1 USO DE PCS NO CONTROLE DE SISTEMAS DE TEMPO REAL

O *hardware* e sua arquitetura têm papel fundamental no suporte ao funcionamento do sistema operacional. Talvez um dos recursos mais importantes fornecido pelo hardware ao sistema operacional são as interrupções. Graças as interrupções, é possível disparar eventos de forma assíncrona, sem a necessidade de realizar *polling*. Quando várias interrupções estão pendentes, o *hardware* do computador seleciona automaticamente a de maior prioridade. Em alguns casos, interrupções podem ser interrompidas por outras interrupções de maior prioridade. Tempos dos ISRs são constantemente pesquisados e estudados por projetistas de sistemas operacionais, sendo cuidadosamente medidos e publicados por fabricantes de software e usuários [Shaw, 2003]. Entretanto, todas as características discutidas nesta seção do texto, indicam uma arquitetura projetada para ter a maior velocidade possível e não o melhor determinismo possível. Todas estas melhorias nas arquiteturas modernas de computadores tornam o sistema impossível de ser analisado do ponto de vista de performance [Laplante, 2004]. Por outro lado, vem observando-se cada vez mais o uso de PCs genéricos para controle de sistemas de tempo real. Isto de fato é possível utilizando-se algumas técnicas e cuidados.

Mantegazza e Dozio [L. Dozio, 2003] propõem o uso de computadores desktop (GPCPU – *General Purpose* CPU) para implementar tarefas de controle de alta frequência a baixo custo. Utilizando-se um RTOS, além de se poder implementar esta plataforma de controle de alta performance a baixo custo, ainda ganha-se a vantagem de uso dos recursos disponíveis em computadores desktop, como as unidades para cálculo de ponto flutuante.

De acordo com Cawleld, é possível implementar sistemas de controle baseados em PCs com alta confiabilidade usando técnicas de tolerância a falhas. Além disto, nenhum sistema operacional para PCs pode ser certificado como livre de erros [Cawleld, 1997].

Outro caso de sucesso recente da possibilidade de uso da arquitetura genérica x86 em sistemas de tempo real, foi sua utilização em aplicações críticas militares em um sistema de defesa desenvolvido pela IBM para a marinha dos Estados Unidos [McKenney, 2008], com diversas restrições temporais. Em uma unidade de controle de

experimentos com plasma, o RTAI substituiu um sistema baseado em LynxOS, com mesma confiabilidade e adicionando ganho de performance [Centioli et al., 2004], além de reduzir custos, trocando um hardware especializado baseado em VME por um hardware genérico baseado em PC. Embora a arquitetura PC nem sempre seja a ideal para sistemas embarcados, devido a problemas de cache e outros recursos que foram feitos para aumentar a performance prejudicando o determinismo, muitos sistemas de tempo real utilizam a arquitetura PC, devido a seu baixo custo graças a sua economia de escala. Um exemplo é o robô autônomo subaquático da universidade de McGill no Canadá, que usa placas de um PC/1044 para o controle do robô chamado de Aqua [Theberge et al., 2006]. Neste robô, um outro conceito interessante é utilizado. Tanto o sistema operacional de tempo real QNX quanto o Linux convencional (não de tempo real) são utilizados simultaneamente em dois processadores diferentes, para beneficiar-se das vantagens de cada um dos sistemas operacionais [Dudek et al., 2007].

2 KERNELS

A tradução direta do inglês da palavra kernel significa núcleo. De fato o kernel é o núcleo de qualquer sistema operacional, provendo uma interface entre hardware e software. Como pode-se ver na Figura 1, o kernel é o único meio através do qual as bibliotecas e aplicativos podem acessar o hardware. Da mesma forma, informações que chegam do mundo externo através do hardware são recebidas pelo kernel e em seguida encaminhas para o software associado a esta informação. Também pode ser visto na Figura, que normalmente os aplicativos finais utilizam bibliotecas para fazer chamadas de sistema (*system calls*) que são chamadas de funções que fazem com que o kernel seja invocado para realizar alguma tarefa que só o sistema operacional pode fazer, como abrir um arquivo, ou obter uma informação da rede.

De acordo com Ganssle [Ganssle, 2004], um kernel deve oferecer os seguintes serviços :

1. Permitir dividir uma aplicação em tarefas;

2. Disponibilizar um relógio (*clock tick*) que permita suspender tarefas até um determinado momento ou para impor tempos limite;

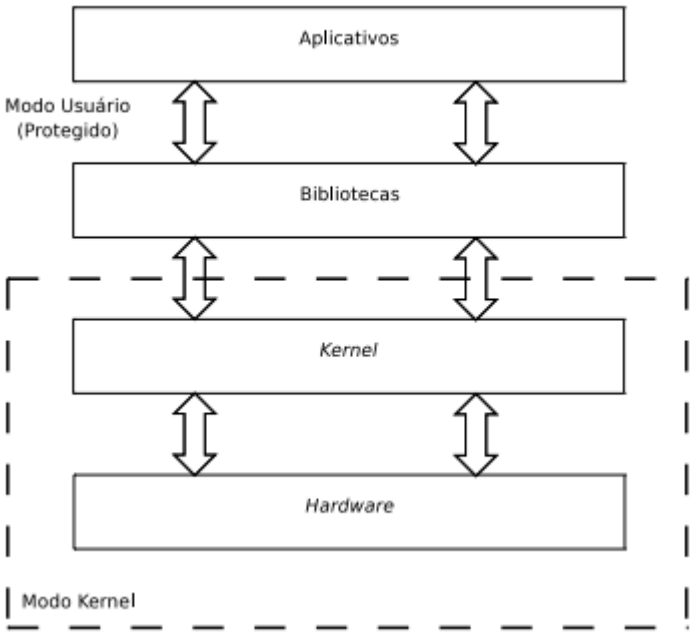

Figura 12: Interfaces de um kernel

3. A possibilidade de tarefas ou rotinas de tratamento de interrupção enviarem mensagens para outras tarefas;

4. Sempre executar a tarefa de maior prioridade que está pronta para execução. O escalonador é executado toda vez que ocorre um evento e existe a possibilidade de uma tarefa mais importante ser executada;

5. Realizar a troca de contexto, que é usada para salvar o estado atual de uma tarefa e restaurar o estado de uma tarefa mais importante.

Os sistemas operacionais podem ser bastante simples, não oferecendo serviços como rede ou acesso a arquivos, mas apenas escalonamento de processos, ou mais complexos, oferecendo diversos serviços para as bibliotecas e aplicativos. Muitos sistemas operacionais consistem simplesmente de um kernel. Um dado muito importante e frequentemente questionado refere-se a sobrecarga imposta por um sistema operacional em um computador. Em geral um sistema operacional de tempo real aumenta o uso do processador de 2% a 4% [Labrosse, 2002], porém muitas vezes um hardware melhor pode não ajudar por limitações no software. De acordo com Weiss [Weiss et al., 1999], existem situações em que 77% do tempo de execução de uma tarefa é consumido pelo RTOS, de forma que o RTOS torna-se um fator limitante da performance do sistema. Independente do seu tipo, um kernel faz com que o tempo de resposta de tarefas de alta prioridade seja virtualmente inalterado ao adicionar novas

tarefas de menor prioridade [Ganssle, 2004]. Isto é muito importante para construir sistemas escaláveis e confiáveis. Normalmente, o kernel utiliza recursos disponibilizados pelo processador e MMU para separar de forma segura o núcleo do sistema e as tarefas em execução, como pode ser visto na Figura 1. Quando o kernel inicia a execução de uma tarefa, ele coloca o processador em um modo chamado e modo protegido (ou modo usuário), reduzindo o número de instruções possíveis e posições de memórias que podem ser acessadas, para que uma tarefa não possa afetar em nenhum aspecto o estado de outras tarefas. Como o modo protegido é bastante limitado, as tarefas precisam fazer pedidos para o kernel, como acessar a rede, usar uma porta de entrada e saída, ou ler e escrever no disco. Quando funções internas do kernel estão sendo executadas, o código em execução tem acesso irrestrito a memória e hardware. Este modo é o modo kernel, que é o modo em que o kernel é executado. Durante a execução em modo protegido, qualquer operação ilegal executada pelos programas vai gerar uma interrupção de software sinalizando o erro, que irá imediatamente interromper a execução da tarefa que causou o erro, e avisar o sistema operacional, para que uma decisão sobre o que deve ser feito seja tomada.

Um kernel pode ser não preemptivo ou preemptivo. A maioria dos *kernels* atuais são preemptivos, de forma que o kernel sempre tenta executar a tarefa de maior prioridade que esteja pronta para execução. Embora um kernel possa ser preemptivo com relação as tarefas que ele esteja controlando, deve-se observar o fato de frequentemente o kernel em si, não ser preemptivo. Isto quer dizer que se um kernel estiver executando alguma tarefa interna do kernel e tiver desativado todas interrupções até terminar esta tarefa crítica, por mais importante que seja o evento ou tarefa que aconteça neste intervalo de tempo, ela só será detectada e tratada quando a execução da tarefa crítica atual terminar.

O próprio kernel do Linux não pode ser preemptado em alguns momentos, pois ele desabilita as interrupções enquanto está executando operações críticas [Aarno, 2004a]. Na verdade, partes do kernel do Linux podem ser preemptadas e algumas partes não, dependendo da sua importância e necessidade de execução atômica. Outra ocasião em que as interrupções podem acabar sendo desabilitadas por segurança está na execução de funções reentrantes. Uma função reentrante basicamente utiliza seus dados de forma que mesmosendo executada várias vezes de forma simultânea (por exemplo, sendo chamada a partir de várias tarefas), sua resposta é correta. A melhor opção para tornar uma determinada porção de código reentrante, é eliminar o uso de variáveis

globais. Contudo, as variáveis globais são a maneira mais rápida de trocar dados em um programa, não sendo possível eliminar todas as variáveis globais em sistemas de tempo real [Ganssle, 2004]. Dessa forma, a abordagem mais comum é desabilitar as interrupções em um código reentrante e reativar as interrupções ao terminar de executar o código reentrante. Isto certamente aumenta a latência do sistema e reduz sua habilidade de responder a eventos externos em tempo [Ganssle, 2004].

Um kernel que ca com as interrupções desabilitadas muito tempo (para tratar regiões críticas), acaba aumentando a latência das interrupções [Farines et al., 2000, Beal, 2005]. De acordo com Labrosse, a especificação mais importante de um sistema de tempo real é a quantia de tempo que as interrupções ficam desligadas [Labrosse, 2002].

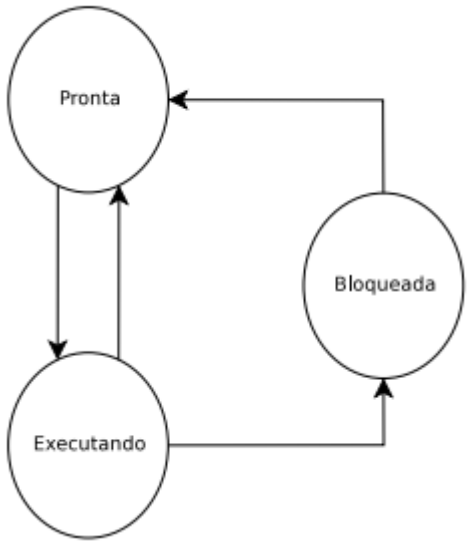

Figura 2: Diagrama simplificado de possíveis estados de uma tarefa

O coração de um sistema de tempo real é o escalonador de tarefas, que decide qual tarefa será executada, quando, e por quanto tempo. A maior parte dos escalonadores utiliza um tempo limite para preemptar a tarefa em execução atualmente. Claramente, o mecanismo de temporização deve ser bastante preciso, caso contrário tarefas podem ser executadas antes ou depois do momento ideal [Kailas e Agrawala, 1997]. A Figura 2 mostra os possíveis estados de uma tarefa e suas transições. Após ser criada, uma tarefa fica pronta. Isto quer dizer que ela está na la aguardando para ser executada. O sistema operacional pode colocá-la em execução em qualquer momento, e

também parar (preemptar) sua execução a qualquer momento, caso uma tarefa de maior prioridade precise ser executada. Caso uma tarefa em execução solicite o uso de algum recurso, como um arquivo em disco, rede, uma porta de entrada e saída, o kernel bloqueia esta tarefa até que o recurso esteja disponível para ser utilizado. Quando o recurso estiver disponível, a tarefa volta a car pronta, para só em seguida poder ser executada novamente.

Quando nenhuma tarefa está sendo executada, normalmente, o sistema operacional executa uma tarefa chamada de Tarefa Ociosa do Sistema (Idle Task) [Ganssle, 2004], que embora não faça nada, representa o tempo ocioso do sistema. Em alguns sistemas com necessidades de economia de energia, é neste momento que coloca-se o processador em modo de baixo consumo, pois qualquer interrupção irá trazer a execução de volta ao estado convencional.

A tarefa de tempo ocioso do sistema também é útil para calcular a porcentagem de uso que o sistema está demandando do processador. Se T é o tempo que a *Idle Task* consome do processador, a carga total de processamento do sistema é (1-T). A frequência com que a interrupção de *clock tick* interrompe os programas para executar o sistema operacional, define também o *time quantum* [Calandrino e Anderson, 2006]. O time quantum, é a menor fatia de tempo de processamento que o escalonador pode fornecer a uma tarefa. Dessa forma, se o *clock tick* for de 1000Hz, o sistema terá um time quantum de 1ms. Se uma tarefa durar 1,1ms, ela precisará de 2 *time quantums*. Existem relatos de sistemas de tempo real com *clock tick* de 3750Hz, levando a um time quantum de 250μs, sem aumentar consideravelmente o overhead do sistema operacional [Calandrino e Anderson, 2006].

O número de tarefas no sistema também contribui para o aumento da latência, pois o sistema operacional costuma desabilitar as interrupções enquanto percorre a lista de tarefas pendentes [Laplante, 2004]. Sabe-se que existe uma relação quadrática entre o overhead causado pelo sistema operacional e o número de tarefas [Kohout et al., 2003]. Na área de sistemas de tempo real, caso exista um sistema com muitas tarefas, é necessário executar medidas de latência para verificar se o escalonador não está desabilitando interrupções por tempos longos e inaceitáveis [Laplante, 2004].

Os *kernels* também podem variar com relação a sua arquitetura interna. A maioria dos *kernels* são classificados como monolíticos. Um kernel monolítico executa todas suas tarefas em *kernel space*, incluindo escalonamento, gerenciamento de memória, gerenciamento de sistema de arquivos em discos, controladores de dispositivos (*device*

drivers) e serviços de sincronização e comunicação entre tarefas e processos (IPC - *Inter Process Communication*).

Já a filosofia que o *microkernel* implementa consiste em executar unicamente as partes essenciais do *kernel* em modo kernel, que normalmente consistem do escalonamento de processos, gerenciamento de memória e IPC. Todo resto do sistema operacional, incluindo até mesmo *device drivers* é implementado como tarefas em modo usuário. A grande vantagem deste modelo é que o sistema operacional torna-se muito robusto. Até mesmo um hardware defeituoso não seria capaz de causar danos ao sistema em execução, já que seu código está executando em modo protegido. A desvantagem é que como o sistema operacional está divido em espaço de kernel e espaço de usuário, existe a necessidade de constantes trocas de contexto, aumentando a sobrecarga do sistema operacional. O sistema operacional mais conhecido que implementa um microkernel é um sistema operacional de tempo real da QNX chamado de Neutrino.

3 SISTEMAS OPERACIONAIS DE TEMPO REAL (RTOS)

Um kernel de tempo real, também chamado de sistema operacional de tempo real (RTOS) permite que aplicações de tempo real sejam projetadas, mantidas, expandidas e alteradas facilmente e com segurança. A adição de tarefas de baixa prioridade ao sistema, não causa influência nenhuma nas outras tarefas de prioridade mais alta já existentes [Labrosse, 2002].

Embora seja possível implementar um sistema de tempo real sem o uso de um sistema operacional, esta tarefa não é fácil e nem aconselhável [Barabanov, 1997]. Além disso, a execução de vários programas de forma simultânea pode causar tempos de resposta inesperados, pois os programas podem interagir de formas não esperadas [Wolf, 2007b].

A escolha de um RTOS (*Real Time Operating System*) é importante para dar suporte a prioridades, interrupções, timers, comunicação entre tarefas, sincronização, gerenciamento de memória e multiprocessamento [Baskiyar e Meghanathan, 2005].

De acordo com Labrosse, existem mais de 150 fornecedores de RTOS. Os pacotes de desenvolvimento podem variar de US$70,00 a US$30.000,00 ou mais, e em alguns casos, é preciso pagar royalties ao fabricante do RTOS para cada venda realizada do produto final [Labrosse, 2002]. Os sistemas operacionais de tempo real podem ser

desenvolvidos do zero, a partir de um projeto específico e formal para o seu desenvolvimento, ou através de uma camada de adaptação (normalmente de baixo nível) entre um sistema operacional existente e o hardware. Normalmente os sistemas desenvolvidos desde o princípio para ser um RTOS seguem um processo de desenvolvimento bastante formal para que o sistema possa ser facilmente validado e certificado por normas e padrões internacionais de segurança. A seguir pode ser vista uma lista de alguns sistemas operacionais de tempo real desenvolvidos desta forma:

• Integrity5 da Green Hills Software que é utilizado em aviões como o A380, o F35 e o Euroghter;

• DEOS (*Digital Engine Operating System*) da Honeywell, que é utilizado pelo Boeing 777 [Krodel e Romanski, 2007] e nos sistemas de controle de vôo de jatos regionais da Embraer [Adams, 2005];

• LynxOS 6 da LynxWorks que é tradicionalmente usado em aplicações militares e aeroespaciais. Também é usado para implementar parte do sistema de controle de tráfego aéreo dos EUA;

• VxWorks7 da Wind River Systems que é tradicionalmente utilizado pela agência espacial dos EUA, a NASA, para implementar robôs de exploração espacial.

A outra maneira de implementar-se um RTOS é adaptando um sistema operacional atual para responder a requisitos de tempo real com confiabilidade. Uma proposta para realizar esta adaptação chama-se ADEOS. O ADEOS (*Adaptative Domain Environment for Operating Systems*) permite dar previsibilidade ao tratamento de interrupções no sistema [Gerum, 2005]. O ADEOS implementa um sistema chamado Optimistic Interrupt Protection [Gerum, 2005] que é descrito em [Stodolsky et al., 1993].

A arquitetura geral de um sistema baseado em ADEOS pode ser vista na Figura 3. Como pode ser observado pela Figura, uma das grandes vantagens do ADEOS, é proporcionar o melhor dos dois mundos. Um mesmo computador pode ser utilizado tanto para tarefas de tempo real quanto de interface com o usuário.

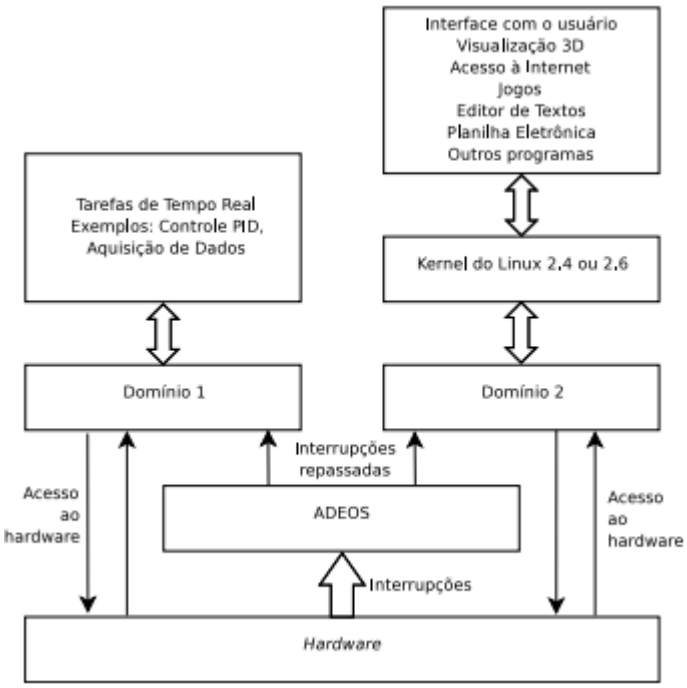

Figura 3: Arquitetura de um sistema ADEOS

O ADEOS foi proposto inicialmente em 2002 [Yaghmour, 2002] e sua arquitetura, em teoria, permite que vários sistemas operacionais sejam executados simultaneamente em um mesmo computador. Com o ADEOS o sistema é dividido em domínios, criando uma la de interrupções (*interrupt pipeline* - *ipipe*) que é distribuída para estes domínios. O ADEOS é responsável por interceptar todas as interrupções do hardware, e roteá-las para um domínio específico, como pode ser visto na Figura 3. O RTAI, Xenomai e RTLinux, três versões de Linux de Tempo real, implementam uma arquitetura baseada em ADEOS, de forma que o ambiente de tempo real é o domínio de maior prioridade, dando uma resposta determinística para as tarefas de tempo real [Barbalace et al., 2008].

Em outras palavras, o que o ADEOS faz é isolar tarefas de tempo real e convencionais, permitindo que o Linux seja executado normalmente, permitindo o uso de vários aplicativos gráficos, rede, dentre outros utilitários. Como o Linux todo, incluindo seu kernel, consiste de um dos domínios do ADEOS de baixa prioridade (ver Figura 3), quando uma tarefa de tempo real precisa ser executada, o Linux todo é preemptado. Como esta operação é bastante rápida, o usuário não percebe a diferença. Entretanto, caso a tarefa de tempo real esteja sendo executada com uma frequência muito grande, o usuário pode ter a impressão que o computador todo travou, já que o

mouse, teclado e tela podem parar de responder, mantendo as tarefas de tempo real em operação. Também é comum dizer que esta camada de adaptação é um *nanokernel*, já que ela é responsável por decidir quando um domínio recebe as interrupções ou não.

Outra abordagem que foi comum durante anos e ainda permanece nos dias atuais é utilizar o Windows NT para controlar sistemas de tempo real. O *time quantum* do Windows NT é de 20ms, que pode ser demasiadamente longo para algumas situações [Dedicated Systems, 2001h]. A análise realizada pela Dedicated Systems mostrou que o tratamento de interrupções no Windows NT é bastante rápido, tendo um valor médio de 3,7µs com máximo de cerca de 22µs. Com o sistema sobrecarregado, foram obtidos valores superiores a 600µs [Dedicated Systems, 2001h]. A conclusão é que o Windows NT é um excelente sistema operacional de propósito geral, porém inadequado para tarefas de tempo real [Martin Timmerman, 2000a].

Com isto, surge a necessidade de extensões capazes de dar uma capacidade realista de tempo real para o Windows NT [Martin Timmerman, 1998] e seus sucessores (Windows 2000, XP e Vista). O Hyperkernel implementa este recurso usando o mesmo conceito de kernel de separação [Martin Timmerman, 2000b], como utilizado no ADEOS pelo RTAI e Xenomai, entretanto possui problemas de determinismo. Mesmo assim uma latência máxima de $19\mu s$ foi observada em testes realizados com o Hyperkernel em um Pentium MMX 200 MHz [Dedicated Systems, 2001c]. Outra extensão para o Windows NT, o INTime, utiliza o mesmo princípio [Martin Timmerman, 2000c], adicionando proteção de memória entre os domínios, conferindo maior conabilidade e estabilidade ao sistema. A latência máxima obtida no mesmo Pentium foi de cerca de $24\mu s$ [Dedicated Systems, 2001d]. Outro produto comercial, o RTX, utiliza a abordagem de implementar suas características de tempo real como um driver do Windows NT. Isto oferece um bom determinismo, mas se um driver de qualquer dispositivo do Windows apresentar problemas, isto irá interferir no sistema de tempo real [Martin Timmerman, 2000f].

O relatório de testes feito pela Dedicated Systems no mesmo Pentium 200 obteve um tempo máximo para tratamento de interrupções de cerca de $15\mu s$ [Dedicated Systems, 2001f]. Em uma comparação entre as três extensões, os especialistas da Dedicated Systems acabaram por concluir que o INTime é a melhor opção, pois apesar de ser mais lento que seus concorrentes, ele oferece maior segurança e proteção [Dedicated Systems, 2001a].

Como vem sendo discutido, as interrupções e consequentemente suas latências, desempenham um papel importante em vários aspectos dos sistemas operacionais e dos sistemas de tempo real. Em um RTOS, o tempo de resposta de interrupção a ser considerado deve ser sempre o de pior caso. Se um sistema responde uma a interrupção em $50\mu s$ 99% das vezes, e responde em $250\mu s$ em 1% das vezes, o tempo de resposta deve ser como considerado $250\mu s$ [Labrosse, 2002].

A latência para tratamento das interrupções pode ser bastante inuenciada pelo sistema operacional. Em geral, pode-se calcular a latência de uma interrupção como: L = [tempo máximo que as interrupções ficam desligadas] + [tempo para iniciar a execução da primeira instrução de uma ISR] A resposta de interrupção é definida como o tempo entre a ocorrência de uma interrupção e o início da execução do código do usuário que trata a interrupção.

No caso de sistemas preemptivos, ela é definida como

R = L + [Tempo para salvar o contexto da CPU] + [Tempo de execução da rotina de entrada em ISR do Kernel].

Outra medida importante de um kernel, é o tempo de recuperação de interrupções, que consiste no tempo que o sistema leva para voltar a sua operação normal após uma interrupção ter sido tratada. No caso de sistemas preemptivos, pode-se calcular este tempo como:

TR = [Tempo para determinar se existe alguma tarefa de maior prioridade pronta] + [Tempo para restaurar o contexto da CPU] + [Tempo para executar a instrução de retorno de interrupção].

Em todas estas situações, o sistema operacional inuencia pelo tempo que mantém as interrupções desligadas, e pelo tempo que leva para tomar suas decisões de escalonamento.

4 SERVIÇOS DE UM RTOS

Em teoria, tanto os sistemas operacionais de tempo real (RTOS) quanto os sistemas operacionais de propósito geral (GPOS), como Linux, MacOS e Windows

possuem mecanismos internos e funcionamento bastante semelhantes. A grande diferença presente nos RTOS são técnicas e otimização utilizadas para favorecer o determinismo e previsibilidade do sistema. Isto é feito utilizando-se algoritmos especiais de escalonamento. De fato, a maioria dos sistemas operacionais não implementam políticas de qualidade de serviço (QoS), dando maior prioridade para tarefas mais importantes e dividindo a CPU de forma mais justa. Os algoritmos de escalonamento usados normalmente em sistemas de tempo real como por exemplo, o *Rate Monotonic* (RM) e o *Earliest Deadline* First (EDF) implementam sistemas de escalonamento que podem garantir uma maior qualidade de serviço para os programas de tempo real. [Ingram, 1999]

5. LINUX REAL TIME

5.1 VERSÃO PARA DISCO FLASH USB

A versão USB tem a utilização prática, já que é possível salvar as alterações de trabalho diretamente na memória USB. O único requisito do disco USB, é ter 400MB de espaço livre. A instalação do MECRTL não apaga os dados já existentes na unidade de disco, sendo que está poderá continuar sendo usada normalmente como já era utilizada antes, entretanto com a nova funcionalidade de poder iniciar um PC com o Linux de tempo real. De qualquer forma, é sempre recomendável fazer um backup dos arquivos importantes antes de realizar esta operação.

Para preparar uma versão USB, faça o download do arquivo mecrtl.zip, e execute os seguintes passos:

Para usuários do sistema operacional Windows:

1. Descompacte o arquivo .zip diretamente na memória Flash USB
2. Várias pastas devem ser criadas no pen drive, como boot, base, modules.
3. Acesse a pasta boot pelo Windows Explorer
4. Clique duas vezes no arquivo de lote bootinst.bat
5. Siga as instruções na tela, e o disco estará pronto para iniciar o Real Time Linux

Para usuários do sistema operacional Linux:

1. Monte a unidade USB em um ponto de montagem (Exemplo: mount /dev/sda1 /mnt/memory)

2. Entre no diretório do ponto de montagem

3. Descompacte o arquivo .zip diretamente no ponto de montagem da memória Flash USB (unzip~/Desktop/mecrtl.zip)

4. Em seguida entre no diretório boot (cd boot)

5. Execute o script bootinst.sh (./bootinst.sh)

6. Siga as instruções na tela, e o disco estará pronto para iniciar o Real Time Linux

Configure a BIOS do computador para dar *boot* via USB, ou pressione as teclas F8 ou ESC durante a inicialização do sistema para escolher o método de boot. (Estas teclas podem variar de acordo com o fabricante do computador).

Quando o boot for realizado pela unidade USB, uma tela com várias opções aparecerá. As duas primeiras opções permitem iniciar o sistema diretamente na interface gráfica baseada no ambiente KDE, sendo que a primeira tenta otimizar as configurações de vídeo, e a segunda usa configurações mais conservadoras para garantir que o sistema seja executado em computadores com recursos de vídeo modestos.

5.2 UTILIZAÇÃO BÁSICA

Na versão USB é possível acessar o sistema em modo texto através do
Usuário: root
Senha: toor.

Um terminal virtual oferece um *prompt* de comando onde é possível digitar comandos e obter as respostas destes comandos, ou mesmo executar programas. Caso esteja em modo gráfico, tanto pela inicialização automática, quanto pelo comando *startx*, você pode abrir terminais virtuais clicando no ícone de um terminal chamado Konsole, disponível no canto inferior esquerdo do ambiente gráfico.

 Utilize este ícone na barra de acesso rápido na parte inferior da tela para abrir o Konsole.

Todos arquivos criados e modificados durante a execução do PENDRIVE são perdidos quando o computador é desligado ou reiniciado, desta forma é importante salvar os arquivos em um local seguro, como uma unidade Flash USB. No Linux não existem drives A, B ou C, como no Windows. No linux existem pontos de montagem, que são diretórios convencionais. Quando um dispositivo é montado em um diretório, tudo que é lido e escrito dentro deste diretório, é mapeado para o dispositivo físico, como um disco rígido externo ou uma memória flash USB.

Utilize o comando df -h para verificar onde sua unidade flash está montada, e seu espaço livre. Na maioria dos casos as unidades flash são apresentadas como /dev/sda1 ou /dev/sdb1 e montadas no diretório /mnt/sda1_removable/. Utilize este diretório (pasta) para acessar e gravar seus arquivos pessoais.

5.3 HELLO WORLD EM ESPAÇO DE USUÁRIO

O primeiro programa não deve ser nenhuma novidade para você. O hello.c pode ser visto na listagem 1, e consiste de um programa escrito em linguagem C que inclui a biblioteca padrão de entrada e saída (stdio), e declara uma função main(). A função main é a primeira função a ser executada quando o programa é executado. Dentro do main, apenas uma função da biblioteca stdio é usada: o prinft, que imprime o texto “Hello World!” na tela. A sequência “\r\n” é legada de antigos dispositivos chamados de teletipos, onde o “\r” faz com que o cursor volte para o inicio da linha (*carriage return*), e o “\n” vai para a próxima linha (*new line*).

Listagem 1 hello.c: Hello World em espaço de usuário

```
#include <stdio.h>
        int main() {
        printf("Hello World!\r\n");
}
```

A listagem do programa hello.c consiste somente do código fonte do programa. Antes de executar este programa, precisamos compilar e executar o linker nele. A compilação consiste em traduzir o código fonte de alto nível em linguagem que nós somos capazes de entender em um arquivo binário com instruções de máquina que o computador é capaz de entender como um programa. Após a compilação, um produto intermediário chamado de objeto é gerado. Este objeto possui as instruções de máquina para executar o programa escrito no hello.c, mas ele ainda precisa da biblioteca stdio que fornece a função printf utilizada. O linker é responsável por fazer esta conexão.

Normalmente, as bibliotecas são linkadas dinamicamente. Isto quer dizer que a biblioteca esta disponível em um arquivo no sistema operacional, e toda vez que o programa é executado o sistema busca a função necessária (neste caso o prinf) neste arquivo externo que é a biblioteca. Nos compiladores gratuitos da *Free Software Foundation*, o gcc (*Gnu Compilers Collection*), tudo isto é feito com um único comando.

A listagem 2 mostra a compilação e execução do programa hello. Ao executar o gcc, o programa executável gerado sempre chama-se a.out, e para executa-lo é preciso digitar ./a.out no terminal. Caso se queria gerar um executável com outro nome, basta usar a opção “-o”, como pode ser visto na listagem de comandos 2. Em alguns casos, é preciso avisar o compilador explicitamente que se deseja utilizar uma determinada biblioteca, como no caso da biblioteca math, declarada nos arquivos de cabeçalho math.h. Para utiliza-la, é preciso executar o gcc com o parâmetro “-lm”.

Listagem 2 Compilando e executando o hello.c

```
bash-3.1$ gcc hello.c
bash-3.1$ ./a.out
Hello World!
bash-3.1$ gcc hello.c -o hello
bash-3.1$ ./helloHello World!
```

Embora a maioria dos programas sejam compilados e linkados dinamicamente às bibliotecas, existem ocasiões, onde é desejável ou essencial se realizar uma compilação estática. Na compilação estática, todas funções externas de bibliotecas são copiadas para dentro do executado gerado, criando um arquivo que pode ser executado independente de bibliotecas externas de forma auto suficiente. Em contrapartida, o arquivo compilado estaticamente possui um tamanho bem maior que aquele linkado dinamicamente. Uma comparação dos tamanhos em bytes pode ser vista no resultado do comando ls, que lista arquivos no linux, disponível na listagem 3.

Listagem 3 Comparação do tamanho em bytes do arquivo fonte hello.c, e dos executaveis compiladosdinamicamente e estaticamente

```
bash-3.1$ ls -lah hello*
-rwxr-xr-x 1 root users 7.9K 2009-01-01 19:44 hello
-rwxr-xr-x 1 root users 497K 2009-01-01 22:01 hello-static
-rw-r-r- 1 root users 64 2009-01-01 19:43 hello.c
```

Para compilar um programa estaticamente, basta utilizar a opção -static do gcc. Um exemplo de compilação e execução do programa hello estaticamente, pode ser visto na listagem 4.

Listagem 4 Compilação estática do programa hello.c

```
bash-3.1$ gcc -static hello.c -o hello-static
bash-3.1$ ./hello-static
Hello World!
```

Um outro comando bastante útil é o ldd, que imprime as dependências de bibliotecas compartilhadas para um programa especifico. Este comando é bastante útil para diagnosticar falhas em um programa, u comportamentos estranhos quando bibliotecas são trocadas ou atualizadas. A listagem 5 mostra o resultado da execução do comando ldd para os programas compilados dinamicamente e estaticamente.

Listagem 5 Lista das bibliotecas compartilhadas usadas pelo executável do programa hello

```
bash-3.1$ ldd ./hello-static
not a dynamic executable

bash-3.1$ ldd ./hello
/lib/libsafe.so.2 (0xb7e000)
linux-gate.so.1 => (0xe000)
libc.so.6 => /lib/tls/libc.so.6 (0xb7dad000)
libdl.so.2 => /lib/tls/libdl.so.2 (0xb7da9000)
/lib/ld-linux.so.2 (0xb7f05000)
```

Até o momento, o programa Hello World foi usado para apresentar alguns conceitos importantes sobre a forma de se programar no Linux. Entretanto, existe uma ferramenta chamada KDevelop que consiste em um ambiente de programação integrado, trazendo muitas facilidades ao processo de desenvolvimento. Dessa forma, a programação e execução do Hello World será descrita agora no ambiente KDevelop. O KDevelop somente funciona no ambiente gráfico de janelas, portanto este deve estar aberto. Para abrir o KDevelop, clique no K no canto inferior esquerdo do ambiente de janelas, e escolha Development, em seguida, KDevelop e finalmente KDevelop (*Multilanguage*). Também é possível abrir o KDevelop diretamente pelo atalho na barra de acesso rápido.

 Utilize este ícone na barra de acesso rápido para abrir o KDevelop.

Para criar o programa Hello World, siga os seguintes passos:

1. No KDevelop, acesse o menu Project →New Project
2. Na lista de opções, escolha C e em seguida Simple Hello World Program
3. Em *Application Name*, digite hello, e clique em Next
4. Em *Location* (local onde será salvo), procure utilizar uma unidade flash USB (/mnt/sda1_removable) para que ao desligar o computador, seu programa não seja perdido
5. Em Author, digite seu nome, e clique em *Next*
6. Em *Version Control System*, mantenha a opção *None*, e clique em *Next*
7. Clique mais uma vez em Next, e finalmente em *Finish*

 (*Build*) Utilize este botão na barra de botões do KDevelop para compilar o programa.

 (*Run*) Utilize este botão na barra de botões do KDevelop para executar o programa.

Se preferir, você pode usar somente o segundo botão, que executa o programa, já que ele também compila o programa automaticamente antes de executá-lo. Ao compilar ou tentar executar o programa pela primeira vez, uma mensagem será exibida perguntando sobre a execução dos scripts *automake* e *autoheader*, que preparam automaticamente o ambiente de configuração. Permita a execução destes clicando em *Run Them*. Isto só precisa ser feito uma vez. A partir deste momento, basta alterar o

programa e clicar no botão Run para executar o programa. A Figura 1 mostra o KDevelop com o programa hello aberto.

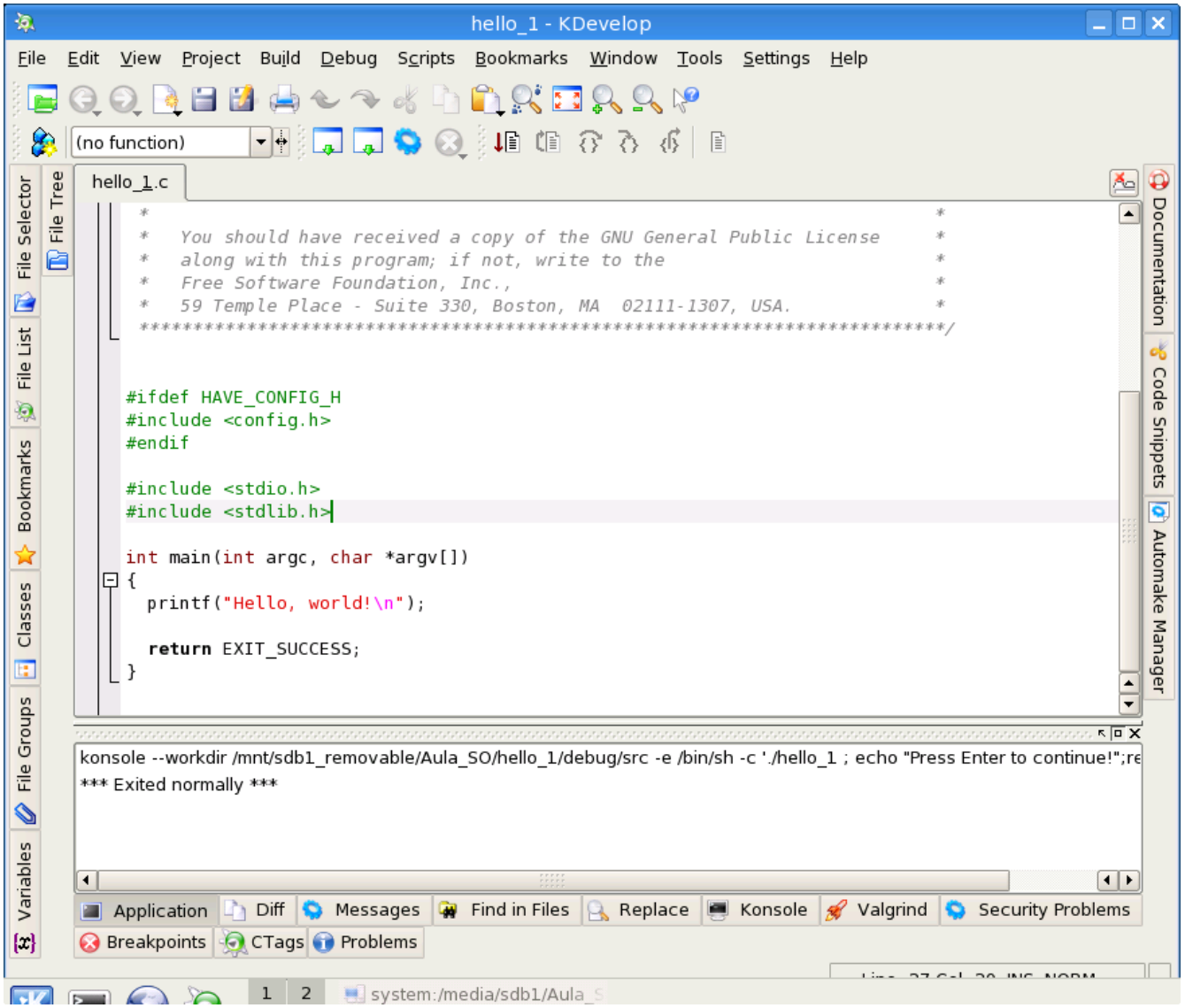

Figura 1: Programa hello no KDevelop

Outra funcionalidade interessante que pode ser vista na Figura 1, é uma guia Konsole dentro do KDevelop, em sua parte inferior. Esta guia permite executar o Konsole dentro do KDevelop, para algum teste ou inspeção rápida. Na mesma barra de guias, também está disponível a guia *Application*, que mostra informações sobre a execução do programa sendo desenvolvido.

5.4 HELLO WORLD EM ESPAÇO DE KERNEL

Até o momento, um programa em espaço de usuário foi apresentado. Este tipo de programa roda em um ambiente protegido pelo sistema operacional, e qualquer operação ilegal ou abusiva do programa, é tratada pelo sistema operacional (kernel), para evitar causar problemas para o resto do sistema. A partir de agora, um programa em espaço de kernel será apresentado. Os programas em espaço de kernel normalmente são os device drivers, e outros subsistemas básicos para manter o sistema rodando, como o escalonador, por exemplo, que é o software responsável por fazer o computador executar várias tarefas ao mesmo tempo. Para executar o Hello World em espaço de kernel, utilizaremos diretamente o KDevelop. Siga os passos a seguir para criar o programa:

1. No KDevelop, acesse o menu Project →New Project
2. Escolha a opção C e em seguida Linux Kernel Module
3. Em Application Name, utilize khello
4. Em Location (local onde será salvo), procure utilizar uma unidade flash USB (/mnt/sda1_removable) para que ao desligar o computador, seu programa não seja perdido e em seguida clique em Next
5. Em Author, coloque seu nome, e em seguida clique em Next
6. Em Version Control System, mantenha a opção none e clique em Next
7. Clique em Next novamente, e nalmente em Finish
8. O KDevelop irá perguntar se arquivos de documentação devem ser criados para o projeto. Permita esta operação clicando na opção Populate.
9. Da mesma forma que anteriormente, é possível compilar este programa (módulo) utilizando o botão (Build) do KDevelop

O programa criado automaticamente pelo KDevelop será semelhante ao listado na listagem 6. Repare que este programa não possui uma função main(), como a maioria dos programas escritos em C. Como foi comentado anteriormente, este programa trata-se de um módulo de kernel que é adicionado a algo que já está em execução, neste caso, o próprio kernel do Linux.

Ao invés da função main(), todo módulo de kernel para Linux (LKM) deve implementar as funções init_module() e cleanup_module() que são executadas quando o módulo é carregado e descarregado do kernel em execução. Os includes deste programa

também são um pouco diferentes, já que em espaço de kernel as bibliotecas convencionais como STDIO e STDLIB não estão disponíveis.

Os includes utilizados apenas avisam o sistema que o programa sendo escrito trata-se de um LKM. Finalmente, as macros MODULE_LICENSE, MODULE_AUTHOR e MODULE_DESCRIPTION fornecem informações textuais sobre o módulo para futuras vericações. Após compilar o LKM com o botão build, você pode utilizar o Konsole disponível dentro do KDevelop com o comando ls para vericar que um arquivo com extensão .ko foi criado. Ao tentar clicar no botão Run para executar este LKM, nada irá ocorrer, já que um módulo de kernel não é um programa convencional. Na verdade, o arquivo gerado nem mesmo é executável. Ele consiste de um arquivo binário, onde a extensão ko é uma abreviação para kernel object. Para colocar o módulo em execução, será necessário carregar este módulo no kernel com o comando:

Listagem 6 khello.c: Hello World em espaço de kernel

```
#include <linux/module.h>
#include <linux/kernel.h>
int init_module(void) {
        printk(KERN_INFO "hello: Hello World\n");
        return 0;
 }

void cleanup_module(void) {
        printk(KERN_INFO "hello: Bye\n");
}
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Module author's name"); MODULE_DESCRIPTION("First test module - Hello World");
```

```
insmod khello.ko

ou,  caso do projeto criado automaticamente pelo KDevelop
insmod khello-driver.ko
```

A partir deste momento, este módulo foi carregado no kernel. Como podemos ver em seu código fonte, ao ser carregado, ele vai utilizar a função printk para imprimir uma mensagem. O printk é uma função interna do kernel do Linux, que imprime mensagens informativas e de depuração para uma memória especial que armazena mensagens do kernel. Para ver o conteúdo destas mensagens, utilize o comando no Konsole:

dmesg

Verifique que na última linha existe uma informação entre colchetes, que trata-se de um valor em segundos relativo ao momento em que a mensagem foi gerada, e na frente deste número, a mensagem produzida pelo printk. O Linux possui um kernel modular, que pode receber a qualquer momento novos módulos, ou ter módulos removidos de execução. Para verificar os módulos em execução, utilize o comando:

lsmod

O lsmod irá listar todos módulos carregados no kernel. Caso a lista seja muito grande, você pode filtrar por um padrão desejado, como por exemplo, “hello” através do comando grep usado em conjunto com o comando lsmod:

lsmod | grep hello

Repare que o módulo está carregado. O número exibido depois do nome do módulo, é quantia em bytes do programa que está sendo usada por este módulo, e o número a seguir (um zero (0)) é o número de dependências deste módulo. Este assunto voltará a ser tratado posteriormente. A quantia de *bytes* ocupadas por um LKM é um parâmetro importante, já que um módulo de kernel ocupa uma memória permanente e só deixa de ocupar esta memória quando o módulo é removido. Para parar a execução do módulo, deve-se utilizar o comando:

rmmod khello-driver

Após remover o módulo, repare que ele não é mais listado ao executar o comando lsmod novamente, e que a mensagem relativa a remoção do módulo (cleanup_module) também foi impressa na memória de mensagens do kernel (dmesg). Com relação ao processo de compilação deste módulo, verifique na opção File Tree do KDevelop que um dos arquivos criados automaticamente é o Makefile. O arquivo Makefile possui regras de compilação de um programa, pois conforme os programas começam car complicados, é necessário criar scripts mais sofisticados com regras de compilação.

O Makefile pode ser executado com o comando make, que na verdade é que o KDevelop faz quando se clica no botão Build. No caso especifico de um módulo do kernel, é essencial usar um Makefile, que chama outro Makefile de dentro do próprio kernel do Linux. Dessa forma, para compilar um módulo de kernel para o Linux, é preciso ter o código fonte do kernel disponível. No caso do MECRTL, o kernel está disponível em /usr/src/linux.

A listagem 7 mostra o conteúdo essencial que um Makefile deve possuir para compilar um módulo de kernel (neste caso o khello).

Listagem 7 Makele para o programa khello.c

```
obj-m += khello.o
all:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

5.5 COMANDOS ÚTEIS

Esta seção oferece alguns comandos úteis para aproveitar melhor os assuntos discutidos neste capítulo. Adicionalmente, maiores informações destes e outros comandos podem ser obtidas executando os comandos com o parâmetro -help, -h ou executando o sistema de manuais do Unix (manpages) para cada comando, como por exemplo “man ldd”, ou “man ls”

5.5.1 LDD

Mostra as bibliotecas compartilhadas que um programa necessita para ser executado.

5.5.2 INSMOD [NOME DO ARQUIVO.KO]

Carrega um módulo de kernel com extensão .ko (kernel object) no kernel em execução

5.5.3 RMMOD [NOME DO ARQUIVO.KO]

Remove um módulo de kernel em execução do kernel, interrompendo sua atividade.

5.5.4 MODINFO [NOME DO ARQUIVO.KO]

Mostra informações sobre um módulo de kernel, como descrição, nome de autor, e parâmetros aceitos.

5.5.5 LSMOD

Lista os módulos de kernel carregados e em execução.

5.5.6 DMESG

Mostra o buffer de mensagens do kernel. As mensagens produzidas com a função printk são gravadas neste buffer. Esta lista de mensagens é circular e tem um tamanho máximo. Quando o máximo é atingido, as mensagens antigas são sobrescritas, mantendo as mais novas.

5.5.7 LE [NOME DO ARQUIVO]

Mostra informações sobre um determinado arquivo, informando se ele consiste de um arquivo texto, um programa executável linkado dinamicamente ou estaticamente, em que processador pode ser executado, e outras informações.

5.5.8 LS

Lista os arquivos do diretório (pasta) atual. A opção -lh mostra o tamanho em bytes dos arquivos. Também é possível executar o ls, usando como parâmetro o nome do arquivo sobre o qual se deseja verificar as informações.

5.5.9 CAT [NOME DO ARQUIVO]

Mostra o conteúdo do arquivo especificado nos parâmetros. Ele pode ser executado com arquivos comuns, ou arquivos especiais, que nunca terminam. Ao tentar executar o cat no arquivo kmsg (cat /dev/kmsg), as mensagens do kernel passarão a ser exibidas continuamente conforme forem geradas.
Para interromper a execução, utilize o comando CTRL+C.

5.5.10 UNAME

Mostra qual é o tipo de Unix em execução. A opçao -a mostra todas informações sobre o sistema,
enquanto a opção -r mostra a versão do kernel em execução.

5.5.11 MAKE

Executa as regras de compilação ou execução no arquivo makefile ou Makefile.

5.5.12 GREP

O grep é um ltro que é bastante útil para ltrar a saída de programas. Por exemplo, na execução do comando cat /dev/kmsg, ou dmesg, várias informações não desejadas podem ser exibidas. Ao usar o grep, apenas as informações especicadas em um ltro são mostradas, como por exemplo:

```
cat /dev/kmsg | grep hello
```

O exemplo acima vai mostrar apenas as mensagens que surgirem no kernel contendo a palavra hello.
Ou

```
dmesg |grep usb
```

só vai mostrar as linhas do dmesg que possuam a palavra usb.

O grep também pode ser usado com expressões regulares, que permitem definir uma quantia enorme de padrões de busca e filtro.

5.5.13 PIPES

Pipes são uma maneira útil e prática de coletar dados e fazer programas se comunicarem no ambiente Unix. Existem diversos tipos de pipes, sendo os mais importantes:

- Redirecionamento de saída para arquivo (>): Este pipe redireciona a saida de um programa para um arquivo. Sua execução consiste em programa > arquivo, como por exemplo dmesg > dmesg.txt que vai gravar um arquivo dmesg.txt como conteúdo do arquivo. Caso o arquivo já exista, o arquivo é substituído e os dados anteriores são perdidos. O pipe lê os dados da saída padrão do programa (STDOUT).
- Redirecionamento de saída para arquivo com concatenação (> >): Este pipe tem o funcionamento igual ao do pipe anterior, entretanto, caso o arquivo de destino já exista, ele não é substituído, mas mantido, e os dados novos são acrescentados ao fim do arquivo.
- Redirecionamento de entrada (<): Este pipe permite que um programa receba em sua entrada padrão (STDIN), que normalmente é o teclado, a partir de um arquivo. Um exemplo seria programa < dados-entrada.txt.
- Redirecionamento da saída de um programa para a entrada de outro (|): Este pipe permite que a saída de um programa seja enviada diretamente para a entrada de outro, permitindo uma forma prática de um programa enviar dados para outro. O exemplo já mencionado cat /dev/kmsg | grep hello, pega a saída do comando cat, que seria o conteúdo do /dev/kmsg e envia para o grep filtrar por hello.

Observação: Os pipes também podem ser usados também para a saída de erro padrão (STDERR), sendo que neste caso é necessário adicionar o símbolo & antes do símbolo de pipe.

5.5.14 startx

Inicia o ambiente gráfico com o gerenciador de janelas KDE.

5.5.15 free

Mostra a quantia de memória RAM livre e utilizada no computador. A opção buffers/cache corresponde a memória temporária que armazena modificações em arquivos antes destes serem gravados em disco, para aumentar a performance. Dessa forma, o Linux tem a tendência de rapidamente usar 100% da memória do computador, entretanto grande parte dela consiste de arquivos que estão mapeados no disco, para aumento de velocidade do sistema.

5.5.16 df

Mostra os discos atualmente montados no sistema, e seu espaço livre. A opção "-h", permite mostrar os valores de espaço livre em bytes. "-h" é uma abreviação para human readble.

5.6 RESUMO

Neste capítulo o sistema operacional baseado em Linux de tempo real foi introduzido. Apesar de nenhuma característica de tempo real ter sido discutida, algumas dicas de comandos foram oferecidas, e uma execução passo a passo foi descrita para criar e executar um programa em espaço de usuário e de kernel no linux. É muito importante realizar e entender bem os passos descritos neste capítulo, pois estes conceitos serão reutilizados ao longo do restando das práticas e exercícios propostas neste texto.

5.7 EXERCÍCIOS

1. Implemente um programa em espaço de usuário que calcule o seno e cosseno de 90 e 45 graus, e imprima o resultado na tela.
2. Implemente um programa em espaço de kernel que conte de 0 a 100.
3. Salve os dois programas criados nos itens 1 e 2, reinicie o computador e tente compilar e executar os dois programas novamente.

4. Implemente um programa em espaço de kernel que calcule 181 valores de seno, variando de 1 em 1, iniciando em 0 graus e terminando em 180 graus.

5. Crie um programa que lê pelo teclado, com a função scanf(), uma lista de 10 valores, e calcula a média destes valores.

6. Crie um arquivo texto contendo 10 valores, e utilize pipes para executar o programa criado no passo 5 recebendo como valores de entrada os valores do arquivo.

7. Grave através de pipes a saída dos programas desenvolvidos nos itens 1, 2 e 4 em arquivos separados (um por programa), e em seguida, a saída de todos programas em um único arquivo.

6 REFERÊNCIAS

[Aarno, 2004a] Aarno, D. (2004a). Control of a puma 560 using linux real-time application interface (rtai). On-Line: http://www.nada.kth.se/ bishop/rtcontrol.pdf, Consultado em Jan/2008.

[Barabanov, 1997] Barabanov, M. (1997). A linux-based realtime operating system. Dissertação de Mestrado, New Mexico Institute of Mining and Technology. [Barbalace et al., 2008]

Barbalace, A., Luchetta, A., Manduchi, G., Moro, M., Soppelsa, A., e Taliercio, C. (2008). Performance comparison of vxworks, linux, rtai, and xenomai in a hard real-time application. Nuclear Science, IEEE Transactions on, 55(1):435439. [Baskiyar e Meghanathan, 2005]

[Beal, 2005] Beal, D. (2005). Linux R as a real-time operating system. Relatório técnico, freescale semiconductor.

[Calandrino e Anderson, 2006] Calandrino, J. e Anderson, J. (2006). Quantum support for multiprocessor pfair scheduling in linux. In Proceedings of the Second International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, p. 3641.

[Cawleld, 1997] Cawleld, D. W. (1997). Achieving fault-tolerance with pc-based control. Relatório técnico, OMNX Open Control, Olin Corporation, OMNX Open Control, Olin Corporation Charleston, TN 37310-0248 dwcawleld@corp.olin.com.

[Centioli et al., 2004] Centioli, C., Iannone, F., Mazza, G., Panella, M., Pangione, L., Vitale, V., e Zaccarian, L. (2004). Open source real-time operating systems for plasma control at ftu. NuclearScience, IEEE Transactions on, 51(3):476481.

[Dedicated Systems, 2001a] Dedicated Systems (2001a). Comparison between hyperkernel4.3, rtx4.2 and intime1.20. Relatório técnico, Dedicated Systems.

[Dedicated Systems, 2001c] Dedicated Systems (2001c). Hyperkernel 4.3. Relatório técnico, Dedicated Systems. [Dedicated Systems, 2001d] Dedicated Systems (2001d). Intime 1.20. Relatório técnico, Dedicated Systems.

[Dedicated Systems, 2001f] Dedicated Systems (2001f). Rtx 4.2. Relatório técnico, Dedicated Systems.

[Dedicated Systems, 2001h] Dedicated Systems (2001h). Windows nt workstation 4.0. Relatório técnico, Dedicated Systems.

[Dudek et al., 2007] Dudek, G., Dudek, G., Giguere, P., Prahacs, C., Saunderson, S., Sattar, J., TorresMendez, L.-A., Jenkin, M., German, A., Hogue, A., Ripsman, A. R. A. A., Zacher, J. Z. A. J., Milios, E. M. A. E., Liu, H. L. A. H., Zhang, P. Z. A. P., Buehler, M. B. A. M., e Georgiades, C. G. A. C. (2007). Aqua: An amphibious autonomous robot. Computer, 40(1):4653.

[Farines et al., 2000] Farines, J.-M., da Silva Fraga, J., e de Oliveira, R. S. (2000). Sistemas de Tempo Real. Escola de Computação 2000, Florianópolis.

[Ganssle, 2004] Ganssle, J., editor (2004). The Firmware Handbook. Elsevier.

[Gerum, 2005] Gerum, P. (2005). Life with adeos.

[Ingram, 1999] Ingram, D. (1999). Soft real time scheduling for general purpose client-server systems. In Proceedings of the Seventh Workshop on Hot Topics in Operating Systems, number 7, p. 130135. Cambridge University. ISBN: 0-7695-0237-7.

[Kailas e Agrawala, 1997] Kailas, K. K. e Agrawala, A. K. (1997). An accurate time-management unit for real-time processors. Relatório Técnico CS-TR-3768, University of Maryland Institute for Advanced Computer Studies.

[Kohout et al., 2003] Kohout, P., Ganesh, B., e Jacob, B. (2003). Hardware support for real-time operating systems. In Hardware/Software Codesign and System Synthesis, 2003. First IEEE/ACM/IFIP International Conference on, p. 4551.

[Krodel e Romanski, 2007] Krodel, J. e Romanski, G. (2007). Real-time operating systems and component integration considerations in integrated modular avionics systems report. Relatório técnico, U.S. Department of Transportation - Federal Aviation Administration.

[L. Dozio, 2003] L. Dozio, P. M. (2003). Linux real time application interface (rtai) in low cost high performance motion control. In Motion Control 2003, a conference of ANIPLA - Associazione Nazionale Italiana per l'Automazione.

[Labrosse, 2002] Labrosse, J. (2002). MicroC/OS-II - The Real Time Kernel. CMP Books, 2 edition.

[Laplante, 2004] Laplante, P. A. (2004). Real-Time System Design and Analysis. John Wiley & Sons.

[Martin Timmerman, 2000a] Martin Timmerman, B. B. (2000a). Executive summary of the evaluation report on windows nt 4.0 workstation - executive summary. Dedicated Systems Magazine.

[Martin Timmerman, 2000b] Martin Timmerman, B. B. (2000b). Hyperkernel 4.3 evaluation – executive summary. Dedicated Systems Magazine.

[Martin Timmerman, 2000c] Martin Timmerman, B. B. (2000c). Intime 1.20 evaluation - executive summary. Dedicated Systems Magazine.

[Martin Timmerman, 2000f] Martin Timmerman, B. B. (2000f). Rtx 4.2 evaluation - executive summary. Dedicated Systems Magazine.

[Martin Timmerman, 1998] Martin Timmerman, Bart Beneden, L. H. (1998). Windows nt real-time extensions - better or worse? Real-Tima Magazine, 3.

[McKenney, 2008] McKenney, P. E. (2008). Responsive systems: An introduction. IBM Systems Journal.

[Stodolsky et al., 1993] Stodolsky, D., Chen, J. B., e Bershad, B. N. (1993). Fast interrupt priority management in operating system kernels. In In Proceedings of

the Second Usenix Workshop on Microkernels and Other Kernel Architectures, number CS-93-152, p. 105110.

[Sutter, 2002] Sutter, E. (2002). Embedded Systems Firmware Demystied. CMP Books.

[Tanenbaum, 2001] Tanenbaum, A. (2001). Modern Operating Systems. Prentice Hall.

[Theberge et al., 2006] Theberge, M., Theberge, M., e Dudek, G. (2006). Gone swimmin' [seagoing robots]. IEEE Spectrum, 43(6):3843.

[Vera et al., 2003] Vera, X., Lisper, B., e Xue, J. (2003). Data caches in multitasking hard real-time systems. In IEEE Real-Time Systems Symposium.

[Weiss et al., 1999] Weiss, K., Steckstor, T., e Rosenstiel, W. (1999). Performance analysis of a rtos by emulation of an embedded system. In Rapid System Prototyping, 1999. IEEE International Workshop on, p. 146151.

[Wolf, 2007b] Wolf, W. (2007b). The good news and the bad news. Computer, 40(11):104105.

[Yaghmour, 2002] Yaghmour, K. (2002). Adaptative domain environment for operating systems. Relatório técnico, Opersys.