Sistemas Operacionais

Gerência do processador

Capítulo 4

Introdução

- O uso da multiprogramação pressupõe a existência de simultânea de vários processos disputando o processador
- Necessidade de “intermediar” esta disputa de forma justa
  - Gerência do processador
    - Algoritmos de escalonamento

Implementação de processo

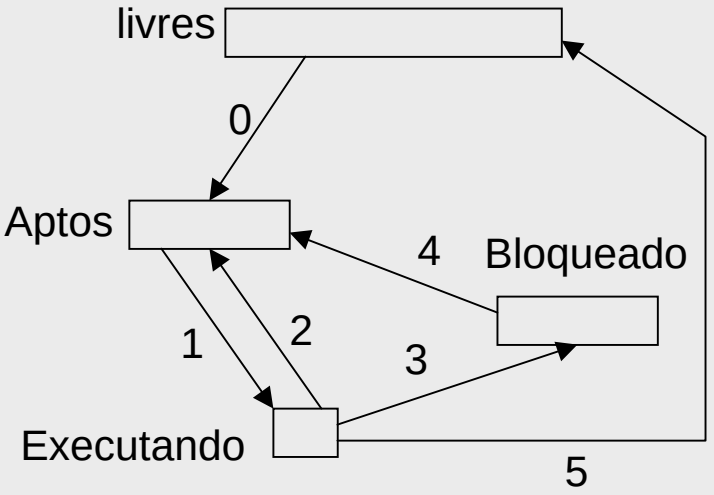
- Processo é um programa em execução
- Possui uma série de estados (apto, executando, bloqueado, etc) para representar sua evolução no tempo
- Necessidade de gerenciar processos
  - Organização de filas de processos nos diferentes estados
  - Determinar eventos que realizam a transição entre os estados
  - Determinar quando e como um processo tem direito a “utilizar” o processador para sua execução
- Sistema operacional necessita manter uma série de informações a respeito do processo

Bloco descritor de processo

- Abstração de processo é implementado através de uma estrutura de dados
  - Bloco descritor de processos (*Process Control Block - PCB*)
- Informações normalmente presentes no descritor de processo
  - Prioridade
  - Localização e tamanho na memória principal
  - Identificação de arquivos abertos
  - Informações de contabilidade (tempo CPU, espaço de memória, etc)
  - Estado do processador (apto, executando, bloqueando, etc)
  - Contexto de execução
  - Apontadores para encadeamento dos próprios descritores de processo
  - etc

Os processos e as filas

- Um processo sempre faz parte de alguma fila
- Geralmente os próprios descritores de processos são utilizados como elementos dessas filas:
  - Fila de livres
    - Número fixo (máximo) de processos
    - Alocação dinâmica
  - Fila de aptos
  - Fila de bloqueados
- Eventos realizam transição de uma fila a outra



Exemplo de implementação de processos (1)

- Estrutura de dados representado bloco descritor de processo

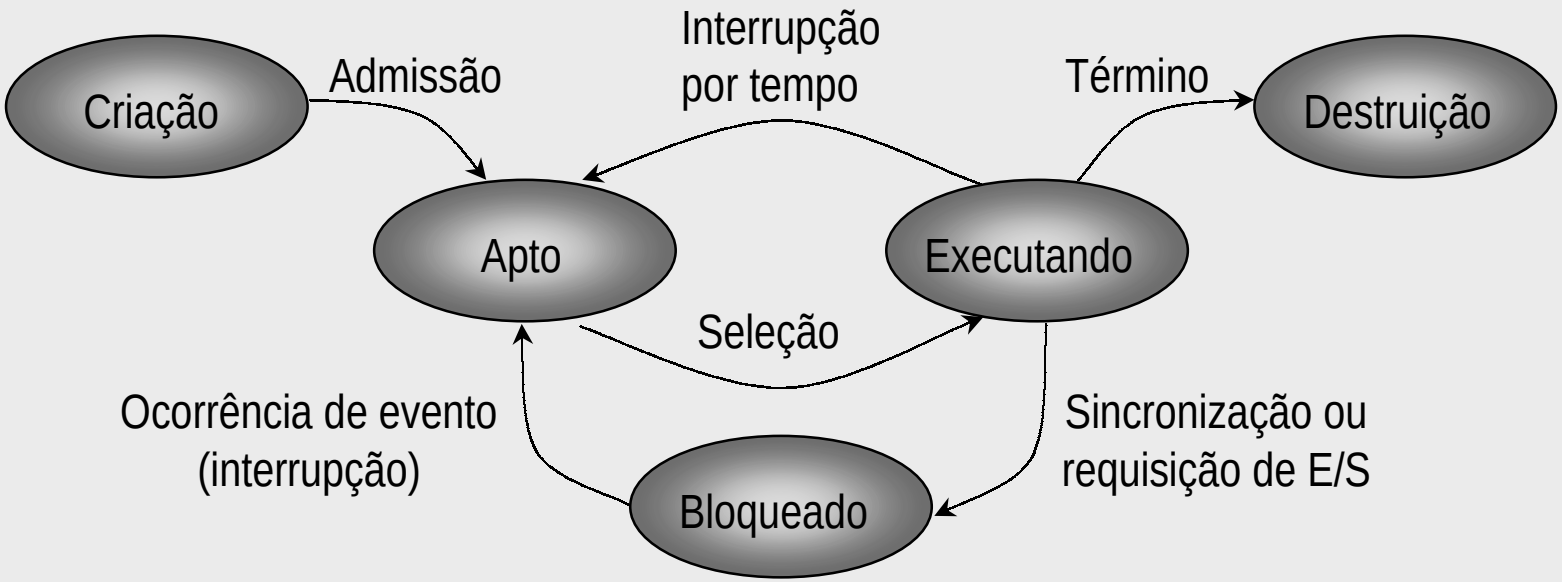
```
struct desc_proc{
        char     estado_atual;
        int      prioridade;
        unsigned inicio_memoria;
        unsigned tamanho_mem;
        struct   arquivos arquivos_abertos[20];
        unsigned tempo_cpu;
        unsigned proc_pc;
        unsigned proc_sp;
        unsigned proc_acc;
        unsigned proc_rx;
        struct   desc_proc *proximo;
}

struct desc_proc tab_desc[MAX_PROCESS];
```

Exemplo de implementação de processos (2)

- Estruturas de filas e inicialização

```
struct desc_proc *desc_livre;
struct desc_proc *espera_cpu;
struct desc_proc *usando_cpu;
struct desc_proc *bloqueados;

/* Inicialização das estruturas de controle */

for (i=0; i < MAX_PROCESS; i++)
        tab_desc[i].prox = &tab_desc[i+1];

tab_desc[i].prox = NULL;
desc_livre = &tab_desc[0];

espera_cpu= NULL;
usando_cpu= NULL;
bloqueado = NULL;
```

Eventos de transição de estados



Criação de processo: tarefas típicas

- Alocar um descritor de processo
- Obter um identificador único para o processo (*pid*)
- Inicializar o descritor de processo
- Inserir apontadores para este descritor em estruturas de controle do sistema operacional
  - e.g.; inserir na lista de aptos
- Criar ou expandir estruturas de dados do sistema operacional para este processo
  - e.g.; contabilização

Seleção de processo para execução

- Segundo um critério, selecionar um processo da lista de aptos para utilizar a CPU (execução)
- Em que momento realizar esta seleção ?
  - Sempre que a CPU estiver livre e houver processos aptos a executar
- Realizado pelo procedimento de escalonamento + *dispatcher*
  - Chaveamento de contexto

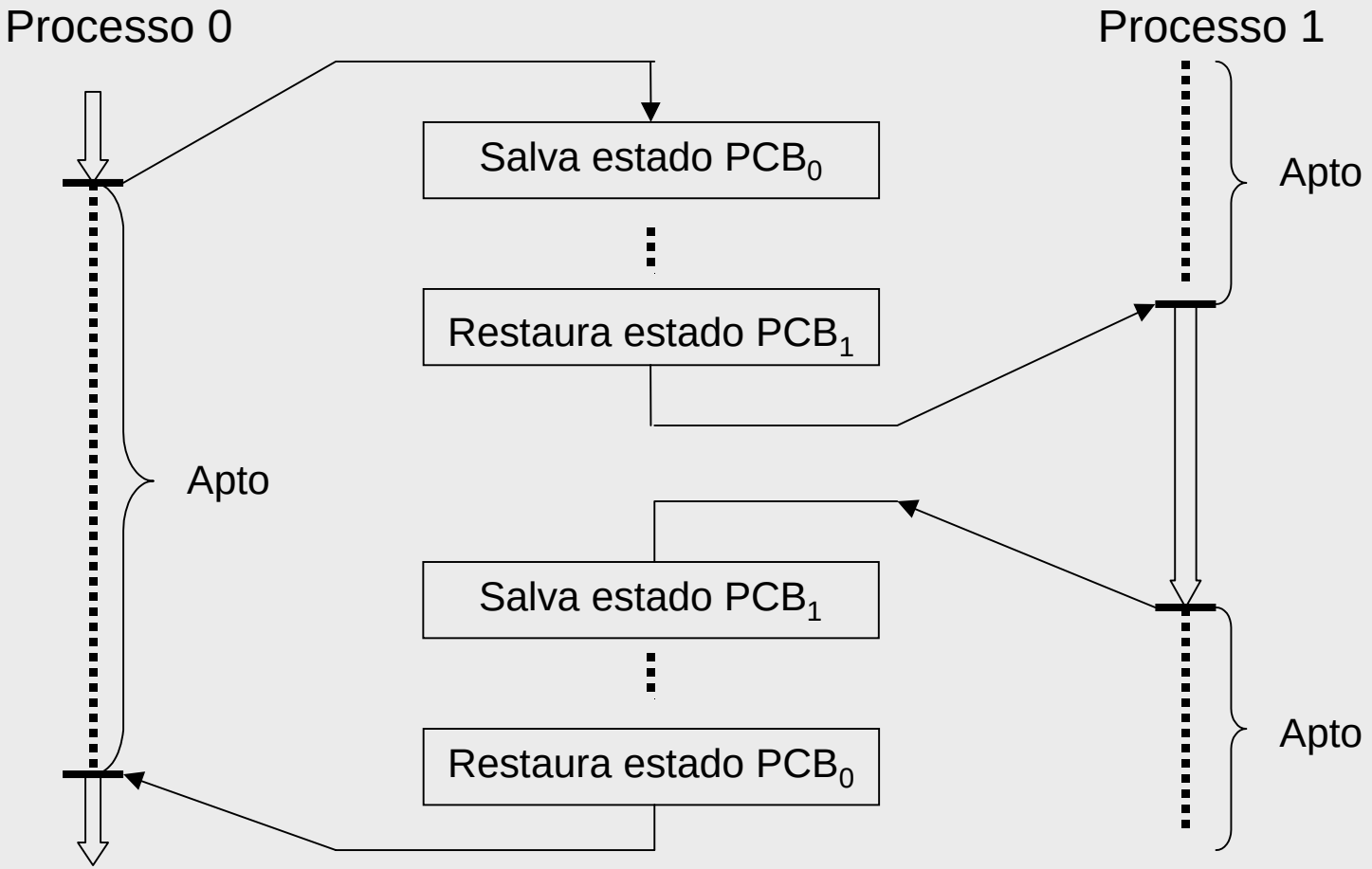
Quando retirar um processo do processador ?

- Término
- Processo de mais alta prioridade ficar apto a executar
- Interrupção de tempo
  - O processo executou por um período de tempo máximo permitido
- Interrupção de dispositivos de E/S
- Interrupção por falta de página (segmento)
  - Endereço acessado não está carregado na memória
  - Memória virtual
- Interrupção por erros

Chaveamento de processos (*dispatcher*)

- Salvar o contexto do processo
- Atualizar o estado do processo que está em execução no seu descritor do processo
- Inserir o descritor do processo na fila apropriada (apto, bloqueado)
- Selecionar outro processo para execução
- Atualizar descritor de processos do processo selecionado
- Atualizar estruturas de dados associadas com gerência de memória e de dispositivos de entrada e saída
- Restaurar contexto do processo selecionado

Chaveamento de contexto



O modelo de processo (1)

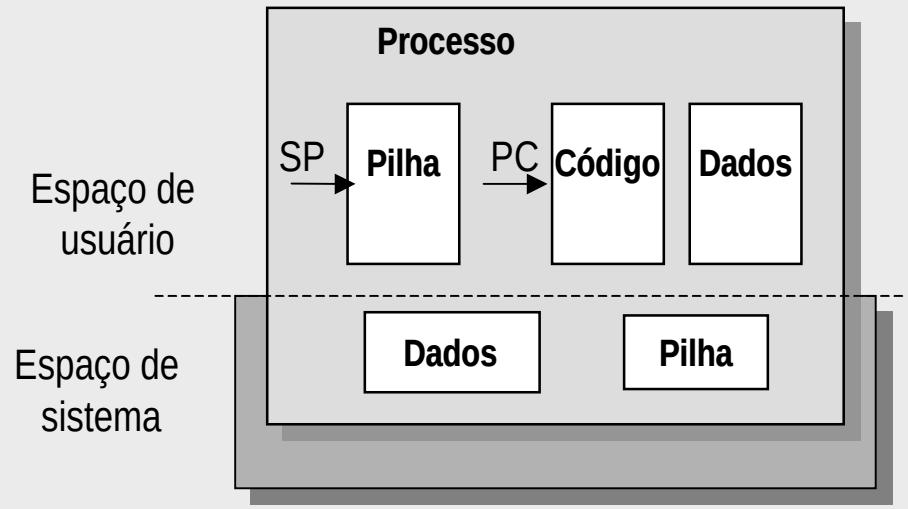
- Conceito dos anos 60
  - Execução seqüêncial de um fluxo de instruções
- Sistema concorrente
  - Conjunto de processos executando em paralelo e cooperando através do compartilhamento de memória
- Estado do sistema
  - Qualquer combinação de estados dos processos
  - Entrelaçamento não determinístico
  - Necessidade de sincronização entre tarefas
    - Semáforos, mutex, variáveis de condição
- Processo é representado por um contexto

O modelo de processo (2)

- Alocação de recursos do sistema para representar um processo:
  - e.g.; Tabelas internas, área de memória, etc
- Alocação de recursos do sistema para execução (processador):
  - Procedimento de escalonamento
  - Comutação de contexto
- Estas duas características são independentes dentro do sistema operacional

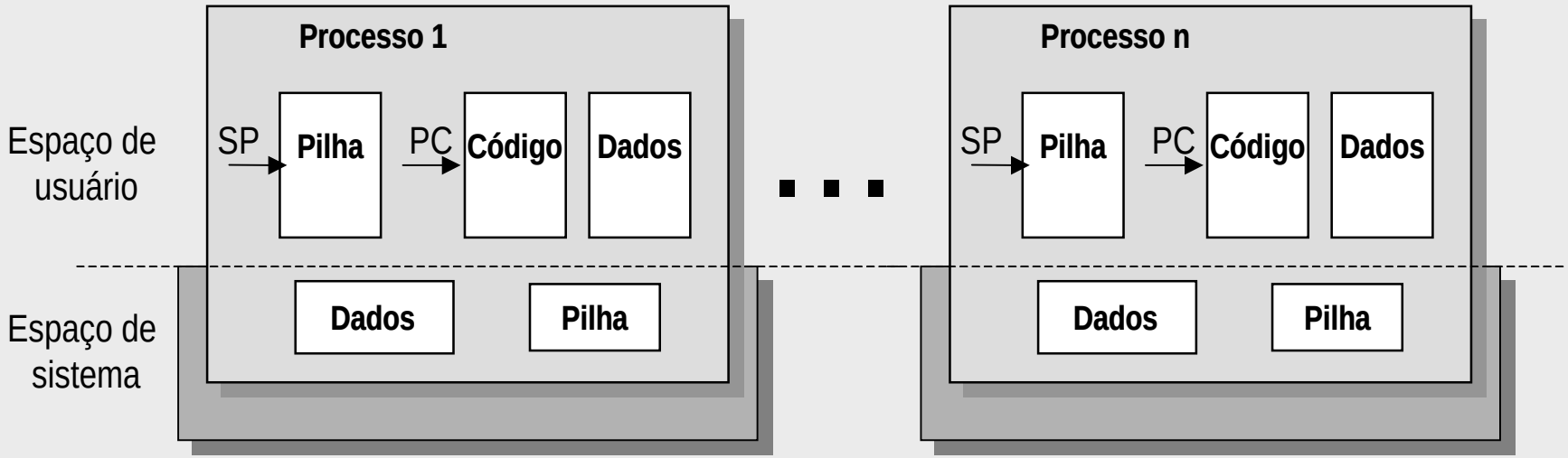
O modelo de processo (3)

- Espaço de endereçamento para armazenamento da imagem do processo
- Informações de recursos alocados são mantidos no descritor de processos
- Contexto de execução (pilha, programa, dados, etc...)



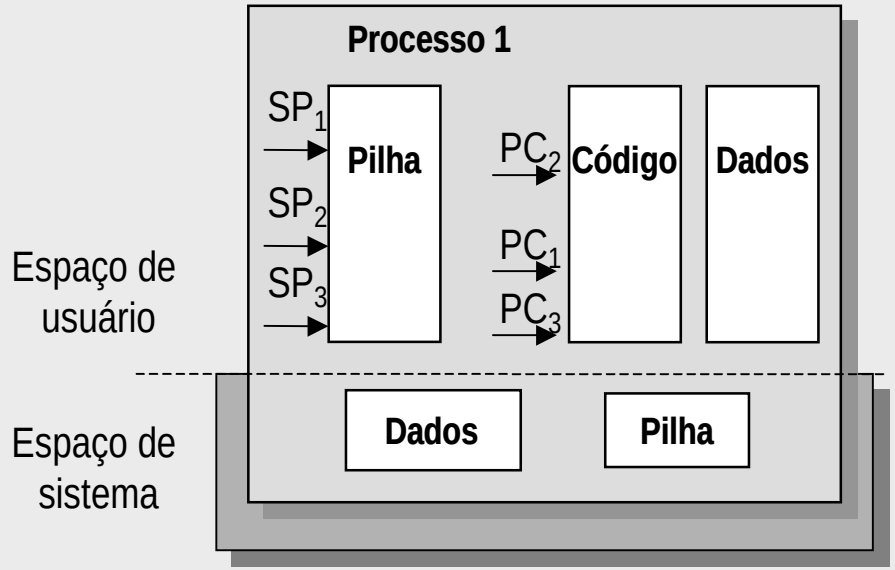
Vários processos

- Um fluxo de controle por processo (*thread*)
- Troca de processo implica em atualizar estruturas de dados internas do sistema operacional
  - e.g.; contexto, espaço de endereçamento, etc...

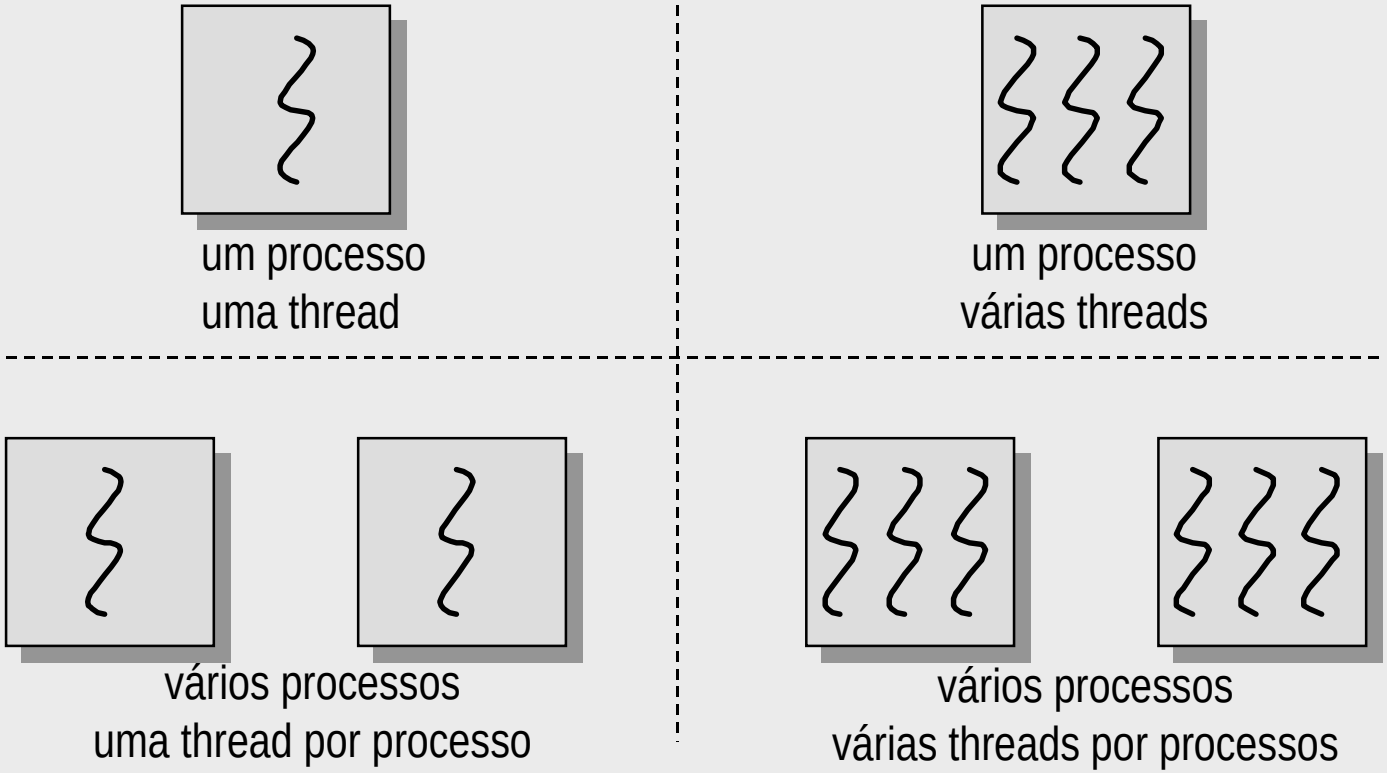


Vários fluxos em um único processo

- Um fluxo de instrução é implementado através do contador de programa (PC) e de um pilha (SP)
- Estruturas comuns compartilhadas
  - Código
  - Dados
  - Descritor de processo
- Conceito de *thread*



Threads e processos



Multiprogramação pesada

- Custos de gerenciamento do modelo de processos
  - Criação do processo
  - Troca de contextos
  - Esquemas de proteção, memória virtual, etc
- Custos são fator Limitante na interação de processos
  - Unidade de manipulação é o processo (arquivo)
  - Mecanismos de IPC (*Inter Process Communications*) necessitam a manipulação de estruturas complexas que representam o processo e sua propriedades
- Solução
  - “Aliviar” os custos, ou seja, reduzir o “peso” das estruturas envolvidas

Multiprogramação leve

- Fornecido pela abstração de um fluxo de execução (*thread*)
  - Basicamente o conceito de processo
- Unidade de interação passa a ser função
- Contexto de uma *thread*
  - Registradores (Pilha, apontador de programa, registradores de uso geral)
- Comunicação por memória compartilhada

Porque utilizar threads ?

- Permitir a exploração do paralelismo real oferecido por máquinas multiprocessadores (modelo M:N ou 1:1)
- Aumentar número de atividades executadas por unidade de tempo (*throughput*)
- Aumentar tempo de resposta
  - Possibilidade de associar threads a dispositivos de entrada/saída
- Sobrepor operações de cálculo com operações de entrada e saída

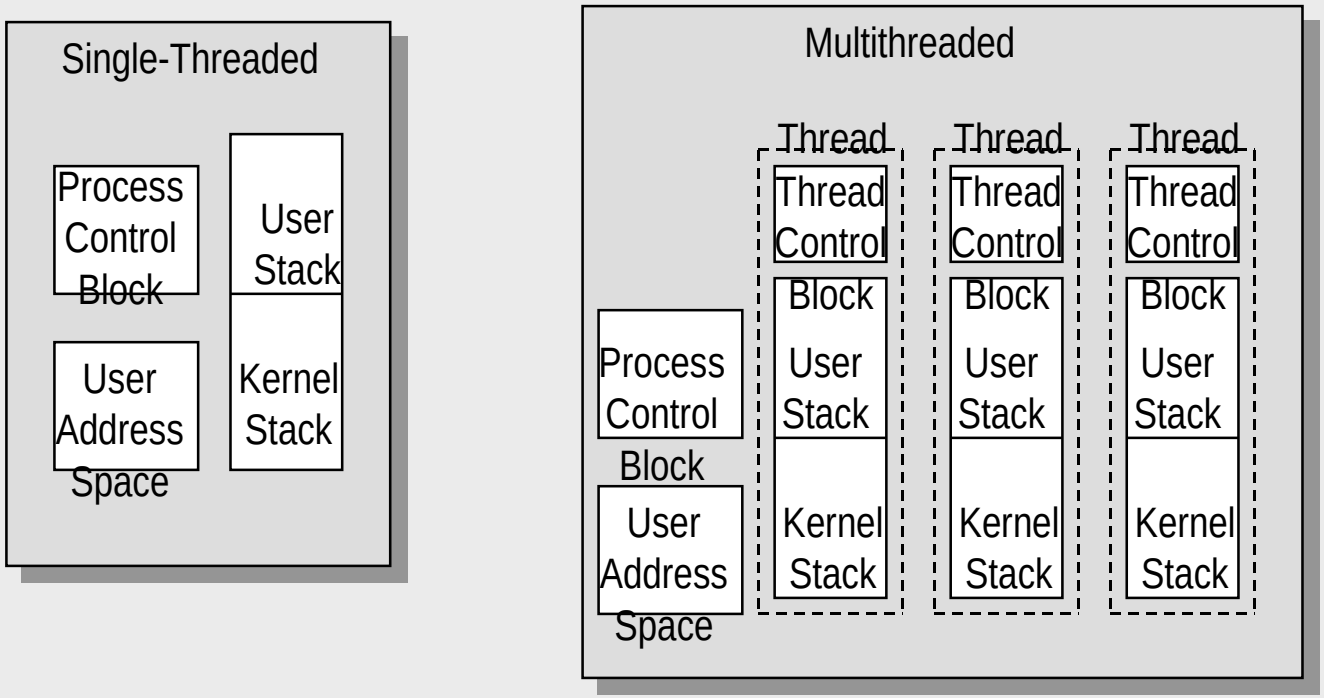
Vantagens de *multithreading*

- Tempo de criação/destruição de *threads* é inferior que tempo de criação/destruição de um processo
- Chaveamento de contexto entre *threads* é mais rápido que tempo de chaveamento entre processos
- Como *threads* compartilham o descritor do processo que as porta, elas dividem o mesmo espaço de endereçamento o que permite a comunicação por memória compartilhada sem interação com o núcleo

Implementação de *threads*

- *Threads* são implementadas através de estruturas de dados similares ao descritor de processo
  - Descritor de *threads*
  - Menos complexa (leve)
- Podem ser implementadas em dois níveis diferentes:
  - Espaço de usuário
  - Espaço de sistema

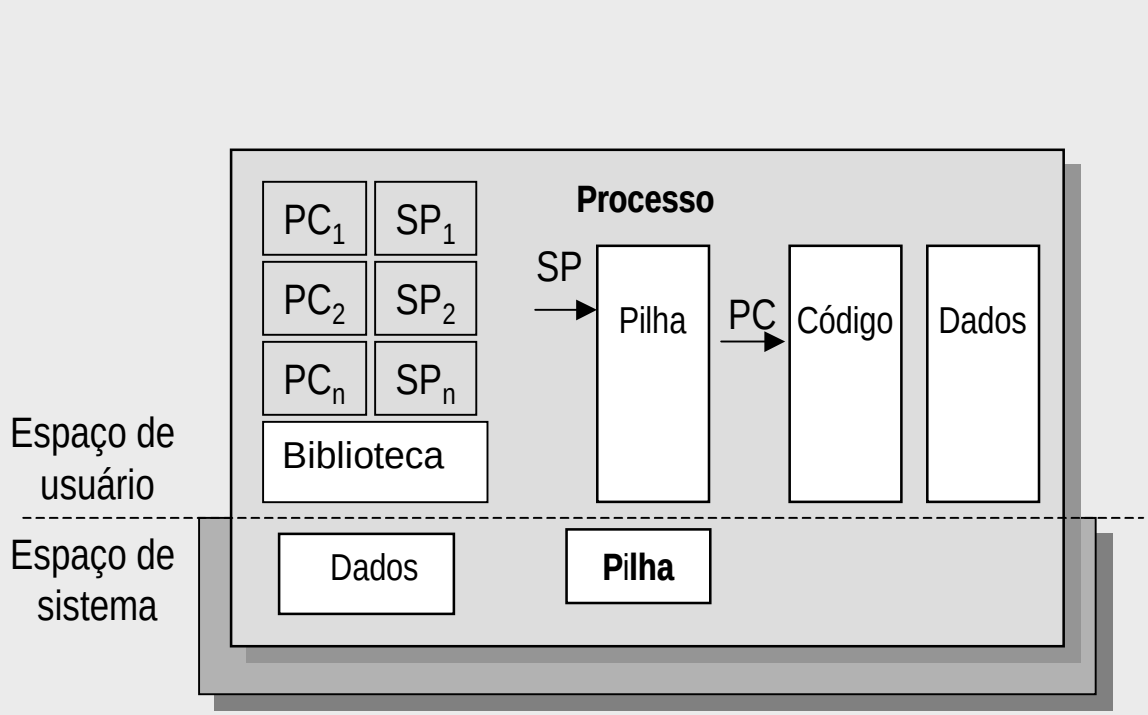
Modelos de processos *single Threaded* e *multithreaded*

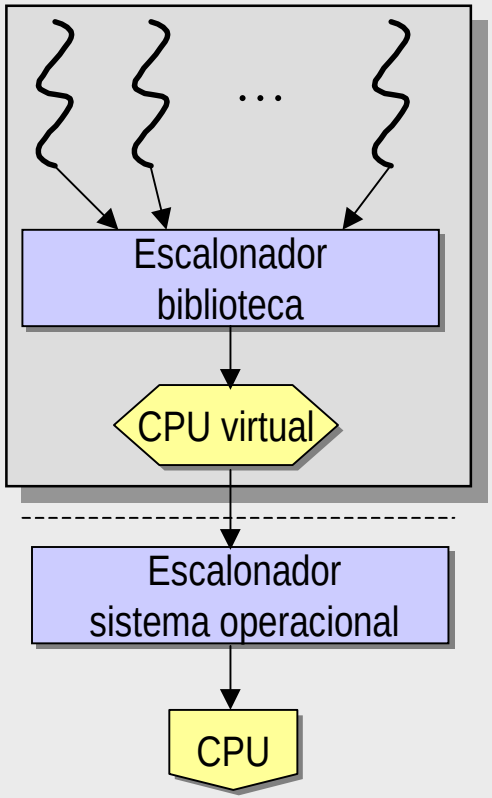


Modelo N:1

- *Threads* a nível de usuário
  - *User level threads* ou ainda *process scope*
- Todas as tarefas de gerenciamento de *threads* é feito a nível da aplicação
  - *Threads* são implementadas por uma biblioteca que é ligada ao programa
  - Interface de programação (API) para funções relacionadas com *threads*
    - e.g; criação, sincronismo, término, etc
- O sistema operacional não “enxerga” a presença das *threads*
- A troca de contexto entre *threads* é feita em modo usuário pelo escalonador embutido na biblioteca
  - Não necessita privilégios especiais
  - Escalonamento depende da implementação

Implementação modelo N:1





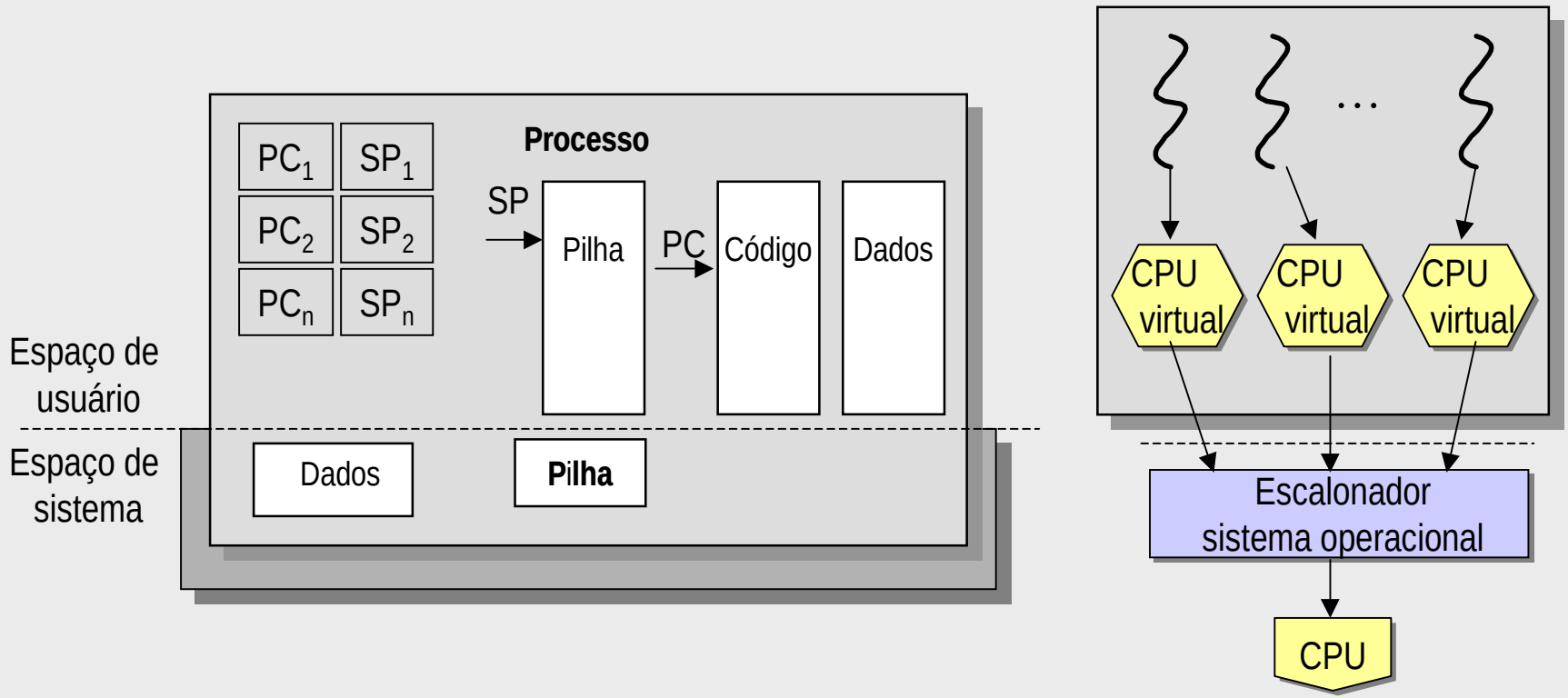
Vantagens e desvantagens

- Vantagens:
  - Sistema operacional divide o tempo do processador entre os processos «pesados» e, a biblioteca de *threads* divide o tempo do processo entre as *threads*
  - Leve: sem interação/intervenção do sistema operacional
- Desvantagens:
  - Uma thread que realiza uma chamada de sistema bloqueante leve ao bloqueio de todo o processo
    - e.g.; operaçoes de entrada/saída
  - Não explora paralelismo em máquinas multiprocessadoras

Modelo 1:1

- *Threads* a nível do sistema
  - *kernel level threads* ou ainda *system scope*
- Resolver desvantagens do modelo N:1
- O sistema operacional “enxerga” as *threads*
  - Sistema operacional mantém informações sobre processos e sobre threads
  - Troca de contexto necessita a intervenção do sistema operacional
- O conceito de threads é considerado na implementação do sistema operacional

Implementação modelo 1:1



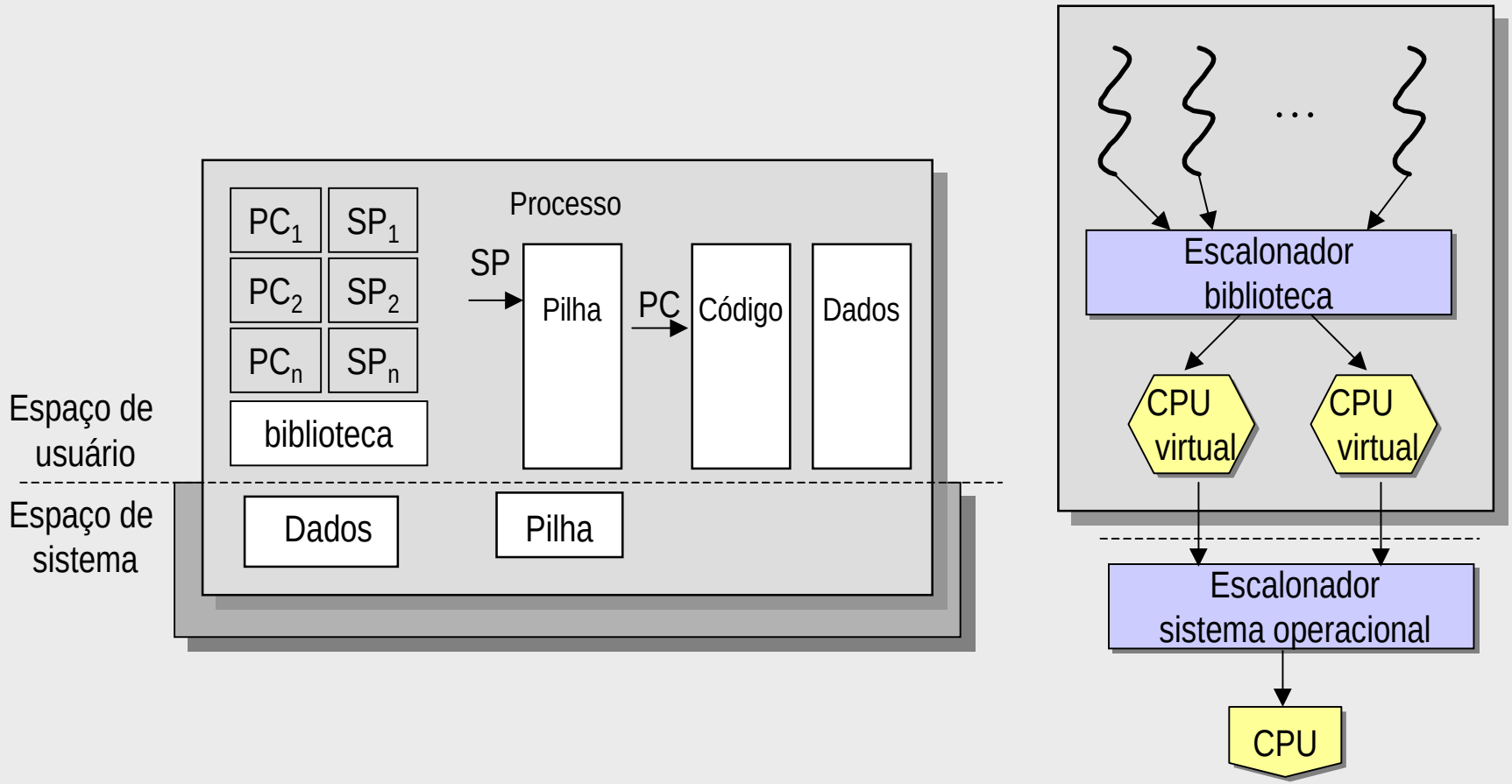
Vantagens e desvantagens

- Vantagens:
  - Explora o paralelismo de máquinas multiprocessadoras (SMP)
  - Facilita o recobrimento de operações de entrada/saída por cálculos
- Desvantagens:
  - Implementação “mais pesada” que o modelo N:1

Modelo M:N

- Abordagem que combina os modelos N:1 e 1:1
- Oferece dois níveis de escalonamento
  - Nível usuário: *threads* sobre unidade de escalonamento
  - Nível sistema: unidades de escalonamento sobre processador
- Dificuldade é parametrizar M e N

Implementação modelo M:N



Escalonamento

- O escalonador é a entidade do sistema operacional responsável por selecionar um processo apto a executar no processador
- Dividir o tempo do processador de forma justa entre os processos apto a executar
- Típico de sistemas multiprogramados: *batch, time-sharing*, multiprogramado ou tempo real
  - Requisitos e restrições diferentes em relação a utilização da CPU
- Duas partes:
  - Escalonador: política de seleção
  - *Dispatcher:* efetua a troca de contexto

Objetivos do escalonamento

- Maximizar a utilização do processador
- Maximizar a produção do sistema (*throughput*)
  - Número de processos executados por unidade de tempo
- Minimizar o tempo de execução (*turnaround*)
  - Tempo total para executar um determinado processo
- Minimizar o tempo de espera
  - Tempo que um processo permance na lista de aptos
- Minimizar o tempo de resposta
  - Tempo decorrido entre uma requisição e a sua realização

Níveis de escalonamento

- Longo prazo
- Médio prazo
- Curto prazo

Escalonador longo prazo

- Executado quando um novo processo é criado
- Determina quando um processo novo passa a ser considerado no sistema, isto é, quando após sua criação ele passa a ser apto
- Controla o grau de multiprogramação do sistema
  - Quanto maior o número de processos ativos, menor a porcentagem de tempo de uso do processador por processo

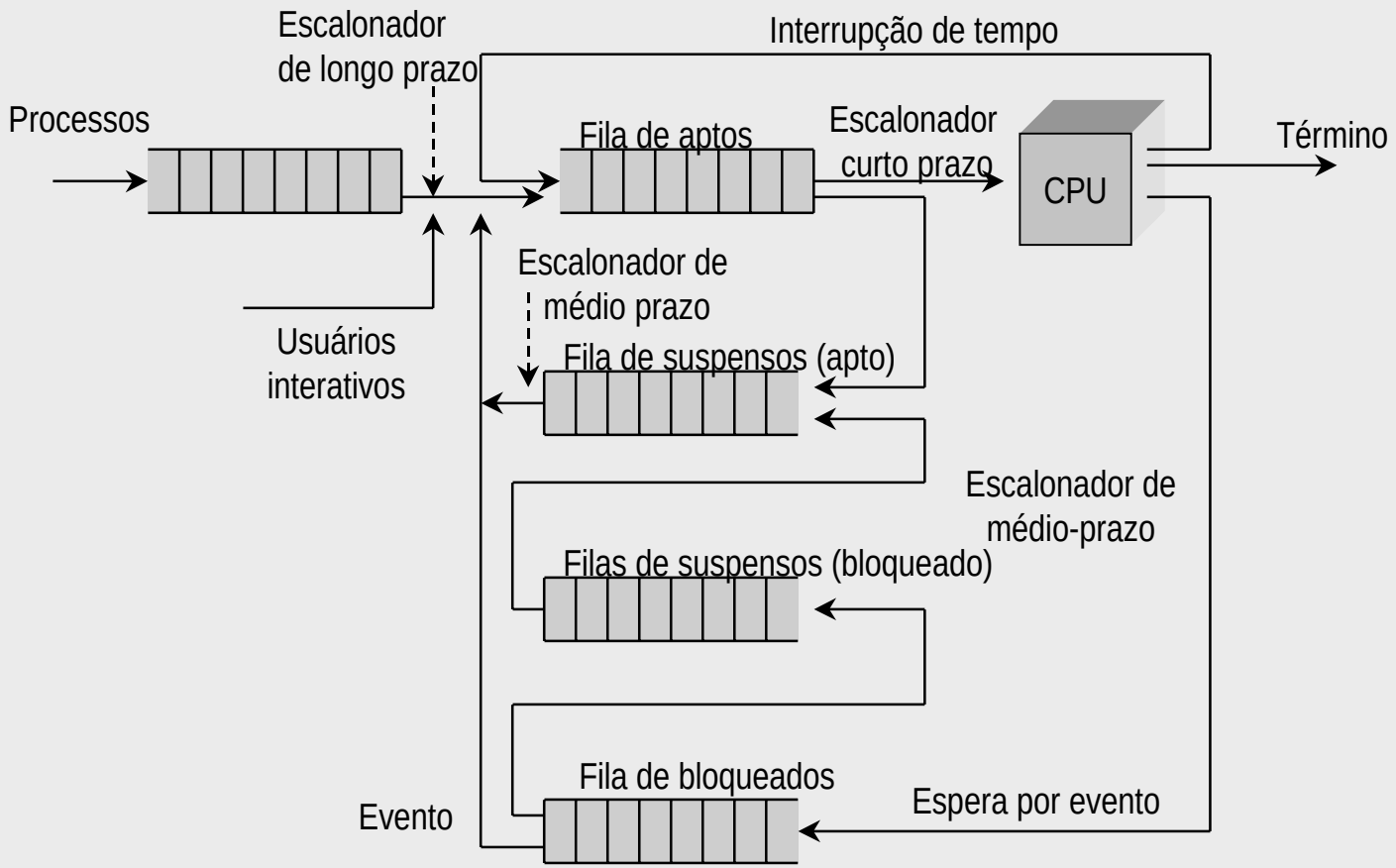
Escalonador médio prazo

- Associado a gerência de memória
  - Participa do mecanismo de *swapping*
- Suporte adicional a multiprogramação
  - Grau de multiprogramação

Escalonador de curto prazo

- Mais importante
- Determina qual processo apto deverá utilizar o processador
- Executado sempre que ocorre eventos importantes:
  - Interrupção de relógio
  - Interrupção de entrada/saída
  - Chamadas de sistemas
  - Sinais (interrupção software)

Diagrama de escalonamento



Tipos de escalonador

- Um vez escalonado, o processo utiliza o processador até que:
  - Não preemptivo:
    - Término de execução do processo
    - Execução de uma requisição de entrada/saída ou sincronização
    - Liberação voluntária do processador a outro processo (yield)
  - Preemptivo:
    - Término de execução do processo
    - Execução de uma requisição de entrada/saída ou sincronização
    - Liberação voluntária do processador a outro processo (yield)
    - Interrupção de relógio
    - Processo de mais alta prioridade esteja pronto para executar

Algoritmos de escalonamento (1)

- Algoritmo de escalonamento seleciona qual processo deve executar em um determinado instante de tempo
- Existem vários algoritmos para atingir os objetivos do escalonamento
- Os algoritmos buscam:
  - Obter bons tempos médios invés de maximizar ou minimizar um determinado critério
  - Privilegiar a variância em relação a tempos médios

Algoritmos de escalonamento (2)

- Algoritmos não preemptivos (cooperativos)
  - FIFO
  - SJF
- Algoritmos preemptivos
  - Round robin (circular)
  - Múltiplas filas
- Existem outros algoritmos de escalonamento
  - High Response Ratio Next (HRRN)
  - Shortest Remaining Time (SRT)
  - Shortest Process Next (SPN)
  - etc...

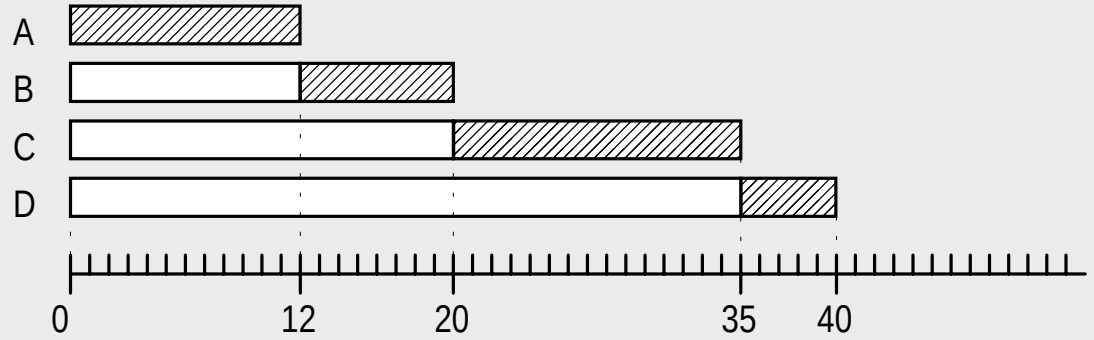
FIFO - *First In First Out* (1)

- *Firts-Come, First-Served* (FCFS)
- Simples de implementar
  - Fila
- Funcionamento:
  - Processos que se tornam aptos são inseridos no final da fila
  - Processo que está no início da fila é o próximo a executar
  - Processo executa até que:
    - Libere explicitamente o processador
    - Realize uma chamada de sistema (bloqueado)
    - Término sua execução

FIFO - *First In First Out* (2)

- Desvantagem:
  - Prejudica processos *I/O bound*
- Tempo médio de espera na fila de execução:
  - Ordem A-B-C-D = (0 + 12 + 20 + 35) / 4 = 16.75 u.t.
  - Ordem D-A-B-C = (0 + 5 + 17 + 25) / 4 = 11.7 u.t.

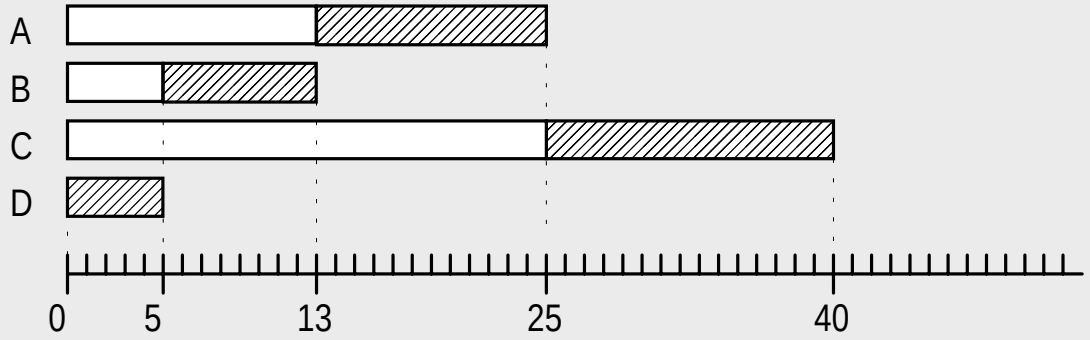
Processo	Tempo
A	12
B	8
C	15
D	5



SJF - *Shortest Job First* (1)

- Originário do fato que o menor tempo de médio é obtido quando se executa primeiro os processos de menor ciclo de processador (*I/O bound*)

Processo	Tempo
A	12
B	8
C	15
D	5



Tempo médio: (0 + 5 + 13 + 25)/4 = 10.75 u.t

SJF - *Shortest Job First* (2)

- Algoritmo ótimo, isto é, fornece o menor tempo médio de espera para um conjunto de processos
- Processos *I/O bound* são favorecidos
- Dificuldade é determinar o tempo do próximo ciclo de CPU de cada processo, porém:
  - Pode ser empregado em processos *batch* (*long term scheduler*)
  - Prever o futuro com base no passado

Prevendo o futuro... (1)

- Pode ser feito utilizando os tempos de ciclos já passados e realizando uma média exponencial

  1. t_n = tempo do n^{esimo} ciclo de CPU
  2. τ_{n+1} = valor previsto para o próximo ciclo de CPU
  3. τ = determina o valor do passado
  4. $\alpha, 0 \leq \alpha \leq 1$
  5. Define - se : $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$.

- Fator α tem o efeito de considerar, de forma ponderada, os ciclos anteriores de processador

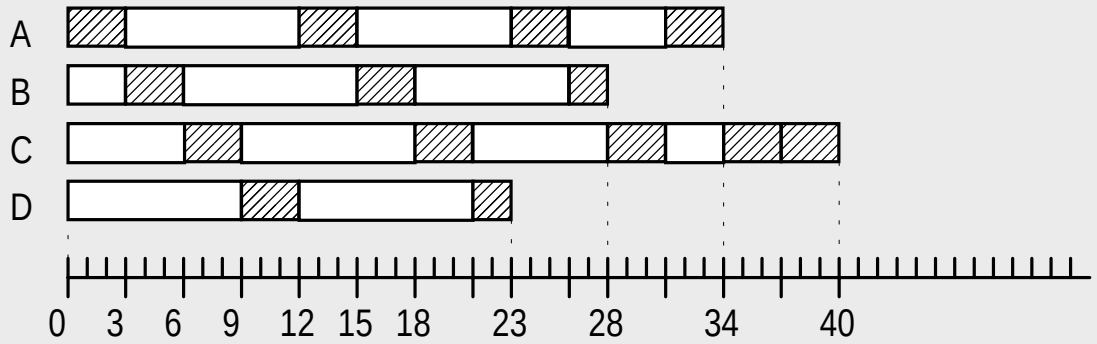
Prevendo o futuro... (2)

- Não considera o último ciclo de processador, só o passado (α =0)
  - $\tau_{n+1} = \tau_n$
- Considera apenas o último ciclo de processador (α = 1)
  - $\tau_{n+1} = t_n$
- Tipicamente se emprega α =0.5
  - Tem o efeito de considerar o mesmo peso para a história atual e a história passada

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\,\alpha\, t_{n-1} + \ldots + (1-\alpha)^j\,\alpha\, t_{n-j} + \ldots + (1-\alpha)^{n+1}\,\tau_0$$

RR - *Round Robin* (1)

- Similar ao algoritmo FIFO
- Cada processo recebe um tempo limite máximo (*time-slice*, *quantum*) para executar um ciclo de processador
- Fila de processos apto é uma fila circular
- Necessidade de um relógio para delimitar as fatias de tempo



RR - *Round Robin* (2)

- Processo perde o processador quando:
  - Libera explicitamente o processador (*yield*)
  - Realize uma chamada de sistema (bloqueado)
  - Termina sua execução
  - Quando sua fatia de tempo é esgotada
- Escalonamento do tipo *round-robin* é preemptivo
- Se *quantum* $\rightarrow \infty$ obtém-se o comportamento de um escalonador FIFO

RR - *Round Robin* (3)

- Problema 1: Dimensionar o *quantum*
  - Compromisso entre *overhead* e tempo de resposta em função do número de usuários (*1/k* na presença de *k* usuários)
  - Compromisso entre tempo de chaveamento e tempo do ciclo de processador (*quantum*)
- Problema 2: Processos *I/O bound* são prejudicados
  - Esperam da mesma forma que processos *CPU bound* porém muito provavelmente não utilizam todo o seu *quantum*
  - Solução:
    - Prioridades

Escalonadores por prioridades

- Associar prioridades a processos *I/O bound* para compensar o tempo gasto em estado de espera (apto)
- Sempre que um processo de maior prioridade que o processo atualmente em execução entrar no estado apto deve ocorrer uma preempção
  - Escalonamento com prioridades é inerente a preempção
  - É possível haver prioridade não-preemptiva
- Escalonador deve sempre selecionar o processo de mais alta prioridade

Prioridade estática versus dinâmica

- Prioridade estática:
  - Um processo é criado com uma determinada prioridade e esta prioridade é mantida durante todo o tempo de vida do processo
- Prioridade dinâmica:
  - Prioridade do processo é ajustada de acordo com o estado de execução do processo e/ou do sistema
    - e.g; ajustar a prioridade em função da fração do *quantum* que foi realmente utilizada pelo processo:

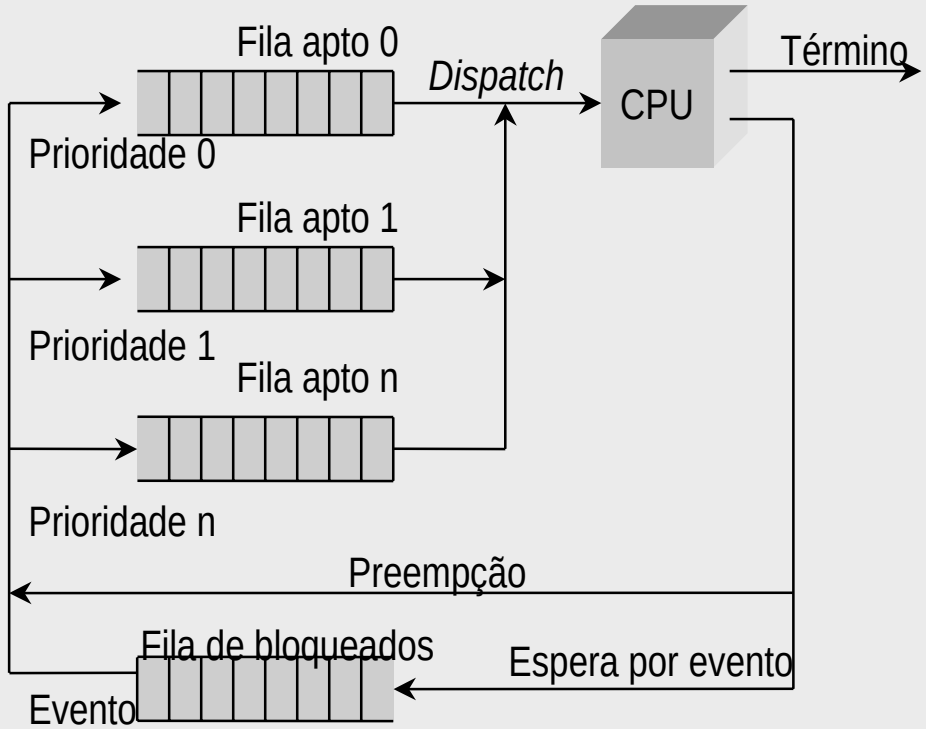
      q = 100 ms

      Processo A utilizou 2ms $\mapsto$ nova prioridade = 1/0.02 = 50

      Processo B utilizou 50ms $\mapsto$ nova prioridade = 1/0.5 = 2

Implementação de escalonador com prioridades

- Múltiplas filas associadas ao estado apto
- Cada fila uma prioridade
  - Pode ter sua própria política de escalonamento (FIFO, SJF, RR)



Exemplo: *pthreads*

- A política de escalonamento FIFO com prioridade considera:
  - Quando um processo em execução é preemptado ele é inserido no ínicio de sua fila de prioridade
  - Quando um processo bloqueado passa a apto ele é inserido no final da fila de sua prioridade
  - Quando um processo troca de prioridade ele é inserido no final da fila de sua nova prioridade
  - Quando um processo em execução “passa a vez” para um outro processo ele é inserido no final da fila de sua prioridade

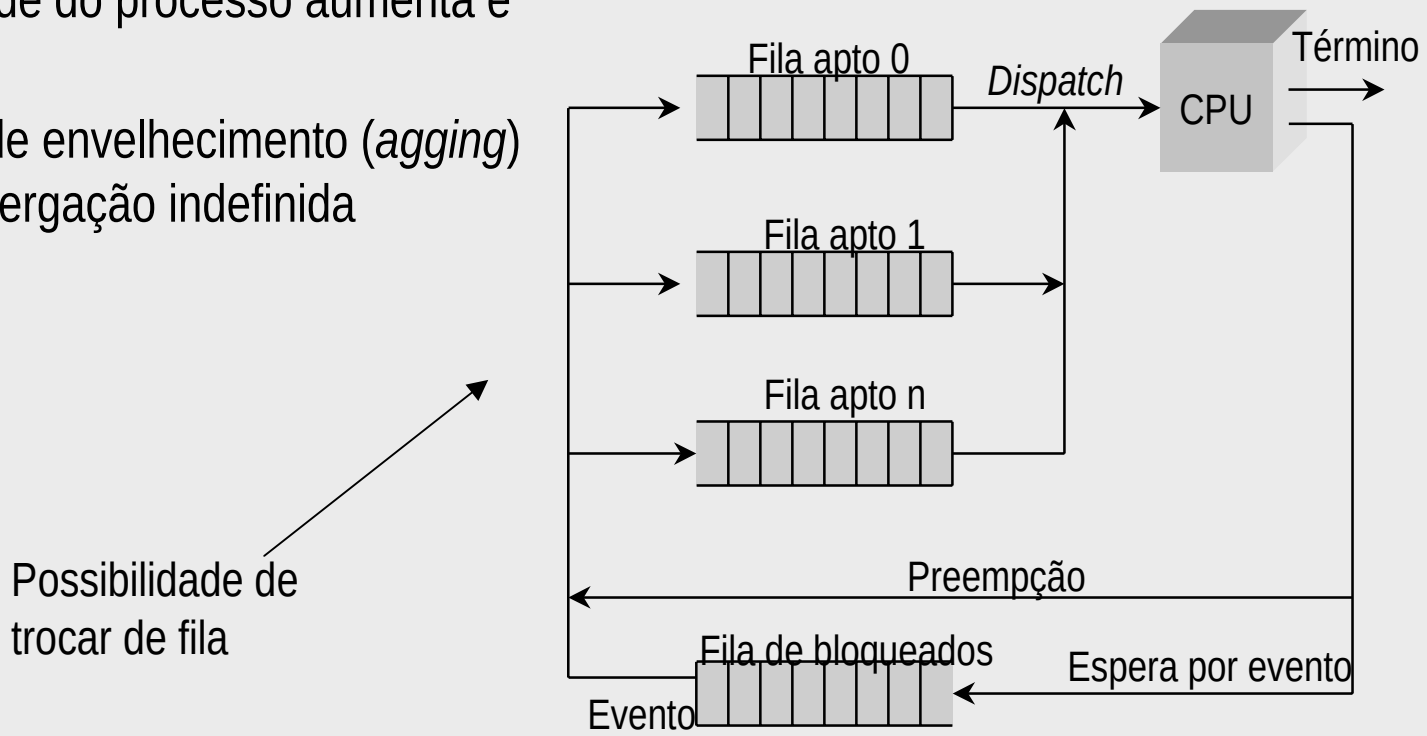
Problemas com prioridades

- Com prioridades um processo de baixa prioridade pode não executar (*starvation*)
- Um processo que durante sua execução troca de comportamento (*CPU bound* a *I/O bound*) pode ficar mal classificado a nível de prioridades e ser penalizado

- Solução:
  - Múltiplas filas com realimentação

Múltiplas filas com realimentação

- Baseado em prioridades dinâmicas
- Em função do tempo de uso da CPU a prioridade do processo aumenta e diminui
- Sistema de envelhecimento (*agging*) evita postergação indefinida



Exemplo: escalonamento em UNIX (1)

- Múltiplas filas com realimentação empregando round robin em cada uma das filas
- Prioridades são re-avaliadas uma vez por segundo em função de:
  - Prioridade atual
  - Prioridade do usuário
  - Tempo recente de uso da CPU
  - Fator *nice*
- Prioridades são divididas em faixas de acordo com o tipo do usuário
- A troca dinâmica das prioridades respeita os limites da faixa

Exemplo: escalonamento UNIX (2)

- Prioridades recebem valores entre 0 e 127 (menor o valor númerico, maior a prioridade)
  - 0-49: processos do kernel
  - 50-127: processo de usuário
- Ordem decrescente de prioridades
  - *Swapper*
  - Controle de dispositivos de entrada e saída orientados a bloco
  - Manipulação de arquivos
  - Controle de dispositivos de entrada e saída orientados a caractere
  - Processos de usuário

Escalonamento tempo real

- *Hard real-time* systems - necessitam completar uma tarefa crítica dentro de uma determinada quantidade de tempo
- *Soft real-time* computing – necessita que um processo crítico receba uma prioridade maior que outros processos

Leituras complementares

- R. Oliveira, A. Carissimi, S. Toscani; *Sistemas Operacionais*. Editora Sagra-Luzzato, 2001.
  - Capítulo 4
- A. Silberchatz, P. Galvin, G. Gane; *Applied Operating System Concepts*. (1st edition). Addison-Wesley, 2000.
  - Capítulo 4, 5 e 6
- W. Stallings; *Operating Systems*. (4th edition). Prentice Hall, 2001.
  - Capítulo 9