Sistemas Concurrentes y Distribuidos - Portafolio de prácticas

29 de septiembre de 2016

Parte I

Seminario 1. Programación multihebra y sincronización con semáforos.

Hebras: introducción.

Una hebra es un flujo de control en el texto (zona con la secuencia de instrucciones a ejecutar) del proceso al que pertenece (que puede ser común a varias hebras).

Los ejemplos utilizan hebras de POSIX, más concretamente, utilizan *hebras del kernel de linux*, esto significa que hebras distintas de un mismo proceso pueden ejecutarse en procesadores distintos (lo cual hace que sean una herramienta ideal para aprovechar el potencial de los sistemas multiprocesador).

Creación de hebras.

Utilizamos la función *pthread_create:*

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

La forma más simple de crear hebras:

```
#include <pthread.h>

void *proc1( void *arg){
        ....
}

void *proc2( void *arg){
        ...
}

int main(){

pthread_t hebra1, hebra2 ;
pthread_create(&hebra1, NULL, proc1, NULL) ;
pthread_create(&hebra2, NULL, proc2, NULL) ;
}

/* Si usamos memoria dinámica sería algo como:

pthread_t *hebra1 ;
pthread_t *hebra2 ;
hebra1 = new pthread_t ;
hebra2 = new pthread_t ;
pthread_create(hebra1, NULL, proc1, NULL) ;
pthread_create(hebra2, NULL, proc2, NULL) ; */
```

Finalización de hebras

- Cuando una hebra encuentra un return en la función que está ejecutando.
- Cuando termina la función que está ejecutando.
- Si el proceso llama explícitamente a pthread_exit durante la ejecución.
- Si otra hebra (B) llama a pthread_cancel(A), B *mata* a A.
- Si una hebra del programa llama a exit, todas terminan.
- Si termina la hebra principal sin haber ejecutado pthread_exit, todas terminan.

Esto último nos indica que si la hebra principal tiene como única función llamar al resto, debemos acabarla con un *pthread_exit.*

pthread_exit es una función que recibe como argumento un puntero void que será el que recibirá la hebra que espera (vía join) a que finalice la hebra que finaliza).

```
pthread_exit(NULL) ;
```

es la forma típica de acabar con una hebra.
[EJEMPLOS1/ejemplo1.cpp]

Operación de unión (join)

mediante la función pthread_join una hebra A espera a que finalice otra hebra B.

tiene dos parámetros: el primero, la hebra objetivo (a la que se esperará) y el segundo, un puntero a la variable que recibirá el dato de tipo void* enviado por la hebra objetivo al finalizar. Devuelve un entero que será 0 si no ha habido ningún error.

No hay que calentarse la cabeza con los parámetros. Si queremos, por ejemplo, esperar a que finalicen dos hebras antes de continuar con la hebra principal podemos escribir:

```
int main(){
        .....

        pthread_join(hebra1, NULL) ;
        pthread_join(hebra2, NULL) ;

        .....

}
```

[EJEMPLOS1/ejemplo2.cpp]

Hebras idénticas

Es el caso en que varias hebras distintas ejecutan el mismo algoritmo *con datos de entrada distintos.*

[EJEMPLOS1/ejemplo3.cpp] Muestra un ejemplo de como pasar y recibir parámetros de distintas hebras idénticas. (Algo complejo y poco práctico, después veremos una forma más simple de hacer lo mismo).

Ejemplo de hebras: cálculo numérico de integrales

Calculamos de forma aproximada el valor de una integral $I = \int_0^1 f(x)dx$ entre los puntos 0 y 1, evaluando la función f en un conjunto de m puntos uniformemente espaciados en el intervalo [0,1] y aproximando I como:

$I = \frac{1}{m} \sum_{i=0}^{m-1} f(x_i)$ donde $x_i = \frac{1+1/2}{m}$

A modo de ejemplo usaremos la función f cuya integral entre 0 y 1 es el número π

$\pi = \int_0^1 \frac{4}{1+x^2} dx$

Lo haremos de forma que:

- Cada hebra evalua f en m/n puntos (con n = número de hebras)
- Cada hebra calcula la suma parcial de los valores de f (de forma independiente)
- La hebra principal recoge las sumas parciales y calcula la suma total
- En un entorno con k procesadores o núcleos el cálculo puede hacerse hasta k veces más rápido. Esta mejora ocurre para valores de m varios órdees de magnitus más grandes que n.

m es el número de muestras y n el número de hebras
(m debe ser múltiplo de n).

[EJEMPLOS1/calculopi]

Tras realizar la elaboración del programa «calculopi», estos son los primeros resultados obtenidos:

```
Ejemplo 4 (cálculo de PI)
valor de PI (calculado secuencialmente)    == 3.14159
valor de PI (calculado concurrentemente) == 3.1415
```

(con n = 4 y m = 1.000.000)
Esto nos permite deducir que el programa funciona correctamente.
Ahora, añadiré lo necesario para calcular el tiempo de calculo de la integral secuencial y concurrentemente y compararé los resultados.

En mi ordenador personal, que cuenta con siete procesadores, el tiempo de ejecución ha sido:

```
Tiempo de ejecución (secuencial) == 0.00744997
Tiempo de ejecución (concurrente) == 0.00326999
Diferencia: 0.00417997
```

Si cambiamos el valor de m por *m=1024*1024*8* y el de n por *n=8* tenemos:

```
Tiempo de ejecución (secuencial) == 0.0649169
Tiempo de ejecución (concurrente) == 0.029612
Diferencia: 0.035305
```

Como podemos ver, la diferencia aumenta cuando aumentamos el número de procesadores a utilizar. No obstante, dado que el número de procesadores es 7, si creamos más de 7 hilos no deberíamos notar la diferencia.

Con m=1024*1024*8 y n=12 obtenemos:

```
Tiempo de ejecución (secuencial) == 0.0579203
Tiempo de ejecución (concurrente) == 0.0245131
Diferencia: 0.0334071
```

Como vemos, apenas hay diferencia entre utilizar 7 o 12 hebras, ya que realmente solo se están ejecutando 7 de ellas al mismo tiempo.