s

Instituto Politécnico Nacional
Escuela Superior de Computo

Sistemas operativos (2CM9)

Maestra: Ana Belem Juárez Méndez

Practica 3: Hilos

Alumno:

García González Aarón Antonio

Septiembre 12, 2019

Índice

Objetivo 3
Introducción 3
Desarrollo 8
Ejercicio 1 8
Ejercicio 2 9
Pregunta 1 11
Ejercicio 3 11
Conclusiones 14
Referencias 14

Objetivo

Utilizar los conocimientos de hilos en aplicaciones de sistemas operativos, con base a las funciones básicas vistas en clase y la definición de hilos

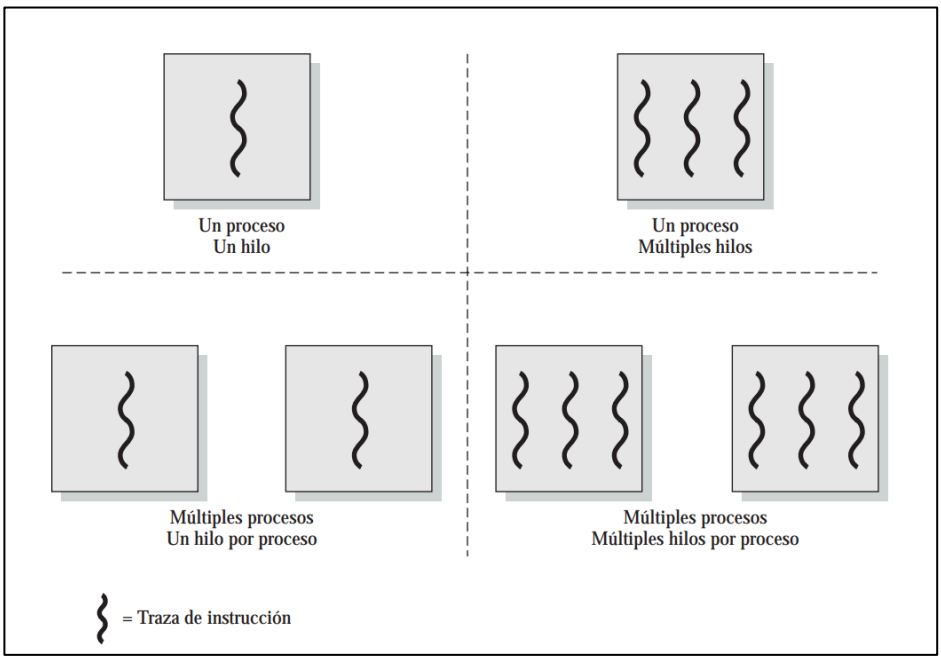
Introducción

¿Qué es un hilo?

Un hilo es una línea de ejecución de un proceso, proceso ligero o subproceso es una secuencia de tareas encadenadas muy pequeña que puede ser ejecutada por un sistema operativo.
Cuando se crea un hilo, el programador indica que código ejecuta. Los hilos de un mismo proceso comparten espacio de memoria, por lo tanto, dos hilos del mismo proceso pueden compartir estructuras de datos y variables. Los hilos de ejecución que comparten los mismos recursos, sumados a estos recursos, son en conjunto conocidos como un proceso. El hecho de que los hilos de ejecución de un mismo proceso compartan los recursos hace que cualquiera de estos hilos pueda modificar estos recursos. Cuando un hilo modifica un dato en la memoria, los otros hilos acceden a ese dato modificado inmediatamente.
Lo que es propio de cada hilo es el contador de programa, la pila de ejecución y el estado de la CPU (incluyendo el valor de los registros).
Los hilos nos permiten aprovechar de la existencia de mas de un procesador en el sistema, puesto que podemos asignar un hilo a cada uno de los procesadores que haya disponibles. La programación con hilos nos permite sacar partido de las arquitecturas multiprocesador que predominan en la actualidad.

Multihilo

Se refiere a la capacidad de un sistema operativo de dar soporte a múltiples hilos de ejecución en un solo proceso. En el enfoque tradicional de un solo hijo de ejecución por proceso, en el que no se identifica con el concepto de hilo, se conoce como estrategia monohilo.



Diferencias entre hilos y procesos

Los hilos se distinguen de los tradicionales procesos en que los procesos son –generalmente– independientes, llevan bastante información de estados, e interactúan solo a través de mecanismos de comunicación dados por el sistema. Por otra parte, muchos hilos generalmente comparten otros recursos de forma directa. En muchos de los sistemas operativos que dan facilidades a los hilos, es más rápido cambiar de un hilo a otro dentro del mismo proceso, que cambiar de un proceso a otro. Este fenómeno se debe a que los hilos comparten datos y espacios de direcciones, mientras que los procesos, al ser independientes, no lo hacen. Al cambiar de un proceso a otro el sistema operativo (mediante el dispatcher) genera lo que se conoce como overhead, que es tiempo desperdiciado por el procesador para realizar un cambio de contexto (context switch), en este caso pasar del estado de ejecución (running) al estado de espera (waiting) y colocar el nuevo proceso en ejecución. En los hilos, como pertenecen a un mismo proceso, al realizar un cambio de hilo el tiempo perdido es casi despreciable.

Funcionalidad de los hilos

Al igual que los procesos, los hilos poseen un estado de ejecución y pueden sincronizarse entre ellos para evitar problemas de compartición de recursos.

Los principales estados de los hilos son: Ejecución, Listo y Bloqueado. No tiene sentido asociar estados de suspensión de hilos ya que es un concepto de proceso. En todo caso, si un proceso está expulsado de la memoria principal (RAM), todos sus hilos deberán estarlo ya que todos comparten el espacio de direcciones del proceso.

- ⇒ Creación: Cuando se crea un proceso se crea un hilo para ese proceso. Luego, este hilo puede crear otros hilos dentro del mismo proceso, proporcionando un puntero de instrucción y los argumentos del nuevo hilo. El hilo tendrá su propio contexto y su propio espacio de la columna, y pasará al final de los Listos.
- ⇒ Bloqueo: Cuando un hilo necesita esperar por un suceso, se bloquea (salvando sus registros de usuario, contador de programa y punteros de pila). Ahora el procesador podrá pasar a ejecutar otro hilo que esté al principio de los Listos mientras el anterior permanece bloqueado.
- ⇒ Desbloqueo: Cuando el suceso por el que el hilo se bloqueó se produce, el mismo pasa a la final de los Listos.
- ⇒ Terminación: Cuando un hilo finaliza se liberan tanto su contexto como sus columnas.

Concurrencia

Numero maximo de flujos de ejecucion secuenciales -hilos- que podria estar ejecutando simultaneamente si dispusiera de un numero infinito de procesadores.

Paralelismo

Numero real de hilos que se estan ejecutando simultaneamente en un momento determinado momento, limitado por el numero de procesores.

POSIX.1c

Existen 8 tipos de datos en POSIX.1c:

Tipo de dato	Descripción
pthread_attr_t	Atributo de hilo
pthread_mutexattr_t	Atributo de mutex
pthread_condattr_t	Atributo de variable de condición
pthread_mutex_t	Mutex (bloqueo con exclusión mutua)
pthread_cond_t	Variable de condición
pthread_t	Hilo (identificador de hilo o ID)
pthread_once_t	Ejecución una sola vez
pthread_key_t	Clave sobre datos específicos de hilo

Gestión de hilos

Las funciones básicas para la gestión de hilos son las siguientes:

Función	Descripción
pthread_create	Crea un hilo
pthread_equal	Verifica la igualdad de dos identificadores de hilo
pthread_exit	Termina el hilo que realiza la llamada
pthread_join	Espera por el termino de un hilo especifico
pthread_self	Regresa el ID del hilo que realiza la llamada
pthread_detach	Configura la liberación de recursos cuando termina (hilo independiente)
pthread_getschedparam	Obtiene la política de planificación y parámetros de un hilo específico
pthread_setschedparam	Establece la política de planificación y parámetros de un hilo específico
pthread_kill	Envía una señal de determinada a un hilo específico
pthread_cancel	Permite a un hilo cancelar otro hilo del mismo proceso

Creando un hilo (pthread_create)

Esta función automáticamente pone en ejecución el hilo que crea. La sintaxis es la siguiente:

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

Donde:

- ⇒ El parámetro thread apunta al ID del hilo recientemente creado.
- ⇒ El parámetro attr representa un objeto atributo que encapsula los atributos de un hilo. Si attr es NULL, el hilo nuevo tendrá los atributos asignados por defecto.
- ⇒ El tercer parámetro, start_routine, es el nombre de la función que es invocada por el hilo cuando comienza su ejecución.
- ⇒ El parámetro arg especifica el parámetro que recibe la función start_routine

Si pthread_create se ejecuta satisfactoriamente retorna 0. Si la rutina no se ejecuta satisfactoriamente retorna un código de error diferente de cero. A continuación se muestra una tabla con los errores que puede generar una invocación a pthread.

Error	Causa
EAGAIN	El sistema no posee los recursos necesarios para crear el nuevo hilo, o se excede el límite total del número de hilos permitidos por el sistema.
EINVAL	El parámetro attr es inválido

EPERM	No se tienen los permisos suficientes para cambiar la política de planificación o cualquier otro parámetro especificado en attr

Sobre el objeto atributo

Los atributos se almacenan en un objeto atributo de tipo pthread_attr_t.

Para la creación y el inicio de un objeto atributo (utilizado en la creación de un proceso ligero) se utiliza la siguiente función:

int pthread_attr_init(pthread_attr_t *attr);

Para la destrucción del objeto de tipo atributo se utiliza:

int pthread_attr_destroy(pthread_attr_t *attr);

El servicio para el establecimiento (set) del atributo correspondiente al estado de terminación es:

int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);

El valor del argumento detachstate puede ser:

- ⇒ PTHREAD_CREATE_DETACHED: El proceso ligero que se cree con este estado de terminación se considerará independiente y liberará sus recursos cuando finalice su ejecución.
- ⇒ PTHREAD_CREATE_JOINABLE: El proceso ligero que se cree con este estado de terminación se considerará como no independiente y no liberará sus recursos cuando finalice su ejecución. En este caso, es necesario que otro proceso espere por su finalización utilizando pthread_join.

El servicio para la obtención (get) del atributo correspondiente al estado de terminación es el siguiente:

int pthread_attr_getdetachstate(pthread_attr_t *attr, int *detachstate);

Cada hilo tiene una pila cuyo tamaño se puede establecer en el momento de la creación, dándole al atributo que se va a utilizar en la creación el valor adecuado. Esto se hace mediante el servicio:

int pthread_attr_setstacksize (pthread_attr_t *attr, int stacksize);

El servicio para obtener el tamaño de la pila es el siguiente:

int pthread_attr_getstacksize (pthread_attr_t *attr, int *stacksize);

Comparando IDs (pthread_equal)

El ID de un hilo es dependiente del SO y puede ser una estructura, por esta razón se deberá utilizar una función para comparar la igualdad de los IDs de los hilos

int pthread_equal(pthread_t t1, pthread_t t2);

Donde t1 y t2 son IDs de los hilos que van a ser comparados.

Si t1 es igual a t2, pthread_equal retorna un valor diferente de cero. Si los IDs de los hilos son diferentes, pthread_equal retorna 0.

Salir (pthread_exit)

Un hilo puede terminar de tres maneras sin terminar el proceso: retornando de su rutina de inicio, cancelado por otro hilo del mismo proceso, o llamando pthread_exit.

void pthread_exit(void *value_ptr);

El valor de value_ptr debe apuntar a datos que existan inclusive después que el hilo termine.

Unir (pthread_join)

La función pthread_join suspende la ejecución del hilo que la invoca hasta que el hilo objetivo, especificado por el primer parámetro, termine.

int pthread_join(pthread_t, void **value_ptr);

El parámetro valor_ptr es un apuntador al valor de retorno que el hilo objetivo pasa a la función pthread_exit o a return. Si value_ptr es NULL, el hilo que invoca join no recibe el valor de retorno del hilo objetivo.

Si la función termina exitosamente retorna 0. Si no termina exitosamente retorna un valor de error diferente de cero.

Error	Causa
EINVAL	El hilo no corresponde a un hilo disponible para la unión
ESRCH	No existe un hilo identificado por ID

Referencias a hilos por su id (pthread_self)

Cualquier hilo puede conocer su ID invocando a:

pthread_t pthread_self(void);

Esta función retorna el ID del hilo que la invoca.

Separar (pthread_detach)

Cuando un hilo termina no libera sus recursos al menos que el hilo este separado. La función pthread_detach modifica las opciones internas del hilo para especificar que el espacio utilizado para el almacenamiento del mismo puede ser reclamado cuando culmine. Los hilos ya separados no reportan su estado cuando culminan.

int pthread_detach(pthread_t thread);

La función pthread_detach recibe un solo parámetro, thread, que es el ID del hilo a ser separado.

Si pthread_detach culmina exitosamente retorna uno. Si no culmina exitosamente retorna un valor diferente de cero. La siguiente tabla muestra los códigos de error de pthread_detach.

Error	Causa
EINVAL	El hilo no corresponde a un hilo disponible para la unión
ESRCH	No existe un hilo identificado por ID

Cancelar (pthread_cancel)

Un hilo invoca pthread_cancel para solicitar que otro hilo sea cancelado. El resultado de la invocación es determinado por el tipo del hilo objetivo y su estado de cancelación.

int pthread_cancel(pthread_t thread);

El parámetro de pthread_cancel es el ID del hilo objetivo a ser cancelado.

Si pthread_cancel se ejecuta exitosamente retorna 0. Sino retorna un valor diferente de cero.

Desarrollo

Ejercicio 1

Realizar un programa con una variable entera global con un valor inicial de cero. Crear un hilo que incremente la variable global en A unidades. Crear otro hilo que la disminuya en B unidades. Al final el hilo principal(main) imprimirá el valor de la variable global.

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

int VARIABLE = 0;
void *aumentar(void *id);
void *decrementar(void *id);

int main(int argc, char const *argv[]){
    int error;
    int *valor;
    int id[2] = {1,2};
    pthread_t hilos[2];

    for(int i = 0; i<2; i++){
        if(i==0){ // Se trata del primer hijo
            error = pthread_create(&hilos[i],NULL,aumentar,&id[i]);
        }else{ // Se trata del segundo hilo
            error = pthread_create(&hilos[i],NULL,decrementar,&id[i]);
        }
        if(error){ // Error al crear hilo
            printf("Error al crear hilo: %d\n", i+1);
            exit(-1);
        }
    }

    for(int i = 0; i<2; i++){
        pthread_join(hilos[i], (void **)&valor);
    }

    printf("El valor final de la variable global es: %d\n", VARIABLE);
    return 0;
}

// Funcion que aumenta en X cantidad a la variable global
void *aumentar(void *id){
    int x;
    printf("\n[H%d]Teclee un numero a sumar: ", *(int*)id);
    scanf("%d",&x);
```

```
    VARIABLE += x;
    pthread_exit(id);
}
// Funcion que decrementa en X cantidad a la variable global
void *decrementar(void *id){
    int x;
    printf("\n[H%d]Teclee un numero a restar: ", *(int*)id);
    scanf("%d",&x);
    VARIABLE -= x;
    pthread_exit(id);
}
```

```
[MacBook-Pro-de-Aaron:p3 aarongarcia$ gcc e1.c
[MacBook-Pro-de-Aaron:p3 aarongarcia$ ./a.out

 [H1]Teclee un numero a sumar:
 5
 [H2]Teclee un numero a restar: 9
 El valor final de la variable global es: -4
```

Ejercicio 2

Crear un programa que realice dos hilos, cada uno debe mostrar algún mensaje en pantalla, el mensaje debe ser mostrado carácter por carácter. El hilo principal(main) debe esperar a que ambos hilos terminen y debe mostrar la palabra 'FIN'.

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

void *messageOne(void *id);
void *messageTwo(void *id);

int main(int argc, char const *argv[]){
    int error;
    int *valor;
    int id[2] = {1,2};
    pthread_t hilos[2];

    for(int i = 0; i<2; i++){
        if(i==0){ // Se trata del primer hijo
```

```
            error = pthread_create(&hilos[i],NULL,messageOne,&id[i]);
        }else{ // Se trata del segundo hilo
            error = pthread_create(&hilos[i],NULL,messageTwo,&id[i]);
        }
        if(error){ // Error al crear hilo
            printf("Error al crear hilo: %d\n", i+1);
            exit(-1);
        }
    }
    sleep(1);
    for(int i = 0; i<2; i++){
       pthread_join(hilos[i], (void **)&valor);
    }

    printf("\nFIN\n");

    return 0;
}

// Funcion que imprime el mensaje 1, caracter a catacter
void *messageOne(void *id){
    char pal[] = "aaron";
    printf("\n[%d]\t", *(int*)id);
    for(int i=0; i<strlen(pal); i++){
        printf("%c\t", pal[i]);
    }

    pthread_exit(id);
}

// Funcion que imprime el mensaje 2, caracter a catacter
void *messageTwo(void *id){
    char pal[] = "miguel";
    printf("\n[%d]\t", *(int*)id);
    for(int i=0; i<strlen(pal); i++){
        printf("%c\t", pal[i]);
    }

    pthread_exit(id);
}
```

```
[MacBook-Pro-de-Aaron:p3 aarongarcia$ gcc e2.c
[MacBook-Pro-de-Aaron:p3 aarongarcia$ ./a.out

[1]     a       a       r       o       n
[2]     m       i       g       u       e       l
FIN
```

Pregunta 1

¿Qué sucede si el hilo principal termina un instante después de haber creado los hilos (modifique el programa del Ejercicio 2 para hacer la prueba)?

```
    for(int i = 0; i<2; i++){
        if(i==0){ // Se trata del primer hijo
            error = pthread_create(&hilos[i],NULL,messageOne,&id[i]);
        }else{ // Se trata del segundo hilo
            error = pthread_create(&hilos[i],NULL,messageTwo,&id[i]);
        }
        if(error){ // Error al crear hilo
            printf("Error al crear hilo: %d\n", i+1);
            exit(-1);
        }
    }
    exit(0);
    sleep(1);
    for(int i = 0; i<2; i++){
       pthread_join(hilos[i], (void **)&valor);
    }
```

```
[MacBook-Pro-de-Aaron:p3 aarongarcia$ gcc e2-2.c
[MacBook-Pro-de-Aaron:p3 aarongarcia$ ./a.out
```

Ejercicio 3

NOTA Si el segundo argumento de la función pthread_create es NULL se utilizan los atributos predeterminados, lo cual implica ser un hilo dependiente. Si un hilo es dependiente no se liberarán sus recursos a menos que otro hilo espere su finalización, esto mediante pthread_join. Se pueden crear hilos sin necesidad de esperar a que dichos hilos terminen, de ser así deben tener el atributo de independientes (PTHREAD_CREATE_DETACHED). La función pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate) permite establecer el estado de terminación de un hilo, si el segundo argumento es PTHREAD_CREATE_DETACHED, el hilo se considera como independiente.

El siguiente programa muestra cómo se crean 10 hilos independientes. Compílalo y ejecútalo.

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

#define NHILOS 10

void funcion_hilos(void){
    printf("\nHilo %d\n", (int)pthread_self()); // pthread_self devuelve el identificador del hilos
    pthread_exit(0);
}

int main(int argc, char const *argv[]){
    int i;
    pthread_attr_t atributos;
    pthread_t hilos[NHILOS];

    //  Se incializan los atributos como independientes
    pthread_attr_init(&atributos);
    pthread_attr_setdetachstate(&atributos, PTHREAD_CREATE_DETACHED);

    for(i=0; i<NHILOS; i++){
        pthread_create(&hilos[i], &atributos, (void*)funcion_hilos,NULL);
    }
    /*El hilo principal se suspende 4 segundos, para esperar a que los hijos terminen, de no ser as
i
        al terminar el hilo principal, todos los hilos terminaran*/
    sleep(1);

    return 0;
}
```

```
[MacBook-Pro-de-Aaron:p3 aarongarcia$ gcc e3.c
[MacBook-Pro-de-Aaron:p3 aarongarcia$ ./a.out

Hilo 236003328

Hilo 239759360

Hilo 237613056

Hilo 236539904

Hilo 238686208

Hilo 238149632

Hilo 239222784

Hilo 240295936

Hilo 237076480

Hilo 240832512
MacBook-Pro-de-Aaron:p3 aarongarcia$
```

Conclusiones

Tuve una pequeña confusion de concepto entre hilos y procesos, con el segundo ejercicio queria resolverlo con la logica de un proceso, pero de haber sido asi, no tendria sentido ya que los hilos son para correr procesos al mismo tiempo, jamas habia trabajado con hilos en lenguaje C, unicamente en Java, una de las aplicaciones mas importantes que tienen los hilos es en los mensajes de texto, tambien si se requiere que la funcion que va a ejecutar determiando hilo, recibe mas de un parametro, es conveniente usar una estructura de datos con los argumentos de dicha funcion y finalmente si un hilo principal (main) termina antes de que los hilos, entonces estos se terminan con este ultimo.

Referencias

[1] Yúbal FM. (2019, 17 julio). Qué es una dirección IP y cómo puedes saber la tuya. Recuperado 9 septiembre, 2019, de https://www.xataka.com/basics/que-es-una-direccion-ip-y-como-puedes-saber-la-tuya

[2] [C] Leer cadena sin límites - El Rincón del C. (s.f.). Recuperado 12 septiembre, 2019, de https://elrincondelc.com/foros/viewtopic.php?t=19774

[3] [why use usleep and not sleep]. (2017, 17 marzo). Recuperado 12 septiembre, 2019, de https://stackoverflow.com/questions/42861913/why-use-usleep-and-not-sleep

[4] pthread_self() in C with Example - GeeksforGeeks. (2017, 1 septiembre). Recuperado 13 septiembre, 2019, de https://www.geeksforgeeks.org/pthread_self-c-example/

[5] UC3M. (s.f.). 11.2. Hilos. Recuperado 12 septiembre, 2019, de http://www.it.uc3m.es/pbasanta/asng/course_notes/c_threads_functions_es.html

[6] Colaboradores de Wikipedia. (2019, 7 septiembre). secuencia de tareas encadenadas muy pequeña que puede ser ejecutada por un sistema operativo. Recuperado 12 septiembre, 2019, de https://es.wikipedia.org/wiki/Hilo_(inform%C3%A1tica)