Ejercicios de concurrencia

Ejercicio

Implementar un programa que ejecute 2 threads de forma tal que uno imprime por pantalla los números pares y otro imprime por pantalla los números impares desde 0 a 19. La salida debe estar ordenada y los thread se deben alternar de forma estricta.

Se debe programar usando mutex y variables condición.

Solución

```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <semaphore.h>
#include <sys/wait.h>
#include <unistd.h>
#include <sched.h>


int dato_comspartido = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t miturno = PTHREAD_COND_INITIALIZER;;
int turno = 0;

void  *par()
{
        int i;
            for (i=0; i<10; i++) {
                    pthread_mutex_lock(&mutex);
                    if (turno == 1)
                            pthread_cond_wait(&miturno, &mutex);
                    printf("Thread 1 %d \n", dato_comspartido++);
                    turno = 1;
                    pthread_cond_signal(&miturno);
                    pthread_mutex_unlock(&mutex);
             }
}

void  *impar ()
{
        int i;
            for (i=0; i<10; i++) {
```

```
                pthread_mutex_lock(&mutex);
                if (turno == 0)
                        pthread_cond_wait(&miturno, &mutex);
                printf("Thread 2 %d \n", dato_comspartido++);
                turno = 0;
                pthread_cond_signal(&miturno);
                pthread_mutex_unlock(&mutex);
        }
}


int main(void) {

        pthread_t th1, th2;

        pthread_mutex_init(&mutex, NULL);
        pthread_cond_init(&miturno, NULL);

        pthread_create(&th1, NULL, (void *)par, NULL);
        pthread_create(&th2, NULL, (void *) impar, NULL);

         pthread_join(th1, NULL);
            pthread_join(th2, NULL);

         pthread_mutex_destroy(&mutex);
         pthread_cond_destroy(&miturno);

}
```

EJERCICIO 16

Realizar un programa que cree 10 "threads", el primer "thread" sumara los números del 001-100 de un fichero que contiene 1000 numeros, y los siguentes "threads" sumaran sucesivamente los numeros que les correspondan: 101-200, 201-300, 301-400, 401-500, 601-700, 701-800, 801-900 y 901-1000 respectivamente. Los hijos devolveran al padre la suma realizada, imprimiendo este la suma total.
Utilice MUTEX para asegurar que no hay problemas de concurrencia entre los threads.

Solución

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

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
void *suma(void *rango);

pthread_mutex_t mtx;
pthread_cond_t cond;
int obtenidoRango;

pthread_attr_t attr;
int f=0;
pthread_t thread[10];

int main() {
    int i=0, n=0, rango=0, *estado, pestado=0, nbytes=0, nreg=0;
    estado=&pestado;
    pthread_attr_init(&attr);

    if((f=open("numeros.dat", O_RDONLY))==-1) {
        fprintf(stderr,"Error en la apertura del fichero\n");
        return(-1);
    }
    nbytes=lseek(f,0,SEEK_END);
    nreg=nbytes/sizeof(int);
    for(i=0;i<10;i++) {
        obtenidoRango=0;
        pthread_mutex_lock(&mtx);
        pthread_create(&thread[i],&attr,suma,&rango);
//        sleep (1);
        while (obtenidoRango==0)
          pthread_cond_wait(&cond, &mtx);
        pthread_mutex_unlock(&mtx);
        rango+=100;
    }
    for(i=0;i<10;i++) {
      pthread_join(thread[i],(void **)&estado);
      printf("Suma Parciales en Prog. Principal: %d\n",*estado);
      n+=*estado;
    }
    printf("Suma Total: %d\n",n);
    printf("Total numeros sumados: %d\n",nreg);
    close(f);
    return(0);
}

void *suma(void *rango) {
int j=0, valor, *suma, num=0;

//sleep(1);

    pthread_mutex_lock(&mtx);
```

```
      valor=*((int *)rango);
    obtenidoRango=1;
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&mtx);

    suma=(int *)malloc (sizeof (int));
   *suma=0;
    printf("Rango: %d a %d\n",valor+1,valor+100);
    lseek(f,valor * sizeof(int),SEEK_SET);
    for(j=0;j<100;j++) {
        read(f,&num,sizeof(int));
        *suma+=num;
    }
    printf("\tSuma Parcial: %d\n",*suma);
    pthread_exit(suma);
}
```