RELATÓRIO DE PROJETO FINAL EM SISTEMAS OPERACIONAIS
IMPLEMENTAÇAO DE RTAI 5.0 COM UBUNTU 16.04

Ibukun C. D. Adjitche[1], **Tiago A. N. de Lima**[2]

[1] Departamento de Ciência da Computação
Universidade Federal de Roraima (UFRR) – Boa Vista, BRAZIL

adjitchedidier1@gmail.com , tiaguzaum@gmail.com

Abstract. *This project consists of making the linux kernel to provide a real-time system. That is, make a change from a general-purpose operating system to real-time systems. In addition, a kernel module will be developed using an RTAI library to test the modified kernel support. It works as follows: a brief description of the kernel kernel as well as the main operating system support divisions, description of linux kernel subsystems, description of linux kernel modules, and presentation of examples, a brief tutorial for development of kernel modules, the process of modifying the Linux kernel of the Ubuntu 16.04 LTS distribution to support real-time systems using an RTAI 5.0 application (the RealTime application interface for Linux). In addition, the kernel modified for real-time shows how the CPU frequency is working for CPU usage. As well as a description of the advantages and disadvantages of the operating system.*

Resumo. *Este projeto consiste na alteração do kernel do linux para prover suporte a sistemas de tempo real. Ou seja, efetuar a alteração de um sistema operacional de propósito geral para sistemas de tempo real. Adicionalmente, será desenvolvido um módulo para o kernel utilizando a biblioteca RTAI para testar o suporte do kernel modificado. Está organizado da seguinte forma: uma breve descrição do funcionamento do kernel do linux, bem como suas respectivas divisões de suporte ao sistema operacional, descrição dos subsistemas do kernel do linux, descrição dos módulos do kernel do linux e apresentação de exemplos, um breve tutorial para o desenvolvimento de módulos para o kernel, o processo de modificação do kernel do linux da distribuição Ubuntu 16.04 LTS para suporte a sistemas de tempo real utilizando a aplicação RTAI 5.0 (the RealTime Application Interface for Linux). Além disso depois do kernel modificado para tempo real, mostrar como alterar a frequência da CPU no sistema operacional para executar na frequência mínima disponível para a CPU. Bem como a descrição das vantagens e desvantagens do sistema operacional ser executado em frequência mínima da CPU.*

1. Introdução

O linux, criado pelo estudante finlandês Linux Torvalds, que na sua primeira versão é um clone do UNIX; apresenta muitas características e ideais que inicialmente faltavam ao MINIX. De natureza Híbrida (UNIX e MINIX), Linux cresceu rapidamente em um sistemas de produção completo, à medida que as memórias virtuais, sistemas de arquivos mais

sotisficados e muitas outras caracteristicas foram acrescentados. Assim como o UNIX, o linux foi projetado por programadores, para programadores, que atuam no ambiente no qual a maioria dos usuários são relativamente sofisticadas e engajadas em projetos de desenvolvimento de software (muitas vezes bastante complexos). Razão pelo qual , um grande número de programadores está ativamente cooperando para produzir um único sistema, de maneira que o linux e suas distribuições tem amplos recursos para permitir que as pessoas trabalhem juntas e compartilhem informações de maneiras controladas.

No intuito de conhecer como funcionam os processos de desenvolvimento de kernel de linux nas comunidades, os módulos e os sistemas de tempo real, esse trabalho venha ser subdivido em três tópicos. O primeiro tópico tratará do kernel do linux, seu processo de desenvolvimento, do seu funcionamento e das subsistemas que o compõe. O segundo tópico, definirá o que são os módulos do kernel, alguns exemplos e um tutorial de como criá-los. O terceiro tópico relatará o processo de implementação de um sistema de tempo real RTAI 5.0 no Ubuntu 16.04 e das vantagens e desvantagens ao setar a frequência do CPU na sua menor velocidade.

2. Kernel do Linux

Entender como funciona o desenvolvimento do Kernel do linux, é preciso dominar os seguintes vocabulários:

- Patch - arquivo contendo diferenças textuais entre arquivos. Utilizado para descrever diferenças em código-fonte de linguagens de programação. Este arquivo é gerado automaticamente por ferramentas de controle de versão e pelo utilitário Unix diff;
- Árvore (Source Tree) - repositório do código-fonte de um sistema de controle de versão;
- branch (ramo) - ramificação de um código fonte onde geralmente são implementadas novas funcionalidades;
- Kernel mainline - Kernel Linux “vanilla” ou principal, originado da árvore principal de desenvolvimento, que é gerenciada por Linus Torvalds;
- merge (mesclar) - é a junção de arquivos de código fonte. Esta operação é realizada por ferramentas de controle de versão como o git. Um exemplo de operação de merge é o merge entre dois branchs;
- Problemas de regressão (regressions) - bugs em funcionalidades que antes estavam corretas e são causados por um determinado evento.

2.0.1. Processo de desenvolvimento do kernel Linux atualmente

- **Processo de merge de patchs**: No começo do desenvolvimento de cada ciclo, “uma janela de merge” é dita aberta. Neste momento, código que é considerado suficientemente estável e aceito pela comunidade de desenvolvimento é incluído no kernel mainline por meio de merges.Os códigos foram previamente coletados, testados e organizados.A janela de merge dura aproximadamente duas semanas. Ao final deste período, Linus Torvalds declara a janela fechada e então disponibiliza o primeiro candidado a versão final deste kernel (RC - Release Candidate). A disponibilização do -rc1 é um sinal que o tempo para fazer o merge de novas

funcionalidades chegou ao fim, e o tempo para estabilizar a próxima versão chegou.

- **Processo de correção do mainline**: Durante as próximas 6 até 10 semanas, somente patches que corrigem problemas devem ser submetidos para o mainline. Raramente alterações significantes serão permitidas e desenvolvedores que insistirem em enviar uma alteração com a janela de merge fechada, tendem a receber uma mensagem nada amigável em resposta. Um exemplo de excessão permitida seria a integração de um driver para um hardware não suportado anteriormente e que não causa problemas de regressão em outras partes do código, isto é, o código utiliza puramente a API disponível naquele momento no kernel.

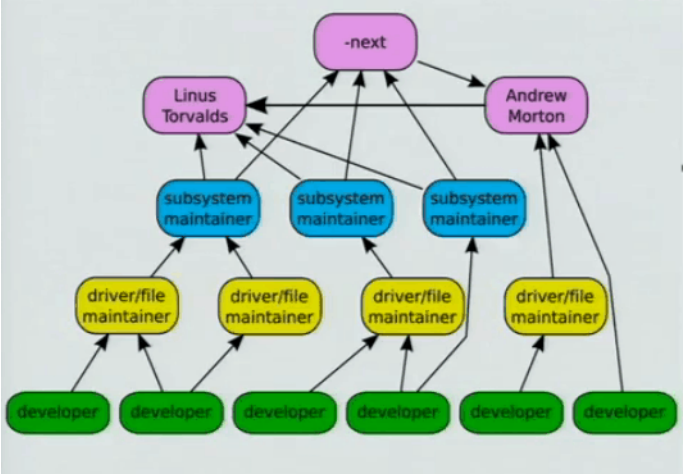


Figura 1. Como o fluxo de um patch é aceitado no MainLine

- **Processo de criação de Versões**: À medida que as correções de bug são feitas no mainline, o números de patchs enviados vai cair com o tempo. Linus lança novas -rc quase que semanalmente. Geralmente os rcs vão de rc6 até rc9 antes que o kernel seja considerado suficientemente estável e a versão final seja lançada. Neste ponto, o processo recomeça.É muito importante que os projetos dos subsistemas tenha um responsável.

- **Processo de estabilização**: A mais importante métrica é a lista de problemas conhecidos de versões anteriores (regressions from previous releases). Nenhum bug é bem-vindo, mas aqueles que causam problemas em coisas que funcionavam anteriormente são considerados especialmente sérios. Por esta razão, patches que causam problemas são prováveis candidatos a serem retirados (reverted) durante o processo de estabilização.Durante o processo de estabilização, o objetivo dos desenvolvedores é resolver o máximo número de problemas (regressions) antes que a versão estável seja lançada. No mundo real, este tipo de perfeição é difícil de ser alcançada. Existem muitas variáveis em um projeto deste tamanho. Existe um ponto onde demorar o lançamento da próxima versão, só torna o problema pior: a pilha de mudanças para a próxima janela de merge vai crescer, criando ainda mais problemas de regressão para a próxima rodada. Portanto, a maioria

dos kernels é lançada com alguns problemas de regressão conhecidos, e com esperança, nenhum é sério.

- **Processo de Manutenção**: Uma vez que o lançamento da versão estável é feita, a manutenção dela é passada para o "stable team". O time estável vai lançar ocasionalmente atualizaçãoes para a release estável. Para ser considerado uma atualização de release, um patch precisa (1) consertar um bug significante e (2) ter sido dado merge com o próximo kernel mainline. Os kernels vão tipicamente receber atualizações não mais que um ciclo de desenvolvimento a contar do seu lançamento.

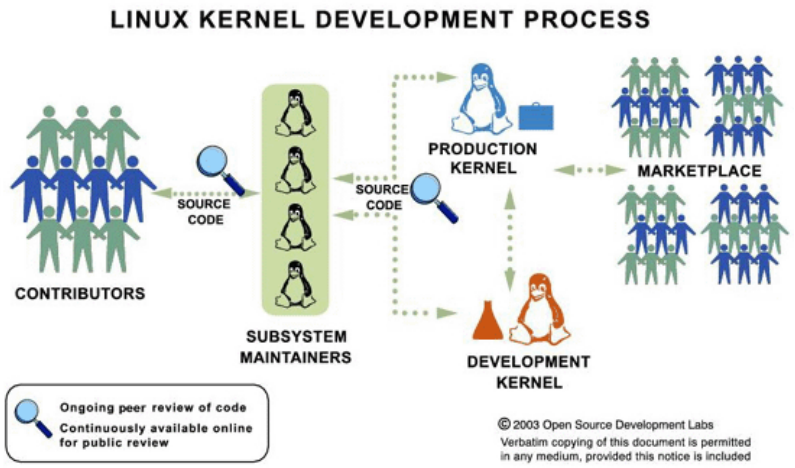


Figura 2. Processo de desenvolvimento atual do kernel Linux

2.1. Funcionamento do Kernel do Linux

Um sistema Linux pode ser considerado um tipo de pirâmide, como ilustrado na **Figura 3**. Na parte de baixo está o hardware, consistindo na CPU, memória, discos, um monitor e um teclado e outros dispositivos. A sua função é controlar o hardware e fornecer uma interface de chamada de sistema para todos os programas.Essas chamadas de sistema permitem que os programas do usuário criem e gerenciem processos, arquivos e outros recursos.

Programas fazem chamadas de sistema colocando os argumentos em registradores (ou às vezes, na pilha), e emitindo instruções de desvio para chavear do modo usuário para o modo núcleo. Dado que não há como escrever uma instrução de desvio em C, é fornecida uma biblioteca, com uma rotina por chamada de sistema. Essas rotinas são escritas em linguagem de montagem, mas podem ser chamadas a partir de C. Cada uma coloca primeiro seus argumentos no lugar apropriado, então executa a instrução de desvio. Assim, para executar a chamada de sistema **read** , um programa C pode chamar a rotina de biblioteca **read** . Como uma nota, é a interface de biblioteca, e não a interface de chamada do sistema, que está especificada pelo POSIX. Em outras palavras, POSIX nos diz quais rotinas de biblioteca um sistema em conformidade deve fornecer, quais são seus parâmetros, o que devem fazer, quais resultados devem retornar. Ele nem chega a mencionar as chamadas de sistema reais.

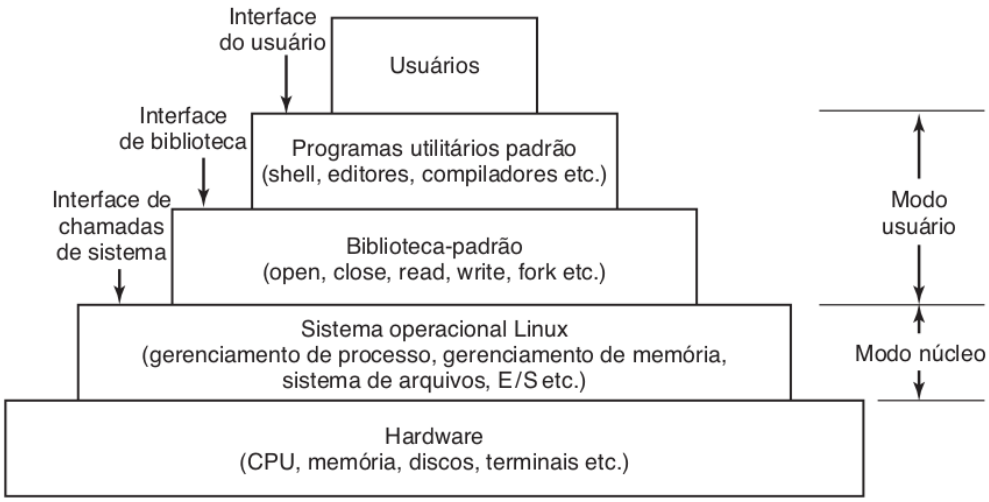


Figura 3. As camadas em um sistema Linux.

Além do sistema operacional e da biblioteca de chamadas de sistemas, todas as versões do Linux fornecem um grande número de programas padrão, alguns dos quais são especificados pelo padrão POSIX 1003.2, e alguns dos quais diferem entre as versões Linux. Entre eles estão o processador de comandos (shell), compiladores, editores, programas de edição de texto e utilitários de manipulação de arquivos. São esses programas que um usuário no teclado invoca. Então, podemos falar de três interfaces diferentes para o Linux: a verdadeira interface de chamada de sistema, a de biblioteca e a interface formada pelo conjunto de programas utilitários padrão.

2.2. Os subsistemas do Kernel do Linux

O núcleo encontra-se diretamente sobre o hardware e possibilita interações com os dispositivos de E/S e a unidade de gerenciamento de memória, e controla o acesso da CPU a eles. No nível mais baixo, como mostrado na Figura 4, ele contém tratadores de interrupção, que são a principal maneira de interagir com dispositivos, e o mecanismo de despacho de baixo nível. Esse despacho ocorre quando acontece uma interrupção. O código de baixo nível para o processo em execução guarda o seu estado nas estruturas de processo do núcleo e inicializa o driver apropriado. O despacho de processo também acontece quando o núcleo completa algumas operações, momento em que um processo de usuário é iniciado novamente.

O código de despacho está em linguagem de montagem e é bastante distinto do escalonamento. Em seguida, dividimos os vários subsistemas de núcleo em três componentes principais. O componente E/S na Figura 4 contém todos os fragmentos de núcleo responsáveis por interagir com dispositivos e realizar operações de E/S de rede e armazenamento.

No nível mais alto, as operações de E/S são todas integradas sob a camada VFS (Virtual File System — Sistema de arquivos virtual). Isto é, no nível mais alto, realizar uma operação de leitura em um arquivo, não importa se ele está na memória ou no disco, é

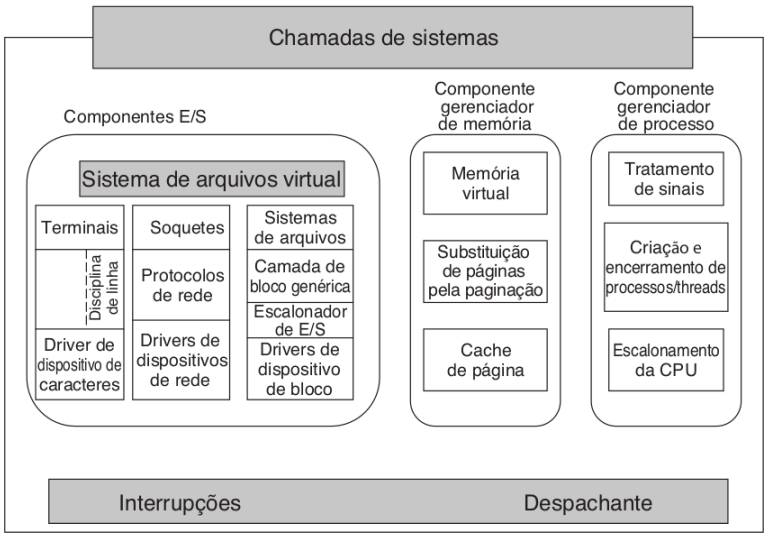


Figura 4. Estrutura do núcleo

o mesmo que realizar uma operação de leitura para recuperar um caractere de uma entrada do terminal.

No nível mais baixo, todas as operações de E/S passam por algum driver de dispositivo. Todos os drivers Linux são classificados como drivers de dispositivos de caracteres ou drivers de dispositivos de blocos, a principal diferença é que buscas e acessos aleatórios são permitidos em dispositivos de bloco, e não em dispositivos de caracteres. Tecnicamente, dispositivos de redes são de fato dispositivos de caracteres, mas eles são tratados de modo um pouco diferente, então provavelmente é melhor que fiquem separados, como foi feito na figura 4.

Acima do nível do driver do dispositivo, o código do núcleo é diferente para cada tipo de dispositivo. Dispositivos de caracteres podem ser usados de duas maneiras. Alguns programas, como em editores visuais, como vi e emacs, querem cada tecla pressionada individualmente. O terminal bruto de E/S (tty) torna isso possível. Outros softwares, como o shell, são orientados por linhas, permitindo que os usuários editem a linha inteira antes de pressionar ENTER para enviá-la para o programa. Nesse caso, a cadeia de caracteres do dispositivo terminal é passada pelo que é chamado disciplina de linha, e uma formatação apropriada é aplicada.

O software de rede é muitas vezes modular, com diferentes dispositivos e protocolos suportados. A camada acima dos drivers de rede lida com um tipo de função de roteamento, certificando-se de que o pacote certo vá para o dispositivo certo ou tratador de protocolo. A maioria dos sistemas Linux contém a funcionalidade completa de um roteador de hardware dentro do núcleo, embora o desempenho seja inferior ao de um roteador de hardware. Acima do código de roteador há a pilha de protocolo real, incluindo IP e TCP, mas também protocolos adicionais. Acima de toda a rede há uma interface de soquete, que permite que os programas criem soquetes para redes e protocolos em particular, retornando um descritor de arquivos para cada soquete usar mais tarde.

No topo dos drivers de disco está o escalonador de E/S, que é responsável por ordenar e emitir solicitações de operações de disco de uma maneira que tente conservar

movimentos de cabeça de disco desperdiçados ou para atender alguma outra política de sistema. Bem no topo da coluna de dispositivos de bloco estão os arquivos de sistemas. O Linux pode, e na realidade tem, múltiplos sistemas de arquivos coexistindo simultaneamente. A fim de esconder as terríveis diferenças arquitetônicas dos vários dispositivos de hardware da implementação do sistema de arquivos, uma camada de dispositivos de bloco genérica fornece uma abstração usada por todos os sistemas de arquivos.

À direita na Figura 4 estão os outros dois componentes-chave do núcleo Linux. Estes são responsáveis pelas tarefas de gerenciamento de memória e processo. Tarefas de gerenciamento de memória incluem manter os mapeamentos memória virtual para física, manter uma cache de páginas recentemente acessadas e implementar uma boa política de substituição de páginas, assim como sob demanda trazer novas páginas de códigos e dados necessários para a memória.A principal responsabilidade do componente de gerenciamento de processo é a criação e término de processos. Isso também inclui o escalonador de processos, que escolhe qual processo ou thread a ser executado em seguida.O núcleo do Linux trata ambos os processos e threads simplesmente como entidades executáveis, e os escalonará com base em uma política de escalonamento global. Por fim, o código para o tratamento de sinais também pertence a esse componente.Embora os três componentes sejam representados em separado na figura 4, eles são altamente interdependentes. Sistemas de arquivos costumam acessar arquivos por meio de dispositivos de bloco. No entanto, a fim de esconder as grandes latências de acessos de disco, arquivos são copiados na cache de páginas na memória principal.

Alguns arquivos podem ser até dinamicamente criados e ter apenas uma representação na memória, como os que oferecem alguma informação de uso de recursos em tempo de execução. Além disso, o sistema de memória virtual pode contar com uma partição de disco ou área de troca de arquivos para criar cópias de partes da memória principal quando ele precisa liberar determinadas páginas e, portanto, conta com o componente de E/S. Existem várias outras interdependências. Além dos componentes estáticos no núcleo, o Linux dá suporte a módulos dinamicamente carregáveis. Esses módulos podem ser usados para acrescentar ou substituir os drivers de dispositivos padrão, sistema de arquivos, rede, ou outros códigos-núcleo. Os módulos não são mostrados na Figura 4.

Por fim, bem no topo está a interface de chamadas de sistema para o núcleo. Todas as chamadas de sistema vêm até essa área, provocando um desvio que chaveia a execução do modo usuário para o modo núcleo protegido e passa o controle para um dos componentes do núcleo descritos anteriormente.

2.3. Os Modulos

Por décadas, drivers de dispositivos UNIX eram estaticamente ligados ao núcleo de maneira que eles estavam todos presentes na memória sempre que o sistema era inicializado. Dado o ambiente no qual o Linux cresceu, comumente minicomputadores departamentais e então estações de trabalho sofisticadas, com seus conjuntos pequenos e inalterados de dispositivos de E/S, esse esquema funcionava bem. Basicamente, um centro de computadores construía um núcleo contendo drivers para os dispositivos de E/S e isso era tudo. Se no ano seguinte o centro comprava um disco novo, ele ligava novamente o núcleo. Nada de especial. Com a chegada do Linux na plataforma PC, subitamente tudo isso mudou. O número de dispositivos de E/S disponíveis no PC é muito maior do que em qualquer

minicomputador. Além disso, embora os usuários do Linux tenham (ou possam conseguir facilmente) todo o código fonte, provavelmente a vasta maioria teria uma dificuldade considerável em acrescentar um driver, atualizar todas as estruturas de dados relacionadas ao driver de dispositivo, religar o núcleo e então instalá-lo como o sistema inicializável (sem mencionar lidar com o resultado de construir um núcleo que não inicializa).

O Linux solucionou esse problema com o conceito de módulos carregáveis.

2.3.1. O que são Módulos do Kernel ?

Os módulos são blocos de códigos que podem ser carregados no núcleo enquanto o sistema está executando. Mais comumente esses são drivers de dispositivos de bloco ou caracteres, mas eles também podem ser sistemas de arquivos inteiros, protocolos de rede, ferramentas de monitoramento de desempenho, ou qualquer coisa desejada.

Quando um módulo é carregado, várias coisas têm de acontecer.

- Primeiro, o módulo tem de ser realocado dinamicamente durante o carregamento.
- Segundo, o sistema precisa conferir para ver se os recursos que o driver necessita estão disponíveis (por exemplo, níveis de solicitação de interrupção) e se afirmativo, marcá-los como em uso.
- Terceiro, quaisquer vetores de interrupção que forem necessários precisam ser configurados.
- Quarto, a tabela de troca de driver apropriada tem de ser atualizada para lidar com o novo tipo de dispositivo principal.
- Por fim, é permitido ao driver executar para desempenhar qualquer inicialização específica de dispositivo que ele possa precisar. Uma vez que todos esses passos tenham sido completados, o driver é completamente instalado, do mesmo modo que qualquer driver instalado estaticamente.

Os módulos do kernel estão localizados no diretório **/lib/modules/versão_do_kernel/*** (onde versao_do_kernel é a versão atual do kernel em seu sistema, caso seja **2.6.23.6** o diretório que contém seus módulos será **/lib/modules/2.6.23.6.**

Os módulos são carregados automaticamente quando solicitados através do programa **kmod** ou manualmente através do arquivo **/etc/modules**.Os seguintes comandos são comandos de gerenciamentos dos módulos:

Insere módulo e os demais módulos necessários:

- **modprobe [options] module**
- **insmod [options] module**

Remove módulo:

- **rmmod [options] module**

Mostar as listas de Módulos instalados:

- **lsmod**

Insere, remove, lista módulos carregados e informações sobre um módulo:

- **modinfo [options] module**

Gera dependência entre módulos:

- **Cria /lib/modules/*/modules.dep**
- **depmod [options] module1.o module2.o...**

Atenção: Não compile o suporte ao seu sistema de arquivos raíz como módulo, isto o tornará inacessível, a não ser que esteja usando initrd.

2.3.2. Exemplos de Módulos

Aqui esta um exemplo de codigo de um modulo que tem como objetivo de ler um dado na porta do computador e de imprimir esse dado na tela. Os macros quem compões o arquivos modulo.h são geralmente:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <asm/io.h>
#include <asm/segment.h>

static int lpt=0x378;
static char data=0;

module_param(data,byte,0);
MODULE_PARM_DESC(data,"dado p/porta");

int init_module(void)
{
printk(KERN_INFO "init\n");
outb(data,lpt);
return 0;
}
void cleanup_module(void)
{
printk(KERN_INFO "cleanup\n");
}

module_init(init_module);
module_exit(cleanup_module);
```

Figura 5. Estrutura do Código do Modulo.

- #include ¡linux/module.h¿
- MODULE_INFO$tab, info$
- MODULE_ALIAS$alias$
- EXPORT_SYMBOL$symbol$
- MODULE_AUTHOR$autor_str$
- MODULE_LICENSE$license$
- ”GPL”, ”GPL v2”, ”GPL and additional rights”, ”Dual BSD/GPL”, ”DUAL MPL/GPL”, ”Proprietary”
- MODULE_VERSION$description_str$
- MODULE_DESCRIPTION$description_str$
- MODULE_SUPPORTED_DEVICE$device_str$

Os módulos podem receber parâmetros ao serem carregados

- Parâmetros são variáveis globais no módulo que podem ser incializadas na carga
- module_param$name, type, perm$ byte, short, ushort, int, uint, long, ulong, charp, bool, invbool
- module_param_array$name, type, nump, perm$
- MODULE_PARM_DESC$parm, desc_str$

2.3.3. Tutorial de desenvolvimento de módulos para o Kernel de Linux

- **Módulo** Abaixo, é apresentado o código em alomundo.c que será o exemplo de modulo para o Kernel. Inicialmente é incluído o cabeçalho module.h que possui

```
#include <linux/module.h>

MODULE_LICENSE("Dual BSD/GPL");

static int alo_inicio(void) {
   printk("Alo, Mundo!\n");
   return 0;
}

static void alo_fim(void) {
   printk("Adeus, Mundo Cruel!\n");
}

module_init(alo_inicio);
module_exit(alo_fim);
```

Figura 6. alomundo.ko.

as definições necessárias para a compilação de módulos.
  - **MODULE_LICENSE()** - esta macro informa a licença do módulo (no exemplo, o código é disponibilizado sob as licenças BSD e GPL).
  - **module_init()** - macro que define quais funções são chamadas quando o módulo é carregado. Neste exemplo, apenas a função alo_inicio() é chamada.
  - **module_exit()** - macro que define quais funções são chamadas antes do módulo ser removido. Neste exemplo, apenas a função alo_fim() é chamada.
  - **printk()** - função que escreve mensagens do kernel em **/var/log/syslog**.

- **MAkeFile**
  Para compilar e gerar o arquivo .ko do módulo, é preciso criar o arquivo Makefile. No arquivo
  - **obj-m** - especifica os arquivos-objeto que serão usados para gerar os módulos carregáveis do kernel. Neste exemplo, será usado o arquivo alomundo.o para gerar o arquivo alomundo.ko.

- **-C** diretório - esta opção muda para o diretório especificado antes de ler o Makefile (vai usar esse ambiente na geração dos módulos).
- **M diretório modules** - vai para o diretório especificado antes de gerar os módulos (os arquivos serão armazenados nesse diretório).
  O Makefile tem dois alvos (targets):

- **all**- compila o arquivo arquivo.c e gera o módulo. Basta digitar
  make all
  Como este é o padrão, pode-se digitar apenas
  make
- **clean** - deleta todos os arquivos gerados na execução do Makefile.
  make clean

```
obj-m := alomundo.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
```

Figura 7. criando o Arquivo MakeFile.

- **Carregando o módulo**
  - Para carregar o módulo sem verificar erros, use o comando insmod:
    sudo insmod alomundo.ko

  - Para carregar o módulo apenas se não houver erros de dependência, use o comando modprobe:
    sudo modprobe -a ./alomundo.ko
- **Verificando os módulos**
  - Para ver quais módulos estão carregados, use o comando lsmod:
    lsmod

  - ou verifique o conteúdo do arquivo modules do diretório /proc:
    less /proc/modules
- **Informações do módulo**
  - Para ver as informações do módulo use o comando modinfo
    modinfo alomundo.ko
- **Dependências**
  - Para verificar as dependências dos módulos use o comando depmod:
    sudo depmod
  - Este comando grava o arquivo **modules.dep** no diretório **/lib/modules/¡versão do Linux¿** com as dependências de todos os módulos.
- **Removendo o módulo**
  - Para remover o módulo, use o comando rmmod:
    sudo rmmod alomundo

  - Também é possível usar o comando **modprobe**. Mas, neste caso, o comando só funciona se o módulo estiver armazenado em **/lib/modules**: **sudo modprobe -r alomundo**
- **Log do sistema**
  - Para ver as mensagens do módulo no log do sistema, use o comando dmesg: **dmesg**
  - Outra forma para verificar as mensagens do log é olhar o arquivo syslog **less /var/log/syslog**

3. RTAI 5.0 Com UBUNTU 16.04

3.1. O que são sistemas de tempos Real ?

Sistemas operacionais são sistemas que oferecem suporte a usuários e suas aplicações através de diversas rotinas, que formam o chamado Kernel. De modo geral, a principal função de um sistema operacional é gerenciar de maneira segura os recursos do sistema (hardware e afins). Um sistema operacional de tempo real tem as mesmas funcionalidades de um sistema operacional com funcionalidades extras. Este tipo de sistema operacional visa não só a corretude dos resultados produzidos, mas também do tempo em que estes resultados são gerados. De acordo com o uso do sistema, diferentes níveis de suporte a tempo real podem ser utilizados. Sistemas Hard Real-Time são aqueles em que para qualquer aplicação as respostas devem ser produzidas dentro de um intervalo de tempo bem definido, ou seja, se uma resposta tardar a chegar o prejuízo será muito grande, como um acidente. Já sistemas Soft Real-Time têm suas respostas dentro de um tempo médio, ou seja, são sistemas que não causam perdas tão grandes caso uma resposta chegue atrasada.

3.2. RTAI

O projeto RTAI teve inicio no "Dipartimento di Ingegneria Aerospaziale del Politecnico di Milano" (DIAPM) em 1996/1997. Seu objetivo era tornar disponível uma ferramenta que permitisse o desenvolvimento de atividades de pesquisa relacionadas a controle ativo de atuadores aerodinâmicos, e que isto fosse possível de ser realizado em computadores pessoais 32 bits, placas de aquisição de dados comuns e através da utilização de linguagens de alto nível para permitir fácil uso por estudantes de graduação. Inicialmente o RTAI foi desenvolvido como um conjunto de patches para o Linux. Com o surgimento do Adeos o RTAI passou a utilizá-lo assumindo uma maneira mais estruturada e flexível de suporte a Tempo Real no Linux. Para obter determinismo de Hard Real Time, o RTAI é o domínio de mais alta prioridade na linha de propagação de interrupções do Adeos, sempre processando as interrupções antes do domínio Linux. Desta maneira a aplicação hard real time no RTAI é servida primeiramente ainda que seja necessário fazer preempt de qualquer outro processo não hard real time.

O RTAI dispõe de três diferentes escalonadores: o Uniprocessor (UP), é otimizado para maquinas uniprocessadas; o Symetric Multi Processors (SMP) permite executar qualquer tarefa pronta em qualquer CPU, enquanto permite definir a seleção de uma CPU específica para uma tarefa; o escalonador Multi uniProcessor, diferentemente, impõe que toda tarefa é amarrada a uma CPU especifica desde o momento de sua criação, permitindo

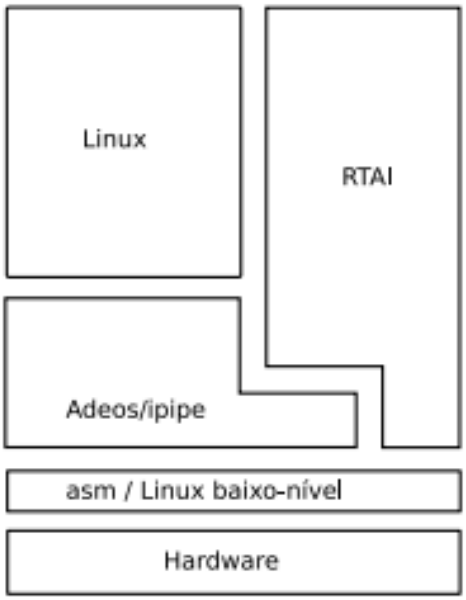


Figura 8. Camadas no RTAI..

a obtenção de melhores performances pela exploração mais eficiente das caches. Também estão disponíveis as seguintes politicas de escalonamento:

- First in Fisrt Out (FIFO) totalmente preemptível, para escalonamento cooperativo voluntário. Uma tarefa segura a CPU até que a libere ou até que uma tarefa de maior prioridade preempte sua execução. Sob esta politica de escalonamento há uma função de suporte para auxiliar o cumprimento dos prazos de execução utilizando prioridades atribuídas estaticamente de acordo com o conceito de Taxa Monotônica.
- Round Robin (RR), como o FIFO mas com um limite de tempo por tarefa, depois da qual a CPU pode ser repassada a qualquer outra tarefa de mesma prioridade esperando na fila para execução.
- Early Deadline First (EDF), para atribuir prioridades dinamicamente de maneira a cumprir o prazo final para termíno de execução das tarefas. Requer do usuário atribuir estimativas razoavelmente boas para o tempo de execução requerido por cada tarefa periódica.

Resumidamente o RTAI oferece os seguintes serviços:

- Gerenciamento básico de tarefas.
- Gerenciamento de memória. Memória compartilhada inter-tarefas, espaço de dados compartilhado entre espaços de usuário e kernel.
- Semáforos.
- Mecanismos de comunicações inter-processos. Incluindo RPC.
- Tratamento de interrupções em espaço de usuário.
- Monitores Watchdog.

3.3. Configuração de Base

A implementação do RTAI foi efetuado na vitual de um Computador DELL:

- Processador:Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz
- HD: 1TB
- Memoria RAM: Memória 8GB

3.3.1. Hardware

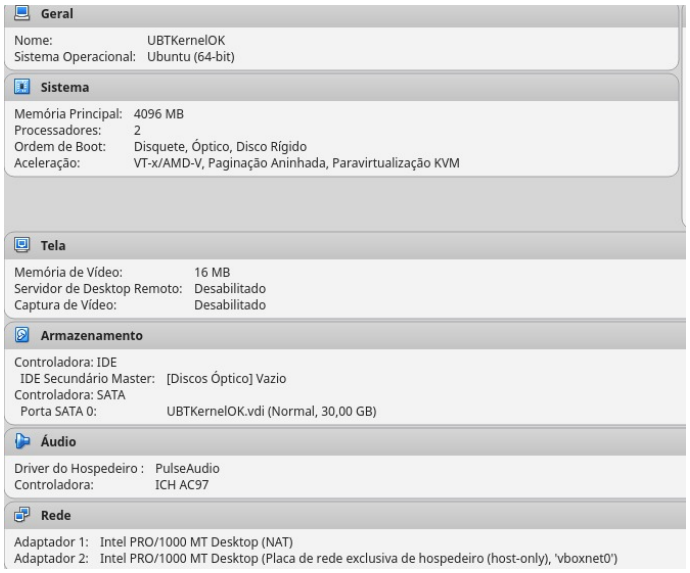


Figura 9. Camadas no RTAI..

3.3.2. Software

- Sistema Operacional: O UBUNTU 16.4 foi escolhido para a implementação do RTAI.
- Compilador: gcc version 5.4.0
- Kernel Configuração: build-essential version(12)

3.4. Processos de Intalação

O Processo de Instalação foi seguindo os passos dentro do Tutorial: ⟨https://sites.google.com/site/thefreakengineer/tutorials/rtai-5-0-1-lubuntu-16-04-x64-1⟩ Do modo geral, as diferentes etapas foram seguidos;

- Os pacotes e dependências:
- Configurar o kernel do Linux.
- Compilar e instalar o kernel do Linux.
- Configuração do Grub
- Configurar o Rtai 5.0
- Compilar o Rtai 5.0
- Primeiro teste em tempo real.

3.5. Configuração depois a instalação do RTAI

Depois a instalação do RTAI, foi efetuado uma reinicialização do sistema, e foi escolhido a opção ubuntu com RTAI no grub para inicializar o Kernel com os modulos do RTAI... Aqui são as imagens testemunhando a apresentação do ambiente depois a instalação do RTAI.

Figura 10. Imagem do GRUB na Inicialização da Maquina Virtual

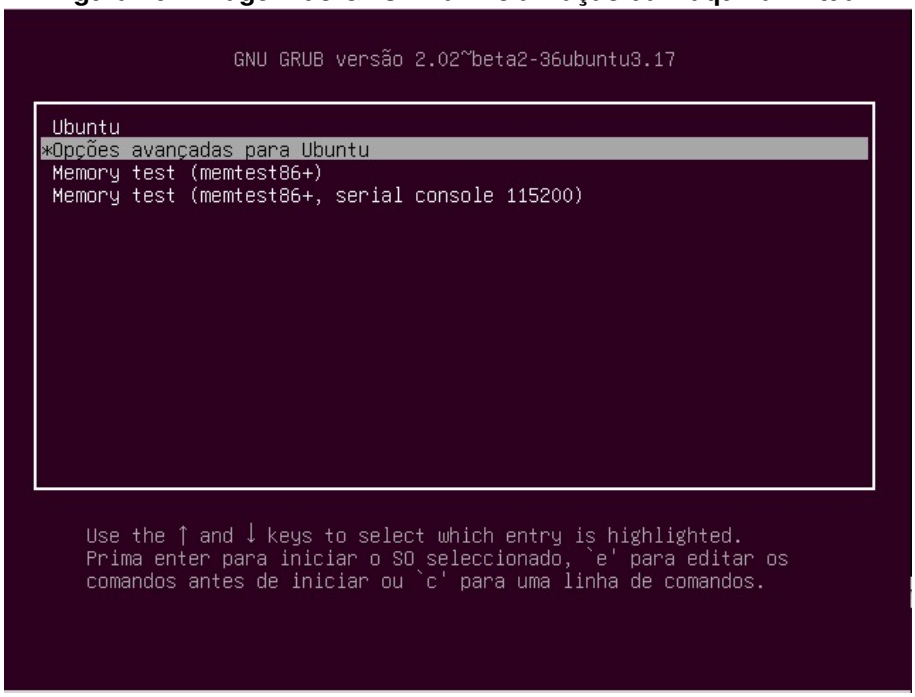


Figura 11. Escolha do Modo Ubuntu com RTAI

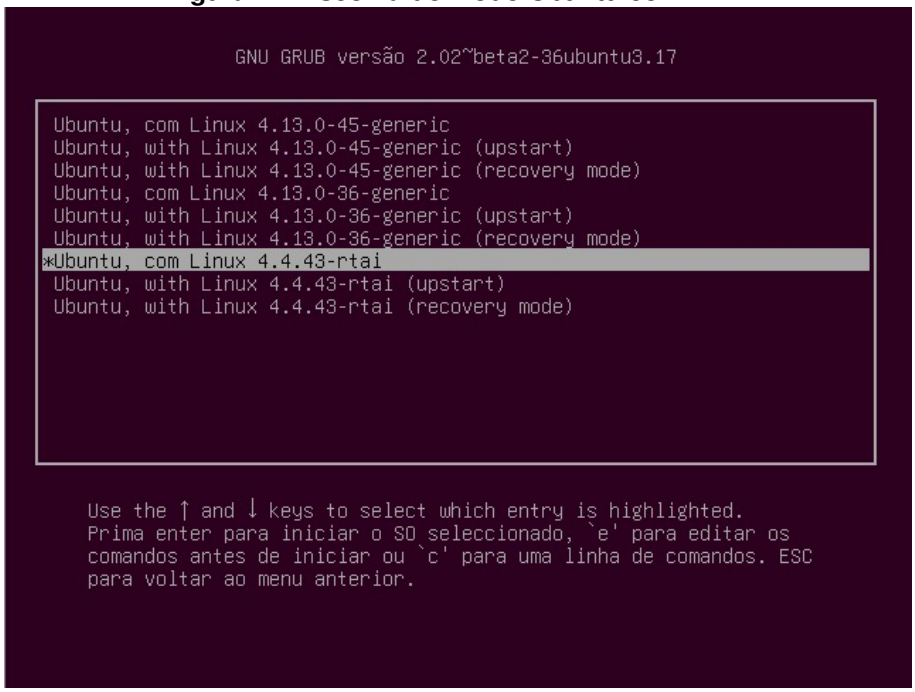


Figura 12. Teste de latency do kernel no RTAI 1

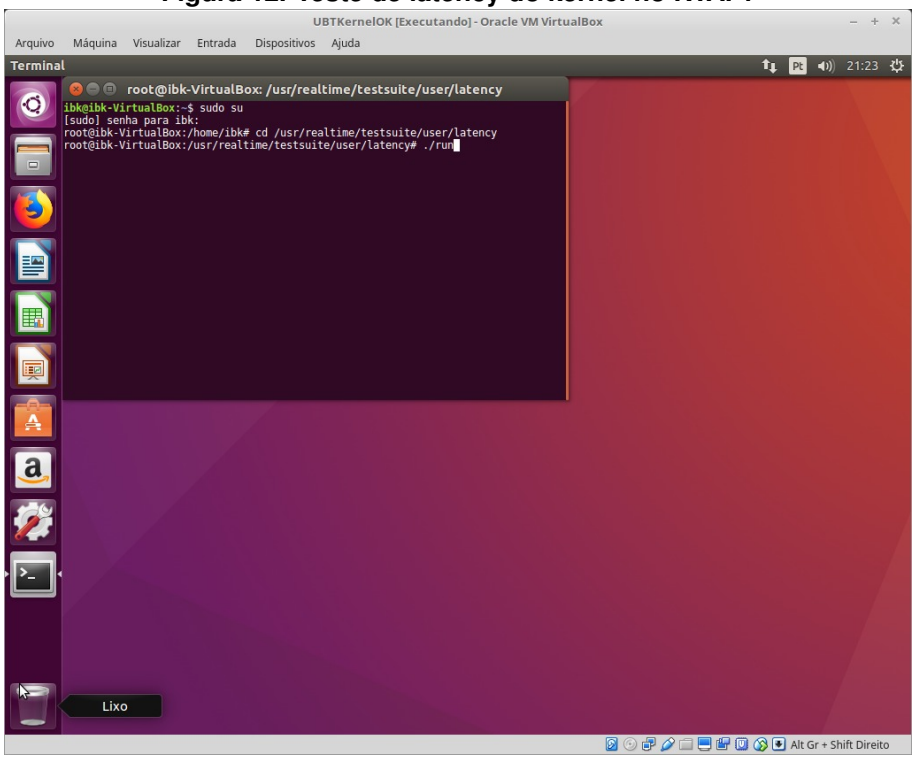


Figura 13. Teste de latency do kernel no RTAI 2

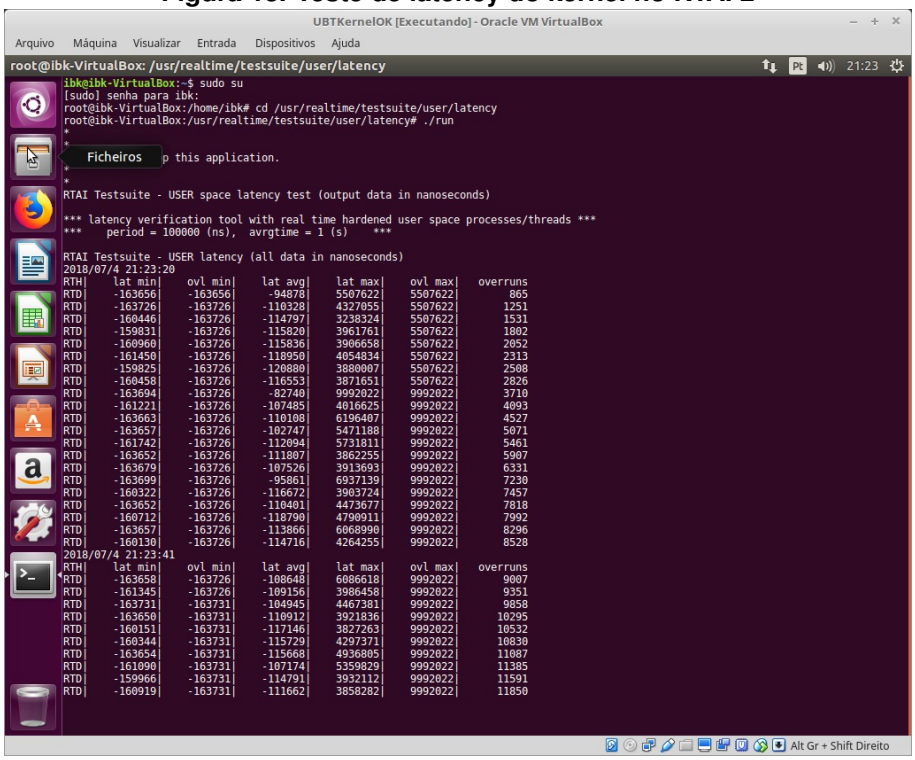
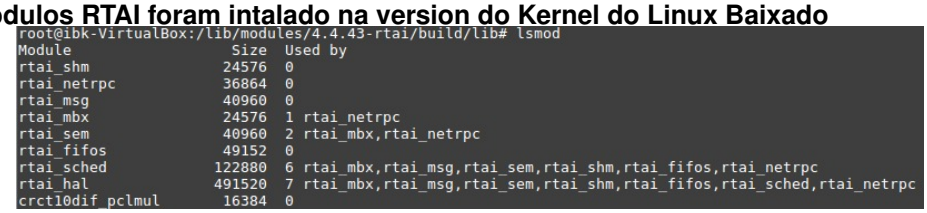


Figura 14. Mostrando a lista do Modulo instalado. Notamos que realmente os módulos RTAI foram intalado na version do Kernel do Linux Baixado



3.5.1. Vantagens e Desvantagens do sistema operacional Executando uma frequência mínima

Vantagens do sisitema operacional executando em frequência minina: baixo consumo de energia, aumento da duração da bateria, redução da geração de calor da cpu e pode aumenta a estabilidade do sisitema e a compatibilidade com aplicativos antigos, maior vida útil do hardware. Uma desvantagem seria a redução do desempenho.

4. Consideração Finais

Neste trabalho foi desenvolvido e alterado o kernel do Linux instalamos o modulo do RTAI 5 no Ubuntu 16.04 em uma máquina virtal, foi reunido e construido neste, um pequeno conhecimento sobre aplicações em tempo real e sobre o kernel e seus módulos. Apesar da facilidade de construção de módulos básicos para o kernel e configuração do RTAI 5, foi difícil encontrar material atualizado para construção de módulos utilizando RTAI e que tivesse informações detalhadas sobre sua construção.

5. Referências

A maior parte do Nosso trabalho foi Baseada no livro (Tanenbaum 2016). Porem os seguintes sites ajudaram muito a ver o lado pratico dos sistemas operacionais se referindo aos assuntos abordados no Relatórios

⟨https://sites.google.com/site/thefreakengineer/tutorials/rtai-5-0-1-lubuntu-16-04-x64-1⟩
Acessado : 25/06/18.

⟨https://www.rtai.org/userfiles/downloads/RTAICONTRIB/RTAI_Installation_Guide.pdf⟩
Acessado : 25/06/18.

⟨http://www.dei.isep.ipp.pt/~npereira/aulas/strl/07/teorica-rtai⟩
Acessado :26/06/18.

⟨https://www.embarcados.com.br/desenvolvimento-kernel-linux/⟩
Acessado : 27/06/18.

⟨http://www.ece.ufrgs.br/~fetter/eng04008/modules.pdf⟩
Acessado : 01/07/18.

⟨http://www.uniriotec.br/~morganna/guia/modulos.html⟩ Acessado : 01/07/18.

⟨https://www.ibm.com/developerworks/br/library/l-linux-kernel/index.html⟩
Acessado : 02/07/18.

⟨https://pt.wikipedia.org/wiki/Linux_(n%C3%BAcleo)⟩
Acessado :02/07/18.

⟨http://www.guiafoca.org/cgs/guia/intermediario/ch-kern.html⟩
Acessado : 03/07/18.

⟨https://www.vivaolinux.com.br/artigo/Gerencia-e-criacao-de-modulos-do-kernel?pagina=2⟩
Acessado : 04/07/18.

⟨http://www.ece.ufrgs.br/~fetter/eng04008/modules.pdf⟩
Acessado : 04/07/18.

⟨http://web.mit.edu/rhel-doc/3/rhel-sag-pt_br-3/ch-kernel-modules.html⟩
Acessado :04/07/18.

⟨https://en.wikipedia.org/wiki/Underclocking#Advantages⟩
Acessado :04/07/18.

Referências

aaaa aaaa.

Tanenbaum 2016 Tanenbaum, A. S. (2016). *Sistemas Operacionais Modernos*. Pearson, 4ª ed edition.