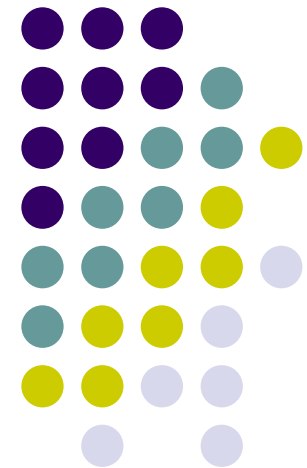


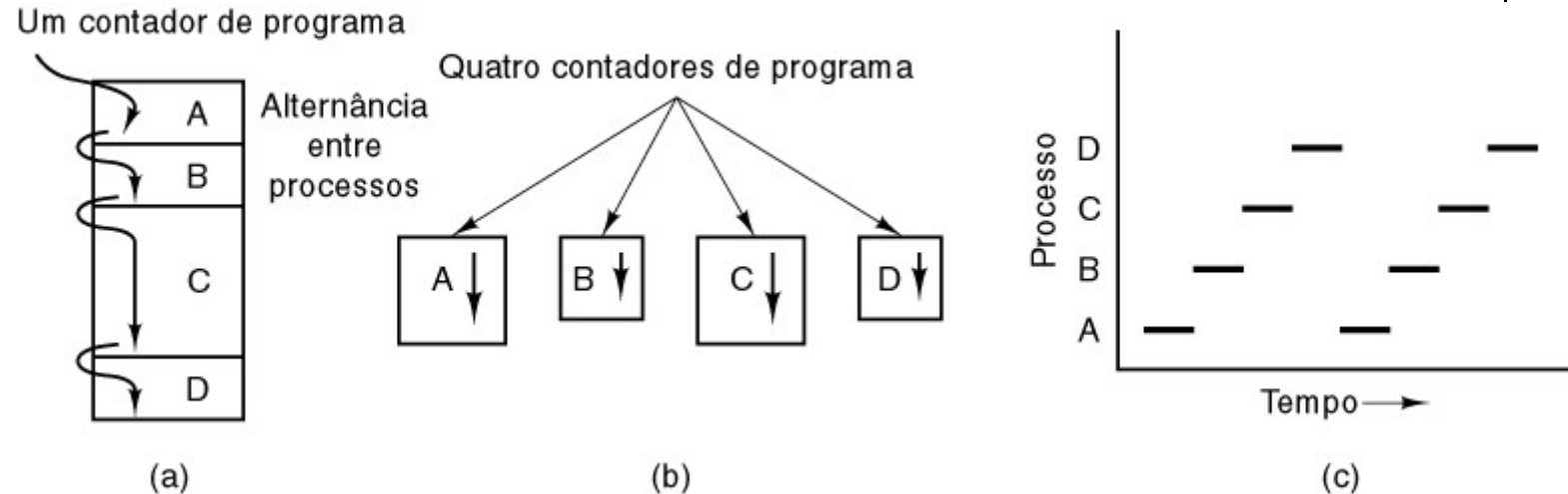
Sistemas de Computação

Processos e Escalonamento de Processos



Processos

O Modelo de Processo



- Considere a multiprogramação de 4 programas:
 - a) O contador de programa ou contador de instrução (PC ou CI) alternadamente assume endereços de cada programa
 - b) Conceitualmente são 4 processos sequenciais independentes
 - c) Somente um programa está ativo a cada momento

Criação de Processos

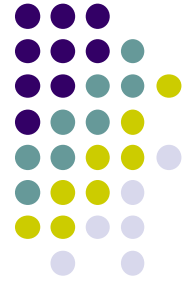


Um sistema pode executar um conjunto muito grande de processos simultâneos/concorrentes.

Principais eventos que levam à criação de processos:

1. Ao iniciar o sistema operacional
2. Um processo pai cria um novo processo (chamada `fork()`)
 - Usuário executa comando / inicia programa através da shell
 - Início de um programa em momento pré-determinado (através do cron daemon)

Término de Processos



Condições que levam ao término de processos:

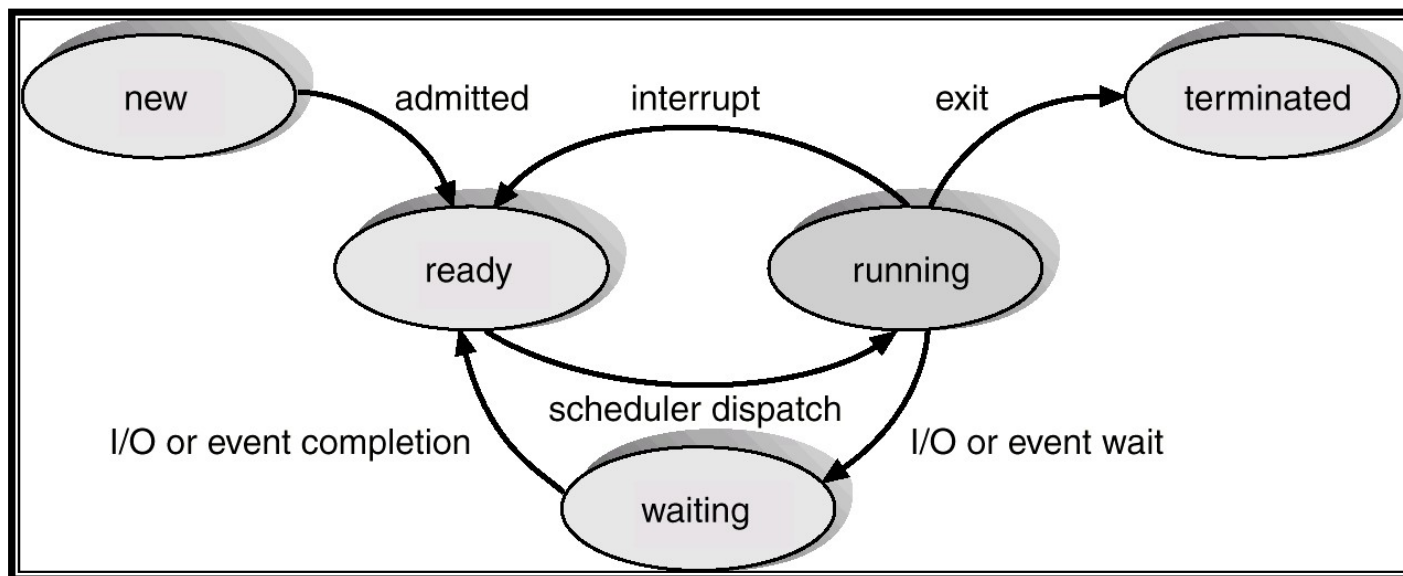
1. Saída normal (voluntária)
2. Saída por erro (voluntária)
3. Erro fatal (involuntário)
4. Cancelamento por um outro processo (involuntário), através de um sinal.

Hierarquias de Processos



- Processo pai cria um processo filho, processo filho pode criar seu próprio processo filho, etc.
- Forma-se uma hierarquia de processos
UNIX chama isso de “grupo de processos”
- Windows não possui o conceito de hierarquia de processos
Todos os processos são criados no mesmo nível

Estados de Processos



- Ao longo de sua execução, um processo pode assumir , os seguintes estados:
 - **new**: processo foi criado.
 - **running**: instruções sendo executadas.
 - **waiting**: Processo aguarda a ocorrência de algum sinal/interrupção.
 - **ready**: Processo aguarda alocação do processador.
 - **terminated**: Processo terminou a sua execução.

Implementação de Processos

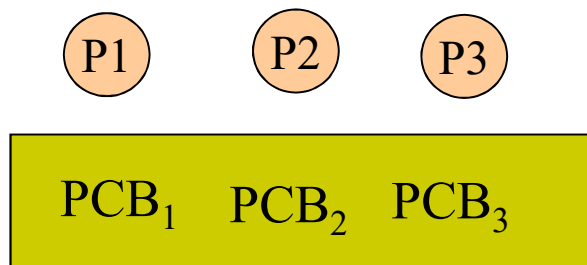


A cada processo estão associadas informações sobre o seu estado de execução (o seu *contexto de execução*),

Estas ficam armazenadas em uma entrada da Tabela de Processos (ou Process Control Block)

Gerenciamento de processos	Gerenciamento de memória	Gerenciamento de arquivos
Registradores Contador de programa Palavra de estado do programa Ponteiro de pilha Estado do processo Prioridade Parâmetros de escalonamento Identificador (ID) do processo Processo pai Grupo do processo Sinais Momento em que o processo iniciou Tempo usado da CPU Tempo de CPU do filho Momento do próximo alarme	Ponteiro para o segmento de código Ponteiro para o segmento de dados Ponteiro para o segmento de pilha	Diretório-raiz Diretório de trabalho Descritores de arquivos Identificador (ID) do usuário Identificador (ID) do grupo

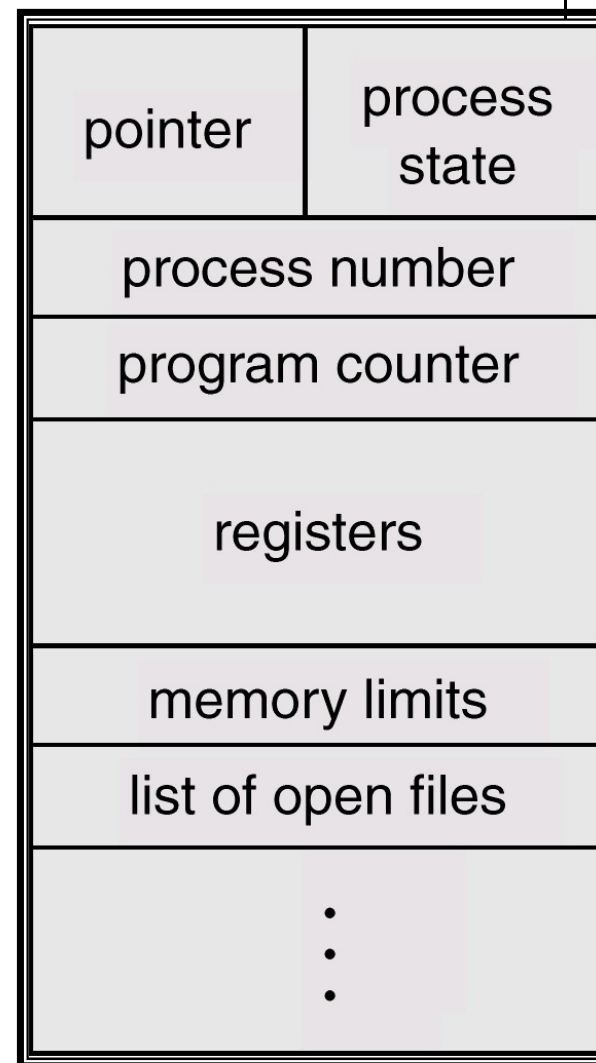
Process Control Block (PCB)



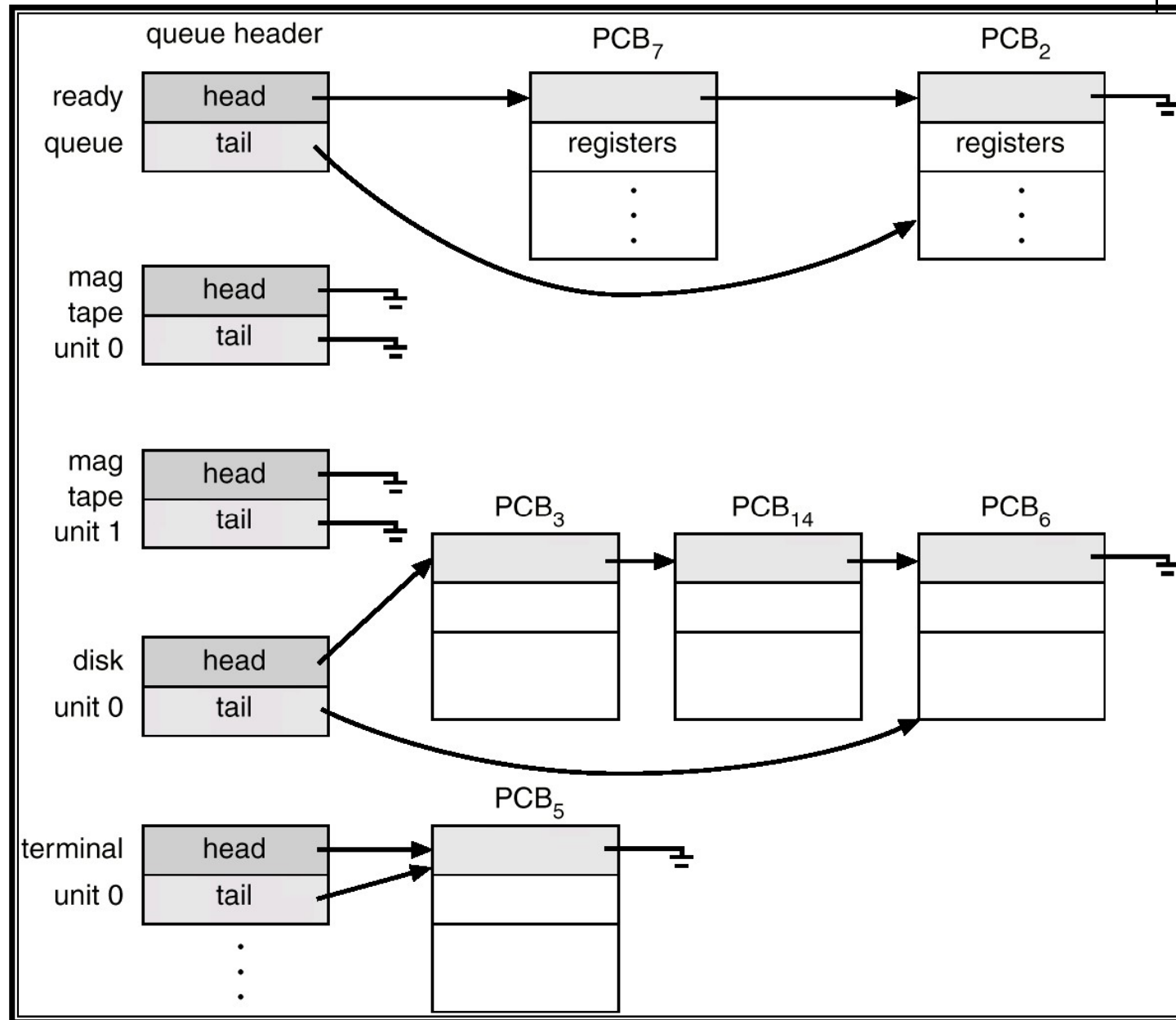
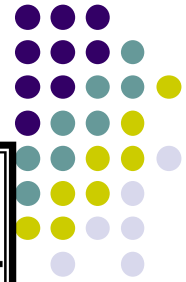
PCB é a representação de um processo no núcleo.

Pode ser uma entrada na *Tabela de Processos* ou um elemento na lista de processos.

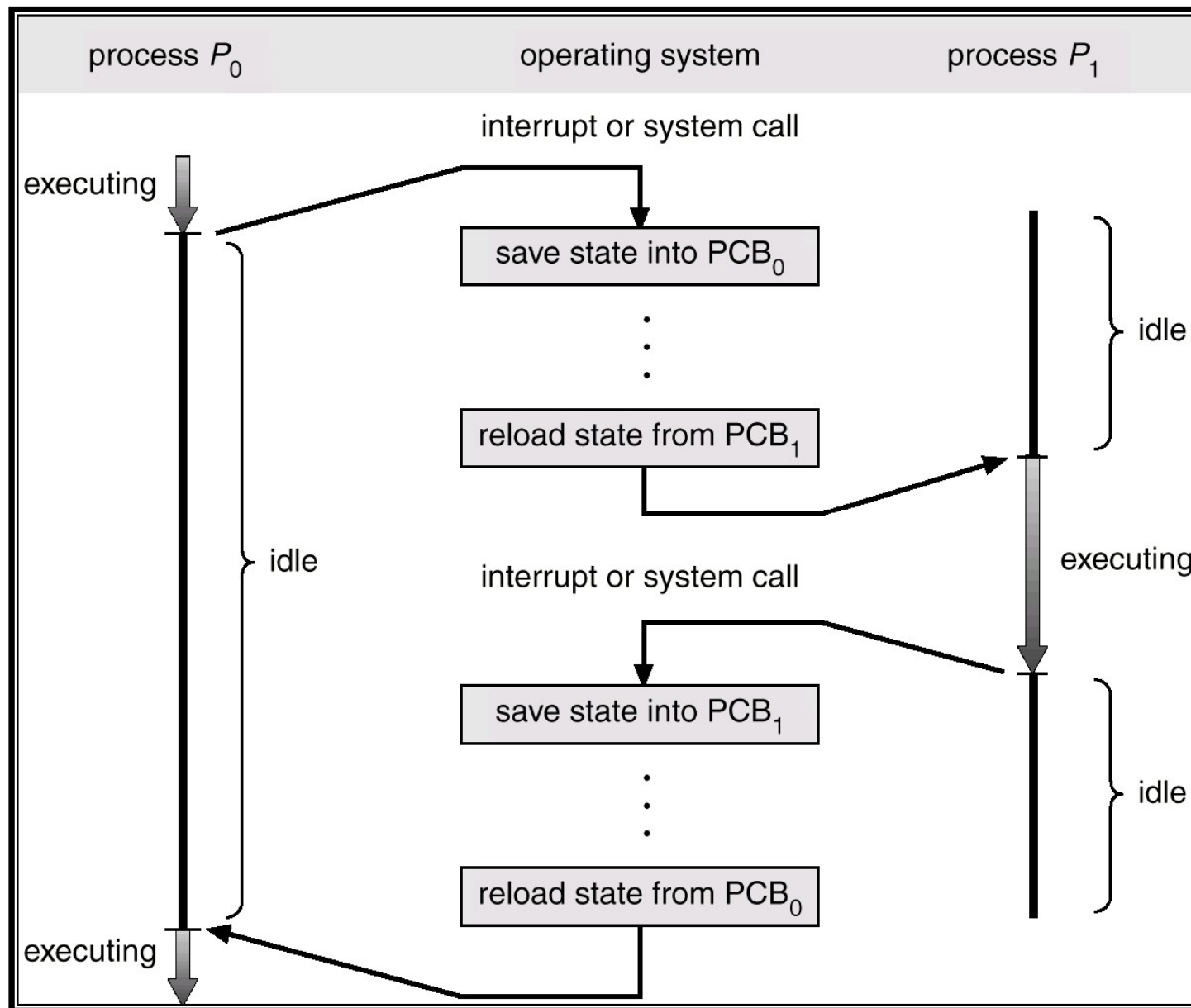
Para ser capaz de reiniciar um processo interrompido (ou esperando) o estado em que deixou a CPU precisa ser restaurado.



Fila dos prontos e de espera por E/S



Troca de Contexto

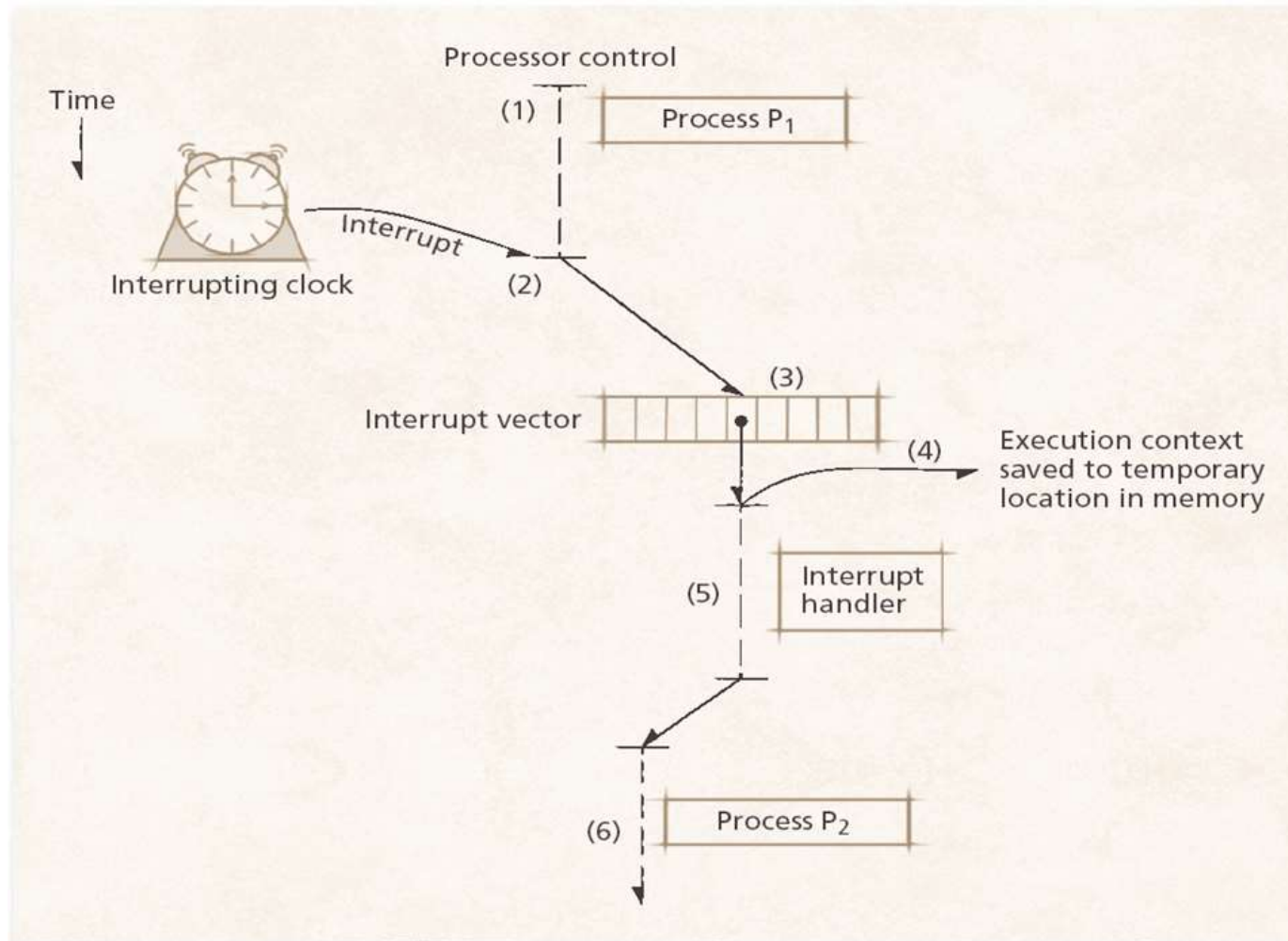


Troca de Contexto

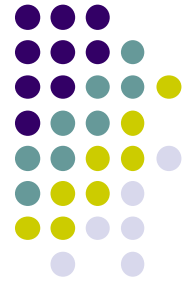


- Consiste de salvar o estado dos recursos em uso (especialmente CPU) do processo interrompido na tabela de processos, e carregar a CPU com um novo estado (PC, registradores, stack pointer, PSW, etc.)
- Esta troca precisa ser:
 - Completa e consistente
 - Eficiente
- O núcleo não pode ser interrompido durante o processo
 - Precisa-se garantir a atomicidade da operação
- Realizado por um tratador de interrupção genérico, tratador de interrupção de primeiro nível

Tratamento de Interrupções



- Para desviar o controle de execução, o valor do contador de programa (PC) da CPU precisa ser trocado.
- Isso é feito pelo HW ao receber uma interrupção.



Interrupções vs. Exceções

O conjunto de interrupções depende da arquitetura do sistema.

A especificação da Intel Architecture IA-32 distingue dois tipos de sinais que um processador pode receber:

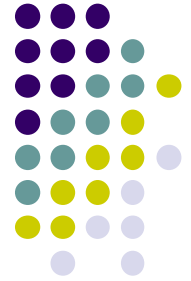
- **Interrupção**

- Notificam o processador que um evento ocorreu, e/ou que o estado de um recurso (p.ex. dispositivo de E/S) mudou
- Gerado por um dispositivo externo ao processador

- **Exceção**

- Indica a ocorrência de um erro, de hardware ou causado por uma instrução sendo executada
- Classificados como faults, traps ou aborts

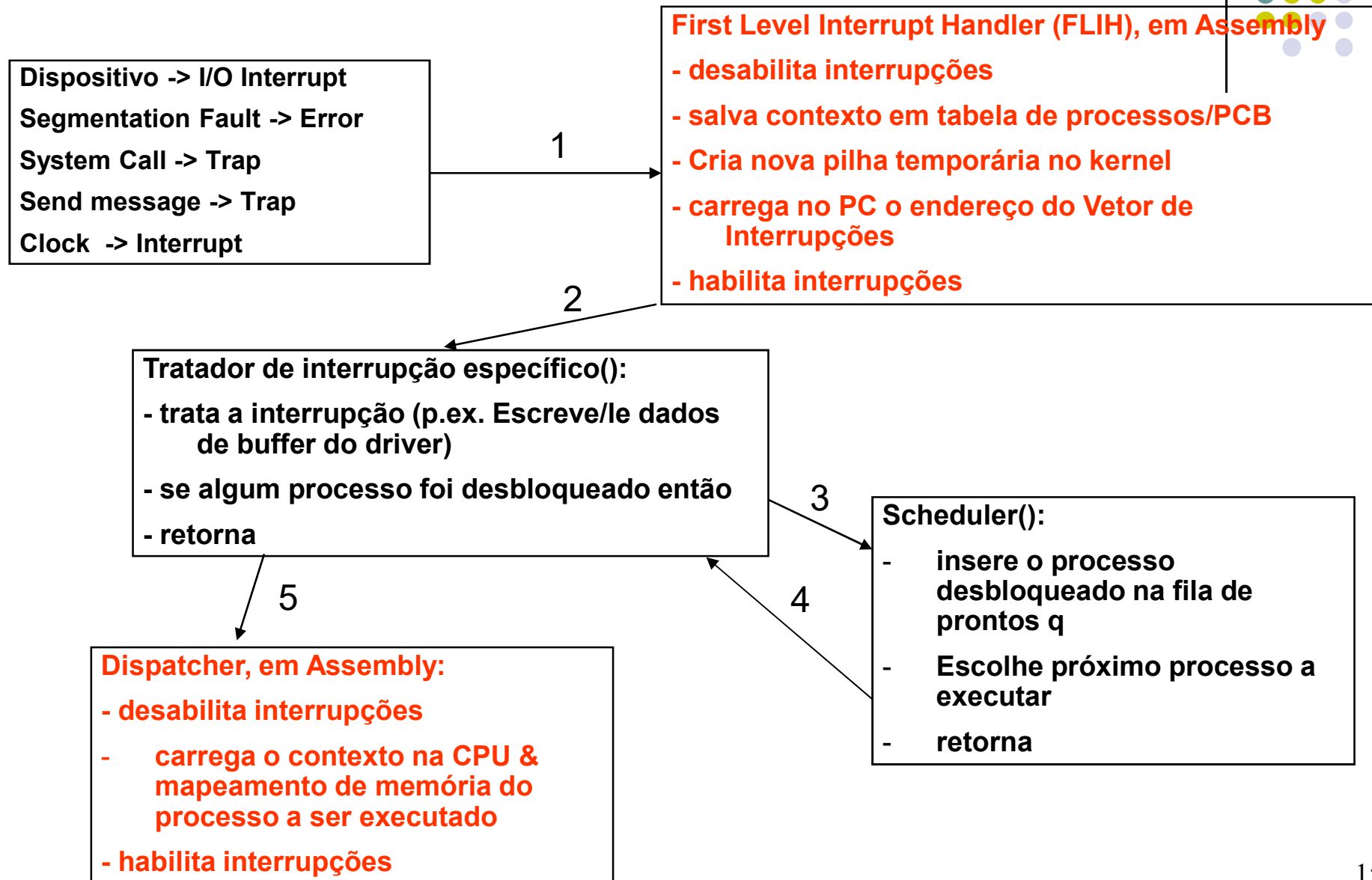
Tipos de Interrupções



Tipos de interrupção reconhecidos pela Arquitetura Intel IA-32:

<u>Tipo</u>	<u>Descrição para cada tipo</u>
I/O	Iniciados pelo HW, notificam o processador de que o estado do dispositivo de E/S mudou (p.ex. E/S finalizada)
Timer	Evento periódico para agendamento de ações e/ou monitoramento de desempenho
Inter-CPU	Em sistemas multi-processados, para comunicação e sincronismo entre processadores

Tratamento de Interrupções



Tratamento de Interrupções

Esqueleto do que o nível mais baixo do SO faz quando ocorre uma interrupção

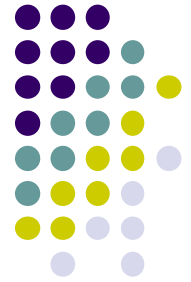


1. O hardware empilha o contador de programa etc.
2. O hardware carrega o novo contador de programa a partir do vetor de interrupção.
3. O procedimento em linguagem de montagem salva os registradores.
4. O procedimento em linguagem de montagem configura uma nova pilha.
5. O serviço de interrupção em C executa (em geral lê e armazena temporariamente a entrada).
6. O escalonador decide qual processo é o próximo a executar.
7. O procedimento em C retorna para o código em linguagem de montagem.
8. O procedimento em linguagem de montagem inicia o novo processo atual.

Vetor de Interrupção:

- Localizado em endereço baixo de memória (núcleo)
- Uma entrada para cada tipo de interrupção (trap, clock, E/S) e cada tipo de dispositivo (floppy, HD, teclado, mouse, etc.)
- Cada entrada contém endereço para um procedimento tratador da interrupção (tratamento do serviço da interrupção)

Escalonamento



- A cada instante um ou mais processos podem estar no estado *pronto*, e.g.:
 - Processos do sistema e de usuários
 - Processos de vários usuários (sistema time-sharing)
 - Mix de processos interativos e batch (simulação, folha de pagamento)
- Escalonador é responsável por gerenciar a fila de prontos, e escolher qual dos processos prontos vai ser o próximo a usar CPU

Escalonamento

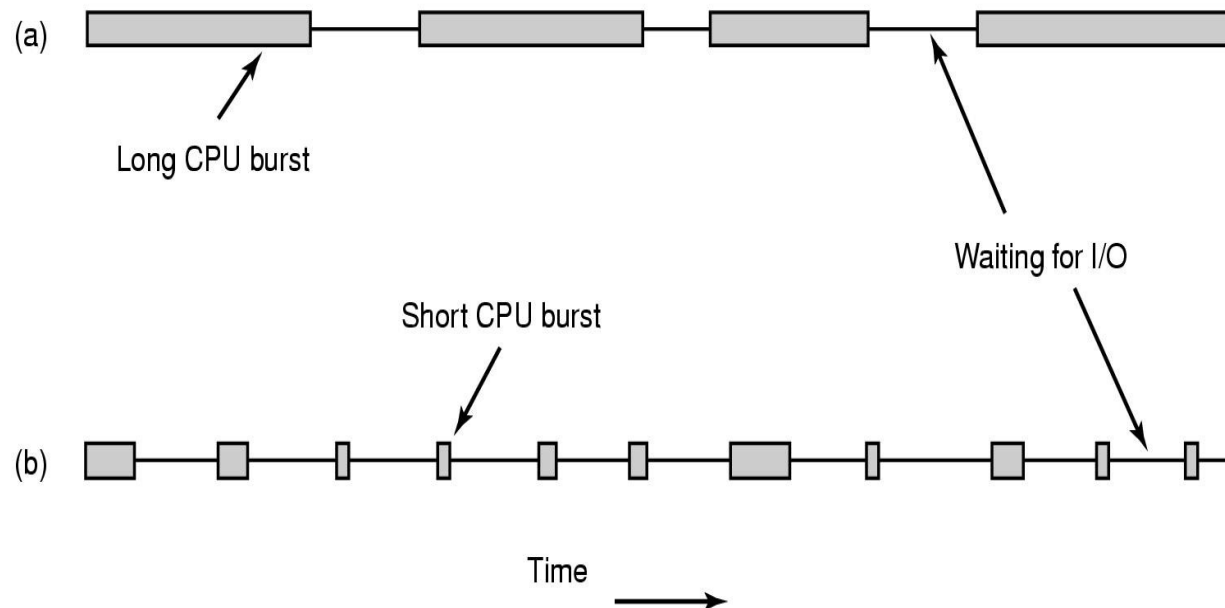


- O algoritmo poderá lidar com objetivos conflitantes, por exemplo:
 - Garantir justiça (*fairness*): cada processo ganha fatia igual da CPU
 - Aumentar eficiência: manter utilização de CPU alta (próxima a 100%)
 - Minimizar tempo de resposta (para processos interativos)
 - Minimizar de tempo de retorno (Δt entre início-fim de processos batch)
 - Maximizar taxa de saída: número de processos processados por unidade de tempo
- Sempre que se beneficia uma classe de processos, prejudica-se as outras classes.

Escalonamento

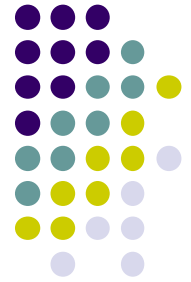


- A política de escalonamento deve ser independente do mecanismo (carregamento da CPU com um contexto)
- Têm parâmetros que precisam ser ajustados para:
 - maximizar a “satisfação geral” dos usuários e
 - garantir execução mais eficiente das tarefas essenciais ao sistema



Principal problema : o comportamento futuro de um processo não é previsível (fases de uso intensivo da CPU e fases de E/S frequente)

Escalonamento



1. Escalonamento de longo prazo
 - ao ser criado, processo vai para fila dos prontos
 - questão: quando isso é feito e em qual posição ele entra?
2. “dispatching” ou curto prazo
 - escolhe um dos processos da(s) lista(s) de prontos para executar

Usa-se o termo *escalonamento* para ambos

Formas de implementar o escalonador



- “embutido” na execução do processo
 - Ao final do tratamento da interrupção, o procedimento para escalonamento é chamado
 - Executa como parte do fluxo de controle do processo que estava em execução
- “autônomo”
 - Executa como um processo independente
 - Pode estar dedicado a uma CPU em uma arquitetura multi-core
 - Em máquinas com 1 processador, é executado uma vez a cada *quantum* de tempo
 - Há uma alternância entre o processo escalonador e os demais processos

Tipos de Escalonamento

Com relação:

- ao momento da invocação do escalonador:
 - **preemptivo**: processo corrente é interrompido (por exemplo, após ter decorrido *quantum* de tempo) e escalonador escolhe outro processo para executar
 - **não-preemptivo**: escalonador só é chamado quando processo termina ou é bloqueado
- ao método de seleção do processo mais prioritário:
 - Uso da função $P = \text{Priority}(p)$
 - Regra de desempate (para processos de mesma prioridade)
 - Escolha randômica
 - Cronológica (FIFO)
 - Cíclica (*Round Robin*)



Escalonamento por prioridade



- Função de prioridade retorna valor P para processo p :

$$P = \text{Priority}(p)$$

- Prioridade estática: não muda ao longo da execução de p ;
 - Prioridade dinâmica: muda em tempo de execução
-
- Prioridades separam todos os processos em níveis:
 - Implementado através de “filas de prontos” multi-nível (e.g. várias *Ready Lists* – RLs)
 - $p @ \text{RL}[i]$ executa antes de $q @ \text{RL}[j]$ se $\text{priority}(i) > \text{priority}(j)$
 - p, q no mesmo nível são ordenados usando outro critério

Algoritmo geral para um escalonador preemptivo para multi-processador



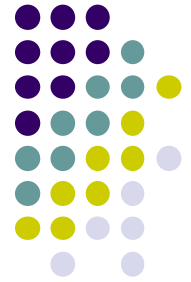
```
Scheduler() {  
    do { // some CPU is idle  
        Find highest priority ready_a process p;  
        Find a free cpu;  
        if (cpu != NIL) Allocate_CPU(p,cpu);  
    } while (cpu != NIL);  
  
    do { // all CPUs are in use  
        Find highest priority ready_a process p;  
        Find lowest priority running process q;  
        if (Priority(p) > Priority(q)) Preempt(p,q);  
    } while (Priority(p) > Priority(q));  
  
    if (self->Status.Type!='running') Preempt(p,self);  
}
```




Parâmetros típicos da Função Prioridade

- Internos (do sistema)
 - Tipo do processo (sistema vs usuário)
 - Quantidade de memória necessária
 - Tempo total de CPU requisitado
 - Tempo de serviço obtido / alcançado
 - Tempo total de permanência no sistema
- Externos
 - Prazo para término de ação (Deadline)
 - Prioridade do usuário: root vs normal (função na empresa, valor desembolsado)

Tipos de Escalonamento



- First In First Out (FIFO)
Execução por ordem de chegada
- Shortest Job First (SJF)
Execução por tempo de CPU
- Escalonamento com múltiplas filas
(ML- múltiplos níveis)
- Algoritmos de escalonamento Multi-level com feedback (MLF)
- Outras políticas de escalonamento

First In First Out (FIFO)

Execução por ordem de chegada



<u>Job</u>	<u>Tempo de CPU</u>
A	8
B	1
C	1



Tempo médio de espera $(0 + 8 + 9) / 3 = 5.7$

B	C	A
---	---	---

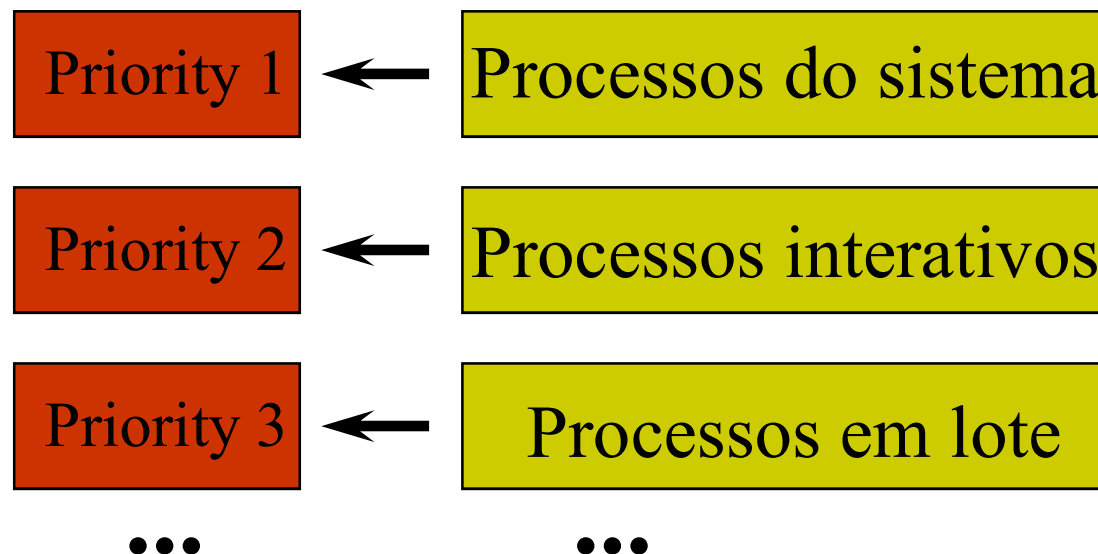
Tempo médio de espera ótimo:

28

Escalonamento com múltiplas filas (ML- múltiplos níveis)



- Para sistemas com mix de processos interativos e em lote
- Processos são classificados segundo prioridade, e cada classe tem sua própria fila de prontos.



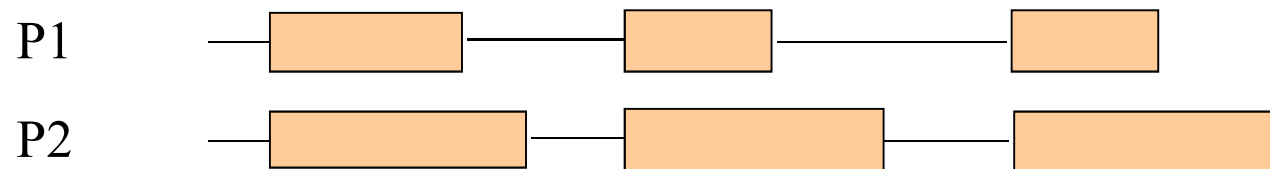
- Executa todos de prioridade 1, depois 2, ...
- Para evitar o *problema de inanição* (alguns processos nunca ganham a vez), pode-se definir períodos de tempo máximos para cada categoria: por exemplo, 70% para 1, 20% para 2, ...

Múltiplas Filas (ML)

Princípio Geral



No ML adaptou-se SJF para processos interativos, considerando o tempo efetivo de CPU entre requisições de E/S



Principal problema: como descobrir qual dos processos prontos requisitará a CPU por menor período de tempo.

Princípio adotado: *Estimar a próxima fatia de tempo necessária, olhando para o passado.*

Exemplo: Seja T_0 a estimativa de tempo de uso de CPU e T_1 o tempo de CPU efetivamente utilizado da última vez. Então, a estimativa para a próxima vez, T_2 , deveria ser ajustada.

$$T_2 = \alpha * T_1 + (1 - \alpha) * T_0.$$

Se $\alpha > 0.5$ dá-se mais importância para o comportamento mais recente, e $\alpha < 0.5$ maior importância para o comportamento passado

Algoritmos de escalonamento Multi-level com feedback (MLF)



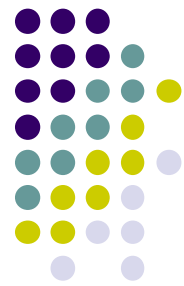
MLF (Multi-level with feedback):

- Similar ao ML, mas com uma prioridade que muda dinamicamente
- Todo processo começa no nível mais alto n
- Cada nível P prescreve em um tempo máximo t_p
- t_p aumenta à medida que P diminui
- geralmente:

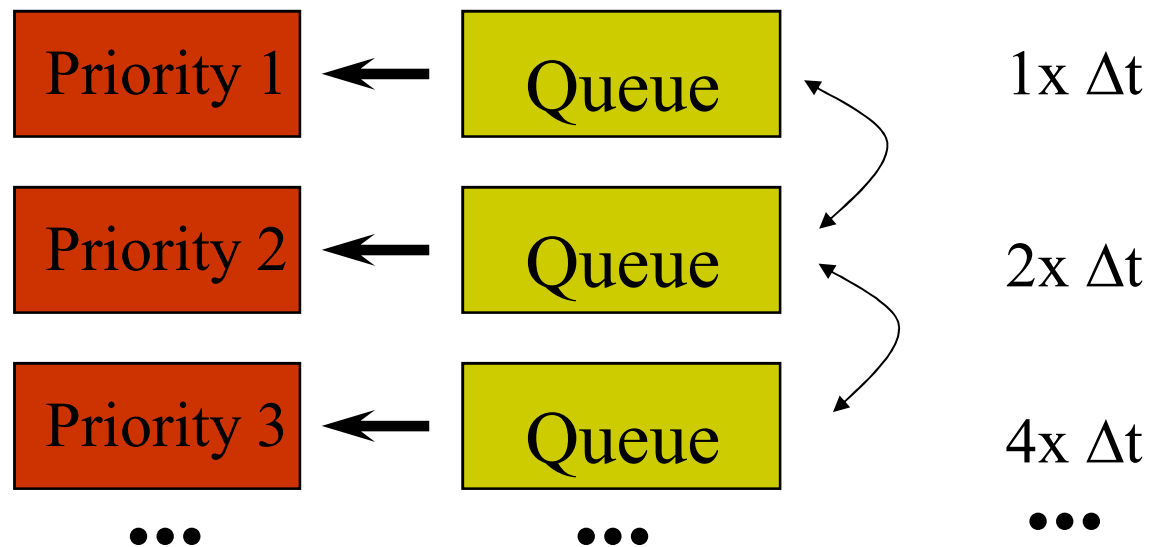
$$t_n = \Delta t \quad (\text{constante})$$

$$t_p = 2 \times t_{p+1}$$

Filas em múltiplos níveis com feedback (MLF)



Idéia: Maior prioridade para processos que precisam de fatia (ou *quantum*) de tempo (Δt) menor. Se um processo repetidamente gasta todo seu quantum Δt , passa para prioridade mais baixa.



- Problema: processos longos, p.ex. que precisam de $100x \Delta t$
 - Percorrem 7 prioridades: 1, 2, 4, 8, 16, 32, 64
- Grande vantagem para processos com alta frequência de E/S

Outras políticas de escalonamento



Escalonamento garantido

- cada um dos n usuários recebe aproximadamente $1/n$ dos ciclos de CPU
- Muito simples, só é feito para processos do usuário (e não de sistema)

Escalonamento por sorteio (*lottery scheduling*)

- Sorteio *quase* aleatório de procesos (todos ou em cada nível de prioridade)
- Vantagem: simplicidade e distribuição uniforme de valores sorteados geralmente garante igualdade de chances
- Para garantir justiça, impõe-se um limite no número de vezes que um processo pode ser sorteado em determinado período

Perguntas?

