Kernel Monolítico e Microkernel

Morgana Macedo Azevedo da Rosa, UCPEL

Resumo—O sistema operacional tem a função de ser a interface entre as aplicações do usuário e o hardware. A parte do sistema operacional que realiza essa interface é chamada de kernel, que recebe as chamadas de sistema das aplicações do usuário e se comunica com o hardware através dos drivers. Essas chamadas de sistema são primitivas que realizam operações de manipulação de dados e dispositivos. O microkernel estrutura o sistema operacional removendo todos os componentes não essenciais do kernel e implementando-os como programas de sistema e de nível de usuário. A estrutura monolítica pode ser comparada como uma aplicação formada por vários procedimentos que são compilados separadamente e depois linkados, formando um grande e único programa executável.

Palavras-Chave—kernel, microkernel, monolítico, nível de usuário, chamadas de sistema.

1 Introdução

O Kernel tem a função de ser uma interface de comunicação entre os programas e o hardware. Essa comunicação entre o programa e o kernel é realizada através de chamadas de sistema.

As Arquiteturas monolíticas eram utilizadas nos primeiros sistemas operacionais e tinham como característica serem sistemas em que todos os códigos-fonte e bibliotecas estão ligados em apenas um grande arquivo executável. Esse tipo de implementação nos dias atuais não tem mais lugar, pois não possibilita manutenções ou modificações de forma rápida e fácil.

Com os avanços da computação, os sistemas operacionais ficaram mais complexos e maiores. Várias equipes de desenvolvimento são necessárias para o desenvolvimento do sistema, para tornar uma tarefa menos complicada, sistemas operacionais foram desenvolvidos em camadas, em que cada camada provê funções de acesso para as camadas superiores. Desse modo as camadas mais internas, mais perto do kernel, tem acesso mais privilegiado do que as camadas mais externas.

Os microkernels são kernels em que os desenvolvedores retiram todos os módulos, ferramentas e aplicações que não são relevantes para determinada arquitetura, tornando este microkernel adaptado aos recursos de hardware do dispositivo. [1]–[3]

- ***Morgana Macedo Azevedo da Rosa**: Engenharia da Computação, Centro Politécnico - CPoli. Universidade Católica de Pelotas - UCPEL.*
E-mail:morganamacedoazevedodarosa@gmail.com

1.1 Motivações

A principal razão para a realização de um projeto sobre kernel, kernel monolítico e microkernel, é a vasta importância para a estrutura de um sistema operacional.

O kernel do sistema operacional Linux, cujo sistema, é um dos mais proeminentes exemplos de desenvolvimento com código aberto e de software livre, é um kernel monolítico, onde os serviços básicos são agrupados dentro do kernel. E algumas de suas funções como o drivers de dispositivo, suporte a rede, sistemas de arquivo, podem ser compiladas e executadas como módulos Loadable Kernel Modules (LKM), que são bibliotecas compiladas separadamente da parte principal do kernel e podem ser carregadas e descarregadas depois de este estar em execução.

A arquitetura do microkernel, na qual o kernel oferece serviços básicos, como comunicação, entrada e saída e gerenciamento de memória e processo, e serviços mais específicos são plugados na camada do microkernel. [4]

1.2 Objetivos

Esse capítulo irá abordar as metas que tal artigo deverá alcançar.

1.2.1 Objetivo Geral

Desenvolver o conhecimento adquirido sobre kernel, microkernel e kernel monolítico, elucidar as comparações arquiteturais, histórico, bem como as características técnicas e os cenários de uso.

1.2.2 Objetivo Específico

- Compreender kernel;
- Compreender microkernel;
- Compreender kernel monolítico;
- A importância do arquitetural do kernel;
- Comparar kernels;
- Aspectos positivos e negativos do microkernel e kernel monolítico;
- Reconhecer as potencialidades das arquiteturas disponíveis.

1.3 Estrutura do Artigo

Esta seção, introduz a motivação e a objetivação arquitetural e sistemática do kernel monolítico e do microkernel, na seção 2 demonstraremos alguns projetos que introduzam em suas aplicações o uso do kernel, no capitulo 3 iremos discutir e apresentar os conhecimentos adquiridos sobre tal tema, no capítulo 4 iremos demonstrar onde podemos utilizar a ferramenta que faz o intermédio entre o hardware e sistema operacional, na seção 5 elucidaremos exemplos de kernels monolíticos e de microkernels, na seção 6 enfatizaremos as conclusões obtidas ao longo do processo de aprendizado.

2 Trabalhos Relacionados

O kernel Linux fornece uma API para segurança—o Linux Security Modules Framework [5]—que permite a monitoração e controle do acesso aos objetos do kernel ou dos processos. Esses objetos podem variar de arquivos no disco até kernel, como a capacidade de interagir com sockets. Através desse framework é possível registrar um módulo no kernel, a fim de supervisionar ações em outros subsistemas e assim ser capaz de aplicar regras de segurança.

Exemplos de frameworks de segurança que utilizam o LSM (Linux Security Modules) são o SELinux [6] e AppArmor [7]. Esses dois sistemas criam, em conjunto com o LSM (Linux Security Modules), um controle de acesso por restrição (MAC2) capaz de limitar o que um processo pode fazer, seja com objetos em nível de usuário ou do kernel. Com isso, incrementa-se o gerenciamento do acesso e as permissões já existentes no Linux, ao atribuir aos processos labels ou tags relacionadas às permissões concedidas.

Enfatizamos também que tais trabalhos relacionados foram citados devido a forma como estruturam o kernel, apesar de serem classificados como frameworks de segurança, utilizam uma porção do kernel do sistema operacional Linux responsável pela segurança (LSM - Linux Security Modules), de forma a não ficar repetitivo, tal trabalho também relacionará outros trabalhos como o kernel do PikeOS, UNIX, Linux, Windows NT, Amoeba, Mach e Sprite .

3 CARACTERÍSTICAS TÉCNICAS

Nesta seção serão apresentadas as discuções arquiteturais de kernel monolítico e microkernel, bem como suas funções operacionais, benefícios e distinções.

3.1 Kernel

É um gerenciador de recursos. Se o recurso que está sendo gerenciado for um processo, uma memória ou um dispositivo de hardware, o kernel gerencia e intermedeia o acesso ao recurso entre os vários usuários concorrentes. É um conjunto de rotinas que oferecem serviços aos usuários de sistema e suas aplicações. As principais funções do núcleo são: tratamento de interrupções, criação e eliminação de processos, sincronização e comunicação entre processos, escalonamento e controle dos processos, gerência de memória, gerência do sistema de arquivos, operações de entrada e saída, contabilização e segurança do sistema. [1], [8]

3.1.1 Chamadas de sistema

Também conhecidas como System Calls ou Syscalls, são primitivas através das quais os programas requisitam algum serviço do kernel do sistema operacional. Estes serviços podem acessar o disco rígido, criar e executar um novo processo. Portanto, chamadas de sistema são meios de comunicação entre o programa, processo e o sistema operacional. As chamadas de sistema são responsáveis de ser a interface entre o programa em execução e o kernel. A comunicação entre o kernel e o hardware é realizada pelos drivers através

dos subsistemas de cada dispositivo, presentes no kernel.

O Linux possui em média, trezentas chamadas de sistema que permitem acesso aos mais diversos tipos de informação, como gerência do processador, gerência de memória. Nos sistema Linux as chamadas de sistema mais comuns são: open (abrir arquivo), read (ler da memória), write (escrever na memória), close (fechar arquivo ou área de memória), wait (esperar por alugum evento), execve (executar alguma chamada), fork (dividir o fluxo de uma execução), exit (finalizar), kill (matar, utilizando algum processo não finaliza). [1], [4], [8]

3.1.2 Kernel Monolítico

O kernel monolítico é uma arquitetura de sistema operacional mais antiga e mais comum. Cada componente do sistema operacional é contido no núcleo e pode comunicar-se diretamente como qualquer outro, usando as chamadas de sistema. O kernel normalmente é executado com acesso irrestrito ao sistema de computador. O Linux, OS/360 e o VMS são caracterizados por possuírem kernels monolíticos altamente eficientes.

Entretanto, porque os kernels monolíticos agrupam componentes todos juntos, é difícil localizar a fonte de problemas e de outros erros. Além disso, como todo código é executado com acesso irrestrito ao sistema, sistemas de kernel monolítico são particularmente suscetíveis a danos provocados por códigos sujeitos a erros ou mal-intencionados.

Apesar da estrutura monolítica ser de longe a mais usada, ela poderia ser chamada de “a grande confusão” [8]. Simplesmente não há estruturação visível na organização monolítica. O sistema operacional é escrito como um conjunto de procedimentos, cada um dos quais podendo chamar qualquer dos demais sempre que necessário. Quando essa técnica é usada, cada procedimento do sistema deve ter uma interface bem definida em termos de parâmetros e de resultados, sendo, cada procedimento livre para chamar qualquer outro se este último realizar algo que o primeiro necessite.

Para construir o programa objeto relativo ao sistema operacional quando essa estrutura for usada, devemos compilar todos os procedimentos individuais, ou os arquivos contendo tais procedimentos, e então liga-los todos juntos em um único arquivo-objeto. Não há nenhuma possibilidade de se implementar restrições à visibilidade de informações, cada procedimento é visível a todos os outros, em contraposição a uma estrutura que contenha módulos ou packages, nas quais podemos controlar quais informações são locais ao módulo e só aquelas designadas como entry points poderão ser acessadas de fora do módulo em que foram definidas.

Mesmo em kernels monolíticos é possível ter um mínimo de estruturação. Os serviços, chamadas de sistema, fornecidos pelo sistema operacional, são requisitados através da colocação de parâmetros em lugares muito bem determinados, tais como registradores ou pilhas, seguindo-se a execução de uma instrução especial de trap, conhecida como chamada ao supervisor ou chamada ao kernel.

A execução deste tipo de instrução chaveia a máquina do modo usuário para o modo kernel, e transfere o controle ao sistema operacional. A maioria dos processadores trabalha em dois modos de processamento: o modo kernel para o sistema operacional, onde é permitida a execução de todas as instruções básicas da máquina, e o modo usuário para os programas de usuário, onde certas instruções, como as de entrada e saída, não podem ser executadas.

O sistema operacional então examina os parâmetros da chamada para determinar quais das chamadas de sistema deve ser executada. Após o sistema operacional entra em uma tabela indexada que identifica a rotina de serviço, que é então chamada. Finalmente a chamada de sistema é concluída, e o controle retorna ao programa do usuário. Assim um programa principal que chama o procedimento de serviço, um conjunto de procedimentos de serviço que executam cada uma das chamadas de sistema, um conjunto de procedimentos utilitários que auxiliam na execução dos procedimentos de serviço. [1], [2], [4], [8]

3.1.3 Microkernel

À medida que o UNIX se expandiu, o kernel tornou-se grande e difícil de gerenciar. Em meados dos anos 80, pesquisadores da Carnegie Mellon University desenvolveram um sistema operacional chamado de Mach que modularizou o kernel, usando a abordagem de microkernels. Esse modulo estrutura o sistema operacional removendo

todos os componentes não-essenciais do kernel e implementando-os como programas de sistema e de nível de usuário. O resultado é um kernel menor. Existe pouco consenso com relação a quais serviços devem permanecer no kernel e quais devem ser implementados no espaço de usuário. Em geral, no entanto, os microkernels geralmente fornecem gerência mínima de memória e processos, além de um recurso de comunicação.

A principal função do microkernel é fornecer um recurso de comunicação entre o programa cliente e os vários serviços que também estão em execução no espaço de usuário. A comunicação é fornecida por meio de troca de mensagens. Por exemplo, se o programa cliente desejar acessar um arquivo, ele deverá interagir com o servidor de arquivos. O programa cliente e o serviço nunca vão interagir diretamente, ao invés disso, se comunicam indiretamente trocando mensagens com o microkernel.

Os benefícios da abordagem do microkernel incluem a facilidade de expandir o sistema operacional. Todos os novos serviços são adicionados ao espaço de usuário e, consequentemente, não exigem modificação do kernel. Quando o kernel precisa ser modificado, as alterações tendem a ser menores, porque o microkernel é um kernel menor. É mais fácil transportar o sistema operacional resultante de um projeto de hardware para outro. O microkernel também fornece mais segurança e confiabilidade, pois a maior parte dos serviços estão sendo executados como processos de usuário, em vez de kernel. Se um serviço falhar, o resto do sistema operacional permanece inalterado.

Vários sistemas operacionais contemporâneos usam a abordagem de microkernel. O UNIX Digital fornece uma interface UNIX com o usuário, mas é implementado com um kernel Mach. O Mach mapeia as chamadas ao sistema UNIX em mensagens para os serviços apropriados de nível usuário. O sistema operacional Apple MacOS X Server baseia-se no kernel Mach. O Windows NT utiliza uma estrutura híbrida. O Windows NT é projetado para executar várias aplicações, incluindo Win32 (aplicações nativas do Windows), OS/2 e POSIX. Fornece um servidor que executa no espaço de usuário para cada tipo de aplicação também executam no espaço usuário. O kernel coordena troca de mensagens entre as aplicações cliente e servidor de aplicação. [1], [2], [4], [8]

3.1.4 Kernel Monolítico X Microkernel

O kernel monolítico é simplesmente um sistema operacional centralizado que foi ampliado com facilidades de rede e integração de serviços remotos. A maioria das chamadas de sistema são feitas mandando um trap para o kernel, com o trabalho sendo feito no kernel, e com este retornando os resultados ao processo usuário. Com tal abordagem, a maioria das máquinas tem de possuir discos, e precisa gerenciar seus próprios sistemas de arquivos locais. A maioria dos sistemas distribuídos que são extensões ou mesmo imitações do UNIX usa esta solução, pois o UNIX, é ele próprio, um grande e monolítico kernel.

A maioria dos sistemas distribuídos utilizam o microkernel, que é mais flexível pelo fato de não fazer quase nada fornecendo serviços como: um mecanismo de comunicação entre processos, um mínimo de funções para gerência de memória, um mínimo de funções de gerência de processos e de escalonamento, funções de entrada e saída de baixo nível.

Em particular, ao contrário do kerenel monolítico, o microkernel não fornece o sistema de arquivos, o sistemade diretórios, a gerência completa de processos, e a manipulação da maioria das chamadas de sistema. Todos os serviços prestados pelo microkernel estão lá por serem difíceis de prestar em outro lugar ou muito caros quando oferecidos fora do kernel.

Todos serviços oferecidos pelo sistema operacional são geralmente implementados como servidores ao nível do usuário. Para procurar um nome, ler um arquivo, ou obter algum outro tipo de serviço, o usuário deve enviar uma mensagem. A vantagem deste método é o fato dele ser altamente modular: existe uma interface muito bem definida para cada serviço, e cada serviço é igualmente acessível a todos os clientes, independente da sua localização no sistema. Além disso, o esquema do microkernel é fácil de implementar, muito fácil de instalar, e de acrescentar novos serviços, uma vez que a adição ou a modificação de determinado serviço não implica parar o sistema e dar o boot no novo kernel, como no caso do sistema monolítico. É precisamente a capacidade de acrescentar, apagar e modificar os serviços que dá a flexibilidade ao microkernel.

No kernel monolítico o sistema de arquivos é construído dentro do próprio kernel, seus usuários não tem escolha, a não ser usar aquela que está implementado.

A única vantagem potencial do sistema com kernel monolítico é sua performance. É muito mais rápido mandar um trap para o kernel, que executa todo o serviço necessário, do que mandar uma mensagem para os servidores remotos. [1], [2], [4], [8]

4 Cenários de uso

O kernel faz a intermediação entre o hardware e os programas executados pelo computador. Responsável por tarefas como:

- Tratamento de interrupções;
- Contabilização e segurança do sistema;
- Operações de entrada e saída;
- Gestão de sistema de ficheiros;
- Gestão de memória, alocação e liberação de memória;
- Escalonamento, controle, sincronização, comunicação, execução e eliminação de processos;
- Gerência de sistema de arquivos;
- Gerenciamento de dispositivos;
- Controle do sistema de arquivos;
- Gerência de memória;
- Acesso à memória;
- Alterar a data e hora do sistema;
- Alterar informações residentes no núcleo do sistema e acessar diretamente posições no disco. [4], [8]

5 Exemplos de sistemas basedos em kernel e microkernel

Este capítulo irá comparar os kernels de alguns exemplos de sistemas.

5.1 PikeOS

O PikeOS é um microkernel comercial desenvolvido pela empresa Sysgo AG e é utilizado para sistemas de tempo real embarcados como aeronaves, automóveis, automação industrial e médica, infraestrutura de rede e eletrônicos. Várias empresas utilizam o PikeOS como sistema de segurança para aplicações de missão crítica. [8]

5.2 UNIX

O kernel do UNIX não é muito bem estruturado, mesmo assim podemos descrever duas partes no kernel. Em seu nível mais baixo está o kernel dependente da máquina, cujo código, consiste nos manipuladores de interrupção, nos drivers de dispositivos do sistema de entrada/saída, e em parte do software para gerência de memória. A outra parte do kernel, conhecida como kernel independente da máquina é a mesma, qualquer que seja a máquina-alvo da implementação, em vista de ele não depender do hardware no qual o sistema estiver rodando. No código independente da máquina, incluem-se a manipulação das chamadas de sistema, a gerência dos processos, o escalonamento os pipes, o tratamento de sinais, a paginação e o swapping, o sistema de arquivos, e a parte de alto nível do sistema de entrada e saída. Felizmente a parte independente de máquina é bem maior que a parte dependente de máquina, razão pela qual é relativamente fácil transportar o UNIX para um novo hardware. [8]

5.3 LINUX

O Linux mantém o modelo tradicional do UNIX o kernel é criado como um binário único e monolítico. O motivo é melhorar o desempenho como todas as estruturas de dados e código do kernel são mantidas em um só espaço de endereçamento, não há a necessidade de trocas de contexto quando um processo chama uma função do sistema operacional ou quando é entregue uma interrupção de hardware. Não apenas o escalonamento central e o código da memória virtual ocupam esse espaço de endereçamento; todo código do kernel, inclusive todos os drivers de dispositivo, sistemas de arquivo e códigos de rede encontram-se no mesmo espaço de endereçamento único. [1], [8]

5.3.1 Sincronização do Kernel

Um processo em modo usuário não pode acessar diretamente os dados do kernel, hardware ou outros recursos críticos de sistema – esses processos precisam se valer do kernel para executar instruções privilegiadas em seu nome. Essas operações são denominadas caminhos de controle do kernel. Se dois caminhos de controle do kernel acessarem os mesmos dados concorrentemente, poderá resultar em uma condição de disputa. Pra que

isso seja enviado, o kernel proporciona dois mecanismos básicos que fornecem acesso mutuamente exclusivo a seções críticas: travas e semáforos. [2]

Spinlock

As travas giratórias (spinlock) permitem que o kernel proteja seções críticas do código do kernel executando em sistemas habilitados para SMP (Symmetric Multi-Processing). Uma trava giratória é uma trava (lock) onde a thread espera em um loop (spins) verificando repetidamente até que lock fique disponível. Para evitar o custo das trocas de contexto, um spinlock gira (spin) e verifica repetidamente para ver se o lock foi destravado. Assim que a trava giratória é adquirida, todas as requisições subsequentes à trava giratória provocam espera (giro) até que a trava seja liberada. O spinning deve ser muito rápido, de tal modo que a latência entre um unlock-lock seja pequena. As travas giratórias podem ser ineficientes, uma vez que a thread permanece ativa sem nada realizar de útil, caracterizando busy waiting (espera ociosa). [2]

Trava de leitor/escritor

A trava de leitor/escritor, em alguns casos, vários caminhos de controle de kernel precisam apenas ler (e não escrever) os dados acessados dentro de uma seção crítica. Para aperfeiçoar a concorrência em uma situação desse tipo, o kernel fornece travas de leitor/escritor, que possibilitam que vários caminhos de controle de kernel retenham uma trava de leitura, mas permitem que somente um caminho de controle de kernel retenha uma trava de escrita sem nenhum leitor concorrente. O caminho de controle de kernel que retém uma trava de leitura de uma seção crítica deve liberar sua trava de leitura e adquirir uma trava de escrita se desejar modificar dados. A tentativa de adquirir uma trava de leitura terá êxito apenas se não houver nenhum outro leitor ou escritor executando concorrentemente dentro de suas seções críticas, podendo melhorar o desempenho, também é possível que os escritores sejam adiados indefinidamente. [2]

Seqlocks

As seqlocks permitem que os escritores acessem dados imediatamente, sem precisar esperar que os leitores liberem a trava. Associam a trava giratória com um contador sequencial. Exigem que os leitores detectem se um escritor modificou os dados protegidos pelo seqlock examinando o valor do contador sequencial do seqlock, sendo apropriados para o tratamento de interrupções. [2]

Semáforos de Kernel

Travas giratórias e seqlocks funcionam bem quando as seções críticas que protegem contêm poucas instruções. Todavia, à medida que aumenta o tamanho da seção crítica, aumenta a quantidade de tempo gasta em espera ociosa, o que causa sobrecarga significativa e degradação do desempenho. Travas giratórias também podem causar deadlocks se o processo adormecer enquanto estiver retendo a trava. Quando uma seção crítica precisa ser protegida por longo tempo, semáforos de núcleo são uma alternativa melhor para implementação de exclusão mútua. Implementados como semáforos contadores.

Antes de entrar em uma seção crítica, é necessário chamar a função down: Se o valor do contador for maior que 0, o contador será decrementado, permitindo que o processo seja executado; se o valor do contador for igual a 0, o processo é adicionado à fila de espera e entra no estado adormecido; reduz a sobrecarga por causa da espera ociosa. Quando o processo sai da seção crítica, tem de chamar a função up: se o valor do contador for maior ou igual a 0, o contador será incrementado; se o valor do contador for igual a 0, um processo que está na fila de espera é acordado para executar sua seção crítica. [2]

5.4 Windows NT

O kernel do Windows NT fornece a base para o executivo e os subsistemas. O kernel nunca é descarregado da memória e sua execução nunca sofre preempção. Tendo quatro prioridades principais escalonamento de threads, tratamento de interrupções e exceções, sincronização de baixo nível do processador e recuperação após uma falta de energia. [4]

5.5 Amoeba

Como exemplo de sistema distribuído temos, o Amoeba, que implementa, um microkernel que deve rodar em todas as máquinas do sistema, que

trata do gerenciamento de processos e linhas de controle, fornece suporte para a gerência de memória, suporta comunicação entre processos e manipula as operações de entrada/saída. O sistema de arquivos e o restante do sistema operacional rodam como processos usuários. Essa divisão de trabalho mantém o kernel pequeno e simples. [4], [9]

5.6 Mach

O microkernel do Mach foi projetado e construído com o intuito de se tornar uma base sobre a qual tanto o UNIX quanto outros sistemas operacionais, pudessem ser emulados eficientemente. Essa emulação é realizada por uma camada de software que roda fora do kernel, no espaço do usuário. O microkernel do Mach, a exemplo de outros microkernels, presta serviço de gerência de processos, gerência de memória, comunicação entre processos, e entrada e saída. Os arquivos, diretórios e demais funções do sistema operacional são tratados no espaço do usuário. A ideia básica do Mach é a de fornecer os mecanismos necessários para a realização das tarefas do sistema, deixando a implementação das políticas para os processos no nível de usuário. O kernel gerencia abstrações como processos, linhas de comando, objetos de memória, portas mensagens. [4]

5.7 Sprite

O microkernel do Sprite, cujo é um sistema distribuído que conecta grande número de estações de trabalho, onde cada estação de trabalho do sistema executa um kernel Sprite, e esses kernels podem se comunicar uns com os outros via RPCs (Remote Procedure Call). Para garantir a transparência, o kernel Sprite distingue entre dois tipos de chamadas: chamadas dependentes de localização são chamadas ao sistema que produzem resultados diferentes para estações de trabalho diferentes; chamadas independentes de localização são chamadas ao sistema que produzem o mesmo resultado para todas as estações de trabalho. O sistema fornece mais chamadas independentes de localização apresentando exatamente a mesma visualização do sistema de arquivos para cada estação de trabalho. Quando uma chamada dependente de localização é requerida, o sistema ou repassa a chamada para o computador nativo para avaliação, ou transfere a informação de estado do processo (como memória virtual, arquivos abertos e identificadores de processo) do computador nativo para o computador alvo. [1]

6 Considerações Finais

O Kernel é o responsável por controlar o hardware e fornecer as chamadas de sistema para que os programas tenham acesso aos serviços do sistema, como criação e gerencia de processos, gerência de memória, sistema de arquivos. Os microkernels são kernels otimizados como relação à necessidade arquitetural, geralmente fornecendo gerencia mínima de memória e processos, além do recurso de comunicação. No kernel monolítico as estruturas de dados e códigos são mantidas em um só espaço de endereçamento, não havendo a necessidade das trocas de contexto quando um processo chama uma função do sistema operacional ou quando é entregue uma interrupção de hardware.

[1], [4], [8]

Referências

[1] Abraham. Silberschatz, Peter. Galvin, and Greg Gagne. Sistemas operacionais: conceitos e aplicações. *Campus*, 1ª, 2000.

[2] Harvey M. Deitel. Sistemas operacionais. *Pearson*, 3ª, 2005.

[3] Alexandre Carissimi. Sistemas operacionais. *Bookman*, 4ª, 2010.

[4] Francis B. Machado and Luiz Paulo Maia. Arquitetura de sistemas operacionais. *Livros Técnicos e Científicos Editora S.A.*, 2ª, 1996.

[5] J. Morris. Linux security module framework. *https://www.linux.com/learn/overview-linux-kernel-security-features*, 2013.

[6] J. Morris. Selinux project wiki. *http://selinuxproject.org.*, 2013.

[7] W. AppArmor. Apparmor project wiki. *http://wiki.apparmor.net.*, 2013.

[8] Andrew S. Tanenbaum. Sistemas operacionais modernos. *Prentice-Hall do Brasil LTDA*, 4ª, 2016.

[9] Ann Mclver Mchoes. Introdução aos sistemas operacionais. *Thomson Learning*, 1ª, 2002.

Morgana Macedo Azevêdo da Rosa Graduanda em Engenharia da Computação na Universidade Católica de Pelotas, na linha de pesquisa CDO-CMOS: circuitos digitais otimizados para processamento digital de sinais e codificação de vídeo. Bolsista de iniciação científica pelo programa – PIBIC/CNPq.