Práctica

Comparación entre creación de procesos y creación de hilos

Cuestionario previo

1. ¿Qué es un proceso?
2. ¿Cuáles son los estados de un proceso?
3. ¿Qué es un hilo?
4. ¿Cuáles son los estados de un hilo?
5. ¿Qué ventajas aporta un hilo respecto a un proceso?
6. ¿Qué es Posix?

Antecedentes

1. Programación en C
2. Manejar conceptos referentes a la administración de procesos
3. Identificar los conceptos de proceso y multitarea

Objetivo

Aprender conceptos básicos sobre multi hilos, aplicar estos conceptos en la realización de programas sencillos que involucren las llamadas a hilos de POSIX y conocer los mecanismos de sincronización de POSIX.

Introducción

Proceso

Un proceso es un programa en ejecución. Cada proceso consta de bloques de código y de datos cargados desde un archivo ejecutable. También es propietario de otros recursos que se crean durante la vida de dicho proceso y se destruye cuando finaliza. Un proceso tiene:

- Su propio espacio de direcciones
- Memoria
- Variables
- Archivos abiertos
- Procesos hijos
- Contador de programa, registros, pila, señales, semáforos, etc.

Los procesos son creados y destruidos por el sistema operativo, así como también este se debe hacer cargo de la comunicación entre procesos, pero lo hace a petición de otros procesos. El mecanismo por el cual un proceso crea otro proceso se denomina bifurcación (*fork*). Los nuevos procesos pueden ser independientes y no compartir el espacio de memoria con el proceso que los ha creado o ser creados en el mismo espacio de memoria.

Básicamente un proceso puede tener tres estados, puede estar en ejecución (está utilizando la CPU), preparado (está detenido temporalmente para que se ejecute otro proceso) o bloqueado (no se puede ejecutar debido a que ocurrió algún evento al que hay que responder adecuadamente), para estos tres estados se pueden tener cuatro posibles transiciones:

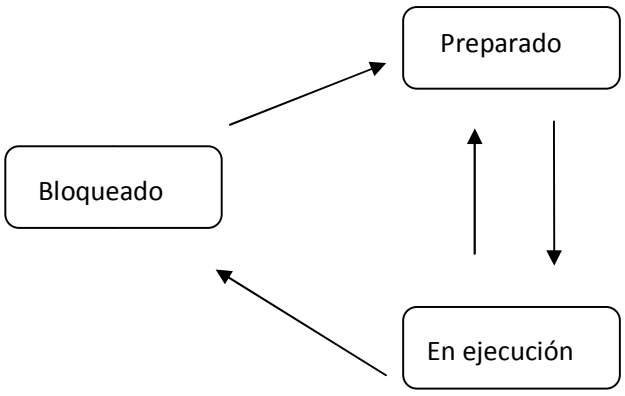

Figura 1. Estados de un proceso

Hilo

Un hilo (thread – hebra o subproceso) es la unidad de ejecución de un proceso y está asociado con una secuencia de instrucciones, un conjunto de registros y una pila, es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo, cuando se crea un proceso, el sistema operativo crea su primer hilo el cual puede a su vez, crear hilos adicionales.

Cada hilo se ejecuta en forma estrictamente secuencial y tiene su propia pila, el estado de la CPU y su propio contador de programa. En cambio, comparten el mismo espacio de direcciones, lo que significa compartir también las mismas variables globales, el mismo conjunto de archivos abiertos, señales, semáforos, etc.

Al igual que los procesos con un solo hilo de control, los hilos pueden encontrarse en uno de los siguientes estados.

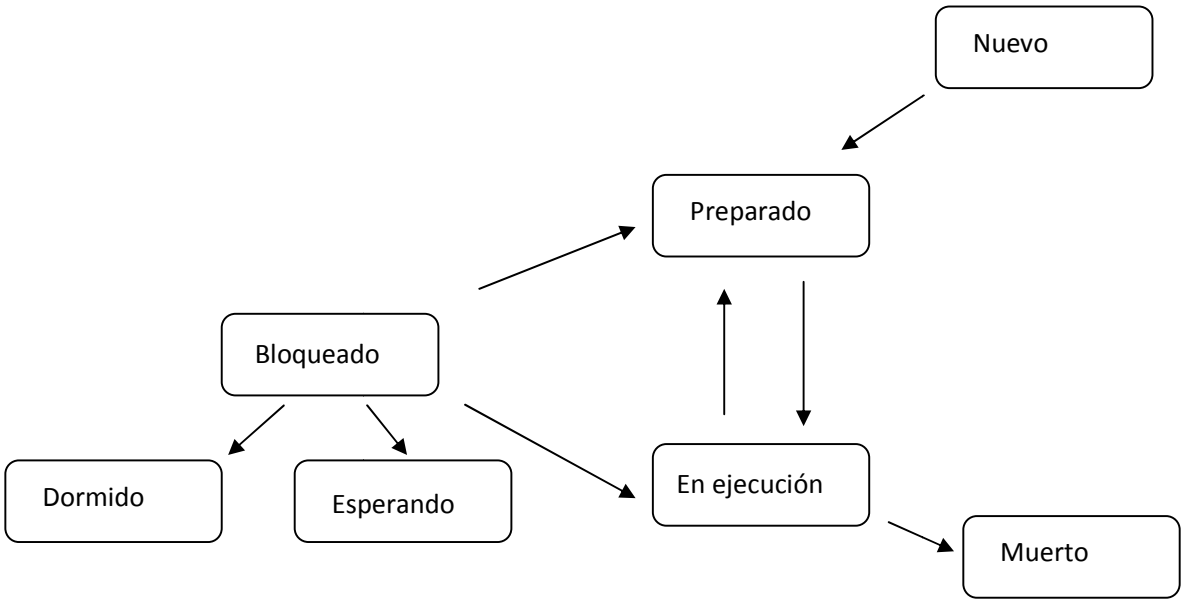

Figura 2. Estados de un hilo

Nuevo. El hilo ha sido creado pero aún no ha sido activado. Cuando se active pasará al estado activado.

Preparado. El hilo esta activo y está a la espera de que le sea asignada la CPU

En ejecución. El hilo esta activo y le ha sido asignada la CPU

Bloqueado. El hilo espera a que otro elimine el bloqueo. Un hilo bloqueado puede estar:

- Dormido. El hilo está bloqueado durante una cantidad de tiempo determinada, después de la cual despertará y pasará al estado preparado.
- Esperando. El hilo está esperando a que ocurra algún evento, cuando ocurra pasará al estado preparado

Muerto. El hilo ha finalizado pero todavía no ha sido recogido por su padre. Los hilos muertos no pueden alcanzar ningún otro estado.

Los hilos comparten un espacio de memoria, el código y los recursos, por lo que el lanzamiento y la ejecución de un hilo es mucho más económica que el lanzamiento y la ejecución de un proceso; muchos problemas pueden ser resueltos mejor con múltiples hilos. Un hilo es básicamente una tarea que puede ser ejecutada en paralelo con otra tarea. Un ejemplo de la utilización de hilos es tener un hilo atento a la interfaz gráfica (iconos, botones, ventanas), mientras otro hilo hace una larga operación internamente. De esta manera el programa responde de manera más ágil a la interacción con el usuario.

Si bien los hilos son generados a partir de la creación de un proceso, podemos decir que un proceso es un hilo de ejecución, conocido como Monohilo. Pero las ventajas de los hilos se dan cuando hablamos de Multihilos, que es cuando un proceso tiene múltiples hilos de ejecución los cuales realizan actividades distintas, que pueden o no ser cooperativas entre sí. Los beneficios de los hilos se derivan de las implicaciones de rendimiento.

1. Se tarda mucho menos tiempo en crear un hilo nuevo en un proceso existente que en crear un proceso.
2. Se tarda mucho menos en terminar un hilo que un proceso,
3. Se tarda mucho menos tiempo en cambiar entre dos hilos de un mismo proceso
4. Los hilos aumentan la eficiencia de la comunicación entre programas en ejecución.

Los sistemas operativos generalmente implementan hilos de dos maneras:

- *Multihilo apropiativo:* permite al sistema operativo determinar cuándo debe haber un cambio de contexto. La desventaja de esto es que el sistema puede hacer un

cambio de contexto en un momento inadecuado, causando un fenómeno conocido como inversión de prioridades y otros problemas.

- *Multihilo cooperativo:* depende del mismo hilo abandonar el control cuando llega a un punto de detención, lo cual puede traer problemas cuando el hilo espera la disponibilidad de un recurso.

El soporte de *hardware* para multihilo se encuentra disponible desde hace relativamente poco tiempo. Esta característica fue introducida por Intel en el Pentium 4, bajo el nombre de HyperThreading.

POSIX

POSIX significa **P**ortable **O**perating **S**ystem **I**nterface (for **U**nix). Es un estándar orientado a facilitar la creación de aplicaciones confiables y portables. La mayoría de las versiones populares de UNIX (Linux, Mac OS X) cumplen este estándar en gran medida. Incluye creación y control de procesos, señales, excepciones de punto flotante, excepciones por violación de segmento, excepciones por instrucción ilegal, errores del bus, temporizadores, operaciones de archivos y directorios (sobre cualquier fs montado), tuberías (*Pipes*), biblioteca C (Standard C), instrucciones de entrada/salida y de control de dispositivo (ioctl).

Para Microsoft Windows *Cygwin* ofrece un desarrollo en gran parte compatible con POSIX y un entorno de ejecución y *Windows Services for UNIX* de Microsoft permite una plena compatibilidad POSIX para ciertos productos de Microsoft Windows, además se cuenta con el *Microsoft POSIX subsystem.*

La biblioteca para el manejo de hilos en POSIX es pthread.

Desarrollo

Procesos

El siguiente código ejemplifica la creación de procesos.

Cuando se llama la función fork, esta genera un duplicado del proceso actual. El duplicado comparte los valores actuales de todas las variables, archivos y otras estructuras de datos. La llamada a fork retorna al proceso padre el identificador del proceso hijo y retorna un cero al proceso hijo. Las siglas PID, del inglés Process IDentifier, se usan para referirse al identificador de proceso.

```
/*================================================================
===
  Código en C para fork() Creación de procesos

================================================================
==*/
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>

#define NFORKS 50000
```

Esta function deberá tener el código el cual deseamos que realicen los procesos hijos.

```
void funcion() {
  int i;
  i= 0;
}
```

La variable pid indicara el identificador del proceso y la variable estado indicará el estado del proceso.

```
main() {
  int pid, j, estado;


Se crearán 50000 procesos hijos

for (j=0; j<NFORKS; j++) {
```

Guardamos en la variable “pid” el resultado de fork(). Cuando se llama la función fork, esta genera un duplicado del proceso actual. Si es cero, resulta que estamos en el proceso

hijo, por lo que haremos lo que tenga que hacer el hijo. Si es distinto de cero, estamos dentro del proceso padre, por lo tanto todo el código que vaya en la parte "else" de esa condicional sólo se ejecutará en el proceso padre

Si la función fork retorna un valor menor a cero, existe un error y debe manejarse.

```
      /*** Manejo de error ***/
    if ((pid = fork()) < 0 ) {
      printf ("El proceso cuyo identificador es= %d fallo\n", pid);
      exit(0);
    }
```

La función getpid(), obtiene el identificador del proceso hijo y la función getppid() obtiene el identificador del proceso padre.

```
/*** Este es el hijo del proceso ***/
    else if (pid ==0) {
      printf("Soy el hijo (%d, hijo de %d)\n", getpid(), getppid());
        funcion();
      exit(0);
    }
```

Con la función waitpid()aseguramos que el padre va a esperar a sus dos hijos antes de continuar.

```
      /*** Este es el estado del proceso  ***/
    else {
      waitpid(pid, &status, 0);
    }
  }
}
```

El código completo se muestra a continuación:

```
/*=========================================================================
===
  Código en C para fork() Creación de procesos

=========================================================================
==*/
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>

#define NFORKS 50000

void funcion() {
  int i;
  i= 0;
```

```
}

main() {
  int pid, j, estado;

  for (j=0; j<NFORKS; j++) {

      /*** Manejo de error ***/
    if ((pid = fork()) < 0 ) {
      printf ("El proceso cuyo identificador es= %d fallo\n", pid);
      exit(0);
    }

      /*** Este es el hijo del proceso ***/
    else if (pid ==0) {
       printf("Soy el hijo (%d, hijo de %d)\n", getpid(), getppid());
       funcion();
      exit(0);
    }

      /*** this is the parent of the fork ***/
    else {
      waitpid(pid, &estado, 0);
    }
  }
}
```

Hilos

Un paquete de manejo de hilos generalmente incluye funciones para: crear y destruir un hilo, itineración, forzar exclusión mutua y espera condicionada. Todas las funciones de hilos del POSIX comienzan con pthread. Entre ellas están:

Función POSIX	**Descripción**
pthread_equal	verifica igualdad de dos identificados de hilos
pthread_self	retorna ID de propio hilo (análogo a getpid)
pthread_create	crea un hilo (análogo a fork)
pthread_exit	termina el hilo sin terminar el proceso (análogo a exit)
pthread_join	espera por el término de un hilo (análogo a waitpid)
pthread_cancel	Termina otro hilo (análogo a abort)
pthread_detach	Configura liberación de recursos cuando termina
pthread_kill	envía una señal a un hilo

Para hacer uso de estas funciones se debe incluir <pthread.h>, el ligado debe incluir -l thread, como en cc -o ejecutable -l thread fuente.c

Los procesos normalmente corren como un hilo único. La creación de un nuevo hilo se logra vía pthread_create.

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

La función debe retornar un * void, el cual es interpretado como el estatus de término por pthread_join

```
/*============================================================================
====
  Código en C para pthread_create() Creación de hilos

============================================================================
==*/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
```

Se crearán 50000 hilos

```
#define NTHREADS 50000


void *funcion(void *null) {
  int i;
  i=0;
```

Termina el hilo sin terminar el proceso

```
  pthread_exit(NULL);
}

main() {
  int rc, i, j, detachstate;
  pthread_t tid;
  pthread_attr_t attr;
```

Se crean los atributos de los hilos por defecto

```
  pthread_attr_init(&attr);
```

Permite cambiar la característica del atributo con **PTHREAD_CREATE_JOINABLE,** podemos esperar al hilo

```
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
```

Se crean los hilos, la función debe retornar un * void, el cual es interpretado como el estatus de término por pthread_join

tid: salida, apuntador a id del hilo.
attr: entrada, para definir atributos del hilo, null para default.
funcion: entrada, función a correr por el hilo.
arg: entrada, argumento de la función del hilo.

```
 for (j=0; j<NTHREADS; j++) {
    rc = pthread_create(&tid, &attr, funcion, NULL);
```

Retorna ID de propio hilo

```
printf("Soy el hilo %d",  pthread_self());
    if (rc) {
      printf("ERROR; el valor que retorna pthread_create() es %d\n", rc);
      exit(-1);
    }
```

Espera por el término de un hilo

```
      /* Esperando por el hilo */
    rc = pthread_join(tid, NULL);
    if (rc) {
      printf("ERROR; el valor que retorna pthread_join() es %d\n", rc);
      exit(-1);
    }
  }
```

Destruye los atributos

```
  pthread_attr_destroy(&attr);
```

Termina el hilo sin terminar el proceso

```
  pthread_exit(NULL);

}
```

El código completo se muestra a continuación

```
/*=========================================================================
====
  Código en C para pthread_create() Creación de hilos

=========================================================================
==*/
```

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NTHREADS 50000

void *funcion(void *null) {
  int i;
  i=0;
  pthread_exit(NULL);
}

main() {
  int rc, i, j, detachstate;
  pthread_t tid;
  pthread_attr_t attr;

        pthread_attr_init(&attr);
        pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

for (j=0; j<NTHREADS; j++) {
    rc = pthread_create(&tid, &attr, funcion, NULL);
        printf("Soy el hilo %d",  pthread_self());
    if (rc) {
      printf("ERROR; el valor que retorna pthread_create() es %d\n", rc);
      exit(-1);
    }
/* Esperando por el hilo */
    rc = pthread_join(tid, NULL);
    if (rc) {
      printf("ERROR; el valor que retorna pthread_join() es %d\n", rc);
      exit(-1);
    }
  }
```

Conclusiones

El resultado esperado de esta práctica es que el alumno obtenga las herramientas necesarias para introducirse en la programación de procesos, hilos y multiprocesos, además de forjarse un criterio para decidir qué tipos de problemas pueden resolverse con procesos y cuales con hilos.

Bibliografía

David R. Butenhof
Programming with POSIX
Addison – Wesley Professional Computing Series
2006

Francisco Javier Ceballos
Java 2, 2a Edición
Alfa Omega
2005

Referencias

http://wwwdi.ujaen.es/~lina/TemasSO/DEFINICIONYCONTROLDEPROCESO/1y2Queesunproceso.EstadoyTransiciones.htm

http://sourceware.org/pthreads-win32/manual/index.html

http://www.gnu.org/software/pth/pth-manual.html

http://blog.txipinet.com/2006/11/03/46-curso-de-programacion-en-c-para-gnu-linux-v/

http://nereida.deioc.ull.es/~pp2/perlexamples/node66.html

http://www.mitecnologico.com/Main/AtributosDeHilos

http://www.infor.uva.es/~benja/creacion-procesos-hilos.html