

Fundamentos de los Sistemas Operativos (FSO)

Departamento de Informàtica de Sistemes y Computadoras (DISCA)
Universitat Politècnica de València

Bloque Temático 2: Gestión de Procesos
Unidad Temática 5

SUT5: Programación con hilos POSIX

fSO

DISCA



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

- **Objetivos**

- Trabajar las llamadas al sistema POSIX relacionadas con la **creación y gestión básica de hilos**
- Trabajar el **concepto de condición de carrera** para comprender la problemática que introduce su existencia en la **programación concurrente**

- **Bibliografía**

- “UNIX Programación Práctica”, Kay A. Robbins, Steven Robbins. Prentice Hall. ISBN 968-880-959-4 . Cap 9

- **Contenido**
 - **Introducción**
 - Creación de hilos
 - Terminación y espera de hilos
 - Identificación de hilos
 - Condición de carrera

- **Procesos POSIX**

- Crean un **hilo** (thread) **inicial** que ejecuta la función **main()**
 - Cualquier hilo puede crear más hilos para ejecutar otras funciones dentro del espacio de direcciones del proceso
- **Todos los hilos** de un proceso están **al mismo nivel**
 - Esto significa que son “**hermanos**”, a diferencia de los procesos cuya relación es “padre-hijo”
- Los **hilos de un proceso comparten las variables y recursos globales** (ficheros, manejadores de señales, etc.) del proceso
 - Cada hilo tiene una **copia privada de sus parámetros iniciales y de las variables locales** de la función que ejecuta

- Llamadas básicas para **gestión de hilos/threads**

	Hilos de ejecución (pthreads)
pthread_create	Crea un hilo para ejecutar una función específica
pthread_attr_init	Inicializa un objeto atributo de hilo a los valores por omisión
pthread_attr_destroy	Destruye el objeto atributo de hilo
pthread_join	Espera la terminación del hilo especificado
pthread_exit	Termina la ejecución del hilo invocador
pthread_self	Devuelve la identidad del hilo que lo invoca
pthread_attr_setdetachstate	Modifica el atributo de desconectado
pthread_attr_getdetachstate	Consulta el atributo de desconectado

- Introducción
- **Creación de hilos**
- Terminación de hilos
- Espera de hilos
- Identificación de hilos
- Condición de carrera

```
#include <pthread.h>
int pthread_create(pthread_t *thread,
                  const pthread_attr_t *attr,
                  void *(*start_routine)(void*),
                  void *arg);
```

- **pthread_create ()**
 - Crea inmediatamente un **nuevo hilo en estado preparado**
 - El hilo creado y el creador **compiten por la CPU** según la política de planificación del sistema
 - Puede ser invocada por cualquier hilo del proceso (no sólo por el “hilo inicial”) para crear otro hilo
- **argumentos**
 - **attr**: atributo que contiene las características del nuevo hilo
 - **start_routine** es la función que ejecutará el hilo
 - **arg** es un puntero a los parámetros iniciales del hilo
 - en **thread** se devuelve el identificador del hilo creado

- Atributos para creación de hilos :

```
#include <pthread.h>

int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
```

donde

- **attr** es el atributo que debe ser creado/destruido
- Estas llamadas
 - Crean/destruyen un atributo de creación de hilos
 - La función “**init**” inicializa “**attr**” con los valores por defecto
 - Estos valores se pueden modificar con funciones específicas
 - Se pueden crear múltiples hilos con los mismos atributos, misma variable “**attr**”

- Para modificar los atributos de creación de hilos:

```
#include <pthread.h>
```

```
int pthread_attr_setdetachstate(pthread_attr_t *attr,  
                                int detachstate);
```

```
int pthread_attr_getdetachstate(const pthread_attr_t *attr,  
                                int *detachstate);
```

- donde
 - **detachstate** indica si otro hilo podrá esperar a la finalización de este hilo, mediante una instrucción `pthread_join`:
 - `PTHREAD_CREATE_JOINABLE`
 - `PTHREAD_CREATE_DETACHED`

- Ejemplo: Hello World. Sin espera de hilos

```
#include <stdio.h>
#include <string.h>
#include <pthread.h>
```

```
//fichero: pthread_hello.c
//compilar: gcc pthread_hello.c -o pthread_hello -lpthread
//mostrar hilos en el shell: $ps -lT
```

```
void *My_Print(void *ptr )
{ char *message;
  message = (char *) ptr;
  write(1,message,strlen(message));
}

int main()
{
  pthread_t thread1, thread2;
  pthread_attr_t attr;

  pthread_attr_init(&attr);
  pthread_create(&thread1, &attr, My_Print, "Hello ");
  pthread_create(&thread2, &attr, My_Print, " World\n");

  return 0;
}
```

- Introducción
- Creación de hilos
- **Terminación y espera de hilos**
- Identificación de hilos
- Condición de carrera

- **Finalización de hilos POSIX**

- Un hilo termina voluntariamente su ejecución cuando:
 - Se ejecuta la última instrucción de su función, o bien
 - el hilo invoca a `pthread_exit`

```
#include <pthread.h>

int pthread_exit(void *exit_status);
```

donde

- **exit_status** es un puntero a una variable mediante la cual un hilo que finaliza con `pthread_exit` comunica un valor a otro que está esperando su terminación en `pthread_join`)
- La finalización del último hilo de un proceso finaliza el proceso
- Si `main()` no finaliza con `pthread_exit` el proceso terminará y, por lo tanto, los hilos del mismo que se encuentren en ejecución también terminarán

- **Espera de hilos**

- Mediante la llamada `pthread_join` se puede **esperar a la finalización de un hilo**, siempre que se haya creado con el atributo `PTHREAD_CREATE_JOINABLE`

```
#include <pthread.h>
```

```
int pthread_join(pthread_t thread, void **exit_status);
```

- **exit_status** es el valor que el hilo terminado con `pthread_exit` comunica al hilo que invoca la llamada `pthread_join`
- Se suspende la ejecución del hilo que invoca a la función, hasta que el hilo esperado finalice

- Ejemplo

```
...  
void *funcion(void *p) {  
    printf("Soy un hilo feliz!\n");  
    sleep(10);  
}  
...  
int main( void ) {  
    pthread_t      id_hilo;  
    pthread_attr_t  atributos;  
  
    printf("Hilo principal: principio\n");  
    pthread_attr_init(&atributos);  
    pthread_create(&id_hilo, &atributos, funcion, NULL);  
    printf("Hilo principal: He creado un hermano\n");  
    pthread_join(id_hilo, NULL);  
    printf("Hilo principal: Acabo!");  
}
```

¡Aviso! Hemos de declarar variable de tipo hilo. Una variable por cada hilo a crear

¿Cual sería el resultado de ejecución si eliminamos el pthread_join?

- Introducción
- Creación de hilos
- Terminación y espera de hilos
- **Identificación de hilos**
- Condición de carrera

- **Identificación de hilos**

```
#include <pthread.h>
```

```
pthread_t pthread_self(void);
```

```
int pthread_equal(pthread_t th1, pthread_t th2);
```

donde

- **pthread_self** devuelve el identificador interno del hilo invocante.
- puesto que este identificador puede no ser escalar, la función **pthread_equal** sirve para comparar dos identificadores.
- La función **pthread_equal** devuelve
 - **cero** (0) si dos identificadores son distintos o **no** son iguales
 - Un valor distinto de cero si son iguales los identificadores

Ejemplo: creación periódica de hilos

```
int main ()
{ pthread_t t1,t2;
  pthread_attr_t attr;
  int period1=1, period2=2;
```

```
if (pthread_attr_init(&attr) != 0)
{ printf("Error: atributtes\n");
  exit(1);
}
```

```
if (pthread_create(&t1, &attr, func_period, &period1) != 0)
{ printf("Error: creating first pthread\n");
  exit(1);
}
```

```
if (pthread_create(&t2, &attr, func_period, &period2) != 0)
{ printf("Error: creating second pthread\n");
  exit(1);
}
```

```
pthread_join(t1, NULL);
pthread_join(t2, NULL);
```

```
}
```

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

```
void *func_period (void *arg) {
  int period, i;
  period= *((int *)arg);
  for (i=0; i<10; i++) {
    printf("Pthread(period %d):", period);
    printf(" %ld\n", (long) pthread_self());
    sleep (period);
  }
}
```

```
//fichero: th_periodic.c
//compilar:gcc th_periodic.c -o th_periodic -lpthread
//mostrar hilos en el shell: $ps -lT
```

- Introducción
- Creación de hilos
- Terminación y espera de hilos
- Identificación de hilos
- **Condición de carrera**

•Ejemplo: “globalvar.c”

- Proceso que incrementa en 40000000 unidades una variable global
- Resultado correcto de ejecución: **Globalvariable=40000000**

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int Globalvariable=0;
```

```
int main()
{   int i;
    long iterations = 40000000;
    for (i=0; i<(iterations); i++){
        Globalvariable++;
    }
    printf("Globalvariable= %d\n",Globalvariable);
    return 0;
}
```

//fichero: globalvar.c
//compilar: gcc globalvar.c -o globalvar

¿¿Resultado
de ejecución??

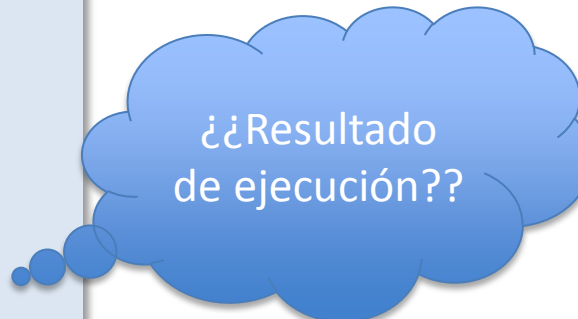
```
//fichero: race_condition.c  
//compilar: gcc race_condition.c -o race_condition -lpthread
```

- Ejemplo “**race_condition.c**”

- Versión concurrente de “globalvar.c”
- **Dos hilos** colaboran para **incrementar variable**
- Cada hilo realiza **20000000 operaciones**
- Resultado de ejecución esperado
Globalvariable = 40000000

```
int main()  
{ long iterations = 20000000;  
  pthread_t t1, t2;  
  pthread_attr_t attr;  
  pthread_attr_init(&attr);  
  pthread_create(&t1, &attr, Addition, &iterations);  
  pthread_create(&t2, &attr, Addition, &iterations);  
  pthread_join(t1, NULL);  
  pthread_join(t2, NULL);  
  printf("Globalvariable= %d\n", Globalvariable);  
  return 0;  
}
```

```
#include <stdio.h>  
#include <pthread.h>  
int Globalvariable=0;  
void *Addition(void *ptr )  
{int i, aux_variable;  
  int *iter = (int *)ptr;  
  for (i=0; i<*iter; i++){  
    aux_variable = Globalvariable;  
    aux_variable++;  
    Globalvariable = aux_variable;  
  }  
}
```



¿¿Resultado de ejecución??