Programação Concorrente

Prof. Ilo Rivero

ilorivero@live.com

Ementa

- Conceitos básicos de sistemas operacionais e multiprogramação.
- Processos concorrentes.
- Sincronização e deadlock.
- Multithreading, controle e sincronização.
- Tratamento de sinais.
- Mecanismos de IPC (Inter Process Comunication): locks, pipes e named pipes.
- Semáforos.
- Memória compartilhada.

Objetivos

- Ter conhecimento de:
  - Programação concorrente
  - Processos concorrentes
  - Sincronização
  - Multithreading
  - Tratamento de sinais
  - Mecanismos de IPC.

Conteúdo Programático

- Conceitos básicos de sistemas operacionais e multiprogramação.
  - Tipos de sistemas operacionais;
  - Linguagens para programação concorrente;
- Conceitos de Interrupção, processo, co-rotina e multiprogramação.
- Processos concorrentes.
  - Especificação da concorrência;
  - Relações de precedência entre processos;
  - Criação estática e dinâmica de processos;
  - Exemplos de programas concorrentes;
- Sincronização.
  - Sincronização para compartilhamento;
  - Sincronização para comunicação;
  - Sincronização tipo barreira;
  - Sincronizações básicas;

Conteúdo Programático

- Deadlocks.
  - Implementação de semáforos no kernel do SO;
  - Implementação de semáforos fora do kernel;
  - Sincronizações básicas com operações P e V;

- Programas clássicos:
  - Um alocador de recurso;
  - Produtor-consumidor com buffer limitado;
  - Jantar dos filósofos;
  - Problema do barbeiro dorminhoco;
  - Problema dos leitores e escritores;

Conteúdo Programático

- Memória compartilhada.
  - O problema da exclusão mútua;
  - Exclusão mútua para 2 processos:
    - Solução de Dekker;
    - Solução de Peterson;
    - Solução com instruções "test and test";
  - Exclusão Mútua com N processos:
    - Algoritmo de Dijkstra;
    - Algoritmo de Eisenberg e McGuire;
    - Algoritmo de Lamport;
    - Algoritmo de Peterson;
    - Algoritmo de Block e Woo;
    - Algortimo de Toscani;

Datas Importantes

ATPS

PROVA

Fevereiro	1	2	3	4	5	6	7	02/02 - Retorno do Corpo Docente
	8	9	10	11	12	13	14	09 a 13/02 - Semana Pedagógica
	15	16	17	18	19	20	21	16/02 - Recesso Escolar Discente - 17/02 - Carnaval - 18/02 Recesso Escolar Discente
	22	23	24	25	26	27	28	23/02 - Início das aulas
Março	1	2	3	4	5	6	7	
	8	9	10	11	12	13	14	09/03 - Abertura do AVA*
	15	16	17	18	19	20	21	
	22	23	24	25	26	27	28	
	29	30	31	1	2	3	4	03/04 - Sexta feira da Paixão - 04/04 Recesso Escolar Discente
Abril	5	6	7	8	9	10	11	
	12	13	14	15	16	17	18	15/04 - Entrega das notas 1ºbimestre
	19	20	21	22	23	24	25	21/04 - Tiradentes
	26	27	28	29	30	1	2	01/05 - Dia do Trabalho - 02/05 Recesso Escolar Discente
Maio	3	4	5	6	7	8	9	
	10	11	12	13	14	15	16	
	17	18	19	20	21	22	23	
	24	25	26	27	28	29	30	
	31	1	2	3	4	5	6	04/06 - Corpus Christi - 05/06 e 06/06 - Recesso Escolar Discente
Junho	7	8	9	10	11	12	13	
	14	15	16	17	18	19	20	15 a 19/06 - Período de Provas 2ºbimestre
	21	22	23	24	25	26	27	24 a 30/06 - Provas Substitutivas - 30/06 - Término do período letivo
	28	29	30					

Avaliações

- Primeiro Bimestre
  - 2 provas valendo 4 pontos (16/03 e 13/04)
  - ATPS valendo 2 pontos – 1 ponto cada etapa (23/03 e 13/04)
- Segundo Bimestre
  - 1 prova valendo 8 pontos (15/06)
  - ATPS valendo 2 pontos – 1 ponto cada etapa (18/05 e 15/06)

Bibliografia

Bibliografia Básica Unidade: Faculdade Anhanguera de Belo Horizonte (FAB)
1) SEIXAS FILHO, Constantino; SZUSTER, Marcelo (orgs.). **Programação Concorrente em Ambiente Windows** : Uma Visão de Automação. 1ª ed. Porto Alegre: UFMG - Universidade Federal de Minas Gerais, 2003.
2) TANENBAUM, Andrew S. **Sistemas Operacionais Modernos**. 2ª ed. São Paulo: Pearson - Prentice Hall, 2008.

Bibliografia Complementar: Faculdade Anhanguera de Belo Horizonte (FAB)
1) TANENBAUM, Andrew S.. **Sistemas Distribuidos**. 2ª ed. São Paulo: Plêiade, 2008.
2) MARQUES, José A.; GUEDES, Paulo. **Tecnologia de Sistemas Distribuídos**. 1ª ed. Lisboa, Portugal: FCA Editora Informática, 1998.
3) ALBUQUERQUE, F.. **TCP/IP Internet** : Programação de Sistemas Distribuídos HTML, JavaScript e Java. 1ª ed. São Paulo: Axcel Books, 2005.
4) TANENBAUM, Andrew S. **Sistemas distribuídos : princípios e paradigmas**. 2ª ed. São Paulo: Pearson - Prentice Hall, 2009.
5) COULOURIS, George et al. **Sistemas distribuídos : conceitos e projeto**. 4ª ed. São Paulo: Bookman, 2008.

O que é Programação Concorrente

- *“Um programa ‘ordinário’ consiste de declarações de dados e instruções executáveis em uma linguagem de programação.”*

M. Ben-Ari, Principles of Concurrent and Distributed Programming

O que é Programação Concorrente

- As instruções são executadas sequencialmente sobre um processador, o qual aloca memória o código e para os dados do programa.

- Um **programa concorrente** é um **conjunto de programas sequenciais ordinários** os quais são **executados em uma** *abstração* **de paralelismo**.

O que é Programação Concorrente

- Usamos a palavra **processo** para programas sequenciais e reservamos a palavra **programa** para o conjunto de processos.

Pseudo-Paralelismo

- Um **programa concorrente** é executado por se **compartilhar o poder de processamento de um único processador entre os processos** desse programa.

- Unidade de Processamento Concorrente: **Processo**

Abstração para Concorrência

- O paralelismo é ***abstrato*** porque não requeremos que **um processador físico** seja usado para **executar cada processo**.

- Pseudo-Paralelismo

Exemplos de Concorrência

- Sobreposição de I/O e Processamento
  (Overlapped I/O and Computation)
- Multiprogramação (Multi-programming)
- Multi-tarefação (Multi-Tasking)

Sobreposição de I/O e Computação

- No início dos tempos dos primeiros SOs, controlar I/O não podia ser feito concorrentemente com outra computação **sobre um único processador**.

- Mas a evolução do SOs, fez surgir a **concorrência**, retirando da computação principal, alguns microsegundos necessários para controlar I/O.

Sobreposição de I/O e Computação

- Entretanto, é mais simples **programar os controladores de I/O** como **processos separados**, os quais são **executados em paralelo** com o **processo de computação principal**.

Multiprogramação

- Uma generalização de sobreposição de I/O dentro de um único programa é sobrepor a computação e I/O de diversos programas.

Multiprogramação

- É a execução concorrente de diversos **processos** independentes sobre um processador.

Time-Slicing

- Fatia de tempo.
- Compartilhar o processador entre diversas computações de processos.
- Ao contrário do que um processo esperar para o término de uma operação de I/O, o processador é compartilhado através de um hardware (timer) usado para interromper uma computação de um processo em intervalos pre-determinados.

Time-Slicing

- Um programa do SO chamado ***Scheduler*** é executado para determinar qual processo deve ser permitido executar no próximo intervalo.

- O ***Scheduler*** pode levar em consideração, prioridades dos processos.

Interactive Time-Sharing Systems

- Usam **multiprogramação com time-sliced**, para dar a um grupo de usuários a ilusão que cada um tem acesso a um computador dedicado.

Multi-Tasking

- Resolvendo um problema por decomposição, dentro de diversos processos concorrentes.

- A execução de diversos aplicativos (programas) por um único usuário, eu uma máquina de um único processador.

Correção de um programa concorrente

- Por causa das possíveis **interações entre os processos** que compreendem um programa concorrente é difícil escrever um programa concorrente correto.

- Para interagirem, processos precisam se **sincronizar** e se **comunicar** diretamente ou não.

Dois processos incrementando uma variável

```
N:  Integer  := 0;

Process P1 is
  begin
    N := N + 1;
  end P1;

Process P2 is
   begin
     N := N + 1;
   end P2;
```

Aplicando a abstração

- Se o compilador traduzir as declarações de alto nível em instruções INC, qualquer intercalação das sequências de instruções dos dois processos darão o mesmo valor.

Exemplo: Computação com a instrução INC

Processo	Instrução	Valor de N
Inicialmente		0
P1	INC N	1
P2	INC N	2

Processo	Instrução	Valor de N
Inicialmente		0
P2	INC N	1
P1	INC N	2

Computação em Registradores

- Por outro lado, se toda computação é feita em registradores, o código compilado pareceria como:

Processo	Instrução	N	Reg (P1)	Reg(P2)
Inicialmente		0		
P1	LOAD Reg, N	0	0	
P2	LOAD Reg, N	0	0	0
P1	ADD Reg, #1	0	1	0
P2	ADD Reg, #1	0	1	1
P1	STORE Reg, N	1	1	1
P2	STORE Reg, N	1	1	1

Resultado

- A figura anterior mostra que algumas intercalações dão resposta errada.

- Então, é extremamente importante **definir exatamente quais instruções são para ser intercaladas**, de forma de o programa concorrente seja correto em sua execução.

Correção de um programa concorrente

- **Programação concorrente** pode **expressar a concorrência** requerida, provendo instruções de programação para a **sincronização** e **comunicação** entre processos.

Ferramentas de Correção

- Um programador pode ser totalmente confundido pelo comportamento que um programa concorrente pode exibir.

- Ferramentas são necessárias para **especificar**, **programar** e **verificar** propriedades desses programas.

Programação Concorrente

- Estuda a abstração que é usada sobre as sequências de instruções atômicas de execução intercalada.

- Define o que significa um programa concorrente ser correto e introduz os métodos usados para provar correção.

Programação Concorrente

- Trata as primitivas e as estruturas de programação concorrente clássicas:
  - Semáforos
  - Monitores
  - Threads

Threads

- A unidade de processamento concorrente mais atual, devido a capacidade de processamento dos processadores ter aumentado.

- O que vamos realmente executar !!!

Processos

- Definição
  - Um trecho de código em execução em uma máquina.
  - Identificado pelo seu PID (Process Identifier).
  - **É a unidade de processamento concorrente em que um processador sob um SO pode processar.**
  - **Unix** (anos 70) foi construído para processar processos.
  - **Solaris** (anos 90) também utiliza processos.

Escalonamento de Processos

- Todo SO tem um programa chamado **Scheduler** (o escalonador do SO) que seleciona, num dado instante, o processo que deve ser executado pelo processador, alternando este entre esses processos.

Escalonamento

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

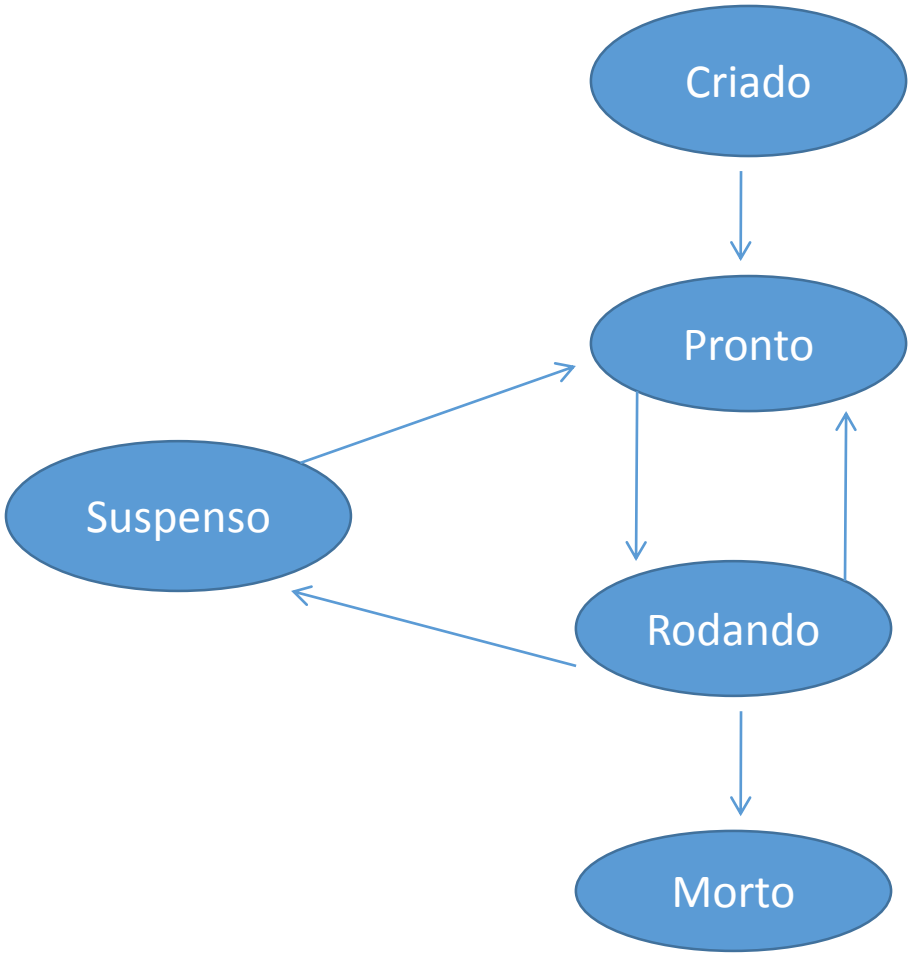
Time-Slicing

- Fracionamento de tempo do processador.
- 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 **scheduler** buscar um outro processo para executar.

Mudança de Estado de um Processo

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

Ciclo de Vida 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.

Estados de Suspensão de Processos

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

Escalonamento Pre-Emptivo

- Resolve as **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.

Threads

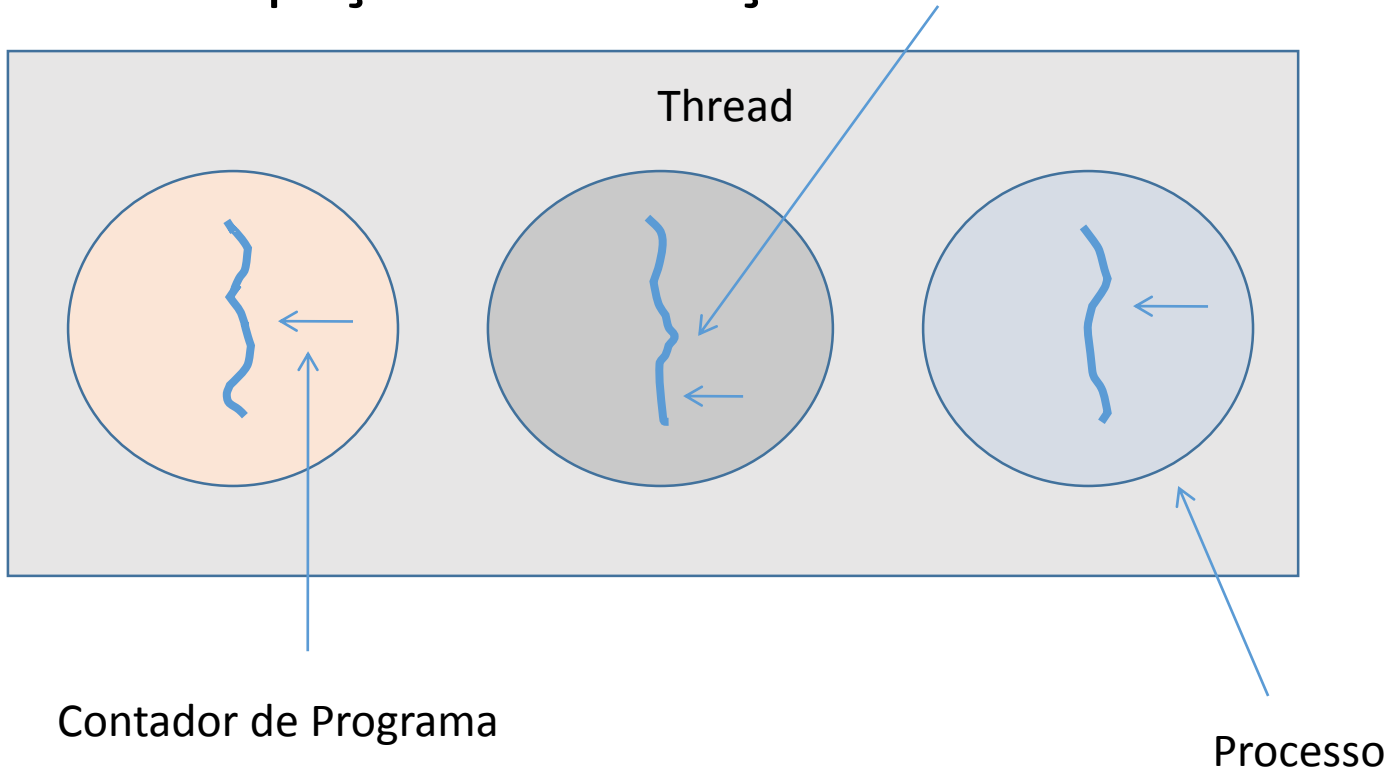
- Definição:
  - Threads (linhas de execução) são atividades (tarefas) concorrentes executadas por um processo.
  - Um processo pode ter uma ou mais threads.
  - Threads pertencentes a um mesmo processo
    compartilham recursos e memória do processo.

Multithreading

- Suporte a Threads:
  - Threads nativas do SO.
  - Suporte de programação multi-thread.
    Exemplo: JVM do Java
  - Linguagem de programação multi-threaded.
    Exemplo: Java

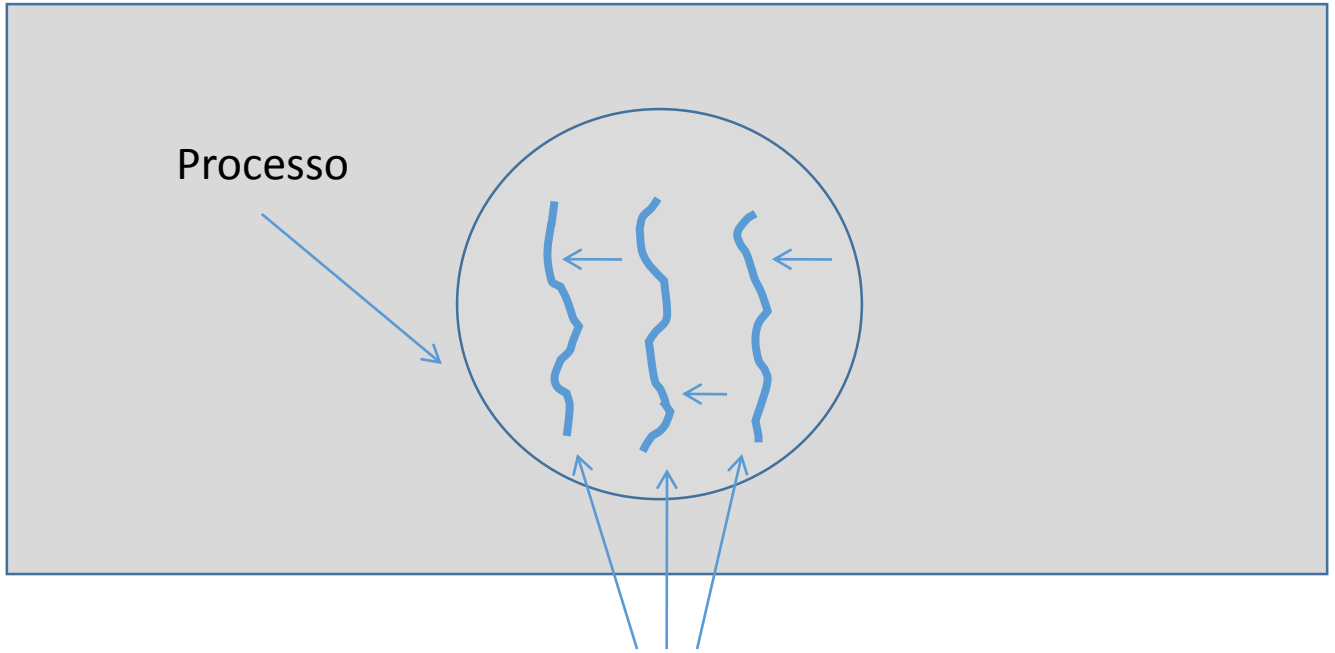
Três Processos cada um com uma Thread

Cada thread tem seu espaço de endereçamento.



Um Processo com três Threads

- Todas num mesmo espaço de endereçamento.



Processos e Threads

- Um confeiteiro.
- O confeiteiro é o **processador**.
- Um Livro de Receitas é um processo.
- Uma receita de bolo.
- Fazer um bolo de aniversário.
- Ingredientes: farinha, ovos, açucar, ...
- Os ingredientes são os **dados**.

Diferença entre Programa e Processo

- O **processo** **é a atividade** que consiste em nosso confeiteiro **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 confeiterio registra onde estava na receita
  (o estado e o contexto do processo são salvos).

Alternando para outro processo

- Confeiteiro procura um livro de pronto-socorro (outro processo).
- Segue a orientações do livro.
- O Confeiteiro alterna do processo (Livro de Receitas pra Fazer o Bolo) para outro, de prioridade mais alta (Administrar cuidado Médico), cada um sendo um processo diferente (receita e livro).

Processo é uma atividade

- Quando a picada for tratada, o confeiteiro volta a fazer o seu bolo, continuando do ponto onde parou, quando abandonou o processo (Fazer o Bolo).

- A ideia é que **processo** é um tipo de **atividade**.

- E em cada atividade para “Fazer o Bolo” ou “Atendimento Médico” existem outras tarefas:, que correspondem às Threads.

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 de Processos com Múltiplas Threads

- Navegador Web
  - Thread de comunicação com a rede.
  - Thread de interação com o usuário navegando.
  - … … …

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 aloca tempo e 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 | Inter-Threads
- **Suporte em S.O.'s:**
  Quase todos | Os mais atuais
- **Suporte em Linguagem de Programação:**
  Quase todas | As mais recentes

Chamadas do Sistema Operacional UNIX

- Criar e executa um processo:
  - **fork()** cria uma cópia do processo atual.
  - **exec()** carrega o código do processo para execução.

Chamadas do Sistema Operacional Unix

- Suspender 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 no Shell.
- Processos podem ser colocados para executar em *background* quando seguidos de um **&**.
- Os processos em execução são listados com o comando **ps –ef** .
- Processos são destruídos com **kill -9 <pid>** .

Chamadas de sistema no Windows

- Criar um Processo:
  - **CreateProcess(<nome>, <comando>, ...)** ou
  - **CreateProcessAsUser(<usuário>,<nome>, ...)**

- Obter o Identificador do Processo:
  - **GetCurrentProcessId()**

- Suspender a Execução:
  - **Sleep(<tempo>)**

- Finalizar o Processo:
  - **ExitProcess(<codigo-retorno>)**

- Destruir um Processo:
  - **TerminateProcess(<pid>, <retorno>)**