Fundação Presidente Antônio Carlos de
Conselheiro Lafaiete
Engenharia de Computação

LINUX SERVER

Uma análise sobre o sistema

Paulo Henrique de Oliveira Rodrigues

Conselheiro Lafaiete, 26 de novembro de 2020

Paulo Henrique de Oliveira Rodrigues

LINUX SERVER
Uma análise sobre o sistema

Trabalho apresentado na Fundação Presidente Antônio Carlos (FUPAC) - Conselheiro Lafaiete , como um dos pré-requisitos para a aprovação na disciplina de Sistemas Operacionais no curso de Bacharel em Engenharia de Computação.

Conselheiro Lafaiete
26 de novembro de 2020

Agradecimentos

Agradeço a Deus por me iluminar nos momentos difíceis, dando força na longa caminhada.
Em especial minha esposa, por me apoiar incondicionalmente e por sempre confiar em meu potencial me incentivando a fazer sempre o melhor e a meu filho pelas risadas calorosas que me renovam sempre o ânimo e me lembram dos meus objetivos.
Ao meu orientador Alex Vitorino por sua paciência e sempre propondo novos desafios que são de grande contribuição em relação ao meu aprendizado e com ensinamentos importantes para consolidação deste trabalho.
Enfim em todos que acreditaram no meu sonho.

"A mão queimada ensina melhor.
Depois disso o conselho sobre o fogo
chega ao coração."– Gandalf - O Cinzento
(J.R.R. Tolkien)

Resumo

O objetivo deste presente trabalho é a consolidação dos conhecimentos adquiridos durante a realização da disciplina de Sistemas Operacionais, dando enfoque aos sistemas baseados em *Linux* utilizados em servidores, salientando suas utilizações e o seu mercado de atuação. O *Linux* é um sistema operacional que vive em um crescimento contínuo e é amplamente usado ele está tanto em sensores a supercomputadores, e podemos vê-lo sendo usados em espaçonaves, automóveis, *smartphones*, relógios e muitos outros dispositivos em nossa vida cotidiana.

Em especial o sistema *Linux* é um sistema de código aberto o que significa que é possível executá-lo para qualquer propósito, estudar seu funcionamento e modificá-lo se assim desejar, ou realizar cópias para terceiros dando total liberdade para sua comunidade.

Ele também opera a maior parte da Internet, todos os 500 maiores supercomputadores do mundo e as bolsas de valores do mundo. Estes funcionam em uma variação do *Linux* preparada para um grande fluxo de tratamento de dados, podendo rodar vários serviços simultaneamente, esta versão é a *Linux* para servidores ou *Linux Server*.

Palavras-chaves: Linux, Servidores, Sistema Operacional.

Lista de ilustrações

Figura 1 – Onde o sistema operacional se encaixa. [1] 1
Figura 2 – Um *pipeline* com três estágios. [1] 4
Figura 3 – Uma *CPU* superescalar. [1] 5
Figura 4 – Uma hierarquia de memória típica. Os números são apenas aproximações. [1] 5
Figura 5 – As memorias cache em um processador moderno. 6
Figura 6 – Do processador a RAM. 7
Figura 7 – Visão de um disco rigido. 8
Figura 8 – Estados do processo. [1] 13
Figura 9 – Comportamento de processos. [1] 17
Figura 10 – Divisão da memória usando *BuddyHeap*.[2] 20
Figura 11 – Um sistema de diretório em nível único contendo quatro arquivos. [1] . 25
Figura 12 – Um sistema hierárquico de diretórios. [1] 25
Figura 13 – Sistema de arquivos contendo um arquivo compartilhado. [1] 27
Figura 14 – (a) Armazenamento da lista de blocos livres em uma lista encadeada. (b) Um mapa de bits. [1] 28
Figura 15 – As cotas são relacionadas aos usuários e monitoradas em uma tabela de cotas. [1] 29

Lista de abreviaturas e siglas

ARM	Acorn RISC Machine
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Process Unit
cd	Change directory
DVD	Digital Versatile Disc
EEPROM	Electrically Erasable PROM
EXT3	Third extended filesystem
EXT4	Fourth extended filesystem
E/S	Entrada e saída
FAT	File Allocation Table
FreeBSD	Free OS descended from the Berkeley Software Distribution
fsck	File system consistency check
GNU	GNU's Not Unix
GNU GPL	GNU General Public License
GUI	Graphical User Interface
HD	Hard Disk
HFS	Hierarchical File System
HPFS	High Performance File System
IBM	International Business Machines Corporation
JFS	Journaled File System
ls	List space
MIT	Massachusetts Institute of Technology
MP3	MPEG-1/2 Audio Layer 3

MS-DOS	Microsoft Disk Operating System
MULTICS	MULTiplexed Information and Computing Service
NTFS	New Technology File System
OS	Operating System
OS X	Operating Systems number 10
PC	Personal Computer
PID	Process Identifier
PPID	Parent Process Identifier
PROM	Programmable Read Only Memory
pwd	Print working directory
RAM	Random Access Memory
ROM	Read Only Memory
SO	Sistema Operacional
UFS	Unix File System
XFS	Extended Filesystem
X86	Processadores baseados no Intel 8086

Sumário

1 INTRODUÇÃO 1
1.1 Revisão sobre hardware de computadores 4
1.1.1 Processadores 4
1.1.2 Memória 5
1.1.3 Discos rígidos 8
1.1.4 Dispositivos de E/S 8
1.1.5 Barramentos 9

2 SISTEMA OPERACIONAL 10
2.1 O zoológico dos sistemas operacionais 11
2.1.1 Sistemas operacionais de computadores de uso pessoal 11
2.1.2 Sistemas operacionais de servidores 11
2.1.3 Sistemas operacionais embarcados 11

3 PROCESSOS E THREADS 12
3.1 Processos 12
3.1.1 Criando processos 13
3.1.2 Término de processos 14
3.2 Threads 16
3.3 Escalonamento 17
3.3.1 Categorias de algoritimo de escalonamento 17

4 GERENCIAMENTO DE MEMÓRIA 19
4.1 Gerência de memória física 19
4.2 Memória Virtual 21
4.3 Swapping e paginação 22

5 SISTEMAS DE ARQUIVOS 23
5.1 Gerenciamento de diretório 24
5.2 Gestão de espaço livre 28
5.3 Cotas de disco 29

Conclusão 30

Referências 31

1 Introdução

Um computador moderno consiste em um emaranhado de peças que contém um ou mais processadores, alguma memória principal, alguma memória secundaria, interfaces de rede diversos periféricos como impressoras, teclado, mouse, monitor e vários outros dispositivos de entrada e saída. Podemos dizer que este é um sistema complexo, para realizar a desafiadora maratona que é compreender como cada parte funciona e gerenciar com maestria esses componentes é um grande desafio [1].
Para isso os computadores modernos são equipados com um SO esse dispositivo de *software* tem a função de fornecer uma plataforma simples e limpa para o usuário de forma a ajuda-lo nas entradas e saídas de dados. Em uma visão simplista podemos ver na figura 1 onde o SO se encontra em relação entre *hardware* e o usuário [1].

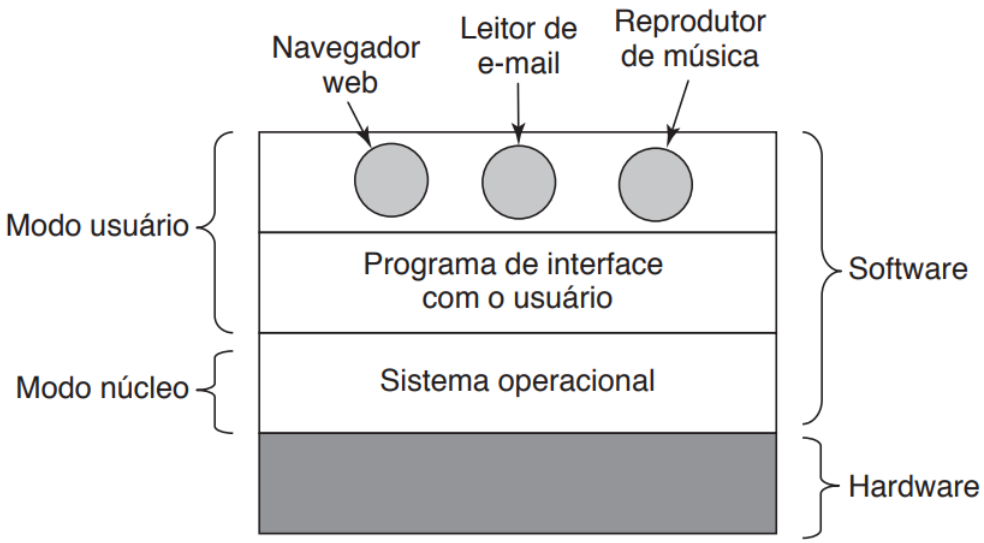

Fig. 1. Onde o sistema operacional se encaixa. [1]

Inicialmente a ideia que temos de um SO é a visão que temos dos ditos sistemas operativos que temos conhecimento que podem ser *Windows*, *Linux*, *FreeBSD*, ou *OS X* mas normalmente a forma de interagir com diretamente com o sistema é através de terminais comumente conhecidos como shell (interpretadores de comando) isto quando baseado em texto ou *GUI (Graphical User Interface)* quando em modo gráfico [1].
Um sistema operacional é projetado para ocultar as particularidades de *hardware* (ditas "de baixo nível") e assim criar uma máquina abstrata que fornece às aplicações serviços compreensíveis ao usuário (ditas "de alto nível") [3].
Assim o sistema trabalha em dois estados o modo núcleo e o modo usuário. Sendo que no modo núcleo (também chamado modo supervisor ou *Kernel mode*). O sistema tem acesso completo aos recursos seja de ao *hardware* ou *software* e pode executar qualquer instrução

que a máquina for capaz de executar [1], [4].
Quando o sistema está em modo *kernel* é considerado que as execuções são de uma fonte confiável e, portanto, pode executar quaisquer instruções e fazer referência a quaisquer endereços de memória (ou seja, locais na memória). O *kernel* tem total controle sobre o sistema e trata todos os outros *softwares* como programas não confiáveis, assim todas as operações em modo usuário que necessitem alterar o sistema solicitam ao uso do *kernel* por meio de uma chamada de sistema para executar instruções privilegiadas, como criação de processos ou operações de entrada / saída [1], [4].

Neste trabalho iremos trabalhar com o sistemas baseados em *Linux* para servidores mas é importante entender um pouco da evolução desse sistema.
Em meados da década de 60 uma iniciativa conjunta do *MIT*, da *Bell Labs* e da *General Electric* decidiram embarcar no desenvolvimento de um "computador utilitário", isto é, uma máquina que daria suporte a algumas centenas de usuários simultâneos em pouco tempo nasce o projeto MULTICS (Serviço de Computação e Informação Multiplexada) [1].
O MULTICS foi projetado para ser um sucesso com suporte para centenas de usuários em uma máquina apenas um pouco mais poderosa do que um PC baseado no 386 da *Intel*. Mas transformá lo em um produto final de fácil comercialização não foi amarga realidade [1].
A *Bell Labs* abandonou o projeto, e a *General Electric* abandonou completamente o negócio dos computadores. Entretanto, o *MIT* persistiu e finalmente colocou o MULTICS para funcionar.E foi instalado por mais ou menos 80 empresas e universidades importantes mundo afora [1].
Um dos cientistas da *Bell Labs* que havia trabalhado no projeto MULTICS, Ken Thompson, decidiu escrever uma versão despojada e para um usuário do MULTICS. Esse trabalho mais tarde desenvolveu-se no sistema operacional UNIX, que se tornou popular no mundo acadêmico, em agências do governo e em muitas empresas [1].
Em 1987, Andrew Tanenbaum lançou um pequeno clone do UNIX, chamado MINIX, para fins educacionais. Em termos funcionais, o MINIX é muito similar ao UNIX [1].
Em 1991 Linus Torvalds começou um projeto inicialmente um emulador de terminal que era utilizado para acessar os servidores em UNIX da universidade Helsinki. Ele escreveu o código para especificamente para o *hardware* que utilizava um computador com um processador 80386 ele realizou o desenvolvimento no minix usando o *GNU C compiler* [5], [6], [7].
O *Linux* também é distribuído sob uma licença de código aberto. O código aberto segue estes locatários principais:

- A liberdade de executar o programa, para qualquer propósito.
- A liberdade de estudar como o programa funciona e alterá-lo para que ele faça o que

você deseja.

- A liberdade de redistribuir cópias para que você possa ajudar seu vizinho.
- A liberdade de distribuir cópias de suas versões modificadas para terceiros.

Esses pontos são cruciais para entender a ideia por trás do *Linux*. O *Linux* se transformou em um sistema de fácil acesso. Com a grande liberdade de se poder modificar o sistema ele proporcionou a criação de diversas distribuições uma vez que qualquer usuário pode criar uma que atenda a suas necessidades [8].
Nos próximos sessões iremos discutir sobre o sistema e aprofundar no *Linux* para assim entender, suas funcionalidade, modo de funcionamento e quais suas principais atuações.

1.1 Revisão sobre hardware de computadores

1.1.1 Processadores

A *CPU* (*Central Process Unit* - no portugues Unidade Central de Processamento) muitas vezes é considerado o cérebro do computador ela é responsável por receber as instruções da memória e processá las, decodificando-as e executando todos os processos em fila de execução num ciclo que é repetido até que o programa termine [1].
Assim os programas são executados. Cada *CPU* é presa a sua arquitetura assim não conseguindo decodificar instruções de arquiteturas diferentes como *ARM* para X86 ou vice e versa. A velocidade de cálculo das *CPU's* é muito maior que o tempo para executar uma instrução registradores internos são utilizados para armazenamento de variáveis e resultados temporários [1].
O SO deve conhecer absolutamente todos os processos destinados ao processador e também todos os registradores gerados pois quando a *CPU* realiza uma multiplexação ela interrompe o programa em execução para recomeçar outro. Assim faz-se necessário que o SO tem que salvar todos os registradores de maneira que eles possam ser restaurados quando o programa for executado mais tarde. Muitas *CPU's* modernas têm recursos para executar mais de uma instrução ao mesmo tempo [1].
Por exemplo, uma *CPU* pode ter unidades de busca, decodificação e execução separadas, assim enquanto ela está executando a instrução não, poderia também estar decodificando a instrução n + 1 e buscando a instrução n + 2. Uma organização com essas características é chamada de *pipeline* como na figura 2 abaixo [1].

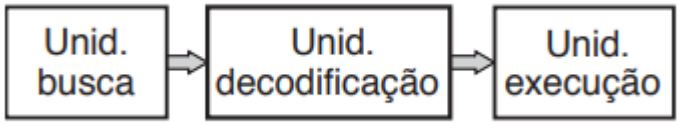

Fig. 2. Um *pipeline* com três estágios. [1]

Ainda mais avançada que um projeto de pipeline é uma *CPU* superescalar, mostrada na Figura 3. Nesse projeto, unidades múltiplas de execução estão presentes. Uma unidade para aritmética de números inteiros, por exemplo, uma unidade para aritmética de ponto flutuante e uma para operações booleanas. Duas ou mais instruções são buscadas ao mesmo tempo, decodificadas e jogadas em um *buffer* de instrução até que possam ser executadas. Tão logo uma unidade de execução fica disponível, ela procura no buffer de instrução para ver se há uma instrução que ela pode executar e, se assim for, ela remove a instrução do *buffer* e a executa [1].

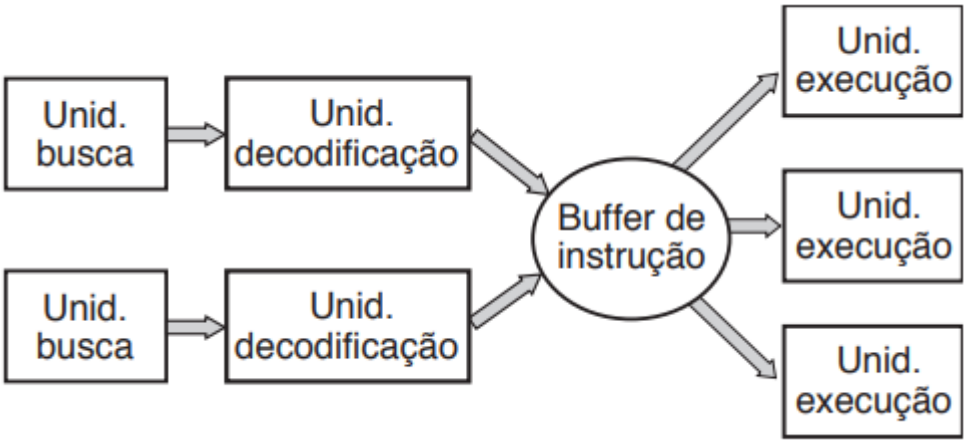

Fig. 3. Uma *CPU* superescalar. [1]

Um *thread* é um tipo de processo leve, mais para o SO cada *thread* é como uma *CPU* separada, considere um sistema com duas *CPU's* efetivas, cada uma com dois threads. O sistema operacional verá isso como quatro *CPU's* chamamos isso de *multithreading* [1].

1.1.2 Memória

A memória é a capacidade de adquirir, armazenar e recuperar informação. essa é uma definição que serve tanto para o meio biológico como para o meio artificial, podemos dizer que a memória é tão importante quanto o processador e que é um dos principais componentes do computador. Com a função de armazenar a informação antes dela seguir para o processador a memória segue uma topologia que define sua utilização pelo SO, as camadas superiores têm uma velocidade mais alta, capacidade menor e um custo maior, o que se altera nas camadas inferiores que têm velocidade mais baixa, capacidade maior e um menor custo [1], [3].

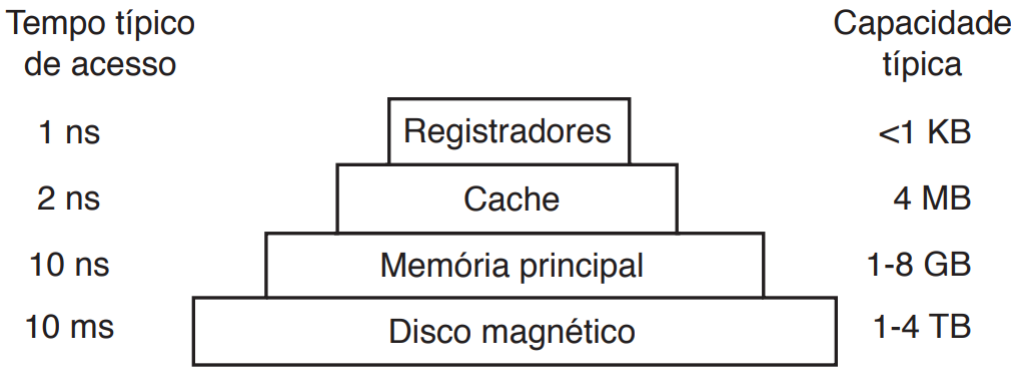

Fig. 4. Uma hierarquia de memória típica. Os números são apenas aproximações. [1]

Na camada superior temos os registradores que são feitos do mesmo material do processador e possuem velocidade similar a deles, posteriormente temos o Cache que é principalmente controlada pelo hardware e pode se separar em até três níveis cada nível mais lento e maior que o anterior. O cache é erroneamente confundido com o buffer do processador mas basicamente ele tem por função servir como um espaço para informação de rápido acesso por exemplo o cache L1 é dividido em memória de instrução e memória para dados. Com isso, o processador vai direto à memória de instrução, se estiver buscando uma instrução, ou vai direto à memória de dados, se estiver buscando um dado o cache L2 possui uma capacidade de armazenamento maior e é um pouco mais lento que o L1 ele serve para armazenar as informações antes delas irem para o cache L1 e assim sucessivamente o cache L3 diferente dos anteriores normalmente se apresenta fora do núcleo do processador e é compartilhado pelos núcleos como podemos ver na figura 5 [1], [3].

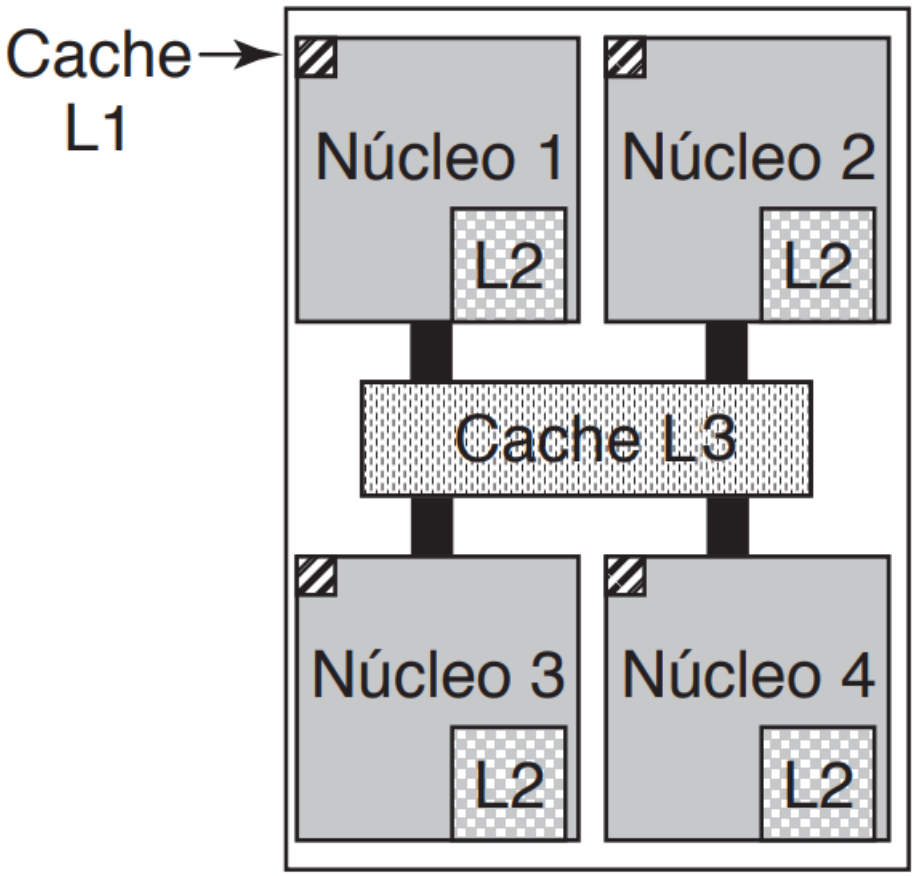

Fig. 5. As memorias cache em um processador moderno.

Logo em seguida vem a memória RAM (Random Access Memory — em portugues memória de acesso aleatório)Todas as requisições da CPU que não podem ser atendidas pela cache vão para a memória principal. Essa memória é responsável por realizar a leitura dos conteúdos quando requeridos, ou seja, de forma não-sequencial e guardar a informação de forma rápida para que seja lida cache assim como manter informações da cache [1], [3].

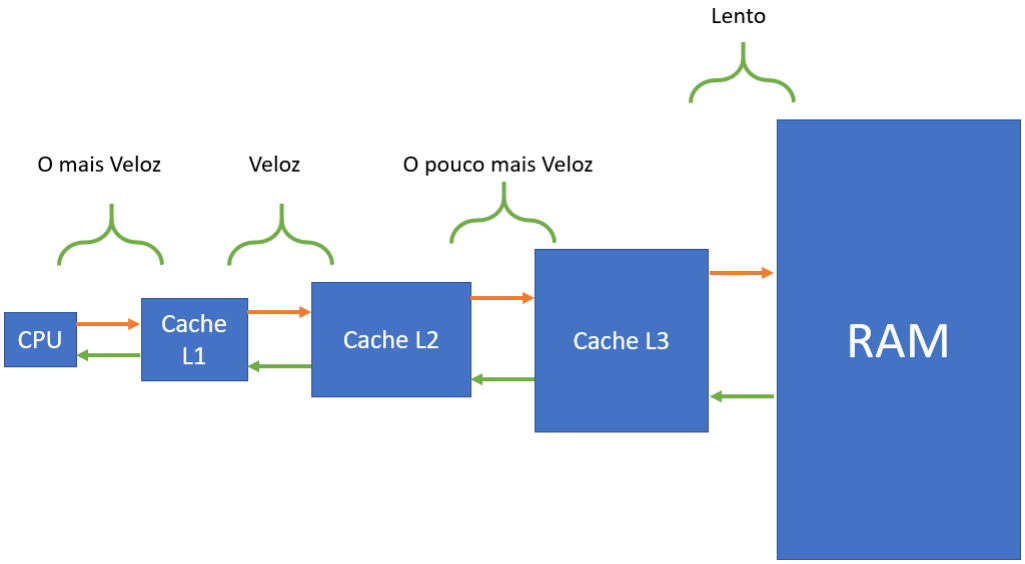

Fig. 6. Do processador a RAM.

A ROM (Read Only Memory — em portugues memória somente de leitura) é programada na fábrica e não pode ser modificada depois. Ela é rápida e barata. Em alguns computadores, o carregador (bootstrap loader) usado para inicializar o computador está contido na ROM [1], [3].

A EEPROM (Electrically Erasable PROM — em portugues ROM eletricamente apagável) e a memória flash também são não voláteis, mas, diferentemente da ROM, podem ser apagadas e reescritas. No entanto, escrevê-las leva muito mais tempo do que escrever em RAM, então elas são usadas da mesma maneira que a ROM, apenas com a característica adicional de que é possível agora corrigir erros nos programas que elas armazenam mediante sua regravação [1], [3].

A memória flash é um tipo particular de EEPROM que vem se tornando popular e bastante comum e sendo utilizada fortemente em equipamentos portáteis. Utilizada em dispositivos como câmeras e reprodutores de música substitui o disco rígido por manter as informações mesmo quando não está conectado diretamente a energia. Podemos dizer que a memoria flash esta entre a memora RAM e o discos rigidos no quesito velocidade [1], [3].

A CMOS (Complementary Metal Oxide Semiconductor - em portugues Semicondutor de óxido de metal complementar) é uma memória volátil e muito utilizada para armazenar a data e hora em sistemas, ela normalmente é alimentada por uma pequena bateria e seu consumo é tão baixo que a bateria que é colocada de fábrica pode durar anos. Ela armazena informações básicas do sistema como data e hora ou por exemplo qual disco deve ser iniciado primeiro para iniciar o sistema operacional [1], [3].

1.1.3 Discos rígidos

Os discos rígidos mais popularmente conhecidos como HD *(Hard Disk)* é o método mais utilizado para armazenamento de dados pelo computador. utilizando-se de braços metálicos posicionados acima de um disco metálico este gera um pulso eletro magnético para gravar a informação no prato metálico em formato de disco. Tecnologia de desenvolvida em 1956 contava apenas com 5 megabytes de tamanho de armazenamento hoje existem discos com capacidade de armazenamento maior que 10 terabytes [1], [3].
Normalmente o sistema operacional é gravado neste disco por ele não ser volátil, ou seja não se apaga mesmo quando o computador está desligado [1], [3].

Fig. 7. Visão de um disco rigido.

1.1.4 Dispositivos de E/S

O sistema operacional não trabalha apenas com CPU e memórias ele tem que gerenciar outros tipos de hardware estes equipamentos em questão tem funções específicas mas basicamente ele trabalham fazendo o *input* e *output* de informações no sistema e são conhecidos como dispositivos de E/S. podemos dizer que estes dispositivos possuem duas características básicas um controlador e o dispositivo em si [1], [3].
O controlador ele proporciona a comunicação entre o SO e o dispositivo em si, como cada dispositivo possui características particulares que podem ser influenciadas pelo fabricante ou tecnologia empregada assim se faz necessário que seja disponibilizado para o sistema *software* auxiliares que ajudam o SO a comunicar com o controlador esse *software* são chamados de Drivers [1], [3].
O dispositivo em si normalmente possui padronização de saídas físicas para que as conexões com o computador sejam simplificadas e por exemplo que uma saída sata seja idêntica

independente do fabricante ou da tecnologia empregada para a construção do dispositivo [1], [3].

1.1.5 Barramentos

A placa mãe é conhecida por comportar todos os periféricos do computador seja as memórias ou a CPU mais sua principal funçao é comportar os barramentos. O barramento é a linha de comunicação entre esses dispositivos. Normalmente os barramentos são divididos por funções específicas ou comunicações específicas como barramento de memória ou de dispositivos de E/S [1], [3].

Barramento de dados – como o próprio nome já deixa a entender, é por este tipo de barramento que ocorre as trocas de dados no computador, tanto enviados quanto recebidos [1], [3].

Barramento de endereços – indica o local onde os processos devem ser extraídos e para onde devem ser enviados após o processamento [1], [3].

Barramento de controle – atua como um regulador das outras funções, podendo limitá-las ou expandi-las em razão de sua demanda [1], [3].

Barramentos de entrada e saída – atua fazendo a ligação com dispositivos de E/S [1], [3].

Barramento de memória – atua diretamente na troca de informação da memória e um dos aspectos fundamentais quanto a esse barramento e a se tratando da quantidade de bits que ele é capaz de levar por vez. Um ótimo exemplo para elucidar isso é em relação à placas de vídeo [1], [3].

2 Sistema Operacional

Mas afinal o que é um sistema operacional?
É difícil entender o que é um sistema operacional pois talvez a única coisa que podemos afirmar é que é um *software* que trabalhe em modo núcleo (e por vezes até isso não é uma completa verdade), para tal precisamos entender que o SO tem duas funções bases que é fornecer uma abstração do *hardware* para programadores de aplicativos e usuários em geral, e o gerenciamento dos recursos de *hardware*.
Em geral o conjunto de instruções, estrutura de barramento, E/S e a organização de memória é complexo e varia dependendo da arquitetura das peças utilizadas no computador. Por esse motivo o SO fornece uma camada de abstração para os aplicativos que se utilizam dos *hardwares* do computador possam criar, escrever e ler arquivos, sem ter de lidar com os detalhes complexos de como o *hardware* realmente funciona.
Como dito anteriormente um computador moderno consiste em um emaranhado de peça e a função do sistema operacional é fornecer uma alocação ordenada e controlada para todas elas além que o SO moderno permite que múltiplas aplicações estejam em memória e sejam executados “ao mesmo tempo”.
Precisamos entender que o sistema não faz execução de todos os recursos ao mesmo tempo isso seria insano de se imaginar mais ele utiliza de filas de processos e de um nivelamento de urgências para definir o que será processado e quando será processado, ele ainda define as utilizações em nível de memória para definir prioridades e utilizando delas para alterar o tempo de acesso ao processador, desta forma mesmo com vários processos “abertos” a execução se dá em filas multiplexadas [1], [3].
Ainda devemos entender que se o sistema possuir vários usuários a necessidade de gerenciar e proteger a memória, dispositivos de E/S e outros recursos é ainda maior, pois cada usuário precisa ter acesso aos recursos de *hardwares* e compartilhar de informações salvas como (arquivos, bancos de dados etc.) e as ações de um usuário pode influenciar ao uso do outro [1], [3].
Podemos dizer então que a função primordial do SO é manter controle, isso seja sobre como conceder acesso a recursos requisitados, sobre quais programas estão usando qual recurso, sobre como conceder recursos requisitados, contabilizar o seu uso, assim como mediar requisições conflitantes de diferentes programas e usuários [1], [3].
O termo *hardware* foi muito utilizado mais qual seriam os *hardwares* de um computador ou uma estrutura basica dos mesmos e como o sistema operacional se utiliza deles para em sua execução [1], [3].

2.1 O zoológico dos sistemas operacionais

Os sistemas operacionais e existem a mais de meio século e durante esse tempo diversos sistemas foram criados. E esses sistemas foram criados para atender a distintas finalidades.
Existem diversas formas e abordagens para a separação destes sistemas operacionais exemplificaremos apenas algumas mais conhecidas.

2.1.1 Sistemas operacionais de computadores de uso pessoal

Os SO destinados a essa função dão suporte a multiprocessos e são multiusuários normalmente durante seu início do sistema ele carrega diversos programas que proporcionem um apoio ao usuário. é comum que os SO de computadores pessoais possua uma interface gráfica (GUI) pois o foco deste sistema é proporcionar um bom apoio ao usuário [1], [3].

2.1.2 Sistemas operacionais de servidores

Assim como os sistemas operacionais de computadores de uso pessoal esses sistemas são multiusuário e executadas multi processos mas diferentemente do sistema anterior normalmente ele não possui um GUI e sim apenas um shell de acesso [1], [3].
Eles são executados em servidores que são computadores pessoais muito grandes, em estações de trabalho ou mesmo computadores de grande porte [1], [3].
Servidores são utilizados principalmente para fornecer serviços como de impressão, de arquivo ou de *web*. Provedores de acesso à internet utilizam várias máquinas servidoras para dar suporte aos clientes, e sites usam servidores para armazenar páginas e lidar com as requisições que chegam [1], [3].
O sistema que trataremos neste trabalho é um sistema de servidor [1], [3].

2.1.3 Sistemas operacionais embarcados

Sistemas embarcados são executados em computadores que controlam dispositivos que não costumam ser vistos como computadores e que não aceitam *softwares* instalados pelo usuário. Exemplos típicos são os fornos de micro-ondas, os aparelhos de televisão, os carros, os aparelhos de *DVD*, os telefones tradicionais e os *MP3 players*. A principal propriedade que distingue sistemas embarcados dos portáteis é a certeza de que nenhum *software* não confiável vá ser executado nele um dia. Você não consegue baixar novos aplicativos para o seu forno de micro-ondas – todo o *software* está na memória *ROM*. Isso significa que não há necessidade para proteção entre os aplicativos, levando a simplificações no *design* [1], [3].

3 Processos e threads

Processo é a abstração mais conhecida no sistema operacional, trata-se de ser uma execução de um programa, além que tudo depende deste conceito para dar suporte a possibilidade de haver operações concorrentes mesmo quando existe apenas uma *CPU* disponível, através dessa abstração é possível criar um suporte a *CPU* transformando uma única *CPU* em múltiplas *CPUs* virtuais [1], [9], [10].

3.1 Processos

Os computadores modernos frequentemente realizam várias tarefas ao mesmo tempo, as vezes isso é tão imperceptível que nem pensamos nisso durante a utilização de um computador, quando nao temos vários *softwares* abertos raramente pensamos que o sistema operacional abriu vários *software* de apoio ao usuário como drivers e *software* de *GUI* entre outros [1].
Todos os *softwares* executáveis no computador, às vezes incluindo o sistema operacional, são organizados em uma série de processos sequenciais, ou, simplesmente, processos. um processo é uma instância do *software* em execução. durante a execução de processos temos a impressão que todos estão sendo executados simultaneamente mas esse é uma ideia errônea a cpu trabalha de forma linear processando processo a processo distintamente. o que pode acontecer é que durante o tempo de carregamento de uma informação vinda do *HD* que pode levar menos de 1 milisegundo para o processador esse tempo é uma eternidade e durante esse tempo de carregamento da informação ele consegue executar diversas informações [1].
Na verdade é como se como se cada processo possui uma *CPU* virtual própria pois a *CPU* real troca a todo momento de processo em processo, mas, para compreender o sistema, é muito mais fácil pensar a respeito de uma coleção de processos sendo executados em (pseudo) paralelo do que tentar acompanhar como a *CPU* troca de um programa para o outro [1].
Todos os processos devem ter suas informações salvas, cada processo pode ter diversos arquivos abertos e esse arquivos possuem ponteiros referindo ao processo, sempre que o processo retorna a execução o sistema chama o *read* para ler as posições salvas. quando o dados não estão em espaço de endereçamento eles são salvos na tabela de processos [1].

Um sistema operacional possui três tipos de processo:

1. Execução – Um processo que realmente está sendo utilizado naquele instante.

2. Pronto – Parado temporariamente para dar espaço a outro processo, porém é executável.

3. • Bloqueado – Impossibilitado de executar enquanto evento externo esperado não acontecer

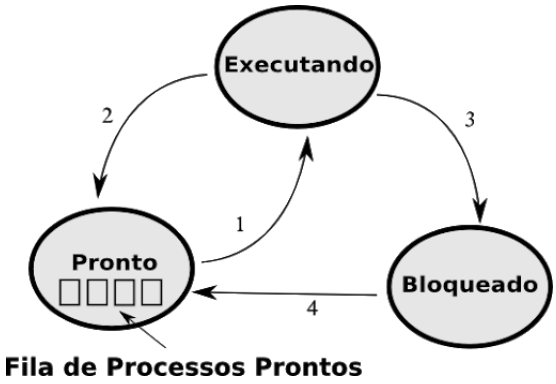

Fig. 8. Estados do processo. [1]

3.1.1 Criando processos

Em suma os SO precisam de uma maneira de criar os processos. Podemos dizer de forma abstrata que o programa é como um livro de receita e que o processo é o ato de cozinhar em si onde se segue passo a passo da receita e que os ingredientes para o preparo são dos dados.
Em sistemas simples ou embarcados pode ser possível ter todos os processos que serão em algum momento necessários quando o sistema for ligado. Mas para sistemas para fins gerais, no entanto, de alguma maneira é necessária para criar e terminar processos, na medida do necessário, durante a operação [1], [9], [10].

Quatro eventos principais fazem com que os processos sejam criados:

1. Inicialização do sistema.

2. Execução de uma chamada de sistema de criação de processo por um processo em execução.

3. Solicitação de um usuário para criar um novo processo.

4. Início de uma tarefa em lote.

Durante o carregamento inicial do SO diversos processos são criados alguns em primeiro plano e outros em segundo plano. Podemos dizer que todos os processos de primeiro plano são de interação com o usuário e que os processo de segundo plano são de

domínio do próprio sistema [1], [9], [10], [11].
No *linux server* cada processo possui um número de *PID* (*Process Identifier* - em portugues Identificador de Processo)este é um número de identificação que o sistema dá a cada processo. Para cada novo processo, um novo número deve ser atribuído, ou seja, não se pode ter um único *PID* para dois ou mais processos ao mesmo tempo [1], [9], [10], [11].
Os sistemas baseados em *Unix* precisam que um processo já existente se duplique para que a cópia possa ser atribuída a uma tarefa nova. Quando isso ocorre, o processo "copiado"recebe o nome de "processo pai", enquanto que o novo é denominado "processo filho". É nesse ponto que o *PPID* (*Parent Process Identifier* - em portugues Identificador de processo pai) passa a ser usado: o *PPID* de um processo nada mais é do que o *PID* de seu processo pai [1], [9], [10], [11].
No *Unix*, há apenas uma chamada de sistema para criar um novo processo: *fork*. Essa chamada cria um clone exato do processo que a chamou. Após a *fork*, os dois processos, o pai e o filho, têm a mesma imagem de memória, as mesmas variáveis de ambiente e os mesmos arquivos abertos. E isso é tudo. Normalmente, o processo filho então executa *execve* ou uma chamada de sistema similar para mudar sua imagem de memória e executar um novo programa [1], [9], [10], [11].

3.1.2 Término de processos

Após criado o processo ele começa a ser executado e realiza qualquer que seja o seu trabalho, e como pode se imaginar pouco depois ele tende a ser finalizado.

O novo processo terminará, normalmente devido a uma das condições a seguir:

1. Saída normal (voluntária)
2. Erro fatal (involuntário).
3. Saída por erro (voluntária).
4. Morto por outro processo (involuntário).

A maioria dos processos ele termina por ter chegado ao seu final, ou seja, quando um compilador termina de traduzir o programa dado a ele, o compilador executa uma chamada para dizer ao sistema operacional que ele terminou. A segunda é um erro fatal por exemplo a execução de um arquivo que não existe no sistema. A terceira possibilidade é quando existe um erro no programa executado por algum tipo de dado inexistente ou incorreto.
A quarta razão é quando o processo sofre influência externa por exemplo por um comando que finalize esse processo. Esse motivo em especial existe uma grande variação de comandos *linux* para executar esse tipo de recurso em especial o comando *kill* [1], [9], [10], [11].

O comando *kill* envia o sinal especificado para o especificado processos ou grupos de processos. Se nenhum sinal for especificado, o sinal termino é enviado. O padrão ação para este sinal é encerrar o processo. Este sinal deve ser usado de preferência ao sinal *kill*, uma vez que um processo pode instalar um manipulador para o sinal termino a fim de executar etapas de limpeza antes de encerrar de maneira ordenada. Se um o processo não termina depois que um sinal termino foi enviado, então o sinal *kill* pode ser usado; esteja ciente de que o último sinal não pode ser capturado e, portanto, não dá ao processo de destino a oportunidade de execute qualquer limpeza antes de encerrar [1], [9], [10], [11].
A maioria dos *shells* modernos tem um comando *kill* embutido , com um uso semelhante ao do comando descrito aqui. As opções *–all , –pid e –queue* e a possibilidade de especificar processos por comando nome, são extensões locais.
Caso prefira, você também pode matar de uma vez só todos os comandos selecionado ao nome de um programa. Para isso, basta usar o comando *killall* seguido do nome do *software* em questão, como *killall vim*.
Porém, o *killall* exige uma certa rigidez ao informar o nome do processo. Caso você não tenha certeza do nome completo, pode tentar o *pkill*, que faz diversas associações com a palavra-chave digitada [1], [9], [10], [11].

3.2 Threads

Cada programa, ou processo, possui normalmente um fluxo de controle. Assim o programa é executado seqüencialmente passo a passo com seu único fluxo de controle. Em sistemas operacionais tradicionais, cada processo tem um espaço de endereçamento e um único *thread* de controle. Neste ponto que as *threads* se destacam, com as *threads* podemos ter mais de um único fluxo de controle em nosso aplicativo [1], [12].
As *threads* em muitas situações, é desejável ter múltiplos *threads* de controle no mesmo espaço de endereçamento executando em quase paralelo, como se eles fossem (quase) processos separados (exceto pelo espaço de endereçamento compartilhado) [1], [12].
Assim o *software* agira como se tivessem sido dividido em varias partes de seu código atuando em paralelo no sistema. *Threads* são, portanto, entidades escalonadas para executarem na *CPU*, por isso a noção de (pseudo) paralelismo, pois as *threads* concorrerão pelo processador juntamente com mais *threads* que tiverem no programa, ou concorrerá apenas com o fluxo do programa [1], [12].

3.3 Escalonamento

O escalonador é um subsistema do SO encarregado de direcionar a prioridade de entrada dos processos na *CPU* os algoritmos avaliam o cenário disposto pelo sistema e com isso determina a lógica empregada para resolver qual processo será processado é importante lembrar que algumas variáveis necessitam de mais processo estes ocuparão a *CPU* por um tempo maior e não precisam da intervenção do usuário. Os processos que precisam de mais entradas e saídas de dados, ou seja, o processo demanda intervenção do usuário [1].

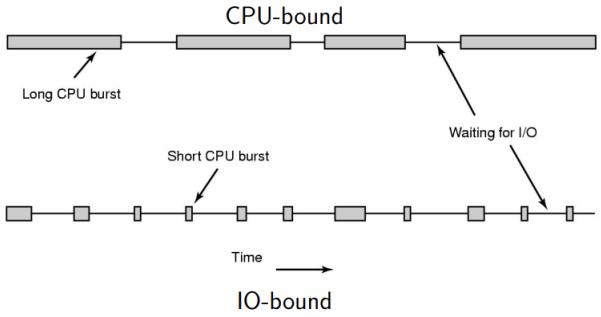

Fig. 9. Comportamento de processos. [1]

3.3.1 Categorias de algoritimo de escalonamento

Diferentes algoritmos de escalonamento são implementados para tratar condição que acontece nas diferentes áreas de aplicação do SO principalmente processos identificados para objetivos diferentes. Cada sistema tem particularidades que devem ser levadas em consideração pelo escalonador [1].

É necessário distinguir três ambientes:

1. Lote
2. Interativo
3. Tempo real

Sistema em lote são bastante comuns por sua otimização no tempo de retorno e maximiza o número de horas de trabalho e muitas vezes aplicáveis a outras situações também, o que torna seu estudo interessante, mesmo para pessoas não envolvidas na computação corporativa de grande porte [1].
Sistemas interativos tem objetivo de otimizar o tempo de resposta, mesmo que nenhum processo execute de modo intencional para sempre, um erro em um programa pode levar um processo a impedir indefinidamente que todos os outros executem. Os servidores *Linux*

também caem nessa categoria, visto que eles normalmente servem a múltiplos usuários (remotos), todos os quais estão muito apressados, assim como usuários de computadores [1].

Em sistemas de tempo real, os processos sabem que eles não podem executar por longos períodos e em geral realizam o seu trabalho e bloqueiam rapidamente. A diferença com os sistemas interativos é que os de tempo real executam apenas programas que visam ao progresso da aplicação à mão [1].

4 Gerenciamento de memória

O processador apesar de sua grande velocidade não consegue armazenar dados logo para que seja possível a execução dos programas pode se dizer que todo computador deve possuir algum tipo de memória. Podemos dizer que a memória é um grande vetor de palavras ou bytes de variados tamanhos sendo que cada um possui seu próprio endereço. A memória principal é um repositório de dados acessíveis e compartilhados pela *CPU* e dispositivos de entrada/saída [13].
Inicialmente os computadores não faziam abstração da memória tendo que realizar a leitura direto da memória física, este tipo de abordagem além de lenta transforma a memória apenas em um grande índice que vai de 0 ao tamanho máximo de endereços da memória fazendo que a informação seja movida de célula a célula de 8 *bits*. Nesta condição quando mais programas forem executado simultaneamente pode ocorre erros sofridos quando um programa apagar os dados salvos por outro [13], [14], [15].
Apesar disto é possível que se execute mais de um programa simultaneamente e de forma funcional, desde que exista apenas um programa de cada acessando a memória, para que não exista conflitos. O conceito *swapping* faz com que um sistema operacional salve o conteúdo inteiro da memória em um arquivo de disco assim existe um backup da informação [1].
Mas a maioria dos computadores possuem uma hierarquia de que como já visto na figura 4 podemos afirmar ainda que quanto mais longe do processador maior a memória e menor é o seu valor. No *Linux*, tratamento da memória se dá de forma subdividida, ele trabalha seu gerenciamento com dois componentes: o primeiro, responsável pela alocação e liberação de memória física (páginas, grupos de páginas e pequenos blocos de memória) e o segundo pela memória virtual [13], [14], [15].

4.1 Gerência de memória física

Através de algoritmos como o *Buddy-heap* que rastreiam as páginas físicas disponíveis, o gerente primário de memória física consegue realizar as tarefas ao qual é responsável como alocação e liberação de todas as páginas físicas e é capaz de disponibilizar intervalos de páginas fisicamente contíguas sob demanda. Este tipo de solução trabalha dividindo a memória em partições para tentar satisfazer as requisições de forma adequada [13], [14], [15].
Quando se utiliza de um espaço de endereçamento o conjunto de endereços que um processo pode usar para endereçar a memória pode ser variado assim uma parceria de regiões alocáveis parceiras adjacentes podem ser combinadas para construir uma região maior ou

solicitações de pequenos blocos de memória que não puderem ser satisfeitas por não existir uma pequena região disponível, resultam na divisão de uma região maior em duas outras parceiras de tamanho igual, repetindo o processo, se necessário, até que se consiga uma região do tamanho desejado [13], [14], [15].
Em sistemas *Linux* as alocações são realizadas de maneira estática por drivers que durante o *boot* do sistema reservam uma área continua de memória, de forma dinâmica pelo alocador de páginas, sendo que as funções do *kernel* não precisam utilizar o mesmo alocador básico para a reserva de memória. Segundo Tanenbaum(2016) os principais sistemas de alocação de memória no *Linux* , são o sistema de memória virtual, o alocador de comprimento variável *kmalloc* e os dois caches de dados persistentes no *kernel* (*cache de buffers* e *cache* de páginas) [1], [13], [14], [15].

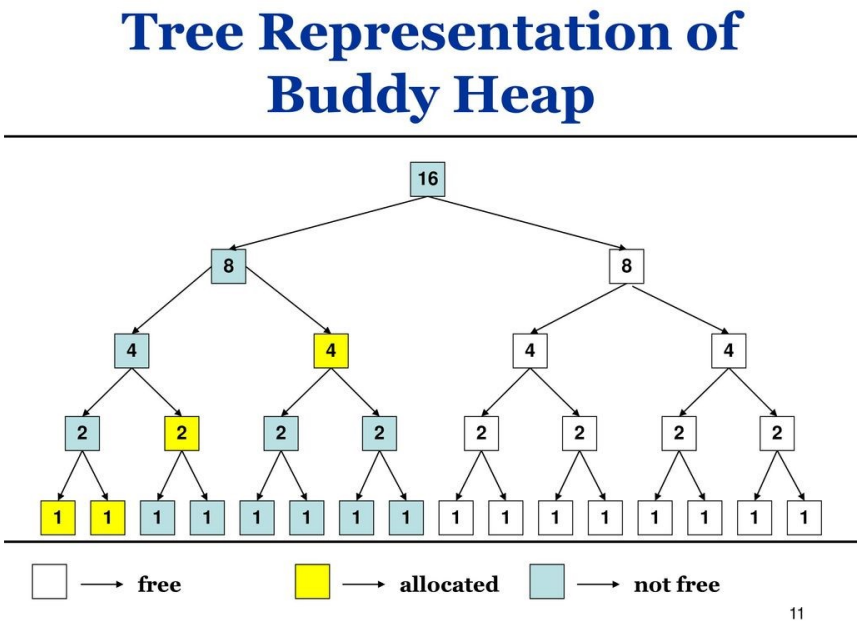

Fig. 10. Divisão da memória usando *BuddyHeap*.[2]

O alocador *kmalloc* é utilizado para solicitações de tamanho arbitrário. Assim o serviço aloca páginas inteiras sob demanda e, em seguida, divide-as em pedaços menores para satisfazer as requisições frequentes de pequenos blocos de memória. A alocação de memória utilizando esse serviço envolve determinar uma lista, entre as listas mantidas pelo *kernel* das páginas alocadas, utilizando o primeiro bloco disponível na lista ou alocando uma nova página e subdividindo-a.
O *kmalloc* não tem permissão e ou autoridade para relocar ou liberar regiões de memória já alocados mesmo que seja esta uma tratativa para uma possível falta de memória, depois de alocadas, essas regiões somente poderão ser reutilizadas mediante uma liberação explicita.
Outros três subsistemas principais que tratam de sua própria gerência de páginas físicas e que interagem intimamente entre si, segundo Silberschatz(2000) são o *cache* de *buffers*, que é o *cache* principal do *kernel* para dispositivos orientados a bloco; o *cache* de páginas que

mantém em *cache* páginas inteiras de conteúdo de arquivos, dados da rede entre outros e o sistema de memória virtual que gerencia o espaço de endereçamento virtual de cada processo [1], [13], [14], [15].

4.2 Memória Virtual

O sistema de memória virtual no *Linux* é responsável pela manutenção do espaço de endereçamento visível para cada processo assim sendo a criação das páginas de memória virtual sob demanda e a gerência do carregamento dessas páginas para o disco, ou o descarregamento de volta para o disco, é responsabilidade desse sistema.
O gerente de memória virtual mantem duas perspectivas do espaço de endereçamento de um processo sendo a primeira, é a visão lógica e a segunda a visão física de cada espaço de endereçamento [13], [14], [15].
Na primeira instruções recebidas pelo sistema de memória virtual referentes ao layout do espaço de endereçamento que consiste em um conjunto de regiões não superpostas, com cada uma representando um subconjunto contínuo e alinhado por página do espaço de endereçamento e sendo descrita internamente por uma única estrutura *vm_area _struct* (é uma estrutura de dados utilizada para descrever uma área da memória virtual para um processo), que define as propriedades da região. Na segunda é armazenada uma tabela de páginas do *hardware* para o processo e é gerenciada por um conjunto de rotinas.
Cada *vm_area _struct* na descrição do espaço de endereçamento contém um campo que aponta para uma tabela de funções que implementam as funções básicas de gerência de página para qualquer região dada da memória virtual [13], [14], [15].
No *Linux* existem diversos tipos de região de memória virtual mais suas propriedades de caracterização são do tipo de armazenamento secundário associado e sua reação a escritas, as regiões de espaço de endereçamento de um processo pode ser privada ou compartilhada sendo feito um processo *Copy-on-write* (é a operação onde uma região privada é copiada para uma nova região a fim de preservar essa região contra escritas a partir de outro processo) quando tentar escrever em uma região privada de outro processo, copiando o conteúdo da região para uma outra região nova e efetua as alterações na região recém-criada, preservando a região privada. Caso está escrita seja feita em região compartilhada o objeto mapeado para tal região é atualizado, de modo que as alterações ficam visíveis de imediato para todos os processos que estiverem mapeando tal objeto.
O *Linux* cria espaços de endereçamento através de duas situações em um novo programa que é chamado solicitando ao sistema de execução um novo espaço de endereçamento, no qual o processo ao receber este espaço ele está completamente vazio, e a rotina carrega o programa para ocupar o espaço, ou quando um novo processo é criado por *fork*, o que significa que uma cópia completa do espaço de endereço é criada para o processo existente [13], [14], [15].

4.3 Swapping e paginação

Com uma crescente utilização da memória através de sistemas de tempo compartilhado ou computadores pessoais orientados graficamente por vezes a memória principal não possuem espaço suficiente para manter todos os processos ativos uma solução para isso é salvar parte destes processos excedentes colocando os no disco liberando assim espaço para novos processos e estes ficando em stand-by esperando serem chamados novamente. Assim sendo dependendo do *hardware* duas técnicas podem ser utilizadas [13], [14], [15].
A técnica mais simples, é o /emphswapping usada para algoritmos de escalonamento, baseando-se em prioridades. Se um processo de prioridade mais alta necessitar ser carregado, o gerenciador de memória poderá descarregar um outro com prioridade mais baixa para o disco, para que o de maior prioridade possa ser executado. Quando for finalizado, o processo descarregado pode então, ser carregado novamente para a memória principal. Segundo Silberschatz(2000), quando o escalonador de CPU executar um processo, ele chama o *dispatcher* (também chamado de agendador de curto prazo, conforme Stallings(2004) descreve, decide qual dos processos na memória, prontos para a execução, será executado pelo processador) , que verifica se o próximo processo na fila está na memória. Se o processo não estiver na memória e não houver região de memória livre, o *dispatcher* descarrega um processo que está na memória *(swap out)* e carrega o processo desejado em seu lugar *(swap in)*, recarregando, então, os registradores de forma usual e transferindo o controle para o processo selecionado [13], [14], [15].
No *Linux*, o processo de paginação pode ser dividido em duas seções: o algoritmo de políticas, primeiramente, que decide que páginas são gravadas no disco e quando esse processo será feito, por meio de uma versão modificada do algoritmo de relógio, que emprega um relógio de passagens múltiplas.
A segunda seção, é o mecanismo de paginação, que suporta tanto partições e dispositivos dedicados, quanto arquivos, sendo que no último, o processo pode ser mais lento devido ao custo adicional provocado pelo sistema de arquivos. O algoritmo utilizado para a gravação das páginas é o algoritmo *next-fit*, para tentar gravar páginas em carreiras contínuas de blocos de disco, visando um melhor desempenho [13], [14], [15].

5 Sistemas de arquivos

Arquivos são unidades lógicas de informação criadas por processos. Podemos pensar nos arquivos como espaços de endereçamento que são usados para modelar o disco em vez da memória *RAM* os processos podem ler estes e através dos dados contidos neles gerar novos caso necessário. As informações contidas nos arquivos devem ser persistentes, ou seja, mesmo após o termino do processo elas devem ser mantidas integras para utilização futura exceto quando o próprio utilizador do sistema deseja excluir este. No *Linux* os arquivos sofrem uma estrutura hierárquica onde podemos ter níveis de permissões variando entre proprietário, grupo ao qual o proprietário pertence e público (ou seja, qualquer usuário do sistema). Os arquivos normalmente possuem três operações básicas escrita, leitura e execução [10], [16], [17].
Arquivos são gerenciados pelo sistema operacional. Como são estruturados, nomeados, acessados, usados, protegidos, implementados e gerenciados são tópicos importantes no projeto de um sistema operacional. Como um todo, aquela parte do sistema operacional lidando com arquivos é conhecida como sistema de arquivos.
Basicamente o sistema de arquivos é um conjunto de estruturas logicas que permite o sistema operacional controlar o acesso a um dispositivo de armazenamento, diferentes sistemas operacionais podem usam diferentes sistemas de arquivos, atualmente, o *NTFS (New Technology File System)* é o sistema de arquivos padrão do *Windows*, enquanto o *ext4* é o do *Linux* [10], [16], [17].

Abaixo podemos ver alguns dos sistemas de arquivos mais conhecidos:

- *EXT3 (third extended filesystem)* – foi adotado como padrão *Linux* a partir de 2001. Introduziu o registro *(journal)* que melhora a confiabilidade e permite recuperar o sistema em caso de desligamento não programado. *EXT3* suporta 16TB (1 terabyte corresponde a 240 bytes) de tamanho máximo no sistema de arquivos, e 2TB de tamanho máximo de um arquivo. Um diretório pode ter, no máximo, 32.000 subdiretórios.

- *EXT4 (fourth extended filesystem)* – passou a ser o padrão *Linux* a partir de 2008. *EXT4* suporta 1EB (1 exabyte corresponde a 260 bytes) de tamanho máximo de sistema de arquivos e 16TB de tamanho máximo de arquivos. É possível ter um número ilimitado de subdiretórios.

- *XFS (Extended Filesystem)* – usado como padrão por algumas distribuições *Linux* desde 2014. *XFS* é um sistema de arquivos desenvolvido em 64 bits, compatível com

sistemas de 32 bits. Ele suporta até 16 EB de tamanho total do sistema de arquivos e até 8 EB de tamanho máximo para um arquivo individual. É considerado um sistema de arquivos de alto desempenho.

• *JFS (Journaled File System)* – é um sistema de arquivos de 64 bits com *journaling* desenvolvido pela *IBM*.

• *HFS (Hierarchical File System)* – é um sistema de arquivos proprietário da *Apple*.

• *FAT (File Allocation Table)* – é um sistema desenvolvido para o *MS-DOS* e usado em versões do *Microsoft Windows* até o *Windows 95*. É suportado praticamente por todos os sistemas operacionais existentes. Existem 3 versões do sistema: *FAT* (12 bits, usado pelos disquetes), *FAT16* (para OS 16 bits ou 32 bits) e *FAT32* (só para SO a 32 bits).

• *NTFS (New Technology File System)* é o sistema de arquivos padrão do sistema operacional *Microsoft Windows*. São algumas características deste tipo de sistema: aceita volumes de até 2 TB; o tamanho do arquivo é limitado apenas pelo tamanho do volume; é um sistema de arquivos muito mais seguro que o *FAT*; *NTFS* podem se recuperar de um erro mais facilmente.

• *HPFS* – é o sistema de arquivos utilizado pelo *OS/2* da *IBM*, com recursos que se aproximam muito dos permitidos pelo *NTFS*.

• *UFS (Unix File System)* – é um sistema de arquivos usados por muitos sistemas operacionais *Unix* e assemelhados [10], [16], [17].

5.1 Gerenciamento de diretório

Sistemas de arquivos normalmente têm diretórios ou pastas, que podem ser de nível único ou hierárquicos.
Segundo Tanenbaum(2016) a forma mais simples de um sistema de diretório é ter um diretório contendo todos os arquivos. Às vezes ele é chamado de diretório-raiz.
Essa decisão foi tomada sem dúvida para manter simples o *design* do *software*. Um exemplo de um sistema com um diretório é dado na Figura 11. Aqui o diretório contém quatro arquivos. As vantagens desse esquema são a sua simplicidade e a capacidade de localizar arquivos rapidamente às vezes ele ainda é usado em dispositivos embarcados simples como câmeras digitais e alguns *players* portáteis de música [1].

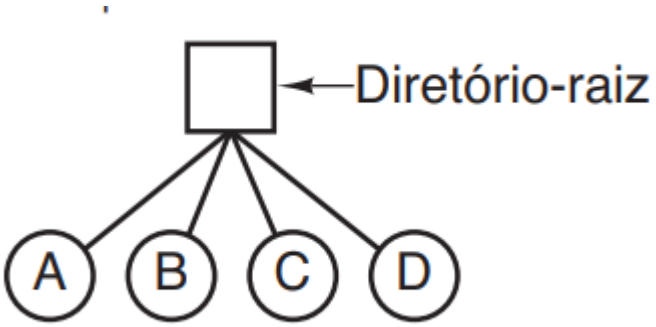

Fig. 11. Um sistema de diretório em nível único contendo quatro arquivos. [1]

Apesar do método anterior ser mais simples em atualmente os usuários em seus dispositivos milhares de arquivos inviabilizando o modelo anterior em consequência, é necessária uma maneira para agrupar arquivos relacionados em um mesmo local[1]. Faz-se necessária uma hierarquia (isto é, uma árvore de diretórios). Com essa abordagem, o usuário pode ter tantos diretórios quantos forem necessários para agrupar seus arquivos de maneira natural. Além disso, se múltiplos usuários compartilham um servidor de arquivos comum, como é o caso em muitas redes de empresas, cada usuário pode ter um diretório-raiz privado para sua própria hierarquia. Essa abordagem é mostrada na Figura 12 [1].

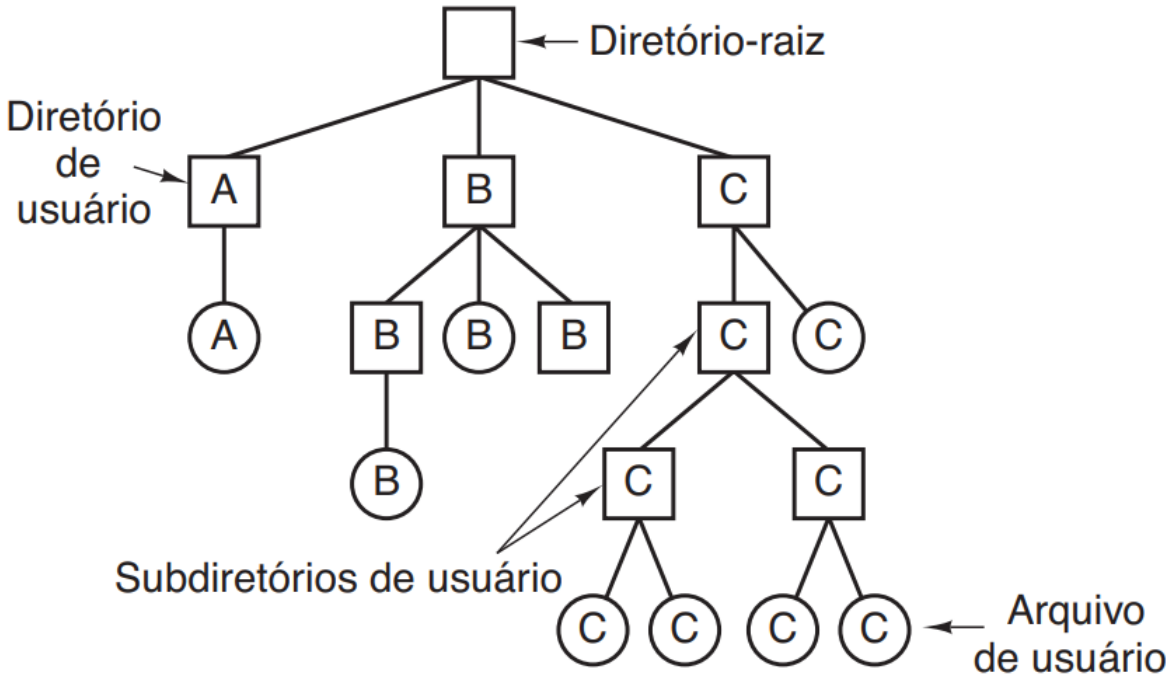

Fig. 12. Um sistema hierárquico de diretórios. [1]

No *Linux* o diretório raiz e o diretório "/" é o diretório com maior hierarquia entre todos os diretórios do sistema assim todos os outros diretórios estão abaixo dele[10], [16], [17].

A seguir são apresentados exemplos de diretórios que normalmente ficam abaixo do diretório raiz.

- bin – diretório com os comandos disponíveis para os usuários comuns (não privilegiados).
- boot – diretório com os arquivos estáticos do *boot* de inicialização.
- dev – diretório com as definições dos dispositivos de entrada/saída.
- etc – diretório com os arquivos de configuração do sistema.
- home – diretório que armazena os diretórios dos usuários do sistema.
- lib – diretório com as bibliotecas e módulos (carregáveis) do sistema.
- lost+found – é usado pelo *fsck* para armazenar arquivos/diretórios/devices corrompidos.
- media – ponto de montagem temporário para mídias removíveis.
- mnt – ponto de montagem temporário para sistemas de arquivos.
- opt – *softwares* adicionados pelos usuários.
- proc – diretório com informações sobre os processos do sistema.
- root – diretório *home do root*.
- run – armazena arquivos temporários da inicialização do sistema.
- sbin – diretório com os aplicativos usados na administração do sistema.
- snap – diretório com pacotes snaps (podem ser executados em diferentes distribuições *Linux*).
- srv – dados para serviços providos pelo sistema.
- sys – contém informações sobre *devices, drivers* e características do *kernel*.
- tmp – diretório com arquivos temporários.
- usr – diretório com aplicativos e arquivos utilizados pelos usuários como, por exemplo, o sistema de janelas X, jogos, bibliotecas compartilhadas, programas de usuários e de administração, etc.
- var – diretório com arquivos de dados variáveis (*spool, logs*, etc).

Três comandos básicos do *Linux server* para se deslocar entre diretórios são:

pwd (print working directory): informa o diretório de trabalho - ou corrente, apresentando o caminho desde o raiz até o diretório atual.

ls (list space): lista, de maneira simples ou com informações variadas, arquivos específicos ou o conteúdo de diretórios.

cd (change directory): serve para fazer a mudança entre os diretórios - como o próprio nome informa o seu uso pode ser para caminhar na pasta local, dentro da própria pasta, ou para qualquer local do sistema.

Compartilhamento de arquivos muitas vezes é necessário que um arquivo se mostre simultaneamente em diretórios diferentes por estar compartilhado entre vários usuários ou por possuir informação que são necessárias para o sistema como na figura 13 abaixo [1]:

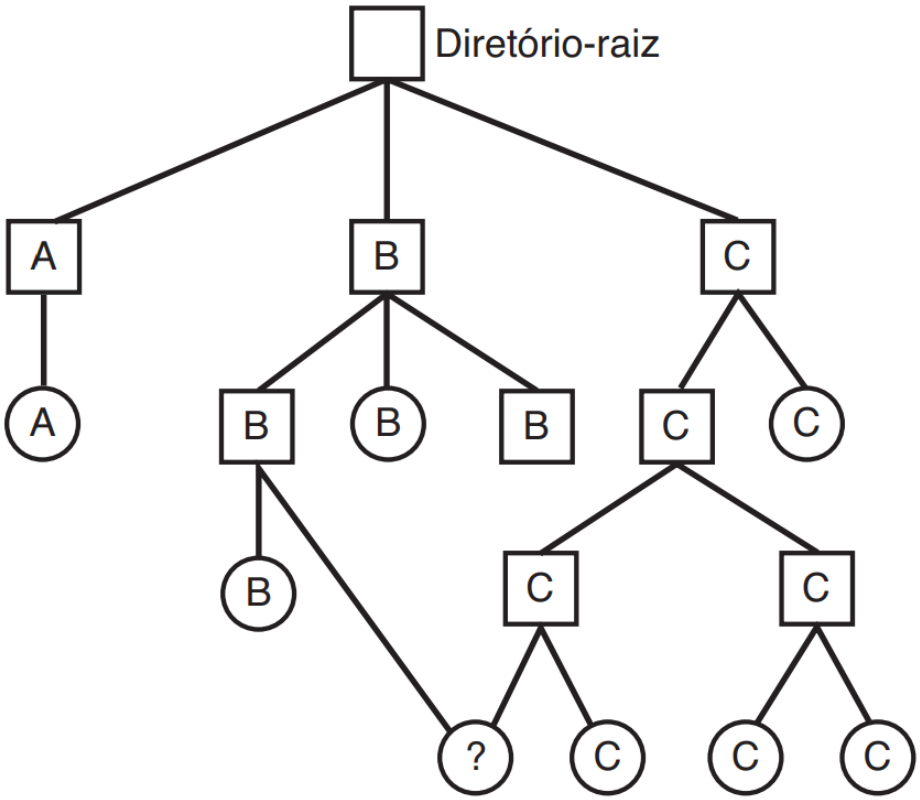

Fig. 13. Sistema de arquivos contendo um arquivo compartilhado. [1]

O compartilhamento de arquivos pode gerar alguns problemas se os diretórios realmente contiverem endereços de disco, então uma cópia desses endereços terá de ser feita no diretório do de cada usuário quando o arquivo for utilizado subsequentemente se cada usuário adicionarem blocos de arquivos em seu diretório pessoal quando utilizarem o arquivo é praticamente como se o usuário estivesse copiando o arquivo para seu diretório assim perdendo o sentido do compartilhamento [1].
Segundo Tanenbaum(2016) ele diz que esse problema pode ser solucionado de duas maneiras. Na primeira solução, os blocos de disco não são listados em diretórios, mas em uma pequena estrutura de dados associada com o arquivo em si. Os diretórios apontariam então apenas para a pequena estrutura de dados. Essa é a abordagem usada em *UNIX* (em que a pequena estrutura de dados é o *i-node*). A segunda solução seria que quando um novo usuário utilize do arquivo seja criado um arquivo *link* que irá conter apenas uma receita do arquivo original contendo seu o caminho do arquivo que está sendo utilizado, essa abordagem é chamada de ligação simbólica [1].

5.2 Gestão de espaço livre

Arquivos são armazenados em disco e os discos possuem limites físicos quanto a tamanho do disco e quantidade de arquivos que podem ser armazenados. Visto isso o gerenciamento do espaço do disco é uma das preocupações do projetista de sistemas de arquivos [1].

O primeiro consiste em usar uma lista encadeada de blocos de disco, com cada bloco contendo tantos números de blocos livres de disco quantos couberem nele. Com um bloco de 1 KB e um número de bloco de disco de 32 bits, cada bloco na lista livre contém os números de 255 blocos livres [1].

A outra técnica de gerenciamento de espaço livre é o mapa de bits. Um disco com n blocos exige um mapa de bits com n bits. Blocos livres são representados por 1s no mapa, blocos alocados por 0s (ou vice-versa) [1].

Para nosso disco de 1 TB de exemplo, precisamos de 1 bilhão de bits para o mapa, o que exige em torno de 130.000 blocos de 1 KB para armazenar [1].

Não surpreende que o mapa de bits exija menos espaço, tendo em vista que ele usa 1 bit por bloco, versus 32 bits no modelo de lista encadeada.

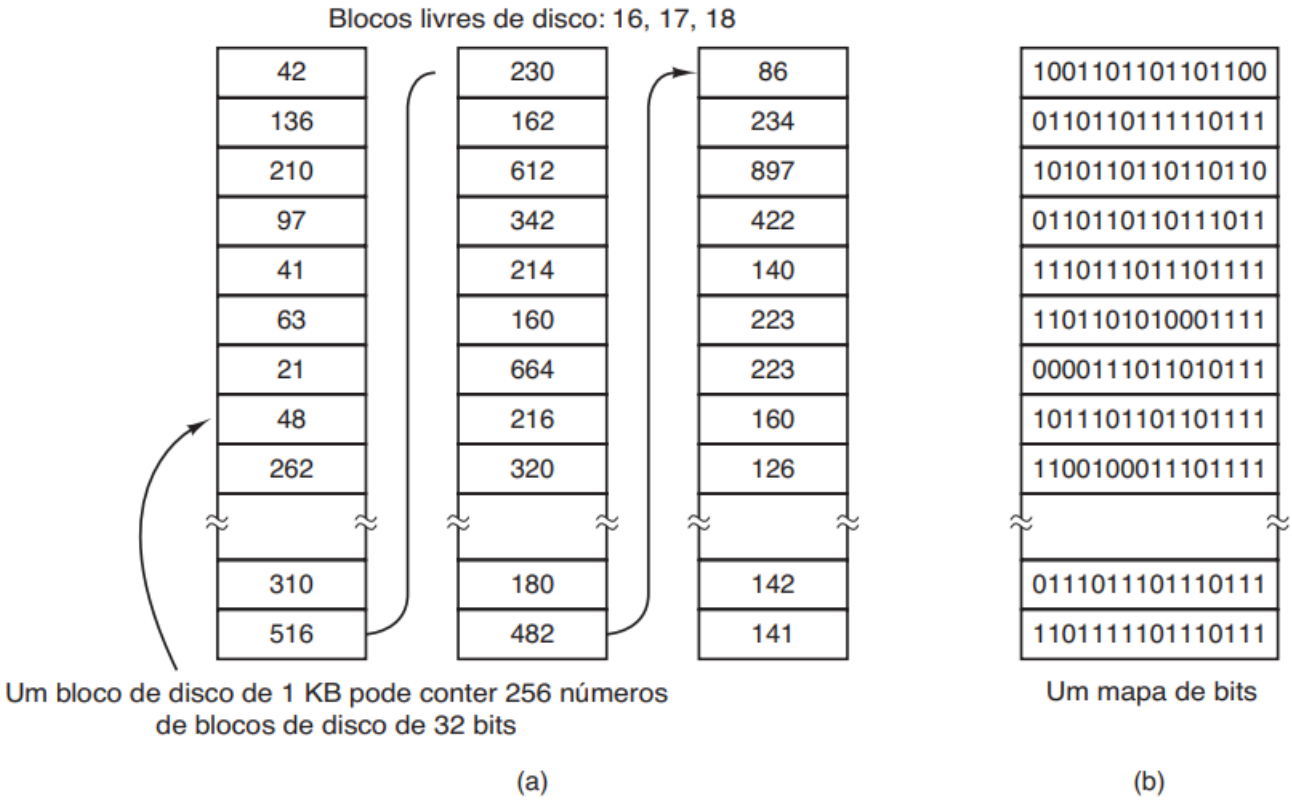

Fig. 14. (a) Armazenamento da lista de blocos livres em uma lista encadeada. (b) Um mapa de bits. [1]

5.3 Cotas de disco

É comum em sistemas de multiusuários que os sistemas definam cotas de disco que é um mecanismo para impor um tamanho máximo de utilização para cada usuário assim ele impede que o usuário utilize o sistema de forma a prejudicar o funcionamento do sistema[1].

. Quando um usuário abre um arquivo é gerada uma tabela de arquivos aberta na memória principal com os atributos endereços de disco são localizados. Quaisquer aumentos no tamanho do arquivo serão cobrados da cota do proprietário, uma segunda tabela é contendo os registros de cotas de todos os usuários com um arquivo aberto, mesmo que esse arquivo tenha sido aberto por outra pessoa. Essa tabela está mostrada na Figura 15. Ela foi extraída de um arquivo de cotas no disco para os usuários cujos arquivos estão atualmente abertos. Quando todos os arquivos são fechados, o registro é escrito de volta para o arquivo de cotas. Quando uma nova entrada é feita na tabela de arquivos abertos, um ponteiro para o registro de cota do proprietário é atribuído a ela, a fim de facilitar encontrar os vários limites [1].

Toda vez que um bloco é adicionado a um arquivo, o número total de blocos cobrados do proprietário é incrementado, e os limites flexíveis e estritos são verificados. O limite flexível pode ser excedido, mas o limite estrito não. Esse método tem a propriedade de que os usuários podem ir além de seus limites flexíveis durante uma sessão de uso, desde que removam o excesso antes de se desconectarem. Os limites estritos jamais podem ser excedidos [1].

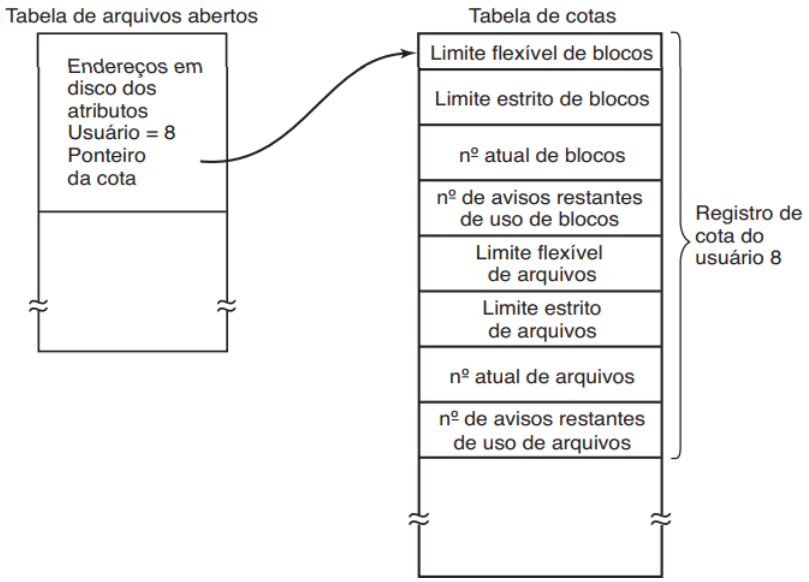

Fig. 15. As cotas são relacionadas aos usuários e monitoradas em uma tabela de cotas. [1]

Conclusão

Durante este estudo foi possível compreender a gigantesca extensão das atividades de um Sistema operacional, com o desenvolvimento do estudo este modelo expos o sistema e tudo o que ele gerencia.

Foram vistos todas as estruturas seja de /emphhardware ou /emphSoftware que o sistema operacional se utiliza ou gerencia para prover uma abstração para o usuário de forma a ser uma utilização transparente em especial o /emphLinux por ser um /emphSoftware Livre implica, por definição, na abertura da possibilidade de se deter todo o conhecimento embutido em uma aplicação. A tecnologia de sistemas de informação deixa de ser uma 'caixa preta' criada por uma 'sociedade superior', passando a estar ao alcance de todos. Outro fator importante apresentado foi a importância dos sistemas na utilização dos recursos computacionais.

Por fim, foi visto no decorrer do trabalho que os sistemas operacionais são de fundamental base de utilização para a área da tecnologia representando uma área de conhecimento que tem muito a agregar para todos aqueles envolvidos com ambientes computacionais.

Referências

[1] A. S. Tanenbaum and H. Bos, *Sistemas operacionais modernos*, P. E. do Brasil Ltda, Ed. 4ª edição, 2016.

[2] A. Norman. (2018) Upper bound for defragmenting buddy heaps. [Online]. Available: https://slideplayer.com/slide/13618569/

[3] D. Comer, *Operating system design : the xinu approach, linksys version.* Boca Raton: CRC Press-Taylor and Francis, 2012.

[4] T. L. I. Project. (2007) Kernel mode definition. [Online]. Available: http://www.linfo.org/kernel_mode.html

[5] L. Torvalds. (1991) "what would you like to see most in minix?". [Online]. Available: https://groups.google.com/g/comp.os.minix/c/dlNtH7RRrGA/m/SwRavCzVE7gJl

[6] M. Magazine. (1993) The choice of a gnu generation an interview with linus torvalds. [Online]. Available: https://gondwanaland.com/meta/history/interview.html

[7] L. Torvalds and D. Diamond, *Just for Fun: The Story of an Accidental Revolutionary.* HarperCollins, 2002. [Online]. Available: https://books.google.com.br/books?id=6zSWd8Ou8BAC

[8] T. L. Foundation. (2020) What is linux? [Online]. Available: https://www.linux.com/what-is-linux/

[9] E. Alecrim. (2005) Processos no linux. [Online]. Available: https://www.infowester.com/linprocessos.php#:~:text=Um%20PID%20(Process%20Identifier)%20%C3%A9,mais%20processos%20ao%20mesmo%20tempo

[10] C. E. Morimoto, *Servidores Linux: Guia prático*, E. Sulina, Ed. 1ª Edição, 2011.

[11] M. Kerrisk. (2007) The linux programming interface. [Online]. Available: https://man7.org/index.html

[12] Higor. (2007) Programação com threads. [Online]. Available: https://www.devmedia.com.br/programacao-com-threads/6152

[13] C. UFSCAR. (2019) Gerenciamento de memória - 2019.1. [Online]. Available: http://www2.comp.ufscar.br/mediawiki/index.php/Gerenciamento_de_mem%C3%B3ria_-_2019.1

[14] A. Silberschatz, P. Galvin, and G. Gagne, *Sistemas Operacionais: Conceitos e Aplicações. Tradução de Adriana Rieche*, Campus, Ed. 8ª edição, 2000.

[15] W. Stallings, *Operating Systems Internals and Design Principles*, P. Hall, Ed. 4ª edição, 2004.

[16] guialinux. (2020) Sistemas de arquivos. [Online]. Available: https://guialinux.uniriotec.br/sistemas-de-arquivos/

[17] L. A. Siqueira, *Certificação LPI-1 101 102*, E. A. Books, Ed. 5ª edição, 2015.