Threads Posix

- *INTRODUÇÃO*

Sistemas operacionais multitarefa tem a característica de poder rodar múltiplos processos ou programas em paralelo, sendo de um usuário ou vários, e sua origem remonta aos anos 70. Ao final dos anos 80, este conceito foi estendido para dentro do processo, permitindo a existência de múltiplos caminhos de execução dentro de um mesmo programa. Somente a partir de 1996 os principais S.O. comerciais passaram a suportar múltiplos threads dentro de um processo.

- *THREADS*

Um thread representa o fluxo de execução de um processo. Cada processo pode ter vários ou pelo menos um thread.
Qual é a vantagem de criar threads no lugar de processos?:

1 - A criação e sincronização dos threads é mais rápida.
2 - A comunicação entre threads é mais eficiente por estarem compartilhando e espaço de endereçamento.
3 – Maior interatividade
4 – Menor tempo de resposta
5 – Melhor eficiência em arquiteturas multiprocessador

Um thread pode ser implementado no nível de usuário ou no nível do kernel. A vantagem de ter os threads no espaço do usuário é que eles podem se implementados sem a necessidade de alterar o núcleo. E a desvantagem é que quando um thread é bloqueado todos os outros threads do processo também são bloqueados, pois o núcleo pensa que existe somente um thread e o processo não é escalonado até que aquele thread seja desbloqueado.
Já implementando os threads no espaço do núcleo, o que é um pouco mais complicado e caro, no exemplo anterior, o núcleo, tendo conhecimento dos threads poderia passar o foco da execução para outro thread do processo.

Thread "usuário"	**Thread "núcleo"**
-Uma biblioteca dá suporte a criação, escalonamento…,que executa em modo usuário. -O núcleo não interfere nos threads. -Vantagens: muito rápidas de gerenciar pois não necessitam do núcleo (chamada de sistema…) -Exemplo: bibliotecas Pthreads (POSIX), threads deSOLARIS.	-O núcleo oferece suporte a threads. -Mais lento…O Sis. Op. pode escalonar mais eficientemente os threads, inclusive em máquinas multi-processadas. -Exemplo: Windows, Solaris, Linux.

Tabela 1.0

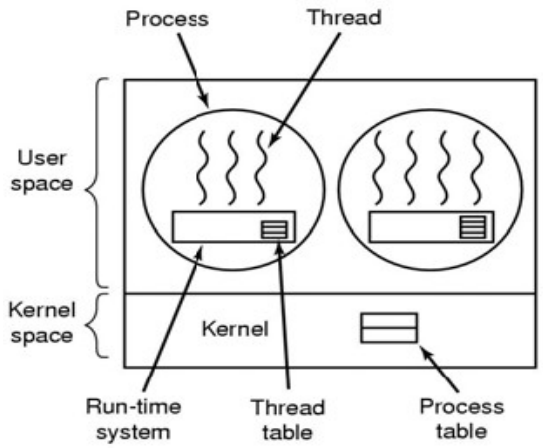

Figura 1.0

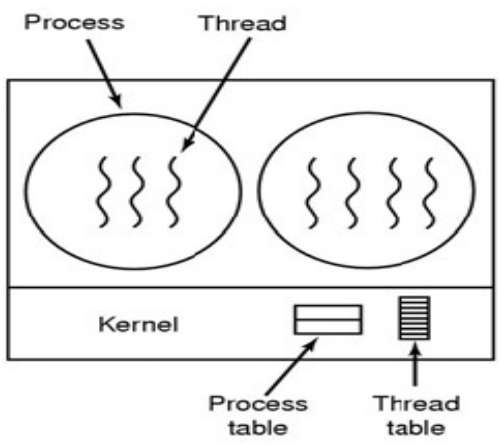

Figura 1.1

Existem 3 formas de conciliar os dois níveis de threads:

1 – (n : 1) N threads de usuário por thread de núcleo.
2 – (1 : 1) 1 threads de usuário por threads de núcleo.
3 - Meio termo entre os dois.

Cada thread compartilha com os outros threads, os recursos do processo ao qual pertence e morre quando o processo a que pertence morre.
Num sistema multithreads, os threads compartilham o espaço de endereçamento mas têm contextos de execução diferentes.

Itens compartilhados por todas os threads num processo	**Itens privativos de cada thread**
Espaço de endereçamento	Contador de programa
Variáveis globais	Registradores
Arquivos abertos	Pilha
Processos filhos	Estado
Alarmes pendentes	
Sinais e tratadores de sinais	
Informação de contabilidade	

Tabela 1.1

- *PTHREADS (Threads POSIX)*

"Antigamente" cada sistema operacional implementava sua própria versão de threads, fazendo com que fosse quase impossível para os programadores desenvolverem aplicações portáveis que utilizassem threads.
Para superar esse problema foi desenvolvido o padrão POSIX 1003.1.c (Portabel Operating Sysrtem Interface) pelo IEEE (Institute of Electrical and Electronics Engineers), sendo sua última versão de 2001.
A especificação POSIX não estabelece se os threads devem ser implementados no núcleo ou no espaço do usuário. Ela define uma API (aplication programming interface) com as funções básicas de criação e controle de threads. É conhecida como Posix Threads ou Pthreads e se trata de uma biblioteca que deve ser chamada nos programas que desejem utilizar estas funcionalidades.
Existem várias plataformas que suportam o padrão Posix:

-Sistemas embarcados (VxWorks, Nucleus);
-Windows NT e derivados (embora o nível de suporte seja discutível);
-Unix (Linux/BSD/Solaris/MacOSX/AIX);
-Mainframes (z/OS, OpenVMS)

Esta API apresenta mais de 60 funções em C. Aqui são apresentadas somente as principais chamadas:

Chamadas a thread	Descrição
pthread_create	Cria um novo thread no espaço de endereço do chamador
pthread_exit	Termina o thread chamador
pthread_join	Espera pelo término de um thread
pthread_mutex_init	Cria um novo mutex
pthread_mutex-destroy	Destrói um mutex
pthread_mutex_lock	Impede um mutex
pthread_mutex_unlock	Libera um mutex
pthread_cond-init	Cria uma variável de condição
pthread_cond_destroy	Destrói uma variável de condição
pthread_cond_wait	Espera por uma variável de condição
pthread_cond_signal	Libera um thread que está à espera sobre uma variável de condição.

Tabela 1.2

Podemos fazer a seguinte classificação:

– Para criar e esperar.
  pthread_create
  pthread_join
– Para determinar o thread ID
  pthread_self
– Para terminar threads
  pthread_cancel
  pthread_exit
  exit [termina todos threads] , ret [termina o thread corrente]
– Para sincronização
  Mutexes
  Condições
  Semáforos

As chamadas:

-Criação de threads:

int pthread_create (pthread_t * thread, pthread_attr_t * attr, void * (*start_routine)(void *), void *arg);

Argumentos:

thread – guarda o thread ID.
attr – é um parâmetro opcional, geralmente se passa NULL
start_routine: é uma função que será executada pela thread. Pega em argumento um "void*" e retorna um void*. O protótipo da função deve ser semelhante a: ***void *a_name(void* parameter);*** Pelo fato do tipo de retorno da função ser um ponteiro para void, pode-se retornar qualquer tipo de dado para o thread principal da aplicação quando o(s) thread(s) extras forem "coletados";

arg – ponteiro sobre o argumento da função. Para passar mais de um argumento, usar um vetor ou uma struct.

A struct '**pthread_attr_t**' inclui:

.A informação se a thread é "detachable" ou não.
.O algoritmo de escalonamento
real-time?
SCHED_FIFO
SCHED_RR
SCHED_OTHER
.O tipo de thread
Threads de núcleo: PTHREAD_SCOPE_SYSTEM
Threads de usuário: PTHREAD_SCOPE_PROCESS
.Outros:
O endereço da pilha
O tamanho da pilha

Em caso de sucesso a função retorna "0" (zero), mas em caso de erro não retornará necessariamente "-1"

-Sincronização de threads:

"Joining" é uma maneira de se obter sincronização entre threads. Para que uma thread fique bloqueada até que uma outra termine:

int pthread_join(pthread_t th, void **thread_return);

th - Thread ID, valor recebido quando o thread foi criado;
thread_return - Ponteiro para ponteiro do tipo void (pode ser utilizado para receber qualquer tipo de dado retornado pela função que rodou no thread extra);

-Finalização threads:

Uma thread termina quando:

-A thread retorna da função que a originou (start_routine)
-A thread chama **pthread_exit**
-A thread é cancelada por outra thread através da função **pthread_cancel**
-O processo inteiro termina

void pthread_exit(void *retval);

retval - é o valor de retorno.

A chamada à função **pthread_exit()** provoca a terminação do thread e a liberação dos recursos que está consumindo. Tendo em conta que ao terminar a função do thread, este será destruído, não haveria necessidade de chamar esta função, somente se for necessário destruir o thread no meio da sua execução.
Se o thread pai termina retornando da sua função principal, seus filhos morrem Se o thread pai termina com pthread_exit, seus filhos NÃO morrem.

-Identificação de threads:

Para saber o thread ID do thread corrente:
pthread_t pthread_self (void);

-Detached threads:

Um thread detached é um thread que não pode ser sincronizado com **pthread_join.** "Ser detached" é um atributo do thread, que pode ser utilizado no momento da criação deste.

-Mutex:

Mutex é uma abreviação de "mutual exclusion" (exclusão mútua).
Variáveis do tipo Mutex são a principal forma que o Pthreads apresenta para a proteção de regiões críticas. Em geral um mutex protege algum recurso, como um buffer compartilhado por dois threads. Para garantir que somente um thread por vez acesse o recurso compartilhado, os threads devem obter o mutex antes de tocar no recurso e liberá-lo assim que terminarem. Os mutex como os semáforos só podem conter valores 0 e 1.

A seqüência de uso de uma variável do tipo mutex normalmente é:

Criar e inicializar uma variável do tipo mutex
Diversas threads tentam efetuar o lock no mutex
Apenas uma thread consegue
Essa thread realiza alguma computação
E depois libera o mutex
Outra thread efetua e o lock e repete o processo
No final, o mutex é destruído

Quando diversas threads competem por um mutex, aquelas que não conseguiram efetuar o lock ficam bloqueadas.
Para criar, destruir, bloquear e desbloquear mutexes, veja as chamadas na tabela 1.2.

-Variáveis Condicionais:

Mutexes são destinados ao uso de curta duração. Eles não são utilizados para sincronização de longa duração, como a espera de um dispositivo de fita. Para esse tipo de sincronização são utilizadas as "variáveis de condição".
Elas podem estar associadas a mutexes.
Para criar, destruir, ficar em modo de espera e liberar variáveis de condição, veja as chamadas na tabela 1.2.

-Observações pessoais (depois de alguns testes...):

1- O Windows não suporta nativamente o modelo PThreads, mas há implementações disponíveis que resolvem esse problema.

2 – Nos sistemas Unix, ao criar várias threads, elas pertencem ao mesmo processo (é só verificar o PID, que vai ser sempre o mesmo).
Já no sistema Linux, se verificar o PID, podemos comprovar que cada thread pertence a um processo diferente (é uma forma particular do Linux de tratar threads).

3 – No Linux, quando compilar o programa, usando GCC, deve também ser compilada a biblioteca Pthread (gcc -lpthread nomePrograma.c -o nomeExec).

4 – Executando no CYGWIN (aplicativo que simula Linux [ou Unix...]no Windows) funciona como nos sistemas Unix: todas as threads pertencem a um mesmo processo. Isto também foi testado no Solaris.

-Bibliografia:

Biblioteca:
->Sistemas operacionais modernos - Autor: A.S.Tanenbaum

Intenet:
->Programação multithread em ambiente Unix: como criar e sincronizar 2 tarefas paralelas - http://linhadecodigo.com.br/artigos.asp?id_ac=1168

->Processos leves: threads - http://asc.di.fct.unl.pt/~pm/SO-04-05/teoricas/08-1503-GP03.pdf

->Pthreads – Mário João Junior - http://equipe.nce.ufrj.br/gabriel/progpar/Pthreads.pdf

->Threads - Paul Crocker - http://www.di.ubi.pt/~operativos/praticos/6-b-threads.pdf

->Pthreads - http://www.ic.unicamp.br/~rodolfo/mo801/aulas/PThreads.html

->A Pthreads Tutorial, written by Andrae Muys - http://www.cs.nmsu.edu/~jcook/Tools/pthreads/pthreads.html

->Multi-Threaded Programming With POSIX Threads - http://users.actcom.co.il/~choo/lupg/tutorials/multi-thread/multi-thread.html

->Programação parlela - http://www.inf.ufsc.br/~frank/INE5645/2.%20Programacao%20Paralela.pdf

->Programação concorrente com Threads POSIX -- http://gsd.di.uminho.pt/teaching/5306O2/2006/guioes/guiao-04