A biblioteca pthreads

Weslley Lima

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

Estrutura e funções da biblioteca

- pthread_t (struct)
- pthread_create
- pthread_join
- pthread_self
- pthread_exit

Exemplo

```
#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;
```

Exemplo

```
iret1 = pthread_create( &thread1, NULL,
  print_message_function, (void*) message1);
iret2 = pthread_create( &thread2, NULL,
  print_message_function, (void*) message2);
pthread_join( thread1, NULL);
pthread_join( thread2, NULL);
printf("Thread 1 returns: %d\n",iret1);
printf("Thread 2 returns: %d\n",iret2);
exit(0);
```

Exemplo

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

- Compilando:
  - **gcc -lpthread exemplo.c -o exemplo**

Descrição da funções

- Uma thread pode ser finalizada de três formas:
  - Chamando pthread_exit
  - return da função que criou a thread
  - Chamando exit no processo pai

pthread_create

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

- Argumentos
  - thread – retorna o id da thread
  - attr – Altera atributos da thread, tal como, tamanho da pilha, endereço da pilha e política de escalonamento. Setada em NULL por default;
  - start routine – função que dará origem à thread. O retorno e os argumentos devem obrigatoriamente ser ponteiro para void;
  - arg – ponteiro para os argumentos da função

pthread_join

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

- Aguarda o término de outra thread
- Argumentos
  - th – suspende a thread até a thread identificada por th terminar.
  - thread_return – Se thread_return não é nulo, o valor do retorno de th é armazenado na localização apontada por thread_return

pthread_exit

```
void pthread_exit(void *retval);
```

- Termina a thread que chamar tal função.
  - retval – Valor de retorno da thread

Sincronização de threads

- Mecanismos disponíveis
  - Mutex – bloqueia o acesso a outras variáveis por outras threads.
  - joins – Faz uma thread esperar até outra completar seu trabalho.
  - Variáveis de condição

Mutex – Exemplo

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

void *functionC();
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
int  counter = 0;
main()
{
   int rc1, rc2;
   pthread_t thread1, thread2;
  if((rc1=pthread_create( &thread1, NULL, &functionC, NULL)))
   {
      printf("Thread creation failed: %d\n", rc1);
   }
```

Mutex – Exemplo

```
if( (rc2=pthread_create( &thread2, NULL, &functionC, NULL)) )
   {
      printf("Thread creation failed: %d\n", rc2);
   }
 pthread_join( thread1, NULL);
 pthread_join( thread2, NULL);
 exit(0);
}
void *functionC()
{
   pthread_mutex_lock( &mutex1 );
   counter++;
   printf("Counter value: %d\n",counter);
   pthread_mutex_unlock( &mutex1 );
}
```

Variáveis de condição

- Permite que threads suspendam sua execução e abandonem o processador até uma condição tornar-se verdadeira;
- Deve sempre ser associada a um mutex para evitar condições de corrida;

Variáveis de condição

- Funções
  - pthread_cond_init
  - pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
  - pthread_cond_destroy
  - pthread_cond_wait – desbloqueia o mutex e aguarda um sinal na variável de condição.
  - pthread_cond_timedwait – impõe limites de espera em caso de bloqueio.
  - pthread_cond_signal – reinicia uma thread que estivesse aguardando por uma variável de condição.
  - pthread_cond_broadcast – acorda todas as threads bloqueadas por uma variável de condição.

Exemplo

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

pthread_mutex_t count_mutex   = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t  condition_var = PTHREAD_COND_INITIALIZER;

void *functionCount1();
void *functionCount2();
int  count = 0;
#define COUNT_DONE  10
#define COUNT_HALT1  3
#define COUNT_HALT2  6
```

Exemplo

```
main()
{
   pthread_t thread1, thread2;

   pthread_create( &thread1, NULL, &functionCount1, NULL);
   pthread_create( &thread2, NULL, &functionCount2, NULL);

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

   printf("Final count: %d\n",count);

   exit(0);
}
```

Exemplo

```
void *functionCount1()
{
  for(;;)
  {
    pthread_mutex_lock( &count_mutex );
    pthread_cond_wait( &condition_var, &count_mutex );
    count++;
    printf("Counter value functionCount1: %d\n",count);
    pthread_mutex_unlock( &count_mutex );
    if(count >= COUNT_DONE) return(NULL);
   }
}
```

Exemplo

```
void *functionCount2()
{
   for(;;)
   {
     pthread_mutex_lock( &count_mutex );
     if( count < COUNT_HALT1 || count > COUNT_HALT2 )  {
       pthread_cond_signal( &condition_var );
     }
     else  {
       count++;
       printf("Counter value functionCount2: %d\n",count);
     }
     pthread_mutex_unlock( &count_mutex );
     if(count >= COUNT_DONE) return(NULL);
   }
}
```

Trabalho

- Escolha um dos problemas clássicos de comunicação entre processos estudado em sala e implemente-o em C utilizando threads para criar os participantes do problema e mutex e variáveis de condição para garantir a exclusão mútua.
- Problemas:
  - Filósofos comilões
  - Leitores e Escritores
  - Produtores e consumidores