Allison Machado - allisonline.net

Humberto Rocha - humberto.io

Minix 3

Brasília - DF, Brasil

19 de junho de 2013

Resumo

Sistemas computacionais são formados por componentes de Hardware e Software. O Hardware é a parte física do sistema, formada por um conjunto abrangente de componentes eletrônicos, enquanto o Software compõe a parte lógica do sistema, que são as instruções capazes de guiar a execução de tarefas específicas. A interação principal entre estas duas partes só é possível pela existência do sistema operacional, que é capaz de abstrair a complexidade da máquina e fornecer um ambiente de trabalho para programadores e usuários do sistema computacional. O Minix é um sistema operacional gratuito, de código aberto, planejado para ser confiável e com um enfoque educacional que permite um aprendizado rico dos principais conceitos de sistemas operacionais - como o kernel, gerência de processos, gerência de memória e inicialização de sistemas. O Minix segue a arquitetura de microkernel, que fornece apenas a estrutura fundamental para a possível construção de um sistema operacional, no entanto seus pontos fortes são a modularidade e a confiabilidade.

Palavras-chaves: Sistemas Operacionais. Minix. Microkernel. Confiabilidade.

Abstract

Computational systems are made of hardware and software components. The hardware is the physical part of the system, it consists of a wide set of electronics components, while software is the logical part of the system which is capable of leading instructions to perform specific tasks. The main interaction between these two parts is only possible because of the operating system, which is able to abstract the machine complexity and provide a straight work environment for developers and computer system users. Minix is a free and open source operating system, designed to be reliable and having an educational focus that allows a rich learning of the main operating systems concepts - such as the kernel, process management, memory management and system startup. Minix follows the microkernel architecture, which only provides the fundamental structure for building an operating system, but its strengths are the modularity and reliability.

Keywords: Operating System. Minix. Microkernel. Reliability.

Lista de abreviaturas e siglas

IPC	*Inter-process Communication* - Comunicação entre Processos;
MMU	*Memory Management Unit* - Unidade de Gerenciamento de Memória;
PID	*Process Identifier* - Identificador de Processos;
UID	*User Identifier* - Identificador de Usuário;
BIOS	*Basic Input/Output System* - Sistema Básico de Entrada/Saída;
IEEE	*Institute of Electrical and Electronics Engineers* - Instituto de Engenheiros Eletricistas e Eletrônicos;
CPU	*Central Processing Unit* - Unidade Central de Processamento;
POSIX	*Portable Operating System Interface* - Interface Portável entre Sistemas Operacionais;
TCP/IP	*Transmission Control Protocol e Internet Protocol* - Protocolo de Controle de Transmissão e Protocolo de Internet;
OSI	*Open Systems Interconnection* - Modelo de Interconexão de Sistemas Abertos.

Sumário

Introdução

Sistema Operacional é um conjunto de software que possui duas grandes finalidades. A primeira é gerenciar os recursos de um sistema computacional como o uso da cpu, o acesso à memória e a entrada/saída; enquanto a segunda é oferecer uma camada de abstração de hardware, criando uma interface bem definida para os programas que utilizarão de seus recursos, o que faz com que o programador não tenha que se preocupar com a implementação das particularidades de hardware mas sim com que recurso ele quer utilizar (MAZIERO, 2013).

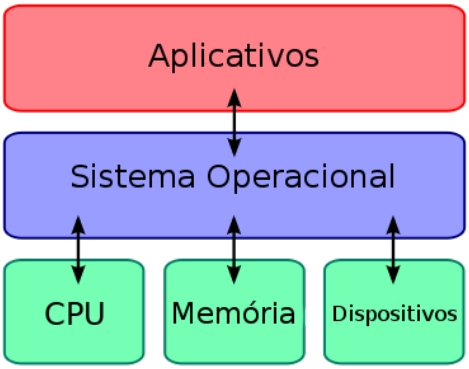

Figura 1 – Abstração do Sistema Operacional

Em um mundo sem sistemas operacionais, quando um usuário precisasse ler ou gravar um CD, por exemplo, seria necessário que o usuário programasse toda a interação com o dispositivo de CD. Logo, todos precisariam conhecer as interrupções, parâmetros passados à controladora e muito mais para acessar o dispositivo. De fato não seria prático, e considerando que existem vários outros equipamentos de Hardware, este problema se estenderia, sendo necessário um conhecimento específico para manipular cada um dos diferentes tipos de dispositivos.

O Sistema operacional é o programa que esconde dos usuários as dificuldades relacionadas à manipulação do hardware e apresenta uma visão limpa e clara das funcionalidades do sistema. Portanto, com o uso do sistema operacional deseja-se oferecer uma "máquina estendida" mais clara e fácil de manusear do que o Hardware. Considerando que computadores modernos possuem processos, memória, temporizadores, discos, mouses, interfaces de rede, impressoras, e uma variedade de outros dispositivos, o Sistema Operacional deve trabalhar para que exista uma alocação ordenada e controlada dos recursos para estes dispositivos, possibilitando que o usuário interaja com a máquina.

Para o desenvolvimento deste estudo foi utilizado como base o sistema operacional Minix. A primeira versão deste sistema foi lançada em 1987, e o principal idealizador

deste projeto foi o cientista da computação Andrew S. Tanenbaum, que imaginava um sistema voltado para fins educacionais. Conforme Tanenbaum e Woodhull (2008, p. 16): "O Minix foi criado com base no sistema Unix e na época do primeiro lançamento a sua licença foi considerada bastante liberal em relação aos outros sistemas existentes, até mesmo em relação ao Unix". Este fato, aliado à característica de que o sistema era bem simples e pequeno do que os outros, fazia com que estudantes conseguissem dissecá-lo com mais facilidade, aprendendo na prática como era o processo de criação de um sistema operacional. Segundo Tanenbaum e Woodhull (2008, p. 11): "O Unix foi projetado para ser eficiente, enquanto o Minix, foi projetado para ser legível".

Objetivos

O objetivo deste trabalho é desenvolver um estudo introdutório sobre o sistema operacional Minix, passando primeiramente por uma descrição das principais arquiteturas dos sistemas operacionais. Portanto serão apresentadas as arquiteturas do tipo Monolítico, em camadas e MicroKernel. Em seguida será feita uma análise da escolha de arquitetura implementada no Minix, suas vantagens e desvantagens.

Um estudo detalhado sobre a natureza, objetivos e características dos processos no Minix também faz parte do escopo deste trabalho. O que é um Processo? Como são criados? Qual a relação de um processo com a inicialização de um sistema operacional? Estas perguntas serão respondidas no decorrer deste estudo.

Também será apresentada o funcionamento da inicialização do Minix bem como uma breve descrição do padrão POSIX.

1 Conceituação

A seguir serão apresentados os conceitos fundamentais para o posterior estudo do Minix no capítulo subsequente.

1.1 O Kernel

O conceito de Kernel é fundamental para a compreensão das diferentes arquiteturas demonstradas a frente. O kernel ou núcleo é o principal componente de um sistema operacional, é a ponte entre as aplicações de usuário e o que ocorre em nível de máquina nos computadores.

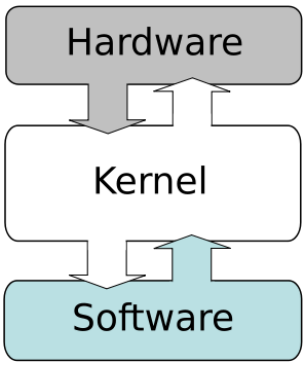

Figura 2 – Localização do Kernel

A principal responsabilidade de um Kernel é o gerenciamento de recursos em um sistema para que exista a comunicação de componentes de Hardware e Software. O Kernel é o centro de um sistema operacional, e deve proporcionar a camada de abstração de recursos em nível mais baixo, por exemplo - a camada de interação com portas de Entrada e Saída, para que uma aplicação possa usufruir de tais recursos. Estes recursos e interfaces são fornecidos geralmente por meio de chamadas de sistema ou comunicação entre processos.

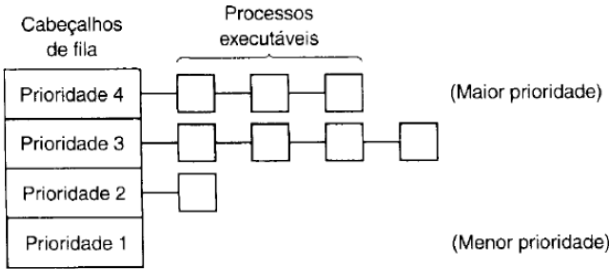

Figura 3 – Escalonador

A seguir uma lista de recursos que são primariamente e fundamentalmente gerenciados pelo Kernel:

- Processador - Parte central de um computador, responsável pelo processamento de informações. O Kernel é responsável por decidir a todo momento quais dos processos criados (por enquanto, entenda processo como um programa em execução) devem ser atendidos pelo processador durante certo intervalo de tempo. Lembre-se de que um único processador atende a apenas uma única instrução por vez.
- Memória Principal - A memória serve como uma mesa de trabalho para que o processador possa realizar suas tarefas. Ela armazena instruções e dados de programas. As instruções devem estar armazenadas de forma ordenada e de acordo com a tarefa de execução pretendida e é formada por segmentos que contém significados e funções específicas. Geralmente existem diversos programas fazendo requisições ao sistema para obter mais memória e espaço de armazenamento, isso pode fazer com que a memória total do computador não seja capaz de atender a plenitude desta demanda, e este é outro problema que é gerenciado pelo sistema operacional, além de decidir qual parte da memória cada processo pode usar.
- Entrada e Saída de Dados (E/S) - O Kernel gerencia as requisições de entrada e saída de dados gerados por aplicações dispositivos como teclados, mouses, drivers de disco, drivers USB, impressoras, monitores, adaptadores de rede e etc. O Kernel cria uma camada de abstração para o acesso a estes dispositivos. Esta camada de abstração permite que aplicações não se preocupem com os detalhes da interação com cada espécie diferente de periféricos, mas apenas com a interação padronizada com o sistema operacional.

O Kernel também deve prover métodos para sincronização e comunicação ente processos, estes métodos são chamados IPCs - *inter-process communication*. Isto é fundamental para qualquer forma de implementação modular, uma vez que funcionalidades contidas em uma parte do sistema podem ser essenciais para o bom funcionamento de outra parte. Lembrando sempre que para que exista uma hierarquia na execução das atividades é necessário um meio de comunicação entre todas as as partes de um sistema.

1.1.1 Uma nota sobre gerência de memória

O kernel tem acesso completo à memória do sistema e deve permitir que os processos acessem com segurança essa memória. Muitas vezes, isso é realizado por meio do endereçamento virtual que implementa os conceitos de paginação ou segmentação (o objetivo deste trabalho não é entrar em detalhes sobre estas definições - consulte University of Alberta (2008) para maiores detalhes). Endereçamento virtual traduz endereços físicos

em endereços virtuais, e um programa enxerga somente endereços virtuais. Um mesmo endereço de memória virtual pode ser acessado/endereçado por dois processos diferentes, mas é como se estes endereços pertencessem apenas ao escopo do programa em execução, portanto a tradução dos endereços para a memória física será diferente. Assim cada programa se comporta como se tivessem toda a memória de trabalho disponível para si, graças ao Kernel.

Em alguns casos, um endereço virtual de memória pode referenciar dados que não estão na memória principal no momento da requisição, a virtualização permite que isso ocorra sem que o programa se preocupe com este fato. Cabe ao Kernel ir até o disco e buscar o dado que não está presente na memória, servir o programa que necessita deste dado e então o programa continua sua execução normalmente.

A memória virtual também permite a criação de partições de memória em áreas disjuntas, é possível reservar uma partição para o kernel (modo Kernel) e outra para as aplicações (modo usuário). As aplicações que rodam em modo usuário não conseguem então enxergar ou acessar a área de memória destinada ao modo Kernel, protegendo então toda a área de trabalho e atuação do Kernel. Esta estrutura de organização e divisão da memória é usada amplamente e na maioria das implementações de sistemas operacionais.

1.2 Modo Kernel x Modo Usuário

Dado que um sistema operacional é responsável pela gerência dos recursos da máquina é imprescindível que o mesmo possua acesso irrestrito a todos os recursos de baixo nível, o que é garantido pelo modo kernel, enquanto os programas aplicativos devem possuir acesso limitado aos recursos afim de garantir a não interferência no processo de gestão e nas configurações do sistema, o que é garantido pelo modo usuário.

1.3 Principais Arquiteturas

É importante conhecer algumas das principais arquiteturas de sistemas operacionais que foram experimentadas ao longo dos anos. Todas possuem pontos positivos e negativos, e ao final deste estudo, será possível entender o tipo escolhido pelo Sistema Operacional Minix e os porquês desta opção.

1.3.1 Estrutura Monolítica

Quando dizemos que uma arquitetura é monolítica significa dizer que o Sistema Operacional é escrito como uma coleção de procedimentos que chamam uns aos outros sempre que necessário. É claro que cada procedimento tem uma interface bem definida em termos de parâmetros esperados e tipo de retorno concedido.

Para construir uma organização neste sentido, primeiramente um programa compila cada módulo (arquivos contendo procedimentos específicos para determinada tarefa) separadamente, e em seguida um programa lincador junta todas as partes.

Em um kernel monolítico, todos os serviços do sistema operacional residem junto com a thread principal do núcleo do sistema, portanto, também residem na mesma área de memória. Esta abordagem fornece acesso rico e poderoso ao hardware. Alguns desenvolvedores afirmam que esta organização é uma das mais simples de se implementar. As principais desvantagens de sistemas monolíticos são as dependências entre os componentes - uma falha em um driver de dispositivo pode travar o sistema inteiro.

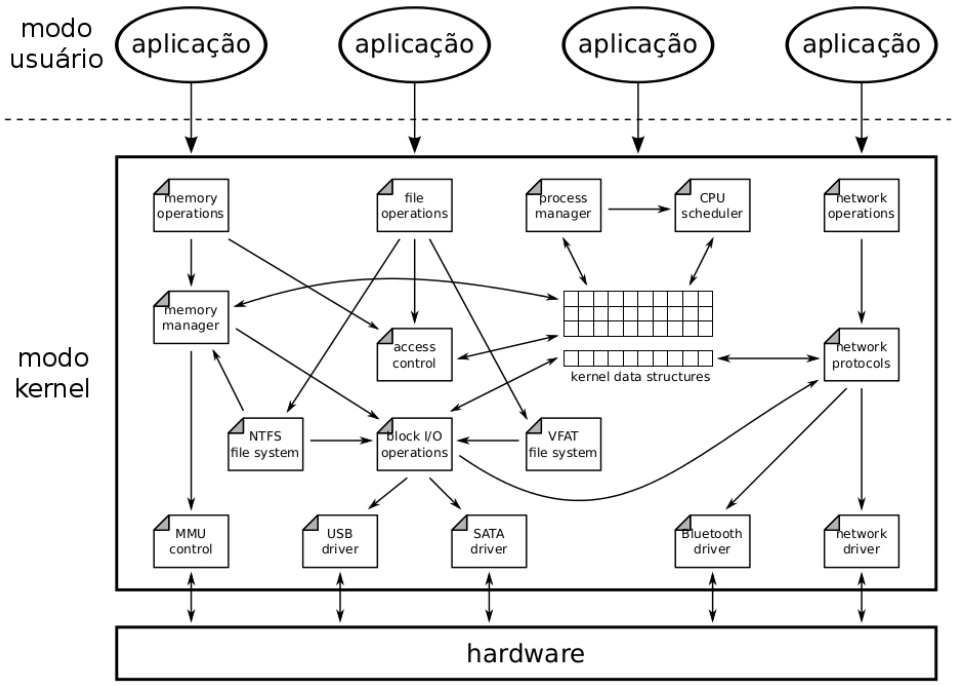

Figura 4 – Kernel monolítico (MAZIERO, 2013)

1.3.2 Sistemas Baseados em Camadas

Os componentes destes tipos de sistema são organizados em módulos que formam uma hierarquia de camadas, onde uma camada é construída em cima da outra. Cada módulo fornece um conjunto de funções que o outro módulo (de um nível superior) pode chamar. Uma camada de um nível especifico fornece serviços a camadas superiores e precisa de serviços das camadas inferiores a ela (este raciocínio é extremamente parecido com as camadas de rede dos modelos TCP/IP e OSI). Teoricamente, a construção deste tipo de sistema se dá partindo da camada mais inferior (mais próxima ao Hardware) e o desenvolvimento caminha até as camadas superiores. A divisão em camadas proporciona a modularidade, onde é possível substituir uma camada por outra sem precisar reconstruir todo o sistema, porém oferece uma performance inferior em comparação com sistemas

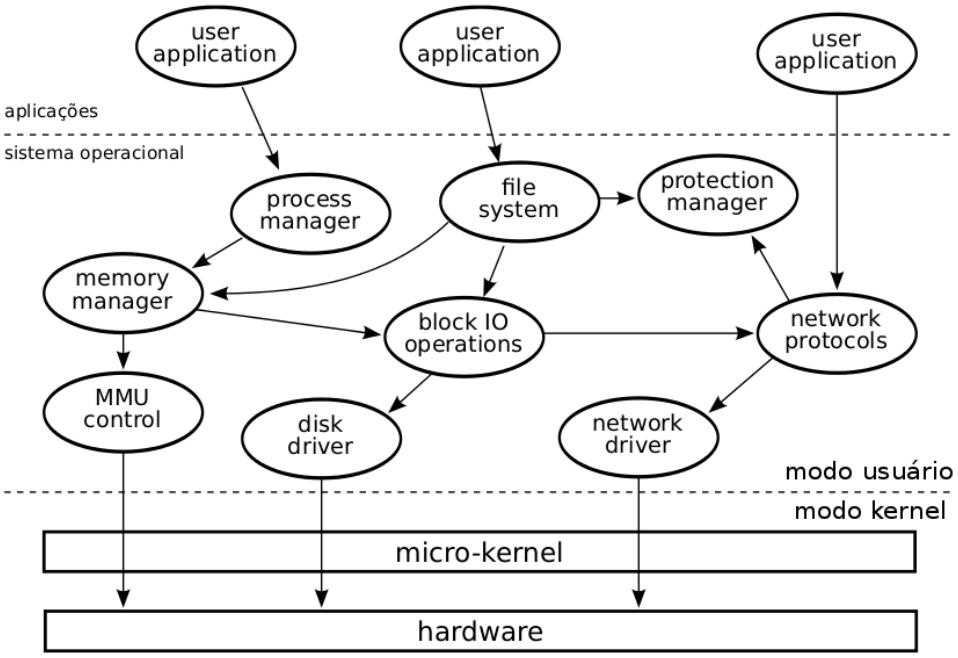

Figura 5 – Microkernel (MAZIERO, 2013)

monolíticos por exemplo, justamente pela necessidade de criar a interação funcional entre as camadas e divisões do sistema (é necessário criar um esquema de troca de mensagens muito eficiente e robusto para minimizar problemas de performance). Um exemplo de sistema que ficou conhecido por este tipo de operação foi o Multics (ver MIT - Massachusetts Institute of Technology (1972) para mais informações sobre o sistema operacional Multics).

1.3.3 Microkernel

Microkernel é o termo que descreve uma abordagem de sistema operacional em que a funcionalidade do sistema é movida para fora do núcleo, e é posta em um conjunto de programas servidores que se comunicam através de um núcleo mínimo. A ideia é deixar a menor quantidade de funções possível no "modo kernel" e, tanto quanto possível no "modo usuário".

Quando nos referimos a modo kernel, significa a área de memória estritamente reservada para a execução do kernel e suas extensões. Em contraste, o espaço do usuário é a área de memória onde todos os aplicativos de trabalho, principalmente do usuário, residem e esta área depende do modo kernel (mais detalhes na seção 1.2 - Modo Kernel x Modo Usuário deste documento).

A abordagem microkernel consiste em definir uma abstração simples do hardware, com um conjunto de primitivas ou chamadas de sistema para implementar os serviços mínimos do sistema operacional, por exemplo o gerenciamento de memória, a multita-

refa, e comunicação entre processos. Serviços "não vitais" como o de redes, por exemplo, são implementados em programas de espaço de usuário, conhecido como servidores. Microkernels são mais fáceis de manter do que kernels monolíticos pois além de reduzir a quantidade de código pertencente ao kernel trás uma maior facilidade em se isolar erros de programação e falhas de sistema. Porém o grande número de trocas de mensagens e mudanças de contexto podem tornar o sistema lento, porque elas normalmente geram mais sobrecarga do que chamadas de função simples. Portanto, em um sistema microkernel, todas as interações entre componentes e aplicações são feitas por meio das trocas de mensagens.

Um microkernel permite que a criação do modo usuário do sistema seja feita numa linguagem de alto nível, e também permite a utilização de diferentes programas gerenciadores no topo do mesmo kernel inalterado, logo é possível customizar o sistema operacional iniciando somente os componentes necessários ou escolhendo os componentes mais adequados às aplicações que serão executadas.

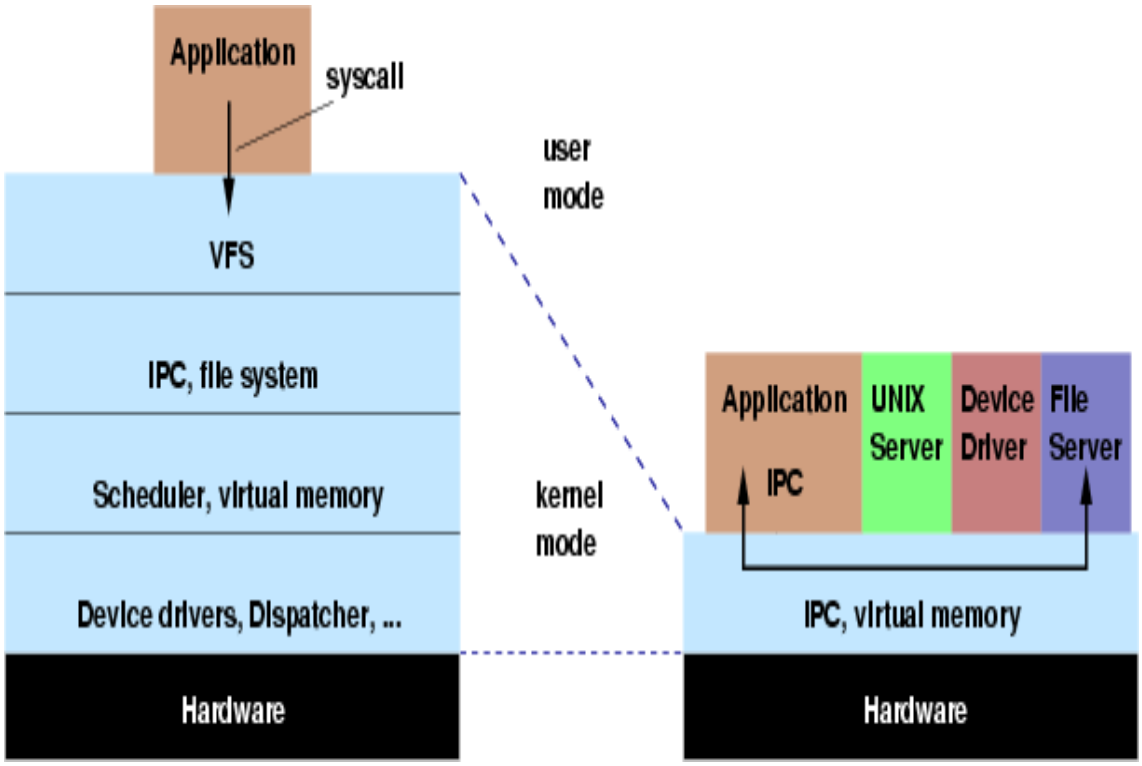

Figura 6 – Comparação entre Kernel Monolítico e Micro Kernel

1.4 Processos

Um conceito chave para o estudo de sistemas operacionais é o conceito de processo. Basicamente, um processo é um programa em execução. Associado a cada processo está o seu espaço de endereçamento, que é uma lista de memória onde há um endereço mínimo

e um máximo que o processo pode usar para endereçar e realizar suas atividades. Esses espaços de endereçamento são divididos em segmentos aonde sempre há, por exemplo, um segmento de texto que armazena as instruções que o processo irá executar, um segmento para o armazenamento de dados do programa e também um segmento de pilha que auxilia na divisão dos escopos de funções e procedimentos existentes. O espaço de endereçamento de um processo engloba todas as informações necessárias para se executar um programa.

Em um ambiente multiprogramado o sistema operacional é responsável por executar a alternância organizada da execução dos processos, já que a CPU só consegue executar um processo de cada vez, e isto se chama escalonamento. A implementação do escalonamento traz vários benefícios a um sistema computacional, entretanto não faz parte do escopo deste trabalho estudar tais conceitos.

Segundo Tanenbaum e Woodhull (2008), (devido à existência do escalonamento) um processo pode se encontrar em um dos seguintes estados:

- Em execução - neste estado o processo está sendo executado, portanto ocupando a CPU.
- Pronto - Temporariamente parado para que outro processo seja atendido, e esperando ser escolhido pelo escalonador.
- Bloqueado - Incapaz de continuar executando, é necessário que um evento externo - como por exemplo a entrada necessária de algum dado - ocorra para liberar o processo.

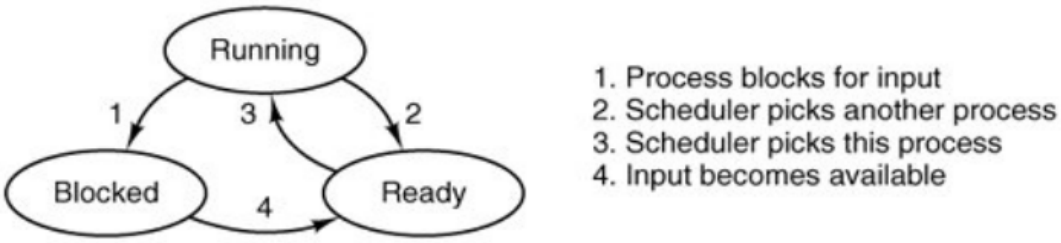

Figura 7 – Possíveis estados de um processo.

Quando o sistema operacional precisa parar a execução de um processo e iniciar outro, é necessário salvar as informações de estado do processo que será pausado, para que estas informações posteriormente possam ser recuperadas e a execução continue de forma correta. Portanto, para a execução deste tipo de atividade, o sistema operacional guarda internamente uma estrutura chamada tabela de processos, que é uma lista de estruturas referentes a processos em execução.

Logo, um processo é formado por seu espaço de endereçamento (muitas vezes chamado de *core image*) e sua entrada na tabela de processos.

Processos podem criar novos processos (que são chamados de processos filhos), que por sua vez também podem criar novos processos dando origem a uma estrutura chamada de árvore de processos. Processos podem cooperar para realizar uma tarefa em comum ou partes da mesma tarefa, e portanto precisam de uma forma para passar informações entre eles. Esta comunicação é chamada de IPC (Comunicação entre processos - *Interprocess Communication*).

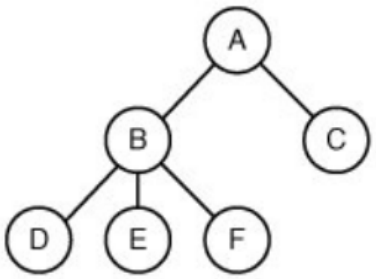

Figura 8 – Exemplificação de uma Árvore de Processos - O processo A criou dois filhos (B e C). Processo B criou três filhos (D, E e F)

Em um ambiente MINIX (objeto de nosso estudo) cada pessoa autorizada a acessar o sistema tem um número de identificação - UID (*User Identifier*) e cada processo armazena o UID do usuário que o iniciou. Um processo filho tem o mesmo UID que o seu pai. Além disso, todos os processos do sistema têm uma identificação única chamada de PID (*Process Identifier*)

1.5 POSIX

O padrão POSIX é um conjunto de normas definidas pela IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos), criadas afim de garantir a portabilidade de software à nível de código através de todos sistemas operacionais baseados em unix. Esta padronização define uma interface mínima de chamadas de sistema que todo sistema POSIX deve implementar.

Para que uma aplicação seja totalmente portável entre sistemas que seguem o padrão, deve-se cuidar em seguir o padrão POSIX também no processo de implementação.

Sistemas Operacionais como o Linux e o Minix3 apresentam conformidade com a maioria dos padrões POSIX, mas a maioria dos sistemas operacionais baseados em unix possuem extensões à biblioteca de funções básicas do C que devem ser tratadas no processo de portabilidade entre os sistemas.(MINIX, 2009).

2 Sistema Operacional Minix

O Minix 3 é um sistema operacional de código aberto que segue o padrão POSIX e possui uma arquitetura de microkernel, onde funcionam em modo kernel o tratamento de interrupções, a programação de CPU e MMU (Unidade de Gerenciamento de Memória - *Memory Management Unit*), o escalonador de processos, a gerência de comunicação entre processos e a gerência de chamadas de kernel. As demais funcionalidades do Sistema atuam como processos independentes em modo usuário.

2.1 Estrutura Interna do Minix

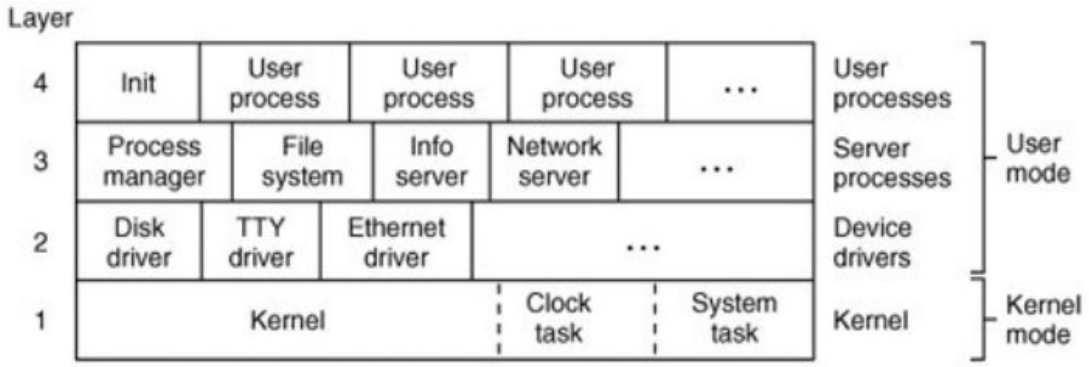

Figura 9 – Minix estruturado em quatro camadas

O Kernel, na camada inferior ou mais baixo nível do sistema, gerencia o escalonamento entre processos e as transações entre seus estados (Na seção 2.2 - Gerenciamento de Processos no Minix, será abordada a ordem de criação e inicialização de processos no sistema operacional Minix). O Kernel também gerencia a troca de mensagens entre os processos. A gerência de troca e envio de mensagens entre processos envolve a verificação do destinatário, a checagem de buffer de memória para o envio e recebimento de dados e a cópia de dados entre entre o remetente e o destinatário. Além disso o Kernel também deve dar suporte a Entrada e Saída (E/S) de dados, envolvendo o tratamento das suas interrupções - processadores modernos exigem que esse tipo de tratamento ocorra de modo privilegiado e restrito, ou seja, no modo Kernel.

Ainda na primeira camada do sistema existem dois módulos que se comportam de maneira semelhante a drivers de dispositivos - são as tarefas de relógio e de sistema. A tarefa de relógio interage diretamente com o Hardware que gera pulsos de tempo e não está acessível ao usuário, somente ao kernel. A tarefa de sistema está relacionada a uma das principais atividades do kernel, fornecer um conjunto de chamadas privilegiadas

de sistema para os dispositivos das camadas superiores como drivers e servidores, de forma que estes dispositivos possam interagir com o Kernel. Mesmo sendo compiladas no mesmo espaço de endereçamento do Kernel, as tarefas de relógio e sistema são escalonadas separadamente e possuem suas próprias pilhas para armazenar parâmetros e escopo de funções.

As três camadas acima do Kernel são tratadas por ele da mesma maneira, cada uma delas é limitada a operar em modo usuário e todas são escalonadas pelo Kernel. Nenhuma delas consegue acessar portas de E/S diretamente e também não conseguem acessar segmentos de memória que não foram alocadas para elas.

A verdadeira diferença entre estas camadas está em privilégios especiais de chamadas ao Kernel. Por exemplo, drivers da camada dois podem pedir a leitura (por meio de uma chamada de sistema) de uma porta de E/S de um dispositivo por conta própria (lembrando que é necessário um driver para cada tipo diferente de dispositivo anexado ao sistema). Analogamente, um driver também tem a permissão de pedir que um dado recém lido seja escrito em um endereço de memória de um processo diferente.

Na terceira camada estão os servidores que fornecem funções úteis aos processos da camada superior de usuário. Dois servidores são essenciais - o gerenciador de processos e o sistema de arquivos. O gerenciador de processos lida com requisições de criação, parada e sinalização de processos, atividades que alteram os estados de execução de cada processo. O sistema de arquivos lida com chamadas de sistema de arquivo que permitem a leitura ou escrita de arquivos, montagem de sistemas de arquivos e troca de diretórios. É importante lembrar que servidores não tem acesso direto à portas de entrada e saída, eles devem primeiramente fazer uma requisição aos drivers que realizarão esta tarefa e retornarão a informação desejada. Embora servidores e drivers sejam processos independentes, eles se diferenciam dos processos de usuário no sentido de que estão sempre em execução enquanto o sistema está ligado e além disso têm prioridade de execução elevada.

Finalmente, a camada 4 contém todos os processos, shells, editores, compiladores e programas de interação com o usuário. Além disso, várias daemons podem estar em execução. Um daemon é um processo de background que executa periodicamente ou sempre espera algum evento e executa uma ação em resposta, como uma trigger de Banco de Dados. De certa forma, um daemon é um servidor que é iniciado de forma independente e é executado como um processo de usuário. Mas também é possível configurar um daemon para ter uma prioridade maior do que a de um processo de usuário comum.

2.1.1 O Servidor de Reencarnação

O servidor de reencarnação é um processo especial que é inicializado após o kernel, sendo ele pai de todos os demais servidores e drivers. Quando um driver ou um processo

morre ele é coletado pelo servidor de reencarnação que checa o motivo da morte do processo reiniciando-o se necessário.

Também é tarefa do servidor de reencarnação checar constantemente o funcionamento dos drivers e servidores esperando dos mesmos uma resposta. Caso ocorra um travamento em um driver ou servidor, cabe ao servidor de reencarnação finalizar e inicializar uma nova instância destes processos.

2.1.2 Minix e a confiabilidade

Uma das maiores preocupações do professor e criador do Minix, Tanenbaum (2010), em relação as suas pesquisas sobre desenvolvimento e manutenção de sistemas operacionais, sempre foi a confiabilidade dos sistemas. Um dos maiores problemas apontados pelo professor, e que pode estar presente em qualquer projeto de desenvolvimento de software - inclusive no desenvolvimento de um sistema operacional, é o inchaço e o crescimento desmedido dos códigos fonte, o que pode causar a insustentabilidade dos projetos. Este fator aliado a falta de modularidade na criação dos sistemas, seriam os principais motivos da falta de confiabilidade e robustez dos sistemas operacionais modernos.

No ano de 2009 em uma conferência da Linux Foundation (consórcio sem fins lucrativos e mantenedor do sistema operacional Linux), o cientista da computação Linus Torvalds, criador e principal mantenedor do Kernel do Linux, afirmou que seu sistema estava se tornando enorme e inchado. (MODINE, 2009). Isto não era uma afirmativa de algum concorrente do Linux, mas do seu próprio criador concluindo que isto era um problema e merecia atenção. O fato é que em 1994 a versão 1.0 do Linux foi lançada com 176.250 linhas de código, a versão atual 2.9 já ultrapassa a marca de 11 milhões de linhas de código. E é importante lembrar que estes números são pequenos em relação ao Windows Vista, por exemplo, que possui aproximadamente 50 milhões de linhas de código. Não há ninguém que entenda o funcionamento completo do sistema operacional Windows, e talvez Linux Torvalds endenta tudo o que se passa no Linux.

Portanto a proposta do professor Tanenbaum é a de repensar os conceitos de sistemas operacionais, e não se deixar levar pela premissa de que com recursos de Hardware cada vez mais poderosos não há a necessidade de se preocupar com tamanho e aspectos relacionados a modularização dos softwares. Estudos feitos em empresas de desenvolvimento de softwares, onde existe um controle rigoroso de qualidade e testes preventivos, apontam que a cada 1000 linhas de código, três falhas (em inglês chamados de bugs) acontecem em média. Portanto, quanto menos linhas de código menos erros e falhas estarão presentes. Em 2010 Tanenbaum fez um comparativo neste segmento entre a quantidade de falhas encontradas no núcleo do Linux e no núcleo do Minix, e ao todo foram encontradas 18000 falhas contra 18 falhas, respectivamente.

Verdadeiramente é possível encontrar benefícios resultantes da escolha da arquitetura feita e aplicada no Minix - o Microkernel, apresentado na seção anterior. Como já foi bem enfatizado uma menor quantidade de linhas de código resultam em menos problemas, porém há mais do que isto. É possível apontar a maior tolerância a falhas que este tipo de sistema apresenta. Por exemplo, um módulo de driver que trava enquanto o sistema está em operação não corrompe o sistema por completo, pois o núcleo do sistema estará executando em uma área de memória separada e se encarregará de reiniciar os serviços problemáticos, como veremos em capítulos posteriores. Em sistemas monolíticos a falha de um driver pode prejudicar o sistema inteiro, visto que fazem parte do mesmo módulo do sistema operacional, não havendo a capacidade de auto reconstrução. Outro fator interessante é o isolamento de programas maliciosos como os vírus. Na arquitetura do Minix um vírus não consegue se espalhar entre módulos distintos pois estão isolados e não é possívelo acesso ou o endereçamento entre camadas sem a verificação e intervenção do núcleo, adicionando mais um ponto de confiabilidade ao sistema.

2.2 Gerenciamento de Processos no Minix

Processos no minix seguem o modelo geral descrito na seção Processos, do Capítulo 1. Foi visto que processos podem sucessivamente criar processos filhos gerando assim uma árvore de processos. Para o completo entendimento dos processos em um ambiente Minix, será descrito a seguir como acontece a criação dos processos desde a inicialização do sistema.

2.2.1 Inicialização do Minix

Quando o sistema é inicializado, a BIOS lê o primeiro setor de um disco bootável. Então a BIOS, copia este setor para uma região fixa de memória e executa as instruções ali encontradas. Discos Rígidos contém partições, e o seu primeiro setor contém um programa chamado masterboot e a tabela de partições do sistema. O masterboot lê a tabela de partições, carrega e executa o primeiro setor da primeira partição marcada como ativa. Este setor contém um bloco chamado de boot (bootblock) que carrega o programa chamado bootloader. É necessário ter em mente que independente do esquema de partições, o controle sempre será passado para o bootblock que carregará o bootloader.

No Minix, o bootloader se encontra por padrão no diretório /boot/boot, e é comumente chamado de Programa Monitor, que além de carregar o Sistema Operacional possui várias outras opções e parâmetros de uso. Este programa não é parte do Sistema Operacional, mas é um programa inteligente, capaz de usar as estruturas de dados do sistema de arquivo para encontrar a imagem atual de sistema operacional - *system image*.

A imagem de boot contém o microkernel, o gerenciador de processos, e o sistema

de arquivos. Também possui o servidor de reencarnação, drivers de memória, de terminais, de log e o init. O servidor de reencarnação é um servidor da camada três que inicia, e se necessário, reinicia drivers de dispositivos. Eventualmente, se um driver de dispositivo falhar durante sua operação, o servidor de reencarnação é responsável por verificar esta falha, eliminar o processo do driver e inicializar uma nova instância do driver, fazendo com que o sistema fique muito mais confiável e tolerante à falhas - esta funcionalidade não está presente na maioria dos sistemas operacionais.

Durante o processo de inicialização, o Kernel executa as tarefas de sistema e relógio (que também estão na camada 1 do sistema), o gerenciador de processos e o sistema de arquivos. O gerenciador de processos e o sistema de arquivos cooperam inicializando outros serviços e drivers e após todas estas rotinas estarem carregadas em memória elas são bloqueadas a espera do init, o primeiro processo de usuário a ser executado.

Init é o primeiro processo de usuário e também o último processo carregado como parte da imagem de boot. O Servidor de Reencarnação é feito pai de todos os processos inicializados da imagem de boot, isto é feito no intuito de informá-lo caso haja algum problema em algum driver ou serviço e este precise ser reiniciado. De acordo com o padrão Unix-like, init executa o shell script localizado no arquivo /etc/rc. Este script roda drivers e servidores que não fazem parte da imagem de boot. Todos esses programas que são inicializados a partir de init serão filhos do processo init que tem PID 1. Um dos programas inicializado por init é o Service, que é a interface de usuário para o Servidor de Reencarnação. O Service inicializa o programa cmos, necessário para leitura do relógio, e o servidor de informações.

Uma importe função do script rc é verificar problemas no sistema de arquivos que pode ter resultado de uma falha anterior do sistema - para isto é necessário que o sistema de arquivos tenha suporte a algum formato de journaling. Finalmente o init lê o arquivo /etc/ttytab, que lista todos os dispositivos terminais potenciais. Os dispositivos que podem ser utilizados como terminais de login.

Quando um usuário digita um nome para login, o programa /usr/bin/login é chamado com o nome de usuário como argumento. O login determina se é necessária uma senha para o usuário especificado, e em caso afirmativo solicita e verifica a senha. Após um login bem-sucedido, login executa o shell do usuário que é por padrão /bin/sh, mas um outro shell pode ser especificado no arquivo /etc/passwd. O shell espera por comandos que devem ser digitados e em seguida gera novos processos para cada comando. Desta forma, as shells são os filhos de init, os processos de inicialização são os netos de init, e todos os processos de usuário no sistema são parte de uma única árvore. De fato, exceto as tarefas compilados no kernel e o gerenciador de processos, todos os processos, tanto processos do sistema e processos de usuário, formam uma árvore.

3 Conclusão

Como pode ser observado, o minix 3 é um sistema operacional robusto que atinge seu propósito em entregar um sistema mais confiável, compacto e organizado. Sistemas operacionais monolíticos têm sofrido com o crescimento desmedido enquanto o kernel do Minix continua compacto e com poucas linhas de código, fazendo com que a manutenção e a tarefa de encontrar problemas se torne mais fácil e eficiente. Em sistemas monolíticos, drivers de dispositivos estão no espaço de endereçamento do kernel, conforme estudado. Isto significa que quando novos periféricos são instalados, softwares desconhecidos ou até mesmo maliciosos podem ser inseridos no kernel, o que pode trazer prejuízos imprevisíveis como, por exemplo, a derrubada de um sistema crítico. No Minix, cada driver está incluído no espaço de endereçamento de usuário, portanto não podem executar instruções privilegiadas por conta própria sem pedir autorização ao kernel - tornando o sistema mais seguro.

É importante lembrar que o Minix possui funcionalidades interessantes e únicas como, por exemplo, o servidor de reencarnação - capaz até de recuperar um driver que eventualmente se encontre em loop infinito - porém, o Minix também apresenta pontos negativos. Uma das principais desvantagens do esquema monolítico é gerenciar toda forma de IPC por meio da intensa troca de mensagens assíncronas. A troca de mensagens envolve aspectos muito mais complexos do que apenas chamadas de rotinas específicas. A troca de mensagens envolve verificação de permissão, limitação e preparo de um buffer temporário, sincronia entre processos e outras questões que se não forem resolvidos de forma performática trarão sérios problemas a todo o sistema. Um dos fatores que mais prejudicam a difusão e utilização do Minix em maior escala é a portabilidade de aplicações e drivers, já que a comunidade de mantenedores do sistema operacional não é muito grande a tarefa de portar aplicações para o sistema se torna difícil, pois é um processo naturalmente demorado.

O sistema operacional Minix juntamente com a documentação feita pelo professor Andrew S. Tanenbaum formam uma fonte de conhecimento aberta e valiosa, portanto, o estudo de sistemas operacionais tendo o Minix como base contribui para um aprendizado mais eficiente. Depois deste estudo, é possível prosseguir na pesquisa sobre sistemas operacionais, até mesmo se o intuito for construir um novo sistema, entretanto é necessário entender que o assunto é muito extenso, sendo necessário muita dedicação e tempo para que resultados saiam do papel e transformem uma proposta em realidade.

Referências

MAZIERO, C. A. *Sistemas Operacionais*: Conceitos e mecanismos. [S.l.], 2013. Disponível em: <http://dainf.ct.utfpr.edu.br/~maziero/lib/exe/fetch.php/so:so-cap01.pdf>. Acesso em: 8.6.2013. Citado 3 vezes nas páginas 5, 10 e 11.

MINIX. *MinixWiki Developers Guide*: Posix and minix 3. [S.l.], 2009. Disponível em: <http://wiki.minix3.org/en/DevelopersGuide/PosixAndMinix>. Acesso em: 17.6.2013. Citado na página 15.

MIT - MASSACHUSETTS INSTITUTE OF TECHNOLOGY. *Multics*: The first seven years. [S.l.], 1972. Disponível em: <http://web.mit.edu/saltzer/www/publications/f7y/f7y.html>. Acesso em: 23.4.2013. Citado na página 11.

MODINE, A. *Linus calls Linux 'bloated and huge'*. [S.l.], 2009. Disponível em: <http://www.theregister.co.uk/2009/09/22/linus_torvalds_linux_bloated_huge/>. Acesso em: 10.6.2013. Citado na página 18.

TANENBAUM, A. S. *MINIX 3*: a modular, self-healing posix-compatible operating system. Bruxelas, Bélgica, 2010. Vídeo (56 min). Disponível em: <http://www.youtube.com/watch?v=bx3KuE7UjGA>. Acesso em: 11.6.2013. Citado na página 18.

TANENBAUM, A. S.; WOODHULL, A. S. *Sistemas Operacionais*: Projeto e implementação. 3. ed. [S.l.]: Bookman, 2008. ISBN 9788577800575. Citado 2 vezes nas páginas 6 e 13.

UNIVERSITY OF ALBERTA. *Understanding Memory*. [S.l.], 2008. Disponível em: <http://www.ualberta.ca/CNS/RESEARCH/LinuxClusters/mem.html>. Acesso em: 12.5.2013. Citado na página 8.