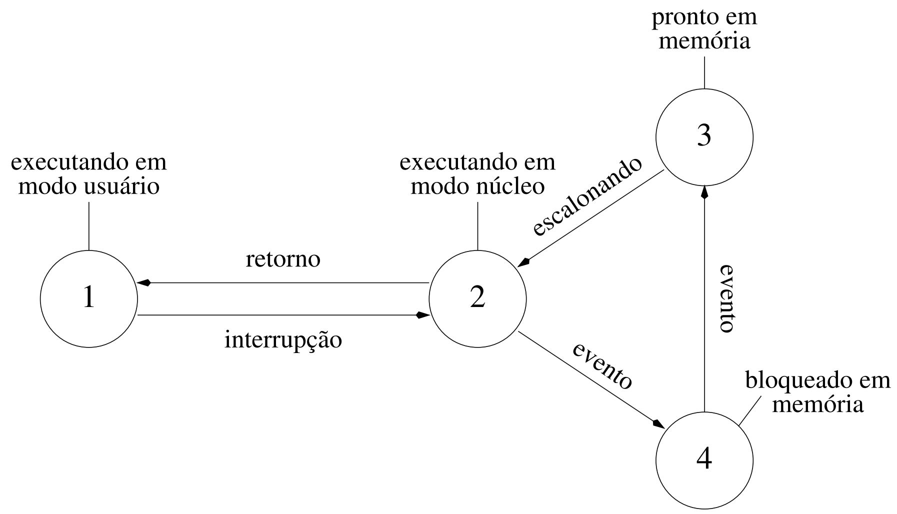
... 1.6 - Arquitetura do UNIX - Processos

- a vida de um processo pode ser representada por um conjunto de estados: executando no modo usuário, executando no modo núcleo, pronto e bloqueado (dormindo).



... 1.6 - Arquitetura do UNIX - Processos

- ... o núcleo protege a sua consistência permitindo chaveamento de contexto apenas quando o processo transita do estado "executando em modo núcleo" para o modo "bloqueado" (transição de 2 para 4).

- ... o núcleo eleva o nível de execução do processador quando da execução de regiões críticas de modo a impedir interrupções que possam provocar inconsistências em suas estruturas de dados (ver níveis de interrupção).

- ... o escalonador de processos realiza, periodicamente, a preempção de processos executando no modo usuário de forma que os processos (neste modo) não monopolizem o processador.

... 1.6 - Arquitetura do UNIX - Processos

- campos importantes da *tabela de processos* - campo de estado, identificadores dos usuários que possuem o processo e conjunto descritor de evento quando o processo está bloqueado.

- ... a área U contém informações que descrevem o processo e que são acessadas somente durante a execução do processo, p. ex., descritores de arquivos, parâmetros internos de entrada/saída, limites do tamanho do processo e arquivo, etc.

- quando um processo evoca a chamada **exec**, o núcleo aloca regiões para o texto, dados e pilha para o processo que está sendo criado, após liberar as posições antigas do processo que estava executando;

- quando o processo evoca **fork**, o núcleo duplica o espaço de endereço do processo antigo permitindo, quando possível, que processos compatilhem regiões ou, caso contrário, faz uma cópia da região;

- quando um processo **exit**, o núcleo libera as regiões que o processo estava usando.

... 1.6 - Arquitetura do UNIX - Processos

- **contexto de um processo** - estado definido pelo:
  - texto no que corresponde ao valor de suas variáveis;
  - valores dos registradores de máquina utilizados;
  - valores armazenados no seu *slot* na *tabela de processos* e na área U;
  - conteúdo da pilha de usuário e da pilha de núcleo.

- ... quando da realização de uma **mudança de contexto**, o núcleo salva informações suficientes de forma que, posteriormente, possa recuperar o contexto do processo e continuar sua execução a partir do ponto que interrompeu;

- ... similarmente, tem-se a **mudança de modo**, ou seja, o núcleo salva as informações necessárias para que o processo possa retornar ao modo usuário.

... 1.6 - Arquitetura do UNIX - Processos

- ... cada chamada de sistema possui um entrada na biblioteca de chamadas de sistema, a qual é codificada em *assembler*, contendo instruções especiais (*traps*) que, quando executadas, provocam uma interrupção resultando em um chaveamento de *hardware* para o modo núcleo passando a utilizar a pilha do núcleo.

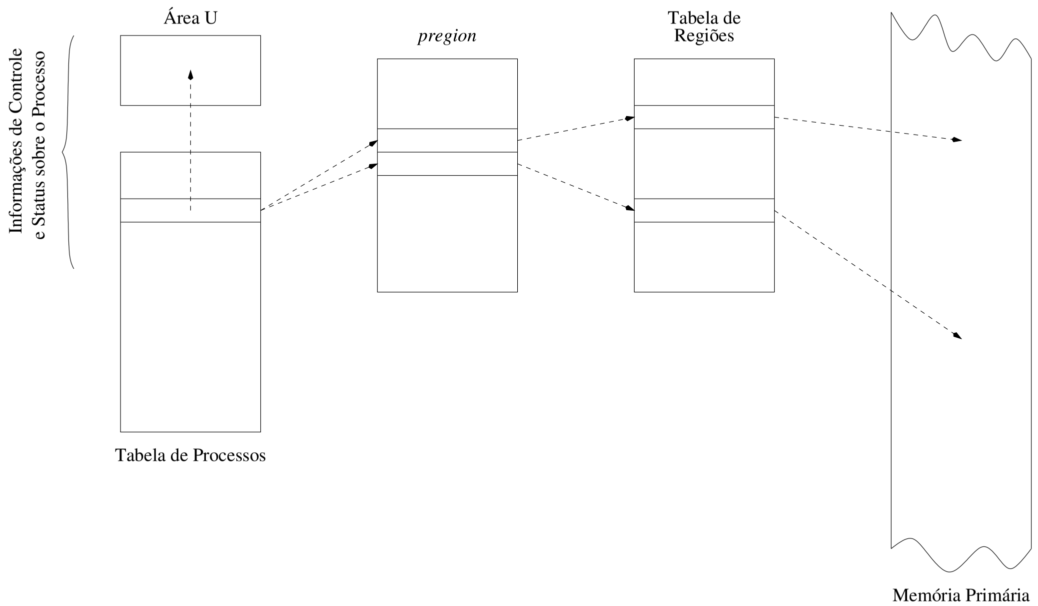
- ... todo processo possui uma entrada na tabela de processos do núcleo e a cada um é alocada uma área que contém dados privados manipulados somente pelo núcleo;

- ... a *tabela de processos* (TP) aponta para uma *tabela de regiões de processo* (*pregion*), cujas entradas apontam para entradas na tabela de região;

- ... região é uma área contígua de um espaço de endereço do processo, tal como: texto ou código, dados e pilha;

... 1.6 - Arquitetura do UNIX - Processos

- entradas na *tabela de região* descrevem atributos da região, ou seja, região contém texto ou dados, é ou não compartilhada, conteúdo está ou não na memória, etc.



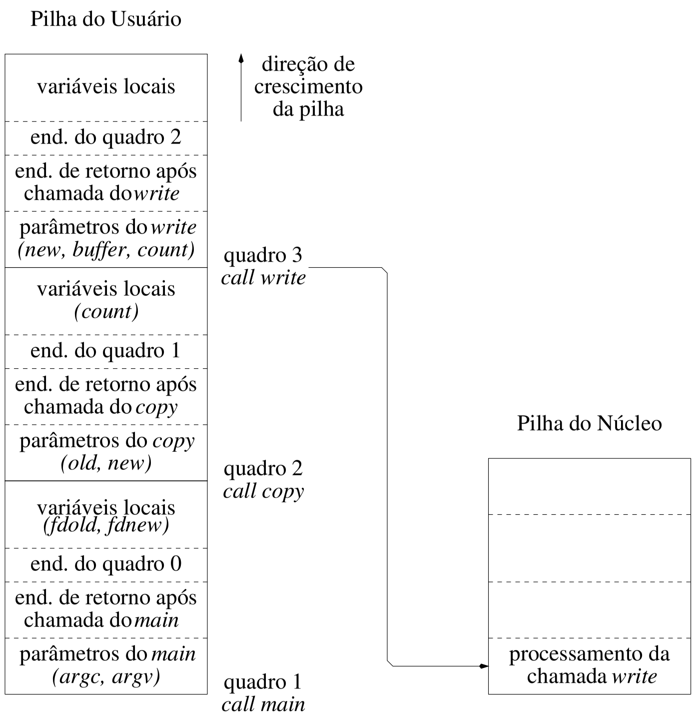
... 1.6 - Arquitetura do UNIX - Processos

- um arquivo executável gerado a partir da compilação de um programa, consiste:

❶ informações que descrevem os atributos do arquivo;
❷ código (texto) do programa;
❸ representação em linguagem de máquina dos dados que possuem valor inicial e uma indicação de quanto espaço o núcleo necessita alocar para os dados sem valor inicial, denominado bss;
❹ outras seções, tais como informações sobre a tabela de símbolos.

- ... o núcleo carrega um arquivo executável durante a execução de uma chamada **exec**, consistindo o processo carregado em 03 partes: texto, dados e pilha.
  – texto de dados correspondem às seções de texto e dados iniciados e não-iniciados;
  – a pilha é criada automaticamente e o seu tamanho é ajustado dinamicamente pelo núcleo e isto é feito em tempo de execução;
  – um quadro na pilha ($frame$) contém parâmetros para a função chamada, suas variáveis locais e os dados necessários para recuperar o quadro anterior na pilha.

... 1.6 - Arquitetura do UNIX - Processos



... 1.6 - Arquitetura do UNIX - Sistema de Arquivos

- *bloco de boot* - contém o código do *bootstrap* que é lido na máquina quando da partida do sistema operacional;

- *super-bloco* - descreve o estado de um sistema de arquivo;

- *lista de inodes* - tem seu tamanho definido quando da configuração do sistema de arquivo; um *inode* corresponde à raiz do sistema de arquivo e é através dele que a estrutura do sistema de arquivo é acessada;

- *bloco de dados* - contém arquivos e dados administrativos e, em sendo assim, só pode pertencer a um único arquivo do sistema.

bloco de *boot*	super bloco	lista de *inodes*	blocos de dados

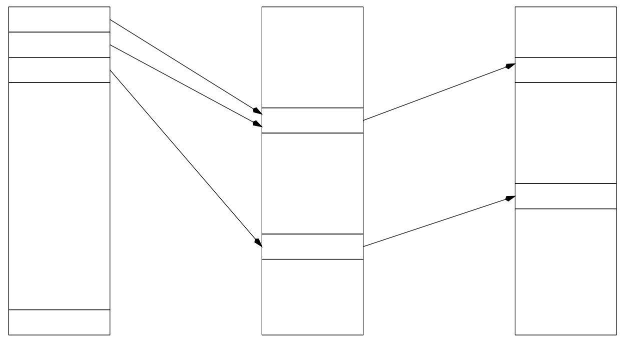
... 1.6 - Arquitetura do UNIX - Processos

- um processo corresponde à execução de um programa e consiste de um conjunto de bytes que o processador interpreta como instrução de máquina, dados e pilha.
- ... o processo executa uma sequência de instruções que é auto-contida e que nã salta para um outro processo — leitura e escrita somente em sua área de dados e pilha, mas nunca na área de dados e pilha de um outro processo;
- ... processos comunicam entre si e com o sistema através de chamadas de sistema;
- ... do ponto de vista prático, um Processo UNIX é uma entidade criada pela chamada **fork**, exceto o primeiro processo do sistema — processo especial criado quando da inicialização do sistema, ou seja, do *boot* e conhecido por processo *init*;
- ... o processo *init* é ancestral de qualquer outro processo no sistema;
- ... o núcleo identifica cada processo através do seu *process identifier* ou *pid* — por exemplo, a chamada de sistema *fork* retorna o valor 0 para o processo filho e o *pid* do processo filho para o processo pai.

... 1.6 - Arquitetura do UNIX - Sistema de Arquivos

- *tabela de arquivo* - mantém o *offset*, no arquivo correspondente, do próximo byte a ser localizado, assim como, os direitos de acesso ao processo;

- *tabela descritora de arquivo do usuário* - identifica todos os arquivos abertos para um dado processo, por exemplo, de um dado usuário.



... 1.6 - Arquitetura do UNIX - Sistema de Arquivos

- ... o núcleo retorna um descritor de arquivo quando das chamadas **open** e **create**, o qual corresponde a um índice na TDAU.

- ... na execução de um **write** ou um **read**, o núcleo utiliza o descritor de arquivo para acessar a TDAU e através desta alcançar a TA e o *inode* do arquivo.

- No UNIX, o núcleo não enxerga o sistema de arquivo como discos, mas como dispositivos lógicos — assim, a conversão entre endereços de um dispotivo lógico e os endereços de um dispositivo físico é realizada pelo *driver* do dispositivo.

- ... um sistema de arquivo consiste de uma seqüência de blocos lógicos, cada um contendo qualquer múltiplo de 512 bytes — o tamanho do bloco não muda em um mesmo sistema de arquivo, mas pode mudar em sist. de arquivo diferentes.

- ... blocos maiores representam um aumento na taxa de transferência entre a memória e o disco mas, por outro lado, demandam mais espaço em memória para manipulá-los.

... 1.6 - Arquitetura do Sist. Oper. UNIX

- ... os programas em linguagem de máquina podem evocar chamadas ao sistema diretamente, ou seja, sem fazer uso da biblioteca;
- por outro lado, programas ordinários (ling. alto nível) evocam chamadas como se tivessem executando funções ordinárias — cabe a biblioteca mapear estas chamadas de funções nas primitivas necessárias para acessar o sistema operacional;
- ... 02 grupos de chamadas de sistema podem ser identificados: chamadas do subsistema de arquivos e chamadas do subsistema de controle de processos;
- os subsistemas de arquivo e de controle de processos interagem quando do carregamento de um arquivo na memória para execução.
- cabe ao subsistema de controle de processos prover mecanismos de gerenciamento de memória, escalonamento de processos e comunicação entre processos.

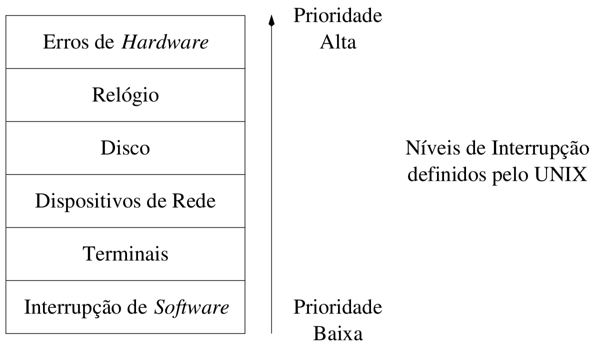
... 1.6 - Arquitetura do UNIX - Sistema de Arquivos

- *inode* - utilizado como representação interna de um arquivo, o *inode* contém uma descrição do *layout* no disco do arquivo de dados, assim como outras informações tais como: proprietário, permissões e instante de acesso.
- ... todo arquivo possui um *inode* (alocado quando da sua criação), entretanto, vários nomes (*links*) todos associados ao mesmo *inode* podem ser atribuídos.
- ... os *inodes* são armazenados no sistema de arquivos e são lidos em uma *tabela de inodes* em memória, quando da manipulação dos respectivos arquivos.
- ... 02 outras estruturas são importantes:
  - *tabela de arquivo* - estrutura global ao núcleo;
  - *tabela descritora de arquivo de usuário* - criada por cada processo.
- ... quando da criação/abertura de um arquivo, o núcleo associa uma entrada de cada uma das tabelas ao *inode* correspondente ao arquivo.

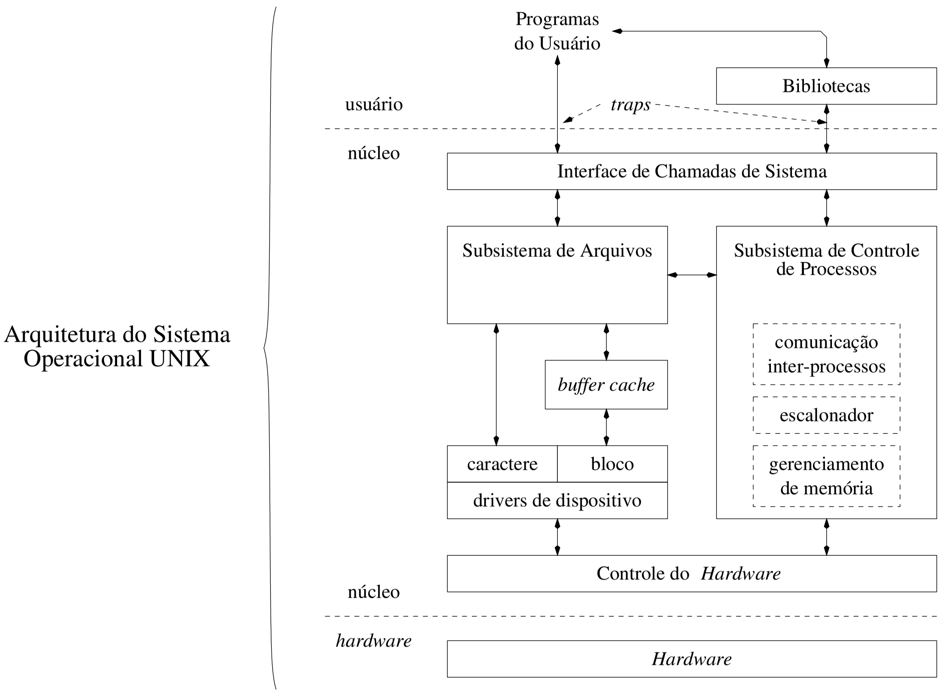
... 1.5 - Aspectos de Hardware

- **gerenciamento de memória** - o núcleo reside permanentemente na memória principal, enquanto os processos ordinários ou parte deles permanecem somente o tempo necessário para a sua execução;

- ... quando da execução de um programa, o núcleo aloca espaço na memória principal através do mapeamento do endereço virtual no endereço físico da máquina (memória principal) que por sua vez depende das características do *hardware* da máquina.



1.6 - Arquitetura do Sist. Oper. UNIX



1.4 - Serviços do Sist. Operacional UNIX

- controle de execução de processos: criação, finalização, suspensão, comunicação e sincronização entre processos;
- escalonamento (ordem de acesso ao processador) de processos;
- alocação de memória principal para execução de processos — pode se fazer necessário que o núcleo mova os processos da memória principal para a secundária;
- alocação de memória secundária para armazenamento e/ou recuperação eficiente dos dados do usuário (este serviço constitui o sist. de arquivo);
- acesso controlado aos dispositivos periféricos tais como terminais, fitas, discos, etc.;
- serviços oferecidos pelo núcleo de forma transparente:
  - ✯ o núcleo detecta que um dado arquivo é um arquivo regular ou um dispositivo, mas esconde esta distinção dos processos do usuário;
  - ✯ o núcleo formata os dados em um arquivo para fins de armazenamento interno, entretanto, este formato é escondido do usuário que recebe um *stream* de bytes.

1.5 - Aspectos de Hardware

- processos no UNIX pode ser executado em 02 níveis: modo usuário e modo núcleo:
  - **modo usuário** - podem acessar as suas instruções e dados, mas não as instruções e dados do núcleo ou de qualquer outro processo;
  - **modo núcleo** - podem acessar endereços do núcleo ou do usuário;
  - ... algumas instruções são privilegiadas e portanto devem ser executadas em modo núcleo resultando em erro quando executadas no modo usuário.
- **interrupções e exceções** - no UNIX, o tratamento dado às interrupções é o mesmo dado às exceções (evento não esperado provocado pelo processo);
- **nível de execução do processador** - a atribuição do nível de execução do processador para um determinado valor mascara a interrupção daquele nível e dos níveis inferiores, ou seja, as dos níveis superiores tornam-se habilitadas;

1.3 - Modularidade no Ambiente UNIX

- **filosofia do sistema:** permitir aos usuários o desenvolvimento de programas pequenos e modulares que possam ser usados como blocos primitivos na construção de programas mais complexos — alguns exemplos:

❶ **redirecionamento de entrada/saída:** processos possuem, convencionalmente, acesso a 03 tipos de arquivos padrões: entrada, saída e erro — que podem ser redirecionados de forma independente.

- **ls** - lista todos os arquivos do diretório corrente na saída padrão;
- **ls > output** - redireciona a saída padrão (p. ex., terminal onde se executa o comando) para o arquivo "output" no diretório corrente;
- **mail mjb < carta** - faz com que o programa *mail* leia o conteúdo da mensagem do arquivo carta e não do terminal (entrada padrão).

... 1.3 - Modularidade no Ambiente UNIX

❷ **pipe:** estabelece um fluxo de dados entre um processo produtor e consumidor — processos podem redirecionar sua saída para um "pipe", por exemplo:

- **grep main first.c second.c third.c** — lista as ocorrências de "main" nos arquivos "first.c", "second.c" e "third.c";
- **grep main first.c second.c third.c | wc -l** — submete a saída do comando anterior a um utilitário que conta o número de bytes, palavras e linhas de um arquivo (wc, opção -l);
- **ps auxw | grep root** — recupera o estado dos processo correntemente sendo executados e seleciona aqueles nos quais a *string* "root" for encontrada.

... 1.2 - Ambiente de Processamento

- ... o UNIX, ao contrário de outros sistemas, não suporta no núcleo programas no nível do usuário; p. ex., programa *shell* responsável pela interpretação de comandos;

- na maior parte das vezes o *shell* executa o comando **fork** e o processo filho executa o comando solicitado através da chamada **exec**;

- o *shell* aceita 03 tipos de comandos:

  ❶ comando interno do *shell*;
  ❷ arquivo executável produzido por compilação;
  ❸ arquivo executável contendo uma seqüência de linhas de comando no *shell*.

... 1.2 - Ambiente de Processamento

- o *shell* aceita 03 tipos de comandos:

  ❶ comando interno do *shell*;
  ❷ arquivo executável produzido por compilação;
  ❸ arquivo executável contendo uma seqüência de linhas de comando no *shell*.

- ... o *shell* normalmente executa um comando sincronamente, ou seja, espera o término do processo associado ao comando antes da leitura de um novo comando;

- no entanto, é possível a execução assíncrona; neste caso, o *shell* lê e executa a próxima linha de comando sem esperar o término do processo associado ao comando anterior, o qual executa em *background*;

- como o *shell* é um programa que situa no nível do usuário, ou seja, não faz parte do núcleo, é fácil modificá-lo para um ambiente particular.

1.2 - Ambiente de Processamento

- ... um programa é um arquivo executável e um processo é uma instância do programa em execução — no UNIX vários processos podem executar simultaneamente;
- criação e sincronização de processos: comandos **fork**, **execl**, **wait** e **exit**.

```
/* Include Files */
#include <stdio.h>
#include <sys/wait.h>
#include <sys/time.h>

int main( int argc, char* argv[] )   /* Main Program */
{
  int pid;
  struct timeval tv1, tv2;
  double t1, t2;

  pid = fork();   /* Cria um Processo */
  if( pid == 0 ) execl( argv[1], NULL );  /* Processo Filho */
  .
  .
```


... 1.2 - Ambiente de Processamento

```
  .
  .
  .
  pid = fork();   /* Cria um Processo */
  if( pid == 0 ) execl( argv[1], NULL );  /* Processo Filho */

  /* Processo Pai continua ... */
  gettimeofday( &tv1, NULL );
  t1 = (double) ( tv1.tv_sec ) + (double) ( tv1.tv_usec ) / 1000000.00;

  wait( NULL );   /* Sincroniza com o término do Proc. Filho */
  gettimeofday( &tv2, NULL );
  t1 = (double)( tv2.tv_sec ) + (double)( tv2.tv_usec ) / 1000000.00;

  printf(``\nO Tempo de Execução de %s é: %lf.\n'', argv[1], (t2 - t1));
}
```

... 1.1 - Sistema de Arquivo

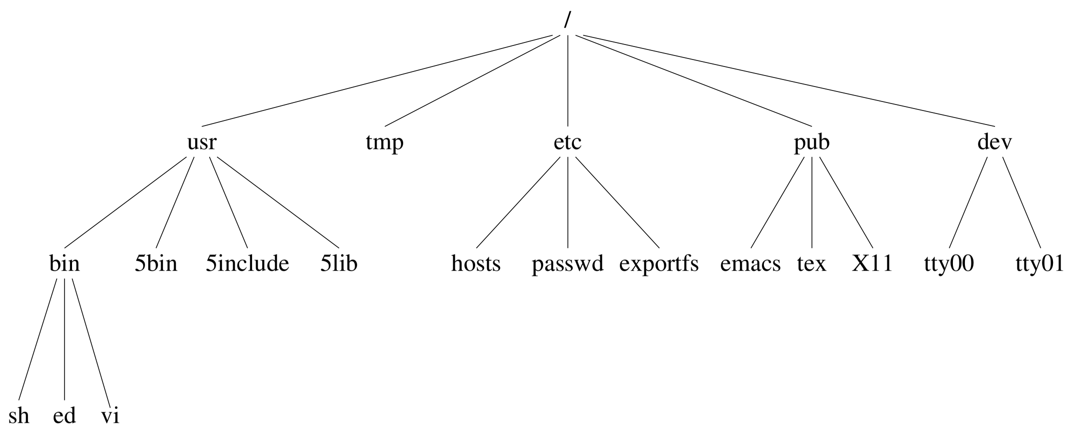
```
#include <stdio.h>
#include <sys/fcntl.h>
#include <sys/stat.h>

char buffer[512];

void copy( int old, int new )     /* Função Copy( ... ) */
{
  int count;
  while( (count = read( old, buffer, sizeof(buffer))) > 0 )
    write( new, buffer, count );
}

main( int argc, char* argv[] )    /* Programa Principal */
{
  int fdold, fdnew;

  if( argc != 3 ) {
    printf(``Uso: copy f1 f2.\n'');  exit(1);
  }
  .
  .
```


... 1.1 - Sistema de Arquivo

```
  .
  .

  /* Abre Arquivo p/ Leitura */
  fdold = open( argv[1], O_RDONLY );
  if( fdold == -1 )  {
    printf(``Impossível abrir %s.\n'', argv[1] );  exit(1);
  }

  /* Cria Arquivo Novo */
  fdnew = creat( argv[2], 0666 );
  if( fdnew == -1 )  {
    printf{``Impossível criar %s.\n'', argv[2] );  exit(1);
  }

  /* Chama Copy(...) */
  copy( fdold, fdnew );
  exit(0);
}
```

1.1 - Sistema de Arquivo

- ... o sistema de arquivo do UNIX é caracterizado por:
  - estrutura hierárquica
  - tratamento consistente dos dados de um arquivo;
  - facilidade na criação/ remoção de arquivos;
  - crescimento dinâmico de arquivos;
  - proteção aos dados de um arquivo;
  - tratamento dos dispositivos periféricos como arquivo regular de dados.



... 1.1 - Sistema de Arquivo

- ... a localização de um arquivo pode ser na forma absoluta ou relativa;
- ... programas no ambiente UNIX não estão a par de como o núcleo armazena dos dados de um arquivo, embora recebam do núcleo uma cadeia de bytes e tenham que interpretar o seu conteúdo;
- ... o acesso aos arquivos é controlado pelas permissões de acesso associadas a cada arquivo — 03 tipos de permissões para 03 tipos de usuários;
- ... programas no UNIX acessam periféricos da mesma forma (com a mesma sintaxe) que utilizam para o acesso aos arquivos regulares;
- ... dispositivos periféricos são protegidos da mesma forma que os arquivos regulares, ou seja, permissões de leitura, escrita e execução para usuários: proprietário do arquivo, grupo de usuários e qualquer outro usuário.

Cap. 1 - Visão Geral do Sist. Oper. UNIX

1 - Visão Geral do UNIX

- ... programas interagem com o núcleo atraves da evocação de um conjunto de chamadas de sistema (p. ex., muitos desses programas são denominados comandos);