Cap. 3 - Sistema de Arquivo do UNIX

- **objetivo:** com base nos conceitos de processo apresentados nas seções anteriores, detalharemos a estruturação do sistema de arquivo no UNIX.

3.1 - *Cache* de *Buffers*

- estrutura do *pool* de *buffers* e cenário para recuperação de um *buffer*.

3.2 - Representação Interna dos Arquivos

- estrutura e acesso aos *inodes*, estrutura de um arquivo regular.

3.3 - Estrutura do Diretório

3.4 - Super-Bloco

3.5 - Atribuição de *Inode* a um Novo Arquivo

3.6 - Alocação de Blocos no Disco

3.7 - Chamadas do Sistema de Arquivos

... Cap. 3 - Sistema de Arquivo do UNIX

- **sistema de arquivo** - permitir que os processos armazenem novas informações ou que recuperem informações previamente armazenadas;
- ... para isso, o núcleo do sistema operacional necessita trazer informações auxiliares para a memória, por exemplo, o super-bloco e os *inodes* disponíveis;
- ... estas informações são manipuladas de forma transparente para o usuário;
- ... com o intuito de minimizar a freqüência de acesso a disco, o núcleo gerencia um *pool* interno de *buffers* denominado de *cache* de *buffers*;
- ... essa estrutura de *software* não deve ser confundida com a memória *cache* em *hardware* utilizada para acelerar as referências à memória principal.

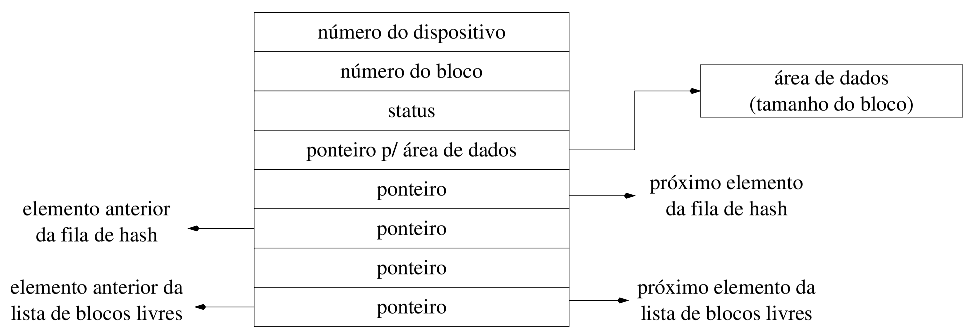
3.1 - Cache de Buffers

- **objetivo** - minimizar a latência das operações de entrada/saída.
- **cache de buffers** - blocos de memóoria utilizados pelo sistema operacional para armazenamento temporário de dados numa operação de entrada/saída.
- ... cada *buffer* é constituído de duas partes: um segmento de memória utilizado para abrigar os dados que são lidos/escritos e um cabeçalho que identifica o *buffer*;
- ... a visão que o núcleo tem do sistema de arquivos no disco é um visão lógica, assim faz-se necessário o *driver* para mapear a esta visão lógica na estrutura física;
- ... nessa estrutura, o núcleo enxerga como sendo formado por vários blocos (*buffers*) de disco, sendo seu conteúdo identificado pelo campos presentes no cabeçalho;
- ... na verdade, o *buffer* é a cópia na memória de um bloco de disco, cópia esta que é mantida até o instante em que o núcleo decide mapear um outro bloco no *buffer*;

✍ um bloco do disco não pode ser mapeado em mais de um *buffer* ao mesmo tempo.

... 3.1 - Cache de Buffers

- o cabeçalho de um *buffer* contém 04 classes de informação: identificação, posicionamento, estado e posição dentro do segmento de memória;
- **identificação** - composto pelos campos número do dispositivo e número do bloco;
- **posicionamento** - apontadores para a lista de blocos livres e fila de hash;
- **posição** - apontador para a área de dados correspondente ao *buffer*;
- **estado** - indica a situação do *buffer* (*locked*, dados válidos, *buffer* com escrita retardada, leitura/escrita do conteúdo do *buffer* no disco, ...).



3.1.1 - Estrutura do Pool de Buffers

- ... trata da alocação dos *buffers*, e neste sentido um *buffer* alocado só é realocado a outro bloco se ele for o menos recentement usado;

- ... para implementação dessa política o núcleo mantém a lista de *buffers* livres organizados segundo a ordem do uso menos recente;

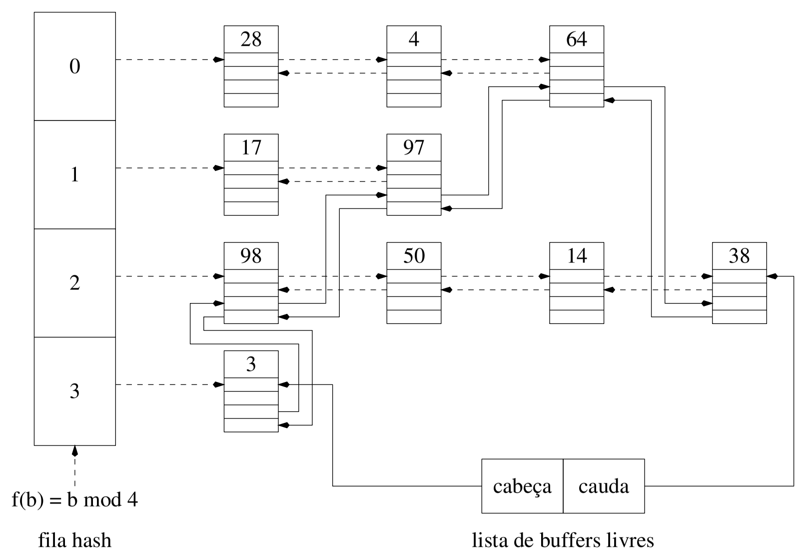
- ... quando o núcleo retorna um *buffer* ao *pool*, ele normalmente o coloca no final da lista sendo que, ocasionamente é colocado na cabeça da lista (em caso de erro);

- ... quando o núcleo necessita acessar um bloco do disco, ele procura um *buffer* que possua a combinação adequada número do dispositivo e número do bloco;

- ... para economizar pesquisa na *cache* de *buffers*, os *buffers* são separados em filas e estas são espalhadas por uma função *hash* que tem como parâmetro o número do dispositivo e o número do bloco;

... 3.1.1 - Estrutura do Pool de Buffers

- ... todo *buffer* está na fila *hash*, e assim não há um significado para sua posição na fila dado que pode encontrar-se simultaneamente na lista de *buffers* livres, caso seu estado seja livre, ou na fila *hash* (p. ex., *buffer* 64).



3.1.2 - Cenários para Recuperação de um Buffer

- ... o *cache* de *buffers* é manipulado por um conjunto de algoritmos que são invocados pelo sistema de arquivos, com o qual determina o número do dispositivo e número do bloco a ser acessado, p. ex., quando um processo solicita leitura de dados;

- ... quando da leitura ou escrita de dados de um determinado bloco do disco, o núcleo verifica se o bloco encontra-se no *cache* de *buffers* e, caso não se encontre, um *buffer* livre é atribuído ao bloco.

✯ ... os algoritmos para leitura/escrita em blocos do disco usam o algoritmo *getbkl* para alocação de *buffers* a partir do *pool* de *buffers*;

✯ ... além do *getblk*, existem outros 04 algoritmos que operam o *cache* de *buffers*:

- *brelse* - libera um bloco, retornando-o à lista de blocos livres;
- *bread* - lê sincronamente o bloco de disco para o *buffer*;
- *breada* - lê assincronamente o bloco de disco para o *buffer*;
- *bwrite* - escreve um bloco do *buffer* para o disco.

... 3.1.2 - Cenários para Recuperação de um Buffer

```
Algoritmo GetBlk
Entrada: número do bloco, número da unidade lógica
Saída: buffer trancado pronto para receber bloco do disco
{
  do {
    if( bloco na fila de hash ) {
      if( buffer ocupado ) {                        /* Cenário #5 */
         sleep( evento "buffer desocupado" );
        continue;
      }
      marca buffer "ocupado";                       /* Cenário #1 */
      remove buffer da lista de buffers livres;
      retorna buffer;
    }
    else {
      if( lista de buffers livres vazia ) {            /* Cenário #4 */
        sleep( evento "lista de buffers livres não vazia" );
        continue;
      }
    remova buffer da lista de buffers livres;
    if( buffer marcado "delayed write" ) {           /* Cenário #3 */
      escreva assincronamente o conteúdo do buffer em disco;
      continue;
    }
    reposiocione buffer na fila da hash;             /* Cenário #2 */
    retorne buffer;
  } while( true );
}
```

1 - o núcleo encontra o buffer correspondente ao bloco na fila hash e o buffer está livre. Neste caso, o algoritmo marca o buffer como "ocupado", remove-o da lista de buffers livres e retorna um ponteiro para o buffer;

2 - o núcleo não encontra o buffer correspondente ao bloco na fila hash e aloca um buffer da lista de buffers livres, reposicionando-o na fila de hash;

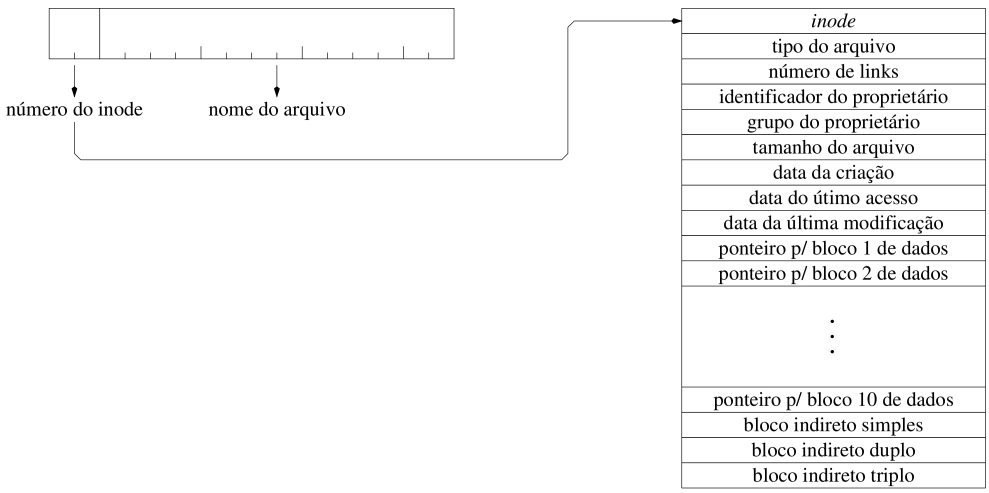
3 - o núcleo não encontra o buffer correspondente ao bloco na fila hash e, na tentativa de alocar um buffer da lista de buffers livres, encontra um buffer na lista de buffers marcado com "delayed write" Neste caso, o núcleo deve escrever o buffer no disco e continuar a procura de um buffer livre para o bloco;

4 - o núcleo não encontra o buffer correspondente ao bloco na fila hash, e a lista de buffers livres encontra-se vazia. Neste caso, o processo é bloqueado até que outro processo devolva um buffer para a lista de buffers livres;

5 - o núcleo encontra um buffer correspondente ao bloco na fila hash, mas o seu buffer está ocupado (p.ex., outro processo está acessando este bloco). Neste caso, o processo é bloqueado até que o outro processo conclua a operação sob este bloco.

3.2 - Representação Interna dos Arquivos

- todo arquivo no UNIX Sytems V contém um único *inode*, que por sua vez possui informações necessárias para um processo acessar um arquivo;
- ... a referência do arquivo é feita pelo nome do arquivo e, através deste, o núcleo determina o *inode* correspondente;



... 3.2 - Representação Interna dos Arquivos

✯ os algoritmos envolvidos na manipulação dos *inodes*, utilizam algoritmos que operam o *cache* de *buffers* discutido anteriormente:

- *iget* - retorna um *inode* previamente identificado, possivelmente através da sua leitura do disco via o *cache* de *buffers*;
- *iput* - libera o *inode*;
- *bmap* - define os parâmetros do núcleo para o acesso a um arquivo;
- *namei* - converte um nome de arquivo (*pathname*) no *inode* correspondente;
- *alloc* - aloca blocos do disco para os arquivos;
- *free* - libera blocos de disco;
- *ialloc* - aloca *inodes* para os arquivos;
- *ifree* - libera *inodes*.

3.2.1 - Estrutura do Inode

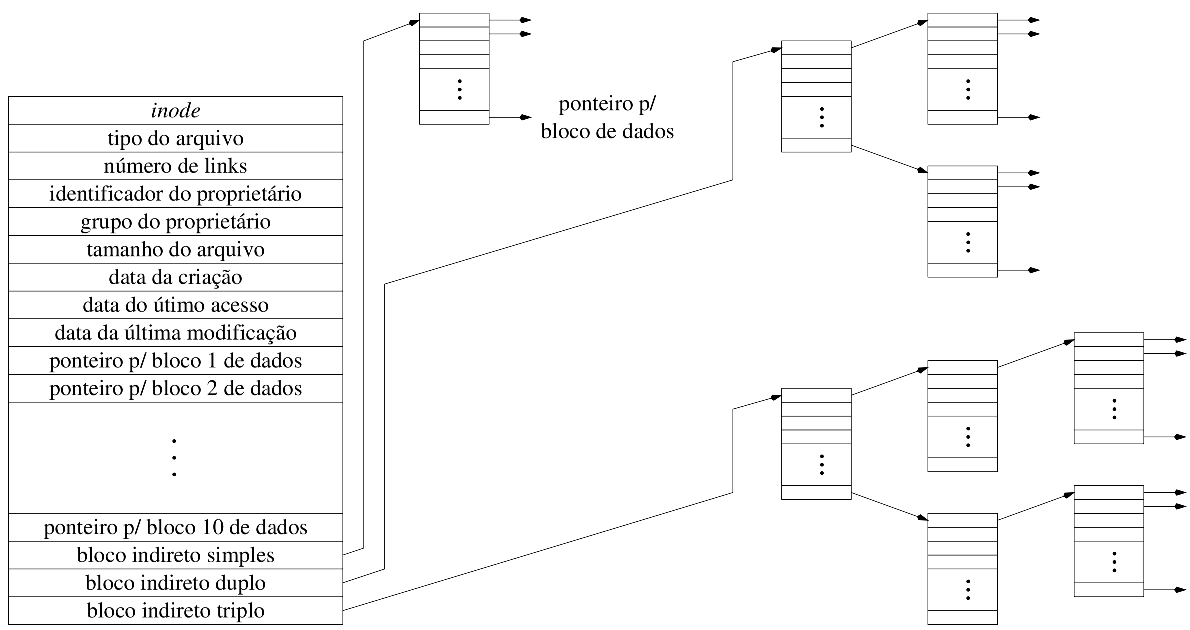
✯ o *inode* é constituído dos campos abaixo descritos e existe estaticamente no disco, assim quando o núcleo necessita manipulá-lo sua leitura é realizada.

- identificador do dono do arquivo: usuário ou grupo;
- tipo do arquivo: regular, diretório, especial ou FIFO (pipes);
- permissão de acesso: proprietário, grupo e outros;
- instante de acesso ao arquivo: última acesso e modificação do *inode*, bem como última modificação do arquivo;
- número de *links* ou conexões associadas ao arquivo;
- endereços no disco dos blocos de dados do arquivo;
- tamanho do arquivo.

- ... o *inode* para ser manipulado é antes copiado para a memória e, além disso, alguns campos são adicionados aos campos do*inode* em disco;

... 3.2.1 - Estrutura do Inode

- ... descrição dos campos de um *inode* no disco:



... 3.2.1 - Estrutura do Inode

✯ ... a cópia do *inode* na memória contém os seguintes campos:

- *status* do *inode* na memória indicando: *inode* está bloqueado; processo espera que *inode* seja liberado; ... o arquivo é um *mount point*;
- número do dispositivo lógico do sistema de arquivos que contém o arquivo;
- número do *inode* que embora não exista no disco pois na memória os *inodes* não são arranjos lineares como no disco;
- apontadores para outros *inodes* na memória, ou seja, fila *hash* e fila de *inodes* livres da mesma forma que os *buffers* são ligados na *cache* de *buffers*;
- contador de referência indicando o número de instâncias que estão ativas.

- ... a diferença mais significativa entre o *inode* na memória e o cabeçalho do *buffer* no *cache* de *buffers* é o contador de referência presente no *inode*, o qual indica o número de instâncias ativas do arquivo.

- ... um *inode* torna-se ativo quando um processo realiza a sua alocação, p. ex., abrindo-se o arquivo e é posto na lista de livres somente se o seu contador for zero.

3.2.2 - Acesso aos Inode

- o núcleo identifica um *inode* específico através do número do seu sistema de arquivo e do seu número dentro deste sistema;

- ... o algoritmo *iget* cria uma cópia do *inode* na memória física, realizando um papel equivalente ao algoritmo *getblk* utilizado no *cache* de *buffers*;

- ... caso o algoritmo não encontre o *inode* na fila *hash* associada ao número do dispositivo e ao número do *inode* procurado, um *inode* é alocado a partir da lista de *inodes* livres e é em seguida trancado ou *locked*;

- ... neste ponto, o núcleo pode ler o *inode* do disco correspondente ao arquivo e copiá-lo para a memória principal — para isso, o núcleo precisa determinar o bloco do disco que contém o *inode* bem como o *offset* do *inode* dentro do bloco;

... 3.2.2 - Acesso aos Inode

- ... para o nro. do bloco e o *offset* do *inode*, o núcleo utiliza as seguintes expressões:

❶ $nro.\ do\ bloco = \frac{\#inode}{\#inodes\ por\ bloco} + \#\ bloco\ inicial\ da\ lista\ de\ inodes$

❷ $offset = (MOD\ \frac{\#\ inode - 1}{\#\ inodes\ por\ bloco}) * \ tamanho\ do\ inode\ do\ disco$

- ... quando o núcleo libera um *inode* através de *iput*, ele decrementa seu contador de referência — se zero, o núcleo escreve o *inode* no disco caso a cópia na memória seja diferente da cópia no disco;

- ... após a cópia ser realizada e na hipótese de ser necessário posteriormente o *inode* é colocado na lista de *inodes* livres.

3.2.3 - Estrutura de um Arquivo Regular

- A indicação dos blocos do disco que constituem um determinado arquivo encontra-se no *inode* associado ao arquivo, ou seja, são utilizados 13 números de blocos:

- ... como visto anteriormente, no UNIX os 10 primeiros números compõem um arquivo de 10 KB e elevando-se este número da 11, 12 e 13 tem-se respectivamente arquiovos de 256 KBytes, 64 MBytes e 16 GBytes.

- ... um processo enxerga um arquivo como uma seqüência de bytes começando no endereço 0 e terminando com o endereço equivalente ao tamanho do arquivo – 1;

- no entanto, para o núcleo o arquivo começa no bloco 0 e continua até o número de bloco lógico correspondente ao tamanho do arquivo;

✍ um exame mais detalhado das entradas dos blocos no *inode* pode mostrar que algumas entradas possuem o valor "zero", indicando que o bloco lógico não contém dados — o que sugere que nenhum processo escreveu dados neste arquivo.

3.3 - Estrutura do Diretório

- ... a forma como o sistema de diretório encontra-se organizado é dependente da maneira que o nome de um arquivo (p. ex., fornecido pelo usuário na abertura de um arquivo) é mapeado nos seus *inodes*;
- No UNIX, essa estrutura é extremamente simples com cada entrada contendo o nome do arquivo e o número do *inode*, representados através de 16 bytes (2 bytes para o número do *inode* e 14 bytes para o nome);
- ... quando um arquivo é aberto, o sistema deve através do seu nome localizar os seus blocos no disco — se o nome for de um caminho relativo, o núcleo acessa o *inode* do diretório corrente e busca pelo arquivo.

3.4 - Super-Bloco

- ... até o momento consideramos a hipótese de que um *inode* havia sido previamente associado a um arquivo e que os blocos do disco já continham os dados;
- ... mas como o núcleo atribui *inodes* e blocos de discos ?!
- para entender este mecanismo é importante conhecer o **super-bloco**, ou seja, um segmento contínuo do disco formado pelos seguintes campos:
  - tamanho do sistema de arquivos;
  - número de blocos livres no sistema;
  - índice do próximo bloco livre na lista de blocos livres;
  - tamanho da lista de *inodes*;‘
  - número de *inodes* livres no sistema de arquivos;
  - lista de *inodes* livres no sistema de arquivos;
  - índice do próximo *inode* livre na lista de *inodes*;
  - campos *locked* para a lista de blocos livres e *inodes* livres;
  - *flag* indicando que o super-bloco foi modificado.

3.5 - Atribuição de Inode a um Novo Arquivo

- ... esta atribuição é feita através do algoritmo *ialloc*, que utilizando-se de uma lista de *inodes* livres do sistema de arquivos mantida como uma *cache* pelo super-bloco;
- ... esta lista não deve ser confundida com a lista que abriga os *inodes* livres no *pool* de *inodes* em memória que por sua vez está associada a arquivos já criados;
- **lista de *inodes* mantida pelo super-bloco** — abriga os *inodes* do disco que não estão alocados a nenhum arquivo e que poderão ser associados aos arquivos que serão criados pelo sistema.

... 3.5 - Atribuição de Inode a um Novo Arquivo

- ... a figura abaixo ilustra o algoritmo *ialloc* para um arquivo sendo criado:

```
Algoritmo IALLOC
Entrada: unidade lógica
Saída: inode locked
{
  do {
    if( super-bloco locked ) {
      if( buffer ocupado ) {
          sleep( evento "super-bloco livre" );
        continue;
      }

    if( lista de inodes no super-bloco vazia ) {
      trava super-bloco;
      acesse posição de onde a busca anterior terminou;
      pesquise inodes livres do disco, copiando-os para a lista do super-bloco;
      destrave super-bloco( evento "super-bloco livre" );
      if( nenhum inode livre encontrado no disco ) {
        retorne NULL;
      }
      marque posição onde a busca terminou;
    }

    /* existe inodes livres no super-bloco */
    acesse um número de inode da lista de inodes livres;
    acesse inode do disco ( algoritmo IGET );

    if( inode não livre ) {
      grave inode no disco;
      libere inode ( algoritmo IPUT );
      continue;
    }

    inicialize inode;            /* inode livre */
    escreva inode no disco;
    decremente o número de inodes livres ( super-bloco );
    retorne inode;
  } while( TRUE );
```

3.6 - Alocação de Blocos no Disco

- ... quando um processo necessita escrever dados em um arquivo, o núcleo aloca blocos do disco para expandí-lo, o que significa associar novos blocos ao *inode*;
- ... para agilizar a determinação de quais blocos do disco estão disponíveis para serem alocados ao arquivo, o super-bloco contém um arranjo que relaciona uma lista ligada em disco de blocos que contém números de blocos livres;
- ... o super-bloco possui parte desta lista ligada de tamanho máximo correspondendo a um bloco e é gerenciada de modo a sempre manter uma indicação de parte dos blocos livres no disco.
- ... ao requisitar a alocação de um bloco no sistema de arquivos, o algoritmo *alloc* reserva o próximo bloco disponível na lista do super-bloco.

... 3.6 - Alocação de Blocos no Disco

- ... o algoritmo *alloc* tem como parâmetro de entrada, a unidade lógica e como parâmetro de saída o *buffer* para o novo bloco.

```
Algoritmo ALLOC
Entrada: unidade lógica
Saída: buffer para o novo bloco
{
    while( super-bloco locked ) {
        sleep( evento "super-bloco livre" );
    }
    remova bloco da lista de blocos livres ( super-bloco );
    if( lista de blocos livres do super-bloco vazia ) {
        trave super-bloco;
        leie próximo bloco da lista de blocos livres no disco ( algoritmo BREAD );
        copie o conteúdo do bloco ( número de blocos livres ) na lista do super-bloco;
        libere o buffer associado ao bloco ( algoritmo BRELSE );
        destrave super-bloco ( evento "super-bloco livre" );
    }

    obtenha buffer da cache de buffers ( algoritmo GETBLK );
    zere o conteúdo do buffer;
    decremente o contador de blocos livres ( super-bloco );
    marque super-bloco como "modificado";
    retorne buffer;
}
```

```
/* Para agilizar a determinação de quais blocos do disco estão */
/* disponíveis para serem alocados a um arquivo, o super-bloco */
/* contém um arranjo que relaciona o número de blocos do disco */
/* que estão livres */

/* Este arranjo constitui-se de uma lista ligada em disco de */
/* blocos que contém números de blocos livres */

/* O super-bloco possui parte desta lista ligada, de tamanho */
/* máximo correspondendo a um bloco, sendo gerenciada */
/* quando da liberação
```

3.7 - Chamadas do Sistema de Arquivos

- ... as chamadas de sistema que compõem o sistema de arquivos utilizam, dentre outros, os algoritmos descritos anteriormente (*getblk*, *ialloc*, etc.);

- No UNIX, cada arquivo é identificado por um descritor de arquivo, ou seja, um número inteiro que é único no sistema de arquivos;

✯ ... as principais chamadas de sistema referentes ao sistema de arquivos são a seguir apresentadas (forma suscinta):

  - **open** — abre um arquivo para leitura/escrita, podendo na chamada indicar se o arquivo deve ser criado caso inexista e com que permissões;
  - **dup** — duplica um descritor de arquivo, retornando um novo descritor associado ao mesmo arquivo;
  - **pipe** — cria um *pipe* no sistema de arquivo, permitindo 02 processos estabelecerem comunicação, sendo um produtor e outro consumidor;
  - **close** — encerra a operação sobre um arquivo aberto;
  - **link** — cria uma conexão simbólica para um arquivo já existente;

... 3.7 - Chamadas do Sistema de Arquivos

✯ ... as principais chamadas de sistema referentes ao sistema de arquivos são a seguir apresentadas (forma suscinta):

  - **unlink** —- remove uma conexão simbólica;
  - **read** — lê dados de um arquivo para um *buffer* de memória;
  - **write** — escreve dados num arquivo de um *buffer* de memória;
  - **lseek** — altera a posi
  - **mknod** — cria um arquivo especial tipo bloco ou caractere, associando-o a algum periférico;
  - **mount** — adiciona uma nova unidade lógica ao sistema de arquivos;
  - **umount** — remove uma nova unidade lógica ao sistema de arquivos;
  - **chdir** — altera o diretório corrente na área U;
  - **chown** — altera o usuário que têm a posse do arquivo;
  - **chmod** — altera permissões de um arquivo;
  - **stat** — fornece informações sobre um arquivo, tipicamente as constantes no *inode* do arquivo.