Sistemas Operacionais

Prof. Robson de Souza

Aulas 11 e 12

Conteúdo: Processos e Threads

Continuação de Threads …

Existem alguns modos de se implementar e utilizar threads, os três principais são:

Modo usuário → São oferecidos por uma biblioteca de rotinas fora do núcleo do s.o.

Modo kernel → São threads oferecidos pelo próprio núcleo do s.o.

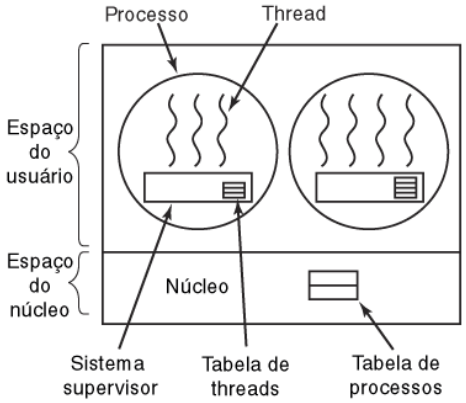
Modo híbrido → Combinação de ambos os modelos.

Em 1995 foi aprovado o padrão POSIX P1003.1c, posteriormente atualizado para POSIX P1003.4a. Conjunto de threads largamente usado em ambientes UNIX. Conhecidos também como Pthreads.

Os threads em **modo usuário** são implementados pela aplicação e não pelo sistema operacional, com isso, é responsabilidade da aplicação manter, sincronizar e escalonar threads. A vantagem disso é a possibilidade de poder trabalhar com threads em ambientes operacionais monothreads.

Esses threads são Gerenciados por biblioteca de threads no nível do usuário. Existem três bibliotecas de threads principais:

- POSIX Pthreads
- Threads Java
- Threads Win32

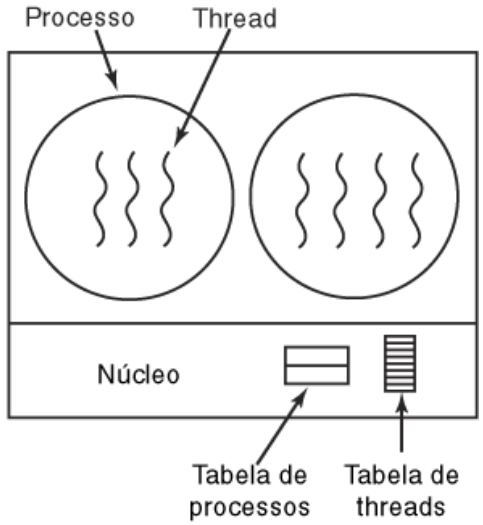


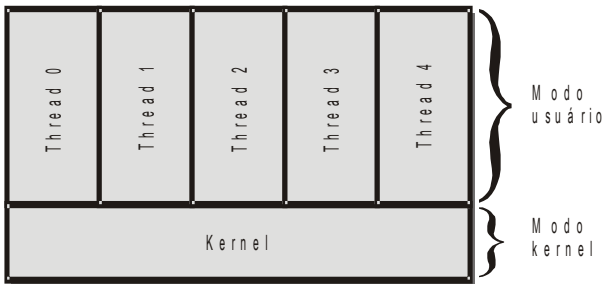
Os threads em **modo kernel**, são implementados diretamente pelo núcleo do s.o. Com isso, a tarefa de escalonar e sincronizar os threads fica com o S.O. Em ambientes com multiprocessadores cada thread pode ser executado por um processador.

Nesse caso, os threads são suportados pelo kernel do Sistema Operacional, por exemplo:

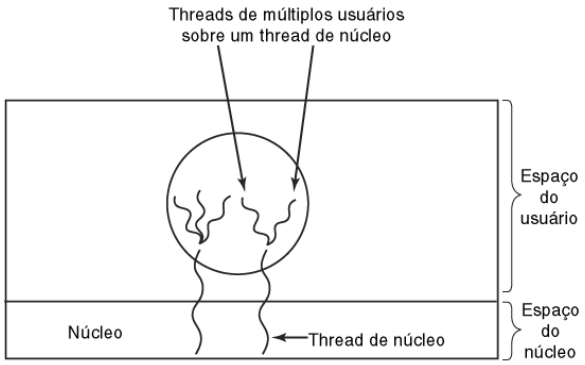
- Windows XP/2000

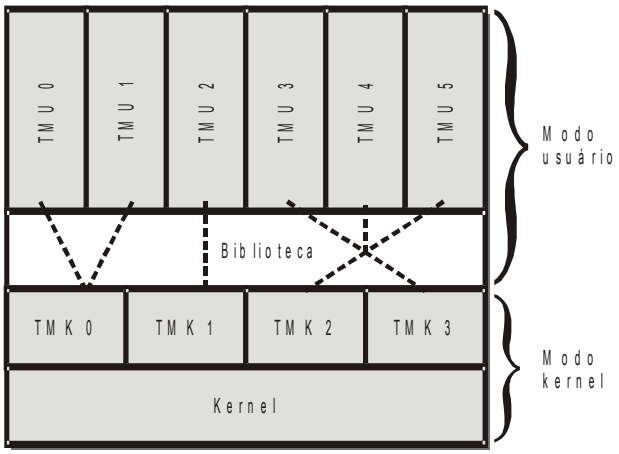
- Solaris
- Linux
- Tru64 UNIX
- Mac OS X





Os threads em **modo híbrido** são uma combinação dos modos anteriores. Um processo pode ter vários threads no modo kernel e por sua vez um thread neste modo pode ter vários threads no modo usuário.

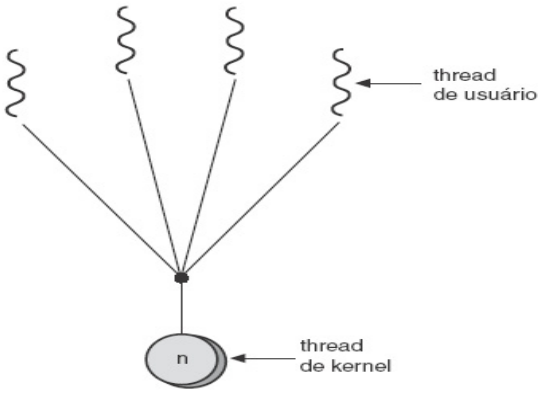




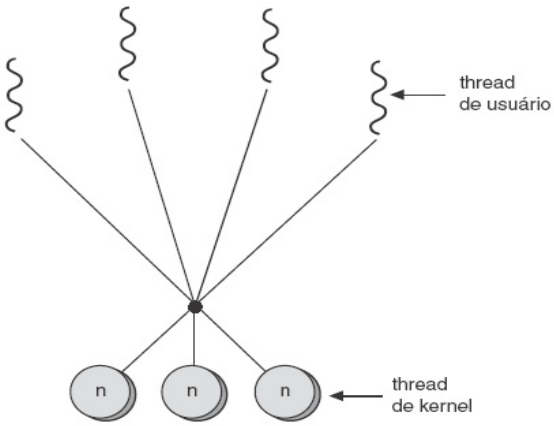
No modo kernel existem três modelos de combinação de threads: **Muitos-para-Um**, **Um-para-Um** e **Muitos-para-Muitos**.

No modelo Muitos-para-Um, Muitos threads no nível do usuário estão associados a um único thread de kernel.
No modelo Um-para-Um, cada thread de usuário está associado a um thread de kernel.
O modelo Muitos-para-Muitos permite que muitos threads no nível do usuário sejam associados a muitos threads no nível do kernel, também permite que o sistema operacional crie um número suficiente de threads de kernel.

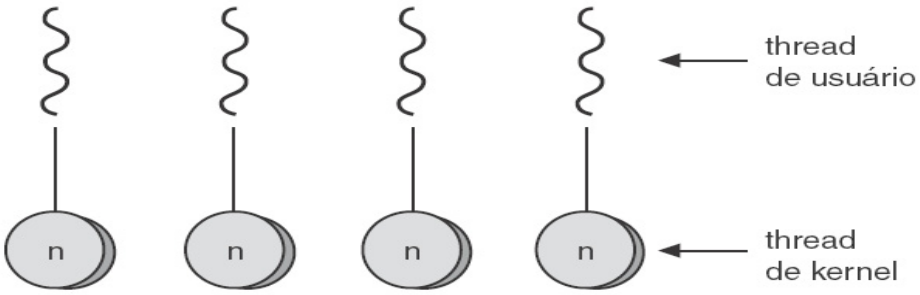
Muitos-para-Um



Muitos-para-Muitos



Um-para-Um



O thread tem um contador de programa, que mantém o controle de qual instrução deve ser executada em seguida, possui registradores, que contém as suas variáveis de trabalho atuais e apresenta uma pilha que contém o histórico de execução.

Como os threads possuem exatamente o mesmo espaço de endereçamento, threads distintos não são tão independentes quanto processos distintos. Threads compartilham as mesmas variáveis globais.

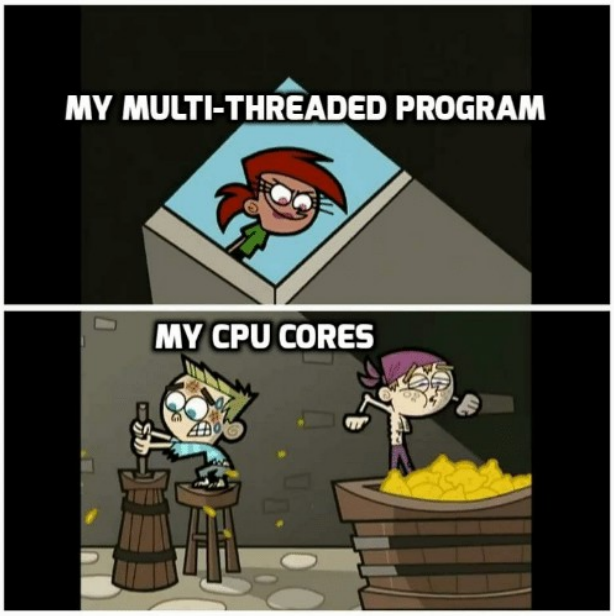
Um thread pode ler, escrever e até apagar completamente a pilha de outro thread, é impossível ter proteção entre threads e é desnecessário, porque os threads tem o objetivo de cooperar e não competir, diferentemente de um processo que é propriedade de um usuário, sendo que diferentes usuários ou aplicações podem ser hostis e competir entre si por um determinado recurso.

A unidade de gerenciamento de recursos é o processo e não o thread, se um thread abre um arquivo por exemplo, este fica visível para os outros threads, que podem alterar o arquivo. O objetivo dos threads é a capacidade de compartilhar um conjunto de recursos, de forma que os threads possam cooperar na realização de uma tarefa.

Quando ocorre a execução de múltiplos threads, os processos normalmente iniciam com um único thread. Esse thread tem a capacidade de criar novos threads chamando uma rotina de biblioteca. Não é necessário e

nem possível especificar qualquer coisa sobre o espaço de endereçamento do thread em criação, isso é executado automaticamente. Pode existir uma hierarquia em threads, mas normalmente essa hierarquia não existe e todos são iguais. Em caso de hierarquia, o thread em criação possui o identificador do thread que o criou.

Existem rotinas de biblioteca para os threads executarem, como rotinas para terminar a execução de um thread, criar um thread, esperar uma saída de um outro thread ou até mesmo desistir voluntariamente da CPU para deixar outro thread executar.



https://me.me/i/my-multi-threaded-program-my-cpu-cores-21289917

Referências bibliográficas:

TANENBAUM, Andrew. 2ª ed. **Sistemas Operacionais Modernos,** Editora Pearson, 2003.

SILBERSCHATZ, Abraham. **Sistemas Operacionais com JAVA,** 6ª ed. Editora Campus

MACHADO, Francis B. **Arquitetura de Sistemas Operacionais,** 4ª ed, LTC, 2007.