SESIÓN N° 06

PROGRAMACIÓN DE HILOS

I OBJETIVOS

- Conocer y aplicar los conceptos sobre Hilos en LINUX.
- Gestionar los Hilos.
- Realizar ejercicios.

Tiempo estimado: 4 horas

II TEMAS A TRATAR

- Hilo o hebra
- Gestión de hilos

III CONSIDERACIONES DE EVALUACIÓN

- No debe utilizar semáforos para sincronizar los hilos
- Los ejercicios pueden ser realizados en colaboración con un compañero.
- Al momento de la revisión los miembros del grupo pueden ser requeridos de realizar modificaciones al código y responder preguntas sobre el mismo.

IV MARCO TEORICO

En esta sesión trataremos cómo usar múltiples hilos de control para efectuar múltiples tareas dentro de un mismo proceso.

Hilo o hebra

Un proceso típico de Unix puede ser visto como un único **hilo de control**: cada proceso hace sólo una cosa a la vez. Con múltiples **hilos de control** podemos hacer más de una cosa a la vez cuando cada hilo se hace cargo de una tarea específica.

Algunos beneficios del uso de hilos son:

- Se puede manejar eventos asíncronos asignando un hilo a cada tipo de evento. Luego cada hilo maneja sus eventos en forma sincrónica.
- Los hilos de un proceso comparten el mismo espacio de direcciones y descriptores de archivos.

- Procesos con múltiples tareas independientes pueden terminar antes si estas tareas se desarrollan traslapadamente en hilos separados. De este modo los tiempos de espera de la primera tarea no retrasan la segunda.
- La creación de un hilo es mucho más rápida y toma menos recursos que la creación de un proceso.

Un hilo contiene la información necesaria para representar un contexto de aplicación dentro de un proceso. Ésta es:

- ID del hilo. No son únicos dentro del sistema, sólo tienen sentido en el contexto de cada proceso.
- Stack pointer
- Un conjunto de registros
- Propiedades de itineración (como política y prioridad)
- Conjunto de señales pendientes y bloquedas.
- Datos específicos del hilo.

Gestión de Hilos

Los sistemas operativos basados en UNIX y en el estándar POSIX de IEEE disponen y aceptan una biblioteca multiplataforma para la manipulación de hilos en lenguaje C. Esta biblioteca se conoce como *pthread.h*, la cual debe ser incluida en los programas. Esta biblioteca permite el uso de funciones como las que se presentan en la Fígura 7.1.

Thread call	Description
pthread_create	Create a new thread in the caller's address space
pthread_exit	Terminate the calling thread
pthread_join	Wait for a thread to terminate
pthread_mutex_init	Create a new mutex
pthread_mutex_destroy	Destroy a mutex
pthread_mutex_lock	Lock a mutex
pthread_mutex_unlock	Unlock a mutex
pthread_cond_init	Create a condition variable
pthread_cond_destroy	Destroy a condition variable
pthread_cond_wait	Wait on a condition variable
pthread_cond_signal	Release one thread waiting on a condition variable

Figura 7.1. Algunas llamadas o funciones de la biblioteca pthread en C

Creación de Hilos

Para crear un nuevo hilo, la función que crea e inicia la ejecución de un nuevo hilo (lo pone en la cola del planificador) tiene esta estructura:

```
int pthread_create(pthread_t *nuevo_hilo, const pthread_attr_t *atributos,
void *<NOMBRE_FUNCION> (void *), void *arg)
```

El parámetro **`nuevo_hilo`** apunta al identificador del hilo que se crea. Este identificador se utiliza en la declaración previa del hilo del tipo **`pthread_t`**. Los atributos del hilo (p.ej. tamaño de pila o política de planificación) se encapsulan en el objeto al que apunta **`atributos`**, normalmente este parámetro es **`NULL`**. El tercer parámetro **`<NOMBRE_FUNCION>`** se refiere al nombre de la función que ejecutará el hilo.

Cuando la función devuelva el control implícitamente, el hilo finalizará. Si la función necesita un parámetro (el único), se especifica con **`arg`** (o **`NULL`** si no lo tiene).

Veamos un ejemplo:

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


void hilo1() {
  char *msg = “Hola ”;
  int i;
  int aleatorio;
  for(i = 0; i < strlen(msg); i++){
     printf("%c", msg[i]);
     fflush(stdout);
     aleatorio = rand();
     sleep(1 + (aleatorio % 5));
  }
  return NULL;
}

void hilo2(){
  char msg = “mundo”;
  int i;
  int aleatorio;
  for(i = 0; i < strlen(msg); i++){
     printf("%c", msg[i]);
     fflush(stdout);
     aleatorio = rand();
     sleep(1 + (aleatorio % 5));
  }
  return NULL;
}

int main(int argc, char *argv[]){
  srand((unsigned)time(NULL));
  pthread_t h1;
  pthread_t h2;
  pthread_create (&h1, NULL, hilo1, NULL);
  pthread_create (&h2, NULL, hilo22, NULL);
  printf("Fin\n");
}
```

Nota: Para compilar un programa en C que utiliza hilos debe linkearse con la biblioteca de hilos así:

```
$ cc -lpthread programa.c -o programa
```

Esperar la terminación de un hilo

Al ejecutar el programa anterior, es bastante improbable que se llegue a ver nada en la consola. La razón es que los hilos que se crean en el programa principal terminan automáticamente cuando el programa principal termina. Por tanto, es necesario un

mecanismo de sincronización que permita esperar a la terminación de un hilo. Este mecanismo es el que implementa la función `pthread_join`.

Para comprobarlo añada las siguientes líneas al programa anterior justo antes de la sentencia para escribir "Fin" en el programa principal y comprueba como la palabra Fin aparece tras la terminación de los dos hilos.

```
pthread_join(h1 , NULL);
pthread_join(h2 , NULL);
```

Pasando parámetros a los hilos

Las funciones que se ejecutan en los hilos tienen acceso a las variables globales declaradas antes que la propia función. Pero si se desea pasar un parámetro concreto a la función que se ejecuta en un hilo dicho parámetro debe ser de tipo puntero a `void`, por lo que se requiere realizar hacer *casting*, tanto al momento del envío del parámetro como dentro de la función manejadora del hilo.

Los argumentos se pasan a la función a través del cuarto parámetro de `pthread_create`.

Cuando se necesita pasar más de un parámetro a una función ejecutada en un hilo hay que recurrir a las estructuras (**`struct`**). Agrupando todos los parámetros en una estructura y pasando un puntero a dicha estructura, la función tendrá acceso a todos los argumentos.

Por ejemplo, veamos cómo hacer una multiplicación de una matriz por un escalar dentro de un hilo.

```
/*Ejercicio Matriz */
# include <stdio .h>
# include <stdlib .h>
# include <string .h>
# include <unistd .h>
# include <pthread .h>

struct parametros {
  int id;
  float escalar ;
  float matriz [3][3];
};
void init (float m [3][3]) {
  int i;
  int j;
  for ( i = 0 ; i < 3 ; i++ ) {
     for ( j = 0 ; j < 3 ; j++ ) {
        m[i][j] = random () *100;
     }
  }
}
void * matrizporescalar( void *arg ) {
  struct parametros *p;
  int i;
  int j;
  p = (struct parametros *) arg ;
  for ( i = 0 ; i < 3 ; i++ ) {
     printf (" Hilo %d multiplicando fila %d\n", p -> id , i);
     for ( j = 0 ; j < 3 ; j++ ) {
        p -> matriz [i][j] = p -> matriz [i][j] * p -> escalar ;
        sleep (5) ;
     }
```

```
  }
}
int main(int argc , char *argv []) {
  pthread_t h1;
  struct parametros p1;
  p1.id = 1;
  p1.escalar = 5.0;
  init (p1.matriz );
  pthread_create (&h1 , NULL , matrizporescalar , ( void *)&p1);
  pthread_join(h1 , NULL);
  printf ("Fin \n");
}
```

Retorno de valores de un hilo

El siguiente programa muestra un ejemplo para devolver un entero desde un hilo al programa padre utilizando casteo.

```
#include <pthread.h>
#include <stdio.h>
void* computo (void* arg) {
  int candidato = 2;
  int n = *((int*) arg);
  while (1) {
      int factor;
      int es_primo = 1;
      for (factor = 2; factor < candidato; ++factor)
         if (candidato % factor == 0) {
             es_primo = 0;
             break;
         }
      if (es_primo) {
         n--;
         if (n == 0)
             return (void*) candidato;
      }
      ++candidato;
  }
  return NULL;
}
int main () {
  pthread_t thread;
  int iesimo_primo = 10000;
  int primo;
  pthread_create (&thread, NULL, &computo, &iesimo_primo);
  pthread_join (thread, (void*) &primo);
  printf("El %d iesimo numero primo es %d.\n", iesimo_primo, primo);
  return 0;
}
```

Liberación de recursos

Los recursos asignados por el sistema operativo a un hilo son liberados cuando el hilo termina. La terminación del hilo se produce cuando la función que está ejecutando termina, cuando ejecuta un `return` o cuando ejecuta la funcón `pthread_exit`. Si la función no ejecuta ninguna de estas dos funciones, se ejecuta automáticamente un `return 0`. Teniendo

en cuenta que las funciones que se ejecutan en hilos deben devolver un puntero a `void`, en este caso, el valor devuelto es un puntero con valor `NULL`.

IV ACTIVIDADES

01. En el siguiente ejercicio se intenta medir el tiempo de creación de 100 procesos (pesados) a partir un proceso padre, y el tiempo de creación de 100 hilos. Observe los resultados de ambos programas.

```
# include <stdio .h>
# include <stdlib .h>
# include <time.h>
int main () {
  struct timeval t0, t1;
  int i = 0;
  int id = -1;
  gettimeofday(&t0 , NULL);
  for ( i = 0 ; i < 100 ; i++ ) {
     id = fork ();
     if (id == 0) return 0;
  }
  if (id != 0) {
     gettimeofday (&t1 , NULL);
     unsigned int ut1 = t1. tv_sec *1000000+ t1. tv_usec ;
     unsigned int ut0 = t0. tv_sec *1000000+ t0. tv_usec ;
     /* Tiempo medio en microsegundos */
     printf (" %f\n", (ut1 -ut0 ) /100.0) ;
  }
}
```

```
# include <stdio .h>
# include <stdlib .h>
# include <time.h>
# include <pthread .h>
struct timeval t0 , t1;
double media = 0.0;
void * hilo(void *arg ) {
  gettimeofday(&t1 , NULL);
  unsigned int ut1 = t1. tv_sec *1000000+ t1. tv_usec ;
  unsigned int ut0 = t0. tv_sec *1000000+ t0. tv_usec ;
  media += (ut1 -ut0 );
}
int main () {
  int i = 0;
  pthread_t h;
  for ( i = 0 ; i < 100 ; i++ ) {
     gettimeofday (&t0 , NULL);
     pthread_create (&h, NULL , hilo , NULL);
     pthread_join(h, NULL);
  }
  /* Tiempo medio en microsegundos */
  printf (" %f\n", ( media /100.0) );
}
```

V
EJERCICIOS

01. Modificar el programa *Ejercicio_Matriz* de tal forma que se puede multiplicar una matriz de 4 x 4 por el escalar 10, con la ayuda de un segundo hilo.

02. Escriba un programa que cree 3 hilos, cada uno de los cuales generará aleatoriamente 10 valores enteros entre 1 y 100. Entre cada generación deberá haber una espera aleatoria entre 1 y 3. El valor generado por cada hilo se deberá sumar en una variable global, de modo que, al finalizar la ejecución de todos los hilos, esta variable contenga la suma total de los valores generados por los 3 hilos. Se debe tener en cuenta que cada hilo deberá informar del número que él genere mostrándolo en la pantalla. Tenga en cuenta que cada vez que un hilo muestre información en pantalla deberá identificarse.

03. Se trata de que usted escriba una solución que utilice hilos que nos permita realizar cualquier actividad que conste de una serie de tareas las mismas que están declaradas en un archivo de texto (denominado *actividad.txt)*, de las cuales varias pueden ser realizadas a la vez, pero algunas dependen de que otras terminen.

    La estructura del fichero tiene la siguiente explicación, los campos están separados por un guion.

    - Primer campo: Identifica el número de la tarea (se podría decir es como el identificador del paso que hay que realizar)
    - Segundo campo: Identifica la descripción de la tarea. (Lo que haríamos al realizar la actividad en ese paso)
    - Tercer campo: Segundos que tardamos en realizar esa tarea (tiempo que va a tardar el hilo en acabar).
    - Cuarto campo: Las tareas que tienen que hacerse antes de comenzar esa tarea (puede contener más de una).

    Por ejemplo, si la actividad consiste en “cocinar un plato”, el contenido del archivo sería el siguiente:

```
T0-Hervir el agua-8
T1-Hacer el sofrito-25
T2-Poner la pasta-1-T0
T3-Hervir la pasta-5-T2
T4-Colar la pasta-2-T3
T5-Mezclar sofrito-2-T1+T4
T6-Servir-3-T5
```

    En programa deberá de leer la información del archivo y en función de ella, simular la ejecución de las tareas descritas en el fichero.

    Siga el siguiente esquema para el programa solicitado:

```
/*Definir una estructura que contendrá los datos de cada tarea
*/
typedef struct {
  char id[3];
  char desc[50];
  int temp;
  char pre[6];
```

```
}tarea;

/*Esta función se encarga de leer por pantalla el nombre del fichero
  y comprobar si el fichero existe.
*/
void *thLectura(void *arg) {

}

/* Esta función se encarga de leer el archivo de texto, y pasar la
   información a la memoria (arreglo de tareas).
   Devuelve el número de tareas de la actividad.
*/
int leerFichero(char fichero[], plato llistaPasos[]){

}

/* Esta función se encarga de llevar a cabo cada una de las tareas
   definidas para la actividad.
   Muestra por pantalla cuando inicia la tarea, espera el tiempo que el
   fichero indica.
   Informa de que ha finalizado la tarea y cierra el hilo.
*/
void *hacerTarea(void *arg) {

}

int main(){
  // crear un hilo para leer por pantalla el nombre de un fichero y
  //  verificar su existencia

  // llamar a la funcion que lee el contenido del fichero a un arreglo de
  //  Tareas

  // Para cada tarea de la actividad que no tenga tareas predecesoras
  // crear un hilo, de lo contrario esperar a que estas tareas
  // predecesores terminen para realizar la tarea.


}
```