# Aula 8: Threads

Como foi exposto anteriormente, os processos podem ter mais de um fluxo de execução. Cada fluxo de execução é chamado de *thread*.

## Visão Geral

Um **thread** é uma unidade básica de utilização da CPU que compreende um ID, um contador de programa, um conjunto de registradores e uma pilha. Compartilha, com outros *threads* pertencentes ao mesmo processo, uma seção de código, sua seção de dados e outros recursos do sistema operacional, como arquivos abertos e sinais.

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. A Figura 1 ilustra a diferença entre um processo com um único *thread* tradicional e um processo com múltiplos *threads*.

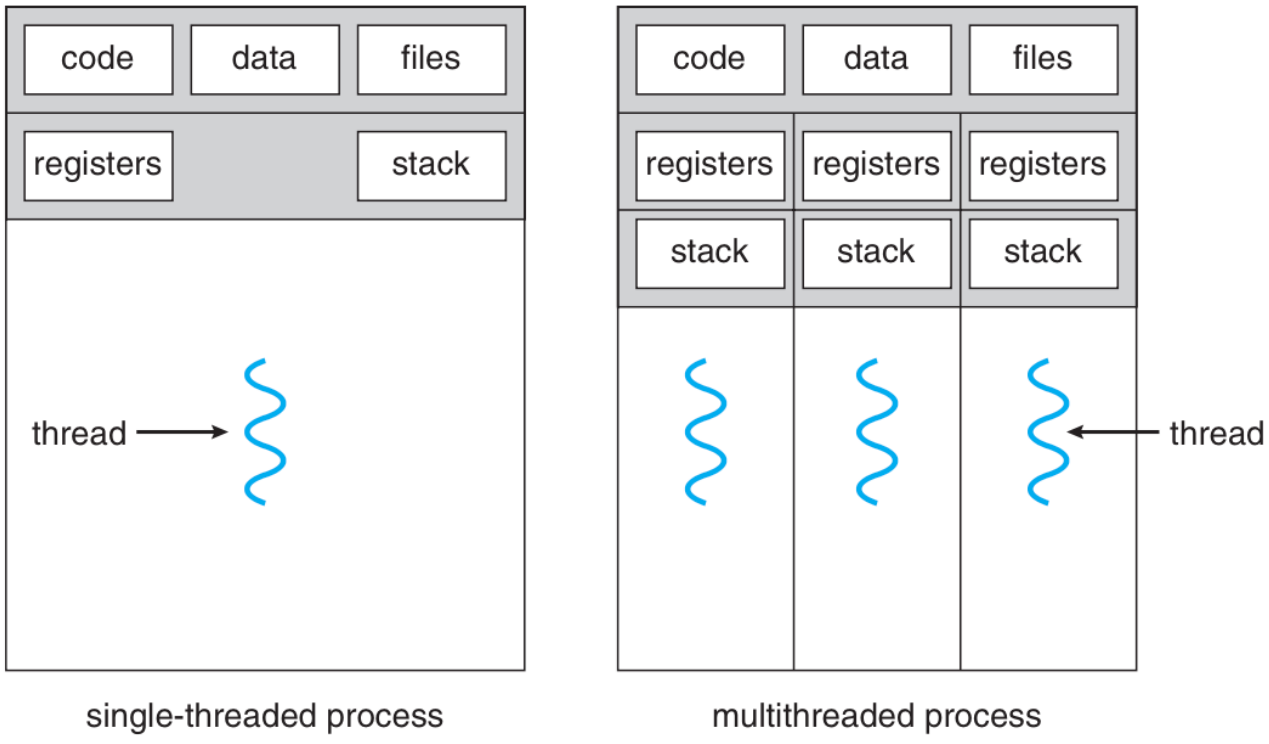


**Figura 1**: Processos com *thread* único e com *multithreads*.

Muitos programas que operam em computadores são **multithreads**. Um programa normalmente é implementado como um processo separado com diversos *threads* de controle. Um navegador Web pode ter um *thread* para exibir imagens ou texto enquanto um outro *thread* recupera

dados de uma rede. Se um servidor operasse como um processo tradicional **com único thread**, ele seria capaz de servir somente um cliente de cada vez. O montante de tempo que um cliente teria que esperar para que sua solicitação fosse atendida poderia ser enorme.

# 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 Usuários

Esses threads estão localizados em um nível superior ao kernel e são implementados através de uma biblioteca de threads no nível de usuários. Esta biblioteca fornece suporte à criação, escalonamento e gerência de threads sem a intervenção do kernel. Eles são mais rápidos e fáceis de gerenciar.

Entretanto, há uma séria desvantagem em usar threads de usuário. Se um thread fizer uma chamada bloqueante, todo o processo estará bloqueado, pois o kernel não tem acesso ao mesmo.

# Threads de Kernel

Estes threads 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 sistemas oferecem suporte tanto para threads de usuário quanto do kernel, resultando em diferentes modelos de geração de multithreads.

## Muitos-Para-Um

No modelo 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. A Figura 2 ilustra um exemplo desse modelo.

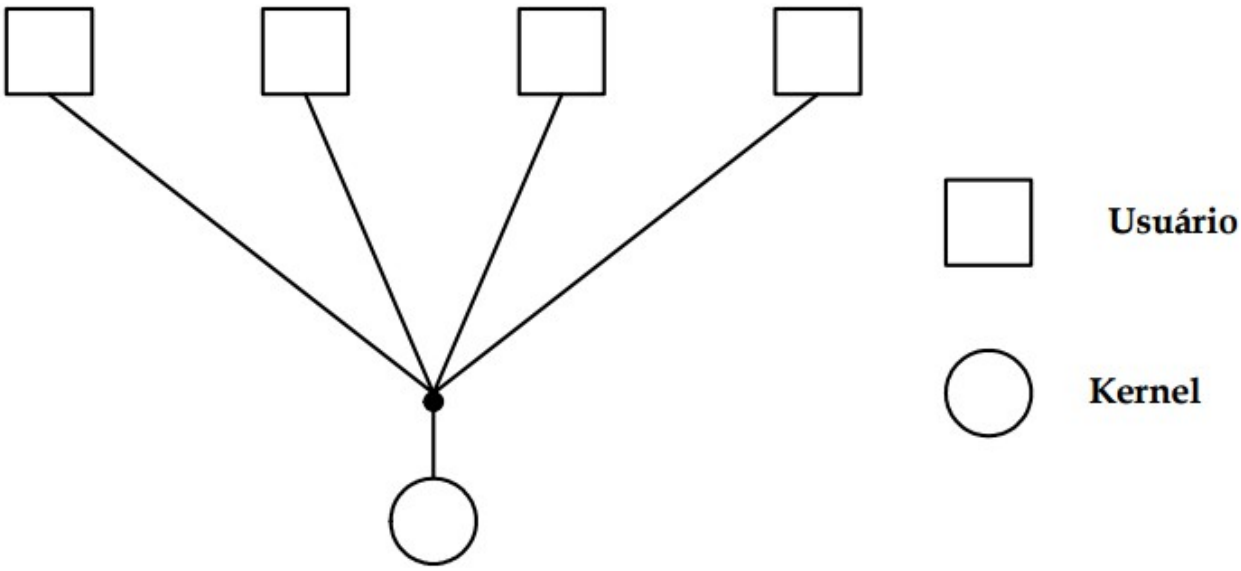


**Figura 2**: Muitos para Um.

## Um-Para-Um

Neste modelo, 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. Este modelo é aplicado nos sistemas Windows NT e OS/2. A Figura 3 ilustra um exemplo de modelo um para um.

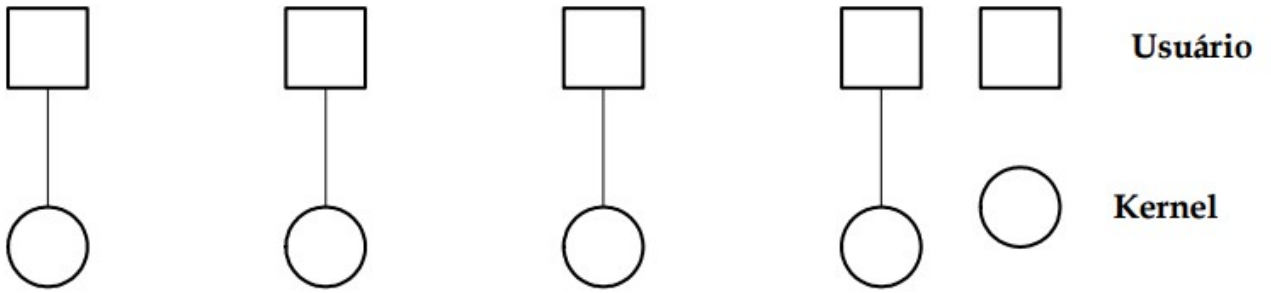


**Figura 3**: Um para Um.

## Muitos para Muitos

Neste modelo, os 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 verdadeira 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. O modelo “Muitos para Muitos” não apresenta estas desvantagens. O Sistema Solaris, IRIX e Digital UNIX utilizam esse modelo multithreading.

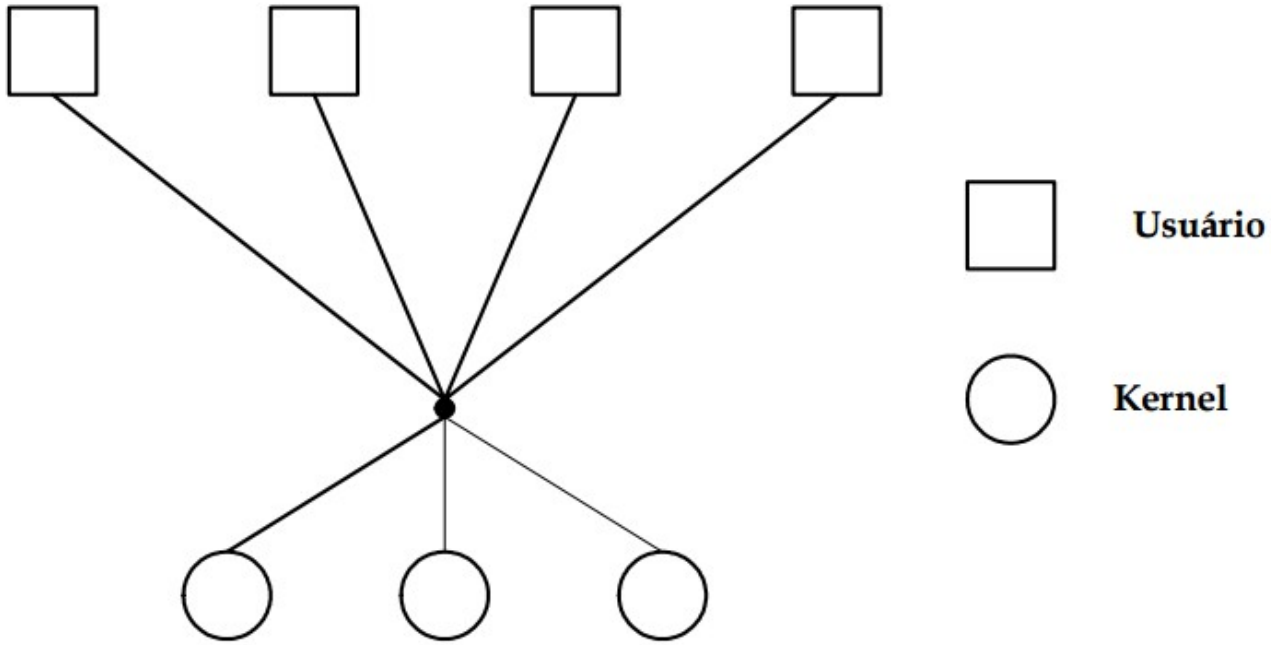


**Figura 4**: Muitos para Um.

# Estados de um Thread

Assim como os processos, um thread pode estar em uma série de estados que são descritos a seguir e ilustrados na Figura 5:

- **Novo**: um thread está nesse estado quando um objeto para o thread é criado: método new();
- **Executável**: chamar os método start() aloca memória para o novo thread na Máquina Virtual Java e chama o método run() para o objeto thread. Quando o método run() de um thread é chamado, o thread passa do estado Novo para Executável, onde pode ser executado pela Máquina Virtual Java. Não existe distinção entre um thread passível de ser executado e um em execução;
- **Bloqueado**: para entrar nesse estado, o thread deve executar alguma instrução bloqueante como operações de I/O ou usar métodos suspend() ou sleep();
- **Terminado**: o método run() do thread termina ou é executado o método stop().

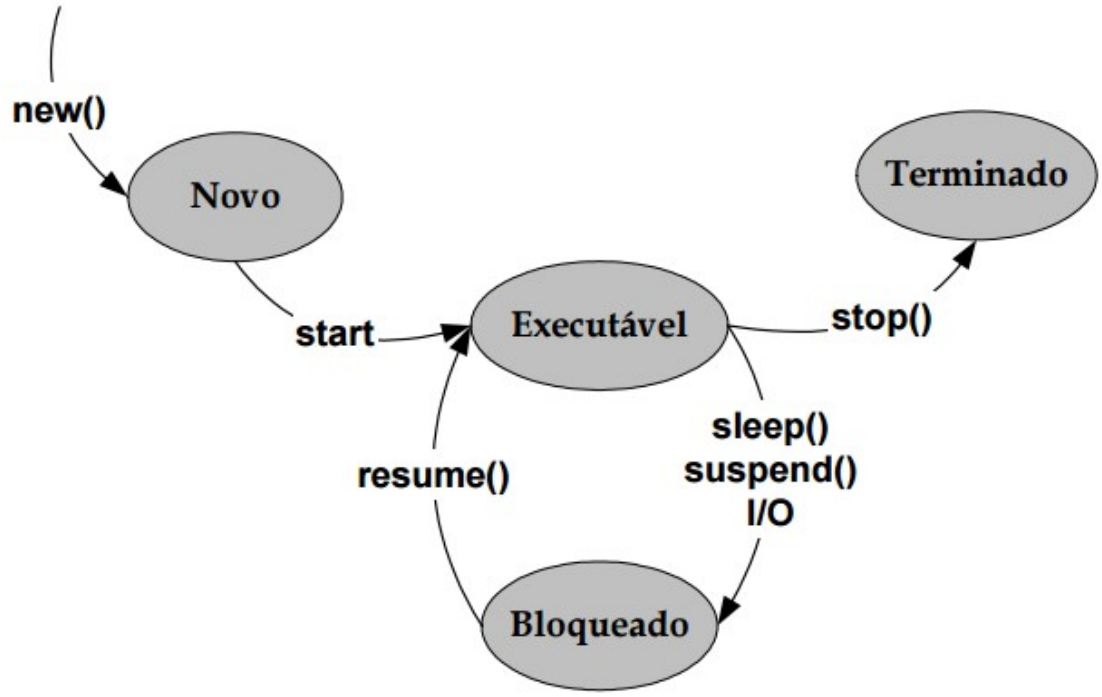


**Figura 5**: Estados de um Thread.

# 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.