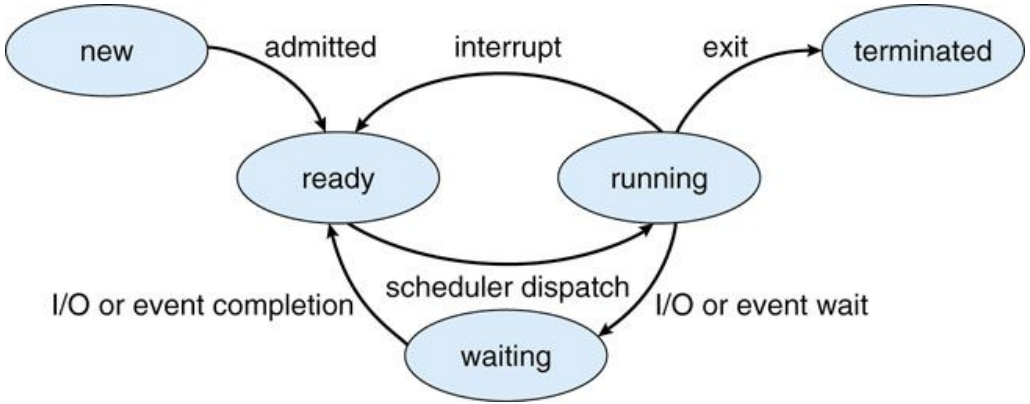
SO: Threads

Sistemas Operacionais
2017-1
Flavio Figueiredo (http://flaviovdf.github.io)

Na aula passada...

- Passamos por processos
- Falamos rapidamente de escalonamento
- Vimos o ciclo de vida de um processo



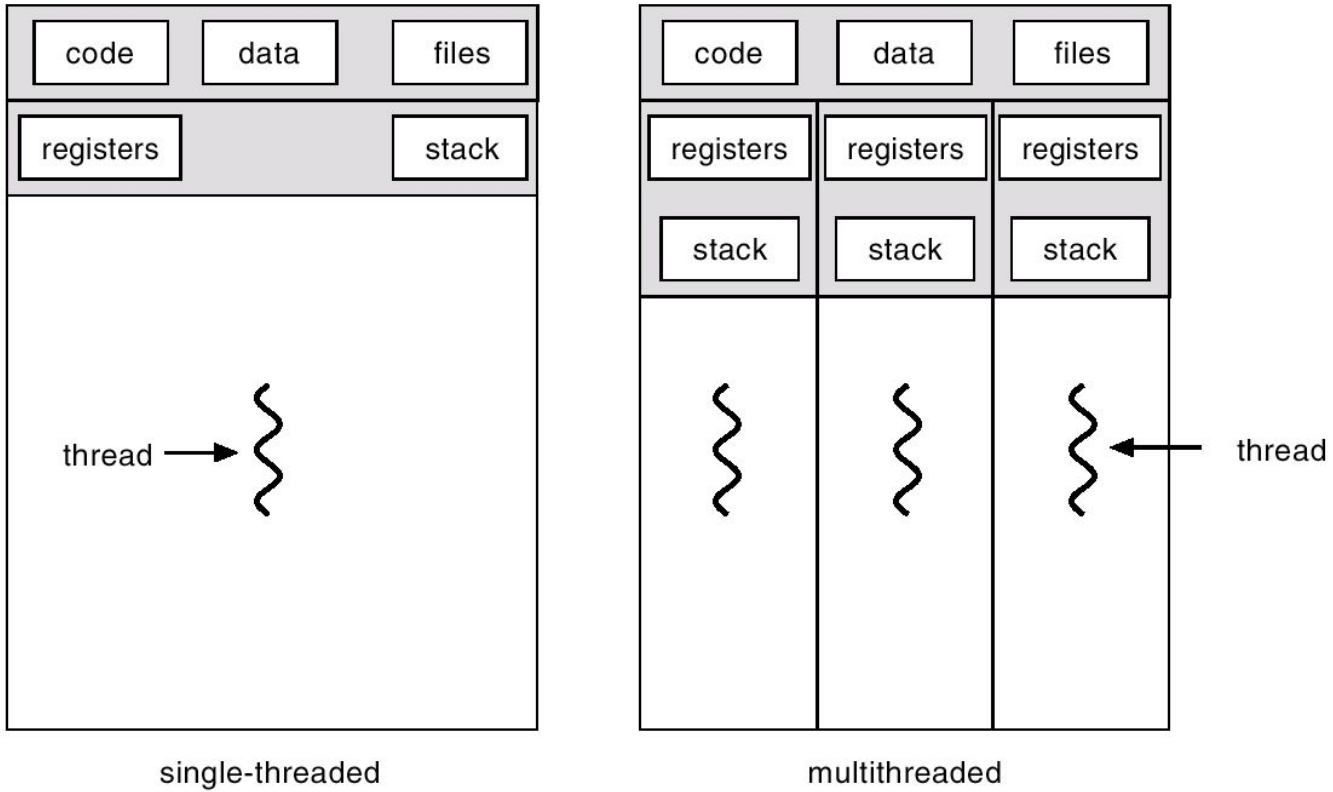
O que são Threads?

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

Threads


code
data
files
registers
stack
thread
single-threaded
code
data
files
registers
registers
registers
stack
stack
stack
thread
multithreaded

Quais são as vantagens de usar threads?

Threads

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

Uma thread pode ser pensada como um “processo leve”

Lembrando de primitivas de sincronização, como você implementaria uma biblioteca de threads?

Imagine que você está em um SO limitado, sem ter multiprocessamento, como o MS-DOS.

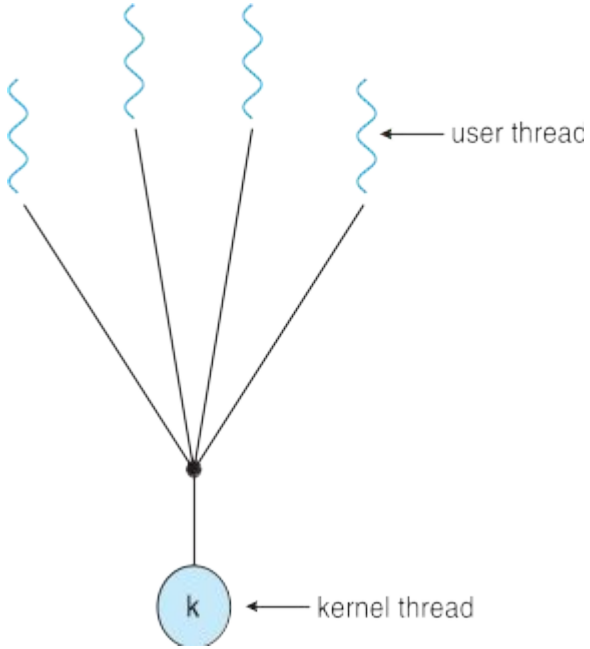
Dois Tipos de Threads

- Kernel Threads
  - Melhor integradas ao escalonador do S.O.
  - Mais overhead

- Green Threads (Nível de Usuário)
  - Mais “leves”, pois overhead se limita ao programa
  - Se o kernel não reconhece, pode ser ineficiente

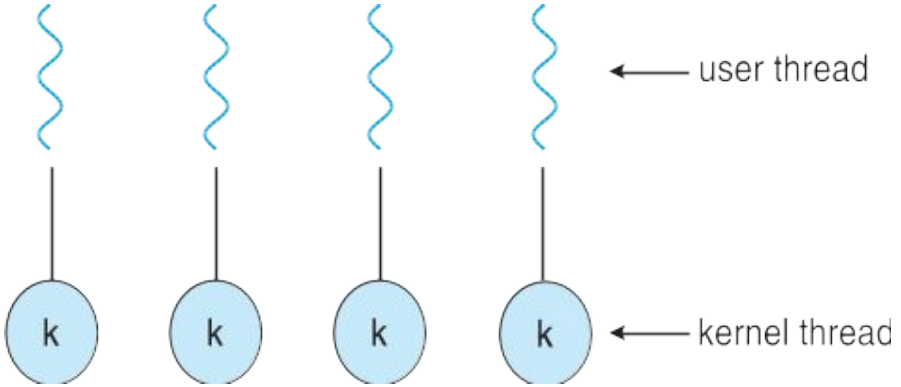
User Threads

- O kernel escalona o seu processo em uma *kernel thread*
- Muitas threads de usuário mapeadas para uma única thread de kernel
  - Ocorre, p.ex., quanto o S.O. não reconhece threads, apenas processos
  - Muitos para um
- Qual a consequência de uma system call nas threads?



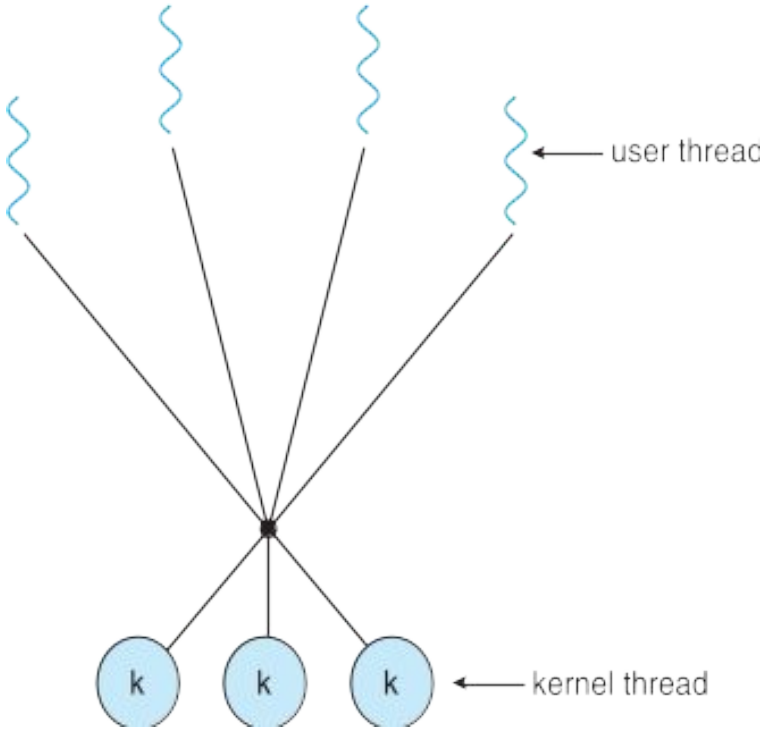
Kernel Threads

- Mapeamento um-para-um
- O kernel sabe o que é uma thread
- Threads são escalonadas diretamente igual aos processos
- Mapeamento é simples, mas “pesado”
  - Toda troca de threads envolve mudança p/ modo protegido
  - Lembre-se da troca de contexto de processos
- Windows, Solaris e Linux são todos assim



Modelo Muitos para Muitos

- Pouco utilizado
- Tenta unir o melhor de 2 mundos
- Threads de usuários leve são mapeadas para um número fixo de threads do kernel



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 em Java

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

Threads em Python

- Python permite threads

- Porém:
  - Python é interpretada e tem um lock global no interpretador
  - Ou seja, o interpretador não pode executar várias threads de uma vez
  - Ganhos apenas de E/S

- Alternativas
  - Bibliotecas de multiprocessos

Aspectos de Implementação

- Um sistema que implementa threads deve definir como lidar com aspectos de execução de processos

- Tratamento de sinais
  - Fork/exec
  - Dados privativos

Bibliotecas de Threads Implícitas

- Bibliotecas que esconde do usuário o uso de threads

- OpenMP
  - laços em paralelo

- Intel Math Kernel Library
  - Primitivas de algebra linear
  - https://software.intel.com/en-us/intel-mkl