**Threads**

**Felipe Castro C. do Sacramento¹,Otávio A. A. Silva¹**

**¹ Faculdade de Engenharia da Computação e Telecomunicações - Instituto de Tecnologia - Universidade Federal do Pará**

**Cidade Universitária “Prof. José da Silveira Netto” – Campus II (Profissional) Belém/PA, Brasil, 66075-110**

{felipeccastro94,tavioalves}@gmail.com

***Abstract.*** *This brief article will describe meta-features using threads in computer systems. Showing that the behavior of such environments and multithreaded monothread, their architecture and how its implementation is done by the operating system.*

***Resumo.*** *Este meta-artigo descreverá breves características do uso de threads em sistemas computacionais. Mostrando qual o comportamento de tal em ambientes monothread e multithread, suas arquiteturas e de que forma é feita a sua implementação pelo sistema operacional.*

**1. Introdução**

Com o passar do tempo o homem percebeu que se trabalhar linearmente, ou seja, um passo por vez em linha de produção para criação de um produto, demanda grande quantidade de tempo e desperdiça recursos, pelo tempo em ocioso de outras linhas de trabalho. Uma das maneiras para se otimizar essa forma de trabalhar foi a geração da divisão de uma ou mais linhas de trabalho, para que enquanto uma estiver trabalhando a outra esteja também, ganhando tempo e deixando de se desperdiçar recursos.

Em um computador, fazendo-se um paralelo, a linha de trabalho seria um processo, e forma de se trabalhar simultaneamente com em um processo, seria possível através da utilização de threads.Ou seja, o thread nada mais é que uma maneira utilizada por um processo para se subdividir e assim poder ser executado mais de um trecho do código do processo.

**2. Processos**

Antes de começarmos a explicar o que vem a ser thread, necessitamos da uma breve atenção ao conceito de processos.

Um processo pode ser definido como um ambiente onde um programa é executado, este ambiente possui informações da execução, recursos do sistema que o programa pode utilizar, espaço de endereçamento, tempo de processador e área em disco. Um processo se subdivide em contexto de software, contexto de hardware e espaço para endereçamento.

O contexto de hardware armazena o conteúdo dos registradores gerais do UCP, além dos registradores de uso específico, como *program counter* (PC), *stack pointer* (SP) e registradores de status.

O Contexto de Software é onde são especificados características e limites dos recursos que podem ser alocados pelo processo como o numero máximo de arquivos que podem ser abertos simultaneamente, prioridade de execução e tamanho de buffer para operação de E/S

O espaço de endereçamento é a área da memória pertencente ao processo onde as instruções e os dados do programa são armazenados para execução.

**3. Thread**

É uma das maneiras utilizadas por um processo para dividir a si mesmo em duas ou mais tarefas que podem ser executadas simultaneamente, em geral, em arquiteturas multiprocessadas.O suporte à thread é fornecido pelo próprio sistema operacional (SO), no caso da Kernel-Level Thread (KLT), ou implementada através de uma biblioteca de uma determinada linguagem, no caso de uma User-Level Thread (ULT).

Se caracterizam por ser processos leves. Justamente dado pelo menor tempo gasto em atividades de criação e escalonamento de threads, se comparadas aos processos. O compartilhamento de memória entre threads maximiza o uso dos espaços de endereçamento e torna mais eficiente o uso destes dispositivos. Pelo fato de possuírem o mesmo espaço de endereçamento podem compartilhar das mesmas variáveis globais, acessar qualquer posição de memória dentro do espaço de endereçamento do processo e fazer modificações, fazendo com que a velocidade de troca de informações sejam bem maiores do que processos, já que não existe um mecanismo de comunicação. Não existe uma forma de proteção do espaço de endereçamento utilizados nos threads, portanto isso fica por conta do programador.

Ou seja, cada threads possui o mesmo contexto de software e compartilha o mesmo espaço de memória do processo pai, porém o contexto de hardware e o fluxo é diferente para cada uma. Por consequência a perda do tempo com o escalonamento das threads é muito menor do que comparada ao escalonamento por processos.

Threads possuem um ciclo de estados, assim como em processos, que são:

1. O endereço da pilha;
2. O contador de programa;
3. Registrador de instruções
4. Registradores de dados, endereços, flags;
5. Endereços das threads filhas;
6. Estado de execução.

**3.1 Categorias**

Threads são basicamente divididas em duas categorias *User Level Thread* e *Kernel Level Thread.*

*User-Level Thread* (ULT): são suportadas pela aplicação, sem conhecimento do núcleo e geralmente são implementadas por pacotes de rotinas (códigos para criar, terminar, escalonamento e armazenar contexto) fornecidas por uma determinada biblioteca de uma linguagem. Possuem como vantagens a possibilidade de implementação em sistemas operativos que não suportam nativamente este recurso, sendo geralmente mais rápidas e eficientes pois dispensam o acesso ao núcleo. Evita assim mudança no modo de acesso, e a estrutura de dados fica no espaço do utilizador, levando a uma significativa queda de *overhead*, além de poder escolher entre as diversas formas de escalonamento em que melhor se adéqua.

*Kernel-Level Thread* (KLT): não é realizada através do código do próprio programa; todo o processo é subsidiado pelo SO. Esse modelo tem a vantagem de permitir o suporte a multiprocessamento e o fato do bloqueio de uma linha de execução não acarretar bloqueio de todo processo, não obstante, temos a desvantagem de ter que mudar o tipo de acesso sempre que o escalonamento for necessário, aumentando assim o tão temido *overhead*.

**3.2 Benefícios do seu uso**

Uma das vantagens da utilização de threads é a subdivisão de uma linha de execução. Quando uma thread está esperado um determinado recurso do sistema, o processo não é totalmente parado, pois quando uma thread entra em um estado de ócio uma outra thread do mesmo processo está aguardando na fila de prontos para dar continuidade a execução do programa.

Podem comunicar-se através das variáveis globais do processo que as criou. A utilização destas variáveis pode ser controlada através de primitivas de sincronização (monitores, semáforos, ou construções similares). Primitivas existem para bloqueio do processo que tenta obter acesso a uma área da memória que está correntemente sendo utilizada por outro processo. Primitivas de sinalização de fim de utilização de recurso compartilhado também existem. Estas primitivas podem “acordar” um ou mais processos que estavam bloqueados.

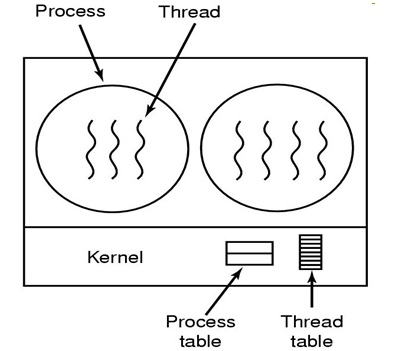
Compartilhamento de recursos é facilitado com a utilização de threads, já que os recursos alocados pelo processo, são compartilhados pelas threads, ou seja, é mais econômica a forma de compartilhamento de recursos.

Pode se utilizar arquiteturas multiprocessadas, podendo-se executar cada uma das threads criadas para um mesmo processo em paralelo. Além do ganho de desempenho, pelo fato de se poder sobrepor atividades.

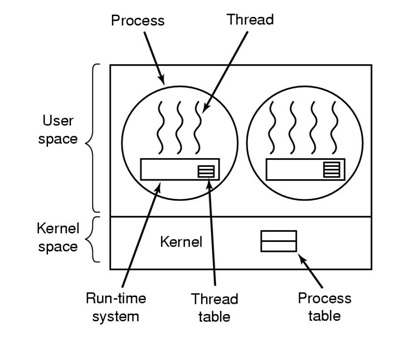
**4. Arquitetura e Implementação de Threads**

Threads podem ser implementadas, como foi dito anteriormente, em modo kernel, suportadas pelo sistema operacional, ou modo usuário, utilizando bibliotecas específicas de cada linguagem de programação.

Em modo kernel, o gerenciamento dos threads é feito por controle do SO, que sabe da existência dos mesmos e controla as suas execuções, podendo escaloná-los individualmente. Ao mesmo tempo que isso representa uma vantagem e uma desvantagem, já que o controle não precisa ser feito pelo programador, mas pela utilização de “system calls” são frequentes as mudanças de modo de acesso.



Em modo usuário, os threads são implementados pela aplicação, devendo haver uma biblioteca na linguagem que os suportem (java.lang.Thread em Java ou thread.h em C), de modo à criar e eliminar threads, troca de mensagens, escalonamento, etc. O fato do SO não ter conhecimento dos threads também (à exemplo do modo kernel) representa uma vantagem e uma desvantagem. Apesar de ser mais rápido e eficiente, o SO pode colocar o processo em “estado de espera”, logo os threads também, ignorando um possível “estado de pronto” dos threads em execução.



**5. Conclusão**

Threads são indispensáveis em sistemas e aplicação atuais. Em tempos em que se busca cada vez mais otimização e eficiência na execução de tarefas (não somente computacionais), threads trazem um conceito de divisão de processo fundamental, por exemplo, para aplicações complexas. As implementações têm suas vantagens e desvantagens, cabendo ao desenvolvedor optar pela qual atende melhor as suas necessidades.

**6. Referências**

http://www.facol.br/sophia/2741/APOSTILA08\_THREADS.PDF

http://pt.wikipedia.org/wiki/Thread

http://www.inf.ufsc.br/~bosco/ensino/ine5645/Onde%20voce%20programa.pptx

http://regulus.pcs.usp.br/~jean/so/AULA%2006%20-%20Threads.pdf