

Universidade de São Paulo
Escola de Engenharia de São Carlos
Departamento de Engenharia Elétrica

*Estação de Controle Multimídia com Interface Web utilizando
sistema operacional Linux embarcado em plataforma ARM*

André Márcio de Lima Curvello

São Carlos
2012

ANDRÉ MÁRCIO DE LIMA CURVELLO

**ESTAÇÃO DE CONTROLE
MULTIMÍDIA COM INTERFACE *WEB*
UTILIZANDO SISTEMA
OPERACIONAL LINUX EMBARCADO
EM PLATAFORMA ARM**

Trabalho de Conclusão de Curso
apresentado à escola de Engenharia de
São Carlos, da Universidade de São
Paulo

Curso de Engenharia de Computação

ORIENTADOR: Prof. Dr. Evandro Luís Linhari Rodrigues

São Carlos

2012

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO,
POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS
DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

C975e Curvello, André Márcio de Lima
Estação de controle multimídia com interface web
utilizando sistema operacional Linux embarcado em
plataforma ARM / André Márcio de Lima Curvello;
orientador Evandro Luís Linhari Rodrigues. São Carlos,
2012.

Monografia (Graduação em Engenharia de Computação)
-- Escola de Engenharia de São Carlos da Universidade
de São Paulo, 2012.

1. Linux. 2. Sistemas Embarcados. 3. Web. 4. ARM.
5. Rede de Computadores. 6. Multimídia. I. Título.

FOLHA DE APROVAÇÃO

Nome: André Márcio de Lima Curvello

Título: “Estação de Controle Multimídia com Interface Web utilizando sistema operacional Linux embarcado em plataforma ARM”

Trabalho de Conclusão de Curso defendido em 20/11/2012.

Comissão Julgadora:

Resultado:

Prof. Dr. José Roberto Boffino de Almeida Monteiro
SEL/EESC/USP

Aprovado

Prof. Dr. Francisco José Monaco
SSC/ICMC/USP

Aprovado

Orientador:

Prof. Associado Evandro Luís Linhari Rodrigues - SEL/EESC/USP

Coordenador pela EESC/USP do Curso de Engenharia de Computação:

Prof. Associado Evandro Luís Linhari Rodrigues

Dedicatória

Dedico esse trabalho aos meus pais, Dálton Mário e Maria Dalva, que sempre acreditaram em meus sonhos e se esforçaram ao máximo para me dar apoio e fornecer os meios e ferramentas para alcançá-los.

Dedico também à minha querida esposa, Viviane, minha grande companheira dessa jornada que é a vida.

Agradecimentos

Primeiramente, agradeço a Deus, autor da vida e da minha existência, e o início e o fim de todas as coisas.

Agradeço à minha amada esposa, Viviane, por ser uma fortaleza de apoio e consolo, à qual pude recorrer em diversas ocasiões, e que me forneceu o descanso e energia necessária para voltar à luta.

Agradeço aos meus pais, que sempre estiveram dispostos e atentos a ajudar e apoiar em tudo quanto precisei. Não teria chegado até aqui sem toda a estrutura que me forneceram.

Agradeço ao Sergio Prado, por ajudar-me bastante em meus primeiros passos no mundo de Linux Embarcado.

Agradeço ao Professor Evandro por todo o apoio que me foi dado, por todas as ideias, críticas e sugestões, que ajudaram esse projeto a chegar ao nível em que está.

Para se ter sucesso, é necessário amar de verdade o que se faz. Caso contrário, levando em conta apenas o lado racional, você simplesmente desiste. É o que acontece com a maioria das pessoas.

Steve Jobs

Sumário

Lista de Figuras	17
Lista de Tabelas	19
Lista de Siglas	21
Resumo.....	23
Abstract	25
1. Introdução	27
1.1 A evolução dos sistemas computacionais	27
1.2 Perspectivas do Cenário Tecnológico Atual	29
1.3 Cenário atual dos Sistemas Embarcados.....	30
1.4 Contextualização do Projeto	33
1.5 Objetivos	34
1.6 Organização do Trabalho	36
2. Fundamentação Teórica e Preparativos	37
2.1 Toolchains	39
2.2 Bootloader.....	41
2.3 Kernel	41
2.4 Sistema de Arquivos.....	42
2.5 Buildroot.....	44
2.6 Inserção de Arquivos na Placa.....	44
2.7 Conceitos de Aplicação <i>Web</i> e Linguagens de Programação para <i>Web</i>	44
2.8 Conteúdo Multimídia	46
2.9 Stream de Vídeo	47
2.10 GPS	47
2.11 Sensor de Temperatura Digital DS18B20	50
2.12 OpenCV.....	50
3 Metodologia	52
3.1 Plataforma de Desenvolvimento.....	52
3.1.1 Especificações do Módulo Principal	55
3.1.2 Especificações da Placa-Base	56
3.2 Ambiente de Desenvolvimento.....	59

3.2.1	<i>Toolchain</i> da FriendlyARM	59
3.2.2	Buildroot.....	60
3.2.3	Detalhes gerais sobre acesso à plataforma e ambientes de programação	65
3.3	Sistema Operacional, Bootloader e RootFS	67
3.3.1	Kernel	68
3.3.2	Bootloader.....	68
3.3.3	Rootfs	69
3.4	Aplicações Utilizadas e Desenvolvidas	71
3.4.1	Servidor Web.....	72
3.4.2	Página Web e Componentes de Controle	73
3.4.3	Persistência de Dados	74
3.4.4	Execução e Controle Multimídia	75
3.4.5	<i>Streaming</i> de Vídeo pela Web.....	76
3.4.6	Captura de Fotos	78
3.4.7	Aquisição de Temperatura Ambiente	79
3.4.8	Aquisição de Dados GPS.....	80
3.4.9	Configuração de endereço IP e Gerenciador de DNS Dinâmico	83
3.4.10	Envio de E-mail	84
3.4.11	Envio de Mensagens por Parâmetros para Transmissão RF	85
4	Desenvolvimento das Soluções e Resultados Obtidos.....	88
4.1	Servidor Web e processador de conteúdo PHP	88
4.1.1	Servidor Web Lighttpd.....	88
4.1.2	Processador de Conteúdo PHP.....	90
4.2	Página web	95
4.3	Configuração de IP e DNS Dinâmico.....	102
4.4	Sessão e variáveis de Sessão	105
4.5	Exibição de conteúdo multimídia.....	106
4.6	Execução e Controle de conteúdo multimídia	118
4.7	Execução de Vídeo e Controle de Saída de Vídeo.....	126
4.8	<i>Streaming</i> de Vídeo	130
4.9	Captura de Imagem da Câmera e Captura Temporizada	135
4.10	Banco de Dados SQLite , Persistência e Autenticação	141
4.11	Sensor de Temperatura ds18b20	144
4.12	GPS	149

4.13	Envio e recebimento de mensagens RF	156
4.14	Painel de Administração, Informações sobre a Estação e Controles Adicionais.....	160
4.15	Painel de Controle Multimídia e Status da Estação	165
4.16	Envio de E-mail.....	171
5	Conclusão	176
6	Novas Possibilidades	179
7	Referências Bibliográficas	181
8	Anexos.....	191
8.1	FriendlyARM.ini.....	191
8.2	Start_uvc.sh.....	191
8.3	Lighttpd.conf	194
8.4	rcS – Script de Inicialização do Sistema.....	196
8.5	www/css/style.css.....	199
8.6	Ds18b20_test.c original.....	204
8.7	s3c6410_gpio.h	206
8.8	Ds18b20.c.....	208
8.9	Makefile do Driver DS18B20	216
9	Apêndices	219
9.1	Scripts de Controle PHP.....	219
9.1.1	www/php/auth.php	219
9.1.2	www/php/camCapture.php	220
9.1.3	www/php/createDB.php.....	223
9.1.4	www/php/foldernav.php	224
9.1.5	www/php/gps.php	229
9.1.6	www/php/mail.php.....	234
9.1.7	www/php/mplayer.php	235
9.1.8	www/php/rf.php	241
9.1.9	www/php/status.php.....	243
9.1.10	www/php/temp.php	247
9.1.11	www/php/variables.php	250
9.2	Arquivos JavaScript Desenvolvidos	254
9.2.1	www/js/ajax.js.....	254
9.2.2	www/js/gps.js.....	258
9.2.3	www/js/mediaControl.js.....	260

9.2.4	www/js/rf.js.....	262
9.2.5	www/js/temp.js.....	263
9.3	Scripts de Páginas PHP	266
9.3.1	www/index.php.....	266
9.3.2	www/pages/admin.php	270
9.3.3	www/pages/câmera.php	276
9.3.4	www/pages/contact.php	277
9.3.5	www/pages/docs.php	278
9.3.6	www/pages/gps.php	280
9.3.7	www/pages/home.php	282
9.3.8	www/pages/images.php	283
9.3.9	www/pages/login.php.....	284
9.3.10	www/pages/musics.php	285
9.3.11	www/pages/rf.php.....	286
9.3.12	www/pages/temp.php.....	287
9.3.13	www/pages/videos.php.....	290
9.4	Códigos em C, Shell Script e Python.....	291
9.4.1	cameraCapture.sh	291
9.4.2	gps.py	292
9.4.3	initCap.sh.....	295
9.4.4	mail.py	296
9.4.5	timeCapture.sh.....	297
9.4.6	usbCapArg.c.....	298
9.4.7	Ds18b20Test.c modificado	299

Lista de Figuras

Figura 1 – ENIAC. Fonte: http://www.britannica.com/EBchecked/topic/183842/ENIAC	27
Figura 2 - Intel 4004. Fonte: http://www.extremetech.com	28
Figura 3 - Apollo Guidance Computer. Fonte: http://www.computerhistory.org	28
Figura 4 - PIC18F87, 8 bits, 48 MHz de clock, 128 KB de Flash, 3,8 KB de RAM - Preço: 4,85 euros. Fonte: www.farnell.com	31
Figura 5 - LPC1754, 32 bits, 100 MHz de clock, 128 KB de flash, 32 KB de RAM - Preço: 4.69 euros. Fonte: www.farnell.com	31
Figura 6 - MSP430 sendo alimentado por frutas. Fonte: www.element14.com	32
Figura 7 – Estrutura genérica do <i>Bootloader</i> , <i>Kernel</i> e <i>Rootfs</i> na memória de um dispositivo Linux Embarcado. Fonte: free-electrons.com	37
Figura 8 - Diferenças entre <i>Native toolchain</i> e <i>Cross-compiling toolchain</i> . Fonte: free-electrons.com	40
Figura 9 - Representação ilustrativa da constelação de satélites que fazem parte do GPS. Fonte: www.garmin.com/aboutGPS	48
Figura 10 - Rapsberry Pi Modelo B. Fonte:	53
Figura 11 - Beagle Board xM	53
Figura 12 - Panda Board ES. Foto:	54
Figura 13 - Tiny6410 com Placa-Base e display LCD	57
Figura 14 - Tiny6410 sem o módulo principal e sem o Display LCD. Elementos da Placa estão legendados.	57
Figura 15 - Vista superior da Tiny6410 no LAVISIM, utilizada no Projeto. Na foto, o Linux acabou de ser inicializado, e a câmera CMOS estava conectada à placa.	58
Figura 16 - Instalação do <i>kit</i> e dos componentes externos no LAVISIM	59
Figura 17 - Tela Inicial do <i>Buildroot</i>	61
Figura 18 - Definição de Arquitetura - <i>Buildroot</i>	62
Figura 19 - Tela de Configuração do <i>toolchain</i> - <i>Buildroot</i>	63
Figura 20 - Tela de seleção de pacotes – <i>Buildroot</i>	64
Figura 21 - Tela com opção de salvar configurações - <i>Buildroot</i>	64
Figura 22 - Putty	65
Figura 23 - Aptana Studio 3	66
Figura 24 – FileZilla acessando o sistema de arquivos da Tiny6410	66
Figura 25 - Modelo da Página <i>web</i> desenvolvida para o projeto	74
Figura 26 - Foto do suporte da Webcam e das caixas de som. Nota-se o aviso de que o ambiente está sendo filmado	77
Figura 27 - Foto da localização e esquemático do sensor DS18B20 na Tiny6410. Fonte: www.minidevs.com	79
Figura 28 - Receptor de Sinal GPS. Fonte: www.andahammer.com	80
Figura 29 - Antena receptora de sinal GPS	81
Figura 30 - Detalhe do conector, encaixado na porta COM4	81
Figura 31 - Receptor de Sinal GPS ligado na porta serial da Tiny6410, e também ligado à antena receptora	82
Figura 32 - Posição da antena receptora de sinais GPS na janela do laboratório	82
Figura 33 - Par de placas SAM9 com <i>Transceivers</i> de RF	85

Figura 34 - Seleção do BusyBox - <i>Buildroot</i>	89
Figura 35 - Seleção do Lighttpd - <i>Buildroot</i>	89
Figura 36 - Exemplo de Funcionamento do PHP. Fonte: www.sixrevisions.com	91
Figura 37 - Seleção dos pacotes e opções para PHP	92
Figura 38 - Seleção das Extensões para PHP, Parte 1 - <i>Buildroot</i>	92
Figura 39 - Seleção das Extensões para PHP, Parte 2 - <i>Buildroot</i>	93
Figura 40 - Tela de versão do PHP	94
Figura 41 - Imagem da tela principal do <i>website</i> desenvolvido.....	101
Figura 42 - Seleção do pacote noip no <i>Buildroot</i>	103
Figura 43 - Exemplo de configuração do NoIP em ambiente Linux.....	104
Figura 44 - Diretório mapeado.	115
Figura 45 - Página de Músicas Exibindo os diretórios em /sdcard/mp3	116
Figura 46 - Página de Músicas exibindo os arquivos dentro da pasta Summer Eletrohits 5. ...	116
Figura 47 - Página de Vídeos exibindo os diretórios em /sdcard/videos.....	117
Figura 48 - Página de vídeos exibindo o conteúdo do diretório /sdcard/videos/Clips.....	117
Figura 49 - Página de Imagens mostrando os arquivos e diretórios em /sdcard/pictures	118
Figura 50 - Página de vídeo com dessaque ao seletor de saída de vídeo.	130
Figura 51 - Tiny6410 executando vídeoclipe com saída exibida no <i>display</i> LCD.	130
Figura 52 - Página "Câmeras" do projeto, exibindo dois <i>streams</i> de vídeo.....	135
Figura 53 - Seleção dos pacotes de OpenCV no Buildroot.....	136
Figura 54 - Dessaque dos controles para tirada de fotos da página Câmera.....	141
Figura 55 - Seleção dos pacotes SQLite no Buildroot.....	142
Figura 56 - Tela de <i>Login</i> da Interface <i>web</i>	143
Figura 57 - Aviso de autenticação para conteúdo protegido.....	144
Figura 58 - Página Final de Exibição e Controle de dados de Temperatura.....	147
Figura 59 - Saída dos dados de Temperatura com atualização, após o usuário clicar no botão "Ativar".....	148
Figura 60 - Saída da Exibição de Temperatura para Kelvin	148
Figura 61 - Saída da Exibição de Temperatura com Seleção de Fahrenheit	149
Figura 62 - Seleção de pacotes para o Interpretador Python - <i>Buildroot</i>	150
Figura 63 - Página de Visualização e Controle de Dados GPS	155
Figura 64 - Seleção dos Pacotes do cURL no Buildroot	157
Figura 65 - Página de Controle de Envio e Recebimento de Mensagens de Radiofrequência.	159
Figura 66 - Resultado mostrado após o usuário digitar uma mensagem e clicar no botão "Enviar Mensagem".	160
Figura 67 - Resultado mostrado após o usuário selecionar um Receptor e clicar no botão "Receber Dados".	160
Figura 68 - Visão geral do Painel de Administração e Controle	165
Figura 69 - Painel de Controle de Funções Multimídia	167
Figura 70 - Painel de Controle de Funcionalidades Multimídia. É possível ver a transição do <i>status</i> do Mplayer na parte <i>Mídia em Execução</i>	170
Figura 71 - Painel de Controle Multimídia. É possível ver que o <i>status</i> do MJPG Streamer é mostrado como ativo.	170
Figura 72 - Página de Contato via E-mail.....	174

Lista de Tabelas

Tabela 1 - Sentenças NMEA-0183 para SiRF	48
Tabela 2 - Descrição dos dados da Sentença GPRMC	49

Lista de Siglas

AJAX	- JavaScript Assíncrono com XML - <i>Asynchronous JavaScript and XML</i>
API	- Interface de Programação de Aplicação – <i>Application Programming Interface</i>
CPU	- Unidade de Processamento Central - <i>Central Processing Unit</i>
CGI	- Interface de Acesso Comum – <i>Common Gateway Interface</i>
CSS	- Folhas de Estilo em Cascata - <i>Cascading Style Sheets</i>
DOM	- Modelo de Objeto de Documento - <i>Document Object Model</i>
FastCGI	- Interface de Acesso Comum Rápida – <i>Fast Common Gateway Interface</i>
GPL	- Licença Pública GNU - <i>GNU Public License</i>
HDMI	- Interface Multimídia de Alta-Definição - <i>High-Definition Multimedia Interface</i>
HTML	- Linguagem de Marcação Hiper-Texto - <i>HyperText Markup Language</i>
I/O	- Entrada e Saída - <i>In and Out</i>
IP	- Protocolo de <i>Internet</i> - <i>Internet Protocol</i>
JPEG	- Junta de Grupos de Especialistas em Fotografia - <i>Joint Photographic Experts Group</i>
LAVISIM	- Laboratório de Visão Computacional e Sistemas Microprocessados
LCD	- Visor de Cristal Líquido - <i>Liquid Cristal Display</i>
MIPS	- Microprocessador sem estágios de <i>pipeline</i> intertravados - <i>Microprocessor without Interlocked Pipeline</i>
MIT	- Instituto de Tecnologia de Massachusetts - <i>Massachusetts Institute of Technology</i>
M-JPEG	- JPEG em Movimento - <i>Motion JPEG</i> .
PHP	- Preprocessador de Hipertexto PHP - <i>PHP: Hypertext Preprocessor</i>
RAM	- Memória de Acesso Aleatório - <i>Random Access Memory</i>
RISC	- Computação com conjunto reduzido de instruções - <i>Reduced Instruction set computing</i>
ROM	- Memória de somente leitura - <i>Read Only Memory</i>
SMTP	- Protocolo de Transferência de Correspondência Simples – <i>Simple Mail Transfer Protocol</i>
SoC	- Sistema em um <i>Chip</i> - <i>System on a Chip</i>
USB	- Barramento Serial Universal - <i>Universal Serial Bus</i>
UVC	- Classe de Vídeo USB - <i>USB Video Class</i>
V4L2	- Segunda Versão do Vídeo para Linux - <i>Video For Linux Second Version</i> .

Resumo

O presente trabalho visou desenvolver uma plataforma modelo para desenvolvimento de soluções embarcadas que utilizassem Linux em arquiteturas ARM ou equivalentes, em ambiente de conexão com rede e *internet*. Estudou-se e observou-se as interações de diversos elementos, componentes e linguagens de programação, e as interações mútuas entre eles para a criação de um ambiente capaz de controlar e fornecer conteúdo e dados multimídia através da *web*, além de interagir com elementos diversos, tais como câmeras, sensor de temperatura e GPS. Todo o conhecimento agregado com o desenvolvimento do trabalho é apresentado na forma de tutorial, permitindo a compreensão de conceitos e elementos importantes e pertinentes ao projeto, e a reprodução do projeto. Ao final, são apresentadas sugestões para o desenvolvimento de novas funcionalidades.

Palavras-chave: Linux, Sistemas Embarcados, Web, ARM, Rede de Computadores, Multimídia.

Abstract

This work aimed to develop a model platform for developing solutions using embedded Linux in ARM or equivalent platforms, in an environment of network and internet connection. It was studied and observed the interactions of various elements, components and programming languages, and the mutual interactions between them to create an environment able to manage and deliver content and multimedia data in the web, and to interact with various elements such as cameras, GPS and temperature sensor. All the knowledge obtained with the development of the work is presented in tutorial way, allowing the understanding of concepts and elements important and relevant to the project, and the reproduction of the project. Finally, suggestions are made for the development of new features.

Keywords: Linux, Embedded Systems, Web, ARM, Computer Networks, Multimedia.

1. Introdução

1.1 A evolução dos sistemas computacionais

Em seu surgimento, os computadores eram totalmente diferentes dos quais temos e vemos atualmente. Eram grandes máquinas que consumiam uma quantidade exorbitante de energia elétrica, geravam muito calor, e ocupavam muito espaço. O melhor e mais memorável exemplo desses primeiros computadores é o ENIAC [13], que era um computador que ocupava o espaço de um andar inteiro e consumia 170 kW de energia para funcionar. Seu poder de processamento era equivalente ao de uma calculadora atual. Uma foto do ENIAC é mostrada na Figura 1.

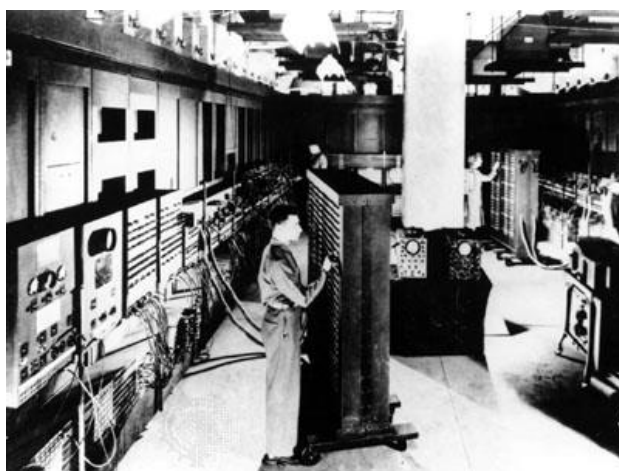


Figura 1 – ENIAC. Fonte: <http://www.britannica.com/EBchecked/topic/183842/ENIAC>

Com a descoberta do transistor por John Bardeen, William Shockley e Walter Brattain no *Bell Laboratories* em 1947 [2], seguida por inovações em tecnologias de fabricação de *chips*¹ de silício e com o surgimento de novas arquiteturas de processadores, como RISC e MIPS, os computadores começaram a ficar menores, a consumir menos energia e a ter uma maior capacidade de processamento. Isso permitiu a transição brutal de grandes e pesadas estações de trabalho, para *chips* contendo literalmente todos os elementos básicos e principais de um computador, como CPU, memória RAM e ROM, e até mesmo alguns periféricos adicionais para controle de I/O, dentre outros elementos. Esses *chips* em questão receberam o nome de microcontroladores.

Na Figura 2 é mostrado o Intel 4004, considerado o primeiro *microprocessador em um chip*, produzido em grande escala pela Intel [44], tendo seu uso inicialmente difundido em calculadoras.

¹ *Chips*, que significa “lascas” em inglês, é um termo usado como sinônimo para pastilhas de silício com circuitos integrados.



Figura 2 - Intel 4004. Fonte: <http://www.extremetech.com>

Inicialmente, os custos para acesso às tecnologias de circuitos de tamanho reduzido para sistemas embarcados eram muito elevados, tanto que somente projetos militares ou governamentais tinham acesso às tecnologias de ponta. Dois exemplos principais são o *Apollo Guidance Computer* [57], e o computador de guia *Autonetics D-17* do míssil nuclear LGM-30 *Minuteman*, como mostrado em [21]. O *Apollo Guidance Computer*, desenvolvido no MIT [56] para servir como computador de guia para os foguetes utilizados no programa espacial *Apollo*, era um sistema computacional de tempo real e também considerado o elemento mais crítico de todo o programa espacial *Apollo*. Uma imagem do *Apollo Guidance Computer* pode ser vista na Figura 3. Já o *Autonetics D-17* foi considerado o primeiro sistema embarcado de produção em massa. Sua primeira versão usava componentes eletrônicos mais primitivos e um disco rígido para armazenar dados. Por sua vez, a segunda versão utilizava componentes eletrônicos integrados e memória do tipo NAND². Só a demanda de memórias do tipo NAND para o programa de fabricação desses mísseis permitiu que o custo desse tipo de memória caísse de mil dólares por *chip* para três dólares por *chip*, o que começou a abrir as portas para o acesso comercial e civil a tais tecnologias.



Figura 3 - Apollo Guidance Computer. Fonte: <http://www.computerhistory.org>

² NAND é um tipo de memória de estado sólido que não precisa de energia para reter os dados armazenados. Maiores informações podem ser encontradas em: <http://whatis.techtarget.com/definition/NAND-flash-memory>

1.2 Perspectivas do Cenário Tecnológico Atual

Considerando os avanços anteriormente citados e a redução no preço dos componentes, uma infinidade de aplicações embarcadas começou a surgir, tais como controles eletrônicos para componentes veiculares, automatização eletrônica e mecânica de processos industriais, calculadoras, agendas eletrônicas, entre outras. Uma quantidade tão grande que hoje estamos rodeados por sistemas embarcados dos mais diversos tipos e tamanhos. É possível encontrar desde máquinas eletrônicas que fazem café com leite na medida e temperatura certa, geladeiras que mostram receitas culinárias obtidas da *internet*, e até mesmo aviões que voam de um lado a outro do planeta sem interferência humana.

A sociedade humana atual é conhecida por *sociedade da informação* [78], tamanha a facilidade de acesso a informações, e a necessidade de que esse acesso seja imediato e de conexões diversas. Além disso, essa demanda por informações tem gerado uma outra demanda por mais e melhores dispositivos eletrônicos.

Outro efeito atual é o chamado *Internet of Things* [32], que descreve o fato de que em breve teremos, possivelmente, mais dispositivos conectados à *internet* do que pessoas. Isso se deve ao fato de atualmente termos televisores do tipo *SmartTV*, celulares *smartphones*, videogames, leitores de livros digitais, geladeiras e mais uma infinidade de dispositivos que são capazes de se conectar à *internet*.

Mesmo com todos os avanços em processos de fabricação e em arquiteturas de processadores, os computadores atuais ainda consomem uma quantidade razoável de energia elétrica. Em média, um *desktop* consome cerca de 140 a 180 W de energia, enquanto que um *notebook* consome em média 60 a 130 W. Esse consumo energético pode não ser como os 170 kW do ENIAC, mas ainda assim é muito grande, se for considerado que o equipamento fique ligado vinte e quatro horas, sete dias por semana. E maior ainda, se for considerada uma casa de família com 4 ou 5 pessoas, cada uma com um computador, teremos um consumo aproximado de 650 W. Pode-se pensar ainda em uma cidade com 1 milhão ou mais de habitantes, e boa parte deles com computadores, o que pode gerar algo em torno de 26 MW de consumo energético, apenas com computadores. Em apenas um *datacenter*, com milhares e milhares desses computadores, o consumo aproximado³ é de cerca de 7,2 MW.

Além do fato de sermos a chamada *sociedade da informação*, vivemos em uma época em que o consumo energético é algo importante e preocupante. De um lado, queremos aparelhos eletrônicos que usem bateria e durem bastante com esse recurso. De outro, queremos usar

³ Pesquisa sobre projetos de Datacenters e seu impacto na infraestrutura do Sistema de Energia. Site: <http://cumminspowerblog.com/pt/category/segmentos/datacenters/>

aparelhos em casa que não tragam um aumento expressivo na conta de energia. Isso tem forçado uma série de pesquisas em elementos eletrônicos com consumo energético cada vez menor, tanto para durarem mais, quando sujeitos à energia de baterias; como também para, ao serem usados em larga escala, não levarem a um colapso da cadeia de suprimento energético atual.

Pensando nessa questão energética, a Intel [43] tem concentrado inúmeros esforços no desenvolvimento de novas soluções de baixo consumo, tal como a nova geração de computadores *Ultrabook* [44], que prometem funcionar por mais de 5 horas com a energia de suas baterias.

Outra arquitetura que tem se destacado bastante é a ARM [7], voltada para dispositivos embarcados e amplamente reconhecida por seu baixo consumo e ótima eficiência. Até pouco tempo, a arquitetura ARM não chamava muita atenção, mas sua evolução para a família de processadores Cortex-A [16], que possuem 2 e até 4 núcleos, podendo alcançar frequências de 1 a 1,5 GHz com um consumo elétrico muito baixo, tem chamado a atenção das empresas, as quais estão utilizando processadores baseados na arquitetura ARM para desenvolver novas soluções. Um exemplo é o *smartphone* Samsung Galaxy SII [76], que possui o processador ARM Cortex-A9 de dois núcleos a 1,2 GHz, e que foi um dos *smartphones* mais vendidos do planeta. Para efeito de comparação, há dez anos mal possuíamos computadores de mesa com processadores de 1 GHz.

1.3 Cenário atual dos Sistemas Embarcados

Considerados esses fatos, tem crescido muito o foco na área de sistemas embarcados, tanto de propósito dedicado como de propósito geral. Os sistemas embarcados de propósito dedicado são componentes computacionais construídos com foco específico na aplicação que devem desempenhar. Exemplos desse tipo de sistemas embarcados são as máquinas de lavar inteligentes. Elas não são um computador de última geração de quatro núcleos, mas sabem a temporização e a força a ser aplicada na lavagem de cada tipo de roupa, como a *Brastemp Ative!* [10]. Já os sistemas embarcados de propósito geral são dispositivos com unidades de processamento com poderio computacional já próximo aos de computadores de uso normal. Exemplos desse tipo de sistemas embarcados são os *smartphones* e *tablets* que existem atualmente, cuja grande maioria utiliza SoCs com núcleos ARM.

Os SoCs são outro grande avanço na área de eletrônica e de sistemas embarcados, em que não há mais um circuito contendo apenas as unidades de processamento e gerenciamento de memória, havendo também componentes que regulam energia, conectividade sem fio, componentes como câmeras e displays LCD, tudo isso integrado em um *chip*.

Atualmente, com o crescimento da demanda, o preço desses componentes tem diminuído bastante, e cada vez mais surgem fabricantes oferecendo produtos a preços cada vez mais competitivos. Vale lembrar que em 2002, um iPAQ⁴ com configurações topo de linha custava cerca de 2 mil reais. Hoje, é possível comprar um *tablet* com Android [4] por menos de 100 dólares e com frete gratuito em lojas como a *Deal Extreme*⁵.

Além disso, os preços de microprocessadores de arquitetura ARM, que é de 32 bits, têm diminuído de tal forma que seu preço está começando a ser equivalente ao preço de microcontroladores de 8 bits, tais como os da família PIC [54]. Ou seja, tem se tornado mais fácil, acessível e vantajoso aos desenvolvedores de soluções utilizarem componentes avançados para desenvolver produtos melhores. Um exemplo desse cenário pode ser visto nas Figuras 4 e 5, as quais mostram o preço do PIC18F87, que é um microcontrolador de 8 bits e custa 4,85 euros a unidade, e o preço do LPC1754, um modelo de microcontrolador ARM, de 32 bits, e que custa 4,69 euros a unidade.



	1841566 PIC18F87J72-IPT	MICROCHIP IC, MCU, 8BIT, 128K FLASH, 80TQFP More Details	111 in stock	Price For: 1 Each 1+ £4.85 ✓ 10+ £4.05 100+ £3.17	<input type="text" value="1"/> 
Check more stock...					

Figura 4 - PIC18F87, 8 bits, 48 MHz de clock, 128 KB de Flash, 3,8 KB de RAM - Preço: 4,85 euros. Fonte: www.farnell.com



	1718543 LPC1754FBD80	NXP MCU, 32BIT, ARM CORTEX M3, 80LQFP More Details	182 in stock	Price For: 1 Each 1+ £4.69 10+ £3.63 100+ £3.31 250+ £3.00 more...	<input type="text" value="1"/> 
Check more stock...					

Figura 5 - LPC1754, 32 bits, 100 MHz de clock, 128 KB de flash, 32 KB de RAM - Preço: 4.69 euros. Fonte: www.farnell.com

⁴ iPAQ é uma linha de dispositivos de mão fabricada pela HP. Maiores informações podem ser encontradas em: <http://welcome.hp.com/country/us/en/prodserv/handheld.html>

⁵ *Deal Extreme* é uma loja virtual de Hong Kong que vende produtos para o mundo inteiro. Endereço virtual: www.dx.com.

Como comentado anteriormente, tem-se também visto o surgimento de produtos e plataformas com foco no baixo consumo energético, fruto de muita pesquisa e de preocupação com consumo, como a família de microcontroladores MSP430 [42], da Texas Instruments. Um exemplo da forma como esses microcontroladores conseguem gerir seu funcionamento e consumir pouca energia é mostrado na Figura 6, em que é possível ver um MSP430 sendo alimentado por frutas.

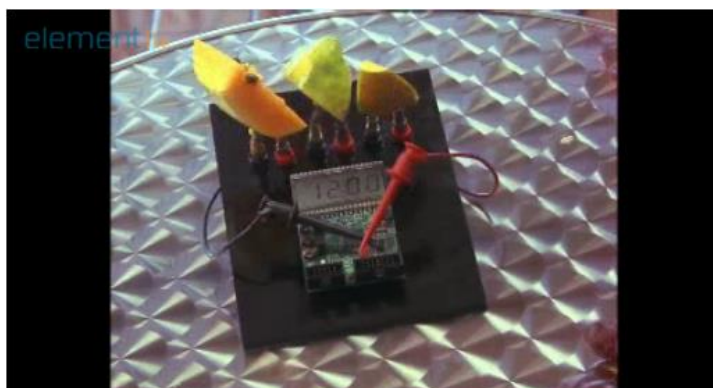


Figura 6 - MSP430 sendo alimentado por frutas. Fonte. www.element14.com

Apesar de terem sido citados componentes de arquiteturas e marcas diferentes, como o PIC da Microchip [54], e o MSP430 da Texas Instruments, uma plataforma embarcada que tem se destacado muito é a ARM [7], como mencionado anteriormente.

A ARM é uma empresa francesa com foco no desenvolvimento de IP⁶. Ela não possui fábricas de componentes eletrônicos, nem vende esses componentes. A empresa somente cria o projeto e licencia seu uso por terceiros. Há uma infinidade de outras empresas e fabricantes que compram licenças de núcleos ARM, e, de acordo com perfis de projetos diversos, vendem produtos contendo microcontroladores ARM propriamente, ou SoCs, agregando o núcleo ARM a outros componentes.

Seguindo essa tendência, vários desenvolvedores, tanto independentes como relacionados a empresas, começaram a criar plataformas de desenvolvimento e soluções que utilizam arquiteturas ARM. Começaram a surgir no mercado aparelhos de GPS, modems, roteadores, *switches*, celulares, televisores e mais uma infinidade de aparelhos eletrônicos com processadores ARM.

⁶ IP, abreviação da expressão em inglês *Intellectual Property*, que significa Propriedade Intelectual em português. Na área de eletrônica, IP significa o projeto, a arquitetura, e não o produto físico em si.

Por se tratar de um sistema operacional de código aberto, o Linux foi logo portado⁷ para a arquitetura ARM. Infelizmente, não há registros muito precisos do primeiro *Kernel*⁸ Linux com suporte para processadores ARM, mas cogita-se que tenha aparecido na década de 90.

Na época do surgimento do *Kernel* Linux para ARM, os preços dos equipamentos e dos *kits* de desenvolvimento com processadores ARM eram elevados, e, portanto, seu uso não era muito difundido no mercado. Porém, com o passar do tempo, com a redução de custos e com o surgimento de novos processadores da família ARM, dentre outros fatores, essa plataforma se popularizou.

Apenas para citar alguns exemplos que podem ser vistos no mercado, há o Kindle [3], um leitor de livros virtuais cujo sistema está construído em cima de um *Kernel* Linux e um processador ARM11. Além disso, a grande maioria dos *smartphones* disponíveis e em uso no mercado utilizam processadores ARM com sistema operacional Android [4], cujo *Kernel* também é Linux. Para que se tenha uma ideia da popularidade dos dispositivos com Android, a empresa Gartner [74] realizou uma pesquisa que indicou uma tendência de que os dispositivos com Android ultrapassem os dispositivos com Windows [55] em 2016.

Além disso, processadores da família ARM têm chamado bastante a atenção pelo fato de possuírem bom desempenho e baixo consumo. Apesar de “ainda” não funcionarem com a energia de frutas, tal como o MSP430, celulares Android com processadores ARM operando em frequências da ordem de 1 GHz funcionam por um ou mais dias na bateria. E essa atenção não tem sido só na área de dispositivos móveis. Como mencionado na parte 1.2, a respeito da preocupação do consumo energético que muitas estações ligadas podem gerar em um *datacenter*, existem pesquisas sérias sobre a utilização de servidores Linux com processadores ARM, tanto que a Dell⁹ doou um protótipo de servidor ARM para a *Apache Software Foundation*, notícia que pode ser vista em [64]. Recentemente, ARM apresentou sua nova família de processadores, capazes de executar instruções de 64 *bits* [38].

1.4 Contextualização do Projeto

O foco desse projeto é no sistema operacional GNU/Linux [49], rodando em plataforma ARM 32 *bits*. A escolha do Linux se deu pelo fato de que o GNU/Linux é de código aberto (normalmente chamado de *opensource*, código aberto em inglês), o que permite total acesso e modificação de qualquer parte e ou componente do sistema. Não são necessários contratos

⁷ É um estrangeirismo da expressão em inglês *ported*, que significa, na área de computação, dizer que um sistema ou aplicação foi adaptado para outro ambiente ou arquitetura.

⁸ É o núcleo o sistema operacional. No caso do sistema Linux, o endereço do projeto é: www.kernel.org.

⁹ Dell é uma empresa multinacional que vende computadores, servidores e presta serviços de TI. Endereço eletrônico: www.dell.com.

empresariais, nem contratos de confidencialidade. O sistema é aberto, bastando respeitar, em alguns casos, o tipo de licença de alguns componentes ou aplicações, que podem ser GPL [33], MIT [56], dentre outros. Embora algumas pessoas¹⁰ julguem ser mais correto chamar o sistema de GNU/Linux, ele será a partir de agora chamado e citado nesse documento apenas de Linux.

O sistema operacional Linux também possui uma ampla, e cada vez maior, quantidade de usuários e desenvolvedores. Basta o uso de alguma ferramenta de busca *web*, como o *Google*, que é possível encontrar fóruns, *websites* de dicas e tutoriais, passo-a-passos, e etc. E mesmo encontrando *websites* com conteúdo em língua estrangeira, o uso de ferramentas de tradução, como o *Google Translate* [36], facilita o acesso e entendimento das informações.

Outro detalhe importante é que o sistema operacional Linux possui um foco em rede desde a sua criação. Além disso, a forma como o sistema é arquitetado permite compilar e carregar módulos do *Kernel* com o sistema em execução, e na maioria dos casos, não é necessário reiniciar o sistema.

Considerando a plataforma, o foco do projeto está na plataforma ARM, em decorrência de sua grande disseminação no mercado e seu uso crescente, justificado por um preço cada vez menor dos processadores. E também pelo fato de já existirem ferramentas de desenvolvimento em um bom grau de maturidade para essa plataforma, além de grupos de desenvolvedores e demais aplicações de Linux. Outras plataformas de 32 bits de sistemas embarcados, como a MIPS [85], são menos conhecidas, e, conseqüentemente, menos utilizadas ou difundidas.

1.5 Objetivos

O trabalho, intitulado *Estação de Controle Multimídia com Interface Web utilizando sistema operacional Linux embarcado em plataforma ARM*, foi escolhido e pensado de modo a alcançar o seguinte objetivo principal:

Fomentar as bases para futuros projetos de sistemas embarcados, através de um guia na forma de tutorial, descrevendo como criar um sistema embarcado de controle multimídia através de uma interface web, e com base nos exemplos e modelos desenvolvidos servir de auxílio para futuros alunos e entusiastas de sistemas embarcados.

¹⁰ Conflito de nomes entre GNU/Linux e Linux é abordado no documentário *Revolution OS*, no qual o criador do *Kernel* Linux, Linus Torvalds, diz achar ridículo chamar Linux por GNU/Linux. Detalhes sobre o documentário podem ser vistos no IMDB: <http://www.imdb.com/title/tt0308808/>.

De modo a permitir alcançar tal objetivo, objetivos secundários foram traçados para servirem de base e apoio ao objetivo principal:

1. Desenvolver um sistema embarcado que permita o uso e o controle de ferramentas e componentes multimídia através da *web*;
2. Desenvolver os componentes de controle e interação multimídia, com a exibição de conteúdo de áudio, vídeo e imagens, dentre outros;
3. Desenvolver uma plataforma de acesso *web* amigável e com acesso rápido a cada um dos componentes e controles;
4. Desenvolver uma solução capaz de interagir com outros componentes *web*, com outros sistemas e com outras plataformas e linguagens de programação;
5. Demonstrar a interação entre componentes e linguagens de programação diferentes;
6. Ser uma solução capaz de ser portada para outras plataformas, sejam elas outros *kits* de desenvolvimento, ou até mesmo computadores *desktop*.

Esses itens foram pensados e escolhidos pelo fato de que o mercado está cada vez mais demandando profissionais capazes de desenvolver soluções embarcadas, tanto para desenvolver soluções e produtos voltados para o usuário final, que deseja produtos amigáveis e com interação multimídia, quanto produtos e soluções voltados para corporações, que devem interagir com elementos diferentes e também fornecer a troca e acesso de dados e informações.

O desenvolvimento de soluções embarcadas encara desafios normalmente não considerados no desenvolvimento de soluções *desktop*, tais como uso de memória, uso de processador e consumo de energia. E, como se não bastassem Essas preocupações, há também um certo cuidado com as ferramentas e arquiteturas utilizadas.

Desse modo, o desenvolvimento desse projeto serve de base e de modelo para mostrar como desenvolver soluções diversas e integradas, como enfrentar e solucionar alguns problemas que podem aparecer no processo de desenvolvimento, entre outras coisas, de modo a auxiliar futuros alunos e pessoas interessadas em desenvolver soluções embarcadas utilizando Linux e em estar minimamente aptos a enfrentar o que o mercado de trabalho espera e demanda.

Dessa forma, unindo os objetivos do projeto à plataforma e sistema escolhidos, esse trabalho então se propõe a extrair a melhor interatividade possível entre os periféricos da FriendlyARM [23] Tiny6410 [27] - que é o *kit* de desenvolvimento utilizado no projeto, o qual possui processador ARM com suporte a Linux - ao criar um projeto que permita o acesso e o controle de diversos dos seus periféricos e componentes através de uma interface *web* amigável.

Os capítulos seguintes mostram todos os detalhes do projeto, as soluções que foram implementadas, detalhes de suas implementações, as modificações que ocorreram ao longo do

tempo, soluções diversas para problemas que ocorreram durante a realização do projeto e desafios relacionados ao tempo necessário para o seu desenvolvimento, dentre outros.

1.6 Organização do Trabalho

O trabalho está organizado da seguinte maneira:

Capítulo 1: É o capítulo introdutório do documento. Relata de maneira resumida a evolução dos sistemas computacionais até chegar aos atuais sistemas móveis e embarcados, assim como também descreve os motivos desse projeto se embasar no sistema operacional Linux [49] em plataforma ARM [7], além de destacar os objetivos do projeto.

Capítulo 2: Trata de introduzir a fundamentação teórica necessária para o entendimento a respeito de elementos importantes do projeto, tais como os conceitos pertinentes ao universo de Linux Embarcado, as ferramentas necessárias para se desenvolver aplicações para Linux em arquiteturas diferentes, conceitos de *web* e linguagens *web*.

Capítulo 3: Esse capítulo trata dos materiais utilizados, tais como *hardware* e *software*, que foram utilizados para o desenvolvimento do projeto. São abordados cada um dos materiais, e também é descrito como cada um foi preparado e planejado para compor o projeto final.

Capítulo 4: Após a descrição dos materiais, esse capítulo descreve o processo de desenvolvimento do projeto de maneira detalhada e os resultados que foram obtidos para cada uma de suas partes.

Capítulo 5: Conclusão do projeto, em que são detalhados os conceitos e lições que foram aprendidas com o seu desenvolvimento.

Capítulo 6: Nesse capítulo são abordadas novas ferramentas e APIS que podem ser utilizadas para desenvolver novas funcionalidades, seja em uma possível evolução do presente projeto, como também em outros projetos derivados deste.

2. Fundamentação Teórica e Preparativos

Um sistema *Linux* embarcado é composto basicamente por três componentes principais: *Bootloader*, *Kernel* e *root filesystem*. Cada um é melhor abordado em seções subsequentes, mas, basicamente, o *bootloader*, na primeira região de memória, é responsável por preparar o *hardware* para a execução do *Kernel*. O *Kernel* é o que gerencia os dispositivos de *hardware* e a interação desses com os *softwares* em execução. Já os *softwares* e demais bibliotecas, arquivos e configurações de sistema e de usuário, ficam armazenados no *root filesystem*, ou sistema de arquivos raiz, em português, que basicamente é o espaço reservado para armazenamento permanente de arquivos no sistema. A Figura 7 mostra o arranjo e a ordem básica desses componentes na memória de um sistema *Linux* embarcado.

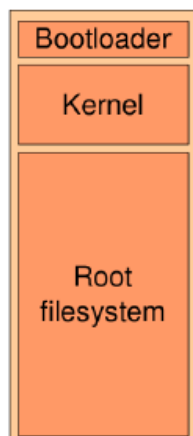


Figura 7 – Estrutura genérica do *Bootloader*, *Kernel* e *Rootfs* na memória de um dispositivo *Linux* Embarcado.

Fonte: Fonte: free-electrons.com

Para começar a desenvolver, codificar, corrigir erros e aprimorar o projeto, foi preciso preparar o ambiente de desenvolvimento para as aplicações que se desejava desenvolver. Caso contrário, nada funcionaria. Afinal de contas, o projeto visa desenvolver aplicações que seriam executadas em uma arquitetura ARM, portanto ferramentas convencionais de compilação, normalmente de arquiteturas x86, não serviriam para esse propósito.

Como referência, a bibliografia [99] é uma ótima fonte de informações, dicas e recursos para desenvolvimento de um sistema *Linux* Embarcado, auxiliando na compreensão mais aprofundada dos componentes do sistema, e de como preparar o sistema manualmente. Além dessa bibliografia, o *site* [20] possui dicas e tutoriais bem didáticos para o desenvolvimento de *Linux* Embarcado.

Para desenvolver aplicações e/ou sistemas completos para ambientes Linux embarcados em plataformas ARM, tem-se duas linhas principais de ferramentas: *buildsystems* e *toolchains*.

O *buildsystem* é um sistema completamente automatizado, capaz de gerar e sintetizar todo o sistema Linux que será embarcado, de maneira customizada pelo usuário desenvolvedor. O *Buildroot* [11] é um *buildsystem*, pois ele é capaz de fazer o *download* de pacotes e de *toolchains*, usar arquivos de configuração definidos pelo usuário, compilar *kernel*, *bootloader* e *rootfs* de formas determinadas pelo usuário desenvolvedor, e gerar os arquivos finais prontos para serem inseridos na memória do dispositivo.

*Toolchain*¹¹ nada mais é do que um conjunto de ferramentas para compilação e depuração de código. Cada *toolchain* fornece basicamente 5 elementos:

1. **Compilador GCC:** É o *GNU C Compiler*, ou Compilador C GNU, traduzido para o português. É compatível com diversas linguagens, como C++ e Java, além de conseguir gerar código para arquiteturas como ARM, x86, MIPS, PowerPC, dentre outras.
2. **Binutils:** É um conjunto de ferramentas para manipular programas binários¹² para arquiteturas específicas, como a ferramenta “as” para ler código-fonte em linguagem *assembly* e gerar um arquivo objeto¹³ correspondente, e a ferramenta “ld”, responsável por fazer a ligação entre um ou mais arquivos objetos em um arquivo executável.
3. **Biblioteca C Padrão:** É a biblioteca responsável por fazer a interface do programa com o *Kernel Linux* através das chamadas do sistema (*System Calls*). Essa biblioteca possui duas implementações mais famosas: a “glibc”, mais usadas em ambientes *Desktop* por ser otimizada para *performance*, e a “uClibc”, mais usada em dispositivos embarcados, por ser otimizada para gerar código menor.
4. **Kernel Headers:** Pelo fato de a biblioteca C padrão conversar com o *kernel* através de chamadas de sistema, é preciso saber quais são as chamadas a serem utilizadas. Elas estão descritas nos *Kernel Headers*, ou cabeçalhos do *Kernel*, de cada versão disponível. Além disso, a biblioteca C padrão precisa ter acesso às constantes e estruturas de dados do *Kernel*.
5. **GDB:** O GDB é o *debugger* padrão utilizado na plataforma *Linux*. Ele não é necessário para compilar aplicações, mas ajuda muito a encontrar problemas em aplicações compiladas. Também auxilia na melhoria e na otimização de aplicações, pois é capaz de mostrar uso de memória e de chamadas ao sistema, o que serve de base para

¹¹ *Toolchain*, traduzido para o português, cadeia de ferramentas. Quando se trata de programação, faz menção ao conjunto de utilitários para compilar, linkar e criar binários executáveis para arquiteturas diversas.

¹² Binários são tipos de arquivos codificados em forma binária, que correspondem aos zeros (0) e uns (1).

¹³ Arquivo objeto é um tipo de arquivo que contém código de máquina realocável, que normalmente não é um executável, mas serve de base para compor arquivos executáveis.

programadores melhorarem suas aplicações.

Obs.1: Uma observação, que serve tanto para a Biblioteca C padrão quanto que para as demais bibliotecas utilizadas, é que as bibliotecas do dispositivo precisam ser as mesmas do *toolchain* utilizado para gerar o sistema ou as aplicações para o sistema, caso contrário, ocorrerão erros na execução das aplicações.

Obs.2: Existe o tipo de compilação que faz uso de *linkagem*¹⁴ estática e *linkagem* dinâmica. A *linkagem* estática é aquela que agrega todas as bibliotecas utilizadas pela aplicação no binário executável gerado. Nesse caso, o tamanho do arquivo binário gerado é maior, mas a chance de ocorrer problemas com bibliotecas incompatíveis é inexistente. Com a *linkagem* dinâmica, o compilador gera o binário executável com referência a locais específicos no sistema, onde estarão as bibliotecas a serem utilizadas, que são carregadas em tempo de execução quando a aplicação é iniciada. Essa forma, porém, é suscetível a casos em que o computador ou dispositivo pode não possuir as bibliotecas necessárias ou equivalentes, o que resulta em erro de execução.

2.1 Toolchains

Com relação aos *toolchains*, existem basicamente dois tipos: *Native toolchain* e *Cross-compiling toolchain*. O tipo *Native toolchain*, traduzido para o português, Conjunto de Ferramentas “Nativo”, visa fornecer a estrutura para compilação de aplicações para a mesma arquitetura do computador em que é realizada a compilação.

Já o tipo *Cross-compiling toolchain*, ou Conjunto de Ferramentas para Compilação Cruzada, visa fornecer uma estrutura para compilar aplicações para outras arquiteturas, diferentes da arquitetura da máquina em que é realizada a compilação. Essas arquiteturas podem ser ARM, MIPS, entre outras, sendo que a arquitetura da máquina em que é realizada a compilação normalmente é x86. Na Figura 8 pode ser visto um modelo visual que explica as diferenças entre compilação nativa e cruzada, a partir de um mesmo código-fonte (*Source-Code*), mas para arquiteturas diferentes.

Há a possibilidade de se realizar a compilação de aplicações no próprio dispositivo rodando *Linux* embarcado, mas como seu desempenho ainda não é satisfatório, essa tarefa ainda é deixada a cargo de máquinas com arquitetura x86.

¹⁴ *Linkagem* é um estrangeirismo adaptado ao português, que vem do termo em inglês *linkage*, que significa unir coisas, ou a ação de unir coisas.

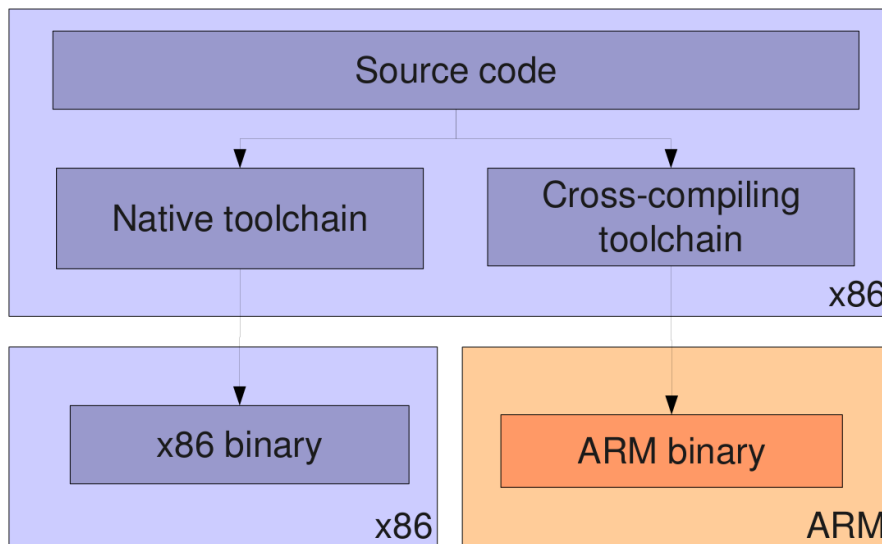


Figura 8 - Diferenças entre *Native toolchain* e *Cross-compiling toolchain*. Fonte: free-electrons.com

Nesse projeto foi utilizado a princípio o *Buildroot* [11], tendo sido usado nos primeiros modelos e testes, com intenso auxílio do Sergio Prado [72] na condução de configurações e correções de erros. Em conjunto, também foi utilizada a *toolchain* que veio no *kit* didático Tiny6410, que é baseada na *toolchain CodeSourcery* [37]. Apesar de ser paga, a *CodeSourcery* é gratuita para uso não-comercial.

Existem outras *toolchains* no mercado, como Monta Vista [58], Pengutronix [68], e até algumas distribuições personalizáveis e compatíveis com uma série de dispositivos, como a OpenEmbedded [66].

Obs.3: Nas próximas partes, são abordados detalhes de cada componente utilizado para a confecção de uma distribuição *Linux* Embarcado, e quais versões desses componentes foram utilizadas no presente projeto. Também são fornecidos dados sobre como esses componentes podem ser *compilados*. Com exceção de uma ou outra aplicação, todas as atividades aqui descritas foram executadas em ambiente *Linux*, mais especificamente a distribuição Fedora 17. Deve-se ter atenção especial aos comandos que começam com “#”, que significa que sua execução se dá em modo de super-usuário. Comandos que começam com “\$” indicam que foram executados como usuários normais. No caso de qualquer dúvida a respeito de uso de console, conceitos e recursos de ambiente *Linux*, recomenda-se a procura de bibliografia específica.

Ainda sobre as *toolchains*, é possível tanto compilar o sistema inteiro, como também compilar aplicações e bibliotecas específicas e inseri-las em um *rootfs*¹⁵ já compilado. A segunda opção é comumente e informalmente chamada de *transplante de aplicações* por profissionais da área, já que se copia os arquivos compilados de uma máquina para outra.

2.2 Bootloader

O *bootloader* é o componente responsável por inicializar o *hardware* para a entrada e ação do *Kernel*. Ele basicamente verifica os componentes de *hardware* presentes, inicia a memória RAM, e carrega o *Kernel* no devido endereço de memória, dando início à sua execução.

É possível encontrar diversos *bootloaders* para dispositivos embarcados, assim como também existem vários *bootloaders* para ambiente *Linux* em computadores e servidores, como o GRUB e LILO. Um dos mais comuns e utilizados em sistemas embarcados, e aquele que foi utilizado nesse projeto, é o U-Boot [88].

O U-Boot é um *bootloader* muito poderoso, dada a sua grande quantidade de opções e o controle que oferece ao desenvolvedor do sistema. O U-Boot fornece uma interface de console ao estilo do terminal *Bash* [34], e permite alterar o *Kernel* e o sistema de arquivos da placa remotamente, através de TFTP [1], dentre outros recursos.

2.3 Kernel

O *Kernel* é o componente principal de qualquer sistema operacional. É ele que gerencia o acesso à memória, ao disco, aos dispositivos do sistema, aos barramentos de comunicação, à CPU, e aos recursos de rede, entre outras coisas.

Com relação ao sistema operacional *Linux*, seu *Kernel* é do tipo monolítico¹⁶, multitarefa (em inglês, *multitask*), ou seja, capaz de executar múltiplas tarefas ao mesmo tempo, e possui de maneira bem delimitada os espaços referentes às aplicações *Kernel* e às aplicações de usuário, normalmente chamadas por *Kernel Space* e *User Space*, respectivamente. Essa delimitação foi criada a título de segurança, já que o *Kernel* normalmente lida com componentes de sistema delicados.

A comunicação entre as aplicações de usuário e o *Kernel* é realizada através das “chamadas ao sistema”, que basicamente são uma série de funções que passam ao

¹⁵ *Rootfs* é a abreviação de *Root File System*, que em português é Sistema de Arquivos Raiz. Compreende todo o conjunto de arquivos, bibliotecas e componentes relacionados ao sistema operacional e demais aplicações agregadas.

¹⁶ *Kernel* monolítico é aquele cujo sistema operacional inteiro opera no *kernel space*.

kernel o que precisa que seja feito, como leitura do disco-rígido, acesso a determinado endereço de memória, entre outros. A bibliografia [53] auxilia bastante na compreensão do funcionamento do *Kernel* do Linux.

Um detalhe curioso do *Kernel Linux* é a forma como ele gerencia os dispositivos do sistema. Basicamente, todos os dispositivos são vistos como arquivos no sistema de arquivos, dentro da pasta “/dev”, no diretório raiz “/”. Isso facilita o uso dos dispositivos pelas demais aplicações, pois basta que se programe como se fosse realizar escrita e leitura em arquivos, com os devidos parâmetros.

Com relação ao gerenciamento de dispositivos, ele também pode ser realizado de maneira estática, pré-determinada, ou dinâmica. O gerenciamento estático é recomendado quando a aplicação em questão não será exposta a componentes diferentes, ou seja, o sistema será executado naquela configuração de *hardware* somente. Já o gerenciamento dinâmico é recomendado quando há modificações de *hardware*, seja em nível de componentes, seja simplesmente se tratando de dispositivos USBs ou outros, como *displays* LCD, dentre outros.

O gerenciamento estático de *drivers* é realizado através de uma lista pré-determinada e fixa de componentes, e o *kernel* trabalha somente com os componentes dessa lista. Já o gerenciamento dinâmico pode fazer uso de dois gerenciadores de dispositivos: o “mdev” [96] e o “udev” [52]. Os dois são essencialmente a mesma coisa, com a diferença de que o “mdev” é o “udev” mais reduzido, recomendado para uso em dispositivos embarcados.

Nesse projeto foi feito uso do gerenciamento dinâmico de *drivers*, ou módulos, com o gerenciador de dispositivos “mdev”.

2.4 Sistema de Arquivos

O sistema de arquivos é o que cuida da leitura e da escrita, e, principalmente, da persistência dos dados trabalhados pelo sistema operacional e das aplicações do usuário.

Quando se trata de sistemas embarcados, normalmente não há o envolvimento de peças como discos-rígidos, que consomem bastante energia e são normalmente utilizados em computadores pessoais e servidores. Com sistemas embarcados, usam-se predominantemente memórias do tipo FLASH.

O lado negativo das memórias FLASH é que elas se desgastam com o tempo, dada a sua estrutura funcional baseada em células lógicas. Por isso, é preciso fazer uso de sistemas de arquivos que tenham um gerenciamento adequado dessas células, isolando as defeituosas, e

sabendo gerir o espaço de modo a conseguir um tempo maior de sobrevivência da memória em geral.

Há, basicamente, quatro tipos principais de sistemas de arquivos para Linux Embarcado:

- **EXT3 [39]:** Abreviação de *Third Extended File System*, em português: Terceiro Sistema de Arquivos Estendido. É amplamente usado em ambientes *Desktop*, sendo muito seguro por possuir mecanismos confiáveis para escrita de arquivos, ajudando na prevenção de danos em ocasiões em que o sistema é indevidamente desligado. Pode ser usado em sistemas embarcados, mas não por muito tempo, já que não possui mecanismos adequados de gestão de células lógicas implementados.
- **YAFFS2 [98]:** Abreviação de *Yet Another Flash File System 2*, ou “Ainda Outro Sistema de Arquivos para Flash 2”, em sua tradução literal para o português. É amplamente usado por dispositivos que possuem sistema *Android*.
- **JFFS2 [81]:** Abreviação de *Journalling Flash File System*. Há pouco tempo era um dos sistemas de arquivos mais utilizados em sistemas embarcados. Com o avanço das tecnologias de memória FLASH, acabou ficando defasado.
- **UBI [61]:** Abreviação de *Unsorted Block Images*. É o sistema de arquivos do futuro para sistemas embarcados, que porém, infelizmente, é o mais complexo e trabalhoso. Funciona com o sistema de arquivos na forma de blocos, e é bem eficiente para o uso dos recursos das células de memória FLASH.

O formato de sistema de arquivos usado no trabalho foi o EXT3. Basicamente, ele foi escolhido por ser menos complexo, e como o sistema estará armazenado em um cartão de memória para desenvolvimento, não se viu problemas em utilizá-lo. Mas é necessário ressaltar que, para produção, recomenda-se fortemente o uso de sistemas de arquivos como UBI ou YAFFS2, que são mais adequados para memórias FLASH.

Durante o período de testes, é altamente recomendado usar uma metodologia de configuração na qual a placa acesse o sistema de arquivos remotamente via NFS [41], em outro computador. Dado o fato de que durante a fase de projeto é realizada intensa compilação cruzada de aplicações, é muito mais viável ter um diretório base no próprio computador, em que as aplicações são inseridas, ao mesmo tempo em que podem ser acessadas remotamente pela placa. Do contrário, seriam necessários sucessivos processos de gravação do sistema de arquivos na memória da placa. Essa metodologia de acesso remoto foi a metodologia utilizada no trabalho, e é descrita passo-a-passo em [29].

2.5 Buildroot

Outra ferramenta formidável, e que se mostrou muito útil para a concretização desse projeto, foi o *Buildroot* [11]. O *Buildroot* é tanto um *buildsystem* quanto um *toolchain*. Ele é capaz de automatizar todo o processo de geração de *Kernel*, *bootloader* e *rootfs*. Além disso, possui uma grande lista de pacotes, os quais podem ser selecionados e compilados para a arquitetura ARM. Tudo isso pode ser feito de maneira automatizada, bastando serem realizadas algumas configurações.

2.6 Inserção de Arquivos na Placa

Após se ter compilado o *Kernel* e o *Bootloader*, e tendo em mãos alguma imagem de sistema de arquivos *rootfs*, é necessário colocá-los no dispositivo. Com ênfase na Tiny6410, há basicamente três formas de inserir tais componentes na placa: Via TFTP [1], via USB e via cartão de memória. Nesse trabalho utilizou-se as imagens carregadas em cartão de memória, o que se mostrou mais prático e viável durante o período de testes. Para detalhes sobre como carregar os componentes em cartão de memória, aconselha-se consultar [30].

2.7 Conceitos de Aplicação Web e Linguagens de Programação para Web

Antes de entrar nos detalhes de uma aplicação *web*, é necessário primeiro iniciar o conceito de *web*. De acordo com as especificações do W3C [91], uma arquitetura *web* é constituída basicamente por uma máquina cliente e por uma máquina servidora, sendo que a máquina cliente realiza requisições do tipo HTTP [91], que é o protocolo base da *web*, à máquina servidora, e essa retorna conteúdo para o usuário, que pode ser uma página *web*, dados de consulta, etc.

Na maioria dos casos, os clientes são representados por computadores normais, mais precisamente, por aplicações denominadas *web browsers*, que são executadas nas máquinas clientes. Os servidores também são aplicações específicas, que respondem a requisições HTTP, rodando em máquinas que normalmente são dedicadas ao propósito de “servir” conteúdo *web*.

Os *web browsers* são aplicações, executadas na máquina cliente, que sintetizam os dados recebidos da máquina servidora em conteúdo compreensível ao usuário, uma vez que a máquina servidora responde às requisições retornando dados estruturados, normalmente no formato HTML [90] e CSS[89], indicando organização, conteúdo e estilo das páginas *web*. O

web browser recebe esses dados, e com base nas informações de organização e estilo, sintetiza o conteúdo para o usuário.

A forma mais comum de a aplicação cliente realizar uma requisição ao servidor é através do método GET [91], que se constitui de requisições normais realizadas pelos endereços URL. Outra forma de requisição, normalmente utilizada para tráfego de informação mais sigilosa, é o método POST [91], que enquadra as requisições em cabeçalhos HTTP [91].

O conteúdo recebido pelo *web browser* cliente normalmente é o que chamamos de página *web*, que se tornou um paradigma amplamente utilizado para difundir dados e informações pela *web*. A linguagem utilizada para escrever páginas *web* é HTML [90], que define *tags* específicas para agrupar e organizar o conteúdo da informação que será exibida para o usuário. Para “embelezar” esse conteúdo, é utilizado CSS [89], que é uma linguagem de estilos que auxilia na aparência e formatação de páginas *web*. Além disso, para permitir um pouco de interatividade, é usado o JavaScript, uma linguagem que é interpretada no *web browser* cliente, que é capaz de realizar cálculos e manipulação de conteúdo DOM HTML, permitindo interatividade tais como Menus, galerias de fotos, gráficos, etc.

De acordo com [82], uma aplicação *web* é qualquer aplicação que use um *web browser* como cliente de acesso, e que faça uso de diferentes recursos para prover serviços, tais como acesso a banco de dados e interatividade com o usuário, entre outros. Ou seja, enquanto que uma página *web* normalmente é um agrupamento de informação estruturada e com estilos [89] de visualização, uma aplicação *web* vai mais além, permitindo interação com o usuário cliente, persistência de dados, resposta a comandos diversos, e normalmente em AJAX [92].

AJAX é a abreviação de *Assynchronous Javascript And XML*, que é o atual estado da arte para troca de dados entre o *web browser* cliente e a aplicação servidora. AJAX permite a realização de requisições *web* de modo transparente ao usuário ou aplicação, e também permite que apenas trechos ou pedaços da aplicação *web* sejam atualizados ou modificados conforme o uso. Ou seja, antes desse padrão, normalmente as páginas de aplicações *web* eram completamente atualizadas sempre que qualquer requisição fosse gerada. Agora, o usuário acessa a aplicação *web* normalmente, e ela é capaz de atualizar seu conteúdo de maneira fluente diante do usuário. Exemplos de aplicações *web* que utilizam AJAX intensamente são Google GMail¹⁷ e Google Drive¹⁸.

¹⁷ Serviço de *e-mail* do Google. Endereço: <http://mail.google.com>

¹⁸ Serviço de armazenamento de dados e documentos do Google. Endereço: <http://drive.google.com>.

Em se tratando do JavaScript, um *framework* que permite facilitar amplamente o manuseio de elementos DOM do HTML, e que também facilita o processo de requisições AJAX, é o jQuery [47]. Basta algumas poucas linhas e já se pode associar requisições específicas a partes também específicas de uma página *web*.

Considerando o lado do servidor de interpretar as requisições, existem formas de tratá-las e produzir retorno para o cliente. As principais formas conhecidas utilizam programas que fazem recurso dos padrões CGI [93] e FastCGI [93]. CGI está caindo em desuso diante do FastCGI, que é um padrão mais novo e otimizado. No caso de se utilizar o CGI como padrão, trata-se de um conjunto de especificações para programas externos interagirem com a aplicação servidora, seja para o disparo de ações específicas, como consultas de banco de dados, seja para a geração de páginas *web* personalizadas.

O padrão mais utilizado é o FastCGI, que possui um desempenho melhor para tratamento das requisições HTTP, consegue suportar um maior número de requisições, e, além disso, é compatível com uma série de linguagens de programação. Há três linguagens que são amplamente utilizadas para gerar conteúdo *web*: Ruby [75], Java [46] e PHP [69]. Apesar de possuírem suas diferenças, próprias das especificações da linguagem de programação, sua essência é basicamente a mesma: disparar ações específicas para requisições específicas e produzir conteúdo *web* personalizado de acordo com as requisições.

Como Ruby e Java são linguagens cujos programas demandam um maior poder de processamento para ser executados, nesse trabalho fez-se o uso de PHP.

2.8 Conteúdo Multimídia

Mídia, de acordo com o dicionário Priberam [73], é qualquer meio utilizado para difusão de informação em formatos tais que explorem os sentidos do ser humano, fazendo uso da fala e da audição, da visão e do tato, principalmente. Exemplos de mídias utilizadas são músicas, que exploram a audição, textos, vídeos e imagens, que exploram a visão, escritas em braile, que exploram o tato, dentre outros.

Multimídia é a combinação de um ou mais meios de comunicação, produzindo mais informação, e também, mais comunicação.

Normalmente, uma aplicação *web* ou *desktop* é chamada multimídia quando ela faz uso de diversas mídias para apresentar informação.

2.9 Stream de Vídeo

Stream vem do inglês “fluxo”. Sendo assim, *stream* de vídeo é um fluxo de vídeo. E vídeo, em seu conceito mais bruto, basicamente é uma sequência temporal de imagens.

De acordo com [83], quando se trata de *stream* de vídeo pela *web*, basicamente se refere a aplicações que fazem uso de protocolos e formas de transmissão de imagens, normalmente utilizando algum mecanismo de compressão, através do protocolo HTTP.

Uma dessas aplicações é o MJPG Streamer [84]. Ele pega frames JPEG de câmeras USB compatíveis com o padrão Linux UVC¹⁹ e faz o *stream* delas como M-JPEG através do protocolo HTTP pela *web*.

2.10 GPS

De acordo com Garmin [31], GPS vem da abreviação da expressão em inglês *Global Positioning System*, que significa Sistema de Posicionamento Global, em português, e que é composto por 24 satélites colocados em órbitas bem precisas. Cada um desses satélites envia informações, tais como hora de envio do sinal e posição do próprio satélite, para a terra. Com base nas informações vindas de vários satélites, um receptor GPS realiza cálculos de triangulações e é então capaz de determinar dados tais como altitude, longitude, latitude e velocidade do receptor. São precisos no mínimo 3 satélites para fornecer dados em duas dimensões, latitude e longitude, e no mínimo 4 satélites para fornecer latitude, longitude e altitude. Assim que a posição do receptor GPS é calculada, informações como velocidade, distância até o destino, entre outras, podem ser obtidas.

¹⁹ UVC é a abreviação de *USB Video Class Linux device driver*, que essencialmente é um padrão que permite suportar câmeras USB com base em um *driver* genérico.



Figura 9 - Representação ilustrativa da constelação de satélites que fazem parte do GPS.

Fonte: www.garmin.com/aboutGPS

Para o uso efetivo de GPS em equipamentos eletrônicos, foram criados alguns protocolos e especificações para representar e transmitir os dados GPS de maneira padronizada. Um dos principais padrões utilizados é o **NMEA 0183** [62].

NMEA é a abreviação de *National Marine Electronics Association* [62], que é uma organização que cuida da educação e do desenvolvimento de equipamentos eletrônicos da Marinha Norte-Americana .

O padrão NMEA 0183 define uma interface de comunicação serial de caracteres ASCII, no qual é determinado um conjunto de sentenças com informações, tais como o tipo de sentença que está sendo transmitida e os dados GPS relativos à sentença, cada qual separado por vírgula (","),. Essa forma padronizada de informações facilita a criação de aplicações que façam uso de dados oriundos de receptores GPS.

Um dos *chips* receptores de sinais GPS mais utilizados é o SiRF [79], sendo a sua versão SiRF Star III uma das mais popularizadas, devido ao seu baixo custo e tamanho. Esse *chip* é compatível com o padrão NMEA 0183, e é o *chip* contido no receptor de sinais GPS utilizado no trabalho. Apesar de o padrão NMEA 0183 estabelecer uma ampla gama de sentenças, o *chip* SiRF trabalha apenas com 5 sentenças, cada qual mostrada na Tabela 1.

Tabela 1 - Sentenças NMEA-0183 para SiRF

Name	SiRF	Descrição
------	------	-----------

GPGGA	Y	Dados de Correção
GPGLL	Y	Dados de Latitude e Longitude
GPGSA	Y	Informações gerais sobre satélites em vista
GPGSV	Y	Dados detalhados de satélites
GPRMC	Y	Dados mínimos recomendados

Pode-se considerar pouco trabalhar com apenas 5 sentenças, vale, porém, lembrar que essas sentenças já fornecem informações suficientes para a maioria das aplicações civis ou de simples usuários.

Como não é o foco desse trabalho adentrar os detalhes de conceito e aplicações a respeito de GPS, mas simplesmente demonstrar uma de suas aplicações, somente a sentença GPRMC será melhor descrita. Para maiores detalhes, recomenda-se procurar bibliografia especializada.

Um exemplo de sentença GPRMC, recebida via transmissão serial em caracteres ASCII do receptor GPS, é mostrada a seguir:

\$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6^a

Como se pode ver, os valores são separados por vírgula. No caso dessa sentença, cada um dos valores possui os significados mostrados na Tabela 2.

Tabela 2 - Descrição dos dados da Sentença GPRMC

Valor	Significado
RMC	Recommended Minimum sentence C
123519	Dado recebido em 12:35:19 UTC
A	Status A=ativo or V=Void.
4807.038,N	Latitude 48 graus 07.038' Norte
01131.000,E	Longitude 11 graus 31.000' LEsse
022.4	Velocidade ao longo do solo em nós.
084.4	Track angle in degrees True
230394	Data – 23 de Março de 1994
003.1,W	Variação Magnética

*6A	Dados de <i>checksum</i> , Sempre começam com *
-----	---

2.11 Sensor de Temperatura Digital DS18B20

O sensor de temperatura soldado na FriendlyARM Tiny6410 é o modelo DS18B20 [43], descrito como um termômetro digital de resolução programável de protocolo *1-Wire* [43]. Fundamentalmente, esse protocolo especifica uma comunicação que requer apenas uma linha de comunicação. Esse modelo de termômetro digital possui uma resolução variável de 9 a 12 bits, e exibe temperatura na escala Celsius. Sua temperatura de operação típica varia de -55 °C a +125 °C.

Cada termômetro DS18B20 possui um código serial único de 64 bits, o que permite que múltiplos termômetros funcionem em um mesmo barramento do tipo *1-Wire*.

Basicamente, o funcionamento do sensor resume-se ao seguinte: determinados comandos hexadecimais são enviados pelo barramento de dados (*1-Wire*), os quais levam o sensor a realizar medidas de temperatura, definir alarmes de temperatura e enviar dados de temperatura. Para cada envio de comandos é definida uma espera aproximada de 750 milissegundos, para que o comando anterior já tenha sido processado quando o próximo for enviado.

Não é o foco desse trabalho descrever o protocolo *1-Wire* e a arquitetura do termômetro DS18B20. Para maiores informações sobre o termômetro e detalhes de sua interface e de como programá-lo, recomenda-se a consulta à documentação do mesmo, que pode ser encontrada em [43].

2.12 OpenCV

OpenCV[65] é uma biblioteca de visão computacional amplamente utilizada pelo fato de possuir uma grande gama de funções implementadas, que facilitam a criação de rotinas e algoritmos de visão computacional.

Para que se tenha uma ideia, a biblioteca OpenCV possui mais de 2500 algoritmos de visão computacional otimizados, prontos para uso, tais como reconhecimento de face, inversões de cores, aplicações de filtros, etc.

Com algumas poucas linhas, já é possível ter uma aplicação capaz de obter uma imagem de um dispositivo de captura de imagem, como uma câmera USB, e realizar algum tipo de processamento em cima dessa imagem.

É compatível com linguagens C e C++, Java e Python. O uso de OpenCV em C ou C++ é, porém, mais recomendável, por possuir maior desempenho e permitir a realização de operações matemáticas pesadas com paralelismo quando executado em computadores com mais de um núcleo.

3 Metodologia

O presente capítulo irá descrever em detalhes a plataforma utilizada para o desenvolvimento do projeto, assim como o ambiente de desenvolvimento, o sistema operacional da plataforma, as aplicações desenvolvidas e as linguagens de programação utilizadas para construí-las, como também componentes externos (câmera, GPS e sensor de temperatura), e os papéis de cada um dos componentes no projeto.

Antes de prosseguir, é altamente recomendável o uso e a familiarização com o sistema operacional Linux em ambiente *desktop*. Esse será o sistema operacional utilizado durante todo o projeto, e as ferramentas utilizadas para criar aplicações serão todas executadas nesse sistema operacional.

3.1 Plataforma de Desenvolvimento

Como dito na Introdução, o custo de componentes eletrônicos, incluindo os processadores ARM de diversos fabricantes, diminuiu de tal forma que hoje é possível encontrar grupos, empresas e projetos fornecendo ótimas plataformas de pesquisa e desenvolvimento de sistemas embarcados, utilizando processadores ARM capazes de rodar Linux. Dentre Essas plataformas podemos citar as seguintes, que foram estudadas como candidatas para a realização do projeto:

- **Raspberry Pi:** É um computador de placa única desenvolvido no Reino Unido pela Fundação Raspberry Pi [70], na intenção de criar uma plataforma para ensino e aprendizado de conceitos básicos de ciências da computação em escolas. Possui um SoC Broadcom BCM2835, que inclui um ARM1176JZF-S de 700 MHz e um processador gráfico VideoCore IV, além de possuir 256 MB de RAM. Além disso, possui saída HDMI, saída de vídeo, saída de som estéreo, alguns conectores GPIO, duas portas USB e conexão de rede Ethernet 10/100. Não possui memória de Estado sólido nem outro tipo de armazenamento, mas usa um *socket* de cartão de memória tipo SD para usar cartões desse tipo como mecanismos de armazenamento e *boot* de sistema. Infelizmente, o *kit* só acompanha a placa e um pequeno manual de instruções. A fonte de alimentação deve ser adquirida separadamente, ou pode ser usado um cabo micro-USB para alimentar a placa. É possível ver o Modelo-B na Figura 10.
 - Preço: US\$ 35,00.
 - Página do Projeto: <http://www.raspberrypi.org/>



Figura 10 - Raspberry Pi Modelo B. Fonte:

- **Beagle Board xM [9]:** É um computador de placa única, com maior poder de processamento. Possui um processador ARM® Cortex™ A8 de 1 GHz e memória RAM de 512 MB, de baixo consumo. É uma placa bem poderosa, possuindo também 4 portas USB, uma porta serial, um conector de energia 5V, entrada e saída de áudio, saída S-Video e HDMI. Possui uma ampla e crescente comunidade de usuários e entusiastas. Infelizmente, o *kit* só acompanha a placa e um cartão de memória. A fonte de alimentação deve ser adquirida separadamente. Uma foto da placa é mostrada na Figura 11.
 - Página do Projeto: <http://beagleboard.org/>
 - Preço: US\$ 149,00



Figura 11 - Beagle Board xM

- **Panda Board ES [67]:** Também é um computador de placa única. Mais poderoso, possui um SoC OMAP4430 que possui um processador ARM Cortex-A9 de núcleo duplo a 1.2 GHz, além de possuir um processador gráfico PowerVR SGX540, que funciona a 384 MHz, e um DSP integrado, o C64x. Além disso, possui 1 GB de

memória RAM DDR2. Como não possui mecanismos próprios de armazenamento de dados, a forma principal é a entrada de cartão de memória tipo SD, que suporta cartões de até 32 GB. Além disso, possui conector de rede Ethernet 10/100, e conectividade Wi-Fi e Bluetooth. Pode enviar sinais de vídeo através de interfaces DVI e HDMI. Possui conectores de áudio 3,5 mm para entrada e saída de áudio. Também possui duas portas USB *host* e uma porta USB OTG. Infelizmente, o *kit* de desenvolvimento é composto apenas pela placa e por um cartão de memória. Demais acessórios, cabos e periféricos devem ser adquiridos separadamente. Uma foto da placa é mostrada na Figura 12.

- Página do Projeto: <http://pandaboard.org/>
- Preço: US\$ 182,00.

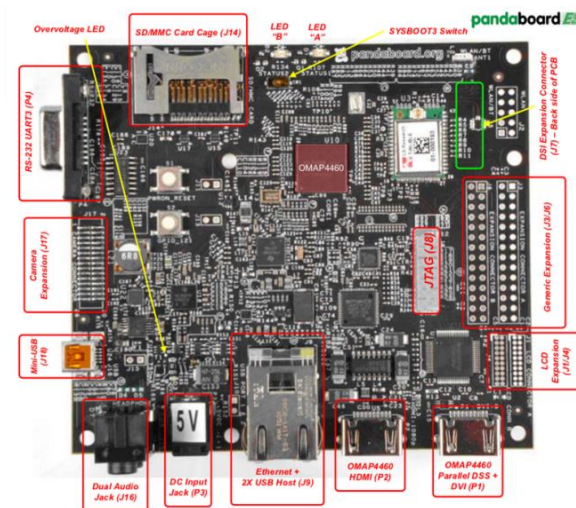


Figura 12 - Panda Board ES. Foto:

- **FriendlyARM Tiny6410 [27]:** É um *kit* composto por duas placas, sendo uma placa-base e outra o módulo principal. Normalmente, ambos são utilizados em conjunto. Considerando ambos os elementos, possui um processador ARM1176JZF-S funcionando a 533 MHz, além de 128 ou 256 MB de memória RAM DDR2 (depende do modelo). Possui memória NAND, podendo ser de 256 MB a 1 GB (também depende do modelo). Como periféricos, possui entrada e saída de áudio estéreo, possui conector de rede Ethernet 10/100, botões, *buzzer*, potenciômetro, algumas portas seriais e portas USB *host* (depende do modelo da placa-base). Também vem com sensor de temperatura e receptor infravermelho. Possui entrada para

cartão de memória do tipo SD. Esse é o *kit* mais completo, vindo com *display LCD touchscreen*, cabos USB e de rede, fonte de alimentação e 2 DVDs com materiais para uso de desenvolvimento. Uma foto da Placa completa é mostrada na Figura 13.

- Página da Empresa: www.friendlyarm.net
- Preço: US\$ 99,00

Por fim, para a realização do projeto, foi escolhida a plataforma FriendlyARM Tiny6410 pelos seguintes motivos:

- O *kit* didático acompanha LCD com *touchscreen*²⁰, cabos de rede e USB, fonte de alimentação, DVDs com material didático (em idioma chinês ou inglês). Os outros *kits* acompanham apenas a placa, e, em um ou outro caso, mais um cartão de memória.
- O *kit* didático possui muitas formas de conectividade, desde várias portas seriais, saída para TV, botões, 3 portas USB, dentre outras, que serão melhor abordadas adiante, numa melhor descrição do *kit*.
- Possui dois grandes fóruns de desenvolvedores: o arm9home.net [8] e o friendlyarmforum.net [24].
- O melhor *kit* com base na relação custo/benefício.

Esse *kit* é composto por uma placa composta por um módulo principal conectado à uma placa-base. O módulo principal contém o processador, memória RAM e NAND, além dos conectores essenciais. Já a placa-base possui componentes diversos, que agregam funcionalidades ao módulo principal. Ambos são descritos detalhadamente logo adiante.

3.1.1 Especificações do Módulo Principal

Com relação ao módulo principal, suas características são as seguintes:

- **Dimensões:** 64 x 50 mm
- **CPU:** 533 MHz Samsung S3C6410A ARM1176JZF-S with VFP-Unit and Jazelle (max freq. 667 MHz)
- **Memória RAM:** 256 MB DDR, *bus* de 32 Bit.
- **Memória Flash:** 1 GB NAND
- **Interfaces:** Serial, SPI, USB, LCD, Câmera CMOS

²⁰ *Touchscreen* é um termo em inglês que denota a funcionalidade de uma tela responder a toques em sua superfície.

- **Saída de Usuário:** 4 x Leds
- **Cabeçalhos de Expansão:** (2,0 mm)
- **Debug:** Conector JTAG de 10 pinos (2,0 mm)
- **Suporte a Sistemas Operacionais:**
 - Windows CE 6
 - Linux 2.6
 - Android
 - Ubuntu

3.1.2 Especificações da Placa-Base

Com relação à placa-base, suas características são as seguintes:

- **Dimensões:** 180 x 130 mm
- **Memória EEPROM:** 1024 Bytes (Barramento I2C)
- **Memória Externa:** *Socket* para cartão de memória tipo SD.
- **Portas Seriais:** 4 x DB9 (padrão RS232)
- **IR:** Receptor Infravermelho
- **USB:** 3 USB *Host* 1.1; 1 miniUSB *Slave* 2.0
- **Saída de Áudio:** Conector stereo 3,5 mm
- **Entrada de Áudio:** Conector 3,5 mm
- **Ethernet:** Conector RJ-45 10/100 (*chip* DM9000)
- **RTC:** Relógio de Tempo Real com bateria
- **Beeper:** PWM *Buzzer*
- **Saída de TV:** CVBS
- **LCD:** Conectores de 40 pinos (2,0 mm) e de 41 pinos para *displays* compatíveis da FriendlyARM e placa VGA.
- **Touchscreen:** 4 pinos (resistivo)
- **Entradas de Usuário:** 8 botões e 1 potenciômetro.
- **Cabeçalhos de expansão:** (2,0 mm)
- **Alimentação Elétrica:** 5 V regulada.

É possível ver em detalhes a placa acoplada à placa-base nas Figuras 13 e 14. Na Figura 13 é possível ver também o *display* LCD firmado na placa-base, e na Figura 14 é possível ver as legendas dos componentes e conectores integrados na placa-base.

Essa será a placa utilizada no desenvolvimento do projeto. Para carregar os arquivos necessários para o funcionamento da placa, que são o *bootloader*, o *Kernel* Linux e o *rootfs*, podem ser tanto usado um cartão de memória configurado para *boot*, como também é possível

inserir os arquivos diretamente na memória NAND da placa, seguindo os passos descritos no Capítulo 3.3 ou na Documentação [30].



Figura 13 - Tiny6410 com Placa-Base e display LCD

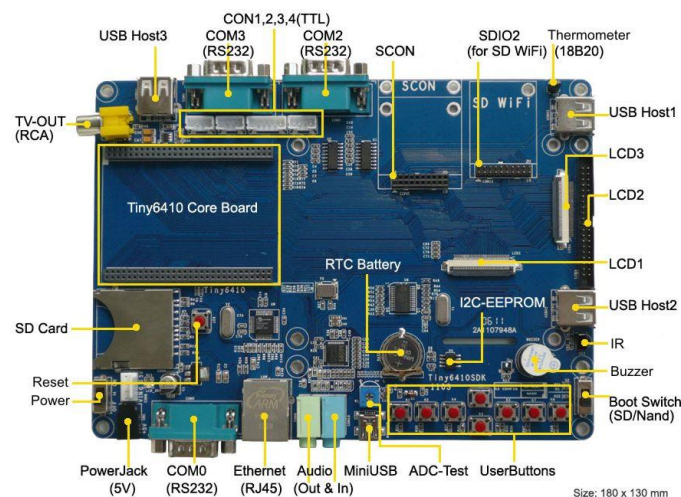


Figura 14 - Tiny6410 sem o módulo principal e sem o Display LCD. Elementos da Placa estão legendados.

Na Figura 15 é possível ver a própria Tiny6410 utilizada no projeto. Nessa figura é possível ver a placa ligada, sendo exibidas algumas das mensagens de inicialização do sistema no *display* LCD. Também é possível ver a câmera CMOS plugada no módulo principal (Essa câmera não foi utilizada no projeto), além do cabo de rede Ethernet plugado no conector de rede, o cabo de áudio conectado na saída de áudio (de cor verde), o cabo da câmera USB plugado na porta USB próxima à saída de vídeo RCA, e o cabo de comunicação e energia do receptor GPS conectado em uma das portas seriais.



Figura 15 - Vista superior da Tiny6410 no LAVISIM, utilizada no Projeto. Na foto, o Linux acabou de ser inicializado, e a câmera CMOS estava conectada à placa.

Na Figura 16 é possível ter uma visão melhor da estrutura montada para o desenvolvimento e funcionamento do sistema desenvolvido no projeto. A Tiny6410 ficou instalada em uma mesa, próxima a uma janela do LAVISIM, laboratório onde ficaram instalados os componentes do trabalho. Juntamente com a Tiny6410, foram colocados um par de caixas de som, conectados à saída de áudio do *kit*, e foi colocado um suporte para a câmera USB, conectada em uma das portas USB do *kit*. Como o ambiente será mostrado pela *web* através de *streaming* de vídeo capturado pela câmera USB, foi afixado ao suporte da câmera USB o aviso “*Você está sendo filmado*”.



Figura 16 - Instalação do *kit* e dos componentes externos no LAVISIM

3.2 Ambiente de Desenvolvimento

O ambiente de desenvolvimento utilizado foi tanto o *Buildroot*, por causa das facilidades que ele oferece para compilar aplicações e bibliotecas, e o conjunto de ferramentas fornecidos pela FriendlyARM, no DVD-A contido no *kit* de desenvolvimento.

O conjunto de ferramentas fornecido pela FriendlyARM fornece o **arm-linux-gcc**, que é o G²¹CC capaz de compilar programas escritos em linguagem C para binários da arquitetura ARM.

O *Buildroot* auxilia muito para compilar pacotes de uma lista pré-definida que o mesmo possui (Por exemplo: MPlayer, PHP, etc), mas o **arm-linux-gcc** permite compilar, para Linux em plataforma ARM, aplicações desenvolvidas manualmente, como simples programas escritos e desenvolvidos especialmente para o projeto.

3.2.1 Toolchain da FriendlyARM

Com relação ao *toolchain* fornecido pela FriendlyARM, o mesmo encontra-se no DVD-A do *kit* de desenvolvimento. Esse *toolchain* é mais do que suficiente para alguns pequenos projetos e pequenas compilações de programas e projetos escritos em Linguagem C ou C++.

Para utilizar esse *toolchain*, é necessário inserir DVD-A no computador que será utilizado como ambiente de desenvolvimento, e acessar a pasta *Linux*, e copiar o arquivo:

²¹ GCC é a abreviação de *GNU C Compiler*, que é o compilador padrão de código C para Linux. Endereço virtual: <http://gcc.gnu.org>.

arm-linux-gcc-4.5.1-v6-vfp-20101103.tgz

Apenas por uma questão de organização, o arquivo foi copiado para a pasta “/”, e dentro desse diretório, foi extraído o pacote comprimido com o seguinte comando:

```
$ tar -xvzf arm-linux-gcc-4.5.1-v6-vfp-20101103.tgz
```

Feito isso, foi criado um diretório, chamado /opt/FriendlyARM/toolschain/4.5.1/.

Para poder utilizar todos os binários dos pacotes fornecidos diretamente pela linha de comando, foram dados com os seguintes comandos:

```
$ export PATH=$PATH:/opt/FriendlyARM/toolschain/4.5.1/bin
```

Após isso, o sistema é capaz de chamar diretamente os binários do *toolchain*, tais como o **arm-linux-gcc**.

Para mais detalhes de como utilizar o *toolchain* e demais componentes fornecidos pela FriendlyARM para compilar e criar o *Kernel*, *bootloader* e *rootfs*, consulte o documento *User's Guide to Mini6410 Linux_041611.pdf* (ou equivalente), que pode ser encontrado dentro da pasta *User Manual* do DVD-A. As informações para configuração do ambiente de desenvolvimento encontram-se na página 138 em diante desse documento.

3.2.2 Buildroot

Para configurar o *buildroot* de modo a gerar os binários e bibliotecas das aplicações que desejamos para o projeto, é preciso baixar o pacote da ferramenta, que será obtido no [link buildroot.uclibc.org/downloads](http://buildroot.uclibc.org/downloads). De preferência, foi baixado o pacote **buildroot-2012-02.tar.gz**.

Após ter sido baixado o pacote, o mesmo foi descompactado em uma pasta com permissão de leitura e escrita, com o seguinte comando:

```
$ tar -xvzf buildroot-2012.02.tar.gz
```

Feito isso, foi criada uma pasta com o nome **buildroot-2012.02**. Para executar corretamente o *Buildroot*, é preciso ter instalado os seguintes pacotes no sistema: **ncurses** e **ncurses-devel**.

Observação: Mesmo com os pacotes mostrados acima, não é garantido o funcionamento total do *Buildroot*, dada a diferença de pacotes existentes nas diversas distribuições Linux em uso atualmente. Quando for iniciada a sintetização do Menu, é preciso ficar atento a qualquer dependência sinalizada, e então utilizar o mecanismo padrão de instalação de pacotes da distribuição para instalar as dependências sinalizadas.

No diretório do **buildroot-2012.02** é digitado o seguinte comando:

```
$ make menuconfig
```

Isso dará início à sintetização do Menu de configuração. Terminado o processo, será possível ver uma tela parecida com a mostrada na Figura 17 no terminal:

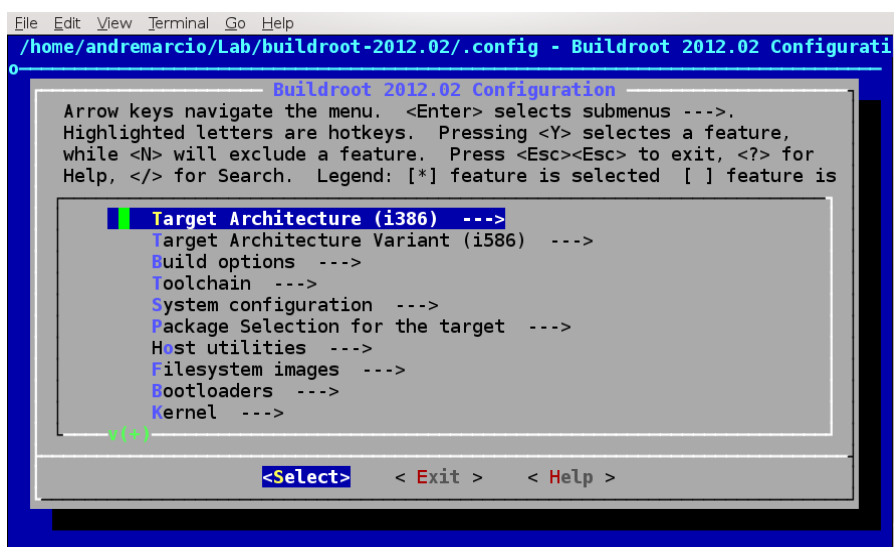


Figura 17 - Tela Inicial do *Buildroot*

Obs: O Menu de configuração é todo controlado através do teclado e de teclas de atalho. Para uso dos atalhos, basta observar as opções, em que há uma letra destacada, normalmente em cor vermelha. Para acessar a opção diretamente, basta pressionar os comandos <Alt> + <Letra destacada>. Com base na tela mostrada na Figura 17, para sair do *Buildroot* bastaria pressionar a tecla “Alt” do teclado juntamente com a letra “E”.

Para configurar o *Buildroot* para a arquitetura do projeto, é preciso definir a arquitetura e a família de processador utilizado. Para isto, é acessada a opção **Target Architecture**. Dentro dessa opção, deve ser marcado o item **arm**.

Após definida a opção **Target Architecture**, é preciso definir a família de processadores, o que é realizado na opção **Target Architecture Variant**, em que deve ser marcado o item **arm1176jzf-s**, que corresponde ao modelo de processador ARM da Tiny6410.

Ao final, as opções marcadas devem ficar como mostradas na Figura 18.

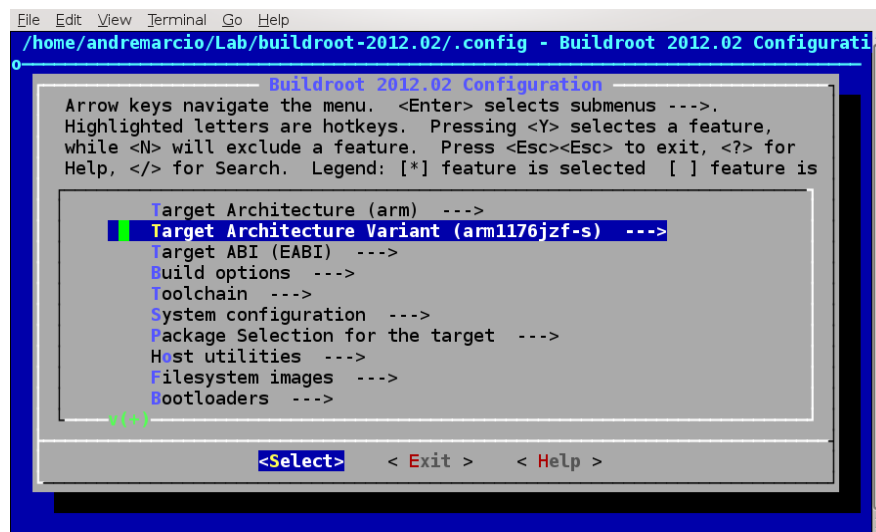


Figura 18 - Definição de Arquitetura - Buildroot

Não é necessário alterar a opção **Target ABI**, que está marcada como **EABI**.

Dentro da opção **Build Options**, o único item que deve ser marcado é o item **Enable Compiler Cache**, que define um *cache* da compilação. Dessa forma, caso ocorram erros durante a compilação, seja do *Kernel*, *bootloader* ou *rootfs*, não será perdido tudo que foi compilado até o momento de algum erro de compilação. Em caso de erros, basta tentar corrigir o que ocasionou o erro, e retornar com o comando de compilação, que o processo retorna do momento em que parou.

Uma das opções mais importantes para o projeto é a opção **Toolchain**. Dentro dessa opção, o **Toolchain type** determina o tipo de *toolchain* utilizado, e **External Toolchain** determina que será feito uso de uma *toolchain* externa. Dentro da opção **Toolchain**, é selecionada a opção **Sourcery Codebench ARM 2011.03**. Depois são marcadas as opções **Download toolchain automatically**, **Enable MMU support** e **Use software floating point by default**. As opções aqui marcadas devem ficar tais como mostradas na Figura 19. Feito isso, é preciso voltar ao Menu inicial para marcar demais opções.

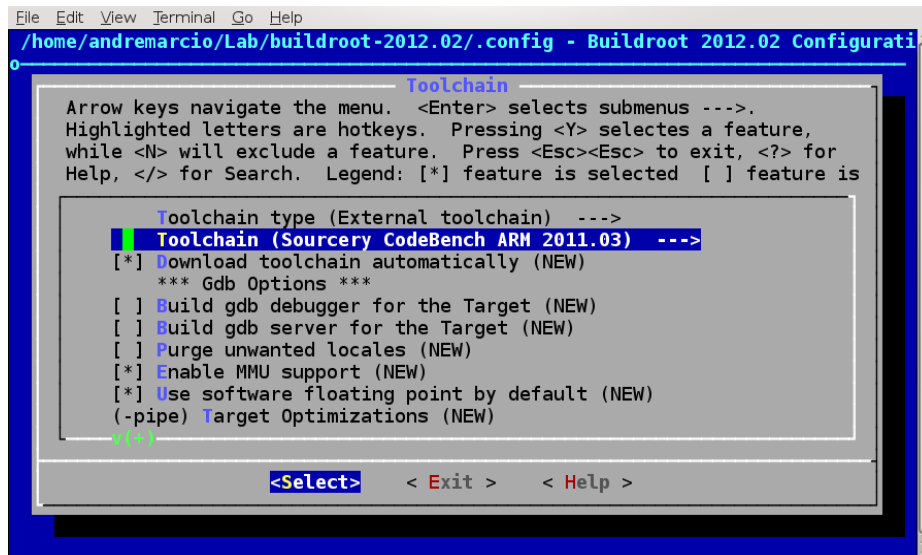


Figura 19 - Tela de Configuração do *toolchain* - Buildroot

Apesar de o *Buildroot* também fornecer uma *toolchain* própria do mesmo projeto, ela não será utilizada. O que acontece é que a *toolchain* do projeto *Buildroot* compila as aplicações usando a biblioteca *uCLibc*. Mesmo compilando um mesmo código para arquitetura ARM, o uso dessa biblioteca inutiliza a inserção das aplicações e bibliotecas no *rootfs* fornecido pela FriendlyARM, que foi compilado utilizando a biblioteca *gLibc*.

As configurações que foram mostradas acima são essenciais para definir as bases do projeto, pois definimos a arquitetura, o processador, além da *toolchain* que será utilizada, e também opções de compilação.

Após isso, é possível determinar configurações de sistema e de pacotes, além de *kernel*, *rootfs* e *bootloader*, que também serão compilados no processo automatizado do *Buildroot*.

Para a maioria dos casos, normalmente já é fornecido pelo fabricante ou projeto um material contendo um *rootfs* pronto. A FriendlyARM fornece um *rootfs* já pronto, como mencionado, contendo diversas aplicações, inclusive um ambiente gráfico muito amigável, o **Qtopia**²².

No caso de compilar apenas alguma aplicação específica, não é preciso compilar todo o *rootfs* novamente. Basta compilar a aplicação propriamente, e transferi-la para o equipamento, normalmente utilizando mecanismos como FTP [94] ou SSH [80]. Esse processo de transferência de aplicações é informalmente citado como *transplante de aplicações* por profissionais da área. Certamente recebe esse nome porque você prepara os binários e

²² Qtopia é um ambiente gráfico desenvolvido para dispositivos móveis. Endereço virtual: <http://qpe.sourceforge.net/>

bibliotecas em uma máquina *host*, que possui uma arquitetura A, e depois os insere em uma máquina ou dispositivo destino, que possui um arquitetura B.

O *Buildroot* automatiza muito bem o processo de compilar uma lista pré-definida de aplicações, que pode ser acessada no item **Package Selection for the target**. Basta selecionar os pacotes e bibliotecas desejadas, salvar o arquivo de configuração do *Buildroot* ao sair do *Buildroot*, e iniciar o processo, que tudo ficará pronto na pasta **output/target** do *Buildroot*. Na Figura 20 é possível ver como é a tela do **Package Selection for the target** do *Buildroot*. E na Figura 21 é mostrada a tela com a opção de salvar o arquivo de configuração do *Buildroot*. Essa tela é mostrada toda vez que alguma modificação for realizada, e é preciso selecionar a opção **Yes** para salvar o arquivo de configuração.

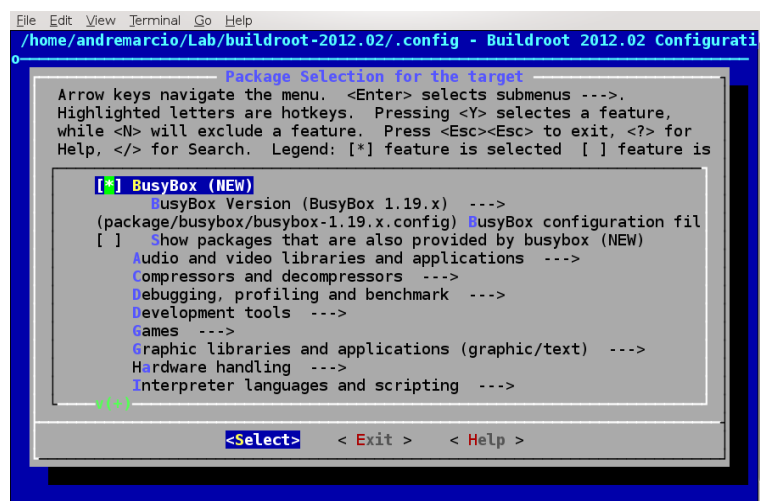


Figura 20 - Tela de seleção de pacotes – *Buildroot*

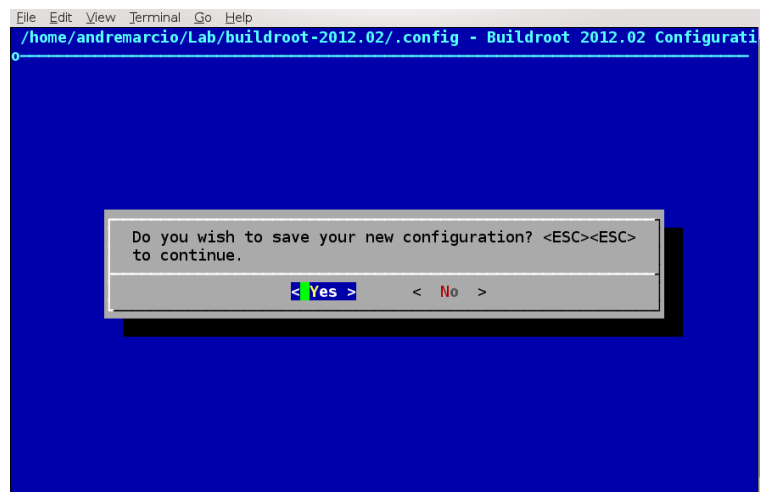


Figura 21 - Tela com opção de salvar configurações - *Buildroot*

A pasta **output** do *Buildroot* é muito importante. Ela possui a seguinte estrutura de diretórios:

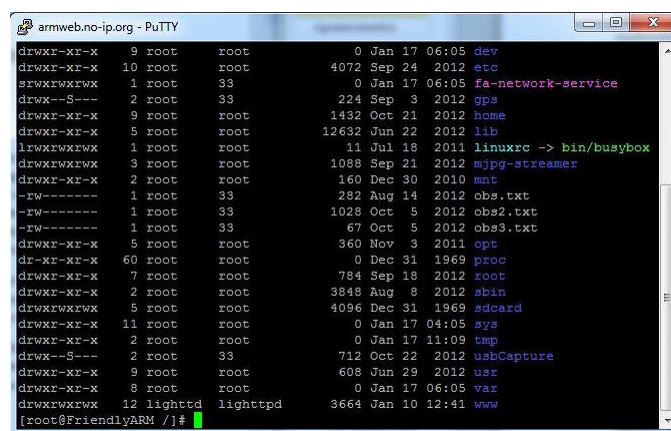
- **build:** Onde o *Kernel* e demais pacotes do rootfs são compilados e agrupados individualmente em diretórios.
- **host:** É onde se encontram as ferramentas para fazer compilação cruzada dos pacotes;
- **images:** É o diretório aonde ficam as imagens geradas do *Kernel*, *rootfs* e *bootloader*.
- **staging:** Nesse diretório fica o *toolchain* gerado pelo *Buildroot*.
- **stamps:** É o diretório de controle do *Buildroot*;
- **target:** É o diretório aonde ficam toda a estrutura de diretórios e arquivos do *rootfs* gerado;
- **toolchain:** Diretório aonde o *toolchain* é compilado.

O diretório que será utilizado sempre é o **target**, pois é nele que Estarão os binários e bibliotecas das aplicações compiladas. E a técnica de *transplante de aplicações* é que será utilizada para transferir as aplicações compiladas pelo *Buildroot* para o sistema Linux da Tiny6410.

3.2.3 Detalhes gerais sobre acesso à plataforma e ambientes de programação

Para acessar o Linux em execução na plataforma, foi primeiramente utilizado o programa **Putty** [71], que fazia acesso via Telnet [86] ao sistema, que já possuía um servidor Telnet (agregado ao BusyBox) em execução e configurado para inicialização automática.

De modo a melhorar a segurança de acesso, foi compilado o OpenSSH para a plataforma, com auxílio do *Buildroot*, e o mesmo foi transferido e colocado em execução, sendo devidamente configurado. Após isso, continuou-se o uso do **Putty** para acesso ao sistema, mas configurado para SSH. Um exemplo de acesso ao sistema Linux da Tiny6410, através do **Putty**, é mostrado na Figura 22



```

armweb.no-ip.org - PuTTY
drwxr-xr-x  9 root  root      0 Jan 17 06:05 dev
drwxr-xr-x 10 root  root     4072 Sep 24 2012 etc
drwxrwxrwx  1 root  root      0 Jan 17 06:05 fa-network-service
drwxr-s---  2 root  root      224 Sep  3 2012 gps
drwxr-xr-x  9 root  root    1432 Oct 21 2012 home
drwxr-xr-x  5 root  root   12632 Jun 22 2012 lib
drwxrwxrwx  1 root  root     11 Jul 18 2011 linuxrc -> bin/busybox
drwxrwxrwx  3 root  root    1088 Sep 21 2012 mjpg-streamer
drwxr-xr-x  2 root  root     160 Dec 30 2010 mnt
drwxr-xr-x  1 root  root     282 Aug 14 2012 obs.txt
-rw-r----- 1 root  root    1028 Oct  5 2012 obs2.txt
-rw-r----- 1 root  root     67 Oct  5 2012 obs3.txt
drwxr-xr-x  5 root  root     360 Nov  3 2011 opt
dr-xr-xr-x 60 root  root      0 Dec 31 1969 proc
drwxr-xr-x  7 root  root     784 Sep 18 2012 root
drwxr-xr-x  2 root  root    3848 Aug  8 2012/sbin
drwxrwxrwx  5 root  root    4096 Dec 31 1969 sdcad
drwxr-xr-x 11 root  root      0 Jan 17 04:05 sys
drwxr-xr-x  2 root  root      0 Jan 17 11:09 tmp
drwxr-s---  2 root  root     712 Oct 22 2012 usbCapture
drwxr-xr-x  9 root  root     608 Jun 29 2012 usr
drwxr-xr-x  8 root  root      0 Jan 17 06:05 var
drwxrwxrwx 12 lightd lighttpd 3664 Jan 10 12:41 www
[root@FriendlyARM /]#

```

Figura 22 - Putty

Para desenvolver a parte de *web*, envolvendo *scripts* PHP, JavaScript, como também arquivos HTML e CSS, foi utilizado o Aptana Studio 3 [6]. Na Figura 23 é mostrada uma tela do Aptana Studio 3.

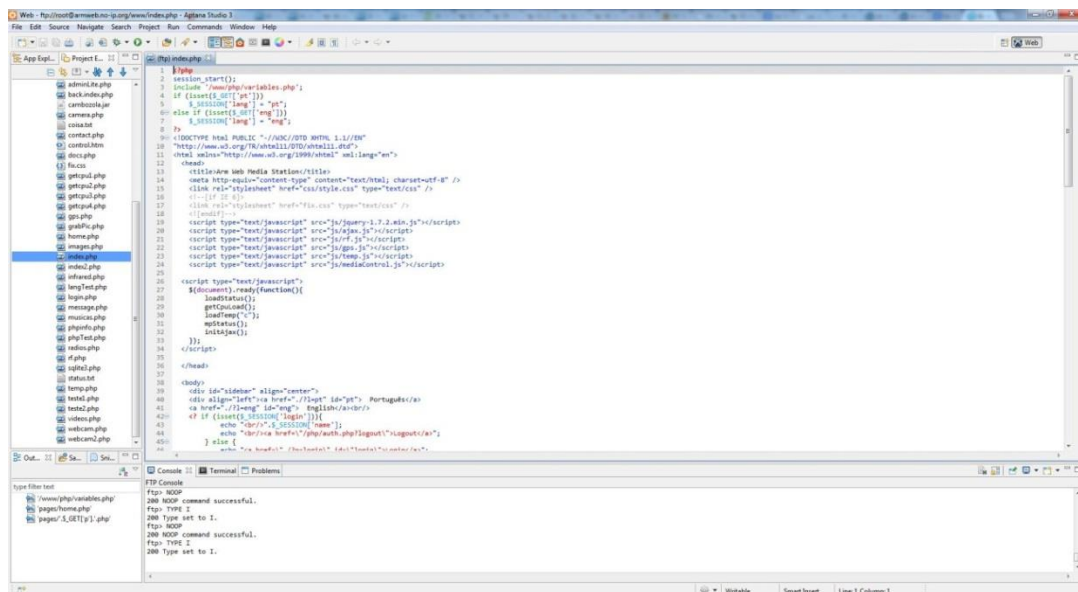


Figura 23 - Aptana Studio 3

Para transferir os arquivos gerados, tanto na parte de *web* como de aplicações gerais, foi utilizado o recurso de servidor FTP [1], já provido pelo BusyBox, da plataforma. De modo a transferir os arquivos da máquina de desenvolvimento para a Tiny6410, foram utilizados os programas FileZilla [22], que é um cliente FTP, e o gerenciador de FTP do Aptana Studio 3 [6]. Na Figura 24 é mostrada a tela do FileZilla acessando o sistema de arquivos da Tiny6410.

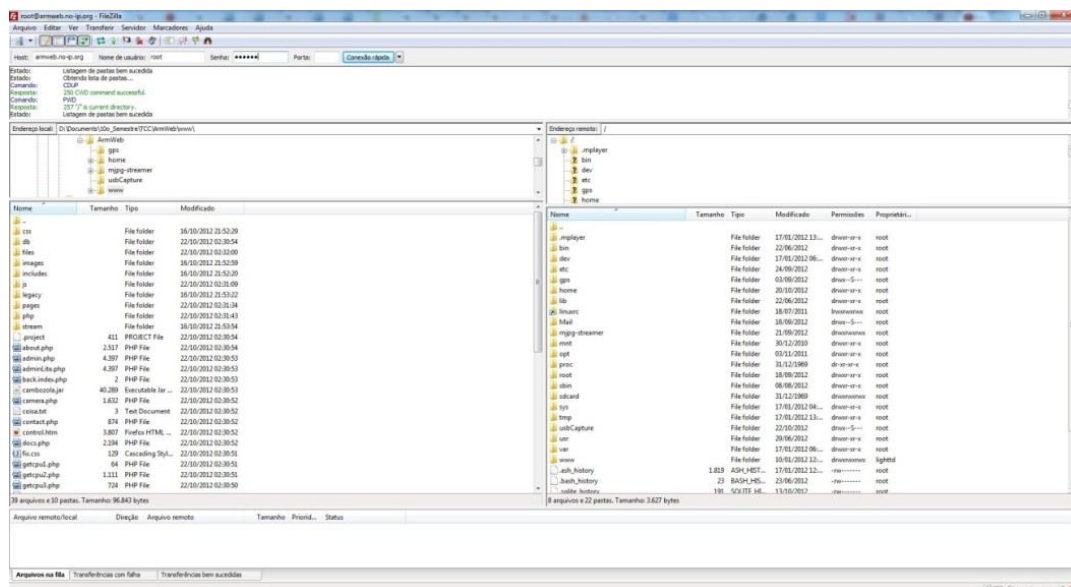


Figura 24 – FileZilla acessando o sistema de arquivos da Tiny6410

Com relação aos arquivos de *Shell Script* e Python, o próprio editor Vi [51] foi utilizado para editar os arquivos, quando na plataforma Tiny6410. Na máquina de desenvolvimento, foi utilizado o GEdit²³.

3.3 Sistema Operacional, Bootloader e RootFS

O sistema operacional utilizado foi o Linux, com Kernel de versão 2.6.38, compilado com a biblioteca gLibc²⁴, utilizando o ambiente de compilação cruzada fornecido pela FriendlyARM.

Antes de entrar em maiores detalhes, vale ressaltar que a Tiny6410 é um *kit* idêntico ao Mini6410 [25], também fornecido e comercializado pela FriendlyARM, e que foi introduzido antes da Tiny6410 no mercado. Por conta disso, a documentação técnica, como manuais de configuração e instalação, é praticamente a mesma, salvo os esquemáticos de detalhes dos *kits*. Então, qualquer referência à Mini6410 ou mini6410 também é uma referência à Tiny6410.

O Kernel foi compilado com suporte aos *drivers* essenciais do *kit* utilizado, a saber: áudio, rede, *display* LCD, saída de vídeo, botões, *buzzer*, suporte a USB e suporte a câmeras USB compatíveis com V4L2²⁵.

O bootloader utilizado foi o U-boot 1.1.5, versão de release **20111018**, configurado para mini6410-sd256, que determina um sistema com 256 MB de RAM capaz de realizar *boot* pelo cartão de memória.

Já o *rootfs* utilizado foi o Qtopia, versão de *release* **20111103**, em formato de sistema de arquivos EXT3, fornecido pela FriendlyARM.

Logo adiante serão fornecidos maiores detalhes sobre os arquivos que foram utilizados, e como devem ser compilados.

É importante destacar que todos os arquivos devem ser compilados com a *toolchain* fornecida pela FriendlyARM.

Todas Essas informações são pertinentes ao processo de compilação manual dos componentes do sistema. A FriendlyARM já disponibiliza os arquivos prontos para serem

²³ GEdit é o editor de textos do projeto GNOME. Endereço virtual: <http://projects.gnome.org/gedit/>

²⁴ GLibc é a biblioteca C padrão do projeto GNU. Página: <http://www.gnu.org/software/libc/>.

²⁵ Camada de suporte a dispositivos de captura de vídeo no sistema Linux. Página: <http://www.ideasonboard.org/uvc/>

carregados, tanto na memória NAND quanto em cartão de memória. Basta acessar o DVD-B e obter os arquivos dentro da pasta *images*. Detalhes desse processo podem ser vistos em [30].

Configurado o *boot* pelo cartão de memória, de acordo com [30], o arquivo de configuração utilizado no projeto é o **FriendlyARM.ini**, mostrado no Anexo 8.1.

3.3.1 Kernel

Com relação ao *Kernel* propriamente dito, foi feito uso da versão 2.6.38, disponibilizada no *kit* FriendlyARM Tiny6410[27], que já contém os arquivos de configuração padrão selecionando todos os *drivers* necessários para o correto funcionamento da Tiny6410.

Para compilar o *Kernel*, é preciso copiar o arquivo “linux-2.6.38-20111116.tgz”, dentro da pasta *linux* do DVD-A do *kit*, e descompactá-lo em uma pasta de preferência com o comando:

```
$ tar -xvzf linux-2.6.38-20111116.tgz
```

Dentro da pasta, podem ser vistos diversos arquivos de configuração. Basicamente, a empresa *FriendlyARM* [11] deixou um pacote pronto para ser usado com a placa em algumas configurações específicas, no que diz respeito à memória RAM e *displays* LCD utilizados. Com relação ao *kit* usado no projeto, temos um equipamento com 256 MB de RAM e com *display* LCD de 4.3 polegadas. Seguindo as instruções de [14], bastou renomear o arquivo “config_mini6410_n43” para “.config” através do comando:

```
# cp config_mini6410_n43 .config
```

E logo após, para criar a imagem, basta dar o comando:

```
# make zImage
```

Ao final, um arquivo, chamado “zImage” é criado. Esse será o arquivo usado para inserir o *Kernel* na memória da placa, tanto em se tratando da memória NAND, interna, quanto do cartão e memória SD, externo.

3.3.2 Bootloader

No *kit* utilizado, o *FriendlyARM* [11] Tiny6410 [13], a empresa disponibilizou uma versão do U-Boot devidamente configurado com os parâmetros de endereços de memória do

hardware, e de demais componentes da placa. Recomenda-se o uso da versão do U-Boot disponibilizado com o *kit*, que é o release **20111018** embora também seja possível fazer uso das versões mais recentes no *site* do projeto [40]. No DVD-A do *kit* é possível encontrar o arquivo “u-boot-mini6410-20111018.tar.gz” na pasta *Linux*. Basta descompactá-lo em um diretório de preferência, com o comando:

```
$ tar -xvzf u-boot-mini6410-20111018.tar.gz
```

Para compilar o U-Boot disponibilizado no *kit*, precisamos observar se iremos fazer uso de *boot* do sistema instalado na memória NAND, ou se faremos uso do sistema instalado em um cartão de memória SD inserido na placa, e também qual a memória RAM da placa utilizada, podendo ser 128 MB ou 256 MB, com relação ao *kit* Tiny6410 [13]. O *kit* utilizado possui 256 MB.

A melhor opção é compilar o U-Boot com suporte a *boot* pelo cartão de memória, pois essa versão também suporta *boot* pela memória NAND.

Para compilar o U-boot, é preciso entrar no diretório gerado após a descompactação do arquivo u-boot-mini6410-20111018.tar.gz, e digitar os seguintes comandos:

```
# make mini6410_sd_config-ram256
# make
```

Esses comandos irão preparar os *Makefiles* para o modelo da placa, e depois irão iniciar todo o processo de compilação.

Após o término do processo, será gerado um arquivo chamado “u-boot.bin”, que deve ser carregado na placa através do utilitário DNW, também disponibilizado no *kit* FriendlyARM, e que pode ser encontrado no diretório “tools”, dentro do DVD-A. Um detalhe importante aqui, é que esse utilitário é desenvolvido para ambiente Windows [55]. Para maiores detalhes de utilização do DNW, consultar bibliografia [30].

3.3.3 Rootfs

O *rootfs* fornecido pela FriendlyARM pode ser encontrado no arquivo **rootfs_qtopia_qt4-20111103.tgz**, que está dentro do diretório *Linux*, no DVD-A incluído no *kit* Tiny6410.

Para descompactar o *rootfs*, copie o arquivo para um diretório de preferência, e basta prosseguir com os seguintes comandos no terminal:

```
$ tar -xvzf rootfs_qtopia_qt4-20111103.tgz
```

Esse comando irá gerar o diretório *rootfs_qtopia_qt4*, que contém toda a raiz do sistema de arquivos que deverá ser inserida na Tiny6410, seja na memória NAND ou no cartão de memória. Porém, esse diretório está em formato bruto, e não em um formato preparado para ser colocado na memória, o qual deve ser UBIFS, EXT3, dentre outros.

Para transformar esse diretório em um arquivo único, no formato de sistema de arquivos necessário, é preciso fazer uso das ferramentas **mktools**, também disponibilizadas pela FriendlyARM, as quais são capazes de transformar diretórios de *rootfs* em arquivos devidamente formatados para um determinado tipo de sistema de arquivos.

Para instalar os **mktools**, é preciso mover o arquivo **mktools-20110720.tgz**, localizado na pasta **Linux** do DVD-A, para a pasta raiz do sistema Linux do computador utilizado no desenvolvimento, e descompactar esse arquivo. Ambos os comandos devem ser feitos em modo super-usuário. Seguem os exemplos de comandos:

```
# cp mktools-20110720.tgz /  
# cd /  
# tar -xvzf mktools-20110720.tgz  
# rm mktools-20110720.tgz
```

Após essa sequência de comandos, os utilitários necessários foram copiados para a pasta */usr/sbin*, do sistema Linux utilizado no computador de desenvolvimento.

Como não se trata de um projeto crítico, a imagem que será gerada será no formato EXT3, que é um formato mais comum. Outros projetos de sistemas Linux embarcados devem fazer uso de sistemas de arquivos em formatos UBIFS ou YAFFS2, que são melhor otimizados para memórias de Estado sólidos tais como NAND.

De posse dos utilitários de geração de sistema de arquivos devidamente formatados, é preciso gerar o arquivo único, formatado para o sistema de arquivos EXT3. De acordo com a documentação [3], o utilitário **mkext3image** será utilizado para esse propósito, com os seguintes comandos:

```
$ mkext3image rootfs_qtopia_qt4 rootfs_qtopia_qt4.ext3
```

Com o término do processo, será gerado o arquivo “rootfs_qtopia_qt4.ext3”, e é esse o arquivo que contém todos os arquivos necessários do sistema, em se tratando de binários, bibliotecas, etc. Para inserí-lo na memória NAND ou no cartão SD, basta seguir as instruções de [30].

3.4 Aplicações Utilizadas e Desenvolvidas

Aqui serão descritas as aplicações que foram escolhidas para o desenvolvimento do projeto, além de algumas pequenas aplicações que foram desenvolvidas para o projeto.

Antes de comentar sobre as aplicações, vale ressaltar que a escolha de cada uma delas se deu com base em uso e consumo de memória e de processamento, além de facilidades de uso. Por se tratar de um sistema embarcado, e não de um computador com amplos recursos, é extremamente preferível e necessário priorizar aplicações que tenham um menor consumo de memória. Do contrário, o sistema pode até funcionar, mas será muito lento, e o péssimo desempenho pode tornar seu uso desagradável.

Como um dos principais objetivos do projeto é fazer uso dos periféricos da Tiny6410 através de uma interface *web*, foram pensadas as seguintes características e aplicações, que fazem uso da grande maioria de periféricos embutidos e externos da Tiny6410:

- **Interface *web*:** A placa possui conexão de rede através de interface Ethernet, dessa forma, é possível acessá-la através de uma rede interna e através da *internet*. Para permitir essa forma de acesso através de um *browser web*, é necessário possuir um **servidor *web*** e uma **página *web***.
- **Controle de Componentes da placa pela *internet*:** Tendo o mecanismo de acesso através de um *browser web*, é preciso alguma forma de controlar programas do sistema Linux da placa.
- **Persistência de Dados:** De modo a permitir guardar dados de configuração e de acesso ao sistema, é preciso uma aplicação capaz de persistir tais dados de maneira confiável e acessível.
- **Execução de Conteúdo Multimídia:** Para executar conteúdo multimídia, é preciso uma aplicação capaz de executar músicas e vídeos, e que permita o controle de volume, a troca de mídias, o controle de execução da mídia corrente, etc.
- **Exibição de Vídeo:** Para exibir vídeos, é preciso uma aplicação multimídia compatível com os *drivers* de saída de vídeo RCA e para o *display* LCD.

- **Captura de Imagens:** Para permitir que o usuário tire fotos do ambiente, é preciso um mecanismo que responda a requisição do usuário disparando uma aplicação na máquina que tire uma foto através da câmera USB, conectada à Tiny6410.
- **Streaming de Vídeo:** De modo a mostrar o ambiente no campo de visão da câmera USB conectada à Tiny6410, é preciso uma aplicação capaz de enviar o *stream* de vídeo pela *web*, e mostrar esse conteúdo na página.
- **Obtenção de Temperatura Ambiente:** Para permitir saber qual a temperatura ambiente do local aonde se encontra a Tiny6410, é preciso uma forma de ler essa temperatura, trabalhar e processar os dados lidos e exibí-los ao usuário.
- **Exibição de Dados GPS:** Para permitir a exibição de dados obtidos por um sensor GPS conectado à placa, é preciso ter um programa que receba Esses dados, e Esses dados precisam ser processados e exibidos para o usuário através da *web*.
- **Envio de E-mail:** Como uma forma de *feedback* do projeto, é bom ter um mecanismo de envio de e-mails. Para tal, as mensagens digitadas precisam ser processadas e enviadas para destinatários específicos.
- **Status do Sistema:** Para saber o *status* geral do sistema, tal como memória disponível, número de processos, uso de CPU, etc, são precisos comandos, *scripts* e maneiras adequadas de mostrar os dados ao usuário.

Cada um dos componentes destacados em negrito serão melhor descritos nas sessões posteriores.

3.4.1 Servidor Web

Primeiramente, para fornecer o acesso à *web*, temos que ter um servidor de conteúdo *web*. Existem diversos atualmente disponíveis para Linux, tais como o Apache [5], Boa [95], tthttpd [87] e o lighttpd [48].

Para o projeto, é necessário um servidor *web* capaz de executar *scripts* em PHP [69], algo que será melhor abordado adiante no Capítulo 4, mas que é uma linguagem que oferece muitas facilidades para manipular páginas *web*, e aplicações da máquina servidora, o que é muito necessário em se tratando de uma Estação de controle de aplicações internas e aplicações multimídia.

Dentre as aplicações servidoras listadas, as que suportam PHP são a *lighttpd* e o Apache. Foi escolhida a *lighttpd* pelo fato de essa consumir, em média, 5 Mb de memória RAM, enquanto que o Apache consome 20 Mb.

Além disso, o *lighttpd* é uma aplicação servidora muito fácil de ser configurada, bastando um simples arquivo de configuração, no qual são definidos permissões de acesso, diretórios a serem monitorados, tipos de arquivos permitidos, e o *socket* PHP.

A versão do *lighttpd* utilizada foi a 1.4.30.

3.4.2 Página Web e Componentes de Controle

Como mencionado no Capítulo 2, toda e qualquer página *web* é escrita em linguagem HTML [90], e para ser agradável visualmente, é necessário possuir *scripts* CSS [89], que definem estilos para cada componente HTML, podendo ser cor, fonte, tamanho do texto, alinhamento de componentes, etc. Sem o CSS, temos aquelas tradicionais páginas *web* do início da *internet*, que eram páginas bem simples e pacatas. Com CSS, temos páginas *web* mais bonitas.

Além disso, para permitir o controle de componentes HTML e a realização de requisições AJAX entre a página *web* exibida no computador cliente e o servidor, representado pela Tiny6410, é necessário o uso de JavaScript [92]. E para facilitar mais a programação das requisições, foi feito uso do *framework* jQuery [47].

Para o manuseio e tratamento mais fino e até mesmo mais adequado da página *web*, é necessária alguma aplicação do lado do servidor, que trate os dados antes de enviá-los ao cliente, e que seja também capaz de realizar ações na máquina servidora. Tal aplicação é o PHP [69].

O PHP é um linguagem de programação executada no servidor, de propósito geral, capaz de criar portais, *sites* com mecanismos de autenticação, é capaz de tratar requisições GET [91] e POST [91], do padrão HTTP [91], e dispara ações, que podem resultar em novas páginas sintetizadas de modo personalizado para o usuário ou ação, ou até mesmo disparar determinados aplicativos e parâmetros na máquina servidora.

A versão de PHP utilizada no projeto foi a versão **5.2.17**, com suporte a SQLite3²⁶, através do PDO²⁷, além de outras opções, que foram habilitadas na compilação.

Pensando em uma forma adequada de organizar a página, concluiu-se que a melhor maneira seria um *layout* com o estilo de três colunas e um rodapé. A coluna da direita deve conter o Menu, com as opções de acesso. A coluna do meio, ou do centro, será responsável por exibir os conteúdos da opção selecionada. E a coluna da esquerda deverá conter um painel de

²⁶ Banco de dados implementado em linguagem C, muito usado em dispositivos embarcados. Página do projeto: <http://www.sqlite.org/>.

²⁷ Abreviação de *PHP Data Objects*. Conjunto de rotinas para acesso a mecanismos de persistência do PHP. Fonte: <http://php.net/manual/en/book.pdo.php>.

control, e um painel de *status* geral da placa. O rodapé serve simplesmente para exibir informações do projeto.

Para exemplificar o modelo proposto, é mostrada na Figura 25 o *layout* planejado. E a melhor maneira de se obter esse *layout* é com o uso de *tags* HTML específicas, além de configurações CSS.

Para controle de elementos multimídia e demais funcionalidades, serão utilizados botões, e Esses irão disparar funções JavaScript, que por sua vez irão disparar requisições ao servidor *web*.

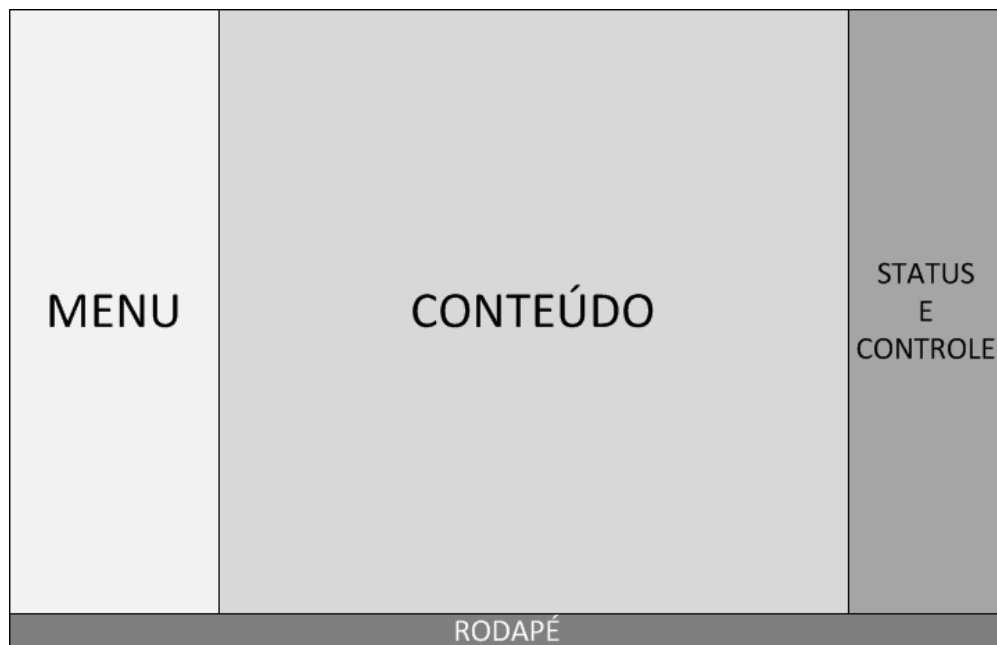


Figura 25 - Modelo da Página *web* desenvolvida para o projeto.

3.4.3 Persistência de Dados

Normalmente, em muitos projetos de sistemas embarcados, o modo de persistência de dados utilizado é a simples escrita e leitura de arquivos de texto. Apesar de prático, esse não é um modelo adequado, principalmente quando o volume de dados pode aumentar, e passa a ser necessário lidar com tabelas e colunas de dados, ao estilo do Modelo Relacional.

Os Bancos de Dados tradicionais e mais conhecidos, tais como MySQL (visto com detalhes na página www.mysql.com), e PostgreSQL (página do projeto www.postgresql.com), são muito pesados para serem executados em plataformas embarcadas, que possuem pequenos recursos de memória e processamento.

Contornando esse fato, existe um Banco de Dados, o SQLite, que é muito leve e comumente utilizado em muitos projetos de sistemas embarcados, e até mesmo em pequenas

aplicações de ambientes *desktop*. Ele utiliza uma pequena quantidade de memória quando em execução, e, além disso, utiliza comandos SQL. O SQLite também pode ser usado por aplicações escritas em diversas linguagens de programação, sejam elas escritas em PHP, para manusear conteúdo *web*, ou até mesmo escritas em *Python* e *C*.

Como o PHP foi compilado com suporte a SQLite para o projeto em questão, seu uso foi essencial como mecanismo de persistência de dados, permitindo a autenticação de usuários, e o armazenamento de dados de configuração do sistema.

A versão do SQLite3 utilizada foi a 3.7.10.

3.4.4 Execução e Controle Multimídia

O *kit* FriendlyARM Tiny6410 possui *display* LCD e saída de vídeo do tipo RCA, capaz de ser ligado em projetores e televisores diversos. Além disso, ele possui também saída de áudio, que pode ser tanto ligada em fones de ouvido, como em aparelhos de som.

Para a realização do projeto, percebeu-se a necessidade de encontrar uma aplicação capaz de ser executada através de linha de comando, o que permite seu controle através dos recursos do PHP (tais quais serão melhor abordados no Capítulo 4), e também capaz de receber comandos externos quando em execução, como por exemplo alteração de volume.

É possível encontrar uma grande quantidade de aplicações capazes de executar músicas via linha de comando, tais como a *mpg123*[59], mas elas não executam vídeos, e não recebem comandos externamente.

A melhor aplicação, adequada aos objetivos do projeto, acabou sendo o **MPlayer**[60], que pode ser executado através da linha de comando do Linux, e pode receber comandos externos quando em execução. O MPlayer possui um modo de execução, chamado *slave*²⁸, que pode ser associado a um arquivo do tipo *pipe*²⁹. Dessa forma, a escrita de comandos específicos no arquivo dispara tais comandos na aplicação, permitindo alteração de volume e de Estado da execução, tais como pausar a música, colocar no mudo, etc.

Um revés inicial foi que a versão normal do MPlayer não era capaz de enviar vídeos para a saída de vídeo RCA do *kit*.

Procurando no fórum de usuários *arm9home.net* [8], foi possível encontrar uma versão do MPlayer modificada, pelo pessoal da FriendlyARM, capaz de mostrar vídeo tanto no *display* LCD, como na saída de vídeo, como também em *ambos*. Tal funcionalidade foi muito bem

²⁸ *Slave* significa escravo, em português.

²⁹ *Pipe* significa tubo em português.

recebida. Essa última versão do MPlayer passou a ser incluída nas *rootfs* fornecidas pela FriendlyARM.

Infelizmente, o pessoal da FriendlyARM ainda não liberou o código-fonte desse MPlayer modificado. Entretanto, ele é compilado utilizando a biblioteca gLibc, e seu uso é liberado e permitido para qualquer fim, sem pagamento de licenças de uso. Para usar a aplicação MPlayer em algum outro projeto, basta copiar os binários e as bibliotecas necessários, processo conhecido por *transplante*, que será melhor explicado no Capítulo 3.

A versão do MPlayer utilizado no projeto foi a versão 1.0rc2-4.5.1.

3.4.5 Streaming de Vídeo pela Web

Um dos principais objetivos do projeto, além do controle de conteúdo multimídia, é a exibição do vídeo capturado por uma câmera USB plugada no *kit* através da *web*, o que permite a visualização do ambiente monitorado pela câmera através da página do sistema. Na Figura 26 é possível ver o arranjo montado para monitoramento do LAVISIM, no qual há um papel escrito “Você está sendo filmado”, indicando que o ambiente é monitorado.

Apesar de existirem diversas formas de *streaming* [83] de vídeo, uma das aplicações mais comuns é a **MJPEG-Streamer** [84], utilizada e recomendada pelo próprio pessoal da FriendlyARM, de tal modo que fornece um tutorial de compilação cruzada e utilização da aplicação [28].



Figura 26 - Foto do suporte da Webcam e das caixas de som. Nota-se o aviso de que o ambiente está sendo filmado.

A maior exigência dessa aplicação é que ela só suporta com câmeras USB compatíveis com o UVC³⁰, que define uma camada de funcionamento de *drivers* de captura de vídeo para Linux. E para que o sistema Linux consiga comunicar com esse tipo de câmeras USB para captura de vídeo e de imagens, é preciso que o *Kernel* seja compilado com suporte a V4L2³¹. O *Kernel* fornecido pela FriendlyARM já é compilado com essa opção.

Nem todas as câmeras USB disponíveis no mercado são compatíveis com UVC, por isso, é necessário pesquisar o *site* do projeto UVC, e verificar se o modelo desejado encontra-se na lista de dispositivos compatíveis. O *site* do projeto é: <http://www.ideasonboard.org/uvc/>.

Com exceção disso, essa aplicação também é de fácil manuseio, bastando um simples *Shell Script* [50] para controlar os parâmetros e automatizar o processo de *streaming*.

O único ponto negativo é a falta de suporte para a câmera CMOS, sendo necessário compilar um *driver* não-oficial para que a câmera CMOS seja reconhecida como uma câmera USB compatível com o padrão UVC e *driver* V4L2. Detalhes sobre o *driver* não-oficial podem ser vistos em: <http://code.google.com/p/mjpg-streamer-mini2440/>.

Configurado o *stream* de vídeo, basta utilizar algumas *tags* HTML e comandos URL para exibir o *stream* de vídeo como se fosse uma imagem na página.

³⁰ Abreviação de *USB Video Controller*, ou Controlador de Vídeo USB, em Inglês. Site do projeto: <http://www.ideasonboard.org/uvc/>.

³¹ Abreviação de *Video For Linux 2*.

3.4.6 Captura de Fotos

Para a captura de fotos, foi necessário utilizar a suíte OpenCV [65], de modo a evitar realizar demasiado trabalho no desenvolvimento de uma aplicação para captura de fotos, acabando por fazer o famoso jargão comum *Reinvenção da Roda*, em que se gasta tempo desenvolvendo algo que já está desenvolvido.

Para o uso da suíte OpenCV, foi necessário realizar a compilação cruzada para a arquitetura ARM das bibliotecas essenciais da suíte, além de escrever uma aplicação que faça a captura da imagem, e fazer a compilação cruzada dessa aplicação.

Com isso, de modo a demonstrar que o uso de OpenCV é possível em aplicações embarcadas, foi escrita uma aplicação que realiza a captura de imagens através da câmara USB, podendo tanto capturar imagens coloridas ou em preto-e-branco, salvando as imagens em formato JPG.

Como há só uma câmara USB, não é possível acontecer o *streaming* de vídeo e a tirada de fotos simultaneamente, pois ambos os programas reservam o dispositivo durante o seu uso. Para isso, bastou criar um *script* em *shell script*, que encerra o processo de *streaming* de vídeo, executa o programa para tirar foto, e após a foto ter sido tirada, o *script* reinicia o *streaming* de vídeo.

Esse *script* foi muito útil na funcionalidade de tirar fotos com uma temporização, fazendo uso do *crontab*³². Com isso, é possível tirar fotos do ambiente monitorado pela câmara USB a cada meia hora, a cada uma hora, dependendo de como for configurado no *crontab*.

Para o comando de disparo de fotos executado via *web*, foram escritas rotinas e funções em PHP que realizam as mesmas funções do *shell script* escrito, ou seja, paralisa o *stream* de vídeo, executa o programa para tirada de foto, e reinicia o *stream* de vídeo.

E para salvar as imagens tiradas pelo programa foi utilizado o diretório **/sdcard/pictures**, localizado no cartão de memória inserido na Tiny6410. Como esse diretório não é mapeado pelo *lighttpd*, as fotos armazenadas nele não irão aparecer na página *web*. Para contornar esse problema foi criado um *link* simbólico do diretório **/sdcard/pictures** para **/www/pictures**.

Maiores detalhes da aplicação serão fornecidos no Capítulo 4.9, que descreve em detalhes a aplicação escrita com a biblioteca OpenCV.

³² *Crontab* é um utilitário de sistemas Linux e Unix, o qual serve para automatizar a realização de tarefas por determinados períodos de tempo.

3.4.7 Aquisição de Temperatura Ambiente

O *kit* FriendlyARM Tiny6410 vem com um sensor de temperatura integrado, que é um sensor DS18B20, que fornece a temperatura em formato digital através do protocolo *1-wire*. Uma imagem da localização do sensor de temperatura e do esquemático de conexão do mesmo é mostrada na Figura 27.

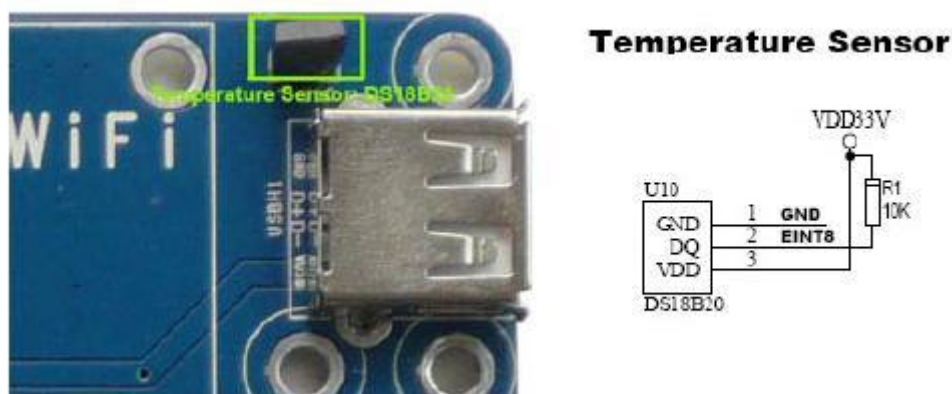


Figura 27 - Foto da localização e esquemático do sensor DS18B20 na Tiny6410. Fonte: www.minidevs.com

Basicamente, o funcionamento do sensor se resume ao seguinte: São enviados determinados comandos hexadecimais pelo barramento de dados (*1-Wire*), e, dependendo do comando, tal como **CONVERT**, associado a **44h** em hexadecimal, o transistor irá processar o comando e gravar 2 *bytes* de dados em sua memória, contendo a informação sobre a temperatura ambiente no momento. Para fazer com que o transistor envie Esses dois *bytes* de dados pelo barramento *1-Wire*, é preciso enviar o comando **READ SCRACHPAD**, associado a **BEh** em hexadecimal. Após esse comando, o sensor de temperatura irá enviar 9 bytes de dados.

Com a utilização de ferramentas de busca *web*, foi possível encontrar o fórum chinês CSDN [14], em que um usuário postou, liberado para qualquer tipo de uso, todos os códigos-fontes necessários para compilar o *driver* e uma aplicação modelo para o sensor de temperatura DS18B20 embutido na Tiny6410. Tanto o *driver* como a aplicação modelo foram escritos em Linguagem C.

Esses arquivos foram baixados e adaptados ao projeto, pois o *driver* precisa ser compilado com referência ao *Kernel* utilizado. Já a aplicação modelo foi modificada para gerar apenas uma saída em sua execução, sendo essa saída a leitura da temperatura em graus *Celsius* naquele instante, em formato texto.

Como essa aplicação gera uma simples saída em formato texto, para exibir a temperatura ambiente na página *web* do projeto basta executá-la com auxílio do PHP, trabalhar esse dado, e mostrá-lo na página *web*.

Além disso, foram agregadas mais algumas funcionalidades, tais como mostrar a temperatura em *Kelvin*, em graus *Fahrenheit*, e até mesmo a geração de um arquivo texto contendo as temperaturas em um dado intervalo de tempo em segundos.

Mais detalhes sobre a implementação dessa parte do projeto é fornecida no Capítulo 4.11.

3.4.8 Aquisição de Dados GPS

Juntamente com o *kit* didático, foi comprado um receptor de sinal GPS, chamado Gstar GS-87, o qual é mostrado na Figura 28.

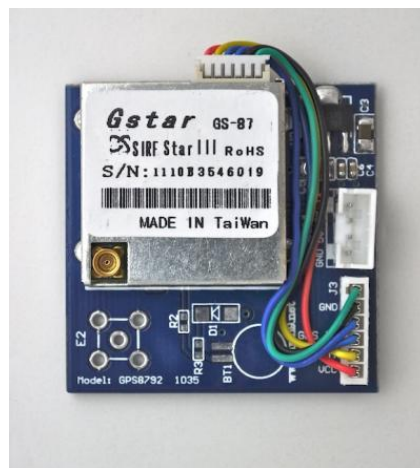


Figura 28 - Receptor de Sinal GPS. Fonte: www.andahammer.com

Esse receptor de sinal GPS faz uso do *chipset* SiRF StarIII, que suporta o padrão NMEA 0183 e as sentenças GGA, GSA, GSV, RMC e VTG, como comentado no Capítulo 2.

O receptor por si só mal consegue captar sinais GPS. Para isso, é usada uma antena especial, igual a antena mostrada na Figura 29.



Figura 29 - Antena receptora de sinal GPS

O receptor de sinal GPS precisa ser devidamente conectado ao *kit*, utilizando um cabo compatível, de modo a permitir o fornecimento de energia elétrica ao dispositivo, e também a comunicação serial com a placa.

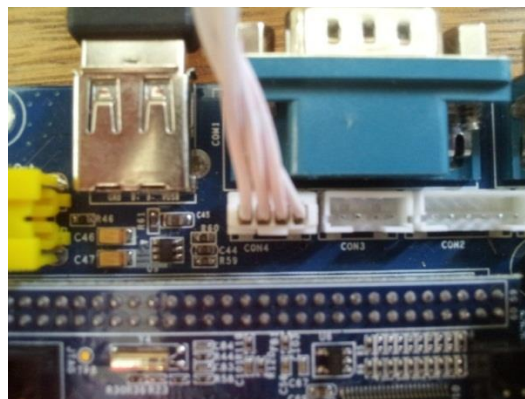


Figura 30 - Detalhe do conector, encaixado na porta COM4

No *kit* Tinyt6410 o receptor GPS foi conectado na COM4, identificada no Linux como **ttysAC3**. A conexão do cabo no *kit* é mostrada na Figura 30, e na Figura 31 é mostrado o receptor de sinal GPS próximo à Tiny6410, conectado ao cabo de alimentação e de transferência de dados, e conectado à antena receptora de sinais GPS.



Figura 31 - Receptor de Sinal GPS ligado na porta serial da Tiny6410, e também ligado à antena receptora

Na Figura 32, é possível ver como a antena receptora de sinais GPS foi colocada em uma das janelas do LAVISIM. Essa foi a melhor posição encontrada, em se tratando do local aonde o *kit* Tiny6410 deve ficar, considerando o comprimento do cabo da antena, e considerando o melhor local capaz de receber sinal GPS no laboratório.



Figura 32 - Posição da antena receptora de sinais GPS na janela do laboratório

Com base nas especificações do *datasheet* do receptor [79], ele envia dados GPS a uma velocidade de 9600 bps (bytesize 8, sem paridade, 1 stop bit) em sua configuração normal. Com posse da porta serial e da velocidade de conexão, basta uma aplicação, ou uma maneira, de obter e filtrar os dados.

É possível criar aplicações em Linguagem C para realizar essa tarefa, mas foi preferível realiza-la utilizando uma simples aplicação escrita em Python. A escolha se deu pelo fato de que é fácil lidar com sentenças de texto em Python, e, por se tratar de uma linguagem interpretada,

quaisquer modificações necessárias no código podem ser realizadas na própria máquina, sem necessidade de um *host* x86 para compilação cruzada.

Dessa forma, uma aplicação em Python foi criada de modo a receber as sentenças de dados GPS, e realizar o *parsing*³³ desses *streams*, permitindo filtrar qual sentença deseja ser mostrada, etc.

Juntamente com o Python, foi utilizado um *script* em PHP, que realiza a divisão e análise dos campos de cada sentença.

A versão do Python utilizada no projeto foi a versão 2.7, e maiores detalhes da implementação das funcionalidades GPS são mostrados no Capítulo 4.12.

3.4.9 Configuração de endereço IP e Gerenciador de DNS Dinâmico

Apesar de a placa possuir conector de rede Ethernet, o sistema Linux não está configurado para obter dados de rede automaticamente. E mesmo que estivesse, não são todas as redes que possuem essa funcionalidade ativada. Dessa forma, para determinar os endereços IP e DNS, necessários para conexão da Tiny6410 com a rede do LAVISIM, foi necessário modificar o arquivo **/etc/eth0-setting** com as seguintes informações de endereçamento:

```
IP=10.235.0.137
Mask=255.255.252.0
Gateway=10.235.0.1
DNS=143.107.182.2
MAC=00:01:02:03:05:13
```

Essas informações são carregadas pelo *script* **/etc/init.d/rcS** (mostrado no Anexo 8.4), assim que o sistema é inicializado. E as informações de endereçamento carregadas correspondem ao endereço IP 143.107.235.39, o qual pode ser acessado externamente à rede interna do LAVISIM.

Inicialmente, o acesso à placa se dava pelo seu puro e simples endereço IP. Apesar de isso não trazer nenhuma complicação para a realização do projeto, um dos objetivos principais foi o de fornecer um ambiente amigável ao usuário. E se o usuário já precisa decorar todo um endereço IP, isso deixa de ser amigável para uma grande parcela de pessoas.

³³ *Parsing* significa análise em português. É um termo muito utilizado em computação para fazer referência à técnica de encontrar trechos específicos ou padrões em textos ou outros tipos de arquivos.

Dessa forma, foi pensado e implementado o uso de um gerenciador de DNS dinâmico, que permite associar endereços mais memorizáveis, de fácil lembrança.

O melhor gerenciador encontrado foi o No-IP, que possui uma aplicação cliente em linguagem C, com o código-fonte aberto, o que torna possível a compilação cruzada para a plataforma do projeto.

Além disso, o No-IP permite que você tenha domínios gratuitos. Analisando a lista de opções, e após pensar sobre um *link* adequado, foi escolhido o endereço *armweb.no-ip.org* para a página do projeto.

A partir de então, qualquer usuário é capaz de acessar a página *web* do projeto através do endereço *armweb.no-ip.org*.

A aplicação No-IP pode ser configurada para verificar o endereço IP da máquina servidora a um dado intervalo de tempo. Já que a placa utilizada no projeto estava instalada em no laboratório LAVISIM da Engenharia Elétrica, possuindo um endereço IP estático associada a ela, esse recurso não se fez necessário.

A versão do cliente No-IP utilizado para Linux foi a versão 2.1.9.

3.4.10 Envio de E-mail

Infelizmente, o *rootfs* fornecido pela FrienlyARM não veio com o pacote *sendmail* [77], que é um pacote muito conhecido em sistemas Linux e Unix para o envio de *e-mail* através do terminal *Shell*.

Para contornar o problema, foram pensadas duas soluções. A primeira envolvia baixar e tentar fazer a compilação cruzada do pacote para a arquitetura ARM, já que o *Buildroot*, na versão utilizada, não possui o pacote **sendmail** para compilação automatizada. Essa até seria uma solução bem viável, mas o *Makefile* da aplicação era demasiado complicado, e dado o tempo que a adaptação do *Makefile* para compilação cruzada poderia tomar, essa solução foi descartada.

A segunda solução envolvia utilizar uma biblioteca do *Python* para envio de e-mail, que é a **smtplib**. Como tanto o Python e a biblioteca estavam na lista de pacotes disponíveis no *Buildroot*, essa foi a solução escolhida.

Para permitir que usuários enviassem *e-mails* ou mensagens de contato a partir da página do projeto, foi desenvolvido também dois *scripts* PHP: um para exibição do formulário

de envio, e outro para receber as mensagens, trata-las, e enviá-las ao *script* Python, que se encarregaria de enviar o *e-mail* com os comandos recebidos do PHP.

Maiores detalhes da implementação dessa parte são mostrados no Capítulo 4.16.

3.4.11 Envio de Mensagens por Parâmetros para Transmissão RF

Um dos objetivos do projeto foi o de também demonstrar a capacidade do sistema de operar com outros sistemas. E um dos sistemas presentes no Laboratório foi o **Sistema de RF**, fruto do Projeto de Iniciação Científica da aluna Patrícia Otoni, intitulado *Ferramentas para Gerenciamento de Conteúdo para Telemetria via Web com Linux Embarcado em Microcontroladores ARM*.

Esse sistema basicamente consiste de um par de placas SAM9-L9260 [63], que executam Linux em plataforma ARM e possuem uma interface de acesso *web*. Esse sistema é capaz de enviar e receber 3 quadros de 32 *bytes* de dados via radiofrequência com o uso de *transceivers*³⁴. Até o presente momento, o tipo de dado enviado é do tipo *string*. Na Figura 33 é possível enxergar o par de placas que compõem o **Sistema de RF**, e cada uma das placas está conectada ao *transceiver* de radiofrequência.

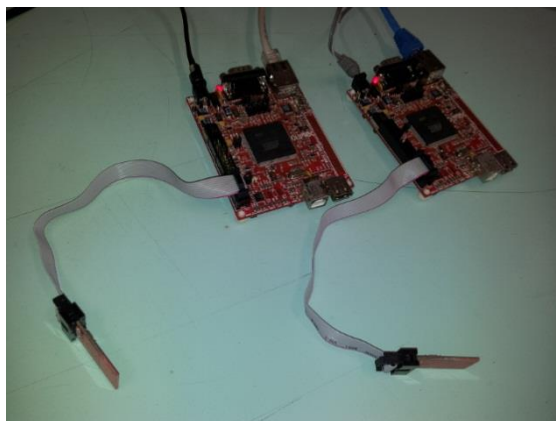


Figura 33 - Par de placas SAM9 com *Transceivers* de RF

O **Sistema de RF** já estava construído em cima da linguagem PHP, em um servidor *web* Apache [5]. No caso desse sistema, o PHP é responsável por obter as *strings* ou textos que o usuário quer enviar de uma placa para a outra, e executar uma aplicação que envia Essas *strings* via radiofrequência pelo *transceiver*. O PHP também era responsável pela parte de

³⁴ *Transceiver* significa transceptor em português. É um componente eletrônico capaz de enviar e receber dados via radiofrequência.

recebimento das mensagens enviadas, mostrando na tela do usuário, pela *web*, as mensagens recebidas.

Como as funções de envio e de recebimento de mensagens já estavam implementadas em código PHP, bastou um pequeno planejamento para permitir o acesso remoto a tais funções, através de passagem de parâmetros via GET, de modo a permitir o envio e recebimento de mensagens em radiofrequência na interface de acesso *web* da Tiny6410 utilizada nesse projeto.

Para permitir que a Estação *web* ARM *Web Media Station* pudesse enviar e receber dados das placas do **Sistemas de RF**, foi também criado um *script* em PHP, chamado **rf.php** e que pode ser visto no **Apêndice 9.1.8**. Além disso, foi criada uma tabela de endereços IP das estações, chamada **rf.sqlite**, utilizando o banco de dados SQLite3.

O *script* PHP consulta a tabela de endereços IP para obter a lista de endereços cadastrados, e mostra Essa lista ao usuário, que pode selecionar qual Estação (ou placa) deseja para enviar e/ou receber mensagens via radiofrequência.

E para enviar as mensagens digitadas pelo usuário a partir da interface *web* da ARM *Web Media Station*, foi utilizado o programa cURL [18]. O *script* **rf.php** recebe os dados digitados pelo usuário, processa Esses dados, verifica qual a máquina escolhida para enviar a mensagem, e utiliza o cURL para enviar a mensagem para a placa escolhida.

Caso o usuário queira apenas ver qual a última mensagem enviada em uma dada placa, o *script* **rf.php** verifica qual a placa escolhida pelo usuário, e utiliza o cURL para enviar a requisição de histórico à placa. O cURL recebe a resposta da requisição, a qual é lida pelo *script* PHP, tratada e sintetizada na página *web* para o usuário.

Maiores detalhes sobre a implementação dessa parte de radiofrequência são mostrados no Capítulo 4.13.

4 Desenvolvimento das Soluções e Resultados Obtidos

Após ter apresentado os objetivos do projeto, conceitos teóricos importantes, detalhes das aplicações e componentes escolhidos, e a relação entre eles para a composição do todo do sistema, o presente capítulo se dedica pura e exclusivamente à parte de desenvolvimento e implementação das soluções propostas.

Serão abordados detalhes de instalação e implementação, em forma de tutorial, como também detalhes de código e linguagem de programação, além das soluções que foram encontradas, e o resultado final obtido.

Todo o trabalho está embasado no *rootfs rootfs_qtopia_qt4-20111103.tgz*, fornecido pela FriendlyARM. De todo jeito, não é demasiado complicado criar e inserir um *rootfs*, como foi mostrado no Capítulo 3. Instruções bem detalhadas também podem ser encontradas na documentação fornecida pela FriendlyARM [30].

4.1 Servidor Web e processador de conteúdo PHP

4.1.1 Servidor Web Lighttpd

O pacote *lighttpd*, que foi o escolhido para ser o servidor de conteúdo *web* para o projeto, não vem instalado por padrão no *rootfs* da FriendlyARM. Dessa forma, precisamos compilar o pacote e transferi-lo para o sistema.

A forma mais prática de compilar o pacote é através do *Buildroot*, que automatiza todo o processo e já fornece os binários e bibliotecas compilados. Há também a opção de baixar o pacote no *site* do projeto [48], e realizar a compilação utilizando o *toolchain* fornecido pela FriendlyARM. Para isso, basta modificar os parâmetros do *Mak³⁵efile* do projeto, para que façam referência aos binários de compilação do *toolchain*.

Para compilar o pacote utilizando o *Buildroot*, tendo já realizado as configurações mostradas no Capítulo 3, é então necessário abrir o painel de configuração do *buildroot*. Para isso, é necessário estar na pasta do *buildroot*, e digitar o seguinte comando:

```
$ make menuconfig
```

Que dará início ao processo de criação do Menu de configuração do *Buildroot*. Dentro desse Menu encontra-se a opção **Package Selection for the target**, responsável por selecionar pacotes para serem compilados.

³⁵ *Makefile* é um arquivo que contém instruções e rotinas que auxiliam a compilação de arquivos.

Dentro dessa opção é preciso habilitar as caixas **BusyBox** e **Show Packages that are also provided by busybox**, conforme mostrado na Figura 34.

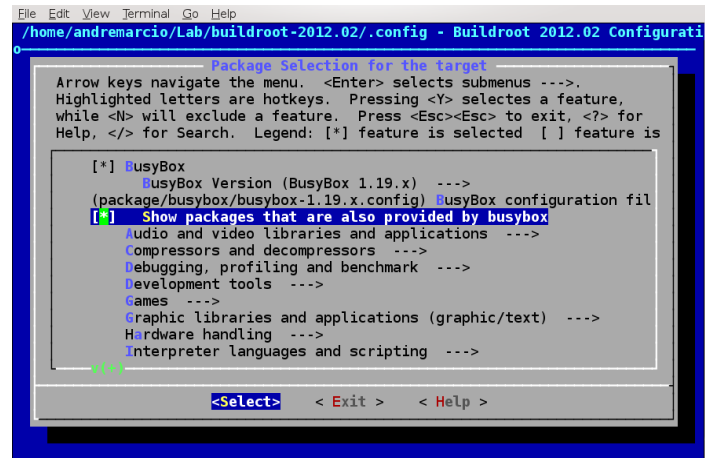


Figura 34 - Seleção do BusyBox - Buildroot

Após isso, na opção **Package Selection for the target** é selecionada a opção **Networking Applications**. Dentro dessa opção, há o pacote do *lighttpd*, marcado para instalação conforme mostrado na Figura 35.

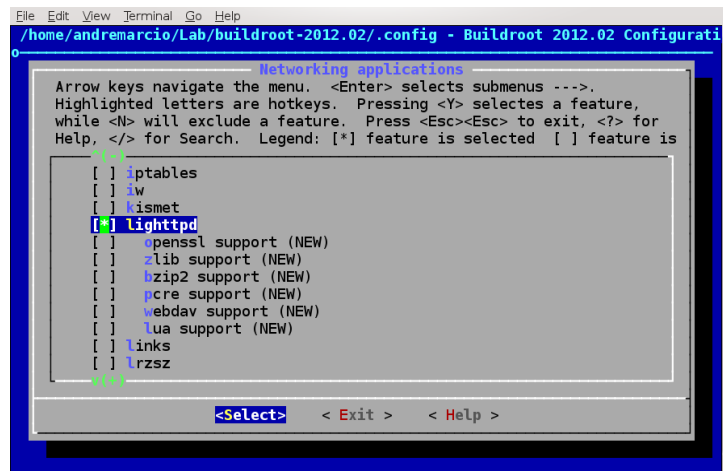


Figura 35 - Seleção do Lighttpd - Buildroot

O pacote é marcado para instalação, e o *Buildroot* é encerrado, salvando o arquivo durante a saída do programa.

Para dar início à compilação do *lighttpd*, foi digitado o comando **make** no diretório do *Buildroot*:

```
$ make
```

Com esse comando, o *Buildroot* começará a baixar os pacotes necessários, e fará toda a compilação dos pacotes e bibliotecas escolhidos.

Terminado o processo de compilação, foi utilizado o FileZilla, para transferir via FTP as pastas “*/usr/bin*” e “*/usr/lib*”, localizadas dentro do diretório **output/target** do *Buildroot*, para a pasta raiz “*/*” do sistema Linux da Tiny6410. Não foi preciso preocupar com conflitos de arquivos.

Com os binários no sistema de arquivos da Tiny6410, é possível inicializar o servidor. Mas antes disso, o *Lighttpd* só funcionará com um arquivo de configuração, que precisa ser criado. O arquivo de configuração **lighttpd.conf**, utilizado nesse projeto, pode ser visto no **Apêndice 8.3**. Recomenda-se colocar esse arquivo de configuração na pasta */etc* do sistema Linux em execução no *kit*.

Para inicializar o servidor, é preciso utilizar o seguinte comando, executado no terminal do Linux em execução na Tiny6410:

```
$ /usr/sbin/lighttpd -f /etc/lighttpd.conf
```

Para automatizar o processo de inicialização do *lighttpd*, basta acrescentar a seguinte linha de comando no final do arquivo */etc/init.d/rcS*, mostrado no Apêndice 8.4:

```
/usr/sbin/lighttpd -f /etc/lighttpd.conf
```

Feito isso, o *lighttpd* será executado com o arquivo de configuração toda vez que o sistema for inicializado.

4.1.2 Processador de Conteúdo PHP

O pilar principal da interface *web* do projeto, além do servidor *web*, é o PHP [69], que é capaz de processar as requisições *web* e retornar informações trabalhadas ou disparar ações na máquina servidora, com base nas requisições recebidas.

Um esquemático do funcionamento do PHP em conjunto com um servidor *web* pode ser visto na Figura 36. Nesse esquemático, dá pra perceber que o *browser* realiza requisições HTTP com o servidor *web*, e dependendo das requisições, o servidor irá encaminhá-las ao PHP. Esse, por sua vez, dependendo das extensões instaladas, realizará acesso a componentes do sistema de arquivos e de bancos de dados.

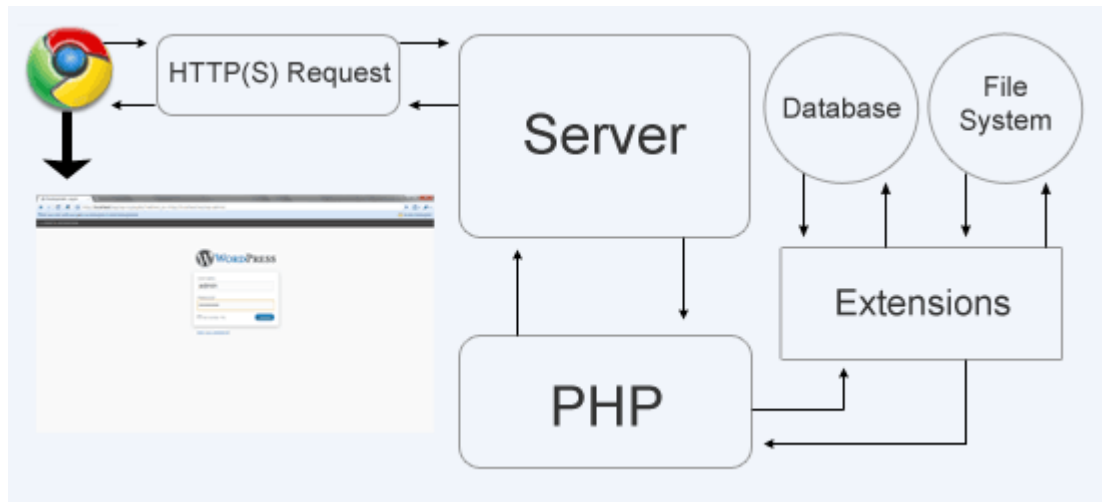


Figura 36 - Exemplo de Funcionamento do PHP. Fonte: www.sixrevisions.com

Para realizar a compilação cruzada dos binários e bibliotecas do PHP , é necessário novamente abrir o Menu de configuração do *Buildroot*, com o comando abaixo, lembrando que o mesmo deve ser digitado com o Terminal de comando na pasta **buildroot-2012.02**:

```
$ make menuconfig
```

Terminado o processo de síntese do Menu, é preciso selecionar a opção **Package Selection for the target**. Dentro desse item, é selecionado o subitem **Interpreter languages and scripting**. E dentro desse subitem são marcadas as seguintes opções:

- Php
- Dentro de **PHP interface**, são marcadas **cli and cgi interface**
- Fastcgi
- Todas as opções dentro de **PHP Extensions**

Um exemplo de como as opções devem estar é mostrado nas Figuras 37, 38 e 39.

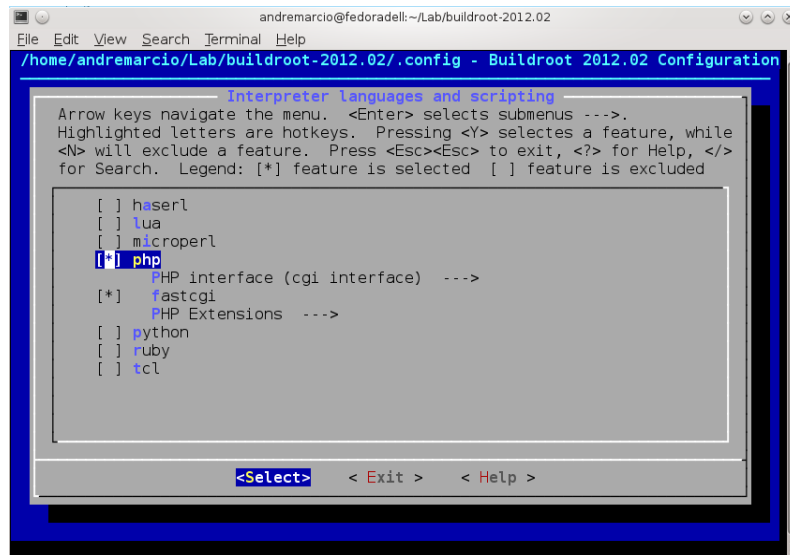


Figura 37 - Seleção dos pacotes e opções para PHP

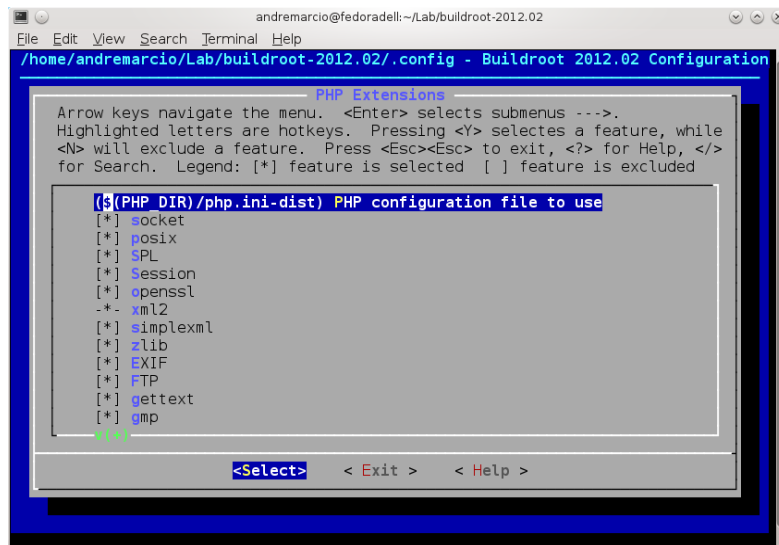


Figura 38 - Seleção das Extensões para PHP, Parte 1 - Buildroot



Figura 39 - Seleção das Extensões para PHP, Parte 2 - Buildroot

Após isso, o Menu de configuração do *Buildroot* é encerrado, e o arquivo de configuração é salvo.

Com o novo arquivo de configuração, é digitado o comando **make**:

```
$ make
```

Dado esse comando, o *Buildroot* irá começar a baixar os componentes e pacotes necessários para compilar o PHP. Além disso, o PHP será compilado com as opções e extensões que foram selecionadas no *Buildroot*.

Terminado o processo, foi utilizado o programa FileZilla [22] para transferir os binários e bibliotecas do PHP, com a cópia dos diretórios “*/usr/bin*” e “*/usr/lib*” da pasta “*output/target*” do *Buildroot* para a pasta raiz “*/*” do sistema Linux da Tiny6410. Não foi preciso preocupar com conflitos de arquivos.

É importante lembrar que o PHP irá funcionar em conjunto com o *lighttpd*. Isso só irá ocorrer se o *lighttpd* criar um *socket* de comunicação com o PHP sempre que receber requisições a arquivos com extensão *.php. E isto é definido no arquivo de configuração que o *lighttpd* recebe em sua inicialização.

Para testar se o PHP está funcionando corretamente, foi criado um arquivo modelo chamado **index.php**, deixando-o na pasta mapeada para arquivos *web*, que é a pasta */www*. Dentro desse arquivo, foi colocado o seguinte trecho de código:

```
<?php
phpinfo();
?>
```

Após salvar o arquivo, o mesmo foi acessado através de um *browser*, colocando o endereço IP ou de domínio associado à placa. Ao acessar o endereço, foi vista a tela de informações da versão do PHP utilizado, tal como mostrado na Figura 40.


PHP Version 5.2.17	
	
System	Linux FriendlyARM 2.6.38-FriendlyARM #14 PREEMPT Wed Nov 16 16:24:53 HKT 2011 armv6l
Build Date	Jun 21 2012 11:52:27
Configure Command	./configure '--target=arm-linux' '--host=arm-linux' '--build=86_64-unknown-linux-gnu' '--prefix=/usr' '--exec-prefix=/usr' '--sysconfdir=/etc' '--program-prefix=' '--disable-gtk-doc' '--enable-static' '--enable-shared' '--mandir=/usr/share/man' '--infodir=/usr/share/info' '--disable-all' '--without-pear' '--with-config-file-path=/etc' '--localstatedir=/var' '--enable-ctype' '--enable-cgi' '--enable-fastcgi' '--enable-sockets' '--enable-posix' '--enable-spl' '--enable-session' '--with-openssl=/home/andremarcio/Lab/buildroot-2012.02/output/host/usr/arm-unknown-linux-gnueabi/sysroot/usr' '--enable-libxml' '--with-libxml-dir=/home/andremarcio/Lab/buildroot-2012.02/output/host/usr/arm-unknown-linux-gnueabi/sysroot/usr' '--enable-xml' '--enable-xmlreader' '--enable-xmlwriter' '--enable-simplexml' '--with-zlib=/home/andremarcio/Lab/buildroot-2012.02/output/host/usr/arm-unknown-linux-gnueabi/sysroot/usr' '--enable-uuid' '--enable-ftp' '--with-gettext=/home/andremarcio/Lab/buildroot-2012.02/output/host/usr/arm-unknown-linux-gnueabi/sysroot/usr' '--with-gmp=/home/andremarcio/Lab/buildroot-2012.02/output/host/usr/arm-unknown-linux-gnueabi/sysroot/usr' '--enable-pcntl' '--enable-pthreads' '--enable-sysvmsg' '--enable-sysvsem' '--enable-sysvshm' '--enable-zip' '--enable-filter' '--enable-calendar' '--with-pcre-regex' '--enable-pdo' '--with-pdo-sqlite=/home/andremarcio/Lab/buildroot-2012.02/output/host/usr/arm-unknown-linux-gnueabi/sysroot/usr'
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc
Loaded Configuration File	/etc/php.ini
Scan this dir for additional ini files	(none)
additional ini files parsed	(none)
PHP API	20041225
PHP Extension	20060613

Figura 40 - Tela de versão do PHP

Após isso, *scripts* PHP podem ser criados para responder a requisições do tipo GET e POST. Requisições GET são as mais comuns e mais fáceis de manipular e usar. Requisições POST são inseridas nos cabeçalhos de pacotes HTTP, o que esconde os dados trafegados, tornando esse tipo de requisição mais apropriado para o tráfego de dados sigilosos ou confidenciais.

Para o tráfego de requisições de dados de controle e de informações gerais, serão usadas requisições GET. Para o tráfego de informações de autenticação de usuário, serão usadas requisições POST.

Para passar algum parâmetro para um *script* PHP em um servidor qualquer, através de um *browser*, basta seguir o seguinte padrão:

www.paginaweb.com.br/script.php?primeiroParametro=valor&segundoParametro=valor&...

Em que www.paginaweb.com.br é um endereço de domínio fictício associado a um servidor *web*, e **script.php** é um nome genérico para um *script* PHP. Após o nome do *script*, o

primeiro parâmetro, *primeiroParametro* deve vir precedido por “?”. Qualquer valor atribuído ao parâmetro deve vir seguido de “=” e o valor desse parâmetro. Após o primeiro parâmetro, o delimitador de parâmetro passa a ser o caractere “&”.

Essas variáveis precisam ser devidamente tratadas no *script* PHP, caso contrário, nenhuma resposta será gerada. Vejamos o seguinte código:

```
<?php
if(isset($_GET['hello'])){
    $msg = $_GET['hello'];
    echo $msg;
}
?>
```

No caso desse código, caso o *script* PHP seja composto por pura e unicamente dessa sequência, ele irá retornar ao usuário os valores que forem atribuídos ao parâmetro **hello** via método GET. No caso de uma requisição tal qual:

www.paginaweb.com.br/helloPHP.php?hello=Oi

O *script* fará o PHP retornar somente “Oi” na requisição, e será somente isso que o usuário irá enxergar na tela, ou que uma aplicação receberá ao realizar Essa requisição.

A partir desses simples exemplos, é possível então gerar *scripts* mais complexos, capazes de responder a requisições específicas sintetizando páginas HTML, capazes de executar aplicativos na máquina servidora com base em requisições também específicas, etc.

4.2 Página web

A página *web* foi desenvolvida fazendo uso de *scripts* PHP juntamente com códigos HTML [90], com componentes de estilo CSS [89] e controle por JavaScript [92], fazendo uso da biblioteca jQuery [47], que facilita muito o manuseio de componentes DOM, do HTML, além de facilitar muito a criação de rotinas de requisições AJAX.

Não é o foco do projeto ensinar os conceitos de *webdesign*³⁶ e *webdevelopment*³⁷, porém, serão apresentados alguns comandos e boas técnicas aprendidos e bem necessários para se criar uma interface amigável e uma boa interface de usuário.

Além das linguagens citadas, o PHP foi essencial para processar conteúdo *web*, tanto de maneira geral, como para usuários devidamente autenticados no sistema.

A página *web* do projeto, muitas vezes também citada como **interface web**, é o principal mecanismo de acesso às funcionalidades do sistema, tais como execução de músicas e vídeos, visualização de *stream* de vídeo de câmeras, envio de mensagens, etc.

A implementação dos arquivos *web*, em se tratando dos arquivos em formatos ***.php**, ***.js** e ***.css** foi toda realizada através do Aptana Studio 3 [6]. Ele foi configurado para acessar o diretório **/www** como diretório raiz, via FTP a Tiny6410.

Toda a estrutura de arquivos da página *web* está dentro do diretório **/www**, do sistema de arquivos do Linux da Tiny6410. Dentro desse diretório, temos os seguintes diretórios e suas respectivas funções:

- **css:** É o diretório aonde ficam os arquivos de estilo CSS.
- **db:** Diretório aonde ficam as bases de dados SQLite utilizadas.
- **files:** É o diretório aonde se encontram alguns arquivos e *scripts* utilizados no projeto. Foram inseridos nesse diretório para facilitar o acesso por *scripts* PHP.
- **pictures:** É um *link* simbólico para o diretório **/sdcard/images**, aonde se encontram imagens tiradas pela câmera USB através da aplicação desenvolvida em OpenCV.
- **images:** É o diretório que contém imagens e ícones utilizados na página *web*.
- **js:** É o diretório aonde se encontram os *scripts* JavaScript escritos para as funcionalidades AJAX e demais funcionalidades de interface da página *web*.
- **pages:** Nesse diretório ficam todas as páginas responsáveis pela exibição de conteúdo na coluna central da página *web*, as quais são mostradas conforme o usuário seleciona a opção desejada no Menu lateral.
- **php:** É o diretório aonde ficam os *scripts* PHP responsáveis por tratar requisições específicas e realizar ações, tais como tocar músicas, autenticar usuário em sessão, obter dados GPS, etc.

³⁶ *Webdesign* é um termo que designa a tarefa de trabalhar em cima do desenvolvimento visual, do *layout* da interface de usuário de um *web site* ou aplicação *web*.

³⁷ *Webdevelopment* é um termo em inglês que designa a tarefa de trabalhar em cima dos códigos, plataformas e serviços que compõem o *web site* ou aplicação *web*.

Arquivos pertinentes a cada um desses diretórios, conforme as funções realizadas, serão melhor abordados nas próximas sessões, conforme cada funcionalidade implementada é comentada.

Com o objetivo de desenvolver o *layout* planejado para o projeto, mostrado na Figura 26, foi criado um arquivo principal, o **index.php**, o qual é mostrado por completo no Apêndice 9.3.1. Esse arquivo é praticamente uma espécie de esqueleto, o qual possui toda a estrutura HTML da página, e é o arquivo que já determina quais *scripts* JavaScript que devem ser carregados pelo *browser web* ao carregar a página *web*, e também instancia algumas variáveis de ambiente definidas pelo arquivo **php/variables.php**, mostrado no Apêndice 9.1.11.

O **index.php** é o arquivo que determina a estrutura dos Menus, a estrutura da coluna central que exhibe as páginas desenvolvidas, e a coluna de *status* e de controle multimídia. É o arquivo que faz referência ao *script* de estilos CSS **style.css**, mostrado no **Anexo 8.5** Além disso, a estrutura do arquivo e do projeto já está preparada para permitir idiomas português e inglês.

Dentro do diretório **/www/pages** temos as seguintes páginas, e suas funcionalidades:

- **home.php**: Página *Inicial* do projeto. Contém breve descrição do projeto.
- **musicas.php**: Página *Músicas*, responsável pela exibição de arquivos e diretórios de músicas.
- **vídeos.php**: Página *Vídeos* responsável pela exibição de arquivos e diretórios de vídeos.
- **câmera.php**: Página *Câmera*, responsável por mostrar os *streams* de vídeos obtidos através do MJPG Streamer, além de possuir controle para captura de imagens.
- **images.php**: Página *Imagens*, responsável pela exibição tabelada das imagens tiradas pela câmera USB.
- **gps.php**: Página *GPS*, responsável pela exibição de dados GPS.
- **temp.php**: Página *Temperatura*, responsável pela exibição de dados da temperatura ambiente.
- **rf.php**: Página *RF*, responsável pelo controle de envio e recebimento de mensagens de radiofrequência.
- **message.php**: Página *Mensagem* para envio de mensagens ao *display* LCD da Tiny6410.
- **docs.php**: Página *Documentos*, que contém documentação e *links web* de referência e bibliografia auxiliar.
- **admin.php**: Página *Administração*, que possui detalhes da Estação e controles administrativos.

- **about.php:** Página *Sobre*, que contém maiores detalhes sobre as funcionalidades de cada uma das partes do Menu.
- **contact.php:** Página *Contato*, possui formulário para contato através de *e-mail*.

O foco dessa parte inicial é primeiramente mostrar como foi pensado a estrutura de Menus e de exibição das páginas criadas. Posteriormente, cada parte será melhor detalhada, a saber, o painel de controle e de *status*, a tela de *login*, etc.

Quando algum usuário digita o endereço IP (ou domínio) da Tiny6410, quem vai responder Essa requisição é o *lighttpd*. E de acordo com seu arquivo de configuração, ele vai procurar arquivos de nomes **index** com as extensões ***.htm**, ***.html** e ***.php**. Como não foram criados arquivos **index** com as primeiras duas extensões, o primeiro arquivo a ser enviado para o cliente é o **index.php**. E também, de acordo com o arquivo de configuração do *lighttpd*, arquivos com a extensão ***.php** estão associados ao PHP, o que fará o *lighttpd* passar qualquer requisição a arquivos com extensão ***.php** para o interpretador PHP. Dessa forma, o interpretador PHP irá pegar o arquivo **index.php**, processá-lo conforme as variáveis de ambiente e de sessão, e retornar a página produzida para o computador cliente.

Assim, ao processar o *script* **index.php**, uma das primeiras coisas que o interpretador irá fazer será iniciar uma sessão e carregar o arquivo de variáveis de execução, como mostrado no trecho abaixo, retirado do *script* **index.php**:

```
<?php
session_start();

include '/www/php/variables.php';
...
```

Depois desse trecho, estão algumas rotinas que servem de molde para modificações futuras que permitirão a mudança de idioma da página. Quando o *script* começa a enviar conteúdo HTML para o *browser*, um dos trechos principais é o seguinte:

```
<script type="text/javascript" src="js/jquery-1.7.2.min.js"></script>
<script type="text/javascript" src="js/ajax.js"></script>
<script type="text/javascript" src="js/rf.js"></script>
<script type="text/javascript" src="js/gps.js"></script>
<script type="text/javascript" src="js/temp.js"></script>
<script type="text/javascript" src="js/mediaControl.js"></script>
<script type="text/javascript">
```

```

$(document).ready(function(){

    loadStatus();
    getCpuLoad();
    loadTemp("c");
    mpStatus();
    initAjax();

});

```

Esse trecho define os arquivos JavaScript que serão carregados no *browser*. Cada um desses arquivos desempenha funções importantes no controle de funcionalidades AJAX (arquivo **js/ajax.js**), envio de requisições multimídia (arquivo **js/mediaControl.js**), controle de requisições de GPS (arquivo **js/gps.js**), Temperatura (arquivo **js/temp.js**) e Radiofrequência (arquivo **js/rf.js**). A função jQuery (“\$(document).ready()”) define algumas funções que devem ser carregadas assim que todo o documento HTML já ter sido carregado no *browser*.

As funções chamadas pela função jQuery estão implementadas no arquivo **js/ajax.js**, mostrado no Apêndice 9.2.1. A função **loadStatus()** requisita o *status* do *stream* de vídeo, se está em funcionamento ou não. A função **mpStatus()** é semelhante, foi faz uma requisição ao servidor para saber o *status* do MPlayer, se está em execução ou não. A função **getCpuLoad()** faz requisição para obter a carga da CPU da Estação, a função **loadTemp()** requisita a temperatura ambiente em graus Celsius, e a função **initAjax()** inicia uma rotina que determina a realização dessas funções de modo temporizado, mostrando Essas informações a cada intervalo de tempo. Detalhes a respeito da implementação dessas funções serão dados nas próximas sessões, conforme forem abordadas as funcionalidades implementadas.

Ainda com relação ao arquivo **index.php**, é mostrada a seguinte estrutura, responsável por sintetizar os dados que irão compor o Menu da página:

```

<div id="menu" align="center">
    <a class="active" href=<? echo "./?p=home";?> ><? echo $Home;?></a>
    <a href=<? echo "./?p=musicas";?>><? echo $Musicas;?></a>
    <a href=<? echo "./?p=videos";?> ><? echo $Videos;?></a>
    <a href=<? echo "./?p=camera";?> ><? echo $Camera;?></a>
    <a href=<? echo "./?p=images";?> ><? echo $Imagens;?></a>
    <a href=<? echo "./?p=gps";?> ><? echo $GPS;?></a>
    <a href=<? echo "./?p=temp";?> ><? echo $Temp;?></a>

```

```

<a href=<? echo "./?p=rf";?> ><? echo $RF;?></a>
<a href=<? echo "./?p=message";?> ><? echo $Mensagem;?></a>
<a href=<? echo "./?p=docs";?> ><? echo $Docs;?></a>
<a href=<? echo "./?p=admin";?> ><? echo $Admin;?></a>
<a href=<? echo "./?p=about";?> ><? echo $Sobre;?></a>
<a href=<? echo "./?p=contact";?> ><? echo $Contato; ?></a>
</div>

```

Essa estrutura está contida na *div* HTML de *id* “menu”, com alinhamento central. A forma como foram organizados os *divs* na página **index.php** é que permitiu esse *div* em particular ficar posicionado na coluna mais à esquerda. Como visto também, o Menu faz uso da *tag* HTML “<a>” com parâmetro “href” para determinar um *link* a ser requisitado, ao clicar naquela palavra ou texto. Esse *link* é uma requisição ao próprio **index.php**, por conta da forma como os *links* foram definidos, que é o padrão “./?p=página”. Com isso, serão feitas requisições ao *script* **index.php** com parâmetro “p” para determinar qual a página que deve ser carregada na coluna central. Os nomes de cada item do Menu são obtidos do arquivo de variáveis, o **php/variables.php**, que foi carregado no início do *script*.

Com base no que foi estruturado para o controle dos Menus, temos o *div* HTML responsável pela coluna central, de *id* “content”, a qual deve exibir o conteúdo das páginas ao usuário, que é o seguinte:

```

<div id="content" align="center">
    <?php
        if(!isset($_GET['p'])){
            include('pages/home.php');
        } else
            include('pages/'.$_GET['p'].'.php');
    ?>
</div>

```

O que esse trecho determina é que, caso não tenha disso instanciado o parâmetro “p” por alguma requisição GET, responsável por transmitir qual a página desejada ao **index.php**, a página que será carregada é a **pages/home.php**. No caso de o usuário clicar em algum dos itens do Menu, esse item irá disparar uma requisição para o **index.php** com a página desejada. E a função **include()** irá instanciar o arquivo ***.php** naquela região do arquivo, o que resultará na exibição da página selecionada pelo usuário.

Dessa forma, ao acessar a página, será inicialmente carregado o conteúdo da **pages/home.php**, que é a página inicial. Conforme o usuário seleciona outras opções no Menu lateral, outros conteúdos serão exibidos na coluna central.

Além do Aptana Studio 3, que foi utilizado para codificação e *upload* dos arquivos necessários ao projeto, foi muito utilizado o *plugin FireBug*³⁸ do *browser Mozilla Firefox*³⁹. Esse *plugin* permite *debuggar*⁴⁰ as requisições *web* e o funcionamento dos *scripts* JavaScript. Ele permite também verificar o tempo que cada componente leva para ser carregado no *browser*.

É possível ver na Figura 41 um exemplo de como ficou a página principal completa da *web site* desenvolvido. Como visto, nessa figura são mostradas as três colunas, a coluna de Menu, a coluna central, de conteúdo, e a coluna de *status* e controle multimídia.



Figura 41 - Imagem da tela principal do *website* desenvolvido.

³⁸ Famoso *debugger* de páginas e aplicações *web*. Página do projeto: <http://getfirebug.com/>

³⁹ Um dos *browsers web* mais comuns em uso. Página do projeto: <http://www.mozilla.org/en-US/firefox/new/>

⁴⁰ *Debuggar* é um estrangeirismo derivado da palavra em inglês *debug*, que significa o ato de verificar o funcionamento de um programa.

4.3 Configuração de IP e DNS Dinâmico

Para ajustar o endereço IP da Tiny6410 conectada à rede do LAVISIM, foi preciso editar o arquivo `/etc/eth0-settings` com as seguintes configurações:

Após isso, basta reiniciar o sistema, com o seguinte comando:

```
# reboot
```

Com esse comando, o sistema Linux da Tiny6410 será reinicializado. Com o sistema novamente inicializado, verifique se o mesmo responde no endereço IP configurado com o seguinte comando:

```
$ ping "endereço ip da Estação"
```

Se houver resposta, a configuração ocorreu com êxito.

Para melhorar o acesso à Estação, foi pensado no serviço de DNS gratuito No-IP, que permite a associação gratuita de domínios **no-ip.org** a endereços IP fixos ou dinâmicos.

Para o serviço de DNS dinâmico e atribuição de domínio, foi necessário proceder com as seguintes tarefas:

1. Criar um registro (*login*⁴¹) no site www.no-ip.com, que permite associar endereços de domínios para endereços IP dinâmicos ou estáticos.
2. Compilar e transplantar o binário da aplicação cliente para a Tiny6410.

De modo a gerar o binário da aplicação No-IP, é possível tanto realizar de modo manual, quanto de modo automatizado com o *Buildroot*. Vale lembrar que o aplicativo foi incluído na lista de pacotes do *Buildroot* em sua versão 2012.02.

Para compilar utilizando o *Buildroot*, foi selecionada a opção **Package Selection for the target**, no Menu principal do Buildroot, e depois a opção **Network applications**. Nessa opção, foi selecionado o pacote **noip** como mostrado na Figura 42. Depois, o *Buildroot* foi encerrado, salvando as configurações. Por fim, comando **make** foi usado para iniciar o processo de compilação.

⁴¹ *Login* é uma expressão em inglês que denota o ato de possuir registro ou de realizar o registro.



Figura 42 - Seleção do pacote noip no Buildroot

Terminado o processo, foi transferido o conteúdo das pastas “/usr/bin” e “/usr/lib”, que se encontram dentro do diretório “/output/target/” do *Buildroot*, para a raiz “/” do sistema na Tiny6410.

Com o binário e as bibliotecas necessárias no sistema, é necessário configurar o No-IP para funcionar e associar o endereço IP da Estação a um domínio configurado no site www.no-ip.com.

O pacote **noip** não vem com um arquivo de configuração, sendo necessário criar uma configuração em sua primeira execução. Isso é possível através do seguinte comando:

```
$ noip2 -C
```

Após esse comando, serão pedidos os dados de *login* e informações necessárias para associação do endereço IP ao domínio. Para o projeto, o endereço *armweb.no-ip.org* foi configurado da maneira mostrada na Figura 43.

```

armweb.no-ip.org - PuTTY
root@armweb.no-ip.org's password:
[root@FriendlyARM /]# noip2
Mail/             home/             obs3.txt           tmp/
bin/              lib/              opt/              usbCapture/
cmd_lcd.txt       linuxrc           proc/             usr/
dev/              mjpg-streamer/   root/             var/
etc/              mnt/             sbin/             www/
fa-network-service obs.txt           sdcard/
gps/              obs2.txt          sys/
[root@FriendlyARM /]# noip2 -C

Auto configuration for Linux client of no-ip.com.

Please enter the login/email string for no-ip.com  andre.ml.curvello@gmail.com
Please enter the password for user 'andre.ml.curvello@gmail.com'  *****

Only one host [armweb.no-ip.org] is registered to this account.
It will be used.
Please enter an update interval:[30] 30
Do you wish to run something at successful update?[N] (Y/N)  n

New configuration file '/etc/no-ip2.conf' created.

[root@FriendlyARM /]#

```

Figura 43 - Exemplo de configuração do NoIP em ambiente Linux.

O comando “\$ noip -C” só cria o arquivo de configuração. Depois é preciso colocar o **noip** em execução, o que pode ser feito pelo seguinte comando:

```
$ noip2 &
```

O caractere “&”no final do comando serve para fazer o programa rodar em *background*⁴², e assim o terminal fica livre para demais comandos necessários.

Para fazer com que o **noip2** seja executado automaticamente com a inicialização do sistema Linux da Tiny6410, é preciso acrescentar a seguinte linha no final do arquivo “/etc/init.d/rcS”:

```
noip2
```

Feito isso, o **noip2** será carregado sempre que o sistema for inicializado, e fará verificação do endereço IP da Estação a cada meia hora (ou 30 minutos).

No caso da Tiny6410 instalada no LAVISIM, apenas a associação de endereço IP configurada no painel de usuário do www.noip.com bastou para a associação de IP, já que o endereço associado à Estação é fixo, e não dinâmico. Mas para alguns projetos, é preciso usar a configuração para IP dinâmico, o qual pode ser mudado pela provedora de *internet*.

⁴² *Background*, significa segundo plano ou fundo. Em termos de computação, um programa é dito como executando em *background* quando sua execução não é visível ou perceptível ao usuário.

4.4 Sessão e variáveis de Sessão

Inicialmente, não foi criado um mecanismo de autenticação de usuário, e de diferenciação de acesso para usuários registrados e não registrados.

Porém, conforme se percebeu a necessidade de que haviam conteúdos de acesso mais restrito, como os *streams* de vídeos que mostram laboratórios e pessoas nesses laboratórios, foi então implementada uma camada de segurança para proteger esse conteúdo, e garantir seu acesso somente a usuários cadastrados no sistema.

Para garantir que as informações de *login* de um usuário sejam vistas por diversos *scripts* em execução no interpretador PHP, existem as variáveis de sessão, que servem para armazenar conteúdo somente quando os arquivos de *script* estão em execução. Porém, para utilizar Essas variáveis de sessão em mais de um *script* PHP, é preciso definir uma sessão no *script*, o que é feito com a seguinte linha de código PHP:

```
session_start();
```

Essa linha deve ser inserida antes de utilizar qualquer variável de sessão, e também antes de qualquer conteúdo ser enviado ao *browser* cliente, seja texto ou código HTML.

E para utilizar variáveis de sessão, basta seguir o seguinte padrão:

```
$_SESSION['variável'] = "conteúdo";
```

Dessa forma, se tivermos dois arquivos PHP, **exemplo1.php** e **exemplo2.php**, e em ambos os arquivos é iniciada uma sessão com a linha **session_start()**, variáveis instanciadas em um arquivo podem ser acessadas no outro. Ou seja, se tivermos a seguinte sequência de código no **exemplo1.php**:

```
<?php
session_start();
    $_SESSION['teste'] = "Oi!";
?>
```

E a seguinte sequência de código no arquivo **exemplo2.php**:

```
<?php
session_start();
echo $_SESSION['teste'];
```

?>

Ao ser executado o **exemplo1.php** a variável de sessão “teste” será instanciada com o valor “Oi!”. E ao ser executado o **exemplo2.php**, será mostrado o valor dessa variável de sessão, que é “Oi!”.

Essa técnica foi utilizada para permitir que, após o usuário ter realizado seu *login*, ele seja visto como autenticado em toda a página *web*. Essa técnica também foi utilizada para definir variáveis de execução para *scripts* tais como **mplayer.php** e demais, que dependem de variáveis que são definidas conforme as ações do usuário.

No Capítulo 4.10, de Banco de Dados e Autenticação, são fornecidos maiores detalhes dos mecanismos associados à autenticação de usuários.

Para maiores informações a respeito do uso de sessões com PHP, recomenda-se a leitura da página *web* <http://www.php.net/manual/en/book.session.php>.

4.5 Exibição de conteúdo multimídia

Para a exibição de conteúdo multimídia, seja ele música, vídeo ou imagens, é preciso mostrar ao usuário os arquivos disponíveis, e permitir ao usuário escolher o arquivo que deseja ver ou executar.

Para alcançar esse objetivo, o PHP foi essencial, pois possui funções capazes de mapear os arquivos de um dado diretório, e juntamente com rotinas de processamento de texto, é capaz de produzir conteúdo customizado ao usuário. Duas funções capazes de mapear diretórios em PHP são:

- **scandir(string \$diretório)**: Recebe um diretório como parâmetro e retorna um vetor contendo todos os arquivos e pastas (também diretórios) desse diretório parametrizado. Maiores detalhes podem ser obtidos em <http://php.net/manual/en/function.scandir.php>.
- **realpath(string \$path)**: Recebe um arquivo ou diretório do sistema como parâmetro e retorna o caminho completo para esse arquivo ou diretório. Maiores detalhes podem ser obtidos em <http://php.net/manual/en/function.realpath.php>.

Um exemplo desse tipo de aplicação é o trecho mostrado abaixo, que está devidamente comentado. Basicamente, é uma rotina que recebe um parâmetro através de uma requisição do tipo GET, e gera uma página de retorno para o usuário.

```

$dir = realpath($chdir); //A variável $dir recebe o caminho completo de
$chdir.
chdir($dir); //O diretório do interpretador desse script PHP é definido
como o diretório recebido.
echo "<p>Diretório: $dir</p>"; //O diretório atual é mostrado ao usuário.
$files = scandir($dir); //Todos os arquivos do diretório são mapeados no
vetor.

if(is_dir($files[1])){ //Verifica se o item do vetor é um arquivo
    $retDir = realpath($files[1]); //Se for, $retDir recebe o caminho
complete do arquivo.
    if(strcmp('/sdcard', $retDir)==0){} //Se for o diretorio /sdcard, nao
faz nada, diretorio protegido.
    else if(strcmp('/sdcard', $retDir)<0)
        echo "<br/><input type=\"button\" value=\"Voltar\"
onclick=\"chdir('$retDir', 'md');\"/><br/>";
    }
}

```

O PHP também possui funções que permitem verificar se um item mapeado é realmente um arquivo ou um diretório, e há até mesmo funções que extraem todo tipo de informação de um item mapeado, tal como diretório base, nome do arquivo, e sua extensão. Tais funções serão abordadas e exemplificadas. Como visto no código acima, duas funções que realizam Essas ações são:

- `bool is_dir(string $filename)`: Retorna verdadeiro (*true*) se o arquivo existir e fazer referência a um diretório, e falso (*false*) caso contrário. Maiores referências em: <http://php.net/manual/en/function.is-dir.php>
- `pathinfo(string $path)`: retorna informações diversas sobre o caminho (*path*) informado, tais como diretório do arquivo, nome do arquivo e extensão do arquivo. Maiores informações em: <http://php.net/manual/en/function.pathinfo.php>

O código completo do *script foldernav.php*, responsável pelas funções de mapeamento de músicas, imagens e vídeos é mostrado no Apêndice 9.1.4.

O *script foldernav.php* percorre os diretórios informados a ele, e retorna uma página ao usuário seguindo o seguinte padrão: Diretórios são exibidos com letras em cor Azul, e arquivos, em letras na cor Preta. Se o usuário clicar em um diretório, esse novo diretório será passado para o *script*, que novamente irá sintetizar uma nova página de retorno, com base da estrutura desse novo diretório. E se o usuário clicar em um arquivo, aí será disparada uma requisição de execução de programa, passando como parâmetro de entrada ao programa o caminho completo do arquivo no sistema.

Infelizmente, foi percebido que uma mesma saída gerada ao usuário seria insuficiente para permitir o controle de música, vídeos e imagens. Ou seja, só mostrar os arquivos existentes em uma dada pasta, diferenciando os diretórios de arquivos propriamente, era insuficiente para permitir maior interatividade e controle.

Isso foi concluído pelo simples fato de que, para permitir controle de programas na máquina servidora através do PHP, são necessários parâmetros específicos, não somente o caminho completo do arquivo. O MPlayer, programa usado para executar vídeos e músicas, não executa imagens. E, para executar músicas, não precisamos definir qual a saída de vídeo desejada. Já para executar vídeos, é preciso definir a saída desejada.

Dessa forma, o *script* **foldernav.php** foi modificado para mapear três tipos de diretórios: Diretórios que contenham outros diretórios e imagens, diretórios que contenham outros diretórios e músicas, e diretórios que contenham outros diretórios e vídeos.

Por motivos de segurança, e para evitar que o usuário que acessar o sistema pela *web* tenha acesso somente ao conteúdo multimídia, as rotinas de navegação fazem uma checagem se o diretório passado como parâmetro é **/sdcard/**, caso contrário, não gerará uma página para o usuário.

Basicamente, o que acontece é que o *script* **foldernav.php** recebe parâmetros passados via requisições GET que ajudam a definir se o diretório a ser mapeado e exibido é para imagens, vídeos ou músicas. Com base nesses parâmetros, o *script* sintetiza uma página, e Essa página gerada faz uso de funções em JavaScript do arquivo **js/mediaControl.js** (que pode ser visto no Apêndice 9.2.3) para permitir o disparo de requisições com base em opções e arquivos selecionados.

O *script* **foldernav.php** só responde a três tipos de requisições:

1. **foldernav.php?img&chdir=diretório**: Esse tipo de requisição define o mapeamento de um diretório de imagens, por causa do parâmetro **img** passado ao *script* **foldernav.php**, e define o diretório a ser mapeado com esse perfil através do parâmetro **chdir** e do atributo *diretório*.

Além disso, a página gerada com esse mapeamento retorna uma tabela de imagens e diretórios. Se o usuário clicar em um diretório, será mapeado esse diretório, novamente seguindo o perfil de imagens.

Se o usuário clicar em uma imagem, ele visualizará a imagem em tamanho completo.

O código referente ao processamento e síntese da página de imagens é mostrado logo a seguir. Como pode ser observado, foi criada uma função chamada *navPics()*, que

recebe o diretório que será mapeado. Esse diretório é então processado, seus arquivos são mapeados, e todo o conteúdo é exibido de maneira tabelada, fazendo uso de *tags* HTML.

Juntamente com as imagens, é também exibido o nome de cada uma delas.

```
function navPics($chdir){
    $dir = realpath($chdir); //Obtém o caminho completo para o diretório.
    chdir($dir); //Muda a execução para o diretório.
    echo "<p>Diretório: $dir</p>"; //Exibe o diretório para o usuário.

    $files = scandir($dir); //Armazena em $files um vetor de arquivos do
    diretório.

    //Essa parte trata da segurança de acesso, restringindo o acesso à pasta
    /sdcard.
    $dirAux = str_replace("/sdcard", "", $dir);
    //Verifica se é diretório.
    if(is_dir($files[1])){
        $retDir = realpath($files[1]);
        if(strcmp('/sdcard', $retDir)==0){}
        else if(strcmp('/sdcard', $retDir)<0)
            echo "<br/><input type=\"button\" value=\"Voltar\"
onclick=\"chdir('$retDir', 'img');\"/><br/>";
    }

    $auxVar=0;

    echo "<table id=\"picTable\">";
    echo "<tr>";

    //Para todos os arquivos mapeados a partir da posição 2...
    for($i = 2; $i < count($files) ; $i++){
        if(is_dir($files[$i])){
            $aux = realpath($files[$i]);
            echo "<td><a style=\"color:#00f;\" href=\"#\
onclick=\"chdir('$aux', 'img');\"> $files[$i] </a></td></tr><tr>";

        }
    }
}
```

```

        if(is_file($files[$i]) && $auxVar != 3){
            $pathParts = pathinfo($files[$i]);
            $aux = $pathParts['basename'];
            echo "<td><a style=\"color:#000;\" href=\"\$dirAux/$aux\"><img
src=\"\$dirAux/$aux\" width=160 height=120/>$aux</a></td>";
            $auxVar = $auxVar + 1;
        } else if(is_file($files[$i]) && $auxVar==3){
            echo "</tr>";
            $auxVar = 0;
            echo "<tr><td><a style=\"color:#000;\"
href=\"\$dirAux/$aux\"><img src=\"\$dirAux/$aux\" width=160
height=120/>$aux</a></td>";

        }
    }
    echo "</td>";
    echo "</table>";
}

//Tratamento da requisição para mapear o diretório de imagens.
if(isset($_GET['img']) && isset($_GET['chdir'])){
    $dirPics=$_GET['chdir'];
    navPics($dirPics);
}

```

2. **foldernav.php?md&chdir=diretório**: Esse tipo de requisição define o mapeamento de um diretório de músicas, por causa do parâmetro **md** passado ao *script* **foldernav.php**.

Além disso, define o diretório a ser mapeado com o perfil de músicas através do parâmetro **chdir**, que tem como valor o diretório a ser mapeado.

Com base nesses parâmetros, o *script* **foldernav.php** irá sintetizar uma página para o usuário contendo diretórios e músicas contidos no diretório passado como valor do parâmetro.

O trecho responsável por mapear e mostrar o conteúdo de diretórios de músicas em PHP é mostrado no código a seguir. Nesse código, há uma função chamada *navMedia()*, que recebe como parâmetro o diretório que será mapeado, processado e exibido ao usuário. No caso de ser um diretório, ele será processado e exibido ao usuário como um *link* que irá disparar a função JavaScript *chdir()* - implementada no

arquivo **js/ajax.js** do Apêndice 9.2.1 - tendo como argumento o tipo de diretório, que é de música, e o diretório a ser mapeado.

```
function navMedia($chdir){
    $dir = realpath($chdir);
    chdir($dir);
    echo "<p>Diretório: $dir</p>";
    $files = scandir($dir);

    //Verifica se é diretório.
    if(is_dir($files[1])){
        $retDir = realpath($files[1]);
        if(strcmp('/sdcard', $retDir)==0){}
        else if(strcmp('/sdcard',$retDir)<0)
            echo "<br><input type='button' value='Voltar'
onclick='chdir('$retDir','md');' /><br>";
    }

    //Para todos os arquivos mapeados a partir da posição 2...
    for($i = 2; $i < count($files) ; $i++){
        if(is_dir($files[$i])){
            $aux = realpath($files[$i]);
            echo "<p><a style='color:#00f;' href='\"#\"'
onclick='chdir('$aux','md');'> $files[$i] </a></p>";
        }
    }

    //Verifica se é arquivo
    if(is_file($files[$i])){
        $aux = realpath($files[$i]);
        echo "<p><a style='color:#000;' href='\"#\"'
onclick='play('$aux');'>$files[$i]</a></p>";
    }
}

if(isset($_GET['md']) && isset($_GET['chdir'])){
    $chdir = $_GET['chdir'];
    if(is_dir($chdir)){
```

```

        navMedia($chdir);
    }
    else echo "error";
}

```

E, ao clicar na música, será disparada uma ação de execução daquela música utilizando a função *play()*, implementada no arquivo **js/mediaControl.js** do Apêndice 9.2.3. E Essa função *play()* dispara uma requisição para executar o arquivo passado como parâmetro ao **mplayer.php**, que será melhor abordado no Capítulo 4.6. Caso o usuário clique em um diretório, será feita uma nova chamada ao *script foldernav.php*, fazendo uso da função *chdir()*, implementada no arquivo **js/ajax.js** do Apêndice 9.2.1. Essa função recebe como parâmetro o valor do diretório clicado, como pode ser visto no código mostrado anteriormente.

A função *play()* recebe como parâmetro o caminho completo do arquivo, e dispara uma requisição ao *script mplayer.php* com o caminho completo do arquivo. A função *play()* é mostrada logo adiante.

```

function play(file){
    $.get("php/mplayer.php?play="+file);
}

```

3. **foldernav.php?vd&chdir=diretório**: Esse tipo de requisição define o mapeamento de um diretório de vídeos, por causa do parâmetro **vd**.

Além disso, define também o diretório que será mapeado com o perfil de vídeos através do parâmetro **chdir**, que terá como valor o diretório que será mapeado.

Com Essa requisição, o *script foldernav.php* irá sintetizar uma página para o usuário contendo diretórios e arquivos de vídeo que estiverem dentro do diretório passado como parâmetro.

São utilizadas duas funções JavaScript para disparar requisições ao *script foldernav.php*, a *chdir()* e *playVideo()*, que é a função *play()*, só que com dois parâmetros, pois agora, além do arquivo de vídeo, é necessário pegar qual a saída desejada, tal como TV, LCD, ou TV e LCD.

```

function navVideos($chdir){
    $dir = realpath($chdir);
}

```



```

chdir($dir);
echo "<p>Diretório: $dir</p>";
//print_r($dir);
$files = scandir($dir);
//print_r($files);

if(is_dir($files[1])){
    $retDir = realpath($files[1]);
    if(strcmp('/sdcard', $retDir)==0){}
    else if(strcmp('/sdcard',$retDir)<0)
        echo "<br/><input type=\"button\" value=\"Voltar\"
onclick=\"chdir('$retDir','vd');\"/><br/>";
    }

//Para todos os arquivos mapeados a partir da posição 2...
for($i = 2; $i < count($files) ; $i++){
    if(is_dir($files[$i])){
        $aux = realpath($files[$i]);
        echo "<p><a style=\"color:#00f;\" href=\"#\"
onclick=\"chdir('$aux','vd');\"> $files[$i] </a></p>";
    }
    if(is_file($files[$i])){
        $aux = realpath($files[$i]);
        echo "<p><a style=\"color:#000;\" href=\"#\"
onclick=\"playVideo('$aux','$('#out').val());\">$files[$i]</a></p>";
    }
}
}

//Tratamento das requisições para sintetizar diretórios de vídeo
if(isset($_GET['vd']) && isset($_GET['chdir'])){
    $chdir = $_GET['chdir'];
    if(is_dir($chdir)){
        navVideos($chdir);
    }
    else echo "error";
}

```

Como pode ser visto no trecho de código mostrado anteriormente, cada arquivo de vídeo é sintetizado com o seguinte trecho de código PHP:

```
echo "<p><a style=\"color:#000;\" href=\"#\
onclick=\"playVideo('$aux',$('#out').val());\">${files[$i]}</a></p>";
```

Esse código, em conjunto com toda a rotina, irá sintetizar *links* HTML que irão disparar a função *playVideo()* ao serem clicados. Cada *link* é sintetizado com a função *playVideo()* fazendo referência ao caminho completo do arquivo de música, e somente o nome do arquivo é exibido como *link*. E para obter a forma de saída de vídeo foi utilizada a funcionalidade jQuery “\$(‘out’).val()”, que obtém o valor selecionado da *tag* HTML de valor “out”, da página de Vídeos (arquivo **pages/vídeos.php** – Apêndice 9.3.12). Os valores de nome de caminho completo do arquivo e do tipo de saída selecionado são passados para a função *playVideo()*, que por sua vez dispara uma requisição ao **mplayer.php** com Esses parâmetros, como mostrado no trecho abaixo, que exibe a função *playVideo()*:

```
function playVideo(file,type){
    $.get("php/mplayer.php?play="+file+"&out="+type);
}
```

A função *chdir()*, utilizada para requisições de mudança de diretórios de vídeo, imagens e músicas, foi implementada em JavaScript, e faz uso de recursos do *framework* jQuery para disparar requisições ao *script* **foldernav.php**. Essa função recebe como parâmetros o diretório que deve ser mapeado, e o tipo desse diretório, que pode ser música (md), vídeo (vd) ou imagem (img). Ao realizar Essa requisição, a função irá receber uma resposta do servidor, e Essa resposta será exibida na *div* HTML de *id* “folder”. As páginas de Músicas, Vídeos e Imagens possuem essa mesma *div*, de modo a permitir o reuso dessa função.

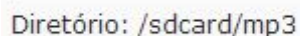
```
function chdir(dir,type){
    if(type === "md"){
        $.get("php/foldernav.php?md&chdir="+dir, function(dir){
            $('#folder').html(dir);});
    }
    else if(type ==="img"){
        $.get("php/foldernav.php?img&chdir="+dir, function(dir){
            $('#folder').html(dir);});
    }
    else if(type ==="vd"){
        $.get("php/foldernav.php?vd&chdir="+dir, function(dir){
            $('#folder').html(dir);});
    }
}
```

```
}
}
```

Para auxiliar o usuário a identificar o diretório que está sendo mapeado, é gerada uma saída no topo de cada uma das páginas, tal como mostrada na Figura 44. Essa saída é criada através do seguinte código PHP, presente nas rotinas de mapeamento de vídeos, imagens e músicas:

```
echo "<p>Diretório: $dir</p>";
```

Esse código basicamente retorna o diretório passado como parâmetro.



Diretório: /sdcard/mp3

Figura 44 - Diretório mapeado.

Para fins de testes e de demonstração das funcionalidades, foram criados 3 diretórios no cartão de memória inserido na Tiny6410, a saber:

- **/sdcard/mp3**– Diretório aonde foram colocadas pastas contendo álbuns completos de músicas.
- **/sdcard/videos**– Diretório aonde foram colocados vídeos de bandas musicais.
- **/sdcard/images** – Diretório aonde ficarão armazenadas as fotos tiradas com a *webcam* plugada na porta USB da Tiny6410.

Como resultado final, é mostrada na Figura 45 o resultado da síntese do diretório **/sdcard/mp3**, que foi definido como diretório padrão ao acessar a página de Músicas. Como não há arquivos de música, são exibidos somente diretórios. Já na Figura 46 é mostrada a síntese do diretório **/sdcard/mp3/Summer Eletrohits 5**. Podem ser vistos todos os arquivos presentes no diretório, com suas extensões.



Figura 45 - Página de Músicas Exibindo os diretórios em /sdcard/mp3

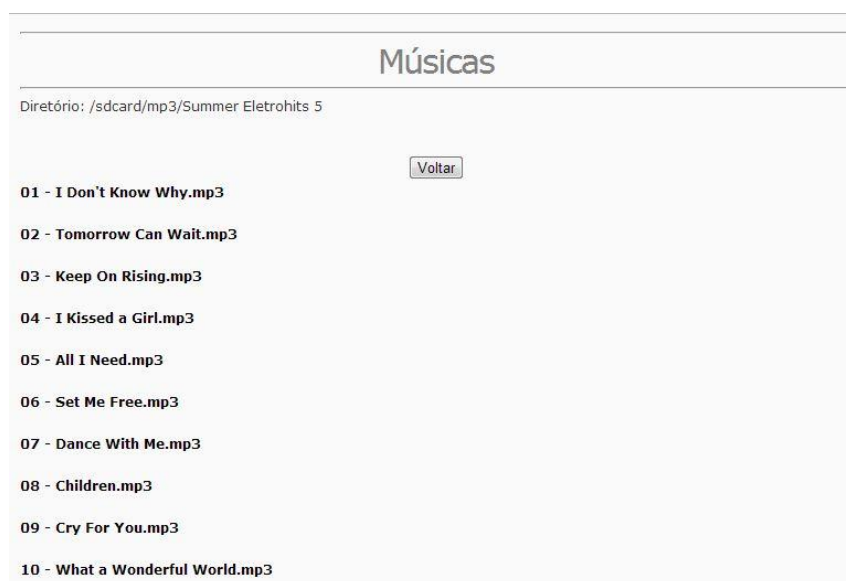


Figura 46 - Página de Músicas exibindo os arquivos dentro da pasta Summer Eletrohits 5.

Já para a parte de Vídeos, é mostrada na Figura 47 o resultado da síntese do diretório **/sdcard/vídeos**, mostrando os dois diretórios presentes, e o menu de seleção de saída de vídeo. Ao clicar no diretório **Clips**, é enviada uma requisição ao servidor, o qual processa esse diretório, e o retorno é exibido ao usuário, o qual pode ser visto na Figura 48, que exibe o único arquivo de vídeo contido no diretório **/sdcard/vídeos/Clips**.

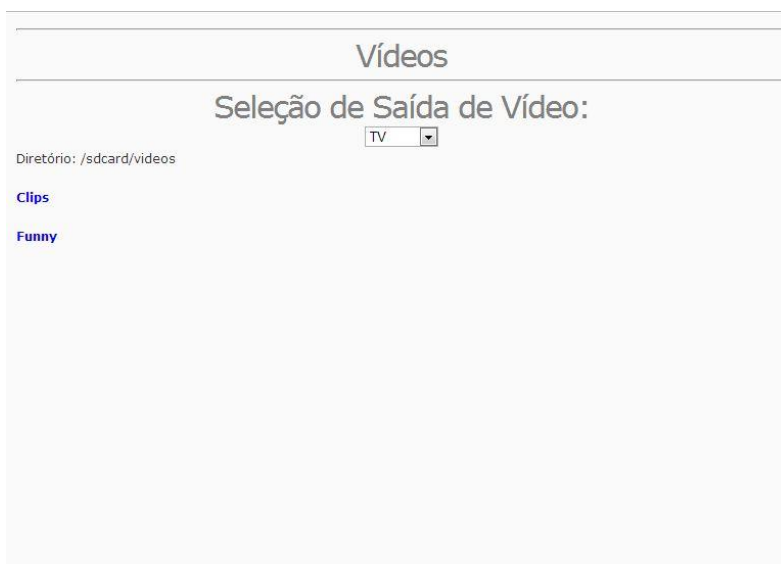


Figura 47 - Página de Vídeos exibindo os diretórios em **/sdcard/videos**

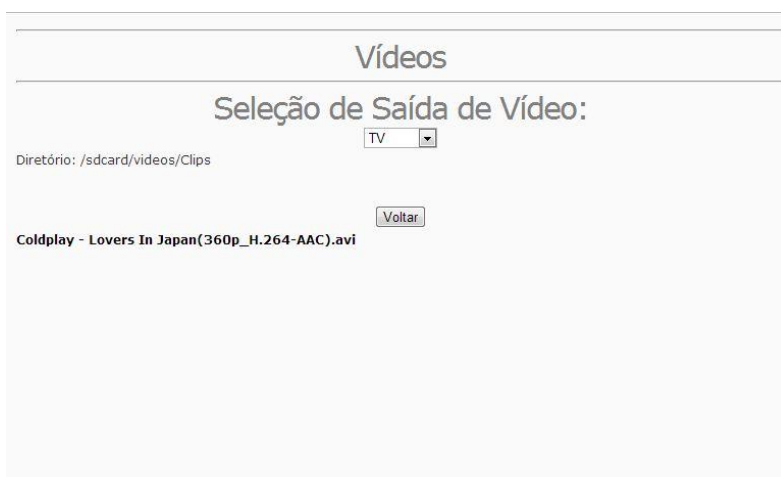


Figura 48 - Página de vídeos exibindo o conteúdo do diretório **/sdcard/videos/Clips**

Com relação às imagens, é mostrada na Figura 49 a tela gerada pela síntese do diretório **/sdcard/pictures**. Como comentado na parte de implementação, pode ser visto que as imagens são exibidas de maneira tabelada, e os nomes das imagens são exibidos também. O único diretório presente é o diretório **time**, responsável por guardar as fotos tiradas de maneira temporizada.

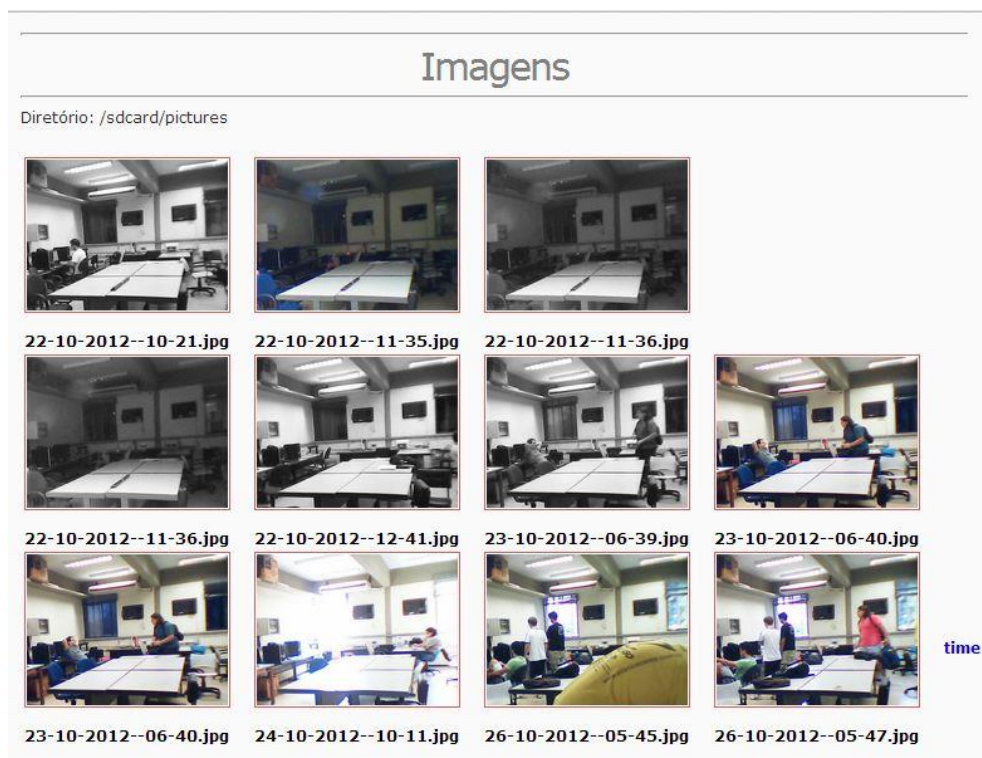


Figura 49 - Página de Imagens mostrando os arquivos e diretórios em /sdcard/pictures

4.6 Execução e Controle de conteúdo multimídia

Para execução de conteúdo multimídia, como vídeos e músicas, basicamente é preciso ter um arquivo multimídia e um programa capaz de executar esse tipo de arquivo. Como comentado no Capítulo 3, o melhor programa para Essa função é o MPlayer, que já vem por padrão nas últimas versões de **rootfs** da FriendlyARM.

De modo a testar o funcionamento e a execução correta do MPlayer, foram inseridas algumas músicas no cartão de memória utilizado no *kit*, no diretório “/sdcard/Musics”. E para executar qualquer música com o MPlayer via linha de comando, basta digitar o seguinte comando no terminal:

```
$ mplayer /"caminho para o arquivo de música"
```

Em que “/caminho para o arquivo de música” é o diretório completo da música no sistema, por exemplo “/sdcard/Musics/musica.mp3”.

Ao digitar esse comando, o arquivo de música começará a ser executado, e caso tenha caixas de som ou fones de ouvido conectados à placa, será possível ouvir a música.

Um detalhe que foi muito pensado para permitir o controle de mídias pela internet foi o fato de o programa permitir um controle maior via linha de comando, o que permite o controle do mesmo programa pelas funções de execução do PHP, que por sua vez podem responder requisições *web*.

Nisso, uma característica muito interessante do **MPlayer** é o seu chamado *slave mode*⁴³, que permite o programa receber uma série de comandos a partir de um arquivo do tipo FIFO⁴⁴. Esse tipo de arquivo pode ser criado com o seguinte comando:

```
$ mkfifo "/caminho para o arquivo"
```

Em que *"/caminho para o arquivo"* é o endereço completo do arquivo no sistema. No caso do projeto, foi criado um FIFO chamado **mplayer.cmd** dentro da pasta *"/www/files/"* com o seguinte comando:

```
$ mkfifo /www/files/mplayer.cmd
```

Dessa forma, qualquer comando, de uma lista de comandos compatíveis, que for escrito no arquivo do tipo FIFO resultará na ação do comando no **MPlayer**. Lembrando que para a escrita de comandos no arquivo FIFO funcionar, o **MPlayer** precisa estar em execução.

Para iniciar o **MPlayer** em *slave mode* associado a um arquivo FIFO, é necessário fazer o seguinte comando, com base no arquivo FIFO utilizado no projeto:

```
$ mplayer -slave -input file=/www/files/mplayer.md /sdcard/Musics/musica.mp3
```

Considerando **musica.mp3** um arquivo de música qualquer. Após esse comando, o **MPlayer** começará a ser executado. Para pausar a execução da música basta proceder com o seguinte comando:

```
$ echo pause > /www/files/mplayer.md
```

Para maiores detalhes a respeito da lista de comandos compatíveis com o *slave mode* do **MPlayer** recomenda-se consultar [ftp://ftp7.mplayerhq.hu/MPlayer/DOCS/tech/slave.txt](http://ftp7.mplayerhq.hu/MPlayer/DOCS/tech/slave.txt).

⁴³ *Slave mode* significa "modo escravo" em português.

⁴⁴ FIFO é a abreviação do termo em inglês *First In First Out*, que em português significa Primeiro a Entrar Primeiro a Sair. Basicamente, este tipo de arquivo descreve um "canal" de comunicação entre um programa e outro elemento qualquer.

Dessa forma, basta utilizar funções específicas do PHP, que executam ações e iniciam processos na máquina servidora, que é então possível controlar a execução do **MPlayer** em *slave mode*. Exemplos de funções que executam ações e processos na máquina servidora com PHP são:

- **exec(string \$command)**: Executa um programa externo, definido pela *string* **\$command**, e normalmente não gera um retorno em sua execução. Maiores detalhes em: <http://php.net/manual/en/function.exec.php>.
- **shell_exec(string \$cmd)**: Executa um comando através de linha de comando *shell*, definido pela *string* **\$cmd**, e pode retornar toda a saída gerada pela execução quando associado a alguma variável. Maiores detalhes em: <http://php.net/manual/en/function.shell-exec.php>.
- **system(string \$command)**: Executa um programa externo, passado pelo parâmetro **\$command**, e retorna somente a última linha da execução desse programa. Maiores detalhes em: <http://php.net/manual/en/function.system.php>

De modo a receber requisições *web*, e disparar ações específicas com base nessas requisições para o **MPlayer**, foi implementado o arquivo **mplayer.php**, que é mostrado por completo no Apêndice 9.1.7. Juntamente a Essa parte, foi também implementado o arquivo **js/mediaControl.js** (o qual pode ser visto no Apêndice 9.2.3), que contém funções escritas em JavaScript com o *framework* jQuery, as quais fazem uso de cada uma das requisições implementadas no **mplayer.php**. Essas funções podem ser associadas a diversos elementos HTML, tais como botões, *links*, etc, como será visto em outras partes do projeto.

A execução de conteúdo multimídia do *script* **mplayer.php** só responde aos seguintes parâmetros:

- **mplayer.php?play=/path**: Essa requisição trata de executar somente músicas, recebendo como parâmetro GET a variável **play**, que possui como valor o caminho completo do arquivo a ser executado. Sempre que recebe uma requisição desse tipo, o *script* termina qualquer execução do **MPlayer** que esteja em andamento, recebe e trata o caminho completo do arquivo passado como parâmetro. Após isso, o *script* escreve em um arquivo qual a mídia que está sendo executada, e depois prepara a *string* de comando para ser executada pela função *play()*, que basicamente executa a função **shell_exec** com o parâmetro recebido. Ocorrendo tudo corretamente, a música começará a ser executada pelo **MPlayer**. Logo adiante é mostrado o trecho de código responsável por tratar Essa requisição.


```

if(isset($_GET['play']) && !isset($_GET['out'])) {

    //Rotina que termina execução de mplayer.
    if(isRunning()){
        killMP();
    }
    $media = $_GET['play']; //Associação de variável.

    $fileplay = escapeshellarg($media); //Tratamento de
caracteres.

    writeStatus($fileplay,$status);
    $out = "off";

    $cmd = "$command $out $fileplay";

    play($cmd);
}

```

Como mostrado no Capítulo 4.5, cada *link* de arquivo de música dispara a função JavaScript *play()* ao ser clicado. E como mostrado novamente no trecho de código abaixo, Essa função faz a requisição GET ao **mplayer.php** passando o caminho completo do arquivo de música a ser executado. Essa função não precisa de um retorno específico, pois o **MPlayer** gera conteúdo de retorno durante toda a execução da mídia.

```

function play(file){
    $.get("php/mplayer.php?play="+file);
}

```

- **mplayer.php?stop:** Esse tipo de requisição basicamente encerra a execução corrente do **MPlayer**. Seu funcionamento é muito simples. Ao receber a requisição **stop**, o *script* **mplayer.php** verifica se o **MPlayer** está em execução, através da função *isRunning()* que retorna valores *booleanos* para determinar se o **MPlayer** está em execução. Caso esteja em execução, o programa é encerrado com a função *killMP()*, que é uma função PHP que obtém o *id* de Processo do PHP e encerra o processo . Caso o programa não

Esseja em execução, é chamada a função *notRunning()*, que basicamente retorna uma mensagem dizendo que o **MPlayer** não está em execução.

```
if(isset($_GET['stop'])){
    if(isRunning())
        killMP();
    else {
        notRunning();
    }
}
```

Para disparar Essa requisição através de componentes HTML, foi implementada a função *stop()* em JavaScript, mostrada no trecho de código a seguir.

```
function stop(){
    $.get("php/mplayer.php?stop");
}
```

- **mplayer.php?pause:** Essa requisição paralisa a execução corrente do **MPlayer** no sistema, fazendo uso da função PHP **shell_exec()** para enviar um comando de *pause* ao arquivo associado a execução em *slave mode* do **MPlayer**. Um detalhe importante a ser observado é que o envio do comando só pode ser realizado com o **MPlayer** em execução, caso contrário o processo gerado pelo **shell_exec()** poderá congelar a execução do interpretador PHP no sistema. Para contornar isso, é verificado se o **MPlayer** está em execução através da função *isRunning()*. Caso não Esseja, é executada a função *notRunning()*, que basicamente retorna uma mensagem dizendo que o **MPlayer** não está em execução.

```
if(isset($_GET['pause'])){
    if(isRunning())
        shell_exec("echo pause > $mpcmd");
    else {
        notRunning();
    }
}
```

Para disparar Essa requisição através de componentes HTML, foi implementada a função *pause()* em JavaScript, mostrada no trecho de código a seguir:

```
function pause(){
    $.get("php/mplayer.php?pause");
}
```

- **mplayer.php?vol=up&u=Numero** e **mplayer.php?vol=dw&u=Numero**: Esse par de requisições realiza basicamente a mesma função, que é a de controlar o volume da execução corrente do **MPlayer**. No caso de o parâmetro **vol** possuir valor “up”, o valor do parâmetro **u** será usado para aumentar o volume, não excedendo o valor 100. No caso de o parâmetro **vol** possuir o valor *dw*, o valor do parâmetro **u** será usado para diminuir o volume, não podendo passar de 0. Esses comandos de controle de volume também fazem uso das funcionalidades do *slave mode* do **MPlayer**, dessa forma, os comandos só são executados caso o **MPlayer** esteja em execução na máquina, caso contrário, é retornada uma mensagem de que o programa não está em execução.

```
if(isset($_GET['vol']) && isset($_GET['u'])){
    if(isRunning()){
        $volume = $_GET['vol'];
        $vol = $_GET['u'];
        //Se o controle for para aumentar o volume...
        if($volume == "up"){
            if ($vol < 100){
                $vol = $vol;
            } else $vol = $maxVolume;
            $mpvol = escapeshellarg("Volume $vol 100");
            shell_exec("echo $mpvol > $mpcmd");
        }
        //Se o controle for para diminuir o volume...
        else if ($volume == "dw"){
            if($vol > 0) {
                $vol = $vol;
            } else $vol = $minVolume;
            $mpvol = escapeshellarg("Volume $vol 100");
            echo $mpvol;
            shell_exec("echo $mpvol > $mpcmd");
        }
    }
}
```

```

        }
    } else {
        notRunning();
    }
}

```

As funções JavaScript escritas para controlar o volume da execução corrente do **MPlayer** são mostradas nos trechos de código a seguir. Por enquanto, o tratamento da intensidade de volume a ser passada fica a controle dessas funções. A função *volumeUp()* incrementa o volume e mostra o resultado na *div* HTML de *id* “*volume*”. Já a função *volumeDown()* decrementa o volume e também mostra o resultado na *div* HTML de *id* “*volume*”.

```

unction volumeUp(){
    if(volume === 100){
        volume = 100;
        $.get("php/mplayer.php?vol=up&u="+volume);
        $('#volume').html(volume);
    }
    else {
        volume+=10;
        $.get("php/mplayer.php?vol=up&u="+volume);
        $('#volume').html(volume);
    }
}

function volumeDown(){
    if(volume === 0){
        volume = 0;
        $.get("php/mplayer.php?vol=dw&u="+volume);
        $('#volume').html(volume);
    }else{
        volume-=10;
        $.get("php/mplayer.php?vol=dw&u="+volume);
        $('#volume').html(volume);
    }
}

```

- **mplauer.php?mute**: Essa requisição faz uso da funcionalidade do *slave mode* do **MPlayer** para colocar a saída de áudio no mudo, ou seja, sem áudio sendo gerado na saída de som. Esse modo não paralisa a execução, simplesmente corta o áudio gerado. A requisição é simples, seguindo a mesma receita mostrada anteriormente. Primeiro é verificado se o **MPlayer** está em execução, e caso Esseja, é gerado um comando do tipo *mute* no arquivo associado à execução *slave mode* do **MPlayer**, o qual coloca o áudio no mudo. Caso o **MPlayer** não Esseja em execução, simplesmente é retornara uma mensagem dizendo que ele não está em execução. O trecho de código PHP a seguir mostra o tratamento dessa requisição.

```
if(isset($_GET['mute'])){
    if(isRunning())
        shell_exec("echo mute > $mpcmd");
    else {
        notRunning();
    }
}
```

Para permitir o disparo dessas requisições através de componentes HTML, foi implementada a função *mute()* em JavaScript, a qual é mostrada no trecho de código abaixo:

```
function mute(){
    $.get("php/mplayer.php?mute");
}
```

- **mplayer.php?prv**: Essa ainda é uma requisição experimental. Até o presente momento, o que pretende-se fazer é enviar também ao *script* **mplayer.php** o diretório que será mapeado. Esse, por sua vez, irá mapear em um vetor todos os arquivos desse diretório. Ao receber um arquivo desse diretório para execução, o *script* irá identificar em qual posição do vetor esse arquivo se encontra. Caso haja uma requisição para executar o arquivo anterior, será verificado se existem arquivos em posições anteriores, e caso positivo, o valor anterior será obtido, verificado se é um arquivo, e passado para execução para o **MPlayer**. Para uso dessas funcionalidades, percebeu-se a necessidade do uso de sessões do PHP, que permitem guardar variáveis utilizadas na execução do *script*. Por se tratar de uma requisição ainda experimental, detalhes de sua implementação não serão tratados aqui, podendo ser vistos no código completo do *script* **php/mplayer.php** mostrado no Apêndice 9.1.7.

De modo a preparar o uso futuro dessa requisição, já foi implementada a função *prev()* em JavaScript, que dispara a requisição para executar o arquivo anterior. Essa função pode ser vista no trecho de código mostrado abaixo.

```
function prev(){  
    $.get("php/mplayer.php?prv");  
}
```

- **mplayer.php?nxt:** Essa também é uma requisição experimental. Funciona da mesma forma que a requisição anterior, com a diferença que faz o **MPlayer** executar a próxima música da lista de execução montada a partir do diretório passado como argumento. Por se tratar de uma requisição ainda experimental, detalhes de sua implementação não serão tratados aqui, podendo ser vistos no código completo do *script* **php/mplayer.php** mostrado no Apêndice 9.1.7.

Também de modo a preparar o uso futuro dessa requisição, já foi implementada a função *next()* em JavaScript, que dispara a requisição para executar o arquivo posterior. Essa função pode ser vista no trecho de código abaixo.

```
function next(){  
    $.get("php/mplayer.php?nxt");  
}
```

As partes do projeto que mais fazem uso das requisições e das funcionalidades aqui descritas são a página *Músicas*, cujo código é descrito e detalhado no Apêndice 9.3.9, a página *Vídeos*, que será abordada no capítulo seguinte, e o Menu de controle multimídia, que será abordado no Capítulo 4.15.

4.7 Execução de Vídeo e Controle de Saída de Vídeo

A sessão anterior descreveu em detalhes a parte de execução e controle de arquivos de música. Executar e controlar arquivos de música é um pouco mais fácil do que lidar com arquivos de vídeo, pois não é necessário definir a saída de vídeo desejada.

Já para a execução de arquivos de vídeo é preciso definir a saída de vídeo desejada, que pode ser o *display* LCD, a saída de TV, ou ambos, TV e *display* LCD. Se nenhuma saída for definida, o **MPlayer** irá exibir o conteúdo de vídeo no *display* LCD. Em contrapartida, a ausência de definição de saída significa um menor controle sobre a execução de vídeo.

Como dito no Capítulo 3, somente as versões recentes do **rootfs** da FriendlyARM possuem o **MPlayer** compilado com suporte para a saída de TV do *kit* Tiny6410.

Com base nas especificações desse **MPlayer** modificado, disponíveis no documento “The MPlayer multimedia-based application development guide 2011-08-15.pdf” [26], fornecido pela FriendlyARM no fórum *Arm9Home.net*, é possível selecionar a saída de vídeo com os seguintes parâmetros, passados ao **MPlayer**:

- a. **mplayer -tvout off**: Esse parâmetro define a saída de vídeo como sendo somente o *display* LCD.
- b. **mplayer -tvout tvonly**: Esse parâmetro define a saída de vídeo como sendo a saída RCA, normalmente utilizada para transmitir o vídeo a televisores e projetores.
- c. **mplayer -tvout tvlcd**: Esse parâmetro define a saída de vídeo como sendo simultaneamente o *display* LCD e a saída RCA.

Para controle da saída de vídeo, os mesmos controles de requisição mostrados na sessão anterior continuam funcionando, pois o *script* PHP utilizado é o mesmo, o arquivo **mplayer.php**, com a adição de mais um parâmetro GET que permite controlar o tipo de saída de vídeo, como mostrado no trecho de código PHP abaixo.

```
if(isset($_GET['play']) && isset($_GET['out'])){

    if(isRunning()){
        killMP();
    }

    $media = $_GET['play'];
    $mode = $_GET['out'];

    $fileplay = escapeshellarg($media);

    if ($mode == "3") {
        $out = "tvandlcd";
        $cmd = "$command $fileplay -tvout $out";
    }
}
```

```

        writeStatus($fileplay, $status);
        play($cmd);
    }
    else if ($mode == "2") {
        $out = "tvonly";
        $cmd = "$command $fileplay -tvout $out";
        writeStatus($fileplay, $status);
        play($cmd);
    }
    else if ($mode == "1") {
        $out = "off";
        $cmd = "$command $fileplay";
        writeStatus($fileplay, $status);
        play($cmd);
    }
}

```

Com base no trecho de código mostrado acima, é possível mostrar que o *script* **mplayer.php** responde unicamente às requisições abaixo, para execução de vídeo:

- **mplayer.php?play=/path/file&out=1** : Essa requisição recebe os parâmetros **play** e **out**, transferidos via GET. O valor de **play** é o caminho completo do arquivo de vídeo, e o valor de **out**, no caso “1”, determina que o vídeo será exibido somente no *display* LCD.
- **mplayer.php?play=/path/file&out=2** : Essa requisição do tipo GET recebe o caminho completo do arquivo de vídeo a ser executado, através do parâmetro **play**, e o valor “2” do parâmetro **out** determina que o arquivo será executado somente na saída RCA.
- **mplayer.php?play=/path/file&out=3** : Já Essa requisição, também do tipo GET, recebe o caminho completo do arquivo de vídeo a ser executado, por meio do parâmetro **play**, e o valor “3” do parâmetro **out** determina que o vídeo será mostrado simultaneamente no *display* LCD e na saída RCA.

A função *playVideo()*, escrita em JavaScript, foi escrita de modo a permitir o disparo de tais requisições, permitindo tanto determinar o arquivo que será executado, como a saída de vídeo utilizada. O trecho de código mostrado abaixo revela que a função *playVideo()* recebe como parâmetros o arquivo a ser executado, e o tipo de saída, e envia uma requisição GET ao *script mplayer.php*.

```
function playVideo(file,type){
    $.get("php/mplayer.php?play="+file+"&out="+type);
}
```

Além dessas requisições, existe ainda uma requisição em fase de protótipo, que é a *mplayer.php?camOut*. Ela trata de agregar tanto o *stream* de vídeo gerado pelo **MJPEG Streamer** quanto a funcionalidade de exibição de vídeo do **MPlayer**. Basicamente, essa requisição dispara um comando que irá criar um túnel de conexão entre o *stream* de vídeo e o **MPlayer**, permitindo exibir no *display* LCD ou na saída de vídeo RCA o vídeo obtido pelo *stream* de captura das câmeras USB. Um protótipo da requisição é mostrado a seguir.

```
if(isset($_GET['camOut'])){
    //      mkfifo a.mjpeg wget -O a.mjpeg http://localhost:8080
    2&gt;/dev/null & mplayer -cache 32 -demuxer 35 a.mjpeg
    // ou mplayer -demuxer lavf http://localhost:8081/stream.mjpg
    shell_exec("mkfifo a.mjpeg wget -O a.mjpeg http://localhost:8082 2&gt;
    /dev/null & mplayer -cache 32 -demuxer 35 a.mjpeg");
}
```

Já com relação à página de Vídeo, foi criada uma caixa de seleção, a qual permite selecionar qual a saída desejada. Quando o usuário clicar em algum arquivo de vídeo, a função *playVideo()* irá verificar qual é a saída selecionada na caixa de seleção, e essa saída será utilizada como parâmetro na função. Um exemplo da caixa de seleção aberta na página de Vídeos é mostrada na Figura 50. Maiores detalhes sobre a página de Vídeo são mostrados no Apêndice 9.3.12.

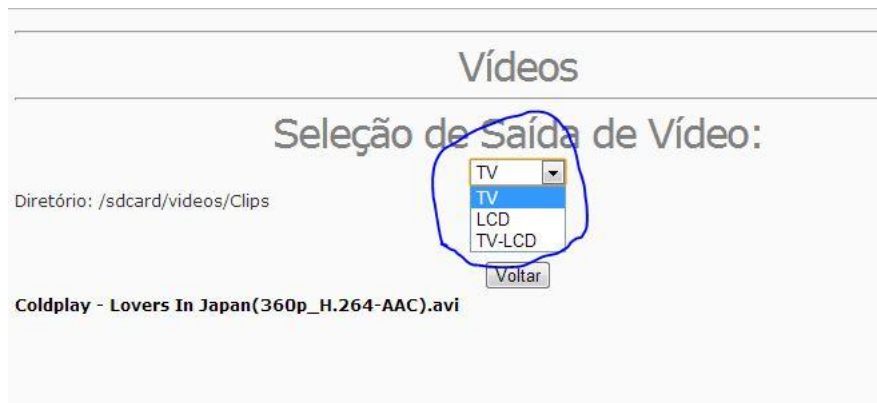


Figura 50 - Página de vídeo com destaque ao seletor de saída de vídeo.

Como resultado de todo o processo, pode ser visto na Figura 51 um exemplo da Tiny6410 executando o videoclipe referente ao arquivo “*Coldplay – Lovers in Japan (360p_H.264-AAC).avi*”. A execução do arquivo de vídeo se deu após o arquivo ter sido clicado na página de vídeo, e a saída de vídeo selecionada foi LCD.



Figura 51 - Tiny6410 executando videoclipe com saída exibida no *display* LCD.

4.8 Streaming de Vídeo

Para o *streaming* de vídeo, foi utilizado o **MJPEG Streamer** [84], que é um programa escrito em Linguagem C, de código aberto, o qual obtém quadros JPEG de uma câmera USB e os transmite no formato M-JPEG através da *web*.

É uma aplicação bem poderosa, com recursos tais que permitem determinar o dispositivo de captura de vídeo, a resolução do vídeo a ser transmitido, a porta a ser mapeada para o *stream* de vídeo, dentre outros detalhes.

O **MJPEG Streamer** por si mesmo já é um servidor *web*, pois é capaz de responder a requisições GET e POST, e é capaz de enviar conteúdo *web*, tal como páginas HTML, como resposta a tais requisições.

Como o projeto já faz uso do *lighttpd* como aplicativo servidor de conteúdo *web*, não foi necessário utilizar o **MJPEG Streamer** para esse propósito. Ele foi utilizado pura e simplesmente para *stream* de vídeo.

Detalhes completos a respeito de como compilar e transferir o **MJPEG Streamer** para a Tiny6410 são mostrados no documento “*ap04-mini6410-webcam.pdf*” fornecido pela FriendlyARM no DVD-A ou no fórum *Arm9Home.net*. Uma versão pronta, já compilada para uso, pode ser baixada no site <http://code.google.com/p/mjpg-streamer-mini2440/>.

Após baixar o arquivo compactado, recomenda-se descompactá-lo em alguma pasta no sistema de arquivos da Tiny6410. No projeto, o **MJPEG Streamer** foi descompactado e utilizado no diretório **/mjpg-streamer**. Dentro desse diretório encontram-se os binários, *plugins* e *scripts* necessários para a execução do *stream* de vídeo.

Os comandos completos para inicialização do **MJPEG Streamer** são mostrados no arquivo **start_uvc.sh**, correspondente ao Anexo 8.2, que é o *script Shell Script* utilizado para inicialização do *stream* de vídeo. Nesse *script*, a linha principal e essencial ao projeto é a seguinte:

```
/mjpg-streamer/mjpg_streamer -o "/mjpg-streamer/output_http.so -p 8082
-w /www" -i "/mjpg-streamer/input_uvc.so -d /dev/video2 -r 352x288"
```

Essa linha determina que o binário do **MJPEG Streamer** será inicializado fazendo uso do *plugin* de saída **output_http.so**, que codifica os dados para envio através do protocolo HTTP, escutando na porta 8082 e mapeando o diretório **/www**. E como parâmetro de entrada, determina que o binário fará uso do *plugin* **input_uvc.so**, que permite utilizar dispositivos de captura de vídeo compatíveis com UVC, e esse *plugin* fará uso do dispositivo **/dev/video2**, que corresponde à câmera USB no sistema Linux, com a resolução de 352 *pixels* por 288 *pixels*.

Como dito no início, o **MJPEG Streamer** também funciona como um servidor *web*, respondendo a requisições HTTP. Ele responde a duas requisições principais, de acordo com a documentação mostrada no site do projeto:

- **?action=stream:** Essa requisição determina que deve ser mostrado um fluxo contínuo de *frames* de imagens.
- **?action=snapshot:** Já essa requisição determina que deve ser mostrado um único frame de imagem JPEG, obtido no instante da requisição.

Inicialmente, ocorream alguns problemas na hora de inicializar o *script*. A princípio era pelo motivo de o *script* mapear **/dev/video0** como dispositivo de captura de vídeo, e na Tiny6410 o dispositivo correto era o **/dev/video2**.

No *script* inicial, a porta mapeada era a porta 8080. Por algum motivo, o uso externo da porta 8080 não foi possível. Não havia nada impedindo tal acesso, pois o arquivo de configuração do *lighttpd* não restringia esse acesso, e não havia nenhuma configuração de *firewall* na Tiny6410 restringindo isso também. Concluiu-se que trava-se de uma restrição da rede interna aos laboratórios da Engenharia Elétrica. Dessa forma, o *script* foi modificado para mapear a porta 8082, o que permitiu o acesso externo ao *stream* de vídeo na porta mapeada.

Antes de realmente descobrir que o problema era na porta 8080, a forma utilizada para mostrar o *stream* de vídeo gerado pelo **MJPEG Streamer** era com a criação de um *socket* de conexão com o *streamer* em um *script* PHP, tal como mostrado no trecho de código a seguir:

```
<?php
/* Usage:  */

$server = "localhost"; // camera server
$port = 8080; // camera server port
$url = "/?action=stream"; // url on camera server
$fp = fsockopen($server, $port, $errno, $errstr, 30);
if( !$fp ){
    echo "$errstr ($errno)<br />\n";
}
else
    $urlstring = "GET ".$url." HTTP/1.0\r\n\r\n";
    fwrite( $fp, $urlstring );

    while( $str = trim( fgets( $fp, 4096 ) ) )header( $str );
    fpassthru( $fp );
    fclose( $fp );

?>
```

Se o arquivo com esse código for chamado **webcam.php** e estiver na pasta monitorada pelo servidor *web*, por exemplo, basta utilizar a seguinte *tag* HTML para observar o *stream* de vídeo na tela da página:

```

```

Essa técnica funciona, mas tem um revés muito grande, que é o consumo de memória que isso gera ao interpretador PHP, podendo em muitos casos congelar o interpretador, o que dará a sensação de que a página não está respondendo.

Após ter descoberto que o problema era a porta 8080, mudando o mapeamento de porta para 8082, o *stream* passou a funcionar da seguinte maneira, que é:

```

```

Dessa forma, é o **MJPEG Streamer** que fica encarregado de tratar o *socket* de conexão HTTP e enviar o *frames* M-JPEG ao *browser* cliente, o que alivia a carga sobre o interpretador PHP, além do fato de deixar o fluxo de vídeo mais contínuo.

A resolução do *stream* de vídeo foi definida como sendo 352 *pixels* por 288 *pixels* para não gerar uma sobrecarga nos pacotes entre o servidor e a máquina cliente, até mesmo pensando no acesso simultâneo da tela de *stream* de vídeo. Caso a resolução fosse maior, isso iria requerer uma banda de conexão e um processamento maior, o que certamente tornaria o processo em geral mais lento, dadas as limitações da Tiny6410.

Para mostrar o *stream* de vídeo gerado pelo **MJPEG Streamer**, foi criada a página Câmera, referente ao arquivo **pages/camera.php**, mostrado por completo no Apêndice 9.3.3.

Por se tratar de conteúdo sigiloso, que é o monitoramento de dois laboratórios da Engenharia Elétrica, os quais possuem amplo fluxo de alunos, professores e funcionários, foi implementada uma camada de segurança na página Câmeras, que só permite o acesso a usuários autenticados.

De modo a permitir ter controle sobre o processo de execução do *stream* de vídeo, foram implementadas algumas respostas a requisições no *script* **status.php**, mostrado por completo no Apêndice 9.1.9. Logo adiante o trecho de código do *script* responsável por instanciar e encerrar o processo do *stream*.

```

if(isset($_GET['m_up'])){
    shell_exec($mjpgFile);
}

if(isset($_GET['m_down'])){
    $pid = shell_exec('pidof mjpg_streamer');
    shell_exec("kill $pid &");
}

```

Como visto no trecho de código, uma requisição do tipo **status.php?m_up** fará o **MJPEG Streamer** ser inicializado. Já uma requisição do tipo **status.php?m_down** fará o **MJPEG Streamer** ser finalizado. Em ambas as requisições, a variável `$mjpgFile` definida como:

```
$mjpgFile = '/mjpg-streamer/start_uvc.sh';
```

Para permitir o controle dessas requisições por elementos HTML, tais como botões, foram implementadas duas funções em JavaScript, mostradas no trecho de código abaixo:

```

function stopStream(){
    $.get("php/status.php?m_down");
    setTimeout(loadStatus,1500);
}

function startStream(){
    $.get("php/status.php?m_up");
    setTimeout(loadStatus,2000);
}

```

A função *stopStream()* envia a requisição para encerrar o *stream* de vídeo, e define uma chamada para a função *loadStatus()* (a qual será melhor explicada no Capítulo 4.16) após 1 segundo e meio (1500 milissegundos). A função *startStream()* é praticamente idêntica, com a diferença que envia uma requisição para iniciar o *stream* de vídeo, e como esse processo pode ser mais demorado, ela define uma chamada para a função *loadStatus()* após 2 segundos, para mostrar ao usuário o *status* do *stream* de vídeo.

No início do projeto, o desenvolvimento todo foi focado com base na câmera USB conectada na Tiny6410. Pouco tempo depois, o aluno João Ligabo, da Engenharia Elétrica, fez

um projeto utilizando o **MJPG Streamer** em uma SAM9-L9260, na disciplina Aplicações de Microprocessadores da Engenharia Elétrica. A câmera utilizada no projeto do João Ligabo monitora o laboratório aonde foi realizada a disciplina. Para poder receber o *stream* gerado pela câmera do projeto do João Ligabo, foi preciso somente acrescentar a seguinte linha de código no site Câmeras:

```

```

Dessa forma, foi possível mostrar mais um *stream* de vídeo no *site* da Tiny6410, o *ArmWeb Media Station*.

Finalmente, o resultado da página Câmeras é mostrado na Figura 52.

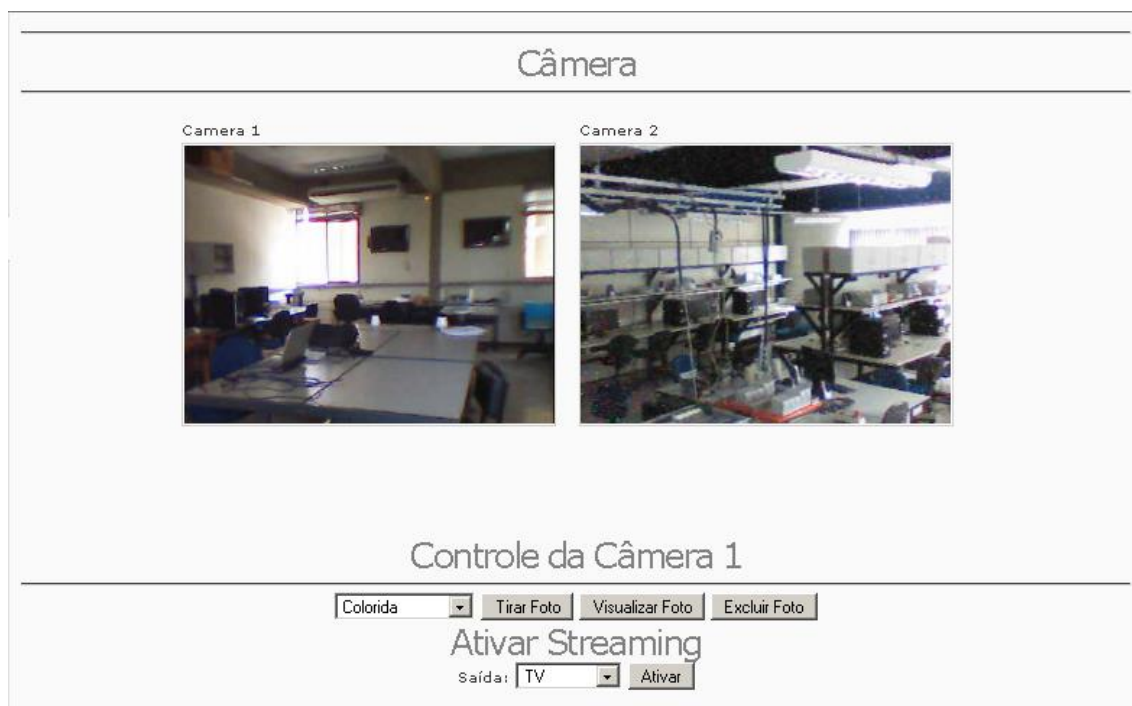


Figura 52 - Página "Câmeras" do projeto, exibindo dois *streams* de vídeo.

4.9 Captura de Imagem da Câmera e Captura Temporizada

A melhor forma de realizar a captura de imagens, obtidas pela câmera USB conectada na Tiny6410, foi por meio da biblioteca OpenCV [65].

O *Buildroot* em sua versão 2012.02 possui o pacote do OpenCV para compilação automatizada, o que facilita bastante o serviço de portar as bibliotecas para a Tiny6410.

Para compilar o OpenCV utilizando o *Buildroot*, basta acessar a seguinte sequência de opções: **Package Selecion for the Target -> Libraries -> Graphics**. Dentro da opção **Graphics**, devem ser marcados os pacotes **opencv**, **jpeg support**, **png support** e **v4l support**. Um exemplo de como devem ficar as opções é mostrado na Figura 53.

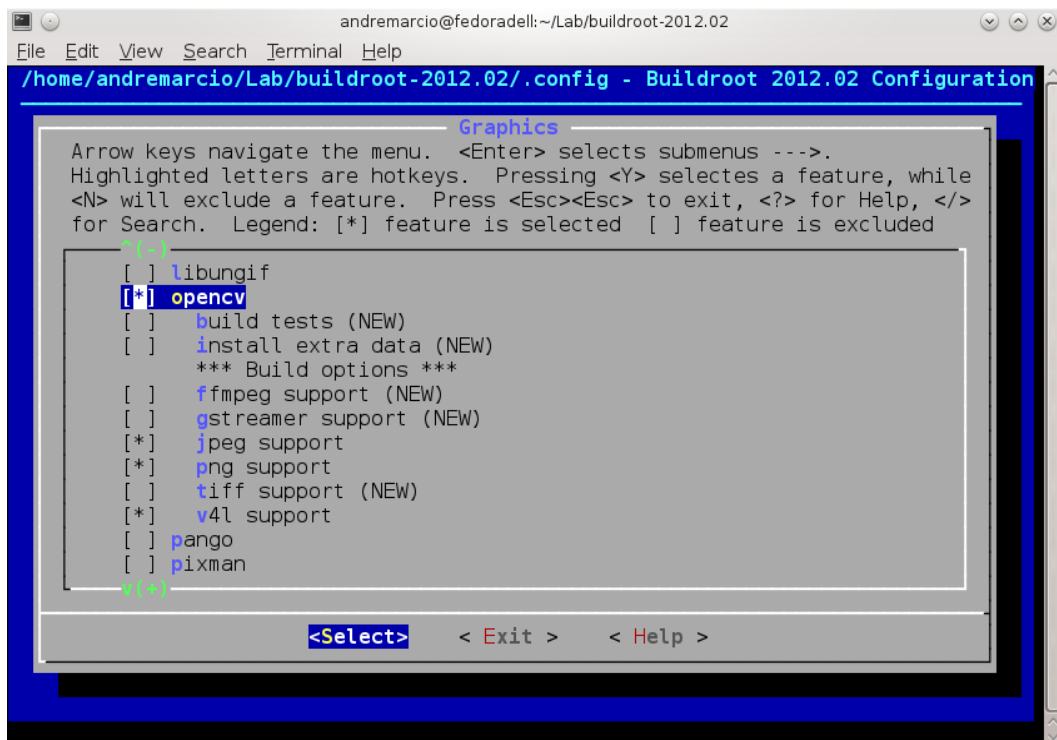


Figura 53 - Seleção dos pacotes de OpenCV no Buildroot

Com as opções marcadas, é preciso sair do *Buildroot*, salvando o arquivo de configurações, e digitar o comando **make** para dar início ao processo de compilação. Terminado o processo, é necessário copiar para a Tiny6410 o conteúdo dos diretórios **/usr/bin** e **/usr/lib**, que se encontram dentro do diretório **/output/target**.

Feito isso, as bibliotecas do OpenCV já se encontram no sistema de arquivos da Tiny6410. E para compilar a aplicação que irá realizar a captura de imagens da câmera USB, utilizando a biblioteca OpenCV, é preciso utilizar a pasta **/output/build/opencv2.3.1a** do *Buildroot*, que é gerada após o processo de compilação citado acima.

Vale lembrar também que esse processo fará uso da *toolchain* fornecida pela FriendlyARM, o que requer o binário **arm-linux-g++** na variável de ambiente **\$PATH** do Linux.

Recomenda-se copiar o diretório **output/build/opencv2.3.1a** para um outro local, de modo a evitar comandos demasiadamente longos, conforme a árvore de diretórios vai aumentando. Para o projeto, o diretório **opencv2.3.1a** foi copiado para a pasta **/home** do usuário.

Para compilar qualquer binário com as bibliotecas geradas para a Tiny6410, crie um arquivo chamado **makefile.sh**, e dentro desse arquivo coloque as linhas mostradas na sequência de código em linguagem *Shell Script* mostrada a seguir.

```
#!/bin/bash
arm-linux-g++ usbCapArg.c
-I/home/andremarcio/opencv-2.3.1a/include/
-I/home/andremarcio/opencv-2.3.1a/include/opencv
-I/home/andremarcio/opencv-2.3.1a/include/opencv2
-I/home/andremarcio/opencv-2.3.1a/modules/core/include
-I/home/andremarcio/opencv-2.3.1a/modules/highgui/include/opencv2
-I/home/andremarcio/opencv-2.3.1a/modules/imgproc/include/
-I/home/andremarcio/opencv-2.3.1a/modules/video/include/
-I/home/andremarcio/opencv-2.3.1a/modules/features2d/include/
-I/home/andremarcio/opencv-2.3.1a/modules/flann/include/
-I/home/andremarcio/opencv-2.3.1a/modules/calib3d/include/
-I/home/andremarcio/opencv-2.3.1a/modules/objdetect/include
-I/home/andremarcio/opencv-2.3.1a/modules/legacy/include
-I/home/andremarcio/opencv-2.3.1a/modules/highgui/include
-L/home/andremarcio/opencv-2.3.1a/lib
-L/home/andremarcio/opencv-2.3.1a/3rdparty/lib
-lopencv_highgui -lopencv_core -lpthread -lz -lrt -llibpng -llibjpeg -
llibtiff -lzlib -lopencv_imgproc -o usbCapArg
```

Esse *script* automatiza o processo de compilação da aplicação, fazendo todas as referências a bibliotecas necessárias. Sem um *script* ou algum outro método para automatizar o processo, todos Esses parâmetros teriam que ser digitados toda vez que a aplicação tiver que ser compilada, o que pode tomar muito tempo.

A aplicação desenvolvida para a captura de fotos é a **usbCapArgs**, um mnemônico para *Usb Capture with Arguments*, que descreve o fato de a aplicação realizar captura de imagens da câmera USB através da passagem de parâmetros. Todo o código da aplicação pode ser visto no Apêndice 9.4.6, que mostra o código-fonte **usbCapArgs.c**, utilizado para compilar o binário da aplicação.

Basicamente, a aplicação **usbCapArgs** funciona da seguinte forma:

```
$ ./usbCapArg tipo nome_da_figura.jpg
```

Em que tipo é um parâmetro que define se a foto a ser tirada é colorida ou preto e branca. Se o valor de tipo for “1”, a foto será colorida, e se for “2”, preto e branca. O *nome_da_figura.jpg* é o parâmetro que determina o nome a ser atribuído à imagem que será tirada. Pode ser qualquer expressão alfanumérica, contanto que possua a extensão ***.jpg**.

Para controlar a ação de tirar fotos através de requisições *web*, foi escrito o *script* PHP **php/camCapture.php**, mostrado por completo no Apêndice 9.1.2.

O *script* **camCapture.php** responde às seguintes requisições do tipo GET:

- **camCapture.php?cap:** Requisição que termina a tirada de uma foto naquele instante, por conta do parâmetro GET **cap**. Como não é passado o tipo de foto a ser tirada, se é colorida ou preto e branca, será tirada por padrão uma foto colorida, e o nome da foto seguirá o padrão de data e hora mostrado.

```
if(isset($_GET['cap']) && !isset($_GET['type'])){
    killStream();
    $img = system($camShellCMD);
    initStream($mjpgFile);
    writeLast($img,$file);
}
```

- **camCapture.php?cap&type=1:** Essa requisição determina a tirada de uma foto do tipo 1, que corresponde a uma foto colorida, no instante da requisição.
- **camCapture.php?cap&type=2:** Essa requisição determina a tirada de uma foto do tipo 2, que corresponde a uma foto preto e branca, no instante da requisição.
- **camCapture.php?last:** Requisição que retorna uma página contendo a última imagem tirada pela câmera.
- **camCapture.php?rmv:** Requisição que remove a última imagem tirada pela câmera.

Para salvar as imagens tiradas pelo **usbCapArgs**, foi escrito o *script* **cameraCapture.sh**, detalhado no Apêndice 9.4.1, e escrito em linguagem *Shell Script*. Esse *script* também funciona da mesma forma que o binário:

```
$ ./cameraCapture.sh tipo nome_do_arquivo.jpg
```

Porém, o funcionamento interno desse *script* é que é diferente. Basicamente, ele instancia os caminhos completos dos binários utilizados, tais como o **usbCapArgs** e **mjpg_streamer**, paralisa o *stream* de vídeo caso ele esteja em execução. Depois disso, o *script* obtém a data no formato “Dia-Mês-Ano—Horas-Minutos”. Esse foi o formato escolhido para ser o nome das fotos, pois basicamente descreve o instante em que a foto foi tirada. Finalmente, dependendo do tipo passado como argumento, a foto será tirada em colorido ou preto e branco, e o *script* utilizará como nome da foto o formato “Dia-Mês-Ano—Horas-Minutos”, seguido da extensão ***.jpg**. Após a tirada da foto, a mesma é movida para o diretório **/sdcard/pictures**, e o *stream* de vídeo é reinicializado.

A foto é movida para o diretório **/sdcard/pictures** por se tratar de utilizar a memória de armazenamento provida pelo cartão de memória externo, que possui mais memória.

Para exibir as imagens tiradas pela câmera, houve um problema devido ao fato de que o diretório **/sdcard/pictures** não é mapeado pelo *lighttpd*. Isto foi facilmente contornado com a criação de um *link* simbólico do diretório **/sdcard/pictures** para **/www/pictures**.

Um dos objetivos agregados ao projeto, com a criação da funcionalidade de tirar fotos pela câmera USB, era também permitir a tirada de fotos de modo temporizado, o que permite acompanhar a movimentação de pessoas e demais atividades no ambiente monitorado pela câmera USB.

Para alcançar esse objetivo, é necessário utilizar alguma ferramenta que permita agendar tarefas no sistema. Mais tarde, foi percebido que o **crond**, utilitário muito comum em sistemas Linux e Unix para agendamento de tarefas, não funcionava corretamente no **rootfs** fornecido pela FriendlyARM. O binário da aplicação estava presente no **rootfs**, mas por algum motivo o mesmo não era inicializado por padrão, e os arquivos de configuração de agendamento de tarefas não estavam presentes.

Para contornar esse problema, foi preciso criar o *script* **initCap.sh**, mostrado no Apêndice 9.4.3, e que faz toda a tarefa de criar os arquivos de configuração do **crond**, de instanciar a execução do **crond**, e de definir um agendamento para tirada de fotos a cada hora e meia.

Outro *script* foi criado para uso específico para tirada de fotos agendadas, que é o *script* **timeCapture.sh**. Esse *script* é praticamente igual ao **cameraCapture.sh**, com a diferença de já definir que as fotos tiradas serão coloridas. E é esse o *script* agendado para funcionamento toda vez que o *script* **initCap.sh** for executado.

Para permitir o disparo das requisições do *script camera.php*, foram criados as seguintes funções em JavaScript, as quais podem ser associadas a componentes HTML pra disparo das requisições:

```
function snapShot(type){
    $.get("php/camCapture.php?cap&type="+type);
}

function getLastPic(){
    $.get("php/camCapture.php?last", function(picture){
        myWindow=window.open("/pictures/"+picture);
        myWindow.focus();
    });
}

function delLastPic(){
    $.get("php/camCapture.php?delLast", function(retValue){
        //Funcao de retorno. Gera janela de aviso com confirmacao de
        imagem deletada.
        alert("Deseja realmente remover a foto: "+retValue+" ?");
    });
}
```

A função *splashot()* dispara a requisição para tirada de foto, recebendo e passando o parâmetro que determina o tipo de foto a ser tirada. A função *getLastPic()* faz a requisição para receber a última imagem tirada, e com os dados recebidos da requisição, essa função sintetiza uma página que será exibida para o usuário, contendo a última imagem tirada. Já a função *delLastPic()* envia a requisição de remover a última imagem tirada.

Por fim, é possível ver os controles implementados para a tirada de fotos da página Câmera na Figura 54. A região de controle está destacada dentro do retângulo de borda azul. O menu de seleção localizado à esquerda do botão **Tirar Foto** permite escolher se a foto deve ser colorida ou preto e branca. O botão **Tirar Foto** dispara a função *snapshot()*, que obtém o tipo de foto escolhido como parâmetro. O botão **Visualizar Foto** abre uma janela com a última imagem tirada pela câmera USB. E o botão **Excluir Foto** simplesmente exclui a última foto tirada pela câmera USB.

Para maiores detalhes a respeito da página Câmera, observar o Apêndice 9.3.3.

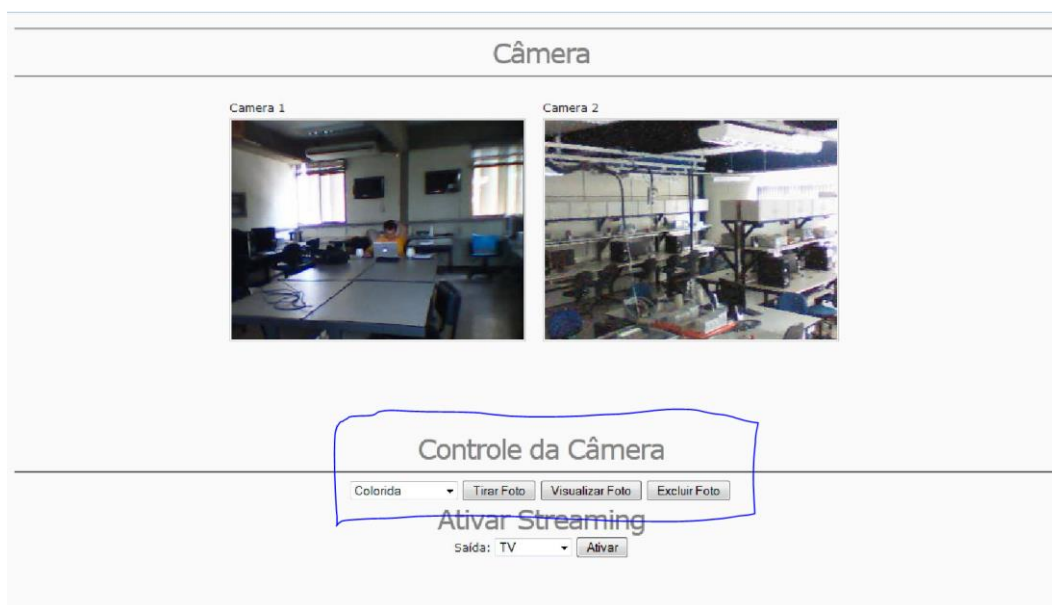


Figura 54 - Dessaque dos controles para tirada de fotos da página Câmera.

4.10 Banco de Dados SQLite , Persistência e Autenticação

Melhor do que escrever dados em forma de texto ou binária diretamente em arquivos, é utilizar um mecanismo de persistência de dados mais confiável, robusto, e fácil de manusear.

Com o passar do tempo no projeto, foi percebida a necessidade de programar um mecanismo de autenticação, de modo a permitir acesso a certos conteúdos da página do projeto somente para usuários autorizados. As funcionalidades de registros em tabelas de Bancos de Dados também foram necessárias na parte que trata do envio e recebimento de mensagens por serviços de radiofrequência, servindo para guardar os endereços IPs das máquinas utilizadas.

Dessa forma, de modo a experimentar e estudar um pouco a respeito do SQLite, que é uma base de dados leve e amplamente utilizada em Sistemas Embarcados, foi desenvolvido um mecanismo de autenticação de usuários, consistindo de uma tabela de usuários cadastrados e um *script* PHP que utiliza de funções de sessão e que realiza as requisições utilizando funções PDO.

Para compilar o pacote SQLite para a Tiny6410, bastou selecionar as opções **sqlite**, **Command-line editing** e **Additional query optimizations**, na opção **Database**, que pode ser

acessada em **Package Selection for the Target/Libraries**. Na Figura 55 é possível ver como devem ficar marcadas essas opções.

Após ter marcado as opções referentes ao **SQLite**, bastou sair do *Buildroot*, salvar o arquivo de configuração e digitar o comando **make** para iniciar a compilação. Com o término da compilação, foi copiado para a Tiny6410 o conteúdo dos diretórios **/usr/bin** e **/usr/lib**, que se encontram dentro do diretório **/output/target** do *Buildroot*. Feito isso, o binário e as bibliotecas do SQLite já estarão disponíveis no sistema.

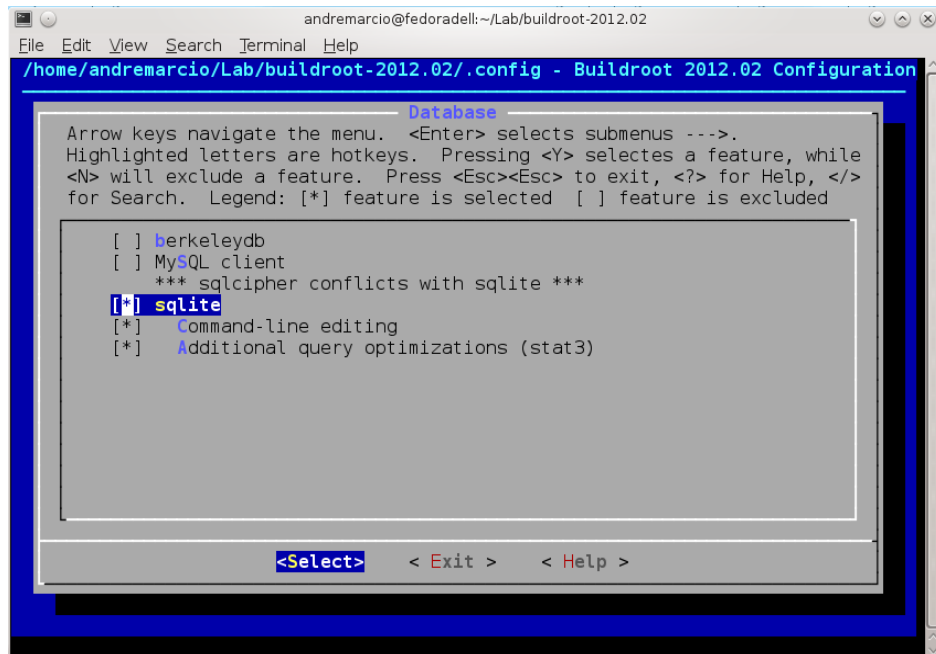


Figura 55 - Seleção dos pacotes SQLite no Buildroot

É preciso novamente lembrar que o PHP deve ser compilado com suporte a SQLite, caso contrário ele não possuirá as funções de acesso e consulta às tabelas do Banco de Dados.

Com o objetivo de preparar as tabelas necessárias para as funções de autenticação de usuário e de máquinas cadastradas para envio de mensagens por serviços de radiofrequência, foi criado o *script* **php/createDB.php**, mostrado por completo no Apêndice 9.1.3. Esse *script* basicamente trata de criar as tabelas necessárias e já preenchê-las com valores padrões.

As principais tabelas utilizadas no projeto foram:

- **db/login.sqlite**: Tabela responsável por armazenar os dados de *login* de usuários autenticados no sistema. É primeiramente preenchida com os dados do administrador, cujo *id* e *password* para acesso são ambos “admin”.

- **db/rf.sqlite**: Tabela responsável por armazenar os endereços IPs de máquinas compatíveis com os serviços de envio e recebimento de mensagens de radiofrequência.

Para o mecanismo de autenticação, foram criados o *script* **php/auth.php**, mostrado por completo no Apêndice 9.1.1, e o *script* **pages/login.php**, mostrado no Apêndice 9.3.8. Ambos os *scripts*, além das páginas Câmera e RF, fazem uso das funcionalidades de sessão para conferir se o usuário é autenticado e controlar o conteúdo a ser exibido.

O *script* **auth.php** recebe os parâmetros de *login* via requisições POST. Esse *script* consulta a tabela **login.sqlite**, e verifica se os dados recebidos, tais como nome de usuário e *password*, constam nessa tabela. Em caso afirmativo, a sessão do usuário é definida como autenticada. Em caso negativo, o usuário recebe a mensagem de erro “Falha no login! Tente novamente!”.

Já o *script* **login.php** é basicamente a tela na qual o usuário insere os dados, e ao clicar no botão **Submit** os dados são enviados via POST para **auth.php**. Um exemplo de como ficou a tela de *Login* na interface *web* é mostrado na Figura 56.



Figura 56 - Tela de *Login* da Interface *web*.

Caso um usuário não autenticado no sistema tente acessar a página Câmeras ou a página RF do Menu, ele receberá a tela mostrada na Figura 57.



Figura 57 - Aviso de autenticação para conteúdo protegido.

4.11 Sensor de Temperatura ds18b20

Os códigos-fontes utilizados na parte do sensor de temperatura foram encontrados em [14]. Esses mesmos códigos-fontes podem ser vistos nos Anexo 8.8, para o código do *driver* do sensor de temperatura, no Anexo 8.7, para o cabeçalho de GPIOs⁴⁵ da Tiny6410 e no Anexo 8.6, para a aplicação modelo que utiliza o dispositivo gerado pelo *driver* do sensor de temperatura. Outro arquivo importante é o *Makefile* do *driver* do sensor DS18B20 mostrado no Anexo 8.9. Esse *Makefile* automatiza o processo de compilação do *driver*.

Para utilizar o *Makefile*, basta modificar a variável **KERNELDIR** para o diretório em que o *Kernel* utilizado no projeto da aplicação está descompactado, e antes de proceder com a compilação é preciso garantir que os binários da *Toolchain* fornecido pela FriendlyARM estejam carregados na variável de ambiente *\$PATH* do Linux.

No caso desse projeto, o *Kernel* utilizado estava descompactado no diretório **/home/andremarcio/Lab/Kernels/Linux-2.6.38**.

Para finalmente compilar o *driver*, é preciso agrupar cada um dos arquivos relacionados ao *driver* em uma pasta específica. Feito isso, basta digitar o comando **make** para iniciar o processo de compilação.

Terminado o processo, o *driver* compilado é o arquivo ds18b20.ko. Para instalar esse *driver* no Linux da Tiny6410, basta transferi-lo para uma pasta do sistema de arquivos utilizando o FileZilla, e no terminal de comando da Estação digite o comando:

⁴⁵ Abreviação de *General Purpose In/Outs*.


```
$ insmod ds18b20.ko
```

Se não tiver ocorrido nenhum erro, o *driver* do sensor de temperatura estará instalado e pronto para uso, tendo sido instanciado como o dispositivo **/dev/ds18b20**.

Para evitar ter que carregar o *driver* do sensor de temperatura toda vez que a Estação for inicializada, recomenda-se acrescentar a linha “insmod /diretório/ds18b20.ko” no arquivo **/etc/init.d/rcS**, em que *diretório* é o caminho completo do arquivo **ds18b20.ko** no sistema de arquivos.

A aplicação teste fornecida no Fórum é a **ds18b20_test.c**. Para compilar essa aplicação com o **arm-linux-gcc**, basta proceder com o seguinte comando:

```
$ arm-linux-gcc ds18b20_test.c -o ds18b20_test
```

Esse comando irá gerar o binário **ds18b20_test**, que funciona em arquitetura ARM. Transfira o binário **ds18b20_test** para a Tiny6410, e digite o seguinte comando no terminal Linux da Estação:

```
$ ./ds18b20_test
```

Após digitar esse comando, irá aparecer uma sequência infinita de linhas, cada qual com a temperatura obtida pelo sensor de temperatura naquele instante.

Essa forma de retorno gerada pelo binário não é interessante para os propósitos dessa aplicação de temperatura, e dificultaria muito a obtenção desses dados por algum *script* PHP. Sendo assim, o arquivo **ds18b20_test.c** foi modificado, removendo-se o *loop* que criava as saídas infinitas de temperaturas, e colocando-se apenas uma única saída a cada chamada do binário. O código fonte do arquivo modificado é mostrado no Apêndice 9.4.7.

Com essa modificação, torna-se fácil utilizar esse binário em conjunto com algum *script* PHP, o qual executaria o binário e obteria a temperatura em forma de *string*.

Com o objetivo de permitir a obtenção da temperatura ambiente, e a exibição desse tipo de dado na página *web* do projeto, foi criado o *script* PHP **temp.php**, o qual é mostrado por completo no Apêndice 9.1.10.

Esse *script* é capaz de responder às seguintes requisições:

- **temp.php?temp=C**: Retorna a temperatura ambiente em graus *Celsius*.
- **temp.php?temp=F**: Retorna a temperatura ambiente em graus *Fahrenheit*.

- temp.php?temp=K: Retorna a temperatura ambiente em *Kelvin*.

Uma amostra do código PHP responsável por tratar as requisições comentadas é mostrada logo adiante. Esse trecho mostra as rotinas responsáveis por obter a temperatura do programa que lê esse dado do sensor de temperatura, e também mostra as rotinas que realizam os cálculos para retornarem a temperatura no formato pedido pela requisição.

```
if(isset($_GET['temp'])){
    $tempType = $_GET['temp'];

    if($tempType === 'C'){
        $temp = shell_exec($tempSensor);
        echo $temp;

    } else if($tempType === 'F'){
        $temp = shell_exec($tempSensor);
        $temp = ($temp/5)*9+32;
        echo $temp;

    } else if($tempType === 'K'){
        $temp = shell_exec($tempSensor);
        $temp = $temp + 273;
        echo $temp;
    }
}
```

Uma funcionalidade que foi experimentada nessa parte do projeto, foi a geração de um arquivo texto contendo dados de temperatura obtidos em um intervalo de tempo. A função *returnData()*, mostrada no trecho de código abaixo, é responsável por receber uma *string* de dados, preparar e retornar o arquivo texto como resultado da requisição.

```
function returnData($string){
    date_default_timezone_set('America/Sao_Paulo');
    $dataInfo = date("d-m-Y--H-i-s");

    header('Content-type: application/txt; charset=utf-8');
    header('Content-Disposition: attachment; filename="tempData.txt"');
    echo $string;
}
```

```

    echo "\r\n";
}

```

As funções que tratam de disparar as requisições de temperatura ao *script temp.php* estão escritas no arquivo **js/temp.js**, mostrado no Apêndice 9.2.5.

E para exibir maiores informações e controles a respeito dos dados de temperatura ambiente obtidos do sensor de temperatura, foi criada a página *Temperatura*, referente ao arquivo **pages/temp.php**, mostrado no Apêndice 9.3.11.

Por fim, a página Temperatura acabou ficando como mostrado na Figura 58. A parte superior, intitulada *Exibição da Temperatura Ambiente*, mostra em uma fonte azul, de tamanho maior, a temperatura ambiente, obtida pelo sensor, no momento em que a página foi carregada.

Figura 58 - Página Final de Exibição e Controle de dados de Temperatura

Para mudar a temperatura exibida, basta alterar o Menu de seleção da parte *Tipo de Temperatura*, que irá disparar a função JavaScript *changeTemp()*, que recebe o tipo de temperatura como parâmetro, e inicia as rotinas AJAX de atualização da temperatura ambiente.

Ao clicar no botão **Ativar**, será disparada a função JavaScript *startTemp()*, que também iniciará rotinas de obtenção da temperatura ambiente em AJAX, só que mostrando o conteúdo de modo gradativo e sequencial na parte *Saída dos dados de Temperatura*, ao mesmo tempo em

que a temperatura também aparece na *Exibição da Temperatura Ambiente*. Um exemplo desse tipo de execução é mostrado na Figura 59.

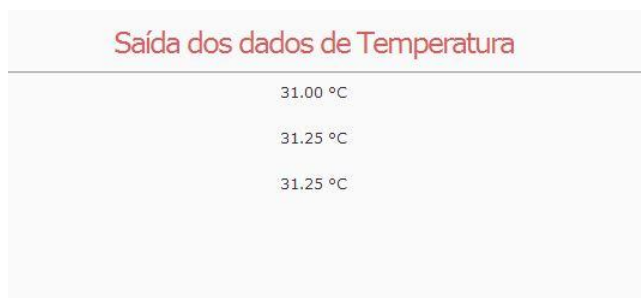


Figura 59 - Saída dos dados de Temperatura com atualização, após o usuário clicar no botão "Ativar".

Caso o usuário deseje obter um arquivo com os dados de temperatura em um intervalo de tempo definido em segundos, basta preencher o intervalo de tempo desejado no campo *segundos* da parte *Obtenção de Arquivo de Dados de Temperatura*. Ao clicar no botão **Salvar Dados**, é realizada uma requisição do tipo POST ao *script temp.php*, determinando a geração de um arquivo texto contendo as temperaturas obtidas no intervalo de tempo em segundos determinado pelo usuário, e no tipo selecionado pelo Menu de seleção da parte de *Obtenção de Arquivo de Dados de Temperatura*.

Para maiores detalhes a respeito de como está estruturado o arquivo referente à página *Temperatura*, favor consultar o Apêndice 9.3.11. Maiores detalhes sobre o *script* que trabalha as requisições de temperatura são mostrados no Apêndice 9.1.10, e as funções JavaScript escritas para o disparo das requisições com componentes HTML são mostradas no Apêndice 9.2.5.

Caso o usuário selecione na parte *Exibição da Temperatura Ambiente* a opção *Kelvin*, será mostrado algo semelhante ao observado na Figura 60.



Figura 60 - Saída da Exibição de Temperatura para Kelvin

Já no caso de seleccionar a opção Fahrenheit, será mostrado ao usuário algo parecido com o observado na Figura 61.



Figura 61 - Saída da Exibição de Temperatura com Seleção de Fahrenheit

4.12 GPS

Como dito nos Capítulos 2 e 3, o *chip* receptor de sinal GPS utilizado, o SiRF StarIII, é compatível com a especificação NMEA 0183, que define sequências de sentenças de dados GPS em caracteres ASCII, enviadas via serial.

Como também visto, o *chip* veio configurado por padrão para operar a 9600 bps, e foi conectado na porta **COM4** da Tiny6410, que é a porta ttySAC3 no Linux. Com base nessas informações, para então ser possível trabalhar e interpretar os dados GPS é preciso ter um programa capaz de realizar as seguintes ações:

1. Ler a porta serial na mesma velocidade com que o receptor GPS envia os dados.
2. Ser capaz de separar as sentenças com base nas vírgulas (“,”).
3. Trabalhar os dados obtidos pelas sentenças
4. Gerar resultado compreensível ao usuário.

Um programa escrito em linguagem C seria capaz de realizar Essas ações, mas como a Linguagem Python possui maiores facilidades para lidar com *strings*, ela foi escolhida para trabalhar como leitora dos dados oriundos do receptor GPS pela porta serial.

Infelizmente, o *rootfs* fornecido pela FriendlyARM não possui o interpretador Python instalado, nem algumas de suas bibliotecas mais importantes. Para compilar o interpretador para

Linux em arquitetura ARM, é feito uso do *Buildroot*, que automatiza o processo. No Menu do *Buildroot*, basta acessar a seguinte sequência de opções para selecionar os pacotes do interpretador Python:

Package Selection for the target -> Interpreter Languages and scripting

Nessa opção, foi selecionado o pacote **Python**. Com a seleção do pacote, aparecerão mais opções. Dentro das novas opções, foi marcado “**.py sources and .pyc compiled**” em **python module format to install**, e foram marcadas todas as opções de bibliotecas em **core python modules** e em **external python modules**. Um exemplo de como as opções devem ficar marcadas é mostrado na Figura 62.

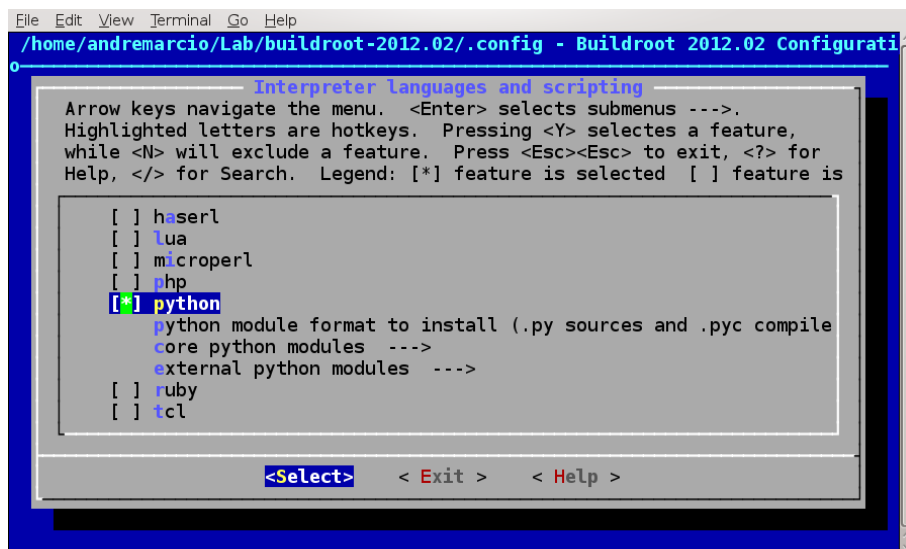


Figura 62 - Seleção de pacotes para o Interpretador Python - *Buildroot*.

Apesar de serem muitas bibliotecas, o tamanho delas em disco é reduzido, e permite o desenvolvimento futuro de novas aplicações. Além disso, a aplicação de envio de e-mail, a **mail.py** (mostrada no Apêndice 9.4.4), foi desenvolvida fazendo uso do interpretador Python e das bibliotecas compiladas.

No final, o *script* Python **gps.py** foi criado, e pode ser visto por completo no Apêndice 9.4.2.

Trabalhar com portas seriais em Python é bastante fácil, bastando instanciar alguns comandos, como mostra o trecho de código a seguir:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
```

```
import serial
import time
import sys

ser = serial.Serial("/dev/ttySAC3",
    baudrate=9600,
    bytesize=8,
    parity=serial.PARITY_NONE,
    stopbits=1,
    timeout=1)
```

Porém, foi percebido um problema ao utilizar o método mostrado para receber dados GPS: Assim que o *script* Python era chamado, ele já começava a ler a sentença GPS que estava sendo enviada. Isso pode parecer normal, mas não é quando é obtida uma sentença que está na metade, ou até mesmo no fim, como por exemplo:

```
N,00042.24,W,173.8,231.8,130694,004.2,W*70
```

Uma sentença desse tipo não permite saber qual é a sentença lida, pois faltam informações.

Dessa forma, esse tipo de leitura é capaz de complicar e gerar erros no processo. Para contornar isso foram adicionadas algumas condições à leitura dos dados da porta serial, de modo a iniciar a leitura somente com o início de uma nova sentença, o que se mostrou eficaz para evitar a leitura de sentenças já correntes.

De modo a permitir alguma maleabilidade com o *script* Python criado, que é o **gps.py**, ele foi escrito de modo a responder por três tipos de entradas:

- **Nenhuma entrada:** Caso o *script* seja executado da seguinte forma:

```
$ ./gps.py
```

A única sentença a ser lida e retornada na execução é a sentença **GPRMC**, definida como padrão, caso não seja passado nenhum argumento.

- **Uma entrada - Tipo de dado GPS:** Caso seja passado um argumento, esse argumento é o tipo de sentença de dado GPS. Para uma entrada como parâmetro, *script* pode ser executado da seguinte forma:

```
$ ./gps.py GPRMC
```

No exemplo mostrado, ele irá retornar uma sentença do tipo **GPGGA**, obtida pela leitura dos dados enviados pelo receptor GPS através da porta serial. Além das sentenças GPS, o valor do argumento passado pode ser *all*, que determina que será obtida a primeira sentença enviada após o comando de execução do programa, seja qual for a sentença.

- **Duas Entradas** - *Tipo de dado GPS e quantidade de amostras*: Nesse caso são passados dois argumentos ao **gps.py**, o primeiro determina o tipo de sentença GPS que será lida, e o segundo determina a quantidade de amostras dessa sentença que deverão ser retornadas com a execução. Para duas entradas como parâmetro, o *script* pode ser executado da seguinte forma:

```
$ ./gps.py all 10
```

No exemplo mostrado acima, o *script* irá retornar 10 sentenças GPS, de qualquer tipo, conforme forem enviadas pelo receptor GPS através da porta Serial. Para seleccionar uma outra quantidade ou um tipo específico de sentença, basta modifica os parâmetros passados como argumento.

Maiores detalhes a respeito do *script* **gps.py** são mostrados em seu código, no Apêndice 9.4.2.

Com um modo minimamente eficiente de obter sentenças GPS, quanto mais tipos específicos de sentenças, também é preciso processar essas sentenças e gerar saída para o usuário.

A melhor forma de fazer isso foi através de um *script* PHP, que é o arquivo **php/gps.php**, o qual é mostrado no Apêndice 9.1.5.

Esse *script* PHP, é responsável por controlar o *script* **gps.py**, receber as saídas geradas pelo **gps.py**, processar as sentenças e gerar uma saída como página *web* para o usuário cliente.

A principal requisição que o *script* **php/gps.php** recebe é a seguinte:

- **php/gps.php?pos:** Essa requisição define o processamento de dados GPS e a síntese de conteúdo HTML com Esses dados ao cliente.

Demais requisições encontram-se ainda em desenvolvimento, e os esqueletos das rotinas que tratam suas implementações podem ser vistos no Apêndice 9.1.5.

Logo adiante é mostrado o trecho de código responsável por tratar a requisição `php/gps.php?pos`.

```
if (isset($_GET['pos'])){
    $gpsShell = "/www/files/gps.py";

    $data = shell_exec("$gpsShell GPGGA");

    $gpsData = explode(",", $data);

    $type = $gpsData[0]; // Atribuição da variável de tipo de sentença.
    $time = $gpsData[1]; // Atribuição da variável de hora UTC.
    $latitude = $gpsData[2]; // Variável de dados de latitude.
    $latType = $gpsData[3]; // Variável de tipo de latitude (n/s)
    $longitude = $gpsData[4]; // Variável de dados de longitude
    $longType = $gpsData[5]; // Variável de tipo de longitude (e,w)
    $fixQ = $gpsData[6]; // Tipo de dado de correção
    $satellites = $gpsData[7]; // Número de satélites em vista.
    $horDil = $gpsData[8]; // Variável não necessária.
    $altitude = $gpsData[9]; // Variável de altitude
    $geoid = $gpsData[10]; // Variável não necessária.
    $checksum = $gpsData[13]; // Variável de verificação de erros.

    ## Tratamento da Hora
    $hours = substr($time, 0,2);
    $minutes = substr($time, 2,2);
    $seconds = substr($time, 4,2);

    ### Tratamento da Latitude
    $latDeg = (int) substr($latitude,0,2);
    $latMin = substr($latitude,2);

    ### Tratamento da Longitude
    $longDeg = (int) substr($longitude,0,3);
    $longMin = substr($longitude,3);

    echo "<table>";
    echo "<tr><td>Tipo de Dado:</td><td>$type</td></tr>";
    echo "<tr><td>Horário
```

```

UTC:</td><td>$hours".".$minutes".".$seconds."</td></tr>";
    echo "<tr><td>Latitude:</td><td> $latDeg"."°"." $latMin"."'". "
$latType</td></tr>";
    echo "<tr><td>Longitude:</td><td> $longDeg"."°"." $longMin"."'". "
$longType</td></tr>";
    echo "<tr><td>Número de Satélites:</td><td>$sattelites</td></tr>";
    echo "<tr><td>Altitude:</td><td> $altitude metros</td></tr>";
    echo "</table>";
}

```

Como pode ser visto no trecho de código mostrado acima, ao receber uma requisição válida, ele já executa o *script* **gps.py** com um argumento, determinando que deve ser retornada uma sentença do tipo **GPGGA**. Para poder processar essa sentença, cujos dados estão separados por vírgula, é utilizada a função PHP **explode()**, que cria um *array* de dados em que cada índice corresponde a valores da *string*, utilizada como parâmetro, separados por vírgula.

Com base nas características da sentença **GPGGA**, definidas pelo padrão NMEA 0183, cada índice do *array* é atribuído a variáveis específicas, que serão trabalhadas para mostrar em uma maneira compreensível ao usuário os dados obtidos, como visto no trecho de código mostrado anteriormente. Um dos problemas enfrentados nessa parte foi o tratamento dos dados de hora, latitude e longitude. Essas informações são enviadas pelo receptor de maneira agrupada, não formatada em uma maneira mais compreensível. Como por exemplo a hora, em se tratando da sentença **GPGGA**, ela é enviada no formato 134510, que significa 13:45:10. O mesmo praticamente acontece para os valores de latitude e longitude. Para formatar esses valores, foi usada a função PHP **substr**, que permite pegar pedaços específicos de uma *string*.

Assim, foram criadas mais variáveis, as quais receberam como valores pedaços específicos das *strings* de dados referentes a hora, latitude e longitude. Ao término das rotinas de processamento dos dados, é retornado ao usuário uma tabela contendo os dados obtidos, de maneira compreensível ao usuário.

A interface de controle de comandos e de visualização de resultados de dados GPS é o site *GPS*, escrito no *script* **pages/gps.php**, o qual é mostrado por completo no Apêndice 9.3.5.

Para auxiliar a realização de requisições ao *script* **php/gps.php**, foi criado também o *script* **js/gps.js**, que determina algumas funções que podem ser disparadas por meio de componentes HTML, tais como botões, *links* e até mesmo figuras.

A principal função JavaScript utilizada é a *gpsPosition()*, mostrada no trecho de código adiante. Essa função trata de fazer a requisição de dados GPS ao *script* **php/gps.php** com o parâmetro **pos**, e mostra a saída recebida do *script* na *div* HTML cujo *id* é “*gpsPos*”. Essa *div* está contida na página *GPS*.

```
function gpsPosition(){  
    $.get("php/gps.php?pos",function(gps){  
        $('#gpsPos').html(gps);  
    });  
}
```

O resultado final da página GPS é mostrado na Figura 63. O *Status de Posição GPS* é sintetizado com a requisição **php/gps.php?pos**, por meio da função JavaScript *gpsPosition()*. Essa requisição leva cerca de 2 segundos para ser processada e exibida ao usuário. As demais partes da página encontram-se ainda em desenvolvimento.

GPS

Status de Posição GPS

Tipo de Dado:	\$GPGGA
Horário UTC:	05:02:57
Latitude:	22° 00.3827' S
Longitude:	47° 53.8749' W
Número de Satélites:	09
Altitude:	850.6 metros

Controle do GPS

Tipo de Dado GPS:

Intervalo de Tempo:

Saída dos dados GPS:

Figura 63 - Página de Visualização e Controle de Dados GPS

4.13 Envio e recebimento de mensagens RF

O objetivo dessa funcionalidade é mostrar a integração entre sistemas diferentes, e ambientes diferentes. O sistema de radiofrequência está em operação em duas placas SAM9, que estão executando o sistema operacional Linux. Com algumas modificações, foi possível garantir, de uma maneira minimamente segura, o acesso externo às funcionalidades de envio e de recebimento de mensagens passadas por radiofrequência.

Para garantir que o sistema de radiofrequência fosse acessado a partir da própria Tiny6410, e não diretamente do *browser* cliente do *site web*, foi pensada e implementada a seguinte estrutura:

- Uma página *web* que deve servir de interface para o usuário selecionar uma máquina emissora, digitar uma mensagem e enviar essa mensagem para a máquina emissora, de modo que a máquina realize a transmissão da mensagem via radiofrequência. Nessa mesma página, o usuário também pode selecionar a máquina receptora e verificar se a mesma recebeu a mensagem enviada. No projeto, essa é a página *Rádio Frequência*, que corresponde ao arquivo **pages/rf.php**, mostrado por completo no Apêndice 9.3.10.
- Um *script* de controle PHP deve receber, através de requisições GET, tanto os dados selecionados quanto digitados pelo usuário na página *web*. Os dados devem ser processados e enviados para a Estação passada como parâmetro, quando a requisição é para enviar mensagem. No caso de uma requisição para receber a mensagem que foi enviada de uma outra Estação, o *script* deve fazer uma requisição específica na Estação alvo, e encaminhar os dados recebidos da requisição. Esse *script* corresponde ao arquivo **php/rf.php**, mostrado por completo no Apêndice 9.1.8.
- Para auxiliar a realização de requisições, foram criados também *scripts* em JavaScript, que permitem o disparo de requisições GET por meio de elementos HTML, tais como botões. Esses *scripts* estão no arquivo **js/rf.js**, mostrado por completo no Apêndice 9.2.4.

E para permitir que o sistema Linux da Tiny6410 realize diretamente as requisições *web* para as estações de transmissão de dados via radiofrequência, foi feito uso do *cURL* [18], que é uma aplicação que, dentre suas inúmeras funcionalidades, permite realizar requisições HTTP via linha de comando.

O pacote dessa aplicação está presente no *Buildroot*, em sua versão 2012.02. Para instalar os pacotes necessários, basta seguir a seguinte sequência de opções no *Buildroot*:

Package Selection for the Target -> Libraries -> Networking

Na opção **Networking**, é preciso habilitar o item **libcurl**. Ao ser habilitado, será mostrado o pacote **curl binary**, o qual também precisa ser marcado. Na Figura 64 é mostrado um modelo de como devem ficar marcados os itens para instalação do *cURL*. Marcados os pacotes, é preciso sair do *Buildroot*, salvar o arquivo de configuração e dar o comando **make** para iniciar o processo de compilação.

Terminado o processo, basta copiar para as pastas equivalentes no sistema da Tiny6410 o conteúdo dos diretórios **/usr/bin** e **/usr/lib**, presenter no diretório **/output/target** do *Buildroot*.



Figura 64 - Seleção dos Pacotes do cURL no Buildroot

Para permitir o suporte à mais estações, essa parte do projeto foi implementada fazendo uso do Banco de Dados SQLite. Basicamente, o *script* **php/rf.php** consulta a tabela **db/rf.sqlite** para obter os endereços IP das estações cadastradas. Cada uma das estações possui as funcionalidades de enviar e de receber mensagens via radiofrequência, as quais podem ser acessadas pelo **php/rf.php**.

O *script* **php/rf.php** responde a duas requisições:

- **php/rf.php?mach=emissor&msg=mensagem:** Essa requisição define uma máquina emissora, pelo parâmetro GET **mach**, e a mensagem que será enviada à máquina, pelo parâmetro GET **msg**. A requisição encaminha qualquer resposta gerada pela máquina emissora.

- **php/rf.php?mach=receptor&hist:** Essa requisição define uma máquina receptora, pelo parâmetro GET **mach**, e que a requisição é para obter a última mensagem enviada, o que é determinado pelo parâmetro GET **hist**. Por restrições das estações de radiofrequência, o *script* aguarda 4 segundos antes de disparar a requisição para a máquina definida como receptora.

As funções implementadas em JavaScript *sendRf()* e *receiveRf()*, servem para fazer as requisições de envio e recebimento de mensagens, e mostram o resultado obtido do **php/rf.php** na parte *Mensagens Obtidas*, que na página *Rádio Frequência* corresponde ao espaço com *id* “tempData”.

O resultado final da página *Rádio Frequência* é mostrado na Figura 65. Na figura, é possível observar a caixa de seleção do emissor, e a caixa de texto a ser digitado. Como função complementar, foi incluída uma função no *script js/rf.js* que conta quantos caracteres foram digitados, dada a limitação de envio de 96 caracteres.

Ao clicar no botão **Enviar Mensagem**, a mensagem digitada na caixa de texto será processada pelo *script sendRf()* do arquivo **js/rf.js**, e enviada ao *script php/rf.php*. Esse, por sua vez, irá receber o destinatário e a mensagem como parâmetros. Antes de enviar a mensagem, o *script* substitui todos os espaços por “%20”, equivalente ao espaço em endereços URL, algo que foi necessário para utilizar corretamente o *cURL*. Caso contrário, sem essa substituição, o *cURL* enviaria somente a primeira palavra do texto digitado.

E ao clicar no botão **Receber Dados**, será disparada a função *receiveRf()*, a qual realiza a requisição para obter a última mensagem recebida, na máquina definida no Menu de seleção **Receptor**, mostrando o resultado na parte *Mensagens Obtidas*.

Radio Frequência

Controle de Mensagens RF

Emissor: Máquina 1 ▼

Texto:

96

Enviar Mensagem

Receptor: Máquina 1 ▼

Receber Dados

Mensagens Obtidas

Figura 65 - Página de Controle de Envio e Recebimento de Mensagens de Radiofrequência.

Na Figura 66 é mostrado um exemplo de funcionamento do envio de mensagem. Como exemplo, foi digitada a mensagem “Ola mundo”, e selecionado como emissor a **Máquina 1**. Logo após isso, foi clicado o botão **Enviar Mensagem**, e é mostrado na parte *Mensagens Obtidas* o resultado retornado pelo *script* **php/rf.php**, que por sua vez também realizou uma requisição à máquina alvo e seu retorno gerado basicamente é a resposta recebida da máquina alvo.

Já na Figura 67 é mostrado um exemplo da opção de receber mensagens. Nesse exemplo, foi selecionado como receptor a **Máquina 2**, e o botão **Receber Dados** foi clicado. Poucos segundos depois, o resultado foi exibido na parte *Mensagens Obtidas*.

Emissor: Máquina 1 ▼

Texto:

Ola mundo!

83

Enviar Mensagem

Receptor: Máquina 2 ▼

Receber Dados

Mensagens Obtidas

143.107.235.53: A mensagem: 'Ola mundo!' foi enviada!

Figura 66 - Resultado mostrado após o usuário digitar uma mensagem e clicar no botão "Enviar Mensagem".

Receptor: Máquina 2 ▼

Receber Dados

Mensagens Obtidas

143.107.235.51: Ola mundo!

Figura 67 - Resultado mostrado após o usuário selecionar um Receptor e clicar no botão "Receber Dados".

4.14 Painel de Administração, Informações sobre a Estação e Controles Adicionais

Para fornecer um relatório geral sobre o *status* do sistema, como memória RAM e memória NAND disponíveis, uso do processador, quantidade de processos em execução, foi desenvolvida uma página inteiramente dedicada para esse fim, que é a página *Administração*, referente ao arquivo **pages/admin.php**, mostrada por completo no Apêndice 9.3.2.

Essa página também trata de permitir que usuários autenticados no sistema possam exercer algum tipo de controle, tal como encerrar ou iniciar o processo de *stream* de vídeo, além de controlar periféricos da placa, como o *display* LCD e o *backlight* do *display* LCD. Caso o

usuário não seja autenticado, será exibido apenas o relatório geral do sistema, correspondente à parte *Dados da Estação*,

Com relação à parte *Dados da Estação*, o código responsável por obter, processar e exibir dados da Estação é mostrado a seguir. Essa sequência de códigos faz uso de algumas funções já prontas do PHP, outras que foram implementadas, como a *uptime()*, e também faz uso de comandos *shell* realizados através da função **shell_exec()**.

```
<h3>Dados da Estação:</h3>
<table id="stationData">
  <tr>
    <td>Tempo em operação:</td> <td><?php echo uptime(); ?></td>
  </tr>
  <tr>
    <td>Processador:</td><td>ARM 1176JZF-S</td>
  </tr>
  <tr>
    <td>Frequência Proc.:</td> <td><?php
$cpufreqAux= shell_exec("cat /proc/cpuinfo | grep BogoMIPS");
$cpufreq = explode(":", $cpufreqAux);
echo "$cpufreq[1] Mhz";
?>
</td>
  </tr>
  <tr>
    <td>Sistema Operacional:</td> <td><?php echo shell_exec("uname
-r");?></td>
  </tr>
  <tr>
    <td>Memória RAM Total:</td> <td><?php
$ramAux = shell_exec("free | grep Mem");
$vecRam = explode(" ", $ramAux);
echo round($vecRam[1]/1024,2) ." MB";
?></td>
  </tr>
  <tr>
    <td>Memória RAM Disponível:</td> <td><?php echo
round($vecRam[4]/1024,2) ." MB";?></td>
  </tr>
```

```

        <tr>
            <td>Memória RAM Utilizada:</td> <td><?php echo
round($vecRam[3]/1024,2) ." MB";?></td>
        </tr>
        <tr>
            <td>Número de Processos Ativos:</td> <td><?php echo
shell_exec("ps | wc -l");?></td>
        </tr>
        <tr>
            <td>Memória FLASH Total:</td> <td><?php echo
round(disk_total_space("/sdcard")/1024/1024,2) ." MB"; ?></td>
        </tr>
        <tr>
            <td>Memória FLASH Utilizada:</td> <td><?php echo
round((disk_total_space("/sdcard")-disk_free_space("/sdcard"))/1024/1024,2)
." MB";?></td>
        </tr>
        <tr>
            <td>Memória FLASH Disponível:</td> <td><?php echo
round(disk_free_space("/sdcard")/1024/1024,2) ." MB"; ?></td>
        </tr>
        <tr>
            <td>Uso da Banda de Rede:</td> <td><?php ?></td>
        </tr>
        <tr>
            <td>Mídia em Execução:</td>
        </tr>
        <tr>
            <td id="midia"></td>
        </tr>
    </table>

```

Como visto no trecho de código mostrado, foram utilizadas funções do PHP tais como **disk_total_space()**, que mostra o espaço total de um diretório passado como argumento, e **disk_free_space()**, que mostra o espaço livre no diretório passado como argumento. A função *uptime()* é mostrada por completo no Apêndice 9.3.2, mas basicamente o que ela faz é pegar os dados retornados pelo comando *uptime* do Linux, e mostra-los de maneira mais compreensível, permitindo ao usuário saber por quanto tempo a máquina está ligada. O uso da função PHP

shell_exec() permitiu executar comandos de console *shell* tais como “*free*”, “*cat /proc/cpuinfo*” e “*ps | wc -l*”, que retornam informações a respeito da memória RAM disponível, a respeito da CPU, e a quantidade de processos em execução, respectivamente.

As partes responsáveis pelo *Uso da Banda de Rede* e da *Mídia em Execução* estão em desenvolvimento para futuras versões do projeto.

Já a parte de *Controle de Sistema*, mostrada na Figura 68, corresponde à seguinte sequência de códigos HTML:

```
<h3>Controle de Sistema</h3>
<table>
<tr>
    <td><input type="button" value="Iniciar QTopia"
onclick="startQtopia()"/></td>
    <td><input type="button" value="Fechar QTopia"
onclick="stopQtopia()"/></td>
</tr>
<tr>
<!-- Trocar parametros aqui entre lcd e backlight -->
    <td><input type="button" value="Ligar Display LCD"
onclick="lcdOn()"/></td>
    <td><input type="button" value="Desligar Display LCD"
onclick="lcdOff()"/></td>
</tr>
<tr>
    <td><input type="button" value="Ligar Backlight"
onclick="backLightOn()"/></td>
    <td><input type="button" value="Desligar Backlight"
onclick="backLightOff()"/></td>
</tr>
<tr>
    <td><input type="button" value="Reiniciar Sistema"
onclick="restartSystem();"/></td>
</tr>
<tr>
    <td><input type="button" value="Controle de Usuários"
onclick="parent.location='index.php?p=users'"/></td>
</tr>
</table>
```

Com base nessa sequência de código HTML, é possível ver como os elementos de controle foram organizados em forma de tabela, por meio das *tags* HTML `<table>`. Dentro dessa *tag*, são usadas as *tags* `<tr>`, que define uma linha, e `<td>`, que define um componente dentro dessa linha. Cada um dos componentes é definido como uma entrada, por conta da tag `<input .../>` do tipo **button**, que define um botão em HTML. Pode-se observar que nas *tags* *input* são definidas funções no parâmetro *onclick*. Com exceção da função disparada no botão **Controle de Usuário**, cada uma dessas funções estão implementadas no arquivo **js/ajax.js**, mostrado no Apêndice 9.2.1. As funções desse arquivo fazem requisições AJAX ao *script* **php/status.php**, que trata de controlar os periféricos conforme as requisições recebidas.

O resultado final da página *Administração* é mostrado na Figura 68.

Administração

Dados da Estação:

Tempo em operação:	11 horas
Processador:	ARM 1176JZF-S
Frequência Proc.:	530.84 Mhz
Sistema Operacional:	2.6.38-FriendlyARM
Memória RAM Total:	211.49 MB
Memória RAM Disponível:	173.37 MB
Memória RAM Utilizada:	38.13 MB
Número de Processos Ativos:	50
Memória FLASH Total:	3630 MB
Memória FLASH Utilizada:	293.4 MB
Memória FLASH Disponível:	3336.6 MB
Uso da Banda de Rede:	
Mídia em Execução:	

MJPEG Streamer Control

Controle da Temporização de Captura de Fotos:

Configuração Atual:

Alterar (Minutos):

Formatar Dados?

Controle de Sistema

Figura 68 - Visão geral do Painel de Administração e Controle

4.15 Painel de Controle Multimídia e Status da Estação

O Painel de Controle Multimídia, localizado na coluna da direita da página do sistema, foi pensado como uma forma de tanto controlar algumas funcionalidades de execução multimídia do sistema, tais como aumentar ou diminuir a intensidade do Volume da música que

está sendo tocada, paralisar a execução da música ou vídeo corrente, dentre outros, como também visualizar informações sobre a Estação, como temperatura e uso de CPU.

Como mostrado no Capítulo 4.5, o *script* **mplayer.php** responde a algumas requisições GET que permitem alterar o a intensidade do volume, paralisar ou encerrar a execução de uma música ou vídeo, *etc.* Dessa forma, bastou a criação de um *script* escrito em JavaScript, chamado **mediaControl.js**, e mostrado no Apêndice 9.2.3 , contendo funções que permitem o controle dessas requisições por meio de componentes HTML, tais como botões.

Fazendo uso então dessas funcionalidades do HTML, foram criados botões para controle de funcionalidades multimídia, e ao pressionar Esses botões, os mesmos disparavam requisições GET, para dado tipo de função associada ao botão.

Por ser um trecho de código pequeno e razoavelmente simples, o código HTML responsável por mostrar de maneira tabular os botões, e também associar funções JavaScript aos mesmos, é mostrado logo adiante.

```
<div id="mediaControl" align="center">
  <h2>Controle</h2>
  <table>
    <tr>
      <td><input type="button" value="Play" onclick="play();" /></td>
      <td><input type="button" value="Pause"
onclick="pause();" /></td>
      <td><input type="button" value="Stop"
onclick="stop();" /></td>
    </tr>
    <tr>
      <td><input type="button" value="Next" onclick="next();" /></td>
      <td><input type="button" value="Prev" onclick="prev();" /></td>
    </tr>
    <tr>
      <td><input type="button" value="Vol+"
onclick="volumeUp();" /></td>
    </tr>
    <tr>
      <td>Volume:</td><td><p id="volume">50</p></td>
    </tr>
  </table>
</div>
```

```

<tr>
    <td><input type="button" value="Vol-"
onclick="volumeDown();" /></td>
</tr>
<tr>
    <td><input type="button" value="Mute" onclick="mute()" /></td>
</tr>
</table>
<hr/>
</div>

```

Esse trecho de código mostrado está presente no *script index.php*, que é o arquivo principal da página do sistema. Pode ser observado no trecho de código que foi utilizada uma estrutura de exibição tabelada, por conta do uso das *tags* HTML *<table>*, que define uma tabela, *<tr>*, que define uma linha, e *<td>*, que define um componente dentro dessa linha. E cada componente utilizado é botão, o que é definido pela *tag* HTML *<input type="button".../>*. Os nomes associados aos parâmetros *value* dos botões serão os nomes exibidos nos botões, na página. E o parâmetro “*onclick=...*” define uma função JavaScript que será chamada no clique daquele botão. Cada botão foi associado a sua correspondente função.

O resultado final da parte responsável pelo controle de funções multimídia é mostrado na Figura 69.



Figura 69 - Painel de Controle de Funções Multimídia

Complementando o painel, foram implementadas e agregadas as funcionalidades de exibição da temperatura ambiente, da carga da CPU, do *status* do *stream* de vídeo e do *status* do MPlayer. Tais funcionalidades também foram implementadas de modo a serem atualizadas em intervalos específicos, graças a funcionalidades da linguagem JavaScript.

O *script* **status.php**, mostrado no Apêndice 9.1.10, foi criado e implementado com o objetivo de fornecer Esses dados a respeito da Estação, e permitir também um certo controle, tal como iniciar o *stream* de vídeo, dentre outros.

Para interagir com as requisições de informação sobre a Estação, foi criado o *script* JavaScript **ajax.js**, que faz uso do *framework* jQuery, e pode ser visto no Apêndice 9.2.1. Esse *script* trata de realizar uma série de requisições do tipo AJAX, ou seja, transparentes ao usuário e que atualizam pedaços da página, e não a página inteira. Para cada tipo de requisição, o *script* associa *ids* HTML específicas, que serão preenchidas com as respostas obtidas das requisições.

Para obter a temperatura ambiente, o *scrip* **status.php** basicamente executa o binário que obtém a temperatura pelo sensor ds18b20 do *kit*, e como esse binário já retorna a temperatura em graus Celsius, basta utilizar esse valor, que é inserido na *tag* que possuir a *id temp* na página.

Para a carga da CPU, a melhor e menos trabalhosa maneira de medir foi através da função PHP `sys_getloadavg()`, que retorna valores de carga da CPU nos últimos 1, 5 e 10 minutos. Pegando-se então o primeiro valor desse *array*, é possível obter a carga da CPU no último 1 minuto. Não é algo muito preciso, mas ajuda a acompanhar a oscilação de carga sobre a CPU da Estação.

```
if(isset($_GET['cpuload'])){

    $load = sys_getloadavg();
    $load = (floatval($load[0]));
    $load = $load*100;
    echo "<p style=\"text-align:center;\">$load %</p>";

}
```

Já com relação aos *status* tanto do MPlayer quanto do MJPG Streamer, é necessário consultar o PI^{46} desses processos. Com a execução do comando “**pidof nome do programa**”, que retorna o número PID do processo, é possível saber se o MPlayer ou o MJPG Streamer estão em execução.

Se a execução do comando “pidof mplayer” retornar um número, significa que o MPlayer está em execução, e aquele número é seu identificador de processo (PID).. Se não retornar um número, significa que o programa não está em execução. Vale ressaltar que é importante saber bem o nome do programa, caso contrário, pode-se pensar que um dado

⁴⁶ PID é a abreviação de *Process ID*.

programa não está em execução, quando na verdade ele possui um nome diferente daquele utilizado na chamada do **pidof**.

Trabalhando com Essas variáveis, o *script* **status.php** retorna a mensagem *MPlayer is up* na cor azul, quando o MPlayer está em execução, ou a mensagem *MPlayer is down*, em vermelho, quando o MPlayer não está em execução. Ambas as mensagens são exibidas na *div* HTML de id *mpStatus*, na página **index.php**.

```
if(isset($_GET['mpstatus'])){
    $pidMp = shell_exec("pidof mplayer");
    if(isset($pidMp)){
        //Ver o que Essa em execução
        echo "<p style='color:blue; text-align:center;'>MPlayer is
Up</p>";
    }
    else echo "<p style='color:red; text-align:center;'>MPlayer is
Down</p>";
}
```

Já para o MJPG Streamer, são exibidas as mensagens *MJPG Streamer is Up* em azul, quando o MJPG Streamer está em execução, e *MJPG Streamer is Down*, em vermelho, quando o MJPG Streamer não está em execução. Ambas as mensagens são exibidas na *div* HTML de id *statusServer*, na página **index.php**.

```
if(isset($_GET['mjpgs'])){

    $mjpgStatus = shell_exec("pidof mjpg_streamer");

    if(strlen($mjpgStatus)===0){
        echo "<p style='color:red; text-align:center;'>MJPG Streamer
DOWN</p>";
    }
    else {
        echo "<p style='color:blue; text-align:center;'>MJPG Streamer
UP</p>";
    }

}
```

Os resultados finais das funcionalidades implementadas para o Painel de Controle Multimídia e Status da Estação podem ser vistos nas Figuras 70 e 71. Nessas figuras, é possível observar a *Temperatura Local*, mostrda no topo do Painel. Na parte *Carga da CPU* é exibida a carga da CPU, e nas partes *MJPEG Stream Status* e *Mídia em Execução* são mostrados os *status* do *MJPEG Streamer* e do *Mplayer*, indicando se estão em execução ou não.



Figura 70 - Painel de Controle de Funcionalidades Multimídia. É possível ver a transição do *status* do *Mplayer* na parte *Mídia em Execução*.

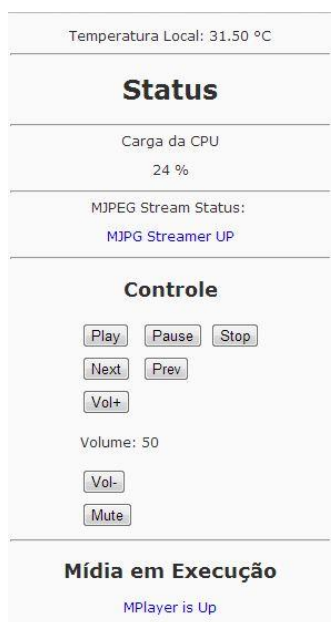


Figura 71 - Painel de Controle Multimídia. É possível ver que o *status* do *MJPEG Streamer* é mostrado como ativo.

4.16 Envio de E-mail

Como dito anteriormente, não é possível utilizar a aplicação *sendmail* do Linux/Unix para envio de e-mail. E a alternativa encontrada e trabalhada foi a de utilizar a biblioteca de Python **smtplib**, que é capaz de realizar conexão com provedores de e-mail que suportem o protocolo de e-mail SMTP⁴⁷.

Essa biblioteca pode ser inserida por padrão, no momento da configuração de compilação do pacote de Python no *Buildroot*. Basta garantir que foi selecionado Python e todas as bibliotecas, na parte de pacotes do *Buildroot*, e também garantir que Esses arquivos Essejam nos devidos diretórios do sistema.

Após isso, para utilizar essa biblioteca, basta colocar uma linha “import smtplib” no início do arquivo, que então é possível utilizar os métodos da biblioteca.

De modo a permitir uma interação entre a página, controlada por PHP, e o *script* em Python, foi utilizada também a biblioteca **sys**, que permite ao *script* Python receber parâmetros de entrada do sistema operacional. E, para lidar com eventuais manipulações de *string*, foi também utilizada a biblioteca **string**.

Para utilizar esse tipo de serviço, e permitir que a placa envie e-mails, foi criada uma conta de e-mail no GMail, chamada tiny6410@gmail.com, e pela qual a placa é então capaz de enviar e receber e-mails.

Em posse das bibliotecas necessárias, e de uma conta de e-mail, é preciso então programar um *script* para receber uma mensagem como parâmetro e enviá-la por e-mail.

A melhor forma pensada para permitir que o sistema envie e-mails foi a de determinar remetentes específicos, os quais são os mantenedores do projeto, e obter o assunto da mensagem e o corpo da mensagem por passagem de parâmetros em linha de comando. Considerando que o *script* desenvolvido foi o **mail.py**, mostrado no Apêndice 9.4.4, a forma de utilizá-lo é:

```
$ ./mail.py “Ola mundo” “E aí pessoal?”
```

De modo a entender melhor o funcionamento do *script*, é mostrado o seguinte trecho de código em Python, muito parecido com o do Apêndice 9.4.4.

```
#!/usr/bin/python
import sys
import smtplib
```

⁴⁷ Abreviação de *Simple Media Transfer Protocol*, é um protocolo comumente utilizado para envio e recebimento de *e-mails*.

```
import string

#Destinatário
FROM = "tiny6410@gmail.com"

#Remetente
to = andre.ml.curvello@gmail.com

#Assunto do e-mail
SUBJECT = sys.argv[1]

#Informações de login
username = "tiny6410"
password = "tiny6410armwe"

#Mensagem a ser enviada
MESSAGE = sys.argv[2]

#Tratamento da mensagem a ser enviada
BODY = string.join((
    "From: %s" % FROM,
    "To: %s" %to1,
    "Subject: %s" % SUBJECT,
    "",
    MESSAGE),"\r\n")

#Estabelecimento da conexão
server = smtplib.SMTP('smtp.gmail.com:587')
server.starttls()

#Autenticação
server.login(username,password)

#Envio da mensagem
server.sendmail(FROM,to1,BODY)

#Fecha a conexão.
server.quit()
```

Essa foi só a parte que tange ao *script* Python. Ainda falta mencionar a página PHP e alguns *scripts* JavaScript que cuidam da passagem de parâmetros.

A página PHP responsável pelo formulário de envio de e-mail de contato é mostrada no Apêndice 9.3.4. Os dados que o usuário irá preencher para contato irão compor o corpo do e-mail que será enviado. E o assunto do e-mail foi decidido como sendo *Recado Data*, sendo Data a informação completa de Data em que a mensagem foi enviada, como por exemplo *Recado 25-10-2012—12-45-16*, que indica uma mensagem enviada no dia 25 de Outubro de 2012, às 12 horas e 45 minutos e 16 segundos.

Os campos preenchidos no formulário de contato são enviados via GET para o *script* **mail.php**, que trata Esses dados, cria o assunto da mensagem com base na data em que foi enviada, e executa o *script* Python passando os parâmetros necessários.

```
<?php

$mailShell = "/www/files/mail.py";

if (isset($_GET['nomeCtc']) && isset($_GET['mailCtc']) &&
isset($_GET['telCtc']) && isset($_GET['message'])) {
    $nomeCtc = $_GET['nomeCtc'];
    $mailCtc = $_GET['mailCtc'];
    $telCtc = $_GET['telCtc'];
    $msg = $_GET['message'];

    $message = "\nNOME: $nomeCtc\r\n
\nE-Mail: $mailCtc\r\n
\nTelefone: $telCtc\r\n
\nMensagem: \r\n $msg";

    $subject = "Recado: ".date("d-m-Y--H-i-s");
    $status = shell_exec("$mailShell \"$subject\" \"$message\"");

    echo $status;
}

?>
```

Na página de Contato, ao clicar no botão **Enviar**, será disparada a função JavaScript *sendEmail()*, mostrada no trecho de código abaixo.

```
function sendEmail(name,mail,tel,msg){  
    var message = msg.replace(/\r\n|\r|\n/g,"\\n");  
    $.get("/php/mail.php?nomeCtc="+name+"&mailCtc="+mail+"&telCtc="+tel+"&  
message="+message,function(status){  
        });  
}
```

Considerando apenas a coluna central da página desenvolvida para o sistema, o resultado final pode ser visto na Figura 72.



Contato

Formulário para Contato

Nome:

E-mail:

Telefone:

Texto:

Figura 72 - Página de Contato via E-mail

Vários testes foram realizados com o envio de mensagens de contato através do *site*, e todos resultaram em êxito, com as mensagens recebidas pelos destinatários programados, comprovando a funcionalidade da forma de envio de e-mail utilizado.

5 Conclusão

A realização do projeto tornou possível utilizar uma série de conceitos, ferramentas e técnicas aprendidos durante o curso de graduação em Engenharia de Computação para a formação do sistema final como um todo.

Os conhecimentos adquiridos em programação, em redes de computadores e em sistemas embarcados foram essenciais para permitir a realização do projeto. Caso contrário, muito tempo seria gasto no aprendizado de tais conhecimentos, e o curso de Engenharia de Computação, ministrado pela Universidade de São Paulo, foi um curso singular, que forneceu os alicerces conceituais para o desenvolvimento do projeto.

O projeto também permitiu mostrar a importância do correto acoplamento entre o desenvolvimento dos componentes de sistema, tais como *Kernel*, *rootfs* e *bootloader*, e das aplicações que são executadas sobre a plataforma que possui esses componentes. Em outras palavras, é preciso projetar não somente as aplicações de usuário, mas todo o sistema que será utilizado.

Além disso, foi possível mostrar como elementos diferentes, componentes diferentes, e até mesmo programas escritos em linguagens de programação diferentes podem interagir entre si para compor um sistema, uma plataforma final.

O projeto também foi capaz de gerar e agregar muita informação a respeito do desenvolvimento de plataformas Linux embarcadas, através do uso de ferramentas de *buildsystem* e *toolchains*, e também pelo desenvolvimento de sistemas e de interfaces *web*.

Por sua organização em forma de tutorial, a presente monografia poderá ser utilizada como valioso ativo no aprendizado de ferramentas e no desenvolvimento de soluções Linux embarcadas. Com isso, conclui-se que o viés didático com que este texto foi escrito alcançou seu objetivo maior.

Considerando a atual cenário de uso de Linux embarcado em dispositivos ARM, e tendo em vista que esse uso é crescente e em larga escala, é de se considerar Linux embarcado como uma plataforma a ser levada a sério. A demanda por produtos desenvolvidos com Linux embarcado leva a também uma demanda por profissionais qualificados para o desenvolvimento de tais produtos em empresas e indústrias.

Dessa forma, para ajudar em uma espécie de pré-qualificação para profissionais em desenvolvimento de Linux embarcado, o presente documento foi escrito em forma bem

detalhada, seguindo uma estrutura de tutorial, para auxiliar futuros alunos a se capacitarem nessa área que demanda cada vez mais profissionais.

Bem recebido, o projeto possui algumas de suas funcionalidades estudadas para a formação de potenciais produtos de mercado, envolvendo áreas de monitoramento remoto, de automação residencial e empresarial.

Com base nas funcionalidades apresentadas, é possível desenvolver produtos, ou simplesmente projetos, envolvendo monitoramento remoto com aquisição de dados de temperatura e de GPS, ao mesmo tempo em que o ambiente é monitorado visualmente. E também é possível desenvolver todo um mecanismo de automação residencial controlado por uma interface *web*, acessado por um dispositivo móvel. Enfim, as possibilidades são diversas, e algumas sugestões são apresentadas no Capítulo 6.

Com foco na criação de produtos, com a redução dos custos em equipamento, como mostrado em 3.1, é possível desenvolver equipamentos com Linux embarcado com preços mais reduzidos, o que facilita a adoção do produto pelo mercado.

6 Novas Possibilidades

Com base nas aplicações desenvolvidas, e em outras disponíveis pela *web*, serão descritas a seguir algumas sugestões de aplicações que poderiam ser desenvolvidas em cima da plataforma utilizada para o presente projeto.

Considerando a disseminação de dispositivos *smartphones*, citada no Capítulo 1, é possível criar uma interface de acesso *web* específica para dispositivos móveis, modelada para as dimensões da tela do dispositivo. Para isso, basta criar um *script* de estilo CSS com os parâmetros necessários, e utilizar de funções do PHP para usar o *script* CSS para dispositivos móveis quando for detectada uma requisição de acesso à página vinda de um dispositivo móvel.

Com o uso do interpretador Python disponível, é possível instalar e utilizar uma série de APIs disponíveis, tais como a Google Drive [35] e o DropBox API [19]. A API do DropBox torna possível o uso dos recursos de armazenamento de dados na nuvem, providos pela empresa.

Já o uso da API do Google Drive torna possível criar e modificar planilhas, fazer *upload* de fotos, etc. E como se não fosse suficiente, é possível implementar aplicações clientes do serviço GTalk, por meio da biblioteca Python XMPPY [97]. Com isso, o sistema poderia interpretar comandos da mesma forma com que se conversa com alguém em um *chat*.

Seria muito interessante também a implementação de alguma forma de interação com mídias sociais, tais como Twitter⁴⁸, que possui uma API em linguagem Python, documentada na página <http://dev.twitter.com/doc>. Com uso dessa API, uma execução de música poderia ser noticiada no Twitter, por exemplo.

Além disso, fazendo uso de algumas bibliotecas JavaScript, tais como HighCharts [40], é possível também gerar gráficos para análise de dados na própria página *web*.

Outro uso que não foi implementado, é a funcionalidade de controle por infravermelho do *kit* FriendlyARM Tiny6410. Essa funcionalidade poderia ser implementada para permitir o controle da Estação através de um controle remoto infravermelho.

Com a compilação da biblioteca OpenCV [65], também é possível criar aplicações na área de Visão Computacional, como por exemplo, algoritmos de biometria para reconhecimento facial, reconhecimento de impressão digital, reconhecimento gestual, assim como algoritmos de

⁴⁸ Rede social cuja característica principal é o envio de mensagens com no máximo 140 caracteres. Página: www.twitter.com.

processamento de imagens para as mais diversas aplicações, como vigilância, inspeção visual automática, reconhecimento de padrões visuais, rastreamento de objetos e estimação de movimentos, dentre tantos outros possíveis.

7 Referências Bibliográficas

- [1] ABOUT.COM. [2012]. *TFTP*. Disponível em: <http://compnetworking.about.com/od/ftpfiletransfer/g/tftp-trivial-file-transfer-protocol.htm>>. Acesso em: 04 de Novembro de 2012.

- [2] ABOUT.COM. [2012]. *The History of The Transistor*. Disponível em: http://inventors.about.com/od/tstartinventions/a/transistor_history.htm>. Acesso em 30 de Outubro de 2012.

- [3] AMAZON. [2012]. *Kindle*. Disponível em: http://www.amazon.com/gp/product/B0051QVESA/ref=sv_kinh_0/180-1566936-3355319>. Acesso em: 04 de Novembro de 2012.

- [4] ANDROID. [2012]. *Android*. Disponível em: <http://www.android.com/>>. Acesso em: 04 de Novembro de 2012.

- [5] APACHE. [2012]. *Apache HTTP Server Project*. Disponível em: <http://httpd.apache.org/>>. Acesso em: 04 de Novembro de 2012.

- [6] APTANA. [2012]. *Aptana Studio 3*. Disponível em www.aptana.com>. Acesso em 30 de Outubro de 2012.

- [7] ARM. [2012]. *ARM – The Architecture for the Digital World*. Disponível em: <http://www.arm.com/>>. Acesso em: 04 de Novembro de 2012.

- [8] ARM9HOME. [2012]. *Arm9home*. Disponível em: <http://www.arm9home.net/>>. Acesso em 04 de Novembro de 2012.

- [9] BEAGLEBOARD.ORG. [2012]. *beagleboard.org*. Disponível em: <http://beagleboard.org>>. Acesso em 30 de Outubro de 2012.

- [10] BRASTEMP. [2012]. *Lavadora Brastemp Ative! Automática*. Disponível em: <http://www.brastemp.com.br/LinhaAtive>>. Acesso em 30 de Outubro de 2012.

- [11] BUILDROOT. [2012]. *Buildroot*. Disponível em: <<http://buildroot.uclibc.org/>>. Acesso em: 04 de Novembro de 2012.
- [12] CATB.ORG. [2012]. *NMEA Revealed*. Disponível em: <<http://catb.org/gpsd/NMEA.html>>. Acesso em: 04 de Novembro de 2012.
- [13] CNET. [2006] *A Computer is Born – ENIAC – monster and Marvel – Debuted 60 years ago*. Disponível em: <http://news.cnet.com/2009-1006_3-6037980.html>. Acesso em 04 de Novembro de 2012.
- [14] COLUMN, Tandesir. [2012]. *Tiny6410 based ds18b20 driver*. Disponível em: <<http://blog.csdn.net/tandesir/article/details/7247558>>. Acesso em 30 de Outubro de 2012.
- [15] CORBET, Jonathan, et. al. *LINUX DEVICE DRIVERS*, 3ed. O'Reilly Media. ISBN: 978-0-596-00590-0.
- [16] CORTEX-A, ARM. [2012]. *Cortex-A9 Processor*. Disponível em: <<http://www.arm.com/products/processors/cortex-a/cortex-a9.php>>. Acesso em: 30 de Outubro de 2012.
- [17] CSDN. [2012]. *Tiny6410 based ds18b20 driver*. Disponível em: <<http://blog.csdn.net/tandesir/article/details/7247558>>. Acesso em: 04 de Novembro de 2012.
- [18] CURL. [2012]. *cURL*. Disponível em: <<http://curl.haxx.se/>>. Acesso em 02 de Novembro de 2012.
- [19] DROPBOX. [2012]. *Dropbox for Developers*. Disponível em: <<https://www.dropbox.com/developers>>. Acesso em: 04 de Novembro de 2012.
- [20] ELECTRONS, Free. [2012]. *Free Electrons – Get the best of your hardware*. Disponível em: <<http://free-electrons.com/>>. Acesso em: 04 de Novembro de 2012.
- [21] EHOW. [2012]. *The History of Embedded Systems*. Disponível em: <http://www.ehow.com/info_12030725_history-embedded-systems.html>. Acesso em: 04 de Novembro de 2012.

- [22] FILEZILLA. [2012]. *FileZilla – The Free FTP Solution*. Disponível em: <<http://filezilla-project.org>>. Acesso em 30 de Outubro de 2012.
- [23] FRIENDLYARM [2012]. *FriendlyARM*. Disponível em: <<http://www.friendlyarm.net/>>. Acesso em 04 de Novembro de 2012.
- [24] FRIENDLYARM. [2012]. *FriendlyARM Forum*. Disponível em: <<http://www.friendlyarm.net/forum>>. Acesso em: 04 de Novembro de 2012.
- [25] FRIENDLYARM. [2012]. *Mini6410*. Disponível em: <<http://www.friendlyarm.net/products/mini6410>>. Acesso em 04 de Novembro de 2012.
- [26] FRIENDLYARM. [2012]. *The MPlayer multimídia-based application development guide*. Disponível em: <<http://www.arm9home.net/read.php?tid-14387.html>>. Acesso em: 04 de Novembro de 2012.
- [27] FRIENDLYARM. [2012]. *Tiny6410*. Disponível em: <<http://www.friendlyarm.net/products/tiny6410>>. Acesso em: 04 de Novembro de 2012.
- [28] FRIENDLYARM. [2012]. *Remotely Controle the Camera on Mini6410 trough a Web Browser*. Disponível em: <<http://arm9download.cncncn.com/tiny6410/um/ap04-mini6410-webcam.pdf>>. Acesso em: 04 de Novembro de 2012.
- [29] FRIENDLYARM. [2012]. *User's Guide to Mini6410 Linux*. Disponível em: <http://www.thaieasyelec.net/archives/Manual/User-Guide-to-Mini6410-Linux_041611.pdf>. Acesso em: 04 de Novembro de 2012.
- [30] FRIENDLYARM. [2012]. *User's Guide to Mini6410 System Installation*. Disponível em: <http://armdevs.googlecode.com/files/User's%20Guide%20to%20Mini6410%20System%20Installation_040211.pdf>. Acesso em: 04 de Novembro de 2012.
- [31] GARMIN. [2012]. *What is GPS*. Disponível em: <<http://www8.garmin.com/aboutGPS/>>. Acesso em: 04 de Novembro de 2012.
- [32] GLOBO.COM. [2012]. *Saiba como a 'internet das coisa' vai mudar seu cotidiano em breve*. Disponível em: <<http://g1.globo.com/tecnologia/noticia/2010/05/saiba-como-internet->

[das-coisas-vai-mudar-seu-cotidiano-em-breve.html](#)>. Acesso em: 04 de Novembro de 2012.

[33] GNU.ORG. [2012]. *GNU General Public License*. Disponível em: <http://www.gnu.org/copyleft/gpl.html>>. Acesso em: 04 de Novembro de 2012.

[34] GNU.ORG. [2012]. *What is Bash?* Disponível em: http://www.gnu.org/software/bash/manual/html_node/What-is-Bash_003f.html>. Acesso em: 04 de Novembro de 2012.

[35] GOOGLE. [2012]. *Google Drive SDK*. Disponível em: <https://developers.google.com/drive/>>. Acesso em: 04 de Novembro de 2012.

[36] GOOGLE. [2012]. *Google Translate*. Disponível em: <http://translate.google.com/>>. Acesso em: 04 de Novembro de 2012.

[37] GRAPHICS, Mentor. [2012]. *Sourcery CodeBench*. Disponível em: <http://www.mentor.com/embedded-software/codesourcery>>. Acesso em: 04 de Novembro de 2012.

[38] HARDWARE. [2012]. *ARM apresenta ARMv8, com instruções de 64-bit*. Disponível em: <http://www.hardware.com.br/noticias/2011-10/arm-64bit.html>>. Acesso em: 04 de Novembro de 2012.

[39] HARDWARE. [2012]. *EXT3*. Disponível em: <http://www.hardware.com.br/termos/ext3>>. Acesso em: 04 de Novembro de 2012.

[40] HIGHCHARTS. [2012]. *HighCharts JS – Interactive JavaScript charts for your web project*. Disponível em: <http://www.highcharts.com/>>. Acesso em 04 de Novembro de 2012.

[41] IBM. [2012]. *Network File System and Linux*. Disponível em: <http://www.ibm.com/developerworks/linux/library/l-network-filesystems/index.html>>. Acesso em: 04 de Novembro de 2012.

[42] INSTRUMENTS, Texas. [2012]. *MSP430 Ultra-Low Power 16-Bit Microcontrollers*. Disponível em: http://www.ti.com/llds/ti/microcontroller/16-bit_msp430/overview.page>.

Acesso em: 04 de Novembro de 2012.

[43] INTEGRATED, Maxim. [2012]. *DS18B20 Programmable Resolution 1-Wire Digital Thermometer*. Disponível em: <<http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>>. Acesso em 30 de Outubro de 2012.

[44] INTEL. [2012]. *Intel*. Disponível em: <<http://www.intel.com.br/>>. Acesso em: 7 de Dezembro de 2012.

[45] INTEL. [2012]. *Ultrabook – Inspired by Intel*. Disponível em: <<http://www.intel.com/content/www/us/en/sponsors-of-tomorrow/ultrabook.html>>. Acesso em: 04 de Novembro de 2012.

[46] JAVA. [2012]. *Java Technology Network*. Disponível em: <<http://www.oracle.com/technetwork/java/index.html>>. Acesso em: 04 de Novembro de 2012.

[47] JQUERY. [2012]. *jQuery: The Write Less, Do More, JavaScript Library*. Disponível em: <www.jquery.com>. Acesso em 30 de Outubro de 2012.

[48] LIGHTTPD. [2012]. *Lighttpd*. Disponível em: <<http://www.lighttpd.net/>>. Acesso em: 7 de Dezembro de 2012.

[49] LINUX, Viva o. [2012]. *Linux*. Disponível em: <<http://www.vivaolinux.com.br/linux/>>. Acesso em: 7 de Dezembro e 2012.

[50] LINUX, Viva o. [2012]. *O que é Shell Script*. Disponível em: <<http://www.vivaolinux.com.br/artigo/O-que-e-Shell-Script>>. Acesso em: 04 de Novembro de 2012.

[51] LINUX, Viva o. [2012]. *Utilizando o Vi – Uma introdução*. Disponível em: <<http://www.vivaolinux.com.br/artigo/Utilizando-o-Vi-uma-introducao>>. Acesso em: 04 de Novembro de 2012.

- [52] LINUX.DIE.NET. [2012]. *udev - Linux man page*. Disponível em: <<http://linux.die.net/man/7/udev>>. Acesso em: 04 de Novembro de 2012.
- [53] LOVE, Robert. LINUX KERNEL DEVELOPMENT, 3ed., 2010. Addison-Wesley Professional. ISBN: 978-0672329463.
- [54] MICROCHIP. [2012]. *8-bit PIC Microcontrollers*. Disponível em: <http://www.microchip.com/ja_jp/family/8bit/index.html>. Acesso em: 04 de Novembro de 2012.
- [55] MICROSOFT. [2012]. *Microsoft Windows*. Disponível em: <<http://windows.microsoft.com/pt-BR/windows/home>>. Acesso em: 04 de Novembro de 2012.
- [56] MIT. [2012]. *Massachusetts Institute of Technology*. Disponível em: <<http://www.mit.edu/>>. Acesso em: 04 de Novembro de 2012.
- [57] MUSEUM, MIT. [2012]. *Apollo Guidance Computer*. Disponível em: <<http://museum.mit.edu/nom150/entries/518>>. Acesso em: 04 de Novembro de 2012.
- [58] MONTAVISTA. [2012]. *MontaVista*. Disponível em: <<http://mvista.com/>>. Acesso em: 04 de Novembro de 2012.
- [59] MPG123. [2012]. *mpg123 – Fast console MPEG Audio Player and decoder library*. Disponível em: < <http://www.mpg123.de/>>. Acesso em: 04 de Novembro de 2012.
- [60] MPLAYER. [2012]. *Mplayer*. Disponível em: <<http://www.mplayerhq.hu/design7/news.html>>. Acesso em: 04 de Novembro de 2012.
- [61] MTD. [2012]. *UBIF – UBI File-System*. Disponível em: <<http://www.linux-mtd.infradead.org/doc/ubifs.html>>. Acesso em: 04 de Novembro de 2012.
- [62] NMEA. [2012]. *National Marine Electronics Association*. Disponível em: <<http://www.nmea.org/>>. Acesso em 30 de Outubro de 2012.
- [63] OLIMEX. [2012]. *SAM9-L9260*. Disponível em: <<https://www.olimex.com/Products/ARM/Atmel/SAM9-L9260/>>. Acesso em 02 de Novembro

de 2012.

[64] OPEN, The H. [2012]. *Dell donates concept ARM server to the ASF*. Disponível em: <<http://www.h-online.com/open/news/item/Dell-donates-concept-ARM-server-to-the-ASF-1736523.html>>. Acesso em 30 de Outubro de 2012.

[65] OPENCV. [2012]. *OpenCV – Open Source Computer Vision*. Disponível em: <<http://opencv.willowgarage.com>>. Acesso em 30 de Outubro de 2012.

[66] OPENEMBEDDED. [2012]. *OpenEmbedded*. Disponível em: <http://www.openembedded.org/wiki/Main_Page>. Acesso em: 04 de Novembro de 2012.

[67] PANDABOARD.ORG. [2012]. *pandaboard.org*. Disponível em: <<http://pandaboard.org/>>. Acesso em 30 de Outubro de 2012.

[68] PENGUTRONIX. [2012]. *Pengutronix*. Disponível em: <http://www.pengutronix.de/index_de.html>. Acesso em: 04 de Novembro de 2012.

[69] PHP. [2012]. *PHP*. Disponível em: <<http://www.php.net/>>. Acesso em: 04 de Novembro de 2012.

[70] PI, Rapsberry. [2012]. *Rapsberry Pi – Na ARM GNU/Linux box*. Disponível em: <<http://www.raspberrypi.org/>>. Acesso em 30 de Outubro de 2012.

[71] PUTTY. [2012]. *Putty*. Disponível em: <<http://www.chiark.greenend.org.uk/~sgtatham/putty/>>. Acesso em 04 de Novembro de 2012.

[72] PRADO, Sergio. [2012]. *Sergio Prado*. Disponível em: <<http://www.sergioprado.org>>. Acesso em: 04 de Novembro de 2012.

[73] PRIBERAM. [2012]. *Dicionário Priberam da Língua Portuguesa*. Disponível em: <<http://www.priberam.pt/dlpo/>>. Acesso em: 04 de Novembro de 2012.

[74] REUTERS. [2012]. *Android to beat Windows in 2016: Gartnet*. Disponível em: <<http://www.reuters.com/article/2012/10/24/us-android-research-idUSBRE89N11J20121024>>. Acesso em: 04 de Novembro de 2012.

[75] RUBY. [2012]. *Ruby Programming Language*. Disponível em: < <http://www.ruby-lang.org/en/>>. Acesso em: 04 de Novembro de 2012.

[76] SAMSUNG. [2012]. *Samsung Galaxy SII*. Disponível em: <<http://www.samsung.com/global/microsite/galaxys2/html/>>. Acesso em: 30 de Outubro de 2012.

[77] SENDMAIL. [2012]. *Welcome to the Sendmail Community*. Disponível em: <http://www.sendmail.com/sm/open_source/>. Acesso em 02 de Novembro de 2012.

[78] INTERESSANTE, Super. [2012]. *A Sociedade da Informação*. Disponível em: <<http://super.abril.com.br/tecnologia/sociedade-informacao-442036.shtml>>. Acesso em: 04 de Novembro de 2012.

[79] CSR. [2012]. *SiRF StarIII*. Disponível em: <<http://www.csr.com/products/26/sirfstariii-gsd3tw>>. Acesso em: 04 de Novembro de 2012.

[80] SECURITY, Search. [2012]. *What is Secure Shell*. Disponível em: <<http://searchsecurity.techtarget.com/definition/Secure-Shell>>. Acesso em: 04 de Novembro de 2012.

[81] SOURCEWARE. [2012]. *JFFS2: The Journalling Flash File System, version 2*. Disponível em: <<http://sourceware.org/jffs2/>>. Acesso em: 04 de Novembro de 2012.

[82] STANFORD. [2012]. *Web Applications*. Disponível em: <<http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=WebApplications>>. Acesso em: 04 de Novembro de 2012.

[83] STREAMINGMEDIA.COM. [2012]. *What is streaming?* Disponível em: <<http://www.streamingmedia.com/Articles/ReadArticle.aspx?ArticleID=74052>>. Acesso em 30 de Outubro de 2012.

[84] STREAMER, Mjpg. [2012]. . Disponível em: <http://sourceforge.net/apps/mediawiki/mjpg-streamer/index.php?title=Main_Page>. Acesso em 30 de outubro de 2012.

- [85] TECHNOLOGIES, MIPS. [2012]. *MIPS Technologies*. Disponível em: <<http://www.mips.com/>>. Acesso em: 04 de Novembro de 2012.
- [86] TELNET. [2012]. *Telnet FAQ*. Disponível em: <<http://www.telnet.org/htm/faq.htm>>. Acesso em: 04 de Novembro de 2012.
- [87] THHTTPD. [2012]. *Thttpd – Tiny/turbo/throttling HTTP SERVER*. Disponível em: <<http://acme.com/software/thttpd/>>. Acesso em: 04 de Novembro de 2012.
- [88] U-BOOT. [2012]. *Das U-Boot – the Universal Boot Loader*. Disponível em: <<http://www.denx.de/wiki/U-Boot/WebHome>>. Acesso em: 04 de Novembro de 2012.
- [89] W3C. [2012]. *Cascading Style Sheets*. Disponível em: <<http://www.w3.org/Style/CSS/>>. Acesso em: 04 de Novembro de 2012.
- [90] W3C. [2012]. *HTML*. Disponível em: <<http://www.w3.org/html/>>. Acesso em: 04 de Novembro de 2012.
- [91] W3C. [2012]. *Hypertext Transfer Protocol – HTTP/1.1*. Disponível em: <<http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>>. Acesso em: 04 de Novembro de 2012.
- [92] W3C. [2012]. *JAVASCRIPT WEB APIS*. Disponível em: <<http://www.w3.org/standards/webdesign/script.html>>. Acesso em: 04 de Novembro de 2012.
- [93] WEBOPEDIA. [2012]. *What is CGI?*. Disponível em: <<http://www.webopedia.com/TERM/C/CGI.html>>. Acesso em: 04 de Novembro de 2012.
- [94] WEBOPEDIA. [2012]. *FTP*. Disponível em: <<http://www.webopedia.com/TERM/F/FTP.html>>. Acesso em: 04 de Novembro de 2012.
- [95] WEBSERVER, Boa. [2012]. *Boa Webserver*. Disponível em: <<http://www.boa.org/>>. Acesso em: 04 de Novembro de 2012.
- [96] WILDANM. [2012]. *mdev – mini udev in busybox*. Disponível em: <<http://wildanm.wordpress.com/2007/08/21/mdev-mini-udev-in-busybox/>>. Acesso em: 04 de

Novembro de 2012.

[97] XMPPPY. [2012]. *xmpppy: The jabber python project*. Disponível em: <<http://xmpppy.sourceforge.net/>>. Acesso em: 04 de Novembro de 2012.

[98] YAFFS. [2012]. *Yaffs2 Specification*. Disponível em: <<http://www.yaffs.net/yaffs-2-specification>>.. Acesso em: 04 de Novembro de 2012.

[99] YAGHMOUR, Karim, et. al. BUILDING LINUX EMBEDDED SYSTEMS. 2003. O'Reilly Media. ISBN: 978-0-596-00222-0.

8 Anexos

8.1 FriendlyARM.ini

É o arquivo de configuração de *boot* do cartão de memória. Esse arquivo define qual sistema deve ser carregado, seja para inicialização ou para instalação. Ele também permite definir os arquivos associados a cada sistema.

```
#This line cannot be removed. by FriendlyARM(www.arm9.net)

LCD-Mode = Yes
LCD-Type = N43

CheckOneButton=No
Action=Run
OS= Linux

VerifyNandWrite=No

StatusType = Beeper | LED

##### Linux #####
Linux-BootLoader = Linux/superboot-20110722.bin
Linux-Kernel = Linux/zImage_n43
Linux-CommandLine = root=/dev/mtdblock2 rootfstype=yaffs2 init=/linuxrc
console=ttySAC0,115200
Linux-RootFs-InstallImage = Linux/rootfs_qtopia_qt4.img
Linux-RootFs-RunImage = Linux/rootfs_qtopia_qt4.ext3
```

8.2 Start_uvc.sh

É o *script Shell Script* utilizado para subir o serviço de *streaming* de vídeo do MJPG-Streamer, mapeando o dispositivo de captura de vídeo, que é a câmera USB, e jogando a saída no diretório **/www**, mapeado pelo *lighttpd*.

```
#!/bin/sh
```

```

# /*****
*****
#
#
#     MJPG-streamer allows to stream JPG frames from an input-plugin
#
#     to several output plugins
#
#
#
#     Copyright (C) 2007 Tom Stöveken
#
#
#
# This program is free software; you can redistribute it and/or modify
#
# it under the terms of the GNU General Public License as published by
#
# the Free Software Foundation; version 2 of the License.
#
#
#
# This program is distributed in the hope that it will be useful,
#
# but WITHOUT ANY WARRANTY; without even the implied warranty of
#
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#
# GNU General Public License for more details.
#
#
#
# You should have received a copy of the GNU General Public License
#
# along with this program; if not, write to the Free Software
#
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
USA    #
#
#
# *****/

```



```

*****/

## This example shows how to invoke mjpg-streamer from the command
line

export LD_LIBRARY_PATH="$(pwd) "
#./mjpg_streamer -o "output_http.so -w ./www" -i "input_uvc.so -d
/dev/camera"

/mjpg-streamer/mjpg_streamer -o "/mjpg-streamer/output_http.so -p 8082
-w /www" -i "/mjpg-streamer/input_uvc.so -d /dev/video2 -r 352x288"

## pwd echos the current path you are working at,
## the backticks open a subshell to execute the command pwd first
## the exported variable name configures ldopen() to search a certain
## folder for *.so modules
#export LD_LIBRARY_PATH=`pwd`

## this is the minimum command line to start mjpg-streamer with
webpages
## for the input-plugin default parameters are used
#./mjpg_streamer -o "output_http.so -w `pwd`/www"

## to query help for the core:
# ./mjpg_streamer --help

## to query help for the input-plugin "input_uvc.so":
# ./mjpg_streamer --input "input_uvc.so --help"

## to query help for the output-plugin "output_file.so":
# ./mjpg_streamer --output "output_file.so --help"

## to query help for the output-plugin "output_http.so":
# ./mjpg_streamer --output "output_http.so --help"

## to specify a certain device, framerate and resolution for the input
plugin:
# ./mjpg_streamer -i "input_uvc.so -d /dev/video2 -r 320x240 -f 10"

## to start both, the http-output-plugin and write to files every 15
second:

```

```
# mkdir pics
# ./mjpg_streamer -o "output_http.so -w `pwd`/www" -o "output_file.so
-f pics -d 15000"

## to protect the webserver with a username and password (!! can
easily get sniffed and decoded, it is just base64 encoded !!)
# ./mjpg-streamer -o "output_http.so -w ./www -c UsErNaMe:SeCrEt"

## If you want to track down errors, use this simple testpicture
plugin as input source.
## to use the testpicture input plugin instead of a webcam or folder:
# ./mjpg_streamer -i "./input_testpicture.so -r 320x240 -d 500" -o
"./output_http.so -w www"
```

8.3 Lighttpd.conf

É o arquivo de configuração utilizado na execução do servidor *web lighttpd*. Define o diretório a ser mapeado, a porta que será utilizada para escutar requisições HTTP, módulos adicionais, arquivos *index* mapeados, dentre outros elementos.

```
server.document-root = "/www"

server.port = 80
#server.username = "lighttpd"
server.groupname = "lighttpd"
#server.bind = "143.107.235.39"
server.tag = "ARM Web Server"

#server.errorlog      = "/var/log/lighttpd/error.log"
#accesslog.filename   = "/var/log/lighttpd/access.log"

server.modules        = (
"mod_access",
"mod_accesslog",
"mod_fastcgi",
"mod_rewrite",
"mod_auth"
```

```

)

# mimetype mapping
mimetype.assign      = (
".pdf"      =>  "application/pdf",
".sig"      =>  "application/pgp-signature",
".spl"      =>  "application/futuresplash",
".class"    =>  "application/octet-stream",
".ps"       =>  "application/postscript",
".torrent"  =>  "application/x-bittorrent",
".dvi"      =>  "application/x-dvi",
".gz"       =>  "application/x-gzip",
".pac"      =>  "application/x-ns-proxy-autoconfig",
".swf"      =>  "application/x-shockwave-flash",
".tar.gz"   =>  "application/x-tgz",
".tgz"      =>  "application/x-tgz",
".tar"      =>  "application/x-tar",
".zip"      =>  "application/zip",
".mp3"      =>  "audio/mpeg",
".m3u"      =>  "audio/x-mpegurl",
".wma"      =>  "audio/x-ms-wma",
".wax"      =>  "audio/x-ms-wax",
".ogg"      =>  "audio/x-wav",
".wav"      =>  "audio/x-wav",
".gif"      =>  "image/gif",
".jpg"      =>  "image/jpeg",
".jpeg"     =>  "image/jpeg",
".png"      =>  "image/png",
".xbm"      =>  "image/x-xbitmap",
".xpm"      =>  "image/x-xpixmap",
".xwd"      =>  "image/x-xwindowdump",
".css"      =>  "text/css",
".html"     =>  "text/html",
".htm"      =>  "text/html",

```

```

".js"      =>  "text/javascript",
".asc"      =>  "text/plain",
".c"        =>  "text/plain",
".conf"     =>  "text/plain",
".text"     =>  "text/plain",
".txt"      =>  "text/plain",
".dtd"      =>  "text/xml",
".xml"      =>  "text/xml",
".mpeg"     =>  "video/mpeg",
".mpg"      =>  "video/mpeg",
".mov"      =>  "video/quicktime",
".qt"       =>  "video/quicktime",
".avi"      =>  "video/x-msvideo",
".asf"      =>  "video/x-ms-asf",
".asx"      =>  "video/x-ms-asf",
".wmv"      =>  "video/x-ms-wmv",
".bz2"      =>  "application/x-bzip",
".tbz"      =>  "application/x-bzip-compressed-tar",
".tar.bz2"  =>  "application/x-bzip-compressed-tar"
)
index-file.names = ( "index.html", "index.php", "index.htm", "index.rb")

fastcgi.server = ( ".php" => ((
"bin-path" => "/usr/bin/php-cgi",
"socket" => "/tmp/php.socket"
)))

```

8.4 rcS – Script de Inicialização do Sistema

É o arquivo de inicialização geral do sistema Linux da Tiny6410. É responsável por instanciar variáveis de ambiente, carregar *drivers* e demais configurações do sistema. Qualquer comando que deva ser executado assim que o sistema for inicializado deve ser inserido neste arquivo.

```
#!/bin/sh
```

```
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin:
runlevel=S
prevlevel=N
umask 022
export PATH runlevel prevlevel

#
#      Trap CTRL-C &c only in this shell so we can interrupt subprocesses.
#
trap ":" INT QUIT TSTP
/bin/hostname FriendlyARM

[ -e /proc/1 ] || /bin/mount -n -t proc none /proc
[ -e /sys/class ] || /bin/mount -n -t sysfs none /sys
[ -e /dev/tty ] || /bin/mount -t ramfs none /dev
/bin/mount -n -t usbfs none /proc/bus/usb

echo /sbin/mdev > /proc/sys/kernel/hotplug
/sbin/mdev -s
/bin/hotplug
# mounting file system specified in /etc/fstab
mkdir -p /dev/pts
mkdir -p /dev/shm
/bin/mount -n -t devpts none /dev/pts -o mode=0622
/bin/mount -n -t tmpfs tmpfs /dev/shm
/bin/mount -n -t ramfs none /tmp
/bin/mount -n -t ramfs none /var
mkdir -p /var/empty
mkdir -p /var/log
mkdir -p /var/lock
mkdir -p /var/run
mkdir -p /var/tmp
```

```

/sbin/hwclock -s

syslogd

MODULE_PATH=/lib/modules/`uname -r`
if [ ! -f ${MODULE_PATH}/modules.dep.bb ]; then
    depmod
fi

modprobe fa_cpu_pfn 2>/dev/null

/etc/rc.d/init.d/netd start
echo "          " > /dev/tty1
echo "Starting networking..." > /dev/tty1
sleep 1
/usr/sbin/lighttpd -f /etc/lighttpd.conf
echo "          " > /dev/tty1
echo "Starting web server..." > /dev/tty1
sleep 1
/etc/init.d/sshd start
echo "          " > /dev/tty1
echo "Starting sshd server..." > /dev/tty1
echo "          "
sleep 1
/usr/sbin/crond
echo "Starting Cron Daemon..." > /dev/tty1
echo "          "
sleep1

echo "          " > /dev/tty1
/etc/rc.d/init.d/alsaconf start
echo "Loading sound card config..." > /dev/tty1
echo "          "

```

```

echo "          " > /dev/tty1
/sbin/insmod /etc/ds18b20.ko
echo "Starting temperature sensor... " > /dev/tty1
echo "          "

/sbin/ifconfig lo 127.0.0.1
/etc/init.d/ifconfig-eth0
fa-network-service

#/bin/qtopia &
#echo "          " > /dev/tty1
#echo "Starting Qtopia, please waiting..." > /dev/tty1

echo "          " > /dev/tty1
/usbCapture/initCap.sh
echo "Starting time photo capture... " > /dev/tty1
echo "          " > /dev/tty

clear > /dev/tty1

```

8.5 [www/css/style.css](http://www.css/style.css)

Esse é o arquivo de estilos CSS, baseado no modelo fornecido pelo projeto MJPG Streamer. Foi originalmente desenvolvido por Andreas Viklund, e determina as cores, fontes, alinhamento de textos e demais componentes HTML da página *web* do projeto.

```

/* andreas07 - an open source xhtml/css website layout by Andreas
Viklund - http://andreasviklund.com . Free to use for any purpose as
long as the proper credits are given for the original design work.
Version: 1.1, November 28, 2005 */
/***** Page and tag styles *****/
body {

    width:100%;

```

```
height:100%;
margin:0;
padding:0;
color:#303030;
background:#fafafa url(bodybg.gif) top left repeat-y;
font:76% Verdana,Tahoma,sans-serif;
}

ul {
    list-style:circle;
    margin:15px 0 20px 0;
    font-size:0.9em;
}

li {
    margin:0 0 8px 25px;
}

a {
    color:#d85d5d;
    font-weight:bold;
    text-decoration:none;
}

a:hover {
    color:#505050;
    text-decoration:underline;
}

img {
    float:left;
    margin:0 15px 15px 0;
    padding:1px;
    background:#ffffff;
    border:1px solid #d0d0d0;
}

a img {
    border-color:#d85d5d;
}
```



```

a img:hover {
    background:#d85d5d;
    border-color:#d85d5d;
}

p {
    text-align:justify;
}

/***** Sidebar area styles *****/
#sidebar {

    float: left;
    width:15%;
    overflow:auto;
    height:100%;
    background:#e0e0e0 url(sidebarbg.gif) top right repeat-y;
    text-align:right;
}

/*body > #sidebar
{position:fixed;}*/

#sidebar h1 {
    margin:20px 18px 0 5px;
    color:#d85d5d;
    font-size:1.6em;
    letter-spacing:-2px;
    text-align:right;
}

#sidebar h2, #sidebar h3
{margin:0 20px 18px 5px; color:#808080; font-size:1.1em; font-
weight:bold; letter-spacing:-1px; text-align:right;}

#sidebar h3

```

```

{margin:20px 18px 4px 5px; color:#606060;}

#sidebar p
{margin:0 20px 18px 5px; color:#606060; font-size:0.8em;}

#sidebar a
{color:#808080}

/***** Navigation menu styles *****/
#menu a
{display:block; width:auto; padding:5px 18px 5px 0; color:#606060;
background:#e0e0e0 url(sidebarbg.gif) top right repeat-y; font-
size:1.8em; font-weight:normal; text-decoration:none; letter-spacing:-
2px;}

#menu a:hover
{color:#303030; background:#f0f0f0 url(sidebarbg.gif) top right
repeat-y;}

#menu a.active
{padding:5px 18px 5px 0; background:#fafafa; border-top:2px solid
#c0c0c0; border-bottom:2px solid #c0c0c0;}

#menu a.active:hover
{color:#505050; background:#fafafa;}

/***** Content area styles *****/

#header {
    clear: both;
    height: 50px;
    padding: 1px;
}

#content {
    float: left;
    margin:auto;
    padding:10px;

```

```
width:60%;
height:100%;
background:#fafafa;
}

#content p
{margin:0 0 20px 0; line-height:1.5em;}

#content h1
{margin:0; color:#d85d5d; font-size:4em; letter-spacing:-5px; text-align:center;}

#content h2
{margin:0; color:#808080; font-weight:normal; font-size:2.5em; letter-spacing:-2px; text-align:center;}

#content h3
{clear:both; margin:30px 0 10px 0; color:#d85d5d; font-weight:normal; font-size: 2em; letter-spacing:-2px;}

#control {
    float: right;
    top:0px;
    margin: auto;
    padding: 10px;
    height: 100%;
    width: 15%;
}

#footer {
    clear: both;
    position: fixed;
    left: 0px;
    bottom: 0px;
    height: 30px;
    width: 100%;
    background-color: #e0e0e0;
}

h4 {
```

```
text-align: left;
}
```

8.6 Ds18b20_test.c original

É a aplicação original de teste do *driver* do sensor de temperatura. Como pode ser visto no código, ela gera um *loop* infinito mostrando a temperatura ambiente em graus Celsius.

```

/*****Copyright (c) *****/
*****

** 文件名称: ds18b20_test.c
** 作    者: lixin
** 版    本: v1.0
** 说    明: ds18b20测试程序,精度:0.25度, 如有需要, 还可以进一步提高。
** 修改记录: 2009-8-27创建
** 最后修改时间: 2009-09-01

*****
*****/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <linux/ioctl.h>

// 函数声明
void ds18b20_delay(int i);

int main()
{
    int fd, i;
    unsigned char result[2];          // 从ds18b20读出的结果,
result[0]存放低八位
    unsigned char integer_value = 0;
    float temperature, decimal_value; // 温度数值,decimal_value为小
数部分的值
    fd = open("/dev/ds18b20", 0);
    if (fd < 0)
    {

```

```

        perror("open device failed\n");
        exit(1);
    }
    while (1)
    {
        i++;
        read(fd, &result, sizeof(result));
        integer_value = ((result[0] & 0xf0) >> 4) | ((result[1] &
0x07) << 4);

        // 精确到0.25度
        decimal_value = 0.5 * ((result[0] & 0x0f) >> 3) + 0.25 *
((result[0] & 0x07) >> 2);
        // decimal_value = 0.5 * ((result[0] & 0x0f) >> 3) + 0.25 *
((result[0] & 0x07) >> 2) + 0.125 * ((result[0] & 0x03) >> 1) + 0.0625 *
((result[0] & 0x01) >> 0);
        temperature = (float)integer_value + decimal_value;
        printf("The temperature is %6.2f\n", temperature);
        //printf("The temperature is %6.4f\n", temperature);
        if (i % 20 == 0)
            printf("\n");
        ds18b20_delay(50);
    }
}

/*****
*****
** 函数名称: ds18b20_delay()
** 函数功能: 延时
** 入口参数: i
** 出口参数: 无
** 备    注:

*****
*****/

void ds18b20_delay(int i)
{
    int j, k;
    for (j = 0; j < i; j++)
        for(k = 0; k < 50000; k++);
}

```

```

}

/*****
*****

**                                     文件到此结束

*****
*****/

```

8.7 s3c6410_gpio.h

É o cabeçalho responsável por definir os *ports* de I/O da Tiny6410. Esse arquivo é utilizado na compilação do *driver* do sensor de temperatura.

```

/*****Copyright (c) *****/
*****

** 文件名称: s3c6410_gpio.h
** 作    者: tandesir
** 版    本: v1.0
** 说    明: s3c6410 gpio操作
** 修改记录: 2011-9-27创建
** 最后修改时间: 2011-10-5

*****
*****/

#ifndef __S3C6410_GPIO_H__
#define __S3C6410_GPIO_H__

#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/interrupt.h>
#include <linux/ioport.h>
#include <linux/gpio.h>
#include <linux/io.h>

#include <mach/hardware.h>

```

```

#include <mach/map.h>
#include <mach/regs-gpio.h>
#include <mach/gpio-bank-n.h>
#include <mach/regs-clock.h>
#include <asm/irq.h>

#include <plat/gpio-core.h>
#include <plat/gpio-cfg.h>

void s3c6410_gpio_cfgpin(unsigned int pin, unsigned int function)
{
    //s3c_gpio_cfgpin(pin,function);
    unsigned int tmp;
    tmp = readl(S3C64XX_GPNCN);
    tmp = (tmp & ~(3<<pin*2)) | (function<<pin*2);
    writel(tmp, S3C64XX_GPNCN);
}

void s3c6410_gpio_pullup(unsigned int pin, unsigned int to)
{
    //s3c_gpio_setpull(pin,to);
    unsigned int tmp;
    tmp = readl(S3C64XX_GPNPUD);
    tmp = (tmp & ~(3<<pin*2)) | (to<<pin*2);
    writel(tmp, S3C64XX_GPNPUD);
}

unsigned int s3c6410_gpio_getpin(unsigned int pin)
{
    unsigned int tmp;
    tmp = readl(S3C64XX_GPNDAT);
    tmp = tmp & (1 << (pin));

    return tmp;
}

void s3c6410_gpio_setpin(unsigned int pin, unsigned int dat)
{
    unsigned int tmp;
    tmp = readl(S3C64XX_GPNDAT);
    tmp &= ~(1 << (pin));

```

```

        tmp |= ( (dat) << (pin) );
        writel(tmp, S3C64XX_GPNDAT); ;
    }

#endif

```

8.8 Ds18b20.c

É o arquivo de *driver* do sensor de temperatura, que após ser compilado irá gerar um arquivo chamado **ds18b20.ko**, que é o módulo de *Kernel* que deve ser carregado no sistema Linux para instanciar o *driver* do sensor de temperatura.

```

/*****Copyright (c) *****/
*****
**  文件名称: ds18b20_drv.c
**  作    者: lixin
**  版    本: v1.0
**  说    明: ds18b20驱动程序.工作过程及时序见ds18b20 datasheet
**  修改记录: 2009-8-27创建
**  最后修改时间: 2009-09-01
*****
*****/

#include <linux/init.h>
#include <linux/module.h>
#include <linux/delay.h>
#include <linux/kernel.h>
#include <linux/moduleparam.h>
#include <linux/init.h>
#include <linux/types.h>
#include <linux/fs.h>
#include <mach/regs-gpio.h>
#include <linux/device.h>
#include <mach/hardware.h>
#include <linux/cdev.h>
#include <asm/uaccess.h>
#include <linux/errno.h>

#include "s3c6410_gpio.h"

```



```

// #define DEBUG
/* 相关引脚定义,方便以后移植 */
#define DEVICE_NAME "ds18b20"
#define DQ          8
#define CFG_IN      0
#define CFG_OUT     1

// ds18b20主设备号(动态分配)
int ds18b20_major = 0;
int ds18b20_minor = 0;
int ds18b20_nr_devs = 1;

// 定义设备类型
static struct ds18b20_device {
    struct cdev cdev;
};
struct ds18b20_device ds18b20_dev;

static struct class *ds18b20_class;

/* 函数声明 */
static int ds18b20_open(struct inode *inode, struct file *filp);
static int ds18b20_init(void);
static void write_byte(unsigned char data);
static unsigned char read_byte(void);
static ssize_t ds18b20_read(struct file *filp, char __user *buf,
                           size_t count, loff_t *f_pos);
void ds18b20_setup_cdev(struct ds18b20_device *dev, int index);

/*****
*****

** 函数名称: ds18b20_open()
** 函数功能: 打开设备,初始化ds18b20
** 入口参数: inode:设备文件信息; filp: 被打开的文件的信息
** 出口参数: 成功时返回0,失败返回-1
** 备    注:

*****
*****/

```

```

static int ds18b20_open(struct inode *inode, struct file *filp)
{
    int flag = 0;
    /*struct ds18b20_device *dev;
    dev = container_of(inode->i_cdev, struct ds18b20_device, cdev);
    filp->private_data = dev;*/

    flag = ds18b20_init();
    if(flag & 0x01)
    {
#ifdef DEBUG
        printk(KERN_WARNING "open ds18b20 failed\n");
#endif
        return -1;
    }
#ifdef DEBUG
        printk(KERN_NOTICE "open ds18b20 successful\n");
#endif
    return 0;
}

/*****
*****

** 函数名称: ds18b20_init()
** 函数功能: 复位ds18b20
** 入口参数: 无
** 出口参数: retval:成功返回0,失败返回1
** 备    注: 操作时序见ds18b20 datasheet
*****
*****/
static int ds18b20_init(void)
{
    int retval = 0;

    s3c6410_gpio_cfgpin(DQ, CFG_OUT);
    s3c6410_gpio_pullup(DQ, 0);

    s3c6410_gpio_setpin(DQ, 1);
    udelay(2);
    s3c6410_gpio_setpin(DQ, 0);           // 拉低ds18b20总线，复位ds18b20

```

```

    udelay(500); // 保持复位电平500us

    s3c6410_gpio_setpin(DQ, 1); // 释放ds18b20总线
    udelay(60);

    // 若复位成功, ds18b20发出存在脉冲(低电平, 持续60~240us)
    s3c6410_gpio_cfgpin(DQ, CFG_IN);
    retval = s3c6410_gpio_getpin(DQ);

    udelay(500);
    s3c6410_gpio_cfgpin(DQ, CFG_OUT);
    s3c6410_gpio_pullup(DQ, 0);
    s3c6410_gpio_setpin(DQ, 1); // 释放总线

    return retval;
}

/*****
*****
** 函数名称: write_byte()
** 函数功能: 向18b20写入一个字节数据
** 入口参数: data
** 出口参数: 无
** 备    注:
*****
*****/
static void write_byte(unsigned char data)
{
    int i = 0;

    s3c6410_gpio_cfgpin(DQ, CFG_OUT);
    s3c6410_gpio_pullup(DQ, 1);

    for (i = 0; i < 8; i++)
    {
        // 总线从高拉至低电平时, 就产生写时隙
        s3c6410_gpio_setpin(DQ, 1);
        udelay(2);
        s3c6410_gpio_setpin(DQ, 0);
    }
}

```

```

        s3c6410_gpio_setpin(DQ, data & 0x01);
        udelay(60);
        data >>= 1;
    }

    s3c6410_gpio_setpin(DQ, 1);          // 重新释放ds18b20总线
}

/*****
*****

** 函数名称: read_byte()
** 函数功能: 从ds18b20读出一个字节数据
** 入口参数: 无
** 出口参数: 读出的数据
** 备    注:
*****
*****/
static unsigned char read_byte(void)
{
    int i;
    unsigned char data = 0;

    for (i = 0; i < 8; i++)
    {
        // 总线从高拉至低，只需维持低电平17ts，再把总线拉高，就产生读时隙
        s3c6410_gpio_cfgpin(DQ, CFG_OUT);
        s3c6410_gpio_pullup(DQ, 0);
        s3c6410_gpio_setpin(DQ, 1);
        udelay(2);
        s3c6410_gpio_setpin(DQ, 0);
        udelay(2);
        s3c6410_gpio_setpin(DQ, 1);
        udelay(8);
        data >>= 1;
        s3c6410_gpio_cfgpin(DQ, CFG_IN);
        if (s3c6410_gpio_getpin(DQ))
            data |= 0x80;
        udelay(50);
    }
    s3c6410_gpio_cfgpin(DQ, CFG_OUT);

```

```

s3c6410_gpio_pullup(DQ, 0);
s3c6410_gpio_setpin(DQ, 1);          // 释放ds18b20总线
return data;
}

/*****
*****

** 函数名称: ds18b20_read()
** 函数功能: 读出18b20的温度
** 入口参数:
** 出口参数:
** 备    注:

*****/
static ssize_t ds18b20_read(struct file *filp, char __user *buf,
                           size_t count, loff_t *f_pos)
{
    int flag;
    unsigned long err;
    unsigned char result[2] = {0x00, 0x00};
    //struct ds18b20_device *dev = filp->private_data;

    flag = ds18b20_init();
    if (flag)
    {
#ifdef DEBUG
        printk(KERN_WARNING "ds18b20 init failed\n");
#endif
        return -1;
    }

    write_byte(0xcc);
    write_byte(0x44);

    flag = ds18b20_init();
    if (flag)
        return -1;

    write_byte(0xcc);
    write_byte(0xbe);

```

```

    result[0] = read_byte();    // 温度低八位
    result[1] = read_byte();    // 温度高八位

    err = copy_to_user(buf, &result, sizeof(result));
    return err ? -EFAULT : min(sizeof(result), count);
}

/*****
 * 字符驱动程序的核心，应用程序所调用的open, read等函数最终会
 * 调用这个结构中的对应函数
 *****/
static struct file_operations ds18b20_dev_fops = {
    .owner = THIS_MODULE,
    .open = ds18b20_open,
    .read = ds18b20_read,
};

/*****
 *****/
** 函数名称: ds18b20_setup_cdev()
** 函数功能: 初始化cdev
** 入口参数: dev:设备结构体; index:
** 出口参数: 无
** 备    注:
 *****/
*****/
void ds18b20_setup_cdev(struct ds18b20_device *dev, int index)
{
    int err, devno = MKDEV(ds18b20_major, ds18b20_minor + index);

    cdev_init(&dev->cdev, &ds18b20_dev_fops);
    dev->cdev.owner = THIS_MODULE;
    err = cdev_add(&(dev->cdev), devno, 1);
    if (err)
    {
#ifdef DEBUG
        printk(KERN_NOTICE "ERROR %d add ds18b20\n", err);
#endif
    }
}

```

```

}

/*****
*****
** 函数名称: ds18b20_dev_init()
** 函数功能: 为温度传感器分配注册设备号，初始化cdev
** 入口参数: 无
** 出口参数: 若成功执行，返回0
** 备    注:
*****
*****/
static int __init ds18b20_dev_init(void)
{
    ds18b20_major = register_chrdev(ds18b20_major, DEVICE_NAME,
&ds18b20_dev_fops);
    if (ds18b20_major<0)
    {
        printk(DEVICE_NAME " Can't register major number!\n");
        return -EIO;
    }

    ds18b20_class = class_create(THIS_MODULE, DEVICE_NAME);
    device_create(ds18b20_class, NULL, MKDEV(ds18b20_major,
ds18b20_minor), NULL, DEVICE_NAME);
#ifdef DEBUG
    printk(KERN_WARNING "register ds18b20 driver successful!\n");
#endif
    return 0;
}

/*****
*****
** 函数名称: ds18b20_dev_exit()
** 函数功能: 注销设备
** 入口参数: 无
** 出口参数: 无
** 备    注:
*****
*****/

```

```

static void __exit ds18b20_dev_exit(void)
{
    device_destroy(ds18b20_class, MKDEV(ds18b20_major, ds18b20_minor));
    class_unregister(ds18b20_class);
    class_destroy(ds18b20_class);
    unregister_chrdev(ds18b20_major, DEVICE_NAME);
#ifdef DEBUG
    printk(KERN_WARNING "Exit ds18b20 driver!\n");
#endif
}

module_init(ds18b20_dev_init);
module_exit(ds18b20_dev_exit);
MODULE_LICENSE("Dual BSD/GPL");
MODULE_AUTHOR("xinli_whut@163.com");

/*****
*****
**
*****
*****/

```

文件到此结束

8.9 Makefile do Driver DS18B20

É o *Makefile* utilizado para compilar o *driver* do sensor de temperatura.

```

# Comment/uncomment the following line to disable/enable debugging
DEBUG = n

# Add your debugging flag (or not) to CFLAGS
# "-O" is needed to expand inlines
ifeq ($(DEBUG),y)
    DEBFLAGS = -O -g -DDEBUG
else
    DEBFLAGS = -O2
endif

EXTRA_CFLAGS += $(DEBFLAGS)
#EXTRA_CFLAGS += -I$(LDDINC)

```



```

ifneq ($(KERNELRELEASE),)
# call from kernel build system

#scull-objs := main.o pipe.o access.o

obj-m      := ds18b20.o

else

KERNELDIR :=/home/andremarcio/Lab/Kernels/linux-2.6.38
PWD        := $(shell pwd)

modules:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules

clean:
    rm -rf *.o *~ core .depend *.cmd *.ko *.mod.c .tmp_versions

depend .depend dep:
    $(CC) $(CFLAGS) -M *.c > .depend

    ifeq (.depend,$(wildcard .depend))
    include .depend
    endif

endif

```


9 Apêndices

9.1 Scripts de Controle PHP

Nessa parte estão todos os *scripts* PHP utilizados para controle de sistema e de aplicações no projeto. Não é a principal função desses *scripts* sintetizar páginas HTML, mas receber requisições e trabalhar em cima dessas requisições.

9.1.1 `www/php/auth.php`

É o *script* PHP que responde a requisições POST, verificando se os dados recebidos de *login* de usuário correspondem a dados válidos cadastrados no banco de dados **login.sqlite**. Caso sejam dados válidos, a sessão de acesso do usuário é considerada como acesso registrado. Caso não sejam dados válidos, a sessão não é validada.

```
<?php

//Carregando o banco de dados...
$db = new PDO('sqlite:/www/db/login.sqlite');
$users = $db->query('SELECT * FROM users');

session_start();

if(isset($_POST['submit'])){
    if(isset($_POST['user']) && isset($_POST['passwd'])){
        $user = $_POST['user'];
        $passwd = $_POST['passwd'];
        foreach ($users as $row) {
            if($user == $row['name'] && $passwd == $row['password']){
                $_SESSION['login'] = true;
                $_SESSION['name'] = $row['name'];

                echo '<META HTTP-EQUIV="Refresh" Content="0;
URL=http://armweb.no-ip.org">';

            } else {
                $_SESSION['login'] = false;
                echo "Falha no login! Tente novamente!";
            }
        }
    }
}
```

```

    }
}

if(isset($_GET['logout']) && ($_SESSION['login'] = true)){
    session_destroy();
    echo '<META HTTP-EQUIV="Refresh" Content="0; URL=http://armweb.no-
ip.org">';
}
?>

```

9.1.2 www/php/camCapture.php

É o *script* de controle de captura de fotos, exibição da ultima foto tirada e remoção da mesma.

```

<?php
include '/www/php/variables.php';

$file = "/www/files/pic.hist";

//Link do script de captura
$camShell = "/usbCapture/cameraCapture.sh";

//Pega data atual - Do servidor!
$data = date('d-m-Y--h-i');

//Tipo padrao.
$type = "1";

$camShellCMD = "$camShell $type $data";

//---- Set of variables

/*

```

```

* Testes mostraram que o script não consegue "sair"
* do controle do mjpg_streamer.
* Dessa forma, fica a critério do PHP o controle sobre
* o processo do MJPG Streamer para a tirada de fotos.
*/

//Escreve em arquivo txt ultima imagem gravada
function writeLast($img,$file){
//base de escrita de arquivo!
$fr = fopen($file,'w+');
fwrite($fr,$img);
fclose($fr);
}

//Pega ultima imagem gravada.
function getLast($file){
    $content = file_get_contents($file);
    return $content;
}

function killStream(){
    $pid = shell_exec("pidof mjpg_streamer");

    if($pid != null){
        shell_exec("kill $pid &");
    } else {

    }

}

function initStream($mjpgFile){
    shell_exec("$mjpgFile &");
}

if(isset($_GET['cap']) && !isset($_GET['type'])){
    killStream();
    $img = system($camShellCMD);
    initStream($mjpgFile);
}

```

```

        writeLast($img,$file);
    }

    //Preparar para caso de tirar foto com efeitos.
    if(isset($_GET['cap']) && isset($_GET['type'])){
        $type = $_GET['type'];

        if($type === "1"){
            killStream();
            $type = "1";
            $camShellCMD = "$camShell $type $data";
            $img = system($camShellCMD);
            initStream($mjpgFile);
            writeLast($img,$file);
        }

        else if($type === "2"){
            killStream();
            $type = "2";
            $camShellCMD = "$camShell $type $data";
            $img = system($camShellCMD);
            initStream($mjpgFile);
            writeLast($img,$file);
        }

        else if ($type === "3"){
            killStream();
            $type = "3";
            $camShellCMD = "$camShell $type $data";
            $img = system($camShellCMD);
            initStream($mjpgFile);
            writeLast($img,$file);
        }
    }

    //Rotina para pegar ultima imagem salva em memória.
    if(isset($_GET['last'])){

```

```

    $img = getLast($file);
    //Retorna o endereço e nome da ultima imagem.
    echo $img;
}

if(isset($_GET['rmv'])){

}

?>

```

9.1.3 www.php/createDB.php

É o *script* responsável por criar as tabelas utilizadas no trabalho, além de inserir dados pré-definidos como padrões nas tabelas geradas.

```

<?php

#Organização das variáveis a serem primeiramente inseridas no Banco de
Dados:

$root = "admin";
$password = "admin";
$email = "tiny6410@gmail.com";
$machine1 = "143.107.235.53";
$machine2 = "143.107.235.51";
$tempDefault = "C";
$timerCamera = 30;

try
{
    //Iniciando as bases de dados.
    $db1 = new PDO('sqlite:../db/admin.sqlite');
    $db2 = new PDO('sqlite:../db/rf.sqlite');
    $db3 = new PDO('sqlite:../db/login.sqlite');

    //Criando as bases de dados
    $db1->exec("CREATE TABLE IF NOT EXISTS admin (Id INTEGER PRIMARY KEY,
temperature TEXT, timerCamera INTEGER, )");

```

```

$dbb2->exec("CREATE TABLE IF NOT EXISTS rf (Id INTEGER PRIMARY KEY,
machine1 TEXT, machine2 TEXT)");

$dbb3->exec("CREATE TABLE IF NOT EXISTS users (Id INTEGER PRIMARY
KEY, name TEXT, password TEXT, email TEXT)");

//Inserindo os dados iniciais
$dbb1->exec("INSERT INTO admin (temperature,timerCamera) VALUES
('$tempDefault','$timerCamera');");
$dbb2->exec("INSERT INTO rf (machine1,machine2) VALUES
('$machine1','$machine2');");
$dbb3->exec("INSERT INTO users (name,password,email) VALUES
('$root','$password','$email');");

#Fechar a conexão com o banco de dados.
$dbb = NULL;
} catch(PDOException $e)
{
    print 'Exception : '.$e->getMessage();
}

?>

```

9.1.4 www.php/foldernav.php

É o *script* responsável por responder a requisições de mapeamento de diretórios de músicas, vídeos e imagens. Com base nas requisições recebidas, os diretórios são mapeados de modo a gerar estruturas de páginas para imagens, vídeos ou músicas.

```

<?php

/*
 * Quando um usuário muda de página, o diretório é mudado.
 * Isso altera a variável dir.
 * Dessa forma, não é necessário carregar toodo o diretório principal nos
comandos. Apenas os índices.
 * Esses índices devem ser validados, de modo que somente arquivos de mídia
sejam executados.

```



```

*
*/

include '/www/php/variables.php';

/*
 * Construir rotinas de validação para impedir recursos que não devem ser
 * acessados.
 */

function navMedia($chdir){
    $dir = realpath($chdir);
    chdir($dir);
    echo "<p>Diretório: $dir</p>";
    //print_r($dir);
    $files = scandir($dir);
    //print_r($files);

    if(is_dir($files[1])){
        $retDir = realpath($files[1]);
        if(strcmp('/sdcard', $retDir)===0){}
        else if(strcmp('/sdcard', $retDir)<0)
            echo "<br/><input type=\"button\" value=\"Voltar\"
onclick=\"chdir('$retDir', 'md');\"/><br/>";
    }

    for($i = 2; $i < count($files) ; $i++){
        if(is_dir($files[$i])){
            $aux = realpath($files[$i]);
            echo "<p><a style=\"color:#00f;\" href=\"#\
onclick=\"chdir('$aux', 'md');\"> $files[$i] </a></p>";
        }

        //Mostrar apenas arquivos com extensões permitíves (ver
        análise de perfil...)
        if(is_file($files[$i])){
            $aux = realpath($files[$i]);

```

```

        echo "<p><a style=\"color:#000;\" href=\"#\
onclick=\"play('$aux');\">${files[$i]}</a></p>";
    }

}

}

function navVideos($chdir){
    $dir = realpath($chdir);
    chdir($dir);
    echo "<p>Diretório: $dir</p>";
    //print_r($dir);
    $files = scandir($dir);
    //print_r($files);

    if(is_dir($files[1])){
        $retDir = realpath($files[1]);
        if(strcmp('/sdcard', $retDir)==0){}
        else if(strcmp('/sdcard',$retDir)<0)
            echo "<br><input type=\"button\" value=\"Voltar\"
onclick=\"chdir('$retDir','vd');\"/><br>";
    }

    for($i = 2; $i < count($files) ; $i++){
        if(is_dir($files[$i])){
            $aux = realpath($files[$i]);
            echo "<p><a style=\"color:#00f;\" href=\"#\
onclick=\"chdir('$aux','vd');\"> ${files[$i]} </a></p>";
        }

        //Mostrar apenas arquivos com extensões permitíves (ver
        análise de perfil...)
        if(is_file($files[$i])){
            $aux = realpath($files[$i]);
            echo "<p><a style=\"color:#000;\" href=\"#\
onclick=\"playVideo('$aux',$('out').val());\">${files[$i]}</a></p>";
        }
    }
}

```

```

    }
}

function navPics($chdir){
    $dir = realpath($chdir);
    chdir($dir);
    echo "<p>Diretório: $dir</p>";

    $files = scandir($dir);
    //Arrumando a questão de nome do diretório.
    $dirAux = str_replace("/sdcard", "", $dir);

    if(is_dir($files[1])){
        $retDir = realpath($files[1]);
        if(strcmp('/sdcard', $retDir)==0){}
        else if(strcmp('/sdcard', $retDir)<0)
            echo "<br/><input type=\"button\" value=\"Voltar\"
onclick=\"chdir('$retDir', 'img');\"/><br/>";
    }

    $auxVar=0;

    echo "<table id=\"picTable\">";
    echo "<tr>";
    for($i = 2; $i < count($files) ; $i++){
        if(is_dir($files[$i])){
            $aux = realpath($files[$i]);
            echo "<td><a style=\"color:#00f;\" href=\"\"#\"
onclick=\"chdir('$aux', 'img');\"> $files[$i] </a></td></tr><tr>";

        }

        if(is_file($files[$i]) && $auxVar != 3){
            $pathParts = pathinfo($files[$i]);
            $aux = $pathParts['basename'];
            echo "<td><a style=\"color:#000;\" href=\"$dirAux/$aux\"><img
src=\"$dirAux/$aux\" width=160 height=120/>$aux</a></td>";
            $auxVar = $auxVar + 1;
        }
    }
}

```

```

        } else if(is_file($files[$i]) && $auxVar==3){
            echo "</tr>";
            $auxVar = 0;
            echo "<tr><td><a style=\"color:#000;\"
href=\"${dirAux}/${aux}\"><img src=\"${dirAux}/${aux}\" width=160
height=120/>${aux}</a></td>";

        }

    }
    echo "</td>";
    echo "</table>";

}

if(isset($_GET['img']) && isset($_GET['chdir'])){
    $dirPics=$_GET['chdir'];
    navPics($dirPics);
}

if(isset($_GET['md']) && isset($_GET['chdir'])){
    $chdir = $_GET['chdir'];
    if(is_dir($chdir)){
        navMedia($chdir);
    }

    else echo "error";
}

if(isset($_GET['vd']) && isset($_GET['chdir'])){
    $chdir = $_GET['chdir'];
    if(is_dir($chdir)){
        navVideos($chdir);
    }

    else echo "error";
}

```

```
?>
```

9.1.5 `www/php/gps.php`

É o *script* responsável por comunicar com o arquivo **gps.py**, mostrado no Apêndice 9.4.2. Além disso, esse *script* processa os dados recebidos, gerando retorno em formato bruto ou em formato compreensível ao usuário. O *script* **php/gps.php** ainda não está plenamente implementando e funcional. Ele será finalizado em versões posteriores do trabalho.

```
<?php
/*
 * Aqui ficam as rotinas de controle de GPS
 * Devem controlar o PYNMEA ou algum outro tipo
 * de parser de dados NMEA.
 * Abre serial, obtém valores, etc...
 * Display na tela.
 */

include '/www/php/variables.php';

$gpsShell = "/www/files/gps.py";
$data = shell_exec($gpsShell);

#### Agrupamento de funções de Parsing de Dados NMEA
function gps($gpsShell){
    $data = shell_exec("$gpsShell GPGGA");

    $gpsData = explode(",", $data);

    //print_r($gpsData);
    $type = $gpsData[0];
    $time = $gpsData[1];
    $latitude = $gpsData[2];
    $latType = $gpsData[3];
    $longitude = $gpsData[4];
    $longType = $gpsData[5];
    $fixQ = $gpsData[6];
    $sattellites = $gpsData[7];
```

```

$horDil = $gpsData[8];
$altitude = $gpsData[9];
$geoid = $gpsData[10];
$checksum = $gpsData[13];

## Tratamento da Hora
$hours = substr($time, 0,2);
$minutes = substr($time, 2,2);
$seconds = substr($time, 4,2);

### Tratamento da Latitude
$latDeg = (int) substr($latitude,0,2);
$latMin = substr($latitude,2);

### Tratamento da Longitude
$longDeg = (int) substr($longitude,0,3);
$longMin = substr($longitude,3);

echo "<table>";
echo "<tr><td>Tipo de Dado:</td><td>$type</td></tr>";
echo "<tr><td>Horário
UTC:</td><td>$hours"."":".$minutes"."":".$seconds."</td></tr>";
echo "<tr><td>Latitude:</td><td> $latDeg"."°"." $latMin"."'". "
$latType</td></tr>";
echo "<tr><td>Longitude:</td><td> $longDeg"."°"." $longMin"."'". "
$longType</td></tr>";
echo "<tr><td>Número de Satélites:</td><td>$sattelites</td></tr>";
echo "<tr><td>Altitude:</td><td> $altitude metros</td></tr>";
echo "</table>";
}

### Retorna a String pura, nao trabalhada.
function rawGPS($gpsShell,$type){
    return shell_exec("$gpsShell $type");
}

# GPS Fix Data
function gpgga($gpsShell){
    $data = shell_exec("$gpsShell GPGGA");

```

```

$gpsData = explode(",", $data);

//print_r($gpsData);
$type = $gpsData[0];
$time = $gpsData[1];
$latitude = $gpsData[2];
$latType = $gpsData[3];
$longitude = $gpsData[4];
$longType = $gpsData[5];
$fixQ = $gpsData[6];
$sattellites = $gpsData[7];
$horDil = $gpsData[8];
$altitude = $gpsData[9];
$geoid = $gpsData[10];
$checksum = $gpsData[13];

## Tratamento da Hora
$hours = substr($time, 0, 2);
$minutes = substr($time, 2, 2);
$seconds = substr($time, 4, 2);

### Tratamento da Latitude
$latDeg = (int) substr($latitude, 0, 2);
$latMin = substr($latitude, 2);

### Tratamento da Longitude
$longDeg = (int) substr($longitude, 0, 3);
$longMin = substr($longitude, 3);

echo "$type<br/>";
echo $hours."H ". $minutes."M ".$seconds."S ". "UTC<br/>";
echo "$latitude<br/>";
echo "$latDeg °<br/>";
echo "$latMin'<br/>";
echo "$latType<br/>";
echo "$longitude<br/>";
echo "$longDeg °<br/>";
echo "$longMin'<br/>";

```

```

    echo "$longType<br/>";
    echo "$sattelites satélites<br/>";
    echo "$altitude metros<br/>";
}

# GPS Minimum recommended Data
function gprmc($gpsShell){
    $data = shell_exec("$gpsShell GPGGA");

    $gpsData = explode(",", $data);

    //print_r($gpsData);
    $type = $gpsData[0];
    $time = $gpsData[1];
    $status = $gpsData[2];
    $latitude = $gpsData[3];
    $latType = $gpsData[4];
    $longitude = $gpsData[5];
    $longType = $gpsData[6];
    $speedGround = $gpsData[7];
    $trackAngle = $gpsData[8];
    $date = $gpsData[9];
    $magnetData = $gpsData[10];
    $magnetType = $gpsData[11];
    $checksum = $gpsData[12];
}

# GPS detailed satellite data
function gpgsv($gpsShell){
    $data = shell_exec("$gpsShell GPGSV");

    $gpsData = explode(",", $data);

    //print_r($gpsData);
    $type = $gpsData[0];
    $sentences = $gpsData[1];
    $sattelites = $gpsData[2];
    $satPRN = $gpsData[3];
    $elevationDeg = $gpsData[4];

```



```

    $azimutDeg = $gpsData[5];
    $SNR = $gpsData[6];

}

function gpgsa($gpsShell){

}

function gpgll($gpsShell){

}

### Função que gera arquivo TXT
function returnTXT($gpsData){
    header('Content-type: application/txt');
    header('Content-Disposition: attachment; filename="gpsData.txt"');
    echo $gpsData;
    echo "\r\n";
}

### Agrupamento de requisições GET
if (isset($_GET['pos'])){
    gps($gpsShell);
}

### Irá pegar apenas uma string do tipo selecionado.
if (isset($_GET['type']) && !isset($_GET['time'])){
    $type = $_GET['type'];
}

### Irá pegar strings do tipo selecionado no dado tempo, ou quantidade.
if (isset($_GET['type']) && isset($_GET['time'])){
    $type = $_GET['type'];
    $time = $_GET['time'];
}

### Irá gerar o arquivo do tipo selecionado no tempo dado.
if (isset($_GET['type']) && isset($_GET['file']) && isset($_GET['time'])){

```

```

$type = $_GET['type'];
$time = $_GET['time'];
}

### Irá gerar o arquivo do tipo selecionado...
if (isset($_GET['type']) && isset($_GET['file']) && !isset($_GET['time'])) {
    $type = $_GET['type'];
    $time = 10; //10 seconds - Basic interval. Or not.
}

?>

```

9.1.6 www/php/mail.php

É o *script* responsável por receber os dados de contato, preenchidos na página *Contato*. Após ter recebido os dados, os mesmos são tratados para depois serem enviados como parâmetros ao *script* **mail.py**, que é detalhado no Apêndice 9.4.4.

```

<?php

$mailShell = "/www/files/mail.py";

if (isset($_GET['nomeCtc']) && isset($_GET['mailCtc']) &&
isset($_GET['telCtc']) && isset($_GET['message'])) {
    $nomeCtc = $_GET['nomeCtc'];
    $mailCtc = $_GET['mailCtc'];
    $telCtc = $_GET['telCtc'];
    $msg = $_GET['message'];

    $message = "\nNOME: $nomeCtc\r\n
\nE-Mail: $mailCtc\r\n
\nTelefone: $telCtc\r\n
\nMensagem: \r\n $msg";

    $subject = "Recado: ".date("d-m-Y--H-i-s");
    $status = shell_exec("$mailShell \"$subject\" \"$message\"");

    echo $status;
}

```

```
}
?>
```

9.1.7 [www/php/mplayer.php](http://www.php/mplayer.php)

É um dos principais *scripts* desenvolvidos no trabalho. É responsável por tratar requisições para execução e controle de conteúdo multimídia, podendo disparar e controlar a execução de vídeos e músicas no **MPlayer**.

```
<?php
    session_start();
    include '/www/php/variables.php';

    /*
     * Setup do MPLAYER em Modo Slave
     */
    $command = "$mplayer $slave";

    $posFile = "";
    //Criar uma função de controle de volume.

    //Volume inicial, final e mínimo.
    //$vol = 50;
    settype($vol, "integer");
    $maxVolume = 100;
    settype($maxVolume, "integer");
    $minVolume = 0;
    settype($minVolume, "integer");

    $dir;
    $files;
    $filePos;
    $index;

    function isRunning(){
        $pidMP = shell_exec("pidof mplayer");
        if ($pidMP != NULL)
```

```

        return TRUE;
    else
        return FALSE;
}

function killMP(){
    $pidMP = shell_exec("pidof mplayer");
    shell_exec("kill $pidMP");
}

//Função para controlar o MPlayer ...
function play($cmd){
    if(isRunning()){
        killMP();
    } else
        shell_exec($cmd);
}

function getMedia($media){
    global $pathparts;
    global $dir;
    global $extension;
    global $files;
    global $filePos;
    global $index;

    //Pega todos os detalhes do Path do arquivo
    $pathparts = pathinfo($media);

    //Pega o diretório...
    $dir = $pathparts['dirname'];
    chdir($dir);

    //Pega a extensão do arquivo ...
    $extension = $pathparts['extension'];

    //Pega o nome do arquiv...
    $filename = $pathparts['basename'];

```

```

        $files = scandir($dir);
        $index = array_search($filename, $files, true);
    }

    function writeStatus($media,$status){
//      $tipo = mediaVerify($media);
        $fd = fopen($status,"w+");
        fwrite($fd,$media);
        fclose($fd);
    }

    function mediaVerify($media){

    }

    function notRunning(){
        echo "MPlayer is not running";
    }

//----- Tratamento das requisições GET -----//

    if(isset($_GET['play']) && isset($_GET['out'])){

        if(isRunning()){
            killMP();
        }

        $media = $_GET['play'];
        $mode = $_GET['out'];

        $fileplay = escapeshellarg($media);

        if ($mode == "3") {
            $out = "tvandlcd";
            $cmd = "$command $fileplay -tvout $out";
            writeStatus($fileplay, $status);
            play($cmd);
        }
    }

```

```

    }
    else if ($mode == "2") {
        $out = "tvonly";
        $cmd = "$command $fileplay -tvout $out";
        writeStatus ($fileplay, $status);
        play($cmd);
    }
    else if ($mode == "1") {
        $out = "off";
        $cmd = "$command $fileplay";
        writeStatus($fileplay, $status);
        play($cmd);
    }
}

else if(isset($_GET['play']) && !isset($_GET['out'])) {
    if(isRunning()){
        killMP();
    }
    $media = $_GET['play'];
    $file_play = $files[$index];

    $fileplay = escapeshellarg($media);

    writeStatus($fileplay,$status);
    $out = "off";

    $cmd = "$command $out $fileplay";

    play($cmd);
}

if(isset($_GET['stop'])){
    if(isRunning())
        killMP();
    else {
        notRunning();
    }
}

```

```

if(isset($_GET['pause'])){
    if(isRunning())
        shell_exec("echo pause > $mpcmd");
    else {
        notRunning();
    }
}

//Só funcionam se tiver mais arquivo na fila! Lista do diretório
atual? GET!

if(isset($_GET['nxt'])){
    //Tratar posição do arquivo no vetor de diretório.
    $posFile++;
    $media = realpath($files[$posFile]);
    $cmd = "$command $out $media";
    play($cmd);
}

if(isset($_GET['prv'])){
    //Tratar posição do arquivo no vetor de diretório.
    $posFile--;
    $media = realpath($files[$posFile]);
    $cmd = "$command $out $media";
    play($cmd);
}

if(isset($_GET['vol']) && !isset($_GET['u'])){
    // Só se for pra incrementar ou decrementar de 10 em 10...
    if(isRunning()){
        $volume = $_GET['vol'];
        $vol = $_GET['u'];
        //Comparar duas strings... Ver isso no Manual. ASAP.
        if($volume === "up"){
            if ($vol < 100){
                $vol = $vol;
            } else $vol = $maxVolume;
        }
    }
}

```

```

        $mpvol = escapeshellarg("Volume $vol 100");
        shell_exec("echo $mpvol > $mpcmd");
    }

    else if ($volume === "dw"){
        if($vol > 0) {
            $vol = $vol;
            echo $vol;
        } else $vol = $minVolume;
        $mpvol = escapeshellarg("volume $vol 100");
        echo $mpvol;
        shell_exec("echo $mpvol > $mpcmd");
    }
} else {
    notRunning();
}

}

if(isset($_GET['vol']) && isset($_GET['u'])){
    if(isRunning()){
        $volume = $_GET['vol'];
        $vol = $_GET['u'];
        //Comparar duas strings... Ver isso no Manual. ASAP.
        if($volume === "up"){
            if ($vol < 100){
                $vol = $vol;
            } else $vol = $maxVolume;
            $mpvol = escapeshellarg("Volume $vol 100");
            shell_exec("echo $mpvol > $mpcmd");
        }

        else if ($volume === "dw"){
            if($vol > 0) {
                $vol = $vol;
            } else $vol = $minVolume;
            $mpvol = escapeshellarg("Volume $vol 100");
            echo $mpvol;
            shell_exec("echo $mpvol > $mpcmd");
        }
    }
}

```



```

    }
    } else {
        notRunning();
    }
}

if(isset($_GET['mute'])){
    if(isRunning())
        shell_exec("echo mute > $mpcmd");
    else {
        notRunning();
    }
}

if(isset($_GET['camOut'])){
    // mkfifo a.mjpeg wget -O a.mjpeg http://localhost:8080
    2>/dev/null & mplayer -cache 32 -demuxer 35 a.mjpeg
    // ou mplayer -demuxer lavf http://localhost:8081/stream.mjpg
    shell_exec("mkfifo a.mjpeg wget -O a.mjpeg http://localhost:8080 2>
/dev/null & mplayer -cache 32 -demuxer 35 a.mjpeg");
}

?>

```

9.1.8 www/php/rf.php

É o *script* responsável por receber e tratar as requisições para envio e recebimento de mensagens do serviço de radiofrequência. Antes de tratar qualquer requisição, o *script* carrega as máquinas registradas no banco de dados **db/rf.sqlite**. Após isso, dependendo dos parâmetros da requisição, o aplicativo *cURL* será utilizado para enviar ou receber mensagens das máquinas escolhidas.

```

<?php
//Abertura do banco de dados RF, responsável pela persistência dos dados dos
endereços IPs das máquinas RF.
$db = new PDO('sqlite:../db/rf.sqlite');

//Definição do programa utilizado para requisições GET/POST

```

```

$cmd = "curl";

//Obtenção das máquinas que irão enviar/receber os dados RF.
#Máquina 1 - Envia
$query = $db->query('SELECT machine1,machine2 FROM rf');
$machines = $query->fetch();
$machine1 = $machines['machine1'];
$machine2 = $machines['machine2'];

if(isset($_GET['mach']) && isset($_GET['msg']) && !isset($_GET['hist'])){
    $emissor = $_GET['mach'];
    if($emissor == "1"){
        $machine = $machine1;
    } else if($emissor == "2"){
        $machine = $machine2;
    }

    //Obtenção da variável msg do tipo GET.
    $msg = $_GET['msg'];
    $msg = str_replace(" ", "%20",$msg);
    //Tratamento dos caracteres de escape
    //Organizando a URL de comando.
    $sendCmd = "$machine/Transfer/rf.php?msg=\"\$msg\"";
    //Escapes para o shell
    $sendCmd = escapeshellarg($sendCmd);
    //Organizando o
    // comando shell
    $cmdMsg = "$cmd $sendCmd";
    //Aplicando o comando e gravando o resultado em $result
    $result = shell_exec($cmdMsg);
    //Retornando o resultado ao usuário.
    echo "$machine: $result";
}

if(isset($_GET['mach']) && isset($_GET['hist']) && !isset($_GET['msg'])){

    $receptor = $_GET['mach'];

    if($receptor == "1"){
        $machine = $machine1;
    }
}

```

```

    } else if ($receptor == "2"){
        $machine = $machine2;
    }
    $histCmd = "$machine/Transfer/rf.php?hist=1";

    $cmdMsg = "$cmd $histCmd";
    sleep(4);
    $hist = shell_exec($cmdMsg);
    echo "$machine: $hist";

//http://143.107.235.53/Transfer/rf.php?hist

//http://143.107.235.51/Transfer/rf.php?msg=%22E%20ae%20pessoal,%20joinha????
?%22
}
?>

```

9.1.9 [www/php/status.php](http://www.php/status.php)

É o *script* responsável por responder requisições de *status* da estação. Serve para informar o *status* do *stream* de vídeo e da execução de músicas e vídeos. Também serve para controlar o processo de *stream* de vídeo e demais periféricos da Tiny6410, como o *display* LCD, *backlight* do *display*, dentre outros. Também permite obter a carga da CPU da estação, e até mesmo permite que o sistema seja reinicializado, através de requisições.

```

<?php
session_start();
/**
 * Escrever aqui funções de avaliação de espaço
 * livre em disco, espaço total, espaço usado
 * processo, uso de ram (/proc)
 * Se o mjpg_streamer está ativo,
 * o que está em execução, etc.
 *
 * Cada parâmetro ele retorna uma coisa diferente.
 *
 */
_include '/www/php/variables.php';

```

```

//Procurar aquelas funções que identificam padrões!
function typeOfMedia($status){
    $pathinfo = pathinfo($status);
    $type = $pathinfo['extension'];
    return $type;
}

function readMedia($status){
    $type = typeOfMedia($status);
    if ($type === "mp3"){
        echo "Musica! $status";
    }
    else if ($type === "avi"){
        echo "Vídeo! $status";
    }
}

if(isset($_GET['mjpgs'])){

    $mjpgStatus = shell_exec("pidof mjpg_streamer");

    if(strlen($mjpgStatus)===0){
        echo "<p style=\"color:red; text-align:center;\">MJPG Streamer
DOWN</p>";
    }
    else {
        echo "<p style=\"color:blue; text-align:center;\">MJPG Streamer
UP</p>";
    }

}

if(isset($_GET['m_up'])){
    shell_exec($mjpgFile);
}

if(isset($_GET['m_down'])){

```

```

    $pid = shell_exec('pidof mjpg_streamer');
    shell_exec("kill $pid &");
}

if(isset($_GET['current'])){
    $current = file_get_contents($status);
    echo $current;
}

if(isset($_GET['mpstatus'])){
    $pidMp = shell_exec("pidof mplayer");
    if(isset($pidMp)){
        //Ver o que Essah em execução
        echo "<p style=\"color:blue; text-align:center;\">MPlayer is
Up</p>";
    }
    else echo "<p style=\"color:red; text-align:center;\">MPlayer is
Down</p>";
}

if(isset($_GET['cpuload'])){

    $load = sys_getloadavg();
    $load = (floatval($load[0]));
    $load = $load*100;
    echo "<p style=\"text-align:center;\">$load %</p>";
}

//Rotina para reinicar a placa. É necessário autenticar com senha.
if (isset($_GET['reset'])){
    $pass = $_GET['reset'];
    if($pass == "1234"){ //Melhorar para pegar do DB.
        echo "Reboot";
        sleep(1);
        shell_exec("reboot");
    }
}
}

```

```

//Rotina para iniciar o QTopia
if (isset($_GET['startQtopia'])) {
    shell_exec("qtopia");

    $pidQt = shell_exec("pidof qtopia");

    if(isset($pidQt)){
        echo 1;
    }
    else echo 0;
}

//Rotina para finalizar o QTopia
if (isset($_GET['stopQtopia'])) {

    $pidQt = shell_exec("pidof qtopia");

    if(isset($pidQt)){
        shell_exec("kill $pidQt");
        echo 1;
    }

    if(!isset($pidQt)){
        echo 1;
    }
    else echo 0;
}

//Rotina para ligar o LCD
if (isset($_GET['lcdon'])) {

    shell_exec("echo 127 > /dev/backlight-1wire");
    echo 1;
}

//Rotina para desligar o LCD
if (isset($_GET['lcdoff'])) {

    shell_exec("echo 0 > /dev/backlight-1wire");
}

```

```

    echo 1;
}

//Rotina para ligar o backlight do display LCD
if (isset($_GET['backon'])) {
    shell_exec("echo 127 > /dev/backlight");
    echo 1;
}

//Rotina para desligar o backlight do display LCD
if (isset($_GET['backoff'])) {
    shell_exec("echo 0 > /dev/backlight");
}

if (isset($_GET['killMP'])) {
    $pidofMP = shell_exec("pidof mplayer");
    shell_exec("kill $pidofMP");
}

?>

```

9.1.10 [www/php/temp.php](http://www.php/temp.php)

É o *script* responsável por receber dados do programa que faz interface com o sensor de temperatura, e de posse destes dados, esse *script* responde a requisições retornando a temperatura ambiente em graus *Celsius*, *Fahrenheit* ou *Kelvin*. Também foi implementada uma funcionalidade que retorna, na forma de um arquivo texto, os valores de temperatura lidos em um intervalo de tempo em segundos.

```

<?php
/**
 * Escrever aqui funções de controle
 * do aplicativo de sensor de temperatura.
 *
 * Abre, pega retorno do valor em console,
 * fecha,

```

```

* e echo...
*
*/
include '/www/php/variables.php';

//Função que pega uma string, e retorna um arquivo texto ao usuário.
function returnData($string){
    date_default_timezone_set('America/Sao_Paulo');
    $dataInfo = date("d-m-Y--H-i-s");

    header('Content-type: application/txt; charset=utf-8');
    header('Content-Disposition: attachment; filename="tempData.txt"');
    echo $string;
    echo "\r\n";
}

//Tratamento de requisições simples de temperatura
if(isset($_GET['temp'])){
    $tempType = $_GET['temp'];

    if($tempType === 'C'){
        $temp = shell_exec($tempSensor);
        echo $temp;

    } else if($tempType === 'F'){
        $temp = shell_exec($tempSensor);
        $temp = ($temp/5)*9+32;
        echo $temp;

    } else if($tempType === 'K'){
        $temp = shell_exec($tempSensor);
        $temp = $temp + 273;
        echo $temp;
    }
}

//Conjunto de parâmetros GET para montagem do arquivo de temperatura.
if(isset($_GET['type']) && isset($_GET['file']) && isset($_GET['time'])){

```



```

$type = $_GET['type'];
$time = (int)$_GET['time'];
$dataInfo = date("d-m-Y--H-i-s");
$timeAval = date("H-i-s");

$string = "### Relatório de Dados de Temperatura ###\n\r";
$string .= "$dataInfo\n\r";
$tempAux = "";
$temp = "";

if($type === 'C'){
    for($i = 0; $i <= $time; $i++){
        $timeAval = date("H-i-s");
        $tempAux = shell_exec($tempSensor);
        $string .= "$timeAval: $tempAux\n\r";
        sleep(1);
        $timeAval = date("H-i-s");
        $tempAux = shell_exec($tempSensor);
        $string .= "$timeAval: $tempAux\n\r";
        sleep(1);
    }

    //Chama a função de retornar o arquivo ao usuário, com a string
gerada.

    returnData($string);
}
else if($type === 'F'){
    for($i = 0; $i <= $time; $i++){
        $tempAux = shell_exec($tempSensor);
        $temp .= ($tempAux/5)*9+32 ."\r\n";
        sleep(1);
        $tempAux = shell_exec($tempSensor);
        $temp .= ($tempAux/5)*9+32 ."\r\n";
    }

    //Chama a função de retornar o arquivo ao usuário, com a string
gerada.

```

```

        returnData($temp);

    } else if($type === 'K'){
        for($i = 0; $i <= $time; $i++){
            $tempAux = shell_exec($tempSensor);
            $temp .= $tempAux + 273 ."\r\n";
            sleep(1);
            $tempAux = shell_exec($tempSensor);
            $temp .= $tempAux + 273 ."\r\n";
        }
        //Chama a função de retornar o arquivo ao usuário, com a string
gerada.
        returnData($temp);
    }
}

?>

```

9.1.11 [www/php/variables.php](http://www.php.net/variables.php)

Esse é o *script* PHP responsável por definir algumas variáveis utilizadas em toda a interface *web* do projeto, tais como endereços de binários e demais *scripts* utilizados, além dos textos em português ou inglês utilizados no projeto.

```

<?php
/*
 *
 * Colocar aqui variáveis importantes utilizadas?
 */

//$GLOBALS['$dir'] = "";
$pid = "";
global $dir;

//-----Variáveis para controle do MPlayer
$mplayer = '/bin/mplayer';
$file = "/www/files/mplayer.cmd";

```

```

$mpcmd = escapeshellarg($file);

$slave = "-slave -input file=$file";
$tvout = "-tvout";
$out = "tvonly"; //Por default, o padrão é tvonly

$status = "/www/files/status.txt";

$geral = "*";

$mjpgFile = '/mjpg-streamer/start_uvc.sh';
$mjpgFile2 = '/mjpg-streamer/start_uvc2.sh';

$tempSensor = "/home/plg/TempSensor/tempSensor";

//---- Variáveis para informação do sistema.
$memFree = "";
$memUsed = "";
$memTotal = "";

//Default: $lang = pt
$lang = "pt";

//-----Nome dos menus-----
--//
$InicialPT = "Inicial";
$InicialEng = "Home";

$MusicasPT = "Músicas";
$MusicasEng = "Musics";

$RadioPT = "Rádios";
$RadioEng = "Radios";

$VideosPT = "Vídeos";
$VideosEng = "Videos";

$CameraPT = "Câmera";
$CameraEng = "Camera";

```

```

$ImagensPT = "Imagens";
$ImagensEng = "Images";

$GPS = "GPS";

$InfraPT = "Infravermelho";
$InfraEng = "Infrared";

$msgPT = "Mensagem";
$msgEng = "Message";

$DocsPT = "Documentos";
$DocsEng = "Documments";

$AdminPT = "Administração";
$AdminEng = "Administration";

$SobrePT = "Sobre";
$SobreEng = "About";

$RFPT = "RF";
$ContatoPT = "Contato";

$TempPT = "Temperatura";

//-----Setando os Status -----
--//
$tocandoPT = "Tocando ";
$tocandoEng = "Playing ";

$assistindoPT = "Assistindo ";
$assistindoEng = "Watching ";

$ouvindoPT = "Ouvindo ";
$ouvindoEng = "Listening ";

//-----Setando os Menus -----

```

```

--//
if($lang === "pt"){
    $Home = $InicialPT;
    $Musicas = $MusicasPT;
    $Radios = $RadioPT;
    $Videos = $VideosPT;
    $Camera = $CameraPT;
    $Imagens = $ImagensPT;
    $GPS = $GPS;
    $Infra = $InfraPT;
    $Mensagem = $MsgPT;
    $Docs = $DocsPT;
    $Admin = $AdminPT;
    $Sobre = $SobrePT;
    $Temp = $TempPT;
    $RF = $RFPT;
    $Contato = $ContatoPT;
}

if($lang === "eng"){
    $Home = $InicialEng;
    $Musicas = $MusicasEng;
    $Radios = $RadioEng;
    $Videos = $VideosEng;
    $Camera = $CameraEng;
    $GPS = $GPS;
    $Infra = $InfraEng;
    $Mensagem = $MsgEng;
    $Docs = $DocsEng;
    $Admin = $AdminEng;
    $Sobre = $SobreEng;
}

//-----Conteúdos das páginas -----
--//

$nomePT = "Estação de Controle Multimídia Web";
$nomeEng = "Web Multimedia Control Station Center";

$descricaoPT = "";

```

```
$descricaoEng = "";
```

```
?>
```

9.2 Arquivos JavaScript Desenvolvidos

Nessa sessão são apresentados os *scripts* JavaScripts escritos para controle de elementos HTML da interface *web* desenvolvida no projeto, e também necessários para envio e recebimento de requisições entre o *browser* cliente e a máquina servidora.

9.2.1 [www/js/ajax.js](#)

É o *script* responsável por boa parte das requisições AJAX, tais como a exibição da temperatura ambiente, uso de CPU, *status* do *stream* de vídeo e do MPlayer, e também demais interações envolvendo controle da Estação e tirada de fotos.

```
var interval=0;

function loadPage(page){

    $('#content').load(page, function(){
        $.getScript("js/ajax.js");
        $.getScript("js/mediaControl.js");
        $.getScript("js/jquery-1.7.2.min.js");
    });

}

function setLang(language){
    if(language=="pt"){
        $.get("index.php?pt");
    } else if (language == "eng"){
        $.get("index.php?eng");
    }
}
```

```

    }

}

function startStream(){
    $.get("php/status.php?m_up");
    setTimeout(loadStatus,2000);
}

function stopStream(){
    $.get("php/status.php?m_down");
    setTimeout(loadStatus,1500);
}

function chdir(dir,type){

    if(type === "md"){
        $.get("php/foldernav.php?md&chdir="+dir, function(dir){
            $('#folder').html(dir);});
    }
    else if(type ==="img"){
        $.get("php/foldernav.php?img&chdir="+dir, function(dir){
            $('#folder').html(dir);});
    }
    else if(type ==="vd"){
        $.get("php/foldernav.php?vd&chdir="+dir, function(dir){
            $('#folder').html(dir);});
    }

}

function loadStatus(){
    $.get("php/status.php?mjpgs", function(status){
        $('#statusServer').html(status);});
}

function mpStatus(){

```

```

$.get("php/status.php?mpstatus", function(mpstatus){
    $('#mpStatus').html(mpstatus);
});
}

function currentPlaying(){
    $.get("php/status.php?current", function(current){
        $('#currMedia').html(current);
    });
}

function getCpuLoad(){
    $.get("php/status.php?cpuload", function(cpuload){
        $('#cpuload').html(cpuload);
    });
}

function message(mensagem){
    $.get("php/mensagem.php?msg="+mensagem , function(retData){

    });
}

function snapShot(type){
    $.get("php/camCapture.php?cap&type="+type);
}

function getLastPic(){
    $.get("php/camCapture.php?last", function(picture){
        myWindow=window.open("/pictures/"+picture);
        myWindow.focus();
    });
}

function delLastPic(){
    $.get("php/camCapture.php?delLast,", function(retValue){
        //Funcao de retorno. Gera janela de aviso com confirmacao de
        imagem deletada.
    });
}

```



```

        alert("Deseja realmente remover a foto: "+retValue+" ?");
    });
}

function initAjax(){
    setInterval('loadTemp("c")',5000);
    setInterval(loadStatus,5000);
    setInterval(mpStatus,5000);
    setInterval(getCpuLoad,1000);
}

function startVideo(type){
    $.get("php/mpplayer?camOut="+type);
}

function formatData(type){
    var pass = prompt("Os dados do tipo "+type+" serão formatados! Digite a senha:");
    $.get("/php/status.php?format="+type+"&pass="+pass);
}

function restartSystem(){
    var pass = prompt("O sistema será reiniciado. Insira a senha: ");
    $.get("/php/status.php?reset="+pass);
}

function sendEmail(name,mail,tel,msg){

    //Trata a mensagem antes de enviar ao script php. Retira caracteres
    que podem causar problemas, tais como espaço, recuo, etc.
    var message = msg.replace(/\r\n|\r|\n/g,"\n");
    $.get("/php/mail.php?nomeCtc="+name+"&mailCtc="+mail+"&telCtc="+tel+"&message="+message,function(status){
        //if(status === "1"){
        //    alert("Email enviado com sucesso!");
        //}

    });
}

```

```
// Funcao que inicia o QTopia da Essacao.
function startQtopia(){
    $.get("/php/status.php?startQtopia",function(status){
        });
}

// Funcao que envia a requisicao para matar o processo do Qtopia.
function stopQtopia(){
    $.get("php/status.php?stopQtopia",function(status){

        });
}

//
function lcdOn(){
    $.get("php/status.php?lcdon",function(status){

        });
}

//
function lcdOff(){
    $.get("php/status.php?lcdoff",function(status){

        });
}

function backLightOn(){
    $.get("php/status.php?backon",function(status){

        })
}

function backLightOff(){
    $.get("php/status.php?backoff",function(status){

        });
}
}
```

9.2.2 [www/js/gps.js](#)

É o *script* JavaScript responsável por interações com o serviço de dados GPS da Estação. Esse *script* dispara ações na página GPS do Menu. Cada um dos botões dessa página está associado a uma função desse *script*. Ele também faz a requisição AJAX que mostra os

dados GPS na tela da página GPS. Este *script* ainda não está completo, e será desenvolvido em versões posteriores do trabalho.

```
var gpsTime;

function getGpsData(){
    switch ($("#select#gpsType").val()){
        case 1:

            break;

        case 2:

            break;

        case 3:

            break;

        case 4:

            break;

        case 5:

            break;

        default:
    }
}

function gpsPosition(){
    $.get("php/gps.php?pos",function(gps){
        $('#gpsPos').html(gps);
    });
}

function startGps(){
    gpsTime = setInterval(getGpsData(),1000);
}
```

```

function stopGps(){
    clearInterval(gpsTime);
}

function getGpsFile(type,time){
    $.get("php/gps.php?type="+type+"&time="+time+"&file",function(retData)
{

    });
}

function getGpsFile(type){
    $.get("php/gps.php?type="+type+"&time="+time+"&file",function(retData)
{

    });
}

function aviso(){
    alert("O tempo de resposta pode demorar, tenha paciência!");
}

```

9.2.3 [www/js/mediaControl.js](#)

É o *script* utilizado para controle de requisições compatíveis com o *script* **php/mplayer.php**, mostrado no Apêndice 9.1.7. Cada uma das funções mostradas podem ser associadas a elementos HTML para disparo de requisições de controle multimídia.

```

var volume=50; //Obter volume via get de sessão?
var status;

function play(file){
    $.get("php/mplayer.php?play="+file);
}

function playVideo(file,type){
    $.get("php/mplayer.php?play="+file+"&out="+type);
}

```

```
function stop(){
    $.get("php/mplayer.php?stop");
}

function pause(){
    $.get("php/mplayer.php?pause");
}

function next(){
    $.get("php/mplayer.php?nxt");
}

function prev(){
    $.get("php/mplayer.php?prv");
}

function volumeUp(){
    if(volume === 100){
        volume = 100;
        $.get("php/mplayer.php?vol=up&u="+volume);
        $('#volume').html(volume);
    }
    else {
        volume+=10;
        $.get("php/mplayer.php?vol=up&u="+volume);
        $('#volume').html(volume);
    }
    //Mudar status do volume no p id.
}

function volumeDown(){
    if(volume === 0){
        volume = 0;
        $.get("php/mplayer.php?vol=dw&u="+volume);
        $('#volume').html(volume);
    }else{
        volume-=10;
    }
}
```

```

$.get("php/mplayer.php?vol=dw&u="+volume);
$('#volume').html(volume);
}

//Mudar status do volume no p id.
}

function mute(){
    $.get("php/mplayer.php?mute");
}

function killMP(){
    $.get("php/status.php?killMP");
}

```

9.2.4 www/js/rf.js

É o *script* JavaScript responsável por tratar, enviar e receber mensagens de e para o servidor, mais precisamente para o *script* PHP **/php/rf.php** (visto no Apêndice 9.1.8) na máquina servidora. Faz uso de requisições AJAX para mostrar o resultado ao usuário.

```

/* Função que envia mensagem, pegando como parâmetro o texto escrito e o
emissor escolhido. A mensagem é tratada, retirando-se alguns caracteres, e
então é enviada. O servidor envia uma resposta,e essa é exibida na div
tempData. */
function sendRf(emissor,txt){
    var message = txt.replace(/\r\n|\r|\n/g," ");
    $.get("php/rf.php?mach="+emissor+"&msg="+message,function(result){
        $('#tempData').html(result);
    });
}

/* Função que recebe uma mensagem do servidor. Recebe como parâmetro o
receptor escolhido. Faz a requisição e mostra a resposta na div tempData ao
usuário. */
function receiveRf(receptor){
    $.get("php/rf.php?mach="+receptor+"&hist=1",function(hist){
        $('#tempData').html(hist);
    });
}

```

```
// Função que realiza a contagem de caracteres.
function countChars(size) {
    var l = size;
    var str = document.getElementById("textArea").value;
    var len = str.length;
    if(len <= 1) {
        document.getElementById("txtLen").value=l-len;
    } else {
        document.getElementById("textArea").value=str.substr(0, 140);
    }
}
```

9.2.5 www/js/temp.js

Esse é o *script* JavaScript responsável pelo controle de elementos e de requisições pertinentes à exibição de temperatura ambiente na página *web* do projeto. Realiza requisições conforme a temperatura escolhida. Possui funções que funcionam de modo temporizado, realizando requisições ao servidor a cada intervalo de tempo, e possui função que paralisa as funções temporizadas.

```
var tempId;
var intervalTemp=0;
function loadTemp(type){

    //Ver uso e aplicação de switch em JavaScript

    if(type === "c"){
        $.get("php/temp.php?temp=C", function(temp){
            $('#temp').html(temp + " °C");});
    }
    else if(type ==="f"){
        $.get("php/temp.php?temp=F", function(temp){
            $('#temp').html(temp + " °F");});
    } else if (type === "k") {
        $.get("php/temp.php?temp=K", function(temp){
            $('#temp').html(temp + " K");});
    }

}
```

```

function loadTempPG(type){

    if(type === "c"){
        $.get("php/temp.php?temp=C", function(temp){
            $('#tempDiv').html("<h3 style=\"color:blue;\">"+temp+"
°C</h3>");});
        }
        else if(type === "f"){
            $.get("php/temp.php?temp=F", function(temp){
                $('#tempDiv').html("<h3 style=\"color:blue;\">"+temp+"
°F</h3>");});
            } else if (type === "k") {
                $.get("php/temp.php?temp=K", function(temp){
                    $('#tempDiv').html("<h3 style=\"color:blue;\">"+temp+"
K</h3>");});
            }
        }

function changeTemp(type){
    clearInterval(tempId);
    if(type === "1"){
        loadTempPG("c");
    }
    if(type === "2"){
        loadTempPG("f");
    }
    if(type === "3"){
        loadTempPG("k");
    }
    else ;
}

function loadTempAll(type){
    intervalTemp = intervalTemp+1;
    //Definindo 10 amostrar na div de saída.
    if (intervalTemp === 10){
        intervalTemp=0;
        $('#tempData').empty();
    }
}

```



```

    }

    if(type === "c"){
        $.get("php/temp.php?temp=C", function(temp){
            $('#tempDiv').html("<h3 style=\"color:blue;\">"+temp+"
°C</h3>");
            $('#tempData').append("<p style=\"text-align:center;\">"+temp+"
°C</p>");}
        );
    }
    else if(type === "f"){
        $.get("php/temp.php?temp=F", function(temp){
            $('#tempDiv').html("<h3 style=\"color:blue;\">"+temp+"
°F</h3>");
            $('#tempData').append("<p style=\"text-align:center;\">"+temp+"
°F</p>");}
        );
    } else if (type === "k") {
        $.get("php/temp.php?temp=K", function(temp){
            $('#tempDiv').html("<h3 style=\"color:blue;\">"+temp+"
K</h3>");
            $('#tempData').append("<p style=\"text-align:center;\">"+temp+"
K</p>");}
        );
    }
}

function startTemp(type){
    //Limpa a Div de Relatorio
    $('#tempData').empty();

    clearInterval(tempId);

    //Avalia cada tipo de temperatura e faz os requests no dado tempo.
    if(type === "1"){
        tempId = setInterval("loadTempAll(\"c\")",5000);
    }
}

```

```

        if(type === "2"){
            tempId = setInterval("loadTempAll(\"f\")",5000);
        }
        if(type === "3"){
            tempId = setInterval("loadTempAll(\"k\")",5000);
        }
        else ;
    }

    //Função para parar os requests Ajax.
    function stopTemp(){
        clearInterval(tempId);
    }

```

9.3 Scripts de Páginas PHP

Nessa parte estão todos os *scripts* PHP. Apesar de serem *scripts* PHP, códigos HTML encontram-se misturados, em decorrência do fato de que PHP permite sintetizar e modificar conteúdo HTML conforme requisições específicas.

Ao contrário da parte de controle PHP, essa parte se reserva pura e unicamente às páginas que lidam com o conteúdo mostrado ao usuário.

9.3.1 `www/index.php`

Esse código é referente ao *script* da página principal do projeto. É o primeiro arquivo a ser chamado pelo *lighttpd* ao receber uma requisição *web*. Carrega algumas variáveis padrões definidas no arquivo **php/variables.php**. Já está preparado para lidar com idiomas diferentes. Além disso, carrega todos os *scripts* JavaScript necessários para controle de elementos HTML e requisições AJAX entre a página e o servidor.

Esse arquivo também define toda a estrutura da página *web* do projeto, tendo um Menu lateral de seleção das opções, uma coluna central para exibição de conteúdo, e uma coluna na lateral direita para controle de funções multimídia e exibição de alguns *status* da Estação.

```

<?php
session_start();
include '/www/php/variables.php';
if (isset($_GET['pt']))
    $_SESSION['lang'] = "pt";
else if (isset($_GET['eng']))

```

```

    $_SESSION['lang'] = "eng";
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <title>Arm Web Media Station</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <link rel="stylesheet" href="css/style.css" type="text/css" />
    <!--[if IE 6]>
    <link rel="stylesheet" href="fix.css" type="text/css" />
    <![endif]-->
    <script type="text/javascript" src="js/jquery-1.7.2.min.js"></script>
    <script type="text/javascript" src="js/ajax.js"></script>
    <script type="text/javascript" src="js/rf.js"></script>
    <script type="text/javascript" src="js/gps.js"></script>
    <script type="text/javascript" src="js/temp.js"></script>
    <script type="text/javascript" src="js/mediaControl.js"></script>

    <script type="text/javascript">
      $(document).ready(function(){
        loadStatus();
        getCpuLoad();
        loadTemp("c");
        mpStatus();
        initAjax();
      });
    </script>

  </head>

  <body>
    <div id="sidebar" align="center">
    <div align="left"><a href="./?l=pt" id="pt"> Português</a>
    <a href="./?l=eng" id="eng"> English</a><br/>
    <? if (isset($_SESSION['login']))){
      echo "<br/>".$_SESSION['name'];
      echo "<br/><a href=\"./php/auth.php?logout\">Logout</a>";
    } else {

```

```

        echo "<a href=\"./?p=login\" id=\"login\">Login</a>";
    }

?>
</div>

<h1>ARM WEB MEDIA STATION</h1>
<h2> <? echo $nomePT; ?>

</h2>

<div id="menu" align="center">
    <a class="active" href=<? echo "./?p=home";?> ><? echo $Home;?></a>
    <a href=<? echo "./?p=musicas";?>><? echo $Musicas;?></a>
    <a href=<? echo "./?p=videos";?> ><? echo $Videos;?></a>
    <a href=<? echo "./?p=camera";?> ><? echo $Camera;?></a>
    <a href=<? echo "./?p=imagens";?> ><? echo $Imagens;?></a>
    <a href=<? echo "./?p=gps";?> ><? echo $GPS;?></a>
    <a href=<? echo "./?p=temp";?> ><? echo $Temp;?></a>
    <a href=<? echo "./?p=rf";?> ><? echo $RF;?></a>
    <a href=<? echo "./?p=message";?> ><? echo $Mensagem;?></a>
    <a href=<? echo "./?p=docs";?> ><? echo $Docs;?></a>
    <a href=<? echo "./?p=admin";?> ><? echo $Admin;?></a>
    <a href=<? echo "./?p=about";?> ><? echo $Sobre;?></a>
    <a href=<? echo "./?p=contact";?> ><? echo $Contato; ?></a>
</div>

<h3 style="text-align: center;">Versão</h3>
<p style="text-align: center;">v3.5 (Oct 13, 2012)</p>
</div>

<div id="content" align="center">
    <?php
    if(!isset($_GET['p'])){
        include('pages/home.php');
    } else
    include('pages/'.$_GET['p'].'.php');

    ?>
</div>

```

```

<div id="control" align="center">
  <table>
    <tr>
      <td>Temperatura Local:</td><td id="temp"></td>
    </tr>
  </table>
  <hr/>
  <h1>Status</h1>
  <hr/>
  <div id="cpuReport">
    Carga da CPU
    <p id="cpuload"></p>
  </div>
  <hr />
  <div id="status">
    MJPEG Stream Status:
    <br/>
    <p id="statusServer"></p>

  </div>

  <hr/>
  <div id="mediaControl" align="center">
    <h2>Controle</h2>
    <table>
      <tr>
        <td><input type="button" value="Play" onclick="play();" /></td>
        <td><input type="button" value="Pause"
onclick="pause();" /></td>
        <td><input type="button" value="Stop"
onclick="stop();" /></td>
      </tr>
      <tr>
        <td><input type="button" value="Next" onclick="next();" /></td>

        <td><input type="button" value="Prev" onclick="prev();" /></td>

      </tr>
    </table>
  </div>

```

```

        <tr>
            <td><input type="button" value="Vol+"
onclick="volumeUp();" /></td>
        </tr>
        <tr>
            <td>Volume:</td><td><p id="volume">50</p></td>
        </tr>
        <tr>
            <td><input type="button" value="Vol-"
onclick="volumeDown();" /></td>
        </tr>
        <tr>
            <td><input type="button" value="Mute" onclick="mute();" /></td>
        </tr>
    </table>
    <hr />
</div>

<div id="current" style="text-align: center;">
    <h2>Mídia em Execução</h2>
    <div id="mpStatus"></div>
    <div id="currMedia"></div>
</div>

</div>

<div id="footer" align="center">
    Universidade de São Paulo - Escola de Engenharia de São Carlos
</div>
</body>
</html>

```

9.3.2 www/pages/admin.php

Esse é o *script* PHP responsável pela página de Administração, do Menu de controle da interface *web* da Estação. Possui uma variedade de funções PHP que tratam de obter dados da Estação, tais como uso de memória, processos em execução, etc.

Além disso, esse *script* faz uso de algumas rotinas AJAX para disparo de requisições de controle à Estação, tais como iniciar ou terminar o *stream* de vídeo, ligar ou desligar o *display* LCD da Estação, dentre outros. Os conteúdos de controle da Estação só são exibidos a usuários autenticados.

```
<?php
session_start();
include "php/variables.php";

function uptime(){

    $upAux = shell_exec("cat /proc/uptime");
    $uptime = explode(" ", $upAux);
    //A partir de agora, uptime principal é $uptime[0];
    $secondsAux = $uptime[0];
    $seconds = intval($secondsAux % 60);

    $minutes = intval($secondsAux / 60 % 60);

    $hours = intval($secondsAux / 3600 % 24);

    $days = intval($secondsAux / 86400);

    if ($days > 0) {
        $uptimeString .= $days;
        $uptimeString .= (($days == 1) ? " dia" : " dias");
    }
    if ($hours > 0) {
        $uptimeString .= (($days > 0) ? ", " : "") . $hours;
        $uptimeString .= (($hours == 1) ? " hora" : " horas");
    }
    if ($mins > 0) {
        $uptimeString .= (($days > 0 || $hours > 0) ? ", " : "") . $mins;
        $uptimeString .= (($mins == 1) ? " minuto" : " minutos");
    }
    if ($secs > 0) {
        $uptimeString .= (($days > 0 || $hours > 0 || $mins > 0) ? ", " : "") .
        $secs;
    }
}
```

```

    $uptimeString .= (($secs == 1) ? " segundo" : " segundos");
    }

    return $uptimeString;
}

?>
<script type="text/javascript">
    $(document).ready(function(){
        loadStatus();
    });
</script>
<h2>Administração</h2>
<hr/>
<h3>Dados da Estação:</h3>
<table id="stationData">
    <tr>
        <td>Tempo em operação:</td> <td><?php echo uptime(); ?></td>
    </tr>
    <tr>
        <td>Processador:</td><td>ARM 1176JZF-S</td>
    </tr>
    <tr>
        <td>Frequência Proc.:</td> <td><?php
            $cpufreqAux= shell_exec("cat /proc/cpuinfo | grep BogomIPS");
            $cpufreq = explode(":", $cpufreqAux);
            echo "$cpufreq[1] Mhz";
            ?>
        </td>
    </tr>
    <tr>
        <td>Sistema Operacional:</td> <td><?php echo shell_exec("uname
-r");?></td>
    </tr>
    <tr>
        <td>Memória RAM Total:</td> <td><?php
            $ramAux = shell_exec("free | grep Mem");
            $vecRam = explode(" ", $ramAux);
            echo round($vecRam[1]/1024,2) ." MB";
        </td>
    </tr>
</table>

```



```

        ?></td>

    </tr>
    <tr>
        <td>Memória RAM Disponível:</td> <td><?php echo
round($vecRam[4]/1024,2) ." MB";?></td>
    </tr>
    <tr>
        <td>Memória RAM Utilizada:</td> <td><?php echo
round($vecRam[3]/1024,2) ." MB";?></td>
    </tr>
    <tr>
        <td>Número de Processos Ativos:</td> <td><?php echo
shell_exec("ps | wc -l");?></td>
    </tr>
    <tr>
        <td>Memória FLASH Total:</td> <td><?php echo
round(disk_total_space("/sdcard")/1024/1024,2) ." MB"; ?></td>
    </tr>
    <tr>
        <td>Memória FLASH Utilizada:</td> <td><?php echo
round((disk_total_space("/sdcard")-disk_free_space("/sdcard"))/1024/1024,2)
." MB";?></td>
    </tr>
    <tr>
        <td>Memória FLASH Disponível:</td> <td><?php echo
round(disk_free_space("/sdcard")/1024/1024,2) ." MB"; ?></td>
    </tr>
    <tr>
        <td>Uso da Banda de Rede:</td> <td><?php ?></td>
    </tr>
    <tr>
        <td>Mídia em Execução:</td>
    </tr>
    <tr>
        <td id="midia"></td>
    </tr>
</table>
<?php if($_SESSION['login'] == false){

```

```

?>
<h3>É preciso estar autenticado para ter acesso aos controles!</h3>
<hr/>
<?> } else { ?>
<hr/>
<div id="mjpgCtrl" align="center">
<h3>MJPEG Streamer Control</h3>
<table>
    <tr><td><input type="button" onclick="startStream();" value="Iniciar
Stream" /></td>
        <td><input type="button" onclick="stopStream();" value="Pausar
Stream" /></td>
    </tr>
</table>
</div>
<hr/>
<h3>Controle da Temporização de Captura de Fotos:</h3><br/>
<table>
    <tr>
        <td>Configuração Atual: </td><td id="tmpAtual"></td>
    </tr>
    <tr>
        <td>Alterar (Minutos): </td><td><input type="number"
id="camTime" size="3"/></td>
    </tr>
    <tr><td><input type="button" value="Aplicar"
onclick="alterTimeCam($('#camTime').val()"/></td></tr>
</table>

<h3>Formatar Dados?</h3>
<select id="format">
    <option id="m">Músicas</option>
    <option id="v">Vídeos</option>
    <option id="i">Imagens</option>
    <option id="a">TUDO</option>
</select>
<input type="button" value="Aplicar"
onclick="formatData($('#format').val());"/>

```

```

<br/>
<hr/>
<h3>Controle de Sistema</h3>
<table>
<tr>
    <td><input type="button" value="Iniciar QTopia"
onclick="startQtopia()"/></td>
    <td><input type="button" value="Fechar QTopia"
onclick="stopQtopia()"/></td>
</tr>
<tr>
<!-- Trocar parametros aqui entre lcd e backlight -->
    <td><input type="button" value="Ligar Display LCD"
onclick="lcdOn()"/></td>
    <td><input type="button" value="Desligar Display LCD"
onclick="lcdOff()"/></td>
</tr>
<tr>
    <td><input type="button" value="Ligar Backlight"
onclick="backLightOn()"/></td>
    <td><input type="button" value="Desligar Backlight"
onclick="backLightOff()"/></td>
</tr>
<tr>
    <td><input type="button" value="Reiniciar Sistema"
onclick="restartSystem();"/></td>
</tr>
<tr>
    <td><input type="button" value="Controle de Usuários"
onclick="parent.location='index.php?p=users'"/></td>
</tr>
</table>
<br/><br/><br/>
<? } ?>

```

9.3.3 www/pages/câmera.php

Esse é o *script* PHP da página Câmera, do Menu lateral da página *web* do projeto. Primeiramente, só irá exibir o conteúdo da página para usuários autenticados no sistema. No caso de o usuário ser autenticado, irá exibir toda a estrutura HTML. Os *streams* são tratados como imagens, e o controle da câmera 1 é mostrado de maneira tabelada, executado por botões que disparam funções JavaScript. Essas funções, por sua vez, disparam requisições AJAX ao servidor.

```
<?php
    session_start();
    if($_SESSION['login'] == true){
?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
    <hr/>
    <h2>Câmera</h2>
    <hr/>
    <br />

<table id="cameras">
    <tr><td>Camera 1</td><td>Camera 2</td></tr>
    <tr><td></td><td></td></tr>
    <!-- Tive problemas com a porta 8080. Mudei para porta 8082,
redirecionei para www padrao. Funciona agora stream direto, sem socket. Tira
carga do PHP. Melhora resposta.-->
</table>
    <br/><br/><br/><br/><br/>
    <div id="controlCamera" align="center">

    <h2>Controle da Câmera 1</h2>
    <hr/>
    <table id="camControl">
        <tr>
            <select id="effect">
                <option value="1">Colorida</option>
```

```

        <option value="2">Preto-e-Branca</option>
    </select>

</tr>
<tr>
    <input type="button" value="Tirar Foto"
onclick="snapShot($('#effect').val());"/>
</tr>
<tr>
    <input type="button" value="Visualizar Foto"
onclick="getLastPic()"/>
</tr>
<tr>
    <input type="button" value="Excluir Foto"
onclick="delLastPic()"/>
</tr>
</table>
<br/>
<br/>
<br/>
<h2>Ativar Streaming</h2>
<hr/>
Saída:
<select id="streamOut">
    <option value="1">TV</option>
    <option value="2">LCD</option>
    <option value="3">TV e LCD</option>
</select>
<input type="button" value="Ativar"
onclick="startVideo($('#streamOut').val());"/>
</div>
<? }
else {
?>
<h3> É necessário estar autenticado no sistema para ver esse conteúdo!</h3>
<? } ?>

```

9.3.4 www/pages/contact.php

É o *script* PHP responsável por sintetizar a estrutura HTML e o conteúdo da página Contato. Também faz referência à função JavaScript que obtêm os dados inseridos pelo usuário

e disparam uma requisição ao *script* **php/mail.php**, passando como parâmetros os dados digitados.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<script type="text/javascript">
  //Carrega o diretório principal
</script>

  <hr/>
  <h2>Contato</h2>
  <hr/>
  <h3>Formulário para Contato</h3>
  <hr/>
  <br/>
  <div id="contactDiv">
    <table>
      <tr>
        <td>Nome: </td><td><input type="text" size="40"
id="nomeCtc"/></td>
      </tr>
      <tr>
        <td>E-mail: </td><td><input type="text" size="40"
id="mailCtc"/></td>
      </tr>
      <tr>
        <td>Telefone: </td><td><input type="text" size="12"
id="telCtc"/></td>
      </tr>
      <tr>
        <td>Texto: </td><td><textarea cols="30" rows="8"
id="message"></textarea></td>
      </tr>
      <tr>
        <td><input type="button" value="Enviar"
onclick="sendEmail($('#nomeCtc').val(),$('#mailCtc').val(),$('#telCtc'
).val(),$('#message').val())"/></td>
      </tr>
    </table>
  </div>
  <br/>
  <hr/>
```

9.3.5 www/pages/docs.php

É o *script* PHP que trata da página *Documentação*. Basicamente, é um código HTML que mostra de maneira organizada informações tais como documentos, bibliografia e *links* específicos tidos como referência no aprendizado e desenvolvimento para Linux Embarcado.

```
<h2>Documentos e Informações Úteis</h2>
<hr/>
```

```

<br/>
<br/>
<div id="press" align="center">
<h3>Apresentação do Seminário Linux Embarcado</h3>
<br/>
<p style="text-align: center; font-size: 20px"><a
href="files/seminario_linux_embarcado.pdf">Download</a></p>
</div>
<div id="links" align="center">
<h3>Links Úteis:</h3>
<br/>
<table id="linksForm">
<tr><td><a href="http://sergioprado.org" ></a></td>
<td><a href="http://free-electrons.com/blog"></a></td>
<td><a href="http://buildroot.uclibc.org/"></a></td>
</tr>
<tr>
<td><a href="http://www.linuxabordo.com.br"></a></td>
<td><a href="http://www.friendlyarm.net/forum"></a></td>
<td><a href="http://www.arm9home.net/"></a></td>
</tr>
<tr>
<td><a href="http://www.cirp.usp.br/arqs/4ciclo/Embarcado.pdf">Outra
apresentação sobre Linux Embarcado</a></td>
</tr>
<tr>
<td><a href="http://www.linuxembarcado.com/">Blog Linux
Embarcado</a></td>
</tr>

```

```

</table>
</div>
<div id="biblio" align="center">
<br/><br/>
<h3>Bibliografia Recomendada:</h3>
<h4>Building Embedded Linux Systems - 2ed, por Karim Yaghmour</h4>
<h4>Essential Linux Device Drivers, Sreekrishnan Venkateswaran</h4>
<h4>Linux Kernel Development - 3ed, Robert Love</h4>
<h4>Understanding the Linux Kernel - 3ed, Daniel P. Bovet, Marco Cesati
Ph.D.</h4>
<h4>Pro Linux Embedded Systems, Gene Sally</h4>
<h4>Embedded Linux Primer: A Practical Real-World Approach - 2ed,
Christopher Hallina</h4>
<h4>Embedded Linux: Hardware, Software, and Interfacing, Craig
Hollabaugh</h4>
<h4>The Linux Programming Interface: A Linux and UNIX System Programming
Handbook, Michael Kerrisk</h4>
<h4><a href="http://lwn.net/Kernel/LDD3/">Linux Device Drivers, 3ed,
Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman</a></h4>
<br/>
</div>

```

9.3.6 [www/pages/gps.php](http://www.pages/gps.php)

Esse é o *script* PHP responsável pela página GPS do Menu de controle da interface *web* do projeto. Basicamente, ele trata de realizar uma requisição AJAX assim que a página é iniciada, para exibir dados GPS da Estação, obtidos do sensor GPS através do *script* **/php/gps.php**. Possui *tags* HTML para controle de conteúdo, e faz uso de botões associados a funções JavaScript para disparo de requisições AJAX ao servidor.

```

<script type="text/javascript">
//Carrega o diretório principal
$(document).ready(function(){
    $.get("php/gps.php?pos",function(gps){
        $('#gpsPos').html(gps);
    });

});

```



```

</script>

<hr/>
<h2>GPS</h2>
<hr/>
<br/>
<h3>Status de Posição GPS</h3>
<table>
    <tr><td><div id="gpsPos"></div></td><td><div
id="gMaps"></div></td></tr>
</table>

<br/>
<hr/>
<br/>
<div id="controlGPS">
    <h3>Controle do GPS</h3>
    <table id="tableCtrlGPS">
        <tr>
            <td>Tipo de Dado GPS:</td>
            <td>
                <select id="gpsType">
                    <option value="1">GPGGA</option>
                    <option value="2">GPGLL</option>
                    <option value="3">GPGSA</option>
                    <option value="4">GPGSV</option>
                    <option value="5">GPRMC</option>
                </select>
            </td>
        </tr>
        <tr>
            <td>Intervalo de Tempo:</td><td><input
type="number" size="4" id="time"/></td>
        </tr>
        <tr>
            <td>
                <input type="button" value="Ativar"

```

```

onclick="startGps(type,time)"/>
        </td>
        <td>
                <input type="button" value="Pausar"
onclick="stopGps()"/>
        </td>
        <td>
                <input type="button" value="Salvar Dados"
onclick="getGpsFile(type,time)"/>
        </td>
    </tr>
</table>
<br/>
</div>
<br/>
Saída dos dados GPS:
<hr/>
<div id="gpsData">

</div>

```

9.3.7 www/pages/home.php

Script da página Inicial do projeto. Basicamente contém uma breve descrição do projeto e dos componentes utilizados, fazendo uso de *tags* HTML para organização de conteúdo.

```

<h1>ARM WEB MEDIA STATION</h1>
<h2>Estação de Controle Multimídia Web</h2>

```

```

<h3>Descrição</h3>

```

```

<p>Esse é um Projeto de Conclusão de Curso, que visa estudar e
desenvolver componentes web e multimídia em cima de uma plataforma ARM,
executando sistema operacional GNU/Linux embarcado. Serão estudados
componentes que permitam o controle da execução de vídeos e músicas através
da web, a exibição desses conteúdos no display LCD e em televisores e caixas
de som, além de componentes como GPS, sensor de temperatura e infravermelho,

```

com a exibição das informações obtidas através da web</p>

```
</a>
```

<p>A idéia por trás desse projeto reside na crescente demanda por equipamentos computacionais que tenham um consumo energético reduzido, o que tem sido muito bem alcançado com a arquitetura de microprocessadores ARM. Além disso, foi possível demonstrar que um equipamento pequeno e razoavelmente barato é capaz de ficar ligado continuamente por dias, dada a robustez do sistema operacional GNU/Linux, além de ser também capaz de operar recursos computacionais mais trabalhosos, como o controle de conteúdo multimídia e o controle de <i>streaming</i> de vídeo pela <i>web</i></p>

```
<br/>
```

<p>Além disso, os exemplos aqui desenvolvidos servirão de base e fundamento para projetos futuros envolvendo dispositivos com Linux Embarcado.</p>

```
<br/>
```

```
<h3>Sobre os Componentes</h3>
```

<p>Essa placa é dotada de componentes tais como: portas USB, entrada/saída de áudio estéreo, display LCD, portas seriais diversas, botões, saída de vídeo RCA e sensor de temperatura ds18b20.</p>

```
<h3>Sobre o Servidor</h3>
```

<p>A aplicação servidora é o Lighttpd, que está vinculado ao PHP através de um socket FastCGI. O PHP cuida de processar as páginas, as requisições do usuário, e dispara ações na máquina servidora.</p>

9.3.8 www/pages/images.php

É o *script* responsável pela página *Imagens*, o qual fornece a estrutura para exibir as imagens tiradas pela câmera USB. Ao ser carregada, essa página faz uma requisição AJAX para mapear o diretório de imagens do cartão de memória com o perfil de imagens. Todos os resultados obtidos das requisições são mostrados na *div* HTML cujo *id* é “*folder*”.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

```

<script type="text/javascript">
//Carrega o diretório principal
$(document).ready(function(){

    $.get("php/foldernav.php?img&chdir=/sdcard/pictures",
function(data){

    $('#folder').html(data);

    });

});

</script>

<hr/>
<h2>Imagens</h2>
<hr/>

<div id="folder">

</div>

```

9.3.9 www/pages/login.php

Esse é o *script* PHP responsável pela página de *login*, a qual faz a autenticação do usuário no sistema, permitindo o acesso a conteúdo restrito.

```

<?php
//session_start();
if($_SESSION['login'] == false)
{
?>
<h1>LOGIN</h1>
<hr/>
<br/>
<br/>
<h3>Autenticação de Usuário</h3>
<hr/>
<br/>
<br/>
<form method="post" action="php/auth.php" id="login">
    ID: <input type="text" size="20" name="user"/>

```

```

        Password: <input type="password" size="20" name="passwd"/>
        <input type="submit" name="submit" value="Submit"/><br/>
</form>
<? } else {
    echo "Usuário já autenticado.";
}

?>

```

9.3.10 www/pages/musics.php

É o *script* responsável pela página *Músicas*. Basicamente, consiste de uma estrutura HTML, realizando uma requisição inicial ao servidor para mapear o diretório **/sdcard/mp3**, de modo a exibir para o usuário os arquivos de áudio e diretórios contidos nesse diretório. As respostas das requisições são mapeadas na *div* de *id* “folder”.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<script type="text/javascript">
//Carrega o diretório principal
$(document).ready(function(){

    $.get("php/foldernav.php?md&chdir=/sdcard/mp3", function(data){
        $('#folder').html(data);
    });

});

</script>

<hr/>
<h2>Músicas</h2>
<hr/>

<div id="folder">

</div>

```

9.3.11 [www/pages/rf.php](#)

É o *script* responsável pela página *RF*, que possui a estrutura HTML que forma os comandos para envio e recebimento de mensagens através do serviço de radiofrequência.

```
<?php
session_start();
?>

<h2>Radio Frequência</h2>
<hr/>

<?php
if($_SESSION['login'] == FALSE){
    echo "<h3>É necessário estar autenticado no sistema para ver
esse conteúdo!</h3>";
}
else {
?>

<h3>Controle de Mensagens RF</h3>
<div id="rfDiv"></div>
<br/>
<hr/>
<br/>
<div id="rfControl">
    <table id="emissorTable">
        <tr><td>Emissor:</td>
            <td>
                <select id="emissorSelect">
                    <option value="1">Máquina 1</option>
                    <option value="2">Máquina 2</option>
                </select>
            </td>
        </tr>
    </table>
    <table id="emissorTableText">
        <tr><td>Texto:</td></tr>
        <tr><td><textarea id="textArea" name="textArea" cols="40"
rows="3" onKeyDown="countChars(96)" onKeyUp="countChars(96)">
            </textarea>
        </td>
    </tr>
    </table>
    </div>
</div>
```

```

        </tr>
        <tr><td><input disabled size="3" value="96" name="txtLen"
id="txtLen"/></td></tr>
        <tr><td><input type="button" value="Enviar Mensagem"
onclick="sendRf($('#emissorSelect').val(),$('#textArea').val())"/></td></tr>
    </table>
</div>
<br/>
<br/>

<table id="receptorTable">
    <tr><td>Receptor: </td>
        <td> <select id="receptor">
            <option value="1">Máquina 1</option>
            <option value="2">Máquina 2</option>
        </select>
        </td>

    </tr>
</table>
<table>

    <tr><td><input type="button" value="Receber Dados"
onclick="receiveRf($('#receptor').val())"/>
    </td>
</tr>
</table>
<hr/>
<h4 style="text-align:center; font-size:18px;">Mensagens
Obtidas</h4>
    <div id="tempData">

    </div>
    <?
    }
    <?>

```

9.3.12 www/pages/temp.php

É o *script* responsável pela página *Temperatura*, que possui os elementos de controle e exibição de dados de temperatura ambiente.

```

<script type="text/javascript">
//Carrega o diretório principal
$(document).ready(function(){

    $.get("php/temp.php?temp=C",function(temp){
        $('#tempDiv').html("<h3 style=\"color:blue;\">"+temp+"
°C</h3>");
    });

});

</script>

<hr/>
<h2>Temperatura</h2>
<hr/>
<h3>Exibição da Temperatura Ambiente</h3>
<div id="tempDiv"></div>
<br/>
<hr/>
<br/>
<div id="tempCtrl">
    Tipo de Temperatura
    <table id="tempControl">
        <tr>
            <td>
                <select id="tempType"
onchange="changeTemp(this.value);">
                    <option value="1">Celsius</option>
                    <option value="2">Fahrenheit</option>
                    <option value="3">Kelvin</option>
                </select>
            </td>
            <td><input type="button" value="Ativar"
onclick="startTemp($('#tempType').val())"/></td>
            <td><input type="button" value="Pausar"
onclick="stopTemp()"/></td>
        </tr>
    </table>

```



```

        <br/>
    </div>
    <div id="tempFileCtrl">
        <h3>Obtenção de Arquivo de Dados de Temperatura:</h3>
        <form id="getTempFileForm" method="get" action="php/temp.php">
            <table id="formTempFile">
                <tr>
                    <td>
                        <select id="tempTypeSelect" name="type">
                            <option value="C">Celsius</option>
                            <option value="F">Fahrenheit</option>
                            <option value="K">Kelvin</option>
                        </select>
                    </td>
                    <td><input type="hidden" name="file"/></td>
                    <td><input type="number" id="tempInterval"
name="time" size="4"/> Segundos.</td>
                </tr>
                <tr>
                    <td><input type="submit" value="Salvar
Dados"/></td>
                </tr>
            </table>
        </form>

        <br/>
        <!--<br/>Intervalo de Tempo:<input type="number" size="4"
id="time"/><br/>-->
    </div>
    <br/>
    <h3>Saída dos dados de Temperatura</h3>
    <hr/>
    <div id="tempData">

</div>

```

9.3.13 `www/pages/videos.php`

Esse é o *script* responsável pela página Vídeos, do Menu de opções da interface *web* do projeto. É um código HTML puro, que basicamente trata de fazer uma requisição ao *script* **php/foldernav.php** para mapear o diretório **/sdcard/vídeos** com o perfil de vídeos, e o conteúdo desse diretório é mostrado na *div* de *id* “folder”.

Esse *script* também possui um elemento HTML que permite a seleção do tipo de saída de vídeo desejada, que pode ser TV, LCD, ou TV e LCD. Essa opção é mapeada por funções JavaScript disparadas quando o usuário clica no vídeo que deseja tocar.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<script type="text/javascript">
//Carrega o diretório principal
$(document).ready(function(){

    $.get("php/foldernav.php?vd&chdir=/sdcard/videos",
function(data){

    $('#folder').html(data);

    });

});

</script>

<hr/>
<h2>Vídeos</h2>
<hr/>
<div id="outputControl">
<h2>Seleção de Saída de Vídeo:</h2>
    <select name="Saída" id="out">
        <option id="tv">TV</option>
        <option id="lcd">LCD</option>
        <option id="tvlcd">TV-LCD</option>
    </select>
</div>

<div id="folder">

</div>
```

9.4 Códigos em C, Shell Script e Python

Nessa parte estão códigos de programas escritos nas linguagens C, Python ou em *Shell Script*.

9.4.1 cameraCapture.sh

É o *script* escrito em *Shell Script* que faz o controle do binário responsável pela captura de imagens da câmera USB. Além disso, ele é capaz de encerrar a execução corrente do *stream* de vídeo, de modo a ter acesso à câmera USB, e dispara o comando de tirar fotos passando como argumento a data no formato Dia-Mês-Ano—Hora-Minuto, seguido da extensão *.jpg. Após tirar a foto, o *script* move o arquivo para uma pasta no cartão de memória, e retorna a execução do *stream* de vídeo.

```
#!/bin/sh

nome=$2
tipo=$1

### Instanciando as variaveis responsaveis pelo binario decaptura
dirUsbCapture='/usbCapture'
capProgram='usbCap'
capLink="$dirUsbCapture/$capProgram $tipo"

### Definindo o diretorio para se salvar as imagens
dirImg='/sdcard/pictures'

### Instanciando o link de relacao com MJPG-Streamer
mjpgDir='/mjpg-streamer'
mjpgFile='start_uvc.sh'
mjpgLink="$mjpgDir/$mjpgFile"

pidMJPGStreamer=$(pidof mjpg_streamer)
if [ -n $pidMJPGStreamer ]
then
    kill $pidMJPGStreamer
fi
```

```

dateTime=$(date +%d-%m-%Y--%H-%M)
fileName="$nome.jpg"
$capLink "$fileName"

mv $fileName $dirImg

$mjpgLink > /dev/null &

echo $fileName

```

9.4.2 gps.py

Esse é o *script* Python responsável por fazer a conexão serial com o receptor GPS, e, com base nos argumentos recebidos, captura somente a sentença \$GPRMC, caso nenhum argumento seja passado, ou captura a sentença escolhida, caso seja passado 1 argumento, ou captura a sentença escolhida em um dado número de amostras, caso 2 argumentos sejam passados.

```

#!/usr/bin/python
# -*- coding: utf-8 -*-

import serial
import time
import sys

ser = serial.Serial("/dev/ttySAC3",
    baudrate=9600,
    bytesize=8,
    parity=serial.PARITY_NONE,
    stopbits=1,
    timeout=1)

param = None
interval = None

def printUsage():
    print "./gps.py param time"

#Funcao para retornar o numero de argumentos passados.

```

```

def getArgs():
    if len(sys.argv)==1:
        return 1
    elif len(sys.argv)==2:
        return 2
    elif len(sys.argv)==3:
        return 3
    elif len(sys.argv)>=4:
        printUsage()

#Funcao para verificar os argumentos passados
def verifyArgs():
    print "Coisa"

#Funcao para, de acordo com o numero de parametros passados, setar as
variaveis principais
def setArgs():
    global param
    global interval

    if getArgs() == 1:
        param = "GPRMC"
    elif getArgs() == 2:
        param = sys.argv[1]
    elif getArgs() == 3:
        param = sys.argv[1]
        interval = int(sys.argv[2])

#Funcao para obter uma unica string de dados NMEA do GPS.
def getSerial(param):
    global ser
    if param == "all":
        #Para evitar de pegar uma string no meio do envio.
        ser.readline()
        #Pega uma string fresca
        gps = ser.readline()
        return gps

    else:
        findAux = -1

```

```

        while cmp(findAux,-1) == 0:
            ser.readline()
            gps = ser.readline()
            findAux = gps.find(param)
        return gps

def getSerialTime(param,interval):
    diff = None

    if param == "all":
        time1 = int(time.time())
        ser.readline()
        while diff <= interval:
            gps = ser.readline()
            time2 = int(time.time())
            diff = time2 - time1
            sys.stdout.write(gps)

    else:
        diff = 0
        findAux = -1
        #time1 = int(time.time())

        ser.readline()

        while diff != interval:
            while findAux == -1:
                gps = ser.readline()
                findAux = gps.find(param)
            diff = diff+1
            #diff = time2 - time1
            sys.stdout.write(gps)

# No caso da chamada padrao...
if getArgs() == 1:
    setArgs()
    sys.stdout.write(getSerial(param))

# No caso de dois argumentos.
elif getArgs() == 2:

```

```

setArgs()
sys.stdout.write(getSerial(param))

# No caso de tr3 argumentos.
elif getArgs() == 3:
    setArgs()
    getSerialTime(param, interval)

```

9.4.3 initCap.sh

Esse foi um *script Shell Script* que precisou ser desenvolvido para instanciar o **Cron** com seus arquivos e diretórios, e mais uma linha de serviço para a tirada de fotos temporizadas.

```

#!/bin/sh

#Inicializando o crond
/usr/sbin/crond

#Criando as pastas que ele precisa
if [ ! -d /var/spool ]; then
mkdir /var/spool
fi

if [ ! -d /var/spool/cron ]; then
mkdir /var/spool/cron
fi

if [ ! -d /var/spool/cron/crontabs ]; then
mkdir /var/spool/cron/crontabs
fi

#Agora que existem as pastas, a pasta principal serah definida
cronDir=/var/spool/cron/crontabs

#Criando o arquivo
touch $cronDir/root

```

```

cronFile=$cronDir/root

#Definindo o script de tirar fotos a cada 30 hora, padrao.
echo "30 * * * * /usbCapture/timeCapture.sh" > $cronFile

```

9.4.4 mail.py

Esse é o *script* Python responsável por enviar *e-mails*, recebendo um assunto e um conteúdo como parâmetros para o *e-mail* a ser enviado. Os destinatários são definidos por padrão, assim como as informações de *login* para envio de e-mail utilizando serviço SMTP.

```

#!/usr/bin/python
import sys
import smtplib
import string

FROM = "tiny6410@gmail.com"

to1 = "andre.ml.curvello@gmail.com"
to2 = "evandro@sc.usp.br"

SUBJECT = sys.argv[1]

username = "tiny6410"
password = "tiny6410armwe"

MESSAGE = sys.argv[2]

print SUBJECT
print MESSAGE

BODY1 = string.join((
    "From: %s" % FROM,
    "To: %s" % to1,
    "Subject: %s" % SUBJECT,
    "",
    MESSAGE), "\r\n")

BODY2 = string.join((

```



```

        "From: %s" % FROM,
        "To: %s" % to2,
        "Subject: %s" % SUBJECT,
        "",
        MESSAGE), "\r\n")

server = smtplib.SMTP('smtp.gmail.com:587')
server.starttls()

server.login(username,password)
server.sendmail(FROM,to1,BODY1)
server.sendmail(FROM,to2,BODY2)
server.quit()

sys.stdout.write("1");

```

9.4.5 timeCapture.sh

Esse foi o *script Shell Script* desenvolvido para a tirada de fotos temporizadas. O uso desse *script* é em conjunto com o **Cron**d, um utilitário para sistemas Linux que controla a execução de tarefas em dados intervalos de tempo. Para isso, é criada uma linha de tarefa no **Cron**d referenciando o *script*, e determinando os intervalos de tempo que se deseja executar. Nesses intervalos, serão então tiradas as fotos do ambiente monitorado pela câmera USB.

```

#!/bin/sh

### Instanciando as variaveis responsaveis pelo binario de captura
dirUsbCapture='/usbCapture'
capProgram='usbCap'
capLink="$dirUsbCapture/$capProgram $tipo"

### Verificando se existe o diretorio de time
if [ ! -d /sdcard/pictures/time ]; then
mkdir /sdcard/pictures/time
fi

### Definindo o diretorio para se salvar as imagens

```

```

dirImg='/sdcard/pictures/time'

### Instanciando o link de relacao com MJPG-Streamer
mjpgDir='/mjpg-streamer'
mjpgFile='start_uvc.sh'
mjpgLink="$mjpgDir/$mjpgFile"

pidMJPEGStreamer=$(pidof mjpg_streamer)
if [ -n $pidMJPEGStreamer ]
then
    kill $pidMJPEGStreamer
fi

nome=$(date +%d-%m-%Y--%H-%M)
fileName="$nome.jpg"
$capLink 1 "$fileName"

mv $fileName $dirImg

$mjpgLink > /dev/null &

echo $fileName

```

9.4.6 usbCapArg.c

Esse é o código em Linguagem C do programa desenvolvido utilizando a biblioteca OpenCV para a tirada de fotos com a câmera USB. Pode receber duas entradas como argumento. A primeira determina se a imagem será colorida, se o valor for “1”, ou preto e branca, se o valor for “2”, e a segunda, é o nome da imagem propriamente. O nome da imagem deve possuir a extensão *.jpg.

```

#include "cv.h"
#include "highgui.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv){

    int i;
    int k = atoi(argv[1]);

```

```

CvCapture* capture = 0;
IplImage* img = 0;
IplImage* img1 = 0;

capture = cvCreateCameraCapture(2);

if(!capture){
return -1;
}

for(i=0; i<10; i++){
img=cvQueryFrame(capture);
    }

//Se for 1, imagem colorida.
if(k == 1){
    cvSaveImage(argv[2],img);
}

//Se for 2, imagem Preto e Branca
else if (k == 2){
IplImage* img1=cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,1);
    cvCvtColor(img,img1,CV_RGB2GRAY);
    cvSaveImage(argv[2],img1);
    cvReleaseImage( &img1 );
}

    cvReleaseImage( &img );
    cvReleaseCapture( &capture );
return 0;
}

```

9.4.7 Ds18b20Test.c modificado

Esse é o arquivo em código C que lê os valores de temperatura obtidos do sensor de temperatura DS18B20. Ele foi modificado do original para não gerar um *loop* infinito de saídas, passando a gerar apenas uma saída quando chamado.

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <linux/ioctl.h>

int main()
{
    int fd, i;
    unsigned char result[2];
    unsigned char integer_value = 0;
    float temperature, decimal_value;
    fd = open("/dev/ds18b20", 0);
    if (fd < 0)
    {
        perror("open device failed\n");
        exit(1);
    }
    for (i=0; i<51;i++)
    {
        read(fd, &result, sizeof(result));
        integer_value = ((result[0] & 0xf0) >> 4) |
        ((result[1] & 0x07) << 4);

        decimal_value = 0.5 * ((result[0] & 0x0f) >> 3) +
        0.25 * ((result[0] & 0x07) >> 2);

        temperature = (float)integer_value + decimal_value;

    }
    printf("%6.2f",temperature);
}

```