BCC264

Sistemas Operacionais

Threads

Prof. Charles Garrocho

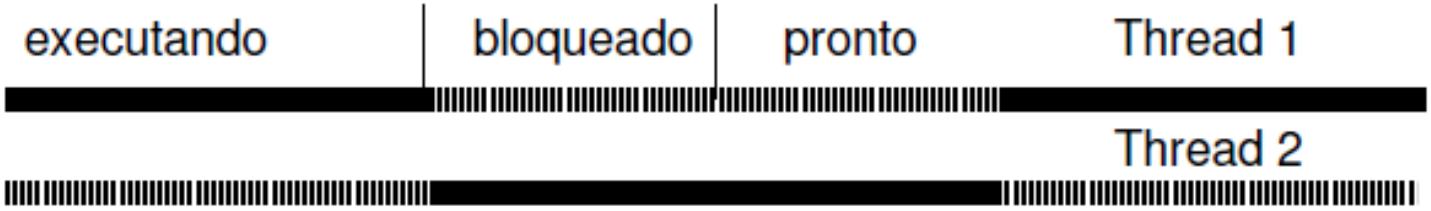
Processo e *threads*

- *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 os recursos, como a memória
  - Exceção apenas para o que identifica cada *thread*
    - Contador de programa
    - Registradores
    - Pilha de execução

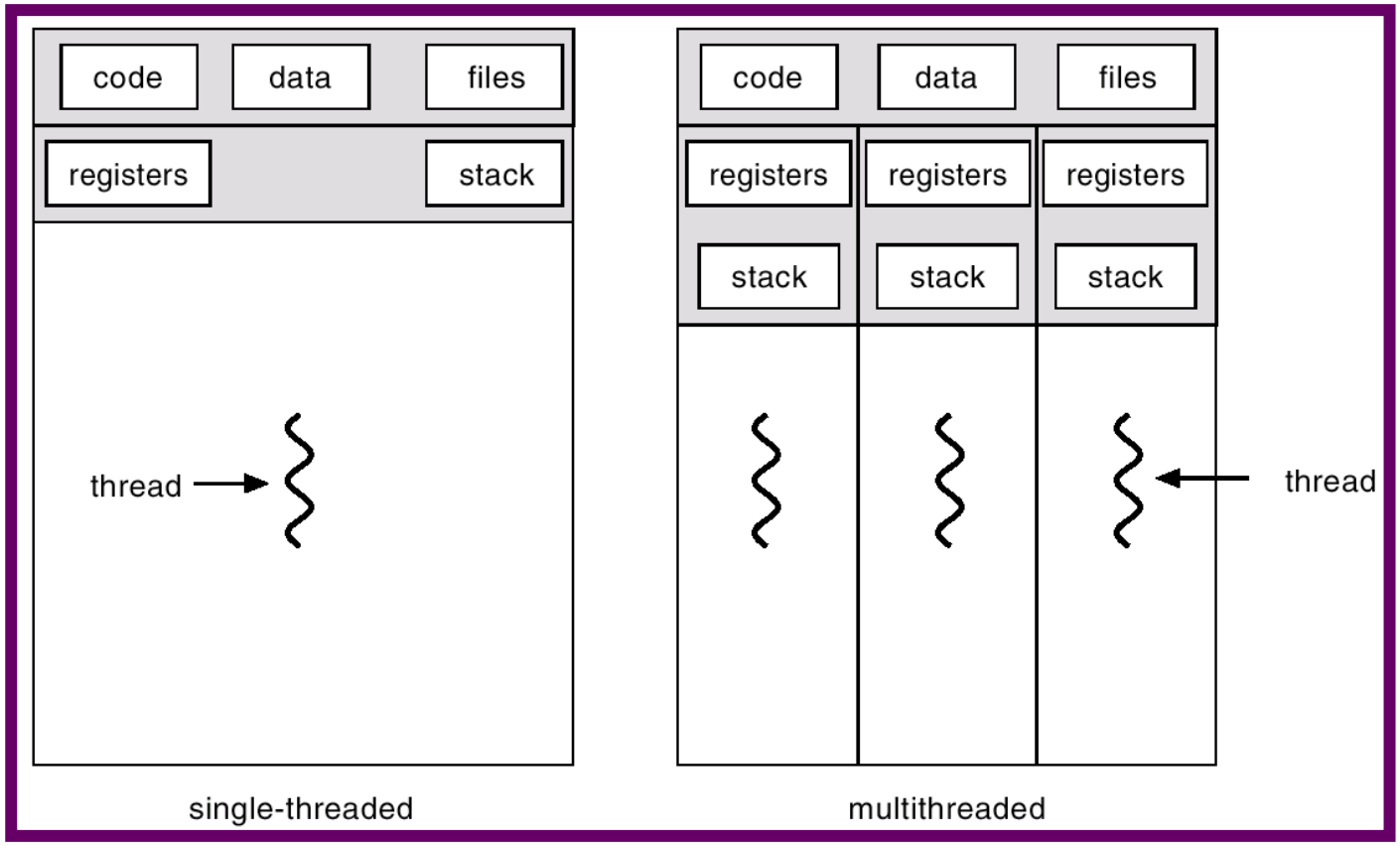
Processo e *threads*

Processos *versus* Threads

- Processo é um ambiente de execução
  - Espaço de endereçamento
  - Recursos de sincronização e comunicação
  - Recursos de mais alto nível
- Thread é uma atividade no sistema
  - Compartilham recursos do processo
  - Objetivo: maximizar o grau de execução concorrente
    - Sobreposição E/S e processamento



Processo e *threads*



Benefícios

- Capacidade de resposta
- Compartilhamento de recursos
- Economia (comparado com processos)

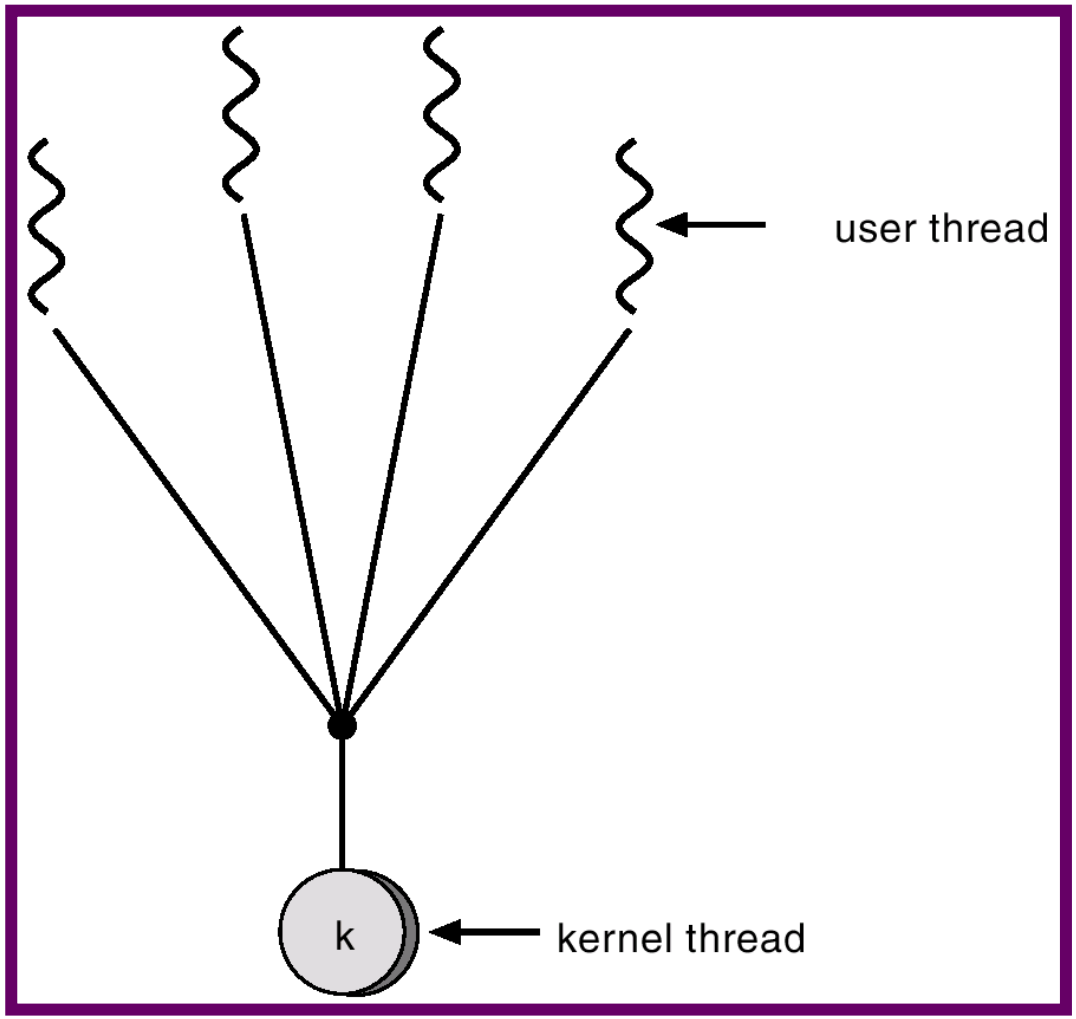
Threads de usuário x kernel

- Implementações com compromissos diferentes
- *Threads* no nível de usuário (bibliotecas)
  - Mais “leves”, pois se limitam ao programa
- *Threads* de kernel
  - Melhor integradas ao escalonador do S.O.
  - Mais *pesadas*
- 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
  - Pacote de threads inserido totalmente no espaço do usuário, incluindo sincronização (ex. exclusão mutua)
  - Executam no topo de um sistema supervisor
  - Cada processo possui uma tabela de threads

Mapeamento muitos-para-um

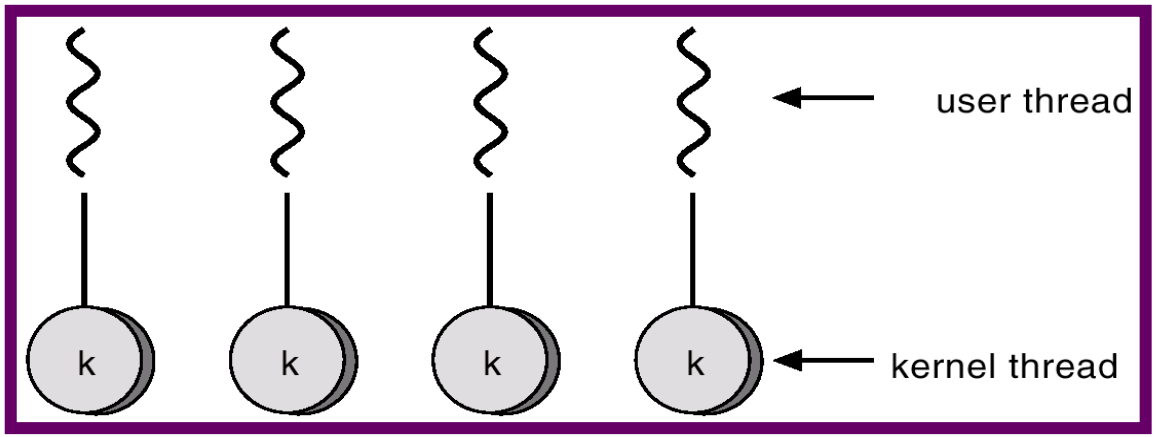


Mapeamento muitos-para-um

- Vantagens
  - São portáveis, por não precisarem de APIs de gerenciamento de SOs
  - Threads podem ser implementadas em SOs que não reconhecem threads
  - Cada processo com seu algoritmo de escalonamento
  - Não é necessário chaveamento entre modo sistema e usuário para criação e destruição de threads (mais leve)
- Desvantagens
  - Implementação das chamadas ao sistema bloqueante (uma thread espera por E/S de forma bloqueante, todas as outras ficam bloqueadas)

Mapeamento um-para-um

- Kernel reconhece *threads* e cada *thread* de usuário é na verdade uma *thread* de kernel
- Mapeamento é simples, mas “pesado”
- Tabela de thread única
- Criação e destruição através de chaveamento entre modo usuário e kernel



Modelo muitos-para-muitos

- Mapeamento flexível entre modo usuário e kernel
- O S.O. cria um número “suficiente” de *thread*
- 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