UNIVERSIDAD NACIONAL DE SAN AGUSTIN

FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA

R. JESÚS CÁRDENAS TALAVERA
Magister en Ciencias Informática con mención
En Tecnologías de Información

GUÍA DE LABORATORIO

SISTEMAS OPERATIVOS

SEMESTRE V

COMPETENCIAS

Comprender el funcionamiento interno de los sistemas operativos y el manejo de procesos ligeros en el sistema
Utilizar, analizar y explicar con ejemplos el uso de Threads y la programación paralela

Laboratorio 6

Threads en C - PThreads

I OBJETIVOS

- El estudiante comprenderá y analizará el funcionamiento interno de los sistemas operativos desde la utilización de comandos hasta la programación de procesos.
- El estudiante deberá de probar, analizar y explicar el comportamiento de los Threads dentro de los procesos. Además de interpretar la funcionalidad que conlleva la programación paralela y su impacto en el Sistema Operativo.

II TEMAS A TRATAR

- Threads
- Lenguaje de programación C

III MARCO TEÓRICO

¿En qué se diferencia un proceso de un Thread?

Un proceso es cualquier programa en ejecución dentro del sistema operativo e independiente de cualquier otro proceso que se encuentre en ejecución. Un proceso tiene su propia zona de memoria y se ejecuta en simultaneo con otros procesos (Fig. 1). Un programa malintencionado no puede inferir con otros procesos en ejecución ni mucho menos a los del sistema operativo.

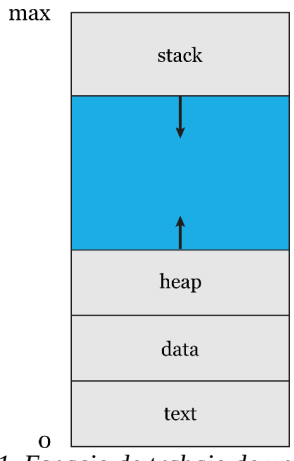

Figura 1. Espacio de trabajo de un proceso

Dentro de los procesos puede haber varios procesos ligeros (Threads). Lo que implica que un proceso puede estar ejecutando varias tareas en simultaneo. Los Threads dentro de un proceso comparten memoria, por lo tanto, si un Thread toca una variable, todos los demás Threads del mismo proceso verán el nuevo valor de la variable. De esta manera, es necesario contar con mecanismos de sincronización que aseguren el correcto funcionamiento de los Threads.

Por lo tanto, lanzar un nuevo proceso resulta en una actividad más costosa que lanzar un proceso ligero (Thread).

1. PThreads

PThreads hacen referencia al estándar POXIS, el cual provee un API para la creación y manipulación de Threads. A continuación, se muestra un ejemplo de la utilización de un Threads en ejecución dentro de un proceso.

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

void* funcion(void* p1);
int c = 0;

int main() {
	pthread_t hilo;
	pthread_attr_t attr;
	int error;

	pthread_attr_init(&attr);
	error = pthread_create(&hilo, &attr, funcion, NULL);

	if (error != 0) {
		perror("error");
		return(-1);
	}
	int i = 0;
	while (i < 300) {
		c++;
		printf("Padre: %d\n", c);
		i++;
	}
	pthread_join(hilo, NULL);
	return(0);
}

void* funcion(void* p11) {
	int i = 0;
	while (i < 300) {
		c--;
		printf("Hijo: %d\n", c);
		i++;
	}
	pthread_exit(0);
}
```

Las partes fundamentales de la creación de nuevos Threads de acuerdo a la librería PThread son:

- **pthread_t** es un puntero a un identificador de un nuevo Thread.

- **pthread_attr_t** son los atributos de creación del Thread. La función *pthread_attr_init* inicializa estos atributos con los parámetros por defecto de un proceso típico del sistema operativo donde se ejecutará un Thread nuevo.
- **pthread_create** se utiliza para crear e iniciar la ejecución de un nuevo Thread. Recibe como parámetros el identificador del Thread a trabajar, los atributos que obtendrá el Thread a ser ejecutado, la referencia a la función a desarrollar por el Thread, y por último, de ser necesario, parámetros de trabajo.
- **pthread_join** permitirá al proceso que invoco los Threads esperar por la finalización de un Thread especifico, pudiendo incluir un tiempo de espera fijo o ilimitado (el valor NULL)
- **pthread_exit** Al finalizar un Thread, notificará al proceso que lo creo que ha finalizado con su actividad, y se podrán reclamar los recursos asignados por el Sistema Operativo.

IV

ACTIVIDADES

1. A continuacion, para poder compilar un programa que maneja hilos, bastará con incluir el flag *-lpthread*. (Fig. 2) Ejemplo:

```
GCC nombre_de_archivo.c -lpthread -o ejecutable
```

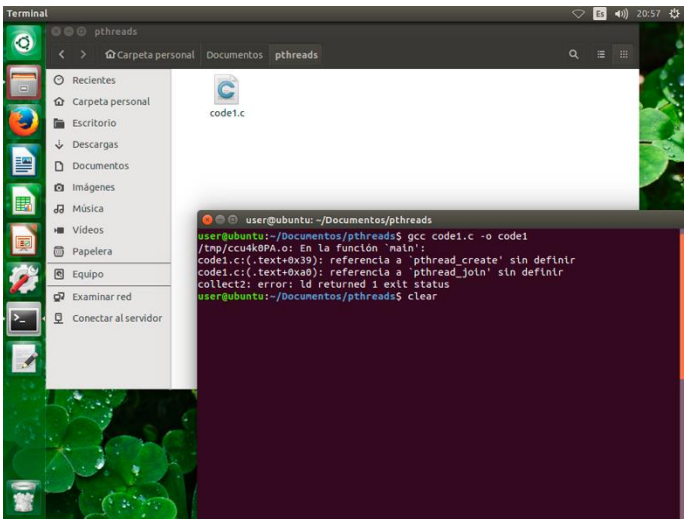

Figura 2. Error de compilación producido al omitir el flag -lpthread

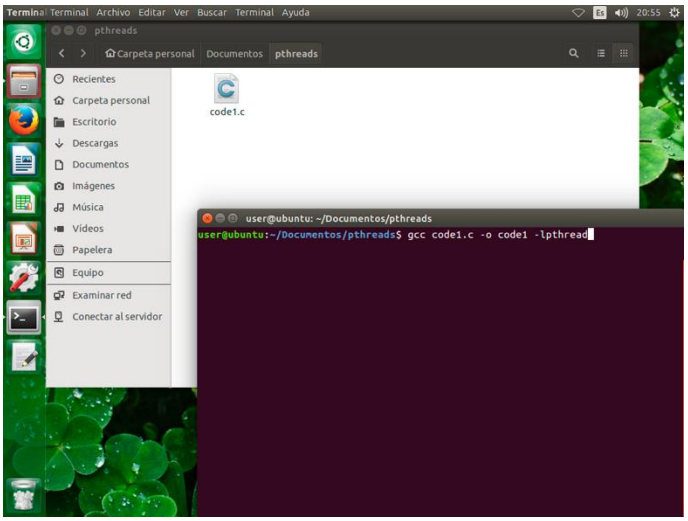

Figura 3. Sentencia de compilación correcta

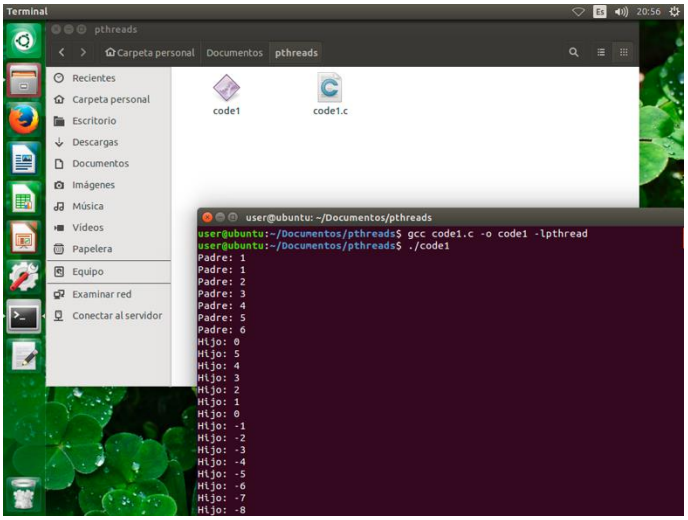

Figura 4. Ejemplo de ejecución de dos tareas (padre e hijo) en paralelo

2. Analice el código del ejemplo la sección PThreads. Describa que actividad realiza el código que se muestra. Detalle los comandos utilizados para poder ejecutar de forma correcta el código.
3. Explique a que se debe el orden de ejecución del proceso principal y del Thread creado, ¿Se ejecutan en paralelo o de forma secuencial?.
4. Del código de la sección PThreads, que ocurre si se remueve la sentencia *pthread_exit(0)*. Elabore un ejemplo de código que demuestre el comportamiento del código sin esta sentencia.
5. De igual manera, que involucra remover la sentencia pthread_join(hilo,NULL). ¿Existen cambios significativos en el comportamiento de los códigos? Elabore un ejemplo en código.
6. 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. Haga pruebas con un valor mayor de procesos y de hilos. ¿Qué conclusiones le merece los resultados observados?

```c
# 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) ;
   }
}
```

```c
# 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) );
}
```

7. Analice el código del siguiente programa. Describa que actividad realiza el código que se muestra.

   Agregue una función que sea invocada por el hilo principa para mostrar el contenido de la matriz antes y despues de su modificación.

```c
/*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");
}
```

V

EJERCICIOS PROPUESTOS

1. Modifique el programa *Ejercicio_Matriz* de tal forma que, un hilo se encargue de multiplicar cada fila de la matriz por un escalar cualquiera N.
2. Modifique el programa *Ejercicio_Matriz* de modo que a través de otro hilo se muestre el contenido de la matriz.
3. Elabore un programa con un mínimo de 4 Threads, que realice la suma de una lista de números (esta estructura contendrá más de 5000 elementos generados de forma aleatoria). Elabore un programa que realice el proceso de forma secuencial para validar los resultados obtenidos.
   Mida los tiempos requeridos por ambos programas (con varios hilos/un solo hilo).
   ¿A qué conclusiones puede usted llegar después de esta experiencia?

VI

CUESTIONARIO

1. ¿Qué diferencia existe entre crear un proceso con PThread y Fork?

2. ¿Por qué no es posible compartir una variable declarada dentro de la función *main( )* entre varios PThreads?

3. Si existen 2 PThreads que modifican una variable global (uno realiza la operación x += 1 y el otro x-=1 , repitiendo 100 veces cada operación) y se muestra cada operación en pantalla. ¿Se llega a obtener siempre el mismo resultado o este cambia en cada ejecución? Escriba un programa y realice varias pruebas que sustenten sus afirmaciones.

VII

ENTREGABLES

- Elaborar documento de sus actividades y los ejercicios realizados. Se debe incluir el comando utilizado, la descripción de la actividad que realiza, y una captura de pantalla (puede ser solo de la consola o terminal usado) de los resultados de las pruebas realizadas.
- En este documento incluir la respuesta al cuestionario y conclusiones.
- Subir al aula virtual el documento en formato PDF.