Sistemas en Tiempo Real – P10

Continuación de la práctica 8

Esta práctica añade/modifica funcionalidades a la práctica 8 (en rojo).

Configuración del equipo

Los PCs actuales son multicore, un aspecto que no contemplamos en esta práctica, de modo que necesitamos que Linux explote un único core. Para ello configuraremos el número de cores útiles en tiempo de ejecución en la línea de comandos. Concretamente el segundo core (van del 0 a x-1, siendo x el número de cores) se deshabilita mediante el comando:

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

Realmente lo que hace el comando es escribir un cero en el fichero online correspondiente al core 1 para deshabilitarlo. Escribir un uno vuelve a habilitarlo. Deshabilitaremos todos los cores excepto el 0.

Marco temporal

El tipo *marco_temp_t* se encargará de almacenar el marco temporal de cada hebra. Es una ampliación de la estructura definida en la práctica 8 y deberá contener los siguientes campos (los campos *C, T, P, id* y *TCritico* son los mismos que los de la práctica 8):

- C → Tiempo de ejecución en milisegundos (tipo int)
- T → Periodo de repetición en milisegundos (tipo int)
- P → Prioridad de la hebra (tipo int)
- id → Identificador de hebra (tipo int)
- Ta → Tiempo de acceso de la tarea (tipo int). Indicará el instante de comienzo de la tarea a contar a partir del instante crítico.
- acciones → Lista de acciones que llevará a cabo la tarea (tipo int *). Es un vector de enteros que contendrá una lista tiempos de ejecución, los cuales corresponden a los tiempos que tardan en ejecutarse las diferentes acciones de la tarea.
- recursos → Lista con los tipos de recursos que se utilizarán en cada una de las acciones de la tarea (tipo int*). Para cada uno de los tiempos de ejecución contenidos en acciones, en este vector tenemos el recurso que utilizará dicha acción. Los posibles valores son -1 para recursos no compartidos y de 0 a K-1 para recursos compartidos, siendo K el número de recursos compartidos. El par (recursos[i], acciones[i]) indica a la tarea qué recurso debe utilizar y durante cuánto tiempo.
- num_acc → Número de acciones y de tipos de recursos que tiene la tarea (tipo int).
- TCritico → instante crítico (tipo timespec). Se utilizará para que todas las tareas comiencen a la vez, evitando así los retardos surgidos de la creación de tareas.

- mutex → Lista de mutex para el control de los recursos compartidos (tipo pthread_mutex_t *). Este vector contendrá un mutex por cada uno de los recursos compartidos de la aplicación, de forma que cada recurso compartido tiene asociado un mutex. Habrá, por tanto, K mutex. Este vector es compartido por todos los marcos temporales.

Función EjecutarRecurso

Esta función se va a encargar de ejecutar un recurso durante un tiempo de ejecución concreto. Para ello, recibe por parámetros la lista de mutex, el recurso que se va a ejecutar (-1 para recursos no compartidos y de 0 a K-1 para recursos compartidos), el tiempo de ejecución del recurso y el identificador de la tarea que realiza la ejecución. El pseudocódigo de la función es el siguiente:

- Si el recurso es un recurso compartido
  - Bloquear el mutex asociado al recurso
- Imprimir el mensaje "Hilo id comienza la ejecución del recurso x que requiere y ms."
- Ejecutar la función *spin_m* con el tiempo de ejecución del recurso
- Mostrar el mensaje "Hilo id termina con el recurso x."
- Si el recurso es un recurso compartido
  - Desbloquear el mutex asociado al recurso

Tarea periódica genérica

Se modificará la tarea periódica de la práctica 8 de la siguiente forma:

- Establecer la prioridad y la política (SCHED_FIFO para todas las hebras).
- Antes de iniciar la acción periódica, se hará una espera hasta el valor indicado por el campo *TCritico* del marco temporal (parámetro de tipo *marco_temp_t* pasado a la hebra) más el tiempo de acceso de la tarea (campo *Ta del marco temporal*). De esta forma, el instante de comienzo de la hebra será *TCrítico+Ta*.
- Se calcula el valor del primer instante de espera, que en la práctica 8 era *TCritico+P* y en este caso será *TCrítico+Ta+P*.
- La acción periódica que realiza la hebra es la siguiente:
  - Muestra el mensaje "Hilo *id* con prioridad *P* inicia la acción periódica\n".
  - Para cada acción del marco temporal
    - Ejecutar la función *EjecutarRecurso* con la lista de mutex, el recurso i-ésimo, el tiempo de ejecución i-ésimo y el identificador, todos ellos campos del marco temporal de la tarea.
  - Muestra el mensaje "Hilo *id* termina la acción periódica\n".
  - Espera hasta el siguiente instante de ejecución.
  - Calcula el siguiente instante de ejecución de la tarea periódica en función del último instante de ejecución y T.

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 crítico (lo recibe por parámetro), su acción periódica será:

- Mostrar el mensaje "********** *x* ms **********" (donde x variará de 10 en 10).
- Esperar al siguiente instante de ejecución
- Calcula el siguiente instante de ejecución de la tarea periódica en función del último instante de ejecución y 10 ms.

Esto se repetirá hasta que se hayan ejecutado 2 segundos (200 iteraciones de 10 ms).

El programa principal

Las funciones del programa principal serán:

- Establecer la prioridad de la tarea encargada de mostrar los tiempos (max FIFO) de la misma forma a como se hacía en la práctica 8.
- Crear e inicializar un vector *marcos* de tipo *marco_temp_t* con un tamaño de X, siendo X el número de tareas. Todos los valores del vector se rellenarán a partir de un fichero de entrada, que cargará el número de tareas y rellenará el vector con todos los valores del marco temporal (excepto el instante crítico y la lista de mutex). Esta parte está ya implementada en el esqueleto de la práctica. El formato del fichero es el siguiente:

  Número_de_tareas
  K techo_prio$_1$ … techo_prio$_K$
  C_1 T_1 Ta_1 P_1 $NumAcc_1$ $R_{1,1}$_$TE_{1,1}$ … $R_{1,NumAcc0}$_$TE_{1,NumAcc0}$
  …
  C_N T_N Ta_N P_N $NumAcc_N$ $R_{N,1}$_$TE_{N,1}$ … $R_{N,NumAccN}$_$TE_{N,NumAccN}$

  siendo K el número de recursos compartidos, techo_prio$_i$ el techo de prioridad del recurso i, C_i el tiempo de ejecución de la tarea i, T_i el periodo de la tarea i, Ta_i el tiempo de acceso de la tarea i, P_i la prioridad de la tarea i, $NumAcc_i$ el número de acciones que tiene la tarea i, $R_{i,j}$ el recurso usado en la acción j de la tarea i y $TE_{i,j}$ el tiempo de ejecución de la acción j de la tarea i. Para recursos no compartidos se usarán los caracteres 'n' o 'N' y para recursos compartidos los valores entre 0 y K-1.
- Crear el vector de mutex, inicializando cada uno de los elementos del vector con el protocolo de techo de prioridad inmediato y el techo de prioridad correspondiente al recurso asociado a dicho mutex. Una vez creado el vector e inicializado todos sus componentes, asignarlo a todos los marcos temporales (la lista de mutex es compartida por todas las hebras).
- Calcular el instante crítico (hora actual más dos segundos) y asignarlo a todos los elementos del vector *marcos*.
- Lanzar todas las hebras.
- Esperar únicamente por la hebra que muestra los tiempos.

- Liberar la memoria de todos los componentes utilizados.

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

Ver el anexo para la descripción de las funciones sobre prioridades en POSIX.

Anexo - Funciones POSIX para prioridad de hebras

- **pthread_attr_init (pthread_attr_t *attr)** → Inicializa los atributos de una hebra a sus valores por defecto.
- **pthread_attr_destroy (pthread_attr_t *attr)** → Destruye attr.
- **pthread_attr_setinheritsched (pthread_attr_t *attr, int inherit)** → Establece, en los atributos de creación de una hebra, si ésta hereda o no la política/prioridad del padre. Por defecto tiene un valor PTHREAD_INHERIT_SCHED (hereda), para establecer la prioridad de un hilo, hay que cambiar este campo a PTHREAD_EXPLICIT_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. sched_param está definida de la siguiente forma:

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

  El campo sched_priority será el valor de prioridad de los atributos de creación de la hebra.
- **pthread_setschedparam (pthread_t thread, int policy, struct sched_param *param)** → Establece los parámetros (param) y la política (policy) de la hebra thread.
- **pthread_self()** → Devuelve el identificador de la hebra actual.
- **pthread_mutexattr_init (pthread_mutexattr_t *attr)** → Crea e inicializa los atributos de creación de mutex attr a sus valores por defecto.
- **pthread_mutexattr_destroy (pthread_mutexattr_t *attr)** → Destruye attr.
- **pthread_mutexattr_setprotocol (pthread_mutex_attr_t *attr, int protocol)** → Establece el protocolo de prioridades de attr. Dicho protocolo puede ser PTHREAD_PRIO_INHERIT (Herencia de prioridad), PTHREAD_PRIO_PROTECT (Techo de prioridad inmediato), PTHREAD_PRIO_NONE (Prioridades estáticas).
- **pthread_mutexattr_setprioceiling (pthread_mutexattr_t *attr, int prioceiling)** → Establece prioceiling como techo de prioridad de attr.
- **pthread_mutex_init** (pthread_mutex_t *mutex, pthread_mutexattr_t *attr) → Inicializa mutex con los atributos establecidos en attr.
- **pthread_mutex_destroy (pthread_mutex_t *mutex)** → Destruye mutex.

Las funciones *pthread_attr_xxxxx* se invocan antes de la función *pthread_create*, ya que ésta última utilizará los atributos modificados por las primeras. Las otras funciones se usarán dentro de la hebra que las necesite. De igual forma, las funciones *pthread_mutexattr_xxxxx* se invocan antes de *pthread_mutex_init* ya que ésta utilizará los atributos creados por las anteriores.

Para establecer la política y la prioridad dentro de una hebra en particular seguiremos la siguiente secuencia:

1. Crear una variable de tipo *sched_param*.
2. Asignar al campo *priority* de la variable definida en 1 la prioridad deseada.
3. Establecemos tanto la prioridad como la política de la hebra usando la función *pthread_setschedparam* (para el identificador de la hebra a modificar, usar *pthread_self()*)