Sistemas Operacionais

Processos no Unix (Linux)

Objetivos

- Apresentar a estrutura de processos no Unix
- Descrever as principais funções associadas à gerência de processos
- Discutir questões sobre estruturas de dados relacionadas e comunicação entre processos

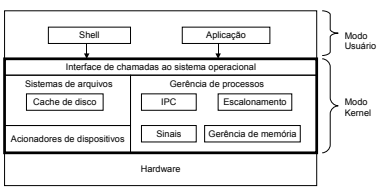
Unix: um pouco de história

- Bell Labs, Ken Thompson, Dennis Ritchie
- 1971: PDP-7, assembly
- 1972: PDP-11, C --- código fonte disponível
- 197?: U Berkeley: Mem. Virtual, TCP/IP
- 1980: AT&T - System V
- POSIX: Padrão de portabilidade

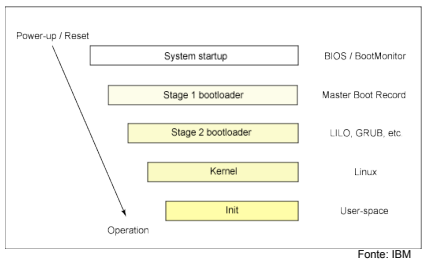
Unix: versões atuais

- Sun Solaris: um dos mais estáveis
- SCO Unix: um dos mais antigos
- Linux: um dos mais populares
- freeBSD: um dos mais eficientes
- netBSD: o com mais plataformas
- IBM AIX: um dos mais controversos
- Apple Mac X: um dos mais diferentes

Unix: estrutura geral do sistema

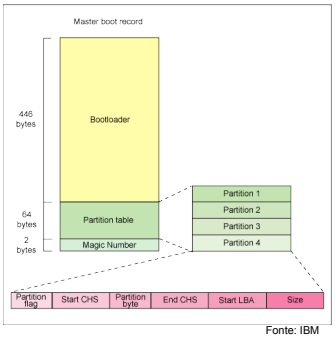


Unix: o processo de boot



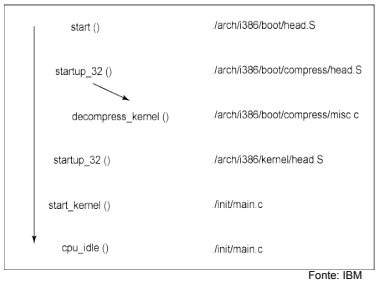
Fonte: IBM

Unix: o processo de boot (*bootloader*)



Fonte: IBM

Unix: o processo de *boot* no *kernel*



Fonte: IBM

Unix: o processo de *boot* (init)

- Primeiro processo a executar
- Nunca termina (causa *shutdown* se morrer)
- Lê arquivos de configuração
- Dispara demais processos

Unix: processos

- Hierarquia de processos: pais e filhos
  - **`fork()`**: cópia do processo pai
  - **`exec()`**: substitui processo por novo programa
- Várias formas de comunicação entre processos: mem., semáforos, pipes, msgs.
- Processos, *threads* de aplicação (pthreads)
- Alguns tem *threads* de núcleo (Solaris, Linux)

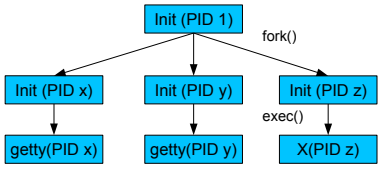
Criação de processos no Unix

- fork()
  - S.O. cria um novo PCB
  - Cria-se uma cópia da memória do processo pai
  - Recursos de E/S são compartilhados
- exec()
  - S.O. busca um programa da memória e o carrega sobre a área do programa que fez a chamada
  - Execução passa para o início do programa principal carregado.

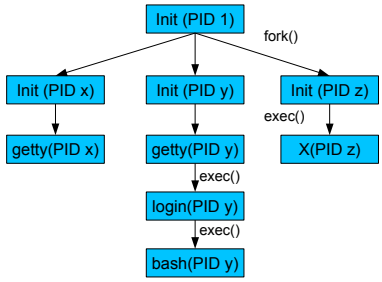
Criação de processos no Unix

```
if ((child_pid=fork()) > 0) {
        /* Aqui é o processo "pai" */
} else if (child_pid==0) {
        /* Processo "filho" */
        if(execl(programfile,/*...*/)<0) {
                perror("Erro execl"); exit(1);
        }
        fprintf(stderr,"Não chega aqui");
} else {
        perror("Erro fork"); exit(2);
}
```

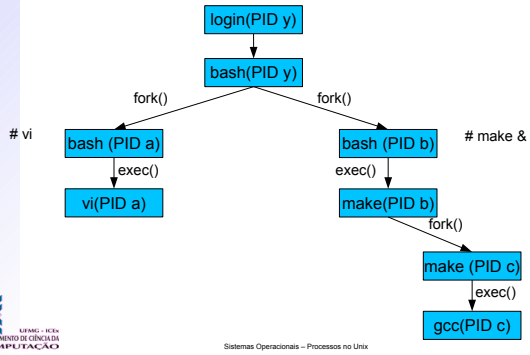
Unix: processos (a partir do init)



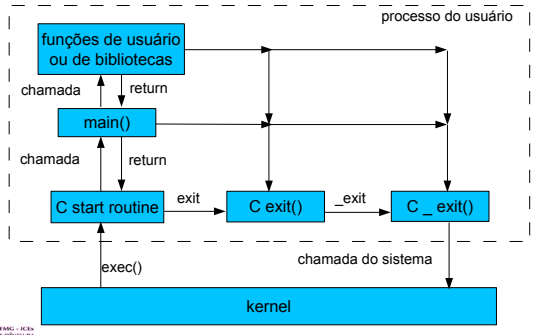
Unix: processos (a partir do init)



Unix: processos (a partir do login)



Unix: processos (início e fim)



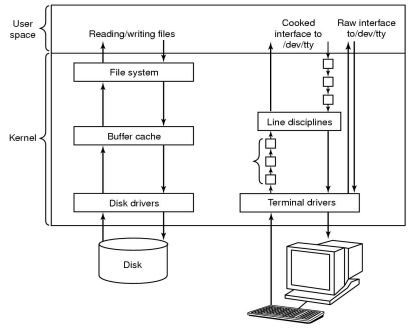
Unix: escalonamento de processos

- Política complexa com prioridades controladas (`nice`)
  - Usuários podem reduzir prioridade de processos
  - Apenas root pode aumentá-las
- Escalonador classifica processos:
  - *I/O bound* - maior parte do tempo em filas
  - *CPU bound* - usam a CPU intensivamente

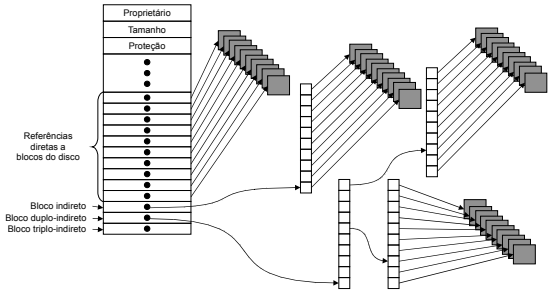
Unix: E/S

- Arquivos: sequências de bytes ($2^{32}-1$)
- Sistemas de arquivos na árvore de diretórios
- Arquivos mapeados através de *i-nodes*
- Permite referências múltiplas (*links*)
- Diretórios como listas encadeadas simples
- Dispositivos vistos como arquivos
  - Afinal, “tudo é um arquivo”
- Dispositivos de bloco, caractere e... ???

Unix: E/S (bloco x caractere)



Unix: i-node



Unix: proteção

- Permissões para o dono, grupo e “outros”
- Arquivos: ler, escrever, executar
- Diretórios: listar, criar/remover arquivos, percorrer
- Outros bits especiais (setuid, swap, etc.)
- Processos executam com permissões do usuário exceto em casos especiais

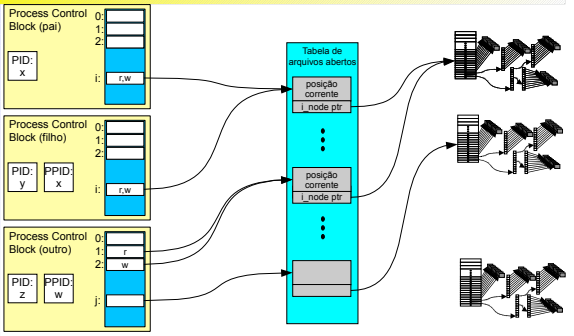
Processos e arquivos

- I-nodes identificam arquivos no sistema de arquivos
- Kernel mantém controle dos arquivos abertos (segurança)
- Cada processo recebe apenas “descritores” de arquivos
  - Tabela em memória no PCB

Tabela de descritores

- Cada entrada mantém dados de um arquivo aberto
  - registra o tipo de acesso solicitado na abertura
  - aponta para a tabela de arquivos abertos pelo kernel (independente de processos)
- Entradas 0, 1, e 2 da tabela têm significado especial:
  - STDIN, STDOUT, STDERR
- Ao executar um fork() a tabela de descritores é duplicada
  - Compartilhamento de arquivos

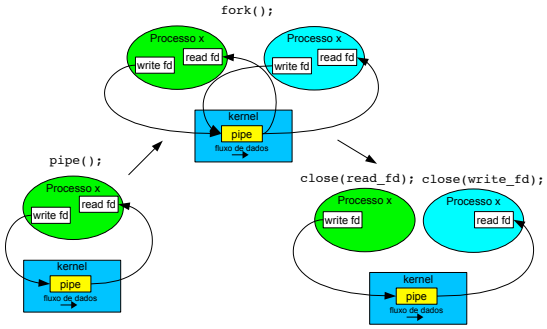
Compartilhamento de arquivos



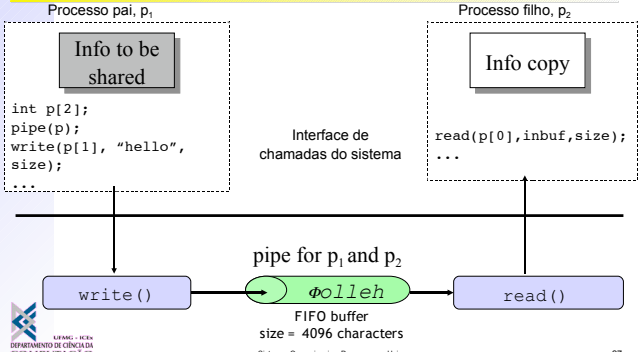
Unix: *pipes*

- Fila de caracteres (FIFO) para comunicação
  - Controle feito pelo kernel
  - Processos podem ler e escrever (operações atômicas)
- Aparecem como descritores de arquivos
  - Criação de pipe `pipe()`: análoga à abertura de arquivo
    - kernel cria a estrutura da fila na sua área
    - retorna um descritor para cada ponta do pipe (0,1)=(in,out)
  - Chamadas de sistema `read`/`write` recebem/enviam caracteres pelo pipe
    - dados são copiados entre área do processo e do kernel

Unix: criação de *pipes*



Unix: *pipes*



Unix: *pipes*

- Descritores de arquivos (e pipes) são copiados no fork

```
int pipeID[2];
. . .
pipe(pipeID); /* preenche pipeID c/ descritores */
. . .
if(fork() == 0) {  /* the child */
  . . .
  read(pipeID[0], childBuf, len);
  /* process the message */ ;
  . . .
} else {            /* the parent */
  . . .
  write(pipeID[1], msgToChild, len);
  . . .
```


Unix: descritores e *pipes*

- Chamada útil: `dup2(int fd_orig, int fd_dest);`
  - duplica o descritor original para a entrada destino
  - permite, por exemplo, ligar a entrada padrão a um pipe
  - p.ex., para executar os comandos "`ls | sort -r`"

```
pipe(pipeID); /* preenche pipeID c/ descritores */
if(fork() == 0) {  /* the child */
  dup2(pipeID[0], STDIN_FILENO);
  execl("/bin/sort", "sort", "-r", NULL);
} else {            /* the parent */
  dup2(pipeID[1], STDOUT_FILENO);
  execl("/bin/ls","ls", NULL);
}
```