UNIX

1. HISTÓRICO

Na década de 60, inúmeros esforços foram direcionados para o desenvolvimento de um verdadeiro sistema operacional de tempo compartilhado que viesse a substituir os sistemas batch da época. Foi nesse período que o MIT (Massachussets Institute of Technology), a Bell Labs (uma subsidiária da AT&T) e a General Electric (GE) se uniram para desenvolver o MULTICS (MULTiplexed Information and Computing Service).

Por diversas razões o projeto MULTICS não foi levado adiante, ficando quase que totalmente esquecido não fosse por Ken Thompson, um dos pesquisadores da Bell Labs. Com base no MULTICS, Thompson desenvolveu sua própria versão do sistema operacional, que veio a se chamar UNICS (UNiplexed Information and Computing Service) e, posteriormente, Unix.

O sistema Unix foi inicialmente desenvolvido em assembly para um minicomputador PDP-7 da Digital. A decisão de escrever o sistema Unix em uma linguagem de alto nível, tornando-o, assim, portável, ocorreu em 1972, quando Thompson e Dennis Ritchie, também da Bell Labs, reescreveram o sistema para uma linguagem chamada B. Essa linguagem foi extensamente modificada, dando origem à linguagem C, na qual, mais tarde, o Unix seria todo reescrito e portado para um minicomputador PDP-11.

Mesmo tendo criado e desenvolvido o Unix, a Bell Labs não podia comercializá-lo na época devido às leis americanas antimonopólio, que impediam seu envolvimento no mercado de computadores. Apesar dessa limitação, as universidades poderiam licenciar o Unix, recebendo inclusive o código fonte do sistema. Como a grande maioria das universidades utilizava computadores PDP-11, não existiam dificuldades para se adotar o Unix como plataforma padrão no meio acadêmico. Em meados da década de 80, o Unix já podia ser encontrado não só no meio acadêmico, mas também na indústria, sendo oferecido por inúmeros fabricantes.

A partir de 1984, a AT&T foi autorizada pelo governo americano a comercializar o sistema que tinha desenvolvido. Diversas versões foram lançadas, sendo a versão System V Release 4 (SVR4) a que se estabeleceu. Enquanto isso, a Universidade de Berkeley, Califórnia, desenvolveu sua própria versão do sistema, batizada de 1 BSD (First Berkeley Software Distribution). Outras versões se sucederam, chegando a 4.4BSD. O Unix de Berkeley introduziu inúmeros melhoramentos no sistema, merecendo destaque o protocolo TCP/IP, hoje um padrão de fato na indústria.

Várias tentativas vêm sendo feitas visando unificar as versões do Unix de Berkeley (BSD) e da AT&T (System V), além das inúmeras outras implementações oferecidas pelo mercado. A primeira tentativa séria nesse sentido foi dada pelo IEEE (Institute of Electrical and Electronics Engineers), através do seu comitê POSIX (Portable Operating System Unix). Como resultado desse trabalho, surgiu o padrão IEEE 1003, que, entre outras definições, estabelece um conjunto de chamadas padrão. Apesar de inúmeras tentativas de padronização e unificação do Unix, sempre aparecem grupos descontentes que tentam impor seus próprios padrões.

Ainda é possível ter acesso ao código fonte do Unix, possibilitando que ele seja estudado e alterado, utilizando-se um simples PC. Existem dois sistemas de domínio público (MINIX e o LINUX) que podem ser copiados pela Internet ou comprados pelo correio por quantias simbólicas. O MINIX foi desenvolvido na década de 80 pelo professor Andrew Tanenbaum, da Vrije Universiteit em Amsterdã, sendo utilizado em inúmeras universidades do mundo com fins educacionais. O LINUX foi desenvolvido pelo finlandês Linus Torvalds na década de 90 e em pouco tempo passou a ser utilizado tanto para fins acadêmicos como comerciais.

As implementações do sistema Unix variam conforme suas versões, plataformas de hardware e fabricantes, principalmente se comparadas às versões originais de Berkeley e da AT&T. No decorrer deste capítulo, tentaremos dar uma visão geral do sistema, sem entrar em detalhes das diferentes versões.

2. CARACTERÍSTICAS

O sistema operacional Unix é um sistema interativo, multiusuário, multiprogramável/multitarefa, que suporta arquitetura com múltiplos processadores e implementa o mecanismo de memória virtual. Entre as muitas razões sugeridas para explicar o sucesso alcançado pelo Unix, incluem-se estas:

- O sistema foi escrito em uma linguagem de alto nível, tornando-o fácil de compreender, alterar e portar para outras plataformas;
- O sistema oferece aos usuários primitivas que permitem que programas complexos sejam construídos a partir de programas bastante simples;
- O sistema de arquivos implementa uma estrutura hierárquica de fácil manutenção e implementação;
- O sistema oferece uma interface consistente para dispositivos periféricos.

Para se conectar ao sistema é necessária uma identificação de usuário (login) e uma senha de acesso, como mostra a Fig. 1. Havendo sucesso, o shell (interpretador de comandos) apresenta um prompt, indicando que está pronto para receber comandos do usuário.

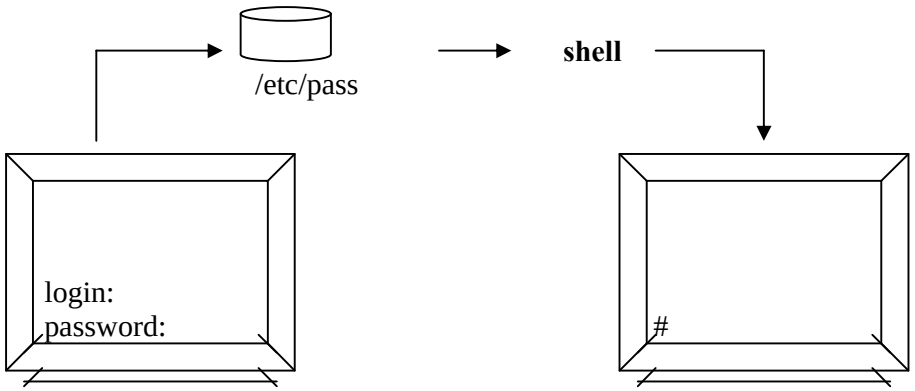


Fig. 1 - Conexão ao sistema.

Devido às várias implementações do Unix, três interpretadores de comandos se tornaram populares.

- o Bourne Shell (sh) foi o primeiro shell disponível e pode ser encontrado em todas as versões do Unix.
- o C Shell (csh) é o padrão para o BSD e está incluído na maioria das versões do Unix.
- o Korn Shell (ksh) é o padrão para o System V, estando também presente nas versões de Unix mais modernas.

Todos esses shells descritos são interfaces orientadas a caracter, mas existem várias interfaces gráficas disponíveis, como :

- o X Windows System (MIT)
- o Open Look (Sun e AT&T)
- o DECwindows (Digital).

O sistema operacional Unix faz distinção entre letras maiúsculas e minúsculas, tratando-as como caracteres diferentes.

3. ESTRUTURA DO SISTEMA

O sistema Unix é famoso por sua portabilidade, sendo oferecido por diversos fabricantes para diferentes plataformas de hardware. A razão dessa portabilidade está na estrutura do sistema, que pode ser comparada à de uma pirâmide (Fig. 2).

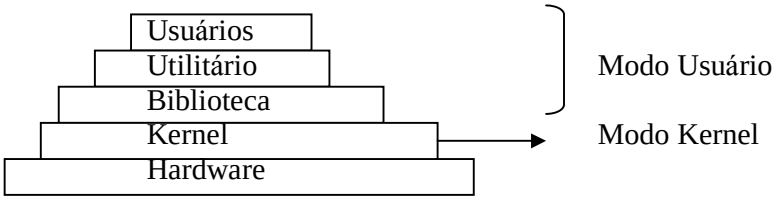


Fig. 2 - Estrutura do Unix.

Hardware

Na base da pirâmide está o hardware, envolvendo o processador, a memória e os dispositivos de entrada/saída.

Kernel

O kernel é responsável por controlar o hardware e fornecer as system calls para que os programas tenham acesso aos serviços do sistema, como criação e gerência de processos, gerência de memória, sistema de arquivos, gerência de E/S etc.

O kernel pode ser dividido em duas partes :

- A parte dependente do hardware consiste nas rotinas de tratamento de interrupções, device drivers, parte do gerenciador de memória, ou seja, tudo que normalmente deve ser reescrito quando se está portando um sistema Unix para uma nova plataforma.
- A parte independente do hardware não tem a princípio nenhum vínculo com a plataforma onde está sendo executada, sendo responsável pelo tratamento das system calls, gerência de processos, escalonamento, pipes, paginação e swapping, gerência de arquivos etc.

Parte do sistema operacional residente em memória, o kernel, comparado com outros sistemas operacionais, oferece um conjunto relativamente pequeno de primitivas, a partir das quais podem ser construídas rotinas de maior complexidade. A estratégia de criar um sistema modular foi muito importante no processo de desenvolvimento do Unix, pois novos programas utilitários podem ser facilmente integrados ao sistema, sem que o kernel tenha que sofrer qualquer tipo de alteração.

Biblioteca

Para cada system call existe um procedimento de biblioteca (library procedure) que permite esconder os detalhes da mudança de modo de acesso (trap), fazendo com que as system calls pareçam chamadas de procedimentos comuns.

Importante ressaltar que o **POSIX** define a interface com a biblioteca de procedimentos e não a interface com as system calls.

Utilitários

A camada mais externa do sistema é a interface com o usuário, conhecida como **shell**. O shell é simplesmente um programa que lê os comandos do usuários, verifica se a sintaxe está correta e passa o controle para outros programas que fazem parte do sistema operacional.

Os usuários se comunicam com o sistema geralmente através de comandos e utilitários, como compiladores, editores, formatadores, etc. O **vi** (visual editor), por exemplo, permite a criação e edição de arquivos textos, como programas ou scripts. O utilitário **awk** permite a manipulação e formatação de textos.

4. PROCESSO

O Unix, como um sistema multiprogramável, suporta inúmeros processos, que podem ser executados concorrentemente ou simultaneamente.

Um processo é criado através da system call **FORK**. O processo que executa o fork é chamado de processo pai, enquanto que o novo processo é chamado processo filho. Cada subprocesso tem seu próprio espaço de endereçamento, ou seja, as variáveis do processo pai não são visíveis por seus filhos. A cada processo do sistema são associados:

PID (Process Identification)	nº que identifica unicamente um processo.
PPID (Parent Process Identification)	nº de identificação do processo pai.
UID (User Identification)	nº de identificação do usuário que criou o processo.
GID (Group Identification)	nº do grupo do dono do processo.

O processo 0 (zero) é o único processo que não tem um pai, sendo criado durante a ativação do sistema. O processo 1, conhecido como **init**, é criado pelo processo 0 de todos os outros processos do sistema. Ao iniciar uma sessão no Unix, o processo init cria um novo processo para a execução do programa shell. Quando um comando é solicitado, dois eventos podem ocorrer:

- o shell cria um processo onde o comando será executado
- ou o próprio shell executa o comando.

Além do processo shell, um usuário pode criar vários outros processos, em uma mesma sessão. A criação de um novo processo pelo processo corrente é denominada **forking**. Uma vez criado o processo filho, o processo pai permanece hibernando até que o processo filho complete a execução de suas tarefas. A qualquer momento, um processo pode criar outros processos, que por sua vez podem criar outros e assim por diante.

Cada processo recebe também uma prioridade, que é ajustada dinamicamente pelo kernel. Processos são escalonados de acordo com a sua prioridade.

Existem processos especiais, chamados **daemon**, responsáveis por tarefas especiais no sistema, como, por exemplo, escalonamento de processos batch (**cron**), gerência de filas de impressão, log de erros etc. O próprio sistema cria os daemons durante a sua inicialização.

Processos no Unix podem se comunicar através do mecanismo de troca de mensagens, utilizando canais de comunicação, conhecidos como **pipes**. Suponha que você queira utilizar a saída de um comando como entrada de um segundo comando. Uma maneira é criar um arquivo temporário para

armazenar a saída do primeiro comando, utilizar o mesmo arquivo como entrada no comando seguinte e, posteriormente, eliminar o arquivo temporário.

O Unix permite que esse tipo de processamento seja implementado de forma bastante simples, utilizando o conceito de **pipeline**. Através da barra vertical (|), ou pipe, a saída de um comando pode ser direcionada para a entrada de um outro comando sem a utilização de arquivos temporários. A idéia do pipeline pode ser comparada a uma linha de montagem, onde uma linha de produção fornece insumos para outra linha, sucessivamente (Fig. 3).

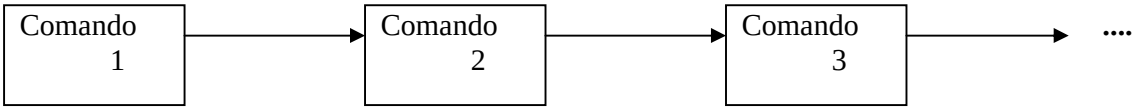


Fig. 3 – Conceito de pipe

Quando o shell aguarda o término da execução de um comando, dizemos que este comando está sendo executado em **foreground**. Por outro lado, quando o prompt é liberado para o usuário antes do término da execução de um comando, dizemos que este comando está sendo executado em **background**.

Um processo em background é um job independente, tendo as mesmas características de um processo em foreground, com uma única exceção: somente os processos em foreground podem executar E/S via terminal. Como um processo background não possui interação com o terminal, podemos direcionar a saída, por exemplo, para um arquivo em disco. Este tipo de processamento é freqüentemente utilizado quando um job tem uma execução demorada, não necessitando de entrada interativa de dados, ou quando o job executa tarefas como compilação, ordenação ou cálculos matemáticos.

A maioria das versões do Unix não implementa a filosofia de filas batch. Um usuário pode ter vários processos em foreground e vários em background executando concorrentemente, constituindo, desta maneira, um ambiente multitarefa.

5. SISTEMAS DE ARQUIVOS

O sistema de arquivos do Unix é baseado em uma estrutura de diretórios em árvore, sendo o diretório raiz (root) representado pela (Fig. 4).

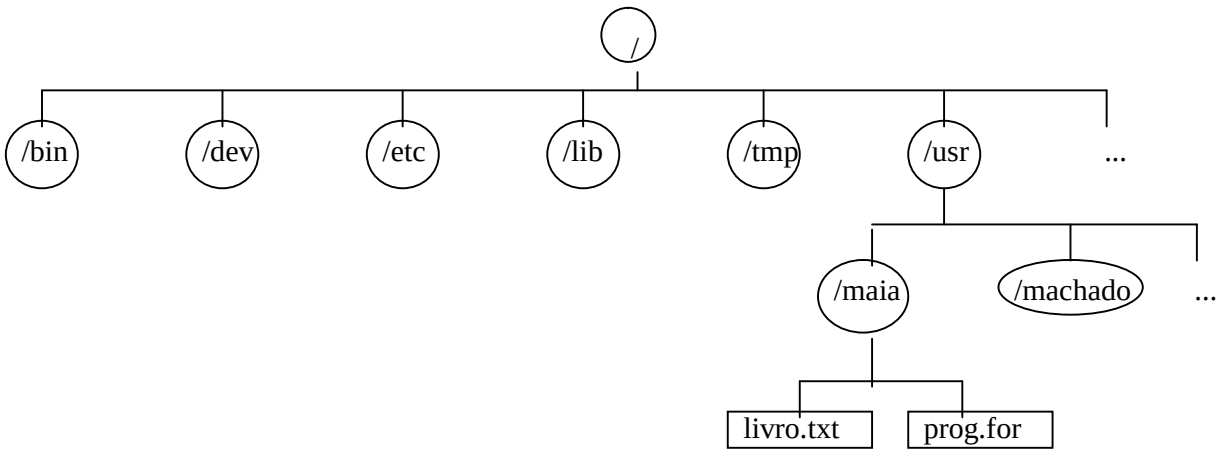


Fig. 4 - Estrutura de diretórios.

No Unix não existe uma dependência entre a estrutura lógica dos diretórios e o local onde os arquivos estão fisicamente armazenados (Fig. 5). Dessa forma, é possível adicionar novos discos ao sistema de arquivos sempre que necessário.

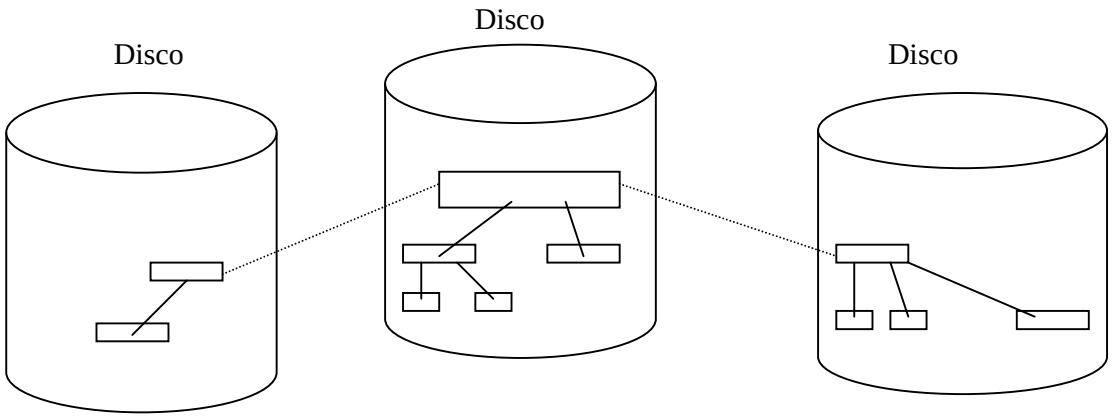


Fig. 5 - Sistema de arquivo.

Esse modelo permite que uma estrutura de diretórios seja formada por diferentes discos, inclusive residentes em estações remotas. O Network File System (NFS) e o Remote File System (RFS) são padrões para a implementação de sistemas de arquivos remotos, respectivamente, para o Unix BSD e o System V.

A estrutura do sistema de arquivos do Unix varia conforme a implementação. No Unix File System (UFS), qualquer disco deve ter a estrutura semelhante à descrita na Fig. 6.

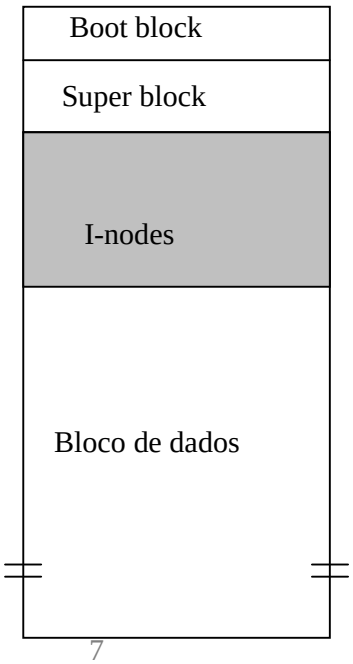


Fig. 6 - Estrutura do sistema de arquivo.

Estrutura	**Descrição**
Boot block	Quando utilizado, serve para realizar a carga do sistema (boot).
Super block	Possui informações sobre a estrutura do sistema de arquivos, incluindo o número de i-nodes, o número de blocos do disco e o início da lista de blocos livres. Qualquer problema com o super block ocasionará a perda de todo o sistema de arquivos.
I-node	Cada i-node descreve um único arquivo, contendo seus atributos, como seu dono, grupo, proteção, permissões de acesso, tipo do arquivo, além da localização dos blocos de dados no disco.
Bloco de dados	Contém os dados propriamente ditos de arquivos e diretórios. Um arquivo pode ser formado por um ou mais blocos, contíguos ou não.

Arquivos e Pathnames

No Unix existem três tipos de arquivos: arquivos ordinários, arquivos diretórios e arquivos especiais.

- Os **arquivos ordinários** são os que contêm dados binários ou caracteres ASCII. Por exemplo, um arquivo criado através de um editor de texto é um arquivo do tipo ordinário. Uma subclasse de arquivos ordinários são os arquivos hidden (oculto), cujos nomes começam sempre com um ponto e possuem funções especiais. Esses arquivos recebem a denominação de hidden porque normalmente não podem ser vistos em uma consulta a um diretório.

- Os **arquivos diretórios** são responsáveis pela manutenção da estrutura hierárquica do sistema de arquivos. As informações são armazenadas em arquivos ordinários, agrupados em diretórios, que, por sua vez, também são agrupados em outros diretórios. Todo diretório contém os nomes de arquivos ponto (.) e dois pontos (..), que correspondem, respectivamente, ao próprio diretório e ao seu diretório pai. Todo usuário possui um diretório default de login, denominado **home directory**.

- O Unix trata todos os dispositivos físicos do sistema como **arquivos especiais**. Cada dispositivo, como terminais, unidades de fita, disco e impressoras, possui um arquivo especial associado a ele. Estes arquivos residem no diretório /dev e especificam o tipo de dispositivo (como terminal ou impressora) e suas características (como configuração e densidade de gravação). Os arquivos especiais podem ser acessados da mesma forma que os arquivos ordinários.

Um **pathname** representa a seqüência de diretórios e subdiretórios, separados por barra (/), que possibilitam a localização de um arquivo dentro da estrutura de diretórios. Existem dois tipos de pathname: absoluto e relativo.

- Um **pathname absoluto** indica a seqüência completa de diretórios a que se deseja ter acesso, a partir da raiz do sistema de arquivos, o diretório root (/).

- Um **pathname relativo** determina a localização de um arquivo a partir do diretório corrente. A especificação de um arquivo que não comece com uma barra (/) é dita uma especificação relativa.

Links

O link proporciona um mecanismo para compartilhamento de arquivos. Um link é uma entrada em um diretório, que faz referência a um arquivo em um outro diretório.

Um arquivo pode possuir múltiplos links, de forma que diferentes nomes de arquivos podem ser utilizados por diversos usuários no acesso às informações de um único arquivo. O número de links de um arquivo é o número de diferentes nomes que ele possui. As vantagens da utilização de links são:

- representam uma economia de espaço em disco, uma vez que existe uma única cópia dos dados
- diversos usuários têm acesso sempre à última versão do arquivo
- é mais conveniente definir links para arquivos a que acessamos freqüentemente do que criar uma cópia em nosso diretório ou usar seu pathname absoluto a cada acesso.

Existem dois tipos de links: soft links e hard links.

- um hard link compartilha o i-node do arquivo original. Por isso, se o arquivo original for eliminado de um diretório, seu conteúdo ainda poderá ser acessado.
- um soft link (link simbólico) é simplesmente um sinônimo para o nome do arquivo original. Se este for eliminado, o link continuará existindo, mas o conteúdo do arquivo não poderá mais ser acessado.

Proteção

Cada arquivo no Unix apresenta três níveis de proteção definidos por três categorias de usuários. Todo arquivo ou diretório tem um dono (**user**) e pertence a um grupo (**group**). Qualquer usuário do sistema que não seja o dono do arquivo e não pertença ao grupo do arquivo enquadra-se na categoria outros (**others**). O administrador do sistema, o chamado **superusuário** (**root**), não pertence a nenhuma das categorias acima, tendo acesso irrestrito a todos os arquivos do sistema.

Para cada categoria de usuário, três tipos de acesso podem ser concedidos:

- leitura (read)
- gravação (write)
- execução (execute)

O sistema Unix não faz distinção entre os acessos de escrita e eliminação, ou seja, um arquivo que pode ser alterado também pode ser eliminado. As tabelas abaixo mostram as permissões para arquivos e diretórios:

Arquivo	**Descrição**
r	Permissão para ler e copiar o arquivo
w	Permissão para alterar ou eliminar o arquivo
x	Permissão para executar o arquivo

Diretório	Descrição
r	Permissão para ler o conteúdo do diretório (listar o nome dos arquivos)
w	Permissão para criar, eliminar e renomear arquivos no diretório
x	Permissão de busca: o usuário que não possuir esta permissão não poderá se posicionar no diretório nem acessar os arquivos na árvore abaixo deste diretório

A proteção assinalada ao arquivo diretório determina o primeiro nível de proteção para todos os arquivos deste diretório e tem prioridade sobre as proteções associadas individualmente a cada arquivo. Por exemplo, caso um usuário não tenha permissão de gravação em um arquivo diretório, não poderá eliminar nenhum arquivo deste diretório, ainda que as proteções dos arquivos indiquem o contrário.

Gerência de Entrada/Saída

A gerência de entrada/saída no Unix é implementada através de device drivers, um device para cada dispositivo. Os device drivers são acoplados ao sistema operacional quando o kernel é gerado. Sempre que um novo dispositivo é acrescentado ao sistema, o driver correspondente deve ser acoplado ao núcleo.

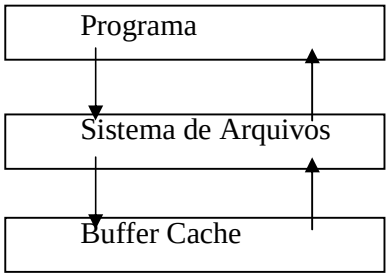
O acesso aos dispositivos, como terminais, discos, impressoras e a rede, é integrado ao sistema de arquivos, através de arquivos especiais. Cada dispositivo está associado a um ou mais arquivos especiais, localizados normalmente no diretório /dev. Por exemplo, uma impressora pode ser o arquivo /dev/1p, um terminal /dev/tty1 e uma interface de rede /dev/net.

No Unix, todas as operações de E/S são realizadas como uma seqüência de bytes, não existindo o conceito de registro ou métodos de acesso. Isso permite enviar uma mesma seqüência de caracteres para diferentes dispositivos de saída, como um arquivo em disco, terminal, impressora ou linha de comunicação. Dessa forma, as system calls de E/S podem manipular qualquer tipo de dispositivo de forma uniforme.

Os arquivos especiais podem ser acessados da mesma forma que qualquer outro arquivo, utilizando simplesmente as system calls de leitura e gravação. O Unix trabalha com dois tipos de operações de entrada e saída: uma orientada a blocos e outra orientada a caracter:

- as **operações orientadas a bloco** estão geralmente associadas a dispositivos com altas taxas de transferência, como discos, e têm o objetivo de minimizar o número de transferências entre o dispositivo e a memória, utilizando buffer caches (Fig. 7). Por exemplo, quando uma operação de leitura a disco é realizada, um bloco é transferido para a memória e, posteriormente, processado.

- d**ispositivos orientados a caracter** estão associados normalmente a dispositivos lentos, como terminais, onde a taxa de transferência entre o dispositivo e a memória é realizada caracter a caracter.



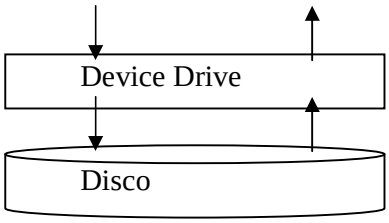


Fig. 7 - Buffer cache.