Threads

Sistemas Operacionais

Charles Tim Batista Garrocho

Instituto Federal de São Paulo – IFSP
Campus Campos do Jordão

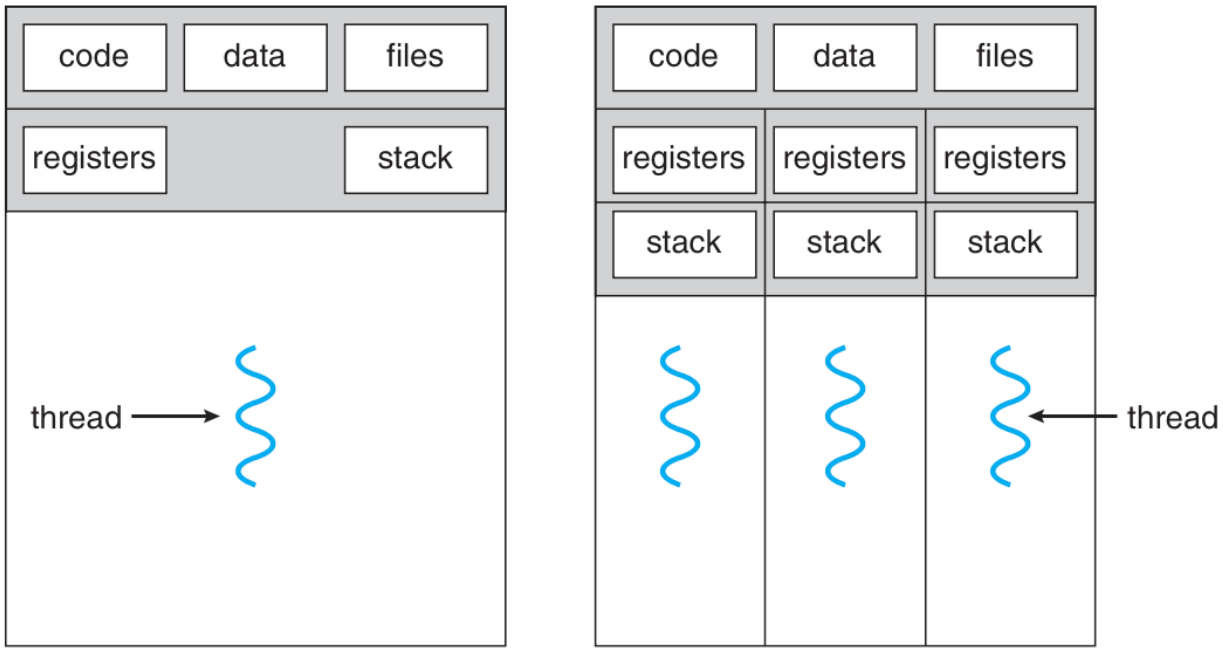
charles.garrocho.com/OSO

charles.garrocho@ifsp.edu.br

Técnico em Informática

Visão Geral

Os processos podem ter mais de um **fluxo de execução**. Cada fluxo de execução é chamado de **thread**.
Um processo tradicional tem um **único thread** de controle. Se o processo possui **múltiplos threads** de controle, ele pode realizar mais do que uma tarefa a cada momento.



single-threaded process

multithreaded process

Benefícios

Os benefícios do uso de threads são:

- **Capacidade de resposta**: ela permite que um processo fique executando, mesmo que uma parte desse processo esteja bloqueada. Isto é possível desde que haja threads independentes. Isso aumenta a capacidade de resposta dos processos;
- **Compartilhamento de recursos**: os threads de um processo podem compartilhar os recursos do mesmo, incluindo memória;
- **Economia**: a alocação de recursos e memória a diversos processos possui um custo computacional muito alto. Então é mais atrativa a implementação de multithreads, pois seu custo computacional é menor;
- **Utilização de arquiteturas multiprocessador**: em uma arquitetura multiprocessador, onde cada thread pode ser executado em um processador diferente;

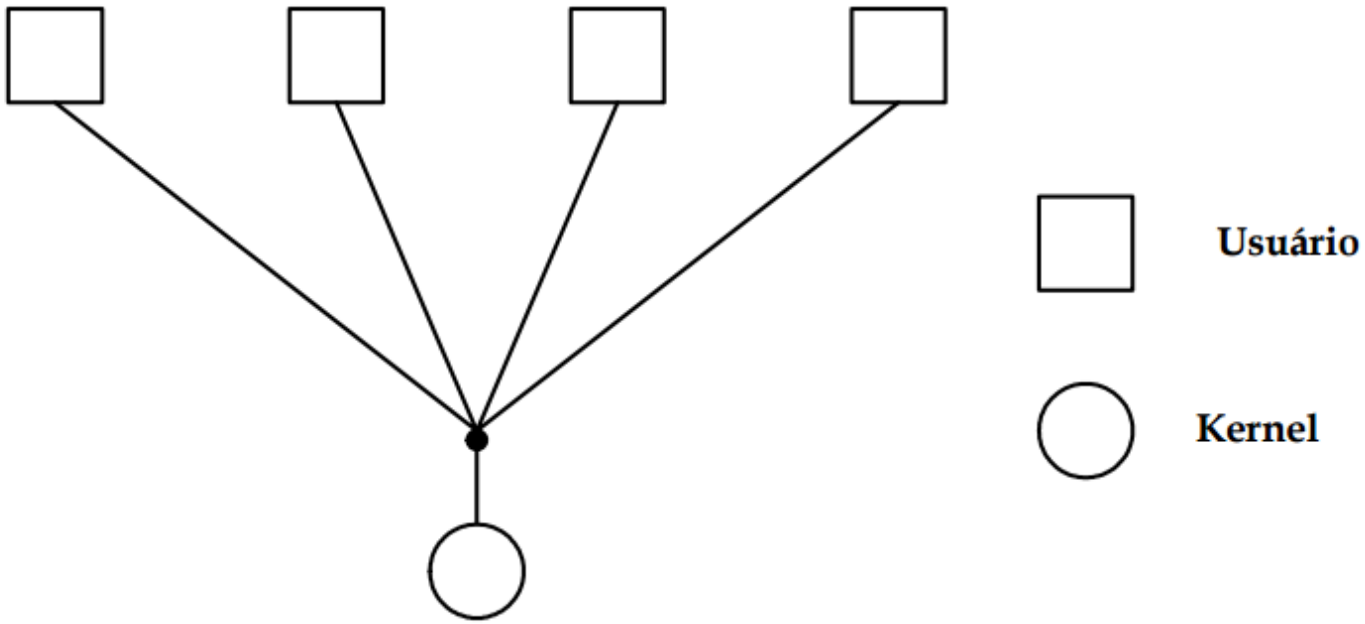
Threads de Usuário e de Kernel

Threads de Usuário estão localizados em um nível superior ao kernel e são implementados através de uma biblioteca que fornece suporte à criação, escalonamento e gerência sem a intervenção do kernel. Eles são mais rápidos e fáceis de gerenciar. Entretanto, se um thread fizer uma chamada bloqueante, todo o processo estará bloqueado, pois o kernel não tem acesso ao mesmo.

Threads de Kernel são suportados diretamente pelo sistema operacional. Todo o gerenciamento dos threads é feito pelo kernel. Se um thread fizer uma chamada bloqueante, o kernel pode chamar outro thread para execução.

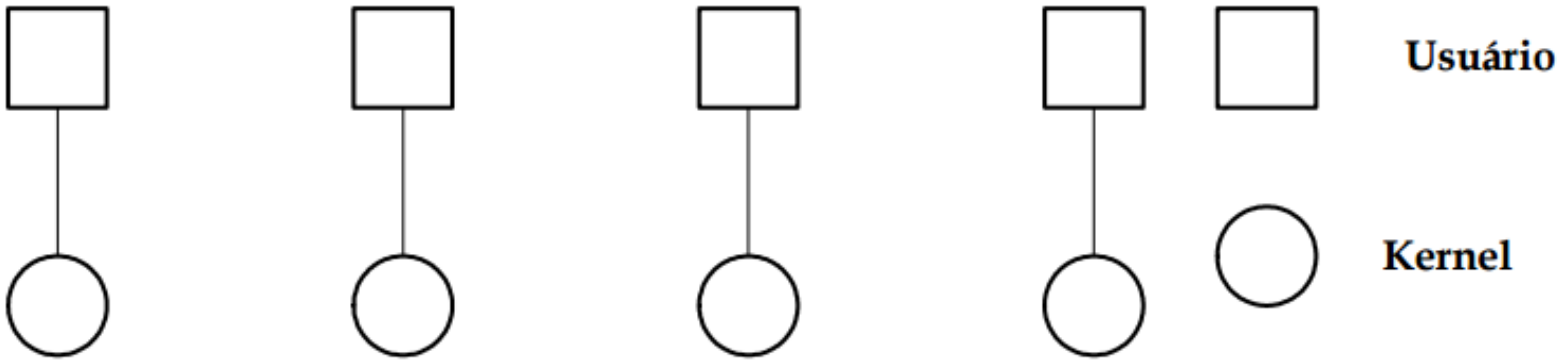
Modelos de Multithreading: Muitos-Para-Um

Vários threads de usuário são mapeados em um único thread de kernel. A gerência de threads fica no nível de usuário, o que o torna mais rápido, porém o processo inteiro será bloqueado em caso de um thread fazer uma chamada bloqueante. Além disso, mesmo em sistemas multiprocessadores não será possível executar multithreads porque o kernel só pode ser acessado por um thread por vez.



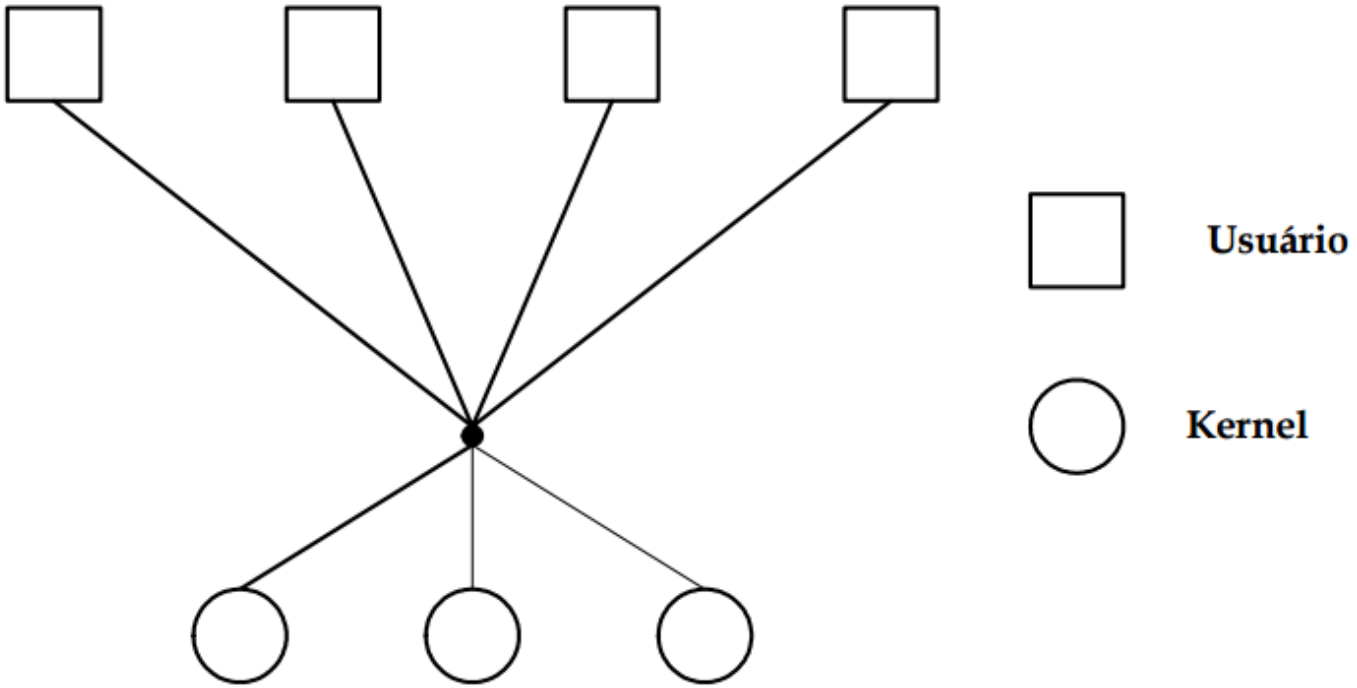
Modelos de Multithreading: Um-Para-Um

Cada thread de usuário é mapeado em um thread de kernel. Há uma maior concorrência que no modelo muitos para um, pois permite que assim que thread execute uma tarefa bloqueante, um outro seja chamado à execução. Entretanto, para cada thread de usuário criado é preciso criar um thread de kernel, o que implica em uma queda de desempenho do sistema.



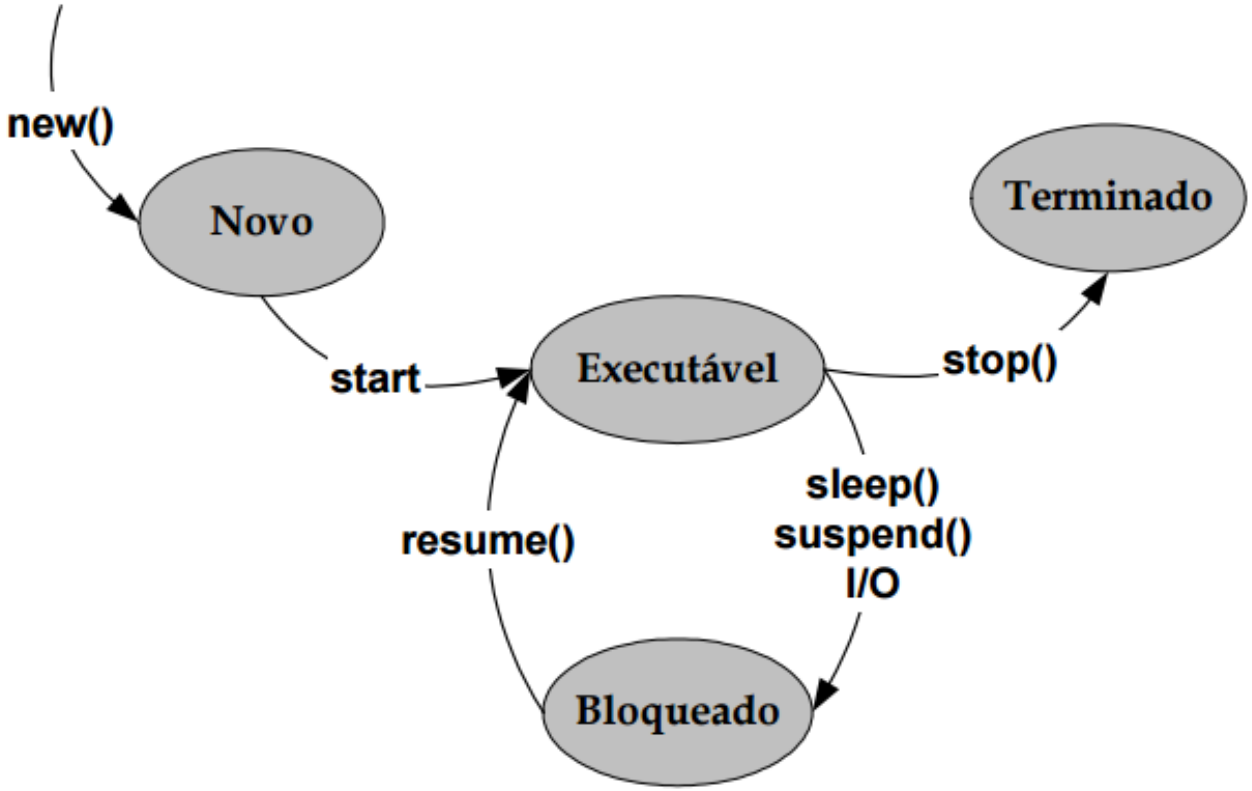
Modelos de Multithreading: Muitos-Para-Muitos

Threads de usuários são multiplexados em um número menor de threads de kernel. O número de threads de kernel pode ser específico para cada aplicação. No modelo Muitos-para-Um não é permitida a concorrência, pois uma chamada bloqueante não permite um outro thread ser chamado. Já no modelo um para um, apesar de permitir a concorrência, é preciso ter muito cuidado, pois não é recomendado criar muitos threads em uma única aplicação, implicando em muitos threads de kernel.



Estados de um Thread

Assim como os processos, um thread pode estar em uma série de estados.



Exercícios

1. Qual a diferença entre processos e threads?
2. Descreva uma aplicação que utilize múltiplas threads para o seu funcionamento. Como seria o funcionamento desta aplicação se ela fosse implementada em uma única thread?
3. Qual a maior vantagem de implementar threads no espaço do usuário? Qual a maior desvantagem?
4. Qual a maior vantagem de implementar threads no espaço do núcleo do sistema operacional? Qual a maior desvantagem?
5. Cite três exemplos de operações que fazem um processo transitar do estado *em execução* para o estado *bloqueado*.
6. Cite os estados e transições de um processo.