CURSO: Tecnologo em Sistemas para Internet - 4º período – Noturno

UNIDADE CURRICULAR: Sistemas Operacionais I - TURMA:

PROFESSOR: Genair C. Viana

Aulas:

- Conceitos de processos e threads;
- Virtual Box;
- Trabalho 1.

28/02/2023

Processos e *threads*

- Vimos o conceito de processo englobando duas características básicas:
  - propriedade de recursos
    - a um processo é alocado um espaço de endereçamento virtual para manter a sua imagem
    - de tempos em tempos o processo pode ter mais memória, além do controle de arquivos, dispositivos de E/S, ...

Processos e *threads*

- unidade de despacho:
  - um processo é uma linha de execução
  - esta linha de execução é intercalada com outras linhas de outros processos
  - cada uma delas tem um estado de execução e uma prioridade
  - é a entidade que é escalonada e despachada pelo SO

Processos e *threads*

- Estas duas características podem ser tratadas de forma independente pelo SO:
  - *thread* ou processo peso leve (*lightweight process*): é a unidade de despacho
  - processo ou tarefa: é a unidade de alocação de recursos

Processos e *threads*

- Em um ambiente *multithreaded*, um processo:
  - é a unidade de alocação e proteção de recursos
  - tem um espaço de endereçamento virtual que mantém a imagem do processo
  - tem acesso controlado
    - a outros processos,
    - a outros processadores
    - arquivos e outros recursos

Processos e *threads*

- Em um processo podem existir uma ou mais *threads* com
  - um estado de execução (pronta, ...)
  - seu contexto salvo quando não estiver executando
    - diferentes valores de PC dentro de um processo
  - sua pilha de execução
    - cada *thread pode chamar procedimentos*
  - acesso a variáveis locais próprias
  - acesso compartilhado com outras *threads* deste processo aos recursos do processo

Benefícios de *threads*

- É mais rápido criar uma *thread* que um processo
- É mais rápido terminar uma *thread* que um processo
- É mais rápido chavear entre *threads* de um mesmo processo
- *Threads* podem se comunicar sem invocar o núcleo já que compartilham memória e arquivos
  - no caso de comunicação entre processos, a intervenção do núcleo é necessária para proteção e sincronização

Contudo,

- Suspender um processo implica em suspender todas as *threads* deste processo já que compartilham o mesmo espaço de endereçamento
- O término de um processo implica no término de todas as *threads* desse processo

Exemplos de uso de *threads*

- um editor de rascunho
  - trabalho em primeiro e segundo planos: E/S e cálculo em planilhas
  - *thread* 1: mostra menu e lê entrada
  - *thread* 2: atualização do rascunho
- Processamento assíncrono
  - salvamento periódico em editores de texto
- Aumento de velocidade de execução
  - paralelismo
- Organização
  - facilidade de projeto e implementação

Estados de uma *thread*

- Estados fundamentais
  - executando, pronto e bloqueado
- Faz sentido o estado “suspensa”?
- O que acontece com as *threads* de um processo quando uma delas bloqueia?

Operações associadas aos estados das *threads*

- *spawn de um processo*
  - *todas as threads também são criadas*
- *spawn de uma thread*
  - *criados somente* **+ *região p/ stack* + *código e dados locais***
- ***bloqueamento/desbloqueamento***
  - ***uma thread fica bloqueada devido ao evento***
  - ***se torna pronta quando o evento ocorre***

Em que nível implementar?

*Threads no n*ível do usuário

- gerenciamento das *threads* é feito pela aplicação
  - sem intervenção do SO
- o núcleo desconhece a existência de *threads*
- *bibliotecas para*
  - *criação e destruição de threads*
  - *envio de msgs*
  - *escalonamento de threads*
  - *salvamento e recuperação de contexto*
- chaveamento entre *threads* não requer privilégio de modo núcleo

e mais ...

  - implementadas através de bibliotecas: executam em qualquer SO
- Porém:
  - chamada ao sistema bloqueia todas as *threads* de um processo (para o SO é só um processo)
  - não aproveita os benefícios do multiprocessamento (estão em algum processo!)

Em que nível implementar?

Nível do núcleo

- gerenciamento das *threads* é feito pelo núcleo
- núcleo mantém a informação de contexto para processo e *threads*
- escalonamento e chaveamento das *threads* é feito pelo núcleo
- bloqueio de uma *thread* não bloqueia as outras

e ainda ...

  - *threads* podem aproveitar a capacidade de multiprocessamento
  - usuário enxerga uma API para *threads* do núcleo
- Porém:
  - a transferência de controle entre *threads* de um mesmo processo requer chaveamento para modo núcleo
  - chaveamento de threads = troca de contexto (no entanto mais barata que troca de contexto de processos)

Comparando implementações

Latências de operação (µs)

Operação	*Threads*: nível usuário	*Threads*: nível núcleo	Processos
Fork nulo	34	948	11.300
Signal-wait	37	441	1.840

Obs.:

1. VAX monoprocessador executando SO tipo Unix
2. chamada de procedimento neste VAX: ≈ 7µs
3. *trap* ao núcleo: ≈ 17µs

Exemplo de uso

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

void thr_func(int *id); /* codigo threads 1 e 3 */
void thr_yield(int *id); /* codigo thread 2 */

int main(){
pthread_t thr1,thr2,thr3; /* declara as threads */
int nThrID1,nThrID2,nThrID3;

nThrID1 = 1;
nThrID2 = 2;
nThrID3 = 3;

/* cria threads: id, inic, funcao, param funcao */
pthread_create(&thr1,NULL,(void* )thr_func,&nThrID1);
pthread_create(&thr2,NULL,(void* )thr_yield,&nThrID2);
pthread_create(&thr3,NULL,(void* )thr_func,&nThrID3);

/* espera fim das threads: id, status de saida */
pthread_join(thr3,NULL);
pthread_join(thr2,NULL);
pthread_join(thr1,NULL);
}
```

Exemplo de uso (2)

```
/* codigo das threads 1 e 3 */
void thr_func(int *id){
printf("Eu sou a thread %d\n",*id);
}

/* codigo da thread 2 */
void thr_yield(int *id){
sched_yield();
printf("Eu sou a thread %d\n",*id);
}


/* compilacao: gcc -o threads -lthread -lposix4 threads.c */
```

Trabalho 1: Gerência de Processos e Threads

Máximo 3 por grupo;
Trabalho vale 2 pontos;
3 pontos para apresentar;
Entrega e apresentação 21/03.