Programación Avanzada

Curso 2023/2024

prof. Claudio Rossi

Practica 6: uso de Threads

Los *threads*, o *procesos ligeros* (o también *hilos*), permite la ejecución paralela de distintas funciones dentro de un proceso.

Objetivos:

1. Uso básico de threads
2. Sincronizacíon de threads
3. Problema propuesto

Comandos de S.O.:

- top: imprime en pantalla información de los procesos y threads activos

Funciones utilizadas:

- int *pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void *), void *arg):* crea un nuevo thread, que ejecutará la función *start_routine*. Los parámetros de la función se pasan como último parámetro (void* arg). El id del thread es guardado en el primer parámetro. El parámetro *attr* es una lista de estructuras que permite especificar comportamentos específicos del thread, como por ejemplo prioridad, política de scheduling, tamaño max de su stack etc. (no usado aquí). Retorna 0 en caso de éxito, o un código en caso de error.

- *int pthread_join(pthread_t thread, void **value_ptr)*: Parecido a la wait: el proceso principal espera a la terminación del thread.

- void *pthread_exit(void *value_ptr)*: termina el thread, retornando *value_ptr* como valor de retorno.

- pthread_t *pthread_self()*: devuelve el id del proceso (unsigned long)

Parte 1.
Se escribirá un programa que lanza dos threads que ejecutarán dos funciones. Una de ellas to tiene ningún parámetro, mientras otra recibe una referencia a una variable de tipo *float*. Comprobar el valor de la variable x que aparece en pantalla !

Programa 1

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


int una_variable_global = -1;

void* function1(void *arg)
{
    int i = 0;
    pthread_t my_id = pthread_self();

    for (i=0;i<5;i++)
      {
      una_variable_global = i;
      printf("\n Hello, soy el thread 1 (%lu) y v.g. vale %d\n",(unsigned
long) my_id,una_variable_global);
      sleep(1);
      }

    printf("\n T1 says: bye bye !\n");


    return NULL;
}

void* function2(void *arg)
{
    pthread_t my_id = pthread_self();
    int i;

    float *la_x;

    la_x = (float*)arg;


      for (i=0;i<8;i++)
      {
      una_variable_global -= 3;
      printf("\n Hello, soy el thread 2 (%lu) y v.g. vale %d\n",(unsigned
long) my_id,una_variable_global);
      sleep(1);
      }

    *la_x = 0.456;

    printf("\n T2 says: Hasta luego lucas !\n");

    return NULL;
}
```

```
int main(void)
{

  pthread_t t1_id,t2_id;
  int i,err;

  float x=0.123;

  err = pthread_create(&t1_id, NULL, &function1, NULL);
  if (err != 0)
    printf("\ncan't create thread :[%s]", strerror(err));
  else
    printf("\n Thread created successfully\n");


  err = pthread_create(&t2_id, NULL, &function2, &x);
  if (err != 0)
    printf("\ncan't create thread :[%s]", strerror(err));
  else
    printf("\n Thread created successfully\n");

  for (i=0;i<15;i++)
      {
      printf("\n Hello, soy el proceso principal: x vale %4.3f y la var.
global vale %d\n",x,una_variable_global);
      sleep(1);
      }    return 0;
}
```

Parte 2.

(A) Sincronización de threads

POSIX proporciona dos primitivas para la sincronización de los threads. La primera, llamada "*Mutex*" permite regula la mutua exclusión (acceso exclusivo a la sección critica). La segunda, llamada "*Condition variables*" son variables compartidas que permiten a un thread quedarse a la espera hasta que otro thread le señale (a través de la variable) que puede seguir.

Funciones utilizadas:

- int *pthread_mutex_lock(pthread_mutex_t *mutex)*: pide acceso al mutex
- int *pthread_mutex_unlock(pthread_mutex_t *mutex)*: libera el mutex
- int *pthread_mutex_init(&mut,&attr)*: inicializa el mutex (*attr* será casi siempre NULL)

- int *pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex)*: pone el thread a la espera (suspenso) de *cond*.
- int *pthread_cond_signal(pthread_cond_t *cond)*: despierta el thread a la espera de *cond*.
- int *pthread_cond_init(&cond, &attr)*: inicializa la cond. var. (*attr* será casi siempre NULL)

Estructuras de datos utilizada:

pthread_mutex_t
pthread_cond_t

Modificar el ejemplo anterior de forma que el thread 1 se suspenda hasta que el thread 2 haya hecho tres iteraciones de su bucle. Para ello se usará una *condition variable*, y un *mutex* asociado a ella. Las dos funciones tendrán la siguiente forma:

```
void* function1(void *arg)
{
    int i = 0;

    printf("\n Hello, soy el thread 1 y me voy a dormir...\n");

    // PONER AQUÍ EL CODIGO NECESARIO PARA QUE
    // ESTE THREAD SE SUSPENDA HASTA SER DESPERTADO POR EL OTRO

    for (i=0;i<3;i++)
      {
       printf("\n Hello, soy el thread 1\n");
       sleep(1);
      }

    printf("\n T1 says: bye bye !\n");


    return NULL;
}
```

```
void* function2(void *arg)
{
   int i;


   for (i=0;i<10;i++)
      {
       printf("\n Hello, soy el thread 2\n" );
        sleep(1);
       if(i==3)
         {
            // PONER AQUÍ EL CODIGO PARA DESBLOQUER EL THREAD 1
        }
      }


    printf("\n T2 says: Hasta luego lucas !\n");

    return NULL;
}
```

```
// Añadir estas variables GLOBALES
pthread_mutex_t  mut ;
pthread_cond_t   cond ;


// en el main, antes de lanzar los threads, inicializar el mutex
// y la condition variable:

pthread_cond_init(&cond,NULL);
pthread_mutex_init(&mut,NULL);
```

(B) alternancia estricta de threads

Modificar el código proporcionado (`thr_sync_strict_alt_lost_wakeups.c`) para resolver el probleba de las "LOST WAKEUPS" y funcione correctamente.