DESARROLLO DE SOFTWARE PARA SISTEMAS EMPOTRADOS

Gestión básica de threads en POSIX

Mario Aldea
Michael González
Héctor Pérez

Gestión de threads POSIX

- Threads POSIX: Conceptos básicos
- Creación de threads
- Terminación de threads
- Identificación de threads

Threads POSIX: Conceptos básicos

- El estándar POSIX distingue entre *procesos* y *threads*
  - Un *proceso* está formado por uno o más *threads*
  - Los threads de un proceso
    - comparten un único espacio de direcciones
    - tienen tiempos de cambio de contexto menores
    - presentan tiempos de creación y destrucción menores

- POSIX define una API para soportar múltiples threads o flujos de control en cada proceso
  - Esta API está definida en lenguaje C
  - La librería se denomina ***pthread***
    - todas las operaciones están definidas en el fichero *pthread.h*

DSSE: M. Aldea, M. González y H. Pérez

Threads POSIX: Conceptos básicos

- Un **thread** representa un flujo de control simple perteneciente a un proceso
  - tiene un identificador de thread (***tid***)
    - este ***tid*** solo es válido para threads del mismo proceso
  - tiene su propia política de planificación y los recursos del sistema necesarios (por ejemplo, su propio *stack)*
  - todos los threads de un proceso comparten un único espacio de direccionamiento

- **Proceso** en una implementación multi-thread:
  - un espacio de direccionamiento con uno o varios threads
  - inicialmente contiene un solo thread: el **thread principal**
  - el thread principal se considera un thread más con una característica particular
    - cuando él termina, todos los demás threads del proceso también

Threads POSIX: conceptos básicos

- Los servicios bloqueantes afectan al thread invocante
  - el resto de threads del proceso siguen ejecutándose
  - por ejemplo, al invocar la función *sleep()*
- Los threads tienen dos estados posibles para controlar la devolución de recursos al sistema:
  - *"detached" o independiente*: *PTHREAD_CREATE_DETACHED*
    - cuando el thread termina, devuelve al sistema los recursos utilizados
    - no se puede esperar su terminación
  - *"joinable" o sincronizado*: *PTHREAD_CREATE_JOINABLE*
    - cuando el thread termina, mantiene sus recursos
    - se liberan cuando otro thread llama a *pthread_join()*
- **Por defecto,** un thread es "*joinable*"

Creación de threads

- Para crear un thread es preciso definir sus atributos en un objeto especial (*pthread_attr_t*)
- El objeto de atributos
  - debe crearse antes de usarlo: *pthread_attr_init*()
  - puede borrarse: *pthread_attr_destroy*()
  - se pueden modificar los atributos concretos del objeto
    - pero no los del thread, que se **fijan** al crearlo
- Los atributos definidos son:
  - tamaño de *stack* mínimo (opcional)
  - dirección del *stack* (opcional)
  - control de devolución de recursos (*"detach state"*)
  - atributos de planificación

Creación de threads: Interfaz de gestión de atributos

```
#include <pthread.h>

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

int pthread_attr_setstacksize (pthread_attr_t *attr,
                               size_t stacksize);
int pthread_attr_getstacksize (const pthread_attr_t *attr,
                               size_t *stacksize);

int pthread_attr_setstackaddr (pthread_attr_t *attr,
                               void *stackaddr);
int pthread_attr_getstackaddr (const pthread_attr_t *attr,
                               void **stackaddr);

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

PTHREAD_CREATE_DETACHED
PTHREAD_CREATE_JOINABLE

Creación de threads

- Función para la creación de un nuevo thread:

```
int pthread_create(pthread_t *thread,
                   const pthread_attr_t *attr,
                   void *(*start_routine)(void *),
                   void *arg);
```

Parámetro	Significado
thread	identificador del thread creado
attr	define los atributos del thread
start_routine	puntero a la función a ejecutar por el thread
arg	argumento que se pasa a la función a ejecutar
Retorno	**Cero si se ha ejecutado con éxito** **Cualquier otro valor en caso de error**

Creación de threads: Ejemplo (1/2)

```
// Dado que se utiliza la librería pthread, debe indicarse en el
// proceso de compilación.
// Ejemplo: gcc main.c -lpthread
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <misc/error_checks.h> // Define la macro no estándar CHK()


// Thread que pone pseudo-periódicamente un mensaje en pantalla
// el periodo se le pasa como parámetro
void *periodic (void *arg) {
        int period;

        period = *((int *)arg);
        while (1) {
                printf("En el thread con periodo %d\n",period);
                sleep (period);
        }
}
```


Creación de threads: Ejemplo (2/2)

```
// Programa principal que crea dos threads pseudo-periódicos
int main ()
{
        pthread_t th1,th2;
        pthread_attr_t attr;
        int period1=2
        int period2=3;
        // Crea el objeto de atributos para crear threads independientes
        CHK( pthread_attr_init(&attr) );
        CHK( pthread_attr_setdetachstate(&attr,
                                         PTHREAD_CREATE_DETACHED) );

        // Crea los threads
        CHK( pthread_create(&th1, &attr, periodic, &period1) );
        CHK( pthread_create(&th2, &attr, periodic, &period2) );

        // Les deja ejecutar un rato y luego termina
        sleep(30);
        printf("thread main terminando \n");
         CHK( pthread_attr_destroy(&attr) );  // Liberamos recursos
        exit (0);
}
```

Terminación de threads

- Función para terminación de threads:

```
#include <pthread.h>
void pthread_exit (void *value_ptr);
```

- Esta función termina el thread y hace que el valor apuntado por *value_ptr* esté disponible para una operación "*join*"
  - se ejecutan las rutinas de cancelación pendientes
  - al terminar un thread es un error acceder a sus variables locales

DSSE: M. Aldea, M. González y H. Pérez

Terminación de threads

- Se puede esperar (*join*) a la terminación de un thread cuyo estado es sincronizado (*joinable*):

```
#include <pthread.h>
int pthread_join (pthread_t thread,
                  void **value_ptr);
```

- También se puede cambiar el estado del thread a "*detached*":

```
#include <pthread.h>
int pthread_detach (pthread_t thread);
```

- **Por defecto,** el estado de un thread es "*joinable*"

Terminación de threads: Ejemplo (1/2)

```
#include <pthread.h>
#include <stdio.h>
#include <misc/error_checks.h> // Define la macro no estándar CHK()

#define MAX   500000
#define MITAD (MAX/2)

typedef struct {
  int *ar;
  long n;
} subarray_t;

// Thread que incrementa n componentes de un array
void * incrementer (void *arg) {
  long i;
  subarray_t * subarray = ((subarray_t *)arg);

  for (i=0; i < subarray->n; i++) {
    subarray->ar[i]++;
  }
  return NULL;
}
```


Terminación de threads: Ejemplo (2/2)

```
// programa principal que reparte entre dos threads el incrementar los componentes de un array
int main() {
  int ar [MAX] = { 0 }; // inicializa los elementos a 0
  pthread_t th1,th2;
  subarray_t sb1,sb2;
  long suma=0, i;

  // crea los theads
  sb1.ar = &ar[0]; sb1.n = MITAD; // primera mitad del array
  CHK( pthread_create(&th1, NULL, incrementer, &sb1) );

  sb2.ar = &ar[MITAD]; sb2.n = MITAD; // segunda mitad del array
  CHK( pthread_create(&th2, NULL, incrementer, &sb2) );

  // sincronizacion de espera a la finalizacion
  CHK( pthread_join(th1, NULL) );
  CHK( pthread_join(th2, NULL) );
  printf ("Ambos threads han finalizado \n");

  for (i=0; i<MAX; i++) // muestra resultados
    suma=suma+ar[i];
  printf ("Suma=%d\n",suma);
  return 0;
}
```

Identificación de threads

- Identificación del propio thread:

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

- Comparación de *tids*:

```
int pthread_equal (pthread_t t1, pthread_t t2);
```