Sistemas en Tiempo Real – Práctica Posix 4

Configuración del equipo

Los PCs actuales son multicore, un aspecto que no contemplamos en esta práctica, de modo que necesitamos utilizar un único core. Para ello configuraremos el número de cores útiles en tiempo de ejecución en la línea de comandos (lo que haremos será desactivar los cores que no utilicemos). Para desactivar un core escribiremos un 0 en el fichero *online* de dicho core mediante el siguiente comando:

echo 0 | sudo tee /sys/devices/system/cpu/cpu1/online

El comando anterior desactiva solamente el core 1 (*cpu1* en la ruta), por lo que para desactivar otros cores hay que cambiar *cpu1* por *cpuX*, siendo *X* el core que queremos desactivar. Escribir un uno vuelve a activarlo y reiniciar el equipo activa todos los cores. Para usar un único core debemos **desactivar todos los cores excepto el 0.**

Modificación de la clase *hilo_t*

Para esta práctica vamos a tener que modificar la clase *hilo_t* implementada en la Práctica Posix 0 ya que debe manejar los datos correspondientes al marco temporal del hilo. Para ello, añadiremos los siguientes elementos:

- Nuevos atributos privados:
  - Tiempo de ejecución en milisegundos (tipo entero).
  - Identificador del hilo (tipo entero).
  - Prioridad del hilo (tipo entero).
  - Política del hilo (tipo entero).
  - Periodo de repetición en milisegundos (tipo *struc timespec*).
  - Instante de comienzo (tipo *struct timespec*). Este dato se utilizará para que todas las tareas comiencen a la vez, evitando así los retardos surgidos de la creación de tareas.
- Nuevos métodos públicos:
  - *Int EstablecerAtributos* → Tendrá los siguientes parámetros (deben estar en el orden indicado):
    - Prioridad (entero) → Indicará la prioridad con que se va a crear el hilo.
    - Política de planificación (entero) → Indicará la política de planificación con la que se va a crear el hilo.
    - Tipo de herencia (entero) → Indicará si el hilo se creará usando los atributos de creación o si heredará esos valores del hilo padre.
    - Periodo de repetición (entero) → Indicará el periodo de repetición del hilo. Este dato vendrá dado en milisegundos y se transformará a *struct timespec* dentro de la clase.
    - Tiempo de ejecución de la tarea periódica del hilo (entero) → Indicará el tiempo de ejecución más desfavorable, en milisegundos, de la tarea periódica que ejecutará el hilo.
    - Identificador del hilo (entero).
    - Instante de comienzo (*struct timespec*) → Este dato indicará el instante en que comenzará a ejecutarse la acción periódica del hilo.

    Este método se va a encargar de asignar los valores de los diferentes parámetros a los atributos de clase a los que corresponden. Además, también establecerá los valores de herencia, política y prioridad en los atributos de creación del hilo (ver el Anexo 1) para que el hilo se cree usando estos valores al llamar al método *Lanzar*.
  - *struct timespec ObtenerPeriodo* sin parámetros → Devolverá el periodo del hilo.

- *struct timespec ObtenerInstanteDeComienzo* sin parámetros → Devolverá el instante de comienzo del hilo.
- *Int ObtenerId* sin parámetros → Devolverá el identificador del hilo.
- *Int ObtenerTiempoEjecucion* sin parámetros → Devolverá el tiempo de ejecución de la tarea periódica del hilo.
- *Int ObtenerPrioridad* sin parámetros → Devolverá la prioridad del hilo.
- *Int ObtenerPolitica* sin parámetros → Devolverá la política de planificación del hilo.

Tarea Periódica Genérica

Se deberá implementar una tarea periódica genérica que tomará como parámetro un puntero a *hilo_t* que permitirá acceder a todos los datos del hilo necesarios para la ejecución de la tarea (debe tener el formato de las funciones usadas para crear hilos, ver el Anexo 1). A continuación, se muestra el pseudocódigo de esta tarea:

```
Inicio Tarea Periódica Genérica
   Transformar el parámetro de entrada de void* a hilo_t*
   Obtener el periodo y el instante de comienzo del hilo
   Calcular el instante siguiente de repetición sumando los dos
    valores anteriores
   Dormir el hilo hasta el instante de comienzo
   Bucle infinito
      Mensaje de comienzo de la acción periódica
      Ejecución de la acción periódica
      Mensaje de finalización de la acción periódica
      Dormir el hilo hasta el instante siguiente
      Calcular el nuevo instante siguiente añadiéndole el periodo
   Fin bucle infinito
Fin Tarea Periódica Genérica
```

Para la ejecución de la acción periódica, se dispone de la función *Ejecutar(int n)* (está ya implementada en el esqueleto de la práctica) que mantiene al hilo invocante en ejecución durante *n* milisegundos (el hilo no se duerme, es una espera activa, por lo que se mantiene en ejecución mientas no haya ningún otro hilo que lo saque del procesador).

Tarea Mostrar Tiempos

Además de la tarea periódica genérica indicada anteriormente, deberá crearse otra tarea periódica, de periodo 10 ms que, tras esperar al instante de comienzo, su acción periódica será mostrar un mensaje en el que se indica el instante de tiempo actual en milisegundos. Esto se repetirá hasta que se haya ejecutado 1 segundo (100 iteraciones de 10 ms).

El programa principal

El programa principal se encargará de leer los datos de las tareas de un fichero de entrada (está ya implementado en el esqueleto de la práctica) así como de configurar y lanzar los diferentes hilos (un hilo de tipo Tarea Periódica Genérica por cada tarea del fichero de tareas más un hilo de tipo Tarea Mostrar Tiempos). Además, esperará a que termine la Tarea Mostrar Tiempos (el resto ejecutan bucles infinitos, por lo que cuando esta tarea termine, terminarán todas al ser la única por la que esperamos). La configuración de los diferentes hilos será la siguiente:

- Tiempo de ejecución, periodo, y prioridad de las tareas periódicas genéricas → Vienen dados en el fichero de tareas (ver más abajo el formato del mismo).

- Tiempo de ejecución, periodo, y prioridad de la tarea mostrar tiempos → 0 (no ejecuta ninguna tarea periódica, con lo que su tiempo de ejecución es irrelevante), 10 ms y prioridad máxima para la política FIFO (establecer a este hilo la prioridad máxima permitirá que saque a los demás hilos del procesador para mostrar las franjas temporales).
- Identificador de hilo → 0 para la tarea mostrar tiempos y de 1 a N (siendo N el número de tareas) para las tareas periódicas genéricas.
- Instante de comienzo → Es igual para todos los hilos y se calculará sumando dos segundos a la hora actual.
- Política → FIFO para todos los hilos.
- Herencia → Es igual para todos los hilos y debe indicar que no se va a heredar del padre.

El formato del fichero de tareas es el siguiente:

```
Número de tareas
C1 T1 P1
…
Cn Tn Pn
```

Siendo C_i, T_i y P_i el tiempo de ejecución, el periodo de repetición y la prioridad respectivamente de la tarea i-ésima.

La práctica se comprimirá en un único archivo que será entregado a través del campus virtual usando la tarea creada para tal efecto.

Anexo 1 – Posix

- Definición de la función que se usará para lanzar un hilo: *void *NombreFuncion(void *parámetro)*.
- Paso de parámetros a un hilo:
  - Parámetro de tipo puntero:

```
void *funcion(void *aux) { //Por ejemplo, el parámetro es un vector de enteros
        int *n=(int *)aux;
 //Resto del código usando n de forma normal
}
```

  - Parámetro de tipo no puntero:

```
void *funcion(void *aux) { //El parámetro es un dato de tipo TDato
        Tdato d=*((TDato *)aux);
 //Resto del código usando d de forma normal
}
```

- **pthread_attr_setinheritsched(pthread_attr_t *attr, int inherit)** → Establece, en los atributos de creación de una hebra, si hereda o no la política/prioridad del padre. Por defecto tiene un valor *PTHREAD_INHERIT_SCHED* (hereda). Para establecer la política/prioridad de un hilo, hay que cambiar este campo a *PTHREAD_INHERIT_SCHED*. Este valor se indica en el parámetro *inherit*.
- **pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy)** → Establece *policy* como política de planificación de la hebra que se cree usando los atributos *attr*. Los valores posibles son *SCHED_FIFO*, *SCHED_RR* (Round Robin), *SCHED_SPORADIC* (servidor esporádico) o *SCHED_OTHER* (depende de la implementación).
- **pthread_attr_setschedparam(pthread_attr_t *attr, const struct sched_param param)** → Asigna los parámetros de planificación *param* a los atributos *attr*. La estructura *sched_param* está definida de la siguiente forma:

```
struct sched_param {
    int sched_priority;
}
```

  Donde el campo *sched_priority* será el valor de prioridad que se establecerá en los atributos de creación de la hebra.