

Sistemas Operacionais

Gerência do processador



2ª edição

Revisão: Fev/2003

Capítulo 4

Sumário

- ➡ ■ Implementação do conceito de processos e threads
 - Escalonamento
 - Escalonadores não -preemptivos
 - Escalonamento
 - Escalonamento preemptivos

Introdução

- Multiprogramação pressupõe a existência simultânea de vários processos disputando o processador
- Necessidade de “intermediar” esta disputa de forma justa
 - Gerência do processador
 - Algoritmos de escalonamento
- Necessidade de “representar” um processo
 - Implementação de processos
 - Estruturas de dados

Representação de processo

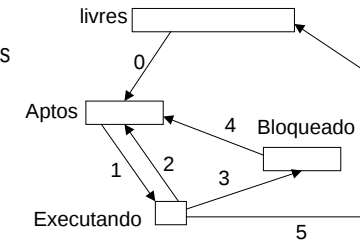
- Processo é um programa em execução
 - Áreas na memória para código, dados e pilha
- Possui uma série de estados (apto, executando, bloqueado, etc) para representar sua evolução no tempo, implica em:
 - Organizar os processos nos diferentes estados
 - Determinar eventos que realizam a transição entre os estados
 - Determinar quando um processo tem direito a “utilizar” o processador
- Necessário manter informações a respeito do processo
 - e.g.: prioridades, localização em memória, estado atual, direitos de acesso, recursos que emprega, etc.

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 em um 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 a própria estrutura de descritores de processos são empregadas 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 bloco descritor 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 bloco descritor 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;
```

Tarefas típicas no PCB durante o ciclo de vida

■ Criação

- Alocação de áreas de memória para código, dados e pilha e de estruturas de dados do sistema operacional
- Inicialização do descritor de processo e inserção em filas do sistema

■ Execução

- Realizam das instruções da área de código
 - Interação com sistema operacional via chamadas de sistema
- Atualização do bloco descritor de processo
 - Retratar estados e recursos que evoluem dinamicamente com a execução
- Suscetível ao acionamento do escalonador/*dispatcher* em resposta a eventos

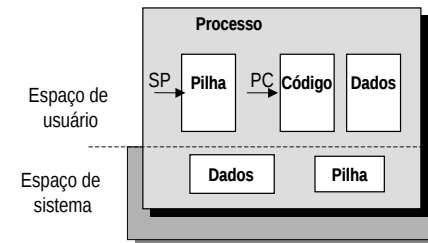
■ Término

- Liberação de recursos e estruturas de dados utilizadas

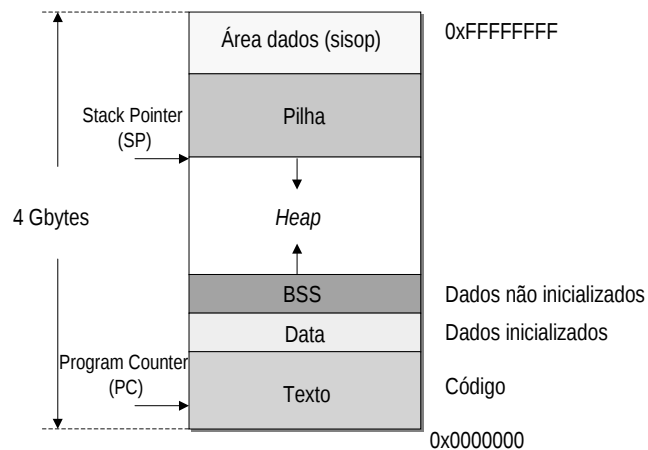
O modelo de processo

■ Processo é representado por:

- Espaço de endereçamento: área p/ armazenamento da imagem do processo
- Estruturas internas do sistema (tabelas internas, áreas de memória, etc)
 - Mantidos no descritor de processos
- Contexto de execução (pilha, programa, dados, etc...)

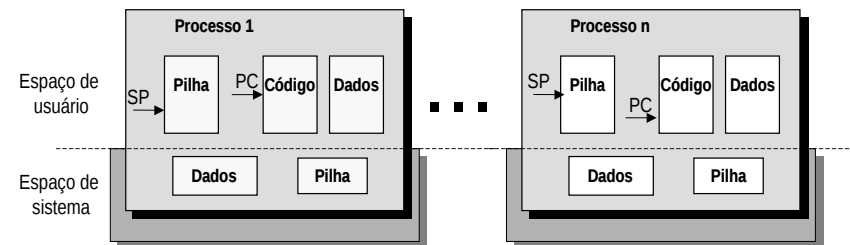


Exemplo: modelo de processo Unix (linux)



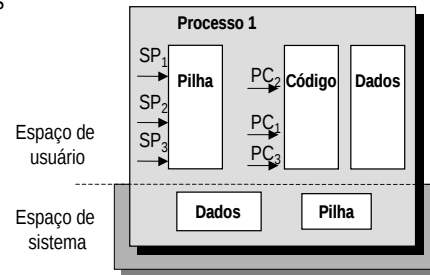
Vários processos

- 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 uma pilha (SP)
- Estruturas comuns compartilhadas
 - Código
 - Dados
 - Descritor de processo
- Conceito de *thread*



13

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 tratamento de estruturas complexas que representam o processo e suas propriedades
- Solução
 - “Aliviar” os custos, ou seja, reduzir o “peso” das estruturas envolvidas

Sistemas Operacionais

14

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 através do compartilhamento direto da área de dados

15

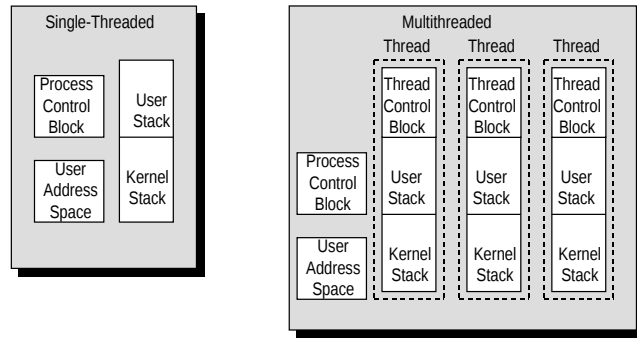
Implementação de *threads*

- *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

Sistemas Operacionais

16

Modelos de processos *single Threaded* e *multithreaded*



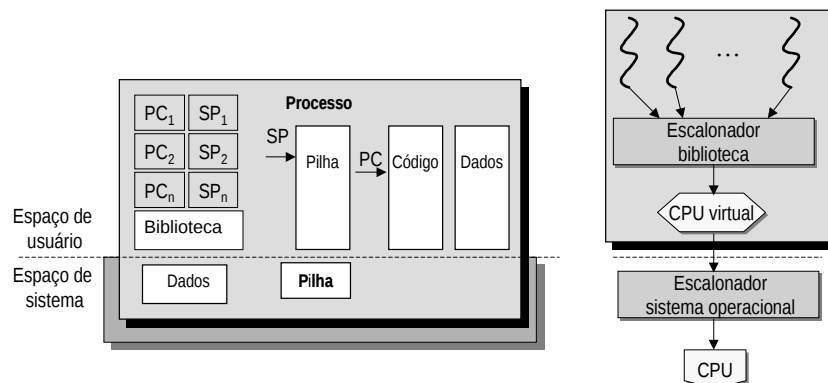
17

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

18

Implementação modelo N:1



19

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.; operações de entrada/saída
 - Não explora paralelismo em máquinas multiprocessadoras

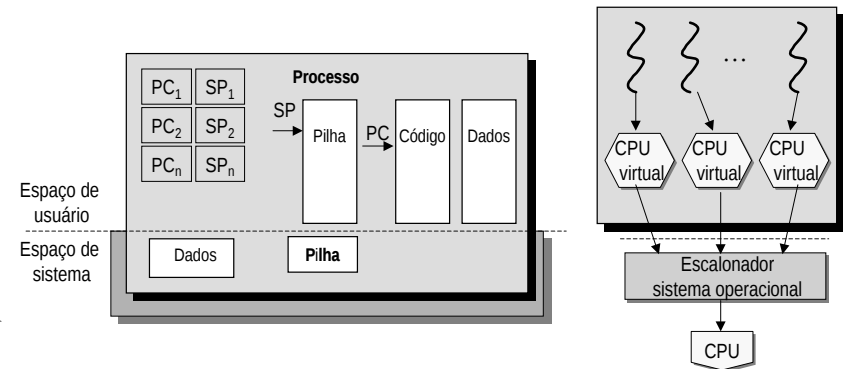
20

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

21

Implementação modelo 1:1



Sistemas Operacionais

22

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

23

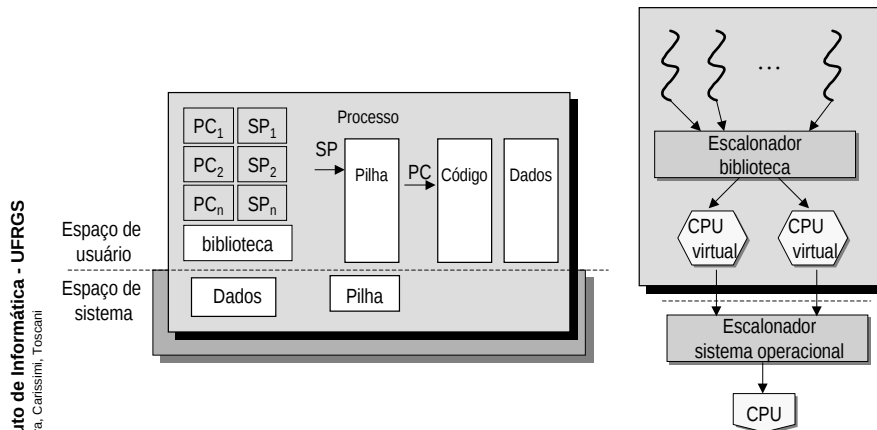
Modelo M:N

- 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

Sistemas Operacionais

24

Implementação modelo M:N



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*)
- Diminuir 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

Leituras complementares

- R. Oliveira, A. Carissimi, S. Toscani *Sistemas Operacionais* Editora Sagra-Luzzato, 2001.
 - Capítulo 4.
- A. Silberchatz, P. Galvin; *Operating System Concepts*. (4th edition). Addison-Wesley, 1994.
 - Capítulo 4

Sumário

- Implementação do conceito de processos e threads
- Escalonamento
 - Escalonadores não -preemptivos
- Escalonamento
 - Escalonamento preemptivos

Escalonamento

- O escalonador é a entidade do sistema operacional responsável por selecionar um processo apto para executar no processador
- O objetivo é dividir o tempo do processador de forma justa entre os processos aptos 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

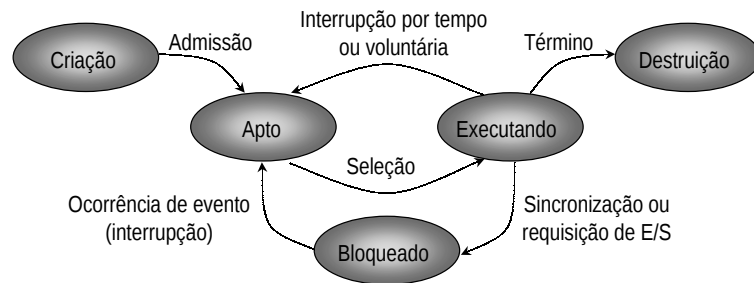
Objetivos do escalonamento

- 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 permanece na lista de aptos
- Minimizar o tempo de resposta
 - Tempo decorrido entre uma requisição e a sua realização

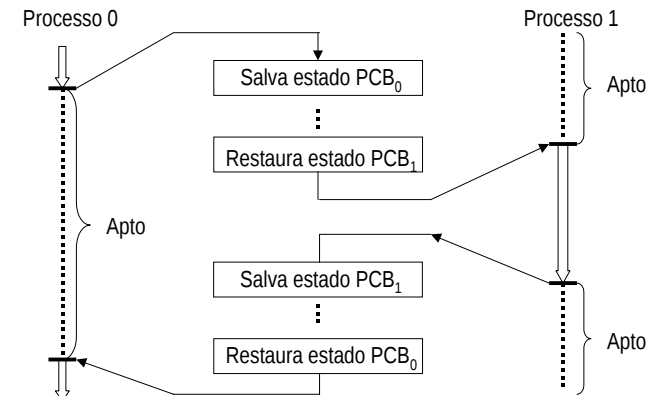
Situações típicas para execução do escalonador

- Dependem se o escalonador é preemptivo ou não, se considera prioridades ou não, etc...
 - Sempre que a CPU estiver livre e houver processos aptos a executar
 - Criação e término de processos
 - Um processo de mais alta prioridade ficar apto a executar
 - Interrupção de tempo
 - Processo executou por um período de tempo máximo permitido
 - Interrupção de dispositivos de entrada e saída
 - Interrupção por falta de página (segmento) em memória
 - Endereço acessado não está carregado na memória (memória virtual)
 - Interrupção por erros

Eventos de transição de estados



Chaveamento de contexto (*dispatcher*)



PCB: *Process Control Block*

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
 - Controle de admissão
- 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

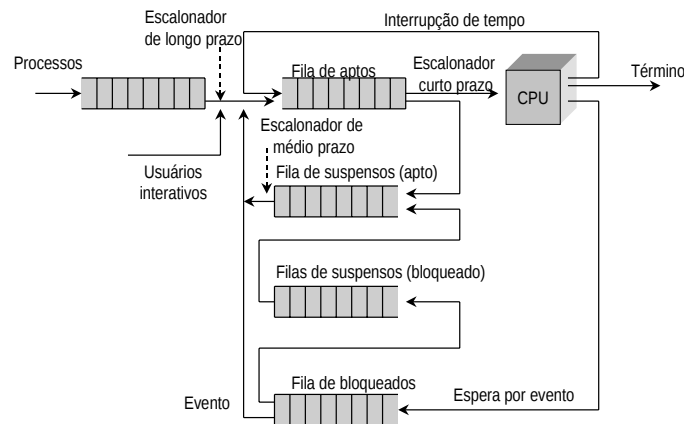
Escalonador médio prazo

- Associado a gerência de memória
 - Participa do mecanismo de *swapping*
- Suporte adicional a multiprogramação
 - Grau de multiprogramação efetiva (diferencia aptos dos aptos-suspensos)

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)
 - *First-In First-Out* (FIFO) ou *First-Come First-Served* (FCFS)
 - *Shortest Job First* (SJF) ou *Shortest Process Next* (SPN)
- Algoritmos preemptivos
 - Round robin (circular)
 - Baseado em prioridades
- Existem outros algoritmos de escalonamento
 - *High Response Ratio Next* (HRRN)
 - *Shortest Remaining Time* (SRT)
 - etc...

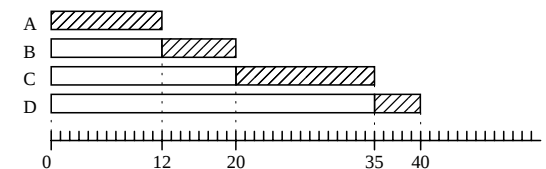
FIFO - *First In First Out* (1)

- *First-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)
 - Termine sua execução

FIFO - *First In First Out* (2)

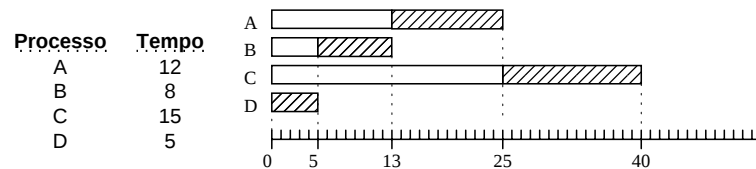
- Desvantagem:
 - Prejudica processos *I/O bound*
- Tempo médio de espera na fila de execução:
 - Ordem A-B-C-D = $(0 + 12 + 20 + 35) / 4 = 16.75$ u.t.
 - Ordem D-A-B-C = $(0 + 5 + 17 + 25) / 4 = 11.7$ u.t.

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*)



Tempo médio: $(0 + 5 + 13 + 25)/4 = 10.75 \text{ u.t}$

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 enésimo ciclo de CPU
 2. τ_{n+1} = valor previsto para o próximo ciclo de CPU
 3. τ = armazena a informação dos ciclos passados ($n-1$)
 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)

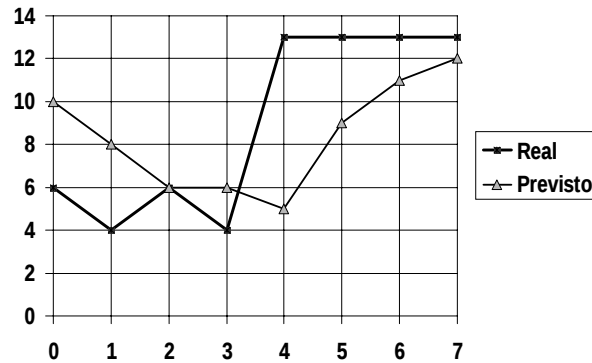
- Não considera o último ciclo de processador, só o passado ($\alpha = 0$)
 - $\tau_{n+1} = \tau_n$
- Considera apenas o último ciclo de processador ($\alpha = 1$)
 - $\tau_{n+1} = t_n$
- Tipicamente se emprega $\alpha = 0.5$
 - Tem o efeito de considerar o mesmo peso para a história atual e a história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

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

Exemplo de “previsão do futuro”

Ciclo de cpu:

Real: 6 4 6 4 13 13 13 Parâmetros: $\alpha=0.5$ $\tau_0=10$
Previsto: 10 8 6 6 5 9 11 12



Leituras complementares

- R. Oliveira, A. Carissimi, S. Toscani; *Sistemas Operacionais*. Editora Sagra-Luzzato, 2001.
 - Capítulo 4
- A. Silberchatz, P. Galvin; *Operating System Concepts*. (4th edition). Addison-Wesley, 1994.
 - Capítulo 4, 5 e 6

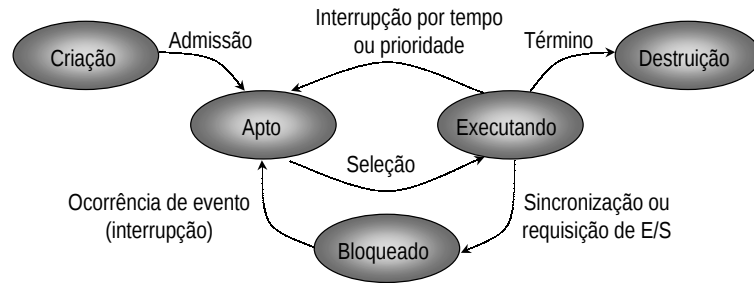
Sumário

- Implementação do conceito de processos e threads
- Escalonamento
 - Escalonadores não -preemptivos
- Escalonamento
 - Escalonamento preemptivos

Tipos de escalonador (lembrando...)

- 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

Eventos de transição de estados

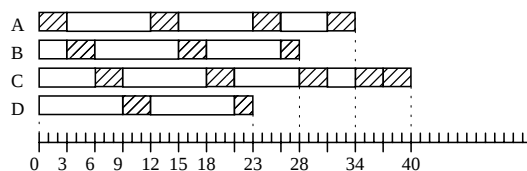


Escalonadores preemptivos

- Por interrupção de tempo
 - Round robin (circular)
- Por prioridades

RR - Round Robin (1)

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



RR - Round Robin (2)

- Por ser preemptivo, um 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
- Se *quantum* $\rightarrow \infty$ obtém-se o comportamento de um escalonador FIFO

Problemas com o *Round Robin*

■ Problema 1: Dimensionamento do *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: Associar prioridades mais altas aos processos *I/O bound* para compensar o tempo gasto no estado de espera (apto)

57

Escalonamento com prioridades

■ Sempre que um processo de maior prioridade que o processo atualmente em execução entrar no estado apto deve ocorrer uma preempção

- A existência de prioridades pressupõem a preempção
- É possível haver prioridade não-preemptiva

■ Escalonador deve sempre selecionar o processo de mais alta prioridade segundo uma política:

- Round-Robin
- FIFO (FCFS)
- SJF (SPN)

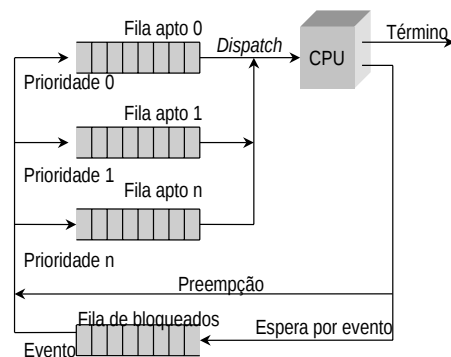
58

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)



59

Exemplo: *pthread*s

■ A política de escalonamento FIFO com prioridade considera:

- Quando um processo em execução é preemptado ele é inserido no início 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

60

Como definir a prioridade de um processo?

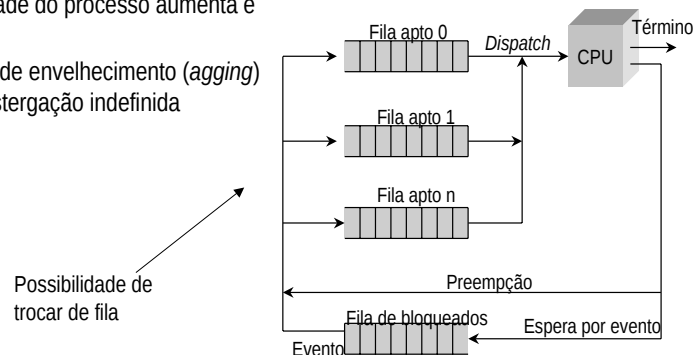
- Prioridade estática:
 - Um processo é criado com uma determinada prioridade e esta prioridade é mantida durante todo o tempo de vida do processo
- Prioridade dinâmica:
 - Prioridade do processo é ajustada de acordo com o estado de execução do processo e/ou do sistema
 - e.g; ajustar a prioridade em função da fração do *quantum* que foi realmente utilizada pelo processo:
 $q = 100 \text{ ms}$
Processo A utilizou 2ms $\mapsto$ nova prioridade = $1/0.02 = 50$
Processo B utilizou 50ms $\mapsto$ nova prioridade = $1/0.5 = 2$

Problemas com prioridades

- Um processo de baixa prioridade pode não ser executado
 - Postergação indefinida (*starvation*)
- Processo com prioridade estática pode ficar mal classificado e ser penalizado ou favorecido em relação aos demais
 - Típico de processos que durante sua execução trocam de padrão de comportamento (*CPU bound* a *I/O bound* e vice-versa)
- 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



Estudo de caso: escalonamento 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

Escalonamento UNIX (1)

- Prioridades recebem valores entre 0 e 127 (menor o valor numérico, 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 UNIX (2)

- Cálculo de prioridade de processos de usuário:
 - Fator nice: valor variando entre 0 (mais prioritário) a 39 (menos prioritário), sendo 20 o valor *default*
 - Uso recente do processador (p_{cpu})

$$decay = \frac{2 \times load_average}{(2 \times load_average + 1)}$$

$$p_{usrpri} = PUSER + \frac{p_{cpu} \times decay}{4} + 2 \times p_{nice}$$

- Onde:
 - *load_average* é o número médio de processos aptos no último segundo
 - PUSER é valor de base de prioridade para usuários (50)

Estudo de caso: escalonamento Linux

- Duas classes em função do tipo de processos (*threads*)
 - Processos interativos e *batch*
 - Processos de tempo real
- Políticas de escalonamento do linux (padrão POSIX)
 - SCHED_FIFO: FIFO com prioridade estática
 - Válido apenas para processos de tempo real
 - SCHED_RR: Round-robin com prioridade estática
 - Válido apenas para processos de tempo real
 - SCHED_OTHER: Filas multinível com prioridades dinâmicas (*time-sharing*)
 - Processos interativos e *batch*

Escalonamento linux (tempo real)

- Linux implementa dois tipos de prioridade
 - Estática: exclusivamente processos de tempo real
 - Dinâmica: processos interativos e batch
- Prioridade é definida pelo usuário e não é modificada pelo escalonador
 - Somente usuários com privilégios especiais
- Seleciona sempre processos de mais prioridade para executar
 - Executa segundo a política selecionada: SCHED_FIFO ou SCHED_RR
- Processo em tempo real tem preferência (prioridade) sobre processos interativos e batch

Escalonamento linux (*timesharing*)

- Baseado no uso de créditos e prioridade
- Sistema de créditos:
 - Cada processo executa um certo número de créditos
 - O processo com maior crédito é o selecionado
 - Cada interrupção de tempo o processo em execução perde um crédito
 - Processo que atinge zero créditos é suspenso (escalonador médio prazo)
 - Se no estado apto não existir processos com créditos é realizado uma redistribuição de créditos para todos os processos (qualquer estado)

$$Créditos = \frac{Créditos}{2} + prioridade$$

Estudo de caso: escalonamento windows 2000

- Unidade de escalonamento é a thread
- Escalonador preemptivo com prioridades
- Prioridades organizadas em duas classes:
 - Tempo real: prioridade estática (níveis 16-31)
 - Variável: prioridade dinâmica (níveis 0-15)
- Cada classe possui 16 níveis de prioridades
 - Cada nível é implementado por uma fila em uma política round-robin
 - Múltiplas filas: classe de tempo real
 - Múltiplas filas com realimentação: classe de tempo variável
 - Threads da classe tempo real tem precedência sobre as da classe variável

Escalonamento windows 2000 (classe variável)

- Dois parâmetros definem a prioridade de uma *thread*:
 - Valor de prioridade de base do processo
 - Prioridade inicial que indica sua prioridade relativa dentro do processo
- Prioridade da *thread* varia de acordo com uso do processador
 - Preemptada por esgotar o *quantum*: prioridade reduzida
 - Preemptada por operação de E/S: prioridade aumentada
 - Nunca assume valor inferior a sua prioridade de base, nem superior a 15
- Fator adicional em máquina multiprocessadoras: afinidade!
 - Tentativa de escalonar uma *thread* no processador que ela executou mais recentemente.
 - Princípio: reaproveitamento de dados na memória cache

Escalonamento não preemptivo com prioridades

- SJF é um forma de priorizar processos
 - A prioridade é o inverso do próximo tempo previsto para ciclo de CPU
- Processos de igual prioridade são executados de acordo com uma política FIFO
- Problema de postergação indefinida (*starvation*)
 - Processo de baixa prioridade não é alocado a CPU por sempre existir um processo de mais alta prioridade a ser executado
 - Solução:
 - Envelhecimento
- O conceito de prioridade é mais “consistente” com preempção
 - Processo de maior prioridade interrompe a execução de um menos prioritário

Leituras complementares

- R. Oliveira, A. Carissimi, S. Toscani; Sistemas Operacionais. Editora Sagra-Luzzato, 2001.
 - Capítulo 4, Capítulo 9 (seção 9.4), Capítulo 10 (seção 10.4)
- A. Silberchatz, P. Galvin; Operating System Concepts. (4th edition) Addison-Wesley, 1994.
 - Capítulo 5
- 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