

Unidade 4

Dispositivos de Entrada e Saída



Introdução

Atualmente, os dispositivos computacionais estão atingindo uma sofisticação muito grande. Para acompanhar essa grande evolução, os sistemas operacionais também mudaram.

Em *desktop*, por exemplo, o número de dispositivos que podemos conectar é bem variado. Usando as portas USB's, conectamos desde a *smartphones* até dois monitores ao mesmo tempo.

Quando se conecta alguma entrada em um sistema operacional, diversos processos nesse sistema acontecem. Não é algo trivial, mas é despercebido pelos usuários. Então o objetivo dessa unidade, em específico a aula 1, é mostrar o funcionamento dos dispositivos dentro de um sistema operacional. Ou seja, o seu funcionamento.

Estudamos até esse momento bastante teoria sobre os sistemas operacionais. Então, chegou a hora de fazer um estudo dentro de um sistema operacional real, nessa unidade. Esse é o propósito da aula 2. Nessa aula, debateremos alguns tópicos relevantes dentro do sistema operacional Linux. Entre os assuntos tratados, veremos comandos internos para a Gerência do sistema Linux como um todo.

Bons estudos!

Objetivos

- Entender como funciona os dispositivos em Sistemas Operacionais;
- Fazer um estudo de caso no Sistema Operacional Linux.

Conteúdo programático

Aula 01 – Gerência de dispositivos.

Aula 02 – Estudo de Caso do Sistema Operacional Linux.



Você poderá também **assistir às videoaulas** em seu celular! Basta apontar a câmera para os **QR Codes** distribuídos neste conteúdo.

Pode ser necessário instalar um aplicativo de leitura QRcode no celular

Aula 01 – Gerência de dispositivos

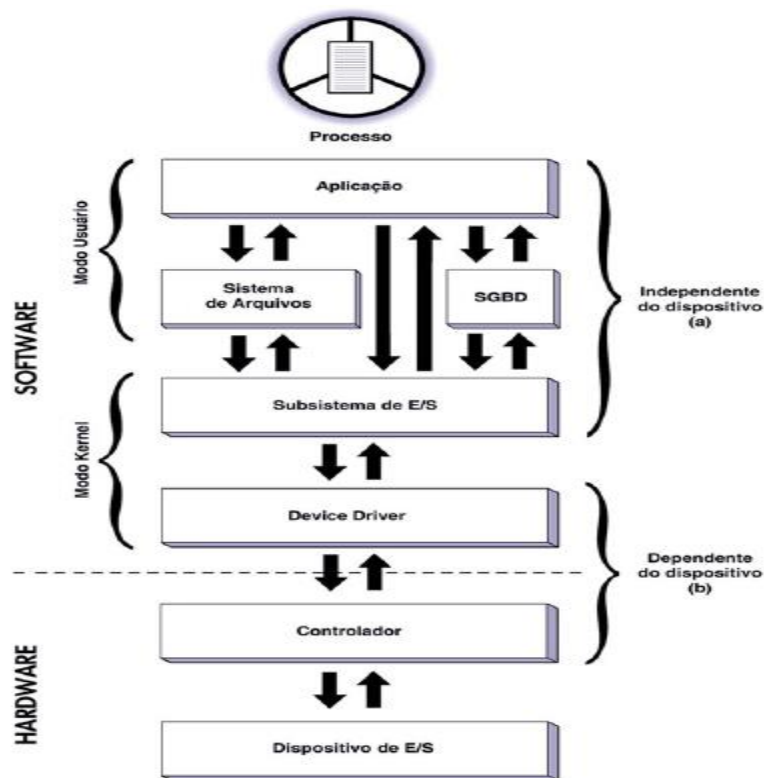
Olá alunos! Estudamos até aqui muitos conceitos importantes relacionados aos sistemas operacionais, tais como: Gerência de memória, gerência de arquivos e processos. Nessa unidade, estudaremos uma das mais complexas funções que o sistema operacional exerce, as operações de entrada e saída.

Operações de Entrada/Saída

Toda vez que o sistema realizar uma operação de E/S (Entrada e saída) ela deve ser a mais simples possível para o usuário e a sua aplicação, dado que ninguém quer perder tempo acessando periféricos de maneira demorada. A implementação das funções de entrada e saída é estruturada através de um modelo de camadas, onde a camada de mais baixo nível esconde as características dos dispositivos das camadas superiores, veja a figura 1 abaixo.

Dessa forma, o sistema operacional (SO) deve ser projetado de forma que possa se comunicar com qualquer dispositivo que possa ser conectado ao *hardware* do computador. Esse conceito é denominado independência de dispositivos.

Figura 1 - Arquitetura de camadas da gerencia de dispositivos



Fonte: Machado; Maia (2013)

Para acessar algum dispositivo de maneira mais simples possível, podemos utilizar de bibliotecas de comandos de entrada/saída que são oferecidas pela maioria das linguagens de alto nível. Os comandos dessas linguagens como *C* e *Java* independem do ambiente operacional em uso.

Ainda, a independência dos dispositivos pode ser alcançada através das *system calls* para E/S. Sendo assim, é possível escrever aplicações que manipule arquivos, estejam eles em discos rígidos ou *pen drives*, sem que precise alterar o código para cada tipo de dispositivo. Portanto, o objetivo das *system Calls* é simplificar a interface entre as aplicações e os dispositivos (MACHADO; MAIA, 2013).

Então, conforme explanado anteriormente, a implementação da gerência de entrada e saída é estruturada através de camadas, conforme a figura 1. Veremos a partir daqui as funções dessas camadas para um melhor entendimento desse assunto.

Subsistema de Entrada/Saída

A função da camada de subsistema de entrada/saída é ser responsável por realizar funções que são comuns a todos os dispositivos, deixando os aspectos mais específicos de cada periférico a critério dos *device drivers*, assunto que vai ser tratado a *posteriori*. Assim de uma forma genérica, podemos considerar que o subsistema de E/S é a parte do sistema operacional responsável pela gerência dos dispositivos.

Uma função exercida pelo subsistema é mapear o nome do dispositivo com seu respectivo *driver*. As camadas superiores conhecem apenas o nome do dispositivo e utilizam esse nome para terem acesso ao periférico. Outra função importante também gerenciada pelo subsistema é a técnica de “bufferização”. Essa técnica, permite reduzir o número de operações de E/S, utilizando uma área de memória intermediária chamada de *buffer* (MACHADO; MAIA, 2013).

Vamos exemplificar. Ao buscar algum dado solicitado, o sistema apresenta além do dado, mais um bloco de dados. Assim, quando o processo requisitar mais um dado além daquele que foi lido, ele já estará no *buffer*. Dessa maneira evita-se uma nova operação de entrada e saída. Essa operação é muito usada no sistema de memória *cache*-processador.

Videoaula 1

Agora assista à videoaula na qual será explanado sobre as características de um sistema de entrada e saída.



Videoaula 1

Utilize o QRcode para assistir!

Agora assista à videoaula na qual será explanado sobre as características de um sistema de entrada e saída.



Device Drivers

Com certeza, caro aluno, você já se deparou com esse conceito na sua rotina diária. Quando você instala um sistema operacional a próxima etapa é instalar os *drivers* corretos para que funcione determinado dispositivo. Pois se você não faz essa instalação corretamente, o *driver* instalado não irá integrar de maneira adequada com o subsistema acima ou com o controlador abaixo, conforme mostrado na figura 1.

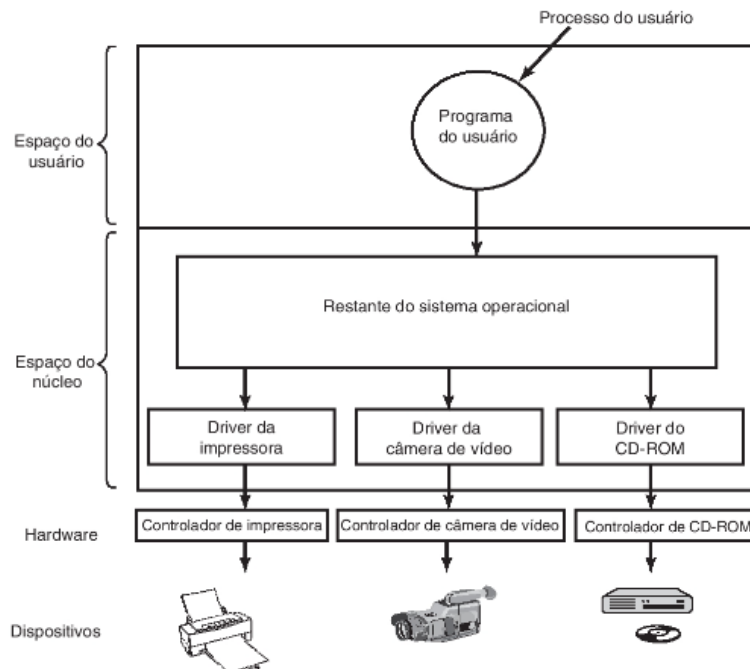
Os *drivers* tem como função fazer uma ponte via *hardware* entre o subsistema de E/S com os dispositivos por intermédio dos controladores. Assim, enquanto os subsistemas de E/S trata de funções que afetam a todos dispositivos, os *drivers* irão tratar somente dos seus aspectos particulares.

Os *drivers* de dispositivos geralmente são códigos escritos pelos fabricantes e fornecidos juntamente com os dispositivos. Dessa maneira, cada sistema operacional utiliza seus próprios *drivers* dos dispositivos, os fabricantes normalmente fornecem *drivers* para os sistemas operacionais mais populares (TANENBAUM, 2015).

Para se acessar qualquer *hardware* de algum dispositivo, o driver conta com o auxílio dos controladores que contêm registradores que dão a eles comandos de acesso diretamente ao *hardware*. O número de registradores e a natureza dos comandos usados dependem exclusivamente de cada dispositivo. Por exemplo, o *driver* de um *mouse* deve aceitar informações do *mouse* dizendo quanto ele se moveu e qual botão foi pressionado.

Vale notar que normalmente os *drivers* fazem parte do núcleo do sistema. No entanto, pode-se projetar *drivers* que executem no espaço do usuário, utilizando as chamadas de sistemas para leitura e escrita nos registradores dos dispositivos. Isso é uma técnica muito importante, dado que isola o núcleo do sistema operacional dos *drivers*, e os *drivers* entre si, eliminado uma das maiores “dores de cabeça” dos sistemas. Uma arquitetura construída dessa forma é altamente confiante. *Drivers* dos dispositivos podem ser posicionados abaixo do restante do sistema operacional, conforme podemos verificar na figura 2 (TANENBAUM, 2015).

Figura 2 - Posicionamento dos *drivers* no sistema operacional



Fonte: Tanenbaum (2015, p. 216)

Como podemos notar, os *drivers* fazem parte do núcleo do sistema operacional, sendo a sua escrita feita geralmente na linguagem *assembly*. Qualquer erro de programação em *drivers* pode comprometer por completo o funcionamento do sistema. Por isso um *device driver* deve ser cuidadosamente desenvolvido e testado.

Depois de minuciosamente projetado e testado, os fabricantes desenvolvem para um mesmo dispositivo, diferentes *device drivers*, um para cada sistema operacional. Toda vez que um novo dispositivo é instalado, o *driver* do dispositivo deve ser adicionado ao núcleo do sistema. Antigamente a inclusão de um novo *driver* no sistema significava a recompilação do kernel, uma operação que exigia a reinicialização do sistema (MACHADO; MAIA, 2013).

Videoaula 2

Agora assista à videoaula na qual será debatido sobre as características de um *DEVICE DRIVER*.



Videoaula 2

Utilize o QRcode para assistir!

Agora assista à videoaula na qual será debatido sobre as características de um DEVICE DRIVER.



Leitura Obrigatória

Leia o tópico 5.3.2 do livro de Andrew Tanenbaum. Nesse tópico, o autor trata sobre os *Drivers* dos Dispositivos. Acesse o *link*:

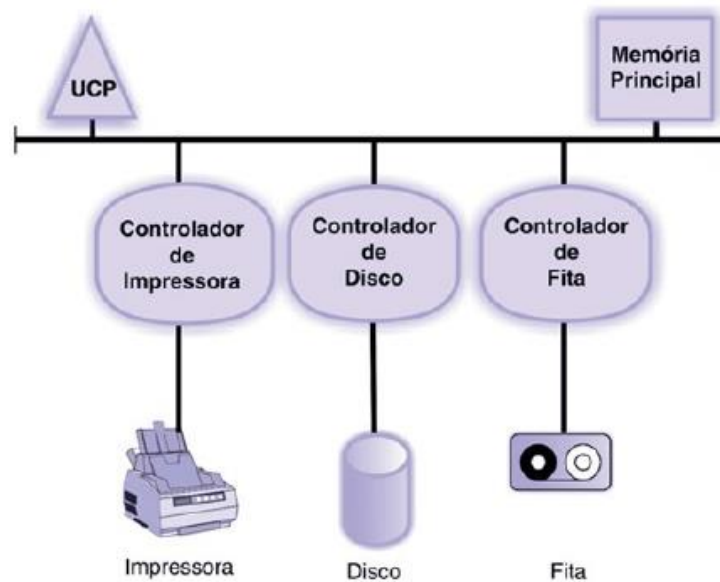
Disponível em:

<https://plataforma.bvirtual.com.br/Leitor/Publicacao/1233/pdf/0?code=ACkKVA8YaGRY1SbyMvjg7uTfbtv49JTITntyblZbA+ztei8cDO+Q6zNJ5GD0ejkVoRd5/zHWcm7GHDrmwIZylA=>
≡. Acesso em: 2 maio 2021.

Controladores de Entrada e Saída

Os controladores de E/S são componentes de *hardware* responsáveis pela manipulação direta dos dispositivos físicos. É com esses componentes que os *device drivers* “conversam”. No computador, os controladores, figura 3, podem ser apresentados na forma de uma placa controladora de circuito impresso que pode ser inserida em um conector de expansão, por exemplo.

Figura 3 - Controladores e dispositivos de E/S



Fonte: Machado; Maia (2013)

O controlador possui memória e registradores próprios utilizados para executar as instruções enviadas pelo *device driver*. As instruções que são de baixo nível, são responsáveis pela comunicação entre o dispositivo E/S e o controlador.

Leitura Obrigatória

Leia o tópico 5.1.2 do livro de Andrew Tanenbaum. Nesse tópico, o autor trata sobre os Controladores dos Dispositivos. Acesse o link:

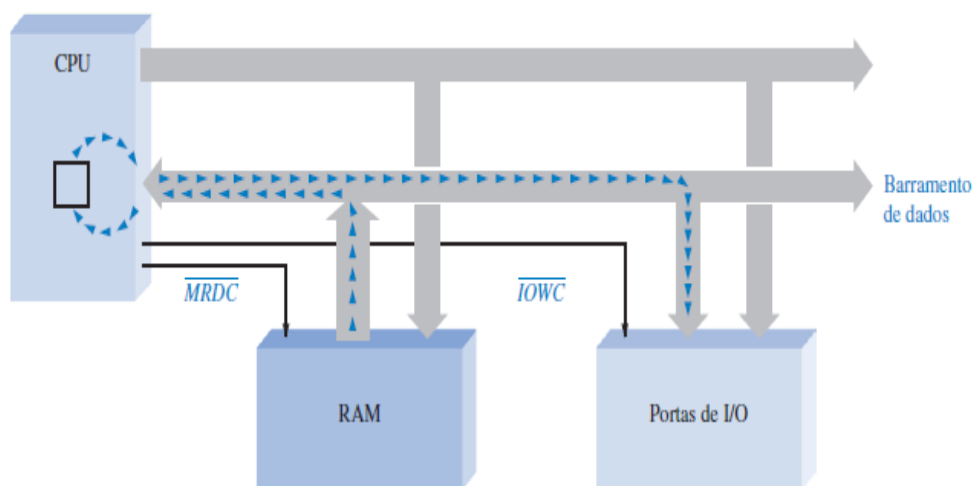
Disponível em:
<https://plataforma.bvirtual.com.br/Leitor/Publicacao/1233/pdf/0?code=ACkKVA8YaGRY1SbyMvjq7uTfbtv49JTITntyblZbA+ztei8cDO+Q6zNJ5GD0ejkVoRd5/zHWcm7GHDrmwIzYlA=>
=. Acesso em: 2 maio 2021.

Quando um controlador faz uma leitura, por exemplo, ele armazena em seu *buffer* interno uma sequência de *bits* oriundos de algum dispositivo até formar um bloco. Depois de verificar a existência de erros, o bloco poderá ser transferido para um *buffer* de E/S na memória principal. Essa transferência é feita pela a UCP (*Central Processing Unit*) ou por um controlador de DMA (*Direct Memory Access*).

Vamos estudar um pouco sobre a transferência feita pelo UCP, veja a figura 4. Por exemplo, quando os dados são transferidos da RAM para um dispositivo periférico, a UCP lê o primeiro *byte* de dado da memória e o carrega em um registrador interno ao microprocessador. Em seguida a UCP escreve o *byte* de dado na porta de E/S apropriada. Essa operação de leitura/escrita é repetida para cada *byte* do grupo de dados a ser transferido (FLOYD, 2007).

Podemos então notar, que se for utilizar grandes blocos de dados serão utilizadas muitas paradas intermediárias pelo microprocessador consumindo assim muito tempo de transferência.

Figura 4 - Transferência de memória para dispositivo E/S controlada pela UCP



Fonte: Floyd (2007, p. 737)

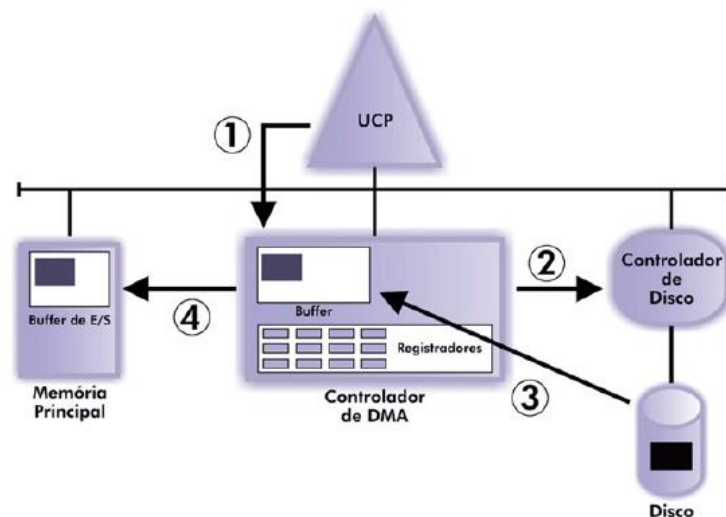
DMA é uma técnica que foi implementada já na década de 50, cuja função é de mover os dados de uma região da memória para a E/S e vice-versa, tendo acesso direto a eles sem a intervenção da UCP. O controlador de DMA é um dispositivo de *hardware* que pode fazer parte de um controlador ou pode ser um dispositivo independente. O sistema operacional somente pode usar o DMA se o *hardware* tem o controlador DMA.

Para as transferências diretas com a memória, o DMA assume o controle dos barramentos do sistema e permite que os dados passem de maneira direta entre a RAM e o dispositivo periférico (FLOYD, 2007).

Então vamos agora exemplificar uma operação de leitura em disco utilizando o DMA, veja a figura 5. De maneira geral, a UCP, através do *device driver*, inicializa os registradores do controlador de DMA inserindo valores que indicam o que transferir e para onde transferir.

Continuando, o controlador de DMA solicita ao controlador de disco a transferência do bloco de disco para o seu *buffer* interno. Terminada essa transferência, o controlador de disco verifica a existência de erros e, caso não haja erros, o controlador de DMA transfere o bloco para o *buffer* na memória principal. Depois de terminar a transferência, o DMA gera uma interrupção avisando ao processador que o dado já se encontra na memória principal. Podemos notar que é uma maneira de poupar a UCP de tanto trabalho.

Figura 5 - Técnica DMA



Fonte: Machado; Maia (2013)

Leitura Obrigatória

Leia o tópico 5.1.4 do livro de Andrew Tanenbaum. Nesse tópico, o autor trata sobre a DMA de maneira bem específica. Acesse o link do livro:

Disponível em:
<https://plataforma.bvirtual.com.br/Leitor/Publicacao/1233/pdf/0?code=ACkKVA8YaGRY1SbyMvjq7uTfbtv49JTITntyblZbA+ztei8cDO+Q6zNJ5GD0ejkVoRd5/zHWcm7GHDrmwlZylA=>
=. Acesso em: 2 maio 2021.

Videoaula 3

Agora assista à videoaula na qual será debatido sobre as características dos controladores.



Videoaula 3

Utilize o QRcode para assistir!

Agora assista à videoaula na qual será debatido sobre as características dos controladores.



Conhecemos até agora os mecanismos que fazem funcionar o contato do sistema operacional com os dispositivos de entrada e saída. Nos falta agora explicar um pouco sobre os dispositivos de entrada e saída.

Dispositivos de entrada e saída

Como sabem, os dispositivos de entrada e saída são responsáveis pela comunicação entre o mundo externo e o computador. Existem dispositivos apenas para

entrada e aqueles unicamente para saída. Também é possível existir um dispositivo que faça a entrada e a saída.

Os dados nos dispositivos de E/S podem ser transferidos através de blocos de informação ou caracteres. Um dispositivo de blocos é aquele que armazena informação em blocos de tamanho fixo, com cada um com seu próprio endereço. A propriedade principal de um dispositivo de bloco é que cada bloco pode ser lido ou escrito de forma independentemente de todos os outros. HD's (*Hard Disc*), *pen drives* são dispositivos de blocos mais conhecidos (TANENBAUM, 2015).

Os dispositivos de E/S de caractere são aqueles que enviam ou recebem um fluxo de caracteres, sem levar em conta qualquer estrutura de blocos. Dessa forma, a sequência de caracteres não é endereçável e não requer posicionamentos. Impressora, interfaces de redes e *mouses* são bons exemplos do tipo.

Existem várias tecnologias para a conexão de dispositivos, sempre é bom lembrar que as conexões sempre são feitas por barramentos. Os dois padrões de conexão mais conhecidos é o SCSI (*Small Computer Systems Interface*) e o padrão SATA (*Serial Ata*). Os dois modelos definem padrões de *hardware* e *software* que permitem fazer conexão computacional com vários dispositivos de fabricantes diferentes.

Um exemplo prático atual de conexão de dispositivos é o padrão USB (*Universal Serial Bus*). USB é um padrão de barramento serial para conectar vários dispositivos a um computador ou algum outro dispositivo eletrônico.

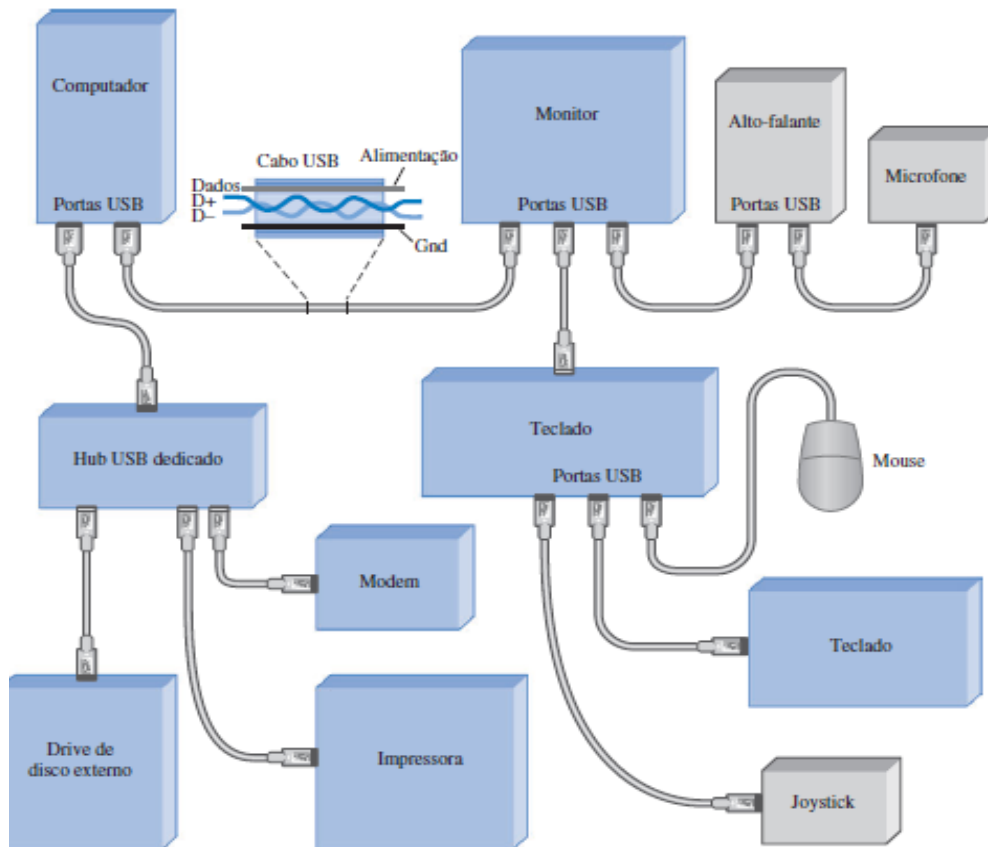
Esse padrão, usa a tecnologia conhecida como *plug and play* que permite que os dispositivos sejam conectados e desconectados sem reiniciar o computador. Também apresenta algumas características que agradam ao usuário, tais como: fornece energia a dispositivos de baixo consumo, permite que muitos dispositivos sejam usados sem a instalação de *driver* do dispositivo específico.

Mas nem tudo é um “mar de rosas” no uso do USB. Essa tecnologia limita o comprimento de um cabo entre dispositivos de alta velocidade a 5 metros e, para dispositivos de baixa velocidade o limite é de 3 metros. Apesar disso, notamos o seu uso nos mais variados equipamentos: Teclados USB, *mouse* USB, unidades USB, *webcams* USB etc.

O cabo USB tem quatro fios (nas versões mais antigas), dois para dados e dois para alimentação, e conecta ao computador a um dispositivo periférico USB, sendo que qualquer

um pode funcionar como *hub* para a conexão de outros dispositivos periféricos USB, veja o exemplo na figura 6.

Figura 6 - Exemplo de conexões com a USB



Fonte: Floyd (2007, p. 746)

A tecnologia USB tem várias versões, a última versão 4.0 foi lançada em 2019. O USB 4.0 tem o dobro da velocidade do padrão mais recente, o USB 3.1. Segue as velocidades do padrão USB até o 4.0:

- USB 1: 12 Mbps (Mega bits por segundo);
- USB 2.0: 480 Mbps;
- USB 3.0: 5 Gb/s;
- USB 3.1: 10 Gb/s;
- USB 3.2: 20 Gb/s;
- USB4: 40 Gb/s.

Por fim, a gama de componentes de entrada e saída se expande de forma exponencial. Apesar da variedade, todos seguem os mesmos princípios de comunicação para o sistema operacional.

Indicação de Vídeo

Se você for curioso, assista à explicação sobre o USB. Esse vídeo explica sobre os tipos e características dessa tecnologia. É bem esclarecedor.

Disponível em: <https://www.youtube.com/watch?v=e22JvQSjQ6s>. Acesso em: 2 maio 2021.

Aula 02 – Estudo de caso – Linux

Olá alunos! Vamos prosseguir com mais um conhecimento na matéria sistemas operacionais. Vamos dar uma pausa nas teorias sobre os SO. Agora nessa aula, vamos fazer um estudo de caso sobre o sistema operacional Linux, em especial na distribuição Ubuntu. A escolha desse sistema é relevante, pois se trata de um sistema bem conhecido e também gratuito.

Linux é um sistema operacional gratuito e de código aberto que pode ser encontrado em diversas distribuições. O *kernel* do Linux, ou seja, o núcleo deste sistema operacional, é baseado no sistema operacional Unix. Criado por Linus Torvalds, a primeira versão do Linux foi lançada em agosto de 1991.

O kernel é responsável pelo gerenciamento das tarefas em quatro áreas gerais do sistema: processos, memória, *device drivers* e chamadas de sistema.

O Linux pode ser considerado um dos avanços mais importantes do século XXI. Isso porque esse sistema capacitou centenas de dispositivos baseados em computador e também ajudou no crescimento da internet. Veja só aluno, o *Google* roda milhares e milhares de servidores Linux para fornecer sua tecnologia de busca. Além disso, o sistema operacional *Android* é baseado em Linux (NEGUS, 2014).

Os sistemas operacionais *Microsoft Windows* e o *Mac OS* com certeza são os mais usados entre os *desktops* atualmente. Porém, esses sistemas operacionais são proprietários e isso difere do Linux. Vamos ver algumas premissas dos *softwares* proprietários:

- Em sistemas proprietários você não pode ver o código para criar outro sistema operacional;
- Não pode alterar o sistema operacional em seus níveis mais básicos, caso ele não atenda suas necessidades;
- Você talvez não consiga ser capaz de executar facilmente seu *software* com um determinado sistema operacional, pois esse sistema pode ser incompatível, por exemplo, com as interfaces do seu projeto.

O Linux é um sistema operacional livre, e por isso segue 4 liberdades essenciais pregada pelo movimento da filosofia do *software* livre (FREE SOFTWARE FOUNDATION, 2021). Leia as seguintes liberdades:

1. A liberdade de executar o programa como você desejar, para qualquer propósito.
2. A liberdade de estudar como o programa funciona, e adaptá-lo às suas necessidades. Para tanto, acesso ao código-fonte é um pré-requisito.
3. A liberdade de redistribuir cópias de modo que você possa ajudar outros.
4. A liberdade de distribuir cópias de suas versões modificadas a outros. Desta forma, você pode dar a toda comunidade a chance de se beneficiar de suas mudanças. Para tanto, acesso ao código-fonte é um pré-requisito.

Precisamos ter em mente que Linux em si é apenas o kernel do sistema, ou seja, um *software* não visível, mas que tem como missão controlar as interações entre o *hardware* e outros programas da máquina.

No começo de tudo, o Linux não passava de pacotes de código fonte espalhados pela internet, onde apenas pessoas com um grau elevado de conhecimento conseguia montar um sistema operacional para uso.

As pessoas sem um conhecimento de computação queriam também montar seus sistemas operacionais, porém de uma maneira simples, dado que nem todos poderiam se dedicar ao estudo de um SO.

Nessa esteira, começou o desenvolvimento de o que chamamos atualmente de distribuições Linux. Distribuições são um conjunto elaborado de *softwares* em torno do kernel que torna o sistema operacional específico para cada tipo de usuário. Temos como exemplo de distribuição: *Ubuntu*, *Red Hat*, *Mint* etc. Segundo o site (<https://distrowatch.com/>) existem centenas de distribuição Linux.

Videoaula 1

Agora, assista à videoaula em que será mostrado o nascimento do Sistema Operacional Linux.



Videoaula 1

Utilize o QRcode para assistir!

Agora, assista à videoaula em que será mostrado o nascimento do Sistema Operacional Linux.



Indicação de Leitura

Para saber mais sobre o surgimento do Linux, leia o tópico 10.1 do livro de Andrew S.Tanenbaum – Sistemas operacionais.

Disponível

em:

<https://plataforma.bvirtual.com.br/Leitor/Publicacao/1233/pdf/0?code=xjdkNQtTJgSUOOvc6gscVviQV6myX1iUwUC9LKqEGmNhZ6dTWyOPnFHJVe4VN/pM/7+zyjXjNx4kiu1fWReROw==>. Acesso em: 02 maio 2021.

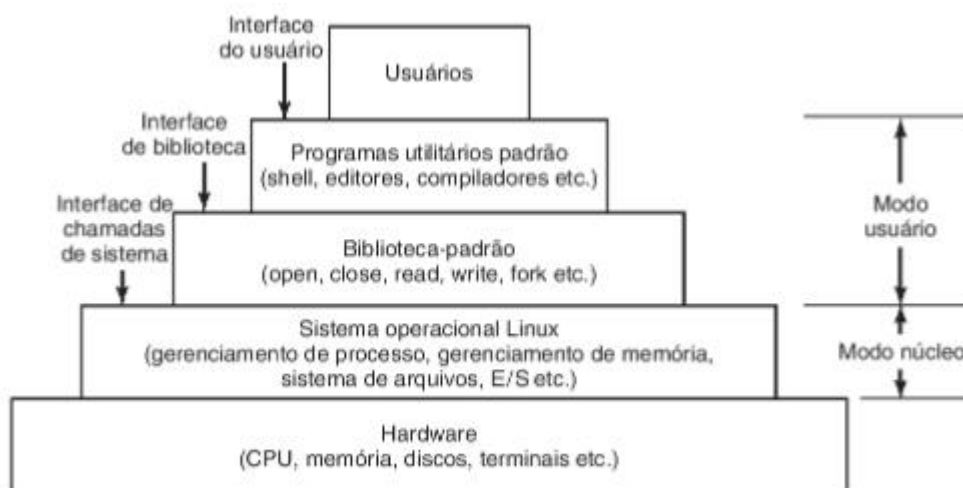
Para demonstrações em nosso estudo, será utilizada a distribuição Ubuntu. Esta distribuição é voltada para usuários finais, facilitando assim o seu uso. Portanto, recomendamos a instalação desse sistema operacional para o aprendizado.

Ubuntu é uma distribuição do sistema operacional Linux, cujo projeto foi criado em 2004 pela Canonical, empresa fundada por *Mark Shuttleworth*. O Projeto Ubuntu é guiado pela ideia de que *softwares* devem ser inclusivos, sendo possível para qualquer um, em qualquer lugar, usar, compartilhar ou modificar.

Muito bem, estamos estudamos alguns aspectos teóricos até o momento para entendermos a base da criação do Sistema operacional Linux. Chegou o momento de fazermos um estudo de caso sobre esse sistema. Vamos pegar as partes que pertencem e que fazem esse sistema funcionar.

Genericamente podemos falar que a arquitetura do Linux é dada conforme a figura abaixo:

Figura 1 - Arquitetura em sistema Linux



Fonte: Tanenbaum (2015, p. 449)

Iniciaremos falando sobre a interface do usuário, no qual ele se comunica com o sistema operacional. Em específico, falaremos do *shell*, que é um interpretador de comandos do Linux. Mas antes, precisamos pontuar algo antes aqui. Para o *desktop*, o sistema Linux usa como janela gráfica o X *Windows System* (<http://www.x.org>). Acredite! No começo não existia janela gráfica. Aliás, até nos tempos atuais existem, por exemplo em servidores.

Muitos ambientes gráficos de *desktop* diferentes estão disponíveis para escolha em Linux. Olhe que legal! Você pode escolher a interface. Observe:

GNOME — O GNOME é o ambiente de *desktop* padrão para o Fedora, o *Red Hat Enterprise Linux* e muitos outros. É considerado um *desktop* profissional, com mais foco na estabilidade do que em efeitos visuais.

K Desktop Environment — O KDE é provavelmente o segundo desktop mais popular para Linux. Ele tem mais opções que o GNOME e oferece aplicativos mais integrados. O KDE também está disponível com o Fedora, RHEL, Ubuntu e muitos outros sistemas Linux.

Xfce — O *desktop* Xfce foi um dos primeiros ambientes de *desktop* “leves”. É bom para usar em computadores antigos ou menos poderosos. Ele está disponível com o RHEL, o Fedora, o Ubuntu e outras distribuições Linux.

Prontos? Vamos começar. Como falamos, o *shell* é o interpretador de comando do Linux. Basicamente, para acessar o terminal no Ubuntu utilize os seguintes passos: utilizando um terminal: no Ubuntu pode-se utilizar o terminal acessando o menu *Aplicativos* > *Acessórios* > *Terminal* na versão clássica do Gnome, ou simplesmente digitando *terminal* na busca do Painel Inicial, observe a figura 2.

Figura 2 - Prompt aberto no Ubuntu



Fonte: o autor (2021)

Notando a figura 2, podemos ter algumas informações. O que se nota primeiramente é o *prompt* **kleber@kleber-VirtualBox:~\$**. A primeira parte do *prompt*, do lado esquerdo do sinal @, exibe o nome de usuário. Após o sinal @ é exibido o nome do computador. O sinal “:” divide o *prompt* em duas partes. À esquerda constam o nome do usuário e do computador. À direita consta o diretório do sistema de arquivos que está sendo acessado. O símbolo “~” indica que o usuário está em sua pasta pessoal.

Como pode ver, começamos a comunicação com sistema. Vamos testar comandos simples para assimilarmos o conceito. Veja a digitação dos comandos abaixo.

```
kleber@kleber-VirtualBox:~$ date
sáb 19 jun 2021 19:03:42 -03
kleber@kleber-VirtualBox:~$ pwd
/home/kleber
kleber@kleber-VirtualBox:~$ ls
'Área de Trabalho'  Downloads  Modelos  Público
Documentos          Imagens    Música    Vídeos
```

O comando **date**, sem opções ou argumentos, exibe dia, mês, data, hora, fuso horário e ano. O comando **pwd** exibe o diretório de trabalho atual **/home/kleber**. O comando **ls** lista os arquivos e diretórios no diretório atual.

A maioria dos comandos tem uma ou mais opções que você pode adicionar para mudar o comportamento deles. Em geral, as opções consistem em uma única letra, precedida por um hífen. Por exemplo, **ls -l -a -t**.

Podemos redirecionar a saída de um comando para um arquivo. Por exemplo, caso queira registrar em um arquivo o conteúdo de um diretório, pode-se usar o seguinte comando **ls -l > listagem.txt**. Digite e teste. Será criado o arquivo **listagem.txt** e seu conteúdo será a listagem do diretório corrente obtida pelo comando **ls -l**.

Se o arquivo já existir e usuário desejar apenas adicionar um conteúdo ao final do arquivo, basta utilizar o comando: **ls -l >> listagem.txt**.

Podemos canalizar também um comando. A canalização passa a saída de um comando para o outro. Por exemplo, caso queira exibir na listagem de um diretório, os arquivos que contenham a palavra “listagem”, basta executar o comando **ls | grep listagem**. Veja o resultado abaixo:

```
kleber@kleber-VirtualBox:~$ ls | grep listagem
listagem.txt
kleber@kleber-VirtualBox:~$
```

Por fim, para sair do *shell* digite **exit**.

Videoaula 2

Agora, assista à videoaula em que será mostrado o uso do Linux em diversas tarefas.



Videoaula 2

Utilize o QRcode para assistir!

Agora, assista à videoaula em que será mostrado o uso do Linux em diversas tarefas.



Observe que tivemos uma minúscula amostra de comandos que podem ser dados por meio do *shell*. Mas na verdade existe uma infinidade de possibilidades que você pode fazer com o *shell*. Pode usar o *shell* como usuário comum ou como administrador. Essa é a primeira porta de comunicação com o sistema, mas não esquecemos que podemos usar a parte gráfica também.

Além do *shell*, temos centenas de aplicativos que podemos usar na camada de interface com o usuário. Nessa camada fica a maioria de aplicativos utilizados por um usuário comum: editores de texto, navegadores etc.

Seguindo então a hierarquia dada pela figura 3, vamos falar da interface de biblioteca agora. Uma vez que o usuário precise fazer uso de rotinas para acessar o sistema, é comum utilizar-se bibliotecas padrões para isso. Assim, para fazer uma chamada *read*, por exemplo, utiliza-se alguma biblioteca de *read* que são disponibilizadas na linguagem C.

Existem dois tipos de API (*application programming interface*) no kernel:

- Uma para comunicação entre o espaço do usuário e o espaço do kernel (*kernel-user space*) – permite que os programas dos usuários acessem os recursos e os serviços do kernel através das chamadas de sistema e das subrotinas da glibc (esses dois conjuntos formam a API do LINUX);
- Para uso dentro do espaço do kernel (*kernel internal*) – permite que os subsistemas do Linux se comuniquem entre si e com os módulos adicionados ao kernel. Muitos dos cabeçalhos usados são criados para uso específico do Linux e não seguem qualquer padronização.

No tópico seguinte vamos explorar alguns itens que fazem parte da interface de chamadas de sistema.

Leitura Obrigatória

Leia o tópico 10.2 de **Andrew S. Tanenbaum** – Sistemas operacionais. Nesse tópico, o autor explana sobre as interfaces do Linux.

Disponível

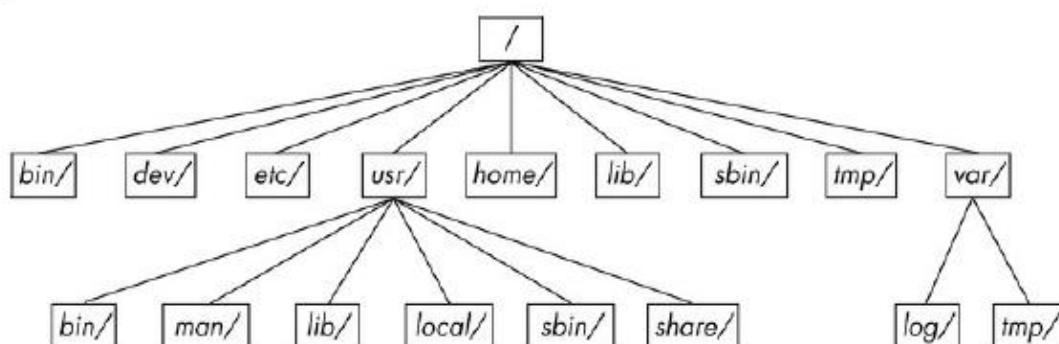
em:

<https://plataforma.bvirtual.com.br/Leitor/Publicacao/1233/pdf/0?code=xjdkNQtTJgSUOOvc6gscVviQV6myX1iUwUC9LKqEGmNhZ6dTWyOPnFHJVe4VN/pM/7+zyjXjNx4kiu1fWReROw==>. Acesso em: 2 maio 2021.

Sistemas de arquivo em Linux

Primeiramente devemos aprender como está estabelecido a hierarquia das pastas no sistemas de arquivos, observe a figura 3. Ela oferece uma visão geral simplificada da hierarquia, mostrando alguns dos diretórios em **/**, **/usr** e **/var**. Observe que a estrutura de diretórios abaixo de **/usr** contém alguns dos mesmos nomes de diretório de **/**.

Figura 3 - Hieraquia de diretórios Linux



Fonte: WARD (2016)

A seguir estão listados os subdiretórios mais importantes do diretório-raiz, segundo (WARD, 2016):

- **/bin** – contêm programas prontos para execução (também conhecidos como executáveis), incluindo a maior parte dos comandos Unix básicos como ls e cp. A maioria dos programas em /bin está no formato binário e foram criados por um compilador C, porém alguns são **shell scripts** em sistemas modernos;
- **/dev** – contêm arquivos de dispositivos;
- **/etc** – esse diretório essencial de configurações do sistema contêm arquivos de configuração de senha de usuário, de inicialização, de dispositivos, de rede e outros arquivos. Muitos itens em /etc são específicos do *hardware* do computador. Por exemplo, o diretório /etc/X11 contém configurações da placa gráfica e de sistemas de janelas;
- **/home** – contêm diretórios pessoais para usuários normais. A maioria das instalações Unix está em conformidade com esse padrão;
- **/usr** – embora pronunciado como “user”, esse subdiretório não tem nenhum arquivo de usuário. Em vez disso, ele contém uma hierarquia extensa de diretórios, incluindo a maior parte do sistema Linux. Muitos dos nomes de diretório em /usr são iguais àqueles do diretório-raiz (como /usr/bin e /usr/lib) e contêm os mesmos tipos de arquivo. (O motivo de o diretório-raiz não conter o sistema completo é principalmente histórico – no passado, servia para manter os requisitos de espaço reduzidos).

Vale notar também que o sistema Linux precisa de um tipo de estrutura de sistema de arquivo para trabalhar, vejamos os principais usados (BUENO, 2015).

Ext2: sistema de arquivos estendido (*extended filesystem*). O Ext 2 foi o primeiro sistema de arquivo usado no Linux. Apesar de não ser mais utilizado atualmente em *desktops*, uma vez que possui capacidade de recuperação de falhas limitado, ainda é indicado para dispositivos móveis de memória *flash*, como *pen drives*;

Ext3: Com relação ao Ext2, apresenta uma significativa vantagem: *Journaling*, ou seja, é um sistema de arquivos “jornalizado”. A expressão *Journaling* refere-se à capacidade de um sistema de arquivos recuperar dados em caso de falhas (queda de energia, por exemplo). Isto é feito por meio de um sistema de log que registra as alterações antes que elas ocorram, permitindo a restauração do sistema de arquivos após falhas.

Ext4: é o sistema de arquivos mais recomendado atualmente. Supera o Ext3 em performance, capacidade de armazenamento e recursos de manutenção.

Leitura Obrigatória

Leia o tópico 10.6 de **Andrew S.Tanenbaum** – Sistemas operacionais. Nesse tópico o autor explana sobre o sistema de arquivos do Linux.

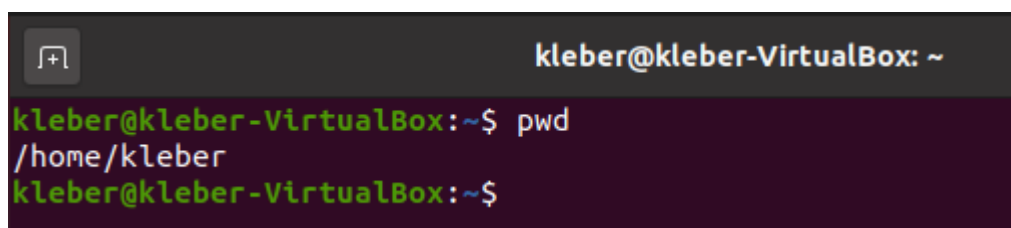
Disponível em:

<https://plataforma.bvirtual.com.br/Leitor/Publicacao/1233/pdf/0?code=xjdkNQtTJgSUOOvc6gscVviQV6myX1iUwUC9LKqEGmNhZ6dTWyOPnFHJVe4VN/pM/7+zyjXjNx4kiu1fWReROw==>. Acesso em: 2 maio 2021.

Legal né? Estamos vendo a estruturação dos sistemas de arquivos do Linux até o momento. Vamos agora navegar nesse sistema, você pode usar o *shell* ou a mesma a parte gráfica de algum aplicativo gerenciador.

No caso, vamos com o poderoso *shell*, esse modo aproxima o usuário do funcionamento do sistema. No Linux, todos os diretórios e partições estão distribuídos a partir da raiz do sistema de arquivo que é conhecido como **root** ou **/**.

Comece então com o comando **pwd** (*print working directory*). Esse comando é o que mostra o nome do diretório local em uma interface de linha de comando. Teste você mesmo! O meu resultado foi: Estou na **/home/kleber**.

A screenshot of a terminal window with a dark background. The title bar at the top reads 'kleber@kleber-VirtualBox: ~'. The terminal shows a prompt 'kleber@kleber-VirtualBox:~\$' followed by the command 'pwd'. The output of the command is '/home/kleber', which is displayed on the next line. The prompt 'kleber@kleber-VirtualBox:~\$' appears again on the following line.

```
kleber@kleber-VirtualBox:~$ pwd
/home/kleber
kleber@kleber-VirtualBox:~$
```

Vamos para outro comando muito utilizado, o **CD**. O comando **cd** muda o diretório de trabalho corrente do *shell*. Vamos para a pasta Documentos agora, repare:

```
kleber@kleber-VirtualBox: ~/Documentos
kleber@kleber-VirtualBox:~$ ls
'Área de Trabalho'  Downloads  listagem.txt  Música  Vídeos
Documentos          Imagens    Modelos      Público
kleber@kleber-VirtualBox:~$ cd Documentos
kleber@kleber-VirtualBox:~/Documentos$
```

Antes de ir para a pasta documento, repare que listei antes os arquivos do meu diretório corrente e depois executei o comando **cd Documentos**. Agora, vamos criar um diretório usando o comando **mkdir(cria diretório)** e o **rmdir(remove diretório)**. Veja:

```
kleber@kleber-VirtualBox: ~
kleber@kleber-VirtualBox:~$ mkdir teste
kleber@kleber-VirtualBox:~$ ls
'Área de Trabalho'  Downloads  listagem.txt  Música  teste
Documentos          Imagens    Modelos      Público  Vídeos
kleber@kleber-VirtualBox:~$
```

Como pode ver, usamos alguns comandos básicos, porém já tivemos a ideia para navegar no sistema de arquivo. As possibilidades de gerência do sistema de arquivos utilizando comandos é imensa. Isso fornece um poder muito grande para o operador.

Indicação de Leitura

Se você quer se aprofundar nos comandos, pode ler esse manual de mais de 500 comandos do Linux explicados.

Disponível em: https://www.linuxpro.com.br/dl/guia_500_comandos_Linux.pdf. Acesso em: 2 maio 2021.

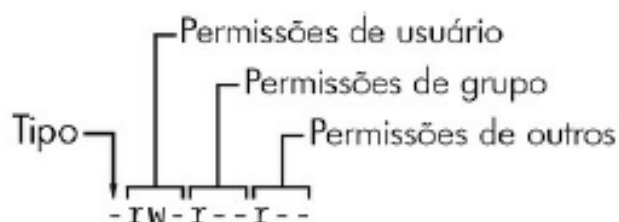
Apesar de ser mostrado em uma unidade anterior, vamos relembrar as permissões que existem no sistema de arquivo do Linux.

Dentro do sistemas de arquivos, os próprios arquivos Linux tem um conjunto de permissões que determina se você pode ler, escrever ou executar o arquivo. Executar **ls -l** faz com que as permissões sejam exibidas. Observe a saída do comando abaixo:

```
kleber@kleber-VirtualBox:~$ ls -l
total 40
drwxr-xr-x 2 kleber kleber 4096 jan 15 19:09 'Área de Trabalho'
drwxr-xr-x 2 kleber kleber 4096 jan 15 19:09 Documentos
drwxr-xr-x 2 kleber kleber 4096 jan 15 19:09 Downloads
drwxr-xr-x 2 kleber kleber 4096 jan 15 19:09 Imagens
-rw-rw-r-- 1 kleber kleber 507 jun 19 19:23 listagem.txt
drwxr-xr-x 2 kleber kleber 4096 jan 15 19:09 Modelos
drwxr-xr-x 2 kleber kleber 4096 jan 15 19:09 Música
drwxr-xr-x 2 kleber kleber 4096 jan 15 19:09 Público
drwxrwxr-x 2 kleber kleber 4096 jun 21 15:31 teste
drwxr-xr-x 2 kleber kleber 4096 jan 15 19:09 Vídeos
kleber@kleber-VirtualBox:~$
```

Na saída do comando acima é mostrado suas permissões **drwxr-xr-x**. Existem quatro partes a saber na saída do comando, conforme mostra a figura 4 abaixo.

Figura 4 - Permissões em arquivos no Linux



Fonte: Ward (2016)

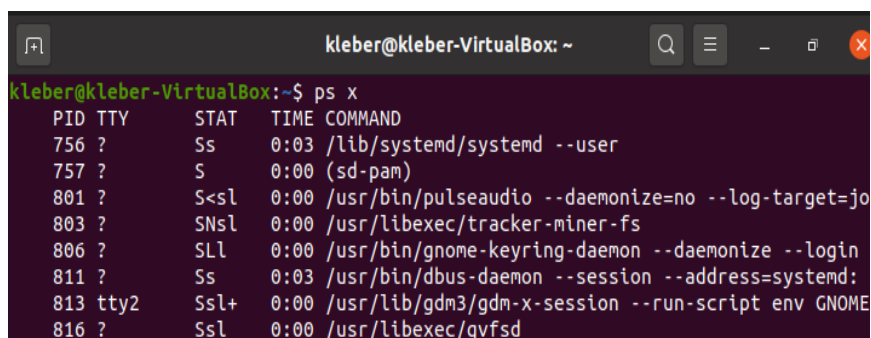
As permissões podem ser separadas em três conjuntos: usuário (*user*), grupo (*group*) e outros (***other***), nessa ordem. As nomenclaturas podem mudar conforme os autores do assunto.

Por exemplo, os caracteres **rw-** da figura 4 correspondem às permissões de usuário, os caracteres **r--** que se seguem são as permissões de grupo e os caracteres **r--** finais são as permissões para outros.

Cada conjunto de permissões pode conter quatro representações básicas:

- **r** – indica que o arquivo pode ser lido;
- **w** – indica que o arquivo pode ser escrito;
- **x** – indica que o arquivo é executável (você pode executá-lo como um programa);
- **-** – não significa nada.

Ainda dentre da interface de chamadas de sistema vamos trabalhar com a manipulação dos processos. Cada processo no sistema tem um ID de processo numérico (PID). Para uma listagem rápida dos processos em execução, basta executar comando **ps x** na linha de comando. Observe a saída do comando:



```
kleber@kleber-VirtualBox:~$ ps x
  PID TTY          STAT       TIME COMMAND
   756 ?        Ss           0:03   /lib/systemd/systemd --user
   757 ?        S            0:00   (sd-pam)
   801 ?        S<sl         0:00   /usr/bin/pulseaudio --daemonize=no --log-target=jo
   803 ?        SnsL         0:00   /usr/libexec/tracker-miner-fs
   806 ?        SLL          0:00   /usr/bin/gnome-keyring-daemon --daemonize --login
   811 ?        Ss           0:03   /usr/bin/dbus-daemon --session --address=systemd:
   813 tty2      Ssl+         0:00   /usr/lib/gdm3/gdm-x-session --run-script env GNOME
   816 ?        Ssl          0:00   /usr/libexec/gvfsd
```

Os campos desse comando estão descritos a seguir:

- PID – é o ID do processo;
- TTY – é o dispositivo de terminal em que o processo está executando;
- STAT – é o *status* do processo, ou seja, o que o processo está fazendo e em que local está a sua memória. Na execução, S significa dormindo (*sleeping*) e R significa em execução (*running*);
- TIME – é a quantidade de tempo de CPU em minutos e segundos que o processo usou até o momento. Em outras palavras, é a quantidade total de tempo gasto pelo processo executando instruções no processador;
- COMMAND – Mostra os comandos utilizados ps.

Como estamos falando de processos criados, já vem em mente também uma forma de encerrar esses processos. Para encerrar um processo, deve-se enviar um sinal com o comando **kill**. Assim: \$ **kill pid**. Um sinal é uma mensagem do kernel para um processo. Ao executar **kill**, você estará pedindo ao kernel que envie um sinal para outro processo.

Videoaula 3

Agora assista a videoaula na qual será mostrado o uso do Linux em diversas tarefas.



Videoaula 3

Utilize o QRcode para assistir!

Agora assista a videoaula na qual será mostrado o uso do Linux em diversas tarefas.



Leitura Obrigatória

Leia o capítulo 5 do *Guia Foca Linux*. Nesse capítulo são tratados diversos comandos a respeito da gerência do sistema operacional. Vale a pena conferir!

Disponível em: <https://www.guiafoca.org/guiaonline/iniciante/ch05.html>. Acesso em: 2 maio 2021.

Quando você está administrando um sistema Linux é importante ter o papel do administrador e de usuário separados. Em um sistema que é utilizado por muitas pessoas, limitar quem pode gerencia-lo deixa o sistema mais seguro. Além disso, um administrador selecionado pode impedir que pessoas estranhas danifiquem de maneira acidental o sistema.

No sistema Linux, geralmente o administrador faz login com uma conta de usuário regular e depois solicita privilégios administrativos acaso precise. Isso é feito utilizando os seguintes comandos:

Comando **su** — Muitas vezes **su** é usado para abrir um *shell* como usuário *root*. Com esse *shell* aberto, o administrador pode executar vários comandos e depois sair para voltar para um *shell* como um usuário regular.

Comando **sudo** — Com sudo, um usuário regular tem privilégios de *root*, mas apenas quando executa o comando sudo. Depois de executar esse comando com sudo, o usuário é imediatamente retornado a um shell e estará atuando como o usuário regular novamente.

Você deve ter reparado, amigo aluno, que estamos aqui mostrando alguns procedimentos básicos para a gerência de arquivos e processos. Porém, o sistema Linux fornece dezenas de controles de sistema, tais como: Monitoração de recursos como, tempo de processador, memória e disco. Dê uma parada estratégica e pesquise sobre esses itens e verá o poder do comando *shell*.

Dispositivos no Linux

Opa! Estamos aqui embaixo, na base da pirâmide do Linux. Ou seja, vamos falar do *hardware*, em específico os dispositivos. Um dispositivo é todo componente de *hardware* e que no caso do Linux é controlado pelo kernel. Exemplo: Impressoras, portas, modems, portas, *Hd's* etc.

No Linux, os dispositivos são tratados como arquivos. No caso, esses arquivos são uma espécie de arquivo especiais e podem ser encontrados no diretório */dev*. Sabendo disso, um programador pode usar operações normais de arquivo para trabalhar com um dispositivo, assim como alguns dispositivos são acessíveis a programas padrão também, não é necessário ser um programador para usar algum dispositivo.

Se você usar o comando **ls /dev** verificará vários arquivos de dispositivos em */dev*. Observe a saída do comando abaixo. Você vai notar muitos dispositivos. Mais muito mesmo.

```
kleber@kleber-VirtualBox:~$ ls /dev
autofs      loop4      stdin      tty37      ttyS0      uinput
block       loop5      stdout     tty38      ttyS1      urandom
bsg         loop6      tty        tty39      ttyS10     userio
btrfs-control loop7      tty0       tty4       ttyS11     vboxguest
bus         loop8      tty1       tty40      ttyS12     vboxuser
cdrom       loop9      tty10      tty41      ttyS13     vcs
```

Para identificar um dispositivo e visualizar suas permissões, use **ls -l**. Veja a saída dos dispositivos abaixo.

```
brw-rw---- 1 root disk      8,  0 jun 25 13:38 sda
brw-rw---- 1 root disk      8,  1 jun 25 13:39 sda1
brw-rw---- 1 root disk      8,  2 jun 25 13:39 sda2
brw-rw---- 1 root disk      8,  5 jun 25 13:39 sda5
crw-rw----+ 1 root cdrom    21,  0 jun 25 13:38 sg0
crw-rw---- 1 root disk     21,  1 jun 25 13:38 sg1
```

Observe o primeiro caractere de cada linha na saída do comando acima. Se esse caractere for **b**, **c** o arquivo será um dispositivo. Essas letras significam *block* (block), *character* (caractere) conforme iremos descrever a seguir.

Dispositivos de blocos: os programas acessam dados de um dispositivo de bloco em porções fixas. Por exemplo, *sda1* na saída do comando acima é um dispositivo de disco. Os discos podem ser facilmente divididos em blocos de dados. Como o tamanho total de um dispositivo de bloco é fixo e fácil de indexar, os processos podem fazer acessos aleatórios a qualquer bloco do dispositivo com a ajuda do kernel (WARD, 2016).

Dispositivos de caractere: os dispositivos de caracteres trabalham com *streams* de dados. Nesse caso, você apenas pode ler ou escrever caracteres em dispositivos de caractere. Os dispositivos de caractere não têm um tamanho; quando você lê de um desses dispositivos ou escreve em um, o kernel normalmente realiza uma operação de leitura ou de escrita no dispositivo. As impressoras ligadas diretamente ao seu computador são representadas por dispositivos de caractere (WARD, 2016).

Se você usava DOS/Windows antes, você acessava o drive C:. No Linux é diferente! Você vai notar um dispositivo no lugar. Vamos então listar os principais tipos de arquivos que estão no **/dev** e seus respectivos dispositivos:

/dev/hdXX

Notamos aqui as Interfaces IDEs, ou seja, tudo que tiver conectado nos cabos IDEs. Como exemplos, podemos citar *HD's* e *CD-ROM's*. O xx significa qual IDE, onde o primeiro x corresponde a qual IDE, e o segundo x (opcional) corresponde a partição. Veja a tabela a seguir:

Dispositivo

Descrição

/dev/hda	IDE Primária Master
/dev/hda1	Partição 1 da IDE Primária Master
/dev/hda2	Partição 2 da IDE Primária Master
/dev/hdb	IDE Primária Slave
/dev/hdb1	Partição 1 da IDE Primária Slave
/dev/hdb2	Partição 2 da IDE Primária Slave

c/dev/fdX

Aqui é o dispositivo equivalente ao drive de disquete, onde o x corresponde a qual *driver*. Caso você tenha apenas um *drive*, esse vai ser o **/dev/fd0**. Disquetes não são mais usados.

/dev/ttySX

Aqui são as portas seriais, estas portas seriais correspondem ao modem, ao *mouse*, e outras coisas ligadas nas portas 'COMs'. Observe:

Dispositivo	Descrição
/dev/ttyS0	COM1 (Porta serial 1)
/dev/ttyS1	COM2 (Porta serial 2)
/dev/ttyS2	COM3 (Porta serial 3)
/dev/ttyS3	COM4 (Porta serial 4)

Indicação de Vídeo

Agora assista a um vídeo sobre Dispositivos em Linux. Além de ser um ótimo vídeo ele explica a fundo esse tema.

Disponível em: <https://www.youtube.com/watch?v=CE2JSiGCgyw>. Acesso em: 2 maio 2021.

Nessa aula falamos exclusivamente sobre como gerenciar o Linux, desde o usuário até os seus dispositivos físicos. Portanto, podemos perceber que esse sistema é algo muito

flexível. Com certeza é um assunto muito amplo, por isso, sugiro a você aluno leia as leituras obrigatórias e siga as dicas dadas aqui nesse texto.

Encerramento da Unidade

Chegamos ao fim dessa unidade de maneira bem sólida. Começamos falando sobre a gerência de entrada e saída. Nesse assunto aprendemos que para o sistema operacional se comunicar com dispositivos de entrada e saída é necessário existir *softwares* para tal comunicação. Esses *softwares* são chamados de *drivers*. Esses *drivers* entram em contato com a controladora do dispositivo físico e assim a comunicação acontece.

Também fizemos um rápido estudo de caso com um sistema operacional. Para isso, passamos pelas interfaces presentes no sistema operacional Linux. Começamos pela interface do usuário, onde estudamos o importante interpretador de comandos do sistema conhecido como *shell*. No *shell* vimos alguns comandos importantes. Após isso, vimos as bibliotecas padrão utilizadas pelo sistema para acesso a determinados processos.

Por fim, chegamos à interface de chamadas de sistema, onde dentre outros itens estudamos os sistemas de arquivos. Verificamos como funciona a hierarquia do sistema de arquivos e também alguns arquivos para navegar nesse sistema. Finalizamos o estudo aprendendo sobre como o Linux trata os dispositivos, notamos que os dispositivos são tratados como mero arquivos.

Por incrível que pareça, esse assunto irá acompanhar você até o final da sua carreira, caso você queira trabalhar na área tecnológica. Portanto, vale a pena se esforçar e aprender bastante sobre sistemas operacionais.



Videoaula Encerramento

Utilize o QRcode para assistir!



Referências

BUENO, Fabrício. **Ubuntu em Linha de Comando**. Garopaba: [s.n], 2015. 153 p.

FREE SOFTWARE FOUNDATION (ed.). **O Sistema Operacional GNU**. Disponível em: <http://www.gnu.org/gnu/gnu.html>. Acesso em: 19 jun. 2021.

FLOYD, Thomas. **Sistemas Digitais: Fundamentos e Aplicações**. 9. ed. São Paulo: Bookman, 2007. 645 p.

MACHADO, Francis Berenger; MAIA, Luiz Paulo. **Arquitetura de Sistemas Operacionais**. 5. ed. Rio de Janeiro: LTC, 2013. 266 p.

NEGUS, Christopher. **Linux – a Bíblia**. Tradução Edson Furmankiewicz. 8. ed. Rio de Janeiro: Alta Books, 2014. 852 p.

TANENBAUM, Andrew S. **Sistemas Operacionais Modernos**. 4. ed. São Paulo: Pearson Education do Brasil, 2015. 864 p.

WARD, Brian. **Como o Linux funciona**. São Paulo: Novatec, 2016. 414 p.



UNIFIL.BR