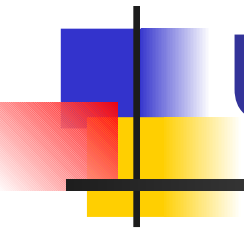
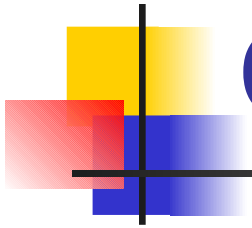


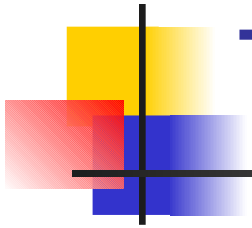
Programação Paralela Utilizando Threads

Cleber Soares Mori
Leandro Botaro Martins
Luis Gustavo Pinheiro Machado
Ricardo dos Reis Cardoso



Concorrência

- Algumas tarefas não dependem de uma execução sequencial
- Várias tarefas poderia estar sendo executadas paralelamente
- Sequencialmente poderíamos levar muito tempo para o término da tarefa

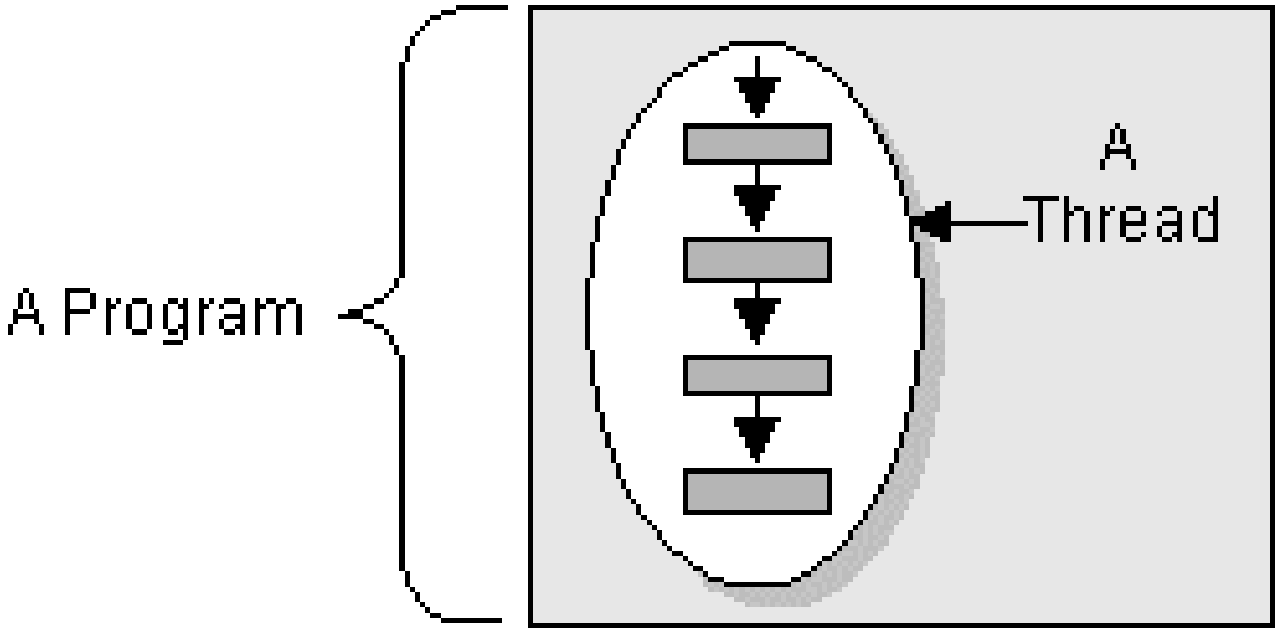


Threads

- O que é um Thread?
  - Prática:
    - Mecanismo para execução de tarefas concorrentemente
  - Definição:
    - São linhas de controle de um mesmo processo, com fluxos de execução independentes e que podem ser executadas concorrentemente, mas que compartilham o mesmo espaço de endereçamento

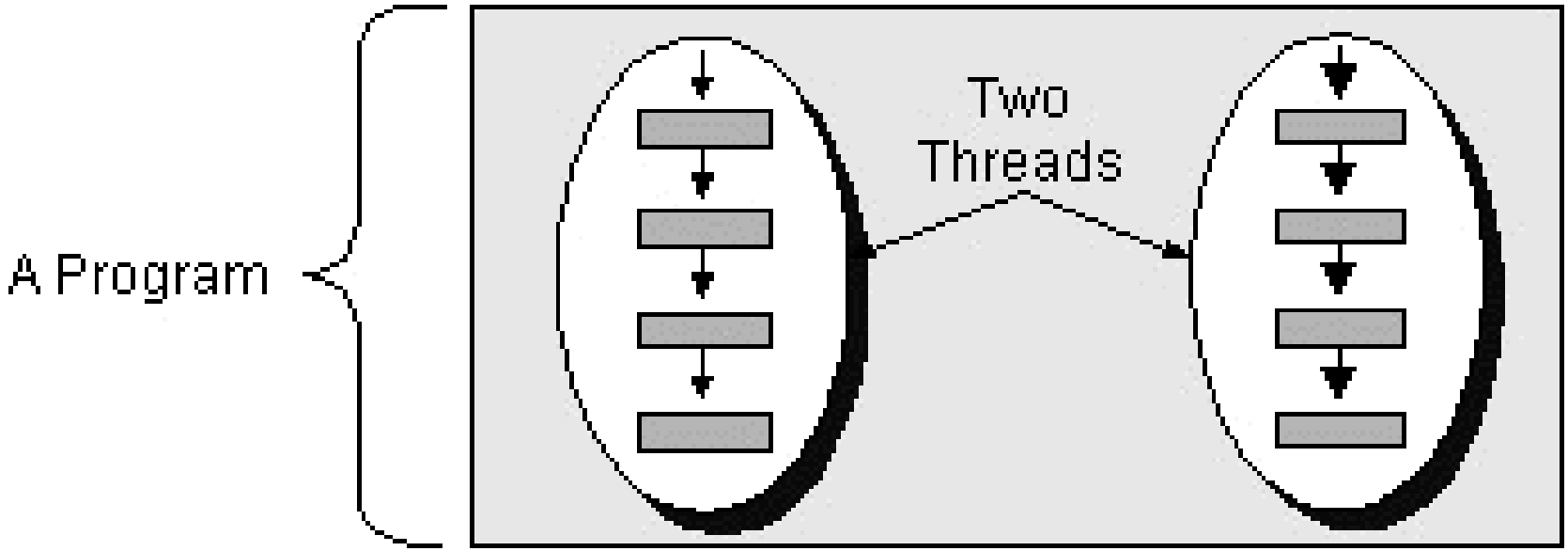
Threads

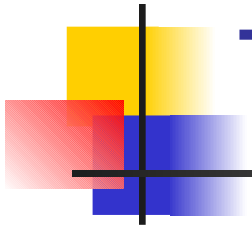
- Processo com 1 Thread



Threads

- Processo com 2 Threads

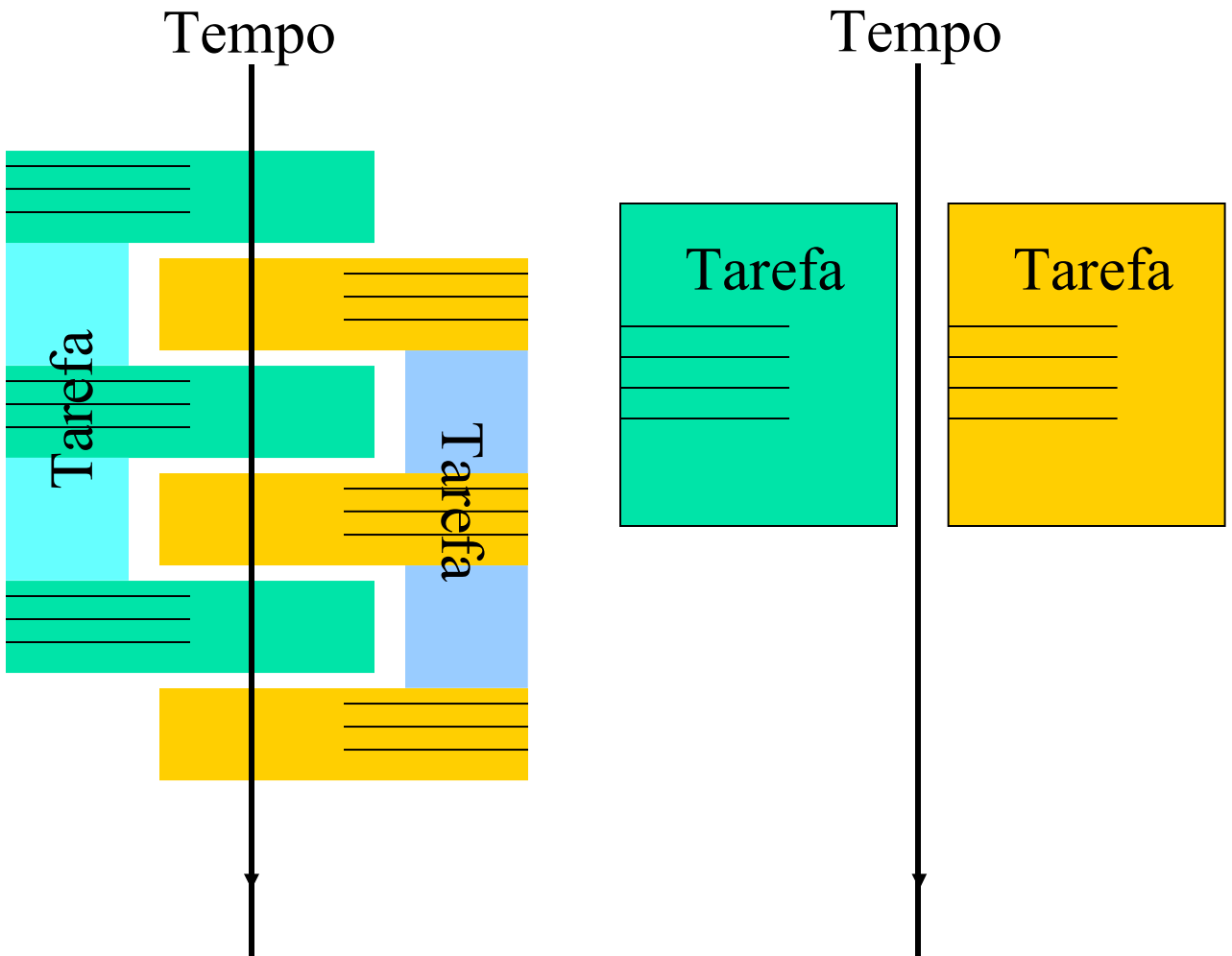


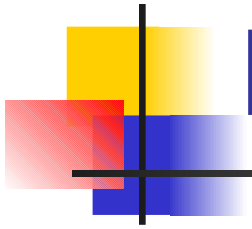


Threads

- Aplicações de Threads
  - Multithreaded Applications
    - Um browser
    - SGBD's
    - Programas Científicos
- Na sua vida você age de maneira concorrente... Por que seus programs não agiriam?

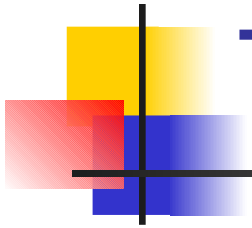
Concorrência X Paralelismo





Processos - Características

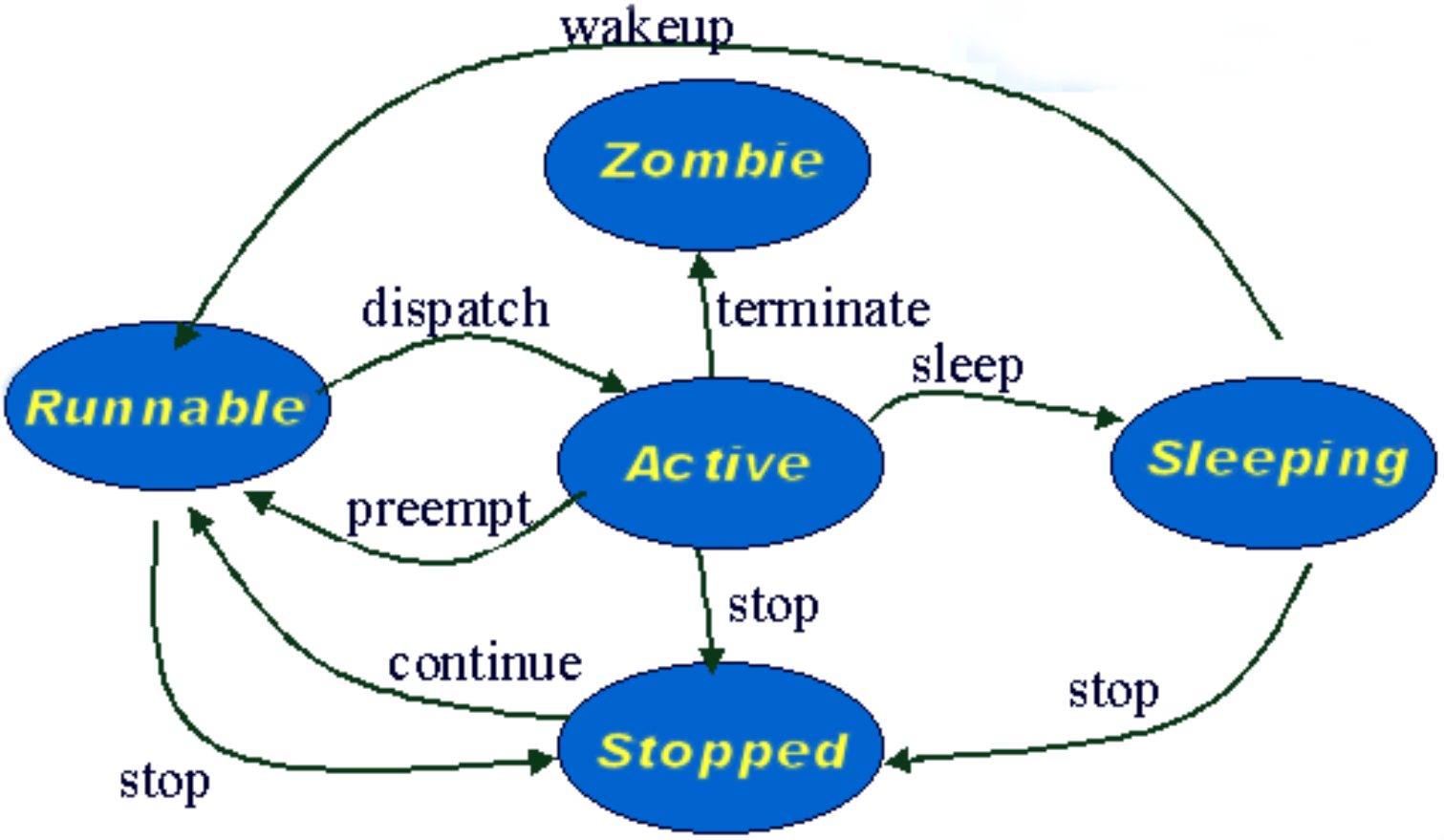
- Possuem seus próprios:
  - Contadores de instruções
  - Pilha de execução
  - Conjunto de registradores
  - Espaço de endereçamento
- Meios de comunicação:
  - Semáforos (Mutex)
  - Monitores
  - Mensagens

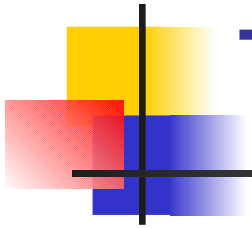


Threads - Características

- Também chamados de Lightweight Processes
- Cada thread possui:
  - Execução individual e sequencial
  - Contador de instruções
  - Pilha de execução
- Outras características
  - Compartilha recursos de um processo (Processamento de espaço de endereçamento)
  - Threads pertencem a um único processo
  - Podem criar Threads-filho
  - Não possuem proteção de memória

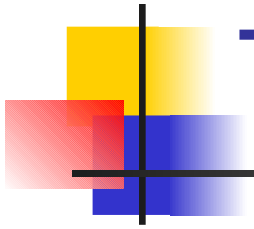
Threads – Ciclo de Vida





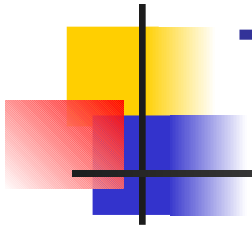
Threads - Versões

- Unix
  - Processos e Threads
- Mach
  - Tasks e Threads
- Conceitos diferentes



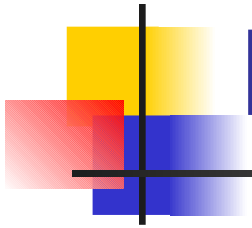
Threads – Sistema Unix

- Processos: Representam um programa em execução, com um conjunto de registradores, espaço de endereçamento e threads associados.
- Threads: Contém um fluxo seqüencial de execução com um conjunto de recursos associado, como tabelas de descrição de arquivos. Seu espaço de endereçamento é compartilhado do processo.



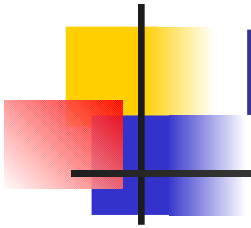
Threads - Sistema Mach

- Task: Contém somente um conjunto de recursos.
- Threads: Tratam de todos os detalhes da execução. Cada Thread possui:
  - Contadores de programa
  - Pilha de execução
  - Conjunto de registradores



Mach x Unix

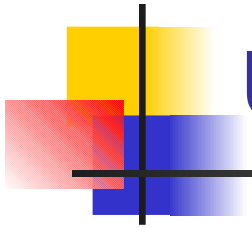
- Como seria um processo Unix em Mach?
  - Modelar um processo Unix como uma Task contendo apenas um Thread
- Vantagem da abordagem Mach?
  - Facilidade na programação paralela
  - Transparência



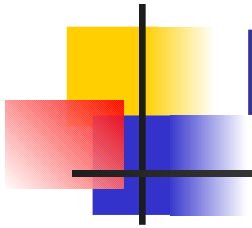
Mach x Unix

- Exemplos de vantagens:
  - Facilidade na programação paralela
    - Aplicação roda facilmente em um ou mais processadores
  - Transparência
    - O sistema aloca processadores adicionais quando for preciso

Programação Paralela utilizando Threads



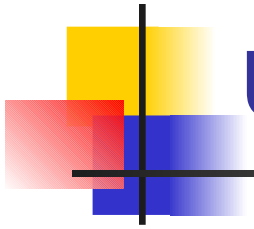
- Por que usar threads?
  - Simplicidade
  - Boa opção para execução paralela de processo (ou pseudoparalela)
  - Maior eficiência do que os tradicionais comandos “fork” de processos



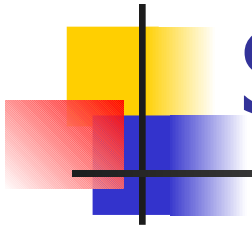
Multithreading - Vantagens

- Multithreading apresenta:
  - Tempo de resposta reduzido
  - Uso eficiente de multiprocessadores
  - Melhora estrutura do programa
  - Uso de menos recursos do sistema
  - Melhoria de desempenho

Programação Paralela utilizando Threads - Exemplo

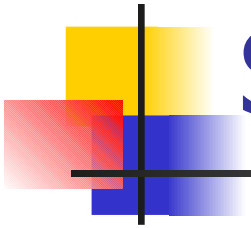


- Internet Browser
  - Eventualmente acessa o disco
  - Bloqueio para esperar os dados
  - Multithreading
    - Apenas uma thread é bloqueada
    - Outras threads podem processar as páginas (Parsing, Gerenciamento de Formulários)
- Múltiplos Processos? Inviável!
  - Deveriam compartilhar o mesmo espaço de endereçamento para uma comunicação eficiente.



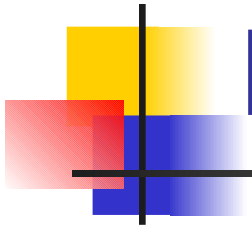
Sincronização de Threads

- Principal dificuldade dos threads
  - Mesmo espaço de endereçamento é compartilhado
    - Evitar que vários threads acessem os mesmos dados.
- Solução
  - Sincronização de Threads!



Sincronização de Threads

- O padrão POSIX provê duas primitivas:
  - Mutex
  - Variáveis de Condição
- Mutex
  - Sincronia de Curta duração
  - Impede entrada mútua em RC
- Variáveis de Condição
  - Sincronia de longa duração
  - Impede entrada mútua em RC até que um recurso esteja disponível

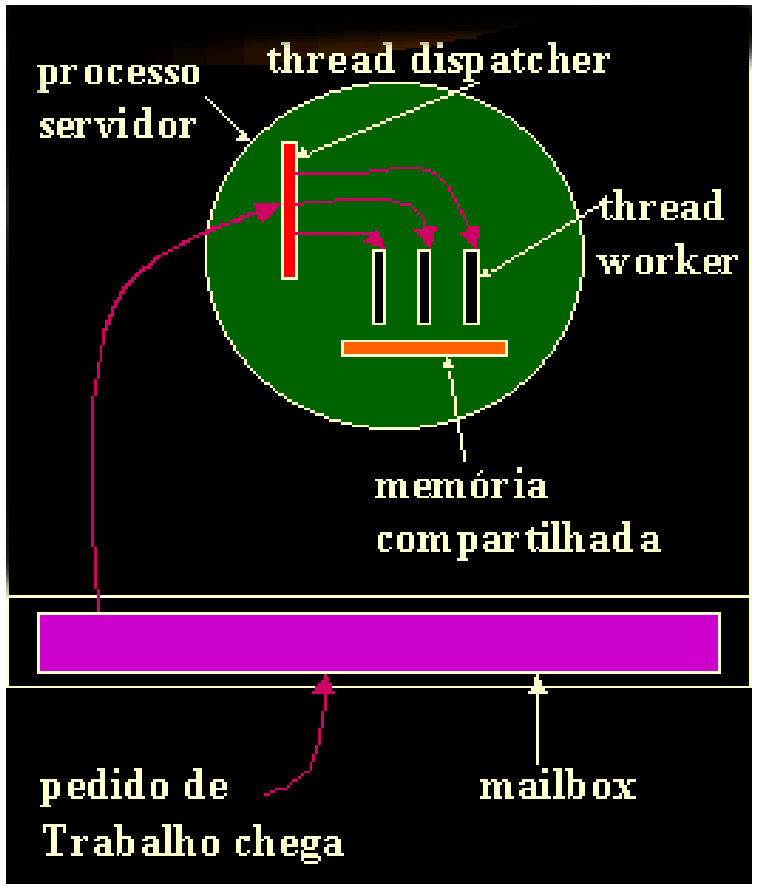


Modelos de Threads

- Há três modelos para implementação de threads:
  - Modelo Dispatcher/Worker
  - Modelo Pipelining
  - Modelo Cooperativo

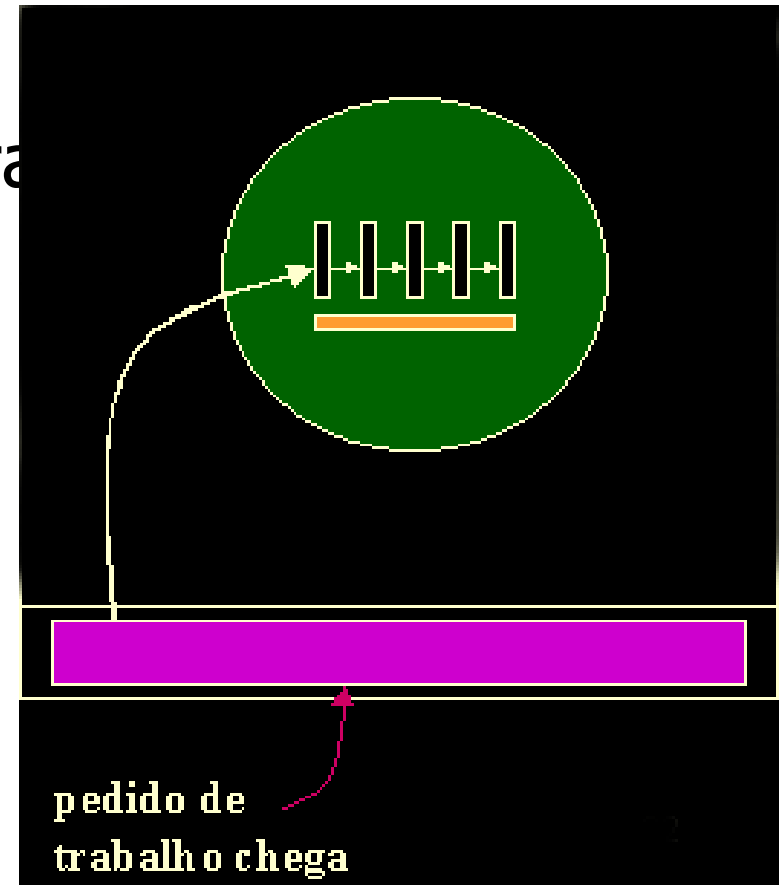
Modelos – Dispatcher/Worker

- O dispatcher procura tarefas a executar na Mailbox
- Se houver uma tarefa e um worker estiver livre, este é escalado para executar a tarefa
- Worker termina a tarefa e avisa ao Dispatcher



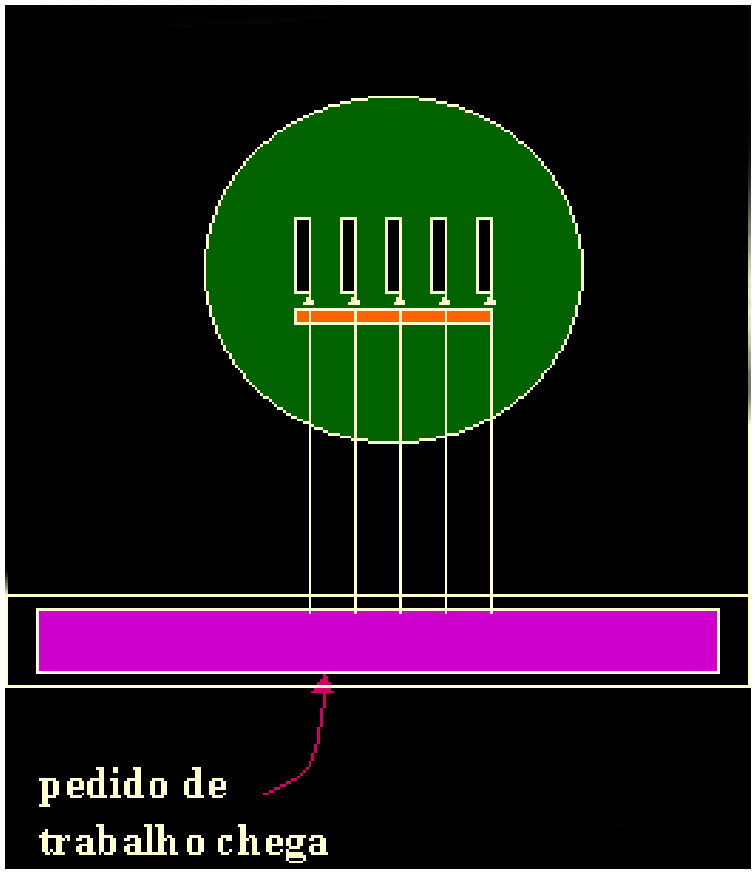
Modelos – Pipelining

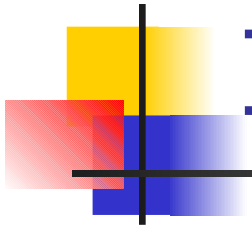
- As tarefas vão sendo passadas de thread para thread de uma maneira sequencial
- Cada thread processa uma parte da tarefa (estágio da pipeling)



Modelos – Cooperativo

- Todos os threads trabalham em uma única tarefa
- A tarefa é dividida em várias partes que são executadas em paralelo
- Cada thread executa uma parte da tarefa

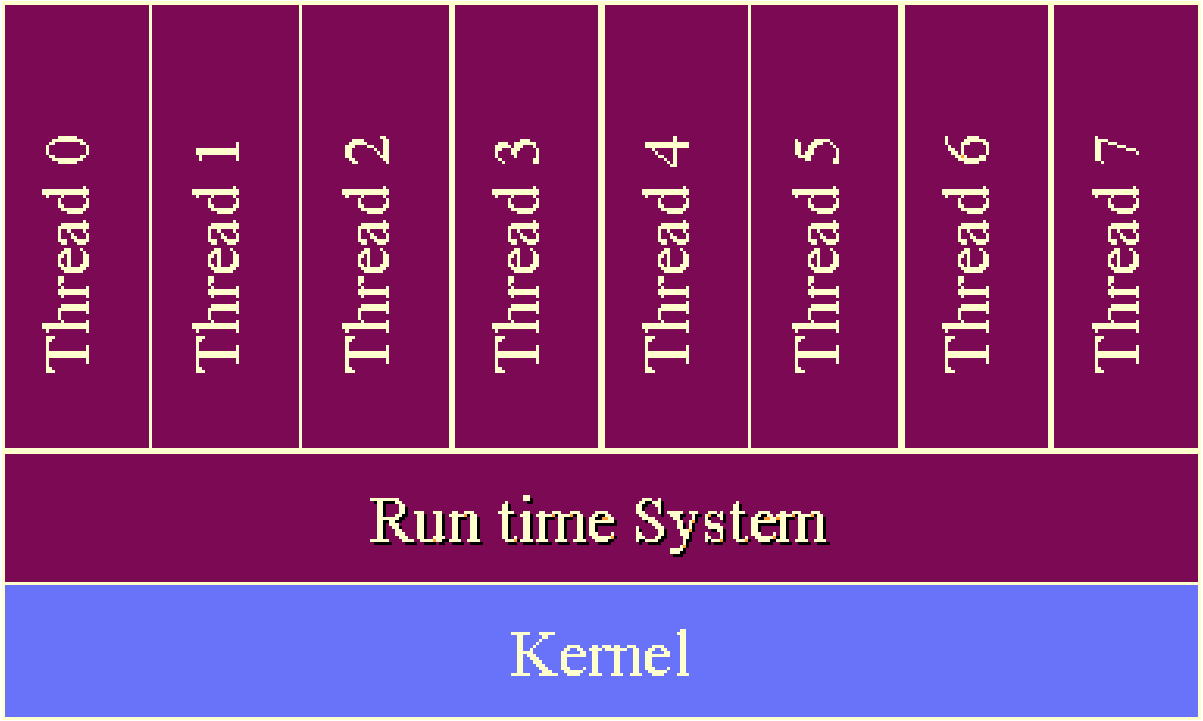


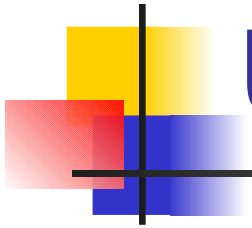


Implementações

- Há dois tipos de implementações:
  - User-Level Threads
  - Kernel-Level Threads

User-Level Threads

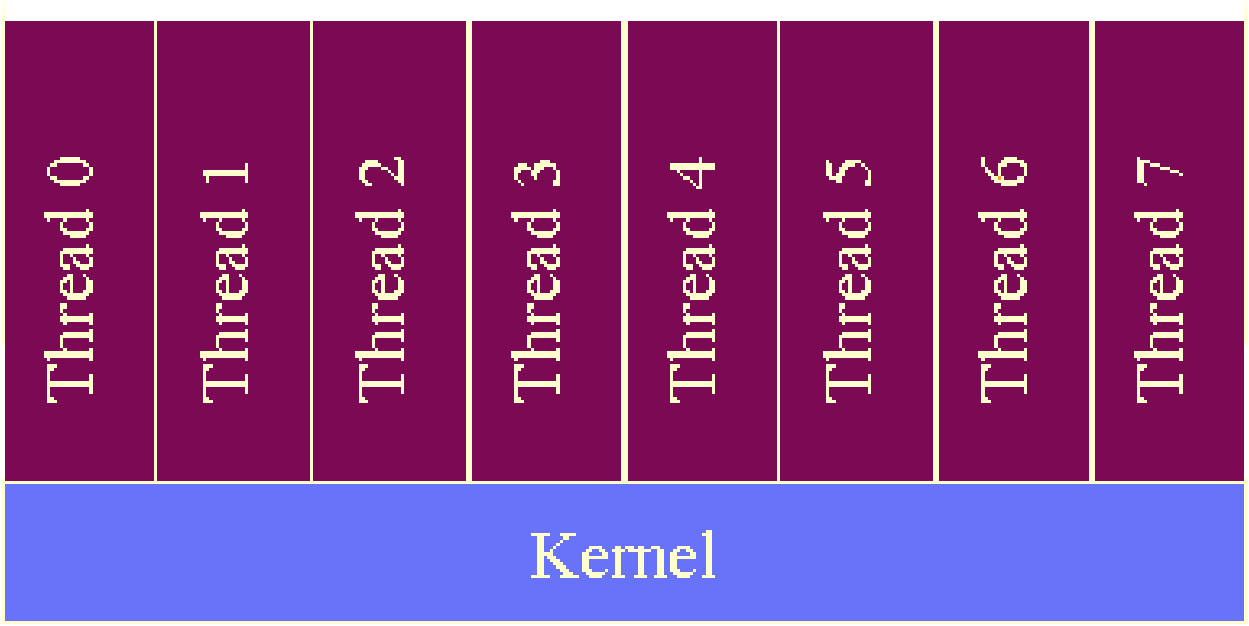


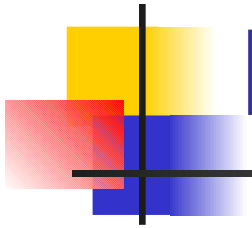


User-Level Threads

- Vantagens
  - Algoritmo de escalonamento próprio
  - Mais escaláveis
  - Troca de contexto mais rápida
- Desvantagens
  - Um Thread pode usar todo o Time-Slice
  - Não tiram vantagem de sistemas SMP's
  - Se um thread for bloqueado, todas as outras do processo também o serão

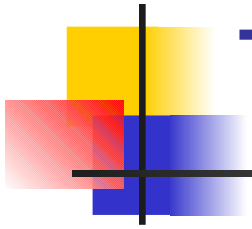
Kernel-Level Threads





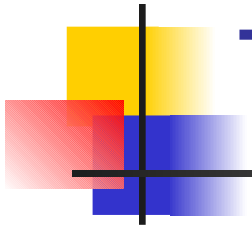
Kernel-Level Threads

- Vantagens
  - Tira vantagens de arquiteturas SMP's
  - Distribuição justa do Time-Slice
  - Um thread não bloqueia outras do mesmo processo
- Desvantagens
  - Muitas tabelas de controle (uma para cada thread)
  - Processos se tornam pesados
  - Mais overhead para passagem entre os modos User-Kernel-User (Threads escalonados dentro do Time-Slice do processo



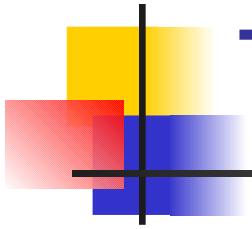
Tipos de Threads

- Alguns sistemas possuem ambas as implementações
- Pode-se juntar as melhores qualidade de cada uma
- Exemplo – Linux:
  - As duas implementações possuem desempenhos equivalentes



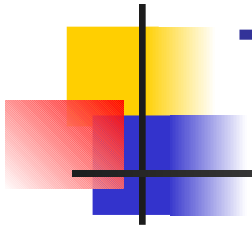
Threads - Vantagens

- Suportam operações concorrentes
  - Exemplo: Tarefas podem ser tratadas em background sem interferir na interação com o usuário
- Tornam os programas mais simples
  - Exemplo: Atualização de muitos campos pode ser feita por vários threads
- Permitem o paralelismo em aplicações
  - Exemplo: Computadores com vários processadores podem executar threads em processadores diferentes sem usar time-sharing



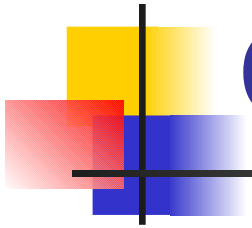
Threads - Desvantagens

- Em relação ao uso:
  - Uso muito complicado para maioria dos programadores
  - Processo de desenvolvimento árduo
- Motivos:
  - Deadlock
  - Difícil implementação de sincronização
  - Difícil depuração
  - Difícil atingir bom desempenho
  - Nem sempre é necessário concorrência
  - Implementações não muito padronizadas



Threads - Desvantagens

- Solução?
  - Análise cuidadosa do problema
  - Verificar se existe real vantagem no uso de threads
- Alternativa?
  - Programação dirigida a eventos



Conclusão

- Os programas paralelos, então, usufruem desta ferramenta para paralelizar vários trechos do código sem gastar muitos recursos, e aproveitando um eventual paralelismo existente no hardware de maneira simples e nativa.