Sesión 10: Threads y memoria compartida – Lectura previa

En esta sesión se va a practicar la creación de pthreads. Esto se va a hacer mediante cuatro ejemplos sencillos.
Los objetivos de esta sesión son:
-Crear threads
-Pasar parámetros a threads
-Esperar la finalización de threads y recoger el estado de finalización
-Crear zonas de exclusión mutua

Es importante que el alumno entienda perfectamente los siguientes ejemplos antes de la clase de laboratorio.
Los ejemplos se pueden encontrar en Albanta en: ~pr_so/public/S10.tar. Para linkarlos se tiene que utilizar la librería de pthreads (-lpthread).

Importante: Para entender bien los ejemplos es necesario dominar a la perfección los punteros en C.

Primer ejemplo: Creación de threads (Fichero: Ex1.c)

```
#define NUM_THREADS 10

void *CodiThread(void *num)
{ char frase[256];

        sprintf(frase,"[thread %d] Tinc identificador %d i acabo\n",(int) num,
                pthread_self());
        write(1,frase,strlen(frase));
        pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
pthread_t identif;
int aux, i;
char frase[256];

        for(i=0; i<NUM_THREADS; i++) {
                sprintf(frase,"[thread principal] Creant thread numero %d\n",i);
                write(1,frase,strlen(frase));
                aux=pthread_create(&identif, NULL, CodiThread, (void *)i);
                if (aux!=0) esc_error("",SISTEMA,TRUE);
                sprintf(frase,"[thread principal] Creat thread %d, amb identificador
                                %d\n",i,identif);
                write(1,frase,strlen(frase));
                }
        sprintf(frase,"[thread principal] acabo\n");
        write(1,frase,strlen(frase));
        pthread_exit(NULL);
}
```

Este ejemplo crea 10 threads que muestran su identificador y terminan.

Este ejemplo muestra la forma de crear un thread. Para esto se utiliza la llamada al sistema *pthread_create*. Sus parámetros son:
-Primer parámetro: variable del tipo *pthread_t,* pasada por referencia, donde se almacenará el identificador del nuevo thread.
-Segundo parámetro: un puntero a una estructura con los atributos del thread que se quiere crear. Si se quieren los atributos de creación por defecto, como es este caso, se pone NULL.

-Tercer parámetro: dirección de la rutina que va a ejecutar el thread. La cabecera de una función que se va a ejecutar con un thread siempre tiene la misma estructura:

```
void * nombre_funcion (void *variable)
```

-Cuarto parámetro: un puntero a los argumentos de la función (se verá en más detalle este parámetro en el siguiente ejemplo)

Para finalizar el thread, se usa *pthread_exit*. Tiene un único parámetro que es un puntero al estado de finalización (se verá más en detalle en el tercer ejemplo).

Segundo ejemplo: Paso de parámetros a un thread (Fichero: Ex2.c)

```
#define NUM_THREADS 6

typedef struct
{
  int numero;
  char *missatge;
} dades_thread;


void *CodiThread(void *a)
{ char frase[256];
dades_thread *mydata;

        mydata=(dades_thread *)a;

        sprintf(frase,"[thread %d] frase: %s\n",mydata->numero, mydata->missatge);
        write(1,frase,strlen(frase));
        pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
pthread_t identif[NUM_THREADS];
int nums[NUM_THREADS];
int aux, i;
dades_thread argument[NUM_THREADS];
char frase[256];

        for(i=0; i<NUM_THREADS; i++) {
                sprintf(frase,"[thread principal] Creant thread numero %d\n",i);
                write(1,frase,strlen(frase));

                /* omplo dades */
                sprintf(frase,"Frase del thread %d",i);
                argument[i].missatge=frase;
                argument[i].numero=i;


                aux=pthread_create(&identif[i], NULL, CodiThread,
                                        (void *)&(argument[i]);
                if (aux!=0) esc_error("",SISTEMA,TRUE);
                sprintf(frase,"[thread principal] Creat thread %d, amb identificador
                                %d\n",i,identif[i]);
                write(1,frase,strlen(frase));
                }
        sprintf(frase,"[thread principal] acabo\n");
        write(1,frase,strlen(frase));
        pthread_exit(NULL);
}
```

Este código crea seis threads que muestran el contenido de una estructura pasada como parámetro.

Este ejemplo muestra la forma de pasar argumentos a una función que va a ser ejecutado mediante un thread.

En el primer ejemplo a cada thread se pasa el orden en el que han sido creados. Esto se hace mediante el contador del bucle. El cuarto parámetro de *pthread_create*, que tiene que ser un puntero a void, es el que se utiliza para pasar este argumento. Si nos fijamos, lo que se hace es convertir (casting de tipos) el entero a un puntero a void mediante:

```
(void*)i
```

Es necesario resaltar que hay diferencia entre la declaración anterior y la siguiente:

```
(void*)&i
```

La primera declaración convierte *i* de entero a puntero de datos. En ningún momento se trabaja con la dirección de *i*, sino con la interpretación de los datos de *i*. En el segundo caso, se pasa la dirección de la variable *i* mediante un puntero que se convierte de (int *) a (void *).

El thread también recibe el parámetro como un (void*). En el primer ejemplo, puesto que se transformó *i* de entero a (void*), dentro del thread se tiene que hacer la transformación inversa: de (void*) a entero. Esto se hace mediante la declaración (int).

En este segundo ejemplo no se utiliza el casting de tipos (de entero a puntero) sino que se pasa directamente un puntero. Esto se hace puesto que los argumentos de la función ocupan más que el espacio de un puntero. Por tanto, en este caso SI que se pasa un puntero a una estructura aunque este puntero hay que transformarlo a (void*). Dentro del thread, se tiene que hacer la transformación inversa, en este caso, de (void*) a (dades_thread*). Una vez hecha esta transformación, ya se puede utilizar el puntero a una estructura de una forma normal.

RECORDATORIO: si se quieren acceder a los campos de una estructura a través de un puntero a esa estructura, se tiene que hacer mediante -> .

Tercer ejemplo: Esperar la finalización de threads y recogida de resultados (Fichero: Ex3.c)

```
#define NUM_THREADS 3

void *CodiThread(void *num)
{ char frase[256];
int segons;

        segons=(random()%10)+1; /*genero un numero aleatori entre 1 i 10 */
        sprintf(frase,"[thread %d] Ara dormire %d segons\n",(int) num, segons);
        write(1,frase,strlen(frase));
        sleep(segons);
        sprintf(frase,"[thread %d] Em desperto i tornare un %d\n",num,segons);
        write(1,frase,strlen(frase));
        pthread_exit( (void *) segons);
}

int main(int argc, char *argv[])
{
pthread_t identif[NUM_THREADS];
int status;
int aux, i;
char frase[256];

        srand(getpid()); /* preparo la generacio de numeros aleatoris */
        for(i=0; i<NUM_THREADS; i++) {
                aux=pthread_create(&identif[i], NULL, CodiThread, (void *)i);
                if (aux!=0) esc_error("",SISTEMA,TRUE);
                sprintf(frase,"[thread principal] Creat thread %d, amb identificador
                                %d\n",i,identif[i]);
                write(1,frase,strlen(frase));
                }
        for(i=0; i<NUM_THREADS;i++) {
                 pthread_join(identif[i], (void **) &status);
                 sprintf(frase,"[thread principal] Mort thread %d, retorna
                                %d\n",i,status);
                 write(1,frase,strlen(frase));
                }

        sprintf(frase,"[thread principal] acabo\n");
        write(1,frase,strlen(frase));
        pthread_exit(NULL);
}
```

Este ejemplo crea 3 threads secundarios. El thread principal espera la finalización de estos threads antes de acabar.

Para esperar la finalización de un thread se hace mediante *pthread_join*. Tiene como parámetros;

-el identificador del thread que se quiere esperar. Este identificador es el que se obtiene con *pthread_create*.
-un puntero a una variable donde se guardará el estado de finalización.

En este caso, como el estado de finalización es un entero, es semejante al paso de parámetros del primer ejemplo. El segundo parámetro de *pthread_join* es un (void**), es decir, un puntero a un puntero a void. En nuestro caso, como en el primer ejemplo, queremos transformar un entero a puntero a void, es decir, usamos:

```
(void*)entero
```

Como, además, tenemos que pasar este nuevo puntero por referencia, tendremos que hacer:

```
&((void*)entero)
```

Por tanto, tenemos un puntero que apunta a otro puntero a void que en nuestro caso, en realidad es un entero, por tanto, tendríamos:

```
(void*)&((void*)entero)
```

que es lo mismo que decir:

```
(void**)&entero
```

Para finalizar un thread y pasar un estado de finalización, se utiliza *pthread_exit* cuyo único parámetro es un puntero al estado de finalización.

En nuestra rutina que ejecutan los threads, se utiliza el casting de tipos para convertir un entero a un puntero a void.

Cuarto ejemplo: Creación de zonas de exclusión mutua (Fichero: Ex4.c)

```
#define NUM_THREADS 10

int mivar=0;
pthread_mutex_t seccio;

void *CodiThread(void *num)
{ char frase[256];
int local;

        pthread_mutex_lock(&seccio);
        local=mivar;
        sprintf(frase,"[thread %d] mivar=%d, li sumo %d\n",(int) num, local,
                        (int) num);
        write(1,frase,strlen(frase));
        local+=(int) num;
        mivar=local;
        pthread_mutex_unlock(&seccio);
        pthread_exit( (void *) NULL);
}

int main(int argc, char *argv[])
{
pthread_t identif[NUM_THREADS];
int aux, i;
char frase[256];

        pthread_mutex_init(&seccio,0);

        for(i=0; i<NUM_THREADS; i++) {
                aux=pthread_create(&identif[i], NULL, CodiThread, (void *)i);
                if (aux!=0) esc_error("",SISTEMA,TRUE);
                sprintf(frase,"[thread principal] Creat thread %d, amb identificador
                                %d\n",i,identif[i]);
                write(1,frase,strlen(frase));
                }
        for(i=0; i<NUM_THREADS;i++) {
                 pthread_join(identif[i], (void **) &status);
                 sprintf(frase,"[thread principal] Mort thread %d, retorna %d\n",
                        i,status);
                 write(1,frase,strlen(frase));
                }
        sprintf(frase,"[thread principal] acabo, var = %d\n", mivar);
        write(1,frase,strlen(frase));
        pthread_exit(NULL);
}
```

Este último ejemplo crea una sección crítica para asegurar el acceso secuencial entre threads a una variable global.

Para crear una sección crítica, lo primero que se necesita es una variable del tipo *pthread_mutex_t* que identifique esta sección. Esta variable, NECESARIAMENTE tiene que ser GLOBAL. En ningún caso puede ser local a la rutina del thread ni al *main*.

Una vez creada esta variable, se tiene que inicializar. Para ello se utiliza *pthread_mutex_init* cuyo primer parámetro es la variable, pasada por referencia y el segundo los atributos (para nosotros, 0).

A partir de ahora, ya se puede utilizar esta variable para declarar zonas de exclusión mutua. El principio de una zona de exclusión se marca con *pthread_mutex_lock* cuyo único parámetro es la variable que identifica esta zona. El final de la zona de

exclusión mutua se señala mediante *pthread_mutex_unlock* cuyo parámetro es la variable que identifica esta zona.