Sistemas Operacionais

Profa. Dra. Kalinka Regina Lucas Jaquie Castelo Branco
kalinka@icmc.usp.br

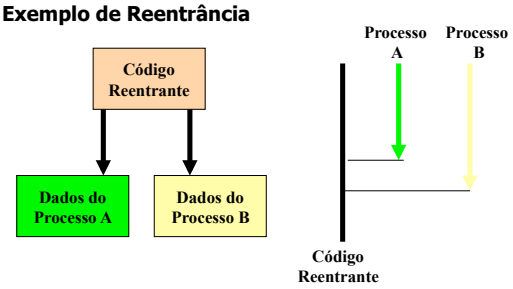
Apresentação baseada nos slides
do Prof. Dr. Antônio Carlos Sementille e da Profa. Dra. Luciana A. F. Martimiano e nas transparências fornecidas no site de compra do livro "Sistemas Operacionais Modernos"

Processos

- **Definição de Processo**
  - **É um programa em execução, incluindo os valores correntes do contador de programa, registradores, e variáveis. O conceito de processo é dinâmico, em contraposição ao conceito de programa, que é estático.**
  - **Nem sempre um programa equivale a apenas um processo**
  - **Em sistemas que permitem a reentrância, o código de um programa pode gerar diversos processos.**

Processos

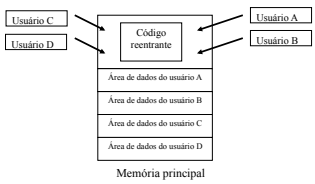
- **Exemplo de Reentrância**



Processos

- **Reentrância**
  - É a capacidade de um código executável (código reentrante) ser compartilhado por diversos usuários, exigindo que apenas uma cópia do programa esteja na memória.
  - Permite que cada usuário possa estar em um ponto diferente do código reentrante, manipulando dados próprios, exclusivos de cada usuário.

Processos



Criação de Processos

Principais eventos que causam a criação de um processo

1. Inicialização do Sistema
2. Execução de uma chamada ao sistema de criação de processo por um processo em execução
3. Uma requisição de Usuário para a criação de um novo processo
4. Início de um Job em lote

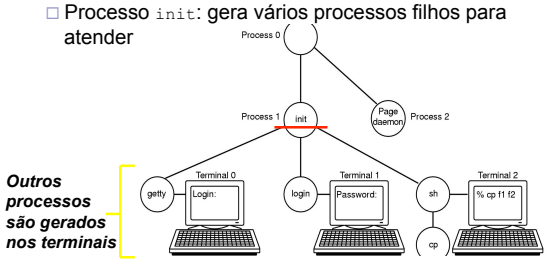
Criando Processos

- UNIX:
  - `Fork;`
    - Cria processo Pai e processo Filho com mesmo endereçamento;
    - Depois o processo Filho tem endereçamento separado;
- Windows:
  - `CreateProcess`
    - Cria processo Pai e processo Filho com mesmo endereçamento sempre;

7

Criando Processos

- Exemplo UNIX:
  - Processo `init`: gera vários processos filhos para atender



Outros processos são gerados nos terminais

8

Término de um Processo

Condições que podem provocar o término de um processo

1. Saída normal (voluntária)
2. Saída por erro (voluntária)
3. Erro Fatal (involuntária)
4. Destruído por outro processo (involuntário)

9

Finalizando Processos

- Condições:
  - Término normal (voluntário):
    - A tarefa a ser executada é finalizada;
    - Chamadas: *`exit` (UNIX) e `ExitProcess` (Windows)*
  - Término com erro (voluntário):
    - O processo sendo executado não pode ser finalizado: gcc filename.c, o arquivo filename.c não existe;

10

Finalizando Processos

- Término com erro fatal (involuntário);
  - Erro causado por algum erro no programa (*bug*):
    - Divisão por 0 (zero);
    - Referência à memória inexistente ou não pertencente ao processo;
    - Execução de uma instrução ilegal;
- Término causado por algum outro processo (involuntário):
  - `Kill` (UNIX) e `TerminateProcess` (Windows);

11

Hierarquias de Processos

- Pai cria um processo filho, processo filho pode criar seus próprios processo
- Formação de hierarquia
  - UNIX chama a isto um “grupo de processos” ("process group")
- Windows não possui conceito de hierarquia de processos
  - Todos os processos são criados da mesma forma

12

Processos

- **Os Processos do Ponto de Vista do S.O.**

Um sistema operacional deve incluir funções para o gerenciamento de processos, como por exemplo:

- **criação e remoção (destruição) de processos;**
- **controle do progresso dos processos;**
- **agir em condições excepcionais que acontecem durante a execução de um processo, incluindo interrupções e erros aritméticos;**
- **alocação de recursos de hardware entre os processos;**
- **fornecer um meio de comunicação através de mensagens ou sinais entre os processos.**

13

Processos

Existem 2 tipos de sistemas, com relação aos processos:

- **Sistemas Estáticos**

Geralmente são sistemas projetados para executar uma única aplicação. Permite que todos os processos necessários já estejam presentes quando o sistema é iniciado.

- **Sistemas Dinâmicos**

Possuem um número variável de processos. Necessita de uma forma de criar e destruir processos durante a execução.

14

Processos

- **O Bloco de Controle de Processos**

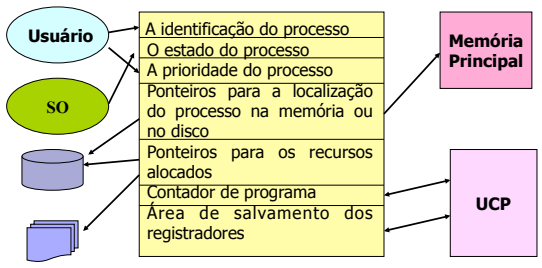
No sistema, cada processo será representado por seu resumo, que consiste no Bloco de Controle de Processo (BCP), também conhecido por Bloco de Controle de Programa ou Descritor de Processo. O BCP consiste de uma estrutura de dados contendo informações importantes sobre o processo, incluindo:

- **A identificação do processo;**
- **O estado do processo;**
- **A prioridade do processo;**
- **Ponteiros para a localização do processo na memória ou no disco;**
- **Ponteiros para os recursos alocados;**
- **Contador de programa;**
- **Área de salvamento dos registradores, etc.**

15

Processos

O Bloco de Controle de Processos



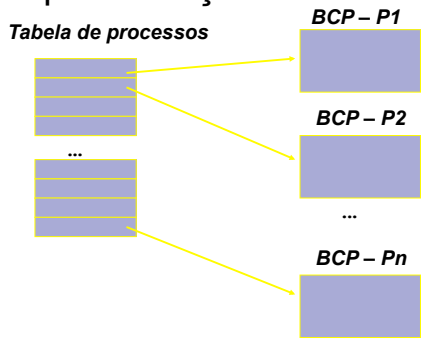
16

Implementação de Processos

- Tabela de Processos:
  - Cada processo possui uma entrada;
  - Cada entrada possui um ponteiro para o bloco de controle de processo (BCP) ou descritor de processo;
  - BCP possui todas as informações do processo → contextos de hardware, software, endereço de memória;

17

Implementação de Processos



18

Implementação de Processos

Process management	Memory management	File management
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Algumas informações do BCP

19

Processos

- **Os Estados de um Processo**

  Desde o momento em que um processo codificado pelo programador for colocado em memória ou disco, até o momento de sua destruição, ele poderá passar por quatro estados:

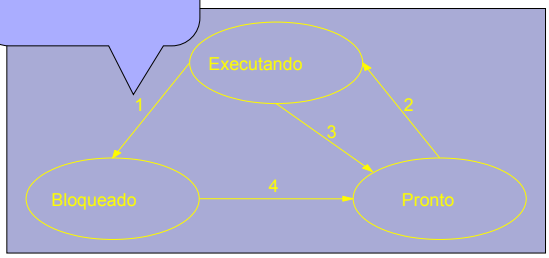
  - **Indefinido:** quando o processo é desconhecido ao S.O. Um processo estará neste estado antes de ser criado e depois de ser destruído. (Neste ponto, ele será apenas um bloco de código no disco ou na memória).
  - **Bloqueado:** quando o processo estiver parado à espera da ocorrência de um evento, equivalente a não estar em andamento progressivo. Ao ser criado, o processo passará por este estado.
  - **Pronto para a execução:** quando o processo só não estiver em execução pelo fato da UCP estar sendo utilizada por outro processo.
  - **Em execução:** quando o processo estiver tendo andamento progressivo normal, usando a UCP.

20

Processos

Um processo sendo executado não pode continuar sua execução, pois precisa de algum evento (E/S ou semáforo) para continuar;

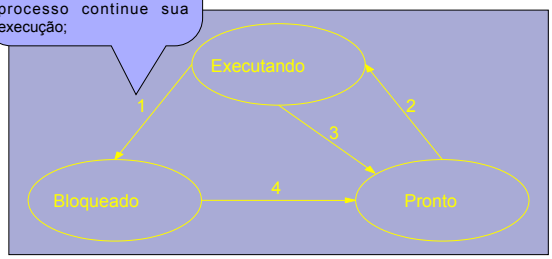
icos:



21

Processos

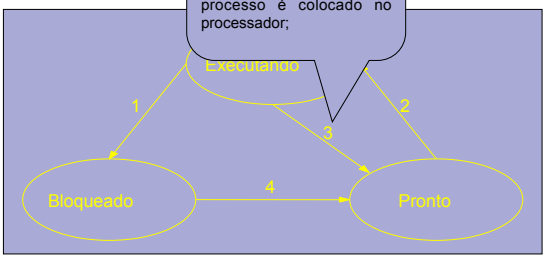
Um processo é bloqueado de duas maneiras:
- chamada ao sistema: `block` ou `pause`;
- se não há entradas disponíveis para que o processo continue sua execução;



22

Estados de

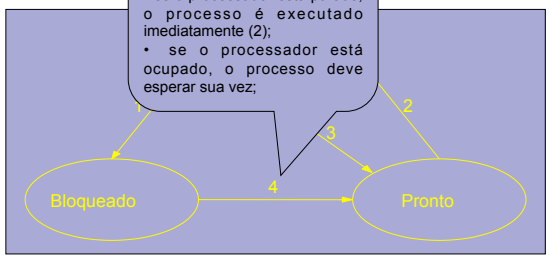
As transições 2 e 3 ocorrem durante o escalonamento de processos:
- o tempo destinado àquele processo acabou e outro processo é colocado no processador;



23

Estados de Processos

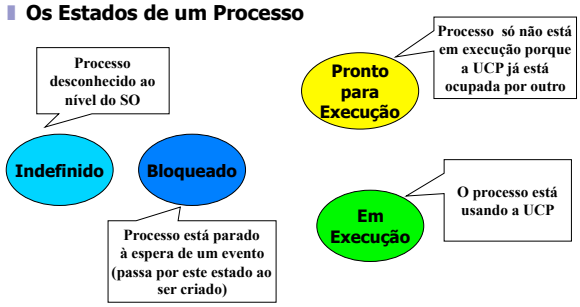
A transição 4 ocorre quando o evento esperado pelo processo bloqueado ocorre:
- se o processador está parado, o processo é executado imediatamente (2);
- se o processador está ocupado, o processo deve esperar sua vez;



24

Processos

- **Os Estados de um Processo**



25

Processos

- **As Ações que provocam mudanças de estado**

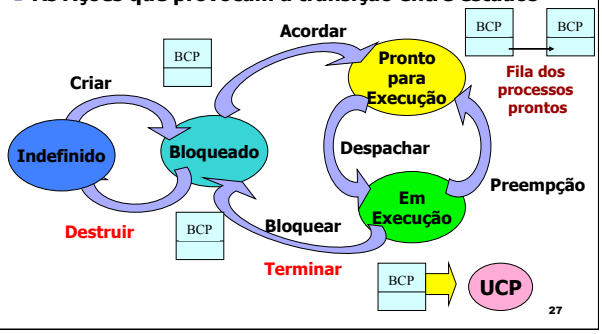
As ações responsáveis pelas transições tem o seguinte significado:

- **Criar:** colocar o processo a memória, tornando-o conhecido ao nível do sistema.
- **Acordar:** liberar o andamento do processo. Esta ação será desempenhada quando ocorrer algum evento ou o disparo inicial do processo, fazendo com que haja a transição do estado bloqueado para o pronto para execução.
- **Despachar:** dar seqüência imediata ao andamento do processo. O processo deverá estar em uma fila, e a ocorrência dessa ação resultará na retirada do processo da fila e sua colocação no estado em execução.
- **Bloquear:** parar o andamento do processo para esperar a ocorrência de um evento. A ação bloquear será desempenhada quando uma determinada condição não for satisfeita, ou quando ocorrer o término do processo.
- **Preempção (Suspender):** parar o andamento do processo por motivos alheios ao mesmo, como acontece na comutação forçada de fatias de tempo. Esta ação retirará o processo da UCP e o colocará numa fila, no estado pronto para a execução, liberando a utilização da UCP.
- **Destruir:** liberar a área de memória ocupada pelo processo, tornando-o desconhecido ao nível do sistema.

26

Processos

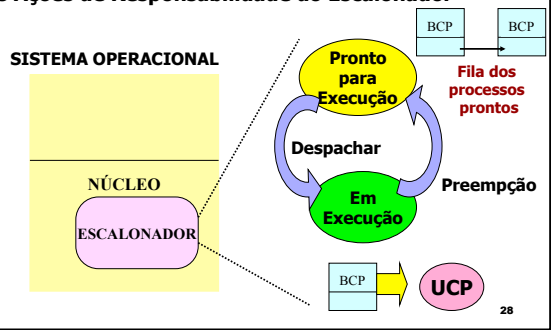
- **As Ações que provocam a transição entre estados**



27

Processos

- **As Ações de Responsabilidade do Escalonador**



28

Processos

- Processos *CPU-bound* (orientados à CPU): processos que utilizam muito o processador;
  - Tempo de execução é definido pelos ciclos de processador;
- Processos *I/O-bound* (orientados à E/S): processos que realizam muito E/S;
  - Tempo de execução é definido pela duração das operações de E/S;
- **IDEAL**: existir um balanceamento entre processos *CPU-bound* e *I/O-bound*;

29

Processos

- **Outras características dos processos**
  - **Os processos não podem ser programados com suposições embutidas sobre temporização**
  - **Em geral, um sistema multiplexa um só processador central entre vários processos. Os instantes em que o processador é realocado de um processo para outro são, em geral, imprevisíveis**
  - **Um algoritmo de escalonamento determina quando parar o trabalho com um processo e atender um diferente**

30

Escalonamento de Processos

- Escalonador de Processos escolhe o processo que será executado pela CPU;
- Escalonamento é realizado com o auxílio do hardware;
- Escalonador deve se preocupar com a eficiência da CPU, pois o chaveamento de processos é complexo e custoso:
  - Afeta desempenho do sistema e satisfação do usuário;
- Escalonador de processo é um processo que deve ser executado quando da **mudança de contexto** (troca de processo);

31

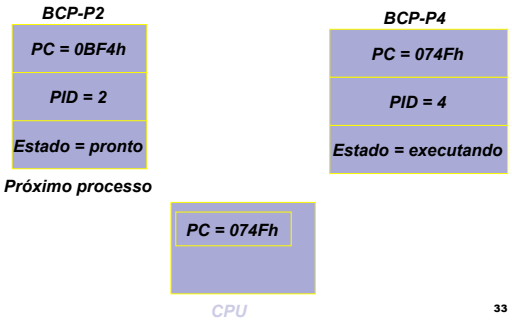
Escalonamento de Processos

- Mudança de Contexto:
  - Overhead de tempo;
  - Tarefa cara:
    - Salvar as informações do processo que está deixando a CPU em seu BCP → conteúdo dos registradores;
    - Carregar as informações do processo que será colocado na CPU → copiar do BCP o conteúdo dos registradores;

32

Escalonamento de Processos

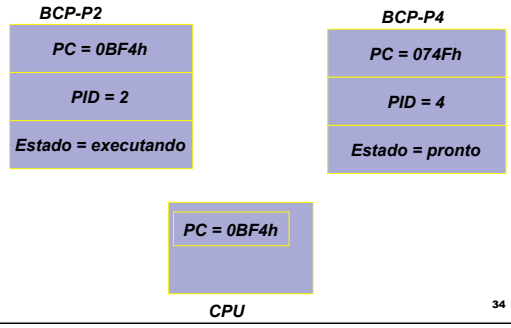
Antes da Mudança de Contexto



33

Escalonamento de Processos

Depois da Mudança de Contexto



34

Escalonamento de Processos

- Situações nas quais escalonamento é necessário:
  - Um novo processo é criado;
  - Um processo terminou sua execução e um processo pronto deve ser executado;
  - Quando um processo é bloqueado (semáforo, dependência de E/S), outro deve ser executado;
  - Quando uma interrupção de E/S ocorre o escalonador deve decidir por: executar o processo que estava esperando esse evento; continuar executando o processo que já estava sendo executado ou executar um terceiro processo que esteja pronto para ser executado;

35

Escalonamento de Processos

- Tempo de execução de um processo é imprevisível:
  → CPU gera interrupções em intervalos entre 50 a 60 hz (ocorrências por segundo);
- Algoritmos de escalonamento podem ser divididos em duas categorias dependendo de como essas interrupções são tratadas:
  - Preemptivo: estratégia de suspender o processo sendo executado;
  - Não-preemptivo: estratégia de permitir que o processo sendo executado continue sendo executado até ser bloqueado por alguma razão (semáforos, operações de E/S-interrupção);

36

Escalonamento de Processos

- Categorias de Ambientes:
  - **Sistemas em Batch**: usuários não esperam por respostas rápidas; algoritmos preemptivos ou não-preemptivos;
  - **Sistemas Interativos**: interação constante do usuário; algoritmos preemptivos; Processo interativo → espera comando e executa comando;
  - **Sistemas em Tempo Real**: processos são executados mais rapidamente; tempo é crucial → sistemas críticos;

37

Escalonamento de Processos

- Características de algoritmos de escalonamento:
  - Qualquer sistema:
    - **Justiça** (*Fairness*)*:* cada processo deve receber uma parcela justa de tempo da CPU;
    - **Balanceamento**: diminuir a ociosidade do sistema;
    - **Políticas do sistema** – prioridade de processos;

38

Escalonamento de Processos

- Características de algoritmos de escalonamento:
  - Sistemas em *Batch*:
    - **Taxa de execução** (*throughput*): máximo número de *jobs* executados por hora;
    - ***Turnaround time*** (tempo de retorno): tempo no qual o processo espera para ser finalizado;
    - **Tempo de espera**: tempo gasto na fila de prontos;
    - **Eficiência**: CPU deve estar 100% do tempo ocupada;
  - Sistemas Interativos:
    - **Tempo de resposta**: tempo esperando para iniciar execução;
    - **Satisfação do usuários**;

39

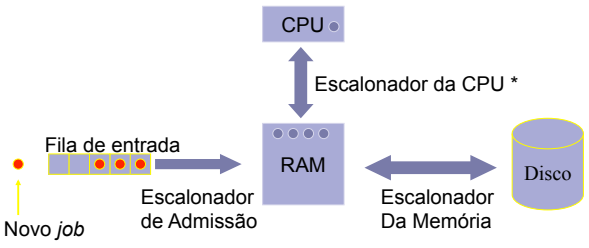
Escalonamento de Processos

- Características de algoritmos de escalonamento:
  - Sistemas em Tempo Real:
    - **Prevenir perda de dados**;
    - **Previsibilidade**: prevenir perda da qualidade dos serviços oferecidos;

40

Escalonamento de Processos Sistemas em *Batch*

- Escalonamento *Three-Level*



41

Escalonamento de Processos Sistemas em *Batch*

- Escalonamento *Three-Level*
  - **Escalonador de admissão**: processos menores primeiro; processos com menor tempo de acesso à CPU e maior tempo de interação com dispositivos de E/S;
  - **Escalonador da Memória**: decisões sobre quais processos vão para a MP:
    - A quanto tempo o processo está esperando?
    - Quanto tempo da CPU o processo já utilizou?
    - Qual o tamanho do processo?
    - Qual a importância do processo?
  - **Escalonador da CPU**: seleciona qual o próximo processo a ser executado;

42

Escalonamento de Processos Sistemas em *Batch*

- Algoritmos para Sistemas em *Batch*:
  - *First-Come First-Served (ou FIFO)*;
  - *Shortest Job First (SJF)*;
  - *Shortest Remaining Time Next (SRTN)*;

43

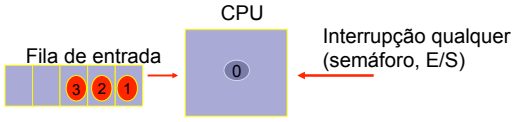
Escalonamento de Processos Sistemas em *Batch*

- Algoritmo *First-Come First-Served*
  - Não-preemptivo;
  - Processos são executados na CPU seguindo a ordem de requisição;
  - Fácil de entender e programar;
  - Desvantagem:
    - Ineficiente quando se tem processos que demoram na sua execução;

44

Escalonamento de Processos Sistemas em *Batch*

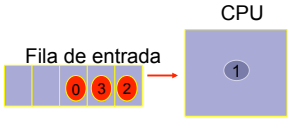
- Algoritmo *First-Come First-Served*



45

Escalonamento de Processos Sistemas em *Batch*

- Algoritmo *First-Come First-Served*



CPU não controla o tempo dos processos! (não-preemptivo)

46

Escalonamento de Processos Sistemas em *Batch*

- Algoritmo *Shortest Job First*
  - Não-preemptivo;
  - Possível prever o tempo de execução do processo;
  - Menor processo é executado primeiro;
  - Menor *turnaround*;
  - Desvantagem:
    - Baixo aproveitamento quando se tem poucos processos prontos para serem executados;

47

Escalonamento de Processos Sistemas em *Batch*

- Algoritmo *Shortest Job First*

A → a
B → b+a
C → c+b+a
D → d+c+b+a

Tempo médio-*turnaround* (4a+3b+2c+d)/4

Contribuição → se a<b<c<d tem-se o mínimo tempo médio;

48

Escalonamento de Processos Sistemas em *Batch*

- Algoritmo *Shortest Job First*

Em ordem:
Turnaround A = 8
Turnaround B = 12
Turnaround C = 16
Turnaround D = 20
Média → 56/4 = 14

Menor *job* primeiro:
Turnaround B = 4
Turnaround C = 8
Turnaround D = 12
Turnaround A = 20
Média → 44/4 = 11

(4a+3b+2c+d)/4 ← Número de Processos

Escalonamento de Processos Sistemas em *Batch*

- Algoritmo *Shortest Remaining Time Next*
  - Preemptivo;
  - Processos com menor tempo de execução são executados primeiro;
  - Se um processo novo chega e seu tempo de execução é menor do que do processo corrente na CPU, a CPU suspende o processo corrente e executa o processo que acabou de chegar;
  - Desvantagem: processos que consomem mais tempo podem demorar muito para serem finalizados se muitos processos pequenos chegarem!

Escalonamento de Processos Sistemas Interativos

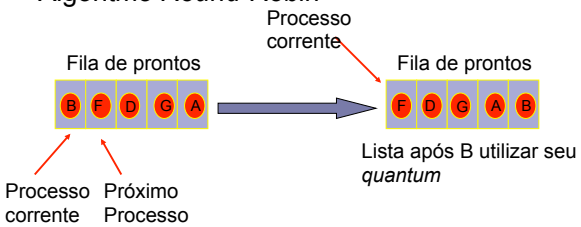
- Algoritmos para Sistemas Interativos:
  - *Round-Robin*;
  - Prioridade;
  - Múltiplas Filas;
  - *Shortest Process Next*;
  - Garantido;
  - *Lottery*;
  - *Fair-Share;*
- Utilizam escalonamento em dois níveis (escalonador da CPU e memória);

Escalonamento de Processos Sistemas Interativos

- Algoritmo *Round-Robin*
  - Antigo, mais simples e mais utilizado;
  - Preemptivo;
  - Cada processo recebe um tempo de execução chamado *quantum*; ao final desse tempo, o processo é suspenso e outro processo é colocado em execução;
  - Escalonador mantém uma lista de processos prontos;

Escalonamento de Processos Sistemas Interativos

- Algoritmo *Round-Robin*



Escalonamento de Processos Sistemas Interativos

- Algoritmo *Round-Robin*
  - Tempo de chaveamento de processos;
  - *quantum*: se for muito pequeno, ocorrem muitas trocas diminuindo, assim, a eficiência da CPU; se for muito longo o tempo de resposta é comprometido;

Escalonamento de Processos
Sistemas Interativos

- Algoritmo *Round-Robin:*

Exemplos:

quantum

Δt = 4 mseg
x = 1mseg → 25% de tempo de CPU é perdido → menor eficiência

Δt = 100 mseg
x = 1mseg → 1% de tempo de CPU é perdido → Tempo de espera dos processos é maior

chaveamento

55

Escalonamento de Processos
Sistemas Interativos

- Algoritmo *Round-Robin:*

Exemplos:

Δt = 4 mseg
x = 1mseg → 25% de tempo de CPU é perdido → menor eficiência

Δt = 100 mseg
x = 1mseg → 1% de tempo de CPU é perdido → Tempo de espera dos processos é maior

quantum razoável: 20-50 mseg

56

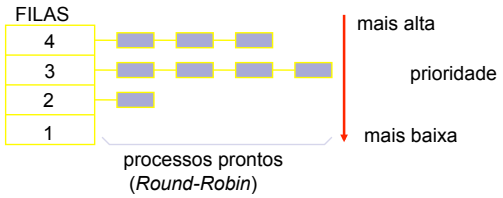
Escalonamento de Processos
Sistemas Interativos

- Algoritmo com Prioridades
  - Cada processo possui uma prioridade → os processos prontos com maior prioridade são executados primeiro;
  - Prioridades são atribuídas dinâmica ou estaticamente;
  - Classes de processos com mesma prioridade;
  - Preemptivo;

57

Escalonamento de Processos
Sistemas Interativos

- Algoritmo com Prioridades



58

Escalonamento de Processos
Sistemas Interativos

- Algoritmo com Prioridades
  - Como evitar que os processos com maior prioridade sejam executado indefinidamente?
    - Diminuir a prioridade do processo corrente e troca-lo pelo próximo processo com maior prioridade (chaveamento);
    - Cada processo possui um *quantum*;

59

Escalonamento de Processos
Sistemas Interativos

- Múltiplas Filas:
  - CTSS (*Compatible Time Sharing System*);
  - Classes de prioridades;
  - Cada classe de prioridades possui *quanta* diferentes;
  - Assim, a cada vez que um processo é executado e suspenso ele recebe mais tempo para execução;
  - Preemptivo;

60

Escalonamento de Processos Sistemas Interativos

- Múltiplas Filas:
  - Ex.: um processo precisa de 100 *quanta* para ser executado;
    - Inicialmente, ele recebe um *quantum* para execução;
    - Das próximas vezes ele recebe, respectivamente, 2, 4, 8, 16, 32 e 64 *quanta* (7 chaveamentos) para execução;
    - Quanto mais próximo de ser finalizado, menos frequente é o processo na CPU → eficiência

61

Escalonamento de Processos Sistemas Interativo

- Algoritmo *Shortest Process Next*
  - Mesma idéia do *Shortest Job First*;
  - Processos Interativos: não se conhece o tempo necessário para execução;
  - Como empregar esse algoritmo: ESTIMATIVA de TEMPO!

62

Escalonamento de Processos Sistemas Interativo

- Outros algoritmos:
  - Algoritmo Garantido:
    - Garantias são dadas aos processos dos usuários:
      - **n** usuários → **1/n** do tempo de CPU para cada usuário;
  - Algoritmo *Lottery*:
    - Cada processo recebe "*tickets*" que lhe dão direito de execução;
  - Algoritmo *Fair-Share*:
    - O dono do processo é levado em conta;
    - Se um usuário A possui mais processos que um usuário B, o usuário A terá prioridade no uso da CPU;

63

Escalonamento de Processos Sistemas em Tempo Real

- Tempo é um fator crítico;
- Sistemas críticos:
  - Aviões;
  - Hospitais;
  - Usinas Nucleares;
  - Bancos;
  - Multimídia;
- Ponto importante: obter respostas em atraso é tão ruim quanto não obter respostas;

64

Escalonamento de Processos Sistemas em Tempo Real

- Tipos de STR:
  - ***Hard Real Time***: atrasos não são tolerados;
    - Aviões, usinas nucleares, hospitais;
  - ***Soft Real Time***: atrasos são tolerados;
    - Bancos; Multimídia;
- Programas são divididos em vários processos;
- Eventos causam a execução de processos:
  - **Periódicos**: ocorrem em intervalos regulares de tempo;
  - **Aperiódicos**: ocorrem em intervalos irregulares de tempo;

65

Escalonamento de Processos Sistemas em Tempo Real

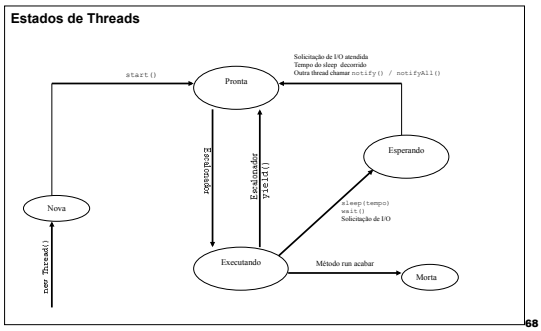
- Algoritmos podem ser estáticos ou dinâmicos;
  - **Estáticos**: decisões de escalonamento antes do sistema começar;
    - Informação disponível previamente;
  - **Dinâmicos**: decisões de escalonamento em tempo de execução;

66

Thread

- "Thread, ou processo leve, é a unidade básica de utilização da CPU, consistindo de: contador de programa, conjunto de registradores e uma pilha de execução."
- "Thread são estruturas de execução pertencentes a um processo e assim compartilham os segmentos de código e dados e os recursos alocados ao sistema operacional pelo processo. O conjunto de threads de um processo é chamado de Task e um processo tradicional possui uma Task com apenas um thread." **Silberschatz**

Thread



Thread

- O conceito de thread foi criado com dois objetivos principais:
  - Facilidade de comunicação entre unidades de execução;
  - Redução do esforço para manutenção dessas unidades.

Thread

- Isso foi conseguido por meio da criação dessas unidades dentro de processos, fazendo com que todo o esforço para criação de um processo, manutenção do Espaço de endereçamento lógico e BCP, fosse aproveitado por várias unidades processáveis, conseguindo também facilidade na comunicação entre essas unidades.

Thread

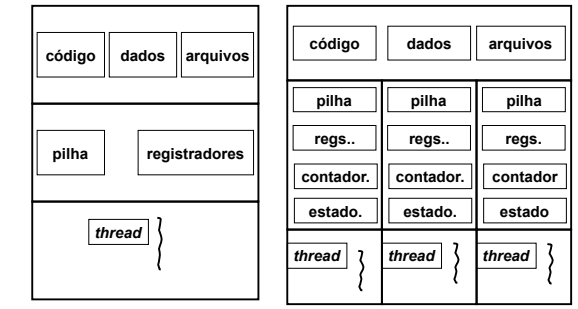
- **Processo ->** um espaço de endereço e uma única linha de controle
- **Threads ->** um espaço de endereço e múltiplas linhas de controle
  - O Modelo do Processo
    - Agrupamento de recursos (espaço de endereço com texto e dados do programa; arquivos abertos, processos filhos, tratadores de sinais, alarmes pendentes etc)
    - Execução
  - O Modelo da Thread
    - Recursos particulares (PC, registradores, pilha)
    - Recursos compartilhados (espaço de endereço – variáveis globais, arquivos etc)
    - Múltiplas execuções no mesmo ambiente do processo – com certa independência entre as execuções
- **Analogia**
  - Execução de múltiplos threads em paralelo em um processo *(multithreading)* e Execução de múltiplos processos em paralelo em um computador

Threads

Itens por Processo	Itens por *Thread*
■Espaço de endereçamento ■Variáveis globais ■Arquivos abertos ■Processos filhos ■Alarmes pendentes	■Contador de programa ■Registradores (contexto) ■Pilha ■Estado

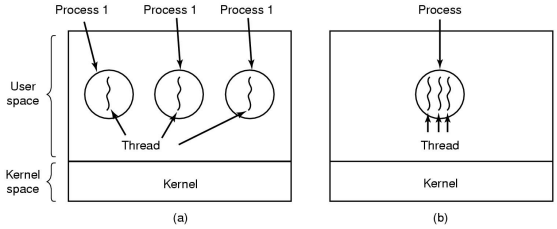
- Compartilhamento de recursos;
- Cooperação para realização de tarefas;

Threads



Processo com uma única *thread* **Processo com várias *threads***

Threads
O Modelo Thread (1)



(a) Três processos, cada um com um *thread*
(b) Um processo com três *threads*

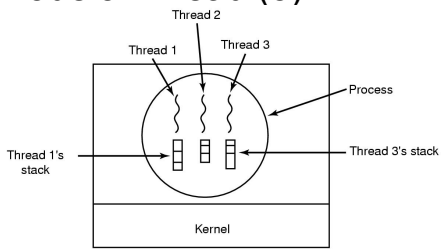
Thread

- Um thread é similar aos programas sequenciais, pois possui um início, sequência de execução e um fim e em qualquer momento uma thread possui um único ponto de execução.
- Contudo, um thread não é um programa, ele não pode ser executado sozinho e sim inserido no contexto de uma aplicação, onde essa aplicação sim, possuirá vários pontos de execuções distintos, cada um representado por um thread.

Thread

- Não há nada de novo nesse conceito de processo com um único thread, pois o mesmo é idêntico ao conceito tradicional de processo.
- O grande benefício no uso de thread é quando temos vários thread em um mesmo processo sendo executados simultaneamente e podendo realizar tarefas diferentes.

O Modelo Thread (3)



Cada thread tem sua própria pilha de execução

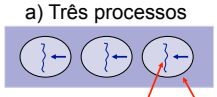
Threads

- Dessa forma pode-se perceber facilmente que aplicações multithreads podem realizar tarefas distintas ao "mesmo tempo", dando idéia de paralelismo.
- Exemplo: navegador web HotJava
  - consegue carregar e executar applets;
  - executar uma animação;
  - tocar um som;
  - exibir diversas figuras;
  - permitir rolagem da tela;
  - carregar uma nova página; entre outros
- para o usuário todas essas atividades são simultâneas, mesmo possuindo um único processador (possível devido a execução de vários threads, provavelmente, uma para cada tarefa a ser realizada.)

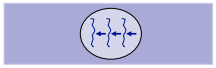
Threads

- Tradicionalmente, processos possuem apenas um contador de programas, um espaço de endereço e apenas uma *thread* de controle (ou fluxo de controle);
- ***Multithreading***: Sistemas atuais suportam múltiplas *threads* de controle;

a) Três processos



b) Um processo com três *threads*



• As três *threads* utilizam o mesmo espaço de endereço

79

Threads

- *Thread* é uma entidade básica de utilização da CPU.
  - ou processos leves (*lightweight process*);
- Processos com múltiplas *threads* podem realizar mais de uma tarefa de cada vez;
- Processos são usados para agrupar recursos; *threads* são as entidades escalonadas para execução na CPU
  - A CPU alterna entre as *threads* dando a impressão de que elas estão executando em paralelo;

80

Threads

- Não há proteção entre threads, pois é desnecessário;
  - Como cada *thread* pode ter acesso a qualquer endereço de memória dentro do espaço de endereçamento do processo, uma *thread* pode ler, escrever ou apagar a pilha de outra *thread*;

81

Threads

- Razões para existência de *threads*:
  - Em múltiplas aplicações ocorrem múltiplas atividades "ao mesmo tempo", e algumas dessas atividades podem bloquear de tempos em tempos;
  - As *threads* são mais fáceis de gerenciar do que processos, pois elas não possuem recursos próprios → o processo é que tem!
  - Desempenho: quando há grande quantidade de E/S, as threads permitem que essas atividades se sobreponham, acelerando a aplicação;
  - Paralelismo Real em sistemas com múltiplas CPUs.

82

Threads

- Considere um servidor de arquivos:
  - Recebe diversas requisições de leitura e escrita em arquivos e envia respostas a essas requisições;
  - Para melhorar desempenho, o servidor mantém um *cache* dos arquivos mais recentes, lendo do *cache* e escrevendo no *cache* quando possível;
  - Quando uma requisição é feita, um *thread* é alocada para seu processamento. Suponha que esse *thread* seja bloqueada esperando uma transferência de arquivos. Nesse caso, outros *threads* podem continuar atendendo a outras requisições;

83

Threads

- Considere um navegador WEB:
  - Muitas páginas WEB contêm muitas figuras que devem ser mostradas assim que a página é carregada;
  - Para cada figura, o navegador deve estabelecer uma conexão separada com o servidor da página e requisitar a figura → tempo;
  - Com múltiplas *threads*, muitas imagens podem ser requisitadas ao mesmo tempo melhorando o desempenho;

84

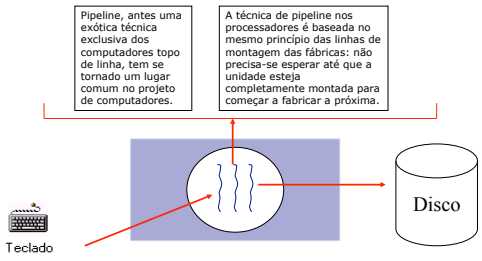
Threads

- Considere um Editor de Texto:
  - Editores mostram documentos formatados que estão sendo criados em telas (vídeo);
  - No caso de um livro, por exemplo, todos os capítulos podem estar em apenas um arquivo, ou cada capítulo pode estar em arquivos separados;
  - Diferentes tarefas podem ser realizadas durante a edição do livro;
  - Várias t*hreads* podem ser utilizadas para diferentes tarefas;

85

Threads

- *Threads* para diferentes tarefas;



86

Threads

- Benefícios:
  - Capacidade de resposta: aplicações interativas; Ex.: servidor WEB;
  - Compartilhamento de recursos: mesmo endereçamento; memória, recursos;
  - Economia: criar e realizar chaveamento de *threads* é mais barato;
  - Utilização de arquiteturas multiprocessador: processamento paralelo;

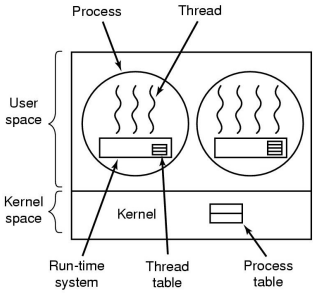
87

Threads

- Tipos de *threads*:
  - Em modo usuário (espaço do usuário): implementadas por bibliotecas no nível do usuário;
    - Criação e escalonamento são realizados sem o conhecimento do *kernel*;
      - Sistema Supervisor (*run-time system*): coleção de procedimentos que gerenciam as *threads*;
      - Tabela de *threads* para cada processo;
    - Cada processo possui sua própria tabela de *threads*, que armazena todas a informações referentes à cada *thread* relacionada àquele processo;

88

Threads em modo usuário



89

Threads em modo usuário

- Tipos de *threads*: Em modo usuário
- Vantagens:
  - Alternância de *threads* no nível do usuário é mais rápida do que alternância no *kernel*;
  - Menos chamadas ao *kernel* são realizadas;
  - Permite que cada processo possa ter seu próprio algoritmo de escalonamento;
- Principal desvantagem:
  - Processo inteiro é bloqueado se uma *thread* realizar uma chamada bloqueante ao sistema;

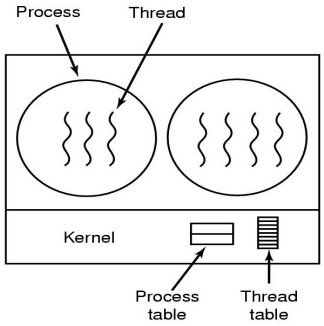
90

Tipos de *Threads*

- Tipos de *threads*:
  - Em modo *kernel*: suportadas diretamente pelo SO;
  - Criação, escalonamento e gerenciamento são feitos pelo *kernel*;
    - Tabela de *threads* e tabela de processos separadas;
      - as tabelas de *threads* possuem as mesmas informações que as tabelas de threads em modo usuário, só que agora estão implementadas no *kernel*;

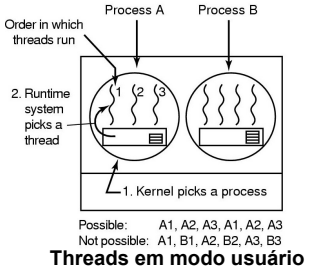
91

Threads em modo *kernel*

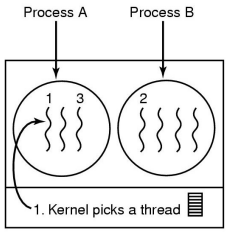


92

Threads em modo Usuário *x* *Threads* em modo *Kernel*



Threads em modo usuário



Threads em modo *kernel*

93

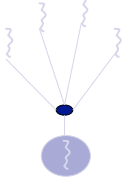
Threads em modo *kernel*

- Tipos de *threads*: em modo *kernel*
- Vantagem:
  - Processo inteiro não é bloqueado se uma *thread* realizar uma chamada bloqueante ao sistema;
- Desvantagem:
  - Gerenciar threads em modo *kernel* é mais caro devido às chamadas de sistema durante a alternância entre modo usuário e modo *kernel*;

94

Threads

- Modelos *Multithreading*
  - Muitos-para-um:
    - Mapeia muitas *threads* de usuário em apenas uma *thread* de *kernel*;
    - Não permite múltiplas *threads* em paralelo;



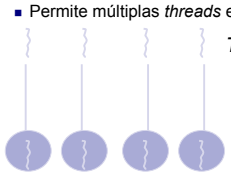
Threads em modo usuário

Thread em modo *kernel*

95

Threads

- Modelos *Multithreading*
  - Um-para-um: (Linux, Família Windows, OS/2, Solaris 9)
    - Mapeia para cada *thread* de usuário uma *thread* de *kernel*;
    - Permite múltiplas *threads* em paralelo;



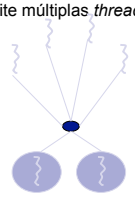
Threads em modo usuário

Threads em modo *kernel*

96

Threads

- Modelos *Multithreading*
  - Muitos-para-muitos: (Solaris até versão 8, HP-UX, Tru64 Unix, IRIX)
    - Mapeia para múltiplos *threads* de usuário um número menor ou igual de *threads* de *kernel*;
    - Permite múltiplas *threads* em paralelo;



Threads em modo usuário

Thread em modo *kernel*

97

Threads

- Estados: executando, pronta, bloqueada;
- Comandos para manipular threads:
  - *Thread_create;*
  - *Thread_exit;*
  - *Thread_wait;*
  - *Thread_yield* (permite que uma *thread* desista voluntariamente da CPU);

98

Threads

- **Porquê threads?**
  - Simplificar o modelo de programação (aplicação com múltiplas atividades => decomposição da aplicação em múltiplas threads)
  - Gerenciamento mais simples que o processo (não há recursos atachados – criação de thread 100 vezes mais rápida que processo)
  - Melhoria do desempenho da aplicação (especialmente quando thread é orientada a E/S)
  - Útil em sistemas com múltiplas CPUs

99