UNIVERSIDAD NEBRIJA

Grado en Ingeniería Informática

Sistemas Operativos

Práctica 2

Índice

Práctica 2: Gestión de Threads (hilos)

Introducción

La práctica anterior nos presentó cómo a través de POSIX, el programador era capaz de gestionar la creación de procesos para configurar sus propias aplicaciones, pudiendo hacer uso de la concurrencia que aporta el sistema operativo multitarea.

En el caso de esta práctica, saltamos a presentar los threads o hilos que los sistemas operativos saben gestionar de forma parecida a cómo lo hacen a nivel de procesos.
Los threads se dicen ser procesos ligeros dado que el cambio de contexto que lleva a cabo el sistema operativo es menos costoso a la hora de su planificación debido a que diferentes hilos dentro del mismo proceso comparten:

- Las instrucciones del proceso.
- Mayoría de los datos.
- Descriptores de ficheros abiertos.
- Señales y manejadores de señales.
- ID de usuario y de grupo.

Por el contrario cada thread tiene único:

- Thread ID.
- Conjunto de registros y puntero a pila.
- Pila para variables locales.
- Prioridad de ejecución.
- Dirección de inicio de la ejecución.

Un thread se representa mediante su TID de tipo pthread_t. Normalmente las llamadas al sistema para gestión de threads devuelven 0 si se ha llevado a cabo con éxito y un valor no nulo en caso de error.

Funciones de gestión básica de threads:

Entre las funciones básicas que se pueden encontrar a la hora de gestionar threads podemos encontrar las siguientes:

Función	Descripición
pthread_create	Crea un thread para ejecutar una función determinada
pthread_exit	Causa la terminación del thread que lo invoca
pthread_attr_init	Inicializa los atributos del thread a su valor por defecto
pthread_join	Hace que el thread que la invoca espere a que termine un thread determinado
pthread_self	devuelve la identidad del thread que lo invoca
pthread_cancel	Solicita la terminación de otro thread

La interfaz de las anteriores funciones se puede ver a continuación.

```
#include <pthread.h>
int pthread_create (pthread_t *tid, const pthread_attr_t
*attr, void *(*start_routine)(void*), void *arg);
```

Parámetros

- *tid*: apunta al thread que se crea
- *attr*: atributos del thread creado (tipo *pthread_attr_t*)
- *start_routine*: puntero a la función que debe ejecutar el thread
- *arg*: puntero a los argumentos que se pasan a la función

```
#include <pthread.h>
int pthread_attr_init (pthread_attr_t *attr);
```

```
#include <pthread.h>
pthread_t pthread_self (void);
```

```
#include <pthread.h>
void pthread_exit (void *value_ptr);
```

Dependencias entre threads: pthread_join, hace que el thread que la invoca espere a que termine un determinado thread

```
#include <pthread.h>
int pthread_join (pthread_t thread, void **value_ptr);
```

- Permite que varios threads cooperen en una tarea
- thread es el TID del thread que esperamos que termine
- En value_ptr se almacena el código de salida del thread que termina (si es NULL no guarda nada)

Ejemplo integral de la gestión de threads

A continuación se muestra un ejemplo que integra la gestión de threads usando algunas de las llamadas principales que se han visto.

```
// Ejemplo de utilización de pthread_join
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

/* Prototipos de las funciones que ejecutan los threads */
void *func1 (void *);
void *func2 (void *);

pthread_t thread1, thread2, thmain;     /* Declaración de los threads */
pthread_attr_t attr;                    /* atributos de los threads*/

/* Definición de las funciones func1 y func2 */
void *func1 (void *arg)
{
          printf("Soy el thread1 y estoy ejecutando func1 \n");
          sleep(4);
          printf("Soy el thread 1 termino \n");
          pthread_exit (NULL);
}
```

```
void *func2 (void *arg)
{
          int err;
          printf("Soy el thread2 y esperando que thread1 termine \n");
          if (err =  pthread_join(thread1, NULL))
                    printf ("Error al esperar a thread1 \n");
          else
          {
            printf("Soy thread2, thread1 ha terminado y ejecuto func2 \n");
            sleep(2);
          }
          pthread_exit (NULL);
}
```



```
/*Función main*/
int main(void)
{
          thmain = pthread_self();

          /*inicializa los parámetros de los threads por defecto*/
          pthread_attr_init (&attr);

          printf("Soy la función main y voy a lanzar los dos threads \n");

          pthread_create (&thread1, &attr, func1, NULL);
          pthread_create (&thread2, &attr, func2, NULL);

          printf("Soy main: he lanzado los dos threads y termino
\n");    pthread_exit(NULL);
}
```

```
[/u0/sitr/sitr001/threads]th2
Soy la funci©n main y voy a lanzar los dos threads
Soy main: he lanzado los dos threads y termino
Soy el thread1 y estoy ejecutando func1
Soy el thread2 y voy a esperar que thread1 termine
Soy el thread1 termino
Soy th2, th1 ha terminado y estoy ejecutando func2
[/u0/sitr/sitr001/threads]
```

Elementos para la sincronización de threads

Para la sincronización de hilos, las implementaciones ofrecen las siguientes herramientas:

- Mutex (esperas a corto plazo).
- Semáforos.
- Variables condicionales (esperas a largo plazo).

Exclusión mutua (mutex o candado): Sirve para obtener acceso exclusivo a las variables o recursos.

Variables de tipo pthread_mutex_t.

Funciones básicas para gestión de sección crítica (mutex):

Función	Descripción
pthread_mutex_init	Inicializa una variable de tipo `pthread_mutex_t`
pthread_mutex_destroy	Destruye una variable mutex
pthread_mutex_lock	Protege la sección crítica (operación P)
pthread_mutex_unlock	Libera la protección de la sección crítica (op. V)
pthread_mutex_trylock	Como `pthread_mutex_lock` pero sin bloqueo

```
#include <pthread.h>
int pthread_mutex_init (pthread_mutex_t *mutex,
                        const pthread_mutexattr_t *attr);
```

Dos parámetros
- Puntero a una variable pthread_mutex_t (mutex)
- objeto atributo de mutex (attr)
- Se inicializa por defecto haciendo el segundo parámetro NULL
- Devuelve 0 si se ha podido inicializar el mutex.

```
#include<pthread.h>
pthread_mutex_t my_lock;
if (pthread_mutex_init(&my_lock, NULL)!=0)
        perror("No puedo inicializar my_lock");
```

```
#include <pthread.h>
int pthread_mutex_destroy (pthread_mutex_t *mutex);
```

- El mutex no se destruye hasta que está completamente vacío (no hay threads en espera)
- Devuelve 0 si se ha podido liberar el mutex.

```
#include <pthread.h>
int pthread_mutex_lock (pthread_mutex_t *mutex);
```

pthread_mutex_lock: operación P de semáforos

- Código de la sección de entrada que protege la sección crítica
- Si el candado está cerrado, el thread que la invoca se suspende hasta que el candado quede libre
- Devuelve 0 si se ha podido bloquear el mutex.

```
#include <pthread.h>
int pthread_mutex_unlock (pthread_mutex_t *mutex);
```

pthread_mutex_unlock: operación V de semáforos

- Libera el candado cuando un thread termina su sección crítica
- Si hay algún thread suspendido en la cola del candado, despierta al primero.
- Devuelve 0 si se ha podido desbloquear el mutex.

Ejemplo:

```
#include<pthread.h>
pthread_mutex_t my_lock;
pthread_mutex_init(&mylock, NULL);

pthread_mutex_lock(&my_lock);
/* Sección crítica */
pthread_mutex_unlock(&my_lock);
```

Entregables:

1. Implementar un programa que reciba dos números enteros como parámetros de entrada y calcule sus factoriales de forma concurrente utilizando dos hilos que se ejecutan en paralelo con el hilo principal. El hilo principal deberá esperar a que terminen los otros dos hilos.

2. El programa que se muestra a continuación cuenta el número de veces que el carácter 'a' o 'A' aparece en el fichero indicado como parámetro de entrada. Modificarlo para que ahora se cree un hilo y sea éste el que ejecute la función de contar.

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define MAXLON 1000

void cuenta (char *nombre) {

  int pos, cont= 0, leidos;
  char cadena[MAXLON];
  int fd;

  fd= open (nombre, O_RDONLY);
  while ((leidos= read (mf, cadena, MAXLON))!= 0)
    for (pos= 0; pos< leidos; pos++)
      if ((cadena[pos]== 'a') || (cadena[pos]== 'A'))
        cont++;
  printf ("Fichero %s: %d caracteres 'a' o 'A' encontrados\n", nombre, cont);
  close (fd);
}

int main (int argc, char *argv[]) {

  if (argc!= 2) {
    printf ("Indica el nombre de un fichero.\n");
    exit(0);
  }
  cuenta (argv[1]);
  return 0;
}
```

3. Generar un programa en C que muestre el funcionamiento de mutex a través de la creación de un par de threads, donde cada uno escriba su propio mensaje por la salida estándar de forma sincronizada secuenciando las tareas.

Nota: Para compilar con la librería thread.h se debe añadir como argumento ***-lpthread*** a la orden gcc.

Error:

```
gcc thread.c -o thread
thread.c: undefined reference to 'pthread_create'
```

Correcto:

```
gcc thread.c -o thread -lpthread
$ ./thread
```

Nota: para liberar el uso de la CPU en un punto intermedio de la ejecución se puede usar la llamada del sistema ***int sched_yield(void)***, en caso de querer comprobar el correcto uso de la sección crítica.