Sistemas Operacionais

Prof. Robson de Souza

Aulas 9 e 10

Conteúdo: Processos e Threads

Relembrando…
Um **processo** é uma abstração de um programa em execução.

O **pseudoparalelismo** ou **multiprogramação** é a impressão de que os programas estão sendo executados ao mesmo tempo, o paralelismo verdadeiro só ocorre quando se tem multiprocessadores.

Modelo de processos

Nesse modelo, todo o software executado no computador (incluindo o S.O) é organizado em um número de processos sequenciais (ou apenas processos), um processo inclui os valores atuais do contador de programa, registradores e variáveis.

Conceitualmente, cada processo tem a sua própria CPU virtual. O tempo que a CPU dá para cada processo não será uniforme, por isso, os processos não devem ser programados com base em suposições de coordenação.

Em caso de processos que possuem algum tipo de requisito de tempo **real crítico** (ex: um processo que não pode ser interrompido), medidas especiais devem ser tomadas.

Um único processador pode ser compartilhado entre vários processos, com algum algoritmo de escalonamento para determinar quando parar de trabalhar em um processo e servir a outro diferente.

O nível mais baixo do S.O é o escalonador, com uma variedade de processos nele. Todo o gerenciamento de interrupções e os detalhes sobre como realmente iniciar e parar processos são ocultados do escalonador. O restante do S.O é estruturado na forma de processos.

Criação de processos

Há quatro eventos principais que fazem com que processos sejam criados:

1 – Início do sistema.
2 – Execução de uma chamada de sistema de criação de processo por um processo em execução.
3 – Uma requisição do usuário para criar um novo processo.
4 – Início de uma tarefa em lote (batch job) → Somente em sistemas em lote em computadores de grande porte.

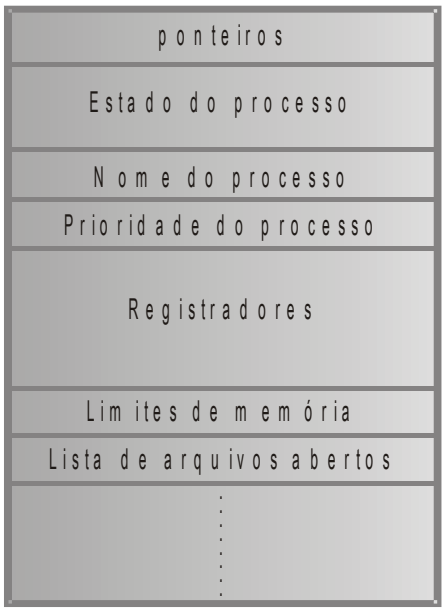
No caso 1, vários processos são criados, começando de um processo inicial que vai gerando outros e assim por diante.
No caso 2, isso é interessante quando a tarefa pode ser dividida em vários processos.
No caso 3, o usuário pode interagir com as janelas do S.O.
No caso 4, os usuários podem submeter (até remotamente) tarefas em lote para o sistema em computadores de grande porte.

Um novo processo (processo filho) é criado por um processo existente (processo pai). O processo pai executa uma chamada de sistema para criar um novo processo (filho) e assim indica qual programa executar nele.

Implementação de processos

Cada entrada na tabela de processos é chamada de **Bloco de Controle de Processos (Process Control Blocks – PCB).** Dentro de um PCB tem-se várias informações relevantes ao processo e necessárias para sua execução, como por exemplo:

- Informações associadas a cada processo
- Estado do processo
- Contador de programa
- Registradores da CPU
- Informação de escalonamento de CPU
- Informação de gerenciamento de memória
- Informação contábil
- Informação de status de E/S



Os PCBs de todos os processos ativos residem na memória principal numa área exclusiva do S.O.

Os grupos de entrada de um PCB podem ser vistos na figura abaixo:

Gerenciamento de processos	**Gerenciamento de memória**	**Gerenciamento de arquivos**
Registradores Contador de programa Palavra de estado do programa Ponteiro de pilha Estado do processo Prioridade Parâmetros de escalonamento Identificador (ID) do processo Processo pai Grupo do processo Sinais Momento em que o processo iniciou Tempo usado da CPU Tempo de CPU do filho Momento do próximo alarme	Ponteiro para o segmento de código Ponteiro para o segmento de dados Ponteiro para o segmento de pilha	Diretório-raiz Diretório de trabalho Descritores de arquivos Identificador (ID) do usuário Identificador (ID) do grupo

Threads

Em S.Os tradicionais, cada processo tem o seu próprio espaço de endereçamento com um único thread de controle, porém, pode ser desejável múltiplos threads de controle no mesmo espaço de endereçamento executando em quase paralelo como se fossem processos separados. Um processo dentro de outro são como "miniprocessos", chamados **threads**.

A principal razão para os threads é que em alguma aplicação podem ocorrer múltiplas atividades ao mesmo tempo. Algumas delas podem ser bloqueadas de tempos em tempos.
São como se fossem processos paralelos, porém os threads compartilham o mesmo espaço de memória, diferentemente dos processos.

Um segundo argumento para o uso de threads é que elas são mais fáceis (mais rápido) de criar e destruir que

os processos, pois não tem quaisquer recursos associados a eles. Pode ser até 100 vezes mais rápido do que criar um processo.

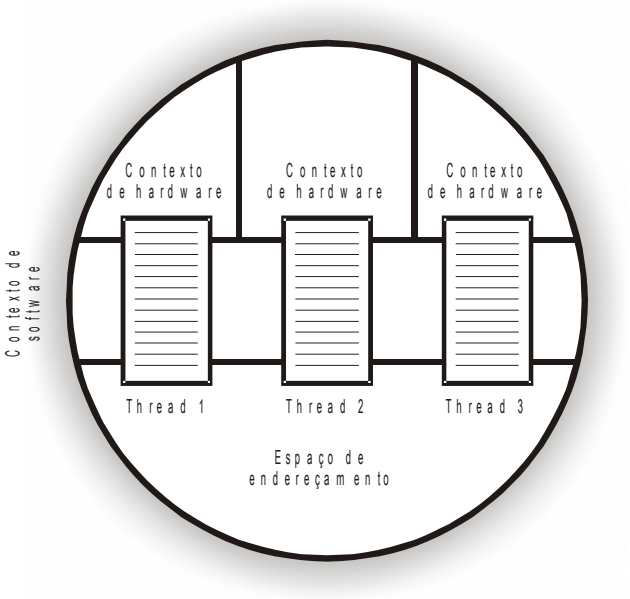
Uma terceira razão, é que embora o uso de threads não resulte em ganho de desempenho quando todos são CPU-bound, no entanto, quando há grande quantidade de computação e de E/S, os threads permitem que essas atividades se sobreponham e, desse modo, aceleram a aplicação.

Finalmente, os threads são úteis em sistemas com múltiplas CPUs, para os quais o paralelismo real é possível.

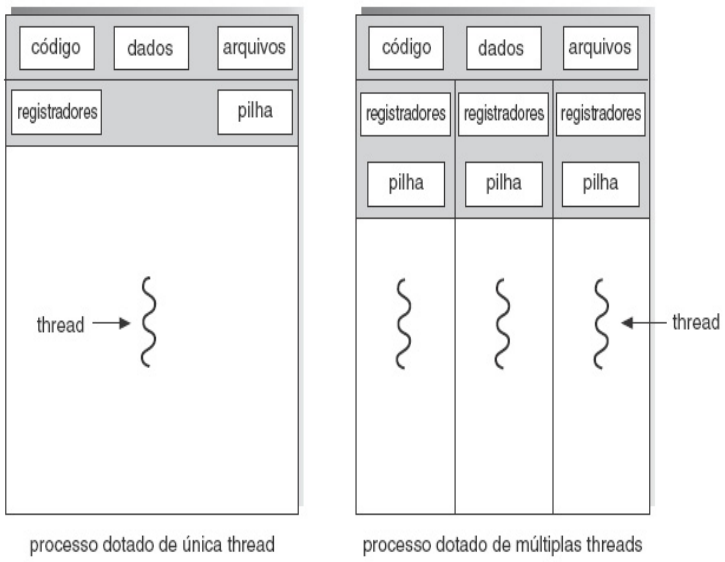
Dentro de um mesmo processo, threads compartilham o mesmo **contexto de software** e **espaço de endereçamento** com os demais threads. Threads são implementados através de uma estrutura denominada **Bloco de controle de threads (TCB).**

Ambiente monothread

Ambiente multithread

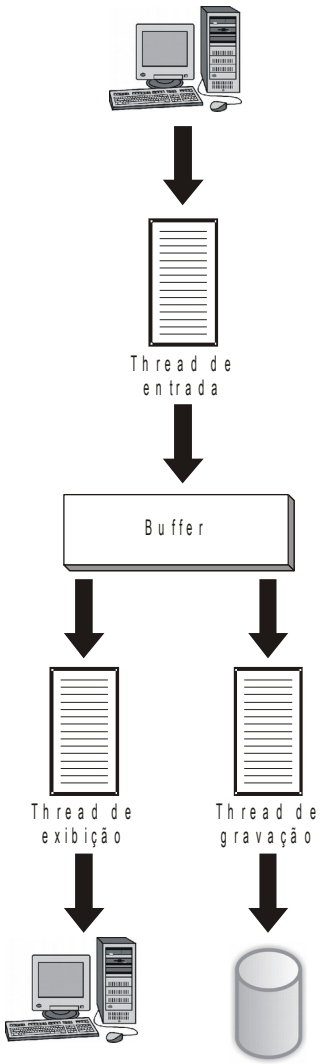


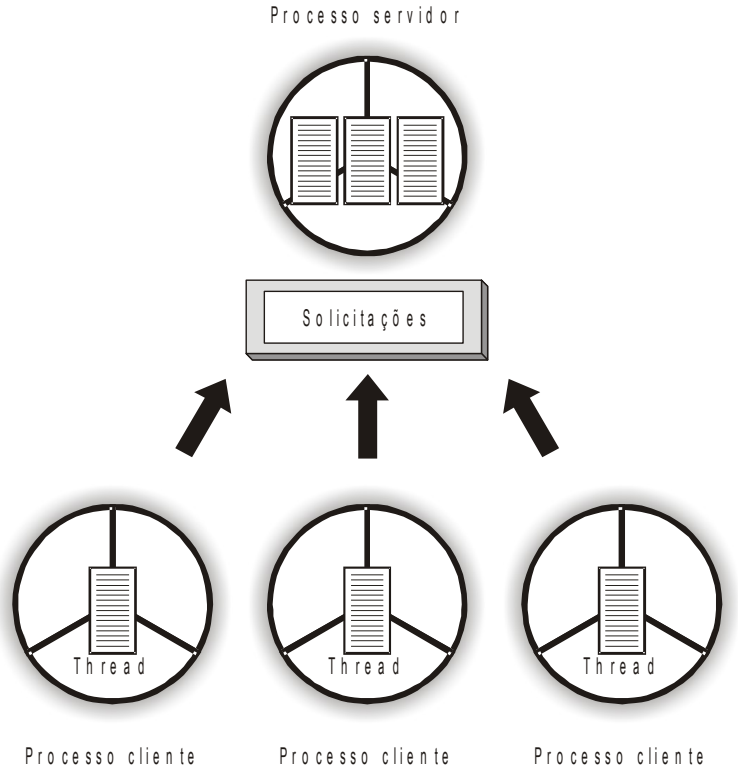
Ambiente mono e multithread



Threads tornam possível manter a ideia de processos sequenciais que fazem chamadas de sistema bloqueante (por exemplo, E/S de disco) e mesmo assim conseguem obter paralelismo. As chamadas de sistema bloqueante tornam a programação mais fácil, enquanto o paralelismo melhora o desempenho.

<u>Aplicações multithread</u>





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.