Lab 6: Sincronització de *Threads*

Objectius:

- Entendre la problemàtica de l'execució concurrent desincronitzada.
- Aprendre les comandes bàsiques de sincronització de *threads* a través de semàfors d'exclusió mútua (mutex).

Fitxers dèxemple:

`exemples_P2.1/mf_mutex.c`

Funcions bàsiques de llibreria:

```
int pthread_mutex_init(pthread_mutex_t *mutex,
                       const pthread_mutexattr_t *attr);
```

La funció `pthread_mutex_init` inicialitza el semàfor d'exclusió mútua `mutex`, amb els atributs que indica el paràmetre `attr`. Habitualment, però, el segon paràmetre es posa a `NULL` per a que el SO utilitzi els valors que té per defecte.

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

La funció `pthread_mutex_destroy` elimina la definició del semàfor d'exclusió mútua `mutex`.

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

La funció `pthread_mutex_lock` tanca el semàfor d'exclusió mútua `mutex`.

```
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

La funció `pthread_mutex_unlock` obre el semàfor d'exclusió mútua `mutex`.

Les funcions d'inicialització i destrucció s'han de cridar al principi i al final del programa, per a cada semàfor d'exclusió mútua que s'hagi de fer servir. Les funcions de tancar i d'obrir els semàfors s'han de cridar al principi i al final d'una secció crítica, per tal d'evitar que els altres fils puguin accedir a algun recurs compartit (evitar conflictes de l'execució concurrent).

Per crear un semàfor d'exclusió mútua s'ha d'utilitzar la següent línia (en llenguatge C):

```
pthread_mutex_t nom_mutex= PTHREAD_MUTEX_INITIALIZER;
```

Programa d'exemple:

```
/*********************************************************************/
/*                      mf_mutex.c                                   */
/*                                                                   */
/*    Compilar i executar:                                           */
/*          $ gcc -Wall  mf_mutex.c -o mf_mutex  -lpthread           */
/*          $ ./mf_mutex  num_threads  n_vegades  n_lletres      */
/*                                                                   */
/*    Executa tants threads com indica el parametre num_threads,     */
/*    on cada thread visualitza un caracter identificatiu            */
/*    (thread 0 -> 'a', thread 1 -> 'b', ...) tantes vegades com     */
/*    indiqui el segon parametre n_vegades, esperant un temps        */
/*    aleatori entre dues visualitzacions; el programa acaba quan    */
/*    s'han escrit un total de n_lletres entre tots els fils.        */
/*                                                                   */
/*    Es diferencia de l'exemple 'multifil.c' en la inclusio d'un    */
/*    semafor per fer exclusio mutua a l'hora d'accedir a la         */
/*    variable global que computa el numero de lletres a escriure.   */
/*********************************************************************/

#define _REENTRANT
#include ....

#define MAX_THREADS     10
#define MAX_VEGADES     50
#define MAX_LLETRES     100
                              /* Variables Globals */
pthread_t tid[MAX_THREADS];   /* taula d'identificadors dels threads */
int lletres;                  /* numero de lletres escrites */
int max_iter;                 /* numero maxim d'iteracions */

pthread_mutex_t mutex= PTHREAD_MUTEX_INITIALIZER; /* crea un sem. Global*/

/* escriure una marca corresponent al num. de thread ('a'+i_thr) en */
/* intervals aleatoris d'entre 1 i 3 segons, les vegades que indiqui */
/* la variable global max_iter */
void * caracter(void *i_thr)
{
  int i;
  char ic='a'+ (intptr_t) i_thr;

  for (i=0; i < max_iter; i++)     /* per a totes les vegades */
  {
    sleep(1 + rand() % 3);         /* dormir entre 1 i 3 segons */
    pthread_mutex_lock(&mutex);          /* tanca semafor */
    if (lletres > 0)               /* si falten lletres */
    {
      printf("%c",ic);             /* escriure marca del thread */
      lletres--;                   /* una lletra menys */
      pthread_mutex_unlock(&mutex);            /* obre semafor */
    }
    else
    { pthread_mutex_unlock(&mutex);            /* obre semafor */
      pthread_exit((void *) (intptr_t)i);   /* sino, forcar sortida thread */
    }
  }
  return((void *)(intptr_t) i); /*retorna numero de lletres escrites pel fil */
}
```

```
int main(int n_args, char * ll_args[])
{
    int i,n,t,t_total,n_thr;

    if (n_args != 4)
    { fprintf(stderr,"comanda: mf_mutex num_threads n_vegades n_lletres\n");
      exit(1);
    }
    n_thr = atoi(ll_args[1]);       /* convertir arguments a num. enter */
    max_iter = atoi(ll_args[2]);
    lletres = atoi(ll_args[3]);
    if (n_thr < 1) n_thr = 1;       /* i filtrar el seu valor */
    if (n_thr > MAX_THREADS) n_thr = MAX_THREADS;
    if (max_iter < 1) max_iter = 1;
    if (max_iter > MAX_VEGADES) max_iter = MAX_VEGADES;
    if (lletres < 1) lletres = 1;
    if (lletres > MAX_LLETRES) lletres = MAX_LLETRES;

    srand(getpid());          /* inicialitza la "llavor" dels aleatoris */
    setbuf(stdout,NULL);            /* anular buffer de sortida estandard */
    printf("Main thread del proces (%d) : ", getpid());

    pthread_mutex_init(&mutex, NULL);          /* inicialitza el semafor */
    n = 0;
    for ( i = 0; i < n_thr; i++)
    {
      if (pthread_create(&tid[n],NULL,caracter,(void *)(intptr_t) i) == 0)
            n++;
    }
    printf("he creat %d threads, espero que acabin!\n\n",n);

    t_total = 0;
    for ( i = 0; i < n; i++)
    {
      pthread_join(tid[i], (void **)&t);
      printf("el thread (%d) ha escrit %d lletres\n",i,t);
      t_total += t;
    }
    pthread_mutex_destroy(&mutex);       /* destrueix el semafor */

    printf("\nJa han acabat tots els threads creats!\n");
    printf("Entre tots els threads han escrit %d lletres.\n",t_total);
    return(0);
}
```

Exercici:

Realitzar la part de sincronisme de la pràctica 2.1.