Sistemas Operacionais

Threads (cap. 5)

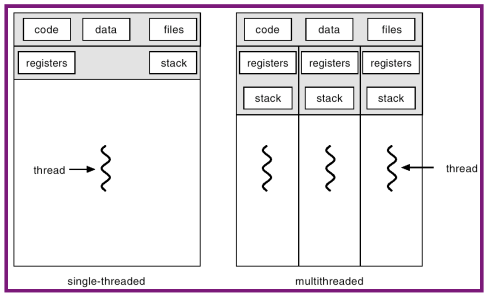
Sumário

- Princípio geral de processos x *threads*
- Modelos operação com múltiplas *threads*
- Questões de implementação e uso
- Diversas versões de *threads*
  - Pthreads
  - Solaris 2
  - Windows 2000
  - Linux
  - Java

Processo e *threads*

- *Thread*: fluxo de controle de instruções
- Processo: espaço de memória e recursos alocados a ele associados (uma *thread* ou mais)
  - Se um processo tem várias *threads*, elas compartilham quase todos os recursos e memória
  - Exceção apenas para o que identifica cada *thread*
    - Contador de programa
    - Registradores
    - Pilha de execução

Processo e *threads*



Benefícios

- Capacidade de resposta
- Compartilhamento de recursos
- Economia (comparado com processos)
- Facilidade de expressão
- Aproveitamento de arquiteturas multiprocessadas
  - Multi-core, *hyperthreading*

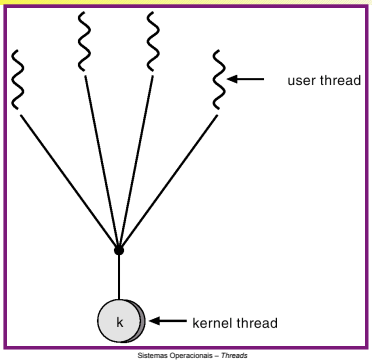
Threads de usuário x kernel

- Implementações com compromissos diferentes
- *Threads* no nível de usuário (bibliotecas)
  - Mais “leves”, pois *overhead* se limita ao programa
  - Se o kernel não reconhece, pode ser ineficiente
- *Threads* de kernel
  - Melhor integradas ao escalonador do S.O.
  - Mais *overhead*
- Mapeamento entre os dois níveis varia entre sistemas e tem compromissos diferentes

Mapeamento muitos-para-um

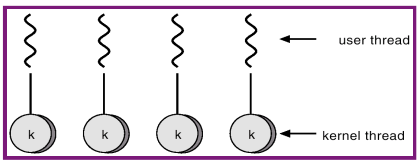
- Muitas *threads* de usuário mapeadas para uma única *thread* de kernel
  - Ocorre, p.ex., quanto o S.O. não reconhece *threads*, apenas processos
- Se uma *thread* faz uma chamada do S.O. bloqueante, todo o processo é suspenso
  - Mesmo que outras threads pudessem executar
  - Para evitar isso, bibliotecas precisam transformar chamadas bloqueantes em não bloqueantes

Mapeamento muitos-para-um



Mapeamento um-para-um

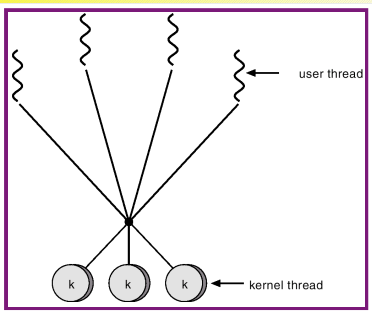
- Kernel reconhece *threads* e cada *thread* de usuário é na verdade uma *thread* de kernel
- Mapeamento é simples, mas "pesado"
  - Toda troca de *threads* envolve mudança p/ modo protegido



Modelo muitos-para-muitos

- Mapeamento flexível entre modo usuário e kernel
- O S.O. cria um número "suficiente" de *threads*
- Bibliotecas de *threads* de usuário fazem mapeamento como preferirem
  - Threads Solaris 2 (*threads x light-weight process*)
  - Windows NT/2000 com pacote *ThreadFiber*
  - Linux com bibliotecas *Pthreads*

Modelo muitos-para-muitos



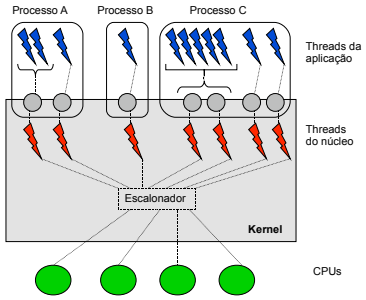
Questões associadas a *threads*

- Relacionamento com a criação de processos
- Cancelamento de threads em execução
- Tratamento de eventos externos a um processo
- Utilização de *pools* (cadeias) de *threads*
- Controle de dados específicos de cada *thread*

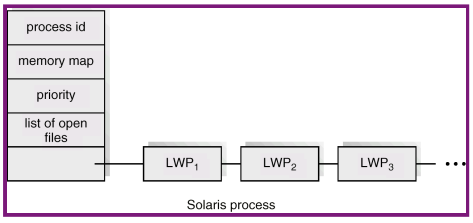
Pthreads (POSIX threads)

- Interface padrão POSIX (IEEE 1003.1c) para criação e sincronização
- Biblioteca de nível de usuário
  - A API define o comportamento da biblioteca
  - Implementação a cargo do desenvolvedor
  - Mapeamento para *threads* de kernel é possível
    - P.ex., implementação em Linux não é exata
- Comum em sistemas Unix em geral

Threads no Solaris 2



Processos no Solaris 2



Threads no Windows 2000

- Mapeamento um-para-um
- Cada *thread* contém:
  - Identificador
  - Conjunto de registradores
  - Pilhas separadas para kernel e usuário
  - Área de dados privativa

Threads no Linux

- Totalmente integradas: linux se refere a tarefas (*tasks*), ao invés de processos e *threads*
- Criação de uma *thread*: *syscall* `clone()`
  - Semelhante ao `fork()`, mas sem duplicação da memória
  - permite a uma tarefa compartilhar os apontadores para o espaço de memória de outra
- Bibliotecas *pthreads* podem (ou não) fazer mapeamento muitos-para-muitos

Threads Java

- *Threads* podem ser criadas:
  - estendendo-se a classe *Thread*
  - implementando-se a interface *Runnable*
- *Threads* são gerenciadas pela JVM
  - Basicamente um-para-um

Estados de Threads Java