UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

DESENVOLVIMENTO DE UM SOFTWARE PARA CONTROLE DE RÁDIO FM NO DISPOSITIVO GCW

MARCOS PAULO DE SOUZA

BLUMENAU
2014

2014/1-15

MARCOS PAULO DE SOUZA

DESENVOLVIMENTO DE UM SOFTWARE PARA CONTROLE DE RÁDIO FM NO DISPOSITIVO GCW

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalha de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Mauro Marcelo Mattos, Dr. – Orientador

BLUMENAU
2014

2014/1-15

DESENVOLVIMENTO DE UM SOFTWARE PARA CONTROLE DE RÁDIO FM NO DISPOSITIVO GCW

Por

MARCOS PAULO DE SOUZA

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Mauro Marcelo Mattos, Dr. – Orientador, FURB
Membro:	Prof. Miguel Wisintainer, Mestre – FURB
Membro:	Prof. Antônio Carlos Tavares, Mestre – FURB

Blumenau, 7 de Julho de 2014

Dedico este trabalho a minha família, meus amigos e todos que acreditaram na realização deste trabalho.

AGRADECIMENTOS

A Deus por estar ao meu lado em todas as noites em que fiquei ocupado neste trabalho.

Aos meus pais Vilmar e Lorena, por terem me apoiado e pela sua paciência nos dias em que fiquei trabalhando sem parar para sua finalização.

Agradeço também a minha namorada Fabiana pela sua paciência e pela sua compreensão nos finais de semana em que fiquei programando a aplicação do rádio.

Ao meu orientador por ter acreditado no potencial desta ideia.

A genialidade é 1% de inspiração e 99% de transpiração.

Thomas Edison

RESUMO

Este trabalho apresenta o desenvolvimento de um software para controle de um *chip* de rádio FM no dispositivo *Game Consoles Worldwide* (GCW). O dispositivo é controlado por uma distribuição Linux. O software foi desenvolvido utilizando a linguagem C em ambiente Linux, utilizando o *framework Simple DirectMedia Layer* (SDL) para construir a interface gráfica. Para controlar o rádio foi utilizado o *Advanced Linux Sound Architecture* (ALSA) e sua *Application Programming Interface* (API). A linguagem de programação utilizada foi a linguagem C.

Palavras-chave: Sistemas operacionais. Linguagem C. Linux. SDL. ALSA.

ABSTRACT

This work presents the development of a software to control a FM radio*microchip* in *Game Consoles Worldwide* (GCW) device. This device is controlled by a Linux operational system. The software was developed using the C Language in Linux environment, using the *Simple DirectMedia Layer* (SDL) to build the user interface. To control the radio was used the *Advanced Linux Sound Architecture* (ALSA) e and the *Application Programming Interface* (API). The programming language used was the C programming language.

Key words: Computer science. C language. Linux. SDL. ALSA.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

ALSA – *Advanced Linux Sound Architecture*

API – *Application Programming Interface*

GCW – *Games Consoles Worldwide*

MIPS – *Microprocessor without Interlocked Pipeline Stages*

SDL – *Simple DirectMedia Layer*

USB - *Universal Serial Bus*

SUMÁRIO

1 INTRODUÇÃO

O sistema operacional Linux está em todos os lugares, desde supercomputadores, celulares, estações de trabalho e até em dispositivos embarcados (COBBAUT, 2013). Conforme Sloss, Symes e Wright (2004, p. 6) sistemas embarcados podem controlar os mais diferentes dispositivos, desde pequenos sensores encontrados em linhas de produção a sistemas de controle de tempo real utilizados nas sondas espaciais da National Aeronautics and Space Administration (NASA).

O Linux é um sistema operacional compatível com o Unix que teve início em 1984. O objetivo original foi construir um sistema operacional livre similar ao Unix envolvendo desde as ferramentas básicas até avançadas de modo a compor um sistema operacional completo e funcional totalmente livre (SYSTEM, 2013).

Conforme Waltrik (2011, p.15), Linux é a designação de uma família de sistemas operacionais *unix-like* que compartilham o mesmo *kernel*, sendo capaz de ser executado em diferentes plataformas de processadores como *Itanium*, *Power Optimization With Enhanced RISC Performance Computing* (PowerPC), x86, *Scalable Processor ARChitecture* (SPARC) e *Microprocessor without Interlocked Pipeline Stages* (MIPS).O Linux, quando executado em um sistema embarcado, é denominado Linux embarcado.

Este está disponível em distribuições que facilitam a sua instalação e manutenção. Dentre as mais famosas estão: Debian, Gentoo, Fedora, openSUSE e Ubuntu (ZANONI, p.36). Em função disto, é muito comum ver o Linux sendo embarcado em televisores, computadores de bordo em automóveis, câmeras fotográficas, aparelhos de *DVD* e vários outros aparelhos eletrônicos.

Uma das áreas de aplicação do Linux é a área de *games* e foi neste contexto que a empresa *Games Consoles Worldwide* (GCW) desenvolveu uma plataforma para *retro games* inicialmente voltada para o dispositivo Dingoo A320 chamada de *OpenDingux*. Esta foi criada para ser um substituto do *firmware* original que vinha no dispositivo.

Esta distribuição possui componentes de software para controlar praticamente todos os periféricos do dispositivo, exceto para controlar o módulo de *hardware* de rádio FM. Neste contexto, este trabalho se propôs a criar um software para controle do módulo de rádio FM baseado no controlador RDA5807 e com isso tornar possível para o usuário do dispositivo ouvir estações de rádio.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é criar um aplicativo capaz de controlar o *microchip* de rádio FM RDA5807 utilizando o *toolchain* desenvolvido pela equipe do GCW e permitir ao usuário poder ouvir estações de rádio no dispositivo GCW.

Os objetivos específicos do trabalho são:

a) disponibilizar uma aplicação para permitir ao usuário operar o rádio FM do dispositivo;
b) validar o uso da ferramenta através de um estudo de caso envolvendo operações de uso no dispositivo real.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho está estruturado em quatro capítulos. O capítulo dois tem como conteúdo a fundamentação teórica que foi necessária para a realização deste trabalho. Este está divido nos seguintes temas: Linux, GCW, SDL, ALSA e *toolchain*.

No capítulo três são apresentados os requisitos, diagrama de estados, implementação e os resultados finais da aplicação.

No capítulo quatro são apresentadas as conclusões e sugestões para melhorias na aplicação.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão abordados conceitos e técnicas que foram necessários para a realização do trabalho. Os assuntos abordados serão: sistema operacional Linux e seus *driver* de dispositivos, dispositivo *GCW*, biblioteca *ALSA*, biblioteca *SDL* e trabalhos correlatos.

2.1 SISTEMA OPERACIONAL LINUX

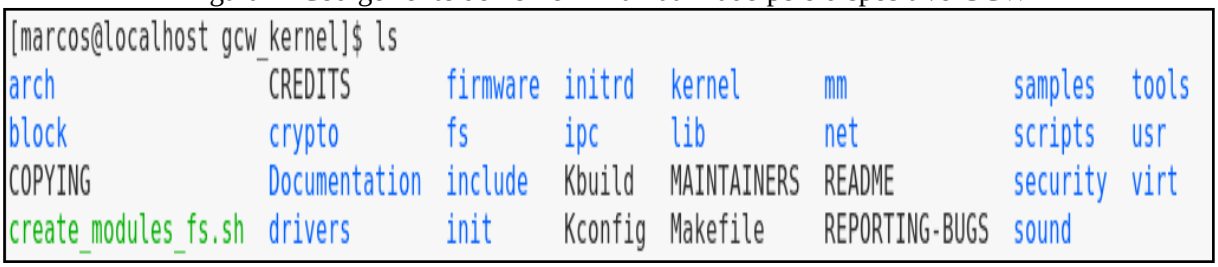
Segundo Campos(2006), o Linux é um sistema operacional criado por Linus Torvalds no ano 1991. Este sistema utiliza a licença de software chamada *GnuPublicLicense* (GPL). Esta licença tem como objetivo de manter o código do sistema aberto, ou seja, possível de ser alterado e ser redistribuído por qualquer um. Por ser aberto e ter livre acesso, muitas empresas ajudam a manter o Linux. Algumas destas são Intel, IBM, HP, Hitachi, entre várias outras.

Como o Linux é um sistema aberto, é comum encontrar-se o Linux sendo usado em várias plataformas, inclusive em plataformas embarcadas como*smartphones*. Além destes, é também muito comum ver o Linux sendo embarcado em televisores, computadores de bordo em automóveis, câmeras fotográficas, aparelhos de *DVD* e vários outros aparelhos eletrônicos (JEONG, 2012).

O sistema operacional Linux é um sistema operacional monolítico, e possui gerenciamento de memória, suporte a vários sistemas de arquivos e suporte a uma gama enorme de dispositivos de *hardware*. O código fonte do *kernel* Linux segue uma estrutura de pasta definida pelo seu conteúdo (RUSLING, 1999). A maior parte do código fonte se concentra no diretório *drivers*, o qual contém o código fonte de todos os *drivers* de dispositivos de todo o sistema (desde *drivers* para dispositivos de rádio, vídeo, teclado, controles de vídeo *games*, e todo o tipo de dispositivos com suporte ao Linux). A Figura 1 mostra uma imagem de como está organizada a estrutura de pastas do *kernel* Linux utilizado pelo GCW.

O Linux utiliza vários sistemas de arquivos para que seja possível interagir com o sistema operacional como, por exemplo, o *sysfs* e *devfs*. Estes auxiliam no controle e gerenciamento do sistema operacional, e proporcionam várias maneiras de interagir com eles. Segundo GOOCH (2001), o sistema de arquivos *devfs* é um poderoso mecanismo para gerenciamento de dispositivos para o Linux. Seus recursos auxiliam tanto desenvolvedores como administradores de sistema. Seu esquema de nomenclatura auxilia administradores de sistema em grandes sistemas, provendo *namespaces* baseados em topologia, e também auxiliam sistemas menores, provendo um *namespace* baseado na classe do dispositivo.

Figura 1- Código fonte do kernel Linux utilizado pelo dispositivo GCW



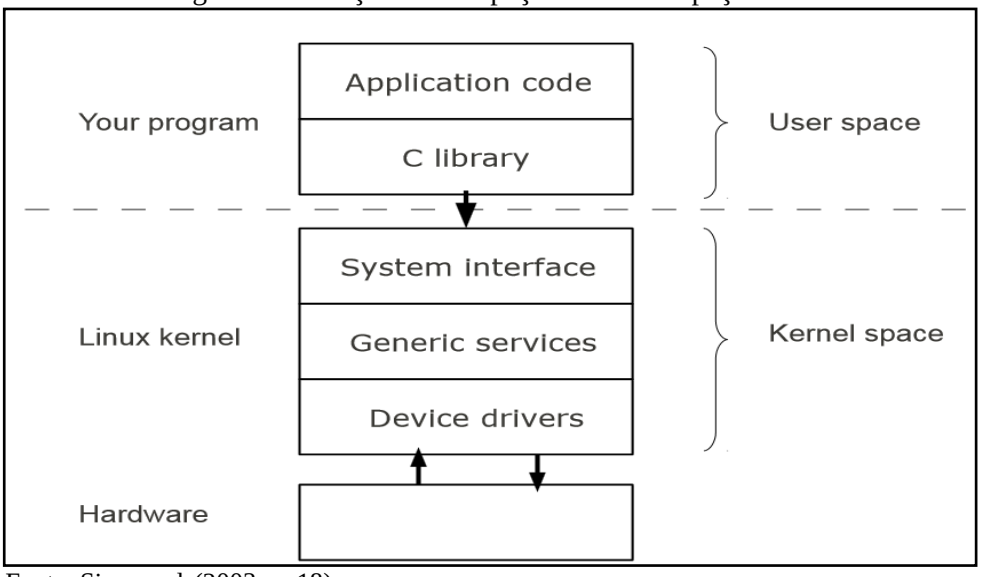
Quanto a parte de desenvolvimento, o *devfs* auxilia na escrita de novos *drivers* de dispositivo, já que não é necessário a alocação de números *oficiais* para os dispositivos. Os números de dispositivo são alocados dinamicamente, conforme os mesmos vão ser adicionados pelo sistema operacional. Tal característica auxilia muito no desenvolvimento do *kernel*, uma vez que as únicas preocupações dos desenvolvedores se focam no desenvolvimento do *driver* de dispositivo. O sistema de arquivos *devfs* reside no caminho absoluto */dev* de qualquer distribuição Linux. Segundo PRADO (2010), o diretório *dev* é o local onde todos os dispositivos plugados no sistema podem ser acessados. Dentro deste diretório pode-se ver arquivos relacionados a dispositivos de disco rígido, teclados, mouses, entre outros.

Existe ainda um outro sistema de arquivos utilizado em projetos com Linux embarcado chamado *squashfs*. Segundo o SQUASHFS (2008), o sistema de arquivos *squasfs* é um sistema de arquivos *somente leitura*, e além disto pode criar arquivos como sendo sistemas de arquivo *somente leitura*. Graças a ferramenta *mksquashfs* é possível criar sistemas de arquivo e arquivos *somente leitura*, e com a chamada *unsquashfs* é possível extrair estes arquivos destas estruturas.

No sistema Linux existe o conceito de programas que executam em espaço de usuário e programas que executam no espaço do kernel Linux. Um processo pode estar em espaço usuário ou espaço de kernel. Segundo BEN-YÉUDA (2004), dependendo de quais privilégios e endereçamento de memória pode-se dizer se um processo está no espaço usuário ou no espaço de kernel. Quando executando em espaço de usuário, um processo tem poucos privilégios e não pode fazer certas coisas. Quando executando no espaço do kernel, um processo tem todas as permissões possíveis e pode fazer qualquer coisa (BEN-YÉUDA, 2004). Quando um processo precisa de permissões mais elevadas para executar alguma tarefa específica no espaço de kernel, é necessário utilizar chamadas de sistema. Um exemplo de uma chamada de sistema *malloc*, que faz alocação de memória. Esta chamada de sistema existe na glibc, uma biblioteca de funções para a linguagem C no ambiente Linux. Ao fazer esta chamada de sistema, o processo que a chama tem o contexto trocado, ou seja, ele executa

esta função no espaço do kernel. Após finalizar a chamada, ele retorna ao espaço do usuário. A Figura 2 mostra um diagrama apresentando como o espaço usuário. Pode executar ações privilegiadas no espaço do kernel.

Figura 2- Interação entre espaço usuário e espaço do kernel



Fonte: Simmonds(2003, p. 18).

2.2 DISPOSITIVO *GCW*

O dispositivo *GCW* foi criado por pessoas que gostavam de jogar *retro games*. O criador do dispositivo tinha o interesse de criar um dispositivo que pudesse servir como uma plataforma para jogos independentes e software de código aberto (KICKSTARTER, 2014).

O dispositivo *GCW* foi inspirado pelo dispositivo *Dingoo A320*, o qual tinha como padrão um *firmware* com emuladores e outras aplicações que dava acesso aos usuários para utilizar alguns dos periféricos, como o *microchip* de rádio FM e microfone. A arquitetura do dispositivo *A320* é *MIPS*. Como o *firmware* matinha os usuários presos às atualizações do fabricante, foi criada a distribuição Linux OpenDingux para ser usada no dispositivo *Dingoo A320*. Um *dual-boot* foi criado para ser possível instalar o OpenDingux em conjunto com o *firmware* original no dispositivo *A320*. Esta distribuição tinha suporte a uma série de emuladores de código aberto para *consoles* da Nintendo. Alguns jogos de computador também foram portados para serem executados dentro do *OpenDingux*. A Figura 3 apresenta uma imagem do dispositivo *Dingoo A320* e o Quadro 1 mostra as especificações do aparelho.

Figura 3- Imagem do console Dingoo A320

Fonte: RADON (2012).

Com o crescimento de usuários do OpenDingux crescia, e então surgiu a ideia da criação de um novo dispositivo para *retro games*, com um *hardware* melhor, e sendo controlado pelo OpenDingux, surgindo assim o *GCW*. A Figura 4 mostra o dispositivo *GCW* que foi concebido utilizando o *Dingoo A320* como referência.

Quadro 1 - Especificação técnica do dispositivo Dingoo A320

CPU:	Ingenics JZ4732 System-On-A-Chip MIPS processor @ 336Mhz (stable overclocks up to 470Mhz have been reported)
RAM:	2x Elpida UPD45128163G5-A75-9JF 128Mbit 133Mhz (total 256Mbit = 32MB of RAM)
Internal Memory:	1x Samsung K9LBG08U0M PCB0 (4GB NAND)
Expansion Slot:	MiniSD Card, officially limited to 8GB (compatible with MicroSDHC via adapter, documented cases of 16GB support)
Display:	2.8" LCD, 320x240, 16M colors, ILI9325 or ILI9331_3
Battery:	3.7V 1700mAh Li-Ion, approximately 7 hours
Audio Output:	Stereo speakers, 3.5mm Headphone jack
Input:	D-Pad, 4 buttons, 2 shoulder buttons, internal microphone
Dimensions:	4.92in x 2.17in x 0.59in (125mm x 55.5mm x 14mm)
Weight:	11.53 oz. (110 g)

Fonte: Adaptado de Dingoonity (2012).

Figura 4 - Imagem do console de retro games GCW

Fonte: Vaughan (2013).

Com o nascimento do GCW, todos os esforços da comunidade foram direcionados para a melhoria da distribuição *OpenDingux*. Este novo dispositivo tem um *hardware* mais sofisticado, conforme pode ser verificado pela. Com *hardware* de maior capacidade, mais recursos podem ser explorados no dispositivo.

Quadro 2 - Especificações técnicas do console GCW

Specifications

CPU:	Ingenic JZ4770 1 GHz MIPS processor
GPU:	Vivante GC860, capable of OpenGL ES 2.0
Display:	3.5 inch LCD with 320x240 pixels; 4:3 aspect ratio is ideal for retro gaming
Operating system:	Linux 3.x (OpenDingux)
Memory:	512 MB DDR2
Internal storage:	16 GB, most of which is available for applications and data
External storage:	micro SDHC up to 32 GB or micro SDXC of 64 GB (SDXC cards must be reformatted before use)
Connectivity:	Mini USB 2.0 OTG Mini HDMI 1.3 out 3.5 mm (mini jack) A/V port for earphone and analog TV-out
Audio:	Stereo speakers, mono microphone
Other:	Accelerometer (g-sensor) and vibration motors
Wireless:	Wi-Fi 802.11 b/g/n 2.4 GHz, can connect to access point or direct device-to-device
Dimensions:	143 * 70 * 18 mm
Weight:	8 oz / 225 g
Battery:	2800 mAh

Fonte: Adaptado de GCW Specs (2013).

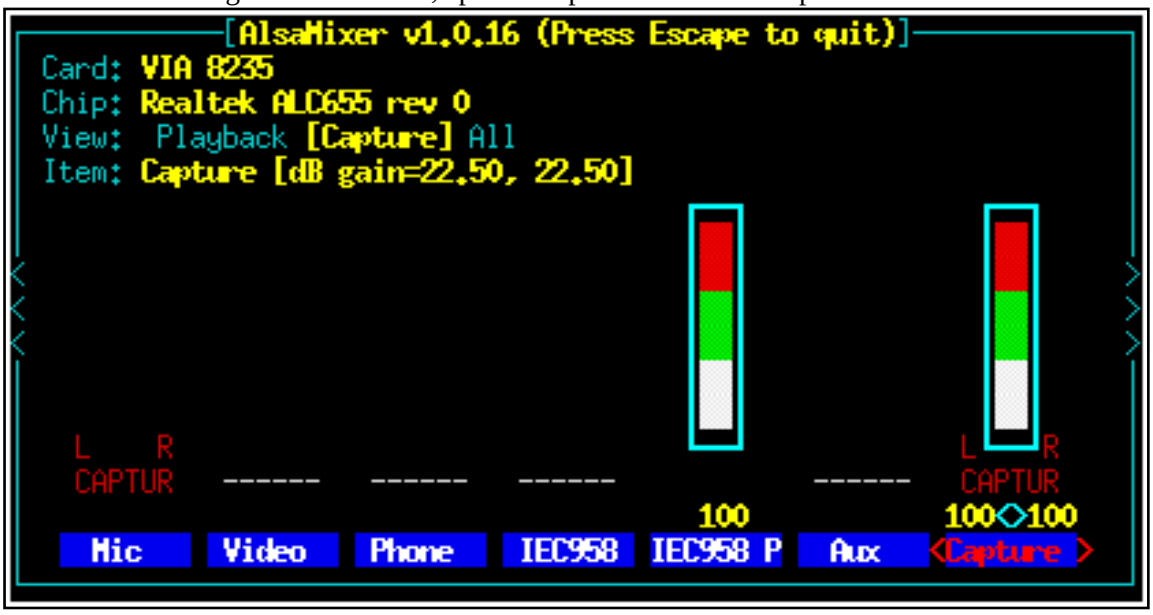
2.3 BIBLIOTECA ALSA

É o *framework* padrão para criação de *drivers* de dispositivos de áudio para controle de placas de som utilizando o sistema operacional Linux. Utilizando o *ALSA* também é possível alterar parâmetros de dispositivos de áudio, como volume, tipo de saída do dispositivo e todas as características que o *hardware* de áudio contiver e que o *driver* de dispositivo desse dispositivo esteja programado. Antes do *ALSA* existia outro framework para criação de tais dispositivos. Esse era chamado *Open Sound System (*OSS), mas ficou depreciado com a criação do *ALSA* (DIMITROV e SERAFIM, 2012).

Na Figura 5 é possível verificar a utilização do *ALSA* no aplicativo *alsamixer*. Este aplicativo em modo *console* é utilizado para efetuar configurações de áudio, como alterar

volume de saída, alterar configurações de saída de áudio e verificar todas as propriedades da placa de áudio do sistema. Este aplicativo épadrão em basicamente todas as distribuições Linux onde existe controle de áudio.

Figura 5- alsamixer, aplicativo para controle de dispositivos de áudio



Fonte: Vaughan (2013).

2.4 BIBLIOTECA SDL

A SDL é uma biblioteca para criação de jogos e multimídia. Utiliza a licença *Gnu Public License* (GPL) e por isso pode ser usada para desenvolver jogos e aplicativos livremente.

É escrita na linguagem C e funciona nativamente com a linguagem C++. Existem adaptações para utilizar SDL em outras linguagens de programação como por exemplo C# e *Python* (SDL WIKI, 2013). A *SDL* suporte oficial para as plataformas Linux, Mac OSX, Windows e iOS e Android. Também existe suporte não oficial para as plataformas *BSD e BeOS (SDL, 2001).

Esta biblioteca torna mais fácil a programação para interação com usuário, como criação de controles, controle de áudio, renderizador para *OpenGL*, temporizadores, múltiplas trilhas de execução, controles de mouse e teclado e interação com *Direct3D* (SDL WIKI, 2013).

A SDL possui uma extensão que é capaz de manipular arquivos True Type Font (TTF) denominada SDL_ttf. Este recurso é voltado para a criação de interfaces com o usuário, pois permite a utilização de vários tipos de fontes de caracteres nas interfaces (ATKINS, 2009). Sua utilização se estende a criação de jogos. Como exemplo disso, a SDL é usada em alguns jogos da produtora de jogos *Valve* (VALVE, 2013) e nos jogos do pacote *Humble Bundle*

(HUMBLE BUNDLE, 2013). A Figura 6 mostra uma imagem do jogo Trine 2 que utiliza a biblioteca SDL.

Figura 6- Figura do jogo Trine 2 que utiliza a biblioteca SDL

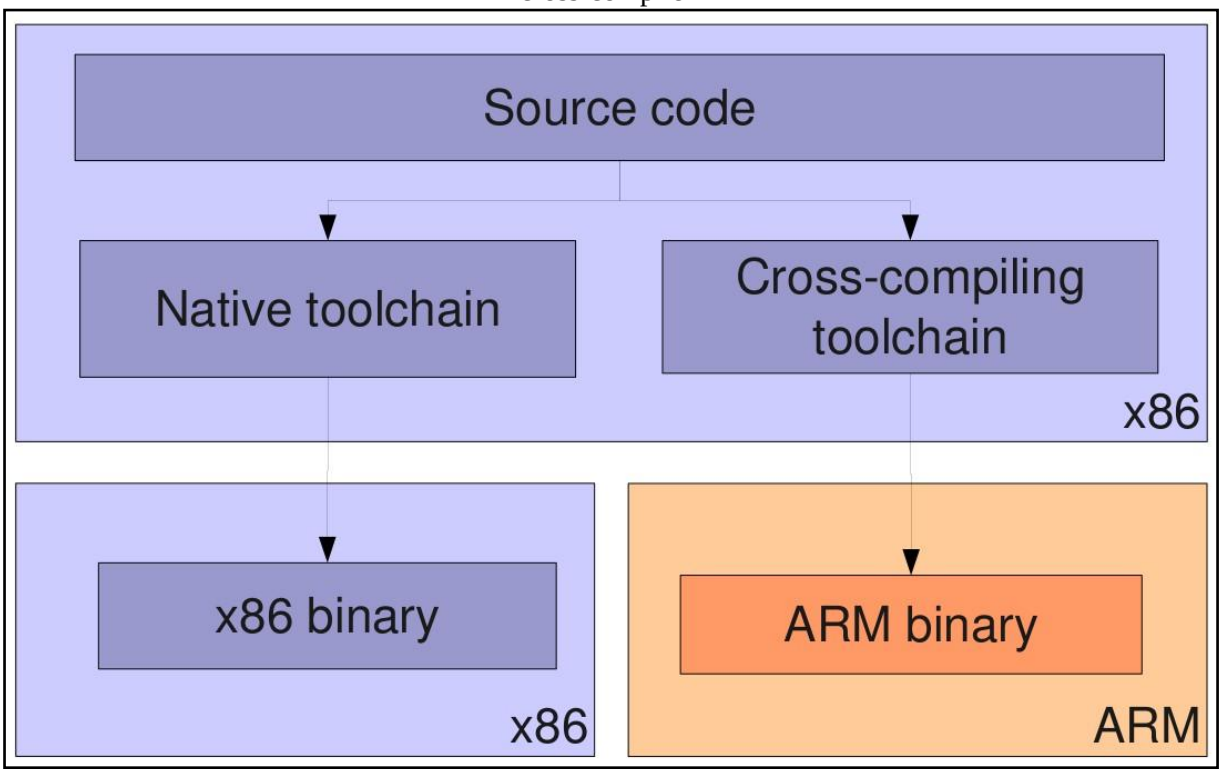
Fonte: TRINE 2(2013).

2.5 TOOLCHAIN

Um *toolchain* é um conjunto de ferramentas para compilação de um programa para uma plataforma. Um *toolchain* contém um pré-processador, um compilador, um montador e um ligador (GCW TOOLCHAIN, 2013). Segundo PRADO (2011), existem dois tipos de *toolchains*, o nativo e o *cross*. O nativo tem como função gerar um executável para a mesma plataforma em que o compilador está sendo executado. O *toolchain cross* tem como objetivo gerar um código executável para uma plataforma diferente da máquina que está compilando o programa.

Um caso típico de uso de um *toolchain cross* pode ser exemplificado no desenvolvimento de aplicações para dispositivos móveis. Dispositivos móveis, em sua maioria, executam em uma arquitetura diferente dos computadores pessoais. Então um desenvolvedor de aplicativos para dispositivos móveisnecessita que um *cross-compiler*, que está incluído no *toolchain*, para poder gerar programas executáveis para a plataforma do dispositivo móvel. A Figura 7 mostra os tipos de *toolchain* e suas plataformas alvo.

Figura 7- Exemplo de estrutura de toolchain gerando para código nativo ou para outra plataforma com cross-compiler



Fonte: PRADO (2011).

Para WALTRIK (2011), a maneira mais simples de embarcar Linux em um dispositivo é fazer *download* de uma imagem do *kernel* Linux, ter um *cross-compiler*, uma biblioteca de funções e alguns aplicativos. Ainda segundo WALTRIK (2011), este procedimento leva muito tempo para ser completado, e com isto se tem necessidade de um *toolchain*para automatizar algumas destas tarefas. Cada *toolchain* tem um estilo próprio, recursos e ferramentas para que o desenvolvedor possa fazer uso e gerar projetos baseados no *toolchain* (WALTRIK, 2011).

2.6 TRABALHOS CORRELATOS

São encontrados alguns trabalhos acadêmicos e projetos independentes que fazem interações com *drivers* de dispositivos ou *framework s*em ambiente Linux. Dois destes são o Minivosc (DIMITROV e SERAFIM, 2012), Toolkit para Linux embarcado (WALTRIK, 2011).

2.6.1 Minivosc

O Minivosc foi um projeto criado para exemplificar a criação de um *driver* de dispositivo de áudiovirtualpara o sistema operacional Linux. Como este *device driver* é virtual, este não requer nenhuma *hardware*deáudio para poder fazer testes sobre ele.

As capacidades do Minivosc se restringem a saída de áudio em formato mono (DIMITYROV e SERAFIM, 2012).

2.6.2 Toolkit para Linux embarcado

O trabalho de Waltriktem como meta a criaçãode um *toolkit* para Linux embarcado para o dispositivo Mini2440. Seu principal objetivo foi acriaçãode uma imagem Linux customizada e otimizada para a placa Mini2440 e ainda criar mini aplicativos para explorar o *hardware* do dispositivo. Estes mini aplicativos servem como modelo para os utilizadores do seu *toolkit* (WALTRIK, 2008).

O trabalho resultou nos seguintes recursos:

a) uma imagem do kernel Linux customizada;
b) uma imagem do sistema de arquivos raiz contendo a distribuição Linux Emdebian combinada com *scripts* e bibliotecas que permitem a inicialização do Lançador de Aplicativos;
c) mini-aplicativos que exploram os principais recursos de *hardware* do Mini2440.

3 DESENVOLVIMENTO

Neste capítulo, são detalhadas a especificação e implementação do aplicativo, através de diagramas e trechos do código fonte implementado.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O trabalho é composto por código na linguagem C em espaço de usuário e alteração de *driver* de dispositivo no espaço do *kernel*. Abaixo estão listados os requisitos funcionais (RF) e os requisitos não funcionais (RFN).

a) permitir ouvir estações de rádio no dispositivo GCW (RF);
b) permitir cadastrar as estações de rádio favoritas (RF);
c) permitir ouvir as estações de rádio utilizando fones de ouvido e alto-falantes (RF);
d) salvar e restaurar as opções do usuário quando o aplicativo for reiniciado (RNF);
e) utilizar a biblioteca SDL para controle da interface com o usuário (RNF);
f) utilizar a linguagem C para desenvolver o aplicativo (RNF);
g) utilizar o *frameworkALSA* para interagir com o áudio do dispositivo (RNF).

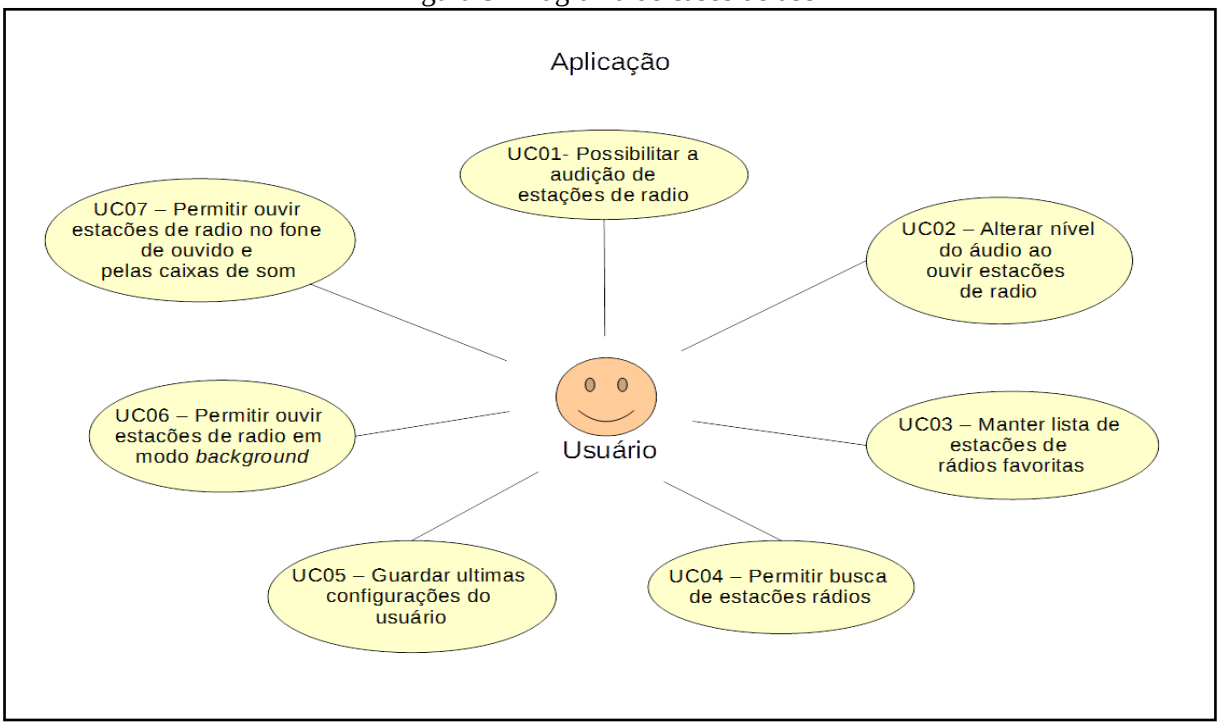
3.2 ESPECIFICAÇÃO

O presente trabalho se divide entre o estudo do *microchip* RDA5807 e o desenvolvimento do software para controle do *microchip*. Por este motivo a especificação se divide entre o detalhamento do *microchip* RDA5807 e o software que faz controle do *microchip.*

3.2.1 Diagrama de casos de uso

No diagrama a seguir, ilustrado pela Figura 8, mostra os casos de uso do sistema. O diagrama foi construído utilizando a ferramenta Libre Office Draw.

Figura 8- Diagrama de casos de uso



Os casos de uso são detalhados nos quadros 3 até 9.

Quadro 3- Caso de uso: possibilitar a audição de estações de rádio

UC01 – Possibilitar a audição de estações de rádio	
Pré Condições	
Cenário Principal	01) O aplicativo é iniciado. 02) O chip de rádio RDA5807 é ligado e sintonizado 03) Uma estação de rádio é configurada no chip de rádio RDA5807
Exceção 01	No passo 03, caso o sinal da estação de rádio esteja fraco o áudio não será ouvido pelo usuário.
Pós-Condição	Usuário capaz de ouvir a estação de rádio desejada.

Quadro 4- Caso de uso: possibilitar alteração do nível de áudio ao ouvir estações de rádio

UC02 – Alterar o nível do áudio ao ouvir estações de rádio	
Pré Condições	Ter uma estação de rádio sintonizada
Cenário Principal	01) O usuário tenta aumentar ou diminuir o volume
Exceção 01	No passo 1, se por acaso o nível do volume for maior que o volume máximo, ou menos que o volume mínimo, o mesmo não é alterado
Pós-Condição	O nível do áudio é alterado conforme solicitado pelo usuário.

Quadro 5- Caso de uso: possibilitar manter uma lista de rádios favorita

UC03 – Manter lista de rádios favoritas	
Pré Condições	Estação de rádio selecionada.
Cenário Principal	01) O usuário escolhe uma estação de rádio de sua escolha. 02) Ao pressionar o botão X a estação de rádio que o usuário está ouvindo é adicionada em uma das 5 rádios favoritas. 03) Ao pressionar o botão A, a rádio favorita selecionada pelo usuário é removida da lista de rádios favoritas.
Exceção 01	No passo 2, se por acaso a rádio que o usuário deseja adicionar já tenha sido adicionada anteriormente e na mesma rádio favorita, então o sistema não adiciona novamente.
Exceção 02	No passo 3, se por acaso a posição atual da rádio favorita que o usuário deseja remover já está vazia, o programa não faz nenhuma alteração.
Pós-Condição	A rádio favorita é adiciona ou removida conforme solicitação do usuário.

Quadro 6- Caso de uso: possibilitar busca por estações de rádio

UC04 – Permitir busca de estações de rádio	
Pré Condições	Não estar executando uma busca de rádio
Cenário Principal	01) Ousuário pressiona os botões L ou R, estes posicionados na parte superior do dispositivo 02) Uma busca de rádios é efetuada para o usuário conforme o botão pressionado. Se o botão pressionado foi o L, a próxima rádio selecionada será de uma frequência de rádio menor do que a rádio atual. Se o botão pressionado tiver sido o botão R, uma rádio de maior frequênciaseráencontrada.
Exceção 01	No passo 2, se por acaso não for encontrada nenhuma rádio com uma frequência menor do que a frequência atual, será efetuada a tentativa de uma de maior frequência que a rádio atual.
Exceção 02	No passo 2, se por acaso não for encontrada nenhuma rádio com uma frequência maior que a frequência atual, será efetuada a tentativa de uma de menorfrequência que a rádio atual.
Exceção 03	No passo 2, se por acaso nenhuma estação de rádio for encontrada,será sintonizada a rádio em que o usuário estava ouvindo antes de fazer a busca.
Pós-Condição	A rádio favorita é adiciona ou removida conforme solicitação do usuário.

Quadro 7- Caso de uso: possibilitar guardar as configurações do usuário

UC05 – Guardar ultimas configurações do usuário	
Pré Condições	Não estar executando uma busca de estação de rádio
Cenário Principal	01) Executar uma alteração de volume, estação de rádio,rádio favorita ou troca de modo de ouvir.
Pós-Condição	Aalteraçãoserá guardada em arquivos de configuração para que a próxima vez que o usuário fechar e reabrir a alteração sua última configuração seja carregada novamente.

Quadro 8- Caso de uso: possibilitar ouvir estações de rádio no modo background

UC06 – Permitir ouvir estações de rádio em modo *background*	
Pré Condições	Não estar executando uma busca de estação de rádio
Cenário Principal	01) Usuário pressiona o botão B no dispositivo
Pós-Condição	A aplicação ira se fechar, mas o áudio da estação de rádio que o usuário estava ouvindo ainda poderá ser ouvida.

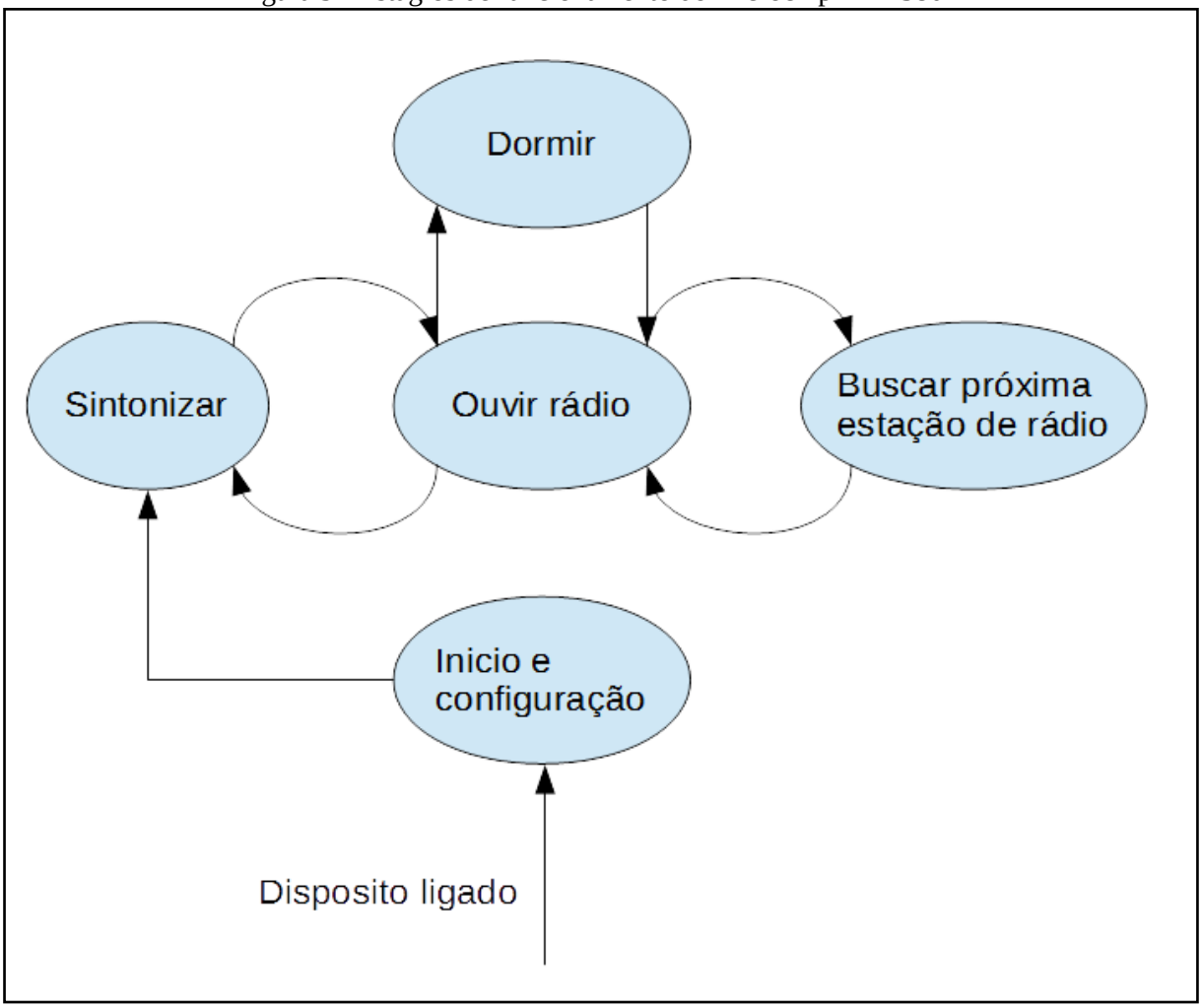
Quadro 9- Caso de uso: possibilitar ouvir estações de rádio pelos fones de ouvido e pelas caixas de som

UC07 – Permitir ouvir estações de rádio pelos fones de ouvido e pelas caixas de som	
Pré Condições	Não estar executando uma busca de estação de rádio
Cenário Principal	01) Usuário pressiona o botão Y do dispositivo
Pós-Condição	Se por acaso o usuário tiver ouvindo estações de rádio pelo fone de ouvido, então a saída de áudio será direcionada para as caixas de som. Se o usuário estiver ouvindo estações de rádio utilizando as caixas de som, então a saída de som será direcionada para os fones de ouvido.

3.2.2 Especificação do microchip RDA5807

O *microchip* RDA5807 tem a função de fazer a recepção de sinais de *rádio frequência* (FM) para o dispositivo. Este *hardware* trabalha no contexto de 5 estados de funcionamento bem definidos, como apresentado na Figura 9.

Figura 9 - Estágios de funcionamento do microchip RDA5807



Fonte: adaptado de RDA5807 Programmer Manual (2008)

Para controle destes estados existem seis registradores de *hardware*no*microchip*. Alterando-se alguns desses registradores faz com que o *hardware* troque de estado. Estes registradores são acessados e configurados pelo *driver* de dispositivo do *microchip*. Este será melhor detalhado na seção a seguir.

3.2.3 Especificação do software de controle do microchip RDA5807

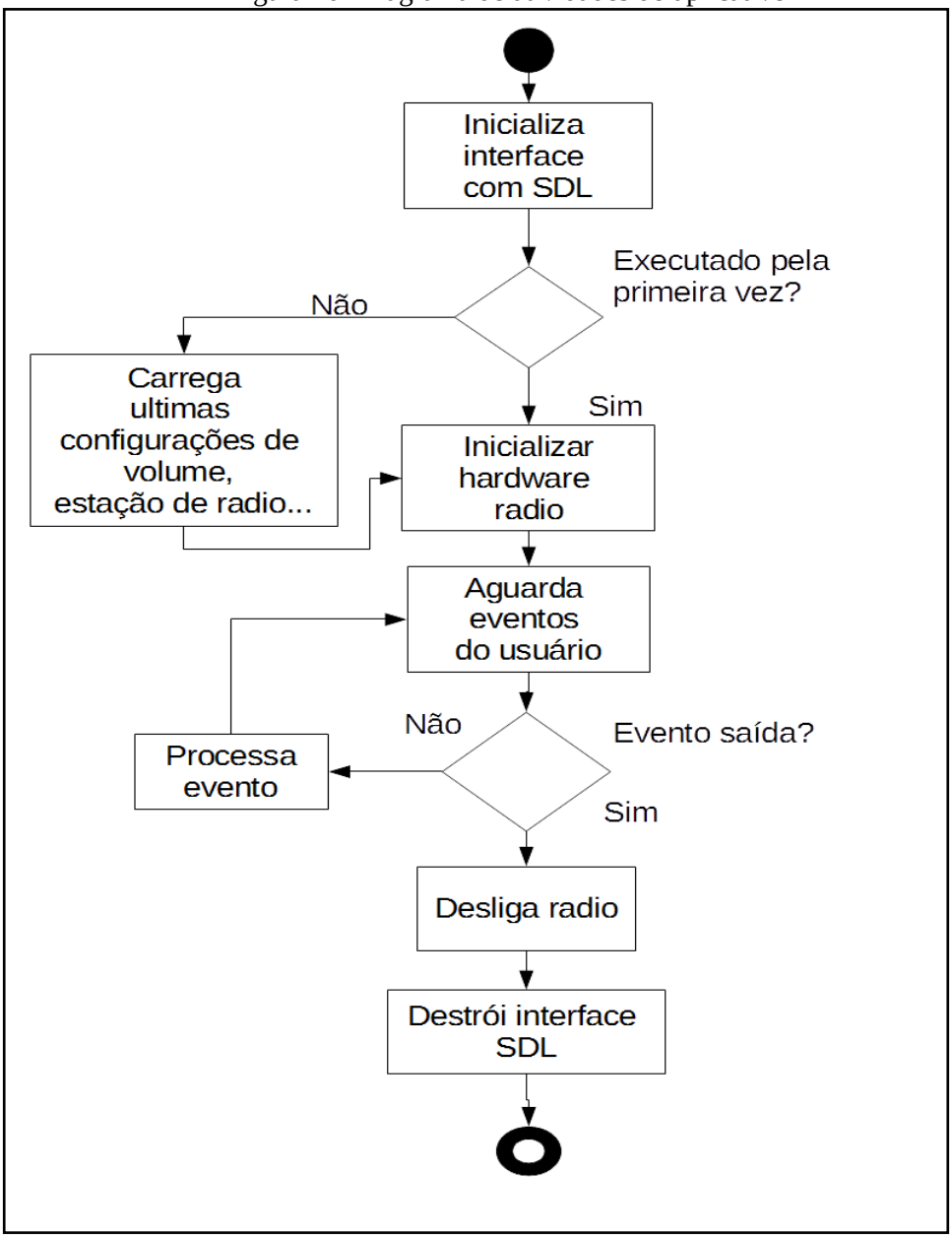
Esta seção se divide em duas partes, sendo a primeira delas descrevendo como funciona o *driver* de dispositivo para o *microchip* RDA5807 e a segunda sendo a implementação do programa aplicativo que interage com o *driver* de dispositivo.

Este trabalho utilizou um *driver* de dispositivo já existente para fazer a recepção e controle dos sinais *FM.*

As funções deste aplicativo foram especificadas utilizando diagramas da UML. Os diagramas de estados e atividades foram desenvolvidos utilizando a ferramenta Libre Office Draw.

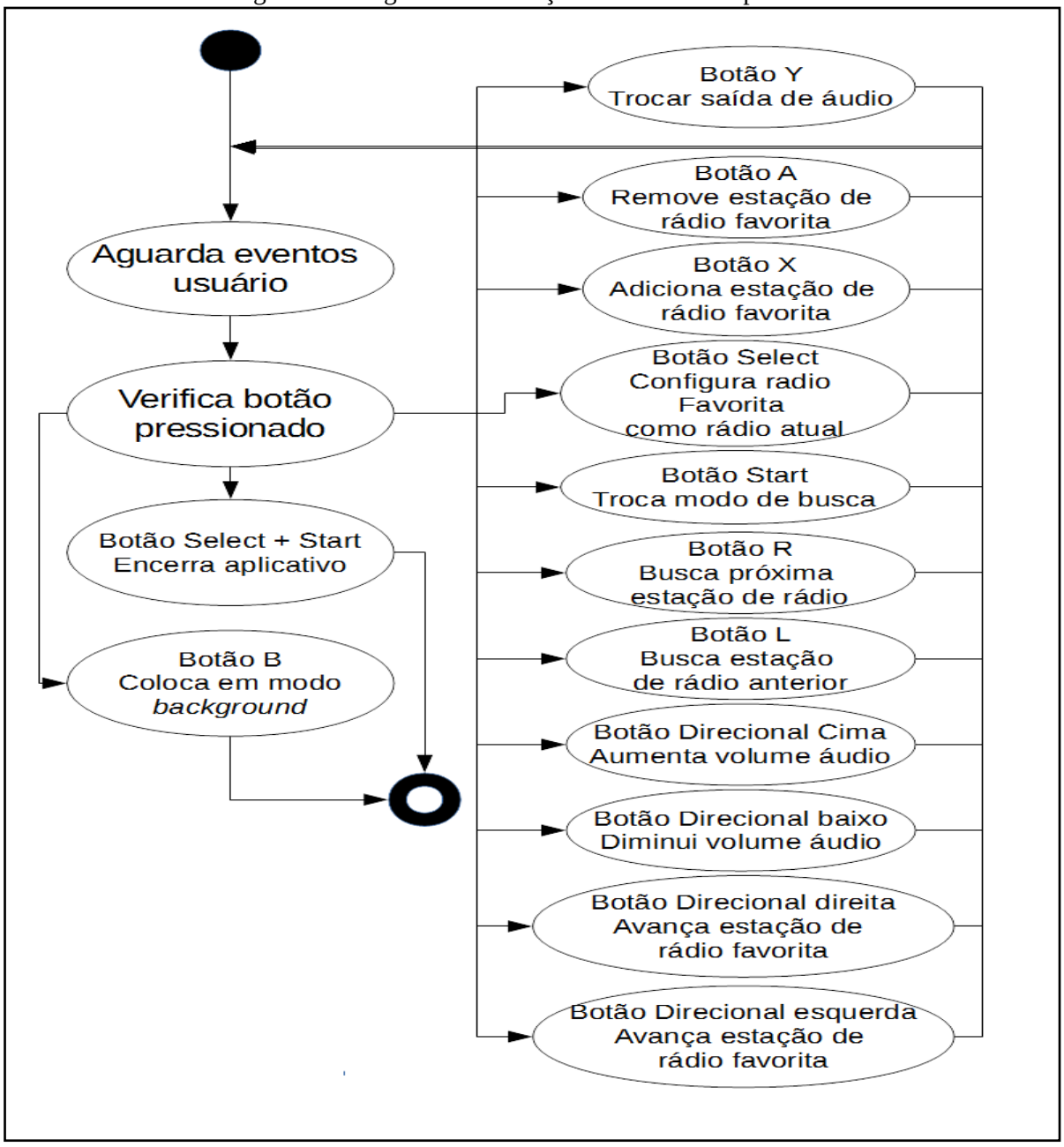
A Figura 10 mostra o diagrama de atividades relacionado as ações do aplicativo. Este diagrama descreve o fluxo de ações que o aplicativo segue conforme as ações do usuário.

Figura 10- Diagrama de atividades do aplicativo



A Figura 11 mostra um diagrama de transição de estados com as condições e comportamentos da aplicação mediante ações do usuário. As ações do usuário se dão pela interação com a interface gráfica do aplicativo.

Figura 11- Diagrama de transição de estados do aplicativo



3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas utilizadas para o desenvolvimento do trabalho. Também são mostrados trechos de código tanto do *driver* de dispositivo quanto da aplicação que interage com o *driver*.

3.3.1 Técnicas e ferramentas utilizadas

Esta seção apresenta as técnicas e ferramentas utilizadas para alteração de um *driver* de dispositivo, criação do aplicativo na camada de usuário e configuração do ambiente para compilação da aplicação para o ambiente *MIPS*.

3.3.2 Configuração do ambiente

Para ser possível fazer a compilação de programa para a plataforma *MIPS* foi necessário a utilização de um *toolchain* do tipo *cross*. O dispositivo GCW contém um *toolchain* para desenvolvedores criarem suas aplicações. Esta está disponível para o sistema operacional Linux, na plataforma 32 *bits*.Apesar de utilizar a plataforma 32 bits, o *toolchain* pode ser utilizado em plataforma 64 *bits*.Para utilizar o *toolchain* basta fazer o *download* do arquivo no *site* do *GCW*. Após o *download*, basta descompactar o arquivo e colocar este na pasta */opt* do sistema Linux. Para fazer utilização do *toolchain* se faz necessário configurar a variável ambiente *PATH* no ambiente Linux. Sem esta configuração se torna mais difícil fazer uso do *toolchain* deve-se sempre passar o caminho absoluto para o compilador em todos os momentos que se deseja compilar o programa novamente. A Figura 12 mostra um *script* para colocar o compilador do *toolchain* no *PATH* do Linux.

Figura 12 - Inserção do caminho do toolchain na variável PATH do Linux

```
export PATH="/opt/gcw0-toolchain/usr/bin:$PATH"
```

Embora o presente trabalho tenha utilizado um *driver* de dispositivo já existente, foram necessários alguns ajustes para obter o pleno funcionamento do mesmo.

O tipo de *driver* do rádio RDA5807 utilizado neste trabalho é do tipo caractere. Esta informação pode ser verificada pela saída do comando *ls -l* verificando o primeiro caractere. Se este for *c*, significa que se trata um *driver* de dispositivo do tipo caractere. Os *drivers* de dispositivo também necessitam ter um *product id* associado ao *driver*, pois este número é utilizado pelo Linux para saber qual o *driver* de dispositivo que vai ser chamado para responder a solicitação. O *chip* RDA5807 tem o número 81 como o *product id*. A Figura 13 mostra o arquivo *rádio0* simbolizando o *chip* de rádio RDA5807 no GCW0, seu tipo de *driver* de dispositivo e seu *product id*.

Figura 13 - Listagem do diretório dev, tipo do driver de dispositivo e productid

```
opendingux:/dev # ls -l radio0
crw-------    1 root     root       81,   0 Dec 27 00:41 radio0
```

Para interagir com *drivers* de dispositivo, é necessário obter um *file descriptor* relacionado ao *driver* e utilizar este na chamada de sistema *ioctl*. Para utilizar este recurso na linguagem C é necessário utilizar a chamada de sistema *open*. A chamada de sistema *ioctl* faz uso deste *file descriptor* para enviar ao Linux uma requisição de *driver* de dispositivo. Ao executar a chamada *ioctl*, o Linux utiliza o *product id* para saber qual o *driver* de dispositivo deve tratar a requisição enviada pelo usuário. Esta chamada *open* é utilizada no programa aplicativo que interage com o *driver* de dispositivos, e seu uso pode ser visto na Figura 14. Esta figura mostra uma parte de uma função escrita na linguagem de programação C. Esta rotina é responsável pela configuração inicial do rádio *FM* quando este está sendo ligado.

Figura 14 - Variável global fd sendo populada pelo file descriptor do arquivo que simboliza o hardware do rádio

```
/* init the fd global variable and set the initial config for seek */
void init_controls(void)
{
        fd = open("/dev/radio0", O_RDONLY);

        if (fd < 0) {
                fprintf(stderr, "Radio device /dev/radio0 not found! Aborting.\n");
                exit(1);
        }

        /* initial values for seek */
        seek.tuner = 0;
        seek.type = V4L2_TUNER_RADIO;
        seek.wrap_around = 1;
}
```

Com a capacidade de interagir com o *hardware* do rádio, é possível fazer configurações no *hardware* mediante a chamada *ioctl.* Originalmente, ao ligar o rádio pela primeira vez, constatou-se que haviam erros no *driver* de dispositivo. Para corrigir os erros identificados a seguir foi necessário fazer *download* do código fonte do kernel utilizado no GCW.

Este código estava hospedado no repositório online *GitHub*, então foi necessário utilizar a ferramenta *git* para poder fazer download das fontes. A Figura 15 mostra o comando executado para baixar o repositório para a estação de trabalho.

Figura 15 - Comando do programa git para fazer download do repositório do kernel do GCW

```
[marcos@localhost ~]$ git clone https://github.com/gcwnow/linux.git
```

O comando da Figura 15 cria uma pasta com o nome *linux*, e dentro desta está todo o código fonte do *kernel* Linux utilizado pelo dispositivo *GCW*. O código fonte do *driver* do dispositivo RDA5807 se encontra na pasta *linux/drivers/media/rádio* e o arquivo fonte tem o nome *rádio-rda5807.c*. O código fonte do *driver* de dispositivo foi escrito utilizando a linguagem de programação C.

A Figura 16 e a Figura 17 mostram o código que habilita a execução do *chip* RDA5807. A Figura 16 mostra como era o código anterior e a Figura 17 mostra o código atual, que teve de introduzir a função *msleep* antes de retornar o controle para a aplicação. A chamada da função *msleep* foi adicionada seguindo o manual de programação do *chip*. O manual indica esperar meio segundo após a atualização do registrador de *hardware* para habilitar o *microchip*. Assim sendo, a chamada de função *msleep* espera como parâmetro o inteiro simbolizando milissegundos, o que faz com que o Linux espere por um tempo determinado pelo *driver* de dispositivo. A chamada ao *usleep* está protegida por uma condição que verifica se a atualização do registrador de *hardware* que habilita o *microchip* foi executada com sucesso. Esta chamada é necessária pois é preciso esperar meio segundo para que a sintonização esteja completa.

Figura 16- Código antigo da função onde é habilitado o microchip RDA5807

```
static int rda5807_set_enable(struct rda5807_driver *radio, int enabled)
{
        u16 val = enabled ? RDA5807_MASK_CTRL_ENABLE : 0;
        int err;
        //dev_info(&radio->i2c_client->dev, "set enabled to %d\n", enabled);
        err = rda5807_update_reg(radio, RDA5807_REG_CTRL,
                                 RDA5807_MASK_CTRL_ENABLE, val);
        if (err < 0)
                return err;
        /* Tuning is lost when the chip is disabled, so re-tune when enabled. */
        if (enabled) {
                err =  rda5807_update_reg(radio, RDA5807_REG_CHAN,
                                          RDA5807_MASK_CHAN_TUNE,
                                          RDA5807_MASK_CHAN_TUNE);
        }

        return err;
}
```

Figura 17- Código da função atual onde é habilitado o microchip RDA5807

```
static int rda5807_set_enable(struct rda5807_driver *radio, int enabled)
{
        u16 val = enabled ? RDA5807_MASK_CTRL_ENABLE : 0;
        int err;
        //dev_info(&radio->i2c_client->dev, "set enabled to %d\n", enabled);
        err = rda5807_update_reg(radio, RDA5807_REG_CTRL,
                                 RDA5807_MASK_CTRL_ENABLE, val);
        if (err < 0)
                return err;
        /* Tuning is lost when the chip is disabled, so re-tune when enabled. */
        if (enabled) {
                err =  rda5807_update_reg(radio, RDA5807_REG_CHAN,
                                          RDA5807_MASK_CHAN_TUNE,
                                          RDA5807_MASK_CHAN_TUNE);

                /* following the rda5807 programming guide, we
                 * need to wait for 0.5 seconds before tune */
                if (!err)
                        msleep(500);
        }

        return err;
}
```

A outra alteração realizada no código do *driver* do dispositivo foi na função onde se faz a busca das estações de rádio. A Figura 18 mostra como era o código antes da correção necessária. O código antigo alterava o registrador de *hardware* e para que o *chip* entrasse em modo de *seek*, ou busca automática de estações de rádio, e retornava o controle ao aplicativo enquanto o *hardware* ainda estava executando o *seek*. Para o correto funcionamento da camada de aplicação, o *driver* de dispositivo precisa retornar somente quando o mesmo encontrar a próxima estação de rádio ou, em caso de exceção, quando não encontrar nenhuma estação de rádio. A exigência pelo sincronismo da camada de aplicação vem da necessidade da aplicação saber qual a estação de rádio corrente. Atualmente o código da camada de

aplicação solicita ao *driver* de dispositivo a estação atual logo após a chamada ao mesmo *driver* de dispositivo para fazer a busca da próxima estação. Por isso é necessário que o *driver* de dispositivo somente retorne à aplicação quando tentar encontrar a próxima estação de radio, ou quando nenhuma for encontrada.

A Figura 19 mostra a função que trata da busca da próxima estação de rádio corrigida. A alteração feita no *driver* de dispositivo introduziu uma "espera ocupada[1]" enquanto o *hardware* continua na busca pela próxima estação de rádio. Esta "espera ocupada" verifica um registrador de *Seek Tune Complete*, ou simplesmente STC, e este altera seu valor quando o *seek* está completo. Existe um caso de exceção na busca pelas estações de rádios. Quando o *seek* não consegue encontrar nenhuma estação, então o *driver* de dispositivo retorna um erro de valor inválido, que é amplamente utilizado dentro do Linux por *driver*s de dispositivo para informar que a requisição não pode ser atendida.

A busca pela próxima estação é feita pelo *hardware* verificando o nível do sinal da estação de rádio.

Figura 18- Código antigo da função do driver do dispositivo responsável pela busca das estações

```
static int rda5807_seek_frequency(struct rda5807_driver *radio,
                                  int upward, int wrap)
{
        u16 mask = 0;
        u16 val = 0;
        int ret, count = 0;

        /* TODO: Seek threshold is configurable. How should the driver handle
         *       this configuration?
         */
        /* seek up or down? */
        mask |= RDA5807_MASK_CTRL_SEEKUP;
        if (upward)
                val |= RDA5807_MASK_CTRL_SEEKUP;
        /* wrap around at band limit? */
        mask |= RDA5807_MASK_CTRL_SKMODE;
        if (!wrap)
                val |= RDA5807_MASK_CTRL_SKMODE;
        /* seek command */
        mask |= RDA5807_MASK_CTRL_SEEK;
        val  |= RDA5807_MASK_CTRL_SEEK;

        return rda5807_update_reg(radio, RDA5807_REG_CTRL, mask, val);
}
```

[1] Segundo Blieberger, BurgStaller e Scholz(2003, p. 1), é um laço de repetição que verifica repetidamente por uma condiçãopré-estabelecida.

Figura 19 - Código da função do driver do dispositivo responsável pela busca das estações

```
static int rda5807_seek_frequency(struct rda5807_driver *radio,
                                  int upward, int wrap)
{
        u16 mask = 0;
        u16 val = 0;
        int ret, count = 0;

        /* TODO: Seek threshold is configurable. How should the driver handle
         *       this configuration?
         */
        /* seek up or down? */
        mask |= RDA5807_MASK_CTRL_SEEKUP;
        if (upward)
                val |= RDA5807_MASK_CTRL_SEEKUP;
        /* wrap around at band limit? */
        mask |= RDA5807_MASK_CTRL_SKMODE;
        if (!wrap)
                val |= RDA5807_MASK_CTRL_SKMODE;
        /* seek command */
        mask |= RDA5807_MASK_CTRL_SEEK;
        val  |= RDA5807_MASK_CTRL_SEEK;

        ret = rda5807_update_reg(radio, RDA5807_REG_CTRL, mask, val);
        if (ret < 0)
                return ret;

        while (1) {
                /*
                 * The programming guide says we should wait for 35 ms for each
                 * frequency tested.
                 */
                msleep(35);

                ret = rda5807_i2c_read(radio->i2c_client,
                                       RDA5807_REG_SEEK_RESULT);
                if (ret < 0)
                        return ret;

                /* Seek done? */
                if (ret & RDA5807_MASK_SEEKRES_COMPLETE)
                        return 0;

                /*
                 * Channel spacing is 100 kHz.
                 * TODO: Should we support configurable spacing?
                 */
                count++;
                if (count > (RDA5807_FREQ_MAX_KHZ - RDA5807_FREQ_MIN_KHZ) / 100)
                        return -ETIMEDOUT;
        }
}
```

Após estas alterações no *driver* de dispositivo, foi necessário recompilar o kernel para fazer uso do novo *driver* de dispositivo. Para compilar o código o usuário necessita estar no diretório raiz do código fonte do kernel GCW, ou seja, dentro da pasta *Linux* (GCW Wiki, 2013). Para fazer a compilação de um *kernel* Linux se faz necessário utilizar a ferramenta *Make*. Esta ferramenta controla a geração de executáveis e arquivos através de código fonte (GNU Make, 2013). Para utilizar a ferramenta *Make* se faz necessário ter um arquivo com o nome *Makefile*, e dentro deste colocar as regras de compilação, quais arquivos e em qual ordem serão compilados. O *kernel* Linux contém este arquivo *Makefile*, sendo necessário somente especificar quais serão os *device drivers* que serão compilados e qual a plataforma alvo. Além de ser necessário o arquivo de configuração para especificar o que será compilado, precisa ser informado qual a arquitetura alvo da imagem. Isto se faz necessário pois o

dispositivo GCW utiliza a arquitetura *MIPS*, então a imagem do kernel Linux deve ser compilada para esta arquitetura.

Para configurar a compilação do Linux foi utilizada uma configuração padrão do GCW, chamada *defconfig*. A Figura 20 mostra os comandos de configuração e a Figura 21 mostra o comando de geração da imagem do kernel para ser utilizada no dispositivo. O parâmetro -j4 serve para executar o comando *Make* executando em 4 *threads* ao mesmo tempo, deixando a compilação mais ágil em computadores com mais de um núcleo de processamento. Após este passo, a nova imagem do kernel e gerada e está pronta para ser colocada no dispositivo.

Figura 20 - Comando para gerar a configuração do kernel no GCW

```
[marcos@localhost gcw_kernel]$ make ARCH=mips gcw0_defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/zconf.lex.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
```

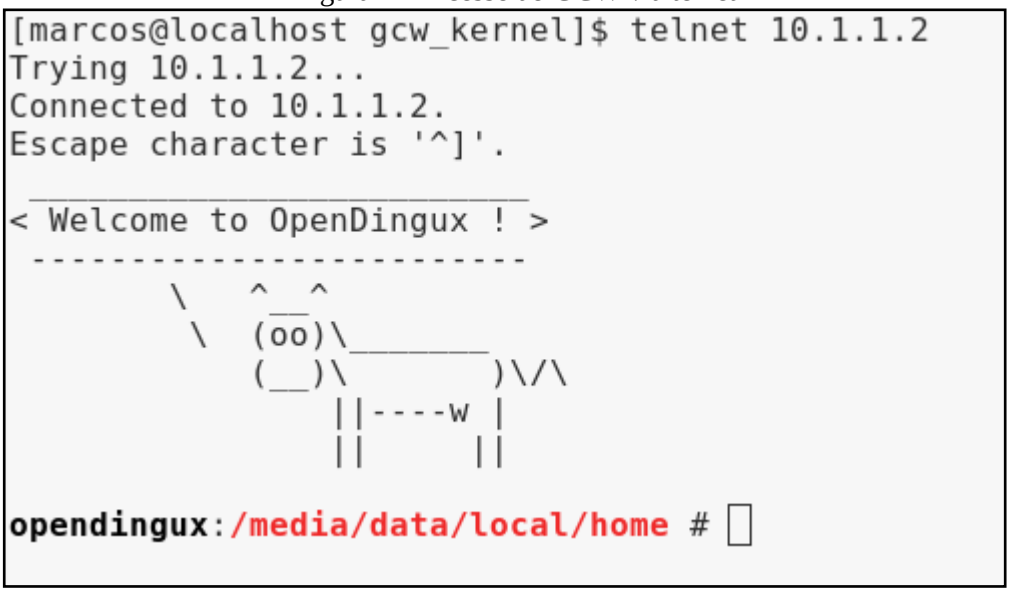
Figura 21 - Comando para gerar imagem de kernel para o dispositivo GCW

```
[marcos@localhost gcw_kernel]$ make ARCH=mips vmlinuz.bin -j4
  CHK     include/config/kernel.release
  CHK     include/generated/uapi/linux/version.h
  HOSTCC  scripts/conmakehash
  CC      scripts/mod/empty.o
  HOSTCC  scripts/mod/mk_elfconfig
  CC      scripts/mod/devicetable-offsets.s
  MKELF   scripts/mod/elfconfig.h
  HOSTCC  scripts/mod/modpost.o
  HOSTCC  scripts/mod/sumversion.o
  HOSTCC  scripts/sortextable
  GEN     scripts/mod/devicetable-offsets.h
  HOSTCC  scripts/mod/file2alias.o
  HOSTLD  scripts/mod/modpost
  UPD     include/config/kernel.release
  CHK     include/generated/utsrelease.h
  UPD     include/generated/utsrelease.h
  CC      kernel/bounds.s
  GEN     include/generated/bounds.h
  CC      arch/mips/kernel/asm-offsets.s
```

O arquivo vmlinuz.bin contém o novo *kernel* compilado, e com o novo *driver* do dispositivo RDA5807. Para instalar este novo *kernel*, é necessário fazer algumas alterações no

dispositivo. É preciso conectar-se ao dispositivo *GCW* via *telnet* para criar um arquivo de configuração, para que esse atualize o *kernel* que será utilizado no dispositivo. Ao conectar o dispositivo GCW em uma estação de trabalho, ele assume o IP 10.1.1.2, sendo desta forma possível se conectar a ele por este IP. A Figura 22 mostra como acessar o dispositivo por *telnet*, através de um cabo *Universal Serial Bus* (USB). O Linux tem um modulo padrão chamado *cdc_ether*, que faz com que seja possível acessar um dispositivo pela rede, via cabo USB.

Figura 22- Acesso ao GCW via telnet



Após ter acesso via *USB*, deve ser criado um arquivo chamado *mount_system* dentro da pasta */etc/local*. Após reiniciar o aparelho, a partição do sistema será montada na pasta */media/system*. Desta forma, foi substituído o *kernel* antigo pela novo que foi compilado. Pode-se transferir o novo kernel via *ftp*, ou até mesmo por *scp*. A Figura 23 mostra a transferência do *kernel* sendo realizada por *scp*.

Figura 23- Comando para transferir a nova imagem de kernel para o GCW

```
[marcos@localhost gcw_kernel]$ scp vmlinuz.bin root@10.1.1.2:/media/system
```

Apos transferir a imagem, o dispositivo deve ser reiniciado para fazer uso do novo *kernel*.

3.3.2.1 Criação do aplicativo para controle do rádio

Na parte da camada de usuário, um aplicativo foi desenvolvido para que o usuário possa interagir com este *driver* de dispositivo utilizando uma interface gráfica.

A resolução de tela do *GCW* é 320 de largura e 240 de altura. Com estes parâmetros, uma interface gráfica foi criada para este dispositivo utilizando a biblioteca *SDL*. Com esta, é

possível gerenciar eventos em botões, e informações a ser mostradas na tela do dispositivo. Juntamente com o *framework SDL*, é utilizada a extensão*SDL_ttf* para mostrar textos na tela utilizando *True Type Fonts.*

Para este controle foi utilizado um método de fazer *polling* nos eventos dos botões, que no framework *SDL* se chama *SDL_WaitEvent*, e tem como parâmetro um *SDL_Event*. E este parâmetro é populado com o evento recebido. Tendo a informação de cada evento recebido do usuário, ações são tomadas. Estas ações podem ser requisições para busca de uma nova estação de rádio, ou para alterar o nível do volume por exemplo. A Figura 24 mostra uma parte do código que gerencia a interface usuário. Esta é trata dos eventos do usuário, que na verdade são botões do dispositivo que foram pressionados.

Após receber eventos do usuário e necessário verificar qual foi o evento realizado. A Figura 24 mostra blocos de seleção condicional, verificando qual foi o evento do usuário. Estes blocos foram necessários, pois com eles foram mapeados os botões do dispositivo e as ações designadas.

Um dos eventos comuns que o usuário pode fazer é a alteração do nível de áudio. O controle do volume é mostrado utilizando retângulos desenhados na tela pelo framework *SDL*. Inicialmente o aplicativo desenha um retângulo na cor preta, e conforme o usuário vai alterando o nível do volume de áudio, o aplicativo vai alterando os retângulospara a cor verde. A implementação se concentra em dois vetores, um de retângulos e outro de cores. O vetor de retângulos é do tipo *SDL_Rect* com 32 posições, que é o número de níveis da placa de áudio do GCW. Existe uma função chamada *setup_volume_bar*, onde os retângulossão inicializados com suas posições na tela, e a função *draw_volume_bar*, que é responsável por mostrar o volume atual na tela, pintando os retângulos de verde. A Figura 25 mostra a função que inicia os retângulos de volume e a Figura 26 mostra a função desenha a barra de volume na tela da aplicação.

Para interagir com o *driver* de dispositivo do *chip* RDA5807 é necessário executar uma chamada de sistema chamada *ioctl*. Juntamente com a *API Video4Linux*, é possível enviar comandos ao *driver* de dispositivo para que seja possívelalterar o estado do *chip* de rádio. A função *setup* inicializa o *hardware* do *chip*, inicializa o sintonizador, seta a frequência inicial e seta o áudio de saída do *chip* de rádio para o máximo que o *chip* suporta, que é o nível 15. A configuração de áudio do *chip* de rádio está fixa para simplificar as configurações necessárias pela aplicação. A Figura 27 mostra função*setup*, que é chamada na inicialização do aplicativo.

Figura 24- Função de polling para receber eventos do usuário

```
while(SDL_WaitEvent(&event)) {
        switch (event.type) {
        case SDL_QUIT:
        case SDL_KEYDOWN:
                button_pressed = SDL_GetKeyName(event.key.keysym.sym);

                /* lock the screen */
                if (!strcmp(button_pressed, "pause")) {
                        lock = 1;
                        break;
                }

                /* unlocked screen */
                if (!strcmp(button_pressed, "unknown key"))
                        lock = 0;

                /* if the screen is locked, do nothing */
                if (lock)
                        break;

                if (!strcmp(button_pressed, "down")) {
                        /* avoid negative values */
                        if (vol) {
                                draw_volume_bar(screen, vol == 1 ? 1 : vol, VOLUME_DOWN);
                                vol--;
                                mixer_control(VOLUME_SET, &vol, &min, &max);
                                handle_sound_level(FILE_VOLUME_WRITE, &vol);
                        }

                } else if (!strcmp(button_pressed, "up")) {
                        if (vol + 1 <= max) {
                                draw_volume_bar(screen, vol == 0 ? 1 : vol + 1, VOLUME_UP);
                                vol++;
                                mixer_control(VOLUME_SET, &vol, &min, &max);
                                handle_sound_level(FILE_VOLUME_WRITE, &vol);
                        }

                /* Change to previous fav radio */
                } else if (!strcmp(button_pressed, "left")) {
                        curr_fav = curr_fav > 0 ? curr_fav - 1 : 0;
                        draw_favrads_rects();

                /* Change to next fav radio */
                } else if (!strcmp(button_pressed, "right")) {
                        curr_fav = curr_fav >= 4 ? curr_fav : curr_fav + 1;
                        draw_favrads_rects();
```

Figura 25- Função que inicializa o vetor de retângulos para mostrar o nível de volume

```
/* Initial position of each rectangle and init the colors */
void setup_volume_bar()
{
        int i, ypos = HEIGHT - 5;

        for (i = 0; i < 32; i++) {
                colors[i][0] = colors[i][2] = 0;
                colors[i][1] = 255;

                rects[i].x = VOLUME_BAR_X_POS;
                rects[i].y = ypos;
                rects[i].w = VOLUME_RECT_WIDTH;
                rects[i].h = VOLUME_RECT_HEIGHT;

                ypos -= VOLUME_BAR_DISTANCE;
        }
}
```

Figura 26 - Função que desenha na tela do GCW o nível do volume a cada alteração feita pelo usuário

```
/* draw the volume bar */
void draw_volume_bar(SDL_Surface *screen, int vol, int mode)
{
        int i = 0;

        Uint32 color = SDL_MapRGB(screen->format, colors[vol][0], colors[vol][1], colors[vol][2]);

        switch (mode) {
        case STARTUP:
                while (i <= vol) {
                        SDL_FillRect(screen, &rects[i], color);
                        i++;
                }
                break;
        case VOLUME_DOWN:
                color = SDL_MapRGB(screen->format, 0, 0, 0);
                SDL_FillRect(screen, &rects[vol], color);
                break;
        case VOLUME_UP:
                SDL_FillRect(screen, &rects[vol], color);
        }

        SDL_Flip(screen);
}
```

Figura 27 - Função setup que faz a inicialização do chip RDA5807

```
/* frequency in MHz */
void setup(float frequency)
{
        control.id = V4L2_CID_AUDIO_MUTE;
        control.value = 0;

        if (ioctl(fd, VIDIOC_S_CTRL, &control) < 0) {
                perror("ioctl: set: mute off");
                fprintf(stderr, "We can't continue without turns mute to off. Aborting.\n");
                exit(1);
        }

        if (ioctl(fd, VIDIOC_G_TUNER, &tuner) < 0) {
                perror("ioctl: set: get tuner");
                fprintf(stderr, "We can't continue without a tuner. Aborting.\n");
                exit(1);
        }

        set_frequency(frequency);

        control.id = V4L2_CID_AUDIO_VOLUME;
        control.value = 15;

        if (ioctl(fd, VIDIOC_S_CTRL, &control) < 0) {
                perror("ioctl: set volume");
                fprintf(stderr, "Using the default volume level.\n");
        }
}
```

No *kernel* do Linux não existem variáveis de ponto flutuante, então as alterações de estações de rádio devem ser convertidas de *megahertz*, que é um formato usual para usuários de rádio no dia-a-dia, para o formato de *hertz*. Segundo *Video4Linux* (2009), a sintonização de frequências acontece em unidades de 62.5 Quilohertz. Sabendo disto, a aplicação mostra ao usuário as estações de rádio no formato em *Megahertzs*, e quando a estação de rádio é enviada ao *driver* é efetuada uma multiplicação por 1.000.000 para converter o valor de *megahertz*para *hertz*. Após ter o valor em *hertz*, a aplicação divide este valor por 62,5 para que este valor seja utilizado pelo *driver* de dispositivo do receptor de rádio FM. A Figura 28 mostra a função *set_frequency* que faz este cálculo e que configura o *driver* de dispositivo para a nova estação. Esta função é utilizada dentro da função *setup*, para configurar a primeira estação de rádio ao iniciar o aplicativo. Esta função é chamada diretamente nos casos onde existe a configuração de busca de rádios manual e nos casos onde se deseja utilizar uma rádio configurada como favorita. Isto acontece pois no modo de busca de rádios automático, o próprio *driver* de dispositivo é quem sabe qual a estação de rádio em que está após a busca.

Para fazer o controle do volume de saída, foi utilizada a biblioteca *ALSA*. Esta biblioteca é utilizada tanto para criação de *drivers* de dispositivo de áudio, como para fazer interações entre aplicativos e dispositivos de áudio. Esta biblioteca teve papel fundamental no trabalho desenvolvido, pois permite a alteração do nível do volume, alteração nos modos do rádio, que se definem em modo “fone de ouvido” e modo “caixa de som”. Para fazer esta

configuração, existe uma *API* para a linguagem de programação C para fazer alterações nos dispositivos de áudio. O *hardware* de áudio do GCW tem um recurso de *bypass*, ou seja, uma maneira de fazer com que o som do rádio saia diretamente pela placa de áudio quando o *chip* de rádioestaligado.Para manipular todas as ações relacionadas a biblioteca *ALSA* a as configurações de áudio foi criada a função*mixer_control*. Esta funçãocontém todas as chamadas para a utilização da *API* do *ALSA*. A Figura 29 mostra uma parte da implementação desta função.

Figura 28- Função para configurar a frequência no driver de dispositivo

```
void set_frequency(float frequency)
{
        /* convert MHz to Hz*/
        int n_freq = (frequency * 1000000) / 62.5;

        freq.tuner = 0;
        freq.frequency = n_freq;
        freq.type = V4L2_TUNER_RADIO;

        if (ioctl(fd, VIDIOC_S_FREQUENCY, &freq) < 0) {
                perror("ioctl: set frequency");
                fprintf(stderr, "We can't continue without a frequency. Aborting.\n");
                exit(1);
        }
}
```

A *API do ALSA* permite tanto configurar funções de áudio como buscar os valores atuais das configurações do áudio. Deste modo a função *mixer_control* também serve para buscar os valores atuais configurados no *driver* de áudio. Por esta razão foram criados macros na linguagem C com atributos nomeados _SET e_GET para serem utilizados ao chamar a função *mixer_controls*. Desta forma é possível buscar valores estados no dispositivo de áudio.

Como as configurações do dispositivo de áudio estão independentes do aplicativo de rádio, podemos manter as configurações após fechar o aplicativo. Desta forma é possível ter o modo *background* no aplicativo. Esta configuração basicamente fecha o aplicativo, mas sem fazer as chamadas de *ioctl* para desligar o dispositivo do *chip* RDA5807, e sem desligar o *bypass* da camada do *ALSA*. Com esta opção o usuário pode fechar o aplicativo e interagir com outras aplicações do dispositivo GCW e ainda assim ouvir estações de áudio. Ao reiniciar o aplicativo é verificada a configuração de *by-pass*, e se esta estiver ativada significa que o rádio está sendo executado em modo *background*. A Figura 30 mostra o código onde é verificado se o rádio está sendo executado em modo *background*. Esta verificaçãoé feita antes de ser iniciada a captura de eventos do usuário.

Figura 29- Função mixer_control que faz a interação do aplicativo com a API do ALSA

```
/* Controls the alsamixer atributes of GCW device */
void mixer_control(int mode, long *volume, long *min, long *max)
{
        snd_mixer_t *handle;
        snd_mixer_selem_id_t *sid;
        snd_mixer_elem_t *elem;

        snd_mixer_selem_channel_id_t channel = SND_MIXER_SCHN_FRONT_LEFT;

        snd_mixer_open(&handle, 0);
        snd_mixer_attach(handle, "default");
        snd_mixer_selem_register(handle, NULL, NULL);
        snd_mixer_load(handle);

        snd_mixer_selem_id_alloca(&sid);
        snd_mixer_selem_id_set_index(sid, 0);

        if (mode == VOLUME_GET || mode == VOLUME_SET) {
                snd_mixer_selem_id_set_name(sid, "Headphone");
                elem = snd_mixer_find_selem(handle, sid);

                if (mode == VOLUME_GET) {
                        snd_mixer_selem_get_playback_volume(elem, channel, volume);
                        snd_mixer_selem_get_playback_volume_range(elem, min, max);
                } else if (mode == VOLUME_SET) {
                        printf("GCW: Volume set to %ld\n", *volume);
                        snd_mixer_selem_set_playback_volume_all(elem, *volume);
                }

                /* adjust volume to Bypass too */
                snd_mixer_selem_id_set_name(sid, "Line In Bypass");
                elem = snd_mixer_find_selem(handle, sid);

                if (mode == VOLUME_SET) {
                        printf("Line in Bypass: Volume set to %ld\n", *volume);
                        snd_mixer_selem_set_playback_volume_all(elem, *volume);
                }

        } else if (mode == HEADPHONE_TURN_ON || mode == HEADPHONE_TURN_OFF) {
                snd_mixer_selem_id_set_name(sid, "Headphone Source");
                elem = snd_mixer_find_selem(handle, sid);

                if (mode == HEADPHONE_TURN_ON) {
                        printf("Headphone Source: Line In\n");
                        snd_mixer_selem_set_enum_item(elem, channel, 1);
                } else if (mode == HEADPHONE_TURN_OFF) {
                        printf("Headphone Source: PCM\n");
```

Figura 30- Parte do código que verifica se o rádio está em modo background

```
/* verify if the radio is running in background */
mixer_control(BYPASS_VERIFICATION, &ret, NULL, NULL);

/* if the radio is running in background, don't set the
 * same things again
 */
if (!ret) {
        /* Initialize the radio by the driver */
        setup(curr_freq);

        /* Set the flag to turn on the capture line */
        mixer_control(mode, &vol, &min, &max);

        /* if we don't have the last_volume file, use the default */
        handle_sound_level(FILE_VOLUME_READ, &vol);

        /* set the sound volume */
        mixer_control(VOLUME_SET, &vol, &min, &max);
}
```

Além do modo *background*, o aplicativo de áudio também pode direcionar a saída de áudio para o fone de ouvido, ou para as caixas de som que estão situadas na parte inferior do dispositivo GCW. Estas configurações são possíveis graças ao framework *ALSA*. Ao utilizar o aplicativo *alsamixer* pode-se ver as possíveis configurações da placa de áudio do dispositivo GCW. A Figura 31 mostra o *alsamixer* sendo executado no dispositivo GCW.

Figura 31- Execução do aplicativo *alsamixer* no GCW

Para ouvir o rádio pelas caixas de som é necessário alterar o atributo da opção "*Line Out Source*" para "*Line In*", além de ser necessário habilitar o atributo "*Speakers*". Este último atributo é um atributo booleano. Desta maneira é possível configurar o GCW para que o áudio seja ouvido pelos fones de ouvido e pelas caixas de som ao mesmo tempo. Contudo, a fim de simplificar a interface do usuário, estas opções são mutuamente exclusivas. A Figura 32 mostra uma parte da função *mixer_control* que trata dos atributos de fones de ouvido e das caixas de som.

Figura 32- Parte de código que configura as opções de saída de áudio

```
} else if (mode == HEADPHONE_TURN_ON || mode == HEADPHONE_TURN_OFF) {
        snd_mixer_selem_id_set_name(sid, "Headphone Source");
        elem = snd_mixer_find_selem(handle, sid);

        if (mode == HEADPHONE_TURN_ON) {
                printf("Headphone Source: Line In\n");
                snd_mixer_selem_set_enum_item(elem, channel, 1);
        } else if (mode == HEADPHONE_TURN_OFF) {
                printf("Headphone Source: PCM\n");
                snd_mixer_selem_set_enum_item(elem, channel, 0);
        }

} else if (mode == SPEAKER_TURN_ON || mode == SPEAKER_TURN_OFF) {
        snd_mixer_selem_id_set_name(sid, "Speakers");
        elem = snd_mixer_find_selem(handle, sid);

        if (mode == SPEAKER_TURN_ON) {
                printf("Speaker turned on\n");
                snd_mixer_selem_set_playback_switch_all(elem, 1);
        } else if (mode == SPEAKER_TURN_OFF) {
                printf("Speaker turned off\n");
                snd_mixer_selem_set_playback_switch_all(elem, 0);
        }

        snd_mixer_selem_id_set_name(sid, "Line Out Source");
        elem = snd_mixer_find_selem(handle, sid);

        if (mode == SPEAKER_TURN_ON) {
                printf("Line Out Source turned on\n");
                snd_mixer_selem_set_enum_item(elem, channel, 1);
        } else if (mode == SPEAKER_TURN_OFF) {
                printf("Line Out Source turned off\n");
                snd_mixer_selem_set_enum_item(elem, channel, 0);
        }

} else if (mode == BYPASS_VERIFICATION) {
```

Também foi disponibilizado o recurso de rádios favoritas. Este recurso procura agilizar a troca de estações de rádios pelo usuário. Esta implementação se mostrou simples, uma vezque foi necessário manter um vetor de *strings* para manter as frequências das estações de rádios e uma função para desenhar estes na tela do GCW. A Figura 33 apresenta a função que mostra rádios favoritas na tela do GCW.

O aplicativo de rádio também tem o recurso de guardar as últimas configurações do usuário. Este recurso guarda as informações da última estação de rádio que o usuário estava ouvindo, a última configuração do nível do volume, o modo de saída de áudio, se este estava sendo ouvido pelos fones de ouvido ou pelas caixas de som, e as rádios favoritas configuradas pelo usuário.Estas configurações são guardadas em arquivos de texto dentro da pasta HOME do usuário. Ao salvar as configurações, uma pasta oculta é criada dentro da pasta HOME, e dentro desta pasta são criados arquivos para salvar cada tipo de preferência. Por exemplo, é criado um arquivo chamado *last_mode* para guardar o último modo de uso, um arquivo chamado *last_volume* para guardar o último volume configurado pelo usuário, um arquivo chamado *last_freq para* guardara última estação de rádio. Esta implementação também contém alternativas para buscar as informações guardadas nos arquivos. Esta busca é utilizada quando o aplicativo é iniciado para fazer a configuração inicial. Se não existirem configurações salvas anteriormente, no caso de ser a primeira execução do aplicativo, valores padrão são utilizados. A Figura 34 mostra uma função para salvar e buscar informações de nível de áudio, e a Figura 35 mostra uma parte do código de inicialização onde são verificadas as últimas configurações do usuário.

Figura 33- Código onde são desenhadas as rádios favoritas

```
/* To be able to draw borders, we need to draw a bigger rect, and after a small one
 * filled by black color */
static void draw_favrads_rects()
{
        Uint32 black_color = SDL_MapRGB(screen->format, 0, 0, 0);
        Uint32 green_color = SDL_MapRGB(screen->format, 0, 255, 0);
        Uint32 white_color = SDL_MapRGB(screen->format, 255, 255, 255);

        int i = 0;

        for (i = 0; i < 5; i++) {
                /* Selected favorite radio has green border */
                if (i == curr_fav)
                        SDL_FillRect(screen, &favrad_rects[i], green_color);
                else
                        SDL_FillRect(screen, &favrad_rects[i], white_color);

                SDL_FillRect(screen, &favrad_rects_border[i], black_color);

                printf("Radio %s\n", favrads.radio[i]);

                int freq_size = strlen(favrads.radio[i]);

                char freq[6];

                if (freq_size == 1)
                        strcpy(freq, "-");
                else
                        strcpy(freq, favrads.radio[i]);

                /* Draw favorite radio into rect */
                desc_fav_rad_info = TTF_RenderText_Solid(desc_fav_rad_font, freq, font_color);
                apply_surface(favrad_rects[i].x + 10, 35, desc_fav_rad_info, screen);
        }
        SDL_Flip(screen);
}
```

Figura 34- Função para guardar e buscar dados de nível de áudio

```
/* save/restore sound level */
void handle_sound_level(int mode, long *volume)
{
        char line[3];

        if (path[0] != '0') {
                sprintf(aux_path, "%s/%s", path, "last_volume");

                if (mode == FILE_VOLUME_READ) {
                        file = fopen(aux_path, "r");

                        /* let the volume parameter as it was */
                        if (!file)
                                return;

                        fgets(line, 3, file);
                        sscanf(line, "%ld", volume);

                } else if (mode == FILE_VOLUME_WRITE) {
                        file = fopen(aux_path, "w");

                        if (!file) {
                                fprintf(stderr, "Cannot set the actual volume level!\n");
                                return;
                        }

                        fprintf(file, "%ld", *volume);
                }
                if (file)
                        fclose(file);
        }
}
```

Figura 35- Função que utiliza a busca do último nível de volume utilizado

```
/* if we don't have the last_volume file, use the default */
handle_sound_level(FILE_VOLUME_READ, &vol);
```

Para executar o aplicativo do rádio no GCW mediante um ícone na tela é necessário que o aplicativo seja armazenado em um arquivo criado via *squashfs*. Junto aos executáveis do aplicativo também deve ser armazenado um arquivo definindo o tipo de aplicação que está sendo armazenada em uma estrutura de *squashfh*. A Figura 36 mostra o arquivo onde são definidos o nome e tipo de aplicativo para este ser mostrado na tela do GCW. A Figura 37 mostra uma parte do arquivo Makefile responsável por criar a imagem *squashfs* com todos os arquivos necessários para pode executar o aplicativo no dispositivo GCW.

Figura 36- Arquivo responsável por fazer a aplicação de rádio ser classificada no GCW

```
[Desktop Entry]

Type=Application
Name=Radio
Comment=A nice radio player for GCW
Exec=radio
Icon=radio
Categories=applications;
```

Figura 37- Comando de criação da imagem *squashfs*

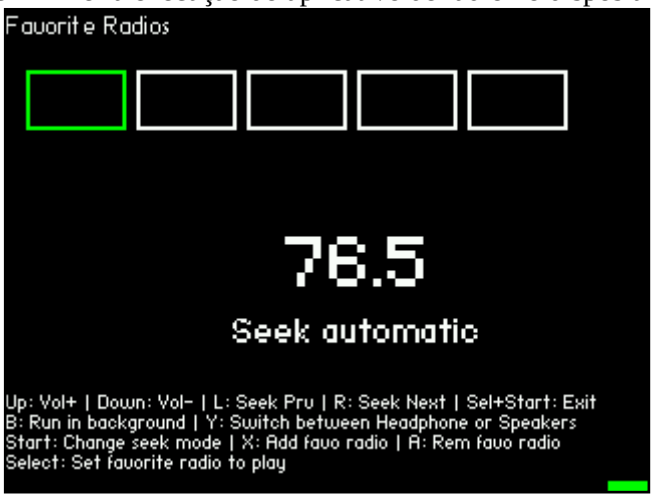
```
bin: build
	mkdir radio_player
	cp radio radio.png Fiery_Turk.ttf README default.gcw0.desktop radio_player
	mksquashfs radio_player radio_player.opk -all-root -noappend -no-exports -no-xattrs
	rm -rf radio_player
```

3.3.3 Operacionalidade da implementação

Está seção demonstra as funcionalidades da aplicação através de uma sequência de imagens.

A Figura 38 mostra o aplicativo em sua primeira execução. Sendo a primeira execução, a estação de rádio inicial é 76.5, pois é o limite mínimo permitido pelo *chip* RDA5807. Na tela inicial tambémestãoasposições das rádios favoritas. Neste ponto todas estas estão vazias, pois não existe nenhuma estação de rádio favorita configurada. Também pode-se verificar a barra de volume no lado esquerdo, a barra com os atalhos na parte inferior da tela e o modo de busca do rádio, que inicialmente é o modo automático.

Figura 38- Primeira execução do aplicativo de rádio no dispositivo GCW



A Figura 39 mostra a aplicação após o usuário aumentar o volume da aplicação. Pode-se verificar que a barra verde situada no lado direito da tela aumentou, simbolizando o atual nível de áudio.

Figura 39- Aplicação de rádio mostrando nível de volume após este ser alterado

A Figura 40 mostra o usuário fazendo uma busca por uma estação de rádio. A busca foi feita utilizando o modo automático, onde o *chip* verifica qual a próxima rádio que tem um nível de sinal razoável e então sintoniza nesta estação.

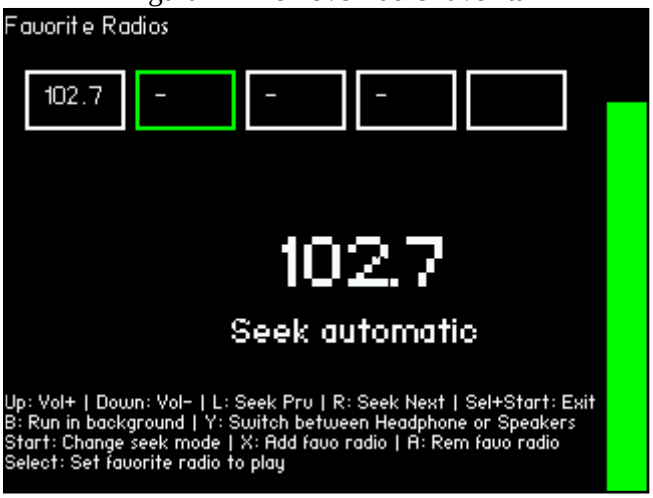
Figura 40- Aplicativo efetuando a busca por nova estação rádio

A Figura 41 mostra duas estações de rádio sendo configuradas como favoritas pelo usuário. Esta açãoé executada pelo usuário ao clicar no botão X do dispositivo GCW.

Figura 41- Adicionando rádios favoritas

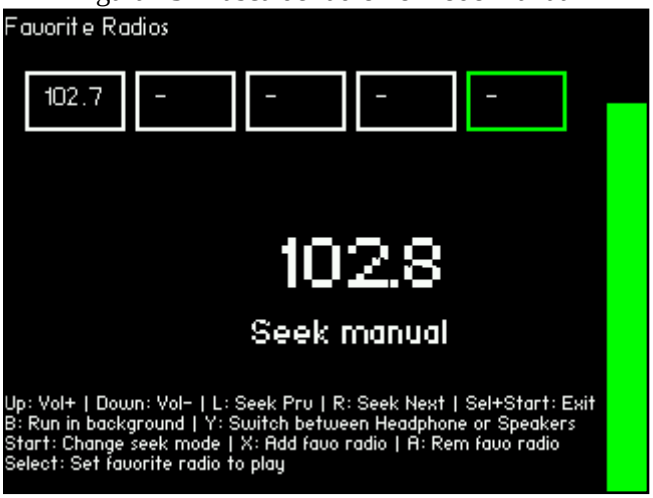
A Figura 42 mostra uma rádio favorita que foi removida. Esta ação é feita pelo usuário quando o botão A é pressionado.

Figura 42- Remover rádio favorita



A Figura 43 mostra uma busca de rádio utilizando o modo de busca manual. Neste modo, a estação de rádio corrente vai sendo incrementada a cada 10kHertz, não sendo necessario utilizar chamadas de busca pelo chip RDA5807.

Figura 43- Busca de rádio no modo manual



3.4 RESULTADOS E DISCUSSÃO

Conforme descrito, os objetivos propostos foram alcançados. Algumas dificuldades foram encontradas nas etapas nas etapas de utilização dos *frameworks SDL* e *ALSA*. Tais dificuldades não estão relacionadas com a documentação das bibliotecas e sim com o aprendizado do escopo em que estas bibliotecas se encontram.

A biblioteca *SDL* foi utilizada para facilitar a criação de interface com o usuário. Seu aprendizado não foi fácil no início. Conforme o trabalho foi sendo desenvolvido, a taxa de produtividade foi aumentando, o que facilitou a finalização do trabalho.

Considerando a resolução pequena do dispositivo GCW, o número máximo de rádios favoritas definidas pelo usuário e de cinco estações.

A única dificuldade encontrada nesta implementação foi o recurso para desenhar uma borda nos retângulos das rádios favoritas. Como não existe o conceito de bordas em retângulos no framework *SDL* a alternativa encontrada foi desenhar dois retângulos, um com a cor da borda e outro retângulo da cor preta menor do que o outro desenhado anteriormente. Desta forma é possível desenhar uma espécie de borda ao redor da frequência favorita do usuário. Os retângulos possuem tamanhos fixos de borda e do retângulo interno.

A biblioteca *ALSA* se mostrou realmente muito útil para o trabalho. Basicamente as dificuldades estiveram relacionadas as configurações que se fizeram necessárias para alterar as configurações da placa de áudio do dispositivo GCW. Não foi encontrada documentação sobre como um desenvolvedor poderia configurar uma placa de áudio utilizando os tipos enumerados nesta biblioteca juntamente com a linguagem C. Após verificar a documentação da *API* e feitos alguns testes, pode-se verificar como seria possível fazer estas configurações. Conforme a biblioteca foi sendo utilizada, esta foi sendo assimilada, e então o trabalho foi fluindo para implementar tudo o que havia sido especificado.

Uma dificuldade desta implementação esteve relacionada ao estudo de quais são os índices das opções que se deseja configurar. Por exemplo, a opção "*Headphone Source*", que tem a opção inicial chamada "PCM". As opções possíveis nesta configuração são "PCM", "*Line In*" e "*Mic*". Para poder configurar o áudio para ser ouvido nos fones de ouvido é necessário que o *driver* de áudio seja configurado como "*Line In*". Pela *API* do *ALSA* esta opção é definida como um tipo enumerado, e com isto é necessário passar um índice para esta configuração. Neste caso, para habilitar o áudio no fone de ouvido, basta usar o índice 1 pela *API* do *ALSA*.

O Quadro 10 mostra um comparativo entre este e os outros dois trabalhos correlatos.

Quadro 10 - Comparação de trabalhos correlatos

Função	**Aplicativo rádio**	**Minivosc**	**Toolkit**
Implementação em*device driver*	X	X	
Implementação na camada do usuário	X		X
Ambiente Linux	X	X	X
Dispositivos embarcados	X		X

4 CONCLUSÕES

Sobre a linguagem de programação utilizada neste trabalho, a linguagem C se mostrou adequada para a implementação do aplicativo de controle de rádio. Isto se deve ao fato de existirem muitos *frameworks*, para diversas funcionalidades diferentes, e alguns deles se adequaram muito bem no desenvolvimento, como o *framework SDL* e o *framework ALSA*. O *toolchain* do GCW se mostrou muito completo em relação ao suporte de diversos *frameworks*, inclusive os utilizados neste trabalho, poupando assim muito tempo de configuração de ambiente. Antes deste trabalho, não existia nenhuma interface gráfica para controle deste dispositivo de rádio.

Sobre o *framework* gráfico *SDL*, este se mostrou adequado para este tipo de trabalho tendo facilitado o desenvolvimento da interface gráfico do dispositivo. Também foi utilizada uma extensão deste *framework*, chamada *SDL_ttf*. Esta foi utilizada para escrever informações de texto na tela utilizando uma fonte *True Type*.

Sobre a distribuição Linux *OpenDingux*, este se mostrou uma distribuição estável considerando-se seu curto período de existência. Pela vasta documentação existente no Linux, foi relativamente simples de fazer chamadas de *driver* de dispositivo para ativar e configurar o dispositivo de *hardware*. Esta documentação teve uma importância crucial no desenvolvimento do trabalho, uma vez que esta era uma parte importante do trabalho.

Embora o trabalho desenvolvido tenha exigido um conhecimento prévio de assuntos como compilação do kernel Linux, os *frameworks* foram sendo dominados no decorrer do trabalho. As principais dificuldades encontradas foram devido a falta de experiência nos *frameworks* gráficos como o *SDL* e *ALSA* e a falta de experiência com chamadas de sistema para ativar *drivers* de dispositivo.

O código fonte do aplicativo se encontra no endereço: https://github.com/marcosps/gcw_radio.

O trabalho atingiu os objetivos propostos embora possua as seguintes limitações:

a) a busca de rádios é feita de forma síncrona;
b) a interface com o usuário pode ser aprimorada;
c) o rádio para de funcionar após o sistema entrar em modo suspenso;
d) o nível de sinal utilizado para a busca de estações de rádio é fixo no *driver* de dispositivo;
e) a interface de usuário disponível somente na língua inglesa.

4.1 EXTENSÕES

Melhorias podem ser introduzidas neste trabalho, tais como:

a) melhorar a interface do usuário, deixando menos parecida com um terminal Linux;
b) utilizar um banco de dados como o SQLite para armazenar as preferências de usuárioaoinvés de utilizar arquivos com texto;
c) permitir uma configuração de nível de sinal aceito pelo rádio, para possibilitar o usuário ouvir estações de rádio de uma maior distância;
d) submeter o *driver* de dispositivo utilizado neste trabalho para a versão oficial do Linux.

BIBLIOGRAFIA

ALSA. Sound card test, 2008. Disponivel em: <http://www.alsa-project.org/main/index.php/SoundcardTesting>. Acesso em: 21 jun 2014.

BEN-YÉUDA, M. **Ten things every Linux programmer should know**. Linux Kernel Workshop. [S.l.]: [s.n.]. 2004.

BLIEBERGER, J.; BURGSTALLER, B.; SCHOLZ, B. Busy wait analysis. In. Ada-Europe International Conference on Reliable Software Tecnhnologies, 8, Toulose, jun. 2003. **Proceedings…** Toulose, 2003, p.142-152.

CAMPOS, A. **O que é Linux**, [S.l], 2008. Disponivel em: <http://br-linux.org/linux/faq-linux>. Acesso em: 9 maio 2014.

COBBAUT, P. **Linux fundamentals**, [S.l.], 2014. Disponivel em: <http://linux-training.be/files/books/LinuxFun.pdf>. Acesso em: 10 abr 2014.

DIMITROV, S.; SERAFIN, S. Minivosc - a minimal virtual oscillator driver for ALSA (Advanced Linux Sound Architecture). In:Linux Audio Conference, 10, California, US, 2012. **Proceedings….**California, 2102. p. 175-182.

DINGOONITY. About, 2012. Disponivel em: <http://wiki.dingoonity.org/index.php?title=Dingoo:About>. Acesso em: 10 abr. 2014.

GCW SPECIFICATIONS. Specifitions, 2014. Disponivel em: <http://www.gcw-zero.com/specifications>. Acesso em: 2014.

GCW WIKI. Building the kernel, 2013. Disponivel em: <https://github.com/gcwnow/linux/wiki>. Acesso em: 14 jun 2014.

GNU. Linux and the GNU System, 2014. Disponivel em: <http://www.gnu.org/gnu/linux-and-gnu.html>. Acesso em: 19 jun 2014.

GNU MAKE. Capabilities of Make, 2013. Disponivel em: <http://www.gnu.org/software/make>. Acesso em: 14 jun. 2014.

GOOCH, R. The Linux device file-system.In:Ottawa Linux Symposium, 2, Ottawa, JP, 2001. **Proceedings…** Ottawa, 2001. p. 1-11.

JEONG, S. Tizen: Overview and Architeture, 2012. Disponivel em: <https://events.linuxfoundation.org/images/stories/pdf/lceu2012_haitzler.pdf>. Acesso em: 15 jun 2014.

KICKSTARTER. GCW-Zero: Open Source Gaming Handheld, 2013. Disponivel em: <https://www.kickstarter.com/projects/gcw/gcw-zero-open-source-gaming-handheld>. Acesso em: 19 jun. 2014.

PRADO, S. **Linux Device Drivers - Parte 1**, 2010. Disponivel em: <http://sergioprado.org/linux-device-drivers-parte-1>. Acesso em: 3 jun 2014.

PRADO, S. **Desmistificando toolchains em Linux embarcado**, 2011. Disponivel em: <http://sergioprado.org/desmistificando-toolchains-em-linux-embarcado/>. Acesso em: 13 jun. 2014.

RADON, M. **Dingoo A320 pocket retro game emulator review**, 2012. Disponivel em: <http://www.gadgetreview.com/2012/01/dingoo-a320-pocket-retro-game-emulator-review.html>. Acesso em: 21 jun. 2014.

RUSLING, D. **The Linux kernel source**, 1999. Disponivel em: <www.tldp.org/LDP/tlk/>. Acesso em: 14 jun. 2014.

SIMMONDS, C. **The embedded Linux quick start guide**, San Francisco, 2010. Disponivel em: <http://elinux.org/images/4/4f/02-linux-quick-start.pdf>. Acesso em: 16 jun. 2014.

SLOSS, A. N.; DOMINIC, S.; CHRIS, W. **ARM system developer's guide**. San Francisco: Elsevier, 2004.

SQUASHFS. **What is SquashFS**, 2008. Disponivel em: <http://tldp.org/HOWTO/SquashFS-HOWTO/whatis.html>. Acesso em: 16 jun. 2014.

TRINE 2. Media, 2011. Disponivel em: <http://trine2.com/site/index.php?page=media>. Acesso em: 17 maio 2014.

VIDEO4LINUX. Video for Linux Two API Specification, 2009. Disponivel em: <http://www.linuxtv.org/downloads/legacy/video4linux/API/V4L2_API/spec-single/v4l2.html>. Acesso em: 15 jun. 2014.

WALTRIK, T. **Toolkit para Linux embarcado**. 2011. 81f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais. Universidade Regional de Blumenau, Blumenau.

ZANONI, F. S. **Avaliacao de arquitetura paralela para smartphones e tablets GNU/Linux e MPI para processamento de dados**. 2013. 98f. Dissertação (Mestrado em Eng.Elétrica) - Programa de Pós Graduação em Eng. Elétrica. Universidade de São Paulo, São Carlos.