Threads

Prof. Edwar Saliba Júnior
Março de 2007

Definição

- **Partes de um processo que compartilham mesmo espaço de endereçamento**
- **Sub-rotina de um programa executada paralelamente ao programa chamador (execução concorrente de sub-rotinas)**
- **Exemplos:**
  - ***Threads* em Java, Delphi e etc.**

Outra Definição

- ***Enquanto processos permitem que o sistema operacional execute mais de uma aplicação ao mesmo tempo, as Threads permitem que a aplicação execute mais de um método (parte da aplicação, sub-rotina) ao mesmo tempo.***

Vantagens e Desvantagens

- **Vantagens:**
  - **Desempenho:**
    - **Não existe necessidade de comunicação entre processos**
    - **Utilização de multiprocessadores para um mesmo processo**
    - **Um programa pode continuar sendo executado mesmo se parte dele estiver bloqueada (um navegador pode permitir a interação do usuário em uma *thread* enquanto uma imagem é carregada em outra *thread*);**

- **Desvantagem: O desenvolvimento de aplicações que fazem uso de *Threads* não é simples.**

Exemplos

- **Navegador Web:**
  - **Uma *thread* para exibir imagens**
  - **Uma *thread* para recuperar dados da rede;**
- **Processador de texto:**
  - **Uma *thread* para sequência de teclas digitadas;**
  - **Uma *thread* para verificação ortográfica e gramatical.**

É bom saber...

- **Para obter benefícios no uso de *Threads*, a aplicação deve permitir que partes do seu código sejam executadas em paralelo de forma independente.**

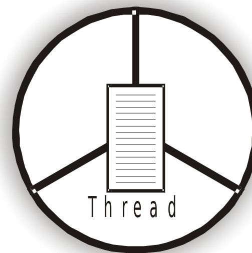
Pergunta-se…

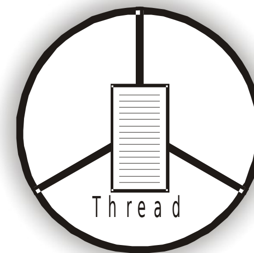
- **Ao invés de *Threads*, por que não dividir a aplicação em vários outros processos?**
  - **A criação de um processo e a mudança de contexto consomem muito mais recursos do que a criação de uma *thread*;**
  - **Como cada processo possui espaço de endereçamento próprio, a comunicação entre processos é difícil e lenta.**

Mono e Multithread

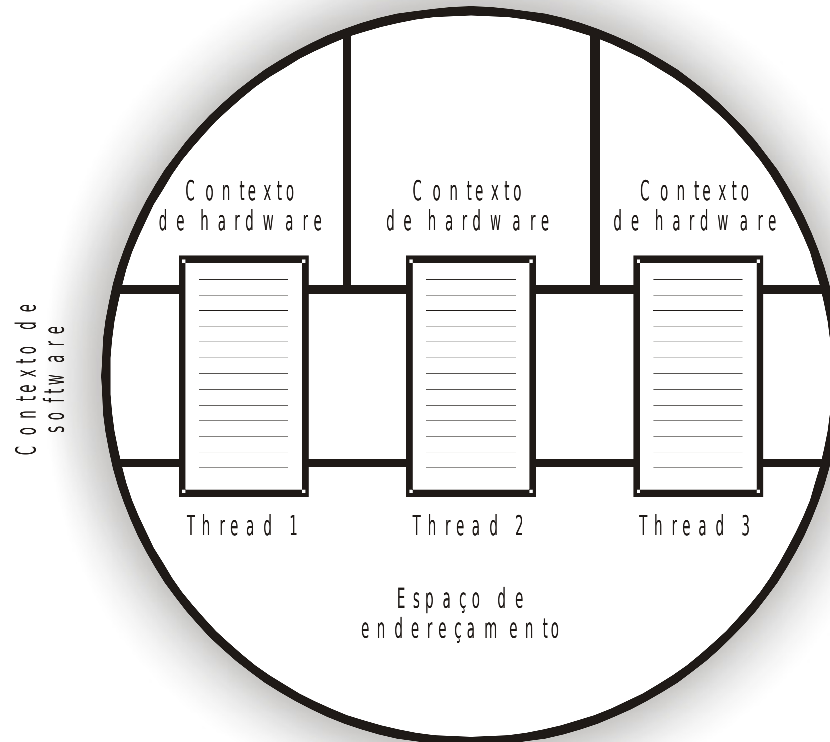
- Os sistemas que suportam apenas uma única *thread* de execução, são chamados de *monothread.* E aqueles que suportam múltiplas *threads,* são chamados de *multithread;*

- **Exemplos de sistemas *Monothreads*:**
  - **MS-DOS e primeiras versões do Windows.**



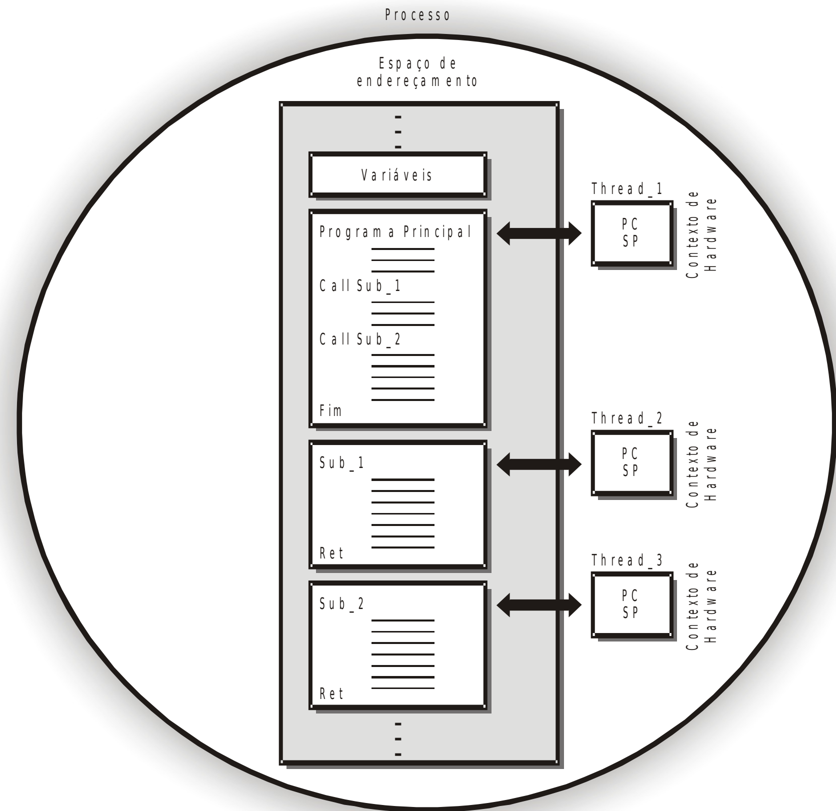


Ambiente Multithread



Ambiente Multithread

- **Cada processo possui pelo menos uma *thread***
- **Exemplo: programa com duas sub-rotinas independentes.**



Ambiente *Multithread*

- **Processo fica responsável pela alocação de recursos, mas a unidade escalonada é a *thread*;**
- ***Threads* de um mesmo processo compartilham contexto de *software* e espaço de endereçamento, mas não compartilham contexto de *hardware*;**
- **TCB: armazena contexto de *hardware* e informações sobre a *thread* (prioridade, estado...);**
- **O TCB está para a *thread* assim como o PCB está para o processo.**

Ambiente *Multithread*

- **Como o espaço de endereçamento é único, a aplicação precisa implementar a sincronização para garantir o acesso seguro à memória.**

Ambientes *Multithreads*

- ***Threads* são extremamente utilizadas em ambientes cliente/servidor. Ex: SGBD;**
- **Um único processo no servidor gera uma *thread* para cada solicitação de cliente**;
- **Lembra da arquitetura Microkernel?**
  - **Pode ser implementada com o uso de *threads* (cada módulo servidor é executado em uma *thread*).**

Ambientes *Multithread*

Processo servidor

Solicitações

Thread

Thread

Thread

Processo cliente

Processo cliente

Processo cliente

Arquitetura

- *Threads* podem ser oferecidos por uma biblioteca de rotinas fora do núcleo do sistema operacional (modo usuário), pelo próprio núcleo do sistema (modo kernel), por uma combinação de ambos (modo híbrido) ou por um modelo conhecido como *Scheduler Activations*;
- A tabela abaixo mostra as diversas arquiteturas para diferentes ambientes operacionais:

Arquitetura

Ambientes	Arquitetura
Distributed Computing Environment	Modo usuário
Compaq OpenVMS versão 6	Modo usuário
Microsoft Windows 2000	Modo kernel
Compaq Unix	Modo kernel
Compaq OpenVMS versão 7	Modo kernel
Sun Solaris versão 2	Modo híbrido
University of Washington Fast Threads	Scheduler activations

Arquitetura

- **Varia de acordo com a implementação do pacote de rotinas necessárias para que a aplicação utilize as *threads*;**

- **<u>Modo Usuário</u>: A responsabilidade pela gerência e sincronização das *threads* é da aplicação; utilizado em sistemas operacionais que não suportam *threads* (pseudo-multithread).**

Arquitetura - Modo Usuário

- TMU possuem uma grande limitação, pois o sistema operacional gerencia cada processo como se existisse apenas um único *thread*. No momento em que um *thread* chama uma rotina do sistema que o coloca em estado de espera (rotina bloqueante), todo o processo é colocado no estado de espera, mesmo havendo outros *threads* prontos para execução.

Arquitetura - Modo Usuário

- Um dos maiores problemas na implementação de TMU é o tratamento individual de sinais. Como o sistema reconhece apenas processos e não *threads*, os sinais enviados para um processo devem ser reconhecidos e encaminhados a cada *thread* para tratamento;
- No caso do recebimento de interrupções de *clock*, fundamental para implementação do tempo compartilhado, esta limitação é crítica. Neste caso, os sinais de temporização devem ser interceptados, para que se possa interromper o *thread* em execução e realizar a troca de contexto.

Arquitetura - Modo Usuário

- Em ambientes com múltiplos processadores, não é possível que múltiplos *threads* de um processo possam ser executados em diferentes CPU's simultaneamente, pois o sistema seleciona apenas processos para execução e não *threads;*
- Esta restrição limita drasticamente o grau de paralelismo da aplicação, já que os *threads* de um mesmo processo podem ser executados em somente

  um processador de cada vez.

Arquitetura

- **<u>Modo Kernel</u>: Sistema operacional provê rotinas para gerenciamento das *Threads* (necessita de mudanças nos modos de acesso).**

Arquitetura – Modo Kernel

- *Threads* em modo kernel (TMK) são implementados diretamente pelo núcleo do sistema operacional, através de chamadas a rotinas do sistema que oferecem todas as funções de gerenciamento e sincronização;
- O sistema operacional sabe da existência de cada *thread* e pode escaloná-los individualmente;
- No caso de múltiplos processadores, os *threads* de um mesmo processo podem ser executados simultaneamente.

Arquitetura – Modo Kernel

- O grande problema para pacotes em modo kernel é o seu baixo desempenho;
- Enquanto nos pacotes em *modo usuário* todo tratamento é feito sem a ajuda do sistema operacional, ou seja, sem a mudança do *modo de acesso*, pacotes em *modo kernel* utilizam chamadas a rotinas do sistema e várias mudanças no modo de acesso.

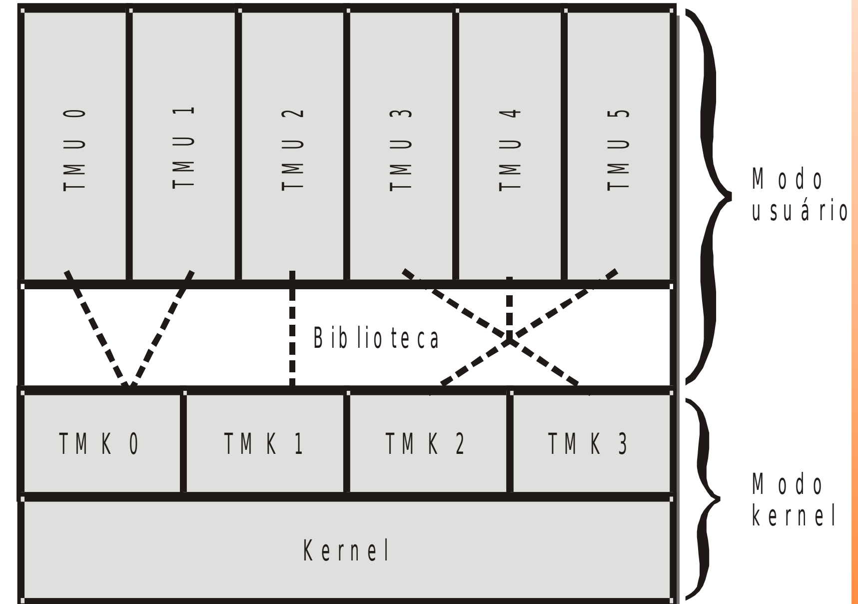
Arquitetura – Modo Kernel

- A tabela abaixo compara o desempenho de duas operações distintas envolvendo a criação, escalonamento, execução e eliminação de um processo/*thread*:

Implementação	Operação 1 (µS)	Operação 2 (µs)
Subprocessos	11.300	1.840
Threads em modo kernel	948	441
Threads em modo usuário	34	37

Arquitetura

- **<u>Modo Híbrido</u>: Cada *thread* em Modo Kernel pode ter várias *threads* em Modo Usuário (problemas de ambas implementações).**



Arquitetura – Modo Híbrido

- A arquitetura de *threads* em modo híbrido combina as vantagens de *threads* implementados em *modo usuário* e *threads* em *modo kernel*;
- Um processo pode ter vários TMK's e, por sua vez, um TMK pode ter vários TMU's;
- O núcleo do sistema reconhece os TMK's e pode escaloná-los individualmente;
- Um TMU pode ser executado em um TMK, em um determinado momento, e no instante seguinte pode ser executado em outro.

Arquitetura – Modo Híbrido

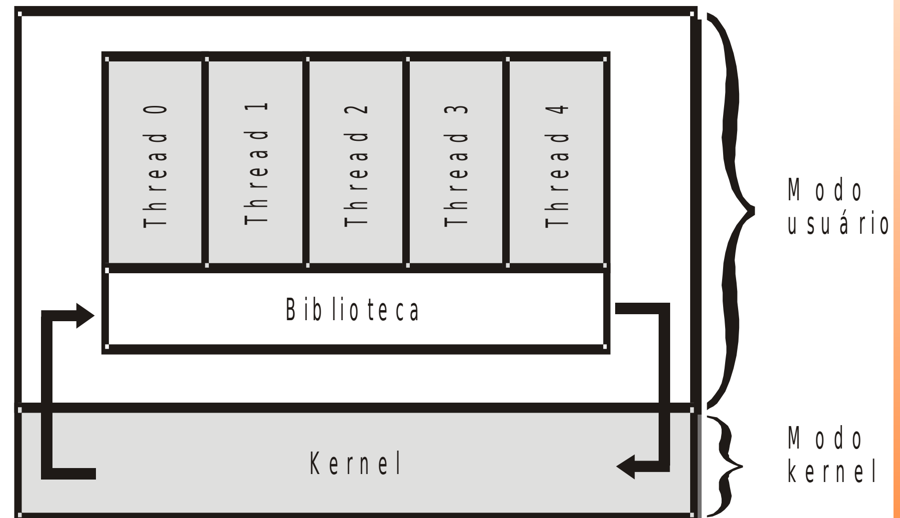
- O programador desenvolve a aplicação em termos de TMU e especifica quantos TMK's estão associados ao processo;
- Os TMU's são mapeados em TMK's enquanto o processo está sendo executado;
- O programador pode utilizar apenas TMK's, TMU's ou uma combinação de ambos.

Arquitetura – Modo Híbrido

- O modo híbrido, apesar da maior flexibilidade, apresenta problemas herdados de ambas as implementações:
  - quando uma TMK realiza uma chamada bloqueante, todos os TMU's, a ela associados, são colocados no estado de espera;
  - TMU's que desejam utilizar vários processadores devem utilizar diferentes TMK's, o que influenciará no desempenho.

Arquitetura

- ***Scheduler Activations***: **Escalonamento em modo usuário para reduzir mudanças no modo de acesso.**



Arquitetura – *Scheduler Activations*

- Os problemas apresentados no pacote de *threads* em modo híbrido existem devido à falta de comunicação entre os *threads* em *modo usuário* e em *modo kernel*. O modelo ideal deveria utilizar as facilidades do pacote em *modo kernel* com o desempenho e flexibilidade do *modo usuário*.

Arquitetura – *Scheduler Activations*

- Introduzido no início da década de 1990 na Universidade de Washington, este pacote combina o melhor das duas arquiteturas, mas em vez de dividir os *threads* em modo usuário entre os de *modo kernel*, o núcleo do sistema troca informações com a biblioteca de *threads* utilizando uma estrutura de dados chamada *scheduler activations*.

Arquitetura – *Scheduler Activations*

- A maneira de alcançar um melhor desempenho é evitar as mudanças desnecessárias de modos de acesso. Caso um *thread* utilize uma chamada ao sistema que o coloque no estado de espera, não é necessário que o kernel seja ativado, bastando que a própria biblioteca em *modo usuário* escalone outro *thread*;
- Isto é possível porque a biblioteca em *modo usuário* e o kernel se comunicam e trabalham de forma cooperativa. Cada camada implementa seu escalonamento de forma independente, porém trocando informações quando necessário.

Reflexão…

. Quais as vantagens de utilização de um ambiente *multithread*?

. Quais os benefícios do uso de *threads* em ambientes cliente/servidor?

Bibliografia

- MACHADO, F. B.; MAIA, L. P. **Arquitetura de Sistemas Operacionais,** 3ª Ed., Rio de Janeiro: LTC Editora, 2002.
- SILVA, Guilherme Baião S. *Slides* da disciplina de Sistemas Operacionais de Arquitetura Fechada. Faculdade INED, 2005.