UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ESCOLA POLITÉCNICA

DEPARTAMENTO DE ELETRÔNICA E DE COMPUTAÇÃO

SISTEMA DE MONITORAÇÃO DE VÍDEO

Autor: _____

Leonardo dos Anjos Chaves

Orientador: _____

Eduardo Antônio Barros da Silva

Examinador: _____

Sérgio Barbosa Villas Boas

Examinador: _____

Lisandro Lovisolo

DEL
Outubro de 2006

DEDICATÓRIA

Dedico esse projeto aos meus pais, Abelardo e Zuleica, pelo amor deles, porque me deram sempre a oportunidade de estudar e acreditaram na minha escolha pela Engenharia. Poucos nessa nação têm tal oportunidade e me sinto privilegiado por conquistar mais essa etapa, graças a eles.

Dedico também a minha amada noiva Luciana, pelos incentivos diários e paciência nas horas difíceis, nas noites sem dormir, nas vezes que deixamos de sair e principalmente por fazer parte do momento mais importante na minha carreira, meu primeiro emprego.

DEL
Outubro de 2006

AGRADECIMENTO

Primeiramente, agradeço a Deus pelo dom da vida e porque sem Ele, não seria capaz de chegar até aqui.

Agradeço aos meus novos amigos e colegas de turma pelos momentos que passamos juntos, dentro e fora da sala de aula, nas provas, churrascos, aniversários, viagens e tantas outras coisas.

Agradeço, de maneira especial, ao meu orientador Eduardo Antônio Barros da Silva, pelos ensinamentos e conselhos que foram e são tão importantes para a minha formação. Agradeço a confiança que depositou em mim durante os projetos de iniciação científica e pelo exemplo de profissional em que tento me espelhar.

RESUMO

Um sistema de monitoração de vídeo é comumente utilizado em aplicações ligadas a segurança. Em sistemas de vigilância é essencial que haja uma ferramenta capaz de capturar as imagens de um ou mais locais com o objetivo de armazená-las. É comum também existir outra ferramenta que seja capaz de exibir os vídeos gravados de maneira que as imagens sejam monitoradas posteriormente ou em tempo real.

Com a intenção de criar um ambiente de desenvolvimento para aplicações que utilizem recursos de sistemas de monitoração, foi idealizado um conjunto de funções definidas nesse projeto. Agrupadas na biblioteca denominada LIBMONITOR, essas funções foram construídas para que as ações de captura, compressão, armazenamento e reprodução sejam utilizadas de maneira rápida, intuitiva e direta.

Esse ambiente, escrito em C++, contém dois grandes módulos: o de captura e o de visualização, os quais são abordados nesse projeto detalhadamente. Além disso, serão apresentadas as pesquisas que definiram a compra dos equipamentos utilizados, como, por exemplo, a câmera de segurança.

Por fim, esse projeto apresenta uma solução para um sistema de vigilância. Duas ferramentas foram desenvolvidas ao longo do projeto para apresentá-la de maneira completa. Essas ferramentas são o VideoCapture e o Cross-Player, que também serão descritas. O objetivo é apresentar uma solução que atenda

aos requisitos de capturar, codificar e armazenar um sinal de vídeo para depois reproduzí-lo, quando e onde for requisitado.

ABSTRACT

A video monitor system is commonly used into security areas. These systems must support some capabilities like image capture and reproduction tools. One of these tools can be a player that will be able to show recorded scenes.

This project aims to develop the programs that support these capabilities and also create a monitoring framework. It should be able to offer a large set of monitor functions. These functions are grouped into a library called LIBMONITOR and they are capable to do capture, compression and storage, as well as to display videos in an easy to use way.

The LIBMONITOR was developed based on C++ language and is divided into two large modules: capture and display, which will be described in details, throught this document. Furthermore, a market researche about the essential equipment to be used in this project, such as the security camera, is also presented.

PALAVRAS-CHAVE

I. MONITORAÇÃO

II. VÍDEO

III. CAPTURA

IV. REPRODUÇÃO

V. CODIFICAÇÃO

ÍNDICE DO TEXTO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ABREVIATURAS

API – *Application Program Interface* – Interface de Programação de Aplicativos

NTSC – *National Television System Committee*. Padrão americano de tv analógica

CMOS – Semicondutor de óxido metálico

CCD – *Charge Coupled Device*

GLOSSÁRIO

Pixel – Elemento de imagem

Stream – seqüência ou fluxo contínuos

Slider – controle de programa para avançar ou recuar, semelhante à barra de progressão, que permite o acesso distinto aos quadros exibidos em um *stream* de vídeo.

Late-bind – Associação tardia

Codec – acrônimo de codificador e decoficador

Capítulo 1

Introdução

Um sistema de monitoração é muito útil quando é necessário observar e/ou gravar um determinado evento. Esse evento pode ser classificado como auditivo, visual, ou até mesmo uma combinação com eles.

Um exemplo evidente aco ntece com os repórteres correspondentes que cobrem um acontecimento em um lugar onde não há infra-estrutura para transmissão ao vivo. Eles utilizam equipamentos gravadores de áudio e/ou câmeras de vídeo para capturar sons e imagens do evento. Assim, essas informações são armazenadas nos dispositivos específicos de cada aparelho, como discos óticos digitais, como CDs e DVDs, ou até mesmo magnéticos como as antigas fitas K7. Essas mídias serão depois transportadas para o estúdio para que a matéria seja adequada ao formato do noticiário que a veiculará.

Outras diferentes aplicações de monitoração foram criadas ao longo do tempo, fomentadas em necessidades diárias, como os exemplos aplicados ao controle de temperatura de um determinado ambiente. Muitos equipamentos em operação devem ser mantidos em um ambiente com temperatura constante, para que não superaqueçam. Isso se aplica também à área de conservação dos alimentos, onde, por exemplo, alguns frigoríficos mantêm seus produtos congelados para que permaneçam conservados durante um período maior de tempo.

No segmento de segurança, sistemas de monitoração de vídeo são úteis para permitir que locais sejam vigiados sem a necessidade da presença física de um profissional. Dessa forma, permite-se que todas as imagens capturadas sejam centralizadas num dispositivo ou local dedicado, oferecendo riscos menores de exposição.

Outra aplicação muito útil para os motoristas da cidade do Rio de Janeiro é a utilização das imagens capturadas pelas câmeras de trânsito. A CET, Companhia de Engenharia de Tráfego, tem um parque de câmeras instaladas ao longo da cidade, que monitoram as principais ruas e avenidas de acesso. Um dos serviços oferecidos pela SMTR, Secretaria Municipal de Transportes da prefeitura do Rio de Janeiro, é disponibilizar essas imagens para as emissoras de televisão que as veiculam nos telejornais, ou disponibilizá-las na internet. Assim, o motorista tem a oportunidade de, antes de sair de casa, saber se o seu trajeto essará engarrafado, com acidentes, ou se encontrará um fluxo de carros normal. Essas informações podem ser obtidas em [20].

Esse projeto consiste no desenvolvimento de um sistema de monitoração de vídeo baseado em um computador pessoal. Durante a implementação dessa solução foi necessário criar um ambiente de programação para permitir que futuras adaptações e a criação de novas funcionalidades aconteçam de forma dinâmica e rápida. Assim, foi idealizada uma biblioteca denominada LIBMONITOR para que as comunicações com o hardware e com as aplicações sejam feitas sem que seja necessário desenvolver códigos de acesso de baixo nível.

A biblioteca é composta por dois grandes módulos: 1) Captura e 2) Visualização, esses viabilizaram a implementação das duas aplicações principais: o VideoCapture e o Cross-Player, descritas ao longo desse documento. Através desses programas é possível capturar vídeos, armazená-los e posteriormente reproduzí-los.

Os capítulos 2 e 3, a seguir, descrevem a motivação e o objetivo do projeto, detalhando o escopo do desenvolvimento das funcionalidades e características da biblioteca e dos programas. Posteriormente, no capítulo 4, é feita uma análise do problema em questão, com a apresentação dos parâmetros de captura e codificação do sinal. Consideramos também a possibilidade de portar essa solução para outras plataformas, reunindo pesquisas e especificações das bibliotecas gráficas e demais recursos relevantes para tal fim.

No capítulo 5 é apresentada a estrutura de programação, com as especificações das bibliotecas auxiliares, as fases e ações que conduziram à implementação do projeto. Além disso, são vistas as necessidades e dificuldades de hardware encontradas e a solução proposta.

A biblioteca LIBMONITOR é descrita no capítulo 6, com duas seções dedicadas aos dois principais módulos: de captura e visualização. É apresentada com detalhes a modelagem das classes e a relação de hierarquia entre elas, bem como as funções e estruturas usadas das bibliotecas auxiliares.

No capítulo 7, concentramo-nos na descrição dos programas VideoCapture e CrossPlayer, nas suas funcionalidades, telas e execuções. Já no capítulo 8, são apresentadas outras aplicações para esse sistema, bem como, diferentes plataformas de desenvolvimento. Por exemplo, é considerado um sistema embarcado como em placas baseadas em DSP (*Digital Signal Processor*). Por fim, no capítulo 9, concluímos o projeto, com considerações sobre trabalhos futuros e sobre o curso de Engenharia Eletrônica e de Computação.

Capítulo 2

Motivação

Para representar a motivação do projeto foram reunidos três argumentos que dizem respeito à cultura, aos costumes da vida urbana nos dias atuais, à comodidade e à facilidade conquistadas através de avanços tecnológicos.

Pedestres de uma grande metrópole, como o Rio de Janeiro, já se depararam em seus condomínios, nos elevadores, essacionamentos, em shoppings e lojas com um aviso que diz: "SORRIA, VOCÊ ESTÁ SENDO FILMADO", como ilustrado na Figura 1. Isto já se tornou comum nesse século XXI, onde a ordem em algumas cidades grandes, como nas capitais brasileiras, é pautada pelo controle e disciplina dos seus moradores. Essa é uma cultura criada devido à necessidade de monitorar os ambientes públicos e/ou particulares que são palcos de insegurança e medo. Desse modo, o uso de uma câmera tenta inibir as possíveis ações de algum criminoso, ou seja, o policiamento é alcançado através de monitoração.

Figura 1: Cartaz comum nos lugares com câmeras

Considerando a pesquisa realizada pela empresa IT Data, a pedido da ABINEE, Associação Brasileira da Indústria Elétrica Eletrônica, o número de computadores nos lares brasileiros cresceu consideravelmente no primeiro semestre desse ano de 2006 [21, 22 e 24]. Segundo essa pesquisa, houve um aumento de 43% na quantidade de computadores nos lares em relação ao mesmo período do ano passado. Os números se tornam mais expressivos se considerarmos que 61% de todos os computadores vendidos durante o primeiro semestre de 2006 são referentes à primeira aquisição deste eletrodoméstico. Em 2005, a venda para a compra do primeiro equipamento foi 6% menor. Resumindo, segundo a PNAD, Pesquisa Nacional por Amostra de Domicílio, de 2006, um em cada cinco lares brasileiros tem computador.

Devemos considerar também que, atualmente, o computador é uma peça indispensável, pois oferece diversas facilidades para o usuário: a criação de trabalhos para a escola, entretenimento, desenvolvimentos de projetos, tesses e simulações com softwares de pesquisa. Além disso, associado à internet, viabiliza o acesso a portais de relacionamentos, imenso volume de informações, a sites de buscas e dos mais variados assuntos.

Assim como o computador, a tecnologia digital está cada vez mais presente nos projetos de engenharia, viabilizando soluções que dependem estritamente desta tecnologia. Nos sistemas de vigilância antigos era comum os usuários se depararem com as imagens de diversas câmeras sendo exibidas em seqüência. Isto porque o sinal de vídeo tinha que ser comutado num equipamento chamado matriz, como ilustrado nas Figuras 2 e 3. Os vídeos eram gravados ainda em fitas VHS e dependendo do tempo disponível para gravação de cada fita, um grande volume de fitas era necessário caso a freqüência de reutilização das mesmas fosse pequena, evitando a regravação. Para que o vigilante pudesse ter uma visão simultânea de todas as câmeras, era preciso instalar monitores independentes para cada fonte de vídeo, ou seja, quanto mais câmeras, mais monitores. Isso tornava a solução custosa, e, além disso, requeria pois dependia que o espaço físico de monitoração comportasse todos os equipamentos.

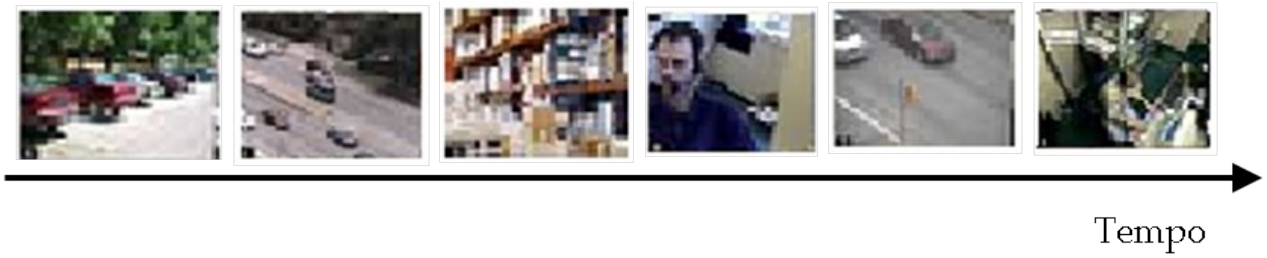


Figura 2: Exibição seqüencial das cenas

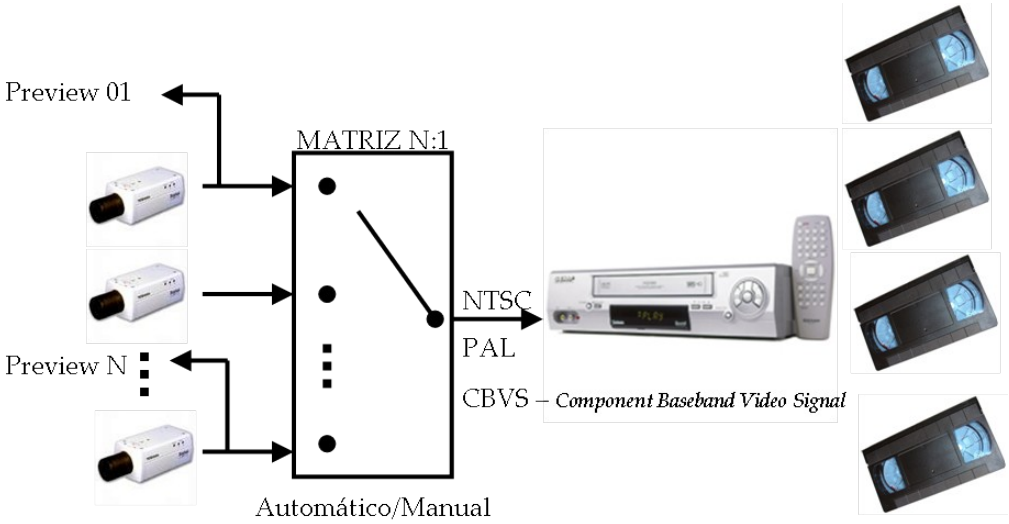


Figura 3:Representação dos sistemas de monitoração analógicos

Com a tecnologia digital, foi possível processar os sinais de vídeo, representando-os com bits, dividindo-os em pacotes e formatando-os numa estrutura de transporte. O *bitstream*, como é chamada a seqüência de bits do sinal, é gerado a partir do multiplexador, ou MUX, representado no diagrama da Figura 4. Essa informação agora pode ser arquivada em mídias digitais, como CDs, DVDs, discos rígidos, entre outros, dependendo da solução utilizada.

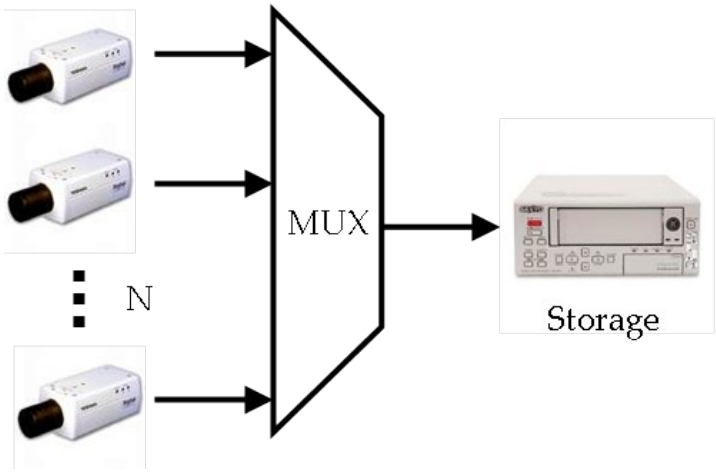


Figura 4: Representação dos sistemas de monitoração digitais.

É simples perceber que além da possibilidade de compressão de dados, utilizando os muitos codificadores já padronizados, os sistemas de monitoração que utilizam tecnologia digital permitem a monitoração simultânea das câmeras, ocupando o espaço de apenas um monitor. Isto é chamado de *MultiView*. Veja a comparação ilustrada na Figura 5.

Figura 5:Comparação entre a monitoração dos sistemas de circuito interno de TV analógica com o MultiView digital

Portanto, o computador pessoal foi escolhido como a plataforma à concepção desse projeto. Os custos das peças que compõem esse projeto também foram avaliados. Com a oferta em larga escala de diferentes câmeras de vigilância e a utilização de um computador com características desatualizadas e, portanto, barato para os dias atuais, foi possível viabilizar a implementação desse sistema. A Figura 6 ilustra esses componentes, essenciais.

Figura 6:Computador e câmera, peças fundamentais para o projeto.

Capítulo 3

Objetivo

O objetivo desse projeto é desenvolver ferramentas que atendam aos requisitos de capturar, armazenar e reproduzir um sinal de vídeo. Essas funcionalidades, como já foram mencionadas, são diretamente aplicadas a sistemas de monitoração. Por isso, é importante que seja criado um ambiente de programação isolado, ou seja, um conjunto de funções reunidas em uma biblioteca que possam ser usadas e adaptadas facilmente, de acordo com as necessidades de cada situação.

Um detalhe importante é que essas alterações sejam feitas na biblioteca sem alterar as características da interface com o usuário. Isto facilita a manutenção e a reutilização das ferramentas.

Como premissa desse projeto, todo o desenvolvimento foi feito com base na linguagem de programação C++, sob a licença GNU GPL [24] e com características multi-plataforma. Isto é, esta solução permitirá que tanto a biblioteca LIBMONITOR

como os programas VideoCapture e Cross-Player, com pequenas adaptações, utilizem sistemas operacionais GNU Linux e Microsoft Windows. Porém, nesse momento, a implementação do projeto será toda descrita com base no sistema operacional GNU Linux. Outra especificação importante é que todo o processamento será restrito apenas para o sinal de vídeo.

Analisando o projeto, chegamos a conclusão que as imagens capturadas devem ser manipuladas seguindo os processos apresentados na Figura 7.

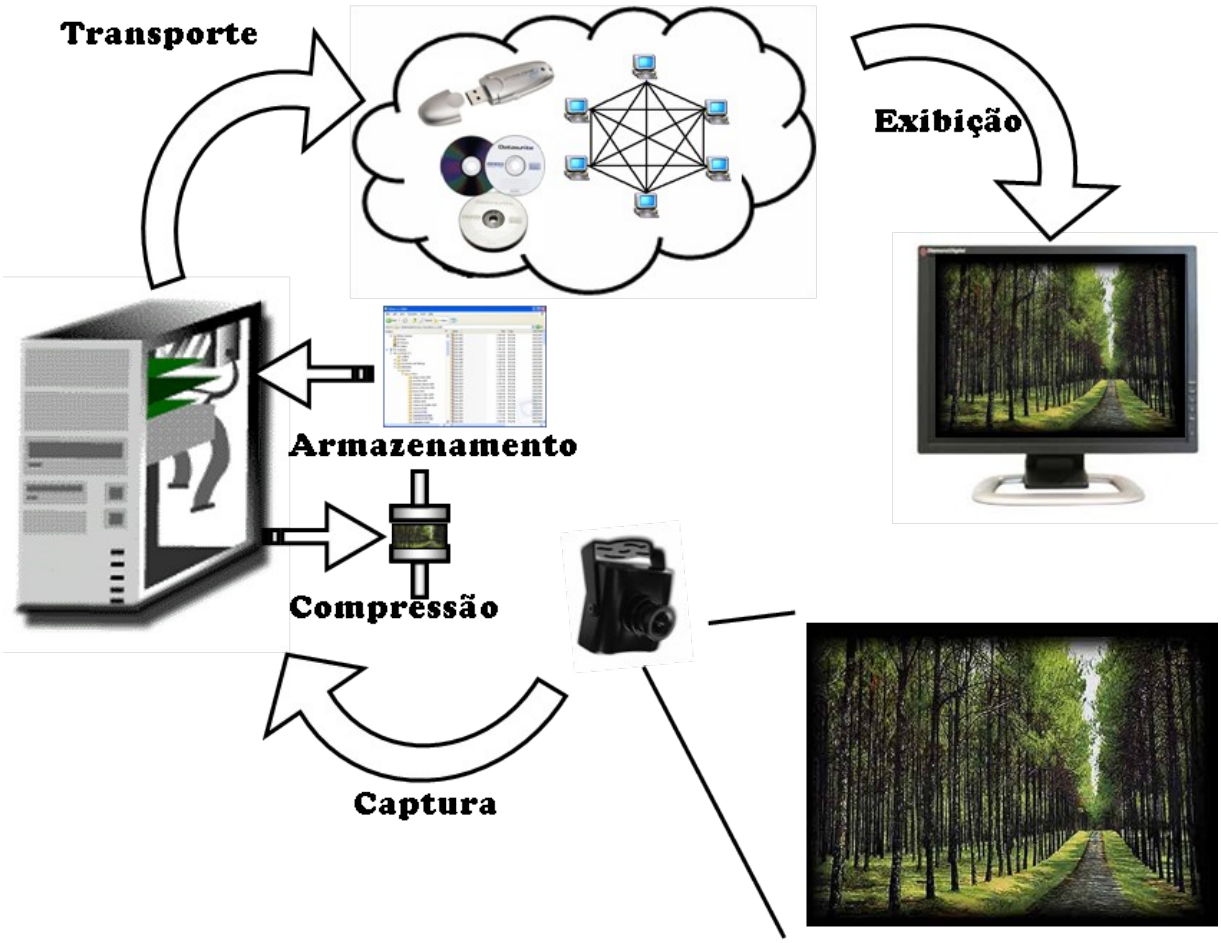


Figura 7: Seqüência da captura até a reprodução.

Capítulo 4

Análise do Problema

Em geral, o vídeo capturado pela câmera é enviado para o computador. A comunicação entre essas duas instâncias é feita pela placa de captura instalada em barramentos ISA ou PCI, ou até mesmo em uma das portas USB, como visto na Figura 08. Essa interface por sua vez, organiza o vídeo em seqüências de quadros não-comprimidos e, através do driver da placa, os disponibiliza num dispositivo acessado via sistema operacional.

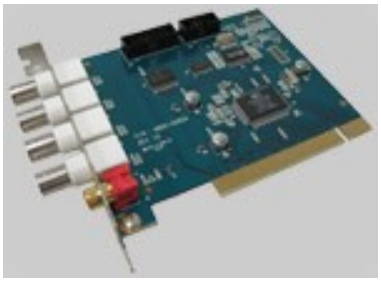


Figura 8: Placa de captura PCI.

Com os quadros armazenados na memória do computador, é iniciado o processo de compressão para que o volume de informação seja reduzido. Logo após o estágio de compressão é que efetivamente armazena-se todo o conteúdo. De posse

do vídeo comprimido, é possível transportá-lo para diferentes locais mais facilmente, utilizando mídias de gravação que atualmente são bastante comuns, como CDs/DVDs, pen-drives, cartão de memória SD, Flash, ou ainda através da transferência de arquivos via rede. Além disso, é necessário que se utilize algum recurso de exibição para que o vídeo seja reproduzido.

Um aspecto importante da compressão com perdas é que essa técnica permite reduzir consideravelmente o volume da informação original com a desvantagem de reduzir a relação sinal-ruído do sinal codificado. Isso significa que quanto mais desejarmos comprimir, ou seja, quanto menor for a taxa de codificação, mais ruído é inserido na informação transmitida. Essa característica é essa diretamente associada à qualidade final do vídeo, por isso é necessário configurar os parâmetros de compressão de maneira que seja alcançada uma qualidade aceitável, levando em consideração também a disponibilidade de armazenamento

De posse dos objetivos e do conceito definidos no escopo, foi estruturado um conjunto de ações para representar cada etapa. A Tabela 1 relaciona as respectivas funções de cada etapa para desenvolver toda a solução do sistema, distribuindo-as em duas fases.

Uma intensa pesquisa foi feita para escolher o hardware necessário no desenvolvimento da solução. Na área de câmeras, as opções eram as câmeras do tipo CMOS (*Complementary Metal Oxide Semiconductor*), CCD (*Charge-Coupled Device*) ou webcam, cuja diferença entre elas está entre o custo, qualidade de captura e resolução. As câmeras CMOS são mais baratas em geral e têm sensores dedicados

para cada elemento de imagem, ou pixel, no quadro, mas com menos qualidade visual. As CCDs são mais caras e podem possuir um sensor que armazena uma imagem de cargas, proporcionais à intensidade de cada pixel, que é lido seqüencialmente. Ela possui um sensor desses para cada componente de cor (vermelho, azul e verde). O custo se eleva ainda mais, pois a lente é normalmente adquirida separada da câmera. Já as webcams têm vantagens por não precisar da placa de captura, pois utilizam conexões do tipo USB ou IP, mas como foram projetadas para capturar imagens a curtas distâncias, não apresentam foco em objetos a longas distâncias. Outro fator relevante é que as webcams IP são, em média, acima de 200% mais caras que câmeras do tipo CMOS.

FASE	ETAPA	AÇÃO
1	CAPTURA	Decodificar o sinal de vídeo.
		Formatar os pixels dos quadros capturados.
	COMPRESSÃO	Ordenar os quadros.
		Configurar os parâmetros de compressão.
	ARMAZENAMENTO	Escolher o local de gravação.
		Nomear o arquivo de vídeo.
2	REPRODUÇÃO	Carregar o arquivo na memória.
		Decodificar o bitstream do vídeo comprimido.
		Exibir os quadros na tela.

Tabela 1: Projeto de desenvolvimento da solução

Para a escolha da resolução do quadro, observou-se que na grande maioria de aplicações de vigilância, utiliza-se a dimensão de 352 colunas por 288 linhas de pixels, comumente chamada de resolução CIF.

O sinal gerado pelo circuito das câmeras de vigilância, em sua grande maioria também, corresponde ao padrão analógico de transmissão NTSC (*National*

Television System Committee), utilizado em países como Essados Unidos, Canadá, Japão, entre outros (Figura 9). Dessa forma, o circuito demodulador da placa de captura deve ser compatível com tal padrão.

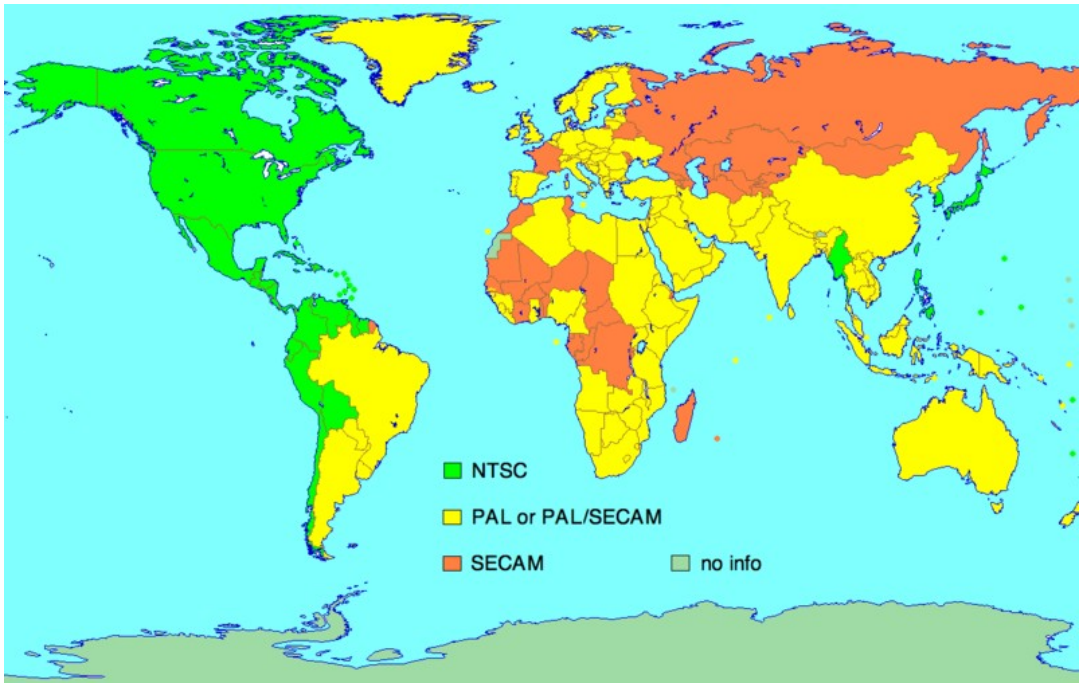


Figura 9: Distribuição dos padrões de transmissão de TV analógica

Seguindo o padrão de televisão adotado nos países das Américas e alguns países da Ásia, foi definido que a taxa de codificação dos quadros seria de 30 quadros por segundo. Essa unidade é mais conhecida como *frames per second* em inglês ou como seu acrônimo fps. Com essa taxa consegue-se ter a sensação de continuidade das cenas, sem interrupções.

Tomando como base, por exemplo, simplesmente a luminância do sinal (vídeo em preto e branco), obtemos: $(352x288) = 101376$ pixels por quadro. Considerando a taxa de 30 quadros por segundo e que cada pixel tem uma resolução de 1 byte ou 8 bits, esse sinal de vídeo atinge a taxa de: $30x101376x8 = 24330240$ bits

por segundo ou 24,3 Mbps. Significa que uma monitoração de apenas uma hora de duração gera um volume de 87,5 Gigabits, ou quase 11 GigaBytes, o que torna inviável o armazenamento desse conteúdo, se o quisermos visualizá-lo em outro momento.

Para solucionar o problema da grande quantidade de bits, usa-se técnicas de compressão. O padrão adotado foi o MPEG-4 Parte 2 [12], graças as altas taxas de compressão que atingem aliada a sua baixa complexidade computacional. Além disso, optou-se por adotá-lo devido às diversas implementações em bibliotecas para computador, seja no sistema operacional GNU Linux ou Microsoft Windows, ou até mesmo para sistemas embarcados que existem para ele. Assim, com essa tecnologia presente nas mais diferentes plataformas, há chances consideráveis de portabilidade da solução.

A pesquisa estendeu-se também pelo conjunto de ações responsáveis por compor o módulo de visualização. Seguindo o processo inverso de captura e codificação, primeiramente é necessário descoprimir o vídeo para que a compressão fosse removida. Assim, com os quadros formatados de acordo com os pixels representados em RGB24 [13], eles eram desenhados no buffer de vídeo. Esse buffer por sua vez era exibido obedecendo a taxa de codificação.

O buffer foi criado com parâmetros que permitem o compartilhamento dos recursos da placa gráfica. Desta forma, os acessos a essa região foram feitos diretamente do hardware para que fosse possível alcançar uma resposta rápida da reprodução. Era preciso também criar recursos de controle sobre vídeo, como

começar, parar, avançar e retroceder, comuns em aparelhos de vídeo cassete e DVD *Players*.

Para otimizar o desenvolvimento e para atender às premissas do projeto, foram pesquisadas diversas bibliotecas auxiliares, evitando que o trabalho de funções básicas fosse refeito. Foram consideradas bibliotecas gratuitas e de preferência multi-plataformas.

Dentre muitas opções para bibliotecas gráficas, como GTK+, usada no desenvolvimento do gerenciador de janelas GNOME; QT, conhecida por ser bastante usada no KDE; e wxWidgets [1], essa última foi a escolhida, pois apresenta uma estrutura de classes, que é semelhante à estrutura do MFC (*Microsoft Foundation Classes*). A MFC é a biblioteca do Windows, usada para criar aplicativos, manipular objetos, janelas, controles, etc, ou seja, o conjunto de classes e funções do sistema operacional mais usado no segmento de desenvolvimento de soluções comerciais.

A wxWidgets foi utilizada para implementar os controles e o gerenciamento das janelas da ferramenta de exibição, o Cross-Player. Seria necessário ainda, adaptar uma janela para que nela fossem desenhados os quadros do vídeo. Para isso acontecer com essabilidade era preciso acessar os recursos de hardware da placa gráfica. Tal conclusão resultou de diversas tentativas de utilização da própria janela da wxWidgets para apresentar os pixels. Na verdade, os quadros eram reproduzidos lentamente e a execução terminava com erros e falhas de maneira aleatória.

Sendo assim, a biblioteca SDL (*Simple DirectMedia Layer*) [4,5] foi a solução encontrada que melhor se encaixava nos requisitos propostos. Ela foi desenvolvida para oferecer acessos, tanto emulados quanto diretos ao hardware, seja nas portas de teclado, mouse e joystick, ou ainda nas placas de áudio, de vídeo 2D ou 3D. Ainda possui versões para Linux, Windows, Windows CE, BeOS, MacOS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX e QNX.

Obrigatoriamente, para que fosse possível desenhar o quadro na memória da placa gráfica, era preciso ter posse do quadro decodificado. Essa foi uma das etapas mais importantes do projeto, pois era preciso selecionar uma biblioteca que decodificasse os pacotes MPEG-4 em tempo de execução, com uma duração menor que o período do quadro $\left(\frac{1}{30} seg\right)$. Tal ação era primordial para que o vídeo fosse reproduzido sem interrupções. Algumas opções foram cogitadas dentre elas a solução Xvid, que é dedicada para codificar e decodificar sinais apenas do padrão MPEG-4. Embora na solução Xvid, os processos de compressão e descompressão fossem rápidos, a solução escolhida foi a libavcodec e libavformat [6,7,8], que acompanham o programa ffmpeg.

Essas bibliotecas, embora sejam bastante usadas em diversos projetos multimídia [25], têm uma documentação muito escassa. Seu principal diferencial é suportar diversos formatos de codificação de vídeo, como H.261, H.263, MPEG 1, 2 e 4, VC-1, além de padrões de áudio, como AAC, MP3, etc. Para uma lista completa e atualizada dos tipos de codecs suportados pela libavcodec, ver [27] para saber sobre vídeo e ver [28] para áudio. A biblioteca libavformat é responsável por dar suporte

aos diferentes tipos de formatos de arquivos. É possível decodificar os sinais encapsulados em arquivos AVI, MPEG-2 Transport Stream, DV, entre outros. A lista completa dos formatos pode ser acessada em [29]

Devemos ser considerar ainda nessa análise a instância responsável por viabilizar a captura. No sistema operacional Linux, a API (*Application Programming Interface*) ou Interface de Programação de Aplicativos, designada para atender a essas especificações é a Video For Linux, que representa o conjunto de funções e chamadas de sistema que essabelece comunicação de leitura e escrita no dispositivo de aquisição. Em outros sistemas operacionais, como o Microsoft Windows, semelhantemente, existe uma API chamada Video For Windows, implementada através da conhecida biblioteca DirectShow.

É através desse tipo de API, que aplicações de alto nível têm acesso a informações de dispositivos de captura de áudio e vídeo. Portanto, a biblioteca, libmonitor, criada para a implementação e uso das ferramentas de captura e visualização utilizou todos os recursos representados na Figura 10.

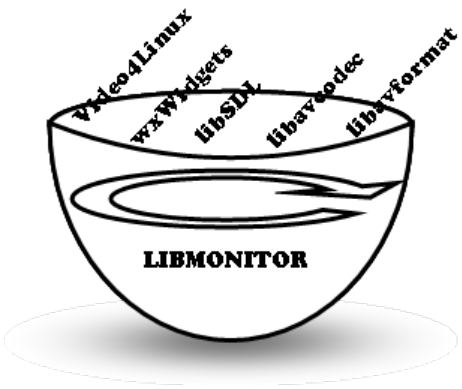


Figura 10: Conjunto de bibliotecas que compõem a libmonitor.

Capítulo 5

Os Recursos

Nesse capítulo serão descritos os recursos de hardware necessários para o desenvolvimento do projeto, bem como as versões de drivers e dos softwares mencionados no capítulo anterior.

Antes da aquisição da placa de captura, foi preciso garantir que o seu funcionamento fosse compatível com os dois sistemas operacionais: Microsoft Windows, e principalmente com Linux, plataforma escolhida para a implementação. O ponto duvidoso era o driver da placa, cuja existência para os sistemas citados seria crítica para a sua definição. Além disso, a placa não poderia entregar os quadros comprimidos, ou seja, realizar a codificação em hardware, o que deveria ser feita em software pelo estágio da compressão já descrito.

É realmente comum produtos de mercado, em se tratando de periféricos para computador, serem lançados com suporte restrito para Windows, com o driver, guias de instalação e manuais escritos apenas para esse sistema. As marcas

analisadas foram, Encore, PixelView e LG, como segue a Tabela 2. Os itens que foram levados em consideração foram:

- Resolução: as dimensões deveriam suportar no mínimo os valores dos sinais padronizados das tvs analógicas atuais (640x480 – NTSC e 768 x 576 – PAL);
- Chipset: para ser compatível com o Linux, segundo [14], a placa deveria ser baseada nos chips da família BTTV, bt848 ou bt878, ou o cx2388x, cx23415, cx23416, todos da Conexant, ou nos chips da Zoran, zr36057, zr36067, zr36120 ou zr36125.
- Captura Comprimida: os quadros capturados não poderiam sofrer nenhum tipo de codificação, salvo o processo de formatação dos pixels, seja em RGB ou YUV [13]. O método de aquisição deverá ser capaz de obter o quadro não-comprimido, através da captura denominada *RAW*.

MARCA	Encore	LG	PixelView	PixelView
MODELO	ENL-TVFM	LTT-200	PlayTV MPEG2	PlayTV Pro Ultra
PADRÃO	NTSC/PAL M e N/SECAM	NTSC/PAL M e N/SECAM	NTSC/PAL M e N/SECAM	NTSC/PAL M e N/SECAM
RESOLUÇÃO	720x480 - 9bits	720x480 - 10 bits	720x480 -8 bits	720x480 - 10 bits
SINTONIZADOR	1x	1x	1x	1x
CVBS	1x	1x	1x	1x
S-VIDEO	1x	1x	1x	1x
ÁUDIO	1x	1x	1x	1x
CHIPSET	CMOS Philips A/D	Philips s/ modelo	Conexant Bt 878	Conexant CX23883
CAPTURA COMPRIMIDA	Não	Não	Sim	Não

Tabela 2: Resultado da pesquisa sobre os dispositivos de captura.

Sendo assim, o único modelo que atenderia as nossas especificações era o da PixelView, PlayTV Pro Ultra, pois os modelos da ENCORE e LG possuem chipsets não suportados pelo Linux. Além disso, o outro modelo da PixelView, PlayTV MPEG2 não permite captura no modo *RAW*, pois a placa realiza captura comprimida dos quadros no padrão MPEG2.

Todos esses dispositivos são produtos comprados por consumidores domésticos, utilizados normalmente para digitalizar as antigas fitas VHS, convertê-las para um DVD, assistir televisão no computador, etc. Atualmente, a placa mais cara pode ser encontrada custando o equivalente à R$150,00 (PlayTV Pro Ultra).

Numa segunda fase de pesquisa, levou-se em consideração as placas já usadas nos sistema de vigilância com número maior de entradas de vídeo e foram selecionadas duas marcas que atendiam os requisitos técnicos: Pico2000 e MiniDVR da TecVoz. Essas placas têm a característica de não possuírem sintonizadores, pois o sinal de vídeo transmitido pela câmera está em banda base. É necessário apenas ter o filtro de canal de 6MHz. A placa Pico2000 tem quatro entradas de vídeo e é importada. A fabricação de novas versões dessa série foi interrompida, portanto, foi descartada comercialmente por encarecer o seu fornecimento e também tecnicamente, por não dispor de melhorias futuras.

Assim, atendendo as especificações descritas, a placa MiniDVR [26] foi escolhida para compor o sistema de monitoração, pois possui o mesmo número de entradas de vídeo que a Pico2000 e mais uma entrada de áudio analógico. Todos os recursos adquiridos para o desenvolvimento do projeto foram custeados pelo Laboratório de Processamento de Sinas – LPS, do Departamento de Eletrônica, tal como a placa de captura e a câmera *VTV Digital security micro câmera VS786C*. A escolha desta câmera, o tipo CMOS e modelo, foi determinada pela indicação do professor Eduardo, pois atendia as premissas de qualidade e preço. As cotações

dessas peças estão no Anexo I e os preços apresentados na Tabela 3. O diagrama básico da instalação dos periféricos é apresentado a seguir, na Figura 11.

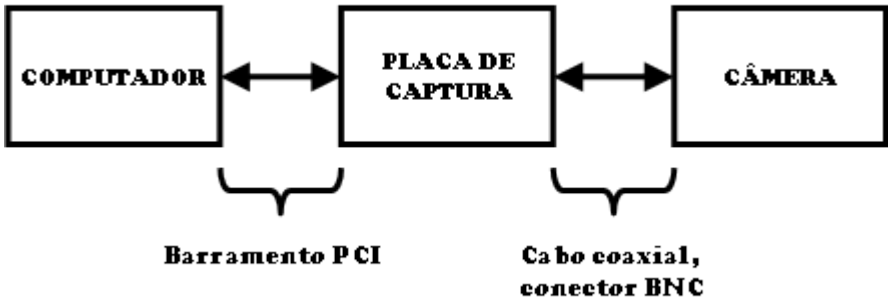


Figura 11: Diagrama de interconexões dos acessórios comprados.

PEÇA	CUSTO
Câmera	R$ 289,00
Placa de Captura	R$ 235,00
TOTAL	R$ 524,00

Tabela 3: Custo das novas aquisições para o projeto.

Para controlar a placa de captura foi usado o driver bttv [16], associado com a API: *Video For Linux* [10], segunda versão, disponível no kernel do Linux. A segunda versão dessa API acompanha as versões do kernel superiores a família 2.6.x. Dessa maneira, foi feito o *download* da versão, 2.6.8.17, disponível em [15].

Para que o novo kernel fosse incorporado nas inicializações seguintes foi necessário recompilar todo o código fonte [11]. Isso demandou um período de pesquisa e conhecimento dos módulos disponíveis para configurar esse novo kernel com suporte à nova placa, para que não houvesse conflito entre os hardwares já existentes no computador, como placa vídeo, de áudio, processador, chipset controladores dos barramentos IDE, PCI e AGP.

Além disso, era preciso conhecer alguns dos protocolos de comunicação que já eram utilizados anteriormente, como os das portas seriais, USB, paralela e interfaces de rede, para que as funcionalidades existentes não fossem interrompidas no novo sistema.

Para configurar o kernel do Linux, é preciso executar um dos comandos:

- make menuconfig
- make xconfig
- make gconfig
- make config

Cada um tem suas próprias características e diferem apenas no formato que apresentarão as informações na tela.

Por exemplo, o método config é baseado no console da linha de comando, é o mais antigo método de configuração do kernel, apresenta as opções em seqüências e é difícil navegar. O menuconfig, também é baseado em console, mas formata as informações com base na biblioteca ncurses que permite desenhar gráficos e estruturas no shell. Isto permite que a navegação pelo teclado seja mais intuitiva e ainda é o mais usado (Figura 12). O xconfig e o gconfig utilizam recursos gráficos do servidor X, abrem janelas de configuração e são os métodos que permitem uma navegação mais fácil através do mouse (Figura 13).

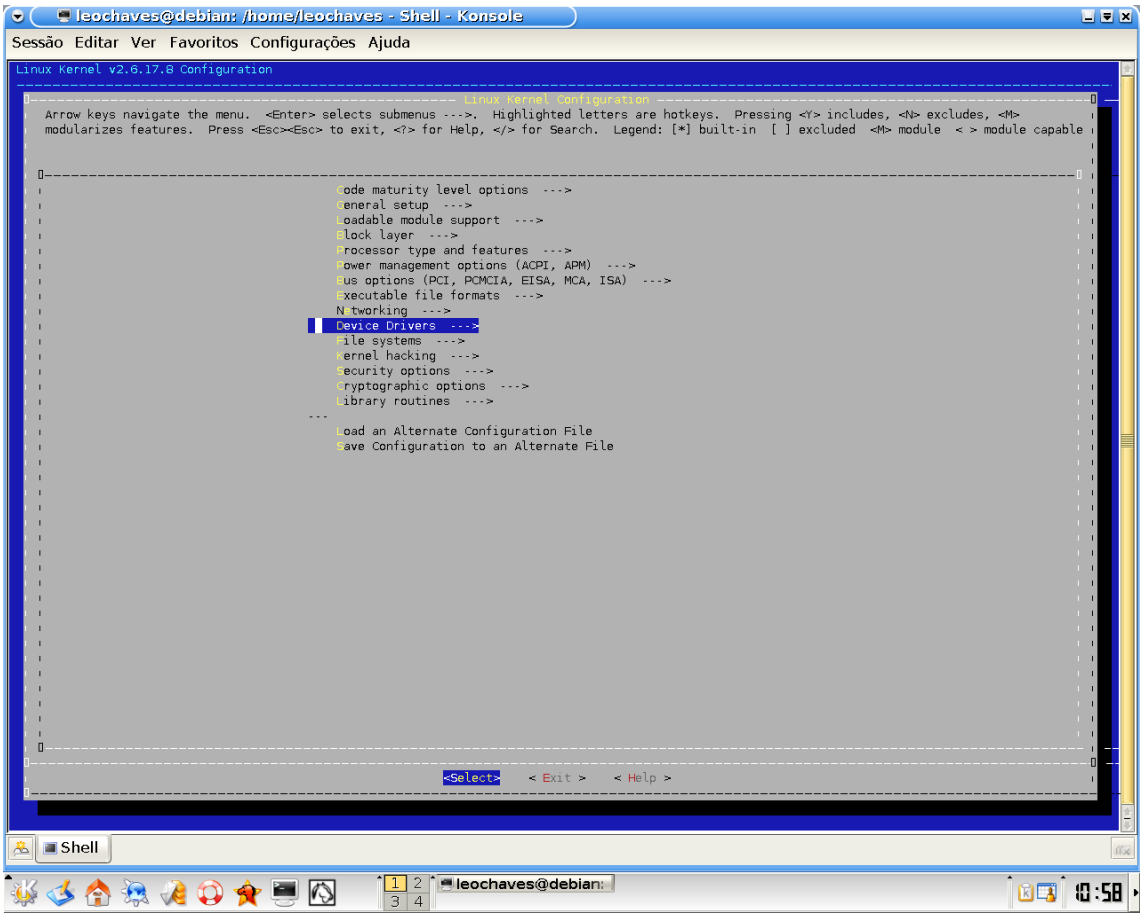


Figura 12: Primeira tela do método menuconfig

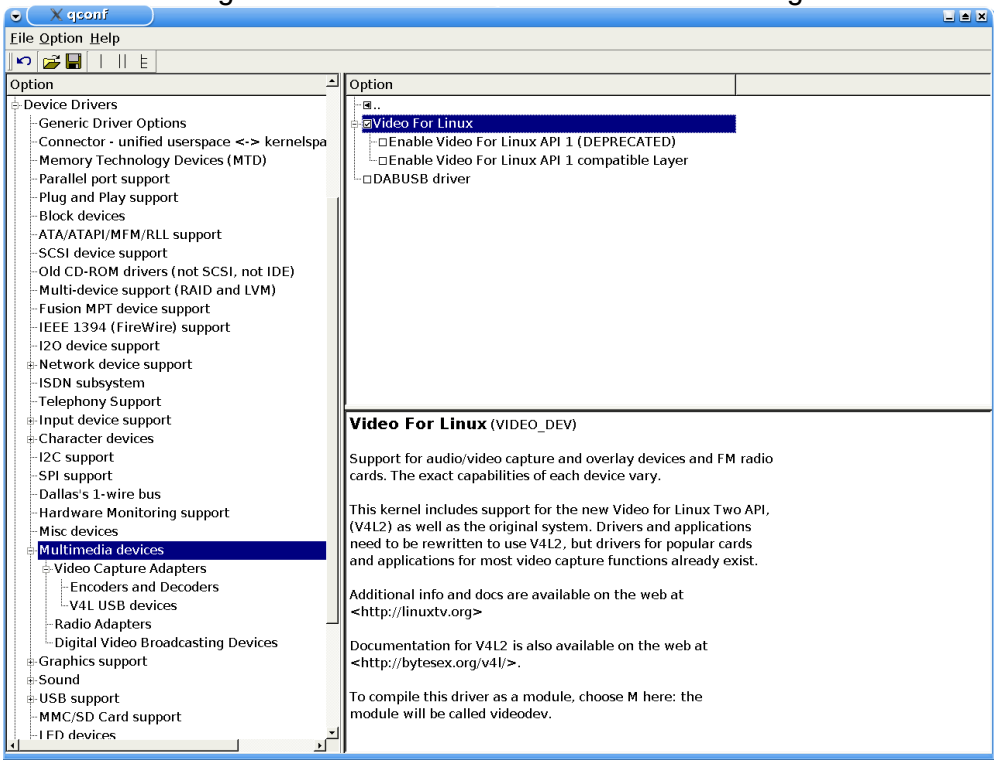


Figura 13: Método xconfig.

É possível configurar as opções como *built-in*, ou seja, integradas ao kernel desde o início do boot e esse driver residirá na memória do computador mesmo que o recurso não seja usado. Outra possibilidade é configurar a opção como módulo, assim, caso o sistema operacional não requisite acesso ao dispositivo, ou ao periférico, o driver em questão não será carregado na memória. Isso diminui o tamanho do kernel e caso as opções sejam otimizadas, escolhidas dedicadamente, para uma configuração específica de PC, alcança-se resultados melhores em velocidade de processamento.

É aconselhável que os recursos necessários e que são normalmente usados, como drivers dos chips da placa mãe, os protocolos de comunicação serial, paralela, TCP/IP, drivers de placas PCI e AGP já instaladas, sejam configurados como *built-in*, pois assim o processo de boot é realizado com maior eficiência e agilidade. Nesse caso, para habilitar a placa de captura, foi preciso seguir pelo menu do kernel denominado Device Driver, depois Multimedia devices e por fim habilitar a opção Vídeo For Linux como *built-in*. Restava apenas escolher que tipo de driver seria compilado para a placa de captura, que nesse caso era baseada no chip BT878A. Essas opções são vistas no sub-menu Video Capture Adapters. Todas essas ações podem ser vistas nas Figuras 14, 15, 16 e 17.

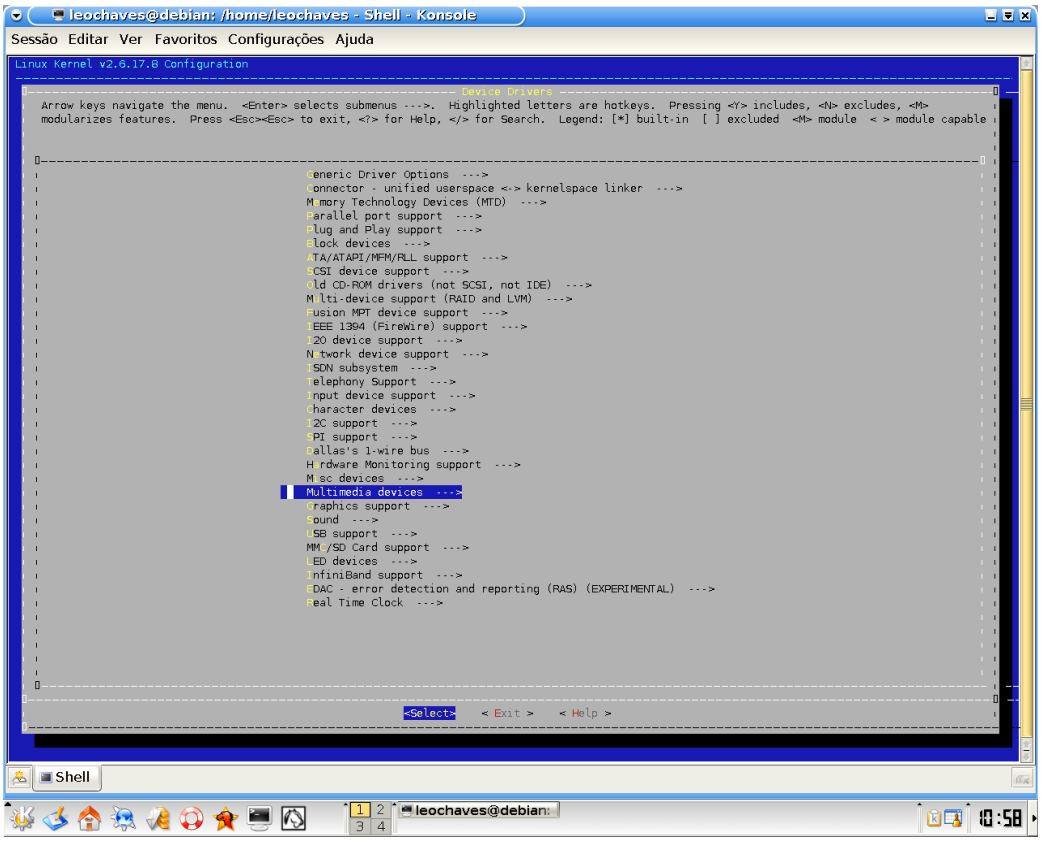


Figura 14: Opção Multimedia devices

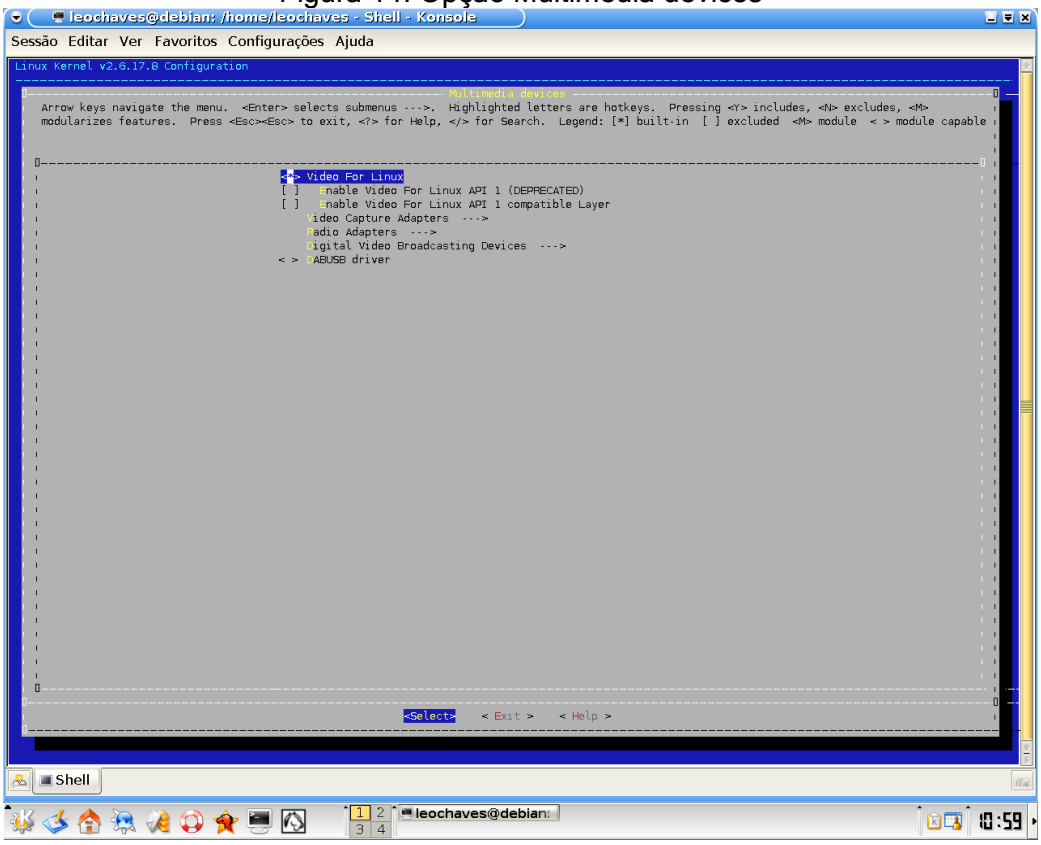


Figura 15: Ativando o Vídeo For Linux

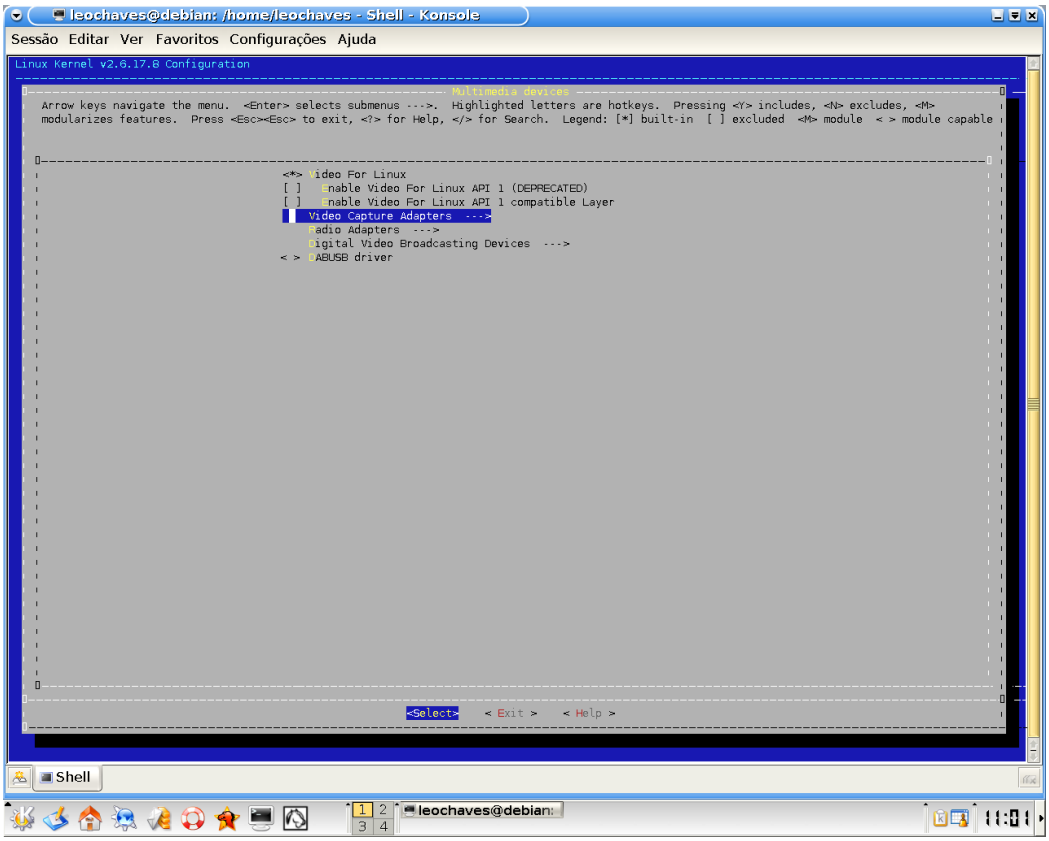


Figura 16: Acessando as opções de drivers

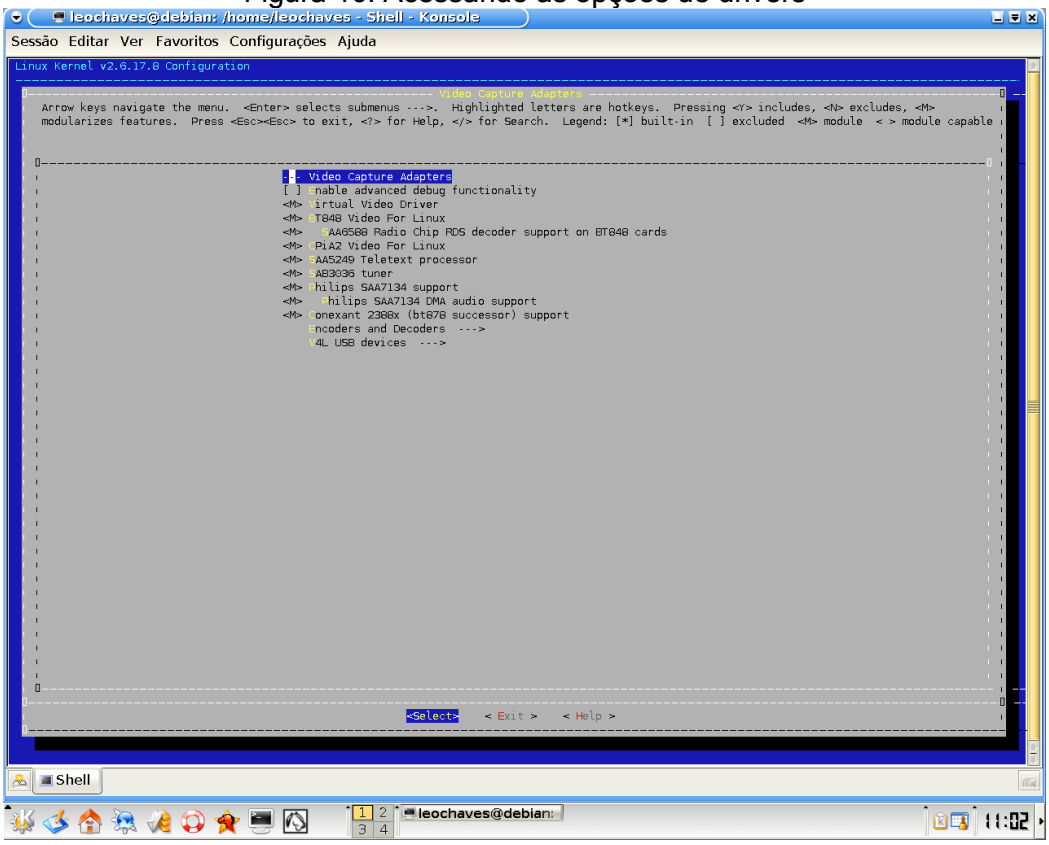


Figura 17: Escolhendo os módulos

Para auxiliar na configuração do kernel, existem três comandos muito utilizados no Linux para saber os dispositivos que sistema reconhece na hora de iniciar:

- dmesg
- lspci
- cat /proc/cpuinfo

Eles identificam as características do computador, todas as marcas de chips e dispositivos que estão instalados, como placas de som, placas de vídeo, controladoras USB. Como era esperada, a placa de captura foi identificada como uma controladora de vídeo baseada no chip BT878, no endereço 0000:00:0a.0, como é visto na linha 8 da Figura 18.

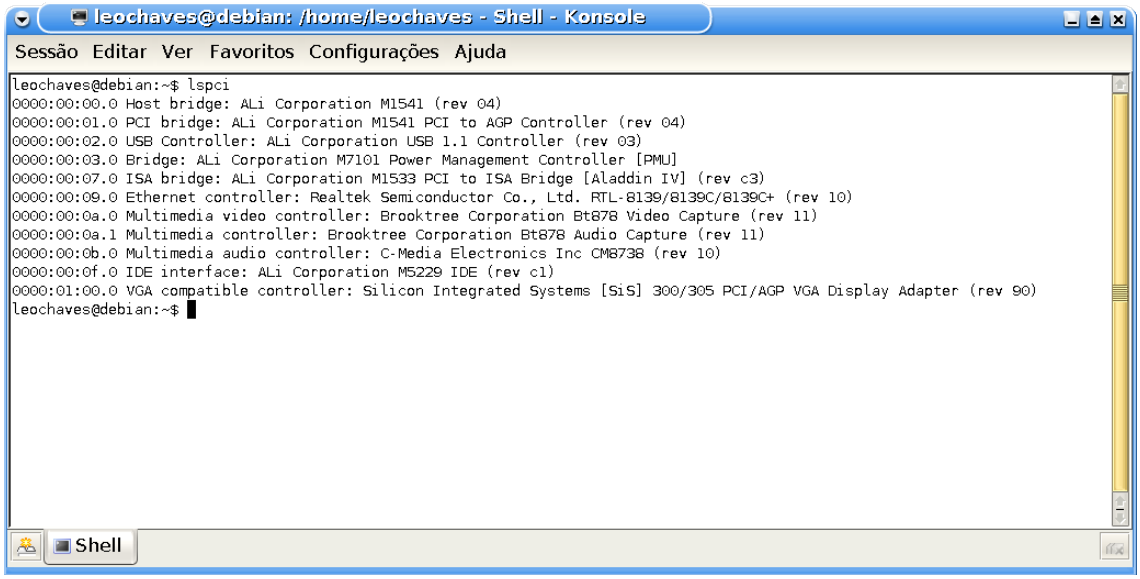


Figura 18: Comando lspci para identificar as placas instaladas no barramento PCI

Para compilar a biblioteca libmonitor, foi construído o script build_all.sh para que não haja necessidade do usuário ter privilégios de administrador. Assim, as bibliotecas auxiliares são compiladas e instaladas dentro dos subdiretórios do próprio usuário. Outro script, chamado build.sh compila somente os arquivos fontes do projeto. A distribuição de arquivos está estruturada conforme a árvore de

diretórios representada na Figura 19, enquanto que as versões das bibliotecas auxiliares usadas nesse projeto estão descritas na Tabela 4.

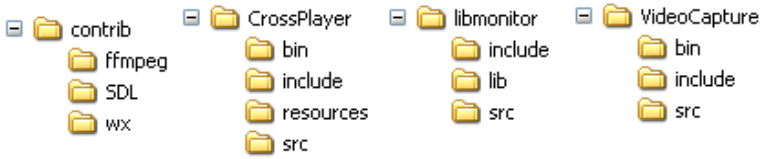


Figura 19:Árvore de diretórios usada no projeto

BIBLIOTECA	VERSÃO
V4L	2.0
wxWidgets	2.6.3
SDL	1.2.11
libavcodec libavformat	Acompanhada na versão 0.4.9-pre1 do FFMPEG

Tabela 4: Lista das bibliotecas auxiliares.

Sob o diretório contrib estão os arquivos das bibliotecas auxiliares, ffmpeg, SDL e wx. No diretório CrossPlayer estão as sub-pastas bin, include, resources e src. Esses locais foram criados para armazenar os respectivos tipos de arquivos: o binário executável; os *.h; as imagens e ícones; e os fontes *.cpp. Os outros diretórios, libmonitor e VideoCapture seguiram as mesmas características, com exceção de libmonitor/lib que armazena os binários da biblioteca, libmonitor.a e libmonitor.so

Capítulo 6

A Biblioteca LIBMONITOR

A biblioteca LIBMONITOR foi desenvolvida com base na linguagem de programação C++ [2], utilizando a biblioteca padrão STL, através da interface de desenvolvimento Anjuta, versão 1.2.2 [17], instalada na distribuição Debian/GNU Linux versão 3.1, conhecida como sarge [18]. Foi utilizado o compilador gcc versão 3.3 [19] e técnicas de programação orientada à objetos como polimorfismo, herança, classes abstratas, bloco *try-catch*, uso de construtores, destrutores virtuais, instruções *inline* e *late-bind* [2].

O computador usado possui as seguintes características: processador AMD K6-2 400MHz, 396 MegaBytes de memória RAM, disco rígido de 10 GigaBytes, sistema operacional Debian/GNU Linux, versão 3.1. A Figura 20 representa como a LIBMONITOR foi idealizada, composta por seus módulos: referentes às duas principais funções de um sistema de vigilância. Essa biblioteca reúne funcionalidades de codificação e decodificação, acesso aos recursos de hardware, controles de entrada

e saída e de gerência das janelas gráficas, de forma independente do sistema operacional. A modelagem e os relacionamentos das classes estão representados na Figura 21, bem como a separação de onde é empregado o uso das funções auxiliares. As linhas pontilhadas delimitam essas camadas.

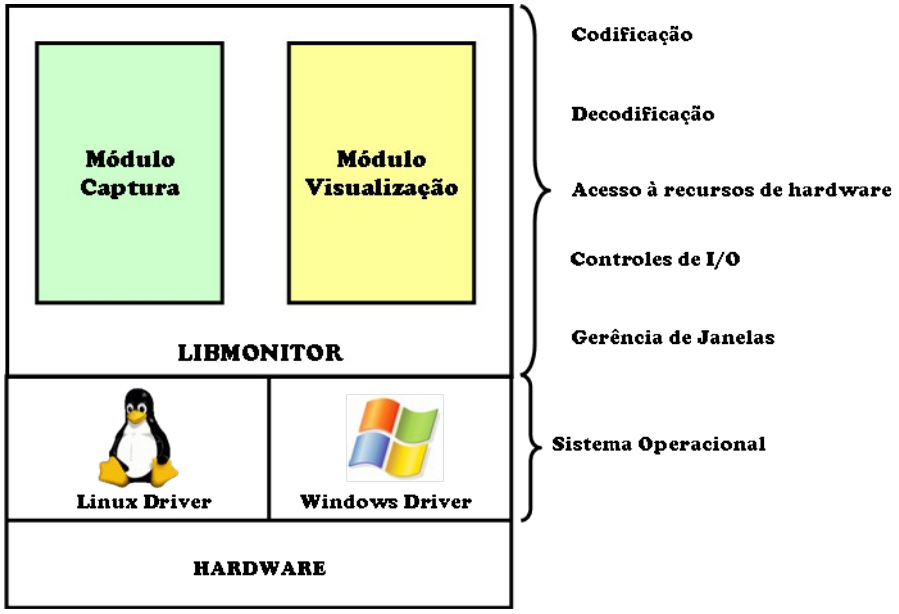


Figura 20: Idealização da LIBMONITOR

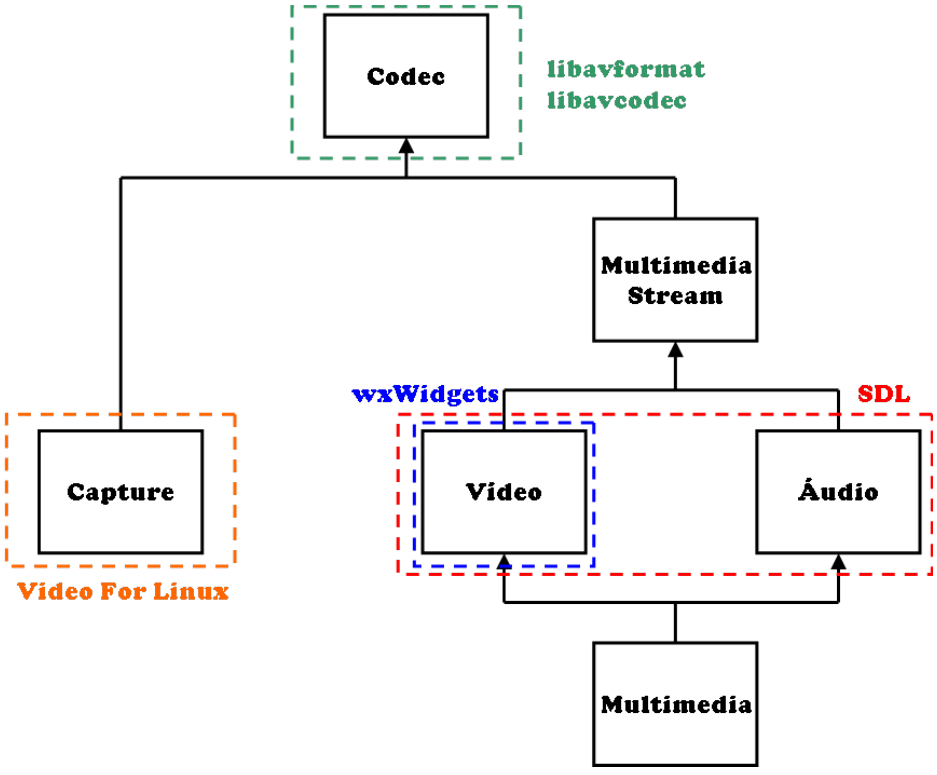


Figura 21: Modelagem das classes

A seguir, serão apresentados os dois módulos em detalhes, com as classes e algumas funções desenvolvidas, bem como a descrição das variáveis criadas. No Apêncie I é apresentado o dicionário da biblioteca libmonitor, com explicações de todas as funções desenvolvidas.

6.1 Módulo Captura

A base desse módulo é, sem dúvida, a API Video For Linux, pois é ela quem, efetivamente, permite o acesso aos dispositivos de captura. As funções dessa API foram definidas para seguir alguns passos básicos, como: abrir o dispositivo, consultar suas propriedades, configurá-las, negociar o formato dos dados, negociar o método de leitura e escrita, o período de captura e o fechamento do dispositivo.

Assim que o driver da placa é carregado na memória, o sistema operacional especifica um caminho e atribui um arquivo para permitir o acesso ao dispositivo. Normalmente, no Linux, esse caminho está no diretório `/dev`, e o arquivo, nesse caso é o `/dev/video0`.

Portanto, a classe Capture implementa, primeiramente, a abertura do dispositivo com o arquivo definido em tempo de execução. Esse método retorna um descritor que é independente por dispositivo. Isto permite que tanto o vídeo quanto o áudio possam ser capturados e identificados por descritores distintos.

A API Video For Linux define estruturas e macros que são usadas na chamada de sistema ioctl (controle de entrada e saída – *Input*/*Ouput*). Essa chamada é utilizada da seguinte forma:

`int ioctl (int fd, int request, void *arg);`

onde **`fd`** corresponde ao descritor, **`request`** é o tipo de requisição da API e **`arg`** é o ponteiro opaco para uma variável cujo conteúdo armazenará as informações, de consultas ou configurações, baseadas na requisição. Essa função retorna o valor -1 caso não tenha sucesso. Por exemplo, para identificarmos o nome do driver que está sendo usado, precisamos executar o seguinte trecho de código:

```
int Xioctl (int fd, int request, void *arg)
{
     int ret;

     do ret = ioctl (fd, request, arg);
     while (-1 == ret && EINTR == errno);

     return ret;
}

int main (int argc, char **argv)
{
     struct v4l2_capabilities cap;
     int fd;

     fd = open ("/dev/video0", O_RDWR | O_NONBLOCK, 0);

     Xioctl (fd, VIDIOC_QUERYCAP, &cap);
```

```
    cout << cap.driver << endl;

    close(fd);

    return 0;
}
```

Todas as requisições da API estão definidas no arquivo de protótipos <videodev2.h> e, particularmente, as de configuração são baseadas na sintaxe *Get-Set*, como pode ser vista nesse exemplo de seleção do formato dos dados:

```
int ioctl (int fd, int request, struct v4l2_format *arg);
```

- VIDIOC_S_FMT;
  - Define o formato da captura com informações apontadas por *arg.
- VIDIOC_G_FMT;
  - Retorna os valores usados na captura no endereço apontado por *arg.

Devemos ler esses exemplos desta forma:

VIDIOC_G_FMT = VÍDEO

VIDIOC_G_FMT = ENTRADA E SAÍDA, do inglês *Input Output*

VIDIOC_G_FMT = CONTROLE

VIDIOC_G_FMT = *GET*

VIDIOC_S_FMT = *SET*

VIDIOC_S _FMT = FORMATO

É na estrutura v4l2_format, por exemplo, que podemos definir ou consultar a dimensão da tela capturada e o formato dos pixels, se RGB 32 bits, RGB 24 bits, YUV

4:2:0 ou 4:2:2 [13], etc. Nesse caso, a captura dos quadros foi implementada utilizando o formato YUV 4:2:0. Outras macros de consulta, como a usada em VIDIOC_QUERYCAP, não usam a sintaxe *Get-Set*. A declaração completa da classe Capture está no Apêndicie I.

A classe Capture implementa o método de leitura baseado na função `read` definida no arquivo <unistd.h>. Seu protótipo é bem simples como pode ser visto abaixo:

```
ssize_t read (int fd, void *buf, size_t count);
```

onde nota-se que `count` refere-se ao número de bytes lidos do arquivo identificado pelo descritor `fd` e o conteúdo é armazendo na variável `buf`. Porém, existem outros métodos de I/O definidos na API Video For Linux e os dispositivos de captura devem ser capaz de executar pelo menos um. O método apresentado é muito comum e normalmente é suportado por diversos dispositivos. Outros no entanto, como *memory mapping*, deve passar por um processo de negociação. No capítulo 3 da referência [10] podem ser analisadas com mais detalhes as diferentes características dos métodos disponíveis.

Cada quadro capturado é submetido ao método de compressão, cuja implementação está na classe Codec. Portanto, o Módulo Captura é composto pela classe Capture para ações de Inicialização, configuração e consultas no dispositivo e pela classe Codec para as ações de codificação (Figura 22). As instruções de como estruturar o formato final do arquivo de vídeo, com os devidos identificadores e cabeçalhos, foram estudadas com base no arquivo fonte `output_example.c` do programa FFMPEG.

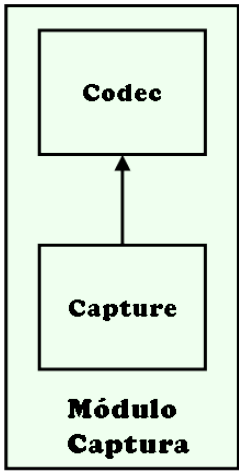


Figura 22: Classes do Módulo Captura

6.2

Módulo Visualização

Esse módulo é responsável por oferecer os métodos necessários para que o desenvolvimento de um programa capaz de gerenciar e reproduzir vídeos e/ou áudios seja fácil e rápido. Para isso, esse módulo foi construído com uma classe que identifica o formato do arquivo e o tipo de codificação do *stream* para extrair a compressão. Outra para fazer a interface de entradas e saídas do módulo e verificar os tipos dos conteúdos presentes, se áudio-visual ou dados, outra que define métodos específicos para as mídias de vídeo e de áudio como desenhar um pixel na tela, e por fim, uma estrutura que implementa funções de controle do que será reproduzido, como avançar, parar, iniciar, etc. Essas classes são respectivamente, Classe Codec, Classe MultimediaStream, Classe Vídeo, Classe Áudio e Classe Multimedia.

É importante ressaltar que nesse módulo é possível criar um *framebuffer* de vídeo na memória da placa gráfica e desenhar nele o quadro desejado. Dessa maneira, essa imagem é simplesmente disponibilizada para o programa principal para que seja representado em uma janela. O método `Show()` é o responsável por isso e está dividido em quatro etapas:

- Durante o processo de modificações no *framebuffer* é preciso executar um tipo de proteção recomendada pela biblioteca SDL. É uma ação de restrição de acesso por outra instância. A função chama-se `SDL_LockSurface()`
- É preciso converter o formato do quadro decodificado para a representação desejada. Nesse caso foi usado RGB24 (8 bits para R, 8 bits para G e 8 bits para B) [13].
- Agora é necessário copiar os valores dos pixels para o *framebuffer*. Foram criados os métodos `DrawPixel()`, `DrawLine()` e `DrawFrame()`, para que o processo de desenhar fosse ituitivo.
- Depois é preciso atualizar o *framebuffer* para validar os novos elementos da imagem: `SDL_UpdateRect()` e finalmente, destravá-lo: `SDL_UnlockSurface()`

O material de estudo que foi consultado para saber sobre o processo de decodificação teve base no artigo de Martin Böheme [7]. Esse artigo declara quais as funções que devem ser utilzadas. Foi visto que, primeiramente, é preciso registrar todos os padrões de *codecs* que a libavcodec suporta. Isto é feito com o método implementado, RegisterAll() da classe Codec. Depois é preciso abrir o arquivo com o método `OpenStream()` definido na classe MultimediaStream e desse modo, o

atributo de Codec, `m_fcx`, recebe toda a informação do arquivo, como por exemplo, quais os tipos e quantos são os conteúdos . Uma vez definido o conteúdo que será exibido, procura-se o codec específico com o método FindDecoder(). Num processo de loop, é feita a leitura parcial do arquivo, selecionando os pacotes apenas do conteúdo escolhido. Então, esse pacote é submetido ao processo de decodificação implementado no método DecodeVideo() que retorna o quadro sem compressão.

É importante ressaltar que a classe MultimediaStream implementa métodos que utilizam contenedores STL. Especificamente o tipo *list* [2] é empregado para resolver os casos de aleatoriedade para acessar números variados de conteúdos de acordo com o arquivo de origem. Dessa forma, podemos manipular o número de vídeos, áudios e legendas que possam existir, de qualquer arquivo multimídia. Para ajudar nesse controle, foi criado a estrutura MultimediaDescriptor (AnexoII) que consegue descrever os tipo de conteúdo já mencionados acima. Portanto, com o método `GetMultimediaList()` é possível passar no seu parâmetro o tipo de mídia que deseja-se pesquisar, e o retorno trará um container com informações dos *streams* desejados. Tal informação é detalhada no Anexo II na Classe MultimediaStream.

Uma outra qualidade importante desse módulo é que de posse de todas as informações dos *streams* presentes, é possível escolher facilmente quais deverão ser reproduzidos. Essa função está implementada no método `SetPlayFlags()`, também da classe MultimediaStream, que através dos parâmetros permite definir os identificadores dos *streams* nas variáveis currentVideoStreamIndex, currentAudioStreamIndex, currentSubtitlesStreamIndex. Esse identificadores são

responsáveis por determinar qual o valor da variável m_playFlags, que informa se será reproduzido um conteúdo de vídeo com ou sem áudio, com ou sem legenda, somente áudio, etc.

Essa informação é feita através da operação lógica OU. Foi definido que o tipo VIDEO tem o valor 1, o tipo AUDIO tem o valor 2 e o tipo SUBTITLE tem o valor 4. Portanto, se quisermos exibir um vídeo sem áudio mas com legenda vamos ter: `m_playFlags` = VIDEO⊕ SUBTITLES. Isto representa 1 0 1 em binário, ou em C/C++ essa operação é feita usando a tecla "|". Para remover essa codificação e saber o que será reproduzido é simples, basta realizar a operação lógica E com cada tipo desejado, ou seja, para tessar se algum áudio será tocado devemos verificar assim:

```
if (m_playFlags&AUDIO)
     cout << Some audio will be played << endl;
else
     cout << None audio will be played << endl;
```

6.2.1

A 1ª versão

Na sua primeira versão, o módulo de visualização foi desenvolvido com as seguintes classes: Resolution, Multimedia, Video e Audio. O relacionamento entre elas corresponde o apresentado na Figura 24. Dessa maneira, as classes Vídeo e

Áudio herdam os métodos e variáveis de Multimedia, que por sua vez herda de Resolution.

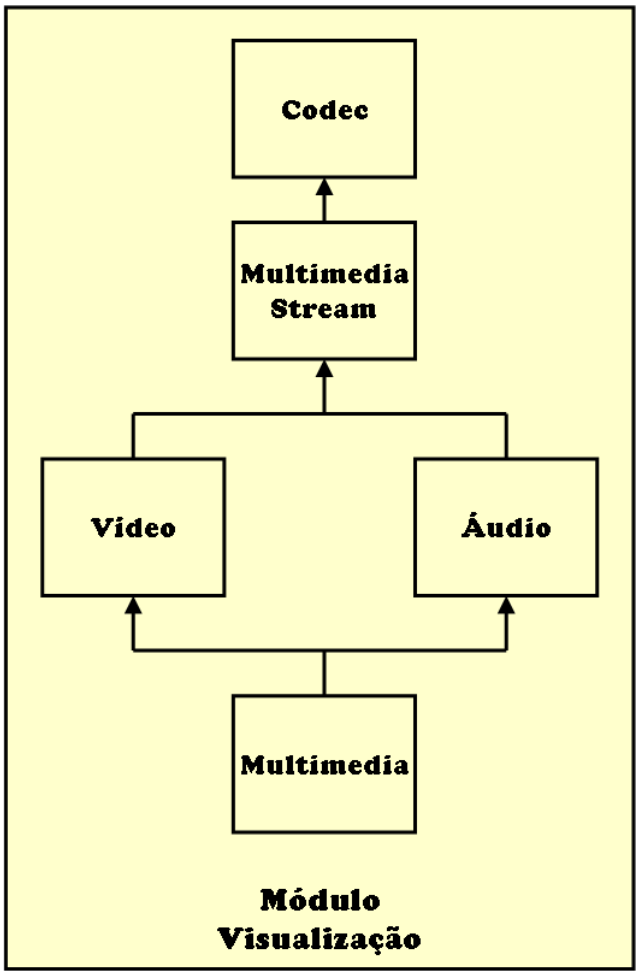


Figura 23: Classes do Módulo Visualização

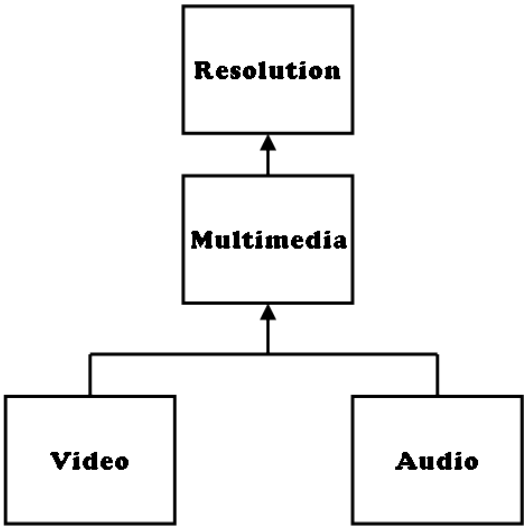


Figura 24: Relacionamento de classes da 1ª versão

A classe Multimedia foi implementada como classe abstrata [2], pois alguns métodos são puramente virtuais, o que nos permite associar o tipo de mídia que será reproduzida em tempo de execução, realizando o que é conhecido como *late-bind* [2]. Nessa versão, não foi feita nenhuma associação com o gerenciador de janelas, portanto o vídeo é reproduzido diretamente no *framebuffer*.

Além disso, sem a capacidade de controlar a dimensão do quadro e sua posição na tela do monitor, não conseguimos também associar um sinal de vídeo com áudio (Figura 25). Embora não estivesse dentro do escopo do projeto, considerar a presença de um sinal de áudio significa preservar a modelagem final das classes.

A linguagem em C++ e o paradigma da programação orientada ao objeto nos conduzem a relacionar as classes de forma hierárquica, seguindo o modelo de classe base e derivada. Assim, reunimos na classe base as principais funções que são comuns às classes derivadas, e essas por sua vez, reúnem as especialidades de cada classe.

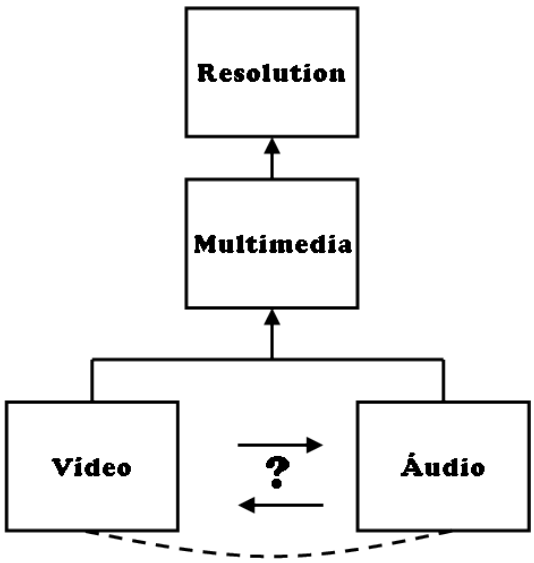


Figura 25: Vídeo sem associação com o áudio.

6.2.1

A 2ª versão

Na segunda versão, novas classes foram criadas e um novo relacionamento entre elas foi essabelecido, para atender a reprodução de vídeo e/ou áudio e/ou legenda (Figura 23). Com isso, novos métodos e atributos foram criados para atender a essa nova adaptação (Anexo II). A função de exibição será executada de forma diferente, toda a imagem gerada no buffer de vídeo será copiada para uma área de controle do gerenciador de janelas, fazendo uma interface entre as bibliotecas libSDL e wxWidgets [9].

A nova classe Video foi estruturada para conter todos os atributos que antes essavam na classe Resolution, assim como o acesso a esses valores de leitura e escrita. Essas variáveis informam, por exemplo, a dimensão do quadro, a taxa de reprodução (quadros por segundo) e a característica do quadro, se entrelaçado ou progressivo.

A classe Multimedia, agora, tem capacidade de reproduzir o(s) *stream(s)* definido(s) pelo atributo `m_playFlags` da classe MultimediaStreams. Ou seja, reproduzirá o video, áudio e legendas ou qualquer combinação entre eles. Porém as funcionalidades de áudio e legendas não essavam no escopo do projeto, mas a biblioteca já está preparada para o futuro desenvolvimento dessas funcionalidades.

Capítulo 7

As Ferramentas

7.1

VideoCapture

Esse programa foi desenvolvido com o objetivo de capturar, através da placa MiniDVR, um sinal de vídeo NTSC no formato RAW e submeter quadro a quadro ao processo de compressão seguindo o padrão MPEG-4 Parte 2. Nessa primeira implementação a interface desenvolvida é baseada em linha de comando. Isto facilita a execução e o acesso remoto se considerarmos que não é necessário que um gerenciador de janelas do Linux seja aberto, como por exemplo, o KDE. A Figura 26 apresenta o fluxograma básico do processo de captura.

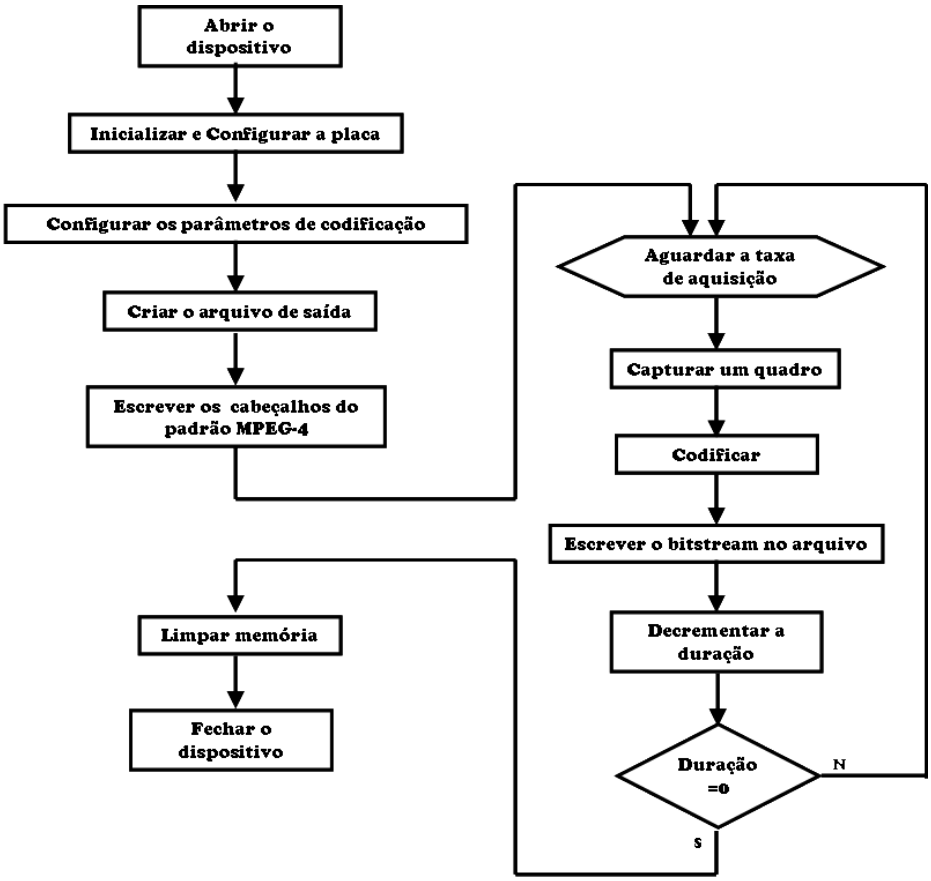


Figura 26: Fluxograma do processo de captura.

7.1.1

Funcionalidades

Ao executar o programa, é possível escolher as seguintes opções:

- Dispositivo: para capturar o vídeo, o driver bttv foi configurado para acessar `/dev/video0`.

- Tamanho da imagem: deve-se escolher o tamanho da imagem original, informando o número de linhas e de colunas.

- Arquivo de saída: o usuário informa o nome do arquivo de saída do programa.

- Taxa de codificação: esse parâmetro informa quantos bits por segundo o *stream* será codificado. Ele é essencial para definir o tamanho e qualidade do vídeo comprimido.

- Taxa de quadro (fps): freqüência com que os quadros serão reproduzidos.

- Duração: Número total de quadros que serão capturados.

7.1.2

Telas

Como o programa VideoCapture é baseado em linha de comando, esta seção não se aplica neste caso para demonstrar a execução. É apresentado abaixo, na Figura 27, apenas com méritos ilustrativos a janela de um terminal gráfico do KDE, com a sintaxe do comando.

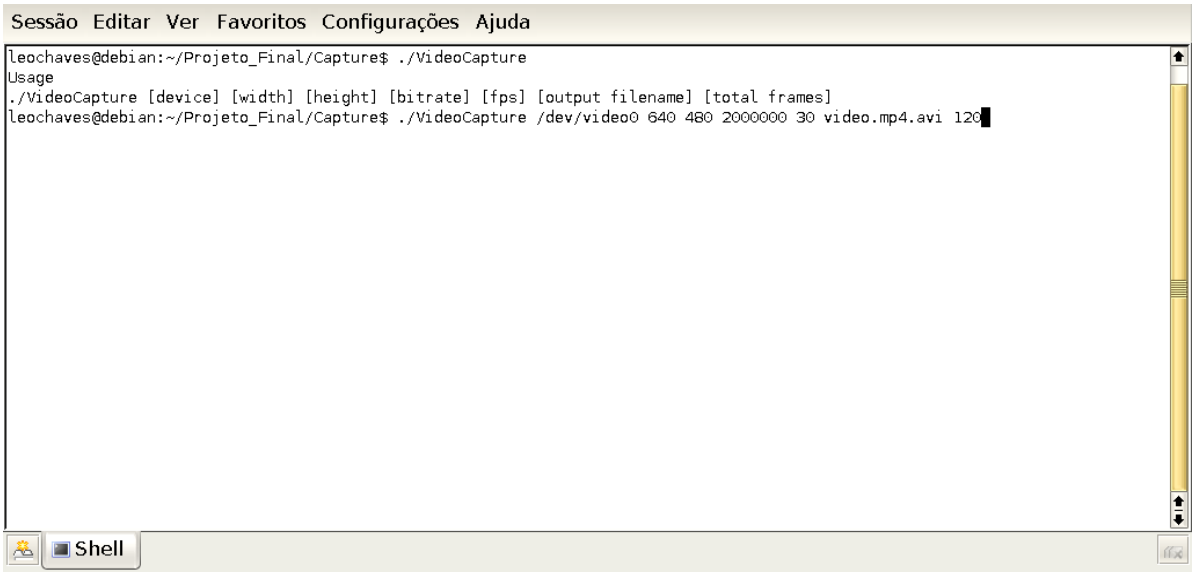


Figura 27: Sintaxe do comando

7.1.3

Execução

A execução é bastante simples. Quando o comando VideoCapture é digitado sem o número e ordem certa dos parâmetros, o programa tem um controle de erros que evita começar o processo de captura e assim, apresenta para o usuário a maneira correta:

```
./VideoCapture [device] [width] [height] [bitrate] [fps]
[output filename] [total frames]
```

7.2

CrossPlayer

A ferramenta CrossPlayer é mais complexa pois envolve diversos controles e processos simultâneos, executados com troca de mensagens entre as instâncias criadas. O controle dessas mensagens é feito pela própria biblioteca wxWidgets, bem como alguns eventos que são transparentes para o desenvolvedor. Existem outros que são precisos controlar, como por exemplo, eventos gerados por botões e acessos ao menu. Os eventos são declarados através da macro DECLARE_EVENT_TABLE [1] definida no ambiente wxWidgets.

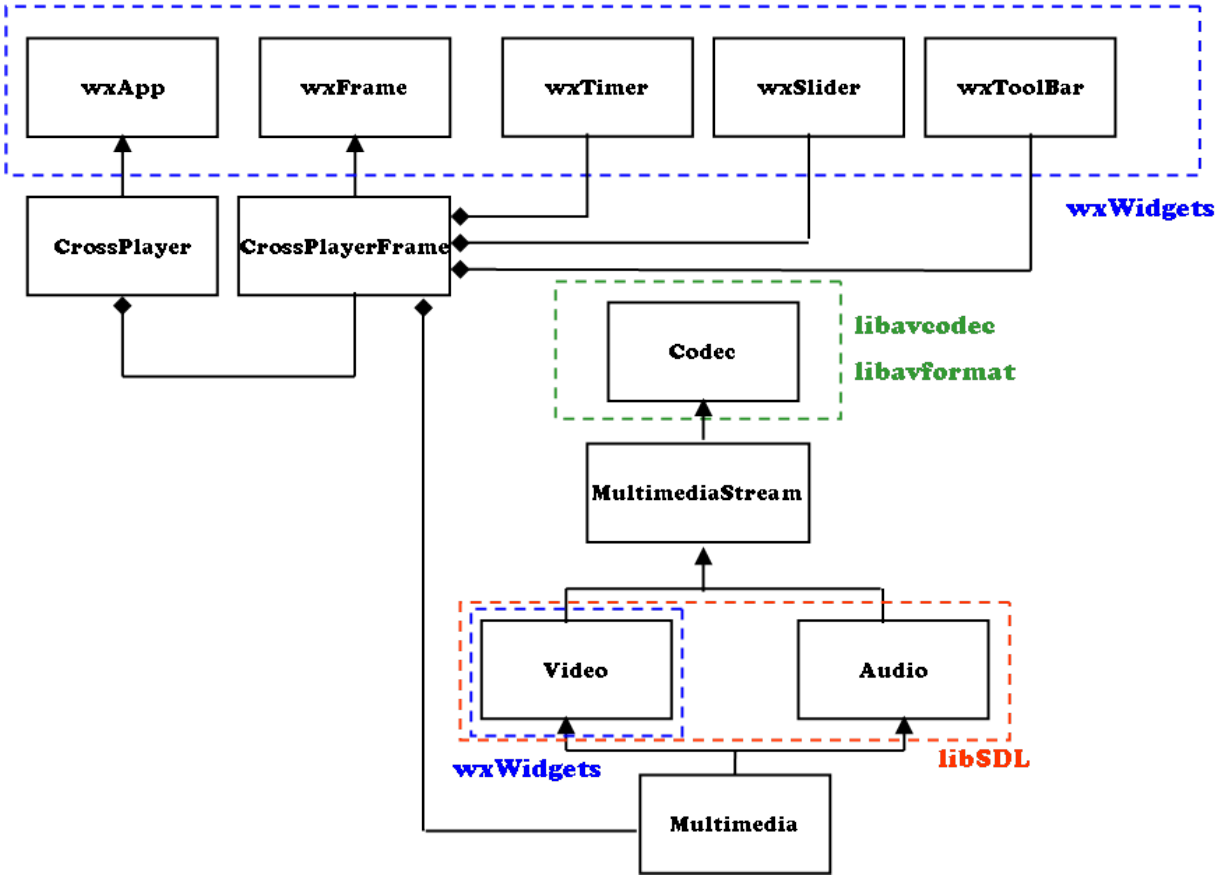


Figura 28: Classes no Cross-Player

Como pode-se observar na Figura 28, a classe CrossPlayer se relaciona com apenas duas classes, herdando de wxApp e contendo CrossPlayerFrame. A classe wxApp é responsável por implementar o loop principal que ficará ativo até a aplicação fechar e a função main(). A classe CrossPlayerFrame por sua vez, herda de wxFrame que corresponde a janela que será apresentada na hora da execução. Essa é a área reservada para organizarmos todo o acesso a aplicação, com menus, controles, etc.

Podemos notar que estão contidos na classe CrossPlayer objetos do tipo wxTimer, wxSlider, wxToolBar e Multimedia. Os objetos dessas classes representam respectivamente, o gerador do tempo para reproduzir o quadro igualmente a taxa codificada; o controle da posição do *stream* semelhante a uma barra de progressão; área onde ficarão os botões e o slider; e o objeto da classe Multimedia é responsável por manipular o arquivo de entrada. Segue um resumo:

- wxApp – classe da aplicação
- wxFrame – classe de gerência de janelas
- wxTimer – classe do temporizador que gera o evento para disparar o processo de exibição de um quadro.
- wxSlider – classe que emite o comando de controle dos quadros exibidos (barra de progressão)
- wxToolBar – classe que acomoda os botões de Play, Pause, Stop, FastForward, Rewind e Infos.

Dentro do diretório CrossPlayer/resources estão os arquivos usados para ilustrar os botões e preencher o desenho do ícone do programa. Convenientemente foi escolhido uma representação de *color-bar* para ser o ícone. A montagem da

interface do programa é feita através da função `Build()` que basicamente realiza as funções descritas no fluxograma da Figura 29. A interface final pode ser vista na Figura 30.

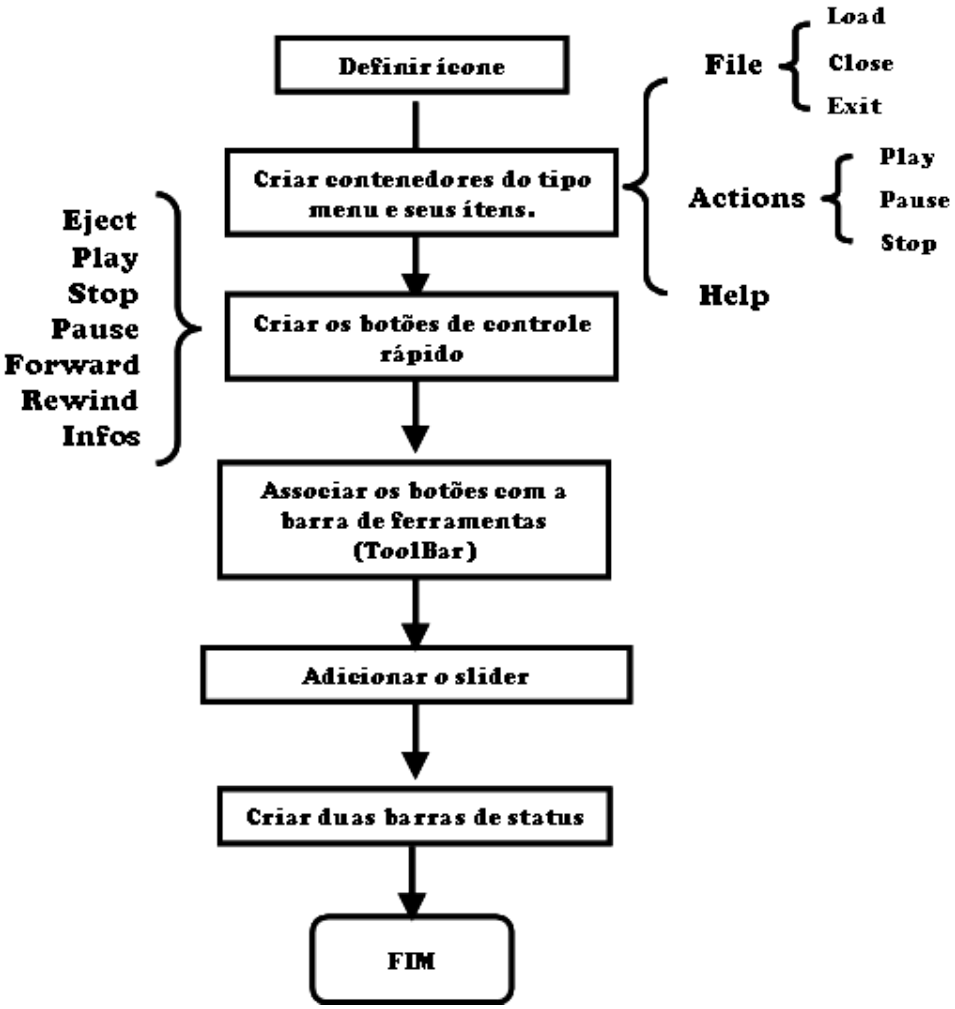


Figura 29: Fluxograma do processo de construção da interface.

Figura 30: Interface do CrossPlayer.

A seguir são apresentados os fluxogramas das ações *Load()* e *Play(),* nas Figura 31 e 32 respectivamente.

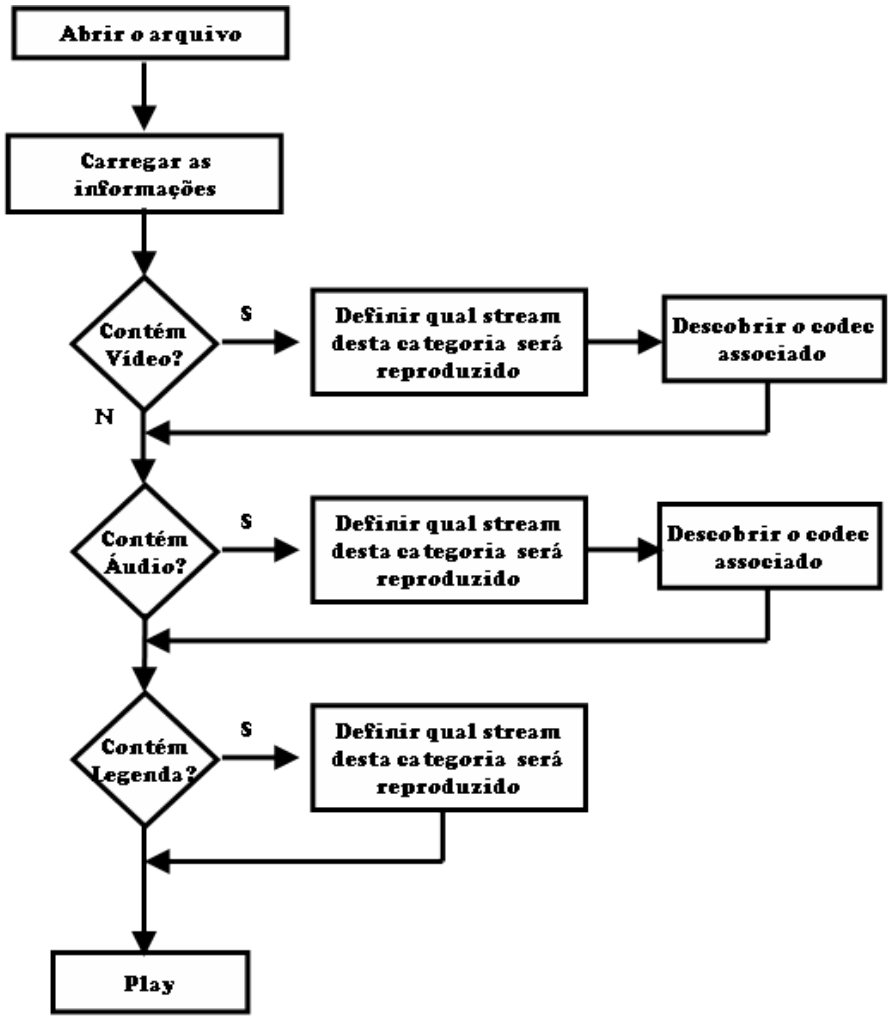


Figura 31: Algoritmo do processo de carga do arquivo

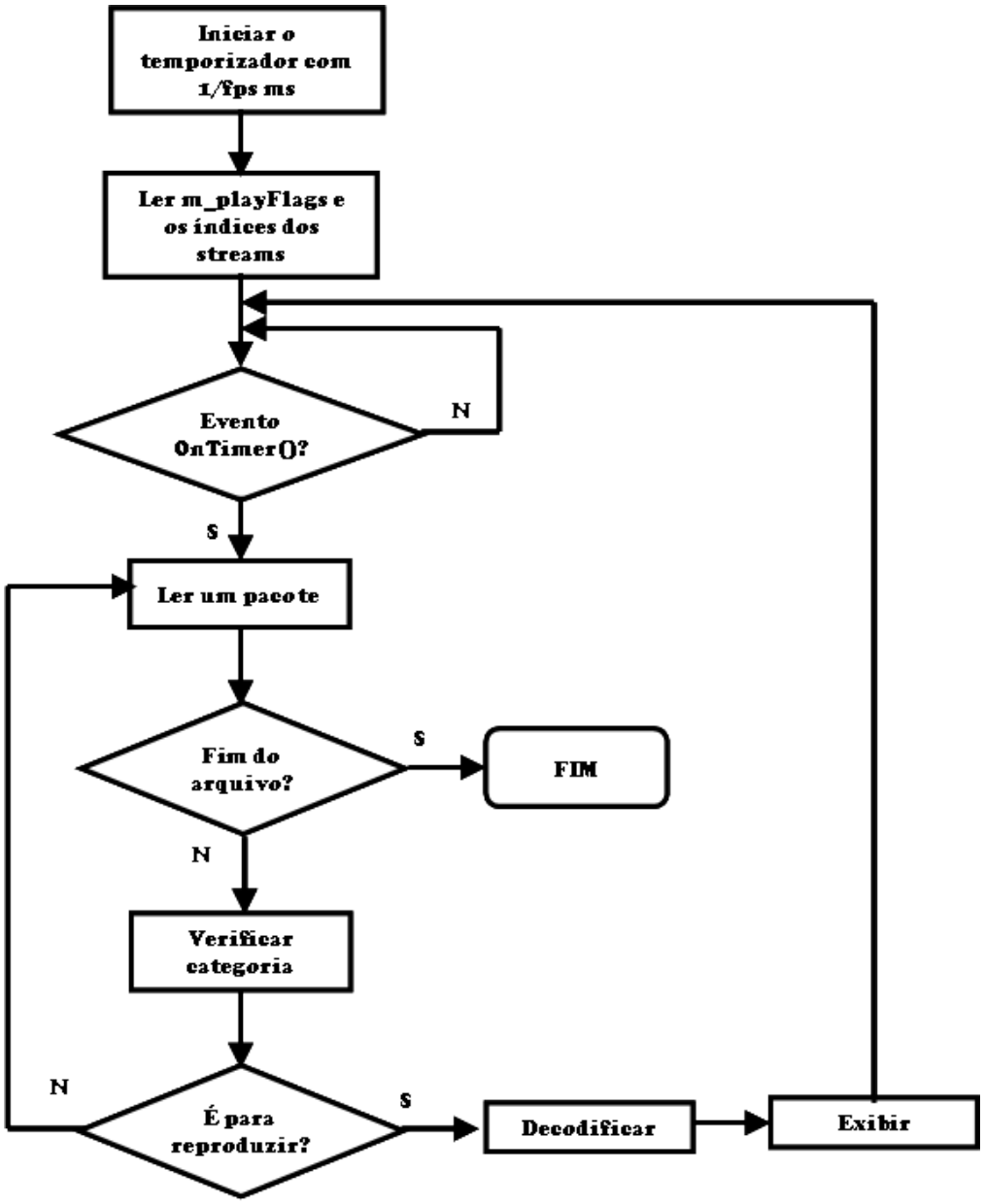


Figura 32: Algoritmo do processo de reprodução

.

7.2.1

Funcionalidades

- Load

  Carrega o arquivo multimídia

- Play

  Reproduz o(s) conteúdo(s) escolhido(s)

- Stop

  Pára a reprodução do(s) conteúdo(s)

- Pause

  Igual ao Stop

- Busca com *slider*

  posiciona a execução de acordo com o número total de quadros.

7.2.2

Telas

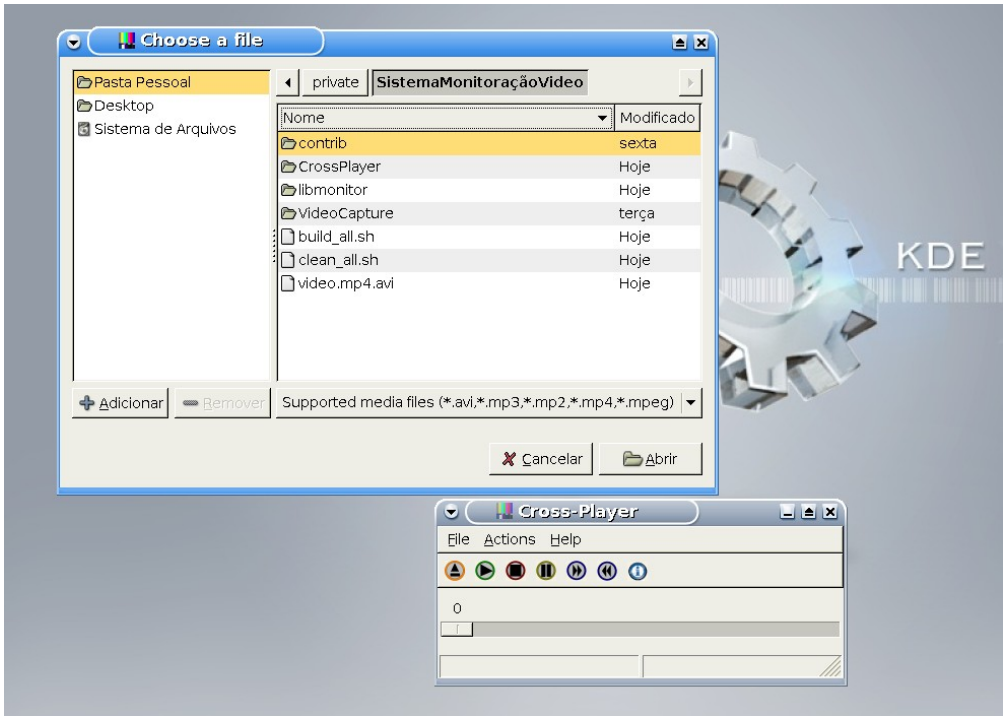


Figura 33: Tela de diálogo para abertura do arquivo multimídia.

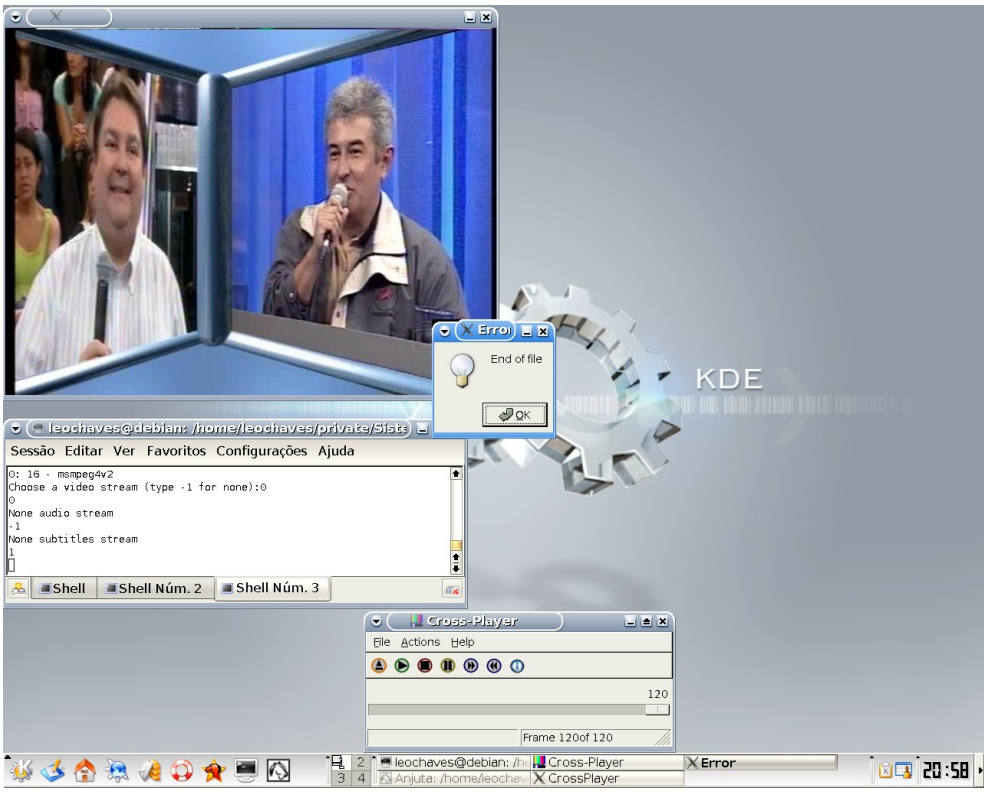


Figura 34: Durante uma reprodução

7.2.3

Execução

Durante a execução, na barra de status e na de progressão (*slider*), é indicado o número atual do quadro que está sendo exibido. A imagem da Figura 34 demonstra o momento durante uma execução do CrossPlayer.

Capítulo 8

Outras Plataformas

Os requisitos de portabilidade deste projeto foram considerados desde a sua concepção. Isto significa que diferentes plataformas, baseadas em computadores pessoais, como os sistemas operacionais Linux ou Microsoft Windows, permitem usar o sistema de monitoração desenvolvido.

Considerando que os recursos necessários para essa solução foram escolhidos para atender às premissas de gratuidade e com versões para os sistemas mencionados, a portabilidade do projeto se torna viável se respeitados a modelagem dos dados e o relacionamento das classes com pequenas adequações no código que se julguem necessárias. Uma opção é fazer com que os mesmos arquivos fonte sejam compilados nos dois sistemas. Para isto acontecer, as alterações devem ser suficientemente capazes de separar trechos do código que serão executados de maneira semelhantes, porém compilados e link-editados diferentemente em cada um dos sistemas. Esses recursos normalmente limitam os trechos de códigos com os

marcadores `#ifdef`, `#ifndef` e `#else`. Outra opção é fazer uma versão específica para cada plataforma. Assim, cada arquivo de código fonte terá uma especificação dedicada para um ou outro sistema.

Além disso, esse projeto contribuiu bastante para que, com o conhecimento adquirido, fosse possível conceber uma solução de monitoração funcionando em sistemas embarcados. Dessa forma, o Laboratório de Processamento de Sinais custeou a compra de uma placa de desenvolvimento baseada na família de DSPs da Texas Instruments DM64x. Essa família é responsável por liderar a grande maioria dos projetos que criam aplicações sobre imagens e vídeo.

Com velocidades que chegam até 720MHz, tais DSPs oferecem suportes dedicados em hardware às três interfaces de entrada e saída de vídeo, além da considerável disponibilidade de memória Cache *on-chip* de 272KB (Figura 35). Em software, são oferecidas diversas bibliotecas com suporte a codificadores em tempo real dos mais diferentes padrões, inclusive MPEG-4, da própria Texas ou de diferentes fabricantes.

A placa de desenvolvimento adquirida pelo LPS, modelo PLUTO (Figura 36), é fabricada pela empresa israelense Ymagic (www.ymagic.com) que oferece, atualmente, outros dois modelos diferentes: Polari e Gemini. O diagrama de blocos da placa PLUTO é apresentado na Figura 37.

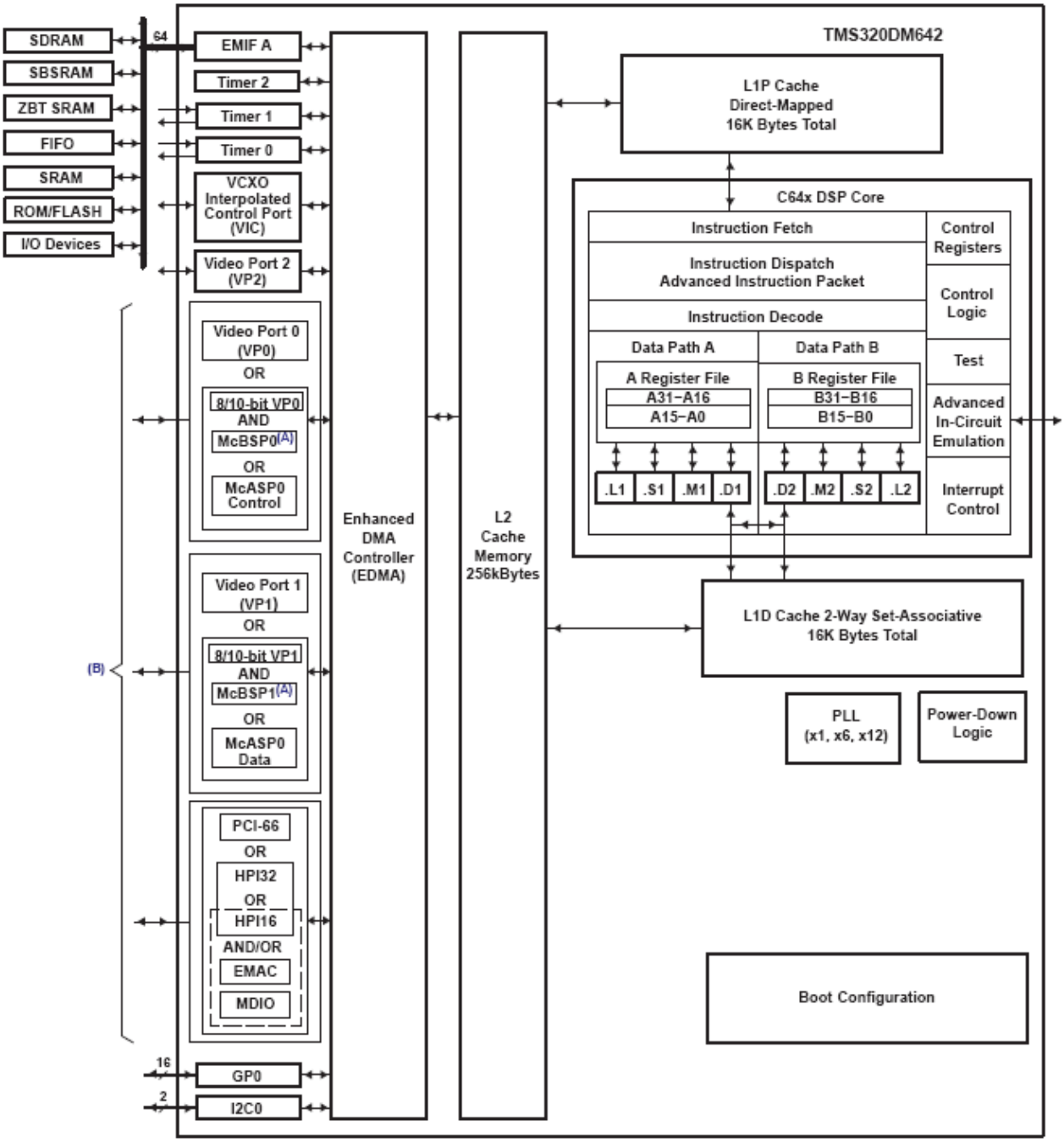


Figura 35: Diagrama de blocos do DSP TMS320DM642.
(fonte: http://focus.ti.com/lit/ds/symlink/tms320dm642.pdf)

Figura 36: Placa PLUTO da Ymagic
(fonte: http://www.ymagic.com/pluto.html)

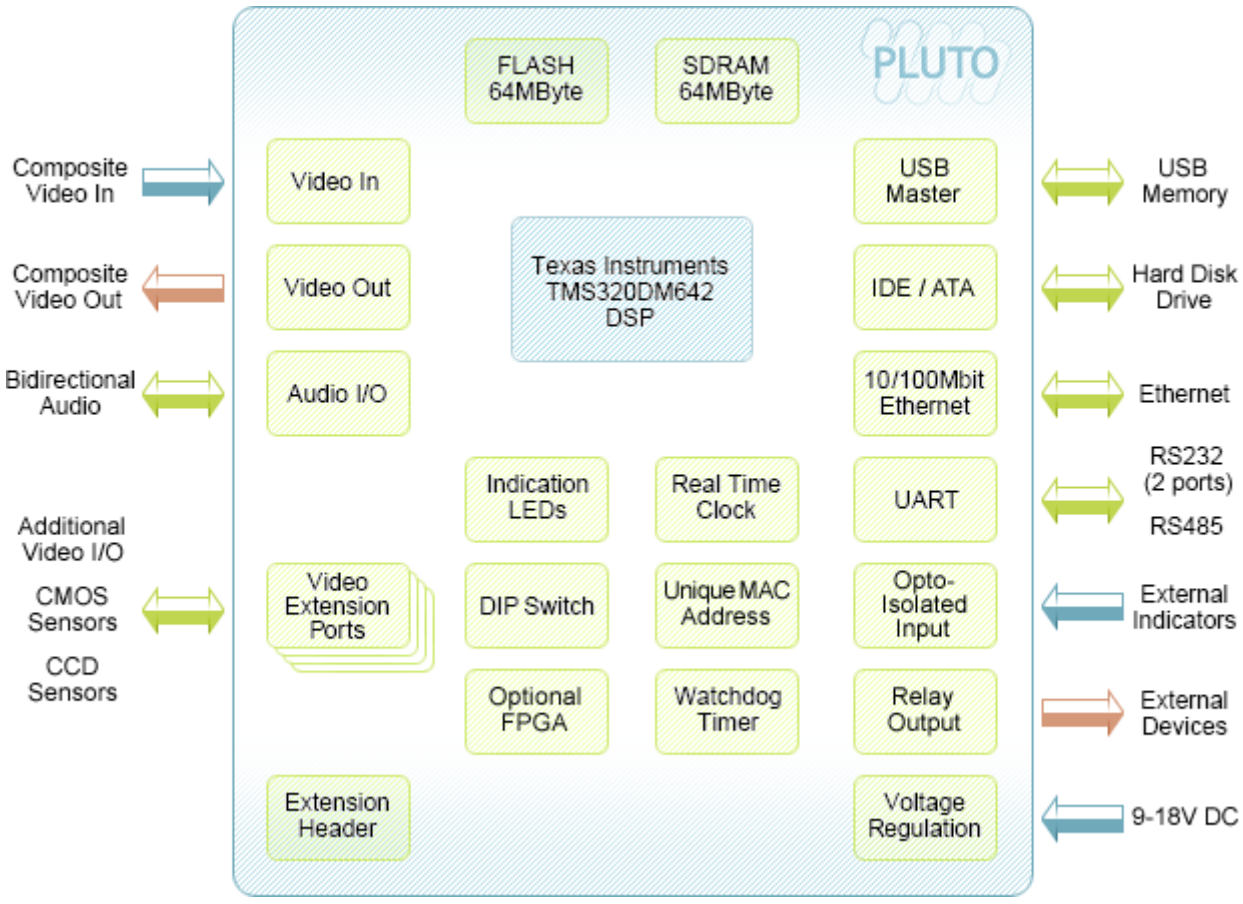


Figura 37: Diagrama de blocos do modelo Pluto
(fonte: http://www.ymagic.com/pluto.html)

Essa placa permite que um disco rígido seja conectado a ela através da interface IDE, aumentando assim a capacidade e tempo de armazenamento. Se analisarmos uma solução de monitoração aplicada a ambientes móveis, como os

transportes urbanos de ônibus, chegaríamos a uma conclusão que não seria adequado submeter os discos rígidos, que possuem controles mecânicos finos, às trepidações comuns desses veículos. Para permitir que o dispositivo de armazenamento não sofra danos por esse motivo, é aconselhável optar por dispositivos semelhantes às memórias *Compact Flash* (CF). Com esse novo cenário podemos imaginar o sistema de monitoração da Figura 38.

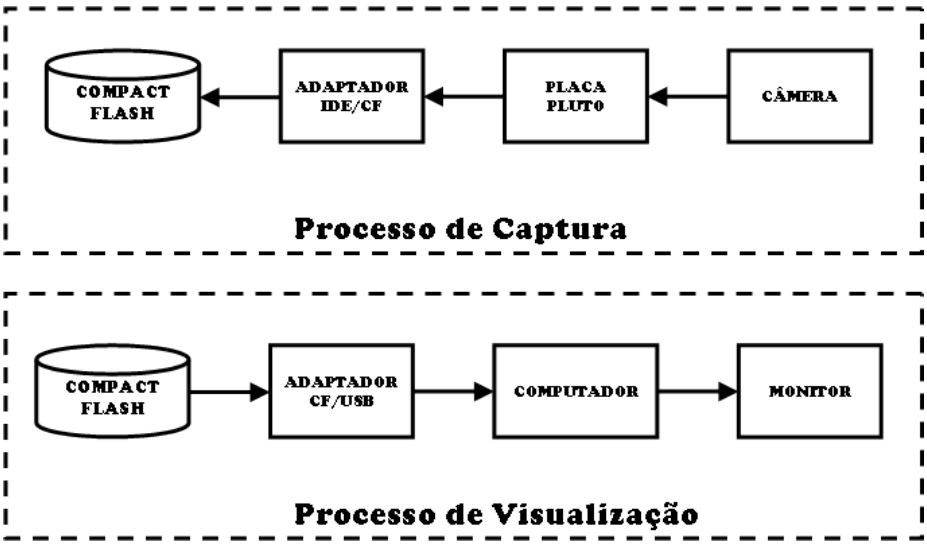


Figura 38: Novo cenário de monitoração

Capítulo 9

Conclusão e Trabalhos Futuros

Os programas desenvolvidos atendem aos requisitos de um sistema de monitoração de vídeo, com as ferramentas VideoCapture e CrossPlayer. Mais do que isso, criou-se uma biblioteca onde todos os recursos auxiliares são transparentes para as funções utilizadas no nível de aplicação final (LIBMONITOR). Assim, foi possível construir métodos intuitivos e que realizam as tarefas pertinentes aos processos de captura, codificação, armazenamento e reprodução de um sinal de vídeo.

Esse é um sistema desenvolvido ao longo do período da disciplina Projeto Final, onde todo o conhecimento adquirido com os estudos e pesquisas deram a esse projeto dois grandes objetivos: 1) difundir o funcionamento desse tipo de aplicação com total segurança e 2) a importância de descrever com detalhes essa implementação. Como resultado final do curso do Departamento de Eletrônica, o sistema desenvolvido agrega conceitos de computação, processamento de sinais e sistemas digitais vistos durante toda a formação.

Desde o aprendizado das linguagens de programação, úteis em quase todas as disciplinas, passando pelas técnicas de compressão e codificação de vídeo e considerando também os projetos com circuitos integrados, a formação obtida nesse curso de Engenharia Eletrônica e de Computação ofereceu condições de definir, desenvolver e concluir de maneira satisfatória um projeto de engenharia eficaz.

Os trabalhos futuros nesse tipo de aplicação são dos mais variados. Primeiro pode-se investir no desenvolvimento de uma solução embarcada utilizando a placa de desenvolvimento da Ymagic comprada pelo LPS. Outra possibilidade é adaptar o sistema apresentado nos seguintes itens:

- Otimizar a taxa de aquisição utilizando outros métodos de captura da API Video For Linux [10] (mapeamento de memória ou ponteiro dinâmico)
- Expandir a captura para as quatro diferentes entradas de vídeo.
- Criar uma janela gráfica onde seja possível monitorar o que está sendo capturado.
- Criar uma solução gráfica integrando os processos de captura e visualização.
- Permitir ao usuário configurar os parâmetros de codificação do vídeo antes da execução.
- Habilitar e desenvolver a captura de áudio associada ao sinal de vídeo.

Bibliografia

[1] Samart, Julian; Hock, Kevin – Cross-Pataform GUI Programming with wxWidgets, Prentice Hall, 2005.

[2] Schildt, Herbert – C++: The Complete Reference Third Edition, McGraw-Hill, 1998.

[3] Smart, Julian; Roebling, Robert; Zeitlin, Vadim; Dunn, Robin – wxWidgets 2.6.3: A portable C++ and Python GUI toolkit, http://www.wxwindows.org/manuals/2.6.3/wx_contents.html, 2006, acessado nos meses de Maio, Julho e Agosto de 2006.

[4] Autores da Biblioteca SDL – Using the Simple DirectMedia Layer API, http://www.libsdl.org/intro.en/toc.html, 2006, acessado nos meses de Maio, Julho e Agosto de 2006.

[5] Autores da Biblioteca SDL – SDL Library Documentation, http://www.libsdl.org/docs.php, 2006, disponível nos formatos HTML, PostScript, Windows Help (chm) e páginas de manual UNIX, acessado nos meses de Maio, Julho e Agosto de 2006.

[6] Böhme, Martin – Using libavformat and libavcodec, http://www.inb.uni-luebeck.de/~boehme/libavcodec_update.html, 2004, acessado nos meses de Maio, Julho e Agosto de 2006.

[7] Minut, Silviu – FFMPEG API Overview example, http://www.cse.msu.edu/~minutsil/linux.html, 2004, acessado nos meses de Maio Julho e Agosto de 2006.

[8] Sem autor, FFMPEG API Documentation, Generated by Doxygen http://cekirdek.pardus.org.tr/~ismail/ffmpeg-docs, 2006, acessado nos meses de Maio, Julho e Agosto de 2006.

[9] CODEnpent – wx-sdl: A Tutorial on combining wxWidgets with SDL, http://code.technoplaza.net/wx-sdl/, 2005, acessado nos meses de Maio, Julho e Agosto de 2006

[10] Schimek, Michael H.; Dirks, Bill; Verkuil, Hans – Vídeo For Linux 2 API Specification, Draft 0.13, 2006, disponível em http://v4l2spec.bytesex.org/, acessado em Agosto de 2006

[11] Lowe, Kwan - Kernel Rebuild Guide, 2004, disponível em http://www.digitalhermit.com/linux/Kernel-Build-HOWTO.html, acessado em Agosto de 2006.

[12] Richardson, Iain E. G. – H.264 and MPEG-4 Video Compression, Wiley, 2003.

[13] Recomendação ITU-R BT.601

[14] Desenvolvedores do Video For Linux - http://www.linuxtv.org/v4lwiki/index.php/Main_Page, acessado em Agosto de 2006.

[15] Desenvolvedores do kernel do Linux – http://www.kernel.org, acessado em Agosto de 2006.

[16] Desenvolvedores do driver BTTV - http://linux.bytesex.org/v4l2/bttv.html, acessado em Agosto de 2006.

[17]Kumar, Naba – Anjuta DevStudio, http://anjuta.sourceforge.net/, disponível na distribuição Debian/GNU Linux 3.1.

[18] Towns, Anthony; Langasek, Steve; Watson Colin; demais desenvolvedores – Debian GNU Linux, http://www.debian.org, acessado desde 2001.

[19] Desenvolvedores do GNU C Compiler - http://gcc.gnu.org/, acessado desde 2001.

[20] Secretaria Municipal de Transportes – Prefeitura do Rio de Janeiro – http://www.rio.rj.gov.br/smtr - acessado diariamente.

[21] Noticiário UOL, acessado em Setembro de 2006, disponível em http://tecnologia.uol.com.br/ultnot/reuters/2006/09/15/ult3949u25.jhtm

[22] Noticiário Folha OnLine, acessado em Setembro de 2006, disponível em http://www1.folha.uol.com.br/folha/dinheiro/ ult91u111025.shtml

[23] Noticiário Último Segundo - IG, acessado em Setembro de 2006, disponível em http://ultimosegundo.ig.com.br/materias/ mundovirtual/2520001-2520500/2520320/2520320_1.xml

[24] Richard Stallman e contribuidores GNU – GNU General Public License version 2 – Free Software Foundation Inc., 1991, disponível em http://www.gnu.org/licenses/gpl.html

[25] FFMPEG Based Projects, http://ffmpeg.mplayerhq.hu/ projects.html, acessado em Agosto 2006.

[26] Especificações da placa de captura MiniDVR - TecVoz, http://www.tecvoz.com.br/dvr/placa_dvr_detalhe.asp?placa=minidvr, acessado em Maio de 2006.

[27] Lista completa dos codecs de vídeo suportados pela biblioteca libavcodec http://ffmpeg.mplayerhq.hu/ffmpeg-doc.html#SEC19 , acessado em Maio de 2006.

[28] Lista completa dos codecs de áudio suportados pela biblioteca libavcodec http://ffmpeg.mplayerhq.hu/ffmpeg-doc.html#SEC20 , acessado em Maio de 2006.

[29] Lista completa dos formatos de bitstreams suportados pela biblioteca libformat http://ffmpeg.mplayerhq.hu/ffmpeg-doc.html#SEC17 , acessado em Maio de 2006.

Apêndice I

Dicionário das Classes LIBMONITOR

I.1

Classe Codec

Arquivos comuns à compilação:

```
#include <ffmpeg/avformat.h>
#include <ffmpeg/avcodec.h>
#include <ffmpeg/avutil.h>
#include "codec.h"
```

I.1.1

DecodeVideo

Nome

`DecodeVideo` – Decodifica um quadro de vídeo

Sinopse

```
int Codec::DecodeVideo (int streamIndex, AVPacket *pkt,
AVFrame *frame);
```

Descrição

Essa função é responsável por decodificar um quadro de vídeo. A informação codificada é representada por *pkt e o resultado é gerado em *frame. O índice

do stream referente ao pacote é passado por streamIndex. Essa função retorna 0 se não for possível decodificar o quadro, senão, retorna diferente de 0.

I.1.2

EncodeVideoFrame

Nome

`EncodeVideoFrame` –Codifica um quadro de vídeo

Sinopse

```
int Codec::EncodeVideoFrame (uint8_t **outbuf, int outbuf_size);
```

Descrição

Codifica um quadro de vídeo e utiliza indiretamente os atributos m_frame e m_ccx, que representam respectivamente o quadro a ser codificado e toda a informação necessária para configurar o padrão e parâmetros de codificação. O quadro codificado é associado com **outbuf. Precisamos informar também em outbuf_size o tamanho de *outbuf.

I.1.3

EncodeVideoFrame

Nome

`EncodeVideoFrame` –Codifica um quadro de vídeo

Sinopse

```
int Codec::EncodeVideoFrame (uint8_t **outbuf, int outbuf_size, FILE *output);
```

Descrição

Semelhante a função anterior com a diferença que permite a gravação do bitstream codificado no arquivo representado por *output.

I.1.4

EncodeVideoFrame

Nome

`EncodeVideoFrame` –Codifica um quadro de vídeo

Sinopse

```
int Codec::EncodeVideoFrame (uint8_t **outbuf, int outbuf_size, AVFrame *frame);
```

Descrição

Semelhante a função I.1.2 com a diferença que o quadro a ser codificado está representado por *frame.

I.1.5

FindDecoder

Nome

`FindDecoder` – Procura por um padrão de decodificação

Sinopse

```
AVCodec *Codec::FindDecoder (CodecID id);
```

Descrição

Procura pelo padrão de decodificação presente na libavcodec, representado pelo identificador id, e retorna um ponteiro paraa estrutura referente ao padrão.

I.1.6

FindEncoder

Nome

`FindEncoder` – Procura por um padrão de codificação

Sinopse

```
AVCodec Codec::FindEncoder (CodecID id);
```

Descrição

Procura pelo padrão de codificação presente na libavcodec, representado pelo identificador id, e retorna um ponteiro para a estrutura referente ao padrão.

I.1.7

InitAVCodec

Nome

`InitAVCodec` – Registra os padrões de codificação/decodificação.

Sinopse

```
void Codec::InitAVCodec ();
```

Descrição

Inicializa todos os padrões codificação e decodificação que a libavcodec implementa. Essa função deve ser chamada no início da execução de cada programa, pois permite que os processos referentes à manipulação dos dados codificados sejam válidos.

I.1.8

InitEncodeProcess

Nome

`InitEncodeProcess` –Inicializa o processo de compressão

Sinopse

```
int Codec::InitEncodeProcess (int width, int height, int
bitrate, int fps, int total_frames, const char *filename);
```

Descrição

Configura os parâmetros de codificação de vídeo e essabelece o padrão de compressão (MPEG 4 Parte 2), gera o cabeçalho referente a esse padrão e formata a estrutura de dados que representará o arquivo com bitstream do vídeo codificado, com, por exemplo, informações de duração. Dessa maneira, garantimos que esse arquivo será compatível com os diferentes programas de reprodução. Retorna 0 caso obtenha sucesso na configuração.

I.1.9

IsSupported

Nome

`IsSupported` – Tessa o codec.

Sinopse

```
bool Codec::IsSupported (CodecID id);
```

Descrição

Tessa se o padrão identificado por id é suportado pela libavcodec. Retorna verdadeiro caso positivo e falso do contrário.

I.1.10

OpenCodec

Nome

`OpenCodec` – Abre/Inicializa um contexto de codificação/decodificação

Sinopse

```
int Codec::OpenCodec (AVCodecContext *ccx, AVCodec *codec);
```

Descrição

Essa função permite que um padrão de codificação/decodificação representado por *codec seja associado com a variável *ccx.Retorna -1 caso ocorra falha na associação.

I.1.11

ReadFrame

Nome

`ReadFrame` – Lê uma seqüência de bits do arquivo

Sinopse

```
int Codec::ReadFrame (AVPacket *pkt);
```

Descrição

Lê um conjunto de bits do arquivo referente a um stream específico, seja vídeo, áudio ou dados. Retorna o número de bytes lidos.

I.1.12

RegisterAllCodecs

Nome

`RegisterAllCodecs` – Registra e inicializa os padrões suportados.

Sinopse

```
void Codec::RegisterAllCodecs ();
```

Descrição

Inicializa e registra todos os formatos de arquivos e de codificação suportados pela libavformat e libavcodec, respectivamento.

I.1.13

MapDataYUV420

Nome

`MapDataYUV420` – Mapeia os dados no formato YUV 420.

Sinopse

```
void Codec::MapDataYUV420 (uint_8 *data, int width, int height);
```

Descrição

Preenche a o atributo m_frame, que separa em três diferentes arrays os dados de luminância Y, chrominância Cb e Cr. A origem dos dados é proveniente de *data que tem os valores dos pixels numa única seqüência de tamanho (width x height) + (width x height)/4 + (width x height)/4 para uma amostragem 4:2:0.

I.1.14

WriteFrame

Nome

`WriteFrame` – Escreve bitstream do quadro codificado.

Sinopse

```
void Codec::WriteFrame (uint_8 *data, int size);
```

Descrição

Usa a própria função da libavformat, av_write_frame, para escrever a informação codificada representada por *data de tamanho size.

I.1.15

UninitEncodeProcess

Nome

`UninitEncodeProcess` – Finaliza o processo de codificação.

Sinopse

```
void Codec::UninitEncodeProcess ();
```

Descrição

É responsável por finalizar o processo de codificação, liberando com o contexto de compressão, bem como formatando os códigos do final do bitstream, dependendo do padrão, e inserindo-os no final do arquivo.

I.2

Classe Capture

Arquivos comuns à compilação:

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <string>
#include <assert.h>
#include <fnctl.h>
#include <unistd.h>
#include <errno.h>
```

```
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/mman.h>
#include <sys/ioctl.h>
#include <sys/select.h>
#include <asm/types.h>
#include <linux/videodev2.h>
#include <ffmpeg/avformat.h>
#include <ffmpeg/avcodec.h>
#include <ffmpeg/avutil.h>
#include "codec.h"
#include "capture.h"
```

I.2.1

CloseDevice

Nome

`CloseDevice` – Fecha o dispositivo de captura

Sinopse

```
void Capture::CloseDevice ();
```

Descrição

Essa função é responsável por fechar o dispositivo representado pelo atributo m_fd.

I.2.2

InitDevice

Nome

`InitDevice` – Inicializa o dispositivo de captura

Sinopse

```
int Capture::InitDevice(int width, int height, int pixelformat);
```

Descrição

Essa função realiza algumas consultas ao dispositivo de captura para negociar a forma de captura do quadro determinada pela dimensão witdh x height, testa dois métodos de captura: *read/write* e *streaming*, além de especificar a formatação dos pixels pelo parâmetro pixelformat.

I.2.3

InitIOMethod

Nome

InitIOMethod – Inicializa algumas variáveis usadas no método de IO.

Sinopse

```
int Capture::InitIOMethod (unsigned int buffer_size);
```

Descrição

Essa função prepara (reserva memória para) o atributo que guardará o quadro capturado, bem como o atributo responsável por armazenar o bitstream do quadro codificado. O parâmetro buffer_size indica quantos bytes serão alocados para o bitstream de codificação. A função retorna 0 caso obtenha sucesso na alocação de memória.

I.2.4

InitIOMethodUserPTR - Experimental

Nome

`InitIOMethodUserPTR` – Método UserPTR de captura

Sinopse

```
void Capture::InitIOMethodUserPTR (unsigned int
buffer_size);
```

Descrição

Tem o mesmo objetivo da função InitIOMethod, porém implementa a alocação de memória de mais unidades, semelhante a um *array*.

I.2.5

MainLoop

Nome

`MainLoop` – Loop principal

Sinopse

```
void Capture::MainLoop (int dur);
```

Descrição

É nesta função que ocorre o loop de captura dos quadros. Este processo tem a duração indicada por dur, que representa o número de quadros que devem ser capturados.

I.2.6

ReadFrame

Nome

`ReadFrame` – Leitura de um quadro

Sinopse

```
int Capture::ReadFrame ();
```

Descrição

Esta função é responsável por preencher a área de memória do buffer com o quadro capturado. O processo é feito através da função *read*. Retorna o número de quadros lidos (1).

I.2.7

ReadFrameUserPTR - Experimental

Nome

`ReadFrameUserPTR` – Leitura de um quadro

Sinopse

```
int Capture::ReadFrameUserPTR ();
```

Descrição

Com o mesmo objetivo da função anterior, difere apenas do método de captura: utiliza *streaming* e tenta capturar mais de um quadro por vez. A área de memória alocada para o buffer é associada com a captura seqüencial dos quadros com o número de vezes igual ao tamanho do array reservado.

I.2.8

FrameProcess

Nome

`FrameProcess` – Funções pós-captura

Sinopse

```
void Capture::FrameProcess (int i);
```

Descrição

Esta função realiza o mapeamento do quadro em YUV 4:2:0, codifica-o e escreve no arquivo final.

I.2.9

FrameProcessUserPTR – Experimental

Nome

`FrameProcessUserPTR` – Funções pós-captura

Sinopse

```
void Capture::FrameProcessUserPTR (uint8_t *data);
```

Descrição

Esta função tem o mesmo objetivo da função anterior, mas manipula os dados que estão em *data.

I.2.10

SupportVideoCapture

Nome

`SupportVideoCapture` – Testa a capacidade de captura

Sinopse

```
bool Capture::SupportVideoCapture ();
```

Descrição

Testa se o dispositivo é capaz de capturar um quadro. Retorna verdadeiro ou falso.

I.2.11

SupportVideoOutput

Nome

`SupportVideoOutput` – Testa a capacidade de reprodução.

Sinopse

```
bool Capture::SupportVideoOutput ();
```

Descrição

Testa se o dispositivo é capaz de gerar um sinal de vídeo para ser exibido/capturado em outro dispositivo. Ou seja, testa se pode ser usado como uma placa gráfica. Retorna verdadeiro ou falso.

I.2.12

SupportVideoOverlay

Nome

`SupportVideoOverlay` – Testa a capacidade de manipular as propriedades do quadro

Sinopse

```
bool Capture::SupportVideoOverlay ();
```

Descrição

Esta função testa a capacidade do dispositivo alterar as propriedades dos quadros capturados, dimensões, colorimetria, brilho, contraste. Tudo isso é

feito na região de memória da própria placa de captura. Retorna verdadeiro ou falso.

I.2.12

SupportVideoVBICapture

Nome

`SupportVBICapture` – Testa a capacidade de interpretar os dados no VBI

Sinopse

```
int Capture::SupportVBICapture ();
```

Descrição

Testa se o dispositivo identifica os dados que são transmitidos no intervalo de retraço vertical (*Vertical Blank Interval*). Atualmente, neste período de varredura vertical, entre as informações dos campos pares e ímpares, não são introduzidos dados, modulados na transmissão analógica para enviar, por exemplo, o closed-caption. Esses dados são normalmente chamados de serviços da linha 21. Retorna verdadeiro ou falso.

I.2.13

SupportVBIOutput

Nome

`SupportVBIOutput` – Testa a capacidade de reprodução com VBI

Sinopse

```
bool Capture::SupportVBIOutput ();
```

Descrição

Testa se o dispositivo é capaz de gerar um sinal de vídeo analógico, introduzindo na transmissão, os dados de algum serviço durante o VBI. Retorna verdadeiro ou falso.

I.2.14

SupportRDSCapture

Nome

`SupportRDSCapture` – Testa a capacidade de receber dados de um sinal de radio.

Sinopse

```
bool Capture::SupportRDSCapture ();
```

Descrição

Testa se o dispositivo suporta a recepção/demodulação de dados binários de um sinal de rádio (Radio Data System). Por exemplo, uma informação transmitida é o nome da estação. Retorna verdadeiro ou falso.

I.2.15

SupportTuner

Nome

`SupportTuner` – Testa a capacidade de receber um sinal modulado.

Sinopse

```
bool Capture::SupportTuner ();
```

Descrição

Esta função testa a capacidade do dispositivo receber/demodular um sinal em rádio-freqüência. Retorna verdadeiro ou falso.

I.2.16

SupportAudioCapture

Nome

`SupportAudioCapture` – Testa a capacidade de capturar sinal de áudio

Sinopse

```
bool Capture::SupportAudioCapture ();
```

Descrição

É com esta função que descobrimos se é possível capturar áudio no dispositivo. Retorna verdadeiro ou falso.

I.2.17

SupportReadWrite

Nome

`SupportReadWrite` – Testa a capacidade de capturar/exibir pelo método READWRITE.

Sinopse

```
bool Capture::SupportReadWrite ();
```

Descrição

Nesta função, sabemos se o dispositivo é capaz de capturar e/ou exibir os quadros usando o método read() e write (). Retorna verdadeiro ou falso.

I.2.18

SupportAsyncIO

Nome

`SupportAsyncIO` – Testa se é permitido ler ou escrever no dispositivo usando métodos assíncronos.

Sinopse

```
bool Capture::SupportAsyncIO ();
```

Descrição

Permite ao usuário saber se o dispositivo é capaz de suportar métodos assíncronos de leitura e escrita. Retorna verdadeiro ou falso.

I.2.19

SupportStreaming

Nome

`SupportStreaming` – Testa a capacidade de usar métodos de entrada e saída baseados em *streaming*.

Sinopse

```
bool Capture::SupportStreaming ();
```

Descrição

Testa a capacidade de usar métodos de entrada e saída baseados em *streaming*. Retorna verdadeiro ou falso.

I.2.20

StartCapturing

Nome

`StartCapturing` – Faz a primeira captura.

Sinopse

```
bool Capture::StartCapturing ();
```

Descrição

Nesta função, quando algum método de entrada e saída baseado em streaming é usado, esta função realiza a primeira captura e ativa o flag VIDIOC_STREAMON para as aquisições seguintes. Retorna zero se positivo.

I.2.21

StopCapturing

Nome

`StopCapturing – Termina o processo de captura.`

Sinopse

```
int Capture::StopCapturing ();
```

Descrição

Esta função configura o flag VIDIOC_STREAMOFF indicando que o processo de aquisição foi terminado. Retorna zero se positivo.

I.2.22

UinitDevice

Nome

`UinitDevice – Retorna ao estado inicial.`

Sinopse

```
void Capture::UinitDevice
```

Descrição

Esta função libera as áreas de memória alocadas para receber os quadros durante a captura.

I.2.23

UinitDeviceUserPTR - Experimental

Nome

`UinitDeviceUserPTR`

Sinopse

`void Capture::UinitDeviceUserPTR` – Retorna ao estado inicial.

Descrição

Esta função realiza a mesma tarefa da função anterior, mas considera outras variáveis usadas na captura em *streaming*.

I.2.24

Xioctl

Nome

`Xioctl –` Chamada do sistema e permite acesso ao dispositivo.

Sinopse

```
int Capture::Xioctl (int fd, int request, void *arg)
```

Descrição

Esta função permite que qualquer requisição válida, representada por *request*, seja feita ao dispositivo *fd* passando a ele o ponteiro **arg*. É nesta chamada que

são feitas as consultas e configurações no dispositivo de captura. Retorna -1 caso ocorra algum erro, ou maior que 0 em casos de sucesso.

I.3

Classe MultimediaStream

Arquivos comuns à compilação:

```
#include <iostream>
#include <list>
#include <string>
#include <ffmpeg/avformat.h>
#include <ffmpeg/avcodec.h>
#include <ffmpeg/avutil.h>
#include "codec.h"
#include "multimediaStream.h"
```

I.3.1

GetMediaList

Nome

`GetMediaList` – Devolve uma lista de conteúdos

Sinopse

```
MediaDescriptorList MultimediaStream::GetMediaList
(CodecType type)
```

Descrição

Retorna uma lista contendo os descritores de todos os streams do tipo indicado por type presentes no arquivo. Essa estrutura foi construída para que suas variáveis pudessem indicar o mínimo de informação necessária para classificarmos o conteúdo. Exemplo: taxa de codificação, padrão de compressão, número de canais (áudio), dimensão da tela, entre outros. É usado o contenedor da STL chamado *list*.

I.3.2

GetNumberOfVideoStream

Nome

`GetNumberOfVideoStream` – Devolve o número de streams de vídeo.

Sinopse

```
short uint MultimediaStream::GetNumberOfVideoStream ();
```

Descrição

Calcula, através do arquivo de entrada, o número de streams de vídeo. Retorna exatamente este valor.

I.3.3

GetNumberOfAudioStream

Nome

`GetNumberOfAudioStream` – Devolve o número de streams de áudio.

Sinopse

```
short int MultimediaStream::GetNumberOfAudioStream ();
```

Descrição

Calcula, através do arquivo de entrada, o número de streams de áudio. Retorna exatamente este valor.

I.3.4

GetNumberOfSubtitlesStream

Nome

`GetNumberOfSubtitlesStreams` – Devolve o número de streams de legenda.

Sinopse

```
shor int MultimediaStream::GetNumberOfSubtitlesStreams ();
```

Descrição

Calcula, através do arquivo de entrada, o número de streams de legenda. Retorna exatamente este valor.

I.3.5

GetPlayFlags

Nome

`GetPlayFlags` – Devolve os parâmetros de exibição.

Sinopse

```
int MultimediaStream::GetPlayFlags ();
```

Descrição

Retorna o valor da variável m_playFlags, que contém a seleção dos streams que serão exibidos.

I.3.6

HasVideoStreams

Nome

`HasVideoStreams` – Testa a existência de streams de vídeo.

Sinopse

```
bool MultimediaStream::HasVideoStreams ();
```

Descrição

Consulta a existência de streams de vídeo e retorna verdadeiro se é encontrado pelo menos um.

I.3.7

HasAudioStreams

Nome

`HasAudioStreams` – Testa a existência de streams de áudio

Sinopse

```
bool MultimediaStream::HasAudioStreams ();
```

Descrição

Consulta a existência de streams de áudio e retorna verdadeiro se é encontrado pelo menos um.

I.3.8

HasSubtitlesStreams

Nome

`HasSubtitlesStreams` – Testa a existência de streams de legendas

Sinopse

```
bool MultimediaStream::HasSubtitlesStreams();
```

Descrição

Consulta a existência de streams de legendas e retorna verdadeiro se é encontrado pelo menos um.

I.3.9

LoadStreamInfo

Nome

`LoadStreamInfo` – Carrega as informaçõe básicas do arquivo multimídia

Sinopse

```
int MultimediaStream::LoadStreamInfo ();
```

Descrição

Acessa as informações do arquivo de entrada. Retorna maior ou igual a 0 caso obtenha sucesso.

I.3.10

OpenStream

Nome

`OpenStream – Abre o arquivo.`

Sinopse

```
int MultimediaStream::OpenStream (const char *filename);
```

Descrição

Esta função abre o arquivo “filename” .

I.3.11

SetFilename

Nome

`SetFilename` – Escreve na variável m_filename.

Sinopse

```
void MultimediaStream::SetFilename (std::string filename);
```

Descrição

Esta função escreve na variável m_filename que representa o nome do arquivo multimídia.

I.3.12

SetFilesize

Nome

`SetFilesize` – Escreve na variável m_filesize.

Sinopse

```
void MultimediaStream::SetFilesize (int size);
```

Descrição

Esta função escreve na variável m_filesize que representa o tamanho do arquivo multimídia.

I.3.13

SetPlayOptions

Nome

`SetPlayOptions` – Escolhe os streams que serão reproduzidos.

Sinopse

```
void MultimediaStream::SetPlayOptions (int
videoStreamIndex = -1, int audioStreamIndex = -1, int
subtitlesStreamIndex = -1);
```

Descrição

Esta função permite escolher os identificadores dos streams que serão reproduzidos. Para que uma categoria não seja reproduzida, simplesmente é atribuído o valor de -1.

I.4

Classe Video

Arquivos comuns à compilação:

```
#include <list>
#include <string>
#include <math>
#include <wx/wx.h>
#include <wx/dcbuffer.h>
#include <wx/dcclient.h>
#include <ffmpeg/avformat.h>
#include <ffmpeg/avcodec.h>
#include <ffmpeg/avutil.h>
#include <SDL.h>
#include "codec.h"
#include "multimediaStream.h"
#include "video.h"
```

I.4.1

CreateScreen

Nome

`CreateScreen` – Cria na tela um *buffer* de vídeo.

Sinopse

```
void Video::CreateScreen ();
```

Descrição

Esta função tem o objetivo de criar uma região na tela capaz de receber informações de elementos de imagem (valor de pixel). Isto significa que esta região é capaz a receber os desenhos de cada imagem que compõe o vídeo.

I.4.2

DrawFrame

Nome

`DrawFrame` – Desenha um quadro na tela.

Sinopse

```
void Video::DrawFrame ();
```

Descrição

Esta função é capaz de desenhar, no vídeo *buffer*, um quadro do stream de vídeo.

I.4.3

DrawLine

Nome

`DrawLine` – Desenha uma linha do quadro.

Sinopse

```
void Video::DrawLine (Uint8 *pLine, int y);
```

Descrição

A função DrawLine é capaz de desenhar a linha, de número y, cujos valores dos pixels estão em pLine.

I.4.4

DrawPixel

Nome

`DrawPixel` – Desenha um pixel do quadro.

Sinopse

```
void Video::DrawPixel (int x, int y, Uint32 pixel);
```

Descrição

A função DrawPixel é responsável por desenhar o pixel na posição (x,y) cujo valor está informado no parâmetro pixel. Cada valor tem a resolução de 32 bits.

I.4.5

GetFrameTime

Nome

`GetFrameTime` – Devolve o período de um quadro em microsegundos.

Sinopse

```
double Video::GetFrameTime ();
```

Descrição

Esta função retorna o período de um quadro, ou seja, o inverso de quadros por segundo.

I.4.6

LoadVideoInfo

Nome

`LoadVideoInfo` – Carrega informações do vídeo.

Sinopse

```
void Video::LoadVideoInfo ();
```

Descrição

É responsável por ler do stream de vídeo, os parâmetros que o caracteriza, por exemplo, taxa, dimensão horizontal e vertical, formato do pixel, nome do codificador, tamanho do GOP, etc...

I.4.7

PlayVideoFrame

Nome

`PlayVideoFrame` – Exibe um quadro do vídeo.

Sinopse

```
int Video::PlayVideoFrame (AVPacket *videoPacket);
```

Descrição

Realiza os processos de decodificação e reprodução do quadro, cujo bitstream comprimido está representado por videoPacket.

I.4.8

Show

Nome

`Show` – Exibe.

Sinopse

```
void Video::Show ();
```

Descrição

Responsável por exibir o quadro nobuffer de vídeo criado pela SDL.

I.4.9

Show

Nome

`Show` – Exibe

Sinopse

```
void Video::Show (wxWindow *win);
```

Descrição

Responsável por exibir o quadro no buffer de vídeo criando pela wxWidgets.

I.4.10

SetWidth

Nome

`SetWidth` – Escreve na variável m_width

Sinopse

```
void Video::SetWidth (unsigned short int w);
```

Descrição

Responsável por escrever o valor da dimensão horizontal do vídeo representado por w.

I.4.11

GetWidth

Nome

`GetWidth` – Devolve o valor da variável m_width

Sinopse

```
unsigned short int Video::GetWidth ();
```

Descrição

Retorna a dimenão horizontal do vídeo.

I.4.12

SetHeight

Nome

`SetHeight` – Escreve na variável m_height

Sinopse

```
void Video::SetHeight (unsigned short int h);
```

Descrição

Responsável por escrever o valor da dimensão vertical do vídeo representado por h.

I.4.13

GetHeight

Nome

`GetHeight` – Devolve o valor da variável m_height

Sinopse

```
unsigned short int Video::GetHeight ();
```

Descrição

Retorna a dimenão vertical do vídeo.

I.4.14

SetInterlaced

Nome

`SetInterlaced` – Escreve na variável m_interlaced

Sinopse

```
void Video::SetInterlaced (bool i);
```

Descrição

Responsável por informar se o formato do vídeo é entrelaçado (i=true), ou progressivo (i=false)

I.4.15

GetInterlaced

Nome

`GetInterlaced` – Devolve o valor na variável m_interlaced.

Sinopse

```
Video::GetInterlaced ();
```

Descrição

Retorna a informação, verdadeira ou falsa sobre o formato do vídeo, progressivo ou entrelaçado.

I.4.16

SetFPS

Nome

`SetFPS` – Escreve na variável m_fps

Sinopse

```
void Video::SetFPS (double fps);
```

Descrição

Responsável por escrever a taxa de quadros por segundo (fps) do vídeo a ser exibido.

I.4.17

GetFPS

Nome

`GetFPS` – Devolve o valor da variável m_fps

Sinopse

```
double Video::GetFPS ();
```

Descrição

Retorna o valor da taxa de quadros por segundo do vídeo.

I.4.18

GetStreamIndex

Nome

`GetStreamIndex` – Devolve o valor da variável m_streamIndex

Sinopse

```
unsigned short int Video::GetStreamIndex ();
```

Descrição

Retorna o índice que identifica o stream de vídeo dentro do arquivo multimídia.

I.4.19

SetStreamIndex

Nome

`SetStreamIndex` – Escreve na variável m_streamIndex.

Sinopse

```
void Video::SetStreamIndex (unsigned short int streamIndex);
```

Descrição

Responsável por escrever na variável m_streamIndex o índice do stream de vídeo, representado por streamIndex.

I.4.20

GetCodecID

Nome

`GetCodecID` – Devolve o valor da variável m_codecID

Sinopse

```
unsigned short int Video::GetCodecID ();
```

Descrição

Retorna o índice do codec referente aos padrões suportados pela biblioteca libavcodec. O índice gravado identifica o padrão usado na decodificação do vídeo exibido.

I.4.21

SetCodecID

Nome

`SetCodecID` – Escreve na variável m_codecID

Sinopse

```
void Video::SetCodecID (unsigned shor int codecID);
```

Descrição

Responsável por escrever na variável m_codecID o índice do codec que será usado na decodificação do vídeo.

I.4.22

GetBitRate

Nome

`GetBitRate` – Devolve o valor da variável m_bitrate.

Sinopse

```
unsigned int Video::GetBitRate ();
```

Descrição

Retorna o valor da taxa, em bits por segundo, do stream de vídeo.

I.4.23

SetBitRate

Nome

`SetBitRate` – Escreve na variável m_bitrate.

Sinopse

```
void Video::SetBitRate (unsigned int bitrate);
```

Descrição

Responsável por escrever a taxa, bitrate, de codificação do stream de vídeo.

I.4.24

GetGopSize

Nome

`GetGopSize` – Devolve o valor da variável m_gopSize

Sinopse

```
unsigned short int Video::GetGopSize ();
```

Descrição

Retorna o valor do *Group Of Pictures*, parâmetro de codificação do vídeo. Este parâmetro representa o período de quadros tipo Intra, que possui análise de compressão apenas espacial.

I.4.25

SetGopSize

Nome

`SetGopSize` – Escreve na variável m_gopSize

Sinopse

```
void Video::SetGopSize (unsigned short int gopSize);
```

Descrição

Essa função determina o valor do GOP do vídeo. (ver I.4.24)

I.4.26

GetCodecName

Nome

`GetCodecName` – Devolve o valor da variável m_codecName

Sinopse

```
std::string Video::GetCodecName ();
```

Descrição

Retorna o nome do codec usado na codificação do stream de vídeo.

I.4.27

SetCodecName

Nome

`SetCodecName` – Escreve na variável m_codecName

Sinopse

```
void Video::SetCodecName (std::string codecName);
```

Descrição

Responsável por escrever o nome do padrão usado na codificação do vídeo.

I.4.28

GetAVFrame

Nome

`GetAVFrame` – Devolve um ponteiro para a variável m_frame

Sinopse

```
GetAVFrame *Video::GetAVFrame ();
```

Descrição

Essa função retorna um ponteiro para a variável m_frame. Esta variável relaciona-se com o quadro que compõe o vídeo. O conteúdo desse ponteiro varia de acordo com a exibição do stream.

I.4.29

SetAVFrame

Nome

`SetAVFrame` – Escreve no ponteiro *m_frame

Sinopse

```
void Video::SetAVFrame(AVFrame *frame);
```

Descrição

Responsável por copiar o ponteiro *frame para o atributo *m_frame.

I.5

Classe Audio

Arquivos comuns à compilação:

```
#include <list>
#include <string>
#include <ffmpeg/avformat.h>
#include <ffmpeg/avcodec.h>
```

```
#include <ffmpeg/avutil.h>
#include <SDL.h>
#include "codec.h"
#include "multimediaStream.h"
#include "audio.h"
```

I.5.1

GetStreamIndex

Nome

`GetStreamIndex` – Devolve o valor da variável m_streamIndex.

Sinopse

```
unsigned short int Audio::GetStreamIndex ();
```

Descrição

Retorna o índice que identifica o stream de áudio dentro do arquivo multimídia.

I.5.2

SetStreamIndex

Nome

`SetStreamIndex` – Escreve na variável m_streamIndex

Sinopse

```
void Audio::SetStreamIndex (unsigned short int
streamIndex);
```

Descrição

Responsável por escrever na variável m_streamIndex o índice do stream de áudio, representado por streamIndex.

I.5.3

GetCodecID

Nome

`GetCodecID` – Devolve o valor da variável m_codecID

Sinopse

```
unsigned short int Audio::GetCodecID ();
```

Descrição

Retorna o índice do codec referente aos padrões suportados pela biblioteca libavcodec. O índice gravado identifica o padrão usado na decodificação do áudio reproduzido.

I.5.4

SetCodecID

Nome

`SetCodecID` – Escreve na variável m_codecID

Sinopse

```
void Audio::SetCodecID (unsigned short int codecID);
```

Descrição

Responsável por escrever na variável m_codecID o índice do codec que será usado na decodificação do áudio.

I.5.5

GetBitRate

Nome

`GetBitRate` – Devolve o valor da variável m_bitrate.

Sinopse

```
unsigned int Audio::GetBitRate ();
```

Descrição

Retorna o valor da taxa, em bits por segundo, do stream de áudio.

I.5.6

SetBitRate

Nome

`SetBitRate` – Escreve na variável m_bitrate

Sinopse

```
void Audio::SetBitRate (unsigned int bitrate);
```

Descrição

Responsável por escrever a taxa, bitrate, de codificação do stream de áudio.

I.6

Classe Multimedia

Arquivos comuns à compilação:

```
#include <iostream>
#include <sstream>
#include <list>
#include <string>
#include <ffmpeg/avformat.h>
#include <ffmpeg/avcodec.h>
#include <ffmpeg/avutil.h>
#include <SDL.h>
#include <wx/wx.h>
#include <wx/dcbuffer.h>
#include <wx/dcclient.h>
#include "codec.h"
#include "multimediaStream.h"
#include "video.h"
#include "audio.h"
#include "multimedia.h"
```

I.6.1

DumpInformation - Experimental

Nome

`DumpInformation` – Escreve no parâmetro str as informações básicas do stream exibido.

Sinopse

```
void Multimedia::DumpInformation (std::string str);
```

Descrição

Essa função escreve na string str as características de todo o conteúdo que será exibido. A string str pode conter: Nome dos codecs de áudio e vídeo, taxas de codificação de áudio e vídeo, número de canais de áudio, resolução e formato do vídeo, índices de cada stream, etc.

I.6.2

PlayAll

Nome

`PlayAll` – Reproduz todo o stream.

Sinopse

```
void Multimedia::PlayAll ();
```

Descrição

Essa função existe apenas por compatibilidade com as versões experimentais, pois comanda uma reprodução sem considerar a taxa de quadros por segundo. Não é usada na ferramenta CrossPlayer.

I.6.3

LoadPlayMediaInfo

Nome

`LoadPlayMediaInfo` – Carrega informações sobre o stream multimídia.

Sinopse

```
void Multimedia::LoadPlayMediaInfo ();
```

Descrição

Essa função reúne as chamadas para carregar as informações de cada categoria (vídeo/áudio/dados) a partir do flag de exibição. Por exemplo, se apenas o vídeo for exibido, essa função chama apenas Video::LoadVideoInfo().

I.6.4

SeekMedia

Nome

`SeekMedia` – Reposiciona o ponto de reprodução.

Sinopse

```
void Multimedia::SeekMedia (int64_t pos, int flags);
```

Descrição

Essa função posiciona o arquivo multimídia em uma determinada posição pos que indica o número do quadro de vídeo. Ela é utilizada na ferramenta Cross-Player quando, através da barra (slicer), ajustamos precisamente o número do quadro. Utiliza uma outra função da biblioteca libavformat (av_seek_frame) que também recebe como parâmetro a variável flags cujo valor é AVSEEK_FLAG_ANY.

I.6.5

GetDuration

Nome

`GetDuration` – Devolve o valor da variável m_duration.

Sinopse

```
unsigned int Multimedia::GetDuration ();
```

Descrição

Retorna a duração do stream multimídia.

I.6.6

SetDuration

Nome

`SetDuration` – Escreve na variável m_duration.

Sinopse

```
void Multimedia::SetDuration (unsigned int dur);
```

Descrição

Responsável por determinar a duração do stream de vídeo. Esse valor representa quantos quadros existem. A duração em segundos pode ser obtida dependendo da taxa de exibição (fps). Portanto, a reprodução levará dur/fps segundos.

I.6.7

GetCurrentSecond

Nome

`GetCurrentSecond` – Devolve o valor da variável m_currentSecond

Sinopse

```
unsigned int Multimedia::GetCurrentSecond ();
```

Descrição

Retorna o valor instantâneo, em segundos, do tempo de reprodução.

I.6.8

SetCurrentSecond

Nome

`SetCurrentSecond` – Escreve na variável m_currentSecond.

Sinopse

```
void Multimedia::SetCurrentSecond (unsigned int sec);
```

Descrição

É responsável por determinar o valor da variável m_currentSecond que indica, em segundos, o tempo de reprodução.

I.6.9

GetCurrentFrame

Nome

`GetCurrentFrame` – Devolve o valor da variável m_currentFrame.

Sinopse

```
unsigned int Multimedia::GetCurrentFrame ();
```

Descrição

Retorna o número do quadro que está sendo exibido.

I.6.10

SetCurrentFrame

Nome

`SetCurrentFrame` – Escreve na variável m_currentFrame.

Sinopse

```
void Multimedia::SetCurrentFrame (unsigned int fr);
```

Descrição

Responsável por definir o número do quadro corrente (fr).

I.6.11

GetTimeStamp

Nome

`GetTimeStamp` – Devolve o valor da variável m_timestamp.

Sinopse

```
int Multimedia::GetTimeStamp ();
```

Descrição

Retorna o período de um quadro (1/fps) multiplicado por 1000.

I.6.12

GetTotalFrames

Nome

`GetTotalFrames` – Devolve o valor da variável m_totalFrames

Sinopse

```
int Multimedia::GetTotalFrames ();
```

Descrição

Retorna o número total de quadros existentes no stream de vídeo.

Apêndice II

Estruturas e Funções FFMPEG

II.1

Estruturas

Tipo	Descrição
AVFormatContext	Esta estrutura representa os formatos dos arquivos, sejam eles de entrada ou saída. Contém informações como o número total de streams, taxa total de bits, tamanho do arquivo, etc.
AVCodecContext	Esta estrutura é a interface principal com o contexto do codificador ou decodificador. Se o caso for codificação de vídeo, a estrutura possui campos que informam a dimensão do quadro, configuração de GOP, fatores de quantização, etc.
AVCodec	Trata-se de uma estrutura que faz parte do processo de codificação/decodificação. Na verdade, o parâmetro que pode ser consultado é o name, que informa o nome do codificador em questão. Normalmente as funções internamente definidas são usadas nas chamadas externas como OpenCodec, CloseCodec, etc.
AVFrame	Contém o quadro do vídeo
AVPacket	É a estrutura usada para ler e escrever os pacotes de bitstream do vídeo codificado.
AVPicture	Contém o quadro do vídeo num determindado formato, por exemplo, RGB32, YUV420, etc.

II.2

Apresentação das Funções

Função	Descrição
av_alloc_format_context	Reserva memória para a estrutura AVFormatContext e retorna um ponteiro para ela.
av_close_input_file	Fecha um arquivo aberto ou criado durante a execução do programa que fica associado à estrutura fcx.
av_free	Libera memória do endereço apontado por *p.
av_init_packet	Inicializa a estrutura apontada por *pkt
av_new_stream	Aloca memória para um novo stream dentro de fcx. É preciso definir um identificador id.
av_read_frame	Faz a leitura do quadro seguinte do arquivo multimídia. Retorna 0 se OK, ou menor que 0 se houver erro ou chegar no final do arquivo. Para casos de leitura de vídeo, *pkt aponta para o endereço de um quadro. No caso de áudio, pkt contém mais de um frames (cada um de tamanho conhecido - PCM, ADPCM). Se o frame de áudio tem o tamanho variável, então pkt contém apenas um frame.
av_register_all	Inicializa libavcodec e registra todos os codecs e formatos.
av_rescale_q	Reescala um inteiro de 64 bits em dois números racionais.
av_set_parameters	Configura os parâmetros *params, no formato *fcx
av_write_frame	Escreve um pacote de dados, áudio ou vídeo, no arquivo multimídia de saída, indicado por fcx. Retorna negativo caso ocorra um erro, 0 se OK e 1 se for o final do stream.
av_write_header	Aloca os dados específicos de cada stream e escreve o cabeçalho no arquivo de saída
av_write_trailer	Termina de escrever os parâmetros do formato do arquivo multimídia, incluindo sua duração. Retorna 0 se OK.
avcodec_alloc_context	Reserva memória para a estrutura AVCodecContext e retorna um ponteiro para ela.
avcodec_alloc_frame	Aloca uma estrutura AVFrame
avcodec_close	Fecha o contexto de codificação apontado por *ccx
avcodec_decode_video	Decodifica um pedaço do stream de vídeo que está associado às informações de *data* com o tamanho size e guarda o resultado na estrutura representada pela variável *frame* . Para isso usa o padrão representado por *codec* . Retorna -1 se ocorrer algum erro, ou retorna o número de bytes usados na decodificação.
avcodec_encode_video	Codifica o quadro apontado por *frame* usando a codificação de acordo com *ccx* e guarda as informações em outbuf. Também deve ser informado o tamanho deste array como é feito com outbuf_size
avcodec_find_decoder	Busca dentre todos os padrões de codificação suportados aquele que possui um identificador igual a *id* e retorna um ponteiro para essa estrutura.
avcodec_find_encoder	Busca dentre todos os padrões de codificação suportados aquele que possui um identificador igual a *id* e retorna um ponteiro para essa estrutura.
avcodec_get_frame_defaults	Escreve em frame, os parâmetros defaults.
avcodec_init	Inicializa os codecs implementados na versão.
avcodec_open	Inicializa/Abre a estrutura apontada por *ccx.
avpicture_free	Libera a memória da estrura AVPicture apontada por *picture.
dump_format	Usado para apresentar as configurações do stream index, contido em fcx, na tela do usuário.
guess_format	Procura adivinhar o formato do arquivo multimídia pela sua extensão, ou pelo mime_type).

Sinopse

Retorno	Função	Parâmetros
AVFormatContext *	**av_alloc_format_context**	(void)
void	**av_close_input_file**	(AVFormatContext *fcx)
void	**av_free**	(void *p)
void	**av_init_packet**	(AVPacket *pkt)
AVStream *	**av_new_stream**	(AVFormatContext *fcx, int id)
int	**av_read_frame**	(AVFormatContext *fcx, AVPacket *pkt)
void	**av_register_all**	(void)
int64_t	**av_rescale_q**	(int64_t a, AVRational bq, AVRational cq)
int	**av_set_parameters**	(AVFormatContex *fcx, AVFormatParameters *params)
int	**av_write_frame**	(AVFormatContext *fcx, AVPacket *pkt)
int	**av_write_header**	(AVFormatContext *fcx)
int	**av_write_trailer**	(AVFormatContext *fcx)
AVCodecContext *	**avcodec_alloc_context**	(void)
AVFrame *	**avcodec_alloc_frame**	(void)
int	**avcodec_close**	(AVCodecContext *ccx)
int	**avcodec_decode_video**	(AVCodec *codec, AVFrame frame, int *ret, int *data, int size)
int	**avcodec_encode_video**	(AVCodecContext *ccx, int *outbuf, int outbuf_size, AVFrame *frame)
AVCodec *	**avcodec_find_decoder**	(CodecID id)
AVCodec *	**avcodec_find_encoder**	(enum CodecID id)
void	**avcodec_get_frame_defaults**	(AVFrame *frame)
	avcodec_init	(void)
int	**avcodec_open**	(AVCodecContext *ccx, AVCodec *codec)
void	**avpicture_free**	(AVPicure *picture)
void	**dump_format**	(AVFormatContext *fcx, int index, const char *url, int is_output)
AVOutputFormat *	**guess_format**	(const char *short_name, const char *filename, const char *mime_type)

Anexo A

Cotações

A.1

Placa MiniDVR

Perth Technology

Master Dealer

" **Preço, Capacitação, Fidelidade e Qualidade !** "

ORÇAMENTO

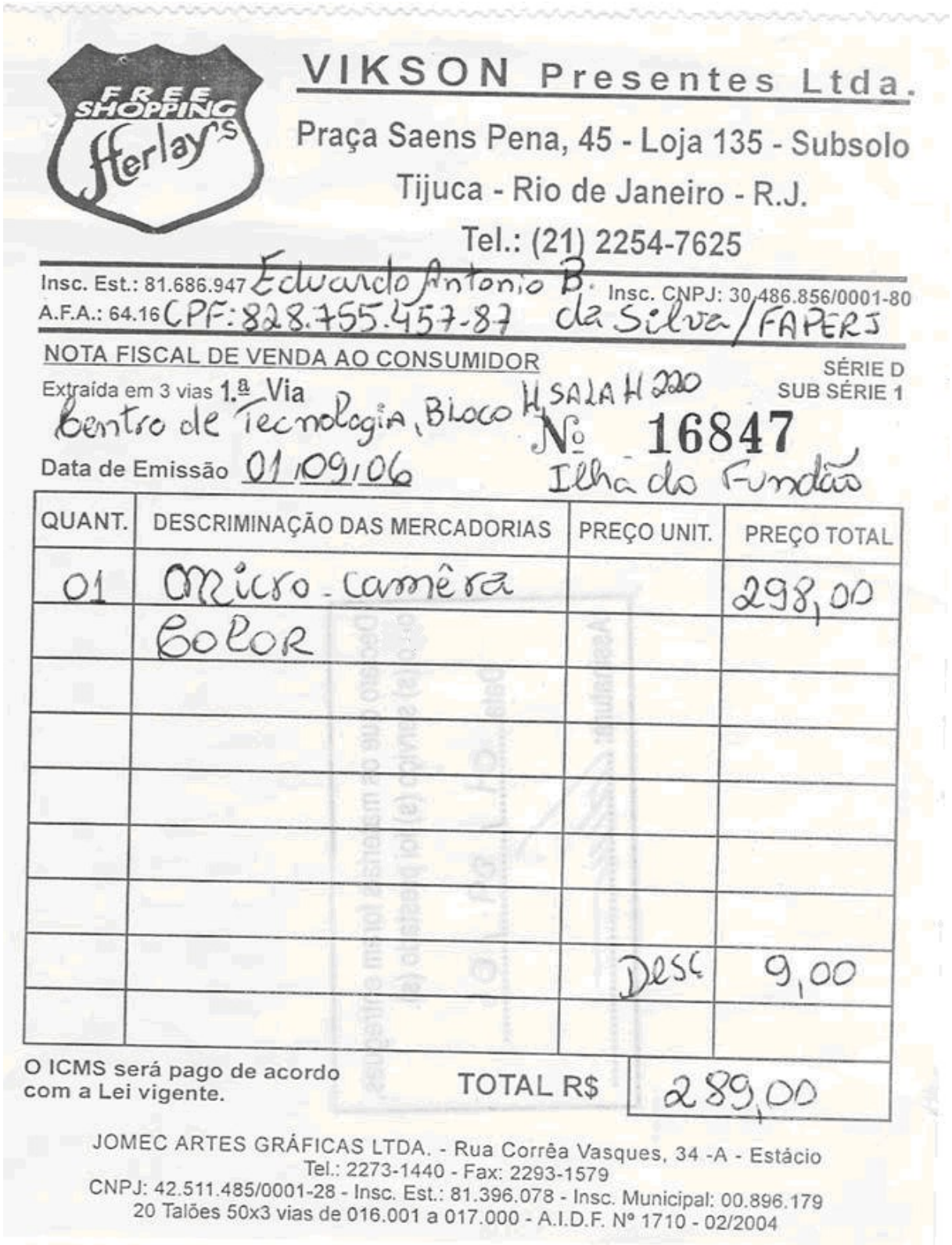
A/C Leonardo Chaves
Rio de Janeiro, 21 de julho de 2006.
Validade da proposta: 5 dias

Qtde	Descrição	Unid	Valor Unitário (R$)	Valor Total (R$)
01	Placa Mini DVR Tecvoz 4 canais / 30 fps	Pç	-	235,00

Rua República do Líbano, 7 – Centro – Rio de Janeiro – RJ
CEP 20061-030 - Tel.: (21) 2111-4986
CNPJ: 04.834.679/0001-75
Insc. Essadual: 77.598.677
Email: comercial@perthtechnology.com.br
Site: www.perthtechnology.com.br
Blog: http://blogdaperth.blogspot.com/

A.2

Câmera de Segurança



FREE SHOPPING Herlay's

VIKSON Presentes Ltda.

Praça Saens Pena, 45 - Loja 135 - Subsolo
Tijuca - Rio de Janeiro - R.J.
Tel.: (21) 2254-7625

Insc. Est.: 81.686.947 Eduardo Antonio B. Insc. CNPJ: 30.486.856/0001-80
A.F.A.: 64.16 CPF: 828.755.457-87 da Silva / FAPERJ

NOTA FISCAL DE VENDA AO CONSUMIDOR

SÉRIE D
SUB SÉRIE 1

Extraída em 3 vias 1.ª Via Centro de Tecnologia, Bloco H, SALA H 220

Nº 16847

Data de Emissão 01/09/06 Ilha do Fundão

QUANT.	DESCRIMINAÇÃO DAS MERCADORIAS	PREÇO UNIT.	PREÇO TOTAL
01	Micro - Camêra		298,00
	Color		
		Desc	9,00

O ICMS será pago de acordo com a Lei vigente.

TOTAL R$ 289,00

JOMEC ARTES GRÁFICAS LTDA. - Rua Corrêa Vasques, 34 -A - Estácio
Tel.: 2273-1440 - Fax: 2293-1579
CNPJ: 42.511.485/0001-28 - Insc. Est.: 81.396.078 - Insc. Municipal: 00.896.179
20 Talões 50x3 vias de 016.001 a 017.000 - A.I.D.F. Nº 1710 - 02/2004