Sistemas Operacionais

Threads

Prof. José Roberto Bezerra

Agenda

- O que são *threads?*
- Por que utilizar *threads?*
- Exemplos de uso
- Aplicação online
- Pthreads

Threads

O que são *threads*?

- Programas “normais” (*single thread*), tem um único fluxo de execução
- Um *Thread* executa dentro de um processo
- Também chamados *lightweight* process (processos leves)
  - São mais fáceis de criar e destruir, pois praticamente não possuem recursos associados
  - No SO Solaris, estima-se que a criação de um processo é 30 vezes mais lenta do que de uma *thread*

O que são *threads*?

- São úteis em sistemas com múltiplas CPUs, nas quais o paralelismo real é possível
  - A medida que é possível dividir uma tarefa em subtarefas, o uso de *threads* se torna mais interessante
- É um fluxo único de controle que compartilha o contexto com um processo

Threads não são processos

- Processos são usados para agrupar recursos
- *Threads* são entidades escalonadas para execução sobre a CPU
- Múltiplos *threads* executando em paralelo em um processo é análogo a múltiplos processos executando em paralelo em um computador

Threads não são processos

- Um *thread* é semelhante a um processo
  - Compartilha o mesmo "espaço de endereçamento"
  - Arquivos abertos
  - Processos filhos
  - Sinais e alarmes
- O chaveamento entre *threads* é similar ao de processos
  - Porém mais rápido e menos custoso para CPU que o chaveamento entre processos
- Cada *thread* possui um contador de programa e uma pilha próprios

Por que *threads*?

- São fáceis de criar e destruir
  - Praticamente não possuem recursos associados
  - Criar um *thread* pode ser até cem vezes mais rápido que criar um novo processo
- Permite que múltiplas execuções de código ocorram no mesmo ambiente do processo
- Quando existe uma mistura entre *threads* orientados a CPU e E/S há ganho de desempenho

Multithreading

- Suporte a nível de usuário e *kernel*
  - Threads de usuário são permitidas acima do kernel e não possuem gerenciamento do kernel
  - Já as threads de kernel são gerenciadas pelo próprio SO
- Presente nos SO atuais
  - Linux
  - Windows XP, 7
  - Mac OS
  - Solaris
  - Tru64

Exemplos de uso

- Editor de textos
- Planilha eletrônica
- Servidor web

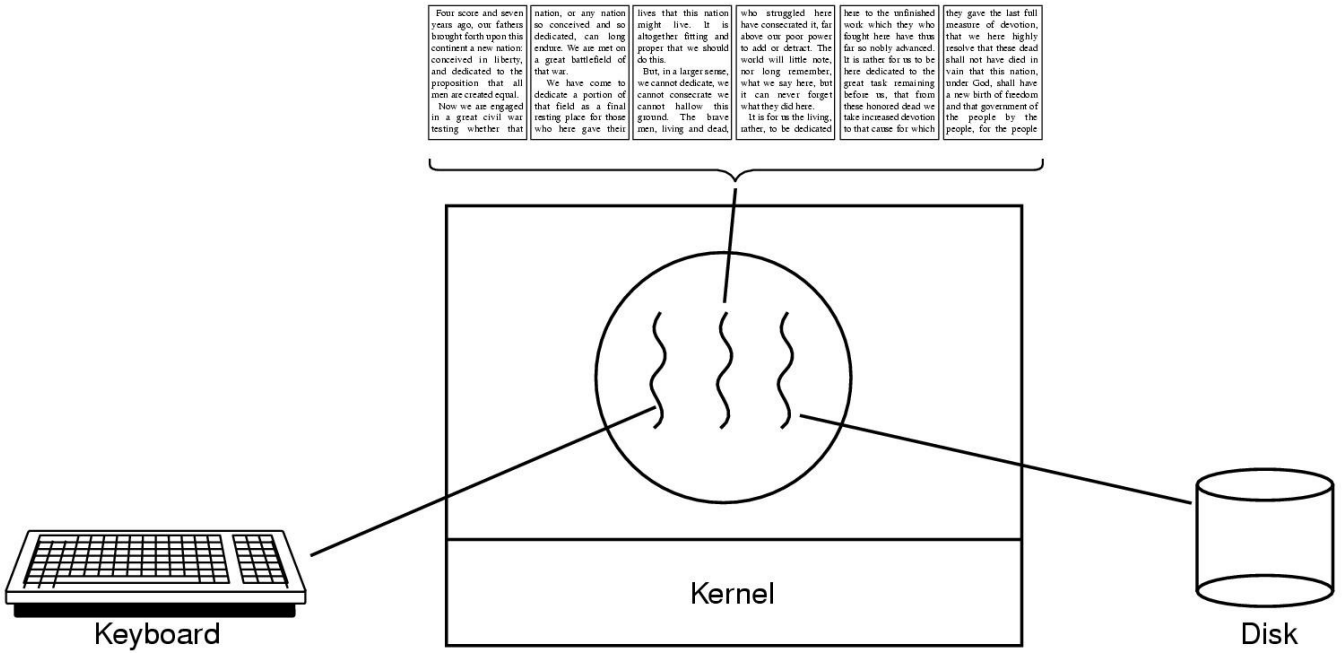
Editor de textos convencional

- Thread única
  - Interação com usuário (mouse/teclado) e Formatação
  - A remoção de um parágrafo na página 1 terá impacto na formatação de todas as 800 páginas
  - Após o comando a interação com o documento pode tornar-se lenta, pois o programa passará a executar a formatação “deixando” de atender requisições dos dispositivos e a visualização do documento

Editor de textos multithreaded

- Multithread
  - Thread para interação com o usuário
  - Thread para formatação do texto
- Exemplo anterior
  - Ao remover texto da página 1, o thread interativo solicita ao thread de formatação para reformatar o arquivo
  - Assim, o thread interativo continua a interação normal com o usuário (mouse/teclado/tela)
  - Segundo thread continua processamento nos momentos de inatividade do usuário
  - Acrescentando um thread para salvamento automático teríamos um esquema mais completo

Editor de textos multithreaded



Um esquema de processador de textos com três processos também funcionaria adequadamente?

Não. Pois, os três *threads* devem precisam operar sobre os mesmos dados (documento) que devem estar compartilhados. Com três processos teríamos três espaços de endereçamento distintos, dificultando operações com o mesmo conjunto de dados.

Planilha eletrônica

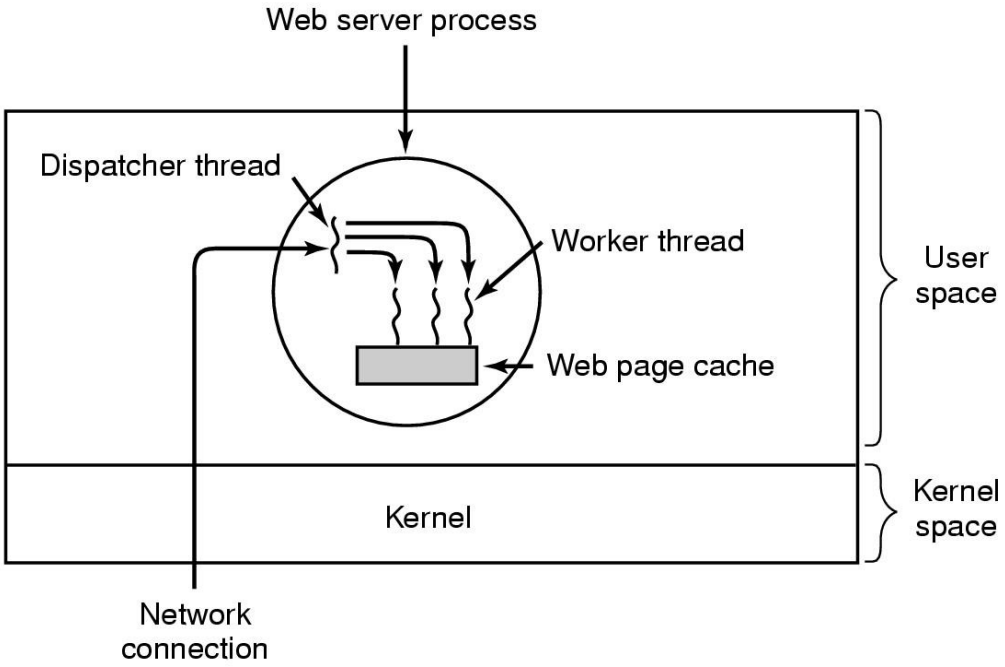
- Permite manter uma matriz com dados passados pelo usuário e aplicar fórmulas complexas obtendo resultados
- A alteração de um dado em uma planilha traz impacto em diversas planilhas, necessitando recálculo
- *Multithread*
  - Interação
  - Cálculos
  - *Backups* automáticos

Servidor *Web*

- Requisições de páginas são feitas a um servidor web que deve enviá-las de volta aos clientes
- Algumas páginas são mais demandadas
  - Tais páginas são armazenadas na memória reduzindo o uso do disco (cache)

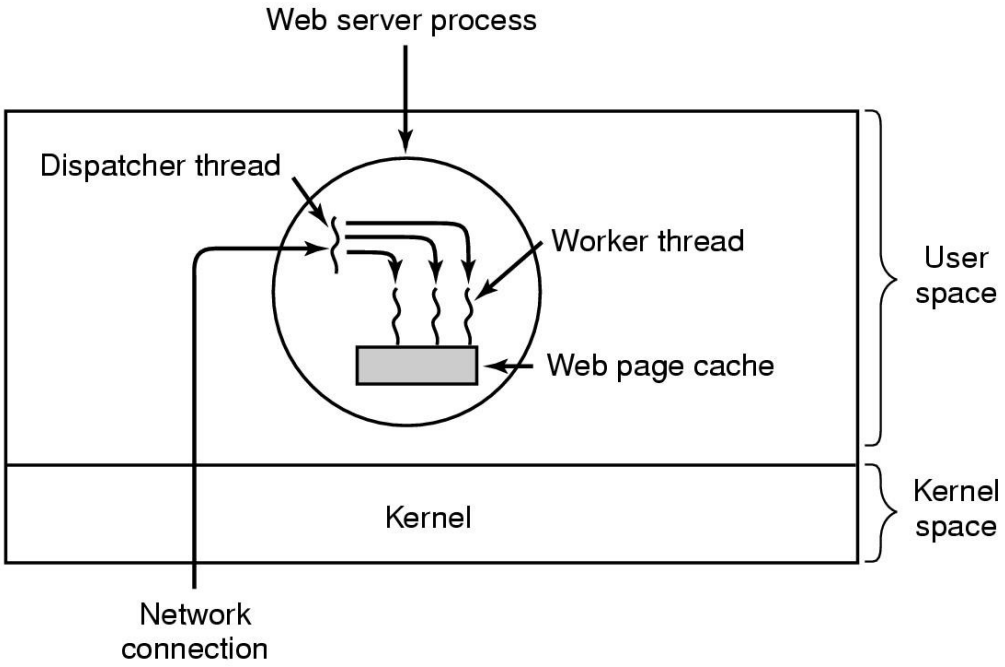
Arquitetura

- Despachante
  - lê as requisições que chegam pela rede
  - Examina a requisição e escolhe um operário para responder
- Operário
  - É desbloqueado para atender a requisição
  - Verifica se a página requisitada está em cache
  - Caso contrário, executa read para ler



Arquitetura

- Permite que o servidor web seja escrito como uma coleção de threads sequenciais
-



Compartilhamento entre *threads*

- Todas as *threads* de um processo compartilham entre si:
  - Espaço de endereçamento
  - Variáveis Globais
  - Processos Filhos
  - Alarmes pendentes
  - Sinais
  - Arquivos abertos
- Cada *thread* possui seu próprio
  - Contador de Programa
  - Registradores
  - Pilha
  - Estado

Compartilhamento entre *Threads*

Proteção

- Todos os threads têm exatamente o mesmo espaço de endereçamento
  - Cada thread pode ter acesso a qualquer endereço de memória do processo
  - Não há proteção de memória entre threads
  - Um thread pode ler, alterar e até apagar a pilha de outro *thread*

Aplicação *online*

- Exemplo de aplicação
  - www.dsc.ufcg.edu.br/~jacques/cursos/map/html/threads/threads1.html

POSIX Threads

Threads POSIX

- IEEE 1003.1c
  - Possibilita a portabilidade em threads
  - Define o pacote chamado Pthreads
  - Suportado pelos sistemas UNIX
  - Possuem propriedades básicas:
    - Identificador
    - Registros
    - Contador de programas
  - Utiliza chamadas de sistema

Chamadas *Pthreads*

Chamada	Descrição
`Phtread_create`	Cria nova *thread*
`Phtread_exit`	Conclui a chamada de *thread*
`Phtread_join`	Espera que um thread específico seja abandonado
`Phtread_yield`	Libera a CPU para que outra thread seja executado
`Phtread_attr_init`	Cria e inicializa uma estrutura de atributos da thread
`Phtread_attr_destroy`	Remove uma estrutura de atributos da thread

Exemplos

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *print_message_function(void *ptr);

void *print_message_function(void *ptr)
{
     char *message;
    message = (char *) ptr;
    printf("%s \n", message);
}
```

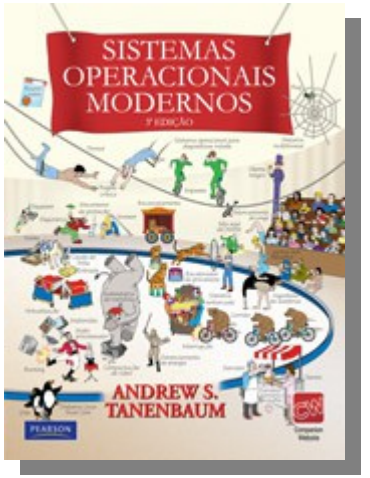
```
main()
{
     pthread_t thread1, thread2;
     char *message1 = "T1";
     char *message2 = "T2";
     int  t1, t2;

    /* Cria threads e aponta para suas respectivas funcoes

     t1 = pthread_create( &thread1, NULL, print_message_fu
(void*) message1);
     t2 = pthread_create( &thread2, NULL, print_message_fu
(void*) message2);

     printf("Thread 1 returns: %d\n",t1);
     printf("Thread 2 returns: %d\n",t2);
     exit(0);
}
```

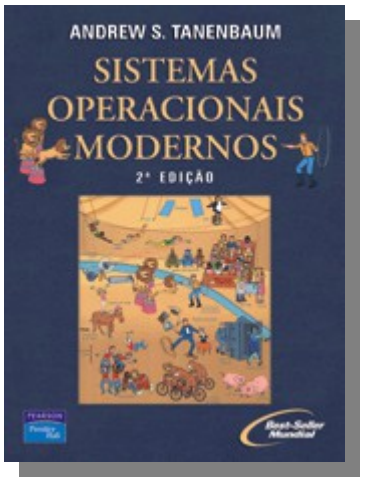
```
main()
{
     pthread_t thread1, thread2;
     char *message1 = "T1";
     char *message2 = "T2";
     int  t1, t2;

    /* Cria threads e aponta para suas respectivas funcoes

     t1 = pthread_create( &thread1, NULL, print_message_fu
(void*) message1);
     t2 = pthread_create( &thread2, NULL, print_message_fu
(void*) message2);

     /* Espera a conclusão das threads para continuar main

     pthread_join( thread1, NULL);
     pthread_join( thread2, NULL);

     printf("Thread 1 returns: %d\n",t1);
     printf("Thread 2 returns: %d\n",t2);
     exit(0);
}
```

```
#define NUM_THREADS     5
void *PrintHello(void *threadid)
{
   long tid;
   tid = (long)threadid;
   printf("Hello World! Thread #%ld!\n", tid);
   pthread_exit(NULL);
}
int main (int argc, char *argv[])
{
   pthread_t threads[NUM_THREADS];
   int rc;
   long t;
   for(t=0; t<NUM_THREADS; t++){
      printf("Function main: criando thread %ld\n", t);
      rc = pthread_create(&threads[t], NULL, PrintHello, (
      if (rc){
         printf("ERROR; codigo de erro de pthread_create()
rc);
         exit(-1);
      }
   }
   pthread_exit(NULL);
}
```

Bibliografia

- Tanenbaum, Andrew S. Sistemas Operacionais Modernos. 3a. Ed. Pearson, 2010
- Tanenbaum, Andrew S. Sistemas Operacionais Modernos. 2a. Ed. Pearson, 2003
- Silbershatz. Sistemas Operacionais com java. 7a. Ed, 2008





Dúvidas e Perguntas

FIM