

Engenharia de Computação



Arquitetura de Sistemas Operacionais

Gerência de Processos

Prof. Anderson Luiz Fernandes Perez

Prof. Martín Vigil

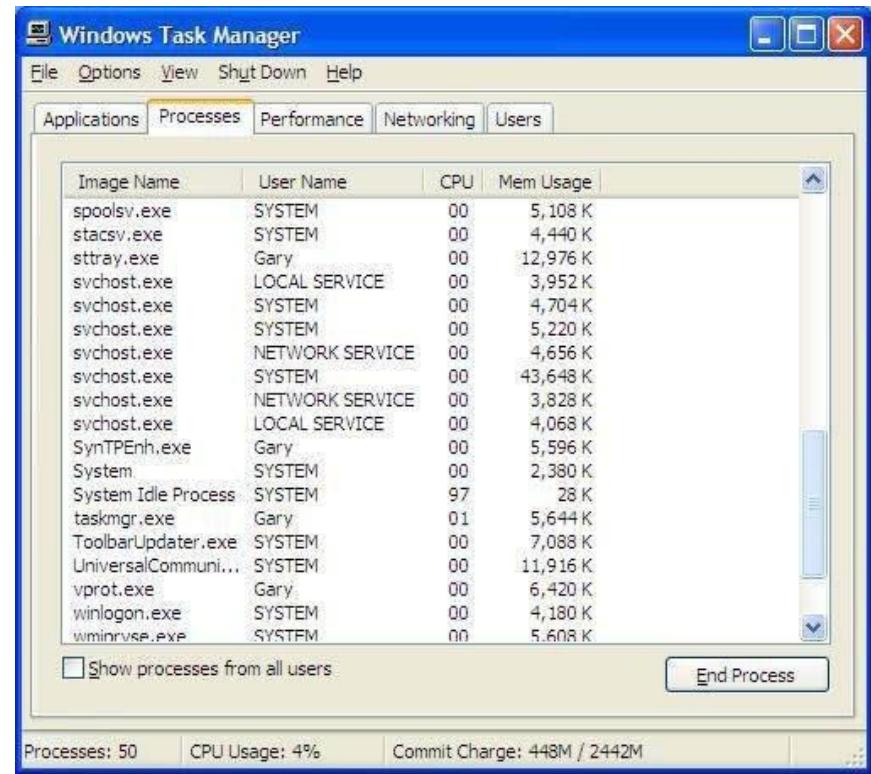
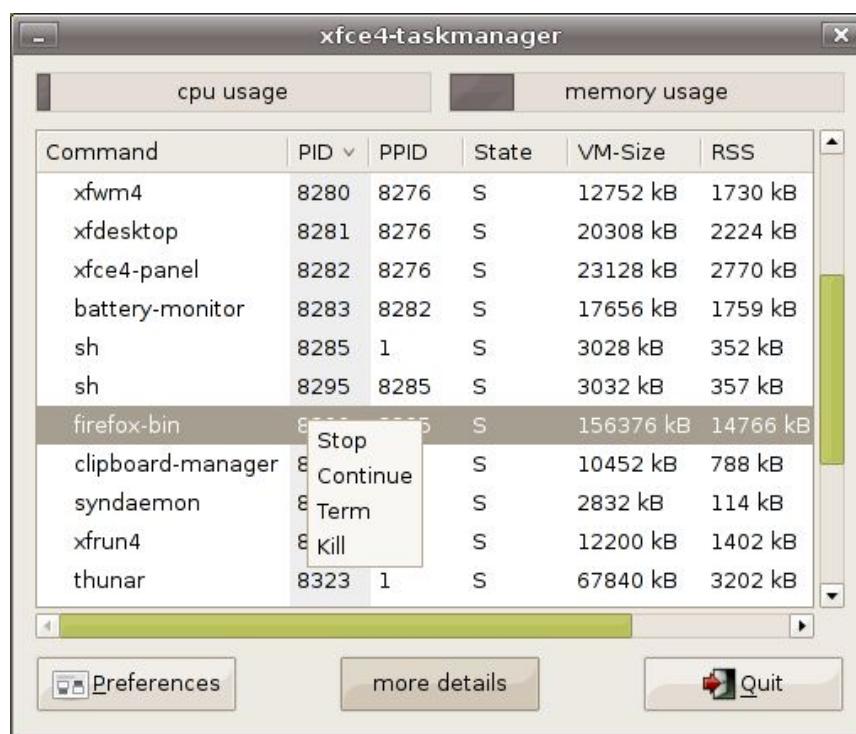
Universidade Federal de Santa Catarina

Campus Araranguá

Processos

Definição de Processo

- Um processo é uma entidade ativa no sistema e representa um programa em execução.



Definição de Processo

- Um processo é representado no SO por uma estrutura de dados chamada PCB (*Process Control Block/Bloco de Controle do Processo*).
- A PCB possui informações sobre o contexto de execução do processo.

Definição de Processo

- Informações Contidas na PCB (*Process Control Block*)
 - Identificação do processo;
 - Estado de execução;
 - Dados do contador de programas (*Program Counter*);
 - Dados dos registradores da CPU;
 - Informações para o escalonamento de CPU, por exemplo prioridade;
 - Informações sobre memória;
 - Informações de contabilidade;
 - Informações sobre as operações de E/S (status das informações de E/S).

Definição de Processo

- Informações Contidas na PCB (*Process Control Block*)



Definição de Processo

- Criação de Processos
 - Cada SO possui chamadas de sistema responsáveis pela criação e destruição de processos.
 - O Unix possui a chamada de sistema *fork()* que quando executada cria uma cópia idêntica ao processo que chamou a execução de *fork()*.
 - O novo processo, criado a partir da chamada de sistema *fork()*, é chamado de processo filho e tem sua própria PCB.

Definição de Processo

- Criação de Processos
 - A sintaxe da chamada de sistema `fork()` é:
 - *pid_t fork(void)*
 - Possíveis valores de retorno:
 - < 0 indica erro;
 - > 0 é retornado ao processo pai, o processo que invocou a execução de `fork()`, e indica a identificação do novo processo criado;
 - $= 0$ é o valor retornado para o processo filho, novo processo criado.
 - A chamada de sistema *fork()* está implementada na biblioteca *unistd.h*.

Definição de Processo

- Exemplo de Criação de Processos em C (UNIX e LINUX) **uso de *fork()***

```
1. #include <stdio.h>
2. #include <unistd.h>
3. #include <stdlib.h>
4. int main()
5. {
6.     int r;
7.     r = fork(); // cria processo
8.     if (r > 0) {
9.         printf("Eu sou o processo pai de %d\n", r);
10.    }
12.    else if (r == 0) {
13.        printf("Processo filho. ID. eh: %d\n", getpid());
14.    }
15.    else {
16.        printf("Erro ao tentar criar processo filho\n");
17.        exit(0);
18.    }
19.    return 0;
20. }
```

Definição de Processo

- Criação de Processos
 - Um processo criado a partir da chamada de sistema *fork()* pode executar outro fluxo de execução.
 - A família de chamadas de sistema *exec* permite que um processo execute um fluxo de execução diferente do fluxo do processo pai.
 - A chamada de sistema *exec()* modifica a imagem do processo que está invocando a função.
 - Os segmentos de pilha, código, dados, heap são sobre-escritos pelo novo programa.

Definição de Processo

- Criação de Processos
 - Chamada de sistema exec – **biblioteca unistd.h**
 - *int execl(char const *path, char const *arg0, ...)*
 - *int execle(char const *path, char const *arg0, ..., char const *envp);*
 - *int execlp(char const *file, char const *arg0, ...);*
 - *int execv(char const *path, char const *argv[]);*
 - *int execve(char const *path, char const *argv[], char const *envp[]);*
 - *int execvp(char const *file, char const *argv[]);*

Definição de Processo

- Criação de Processos
 - Chamada de sistema *exec*
 - Convenção para a chamada de sistema *exec*:
 - **execxy**
 - *Para x:*
 - » *l* – lista de argumentos terminada por NULL;
 - » *v* – vetor de argumentos.
 - *Para y:*
 - » *p* – a variável de ambiente (*PATH*) será buscada;
 - » *e* – um vetor de variável de ambiente deverá ser fornecido.

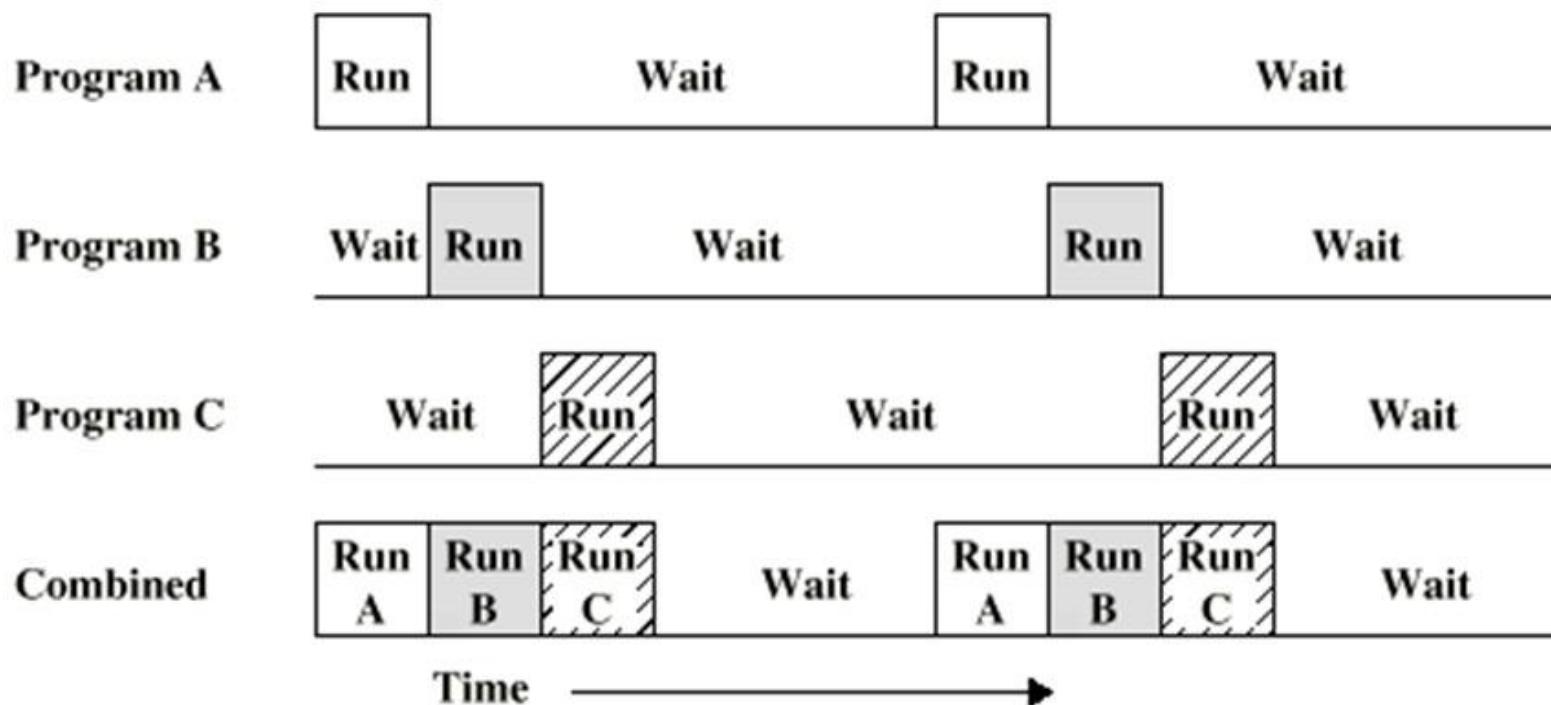
Definição de Processo

- Chamadas de Sistema Relacionadas a Processos (**ambiente Unix e Linux**)
 - *fork()* – cria um novo processo;
 - *exit()* – força a destruição do processo;
 - *getpid()* – retorna o ID do processo;
 - *getppid()* – retorna o ID do pai do processo;
 - *wait()* – aguarda a execução do processo filho;
 - *sleep()* – aguarda um tempo em segundos.

Gestão de Processador

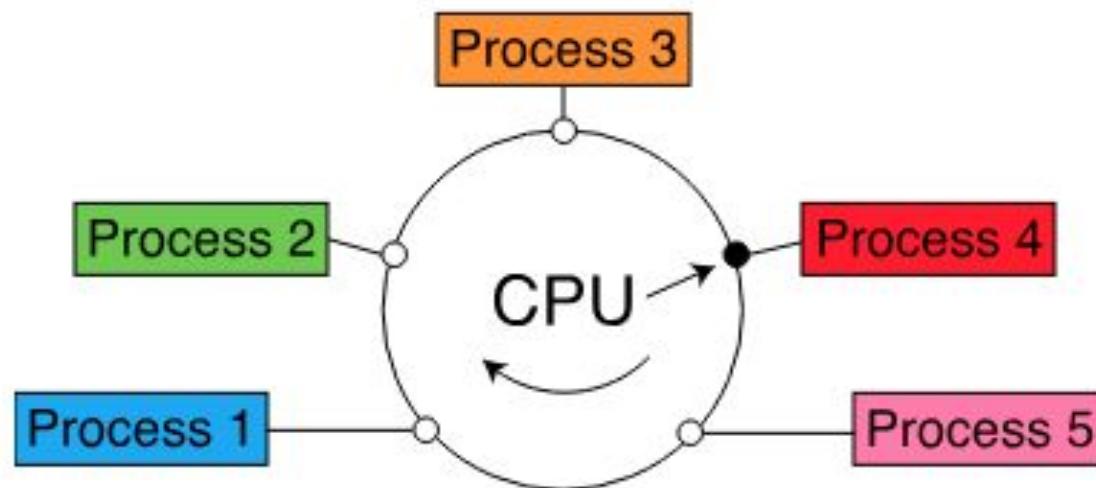
Multiprogramação

- Técnica onde mantêm-se diferentes processos na memória e intercala-se a CPU entre eles para **evitar CPU ficar ociosa**.



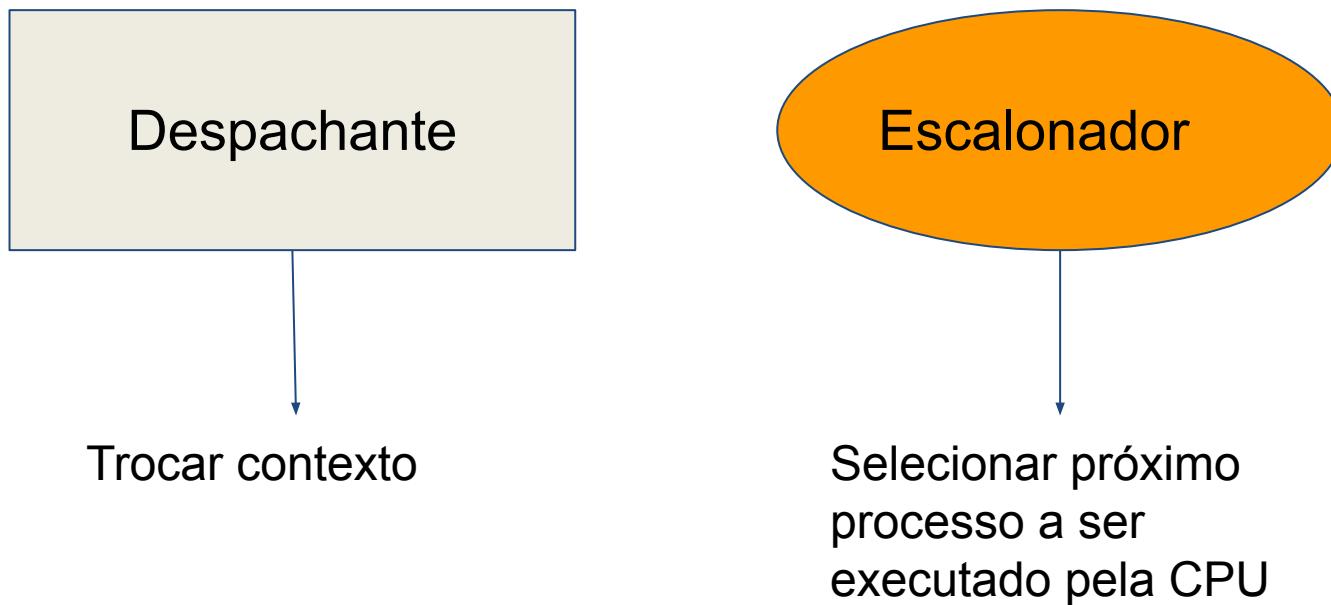
Multitarefa (*timesharing*)

- Caso especial multiprogramação para **reduzir tempo de resposta** para usuários



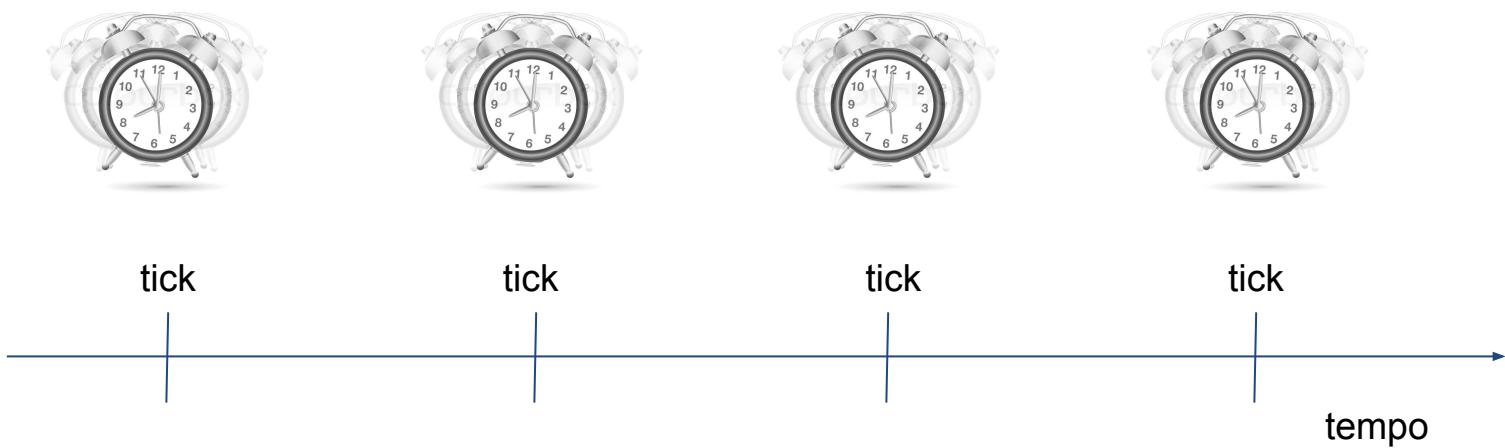
Troca de Contexto

- A **troca de um processo em execução** (ou seja que está com a posse do processador) **por outro**, é definida como **troca de contexto**.
- Os **dados de contexto** de um processo são armazenados em sua **PCB** (*Process Control Block*).
- A **troca de contexto** é realizada pelo **despachante** (do Inglês *dispatcher*) e a **escolha de qual tarefa** deverá assumir o processador é realizada pelo **algoritmo de escalonamento de tarefas**.



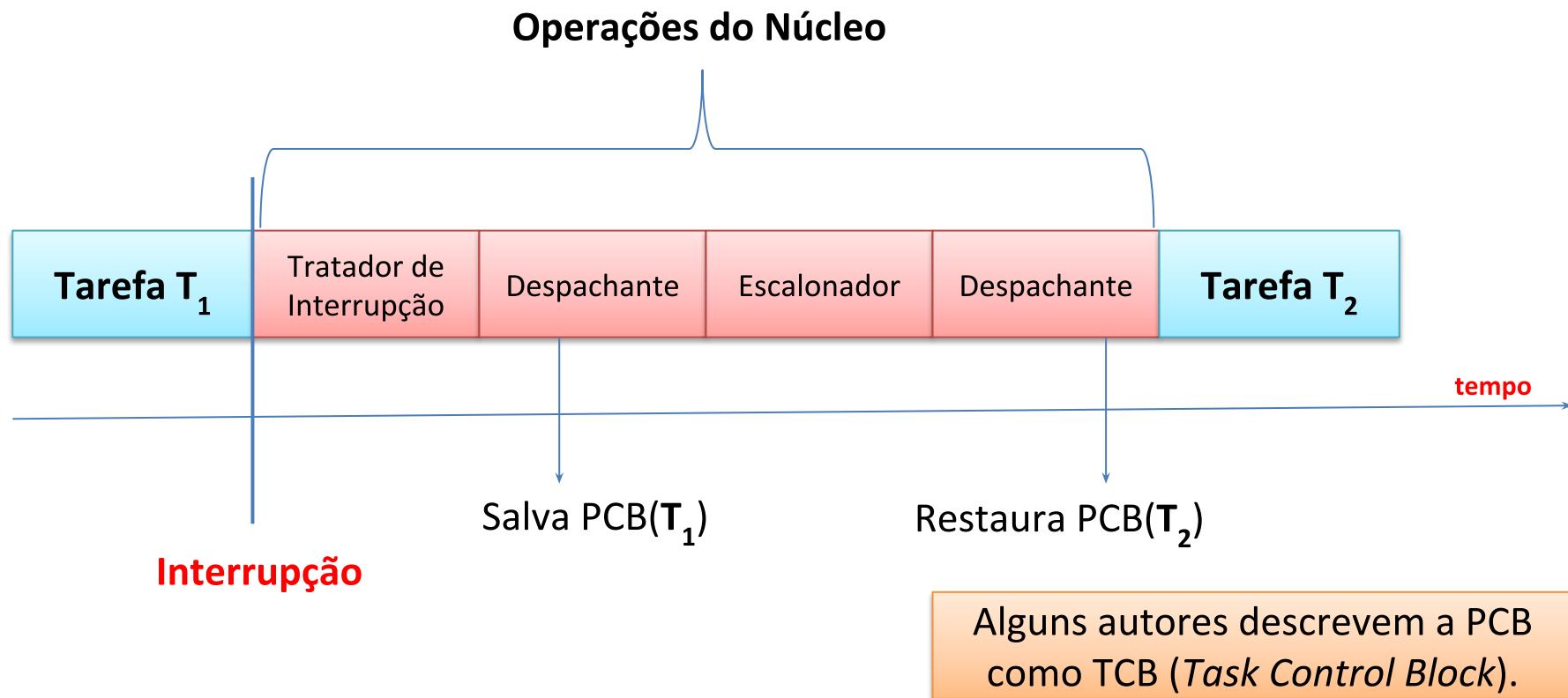
Interrupções

- Eventos que interrompem a CPU
- Interrupção de relógio (clock ticks)



Troca de Contexto

- Passos de uma Troca de Contexto



Troca de Contexto

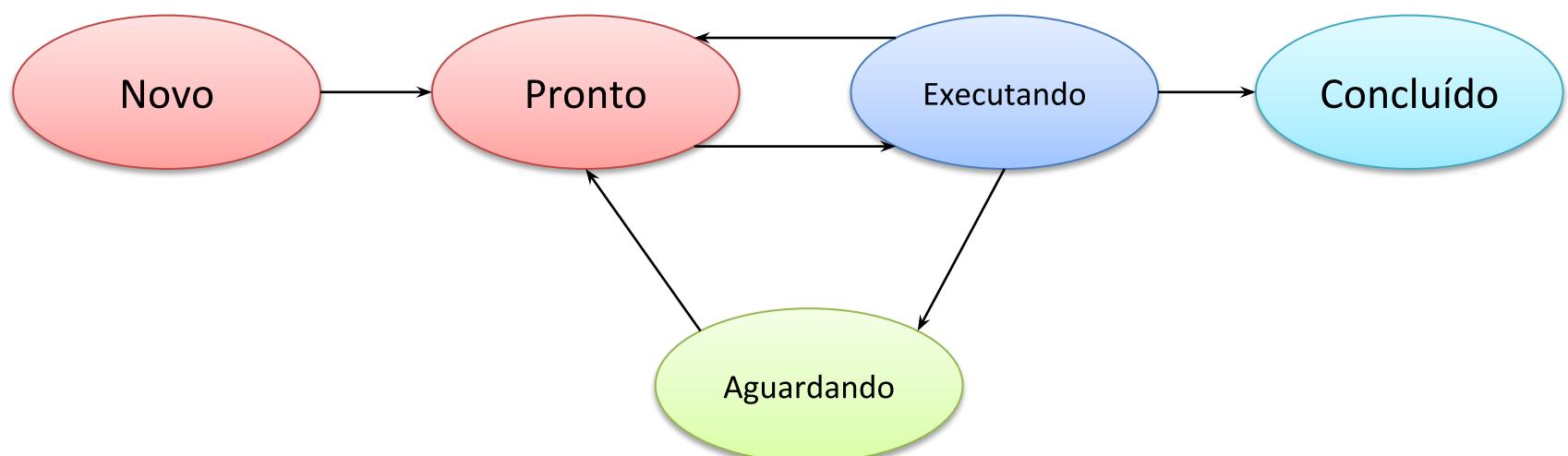
- Estados de um processo
 - Um processo pode estar em um dos **seguientes estados durante sua execução:**
 - **Novo** – acabou de ser criado;
 - **Pronto** – está pronto para ser executado. Está na fila de processos aptos;
 - **Executando** – está em execução;
 - **Aguardando** – está aguardando um evento;
 - **Concluído** – concluiu sua execução.

Troca de Contexto

- A troca de contexto **pode ocorrer quando:**
 - O tempo de execução do processo atual se esgotou;
 - Ocorrer uma interrupção de hardware;
 - Ocorrer uma interrupção de software (execução de uma chamada de sistema – **conhecida como trap**);
 - Ocorrer um erro de execução.

Troca de Contexto

- Diagrama de Troca de Contexto

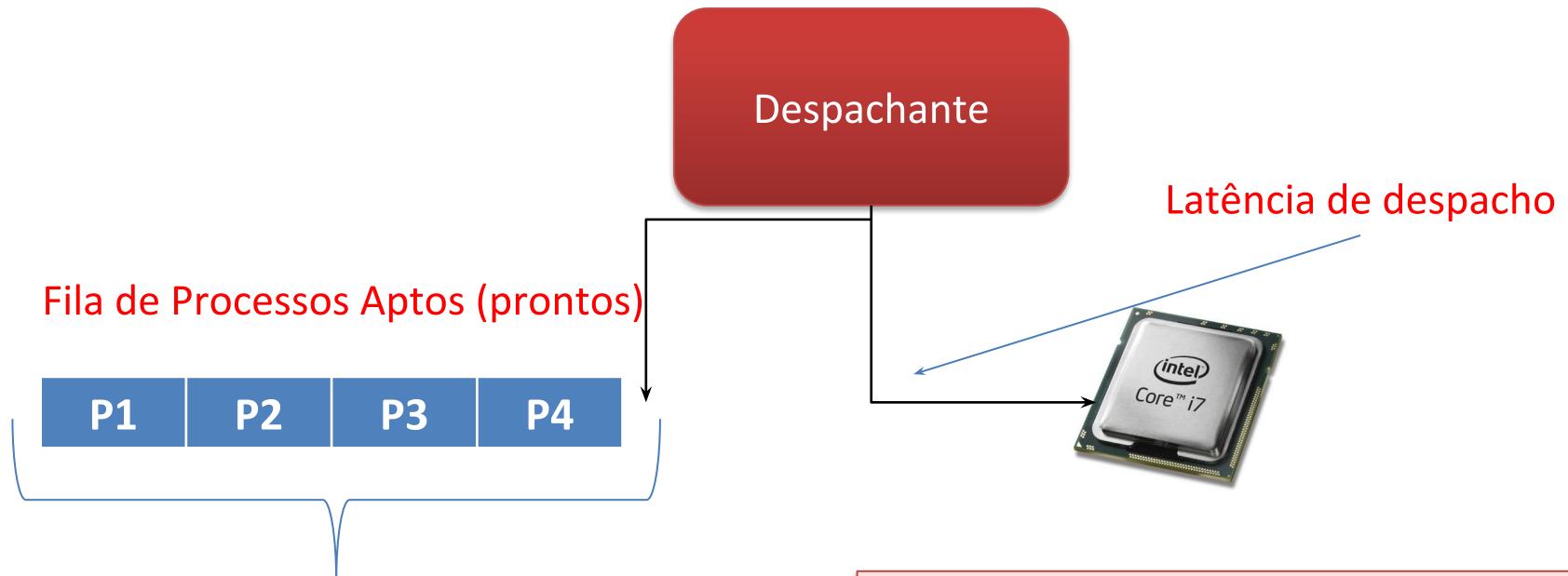


Escalonamento de Processos

- A CPU é um dos principais recursos do computador, desta forma é necessário mantê-la ocupada o maior tempo possível.
- Em um sistema multiprogramado existe a necessidade de selecionar qual processo e por quanto tempo ele ocupará a CPU.
- Processos em execução alternam entre picos de CPU e picos de E/S.

Escalonamento de Processos

- Esquema do Escalonamento de Processos



Política de Escalonamento –
Algoritmo de Escalonamento de
Processos

Observação: a fila de processos aptos não é necessariamente uma estrutura de dados do tipo FILA, pode ser implementada como lista encadeada ou árvore.

Escalonamento de Processos

- Quando Escalonar a CPU?
 - Quando um processo passa do estado de execução para o estado de espera;
 - Quando um processo passa do estado de execução para o estado de pronto;
 - Quando um processo passa do estado de espera para o estado de pronto;
 - Quando um processo termina.

Escalonamento de Processos

- Critérios para a Escolha de um Algoritmo de Escalonamento de Processos
 - **Utilização da CPU**: manter a CPU o máximo do tempo ocupada;
 - **Throughput** (vazão): quantidade de processos que são concluídos por unidade de tempo;
 - **Turnaround**: o intervalo de tempo a partir da criação do processo até a sua conclusão;
 - **Tempo de Espera**: é a soma dos períodos gastos em espera na fila de prontos;
 - **Tempo de Resposta**: tempo entre envio de uma solicitação e a primeira resposta ao usuário.

Escalonamento de Processos

- Tipos de Escalonadores de Processos
 - **Preemptivo:** os escalonadores preemptivos alternam o uso da CPU quando a tarefa esgotou o tempo de execução (*quantum*) ou invocou uma chamada de sistema ou ainda quando uma interrupção é gerada.
 - **Cooperativo:** neste tipo de escalonador a tarefa somente libera a CPU quando a tarefa realiza alguma operação de E/S, conclui sua execução ou libera o processador voluntariamente (*sched_yield*).

Tarefa

Ler Seções 3--3.3, Silberschatz, "Operating System Concepts"

Algoritmos de escalonamento

Algoritmos de escalonamento para sistemas de lote

- Sistemas de lote atuais:
 - Transações bancárias
 - Relatórios periódicos
- Algoritmos clássicos
 - First-Come, First-Served
 - Shortest-Job-First
- Algoritmos de tempo real

Escalonamento de Processos

- Algoritmo FCFS (*First-Come, First-Served*)
 - Nesse algoritmo o processo que solicita a CPU primeiro será o primeiro a ser atendido (executado).
 - Utiliza a estrutura de dados fila (FIFO).
 - As PCBs dos processos criados são inseridas no final da fila e a PCB que está na cabeça da fila indica o próximo processo a utilizar a CPU.
 - Esse algoritmo é não preemptivo.
 - O tempo médio de espera do FCFS **pode** ser longo.

Escalonamento de Processos: Algoritmo FCFS

- Exemplo: processo aptos

ID	Tempo de Criação (chegada)	Tempo de CPU
P1	0	24
P2	0	3
P3	0	3

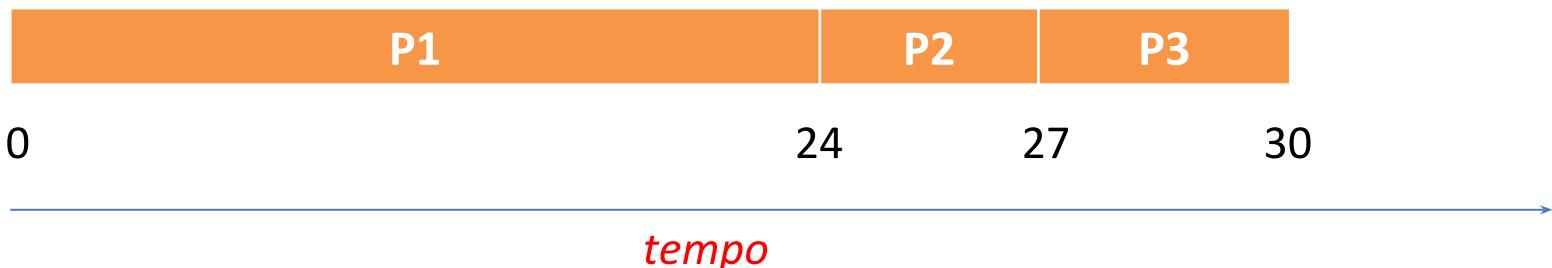
Diagrama de Gantt



Escalonamento de Processos:

Algoritmo FCFS

- Ordem de chegada: P1, P2, P3
- Tempo de Espera:
 - *Processo P1: 0*
 - *Processo P2: 24*
 - *Processo P3: 27*
 - Tempo total de espera: **51 milissegundos**
 - Tempo médio de espera: **$51 / 3 = 17$ milissegundos**



Escalonamento de Processos: Algoritmo FCFS

- Nova ordem de chegada foi P2, P3 e P1.
- Tempo de Espera:
 - *Processo P1: 6*
 - *Processo P2: 0*
 - *Processo P3: 3*
 - Tempo total de espera: **9 milissegundos**
 - Tempo médio de espera: **$9 / 3 = 3$ milissegundos**



Escalonamento de Processos: Algoritmo FCFS

- Vantagens
 - Simplicidade
 - Todos os processos serão executados
- Desvantagens
 - Processo longo atrasa os próximos processos



Escalonamento de Processos

- Algoritmo SJF (*Shortest Job First*) – Job mais curto Primeiro
 - Associa a cada processo o intervalo do próximo pico de CPU do processo (tempo necess. de CPU)
 - Quando a CPU está disponível ela é atribuída ao processo que tem o próximo pico de CPU mais curto.
 - Quanto dois processos tiverem o próximo pico de CPU iguais a política a ser utilizada é a FCFS.

Escalonamento de Processos:

Algoritmo SJF

- Os processos chegam **ao mesmo momento**
- A ordem de execução será: P4, P1, P3 e P2

ID	Tempo de Criação (chegada)	Tempo de CPU
P1	0	6
P2	0	8
P3	0	7
P4	0	3



Escalonamento de Processos: Algoritmo SJF

- Tempo de Espera:
 - *Processo P1*: 3
 - *Processo P2*: 16
 - *Processo P3*: 9
 - *Processo P4*: 0
- Tempo total de espera: **34 milissegundos**
- Tempo médio de espera: **$28 / 4 = 7$ milissegundos**



Escalonamento de Processos

- Algoritmo SJF (*Shortest Job First*) – Job mais curto Primeiro
 - O algoritmo SJF poderá ser implementado **com** ou **sem** preempção.
 - Com preempção: processo com menor tempo de CPU **restante** é o próximo a ser escalonado

Escalonamento de Processos: SJF Preemptivo

ID	Tempo de Criação (unid. de tempo)	Tempo de CPU (unid. de tempo)
P1	0	8
P2	1	4
P3	2	9
P4	3	5



Escalonamento de Processos: SJF Preemptivo

- Tempo de Espera:
 - Processo P1: $10 - 1 = 9$
 - Processo P2: $1 - 1 = 0$
 - Processo P3: $17 - 2 = 15$
 - Processo P4: $5 - 3 = 2$
 - Tempo total de espera: **26 milissegundos**
 - Tempo médio de espera: **$26 / 4 = 6,5$ milissegundos**



Escalonamento de Processos:

Algoritmo SJF

- Vantagens
 - Melhor throughput
- Desvantagens
 - Prever tempo de CPU de um processo é difícil
 - Processos longos podem nunca ser executados (starvation)

Escalonamento de Processos

- Algoritmo Baseado em Prioridades
 - A cada processo é associado um número que indica a prioridade.
 - Processos com prioridade maior têm preferência de execução sobre os demais.
 - A prioridade pode ser definida da seguinte forma:
 - Número menor – prioridade maior
 - Número maior – prioridade menor
 - Ou
 - Número maior – prioridade maior
 - Número menor – prioridade menor
 - Geralmente se utiliza números baixos (menores) para indicar maior prioridade.
 - Esse algoritmo pode ser preemptivo ou não preemptivo.

Algoritmo baseado em prioridades (sem preempção)

ID	Tempo de Criação (chegada)	Prioridade	Tempo de CPU
P1	0	3	10
P2	0	1	1
P3	0	4	2
P4	0	5	1
P5	0	2	5



Algoritmo baseado em prioridades (sem preempção)

- Tempo de Espera:
 - Processo P1: 6
 - Processo P2: 0
 - Processo P3: 16
 - Processo P4: 18
 - Processo P5: 1
- Tempo total de espera: **41 milissegundos**
- Tempo médio de espera: **$41 / 5 = 8,2$ milissegundos**



Algoritmo baseado em prioridades (com preempção)

ID	Tempo de Criação (chegada)	Prioridade	Tempo de CPU
P1	0	3	10
P2	1	1	1
P3	2	4	2
P4	4	5	1
P5	5	2	5



Algoritmo baseado em prioridades (com preempção)

- Tempo de Espera:

- Processo P1: $2 - 1 + 10 - 5 = 6$
 - Processo P2: $1 - 1 = 0$
 - Processo P3: $16 - 2 = 14$
 - Processo P4: $18 - 4 = 14$
 - Processo P5: $5 - 5 = 0$
-
- Tempo total de espera: **34 milissegundos**
 - Tempo médio de espera: **$34 / 5 = 6,8$ milissegundos**



Escalonamento de Processos

- Algoritmo Baseado em Prioridades
 - Um algoritmo de escalonamento de processos baseado em prioridades estáticas pode levar alguns processos ao bloqueio indefinido ou inanição.
 - O bloqueio indefinido acontece quando existe uma alta atividade de processos de com prioridade maior, consequentemente os processos com prioridade menor ficam indefinidamente aguardando para serem executados.
 - Uma solução para o problema do bloqueio indefinido é usar a técnica de envelhecimento de prioridade.

Escalonamento de Processos

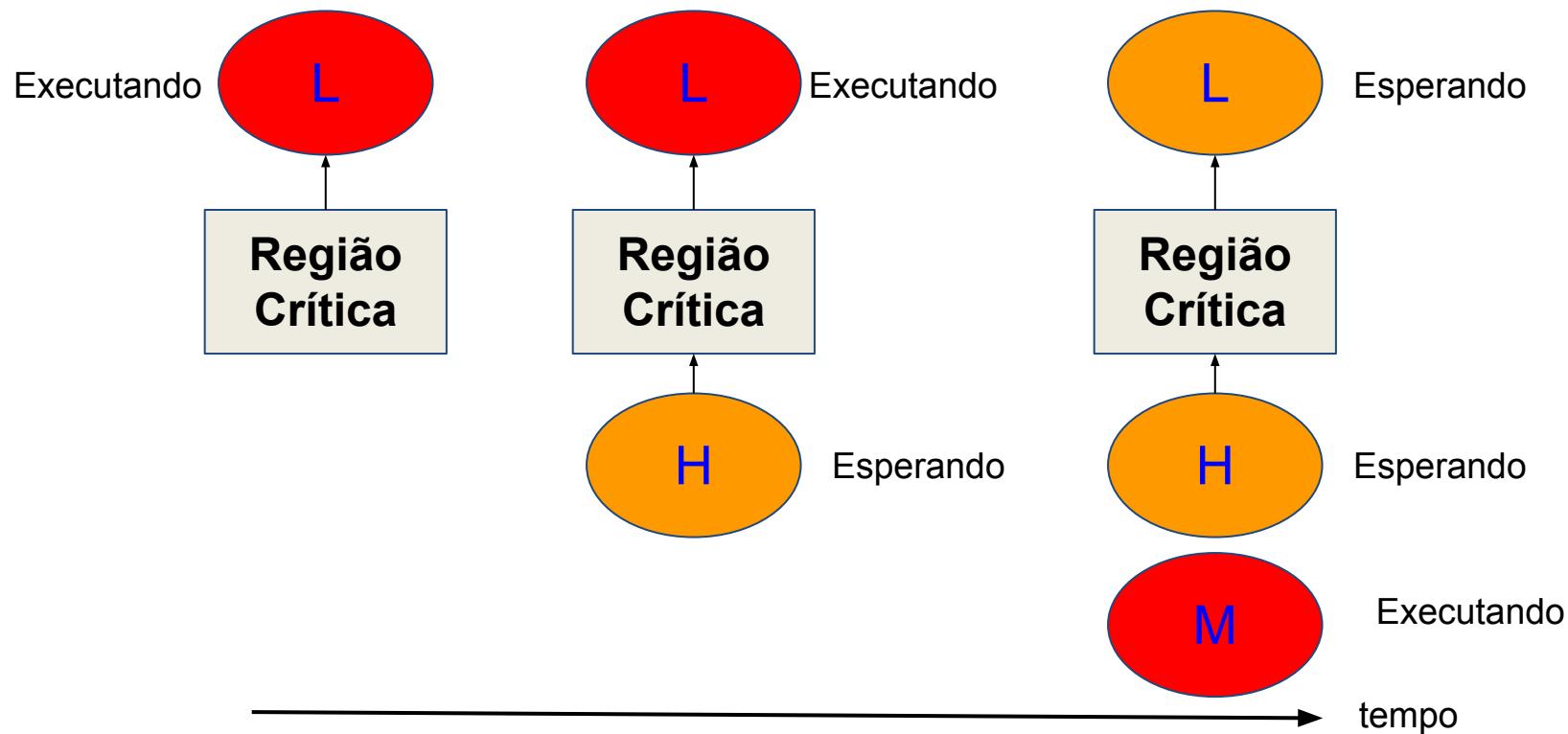
- Algoritmo Baseado em Prioridades
 - Curiosidade



Quando o IBM 7094 foi desligado no MIT em 1973, foi encontrado um processo de baixa prioridade que tinha sido submetido em 1967 e ainda não tinha sido executado.

Escalonamento de Processos

- Problema da Inversão de Prioridade
 - Processos H, M e L, com priorid. $P(H) > P(M) > P(L)$
 - Região crítica com acesso mutualmente exclusivo



Escalonamento de Processos

- Exemplo real de Inversão de Prioridades
 - O projeto Mars Pathfinder tinha o objetivo de enviar a Marte uma sonda para pesquisar a atmosfera e outros fatores daquele planeta.
 - A sonda espacial Mars Pathfinder enviada para Marte em 1997 pela NASA tinha um sistema de controle com 97 threads com vários níveis de prioridade.
 - O sistema de controle da Pathfinder executava sobre o sistema operacional VxWorks.

Escalonamento de Processos

- Solução para a Inversão de Prioridades
 - Utilizar no máximo duas prioridades
 - Herança de prioridade
 - $P(L) = P(H)$ apenas quando H estiver esperando por L
 - Evita M interromper L

Escalonamento de Processos

- Algoritmo *Round-Robin* (**revezamento**)
 - Este algoritmo foi projetado para sistemas de tempo compartilhado.
 - É parecido com o algoritmo FCFS mas com preempção baseada em um tempo conhecido como *quantum*.
 - Geralmente um *quantum* de tempo tem duração de 10 a 100 milissegundos.
 - A fila de aptos é implementada como uma fila circular, sendo que novos processos são adicionados ao final da fila.

Escalonamento de Processos

- Algoritmo *Round-Robin* (**revezamento**)
 - Exemplo: quantum de 4 s

ID	Tempo de Criação (chegada)	Tempo de CPU
P1	0	24
P2	0	3
P3	0	3



Escalonamento de Processos: Algoritmo Round-Robin

- Tempo de Espera:
 - Processo P1: $10 - 4 = 6$
 - Processo P2: $4 - 0 = 4$
 - Processo P3: $7 - 0 = 7$
 - Tempo total de espera: **17 milissegundos**
 - Tempo médio de espera: **$17 / 3 = 5,66$ milissegundos**



Escalonamento de Processos: Algoritmo Round-Robin

- Tempo de Quantum vs. Tempo de troca vs.
Tempo de resposta

Escalonamento de Processos

- Algoritmo de Múltiplas Filas
 - Esse algoritmo classifica os processos em diferentes grupos.
 - Para cada grupo de processos é criada uma fila de aptos.
 - Os processos são atribuídos permanentemente em uma das filas de prontos, com base em alguma propriedade do processo.
 - Cada fila tem seu próprio algoritmo de escalonamento.

Escalonamento de Processos

- Algoritmo de Múltiplas Filas

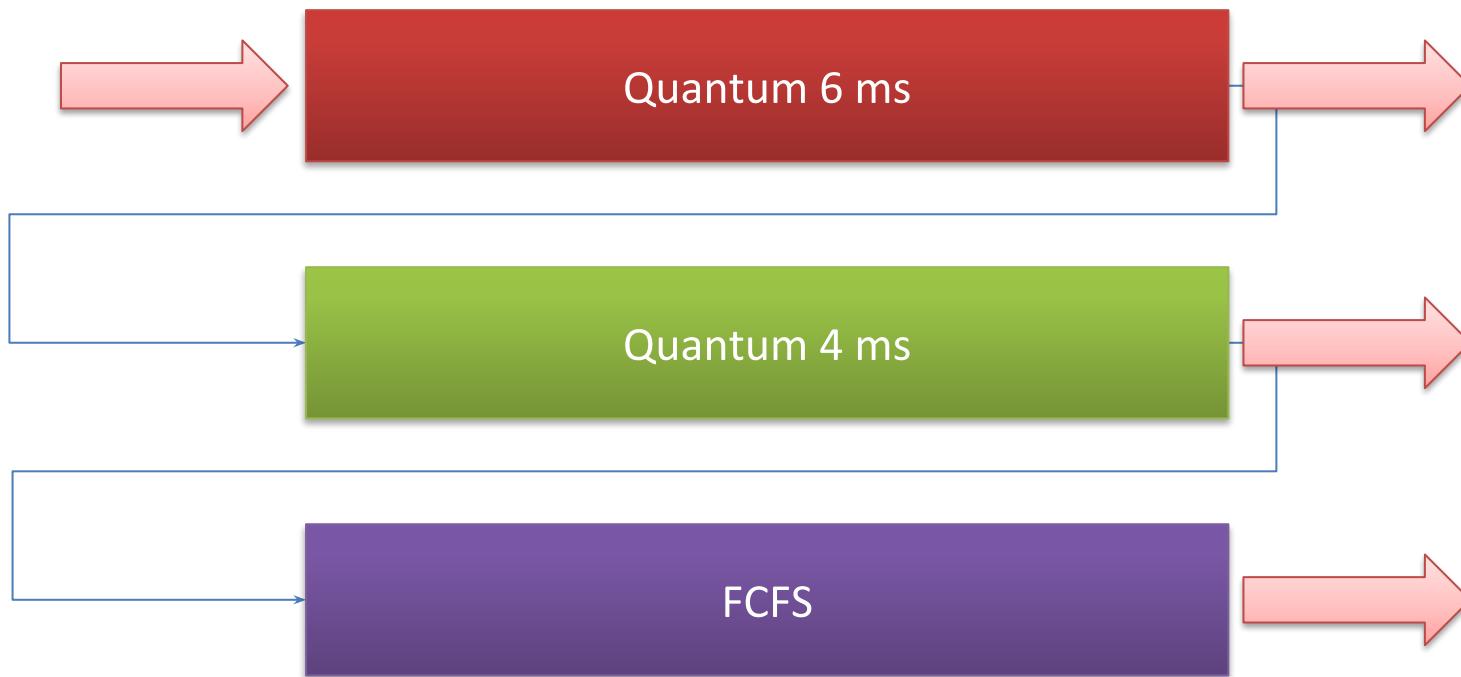


Escalonamento de Processos

- Algoritmo de Múltiplas Filas com Realimentação
 - Várias filas de prontos são criadas de acordo com as características do processos.
 - Um processo colocado em uma fila pode ser deslocado para outra em outro instante de execução.
 - Cada fila tem seu próprio algoritmo de escalonamento.

Escalonamento de Processos

- Algoritmo de Múltiplas Filas com Realimentação



Escalonamento de Processos

- Ver manual do comando *nice* no Linux

Escalonamento de Processos

- Escalonamento em Sistemas de Tempo Real
 - Um sistema de tempo real deve produzir resultados dentro de certos limites de tempo (*deadline*).
 - Resultados produzidos após o *deadline* podem ser considerados inúteis.
 - Um sistema de tempo pode ser classificado em dois tipos:
 1. Sistemas de tempo real críticos;
 2. Sistemas de tempo real não críticos.

Escalonamento de Processos

- Escalonamento em Sistemas de Tempo Real
 - Sistema de Tempo Real Crítico
 - Possuem restrições de tempo mais rigorosas, ou seja, atrasos na execução de alguma tarefa **não** são permitidos.
 - Uma determinada tarefa que não cumpre o *deadline* é considerada inútil.
 - Atrasos na execução de alguma tarefa pode levar todo o sistema ao colapso.
 - **Exemplos:**
 - Equipamentos médicos (marca passo);
 - Sistemas automotivos (freios, injeção, controle de estabilidade);
 - Sistemas de controle de tráfego aéreo (radares);
 - Controle de plantas nucleares.

Escalonamento de Processos

- Escalonamento em Sistemas de Tempo Real
 - Sistema de Tempo Real não Crítico
 - As restrições de tempo são mais brandas do que os sistemas de tempo real crítico.
 - Possíveis atrasos podem levar o sistema a degradação, mas não ao colapso total.
 - Exemplos:
 - Sistemas de medição (satélites, microscópio);
 - Transmissão de áudio e vídeo;
 - Jogos (videogames).

Escalonamento de Processos

- Escalonamento em Sistemas de Tempo Real
 - Tipos de Tarefas em um Sistema de Tempo Real
 - **Periódicas:** exigem a CPU em intervalos constantes. Possuem um tempo de processamento fixo t após adquirir a CPU, um *deadline* d durante o qual deverá ser atendido pela CPU e um período p . A relação entre período, deadline e tempo de processamento é: $0 \leq t \leq d \leq p$. As tarefas periódicas são associadas a sistemas de tempo real crítico.

Escalonamento de Processos

- Escalonamento em Sistemas de Tempo Real
 - Tipos de Tarefas em um Sistema de Tempo Real
 - **Aperiódicas ou assíncronas:** a ativação de uma tarefa aperiódica responde a eventos internos ou externos, definindo uma **característica aleatória**. São associadas a sistemas de tempo real brando.
 - **Esporádicas:** correspondem a um subconjunto das tarefas aperiódicas. Possuem como característica a restrição de um **intervalo mínimo conhecido entre duas ativações consecutivas**, por isso podem ter atributos de tarefas críticas. Podem ser associadas a sistemas de tempo real crítico.

Escalonamento de Processos

- Escalonamento em Sistemas de Tempo Real
 - Algoritmo de Taxa Monotônica (*Rate Monotonic - RM*)
 - Define uma política de escalonamento baseada na alocação de prioridade estática com preempção.
 - Cada tarefa recebe uma prioridade inversamente proporcional a seu período, ou seja, **quanto menor o período, mais alta a prioridade e quanto maior o período, menor a prioridade**.
 - Considera que o tempo de processamento de uma tarefa periódica é o mesmo em cada pico de CPU.

RT Monotônico é *Escalonável*?

- m = total de eventos periódicos
- C = tempo de execução máximo do evento
- P = período do evento

$$\sum_{i=1}^m \frac{C_i}{P_i} \leq 2(2^{1/m} - 1)$$

Escalonamento de Processos

- Escalonamento em Sistemas de Tempo Real
 - Algoritmo de Taxa Monotônica (*Rate Monotonic - RM*)
 - Exemplo (*sem perda de deadline*):
 - Processos aptos

ID	Tempo de Criação (chegada)	Tempo de CPU	Período	Deadline
P1	0	20	50	50
P2	0	35	100	100

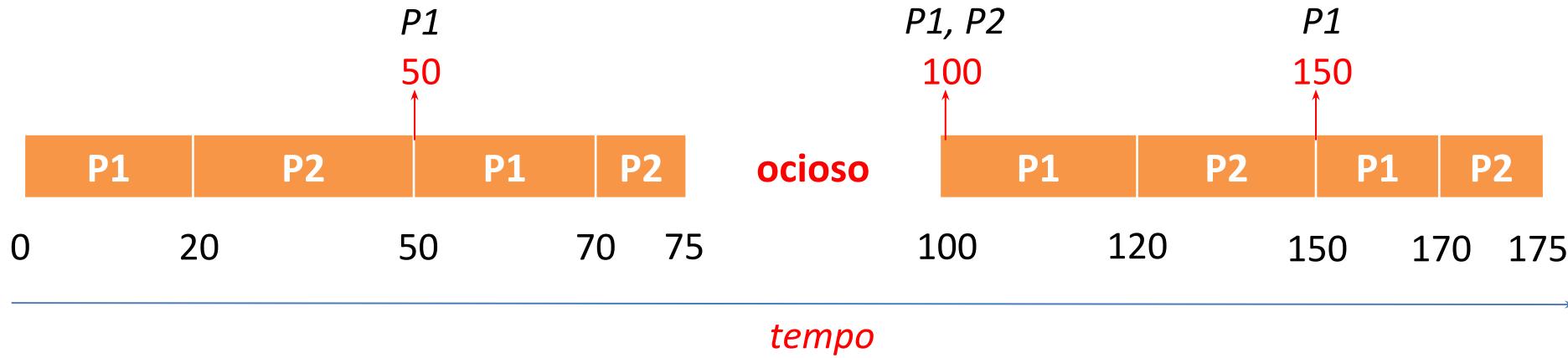
O deadline das tarefas é igual ao
início do próximo período.

Escalonamento de Processos:

Escalonamento em RT

- Algoritmo de Taxa Monotônica (*Rate Monotonic - RM*), Exemplo (*sem perda de deadline*):

ID	Tempo de Criação (chegada)	Tempo de CPU	Período	Deadline
P1	0	20	50	50
P2	0	35	100	100



Escalonamento de Processos

- Escalonamento em Sistemas de Tempo Real
 - Algoritmo de Taxa Monotônica (*Rate Monotonic - RM*)
 - Exemplo (*com perda de deadline*):
 - Processos aptos

ID	Tempo de Criação (chegada)	Tempo de CPU	Período	Deadline
P1	0	25	50	50
P2	0	35	80	80

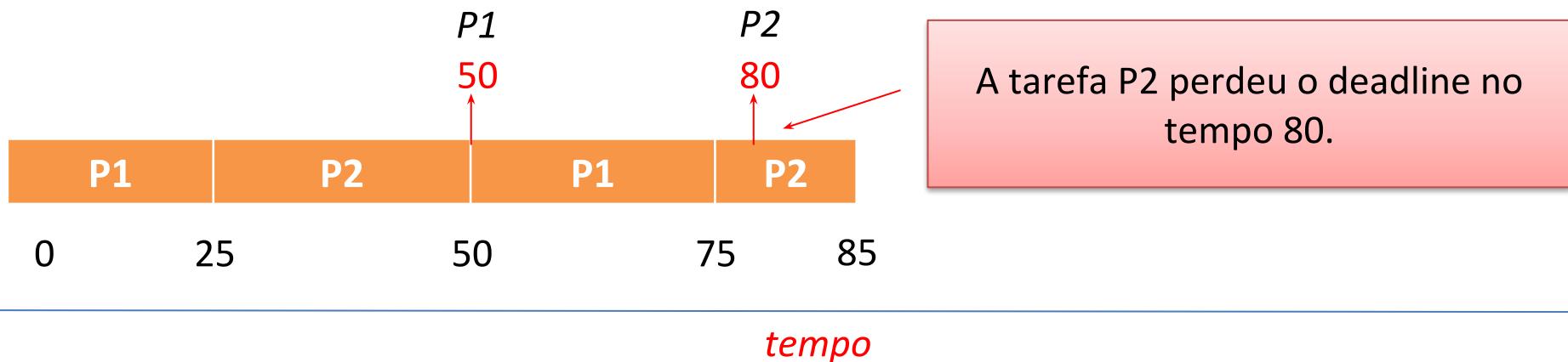
O deadline das tarefas é igual ao
início do próximo período.

Escalonamento de Processos:

Escalonamento em RT

- Algoritmo de Taxa Monotônica (*Rate Monotonic - RM*), Exemplo (*com perda de deadline*):

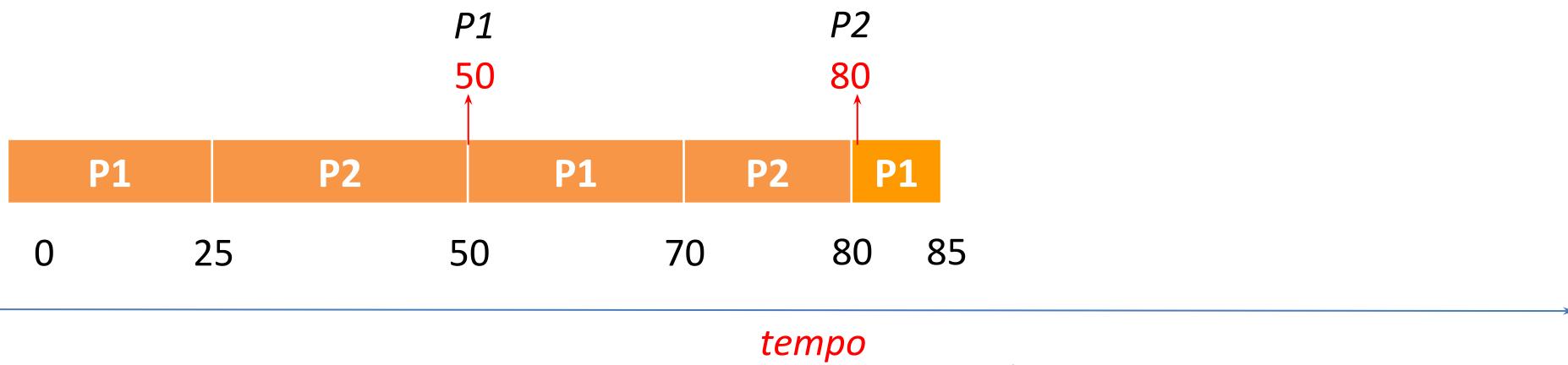
ID	Tempo de Criação (chegada)	Tempo de CPU	Período	Deadline
P1	0	25	50	50
P2	0	35	80	80



Resolvendo a perda de deadline no slide anterior...

- Algoritmo de Taxa Monotônica (*Rate Monotonic - RM*)

ID	Tempo de Criação (chegada)	Tempo de CPU	Período	Deadline
P1	0	25	50	50
P2	0	35	80	80



Escalonamento de Processos

- Escalonamento em Sistemas de Tempo Real
 - Algoritmo do Deadline mais Cedo Primeiro (*Earliest Deadline First - EDF*)
 - As prioridades são atribuídas dinamicamente, de acordo com os *deadlines* das tarefas.
 - Quanto mais próximo o deadline, maior a prioridade e quanto mais longínquo o deadline, menor a prioridade.
 - As prioridades podem ser ajustadas de modo a refletirem o *deadline* do processo recém tornado executável.

Escalonamento de Processos

- Escalonamento em Sistemas de Tempo Real
 - Algoritmo do Deadline mais Cedo Primeiro (*Earliest Deadline First - EDF*)
 - Exemplo:
 - Processos aptos

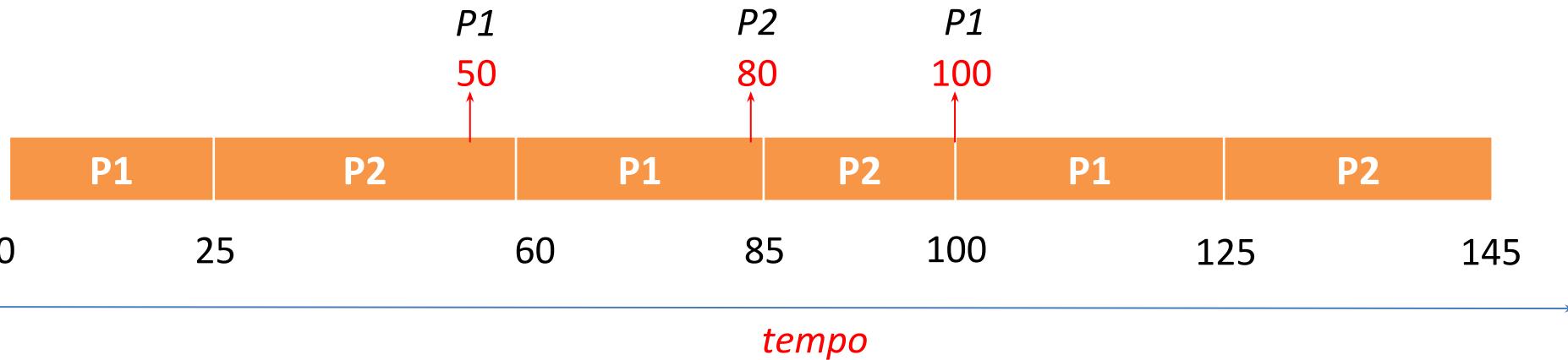
ID	Tempo de Criação (chegada)	Tempo de CPU	Período	Deadline
P1	0	25	50	50
P2	0	35	80	80

O deadline das tarefas é igual ao
início do próximo período.

Escalonamento de Processos

- Algoritmo do Deadline mais Cedo Primeiro (*Earliest Deadline First - EDF*). Exemplo (*sem perda de deadline*):

ID	Tempo de Criação (chegada)	Tempo de CPU	Período	Deadline
P1	0	25	50	50
P2	0	35	80	80



Escalonamento de Processos

- Avaliação de Algoritmos de Escalonamento de Processos
 - O **desempenho** das aplicações (tempo de resposta) está **fortemente relacionado ao tipo de escalonamento de processos empregado**.
 - A **escolha** do algoritmo de escalonamento de processos **deve ser feita seguindo critérios rígidos**.
 - A **avaliação** de algoritmos de escalonamento de processos **pode ser feita por: modelagem determinística, simulação e implementação**.

Escalonamento de Processos

- Avaliação de Algoritmos de Escalonamento de Processos
 - Modelagem Determinística
 - Esse método considera uma carga de trabalho predeterminada específica e define o desempenho de cada algoritmo para esta carga de trabalho.
 - Essa abordagem é simples e rápida.
 - Fornece números exatos que permite comparar os algoritmos.

Escalonamento de Processos

- Avaliação de Algoritmos de Escalonamento de Processos
 - Simulação
 - Muitas vezes não se dispõe de dados determinísticos para uma análise analítica.
 - Essa abordagem envolve a programação de um modelo de sistema computacional.
 - Estruturas de dados representam os componentes do sistema.
 - O estado da simulação é controlado por um relógio, representado por uma variável no simulador.
 - Com essa técnica é possível obter estatísticas que indicam o desempenho do algoritmo.

Escalonamento de Processos

- Avaliação de Algoritmos de Escalonamento de Processos
 - Simulação
 - um sistema computacional é descrito como uma rede de servidores onde cada servidor possui uma fila de processos em espera.
 - Conhecendo as taxas de chegada e as taxas de serviço (utilização da CPU) é possível calcular a utilização, o tamanho médio da fila e o tempo médio de espera.

Escalonamento de Processos

- Avaliação de Algoritmos de Escalonamento de Processos
 - Implementação
 - A melhor maneira de avaliar um algoritmo de escalonamento de processos é implementá-lo e inseri-lo em um sistema operacional.
 - Essa abordagem insere o algoritmo real em um sistema real sob condições reais.
 - A desvantagem dessa técnica é o custo empregado, uma vez que o sistema real será modificado.



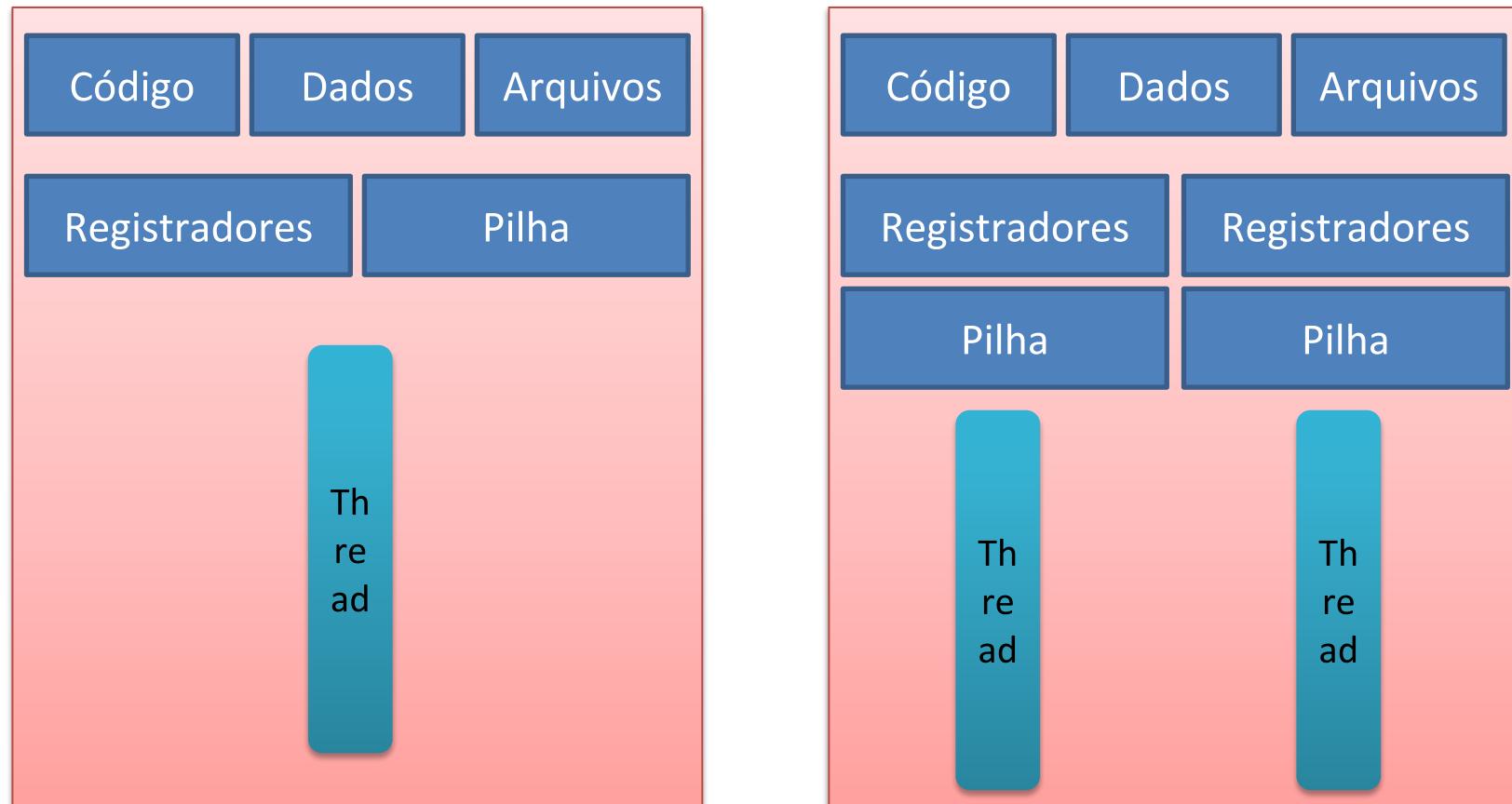
Threads

Definição de Thread

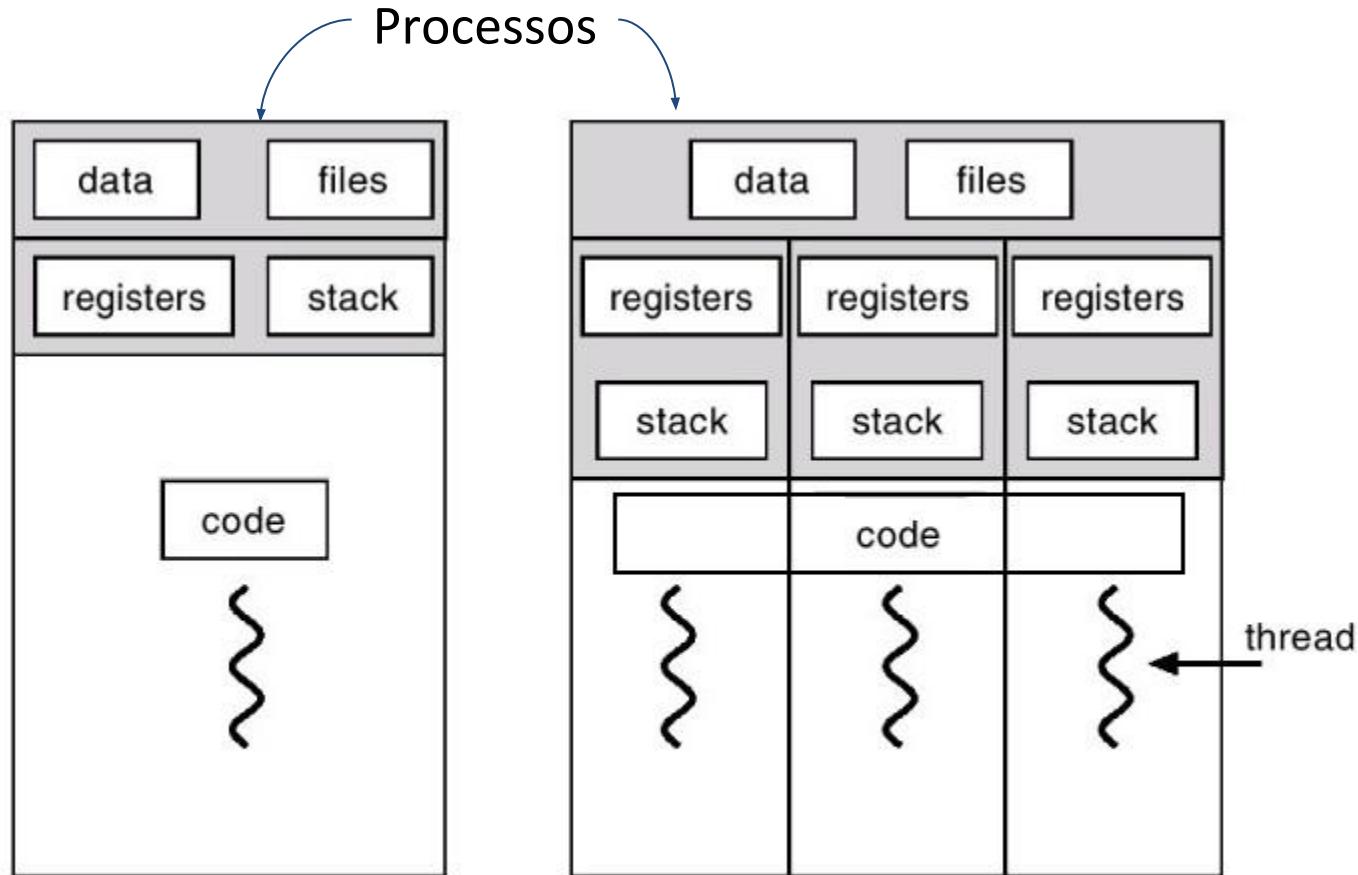
- Um processo possui ao menos um fluxo de execução.
- Um fluxo de execução é conhecido como thread.
- Processos podem possuir mais de uma thread (**processos multithread**).
- Cada thread pode executar uma tarefa específica dentro do processo.

Definição de Thread

- Processo Monothread e Multithread



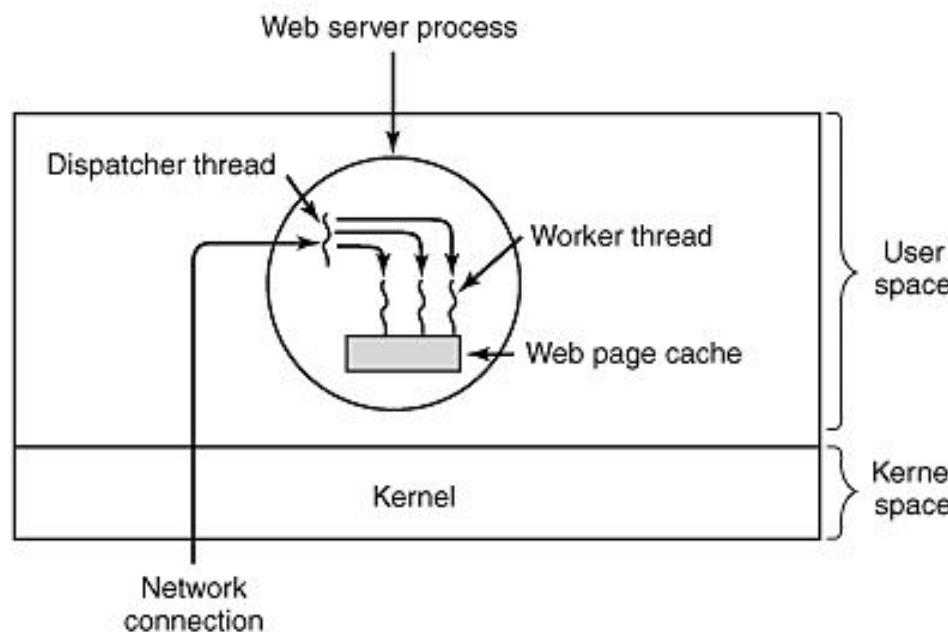
Monothread vs Multithread



Fonte: <https://www.cs.miami.edu/home/visser/Courses/CSC322-09S/Content/UNIXProgramming/UNIXThreads.shtml>

Por que usar threads?

- Decompor um processo em partes menores com custo menor de troca de contexto
- Exemplo: servidor web



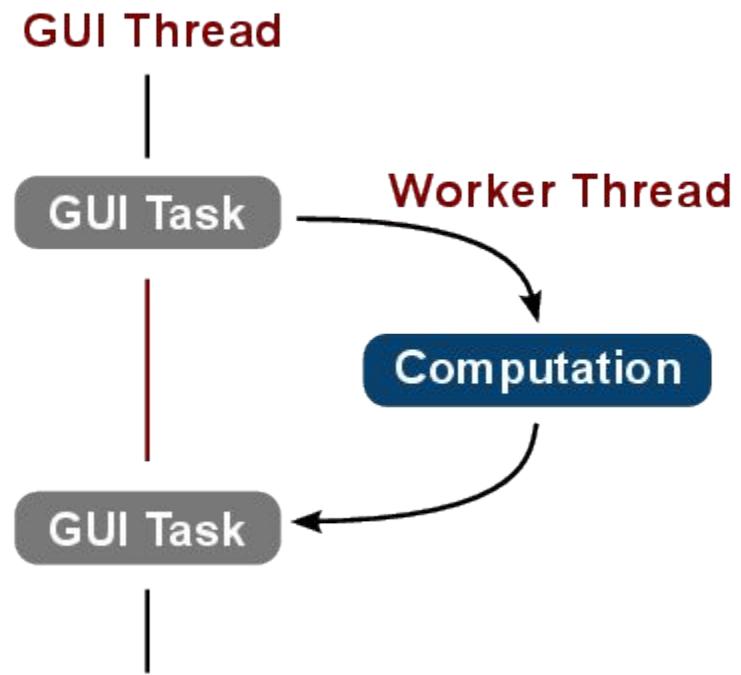
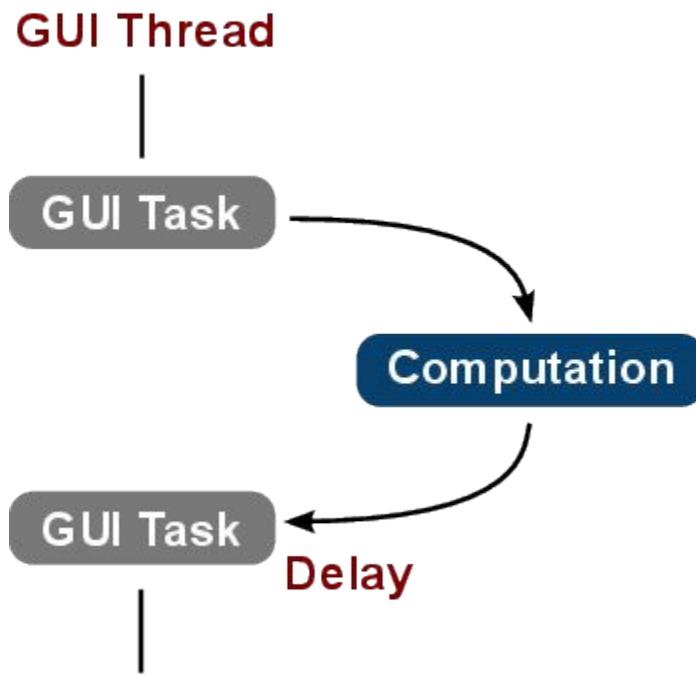
Por que usar threads?

- **Capacidade de resposta:** em um programa interativo é possível manter parte do programa executando enquanto outra parte dele estiver bloqueada esperando de uma operação de E/S.
- **Compartilhamento de Recursos:** um sistema multithread possui várias threads compartilhando o mesmo espaço de endereçamento.

Exemplo de responsividade

monothread

multithread

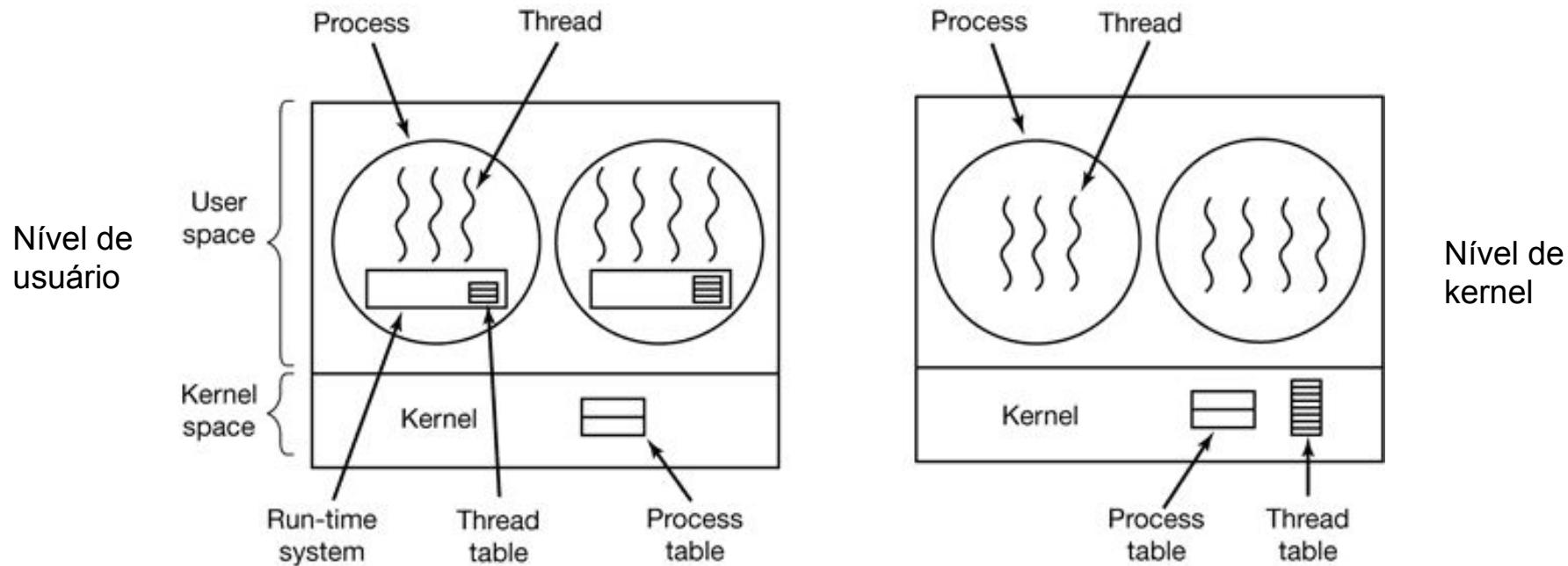


Por que usar threads

- **Economia:** a criação e o gerenciamento de threads pelo SO é mais econômico do que a criação e o gerenciamento de processos.
- **Escalabilidade horizontal:** se o computador possuir vários processadores (ou núcleos de processamento) é possível manter cada thread executando em um processador distinto.

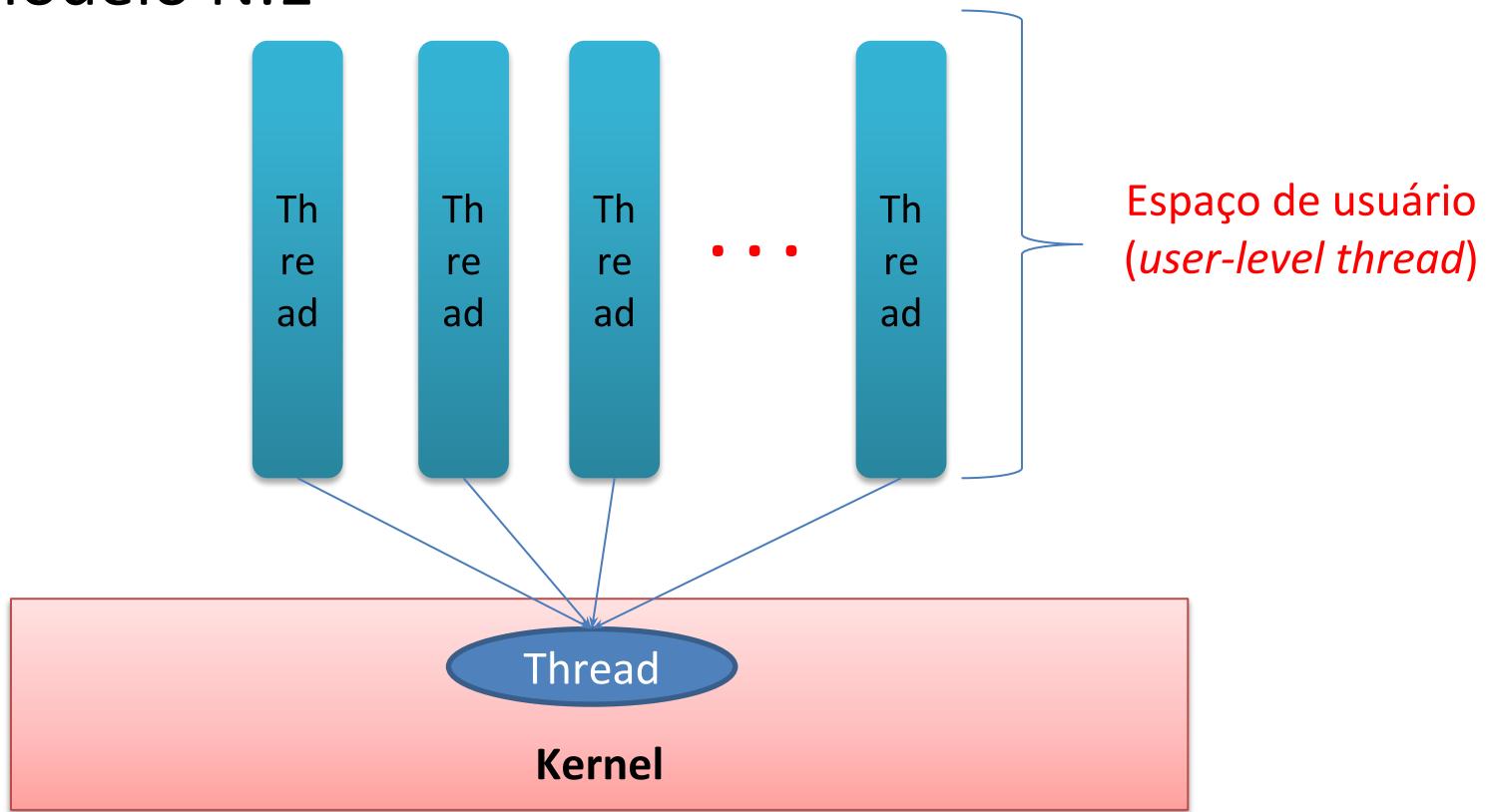
Definição de Thread

- As threads podem ser gerenciadas em 2 níveis



Definição de Thread

- Modelo de Threads
 - Modelo N:1

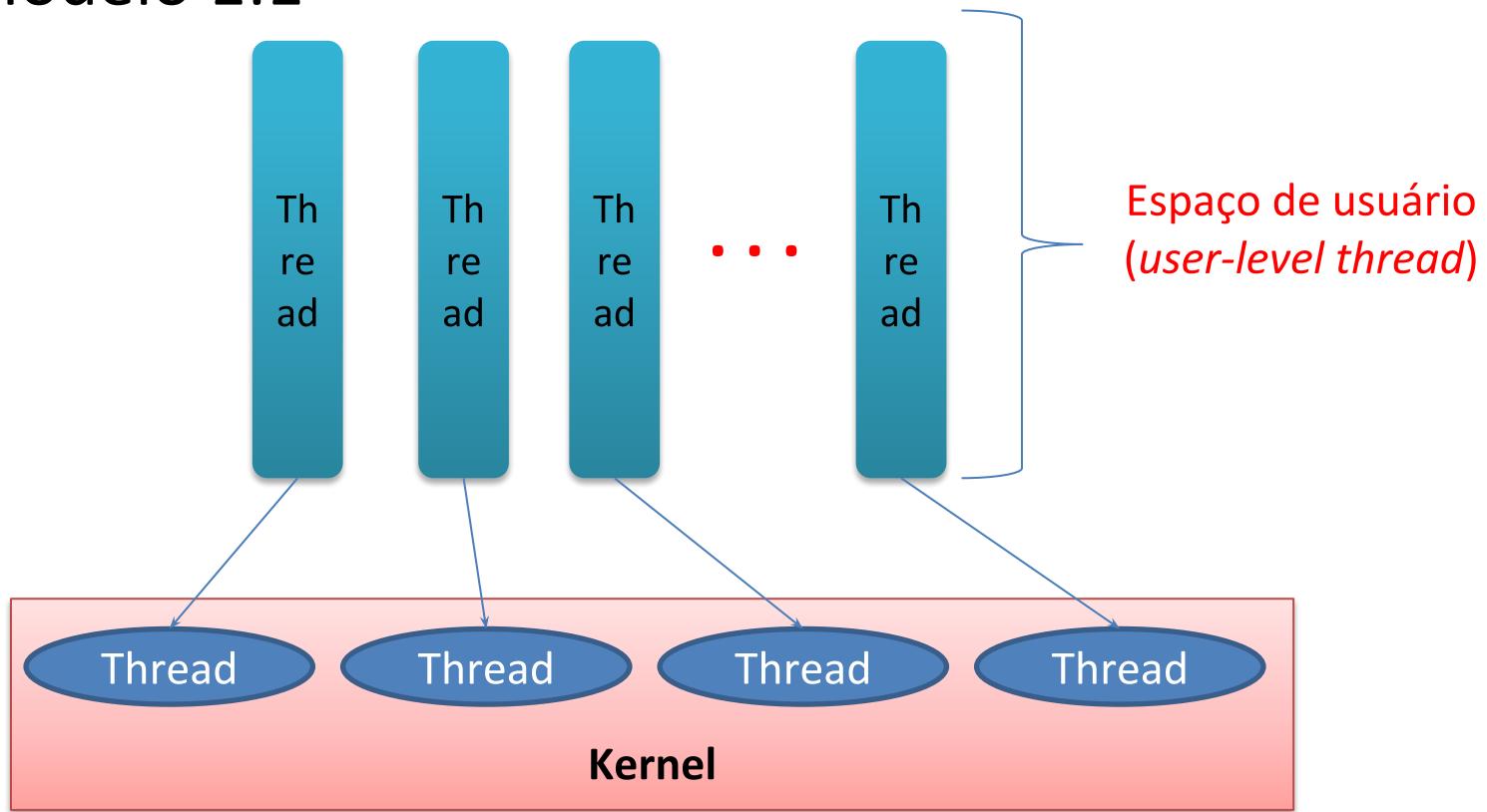


Definição de Thread

- Modelo de Threads
 - Modelo N:1 (user-level threading)
 - Os primeiros SO's não reconheciam as threads, desta forma era necessário o uso de uma biblioteca de threads em nível de usuário.
 - Neste modelo todas as threads implementadas em nível de usuário são representadas por uma única thread no nível de kernel.
 - A vantagem deste modelo é a simplicidade de gerir threads
 - A desvantagem é que se uma thread bloquear para realizar operações de E/S, por exemplo, as demais também param.

Definição de Thread

- Modelo de Threads
 - Modelo 1:1

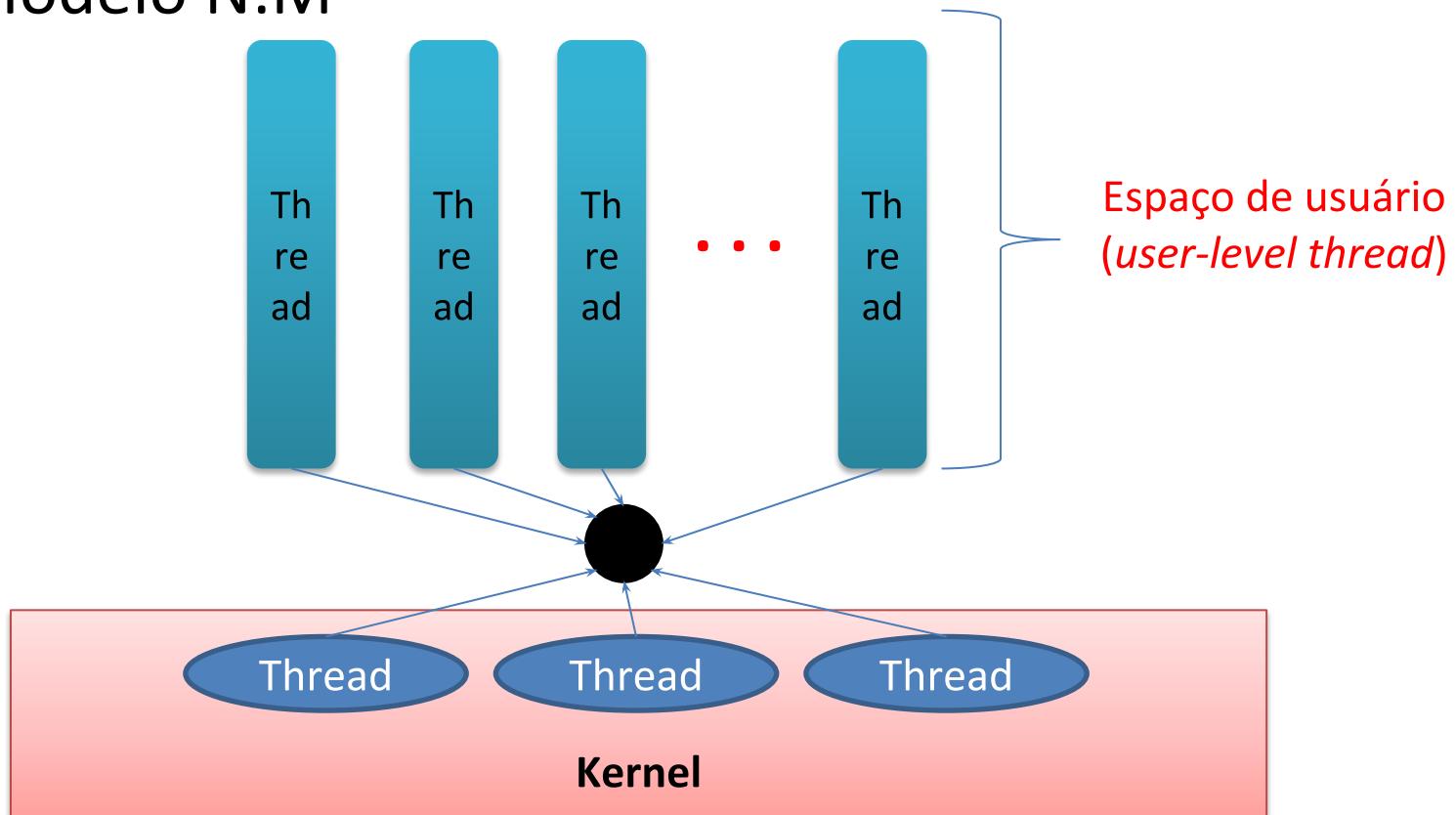


Definição de Thread

- Modelo de Threads
 - Modelo 1:1 (kernel-level threading)
 - Neste modelo cada **thread presente no nível de usuário tem uma correspondente no nível de kernel**.
 - A vantagem deste modelo é que se uma thread bloquear, as demais continuam sendo escalonadas para serem executadas.
 - A desvantagem é que torna o SO muito mais complexo em virtude do gerenciamento individual de cada thread.
 - Exemplos: macOS, iOS

Definição de Thread

- Modelo de Threads
 - Modelo N:M



Definição de Thread

- Modelo de Threads
 - Modelo N:M (hybrid threading)
 - Este modelo é uma junção dos dois anteriores. Neste caso, as **N threads presentes no nível de usuário possuem M threads no nível de kernel**, sendo **N≥M**.
 - A vantagem deste sistema é que cada thread continua sendo vista individualmente pelo SO.
 - Usuário pode criar quantas threads quiser sem comprometer performance
 - Exemplo: IRIX, HP-UX

Exemplo POSIX Thread (pthread)

```
1. #include <pthread.h>
2. #include <stdio.h>
3. void *f_thread();
4. int main()
5. {
6.     pthread_t tid;
7.     pthread_create(&tid, NULL, f_thread, NULL);
8.     pthread_join(tid, NULL);
9.     return 0;
10. }
```

```
11. void *f_thread()
12. {
13.     for (;;) {
14.         printf("Thread executando...\n");
15.     }
16. }
```

Exemplo Java: Classe Thread

```
1. class testeThread extends Thread {  
2.     public void run()  
3.     {  
4.         while (true) {  
5.             System.out.println("Thread em Java ...");  
6.             try {  
7.                 sleep(2);  
8.             } catch (InterruptedException e) {  
9.                 e.printStackTrace();  
10.            }  
11.        }  
12.    }  
13. }  
14. public class Main {  
15.     public static void main(String[] args) {  
16.         testeThread tt = new testeThread();  
17.         tt.start();  
18.         try {  
19.             tt.join();  
20.         } catch (InterruptedException e) {  
21.             e.printStackTrace();  
22.         }  
23.     }  
24. }
```

Exemplo Java: Interface Runnable

```
1. class testeThread implements Runnable {           13. public class Main {  
2.     public void run() {                         14.     public static void main(String[] args) {  
3.         while (true) {                         15.         Thread tt = new Thread(new  
4.             System.out.println("Thread executando..."); 16.             testeThread());  
5.             try {                                17.             tt.start();  
6.                 Thread.sleep(2);                18.             tt.join();  
7.             } catch (InterruptedException e) {        19.             } catch (InterruptedException e) {  
8.                 e.printStackTrace();            20.                 e.printStackTrace();  
9.             }                                21.             }  
10.        }                                    22.         }  
11.    }                                    23.     }  
12. }
```

Exemplo Windows Thread¹

```
#include <windows.h>
#include <tchar.h>
#include <strsafe.h>

#define MAX_THREADS 3
#define BUF_SIZE 255

DWORD WINAPI MyThreadFunction( LPVOID lpParam );
void ErrorHandler(LPTSTR lpszFunction);

typedef struct MyData { ... } MYDATA, *PMYDATA;

int _tmain()
{
    PMYDATA pDataArray[MAX_THREADS];
    DWORD dwThreadIdArray[MAX_THREADS];
    HANDLE hThreadArray[MAX_THREADS];

    // Create MAX_THREADS worker threads.

    for( int i=0; i<MAX_THREADS; i++ )
    {
        pDataArray[i] = (PMYDATA)
            HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY,
            sizeof(MYDATA));
    }
}
```

```
if( pDataArray[i] == NULL ) { ExitProcess(2); }

hThreadArray[i] = CreateThread(
    NULL,
    0,
    MyThreadFunction,
    pDataArray[i],
    0,
    &dwThreadIdArray[i]);

if (hThreadArray[i] == NULL) {
    ErrorHandler(TEXT("CreateThread"));
    ExitProcess(3);
}

WaitForMultipleObjects(MAX_THREADS,
hThreadArray, TRUE, INFINITE);

for(int i=0; i<MAX_THREADS; i++) {
    CloseHandle(hThreadArray[i]);
    if(pDataArray[i] != NULL){
        HeapFree(GetProcessHeap(), 0,
        pDataArray[i]);
        pDataArray[i] = NULL;
    }
}
return 0;
```

¹ [Microsoft](#)

Escalonamento de Processos

- O escalonamento de threads pode ser realizado de duas maneiras:
 1. **No processo (*Process Contention Scope – PCS*)**: a disputa pelo processador acontece em nível de processo, ou seja, a biblioteca de gerenciamento de threads é responsável por determinar qual thread entrará em execução.
 2. **No sistema (*System Contention Scope – SCP*)**: a disputa pelo processador acontece no nível de sistema. O SO determinará qual thread entrará em execução. Implementado no modelo 1:1 de gerenciamento de threads.

Escalonamento de Processos

- Escalonamento em Sistemas com mais de um Processador (**multiprocessadores**)
 - Abordagem assimétrica (*mestre-escravo*)
 - Existe um processador mestre que manipula as estruturas de dados do sistema.
 - Os demais processadores são considerados escravos e são destinados a execução de processos que o mestre determinar.

Escalonamento de Processos

- Escalonamento em Sistemas com mais de um Processador (**multiprocessadores**)
 - Abordagem simétrica
 - Cada processador é responsável pelo seu próprio escalonamento.
 - Nesta abordagem pode existir uma única fila de prontos para todos os processadores ou cada processador pode ter sua própria fila de prontos.
 - Deve-se ter mecanismos que impeçam que dois processadores não selecionem o mesmo processo para executar.

Tarefa

- Pesquise como o algoritmo Work-Steal funciona.
- Ele é simétrico ou assimétrico?