Sistemas Operacionais I

Conceito de Processo em Sistemas Operacionais

Prof. Carlos Eduardo de B. Paes
Departamento de Ciência da Computação
PUC-SP

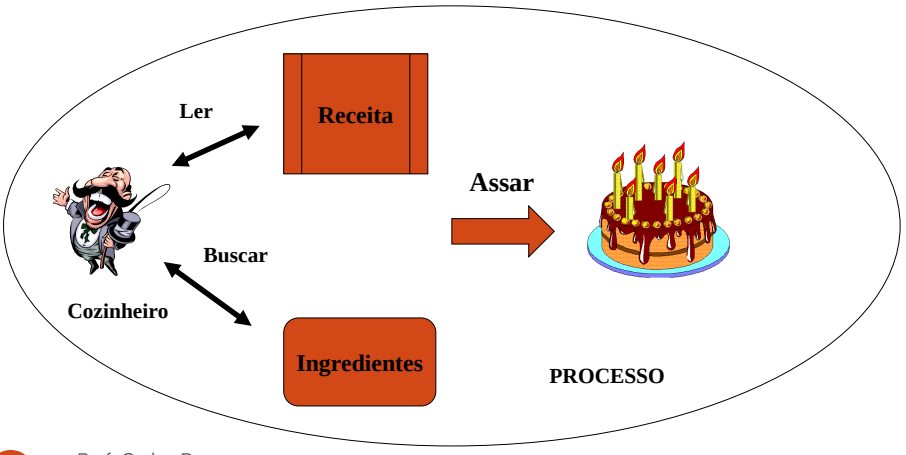
Conceitos de Processo

- Atividades da CPU: jobs, tarefas(task) e processos
- Um processo é um programa em execução, é uma entidade ativa. Um processo inclui o seguinte:
  - o código do programa (na memória, chamado de segmento de código)
  - a atividade corrente, representada pelo valor do PC (que especifica a próxima instrução a executar) e pelo conteúdo dos registradores do processador
  - um conjunto de recursos associados
  - A pilha do processo (na memória principal), contendo parâmetros de sub-rotinas, endereços de retorno e variáveis locais
  - um segmento de dados (na memória principal) contendo variáveis globais.

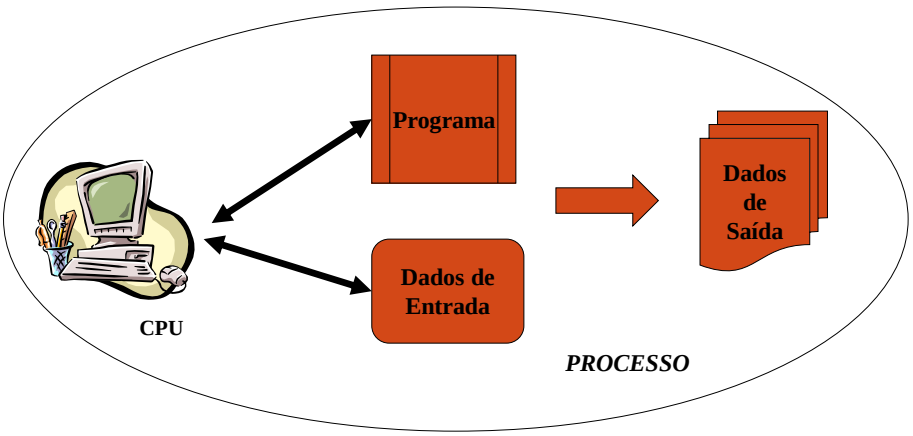
Conceitos de Processo

- Um programa não é um processo, é uma entidade passiva, uma seqüência de instruções armazenadas em um arquivo em disco
- Embora 2 processos possam estar associados ao mesmo programa (reentrância), eles são 2 seqüências de execuções paralelas
- Um processo pode criar outros processos enquanto executa

Conceitos de Processo
Programas X Processos



Conceitos de Processo
Programas X Processos



Multiprogramação

- SO "suporta" a multiprogramação se ele permite que vários processos sejam executados concorrentemente (concorrendo pela CPU) . SO é monoprogramado quando ele não permite isto
- Objetivo da multiprogramação: ter vários processos executando ao mesmo tempo, para maximizar a utilização da CPU e dos demais recursos do hardware

Multiprogramação

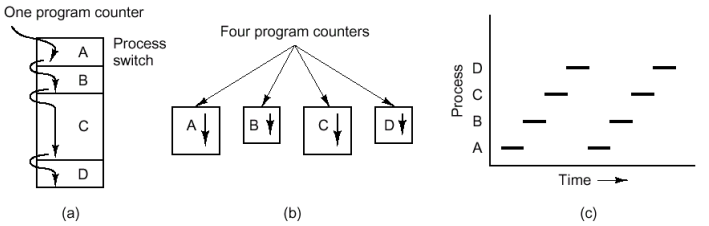
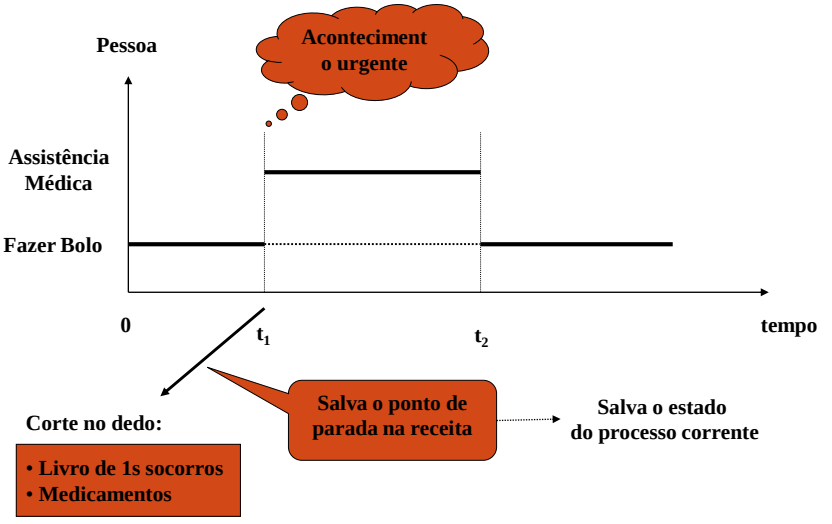


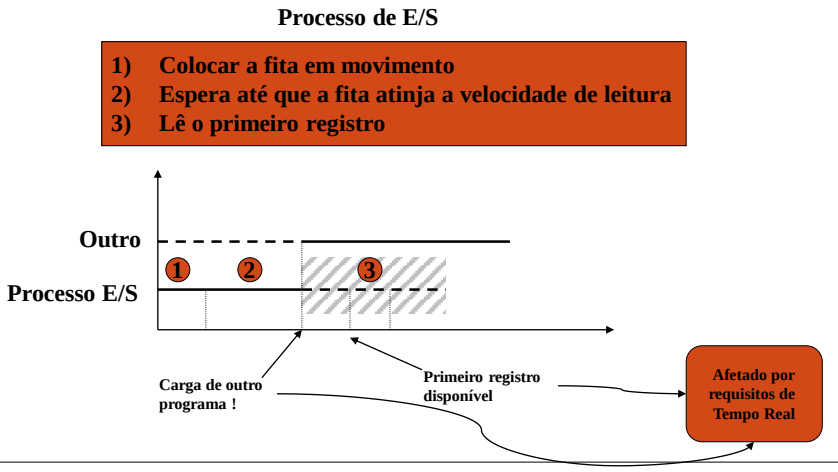
Figure 2-1. (a) Multiprogramming of four programs. (b) Conceptual model of four independent, sequential processes. (c) Only one program is active at any instant.

Multiprogramação



Multiprogramação

- Problema: Leitura de um registro na fita



9

Estados de um Processo

- À medida que um processo executa, ele muda de estado. Estado de um processo é definido pela sua atividade corrente. Cada processo pode estar em um dos seguintes estados:
  - Novo: o processo está sendo criado
  - Executando: as instruções do processo estão sendo executadas
  - Suspenso ou Bloqueado: o processo está esperando que algum evento ocorra (tal como o término de um operação de E/S ou o recebimento de um sinal)
  - Pronto: o processo está esperando para ser atribuído a um processador
  - Terminado: o processo terminou sua execução

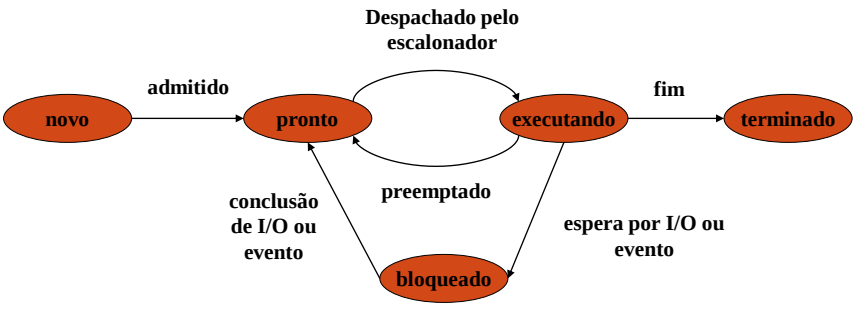
10

Estados de um Processo

- Somente um processo pode estar executando em um processador, em um determinado instante.
- Muitos processos podem estar prontos e bloqueados

Estados de um Processo

- Possíveis transições de estado de um processo



Bloco de Controle de um Processo

- SO mantém, para cada processo ativo, uma estrutura de dados, chamada bloco de controle do processo (PCB-*Process Control Block*)
- O PCB contém vária informações sobre o processo.
- Principais informações mantidas no PCB:

Process Control Block Informações

- Estado do processo: novo, executando, bloqueado ou terminado
- Contador de programa (cópia do PC): Indica o endereço da próxima instrução a ser executada
- Registradores da CPU (cópia): Os registradores variam em número e tipo, dependendo da arquiterura (acumuladores, registradores de índice, stack pointers, registradores de propósito geral, e flags). – **Troca de Contexto**

Process Control Block
Informações

- Informações para escalonamento da CPU: Prioridade do processo, ponteiros para filas de escalonamento e outros parâmetros de escalonamento.
- Informações para gerência de memória: Cópia dos registradores base e limite, tabelas de páginas ou tabelas de segmento, dependendo do esquema de gerência de memória utilizado pelo SO

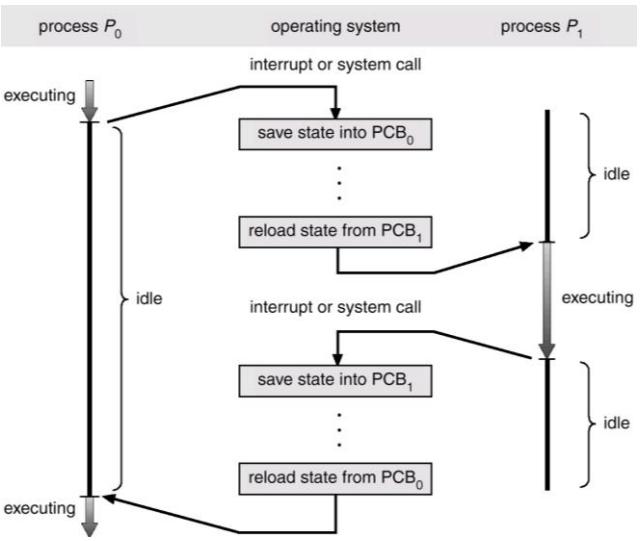
Process Control Block
Informações

- Informações de controle: Quantidade de tempo de CPU e de tempo total utilizado, limites de tempo, identificação do processo, número de controle (ex: PID), etc…
- Informações para gerência de E/S: Lista de dispositivos de E/S alocados para o processo, lista de arquivos abertos pelo processo, etc…

Process Control Block Informações

- Exemplo do PCB de um processo no sistema Linux
- Exemplo do PCB de um processo no sistema Sun Solaris (Unix)

Troca de Contexto



Troca de Contexto

- Importante: qualquer tarefa realizada pelo SO introduz *overhead* (custo adicional) para realização desta tarefa.
  - SO deve ser eficiente para minimizar *overhead*.
  - Tempo gasto para troca de contexto é *overhead* do SO para oferecer a multiprogramação.
  - Durante troca de contexto a CPU não está dedicada a nenhum processo dos usuários.

Filas de Escalonamento e Escalonadores

- Para um sistema com apenas um processador, nunca haverá mais de um processo executando.
- Se houver mais processos, os processos restantes terão que esperar até que a CPU seja alocada a cada um deles
- Da mesma forma, os processos esperam para serem carregados na memória e para utilizarem dispositivos de E/S

Filas de Escalonamento e Escalonadores

- Processos esperam ser atendidos em filas de escalonamento
- Quando um processo é submetido e ainda não foi carregado na memória principal, ele é colocado em uma fila de jobs, e seu estado é novo.
- Nem todos os SOs possuem esta fila

Filas de Escalonamento e Escalonadores

- Os processos que já estão na memória principal e estão no estado de pronto, são mantidos na fila de prontos, onde esperam para utilizar a CPU.
- Quando um processo *P* ganha a CPU, ele executa por um período até, por exemplo, fazer um pedido, de uma operação de E/S, passando para o estado bloqueado.
- Caso o dispositivo esteja ocupado com um pedido de E/S de algum outro processo, então, *P* terá que esperar pelo dispositivo

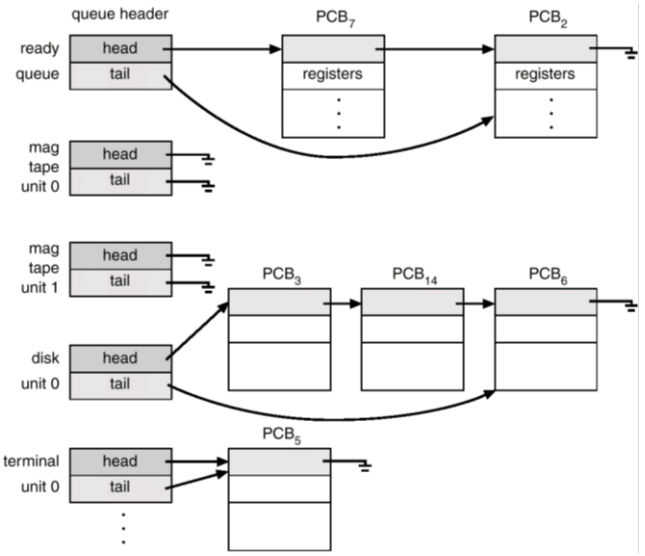
Filas de Escalonamento e Escalonadores

- Após o dispositivo de E/S (dedicado ou compartilhado) ser alocado a P, a operação de E/S será realizada. P continua no estado de bloqueado, esperando pelo fim da operação de E/S, quando passará para o estado de pronto
- Os processos esperando por um determinado dispositivo de E/S são mantidos na fila de E/S desde dispositivo. Cada dispositivo possui sua fila E/S

Filas de Escalonamento e Escalonadores

- Todas estas filas não são necessariamente FIFO
- Na verdade, os PCBs dos processos são mantidos nas filas. Filas são geralmente implementadas como listas encadeadas.

Filas de Escalonamento e Escalonadores



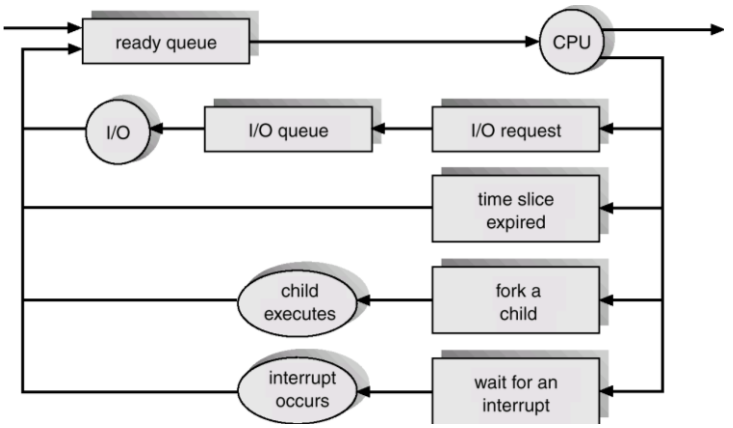
25

Filas de Escalonamento e Escalonadores

- Esquema de escalonamento de processos do SO é representado por uma rede de filas.

26

Filas de Escalonamento e Escalonadores



Filas de Escalonamento e Escalonadores

- Escalonador de longo prazo: seleciona processos da fila de jobs e carrega-os para memória, para competirem pelo CPU
- Escalonador de curso prazo: seleciona um processo, dentre os processos no estado de pronto da fila de prontos, e aloca a CPU para ele.

Filas de Escalonamento e Escalonadores

- ECP: seleciona um processo para CPU bastante freqüentemente. Precisa ser rápido, devido ao curto intervalo entre execuções
- ELP: executa muito menos freqüentemente. Controla o grau de multiprogramação do sistema (número de processos na memória principal, compartilhando a CPU)

Filas de Escalonamento e Escalonadores

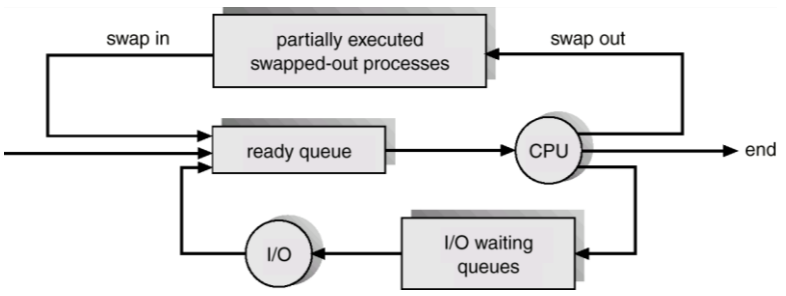
- Tempo gasto com escalonamento é overhead do SO para oferecer multiprogramação.
- Alguns SOs não possuem escalonador de longo prazo. Eles carregam na memória todo novo processo, passando-o para o escalonador de curto prazo

Filas de Escalonamento e Escalonadores

- Alguns SOs adicionam um nível intermediário de escalonamento: escalonador de médio prazo. Às vezes pode ser vantajoso remover processos da memória (e do compartilhamento da CPU), e reduzir o grau de multiprogramação. Mais tarde, os processos são recarregados na memória, e sua execução continua de onde parou
- Escalonador de médio prazo escolhe:
  - Dentre os processos na memória, quais vão ser retirados
  - Depois, dentre os processos retirados, quais vão ser recarregados.

Filas de Escalonamento e Escalonadores

Escalonamento de médio prazo



Operações com Processos

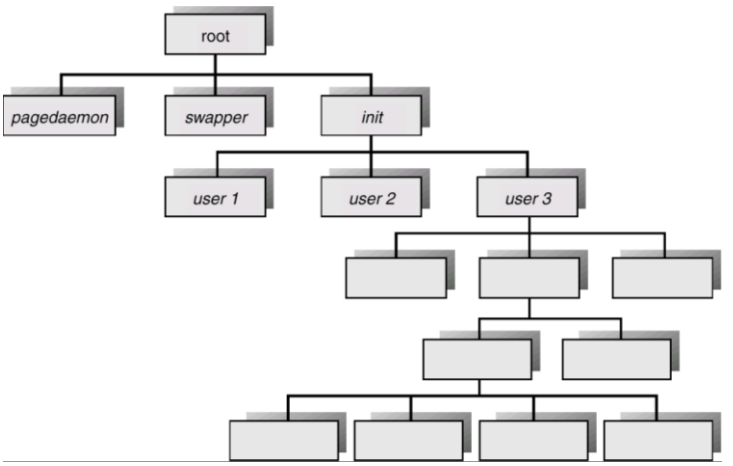
- Processos são criados e terminados dinamicamente
- SO → provê mecanismos para criação e terminação de processos

Operações com Processos
Criação de processo

- Processo → durante sua execução pode criar novos processos por meio de chamadas ao sistema do tipo "create process"
  - Processo que criou → chamado de processo pai
  - Processo criado (novo processo) → chamado de processo filho
  - Cada novo processo pode criar outros processos, formando uma árvore de processos

Operações com Processos

Árvore de processos típica de um sistema UNIX



Operações com Processos

Criação de processo

- Quando um processo P_p cria um novo processo P_f, em relação à execução de P_p e P_f:
  - P_p continua a executar concorrentemente com seus processos filhos; ou
  - P_p espera até que alguns ou todos os seus processos filhos tenham terminado

Operações com Processos
Criação de processo

- Em relação ao espaços de endereçamento de :
  - P_f é uma duplicação de ; ou
  - P_f tem um programa diferente carregado nele
- Exemplo:
  - **fork()** → system call que cria um novo processo
  - **execve()** → system call que substitui a imagem (espaço de endereçamento de memória) de um processo por outro

Operações com Processos
Terminação de processos

- Um processo termina quando executa seu último comando e faz chamada ao sistema do tipo "exit", pedindo para o SO destruí-lo.
- Todos os recursos do processo (memória, arquivos, buffers, …) são desalocados pelo SO
- Processo pode causa a terminação de outro, por meio de uma chamada ao sistema do tipo "terminate process", passando o ID do processo a ser terminado (normalmente somente o processo pai pode terminar os seus processos filhos)

Operações com Processos
Terminação de processos

- Alguns SOs não permitem que um processo filho exista, se seu pai já terminou.
- Quando um processo termina, todos os seus filhos também são terminados → chamado de terminação em cascata

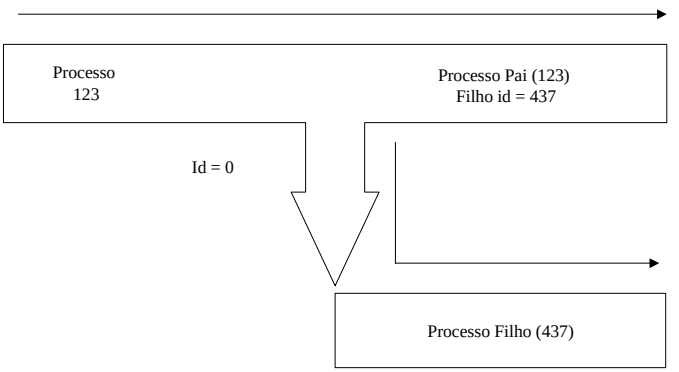
Operações com Processos
Exemplo – fork()

- Chamada de sistema *fork*:
  - Origina uma hierarquia de processos (tree)
  - Duas possibilidades em termos de execução:
    - Processo pai executa concorrentemente ao filho
    - Processo pai espera até o(s) filho(s) terminarem, usa chamada de sistema *wait*
  - Duas possibilidades em termos de espaço de endereçamento:
    - Processo filho é uma duplicação do processo pai
    - Processo filho tem um programa carregado nele utilizando a chamada execve

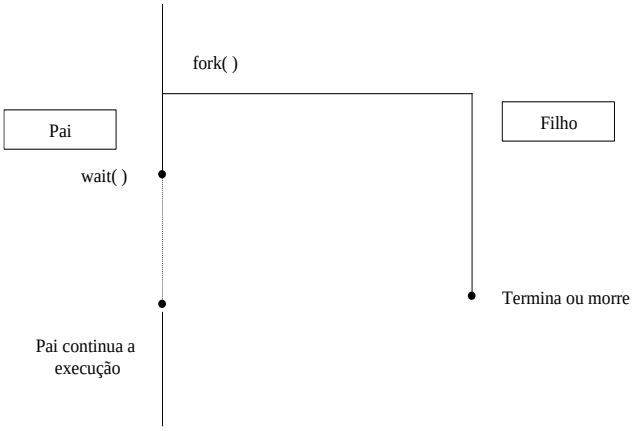
Operações com Processos

```
int pid;

pid = fork();

if (pid == 0) {
        /* código do processo filho */
        exit(0);
}
/*código do processo pai */
wait(...);
```


Operações com Processos



Operações com Processos



Operações com Processos

- Terminação
  - Executa última instrução ou chamada *exit*
  - Pode enviar dados para o pai via wait;
  - Um processo pode causar a terminação de outro (via system call abort). Normalmente somente o pai a utiliza
  - Razões para o processo filho ser terminado:
    - Processo filho excedeu recursos alocados
    - Tarefa solicitada ao processo filho não é mais necessária

Implementação de Processos (MINIX)

- O Sistema Operacional mantém uma tabela (matriz de estruturas), chamada de tabela de processos, com uma entrada para cada processo
- Cada entrada → PCB do processo.
- Os seguintes módulos são mantidos:

Implementação de Processos (MINIX)

Process management	Memory management	File management
Registers	Pointer to text segment	UMASK mask
Program counter	Pointer to data segment	Root directory
Program status word	Pointer to bss segment	Working directory
Stack pointer	Exit status	File descriptors
Process state	Signal status	Effective uid
Time when process started	Process id	Effective gid
CPU time used	Parent process	System call parameters
Children's CPU time	Process group	Various flag bits
Time of next alarm	Real uid	
Message queue pointers	Effective	
Pending signal bits	Real gid	
Process id	Effective gid	
Various flag bits	Bit maps for signals	
	Various flag bits	

Figure 2-4. Some of the fields of the MINIX process table.

Implementação de Processos (MINIX)

Múltiplos processos seqüenciais

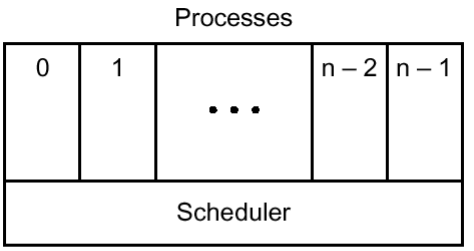
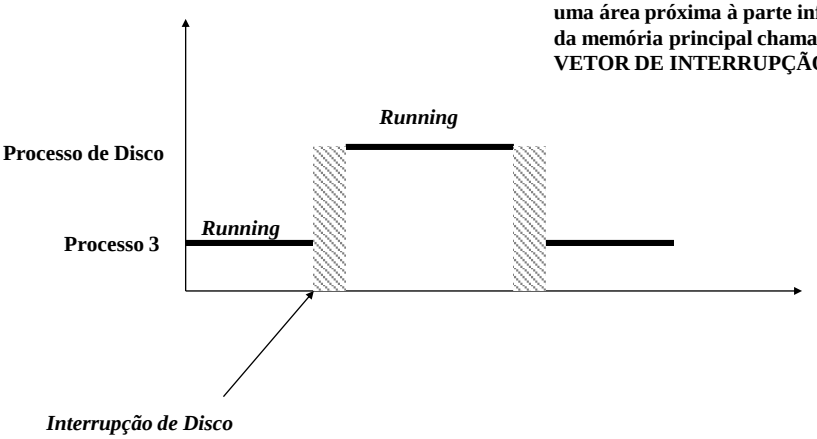


Figure 2-3. The lowest layer of a process-structured operating system handles interrupts and scheduling. Above that layer are sequential processes.

Implementação de Processos (MINIX)

Associado a cada dispositivo de E/S (ex: disquete, discos rígidos, temporizadores e terminais), existe uma área próxima à parte inferior da memória principal chamada VETOR DE INTERRUPÇÃO



Implementação de Processos (MINIX)

1. Hardware stacks program counter, etc.
2. Hardware loads new program counter from interrupt vector.
3. Assembly language procedure saves registers.
4. Assembly language procedure sets up new stack.
5. C interrupt service runs (typically reads and buffers input).
6. Scheduler marks waiting task as ready.
7. Scheduler decides which process is to run next.
8. C procedure returns to the assembly code.
9. Assembly language procedure starts up new current process.

Figure 2-5. Skeleton of what the lowest level of the operating system does when an interrupt occurs.