Processos

- Conceito de processo
- Escalonamento de processos
- Operações em processos
- Manipulação de processos em Linux/Unix

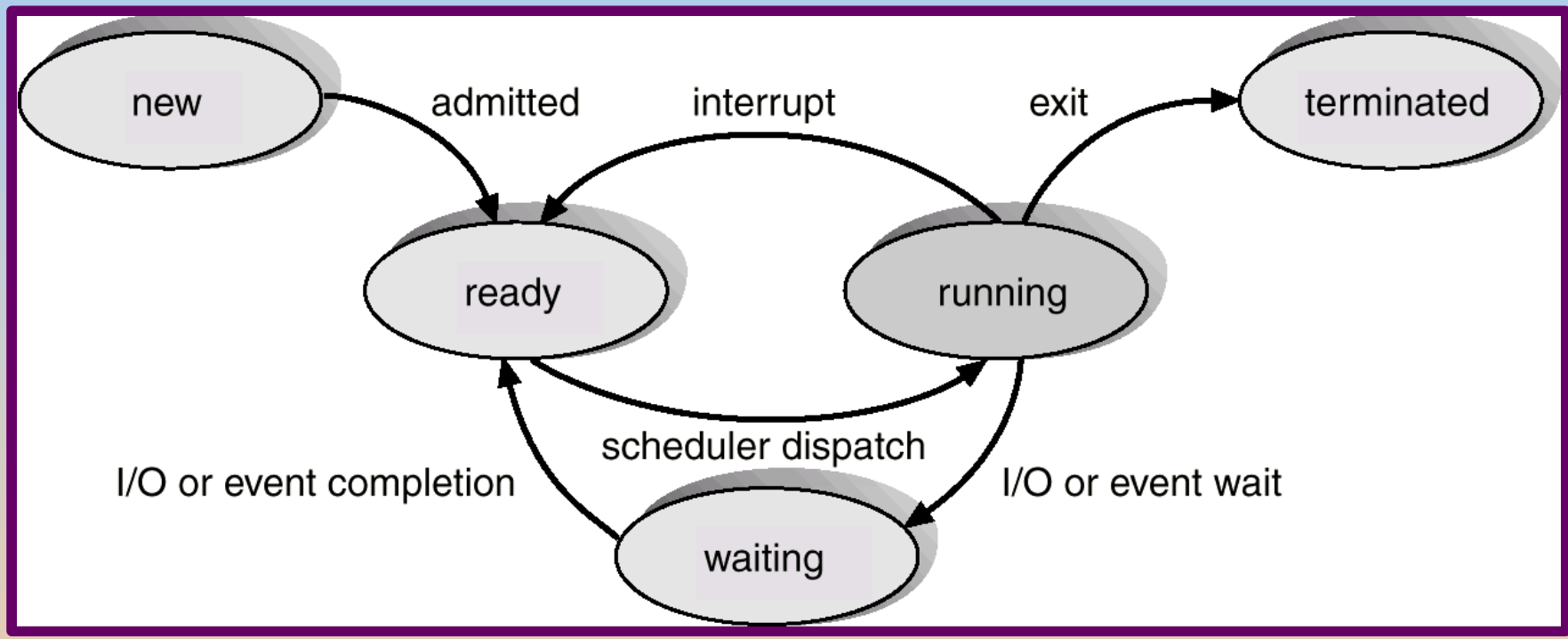
Conceito de Processo

- Uma sistema operacional executa uma série de programas:
  - Sistemas em lote(batch) – jobs
  - Sistemas de tempo compartilhado – programas do usuário ou tarefas
- Processo – um programa em execução; a execução de um processo precisa progredir sequencialmente.
- Um processo inclui:
  - Contador de programa
  - Pilha de execução
  - Seção de dados

Estados de um Processo

- Enquanto um processo executa, ele pode passar por vários estados:
  - **Novo(new)**: Processo que acabou de ser criado.
  - **Em execução(running)**: Instruções estão sendo executadas.
  - **Supenso(waiting)**: O processo está esperando algum evento acontecer.
  - **Pronto(ready)**: Um processo que está esperando para usar a CPU.
  - **Terminado(terminated)**: O processo cuja execução terminou..

Diagrama de Estados para um Processo

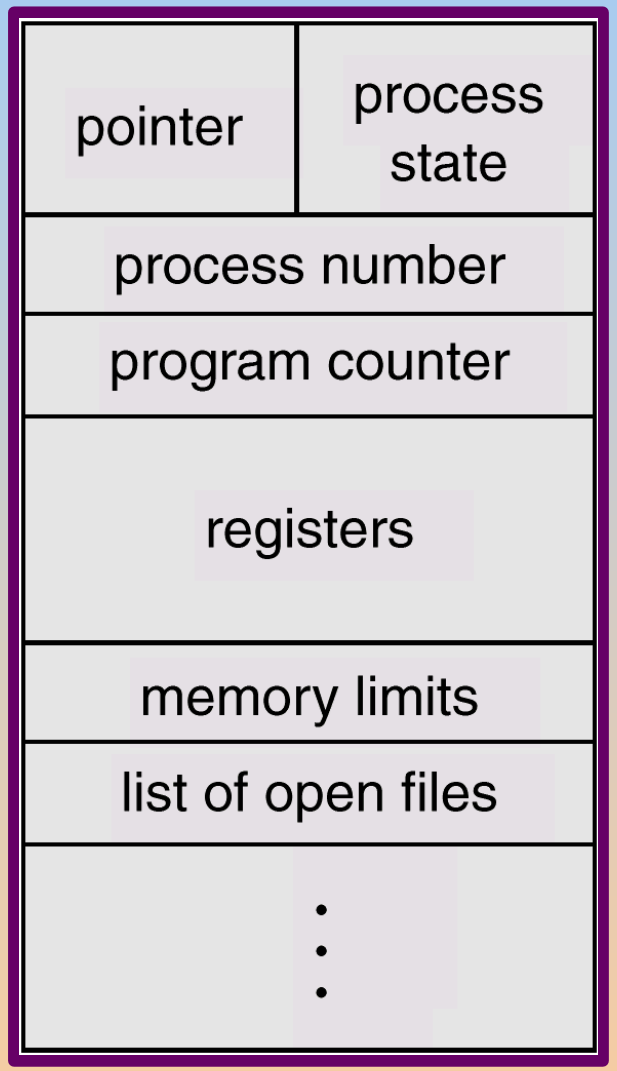


Process Control Block (PCB)
Bloco de Controle de Processo

Armazena informação associada a cada processo:

- Estado do processo
- Contador de programa
- Registradores da CPU
- Informação de escalonamento da CPU
- Informação de gerenciamento de memória
- Tempo de CPU utilizado
- Status de E/S alocados para o processo

Exemplo de um PCB

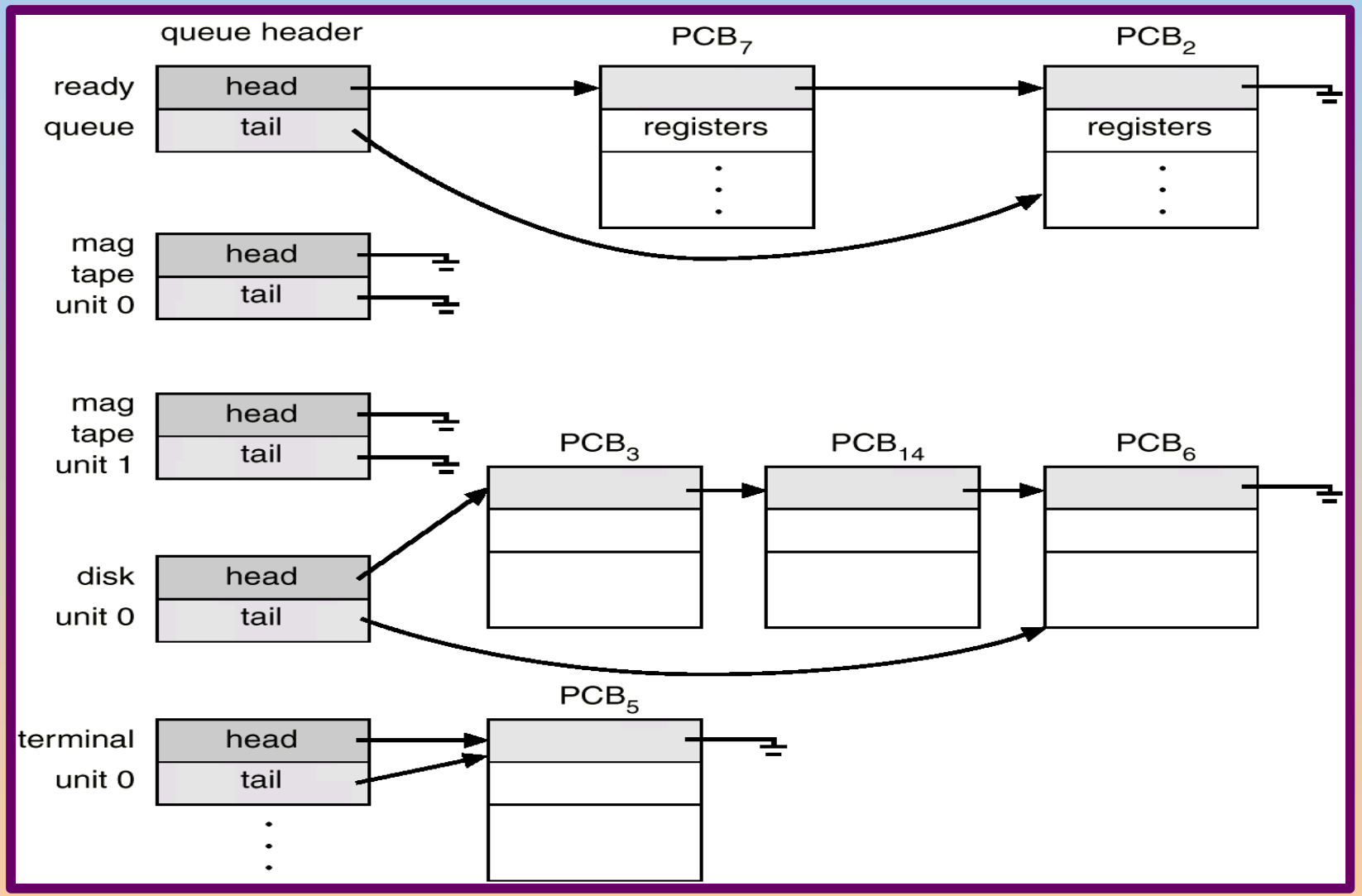


Filas de escalonamento de processos

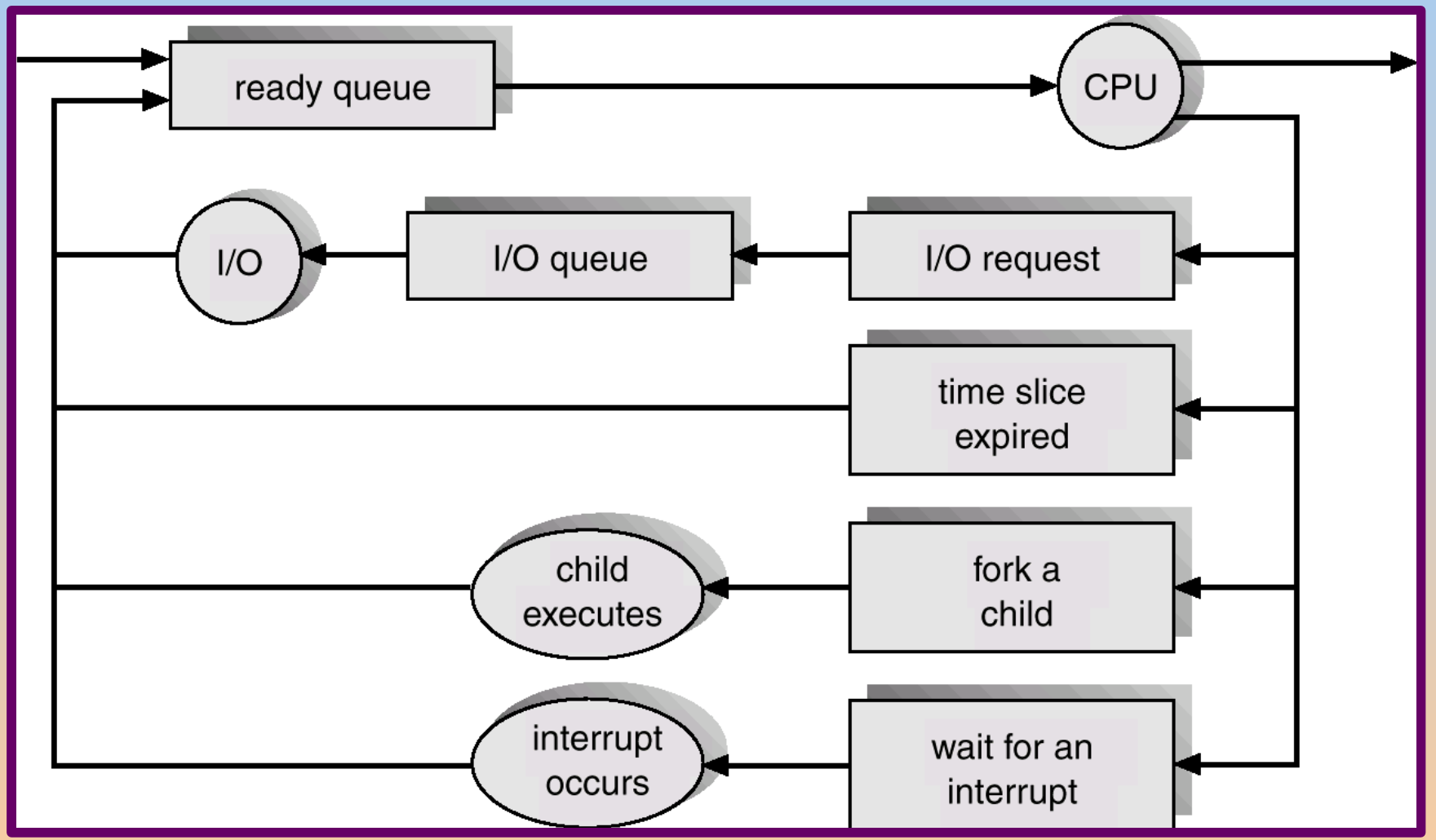
- Fila de tarefas – conjunto de todos os processos sob controle do sistema.
- Fila de processos prontos – conjunto de todos os processos alocados na memória, prontos e esperando para execução.
- Fila de espera para dispositivos – conjunto de todos os processos esperando por algum dispositivo de E/S.

Durante seu ciclo de vida, um processo pode migrar entre estas várias filas.

Filas de espera por dispositivos e de processos prontos



Representação de escalonamento de processos



Tipos de processos

- Processos podem ser divididos em:
  - Processos dependentes de E/S(I/O-*bound process)* – gastam mais tempo com operações de E/S do que com computações na CPU; precisam de tempo de utilização de CPU maior.
  - Processos dependentes da CPU(*CPU-bound process)* – gastam mais tempo fazendo computações.

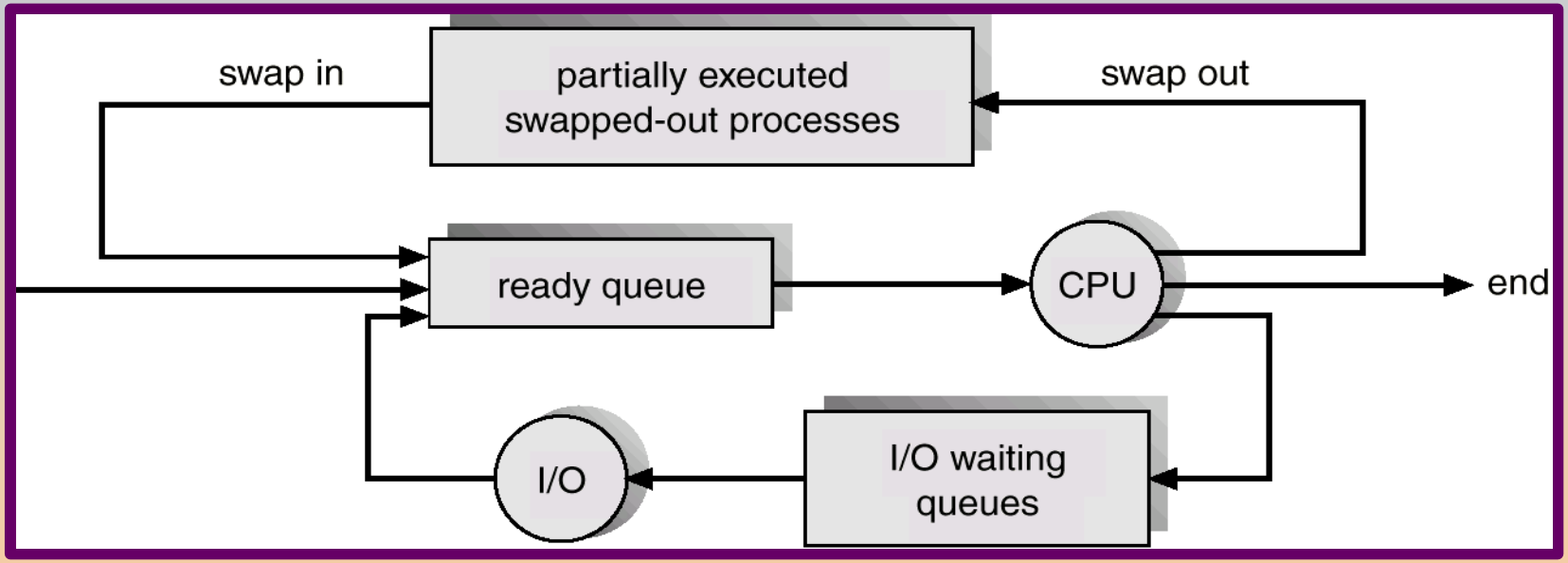
Escalonadores(Schedulers)

- **Escalonador de processos** (job scheduler) – seleciona que processos deverão ser carregados, normalmente do do disco, na fila de processos prontos.

- **Escalonador de CPU** (CPU scheduler) – seleciona um dentre os processos que estão prontos para serem executados e a aloca a CPU para esse processo.

Escalonador Intermediário

Alguns sistemas ainda podem incluir um mecanismo de escalonamento intermediário. A idéia básica é que pode ser vantajoso remover processos da memória principal, reduzindo o grau de multiprogramação.



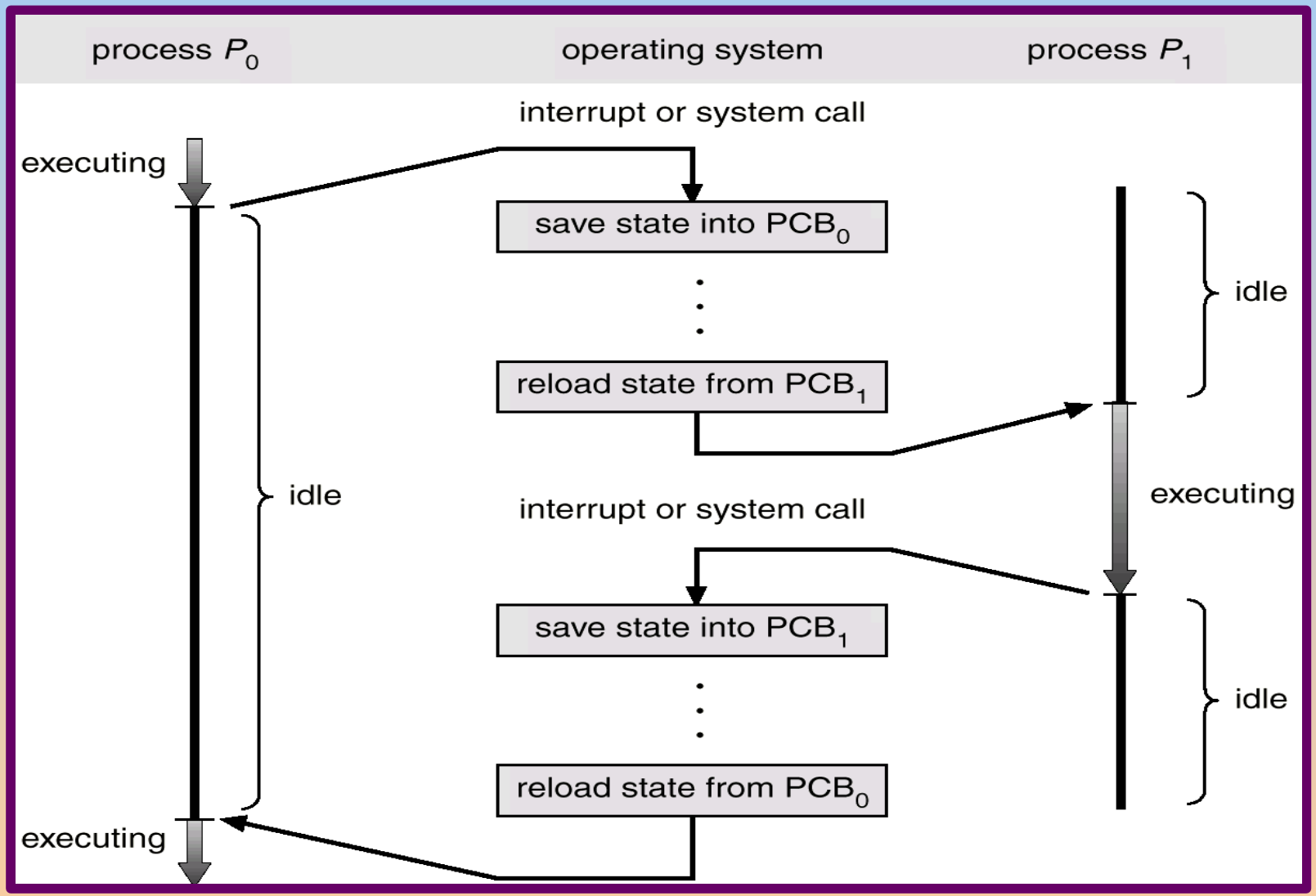
Escalonadores(Cont.)

- Escalonador de CPU é invocado muito frequentemente (millisegundos) $\Rightarrow$ (precisa ser rápido).
- Escalonador de processos é invocado pouco frequentemente (segundos,minutos) $\Rightarrow$ (pode ser lento).
- O escalonador de processos controla o grau de multiprogramação .

Mudança de contexto de processos

- Quando a CPU troca de processo, o sistema operacional precisa salvar o processo velho e carregar o estado salvo do novo processo.
- Tempo para mudança de contexto é um custo para o sistema operacional, pois ele não realiza nenhum trabalho útil durante a mudança de contexto.
- Tempo para mudança de contexto é altamente dependente do hardware.

Mudança de contexto de processos



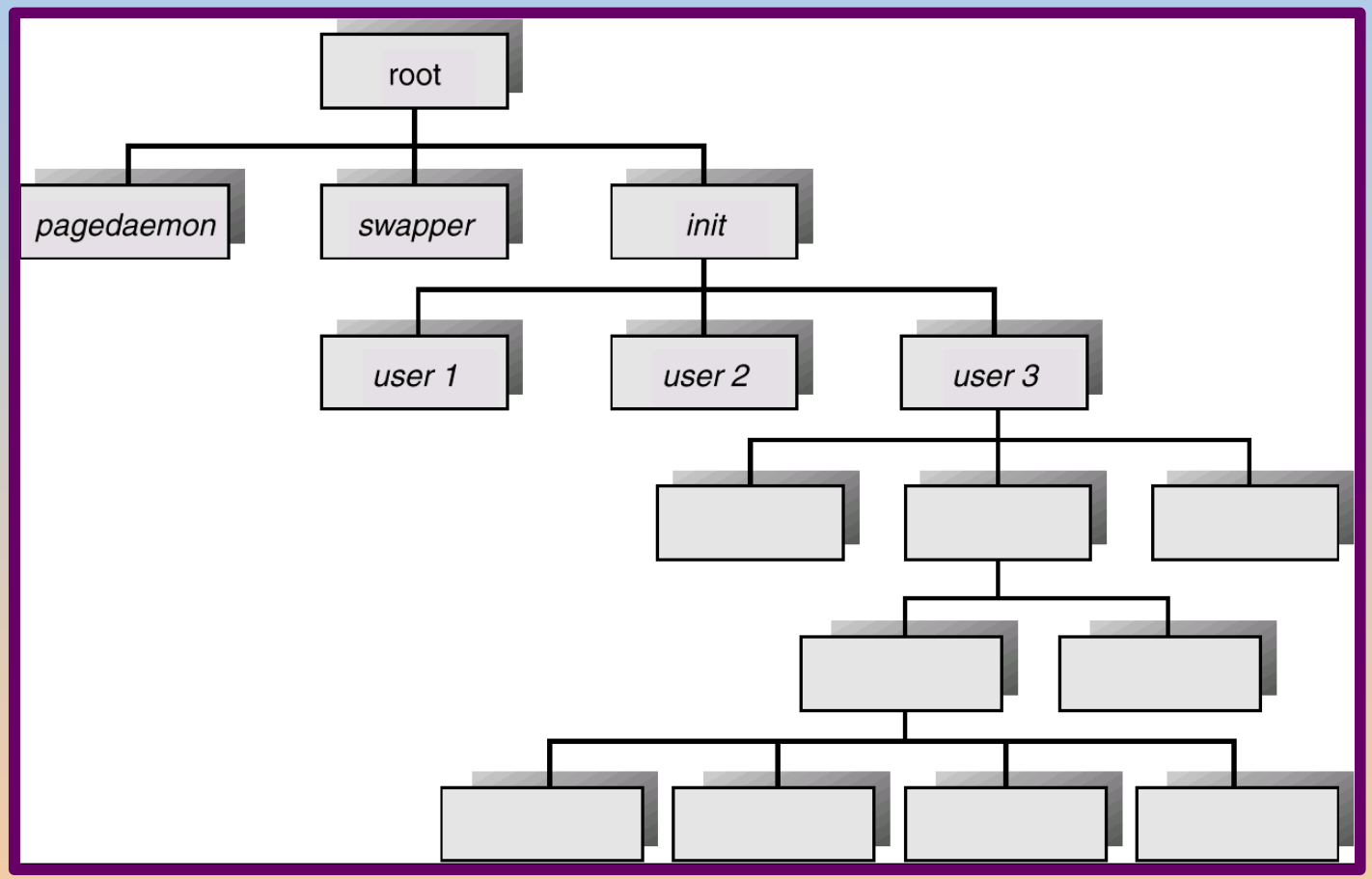
Criação de Processos

- Um processo-pai cria processos-filho que, por sua vez, criam outros processos, formando uma árvore de processos.
- Compartilhamento de recursos
  - Pai e filhos compartilham todos os recursos.
  - Filhos compartilham um subconjunto dos recursos do pai.
  - Pai e filhos não compartilham recursos.
- Execução
  - Pai e filhos executam concorrentemente.
  - Pai espera até que filhos terminem a execução.

Criação de processos (Cont.)

- Espaço de endereçamento
  - Filho é uma duplicata do pai (filho executa a mesma coisa que o pai).
  - Filho tem um programa que deve ser carregado para sua execução.
- Exemplos UNIX/Linux
  - A system call **fork** cria um novo processo
  - A system call **exec** é usada após um fork para substituir o espaço de memória do processo com um novo programa.

Árvores de Processos num sistema UNIX



Exemplo I – Criação de Processos em UNIX

```
#include <unistd.h>

void main(){
    pid_t pid;
    pid=fork();     /* Tenta criar  um novo processo */
    if (pid==0)
            printf("Processo-filho rodando…");
    else if (pid>0)
              printf("Processo-Pai. Processo filho tem pid %d",pid);
            else printf("Processo não criado…");
}
```

Exemplo II – Execução de programas pelo processo-filho

```
#include <unistd.h>

void main(){
    pid_t pid;
    pid=fork();       /* Tenta criar  um novo processo */
    switch(pid){
        case –1:  printf(“Processo não criado…”);
                  break;
        case 0:   execl(“/bin/ls”,”ls”,”-l”,(char *)0);
                  break;
        default:  wait ((int *) 0);
                  printf(“Processo filho terminou de executar ls”);
                  exit(0);
    }
}
```

Finalização de Processos

- Processo executa o último bloco e solicita término para o sistema operacional (**exit**).
  - Ocorre saída de dados do filho para o pai (via **wait**).
  - Os recursos do processo terminado são desalocados pelo sistema operacional.
- Processo pai pode terminar a execução dos processos-filho: (**abort**) se:
  - Filho excedeu algum limite de utilização de algum recurso que lhe foi alocado.
  - A tarefa atribuída ao filho não é mais necessária.
  - O processo-pai está terminando:
    - O SO não permite que os processos-filho continuem se o pai termina.
    - Ocorre um término em cascata.