

Unidade 2

Programação Concorrente

- Processos
- Threads
- Concorrência em Java

Processos

- Definição
 - Um programa em execução em uma máquina.
 - Identificado pelo seu PID (Process Identifier).
 - A unidade de processamento em que um SO processa.
- Execução dos Processos
 - Um processador pode executar somente um processo a cada instante.
 - Em um **S.O. multi-tarefa**, processos se alternam no uso do processador – cada processo é executado durante um *quantum* de tempo.
 - Se houver N processadores, N processos podem ser executados simultaneamente.

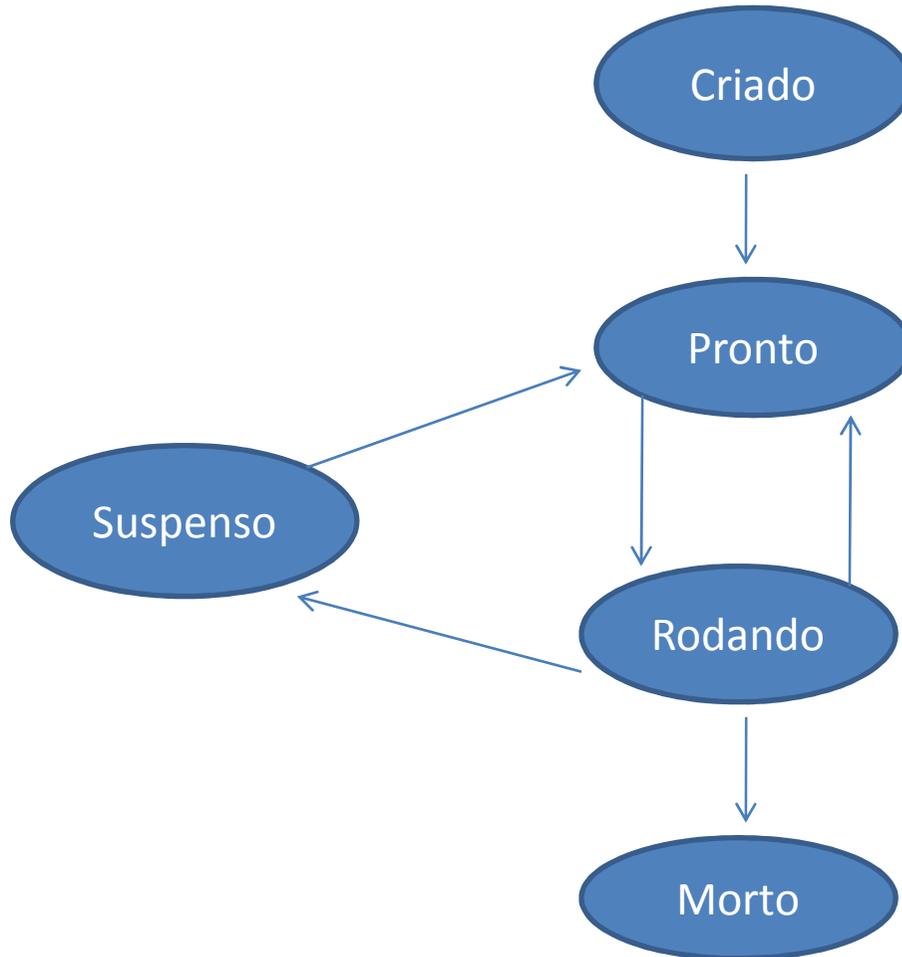
Escalonamento de Processos

- O escalonador do S.O. seleciona o(s) processo(s) que deve(m) ser executado(s) pelo(s) processador(es).

Escalonamento

- Algoritmo de Escalonamento
 - Define a ordem de execução de processos com base em uma fila, prioridade, deadline, ...
 - Em geral, os sistemas adotam uma política de melhor esforço para atender a todos os processos de maneira justa e igualitária
 - Processos do sistema e aplicações críticas (um alarme, por exemplo) exigem maior prioridade.

Ciclo de vida simplificado de um processo



Estados de um Processo

- **Pronto:** processo pronto para ser executado, mas sem o direito de usar o processador.
- **Rodando:** sendo executado pelo processador.
- **Suspenso:** aguarda operação de I/O, liberação de um recurso ou fim de tempo de espera.

Mudanças de Estado

- Processos trocam de estado de acordo com:
 - Algoritmo de escalonamento
 - Troca de mensagens
 - Interrupções de hardware ou software

Time-Slicing

- Divide o tempo do processador entre processos de **igual prioridade**.
- Isto é implementado por um Timer (hardware) o qual interrompe o processamento periodicamente, para permitir o **escalonador** buscar um outro processo para executar.

Escalonamento Pre-Emptivo

- **Prioridades dos processos.**
- Deve ser implementado para garantir que **um processo de alta prioridade possa executar logo que torna-se pronto**, mesmo que signifique **suspender a execução de um processo de mais baixa prioridade.**

Contexto de um Processo

- O estado do processo.
- Informações para escalonamento.
- Dados para contabilização de uso.
- Um segmento de código.
- Um segmento de dados.
- Os valores dos registradores.
- O contador de programa.
- Uma pilha de execução.
- Arquivos, portas e outros recursos alocados.

Mudança de Contexto

- Processos escalonados mudam de contexto.
- O processo em execução é suspenso, e um outro processo passa a ser executado.
- Ocorre por determinação do escalonador ou quando o processo que estava sendo executado é suspenso.
- O contexto do processo suspenso deve ser salvo para retomar a execução posteriormente.

Motivo da Suspensão de Processos

- **Dormindo** – em espera temporizada.
- **Bloqueado** – aguarda I/O.
- **Em Espera** - aguarda uma condição ser satisfeita.

Chamadas do Sistema Operacional UNIX

- Criar um Processo:
 - **fork()** cria uma cópia do processo atual.
 - **exec()** carrega o código do processo.

Chamadas do Sistema Operacional Unix

- Suspende a Execução:
 - **sleep(<tempo>)** ou
 - **wait()** - reinicia com **kill(<pid>,SIGWAIT)**
- Obter Identificador do Processo:
 - **getpid()**
- Aguarda o Fim dos Processos Criados: **join()**
- Finalizar o Processo: **exit(<codigo-retorno>)**
- Destruir um Processo: **kill(<pid>,SIGKILL)**

Interação com o Usuário no Unix

- Processos são criados através da interface gráfica ou de comandos digitados na Shell.
- Comandos bloqueiam a Shell, a não ser que sejam seguidos de um **&**.
- Os processos em execução são listados com o comando **ps -ef** (**ps -aux** em versões antigas)
- Processos são destruídos com **kill -9 <pid>**

Chamadas na API do Windows

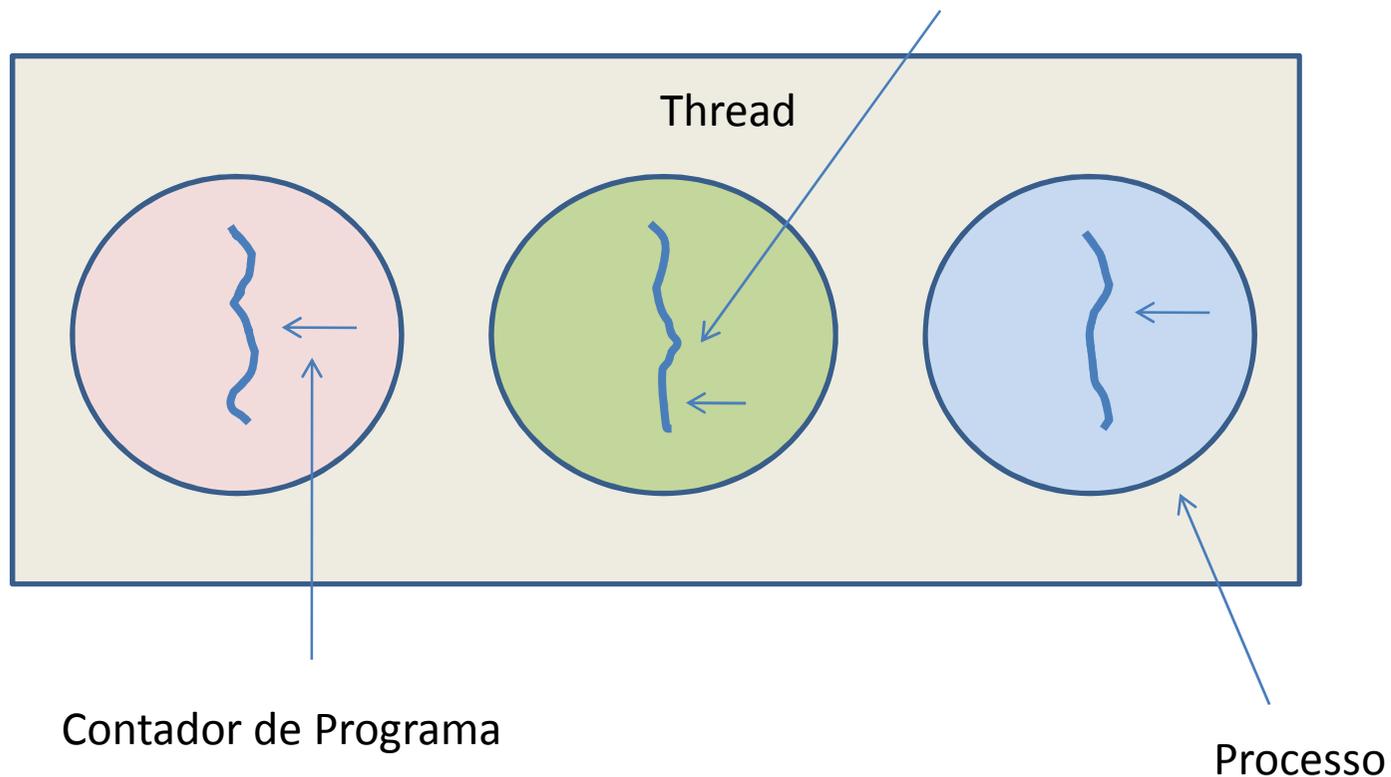
- Criar um Processo:
 - **CreateProcess(<nome>, <comando>, ...)** ou
 - **CreateProcessAsUser(<usuário>, <nome>, ...)**
- Obter o Identificador do Processo:
 - **GetCurrentProcessId()**
- Suspende a Execução:
 - **Sleep(<tempo>)**
- Finalizar o Processo:
 - **ExitProcess(<codigo-retorno>)**
- Destruir um Processo:
 - **TerminateProcess(<pid>, <retorno>)**

Threads

- Definição:
 - Threads (linhas) de execução são atividades concorrentes executadas por um processo.
 - Um processo pode ter uma ou mais threads.
 - Threads pertencentes a um mesmo processo compartilham recursos e memória.
- Suporte a Threads:
 - Threads nativas do S.O.
 - Suporte de programação multi-thread.
 - Linguagem de programação multi-threaded.

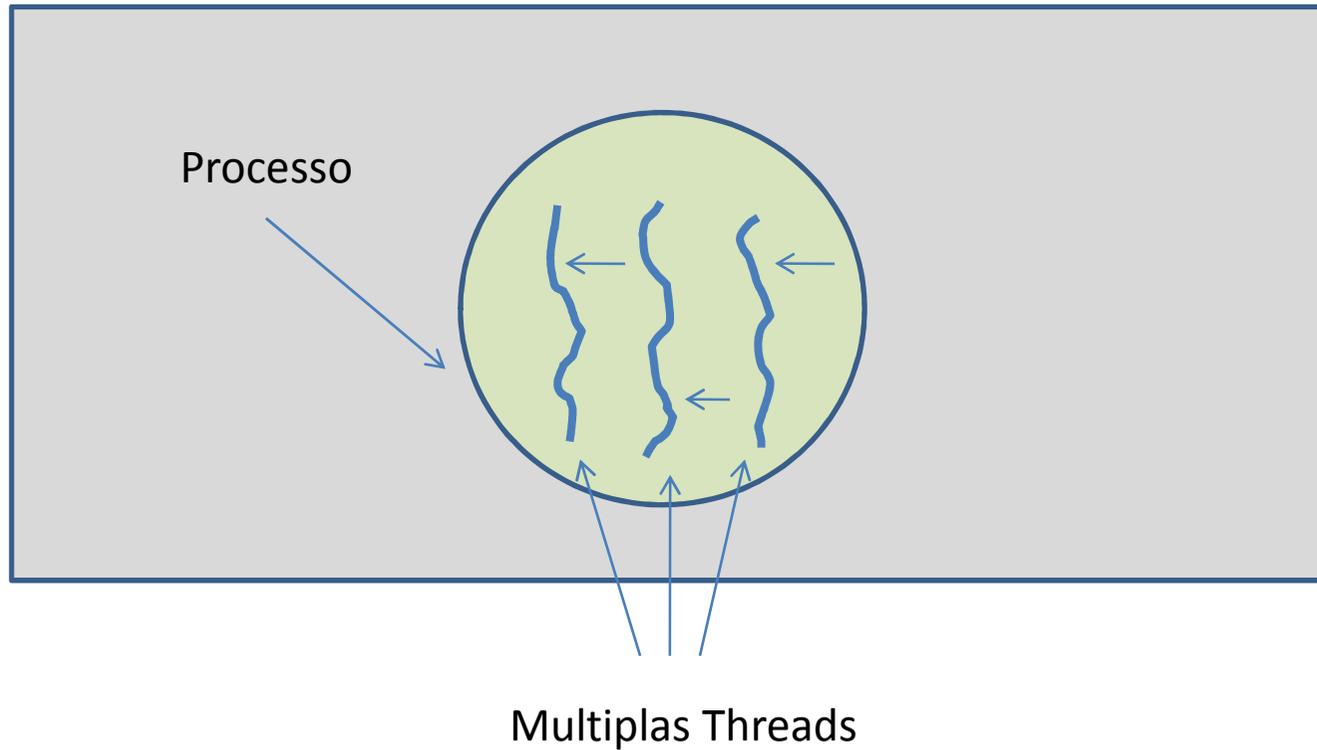
Três Processos cada um com uma Thread

Cada thread tem seu espaço de endereçamento.



Um Processo com três Threads

- Todas num mesmo espaço de endereçamento.



Diferença entre Programa e Processo

- Um confeiteiro
- Uma receita bolo
- Assar um bolo de aniversário
- Ingredientes: farinha, ovos, açúcar, ...
- A receita é o **programa**.
- O confeiteiro é o **processador**.
- Ingredientes: **dados**.

Diferença entre Programa e Processo

- O **processo** é a **atividade** que consiste em nosso confeitoiro **ler a receita** (Thread), **buscar os ingredientes** (Thread), **bater o bolo** (Thread) e **cozinhar o mesmo** (Thread).

Alternando para outro processo

- Confeiteiro
- Filho do confeiteiro
- Abelha
- Ferrada da abelha no filho do confeiteiro
- Confeiteiro precisa socorrer o filho.
- O confeitario registra onde estava na receita (o estado do processo atual é salvo).

Alternando para outro processo

- Confeiteiro procura um livro de pronto-socorro. Segue a orientações do livro (outro programa).
- O Confeiteiro alterna do processo (Cozimento) para outro, de prioridade mais alta (Administrar cuidado Médico), cada um tendo um programa diferente (receita x livro).

Processo é uma atividade

- Quando a picada for tratada, o confeiteiro volta ao seu bolo, continuando do ponto onde parou, quando abandonou o processo (Cozimento).
- A ideia é que **processo** é um tipo de **atividade**.
- Processo tem entrada, saída e estado.

Threads

- Da mesma forma que os processos.
- Cada *thread* tem seu estado e segue um ciclo de vida particular.
- A vida da *thread* depende do seu **processo**.

Exemplos com Múltiplas Threads

- Servidor de Arquivos
- Navegador Web
- ???

Threads

- **Escalonamento**

- **Por Processo:** escalonador aloca tempo para execução dos processos, os quais definem como usar este tempo para executar suas threads.

P1

P2

P3

t11 t12

t21 t22 t23

t31 t32

- **Por Thread:** escalonador define a ordem na qual as threads serão executadas.

t11

t31

t21

t32

t23

t12

t22

t16

Troca de Contexto

- Quando duas *threads* de um mesmo processo se alternam no uso do processador, ocorre uma **troca de contexto parcial**.
- Numa **troca parcial**, o contador de programa, os registradores e a pilha devem ser salvos.

Troca de Contexto

- Uma **troca de contexto parcial** é mais rápida que uma **troca de contexto entre processos**.
- Uma **troca de contexto completa** é necessária quando uma *thread* de um **processo que não estava em execução** assume o processador.

Processos x Threads

- **Troca de Contexto:** Completa | Parcial
- **Comunicação:** Inter-Processo | Intra-Processo
- **Suporte em S.O.'s:**
Quase todos | Os mais atuais
- **Suporte em Ling. Prog.:**
Quase todas | As mais recentes

Threads POSIX

- **Padrão adotado pelos UNIX e outros S.O.'s.**
- Criar uma Thread: **pthread_create(<thread>, <atrib>, <rotina>, <args>);**
- Obter Ident. da Thread: **pthread_self()**
- Suspende Execução:
pthread_delay_np(<tempo>)
- Finalizar a Thread: **pthread_exit (<retorno>)**
- **Linux** usa as mesmas rotinas, mas **cria um processo por thread** → **não é 100% POSIX**

Threads em Windows

- Criar uma Thread:
CreateThread (<atribos>, <tam_stack>, <rotina>, <params>, <flags>, <thread_id>)
- Obter Ident. da Thread: **GetCurrentThreadId()**
- Suspende Execução: **Sleep(<tempo>)** ou **SuspendThread (<thread>)** -> retomar com **ResumeThread(<t>)** ou **SwitchToThread(<t>)**
- Finalizar a Thread: **ExitThread(<retorno>)**
- Destruir uma Thread:
TerminateThread (<thread>, <retorno>)