Universidade Federal de Mato Grosso do Sul

Faculdade de Computação

REDES DE COMPUTADORES

LABORATÓRIO

LAB -08

profa. Hana Karina S. Rubinsztejn - hana@facom.ufms.br

Conteúdo

- Multiprocessamento
  - Threads
- Exercícios

Processos X Threads

- Quando realizamos concorrência por processos (usando a diretiva **fork**), a memória é copiada do pai para o filho, todos os descritores são duplicados no filho e assim por diante.

- Por outro lado, quando realizamos concorrência por ***threads***, todas as *threads* de um processo compartilham a mesma memória global

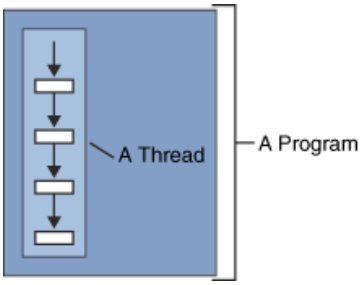
Processos X Threads

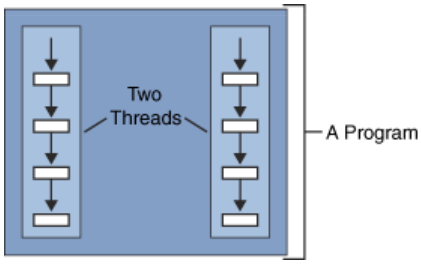
- Além das variáveis globais, também são compartilhados:
  - Instruções do processo
  - Descritores de arquivos
  - Manipuladores de sinais
  - Diretório atual

- Mas cada *thread* tem seu próprio:
  - ID (identificador)
  - Conjunto de registradores, ponteiro da pilha, contador do programa
  - Prioridade
  - Máscara de sinal

Threads

- **Definição:** uma *thread* é um fluxo único de controle sequencial dentro de um programa. Ou seja, uma *thread* não é considerada como um programa.



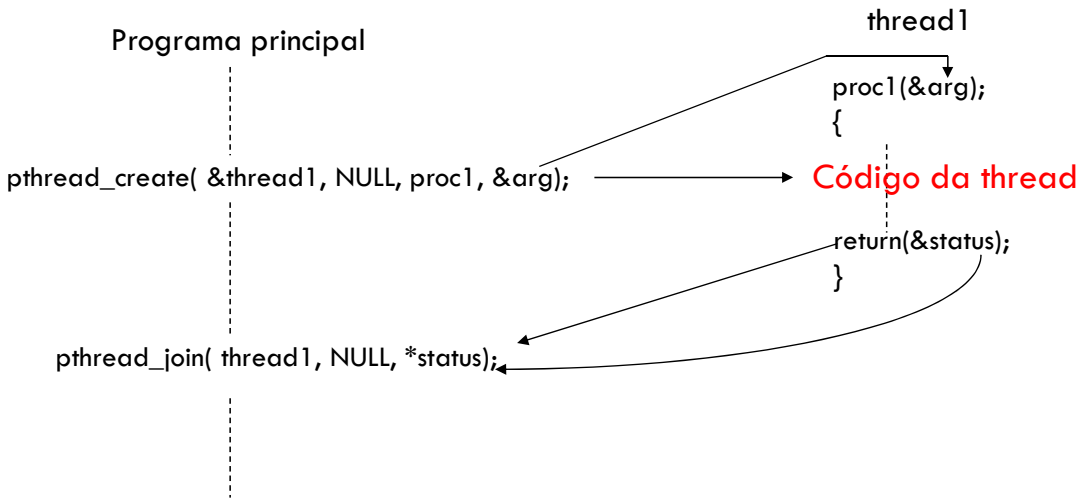


- Alguns autores chamam *thread* de **"contexto de execução"**

Threads

- Um browser é um exemplo de uma aplicação multithreaded. Várias coisas podem ocorrer ao mesmo tempo:
  - *download* de um applet
  - *download* de uma imagem
  - tocar uma animação
  - tocar um som
  - imprimir uma página em *background*
  - *download* de uma nova página
  - olhar 3 applets de ordenação trabalhando

Criação de Thread - Pthreads



Criação de Thread - pthreads

- `#include <pthread.h>`

- ```
  int pthread_create(
  pthread_t *tid,                 // Identificador da thread
  cont pthread_attr_t *attr,      //  Estrutura de configuração
  void *(*func) (void *),         //  Função tratadora
  void *arg);                     // Parâmetros da função tratadora
  ```

- Retorno:
  - 0 se OK,
  - valor Exxx positivo em caso de erro.
  ```

Thread Associável

```
int pthread_join(pthread_t tid, void **status);
```

- Retorno: 0 se OK, valor Exxx positivo em caso de erro.

- A rotina ***pthread_join()*** espera pelo término de uma thread específica. Similar à função `waitpid()` de processos;

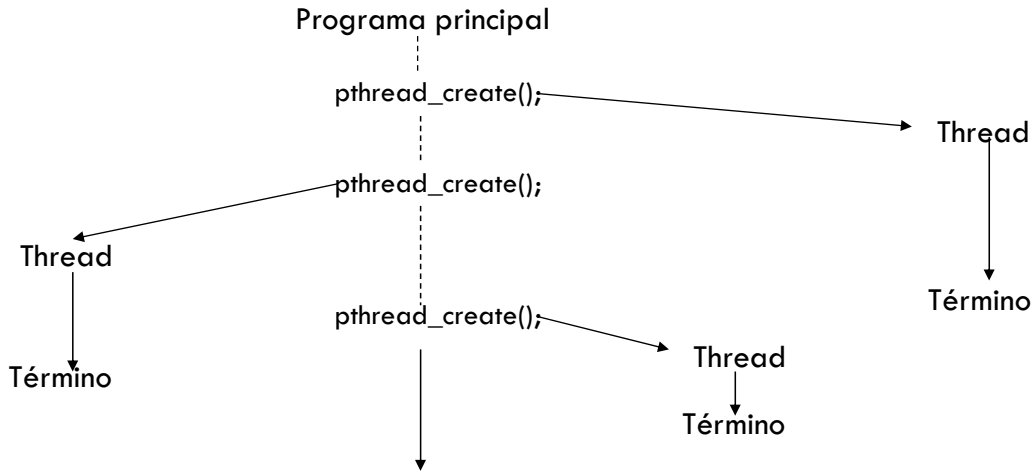
```
for (i = 0; i < n; i++)
      pthread_create(&thread[i], NULL, (void *) slave,
                     (void *) &arg);

...thread mestre
...thread mestre

for (i = 0; i < n; i++)
       pthread_join(thread[i], NULL);
```

Thread Separável

- Pode ser que uma thread não precisa saber do término de uma outra por ela criada, então não precisa executar a operação de união (*join*).
- Neste caso, diz-se que a *thread* criada é *detached* (desunida da thread pai progenitora).
- A função **pthread_detach(pthread_t tid)** altera a thread especificada para que ela seja separável: **pthread_detach(pthread_self());**



Finalização de Threads

- Ou **pthread_join** () ou **pthread_detach** () deve ser chamada para cada *thread* criada pela aplicação, de modo que os recursos do sistema usados pela *thread* possam ser liberados.
  - Mas note que os recursos de todas as threads são liberadas quando o processo pai termina.

Threads: Seção Crítica

- Quando dois ou mais *threads* podem simultaneamente alterar às mesmas variáveis globais, pode ser necessário sincronizar o acesso a esta variável para evitar problemas.

- Código nestas condição diz-se "uma seção critica".
  - Por exemplo, quando duas ou mais threads podem simultaneamente incrementar uma variável x.
  - /* código – Secção Critica */ x = x +1 ;

- Uma seção crítica pode ser protegida utilizando-se ***pthread_mutex_lock***() e ***pthread_mutex_unlock***()

Exemplo 1:

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

void *print_message_function( void *ptr );

main()
{
     pthread_t thread1, thread2;
     char *message1 = "Thread 1";
     char *message2 = "Thread 2";
     int iret1, iret2;

     /* Create independent threads each of which will execute function */
      iret1 = pthread_create( &thread1, NULL, print_message_function,
                                   (void*) message1);
     iret2 = pthread_create( &thread2, NULL, print_message_function,
                                   (void*) message2);
```

Exemplo 1:
A função que vai utilizar thread

```
     /* Wait till threads are complete before main continues. Unless we */
      /* wait we run the risk of executing an exit which will terminate */
      /*  the process and all threads before the threads have completed. */
     pthread_join( thread1, NULL);
     pthread_join( thread2, NULL);

     printf("Thread 1 returns: %d\n",iret1);
     printf("Thread 2 returns: %d\n",iret2);
     exit(0);
 }

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

Exemplo 2: Threads e sockets

```
void *thread_faca_alguma_coisa(void *arg)

int main(int *argc, char *argv[])
{
    ...
    int *iptr;
    pthread_t tid;
    ...
    for( ; ; )
    {
        iptr = (int *) malloc (sizeof(int));
        *iptr = accept(listenfd, (SA *) NULL, NULL);
        pthread_create(&tid, NULL, &thread_faca_alguma_coisa, iptr);
    }
    close(listenfd);
    exit(0);
}
```

- **Note que e necessário alocar iptr para cada thread.**

Exemplo 2: Threads e sockets
A função que vai utilizar thread

```
void *thread_faca_alguma_coisa(void *arg)
{
    int connfd;
    char buff[MAXLINE];
    time_t ticks;

    connfd = *((int *) arg);
    pthread_detach(pthread_self());

    ticks = time(NULL);
    snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
    write(connfd, buff, strlen(buff));

    close(connfd);
    return(NULL);
}
```

Compilando o código

- Para compilar seu código com threads:
  - $ gcc codigo.c -pthread -o codigo

Exercícios

- Exemplo – Execute o servidor echo implementado com *threads*.
- **Exercício 05 – Entregar no moodle:**
  - **Difusor Echo**: Faça com que o servidor multi-thread ecoe para todos os clientes a mensagem recebida.
    - Dicas: faça um vetor de clientes ativos (que guarda os sockets dos clientes), e armazene na estrutura da thread cliente além do socket, também sua posição no vetor
    - Altere o cliente para que ele funcione em modo multi-thread, isto é, que ele seja capaz de escutar simultaneamente mensagens vindas do teclado e da rede.

Tutorial p/ Threads

- **POSIX thread (pthread) Libraries**

- http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html