Departamento de Informática
Sistemas Distribuidos

Ejercicios de laboratorio 2

En Aula Global en el apartado Laboratorio2 se tiene disponible este enunciado y el material de apoyo necesario para hacer los ejercicios propuestos. Descargue el material de apoyo en una máquina Linux (`lab2.tar`) y descomprímalo de la siguiente manera: `tar xvf lab2.tar`.

Ejercicio 1 En Aula Global se dispone del código del siguiente programa (`p1.c`), así como un archivo *Makefile* que permite compilar dicho programa (así como el resto de los programas descritos en el resto de los ejercicios). Para compilar este programa ejecute `make p1`.

```
#include <sys/time.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <pthread.h>

#define NUM_THREADS     2
#define ITER            10

void funcion(int *id) {
        int j;
        int s;
        double k;
        int mid = *id;  // cada thread recibe un número (0 o 1)

        for(j=0 ; j < ITER; j++) {
                k = (double) rand_r((unsigned int *) &s) / RAND_MAX;
                usleep((int) (k * 100000)); // duerme entre 0 y 100 ms
                printf("Ejecuta el thread %d iteracion %d \n", mid, j );
        }

        pthread_exit(NULL);

}
int main(int argc, char *argv[])
{
        int j;
        pthread_attr_t attr;
        pthread_t thid[NUM_THREADS];
        struct timeval t;

        gettimeofday(&t, NULL);
        srand(t.tv_sec);        // se inicializa la semilla de nº pseudoaleatorios

        pthread_attr_init(&attr);

        for (j = 0; j < NUM_THREADS; j++)
                if (pthread_create(&thid[j], NULL, (void *) funcion, &j) == -1){
                        printf("Error al crear los threads\n");
                        exit(0);
                }

        for (j = 0; j < NUM_THREADS; j++)
                pthread_join(thid[j], NULL);

        exit(0);

}
```

Este programa crea dos procesos ligeros, cada uno de los cuales ejecuta una serie de iteraciones en las que el thread se bloquea un tiempo aleatorio entre 0 y 100 ms e imprime su identificador y el número de la iteración. Compile (`make p1`) y ejecute el código anterior varias veces. Identifique los problemas de ejecución que observa

Ejercicio 2 Modifique el programa anterior de forma que cada uno de los threads imprima su identificador

(0 o 1) de forma correcta. El nombre de este programa será `p2.c`. Para compilarlo utilice `make p2`.

Ejercicio 3 Modifique el programa anterior de forma que el código siguiente de cada uno de los threads se ejecute en exclusión mutua. Utilice para ello un mutex. El nombre de este programa será `p3.c`. Para compilarlo utilice `make p3`. Esta solución sirve para poder acceder en exclusión mutua a algún tipo de recurso o fragmento de código que sea necesario proteger. En este caso no existe ningún problema dado que la ejecución de este fragmento de código no da lugar a posibles condiciones de carrera. El objetivo es ver cómo resolver esta situación en una aplicación en la que sea necesario.

```
k = (double) rand_r(&s) / RAND_MAX;
usleep((int) (k * 100000)); // duerme entre 0 y 100 ms
printf("Ejecuta el thread %d iteracion %d \n", mid, j );
```

¿Qué problemas observa?

Ejercicio 4 Modifique el programa del ejercicio 2 de forma que los dos threads ejecuten el código:

```
k = (double) rand_r(&s) / RAND_MAX;
usleep((int) (k * 100000)); // duerme entre 0 y 100 ms
printf("Ejecuta el thread %d iteracion %d \n", mid, j );
```

de forma alternada, primero el thread 0, luego el 1, luego el 0 y así sucesivamente. El nombre de este programa será `p4.c`. Para compilarlo utilice `make p4`.

Ejercicio 5 Modifique el programa anterior de forma que se creen 10 threads que ejecuten todos el código de la función `función`. Todos los threads tienen que alternar la ejecución del bucle de la función: primero el 0, luego el 1…, luego el 9, luego el 0, luego el 1, y así sucesivamente. El nombre de este programa será `p5.c`.

Ejercicio 6. En el material de apoyo se proporciona el código de un programa (`pi.c`) que calcula el número pi mediante el cálculo de la siguiente integral definida con el método de los trapecios.

$$\int_0^1 \sqrt{4\left(1-x^2\right)}\,dx = \frac{\pi}{2}$$

El programa `pi2.c` es una versión similar que está pensada como un primer paso para paralelizar el código y que pueda ser ejecutada por varios threads. Modifique el código `pi2.c` para que el cálculo sea hecho por una serie de threads en paralelo. El programa permite obtener el tiempo de ejecución. Compruebe cómo es el tiempo de ejecución de la versión secuencial respecto a la versión paralela. Para poder compilar estos programas deberá modificar previamente el archivo `Makefile` para incluir las reglas necesarias para su compilación.

Ejercicio 7. En el material de apoyo se proporciona el código de un programa productor-consumidor (`p6.c`). El programa copia un archivo pasado como parámetro en otro. El proceso productor lee del fichero e inserta los caracteres en un buffer, y el consumidor extrae los caracteres del buffer y los escribe en el fichero de salida. Compile, ejecute y analice el funcionamiento del programa. Modifique, a continuación, el programa anterior para que la ejecución de los dos procesos sea correcta. Tenga en cuenta que cuando el proceso productor finaliza la lectura del fichero debe notificar al consumidor dicho evento.

Ejercicio 8. En el material de apoyo se proporciona el código (`p7.c`) de un programa que intenta resolver el problema de los lectores-escritores (dando prioridad a los lectores). Ejecute el programa e identifique el problema existente. Modifique, a continuación, el programa anterior para que la ejecución de los procesos sea la correcta.

Nota: El fichero `Makefile` proporcionado permite compilar los 7 programas descritos anteriormente.

Ejercicio 9. Resuelva todos los problemas anteriores en Python.