
Criando Dispositivos Embarcados com Linux

Vitor Neves Garbellini

Centro Universitário Barão de Mauá
Curso de Ciência da Computação
Ribeirão Preto - SP
<http://www.baraodemaua.br/>

VITOR NEVES GARBELLINI

Criando Dispositivos Embarcados com Linux

Monografia apresentada ao Curso de Ciência da Computação do Centro Universitário Barão de Mauá, como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Marcelo Barros de Almeida

Ribeirão Preto - SP
Centro Universitário Barão de Mauá
Curso de Ciência da Computação
Dezembro - 2006

Agradecimentos

Primeiramente a DEUS pela grandiosa força para a conclusão deste curso.

Ao meu orientador Prof. Dr. Marcelo Barros de Almeida pela paciência, instrução e pelo grande conhecimento passado. Com sua ajuda que consegui estudar esta área.

Aos meus pais, Maria Arminda e Percio, ao meu irmão, Henrique, a minha namorada Camila e a toda minha família, que me ajudaram a concluir mais uma etapa da minha vida.

A todos os professores do Centro Universitário Barão de Mauá, que contribuíram de alguma forma na minha formação.

Aos colegas pela convivência e amizade durante o curso.

Resumo

Os sistemas embarcados são computadores de propósito específico, atendem a tarefas bem definidas e específicas. Estes dispositivos embarcados estão presentes nas mais diversas áreas, dentre eles podemos destacar celulares, televisores, aviões, controladores industriais, etc. Podem ser projetados sob várias plataformas e sistemas operacionais. Um ponto fundamental são as licenças, alguns admitem licença livre, outros royalties por unidades produzidas e patentes envolvidas.

O Linux é uma alternativa viável para um projeto de sistema embarcado, como é livre, se torna um sistema operacional muito customizável, com a capacidade de se moldar exatamente para atender os requisitos de uma determinada aplicação. Alguns conceitos são importantes para a construção de sistemas embarcados com Linux, muitos provem do seu processo de inicialização. Com os conceitos apresentados, um exemplo prático de um sistema Linux sob uma plataforma x86 é totalmente emulado com o auxílio de um programa denominado Qemu. O qual é suficiente para o entendimento dos conceitos apresentados ao decorrer do trabalho.

A grande demanda gera um mercado sólido, que está em constante crescimento. A tendência da competição no mercado, coloca o Linux como uma solução para os fabricantes reduzirem seus custos. Esta alternativa ainda não é muito explorada no Brasil, diferentemente de países mais evoluídos tecnologicamente.

Palavras chave: kernel, initrd e rootfs.

Abstract

Embedded systems are specific purpose computers, designed to attend particular tasks. These embedded devices are present in a large number of applications and equipment, such as TVs, airplanes, industrial controllers, and so. They may be designed under several platforms and operating systems, some of them presented in this work. Licenses are a fundamental point when choosing the operation system, since some projects admit free licenses while others prefer to pay royalties per unit produced or to deal with patents issues.

Nowadays, Linux is a viable option for an embedded system project. Besides being free, it is a very customizable operational system, able to mold itself to attend a determined application's requirement. Several concepts, important for building embedded systems with Linux, are discussed in the present work. Moreover, it is showed how a practical example of Linux system under a x86 platform may be totally emulated with the help of a program called Qemu. This example certifies the understanding of the concepts presented in this work.

Although the presence of Linux in embedded systems is constantly growing in the world, this alternative is still not well exploited in Brazil.

Keywords: kernel, initrd and rootfs

Sumário

Agradecimentos	i
Resumo	ii
Abstract	iii
Lista de Figuras	viii
Lista de Tabelas	ix
Lista de Abreviaturas e Siglas	x
1 Introdução	1
1.1 O que é um sistema embarcado	2
1.1.1 As Funcionalidades na Vida dos Usuários	4
1.2 Mercado	4
1.3 Motivação	8
1.4 Objetivos do Trabalho	9
1.5 Organização do Trabalho	10
2 Conceitos Básicos em Sistemas Embarcados	11
2.1 Introdução	11
2.2 Ambientes de Desenvolvimento	12
2.2.1 Estações Linux	13
2.2.2 Estações Windows	13
2.3 Plataformas Alvo	14
2.3.1 ARM	14
2.3.2 ColdFire	16
2.3.3 Motorola/IBM PowerPC	17

2.3.4	x86	17
2.4	Sistemas Operacionais Relacionados	18
2.4.1	Microsoft Windows CE	19
2.4.2	Linux	21
2.4.3	eCos	21
2.4.4	Inferno	22
2.4.5	uClinux	23
2.5	Questões Legais: Diretos e Deveres	25
2.5.1	GPL e LGPL	26
2.5.2	Licença BSD	27
2.5.3	Licença X Embarcados	28
2.6	Exemplos	28
2.7	Conclusão	32
3	Sistemas Embarcados com Linux	33
3.1	Vantagens e Desvantagens	33
3.2	Linux Embarcado X Linux Comum	35
3.2.1	Bibliotecas C	36
3.2.2	Ambiente Gráfico	38
3.3	Processo de Inicialização	38
3.3.1	Kernel Linux	40
3.3.2	initrd	43
3.3.3	rootfs	43
3.4	Personalizando o kernel Linux	44
3.4.1	Compilação	46
3.4.2	Descompactação/Execução em RAM	46
3.4.3	Tempo de Partida	47
3.4.4	Unindo Tudo Com Buildroot	48
3.4.5	Sistema Raiz com Busybox	49
3.5	Supporte à Tempo Real	49
3.5.1	Real Time Linux (RT-Linux)	51
3.5.2	Linux/RK	52
3.6	Toolchains	53
3.7	Toolkits Gráficos	54
3.7.1	DirectFB	55
3.7.2	Qtopia	55
3.7.3	TinyX	56
3.7.4	Conclusão	57

4 Linux Embarcado na Prática	58
4.1 Qemu	59
4.1.1 Instalação do Qemu	59
4.2 Configurando e Compilando o Kernel Linux	60
4.2.1 Personalizando o Kernel	61
4.2.2 Compilação	77
4.3 Busybox	77
4.3.1 Personalizando o Busybox	78
4.3.2 Compilação do Busybox	78
4.3.3 Conclusão	83
5 Conclusão	85
5.1 Trabalhos Futuros	87
Referências Bibliográficas	89

Lista de Figuras

1.1	Quais Fontes do Linux São Mais Utilizadas	5
1.2	Tipos de Distribuições Mais Utilizadas	6
1.3	A Empresa Quer Pagar Por Suporte ou Serviços? . .	7
1.4	A Empresa Quer Pagar Por Royalties?	8
2.1	Plataformas Mais Utilizadas e a Previsão de Uso . .	15
2.2	Modelos e Tamanhos dos Principais Tipos de Placas x86	18
2.3	Sistemas Operacionais Mais Utilizados	19
2.4	Tela de Funcionalidades 1	29
2.5	Tela de Funcionalidades 2	30
2.6	Drew Tech DashDAQ Funcionando	30
2.7	Netstix 400XM-CF	31
2.8	Jack-PC EFI-6900	31
3.1	Processo de Boot	39
3.2	Visual Gráfico Com Qtopia	56
4.1	Plataformas Emuladas Pelo QEMU	60
4.2	Plataformas Suportadas Pelo QEMU	61
4.3	Utilizando o comando make config	63
4.4	Utilizando o comando make menuconfig	63
4.5	Utilizando o comando make xconfig	64
4.6	Configurando o Kernel - x86 1/2	64
4.7	Configurando o Kernel - x86 2/2	65
4.8	Configurando o Kernel - PCI 1/2	65
4.9	Configurando o Kernel - PCI 2/2	66
4.10	Configurando o Kernel - elf 1/2	66
4.11	Configurando o Kernel - elf 2/2	67

4.12	Configurando o Kernel - Rede 1/3	67
4.13	Configurando o Kernel - Rede 2/3	68
4.14	Configurando o Kernel - Rede 3/3	68
4.15	Configurando o Kernel - Drivers	69
4.16	Configurando o Kernel - RAM 1/2	69
4.17	Configurando o Kernel - RAM 2/2	70
4.18	Configurando o Kernel - IDE 1/2	70
4.19	Configurando o Kernel - IDE 2/2	71
4.20	Configurando o Kernel - Placa de Rede 1/5	71
4.21	Configurando o Kernel - Placa de Rede 2/5	72
4.22	Configurando o Kernel - Placa de Rede 3/5	72
4.23	Configurando o Kernel - Placa de Rede 4/5	73
4.24	Configurando o Kernel - Placa de Rede 5/5	73
4.25	Configurando o Kernel - Suporte a Sistemas de Arquivos	74
4.26	Configurando o Kernel - Ext2	74
4.27	Configurando o Kernel - Proc 1/2	75
4.28	Configurando o Kernel - Proc 2/2	75
4.29	Configurando o Kernel - Linguagem Nativa 1/2 . . .	76
4.30	Configurando o Kernel - Linguagem Nativa 1/2 . . .	76
4.31	Configurando o Busybox com Estático 1/3	78
4.32	Configurando o Busybox com Estático 2/3	79
4.33	Configurando o Busybox com Estático 3/3	80
4.34	Verificando o Tamanho dos Arquivos Gerados	81
4.35	Simulando o Sistema Criado no Qemu	83
4.36	Sistema Carregado no Qemu	83

Lista de Tabelas

1.1	Mercado de Linux Embocado	4
1.2	Windows Embocado X Linux Embocado	9
2.1	Arquiteturas e SOs Compatíveis com Inferno	23
3.1	Evolução do kernel	42
3.2	Estrutura Básica de Diretórios	45
3.3	Exemplos dos Principais Programas	45

Lista de Abreviaturas e Siglas

- . ..ADSL: Asymmetric Digital Subscriber Line
- AP: Acess Ponit
- API: Application Programming Interface
- CD: Compact Disc
- DRM: Digital Rights Management
- FAQ: Frequently Asked Questions
- GB: Gigabyte
- IDE: Integrated Drive Electronics
- ISDN: Integrated Service Digital Network
- LCD: Liquid Crystal Display
- MB: Megabyte
- MBR: Master Partition Table
- MMU: Memory Management Unit
- MP3: MPEG-1/2 Audio Layer 3
- PC: Personal Computer
- PCI: Peripheral Component Interconnect
- PDA: Personal Digital Assistants
- RISC: Reduced Instruction Set Computer
- SCSI: Small Computer System Interface
- SO: Sistema Operacional
- USB: Universal Serial Bus
- VoIP: Voice Over IP

Capítulo 1

Introdução

Com o crescimento da tecnologia, a cada dia uma imensa gama de eletrônicos novos são colocados no mercado com uma velocidade que muitas vezes nem os usuários conseguem acompanhar. Isso atinge também, diretamente, a área de dispositivos embarcados, que tem seu caminho traçado com a evolução da tecnologia.

Os sistemas embarcados são sistemas considerados de propósito específico, que diferentemente dos sistemas de propósito geral, são projetados para atenderem tarefas bem definidas e específicas, estes sistemas tem o objetivo de realizar suas tarefas utilizando o mínimo de recursos de hardware e software reduzindo assim o custo do projeto.

Antigamente, quando a tecnologia não estava tão evoluída como hoje, esses dispositivos eram apenas utilizados em equipamentos onde seu uso era imprescindível como aviões, controladores industriais e linhas de produção. Mas hoje isso mudou muito, com a redução de custo da tecnologia, estes dispositivos ficaram bem mais comuns e estão espalhados nos mais variados produtos em diversos segmentos como carros, celulares, DVD players, geladeiras, televisores, MP3s, etc. A tendência do crescimento desses dispositivos cria um mercado a parte, com uma vasta gama de empresas que se especializaram em

prover soluções em todos os aspectos, para todas as finalidades. A grande maioria destes dispositivos, mesmo se tratando de computadores de fato, jamais serão chamados como tal. Seus usuários muitas vezes nunca terão esse tipo de visão de seu equipamento.

Ao longo deste trabalho será abordada boa parte das opções que um desenvolvedor tem a sua disposição ao iniciar um projeto desta natureza, e a série de escolhas que ele terá que tomar para seu produto final funcionar satisfatoriamente. Será visto desde a base do projeto, como a plataforma embarcada do dispositivo, apresentando as principais e mais conhecidas, até os sistemas operacionais suportados e designados por esses dispositivos. Também se discutirá conceitos importantes como as licenças envolvidas, suas vantagens e desvantagens.

O sistema operacional(SO) escolhido para um estudo mais minucioso foi o Linux, onde será apresentado o tratamento especial para que ele se adeque ao funcionamento destes dispositivos, que em muitos casos são restritos em quase todos os aspectos. Outro aspecto fundamental é apresentar as várias opções e estilos de projetos que poderão ser aplicados utilizando este SO. Uma iniciação prática ao tema será apresentada, em um capítulo específico. A partir dela espera-se que vários conceitos teóricos sejam devidamente elucidados. Para essa explanação aplicaremos um exemplo prático de um sistema embarcado Linux através de um programa de virtualização denominado QEMU. Uma plataforma baseada em processadores x86 pode ser completamente virtualizada com o QEMU.

1.1 O que é um sistema embarcado

Sendo objetivo, um sistema embarcado pode ser considerado um dispositivo que contenha um computador dentro, onde normalmente seu usuário não necessariamente sabe desta existência[Abb03]. Ou, à partir de várias definições, conseguimos extrair que um sistema em-

barcado é composto de um computador de propósito específico que é encapsulado pelo dispositivo que ele controla, diferentemente de um computador comum que é de propósito geral, com diversos requisitos diferentes e recursos disponíveis. Enquanto um sistema embarcado visa seu resultado com o menor custo possível, reduzindo o tamanho da memória e o poder de processamento, um computador pessoal pode rodar diversos programas, armazenar arquivos, acessar a web, música, etc, permitindo que o usuário faça tudo [Fou06a].

A arquitetura dos sistemas embarcados é intencionalmente simplificada e projetada para reduzir custos. Por exemplo, vários sistemas embarcados se comunicam através de portas seriais, essas podem ser até 300 vezes mais lentas que em um computador pessoal. Como um sistema embarcado pode ser produzido em grande escala, quando se fabrica milhões de unidades, uma pequena redução de custo acarreta uma grande economia para seu fabricante, resultando em um lucro maior a empresa[Fou].

Em geral, esses sistemas são restritos e nem sempre pode se contar com memória abundante, monitor, dispositivos de armazenamento, teclado, etc. Isso varia muito de acordo com as especificações do projeto e cabe ao projetista fazer um estudo e analisar a real necessidade da aplicação. Quando se inicia um projeto, deve-se avaliar se o dispositivo será utilizado por um usuário final ou fará parte de uma linha de montagem automatizada. Analises como esta ajudarão a delimitar o escopo e os recursos necessários do sistema embarcado. Em outros casos a aplicação pode necessitar de respostas em tempo real, sem nenhuma possibilidade de erro ou atraso, como em uma nave espacial ou em um míssil por exemplo. Isso também é analisado e seu sistema terá que prover o suporte a respostas em tempo real, admitindo assim um sistema capaz de suportar esses requisitos.

Mercado	Faturamento	Projeção para 2006
Semicondutores	\$165 Bi em 2004	\$206 Bi
SOs,prods,servs	\$250 Mi em 2003	\$1.1 Bi
Linux SOs	\$140 Mi em 2003	\$300 Mi

Tabela 1.1: Mensurando o Mercado (fonte: [Ban04])

1.1.1 As Funcionalidades na Vida dos Usuários

As utilidades providas pelos sistemas embarcados vem mudando até o estilo de vida e os hábitos das pessoas. Por exemplo, hoje em dia através de um celular podemos facilmente passar uma mensagem de texto para qualquer lugar do país, com um custo bem acessível. Muitas crianças não tem mais brinquedos convencionais, passam boa parte da infância na frente de vídeo games de alta tecnologia, que no fundo são computadores, jogando em celulares, escutando seus MP3s, usando suas câmeras digitais, etc.

1.2 Mercado

Hoje a Ásia detém uma estrutura tecnológica invejável na área eletrônicos, o que lhe capacita a fornecer produtos para os maiores mercados do mundo, com um baixo custo de mão de obra e alta produtividade que chega a ser espantosa. Com os embarcados é ainda mais forte esta relação, pois são intimamente ligados com tecnologia. Existem estudos que indicam que o Linux é um dos sistemas operacionais mais utilizados [Som04].

Mais qual o tamanho de mercado? Quanto movimenta? Quais as projeções?, na Tabela 1.1 estão apresentados alguns dados que podem nos ajudar a mensurar este mercado, com informações que nos dão uma noção de como seu crescimento é rápido:

Em pesquisa espontânea realizada pelo site LinuxDevices em 2006,

podemos analisar alguns dados importantes na área de sistemas microprocessados e ter uma idéia do que as empresas buscam e usam no momento, além do porque destas escolhas. Tomaremos esta pesquisa como base apenas para dar uma idéia do mercado, já que existem empresas especializadas em pesquisas desse tipo. E, se contratadas, podem fazer uma pesquisa específica moldada para a sua necessidade, claro que com custos altos para o cliente. O Linux Devices faz este tipo de pesquisa anualmente, por isso alguns gráficos estão com dados de anos anteriores agregados aos obtidos este ano [Hol06a].

A Figura 1.1 mostra, em empresas que utilizam Linux embarcado, quais são as distribuições (comerciais ou não) de sua preferência e quais estão implantadas no momento em seus projetos. Isso vem mudando um pouco em relação ao ano passado, que acompanhava uma tendência do uso de uma grande variedade de fontes diferentes e hoje a tendência, como apresenta o gráfico, é de uma dominação das distribuições mais conhecidas como kernel.org, uCLinux, Debian, etc.

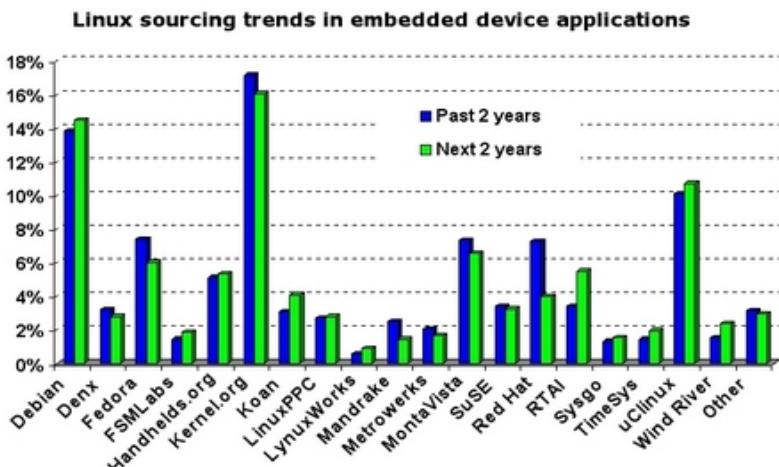


Figura 1.1: Quais fontes Linux mais utilizadas? (fonte: [Hol06a])

A Figura 1.2 mostra as origens do kernel em relação a quatro ca-

tegorias diferentes: comerciais, não comerciais, acompanham o hardware e compartilhados, respectivamente:

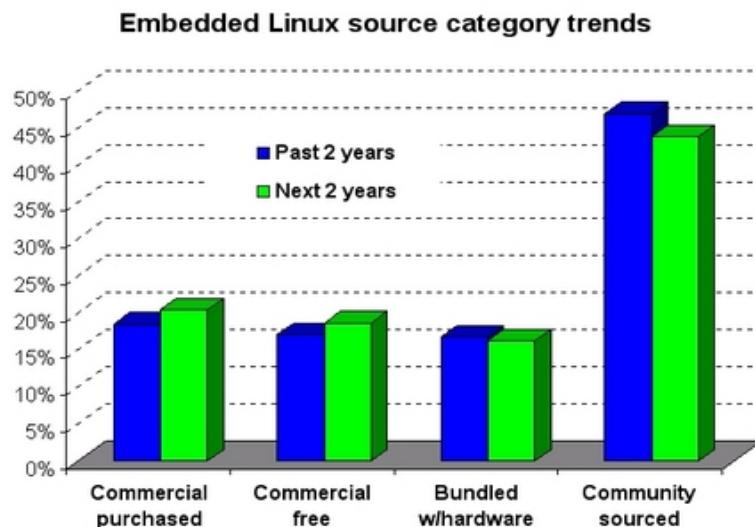


Figura 1.2: Que tipo de distribuição sua empresa utiliza? (fonte: [Hol06a])

Quando perguntadas se as empresas estão dispostas a pagar, se necessário, por suporte ou serviços para Linux embarcado, a grande maioria admite esse tipo de custo ao projeto, como representa a Figura 1.3.

Agora, se o assunto é pagamento de royalties por unidades produzidas, a história se inverte totalmente conforme ilustrado pela Figura 1.4.

Apesar de ser uma pesquisa espontânea, o fato é que o Linux vem crescendo, cada vez mais firmando a sua estabilidade e liderança do mercado, sendo utilizado em milhares de projetos novos a cada ano.

No entanto o uso do Linux ainda é segmentado. Na Tabela 1.2 é apresentada uma comparação entre sistemas embarcados produzidos em várias áreas, evidenciando a porcentagem delas que foram desenvolvidas com Linux e com Windows embarcado. Fica evidente que cada um tem sua força em determinadas áreas do mercado.

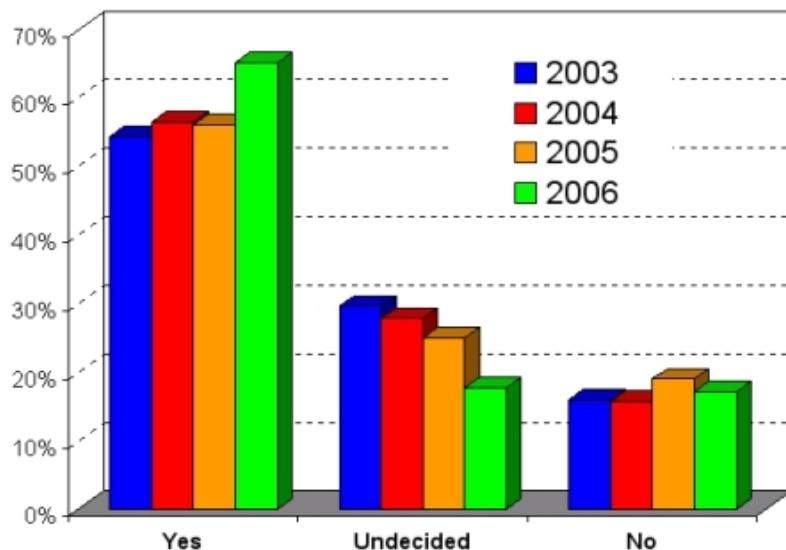


Figura 1.3: Sua empresa está disposta a pagar por suporte ou serviços?(fonte: [Hol06a])

No mercado exterior o Linux já está bem consolidado, porém no mercado brasileiro ainda está começando a ser alvo de interesse das empresas. Grandes fabricantes de microprocessadores para sistemas embarcados estão incentivando a utilização do Linux nestes sistemas no Brasil, como em seminário direcionado a parceiros e clientes que a Freescale promoveu em São Paulo em Setembro de 2006. Este seminário tinha a intenção de mostrar a forte tendência do uso de Linux nesse tipo de solução como explicou Arnaldo Carvalho, um dos fundadores da Conectiva e consultor sênior da Mandriva Conectiva que, ao lado de Thiago Galesi, analista de sistemas e desenvolvedor da empresa, estavam no comando do seminário. “*A escolha do Linux como sistema operacional em um sistema embarcado tem várias vantagens e a maior delas é que, por se tratar de um software livre, ele pode ser ajustado e moldado especificamente para o dispositivo que foi solicitado*“ [Ish06].

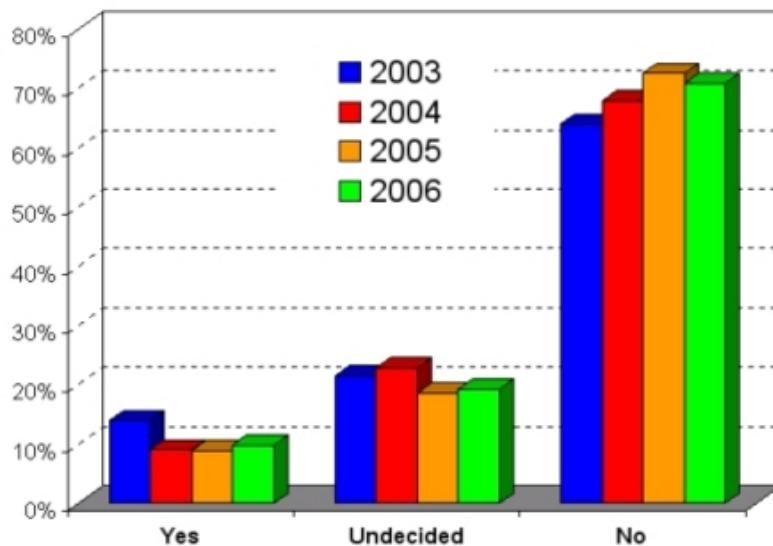


Figura 1.4: Sua empresa está disposta a pagar royalties por unidade produzida? (fonte: [Hol06a])

1.3 Motivação

A principal motivação das empresas ao admitir o Linux como a base de um projeto é que, além de conseguir um sistema moldado exatamente para suas necessidades, o custo é baixo. Além disso não agrega pagamento para utilização, sua licença é livre e seu código é aberto, isto é a empresa pode alterar seu código de acordo com as necessidades de seu projeto.

Com relação ao suporte, ao contrário do que poderia se imaginar, é relativamente grande. Um grande número de usuários e desenvolvedores podem ser contactados através de fóruns e listas de e-mails.

Além disso, o Linux é um sistema estabelecido em mercado, como os de servidores, com amplo suporte a vários tipos de hardware, facilmente personalizável, estável e com baixo índice de falhas.

Tipo de Dispositivo	Win. Embarcado	Linux Embarcado
PDAs, handhelds	120	141
Telefones Celulares	55	30
VoIP	13	15
Robôs	não disp.	11
Áudio e vídeo	23	68
Pequenos dispositivos	45	18
Tablets/webpads	40	18
Gateways,Servs.,APs	não disp.	91
Outros	52	69

Tabela 1.2: Windows Embarcado X Linux Embarcado (fonte: [LD06])

1.4 Objetivos do Trabalho

O objetivo deste trabalho é apresentar uma visão geral dos sistemas embarcados, inicialmente uma visão das diversas plataformas que eles utilizam, obordando opções variadas de sistemas operacionais, com finalidades específicas.

Adicionalmente, este trabalho pretende mostrar a tendência de crescimento destes dispositivos no dia-a-dia, focando o inevitável crescimento da demanda e, através da apresentação dos principais tipos de licença, tentar orientar a melhor escolha para um desenvolvedor.

Finalmente, será discutido o emprego do Linux em sistemas embarcados. O Linux é uma opção de sistema operacional com grande potencial a ser explorado nesta área. Serão apresentadas as suas vantagens e desvantagens, através de simulação, uma demonstração prática de uso. Isto irá munir o leitor dos conceitos necessários para o entendimento deste processo, definir a tendência do uso do Linux nesse meio e o porque desse crescimento.

1.5 Organização do Trabalho

Este trabalho está organizado de forma a permitir uma evolução gradual do leitor no conceito relacionado a Linux embarcado. No capítulo 2 são apresentados vários conceitos gerais em sistemas embarcados e as principais opções de plataforma de hardware e sistemas operacionais. No capítulo 3 é detalhado o processo de construção de sistemas embarcados com Linux. Estes conceitos são colocados em prática na Capítulo 4, onde um exemplo completo de construção de sistema é apresentado. Finalmente, as conclusões são apresentadas no Capítulo 5.

Capítulo 2

Conceitos Básicos em Sistemas Embarcados

O objetivo deste capítulo é apresentar uma visão inicial na área de sistemas embarcados, partindo de algumas definições básicas, para então apresentar algumas opções de hardware e software bem conhecidas que são suportadas e muito utilizadas em sistemas embarcados. Também serão evidenciadas as licenças que regem softwares livres e bibliotecas, muito utilizados na área.

2.1 Introdução

A área de sistemas embarcados é relativamente grande e com uma terminologia bastante específica. Por mais que os sistemas embarcados guardem semelhança com sistemas tradicionais, a quantidade de pontos onde eles diferem é considerável. Neste capítulo serão apresentados os conceitos básicos relacionados a sistemas embarcados, sua terminologia e jargões.

Também serão abordadas algumas plataformas e sistemas operacionais mais utilizados nos mais variados tipos de projetos embarcados, mostrando suas características específicas e ideais para cada

tipo de uso, além de algumas relevantes para determinados tipos de dispositivos embarcados.

2.2 Ambientes de Desenvolvimento

O ambiente de desenvolvimento é um computador escolhido pelo desenvolvedor para criar/desenvolver o sistema embarcado. Isso significa que essa será a sua base, onde contará com inúmeras ferramentas e recursos para utilizar em seu projeto, os quais normalmente ficam inviáveis em um sistema embarcado.

Ao se iniciar um novo projeto embarcado, algumas decisões relacionadas ao ambiente de desenvolvimento precisam ser tomadas. As restrições de memória, disco e processamento de um sistema embarcado em geral o colocam numa posição desfavorável para que o desenvolvimento seja feito diretamente nele. A situação comum é aquela onde todo o desenvolvimento é realizado em um ambiente de desenvolvimento denominado de hospedeiro, ou *host*. Este ambiente é geralmente farto em recursos computacionais, tornando ágil o desenvolvimento. O sistema embarcado, de certa forma, é usado mais para testes finais e depurações específicas da plataforma.

Em seu ambiente, o desenvolvedor pode contar com ferramentas de muita utilidade para um projeto desse tipo, como simuladores e emuladores, com os quais pode-se facilmente emular uma determinada plataforma e/ou simular um determinado sistema operacional, em cima do SO instalado no ambiente. Uma outra ferramenta que será imprescindível, dependendo do projeto, será o compilador cruzado. O compilador cruzado é utilizado em projetos que as plataformas, de desenvolvimento e destino, são diferentes. Sua função é gerar executáveis compatíveis com a plataforma destino, mais detalhes serão apresentados no capítulo seguinte.

Para uma completa abordagem, inicialmente trataremos das plataformas de desenvolvimento mais utilizadas em sistemas embarca-

dos, salientando seu uso e vantagens, para então detalharmos as *targets*. A *target* é o dispositivo final, será o destino da aplicação, onde ela será executada.

2.2.1 Estações Linux

Estações com Linux são mais encontradas em projetos que utilizam o próprio Linux, devido a intimidade necessária com o sistema para atingir resultados satisfatórios, sem contar com a variedade de hardware suportada pelo Linux. Um PC comum, rodando uma das muitas distribuições Linux disponíveis, como Debian, Mandrake, Slackware ou Suse, pode satisfazer as necessidades do projeto.

Em geral vai ser da escolha do desenvolvedor a distribuição que será utilizada, a que o mesmo se adapta melhor. O que vai variar mesmo é o compilador cruzado do sistema que terá que ser compatível com a plataforma alvo.

2.2.2 Estações Windows

Apesar de ser pago, o Windows é muito utilizado por empresas como ambiente de desenvolvimento padrão. Ele é adequado a aplicações Windows Embedded, pelo fato de ter uma alta integração entre produtos do mesmo fabricante, facilitando o desenvolvimento.

A ferramenta de desenvolvimento mais recente e que suporta os principais sistemas operacionais para embarcados Microsoft é o Windows Embedded DevWire. Outra alternativa que também é muito utilizada é o Windows Embedded Studio, ambos possuem uma interface com o usuário simples. A programação nesse tipo de ambiente é relativamente de alto nível.

Vale lembrar que um ambiente de desenvolvimento pode trabalhar em um projeto que se utiliza de outro sistema operacional, por exemplo, utilizar o Windows para desenvolver aplicações para um sistema embarcado Linux.

2.3 Plataformas Alvo

A plataforma alvo é o objeto final do projeto embarcado, onde realmente a aplicação irá ser executada. A escolha de uma plataforma para um sistema embarcado não é fácil, dada a diversidade de placas, processadores, barramentos, etc. A escolha irá depender grandemente da área de aplicação, do poder de processamento necessário, do tamanho do projeto, entre outros parâmetros. Como exemplo, serão abordadas nas próximas seções algumas plataformas embarcadas comuns e muito utilizadas hoje no mercados de sistemas microprocessados.

As plataformas destino são disponibilizadas no mercado em diferentes plataformas, cada uma com sua característica própria, algumas mais indicadas para aplicações bem pesadas, outras são mais eficazes para um sistema mais simples e compacto.

Abordaremos as principais plataformas que dominam grande parte do mercado de sistemas microprocessados, e que são aplicadas em diversos projetos nas mais variadas áreas.

Em pesquisa espontânea realizada pelo site Embedded [Hol06a], podemos constatar as plataformas mais utilizadas no momento e a previsão de uso nos próximos 2 anos, conforme mostra a Figura 2.1.

As seções seguintes discutirão algumas dessas opções.

2.3.1 ARM

Esta plataforma foi inicialmente desenvolvida pela Arcon Computers, sua primeira versão comercial surgiu em 1986. Seus processadores são de 32 bits, usam 16 registradores de uso geral podendo admitir um conjunto de instruções extensível, fazendo o uso de co-processadores (suporta até 16) [PHG05]. É um típico processador RISC com suporte à gerenciamento de memória (MMU). Um ponto forte é o seu baixo consumo de energia.

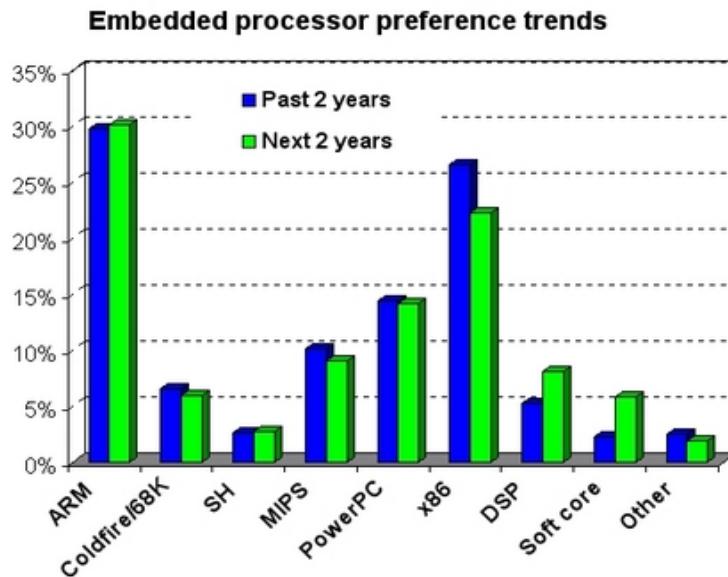


Figura 2.1: Plataformas Mais Utilizadas e a Previsão de Uso. (fonte: [Hol06a])

Na década de 90, a plataforma ARM tornou-se líder de mercado de processadores embarcados de alto desempenho com baixo consumo de energia. Em 98 foi introduzida nas bolsas de Londres e NASDAQ [dR04].

Ao contrário dos outros fabricantes, hoje a ARM Holdings Ltd. projeta *cores* de processadores. Isto é, desenvolve a estrutura básica do processador (*core*) e licencia-os grandes empresas que os utilizam em seus produtos, dentre elas Samsung, Intel e Toshiba. Conseqüentemente, existe uma extensa gama de diferentes modelos espalhados nos mais diversos sistemas embarcados.

A ARM classifica os seus processadores em diversas famílias, entre elas:

- ❑ ARM1, ARM2, ARM3.
- ❑ ARM6, ARM60, ARM610.

- ❑ ARM7, ARM7D, ARM7DI, ARM7 10, ARM7500, ARM7TDMI, ARM720T.
- ❑ ARM9TDMI, ARM940T, ARM9E, ARM948E, ARM968E, ARM926EJ.
- ❑ ARM10, ARM1020E.

Em muitas de suas variantes, os processadores possuem basicamente o mesmo conjunto de instruções, portanto ficam compatíveis com os mesmos softwares, ou a compatibilidade pode ser conseguida através de pequenas alterações.

Algumas aplicações com processadores ARM:

- ❑ Gameboy Advance Games Console
- ❑ PDAs e Pockets PCs de diversos fabricantes Compaq/HP/Ericsson
- ❑ Modems ADSL/Cable D-Link, Zoom, etc
- ❑ Celulares de diversas marcas

2.3.2 ColdFire

É fabricado pela Motorola, um dos líderes mundiais na fabricação de sistemas e serviços eletrônicos avançados [ANP04]. Esse microprocessador foi muito utilizado, mas com a chegada da família ARM a concorrência aumentou muito e boa parte do seu mercado foi perdida, conforme visto em pesquisa de mercado apresentada no Capítulo 1.

Alguns exemplos e suas especialidades:

ColdFire V4: equipa uma linha de impressoras à laser da HP.

ColdFire MCF547X: Microprocessadores com tecnologia de ponta, projetados para aplicações de rede com suporte a encriptação.

ColdFire MCF525X: Variante de controlador projetado para aplicações ligadas a áudio, utilizado em players portáteis e soluções automotivas com suporte a CD, HDD e USB.

ColdFire MCF5223X: Microprocessadores com um custo viável para utilização em controle de aplicações industriais via rede, suporte a Ethernet e encriptação. Disponibiliza 256K de memória flash [Sem06].

2.3.3 Motorola/IBM PowerPC

Esta arquitetura é proveniente da colaboração de 3 empresas: IBM, Motorola e Apple, que em 1991 projetaram esta arquitetura para servir como base para a próxima geração de computadores pessoais IBM e Macintosh. Posteriormente essa parceria terminou.

Uma característica dela é que se divide em três níveis/ambientes de programação UISA, VEA e OEA [Met04]. Comparado com x86 e ARM, o PowerPC é uma arquitetura bem compatível com Linux. Isso se confirma devido a grande quantidade de processadores Modelo PowerPC 603e, que é muito utilizado em sistemas móveis, notebooks e PCs.

2.3.4 x86

Esta plataforma iniciou-se através dos primeiros micros 386, introduzidos no mercado pela Intel, que posteriormente lançou descendentes muito conhecidos como o 486 e o Pentium. Outros fabricantes também produziram processadores compatíveis, tais como a AMD e National Semiconductcor, mas a Intel ainda é a principal referência e a maior produtor de processadores desta família.

Esta é a mais popular plataforma usada hoje para rodar Linux, porém representa uma pequena fatia do mercado tradicional de sistemas embarcados onde seus projetistas dão preferência para as plataformas ARM e PowerPC, devido a baixa complexidade e menores custos.

Uma vantagem de usar esta plataforma é que seu sistema embarcado será igual ou muito similar a um host de desenvolvimento ou

um servidor, apenas com pequenas mudanças no kernel para suporte adequado para placas e processadores específicos

Quando se trata de projetos embarcados com a plataforma x86, não é comum utilizarmos as mesmas placas utilizadas em um computador convencional, existe uma grande variedade de opções diversos tamanhos e recursos, como ilustra a Figura 2.2. Hoje a VIA representa uma nova força no mercado, com processadores x86 de baixo custo.

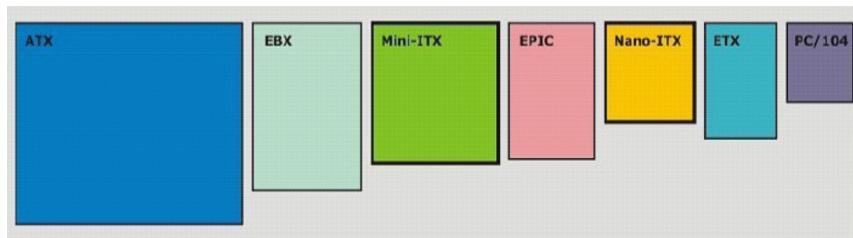


Figura 2.2: Modelos e Tamanhos dos Principais Tipos de Placas x86.
(fonte: [Dev06])

2.4 Sistemas Operacionais Relacionados

Com a crescente demanda do mercado, hoje existe uma grande diversidade de sistemas operacionais disponíveis e compatíveis com dispositivos embarcados, várias opções para diversas plataformas e microprocessadores. A melhor escolha vai depender novamente da necessidade de sua aplicação e sua finalidade, fins comerciais ou não (licenças), sistema distribuído, aplicação robusta, etc.

Podemos também, através da pesquisa realizada pelo site Linux Devices [Hol06a], verificar os líderes de mercado em sistemas operacionais destinados a dispositivos embarcados, conforme mostra o gráfico 2.3.

Nas seções seguintes detalharemos alguns dos mais conhecidos e utilizados SOs, ressaltando seus pontos fortes e suas principais

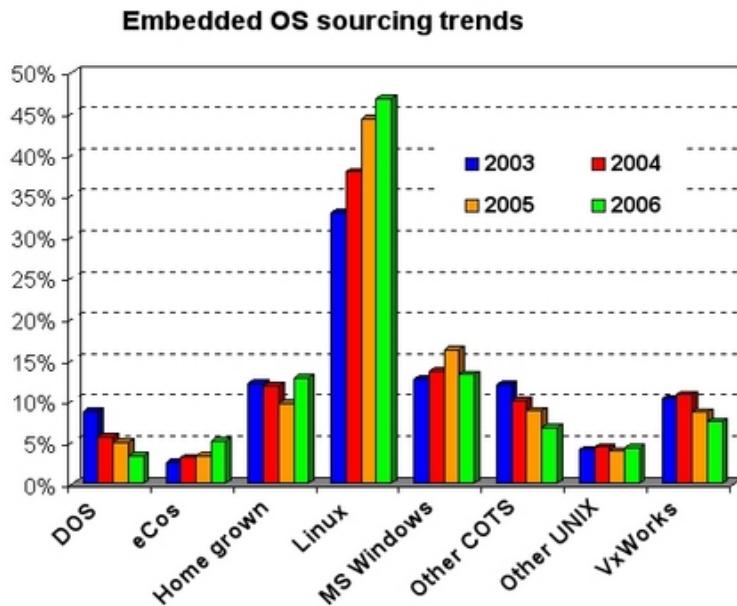


Figura 2.3: Sistemas Operacionais Mais Utilizados. (fonte: [Hol06a])

utilizações.

2.4.1 Microsoft Windows CE

A Microsoft desenvolveu seu primeiro sistema operacional para embarcados em 1996, o Windows CE 1.0 que visava inicialmente dispositivos de pequeno porte com poucos recursos, como era o caso dos primeiros gerenciadores pessoais de informação (PIM's, Palm's, etc).

Esse sistema evoluiu bastante em 1999, quando a Microsoft lançou seu segundo membro desta família, o Windows NT Embedded. Logo este sistema ganharia seu espaço no mercado devido principalmente a sua confiabilidade. Hoje esta família cresceu muito, já existem varias versões: Windows Mobile 2003, Windows Mobile 2005, Windows CE 5.0, Windows XP Embedded, Windows CE .NET.

Este sistema é muito procurado por empresas que buscam versatilidade, devido a existência de diversos módulos customizáveis já

inclusos no sistema padrão. Garante aos desenvolvedores uma grande compatibilidade de aplicações com as APIs Win32 e com a tecnologia .NET, minimizando-se a readaptação das aplicações. Provê suporte as plataformas x86, MIPS, ARM e Hitachi SuperH [Roc05].

Um ponto em que se difere dos concorrentes, é que a Microsoft disponibiliza ferramentas de avaliação dos sistemas Windows XP Embedded e Windows CE .NET o que permite que o projeto seja finalizado sem custo de aquisição. Após a ferramenta estar desenvolvida, testada e aprovada é adquirida a ferramenta de desenvolvimento final. Nenhum trabalho extra é necessário, pois a ferramenta é importada da ferramenta de avaliação para a de desenvolvimento final [Cor04]. Diferentemente dos concorrentes pagos que comercializam a ferramenta de desenvolvimento e se a tecnologia ou se a ferramenta forem ineficientes, todo capital investido é perdido. Porém, a sua licença é paga por unidades produzidas, denominada de *royalties*.

2.4.1.1 Windows XP Embedded

Na ferramenta de desenvolvimento do Windows XP Embedded, é possível selecionar dentre um conjunto de 10.000 componentes provenientes do Windows XP PRO, apenas escolhendo os necessários à sua aplicação. Provê também, uma alta compatibilidade com os serviços multimídia e web browsing, salientando que existe uma alta disponibilidade de drivers para os dispositivos do sistema, sem custo de desenvolvimento [Cor04].

Pontos Importantes:

- ❑ Seu suporte a tempo real, é feito através da aquisição de sistemas de terceiros, deixando assim de ser um recurso nativo.
- ❑ Média do tamanho da imagem do sistema: entre 100 e 150 MB, um tanto alto para sistemas embarcados.

- Muito utilizado em caixas automáticos, pontos de vendas, coletores de dados, etc.

2.4.1.2 Windows CE .NET

O Windows CE .NET disponibiliza cerca de 450 ferramentas com o propósito de gerar um sistema operacional bem customizado, para atender necessidades específicas. Dentre seus componentes está o .NET Compact Framework, este propicia que aplicações desenvolvidas com Visual Studio .NET sejam executadas sem alterações, diminuindo o treinamento específico para desenvolvimento em sistemas embarcados [Cor04].

Pontos Importantes:

- Disponibiliza recursos nativos de tempo real.
- Média do tamanho da imagem do sistema de 25MB, com Internet Explorer e Windows Media Player inclusos.

2.4.2 Linux

A utilização do Linux em um projeto embarcado pode ocorrer de duas maneiras, ou utilizar um sistema já específico para este tipo de aplicação como o uLinux, que será detalhado adiante, ou a partir de uma fonte do kernel, customizá-lo para sua aplicação.

Várias vantagens desta escolha foram abordadas em seções anteriores. Se bem utilizado, pode gerar sistemas finais bem compactos e ajustados, muita estabilidade e grande familiaridade com rede.

2.4.3 eCos

O eCos é um sistema operacional de código aberto, sem *royalties*, que suporta as ferramentas de código aberto GNU. Este sistema está sob uma licença específica denominada eCos License.

O eCos provê toda a funcionalidade necessária para aplicações embarcadas, dentre elas *drivers*, MMU, bibliotecas compatíveis, etc, além disso, disponibiliza também suporte a aplicações em tempo real e todos os mecanismos necessários para este tipo de aplicação. O desenvolvedor que utiliza este sistema tem nas mãos todas ferramentas necessárias para o desenvolvimento de aplicações embarcadas, incluindo software eCos de configuração e criação do sistema, compiladores GNU, debugadores e simuladores.

Funcionalidades que ele assegura ao sistema:

- ❑ kernel de tempo real.
- ❑ Rico em primitivas de sincronismo (sistemas distribuídos).
- ❑ Temporizadores, contadores e alarmes.
- ❑ TCP/IP, USB, Serial, Ethernet.
- ❑ Suporte para *debug*.
- ❑ ISO C e bibliotecas *math*.

Seu ponto forte é a capacidade de moldá-lo de acordo com a necessidade de sua aplicação. Isto lhe assegura uma boa customização do sistema, provendo alta performance de run-time e um tamanho otimizado devido a exclusão de funcionalidades desnecessárias. Hoje o eCos suporta várias plataformas e suas variantes, entre elas: ARM, Hitachi H8300, Intel x86, MIPS, Matsushita AM3x, Motorola 68k, PowerPC, SuperH, SPARC and NEC V8xx [eCo06].

2.4.4 Inferno

O Inferno é um sistema operacional bem diferente dos que vimos até agora, pois pode ser utilizado para criar e suportar aplicações distribuídas. Utiliza um protocolo chamado Styx para acessar e gerenciar os recursos locais e remotos.

Arquiteturas	Sistemas Operacionais
Intel x86	Windows NT/2000/XP
Intel XScale	Irix
IBM PowerPC	Linux
ARM	MacOS X
Sun SPARC	FreeBSD

Tabela 2.1: Arquiteturas e SOs Compatíveis com Inferno (fonte: [Nuo06])

Suas aplicações são programadas em uma linguagem específica denominada Limbo, onde seus binários são representados da mesma forma em todas plataformas e executados usando a tecnologia Just-in-time fazendo o uso de uma máquina virtual [Nuo06].

O Inferno pode rodar também sobre um outro sistema operacional, consequentemente sobre outras arquiteturas. Suporta os mais conhecidos SOs e arquiteturas como mostra a Tabela 2.1.

A segurança deste sistema se dá através do uso de um único protocolo para toda comunicação de rede, com isso podemos evidenciar a segurança e provê-la em um único nível do sistema. O inferno oferece suporte total a autenticação e conexões encriptadas utilizando uma variedade de algoritmos de encriptação, dentre eles:

- ❑ IDEA, 56 bit DES, 40, 128 e 256 bit RC4.
- ❑ MD4, MD5 e SHA

O inferno está em sua 4^a edição (2005), e está sob uma mistura de licenças GPL/LGPL/MIT, uma visão geral das licenças que regem software livre será vista em uma próxima seção. Seu código fonte é aberto [Fou06c].

2.4.5 uLinux

Este SO está muito ligado a um conceito de gerenciamento de memória (MMU), então primeiramente entenderemos o MMU, para assim

abordarmos realmente o uClinux. MMU é a abreviatura de Memory Management Unit, e é a capacidade que um processador/microcontrolador tem de gerenciar a memória do sistema, através informações incluídas em seu microcódigo, permitindo assim que as aplicações rodem em um espaço de memória virtual cabendo ao SO a tarefa de controle e mapeamento entre os endereços virtuais e o endereçamento real. Este tipo de funcionalidade garante ao sistema uma segurança considerável, impedindo que processos acessem áreas restritas de outros, evitando erros graves.

O uClinux é uma variação do kernel Linux que inicialmente foi desenvolvida visando os processadores/microcontroladores que não possuem gerenciador de memória, situação comum em sistemas embarcados bem simples, com recursos escassos e com custos reduzidos. A parte de gerenciamento de memória foi reescrita e hoje ele suporta sistemas com ou sem MMU, além de várias modificações no kernel. Uma nova biblioteca foi criada a uClibc. Esta está preparada para ser compilada em arquiteturas com ou sem MMU também. O sistema mantém a compatibilidade com outras bibliotecas [dA03].

Sua primeira versão foi desenvolvida em 1998, por Kenneth Albanowski e D. Jeff Dionne, baseada na versão 2.0 do kernel e voltado para o microcontrolador Motorola DragonBall MC68328, muito utilizado na época para controlar PDAs. Atualmente o uClinux está baseado no kernel 2.6, suportando diversas plataformas como mostram os itens abaixo:

- ❑ Motorola DragonBall
- ❑ Motorola ColdFire
- ❑ ADI Blackfin
- ❑ ETRAX
- ❑ Motorola QUICC

- ❑ ARM7TDMI and MC68EN302
- ❑ Intel i960
- ❑ PRISMA
- ❑ Atari 68k
- ❑ Microblaze
- ❑ NEC V850E
- ❑ H8

Os binários específicos para cada arquitetura suportada e uma lista com vários dispositivos que utilizam o uClinux como seu sistema operacional, podem ser encontrados no site [uCl06].

Sua interface do programa de aplicação(API) é praticamente igual a do Linux, permitindo assim que diversas aplicações Linux sejam suportadas, o que é uma vantagem em potencial. Outro ponto forte deste sistema é seu tamanho bem reduzido, cerca de 1-2 MB ou menor.

2.5 Questões Legais: Diretos e Deveres

Um assunto relevante quando se trata de sistemas embarcados é o do processo de licenciamento. A diversidade também é grande, indo desde o pagamento de *royalties* por unidade produzida, como acontece com sistemas baseados em tecnologia Windows ou WindRiver, até sistemas completamente gratuitos e sem grandes restrições, como os sistemas baseados em BSD. Entender os tipos de licenças envolvidas com o seu desenvolvimento é também um requisito fundamental e que deve ser levado a sério.

Nas próximas seções, serão abordadas os principais tipos de licenças relacionadas com sistemas operacionais livres, evidenciando

os direitos e deveres de cada uma. Vale ressaltar que esta é uma análise superficial. Uma verificação mais profunda deve ser feita por um advogado especialista.

2.5.1 GPL e LGPL

A GPL foi criada em 1989 por Richard Stallman e publicada em 1991 pela Free Software Foundation. Esta licença garante permissão para copiar, distribuir ou até mesmo vender cópias de um software que é regido sob este tipo de licença.

Além disso, você tem acesso ao código fonte do programa, com isso você tem direito de alterá-lo de acordo com suas necessidades. Entretanto é imprescindível que seja mantido sob a mesma licença. É de responsabilidade de todo desenvolvedor/usuário compartilhar todo material adquirido e/ou modificado e jamais se apropriar de resultados obtidos, ou seja, não pode ser encorporado a software proprietário. Hoje, a GPL se encontra na versão 2.

Uma vantagem desta licença é que os softwares a utilizam são muito compartilhados e discutidos via Internet através de fóruns e sites espalhados pelo mundo, se aperfeiçoando a cada dia. Destaca-se ainda que o usuário que manipula um código fonte aperfeiçoa sua programação e consegue uma melhor visualização da estrutura do programa e suas funcionalidades.

A grande maioria de softwares livres são regidos por esta licença, sendo o maior exemplo de software com este tipo de direitos o kernel Linux, que pode ser visto como uma grande propaganda para incentivar outros a utilizá-la também. Ressalto que o Free é relacionado a conhecimento, não a oportunidade.

Para maiores informações acesse o site do projeto GNU que regulamenta esta licença: <http://www.gnu.org/copyleft/gpl.html>, onde você encontra licença completa. Ou se necessário pode acessar o site <http://forum.ajudalinux.info/index.php?topic=8.0> onde se encontra uma tradução não oficial da mesma.

2.5.1.1 LGPL

Em 1999 foi criada a LGPL (Lesser General Public License) que é uma derivação da GPL. A diferença é que os materiais sob esta licença podem ser incorporados a materiais proprietários, e as alterações feitas nos materiais devem também manter a licença inicial, desde que não seja feita nenhuma modificação no material LGPL.

Esta licença é principalmente utilizada em bibliotecas e programas, para permitir sua linkagem (dinâmica) em programas que não são abertos. A idéia por trás disso é promover essas bibliotecas.

2.5.1.2 GPL Versão 3

A terceira versão da licença GPL ainda é um rascunho, visto que seu primeiro esboço foi publicado em Janeiro de 2006 e não está totalmente definida. Pelo que se vê isso deve demorar um pouco, pois ainda existem vários pontos muito discutidos em diversas conferências espalhadas pelo mundo. O principal ponto de divergência são os DRMs (Digital Rights Management) ou Gestão de Direitos Digitais, que são restrições tecnológicas aplicadas à distribuição de conteúdo digital [Ter06].

Provavelmente estas discordâncias deveram durar algum tempo, devido a importância de ser uma licença que pretende-se tornar global e substituir a GPL2. Algumas empresas já tem seu ponto de vista, como o projeto busybox, que rejeita a qualquer custo a GPL3 e pretende manter o seu licenciamento na versão 2 da GPL.

Maiores informações podem ser obtidas no site oficial do projeto: <http://gplv3.fsf.org/draft>.

2.5.2 Licença BSD

A licença BSD foi inicialmente criada para os sistemas BSD (sistema derivado do Unix), mas hoje vários outros admitem esta licença. Bem mais liberal que a GLP, permite que o software distribuído sob

esta licença tenha seu uso comercial com a simples condição, de que os créditos dos autores originais sejam mantidos. Partindo deste requisito você tem direito de fazer o que bem entender sem nem mesmo disponibilizar seu código fonte, por exemplo [dH05b].

Para dar uma idéia de como é aberta e livre, pode-se agregar a um software trechos de código de outro mantido sob esta licença, como fez a Microsoft com o Windows NT, onde incluiu a pilha TCP/IP e outros componentes provenientes do FreeBSD no Windows.

2.5.3 Licença X Embarcados

Se um projeto pode admitir um sistema operacional sob licença aberta, porque muitos projetos admitem sistemas operacionais sob licenças pagas? Novamente depende muito do projeto, normalmente um desenvolvedor admite uma licença paga porque os sistemas operacionais sob este tipo de licença normalmente são feitos por módulos de fácil aplicação ao dispositivo, ganhando um tempo valoroso no projeto, que pode ser transformado em dinheiro dependendo da aplicação. Pode acontecer também de o projeto já estar em andamento há um bom tempo, ou estar sendo reutilizado, e todo o material adquirido e desenvolvido está relacionado a este SO, tais como códigos, módulos, etc. Assim fica praticamente impossível reiniciar o projeto adaptando-o para um novo sistema operacional de código livre.

2.6 Exemplos

Os sistemas embarcados estão presentes em boa parte dos dispositivos que utilizamos no nosso cotidiano e evoluindo muitos outros. Serão apresentados alguns exemplos de sistemas embarcados para projetar um pouco da evolução contínua desta área, citando como funcionam, seus custos e disponibilidade no mercado:

O computador de bordo para automóveis Drew Tech DashDAQ,

está projetado para ser lançado no mercado americano em 2007 com um custo em torno de \$USS 600. Totalmente baseado em Linux, fará conexão com os veículos através de uma porta específica OBD2, que será padrão nos veículos mais modernos, onde conseguirá coletar informações de diversos sensores espalhados pelo veículo, apresentando uma rica variedade de diagnósticos ao usuário, como velocidade atual e seus picos, portas abertas, temperatura do óleo, combustível, rotação do motor, consumo, agendamento de revisões. Como seu código é aberto mesmo que uma determinada funcionalidade não esteja disponível inicialmente no sistema padrão, ela poderá ser adicionada, se tornando um sistema customizado [Hol06b].

Seu sistema será composto de um microprocessador ARM de 200 MHz, rodando um kernel 2.4 com 64MB de memória disponível, com slot SD/MMC para expansão podendo atingir até 8GB. Seu ponto forte é o visor de LCD colorido de alto brilho com 4 polegadas, sensível ao toque, que atinge uma resolução de até 480 x 272 pixels. Como mostram as Figuras 2.4, 2.5 e 2.6.



Figura 2.4: Tela de Funcionalidades 1. (fonte: [Hol06c])

Um produto que atrai muito pelo seu tamanho reduzido, é o Netstix 400XM-CF Computer, um computador produzido pela GumStix, que utiliza um processador Intel Xcale PXA255 de 400 MHz, com 64 MB de memória RAM e 16 MB de memória flash, porta Ethernet 10/100base T e um slot de expansão para cartões Compact Flash tipo II. Utiliza Linux com kernel versão 2.6 e acompanha uma série



Figura 2.5: Tela de Funcionalidades 2. (fonte: [Hol06c])

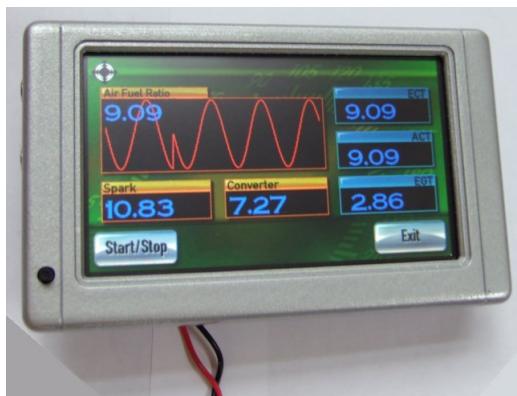


Figura 2.6: Foto do Drew Tech DashDAQ em Funcionamento. (fonte: [Hol06c])

de aplicativos e ferramentas junto com código fonte aberto, seu custo no mercado americano está em torno de \$USS 210 [Gum06a]. Conforme podemos analisar na Figura 2.7 seu tamanho é comparado a de um isqueiro:

Uma outra inovação foi lançada este ano (2006) pela empresa ChipPC Technologies. É uma tomada de parede com um computador embutido, com processador AMD AU1550 Alchemy, com 4 portas USB, saída de vídeo DVI (Digital Video Interface) que pode suportar resoluções de até 1600x1200 e saídas de áudio. Roda Windows CE nos 128MB de memória DDR e 64MB de memória flash, em forma de Disk On Chip (DoC). Uma alternativa ideal onde o espaço é escasso

netstix 400xm-cf computer

*cables not included

Figura 2.7: Netstix 400XM-CF. (fonte: [Gum06b])

e um PC comum não seria a melhor opção, visto que só é necessário um teclado, um mouse e um monitor para o sistema funcionar como um computador convencional [Eth06].

**Figura 2.8:** Jack-PC EFI-6900. (fonte: [Wik06])

Aplicações mais convencionais merecem ainda destaque:

- Firewalls.
- Celulares.

- ❑ Roteadores.
- ❑ Thin Clientes.
- ❑ Telefones IP (VoIP).
- ❑ Dispositivos Multimídia.

2.7 Conclusão

Este capítulo apresentou conceitos gerais em sistemas embarcados, discutindo as principais opções de plataformas e sistemas operacionais, além das licenças envolvidas neste processo.

O próximo capítulo terá seu foco no uso do Linux como sistema operacional para sistemas embarcados.

Capítulo 3

Sistemas Embarcados com

Linux

Como visto nos capítulos anteriores a área de sistemas embarcados é muito vasta, com várias plataformas, diversos sistemas operacionais, licenças, etc. Nesse capítulo será apresentado a utilização do sistema operacional Linux e as várias possibilidades que ele oferece para moldar um sistema para os mais diversos dispositivos embarcados.

Para podermos trabalhar com sistemas embarcados que utilizam-se do Linux como sistema operacional, é necessário que sejam introduzidos uma série de novos conceitos que são essenciais no manuseio e prática deste SO.

3.1 Vantagens e Desvantagens

Muito do que for tratado nesta seção poderá ser analisado como uma vantagem ou uma desvantagem de acordo com a finalidade e o propósito do seu uso. Apesar de ser um sistema de código aberto, o Linux tem uma boa qualidade do seu código que está em constante atualização através dos seus vários desenvolvedores espalhados pela Internet. A cada momento inúmeras aplicações novas, aprimoramen-

tos ou correções de erros aparecem. Se, de um lado, esta constante evolução parece interessante, por outro lado pode ser um transtorno para quem pretende manter um produto que necessita de atualizações freqüentes. Provavelmente você perderá um bom tempo para se localizar, saber o que aconteceu e como está funcionando.

Pelo grande potencial de moldá-lo de acordo com a sua aplicação consegue-se admitir um sistema com um tamanho relativamente pequeno(1-3MB). Isso se torna uma grande vantagem quando trabalhamos com sistemas restritos.

O Linux suporta uma série de sistemas de arquivos diferentes, indo desde os nativos da plataforma, como ext2, ext3 e raiser, até o suporte a sistemas fechados, como os arquivos do Windows, Apple e Novell. Fica a escolha do desenvolvedor decidir o que melhor satisfaz a aplicação ou o que lhe for mais confiável.

Um projeto baseado em Linux pode se tornar demorado se comparado a outras distribuições como o WinCE por exemplo. No CE não existe a idéia de suportar qualquer hardware. Geralmente existe um conjunto de placas de desenvolvimento certificadas para rodar com o Windows CE e o desenvolvedor irá escolher os módulos do CE necessários ao seu projeto. No caso do Linux, a diversidade de programas e plataformas pode se tornar um problema e envolve uma análise mais requintada, consequentemente com tempo maior de projeto. Isso pode se tornar inviável de acordo com o mercado. Por exemplo, pode ser que com tempo gasto no projeto, seu produto já tenha perdido o lugar no mercado ou parte dele.

Mais um ponto forte é o de disponibilizar a qualquer pessoa um amplo suporte através de diversos fóruns, listas, e-mails, FAQ's, exemplos, etc. Normalmente não temos custo por este suporte, mas como visto em pesquisa mostradas anteriormente, verificamos que o importante é ter esse suporte. Em alguns sistemas você não consegue suporte, nem mesmo pago. Por exemplo, versões antigas de um sistema operacional fechado podem simplesmente não serem mais

mantidas pela empresa.

Não se deve esquecer que o Linux é um sistema de propósito geral e pode não suprir todos os requisitos necessários em determinadas situações. Sistemas de missão crítica, isto é, que necessitam de requisitos de tempo da ordem de micro segundos e suporte a tempo real, podem não encontrar no Linux uma boa alternativa.

3.2 Linux Embarcado X Linux Comum

Embora o Linux que roda em um PC seja essencialmente o mesmo usado em um sistema embarcado, este último não poderá utilizar as mesmas funcionalidades de um sistema de propósito comum que esbanja recursos. Muitos serviços e aplicativos acabam sendo desnecessários para o propósito esperado e devem ser removidos.

Na interface com o usuário de um sistema comum, são disponibilizados diversos aplicativos de acesso a Internet, ferramentas de escritório, servidores web, manipulação de imagens, multimídia, etc. Já em sistemas embarcados o sistema provavelmente terá uma interface personalizada para atender as especificações do projeto.

Apesar de poderem compartilhar diversas bibliotecas, é comum que a biblioteca C da GNU seja substituída por outras mais compactas. A biblioteca GNU C, apesar de muito completa, acaba ultrapassando as necessidades de um sistema deste tipo. Existem várias opções de bibliotecas mais ajustadas para sistemas embarcados, abordadas com mais detalhes em uma seção seguinte.

Mesmo os utilitários de linha de comando, comuns em sistemas Linux, são disponibilizados através de versões mais leves sem as ferramentas de desenvolvimento, e apenas adicionados os que realmente forem necessários. Uma ferramenta chamada Busybox pode ser utilizada para facilitar esse processo de criação dos utilitários minimalistas de linha de comando [And06].

O kernel Linux é um ponto fundamental para diferenciarmos. Em

um sistema comum ele é muito completo, tem suporte a inúmeras funcionalidades, pode carregar drivers da maioria dos dispositivos de hardware existentes no mundo, características que são desnecessárias em um sistema embarcado. Para estes sistemas ele deve ser configurado para se tornar bem leve, mantendo apenas as características necessárias e os *drivers* dos dispositivos que estão presentes no equipamento. Assim podemos partir de um kernel comum e configurá-lo com as opções necessárias antes da compilação ou admitirmos um kernel específico para sistemas restritos como o uClinux, que já está bem próximo do que deseja o desenvolvedor [dA06].

3.2.1 Bibliotecas C

A utilização de bibliotecas específicas é comum, pois em um projeto embarcado fica praticamente impossível utilizar uma biblioteca que normalmente é utilizada em um ambiente Linux tradicional. Para isso existem bibliotecas bem mais enxutas, desenvolvidas especialmente com este objetivo, que terão a capacidade de gerar arquivos binários menores, o que é uma vantagem indiscutível. Quando se trabalha com dispositivos com recursos restritos.

Outro ponto fundamental está relacionado com as licenças empregadas. A biblioteca C da GNU (glibc) é baseada na GPL. Desta forma qualquer linkagem com ela irá gerar, necessariamente, um programa sob GPL. Isto pode não ser desejável, dependendo da natureza do projeto.

Seguem alguns exemplos de bibliotecas que são utilizadas em projetos deste tipo.

3.2.1.1 Diet libc

Projeto criado e mantido por Felix Von Leitner, com o intuito de minimizar o tamanho e aumentar a performance da biblioteca C. As principais vantagens em relação a glibc são o tamanho menor e a

maior velocidade na execução do código. Tem grandes semelhanças com a uClibc. O autor aconselha a linkagem estática da biblioteca [Yag03].

Um ponto importante é que esta biblioteca, diferentemente da maioria, não está sob licença LGPL e sim licenciada sob GLP. Desta forma se for utilizada para linkar um determinado código, o binário gerado será considerado um trabalho derivado e será regido sob o mesmo tipo de licença. Existe uma versão comercial desta biblioteca, caso haja a necessidade de um produto que não que não possa usar licença GPL.

A Diet libc suporta as seguintes plataformas: ARM, SPARC, MIPS, x86, PowerPC. E está disponível para download no site <http://www.fefe.de/dietlibc/>. O pacote acompanha um manual de instalação e algumas dúvidas esclarecidas através de um arquivo de FAQ.

3.2.1.2 uClibc

Esta biblioteca foi criada junto com o projeto uClinux, visto no Capítulo 2, para suportar processadores com ou sem MMU. Suporta as principais arquiteturas utilizadas em sistemas embarcados, pelo fato de incluir um carregador nativo específico para cada arquitetura. Caso contrário seria necessário utilizar um carregador proveniente da libc para ser utilizado em conjunto com a uClibc [Yag03].

Apesar de não ser igual a GNU C, a uClibc garante a maioria de suas funcionalidades, sendo bastante compatível com ela. Obviamente que não está totalmente completa, muitas funções que são usadas com menos freqüência foram descartadas. Em geral a grande maioria dos aplicativos que podem ser compiladores pela biblioteca GNU C, também poderão ser compilados e executados com a uClibc.

Pode ser baixada no site de seu projeto <http://uclibc.org>, onde pode-se encontrar também perguntas mais freqüentes, é mantida sob licença LGPL.

Arquiteturas Suportadas: ARM, cris, x86, i960, h8300, m68k, mips/mipsel, PowerPC, SH, SPARC e v850.

3.2.2 Ambiente Gráfico

Em sistemas tradicionais, com recursos abundantes, podem ser utilizados versões bem completas e conhecidas como KDE, Gnome, etc. Já em sistemas embarcados é comum não se ter interface gráfica ou, caso ela esteja presente, que pelo menos seja compacta e leve. Como, em geral, não se necessita de tantos recursos e nem mesmo existe processamento nem memória em abundância, são utilizadas alternativas enxutas criadas especialmente para dispositivos embarcados. Mesmo sendo para sistemas embarcados, seu uso causa um aumento considerável no tamanho da aplicação. Em geral, com 2 a 4 MB adicionais já se consegue uma funcionalidade gráfica razoável.

Conseqüentemente é apenas utilizado em projetos onde seu papel é imprescindível, em aplicações que realmente necessitam de uma alta interatividade com o usuário. Algumas seções à diante trataremos detalhadamente das principais versões de Toolkits gráficos, suas respectivas características e principais aplicações.

3.3 Processo de Inicialização

Muitas pessoas, ao ligarem um computador, nem imaginam que existem várias etapas a serem seguidas até o sistema operacional estar realmente carregado na memória e assim estar pronto para a utilização do usuário. Será dada uma pequena introdução de como isso se procede com os principais conceitos necessários para o entendimento. A abordagem a seguir terá sua base um PC comum, porém o processo é basicamente o mesmo em todas as plataformas.

O processo de boot envolverá uma série de passos e conceitos, listados a seguir, como ilustra a Figura 3.1.

- ❑ BIOS
- ❑ Bootloader
- ❑ kernel
- ❑ initrd
- ❑ rootfs (sistema raiz)

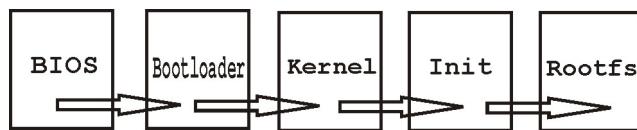


Figura 3.1: Processo de Boot.

Cada passo será descrito à seguir.

Quando o usuário liga o sistema, a primeira etapa a ser executada é um teste de hardware chamado "Power-On Self-Test", onde o sistema verifica a integridade de seus componentes e faz uma comparação com as configurações do último uso, armazenadas na BIOS. A BIOS é um programa que fica armazenado em memória ROM, ela contém rotinas básicas de entrada e saída e executa funções necessárias para inicialização de hardware [Goo06]. Se tudo ocorrer bem, o sistema prossegue com o processo verificando na BIOS qual a sua primeira fonte de inicialização (também conhecido como boot).

Caso o sistema admita, por exemplo, um Hard Disk como sua primeira fonte para o boot, ele irá acessar a primeira trilha do disco (cilindro 0, cabeça 0, setor1) chamada de MBR (Master Boot Record), onde encontrará todas as informações necessárias para dar continuidade ao processo através de um bootloader [Car05].

Bootloader ou gerenciador de boot, é o nome que se dá ao código localizado na MBR responsável em dar continuidade a inicialização do sistema, carregando o kernel. Neste caso, este código reside nos

primeiros 446 bytes da MBR (de um total de 512 bytes). Quando carregado, lista todas as partições existentes através de um arquivo de configuração [Car05].

Os bootloaders mais conhecidos que suportam a plataforma x86 são o LILO (LInux LOad) e o GRUB (GRand Unified Bootloader), cada um tem uma forma diferente de proceder com a inicialização e fazer o acesso ao kernel. Podem estarem configurados para inicializar vários sistemas operacionais, ou até um mesmo sistema operacional com versões de kernel diferentes.

No entanto, existem diversos outros bootloaders, capazes de iniciarizar plataformas diferentes. Como são voltados para sistemas embarcados, acabam agregando funcionalidades específicas como gravação/limpeza de flash, boot remoto via nfs ou tftp, consoles seriais para acesso, etc.

Alguns exemplos de bootloaders:

u-boot: Suporta as plataformas ARM, PowerPC e MIPS. O u-boot é utilizado em sistemas Linux.

MicroMonitor: Suporta as plataformas ARM, SH2, ColdFire e XScale. É utilizado para carregar vários sistemas operacionais, dentre eles: Linux, eCos, pSOS, etc.

Redboot: Suporta boot remoto (tftp), seu código fonte é aberto. Muito utilizado pela Intel.

LinuxBIOS: Suporta as arquiteturas x86 e PowerPC, é regido sob licença GPL. É inteiramente programado em C.

3.3.1 Kernel Linux

A palavra kernel foi muito difundida com o surgimento do sistema Linux, mas este está presente em todos os sistemas operacionais. É o núcleo do sistema operacional, representa a camada mais baixa de

interface com o hardware, sendo responsável por gerenciar os recursos do sistema computacional como um todo. É nele que estão definidas funções para operação com os periféricos do sistema, gerenciamento de memória (MMU), entre outros. Podemos definir que o kernel é um software que fornece aos aplicativos do usuário uma interface para que utilize de forma correta os recursos do sistema [Fou06b]. São geralmente desenvolvidos em linguagens de baixo nível como C ou Assembly.

O kernel Linux surgiu em 1991, foi criado por Linus Torvalds, na época um estudante de informática finlandês. O Linux foi criado inicialmente como um hobby para aumentar as funcionalidades de seu computador 80386 PC. Em 5 de outubro de 1991 circulava pela Internet a seguinte mensagem assinada por Linus: *“...Como eu mencionei há um mês, estou trabalhando em uma versão free de um sistema semelhante ao Minix para computadores AT-386. Ele já alcançou o estágio de ser usável (embora possa não ser, dependendo do que você quer fazer), e pretendo distribuir o código fonte. É apenas a versão 0.02..., mas já consegui rodar o bash, gcc, gnu-make, gnu-sed, compress, etc., nele”*.

A partir daí seu código se espalhou rapidamente pela Internet, caindo na mão de várias pessoas que se interessaram e deram incentivo ao projeto. Provavelmente Linus jamais imaginou que em menos de 10 anos depois atingiria 10 milhões de seguidores [Wat03].

O kernel do Linux é disponibilizado livremente sob licença GPL2, no seu site próprio (<http://kernel.org/>). Como está sempre em evolução e a busca por melhorias é constante, suas versões são fornecidas sempre com três algarismos de identificação. O primeiro e o segundo indicam a série do kernel, enquanto o terceiro indica a versão da série. O segundo algarismo é usado para diferenciar entre versões de desenvolvimento e de produção. Se for um número ímpar significa que é uma série ainda em desenvolvimento, ou seja, uma versão instável em fase de testes e aperfeiçoamento. Caso seja

Ano	kernel	Características
1991	0.02	Versão pública inicial
1994	1.0	Suporte à uma única arquitetura(i386)
1995	1.2	Outras arquiteturas: Alpha, Mips, Sparc
1996	2.0	Modularidade e Multiprocessamento(2)
1999	2.2	USB, ISDN, Masc.IP, Vídeo4Linux
2001	2.4	IPTables, Multiproc.(16), Sup.64MB RAM
2003	2.6	+Preempitível, Melhor Escalabil., uClinux

Tabela 3.1: Evolução do kernel (fonte: [dBA05])

par indica uma série estável e pronta para o uso. Estes números vão sendo incrementados conforme novos recursos são adicionados ou alterados, salientando que nem sempre é preciso rodar sua última versão, a não ser que nela tenha algum suporte específico necessário para sua aplicação ou *bugs* críticos solucionados [Ale04].

Alguns exemplos de versões do kernel:

- ❑ 2.6.2 - Versão estável, série par (6).
- ❑ 2.4.5 - Versão estável, série par (4).
- ❑ 2.5.19 - Versão instável, série ímpar (5).

A Tabela 3.1 mostra como o kernel Linux, ou o sistema operacional Linux, evoluiu desde a sua criação em 1991 até os dias de hoje, salientando algumas de suas versões que ficaram conhecidas por suas novas e marcantes inovações no sistema [dBA05].

Como o Linux é livre, seu kernel é disponibilizado junto de seus arquivos fontes e podemos configurá-lo antes de compila-lo, ou seja, é possível definir um kernel Linux específico para sua utilização, o que é uma grande vantagem especialmente quando se fala de sistemas embarcados.

3.3.2 initrd

O initrd é um pequeno sistema de arquivos montado na partida pelo kernel Linux. É usado para auxiliá-lo na montagem do sistema raiz (rootfs). Através do initrd é possível tratar a diversidade de plataformas existentes de maneira mais geral.

O initrd pode, por exemplo, ser capaz de fazer uma busca do sistema raiz nas diversas fontes de armazenamento do sistema, como discos, pendrives, CDs, etc. O initrd pode também, por exemplo, permitir que um rootfs encriptado ou compactado seja corretamente acessado. Em geral, este tipo de funcionalidade específica não é suportada pelo kernel Linux diretamente.

Uma outra utilidade é a de ter a capacidade de carregar um determinado módulo, dependendo da configuração de hardware no momento da inicialização, deixando o sistema mais genérico, mais simples e consequentemente mais rápido.

Pode-se figurar como uma etapa na inicialização do sistema onde podem ser executados vários scripts com funções diversas, como por exemplo, fazer uma atualização no sistema com o intuito de encontrar alguma alteração dos dispositivos de hardware para passar estas informações ao kernel.

No caso dos sistemas embarcados o initrd pode não ter muita utilidade, uma vez que o kernel destes dispositivos já é configurado especificadamente para prover o suporte ideal de acordo com o hardware do sistema, dificilmente alterado ao longo de sua vida útil.

No entanto, caso o sistema embarcado tenha o sistema raiz fornecido remotamente, pode ser interessante considerar uma abordagem de partida com o uso do initrd.

3.3.3 rootfs

O rootfs é o conjunto das aplicações e serviços de um sistema baseado em Unix, é carregado pelo kernel na inicialização do sistema para

assim poder ser executado e estar a disposição do usuário.

Cada sistema tem apenas um rootfs, ele pode estar comprimido em apenas um arquivo (imagem), comum em sistemas embarcados, pode ser uma partição (sistemas comuns), pode estar em uma rede ou mesmo em um disco SCSI, SATA ou IDE.

Após subir o rootfs para à memória do sistema, o kernel executa o script /sbin/init que contém instruções de como proceder com a inicialização do sistema. O init é responsável por finalizar o processo de inicialização do sistema. Ele requisita o arquivo /etc/inittab, que é constituído de uma tabela que mantém passo a passo de como isso dever ser feito [Yag03].

O init é o serviço pai que o sistema executa na inicialização (bootup), sendo responsável pela inicialização de outros serviços requeridos como por exemplo sh, telnet, ftp, web server e etc.

Na Tabela 3.2 é apresentada uma estrutura padrão de um rootfs. Cada um de seus diretórios tem uma função específica e alguns apenas são necessários em sistemas de multiusuários e/ou servidores. Como a maioria dos sistemas embarcados com Linux não são multiusuários, esta árvore pode ser bem mais simples do que um rootfs de um sistema comum.

3.4 Personalizando o kernel Linux

Como veremos na prática no capítulo seguinte, de acordo com as opções selecionadas no menu antes da compilação, podemos personalizar o kernel. Essa configuração dependerá diretamente de cada projeto, de acordo com suas características e necessidades. Obviamente que o seu tamanho é proporcional a quantidade de funcionalidades que serão suportadas, com implicações no tempo que o sistema demora para iniciar e na segurança. Quanto mais recursos, mais código para ser carregado e maior o tempo para que isso seja feito. Este volume implica também diretamente na segurança, quanto mais

Diretório	Conteúdo
bin	Binários essenciais
boot	Arquivos estáticos utilizados pelo bootloader
dev	Arquivos especiais e dos dispositivos
etc	Arquivos de configuração do sistema
home	Diretórios do usuário
lib	Bibliotecas essenciais, incluindo C e módulos do kernel
mnt	Ponto de montagem dos sists. de arquivos temporários
opt	Pacotes de softwares extras
proc	Sist. de arquivos virtual, para o kernel e outros procs.
root	Diretório do administrador
sbin	Binários de administração do sistema
tmp	Arquivos temporários
usr	Contém a maioria das aplicações do usuário
var	Armazenamento de valores de variáveis

Tabela 3.2: Estrutura Básica de Diretórios (fonte: [Yag03])

Serviço	Nome Programas.
Arquivo	cp, ls, mv, rm, rmdir, chmod, sync
Editor	cat, more, vi, find, grep
Rede	ping, ifconfig, netstat, route
Processo	kill, ps, insmod, lmmode, rmmod

Tabela 3.3: Exemplos dos Principais Programas (fonte: [Yag03])

código se tem, maior a probabilidade de ocorrerem erros.

As opções do kernel vão desde selecionar que tipo de arquitetura o seu sistema utiliza até drivers para os dispositivos de hardware específicos, passando por protocolos, barramentos, freqüências de processadores, etc. São inúmeras as opções que podem estar ativadas ou não. Após selecionar o que o sistema necessita, o kernel é compilado, não havendo a possibilidade de retirar uma determinada funcionalidade depois do sistema gerado. Algumas características podem ser compiladas como módulos, permitindo que sejam ou não carregadas, guardando uma certa flexibilidade.

3.4.1 Compilação

Como veremos mais detalhadamente no capítulo a seguir, após serem selecionadas as opções ideais do kernel, através de um menu, elas ficam salvas em um arquivo chamado `.config`, que será consultado no momento da compilação.

Quando realizamos a compilação do kernel são gerados os arquivos executáveis em um diretório:`/dirkernel/arch/i386/boot/bzImage`. O `dirkernel` representa o diretório onde as fontes descompactadas do kernel estão localizadas. O próximo passo é juntarmos o `rootfs` e os arquivos gerados do kernel, para assim darmos partida no sistema.

3.4.2 Descompactação/Execução em RAM

Quanto o bootloader inicia, ele tem os dados de onde o kernel se encontra. Carrega o kernel na memória e fornece informação para ele onde se encontra o `initrd` e/ou o `rootfs`. O `initrd` é geralmente carregado em memória. Apenas o `rootfs` é normalmente encontrado em um dispositivo não volátil, como um disco, por exemplo.

3.4.3 Tempo de Partida

Um aspecto muito importante é o tempo que o sistema demora para carregar e estar pronto para receber requisições do usuário/aplicação (tempo de inicialização). Obviamente que o usuário não quer ficar esperando seu aparelho de DVD ou um celular demorar um minuto para iniciar.

O tempo de inicialização está relacionado com o tamanho do kernel, a quantidade de funcionalidades que ele assegura, com a forma como é carregado o rootfs na memória, como é feito o acesso ao rootfs, que tipo de armazenamento o sistema dispõe, etc. Esses aspectos estão detalhados nas seções seguintes.

3.4.3.1 Redução de Tamanho

Podemos reduzir bem o tamanho simplesmente configurando-o de acordo com o mínimo necessário para ele executar, mas mesmo assim ficam pequenas funções ativadas sem necessidade. Uma boa alternativa para reduzir ainda mais, é uma ferramenta que auxilia nesse processo, denominada Linux-Tiny. O Linux-Tiny é composto de um conjunto de patches para serem aplicadas ao kernel que removem alguns aspectos desnecessários do mesmo. Por exemplo, remover mensagens de erros geradas pelo kernel, inútil muitas vezes em um sistema embarcado. A redução pode chegar a 300Kb [Ben05].

Um teste utilizando essa ferramenta foi feito, com o kernel 2.6.5, com suporte a IDE, TCP, ext2, NIC e seu tamanho chegou a 363K. Uma característica importante é que a configuração do Linux-Tiny é feita independente da do kernel, o que pode facilitar o trabalho.

3.4.3.2 Armazenamento e Execução no Local

O armazenamento em sistemas embarcados normalmente não é feito através de um Hard Disk (HD) convencional, como nos sistemas de propósito geral. É preceito comum que o sistema embarcado não

tenha partes móveis, eliminando discos rígidos ou dispositivos semelhantes. Além disso o alto consumo de energia, tamanho e faixa de temperatura de operação podem inviabilizar alguns projetos.

Uma boa opção para implementação em sistemas deste tipo é a memória flash, com consumo e tamanho reduzidos. A única desvantagem deste tipo de memória é o seu custo, que é elevado se comparado a outros dispositivos, levando em conta o custo por megabyte.

Com a utilização de uma memória flash, podemos ativar o suporte a funcionalidade XIP no kernel, para que os dados sejam executados diretamente na flash, consequentemente o tamanho da memória RAM necessária em nosso dispositivo embarcado diminui, baixando o custo e claro caindo no nosso objetivo central que é atingirmos um alto desempenho, diminuindo o tempo de partida do sistema.

Existem no mercado alguns chips de memória flash projetados especialmente para prover a função de disco IDE, como o DiskOnChip (DoC). O DoC é um dispositivo que pode ser acoplado diretamente a placa do sistema reduzindo o tamanho geral. Porém eles também tem algumas desvantagens. Por exemplo, seus drivers podem não ser de código aberto, acarretando custo para o projeto [Hol00a]. Além disso, dispositivos baseados em flash tem sempre um número de escritas máximas, em geral com vida útil menor que um HD.

Mais um tipo de memória vem entrando no mercado, a DoM (DiskOnModule). O DoM têm basicamente as funções da DoC sem suas desvantagens, ou seja, vida útil comparada a do HD e não tem problemas de drivers como a DoC.

Mais detalhes podem ser obtidos no site: <http://www.linuxdevices.com/products/PD7536396952.html>.

3.4.4 Unindo Tudo Com Buildroot

O Buildroot é um projeto mantido por Erik Andersen, que é composto de uma combinação de makefiles e patchs que possibilitam a

composição de um sistema mínimo constituído de um compilador cruzado (toolchain) e um rootfs, que por sua vez é composto de aplicações, mais bibliotecas e kernel destinado a uma plataforma destino. E para garantir a segurança não há a necessidade de permissões de root para criar ou executar o buildroot.

Este projeto cresce a cada dia, se tornando conhecido como projetos já estabelecidos como o uClibc e Busybox. O Buildroot emprega o Busybox para gerar parte de seus executáveis [And05]. Ele é mantido sob licença LGPL.

3.4.5 Sistema Raiz com Busybox

O Busybox é um projeto que tem o mesmo mantenedor do Buildroot, Erik Andersen. Em seu projeto ele combina versões mais simples dos mais comuns utilitários UNIX. Os utilitários que ele gera tem menos opções do que suas versões completas GNU, mesmo assim as funcionalidades incluídas atendem os requisitos deste tipo de sistema. Esses utilitários são gerados em um pequeno arquivo executável [And06].

Esta é uma ferramenta ideal para construção de um rootfs, para um sistema simples ou embarcado. Através de um menu de configuração, o desenvolvedor pode customizar o rootfs que será gerado. Diminuindo consideravelmente o tempo de criação do mesmo, caso fosse criado manualmente. O Busybox é modular, os comandos ou funções podem facilmente incluídos ou excluídos. Admite licença GPL2.

3.5 Suporte à Tempo Real

Mais uma decisão que o desenvolvedor terá que tomar ao projetar/desenvolver um sistema embarcado, é se o sistema irá necessitar de suporte à tempo real ou não. Dependendo da aplicação aplicação o sistema operacional deverá estar preparado para garantir que a

requisição de um determinado processo será aceita e respondida em um determinado espaço de tempo. Ou então que o atendimento de uma interrupção tenha uma latência máxima. Em sistemas de missão crítica o não cumprimento das restrições temporais de execução pode levar a falhas.

Em um PC rodando Linux, suponha que exista um processo requisitando processamento. Suponha também que existam inúmeras aplicações requisitando tempo de processador. O tempo de resposta do processo irá variar de acordo com a concorrência, e é exatamente essa variação que dependendo da aplicação nunca poderá ocorrer. Para garantir estas restrições, os Sistemas Operacionais de Tempo Real (Real Time Operational Systems's, ou RTOS) utilizam estratégias especiais de escalonamento de processos.

De maneira geral, os sistemas de tempo real podem ser divididos em dois grandes grupos:

Hard Real Time: Usado em sistemas críticos. Faz a implementação do sistema para trabalhar com tempos de resposta máximos e exatos, ou seja, caso ocorra uma falha no cumprimento dos limites temporais, isso poderá acarretar em prejuízos incalculáveis ou catástrofes. Por exemplo, utilizado na aviação, sistemas industriais, equipamentos militares, etc. “*Se a ação deve acontecer absolutamente em um certo momento (ou em certo intervalo), tem se um sistema em tempo-real hard*” [Tan01].

Soft Real Time: Sistemas que necessitam de um suporte à tempo real, mais sem a necessidade de respostas tão exatas quanto a implementação Hard. A falha no cumprimento do tempo não acarreta em danos ou prejuízos significativos. Por exemplo, considere a reprodução de um aparelho de DVD. “*Esses sistemas são chamados de soft real-time, em que a perda ocasional de um deadline é aceitável*” [Tan01].

O Linux por si só não tem suporte à tempo real, porém existem

patches ou micro-kernels que podem ser aplicados a um sistema linux convencional que alteram o escalonamento do sistema, tornando-o um sistema com suporte à tempo real. Um dado muito importante é que existem cálculos específicos que ajudam e garantem a viabilidade do projeto. Podem existir casos onde não é possível atender os requisitos do projeto, ou seja, a necessidade das aplicações é maior do que o sistema pode atender.

Algumas das principais características deste tipo de sistema:

- ❑ Proporcionam praticamente os mesmos serviços que os SO's comuns, mas adicionam o comportamento previsível perante ao tempo.
- ❑ São sistemas que satisfazem à fortes condições de tempo de resposta à correção lógica e/ou temporal.
- ❑ Precisão e desempenho estão fortemente ligados.
- ❑ Estão preparados para o tratamento de entradas assíncronas.
- ❑ Previsibilidade no pior caso de carga e falhas.

A seguir serão apresentados alguns exemplos de sistemas Linux que dão suporte a tempo real, ressaltando que os exemplos foram escolhidos com a intenção de mostrar as diferentes formas que estes sistemas admitem para obter um resultado satisfatório.

3.5.1 Real Time Linux (RT-Linux)

O desenvolvimento do RT-Linux foi iniciado no Department of Computer Science no New México Institute of Tecnology, porém hoje é mantido principalmente por uma empresa chamada FSMLabs.

É um descendente do linux que se propõe a suportar aplicações com restrições temporais críticas utilizando o escalonamento Hard. Para isso implementa um microkernel com suporte à tempo real, para

trabalhar em conjunto com o kernel Linux. Esse microkernel classifica o kernel do Linux como uma tarefa de mais baixa prioridade, usa o conceito de máquina virtual para tornar o kernel convencional e as demais aplicações totalmente interrompíveis [dO04].

Quando chega uma requisição de interrupção esta é inicialmente tratada pelo microkernel de tempo real, que é repassada para sua tarefa Linux (kernel) no momento em que o sistema não prevê nenhuma tarefa de tempo real para executar.

Uma outra opção semelhante é o RTAI Linux, utiliza basicamente a mesma forma de prover funcionalidades e tratamentos para sistema em tempo real. Uma distribuição preferida pela comunidade free, que rejeita as licenças e patentes envolvidas no RT-Linux.

3.5.2 Linux/RK

Este projeto foi desenvolvido no Real-Time and Multimedia Systems Laboratory na Carnegie Mellon University em Pittsburgh (PA/USA). O projeto implementa o conceito de *resource kernel*, inserindo no kernel convencional do Linux um módulo via patch, denominado *Portable Resource kernel*, que é responsável por um controle ao acesso físico do sistema, assegurando garantias e impondo limites necessários neste tipo de sistemas [dO04].

Todas as aplicações devem reservar suas fatias de cada recurso préviamente, para proporcionar ao kernel a disponibilidade no momento em que a aplicação necessitará do mesmo. Assim, muni o kernel de informações para que consiga o controle total do sistema e a capacidade de prever eventuais complicações.

O módulo que fica associado ao kernel padrão é chamado de RK, de onde deriva o seu nome. O Linux/RK agrega ao sistema vários outros serviços como controle de admissão (aceita ou não uma reserva), escalonamento de recursos, monitoração de usos e imposição de reservas.

3.6 Toolchains

É comum, principalmente em sistemas embarcados, que a plataforma adotada para o desenvolvimento de uma determinada aplicação seja diferente da plataforma que a mesma será executada. Por exemplo, você pretende desenvolver uma aplicação para rodar em uma plataforma ARM à partir de um notebook x86/Linux. Usando algumas ferramentas do GNU toolkits como gcc, gas e ld você pode especificar e criar um compilador cruzado (cross-compiler) capaz de compilar aplicações para outras plataformas [Hol04].

Os compiladores cruzados podem ter outros usos também, por exemplo, evitar o uso de uma máquina com baixo poder de processamento para outra, que disponibiliza alto desempenho, com a intenção de reduzir o tempo de compilação de dias para minutos, dependendo do caso.

Como normalmente é utilizado para gerar os executáveis nativos da arquitetura do sistema destino, consequentemente existem toolchains específicos para cada arquitetura. Quando o sistema destino utiliza a mesma arquitetura que a plataforma de desenvolvimento seu uso é desnecessário.

No mercado existem empresas especializadas neste tipo de ferramenta, desde opções grátis até outras de custos elevados. Podemos encontrar no site: <http://www.microcross.com/html/press-15.html> um exemplo de uma empresa que comercializa um compilador cruzado para ambiente Linux (GNU X-Tools 3.4) à 1000 dólares por plataforma suportada. Este valor inclui também suporte. E uma outra que disponibiliza um toolchain grátis com suporte às seguintes arquiteturas: ARM (XScale), PPC4xx, PPC-e500, PPC60x, PPC8xx and x86. Maiores informações e os arquivos de instalação podem ser encontrados no site do fabricante <http://www.sysgo.com/en/downloads/freetoolbox/>.

Existem pacotes binários específicos para cada arquitetura que

facilitam o trabalho do desenvolvedor, porém pode existir a necessidade de que isso seja feito manualmente. Esse processo passo-a-passo é bem trabalhoso e podem haver algumas incompatibilidades de bibliotecas com determinadas arquiteturas. Isso deve ser verificado antes do inicio do trabalho para evitar perca de tempo. Ferramentas como o Buildroot e Crosstool ajudam a diminuir o tempo despendido nestas tarefas.

3.7 Toolkits Gráficos

Algumas aplicações embarcadas apresentam o requisito de interface com o usuário, como celulares, PDAs, kiosques de informação, caixas eletrônicos, etc. Nestes casos, é necessário que o desenvolvedor use algum tipo de biblioteca gráfica, freqüentemente conhecido como Toolkit Gráfico.

Estes Toolkits são geralmente menores do que um servidor X tradicional, que consumiria muitos recursos, nem sempre disponíveis em um sistema com restrições.

Os Toolkits não são apenas utilizados em sistemas embarcados e podem ser aplicados em computadores simples que jamais suportariam um ambiente X completo. São utilizados também para fazer a parte gráfica de CDs botáveis.

Vários Toolkits realizam o acesso à placa gráfica através de um recurso provido pelo kernel chamado de framebuffer. O framebuffer é um recurso desenvolvido originalmente para permitir a visualização de imagens em modo texto, acessando diretamente a memória da placa de vídeo. Por exemplo, o pinguim apresentado no topo da tela ou algum tipo de menu gráfico durante o boot do sistema operacional indica que o framebuffer está ativado e funcionando.

A seguir serão apresentados alguns Toolkits Gráficos em uso atualmente.

3.7.1 DirectFB

É uma biblioteca para o GNU/Linux que, através de uma API, simplifica as operações gráficas para que os aplicativos se comuniquem diretamente ao hardware de vídeo através do framebuffer. O DirectFB provê potência gráfica à sistemas embarcados mostrando um novo padrão para a utilização em sistemas Linux [Hol01].

Foi projetada para trabalhar com sistemas embarcados, oferece a máxima performance de aceleração do hardware, utilizando o mínimos de recursos possíveis [Hun04].

Algumas de suas características:

- ❑ pequeno tamanho
- ❑ possibilidade máxima de aceleração de hardware
- ❑ suporta operações gráficas avançadas
- ❑ não tem dependências de bibliotecas, exceto da libc

Também pode ser utilizada em aplicações de PC, existe até um projeto de uma mini-distribuição Linux que utiliza DirectFB como ambiente gráfico que utiliza framebuffer [dH05a].

3.7.2 Qtopia

Qtopia é um toolkit gráfico muito completo e bem documentado o que pode significar redução no tempo do projeto. Ele é baseado em uma biblioteca gráfica chamada QT. A união do Linux com o Qtopia gera suporte para aplicações diversas, nos mais variados sistemas embarcados. Seu código fonte é aberto, podendo ser modificado e facilmente aplicá-lo em um outro dispositivo ou projeto. Com cerca de 2MB já é possível ter uma interface bem funcional.

Existe até uma plataforma própria, denominada Qtopia Platform, que utiliza um Core de processador próprio (Qtopia Core). A Trolltech, empresa por trás do Qtopia, afirma que está preparada para

criação de dispositivos Linux com uma rica gama de aplicações para o usuário. A plataforma foi projetada de forma bem otimizada e com alta eficiência.

QT embedded vem sendo usada em PDAs e celulares, por exemplo. A Motorola já desenvolve celulares baseados em Linux, que utilizam o Qtopia como toolkit gráfico. Alguns modelos podem ser encontrados no mercado E680, A780, A760, A768 e A1000.

Para maiores informações sobre Qtopia, acesse o site <http://www.trolltech.com/products/qtopia/devices> e veja uma lista de vários dispositivos de alta tecnologia que optam por este toolkit gráfico.



Figura 3.2: Visual Gráfico com Qtopia. (fonte: [Wik06])

3.7.3 TinyX

O TinyX foi desenvolvido por Keith Packard, é baseado no XFree86 [Pac05]. Este é mais um ambiente gráfico moldado para sistemas pequenos ou embarcados. É o mais leve dos apresentados sendo que a

instalação não necessita mais que um 1MB de memória para que ele seja ativado [Hol00b]. Acesse o site do desenvolvedor para informações mais detalhadas <http://www.xfree86.org/current/TinyX.1.html>. No site existe também um lista com vários comandos específicos, e suas respectivas funções.

3.7.4 Conclusão

Os principais conceitos relacionados à construção de sistemas embarcados com Linux foram apresentados e discutidos neste capítulo. Estes conceitos são de essenciais para que o exemplo construído no capítulo seguinte seja compreendido adequadamente.

Capítulo 4

Linux Embarcado na Prática

Neste capítulo será aplicada toda teoria vista nos capítulos anteriores em um exemplo prático que visa ilustrar o trabalho. Para permitir a reprodução dos resultados e evitar a dependência de uma placa embarcada específica, optamos por utilizar um emulador chamado Qemu, que fará a emulação de uma plataforma x86.

Quando iniciamos a criação de um sistema embarcado podemos admitir duas abordagens distintas:

- ❑ Top-down: parte do sistema completo e retira o que for desnecessário.
- ❑ Bottom-up: cria inicialmente o sistema mínimo e posteriormente adiciona o necessário.

No nosso exemplo iremos partir do mínimo e adicionar o que for necessário, seguindo a abordagem Bottom-up, mais didática e adequada para este exemplo.

4.1 Qemu

O Qemu é um emulador genérico de processadores com código fonte é aberto. Trabalha basicamente em dois modos de operação:

Emulação completa do sistema: neste modo o QEMU emula todo o sistema desde o processador, memória e seus periféricos. Pode suportar vários sistemas operacionais, cada qual para a plataforma suportada.

Emulação em modo de usuário: suportado apenas em estações Linux, pode carregar os processos compilados de um sistema em outro [Bel06].

No site do desenvolvedor (<http://www.qemu.org>) é disponibilizado um módulo de aceleração, para ser usado no caso de emulação de PC em um PC. Ele habilita a plataforma alvo para rodar boa parte do código da aplicação diretamente no processador do host, aumentando consideravelmente a velocidade de execução.

Na Figura 4.1 podemos verificar as plataformas que suportam a execução do QEMU. Já a Figura 4.2 apresenta os sistemas emulados.

No site http://www.oszoo.org/wiki/index.php/Category:Operating_System_Images estão disponíveis inúmeras imagens de diversos sistemas operacionais para download, prontas para serem carregadas/emuladas no QEMU.

4.1.1 Instalação do Qemu

A Instalação do QEMU no Linux é bem simples, é feita a partir de um pacote padrão, normalmente é encontrado no formato `qemu-x.y.z.tar.gz`. Basta seguir os comandos para descompactar, entrar no diretório, configurar e compilar, como ilustrado a seguir:

```
tar -zxvf qemu-x.y.z.tar.gz
```

Host CPU	Status
x86	OK
x86_64	OK
PowerPC	OK
Alpha	Testing
Sparc32	Testing
ARM	Testing
S390	Testing
Sparc64	Dev only
ia64	Dev only
m68k	Dev only

Figura 4.1: Plataformas Emuladas (fonte: <http://www.qemu.org>)

```
cd qemu-x.y.z.tar.gz
./configure
make
make install (com direitos de administrador do sistema)
```

Existem também pacotes pré-compilados para vários SOs.

4.2 Configurando e Compilando o Kernel Linux

Vamos iniciar o exemplo com a configuração do kernel, primeiramente escolhendo as opções que melhor atendam nossa aplicação para depois compilarmos. Assim obteremos seus executáveis para agregar aos outros módulos do sistema e fazermos a simulação através do QEMU.

Para partirmos é necessário que se tenha uma versão do kernel, que pode ser facilmente obtida através do site oficial (<http://kernel.org>). Com ele disponível, deve-se descompactá-lo, e assim estar pronto para seguir os passos seguintes que serão descritos

Target CPU	User emulation	System emulation
x86	OK	OK
x86_64	Not supported	OK
ARM	OK	OK
SPARC	OK	OK
SPARC64	Dev only	Dev only
PowerPC	OK	OK
PowerPC64	Not supported	Dev only
MIPS	OK	OK
m68k	Dev only	Dev only
SH-4	Dev only	Dev only

Figura 4.2: Plataformas Suportadas (fonte:<http://www.qemu.org>)

nas próximas subseções. No exemplo será usada a versão 2.6.17.1.

4.2.1 Personalizando o Kernel

Como estamos interessados em um kernel mínimo, vamos inicialmente remover todas as configurações *default* do kernel com o comando a seguir: `make allnoconfig`.

Com as opções limpas, podemos abrir o menu de configuração do kernel. Esse menu de configuração do kernel é disponibilizado de várias formas diferentes, desde um menu texto que vai perguntando ao usuário sim ou não para cada funcionalidade (opção bem exaustiva) até outras opções gráficas de configuração mais rápida.

Utilizando o comando `make config`, temos uma opção em modo texto conforme a Figura 4.3. Com o comando `make menuconfig` é gerado um menu em modo texto apresentado na Figura 4.4. Já com o comando `make xconfig`, é aberto um menu em modo gráfico que necessita de um servidor X instalado na máquina, como na Figura 4.5.

Todas as opções que a nossa aplicação necessitará serão escolhidas

e apresentadas nas Figuras de 4.6 à 4.30. Resumidamente, iremos habilitar os seguintes recursos:

- ❑ Processador x86 genérico.
- ❑ Suporte PCI para a placa de rede.
- ❑ Formato de executável *elf*.
- ❑ Serviços básicos de rede (TCP/IP).
- ❑ Drivers: RAM, IDE, placa de rede.
- ❑ Suporte ao sistema de arquivos Ext2 e /proc.
- ❑ Linguagem nativa.

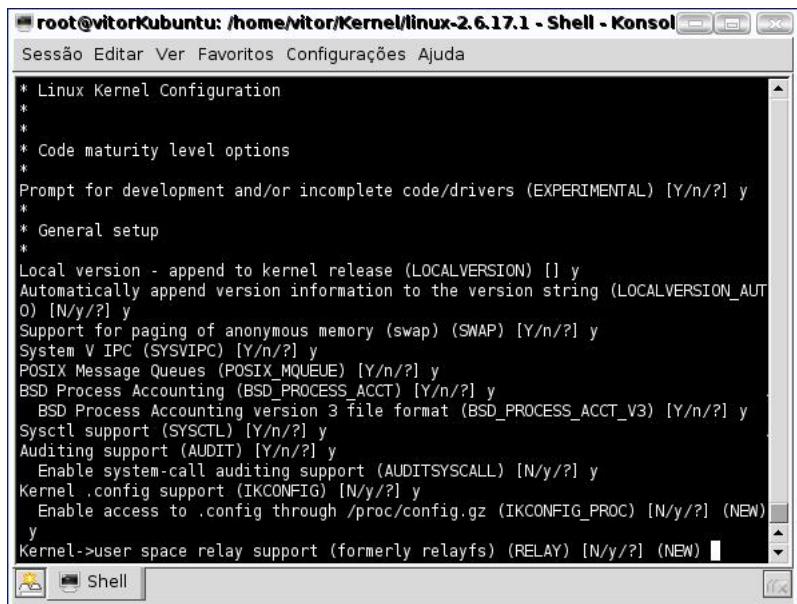


Figura 4.3: Menu Kernel - Utilizando o comando make config

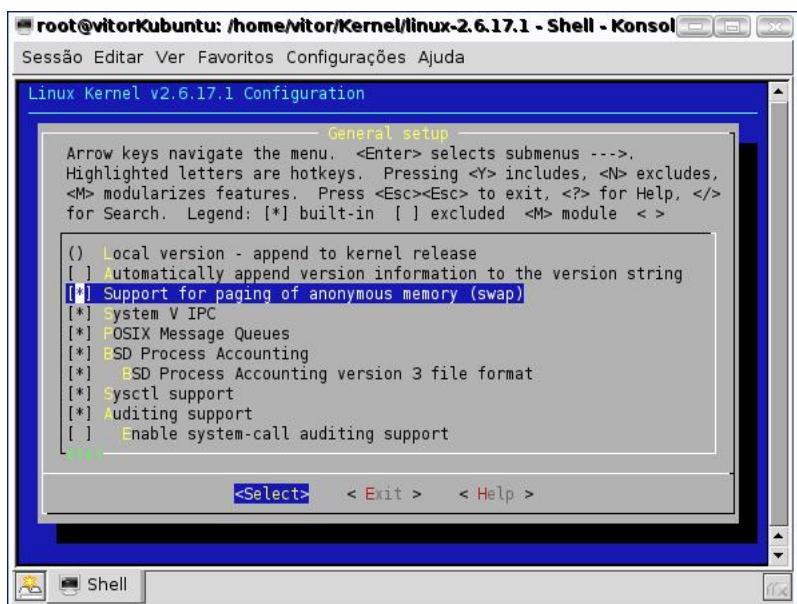


Figura 4.4: Menu Kernel - Utilizando o comando make menuconfig



Figura 4.5: Menu Kernel - Utilizando o comando make xconfig

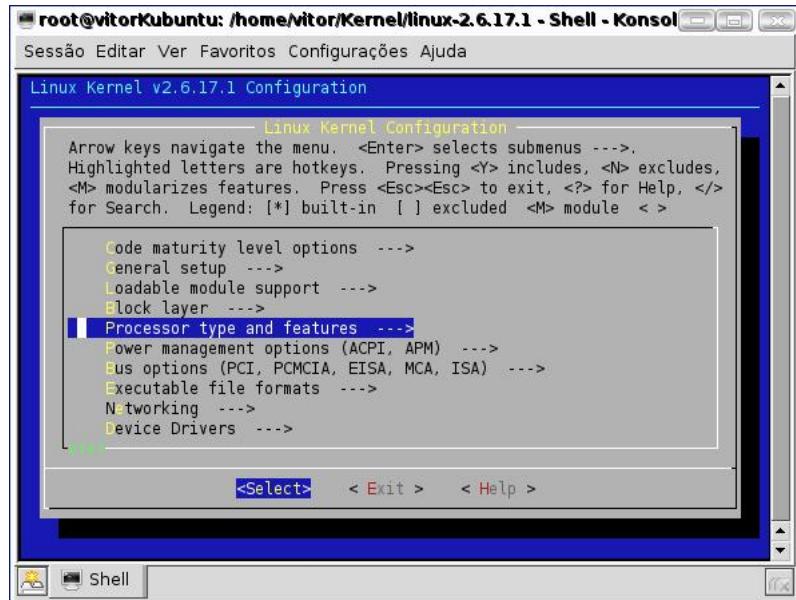


Figura 4.6: Configurando o Kernel - x86 1/2

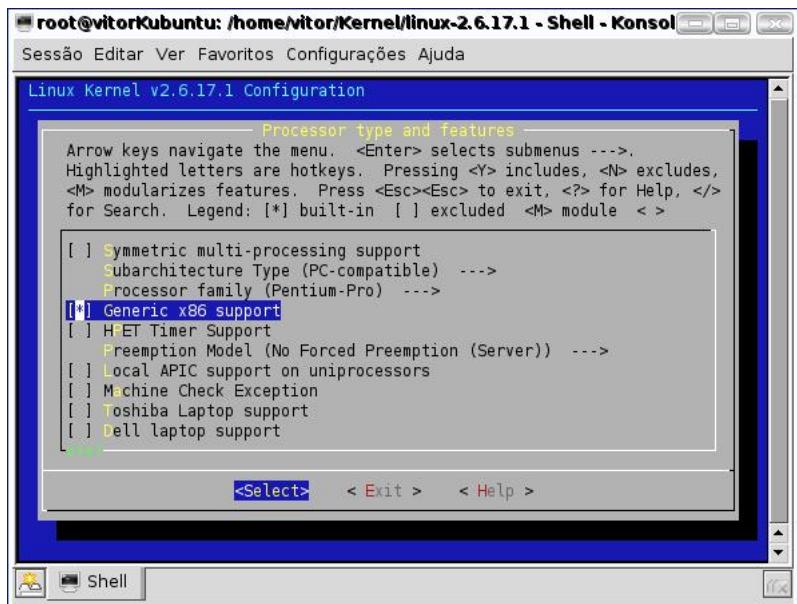


Figura 4.7: Configurando o Kernel - x86 2/2

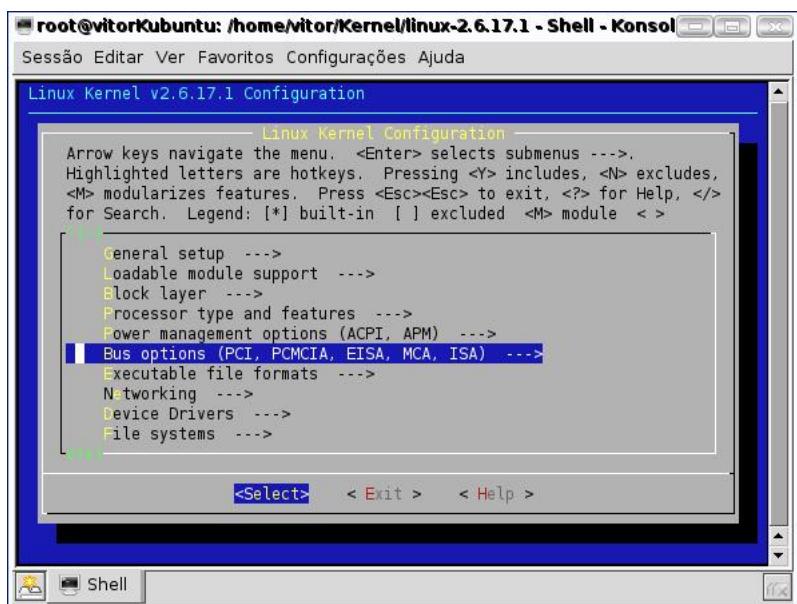


Figura 4.8: Configurando o Kernel - PCI 1/2

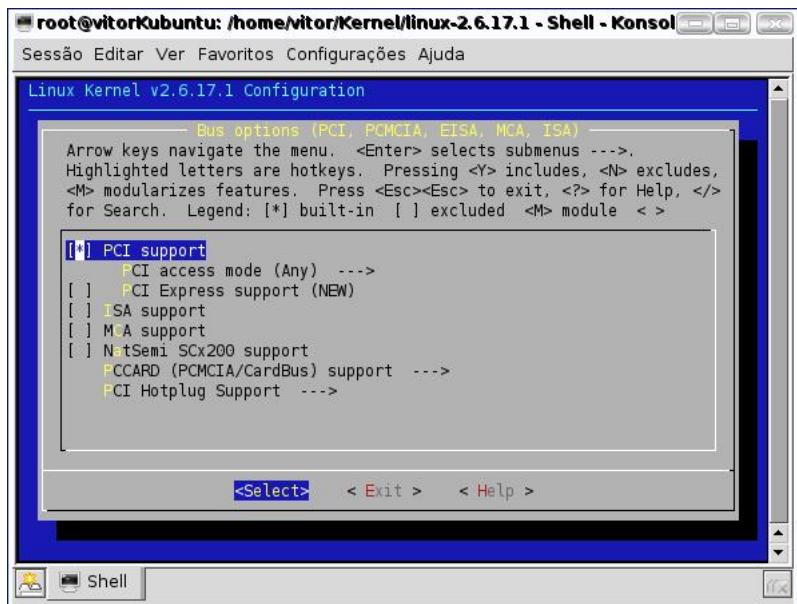


Figura 4.9: Configurando o Kernel - PCI 2/2

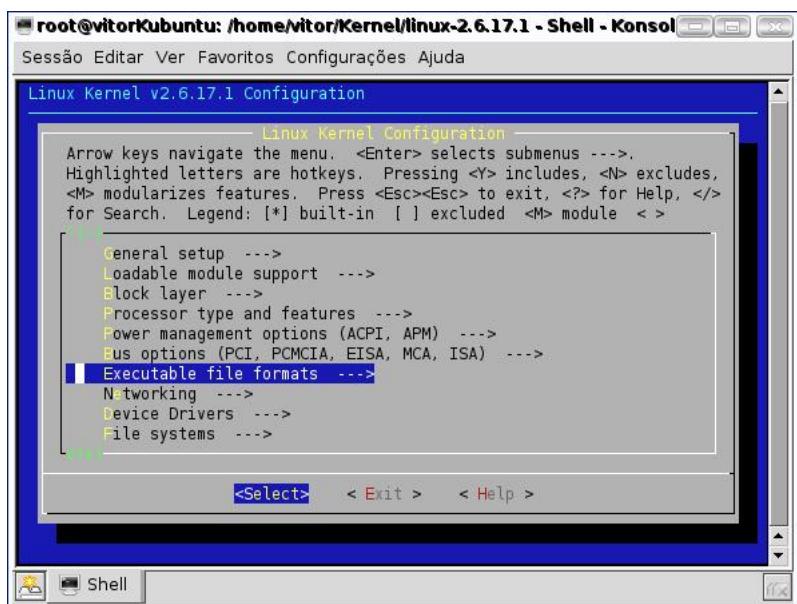


Figura 4.10: Configurando o Kernel - elf 1/2

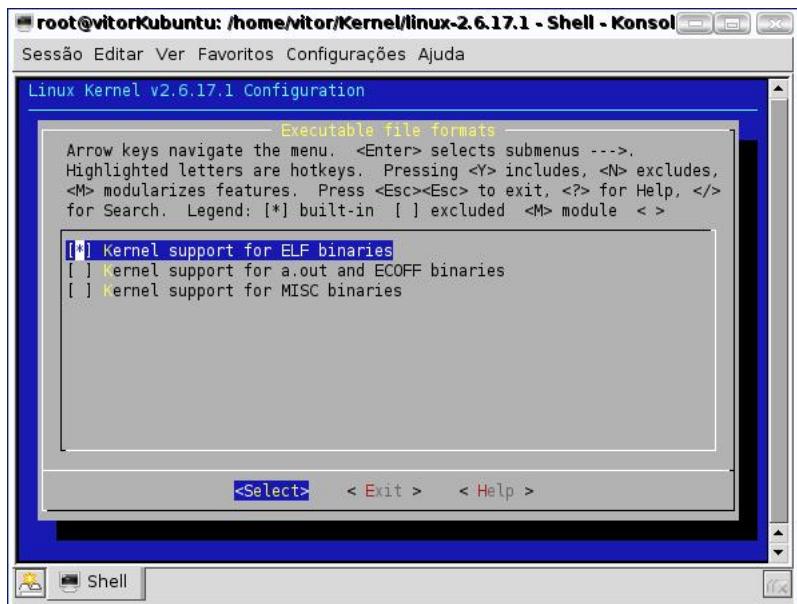


Figura 4.11: Configurando o Kernel - elf 2/2

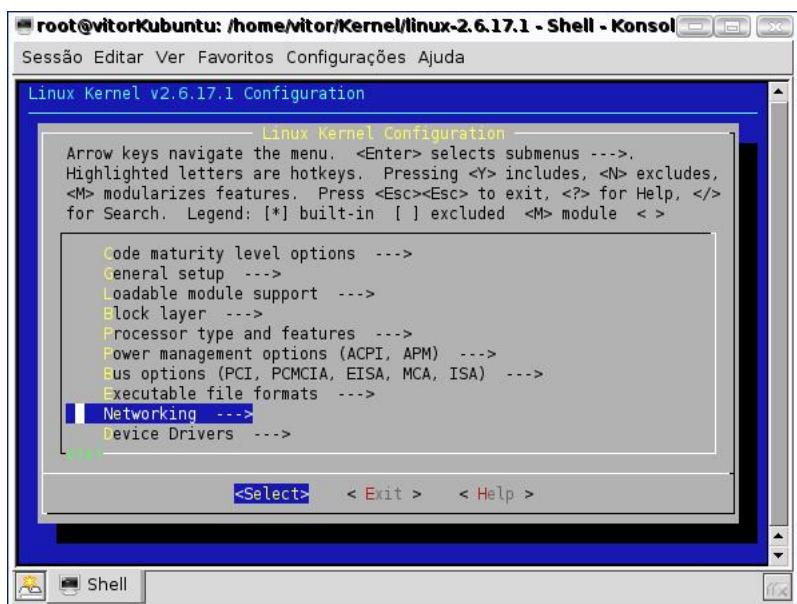


Figura 4.12: Configurando o Kernel - Rede 1/3

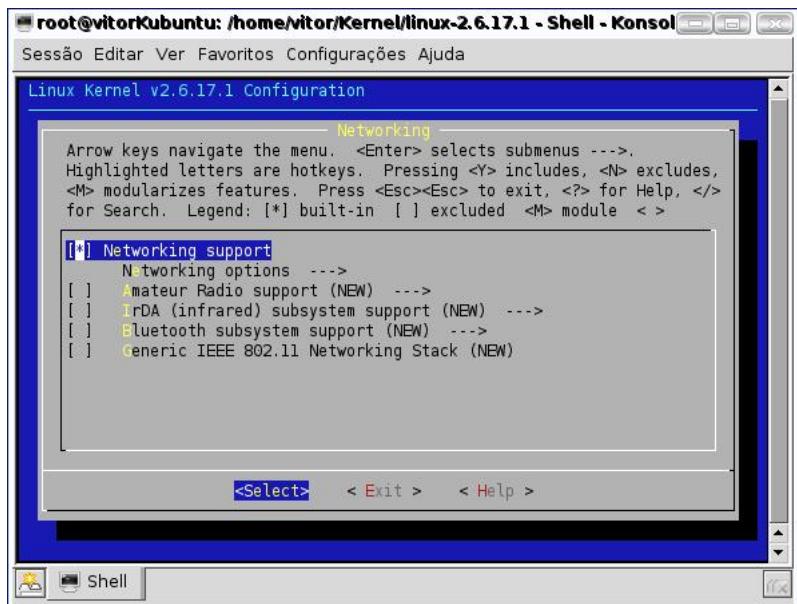


Figura 4.13: Configurando o Kernel - Rede 2/3

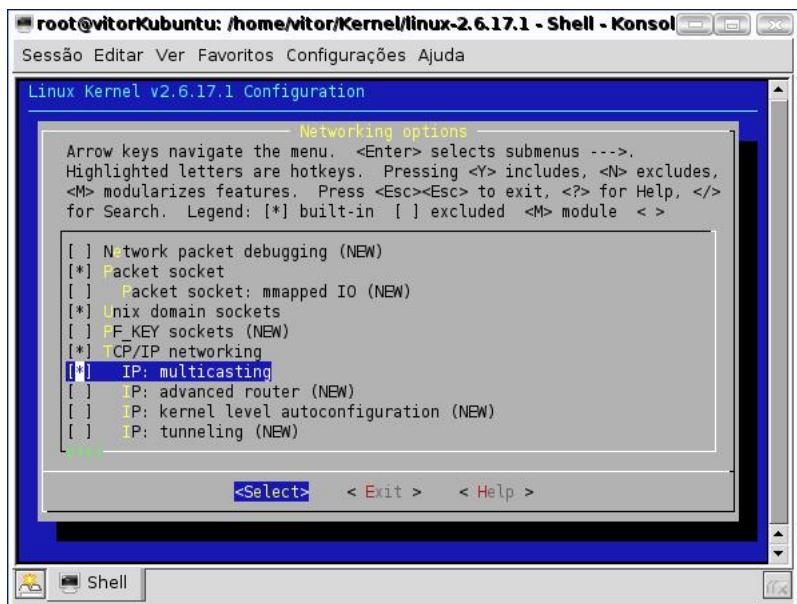


Figura 4.14: Configurando o Kernel - Rede 3/3

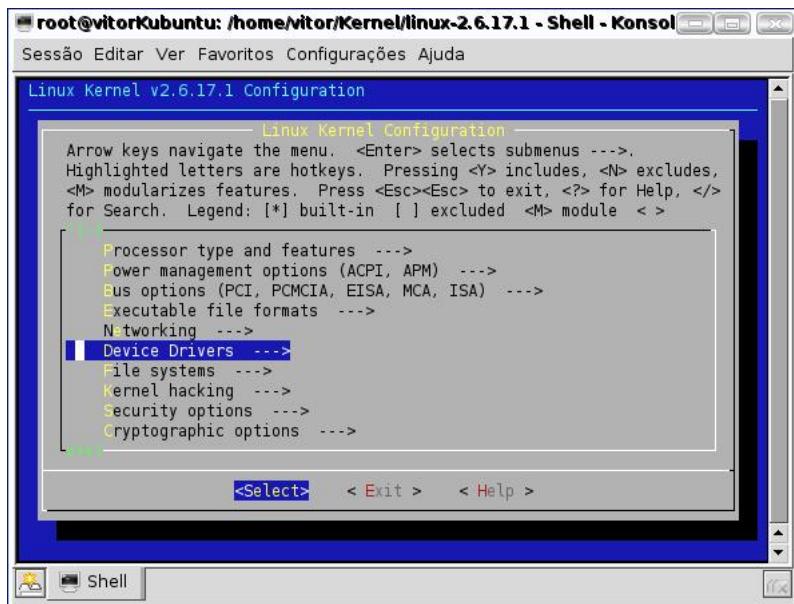


Figura 4.15: Configurando o Kernel - Drivers

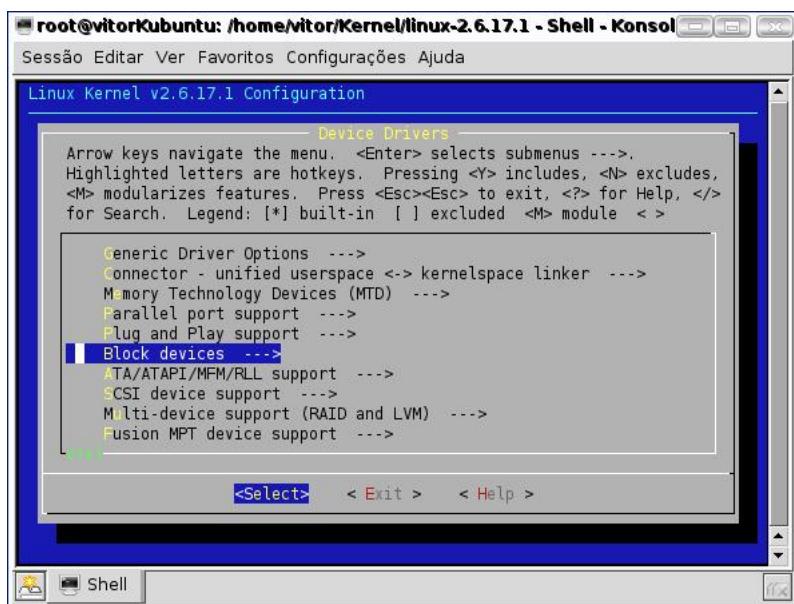


Figura 4.16: Configurando o Kernel - RAM 1/2

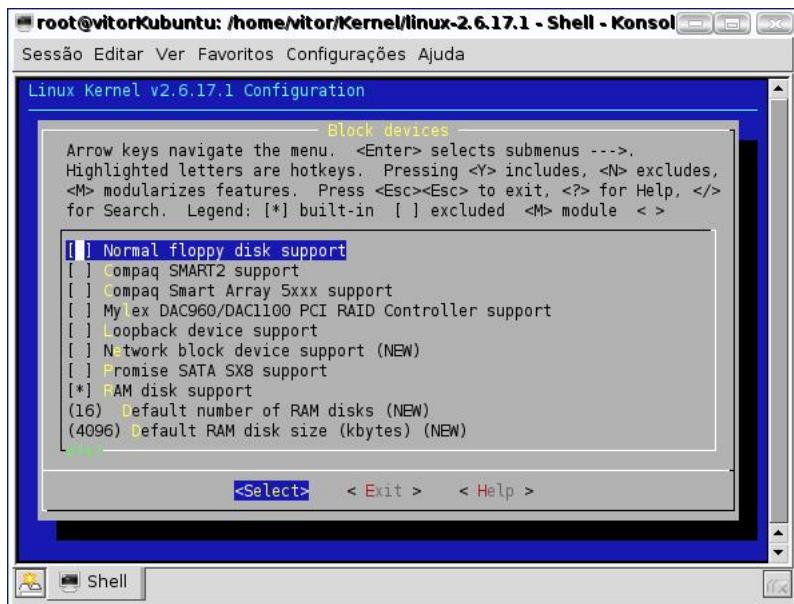


Figura 4.17: Configurando o Kernel - RAM 2/2

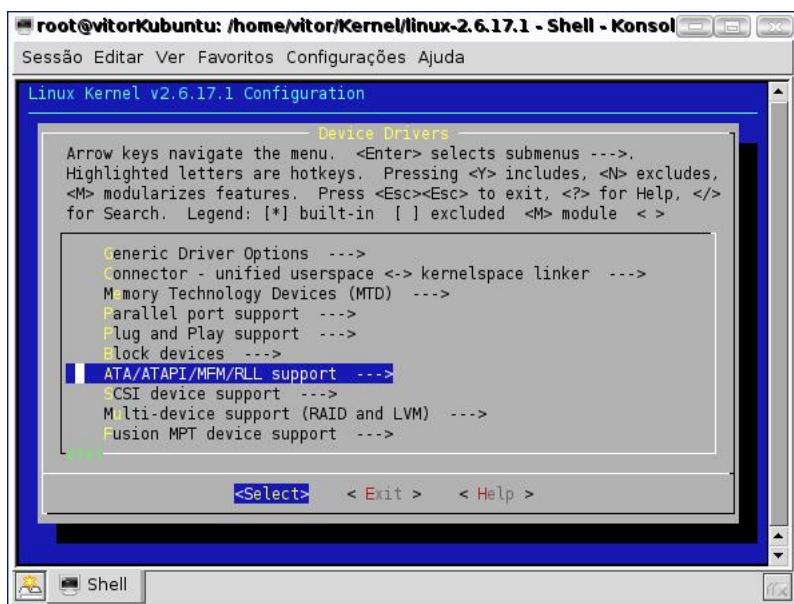


Figura 4.18: Configurando o Kernel - IDE 1/2

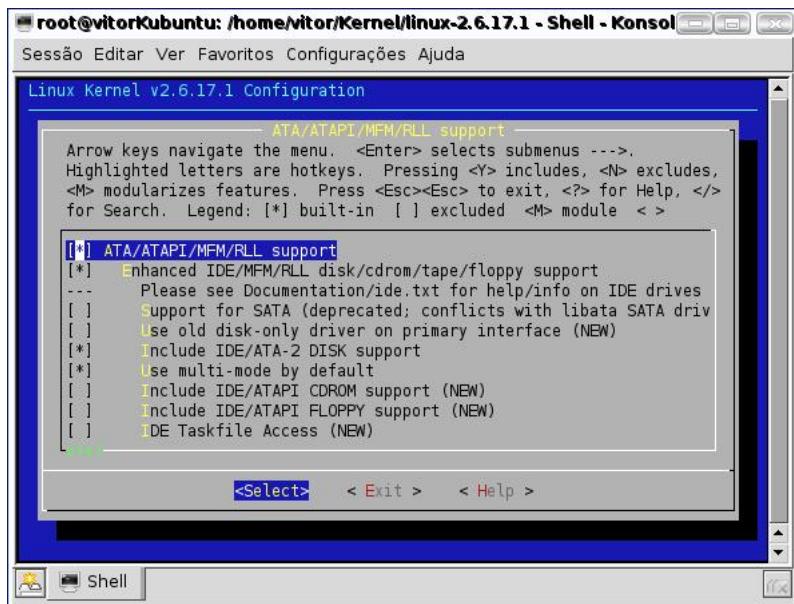


Figura 4.19: Configurando o Kernel - IDE 2/2

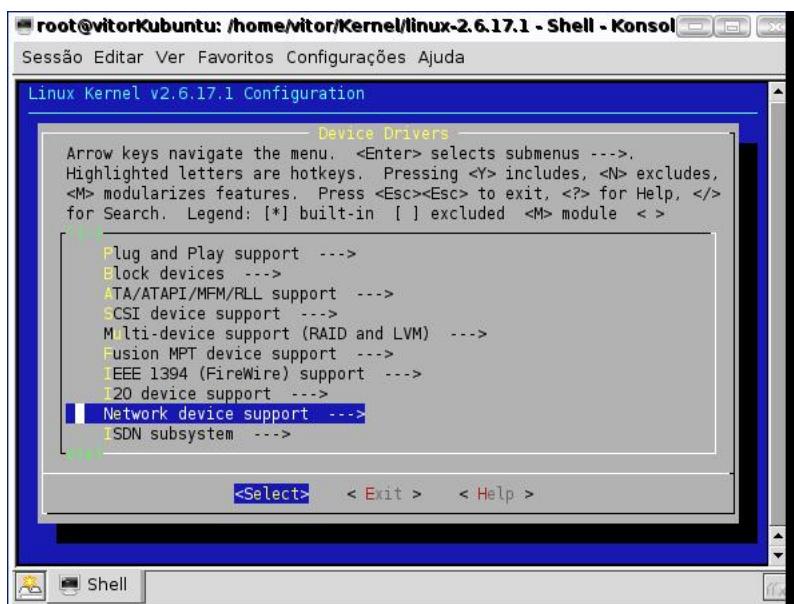


Figura 4.20: Configurando o Kernel - Placa de Rede 1/5

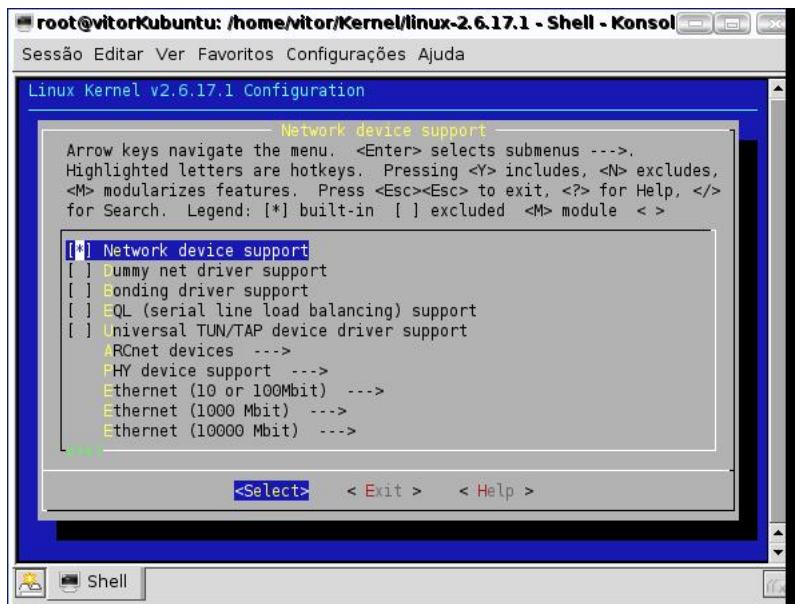


Figura 4.21: Configurando o Kernel - Placa de Rede 2/5

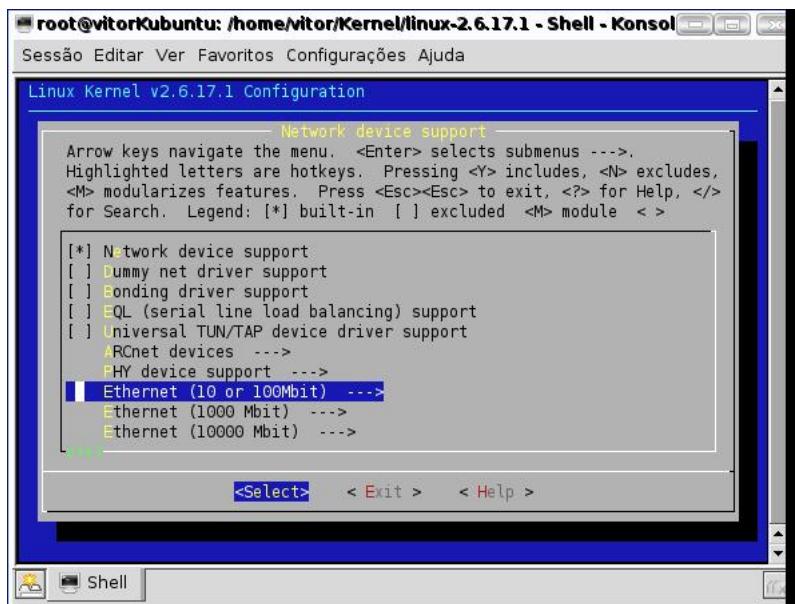


Figura 4.22: Configurando o Kernel - Placa de Rede 3/5

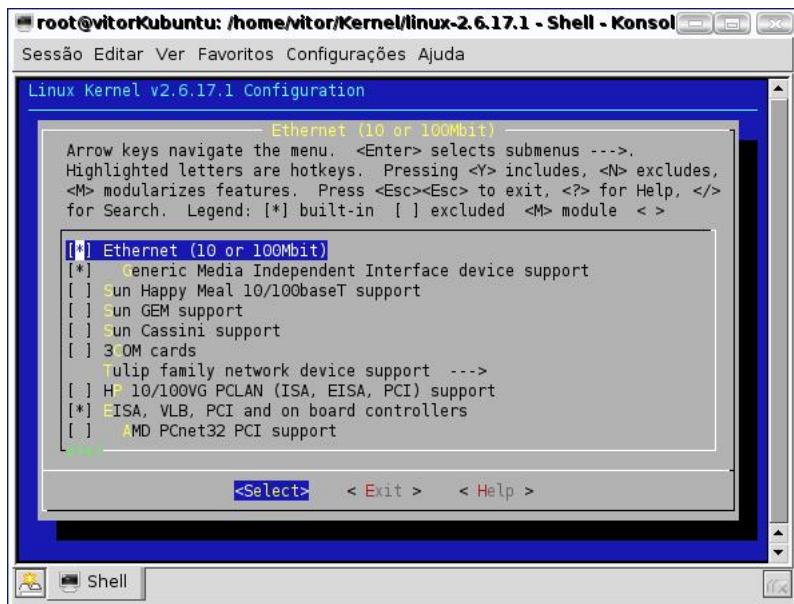


Figura 4.23: Configurando o Kernel - Placa de Rede 4/5

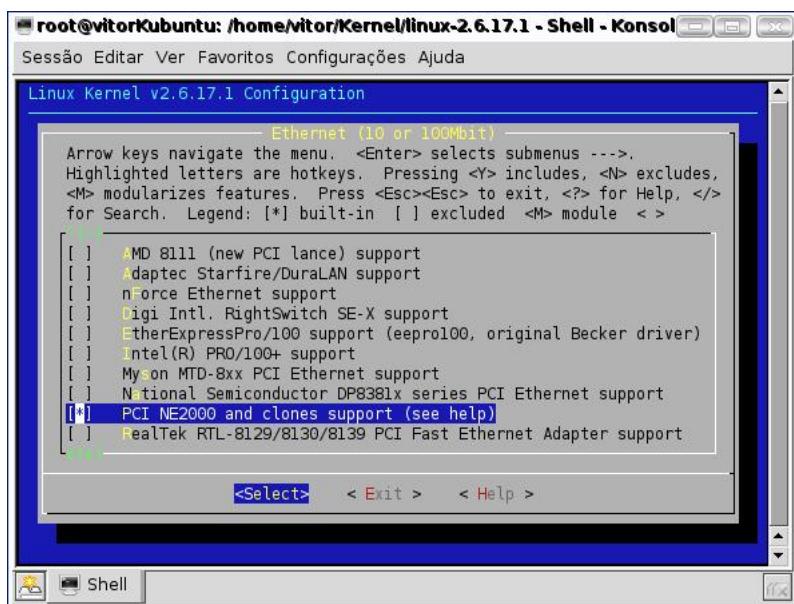


Figura 4.24: Configurando o Kernel - Placa de Rede 5/5

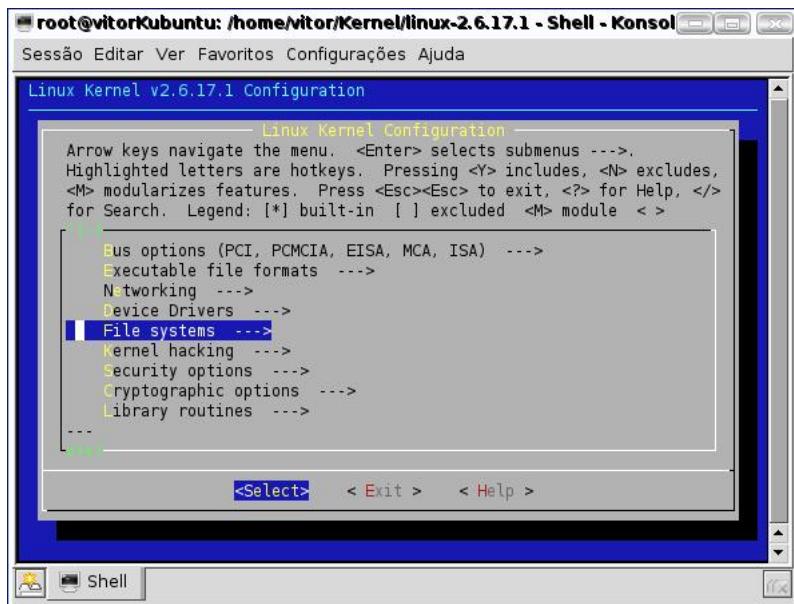


Figura 4.25: Configurando o Kernel - Suporte a Sistemas de Arquivos

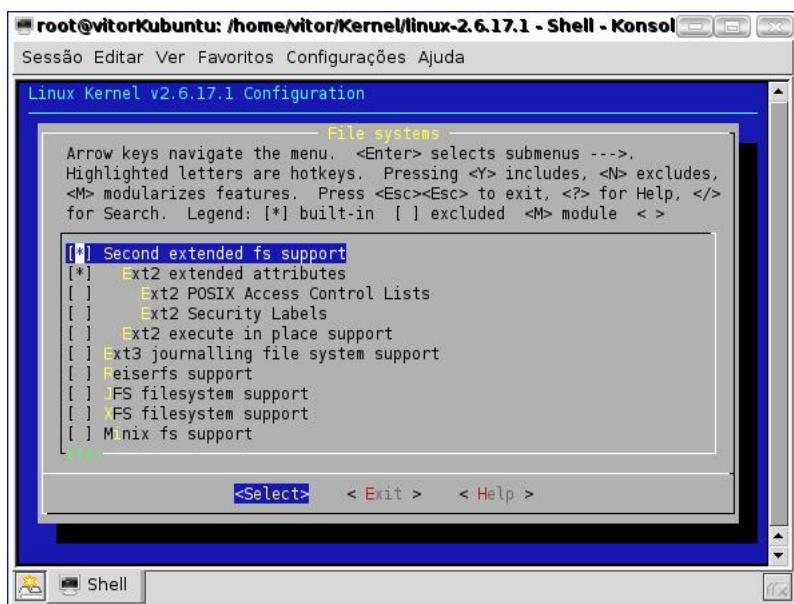


Figura 4.26: Configurando o Kernel - Ext2

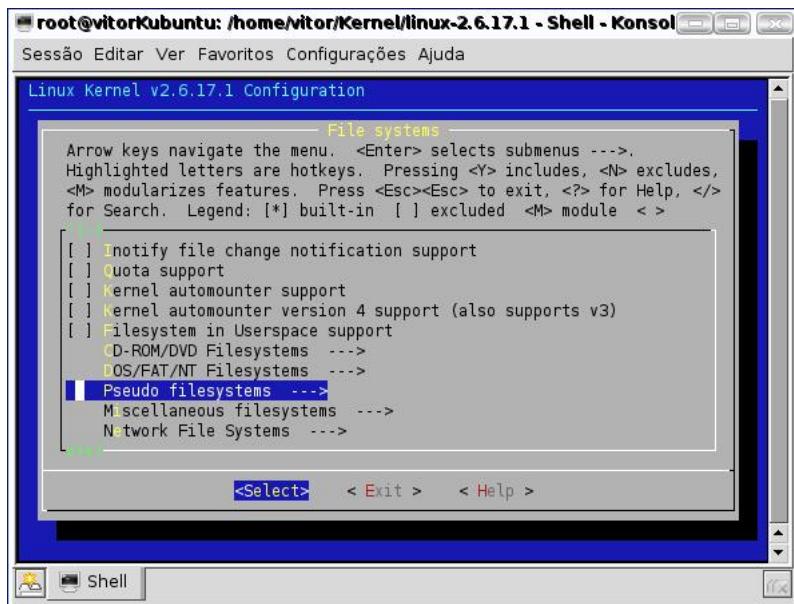


Figura 4.27: Configurando o Kernel - Proc 1/2

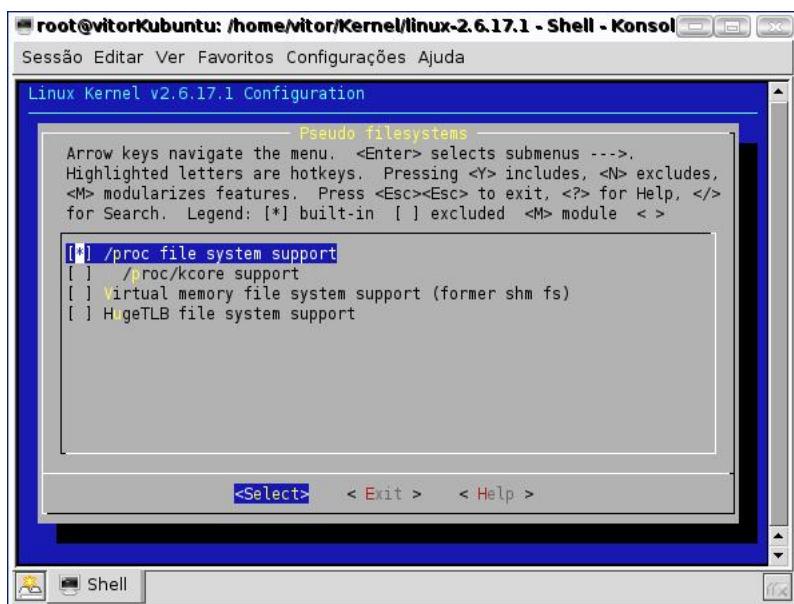


Figura 4.28: Configurando o Kernel - Proc 2/2

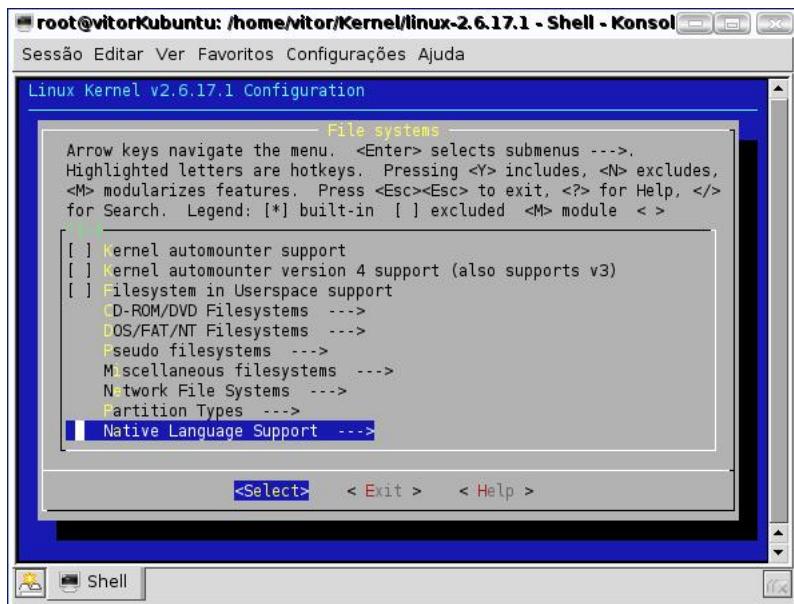


Figura 4.29: Configurando o Kernel - Linguagem Nativa 1/2

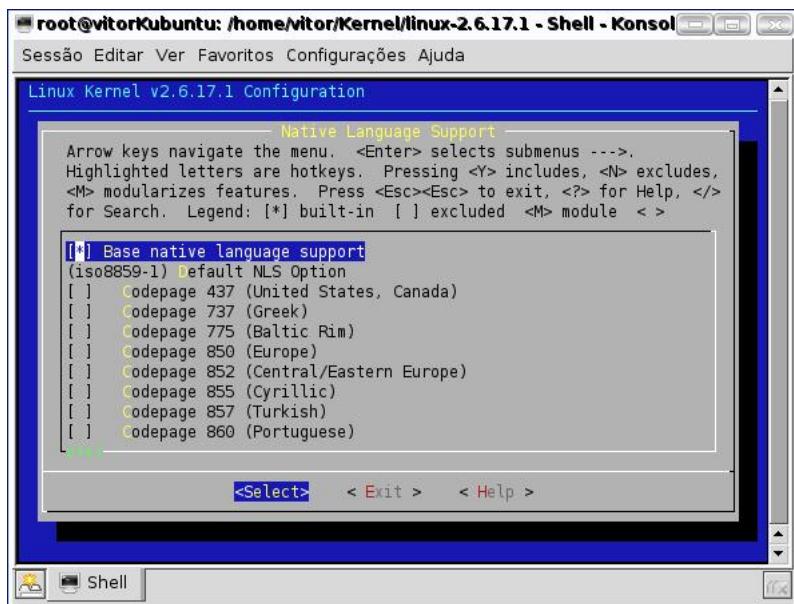


Figura 4.30: Configurando o Kernel - Linguagem Nativa 2/2

4.2.2 Compilação

Com as opções escolhidas, ao sair do menu elas são salvas e carregadas no arquivo `.config`, que será acessado quando o mesmo for compilado. Um comando importante de apresentar é o `make defconfig`, com ele podemos carregar as configurações padrões do kernel no arquivo `.config` e retornar às condições *default*.

Com as opções escolhidas, então podemos compilar kernel, para isso utilizamos os comandos `make` e `make install`. O kernel é gerado no diretório `/dirkernel/arch/i386/boot/bzImage`.

O uso de módulos do kernel é um processo prático, possibilita que novos módulos sejam adicionados sem que haja a necessidade da recompilação do mesmo. São compilados com o comando `make modules` e instalados com `make modules-install`.

4.3 Busybox

Feita a configuração e compilação customizada do kernel, agora partiremos para a geração de nosso rootfs. Para isso utilizaremos o Busybox. Outra opção é criar o rootfs manualmente, o que é bem mais trabalhoso e exigiria a compilação de cada aplicativo, além da criação manual da árvore de diretórios e dispositivos.

O Busybox utiliza um menu similar ao do kernel onde são checadas as opções desejadas e posteriormente acessadas na compilação. O Busybox disponibiliza ao desenvolvedor uma vasta gama de opções e serviços, mas dependendo da aplicação ainda pode ser necessário agregar serviços/aplicações específicas para um determinado projeto.

Veremos detalhadamente nas subseções seguintes um exemplo de como deve ser feito a configuração e compilação do busybox para gerarmos um rootfs mínimo, com poucas funções.

4.3.1 Personalizando o Busybox

Similar ao modo como ajustamos as opções para o kernel, podemos também configurar o busybox, só que agora utilizando o comando `make defconfig` para alterarmos as opções para as configurações padrões, depois entrar no menu através do mesmo `make menuconfig`. O Busybox será configurado para gerar um binário estático, evitando a libc, seguindo o menu nos itens: `Busybox Setings->Build Options->Build Busybox as Static Binary`, como mostram as Figuras 4.31, 4.32 e 4.33.

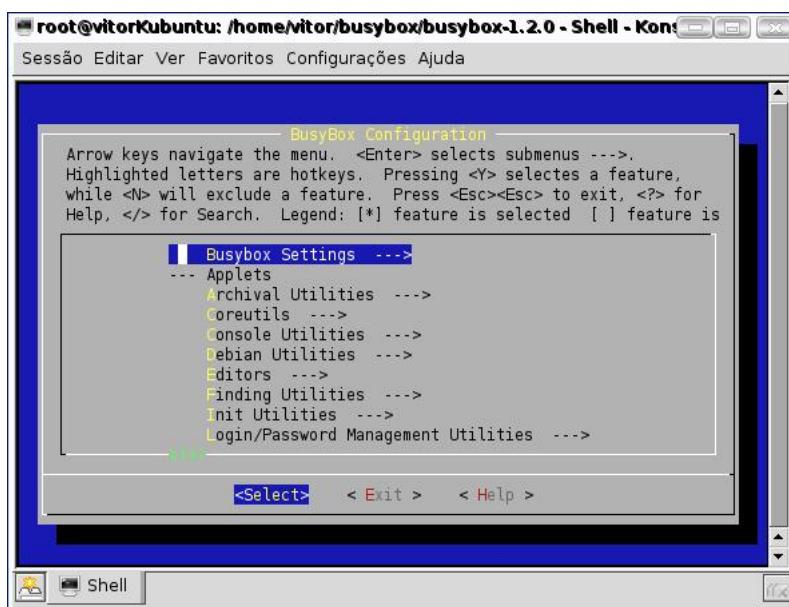


Figura 4.31: Configurando o Busybox com Estático 1/3

4.3.2 Compilação do Busybox

Seguindo a mesma lógica, compilamos o Busybox através do comando `make` e instalamos com o comando `make install`. Assim nossos executáveis são gerados no diretório `busybox/_install`. A

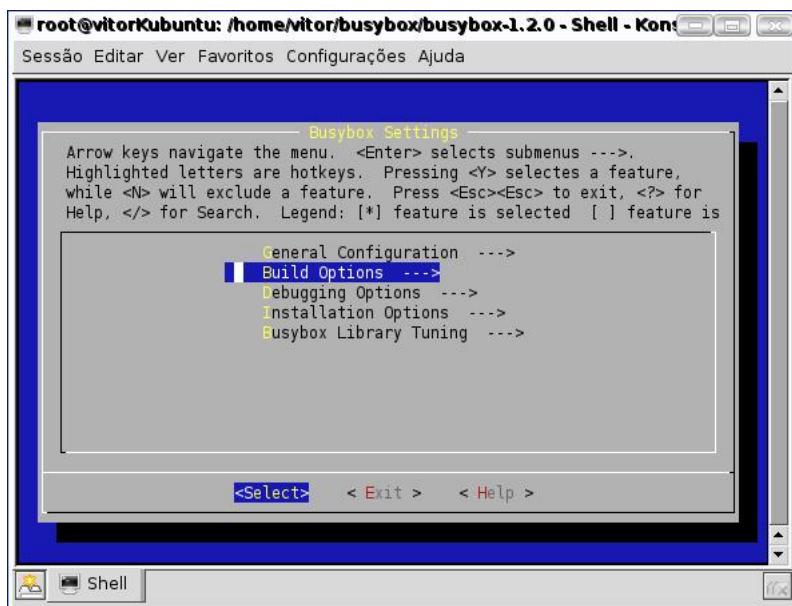


Figura 4.32: Configurando o Busybox com Estático 2/3

próxima tarefa será criar um sistema de arquivos onde as ferramentas geradas pelo Busybox serão copiadas. Para isso, um arquivo de 2MB será criado e formatado como se fosse um sistema de arquivos do tipo ext2. Depois, este arquivo será populado com os arquivos desejados. Dessa forma, criamos o rootfs.

Um dado relevante de sabermos, é qual o tamanho do sistema gerado. Dentro do diretório que foi gerado o rootfs, com o comando du -sh é apresentado o seu tamanho, já para o kernel, também no diretório onde ele foi gerado, utilizamos o comando ls -l <kernel>, conforme mostra a Figura 4.34.

A criação, formatação e montagem do rootfs pode ser feita da seguinte forma:

```
dd if=/dev/zero of=rootfs.img count=2 bs=1024k
/sbin/mkfs.ext2 -i 1024 -m 0 -F rootfs.img
mkdir rootfs
sudo mount -o loop rootfs.img rootfs
```

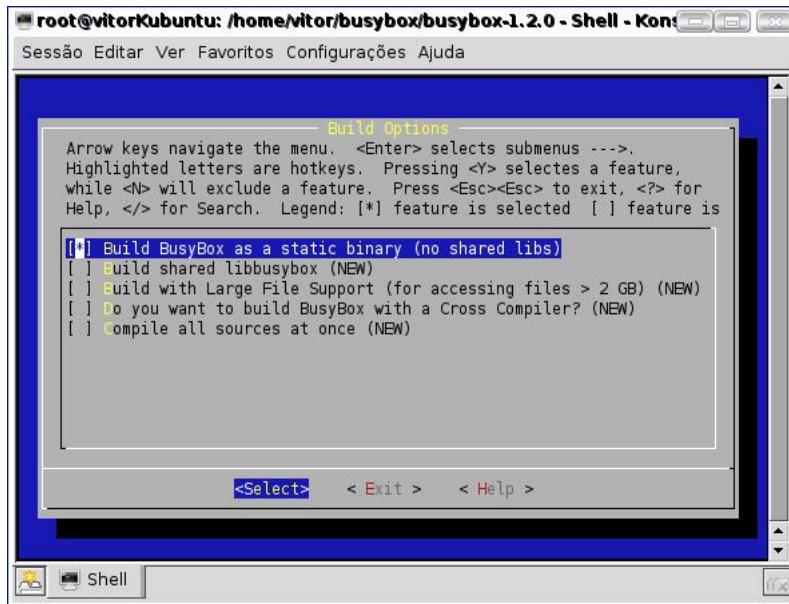


Figura 4.33: Configurando o Busybox com Estático 3/3

Montado, podemos popular nosso rootfs, que até o momento está vazio, com os binários gerados pelo Busybox, com o comando `cp -a /busyboxDir/_install/* /rootfs/`.

Agora criaremos todos os diretórios do nosso rootfs (dev,etc,proc,sys e tmp), os dispositivos (console, null e hda), os scripts de inicialização: init e rcS, e o ponto de referência da inicialização, o inittab com os seguintes comandos:

```
mkdir -p rootfs/{dev,etc,etc/init.d,proc,sys,tmp}
sudo mknod rootfs/dev/console c 5 1
sudo mknod rootfs/dev/null c 1 3
sudo mknod rootfs/dev/hda b 3 0
touch rootfs/etc/inittab
touch rootfs/etc/init.d/rcS
chmod a+x rootfs/etc/init.d/rcS
```

Um conteúdo mínimo para o inittab pode ser dado por:

```
vitor@vitorKubuntu:~/Kernel/linux-2.6.17.1/arch/i386/boot$ ls -l
total 1968
-rwxr-xr-x 1 root root    512 2006-12-06 05:27 bootsect
-rw-r--r-- 1 root root  1580 2006-12-06 05:27 bootsect.o
-rw-r--r-- 1 vitor vitor   2156 2006-06-20 06:31 bootsect.S
-rw-r--r-- 1 root root 935624 2006-12-06 05:27 bzImage
drwxr-xr-x 2 vitor vitor   4096 2006-12-06 05:27 compressed
-rw-r--r-- 1 vitor vitor   6124 2006-06-20 06:31 edd.S
-rw-r--r-- 1 vitor vitor  1413 2006-06-20 06:31 install.sh
-rw-r--r-- 1 vitor vitor   4737 2006-06-20 06:31 Makefile
-rw-r--r-- 1 vitor vitor   330 2006-06-20 06:31 mtools.conf.in
-rwxr-xr-x 1 root root  4602 2006-12-06 05:27 setup
-rw-r--r-- 1 root root   8276 2006-12-06 05:27 setup.o
-rw-r--r-- 1 vitor vitor  26129 2006-06-20 06:31 setup.S
drwxr-xr-x 2 vitor vitor   4096 2006-12-06 05:27 tools
-rw-r--r-- 1 vitor vitor  40535 2006-06-20 06:31 video.S
-rwxr-xr-x 1 root root 930504 2006-12-06 05:27 vmlinuz.bin
vitor@vitorKubuntu:~/Kernel/linux-2.6.17.1/arch/i386/boot$ ls -l bzImage
-rw-r--r-- 1 root root 935624 2006-12-06 05:27 bzImage
vitor@vitorKubuntu:~/Kernel/linux-2.6.17.1/arch/i386/boot$ cd /
vitor@vitorKubuntu:/$ cd home/vitor/busybox/busybox-1.2.0/_install/
vitor@vitorKubuntu:~/busybox/busybox-1.2.0/_install$ du -sh
1,6M .
vitor@vitorKubuntu:~/busybox/busybox-1.2.0/_install$
```

Figura 4.34: Verificando o Tamanho dos Arquivos Gerados

```
::sysinit:/etc/init.d/rcS
::askfirst:-/bin/sh
::restart:/sbin/init
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
```

A primeira linha instrui o init a executar o script rcS como a primeira ação de partida do sistema. A segunda define que o programa sh será o shell do sistema. As demais controlam as ações a serem tomadas em caso de re-inicialização, pressionamento das teclas ctrl+alt+del e desligamento, respectivamente.

Hora de definirmos o conteúdo do rcS:

```
#!/bin/ash
export PATH=/bin:/sbin:/usr/bin:/usr/sbin
# remonta o rootfs para leitura e escrita
mount -n -o remount,rw rootfs /
```

```
# monta os diretórios proc e sys
mount -n -t proc none /proc
mount -n -t sysfs sysfs /sys
# remove e cria novamente a mtab
rm /etc/mtab
cp /proc/mounts /etc/mtab
# configura rede local
/sbin/ifconfig lo up
/sbin/ifconfig eth0 192.168.0.2 netmask
255.255.255.0 up
/sbin/route add default gw 192.168.0.1
export TERM=linux
```

Uma vez criado o rootfs, é interessante colocar todos os arquivos com dono e grupo como administradores: `chmod root:root /rootfs/* -R`

Finalmente, o comando `sync` permite atualizar o conteúdo do rootfs, executando qualquer operação de escrita.

O Qemu é então executado, para iniciar a emulação. São passados para ele o kernel a ser usado, o sistema raiz e algumas opções usadas pelo kernel.

```
> qemu -kernel /arch/i386/boot/bzImage -hda rootfs.img
-append "root=/dev/hda clock=pit"
```

Desta forma, o sistema agora está em execução, pronto para ser introduzida uma aplicação específica, como por exemplo, um servidor web. Note que até a rede irá funcionar, deixando a emulação bem real, conforme pode-se analisar nas Figuras 4.35 e 4.36.

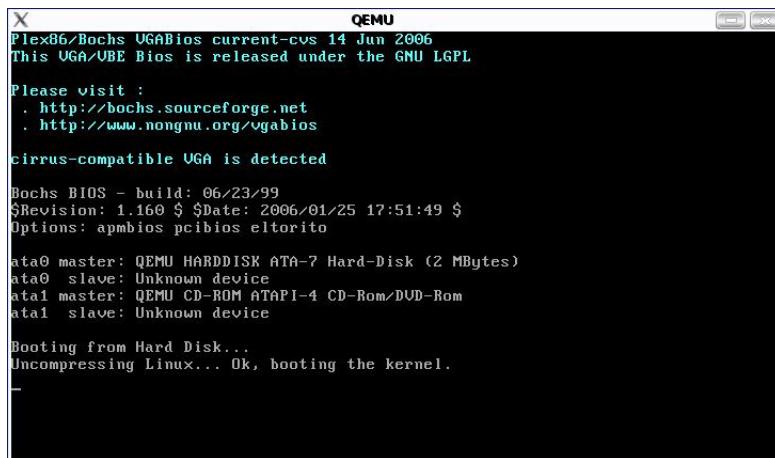


Figura 4.35: Simulando o Sistema Criado no Qemu.

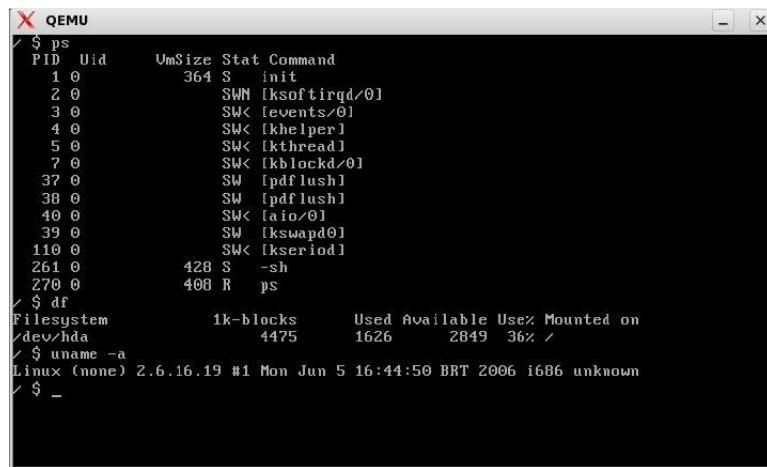


Figura 4.36: Sistema Carregado no Qemu.

4.3.3 Conclusão

Este capítulo mostrou que é possível criar sistemas embarcados com Linux, mesmo sem que se tenha uma plataforma de hardware.

Dessa forma, o trabalho de criação de um sistema embarcado pode ser acelerado quando se conta com ferramentas como o Qemu

e o Busybox.

Capítulo 5

Conclusão

Mesmo não levando em conta a área de sistemas embarcados, observamos o crescimento do Linux e a força que este sistema ganha a cada dia, sendo utilizado em diversas áreas como militar, governamental e corporativa. Isto decorrente das várias vantagens como baixo custo, alta confiabilidade, desempenho satisfatório, suporte, etc. Seu uso em desktops também cresce, com interfaces e recursos cada vez mais próximos das dos concorrentes.

Na área de sistemas embarcados, devido a grande demanda do mercado, seu uso vem em crescimento contínuo, principalmente em países do primeiro mundo e na Ásia. No Brasil, o Linux embarcado ainda caminha devagar: poucas empresas, material escasso, baixa oferta de treinamento e poucos profissionais qualificados. Tudo isto, aliado ao desconhecimento do sistema, acaba gerando uma demanda pequena.

Por se tratar de um mercado em grande expansão, com concorrência acirrada, a tendência é as empresas buscarem alternativas de sistemas operacionais que gerem custos menores para o projeto, fugindo cada vez mais do pagamento de *royalties* por unidade produzida e que tenha possibilidades reais de ganho de produtividade. E o Linux vem, com essa brecha, cada vez mais se consolidando como

uma excelente opção para ser acoplado aos mais variados projetos. Muito pela sua adequação de acordo com a aplicação, por esta capacidade de ser facilmente customizável e pelo amplo suporte a vários tipos de processadores e periféricos.

Um projeto com Linux pode ser, pelo menos inicialmente, mais complexo do que um projeto que admite Windows CE por exemplo, ou pode exigir pessoal mais qualificado na área para que o projeto seja bem sucedido. Porém o produto final fica bem mais ajustado para sua especificação e o desenvolvedor sabe exatamente o que está rodando por trás do seu sistema. A disponibilidade de código fonte é uma segurança adicional para o desenvolvedor.

Em determinados segmentos do mercado, como o caso de servidores web e firewalls, o uso do Linux é indiscutível, pela sua capacidade de rodar muito bem com poucos recursos e a alta familiaridade com recursos de rede.

No exterior o uso do Linux já está bem mais avançado e sendo cada vez mais inserido nas empresas, mesmo se tratando de pesquisas. Empresas como a Motorola vêm buscando uma linha cada vez maior de celulares com Linux. Fornecedores clássicos de sistemas embarcados como a WindRiver, vem fazendo esforços para prover soluções baseadas em Linux. No site <http://www.linuxdevices.com>, são encontrados uma infinidade de dispositivos que fazem uso deste sistema operacional, mostrando que muito já foi feito. Entretanto estamos ainda atrasados.

Mesmo em relação a material pesquisado, os principais livros, artigos, e sites encontrados estão em inglês, mais uma forma de se mostrar como a área é pouco explorada no Brasil. Aqui abre um espaço para ressaltar o site <http://www.linuxabordo.com.br>, uma iniciativa do Prof. Dr. Marcelo Barros de Almeida, uma das poucas fontes onde pode ser encontrado material em português de qualidade.

Um ponto fundamental do trabalho foi a utilização do QEMU. Utilizamos poucas de suas diversas funcionalidades, mas essa pode

ser uma ferramenta essencial em um projeto. Com ele dá pra se fazer quase de tudo em termos de simulação. Uma empresa que deseje projetar um determinado dispositivo, com a utilização deste emulador ela pode testar a sua aplicação em várias plataformas, teste de diversas funcionalidades que o sistema poderá admitir, sem a necessidade de perder tempo com a plataforma de hardware, deixando esta apenas para os testes finais e debugar eventuais erros nas aplicações.

Finalmente, ao contrário do que muitos pensam, é possível também usar sistemas como o Linux para executar e criar aplicações proprietárias, sem ferir nenhuma licença. O kernel Linux permite o seu uso sem maiores problemas. Apenas se modificações forem feitas diretamente nele é que se torna necessário a divulgação. O Busybox, se apenas usado como um executável qualquer, também não obriga que o código fonte seja liberado.

5.1 Trabalhos Futuros

Este trabalho apresentou vários conceitos sobre Linux embarcado e um exemplo prático para apresentar a aplicação dos mesmos, isso ainda é pouco ao analisar as várias opções existentes. Como visto, a área de sistemas embarcados com Linux é muito abrangente, poderíamos dar continuidade a este trabalho com uma série de abordagens, alguns trabalhos que dariam continuidade no estudo do Linux para sistemas embarcados.

Algumas abordagens:

Sistema Operacional: Utilizar um SO com suporte a tempo real, testar e analisar os resultados obtidos, verificar quais mantém uma maior estabilidade. Analisar qual é mais preciso e concluir quais as melhores opções para as diversas áreas que os utilizam.

Interface com o usuário: Implantar uma interface com o usuário através de toolkits gráficos, analisar desempenho das opções

existentes, analisar o aumento de tamanho do sistema, como são desenvolvidas interfaces customizadas, etc.

Qemu: No exemplo prático ele foi utilizado como simulador da plataforma x86, ele suporta outras plataformas também, fazer testes da simulação de uma outras plataformas, por exemplo ARM.

Outros Simuladores: Existem outras opções de simuladores a serem testadas. O Skyeye é uma, fazer comparações entre os simuladores, qual tem o melhor desempenho? qual é mais leve? quais as plataformas suportadas?, são análises que agregariam dados importantes ao trabalho.

Referências Bibliográficas

- [Abb03] Doug Abbott. *Linux for Embedded and Real-Time Applications*. Elsevier Science, 200 Wheeler Road Burlington, MA 01803, 2003.
- [Ale04] Emerson Alecrim. <http://www.infowester.com/linuxkernel.php>, 2004. Último Acesso: 20/10.
- [And05] Erik Andersen. <http://buildroot.uclibc.org/>, 2005. Último Acesso: 03/11.
- [And06] Erik Andersen. <http://www.busybox.net/about.html>, 2006. Último Acesso: 12/10.
- [ANP04] ANPEI. <http://www.anpei.org.br/engenhar/x01-2004/entrevista.aspx>, 2004. Último Acesso: 10/10.
- [Ban04] Linux Bangalore. <http://linux-bangalore.org/2004/schedules/DownloadFile.php?talkcode=F0200032\&type=other>, 2004. Último Acesso: 17/10.
- [Bel06] Fabrice Bellard. <http://fabrice.bellard.free.fr/qemu/about.html>, 2006. Último Acesso: 06/11.
- [Ben05] Onno Benschop. <http://itmaze.com.au/articles/linux-tiny/>, 2005. Último Acesso: 18/10.

- [Car05] Reinaldo Carvalho. <http://www.nautilus.com.br/~rei/material/artigos/sistema-de-boot.html>, 2005. Último Acesso: 02/11.
- [Cor04] Microsoft Corporation. http://www.compusoftware.com.br/embedded/espanol/html/conteudo/txt_inside03.htm, 2004. Último Acesso: 21/10.
- [dA03] Alan C. Assis; Marcelo Barros de Almeida. uclinux: O linux dos pequenos. *Revista do Linux*, pages 48–50, 2003.
- [dA06] Marcelo Barros de Almeida. <http://www.linuxabordo.com.br/download/artigos/linuxabordo.pdf>, 2006. Último Acesso: 20/09.
- [dBA05] Marcelo de Barros Almeida. <http://www.linuxabordo.com.br/download/artigos/ahistoriadetux.pdf>, 2005. Último Acesso: 17/10.
- [Dev06] Linux Devices. <http://www.linuxdevices.com/articles/AT9547755813.html>, 2006. Último Acesso: 08/09.
- [dH05a] Guia do Hardware. <http://www.guiadohardware.net/termos/directfb>, 2005. Último Acesso: 10/10.
- [dH05b] Guia do Hardware. <http://www.guiadohardware.net/termos/licenca-bsd>, 2005. Último Acesso: 21/10.
- [dO04] Rômulo Silva de Oliveira. Adaptações do linux para suportar aplicações com requisitos de tempo real. Master's thesis, Universidade Federal de Santa Catarina, 2004.
- [dR04] Vítor da Rosa. <http://www.inf.ufrgs.br/~flavio/ensino/cmp237/arm7.pdf>, 2004. Último Acesso: 22/10.

- [eCo06] eCos. <http://ecos.sourceforge.org/>, 2006. Último Acesso: 23/10.
- [Eth06] Power Over Ethernet. http://www.poweroverethernet.com/products.php?article_id=378, 2006. Último Acesso: 25/10.
- [Fou] Wikimedia Foundation. http://encyclopedia.kids.net.au/page/em/Embedded_system. Último Acesso: 29/10.
- [Fou06a] Wikimedia Foundation. http://en.wikipedia.org/wiki/Embedded_system, 2006. Último Acesso: 29/10.
- [Fou06b] Wikimedia Foundation. <http://pt.wikipedia.org/wiki/Kernel>, 2006. Último Acesso: 20/10.
- [Fou06c] Wikimedia Foundation. <http://www.answers.com/topic/inferno-operating-system>, 2006. Último Acesso: 23/10.
- [Goo06] Google. http://www.google.com.br/search?hl=pt-BR&lr=&cr=countryBR&defl=pt&q=define:BIOS&sa=X&oi=glossary_definition&ct=title, 2006. Último Acesso: 22/11.
- [Gum06a] Gumstix. http://www.gumstix.com/store/catalog/product_info.php?products_id=170, 2006. Último Acesso: 25/10.
- [Gum06b] Gumstix. http://www.gumstix.com/store/catalog/product_info.php?products_id=170, 2006. Último Acesso: 15/10.
- [Hol00a] Ziff Davis Publishing Holdings. <http://www.linuxdevices.com/articles/AT8844506693.html>, 2000. Último Acesso: 21/11.

- [Hol00b] Ziff Davis Publishing Holdings. <http://www.linuxdevices.com/news/NS8925620279.html>, 2000. Último Acesso: 28/10.
- [Hol01] Ziff Davis Publishing Holdings. <http://www.linuxdevices.com/links/LK9936231852.html>, 2001. Último Acesso: 29/10.
- [Hol04] Ziff Davis Publishing Holdings. <http://www.linuxdevices.com/news/NS5853072179.html>, 2004. Último Acesso: 31/10.
- [Hol06a] Ziff Davis Publishing Holdings. <http://www.linuxdevices.com/articles/AT7070519787.html>, 2006. Último Acesso: 04/10.
- [Hol06b] Ziff Davis Publishing Holdings. <http://www.linuxdevices.com/articles/AT7679125282.html>, 2006. Último Acesso: 25/10.
- [Hol06c] Ziff Davis Publishing Holdings. <http://www.linuxdevices.com/articles/AT7679125282.html>, 2006. Último Acesso: 15/10.
- [Hun04] Andreas Hundt. http://www.directfb.org/docs/DirectFB_overview_V0.2.pdf, 2004. Último Acesso: 29/10.
- [Ish06] André Ishii. http://br-linux.org/linux/mandriva_conectiva_e_freescale_realizam_seminarios_sobre_linux_embarcado, 2006. Último Acesso: 26/10.
- [LD06] Windows for Devices Linux Devices. <http://www.linuxdevices.com/articles/AT6743418602.html>, <http://www.linuxdevices.com>, [http:](http://)

- //www.windowsfordevices.com, 2006. Último Acesso: 13/10.
- [Met04] Jerry Metz. Portabilidade do linux e viabilidade em desktop. Master's thesis, Universidade Federal de Lavras, 2004.
- [Nuo06] Vita Nuova. <http://www.vitanuova.com/inferno/index.html>, 2006. Último Acesso: 23/10.
- [Pac05] Keith Packard. <http://www.xfree86.org/current/TinyX.1.html>, 2005. Último Acesso: 28/10.
- [PHG05] Uirauna Imirim Caetano Pedro Henrique Gomes, Tatiane Silvia Leite. <http://www.ic.unicamp.br/~rodolfo/Cursos/mc722/2s2005/Trabalho/g20-arm-apresentacao.pdf>, 2005. Último Acesso: 22/10.
- [Roc05] Rodrigo Rocha. http://www.compusoftware.com.br/embedded/html/conteudo/txt_inside05.htm, 2005. Último Acesso: 20/10.
- [Sem06] Freescale Semiconductor. <http://www.freescale.com/webapp/sps/site/overview.jsp?nodeId=0162468rH3Y TLC00M9>, 2006. Último Acesso: 18/10.
- [Som04] Patanjali Somayaji. <http://linux-bangalore.org/2004/schedules/DownloadFile.php?talkcode=F0200032&type=other>, 2004. Último Acesso: 04/10.
- [Tan01] Andrew S. Tanenbaum. *Modern Operational Systems*. Pearson, 80 Strand, London, WC2R 0RL, 2001.
- [Ter06] Terra. <http://www.softwarelivre.org/news/5581>, 2006. Último Acesso: 21/10.

- [uCl06] uClinux. <http://www.uclinux.org/ports/>, 2006. Último Acesso: 15/10.
- [Wat03] Leslie Harley Watter. <http://www.pr.gov.br/batebyte/edicoes/1999/bb93/estagiario.htm>, 2003. Último Acesso: 12/10.
- [Wik06] Wikipedia. <http://de.wikipedia.org/wiki/Bild:QtopiaPhone.jpg>, 2006. Último Acesso: 16/10.
- [Yag03] Karim Yaghmour. *Building Embedded Linux Systems*. O'Reilly, 1005 Gravenstein Highway North Sebastopol, CA 95472, 2003.