Práctica 11 Algoritmos de Planificación

# Algoritmos de planificación

Un sistema de tiempo real se caracteriza por la fuerte interacción que mantiene con el entorno que le rodea, debiendo responder a diferentes eventos generados por éste en unos plazos de tiempo preestablecidos. La naturaleza compleja y concurrente del entorno, conduce a la utilización de arquitecturas de software concurrente para los sistemas de tiempo real. En este tipo de arquitecturas, el sistema global se divide en un conjunto de actividades concurrentes, denominadas tareas, cada una de ellas encargada de responder a un determinado evento o conjunto de eventos generados por el entorno. El software producido siguiendo este modelo será más próximo al sistema real, resultando más sencillo y comprensible.

Dado que el número máximo de tareas que es posible ejecutar simultáneamente en una computadora es limitado (como máximo igual al número de procesadores), es necesario definir un conjunto de reglas que permitan determinar qué tarea o tareas deben ser ejecutadas en cada momento. Estas reglas constituyen los denominados algoritmos o políticas de planificación, y de su elección depende en gran medida el que se satisfagan o no las restricciones temporales impuestas al sistema.

Los sistemas operativos de tiempo real utilizan algoritmos de planificación en tiempo de ejecución basados en prioridades, en los que los problemas de contención de recursos se resuelven en el instante que se producen, eliminándose la necesidad de que exista un plan de ejecución previo. En dichos algoritmos, a cada tarea le es asignada una prioridad y, en función de ella, se resuelven los posibles conflictos de utilización del procesador y de los demás recursos del sistema.

Dependiendo de si la prioridad de las tareas es constante o cambia en función del estado del sistema los algoritmos de planificación en tiempo de ejecución se dice que están basados en prioridades estáticas o dinámicas. Además, sea cual sea el tipo de prioridades, puede diferenciarse entre políticas expulsivas y no expulsivas: en una política de planificación no expulsiva la tarea en posesión del procesador no verá detenida su ejecución hasta que así lo desee o bien hasta que se bloquee al tratar de acceder a algún recurso no disponible en ese momento. Por el contrario, en una política expulsiva una tarea perderá la posesión del procesador en el momento en que aparezca otra tarea de mayor prioridad lista para ejecutar. El planificador no expulsivo es más sencillo de implementar, aunque a cambio, presenta la importante desventaja de que su uso puede provocar retrasos importantes en la ejecución de las tareas más prioritarias.

En los sistemas inmersos las prioridades estáticas son mucho más utilizadas que las dinámicas. Ello se debe a: que la sobrecarga causada por el algoritmo suele ser menor para este tipo de políticas, que su base teórica fue desarrollada con anterioridad y, a que son estables ante sobrecargas del sistema. Por otro lado, los algoritmos de planificación basados en prioridades dinámicas presentan dos importantes ventajas: aprovechan al máximo la potencia del procesador, de forma que con ellos son panificables conjuntos de tareas que no lo serían de haber utilizado otros algoritmos (en particular los basados en prioridades estáticas), y se adaptan muy bien a entornos altamente dinámicos en los que la carga del sistema no puede ser conocida de antemano. Estas ventajas, junto con el aumento de la potencia de los procesadores y el desarrollo de la base teórica necesaria, permiten que en la actualidad pueda resultar práctica y factible la utilización de este tipo de algoritmos en sistemas inmersos.

El más importante de los algoritmos basados en prioridades dinámicas es el EDF (“Earliest Deadline First”) [Liu, Layland], en el que la prioridad de una tarea es tanto mayor cuanto más cercano se encuentre el instante en que finaliza su plazo de ejecución, o lo que es lo mismo, cuanto más cerca se encuentre de incumplir los requisitos temporales marcados para ella. El plazo de ejecución de una tarea no es un valor constante: cuando la tarea finaliza el trabajo correspondiente a una activación, su plazo pasa a ser el referido a la activación siguiente. Con esta política una tarea se mantendrá en ejecución hasta que se bloquee, finalice el trabajo correspondiente a la presente activación o sea expulsada como consecuencia de la activación de otra tarea con un plazo de finalización más cercano.

## Ejemplo de una política definida en aplicación

El siguiente ejemplo muestra el código de un conjunto de hilos periódicos (tareas periódicas) planificados bajo una política de planificación EDF (archivos examples/appsched/edf\_sched.c y examples/appsched/edf\_threads.c).

Las tareas pueden estar en uno de los siguientes tres estados: ACTIVE, cuando se encuentra ejecutando o en disposición de hacerlo; BLOCKED, cuando su ejecución ha sido detenida porque ha tratado de acceder a algún recurso no disponible en ese momento; TIMED, cuando ha terminado su ejecución actual y está esperando por su próximo periodo.

typedef enum { ACTIVE, BLOCKED, TIMED } th\_state\_t;

El planificador EDF usa una estructura autoreferenciada para manejar los datos de las tareas

typedef struct thread\_data {

struct thread\_data \*next;

th\_state\_t th\_state;

struct timespec period;

struct timespec next\_deadline; /\* absolute time \*/

int id;

timer\_t timer\_id;

pthread\_t thread\_id;

} thread\_data\_t;

El planificador, también tiene una lista de hilos que son registrados para ser planificados por él.

list\_t RQ = NULL;

Una variable entera inicializada a 1, para asignar diferentes identificadores a cada hilo.

int threads\_count = 1;

Un apuntador a thread\_data\_t para indicar el hilo actualmente elegido para ejecutar.

thread\_data\_t \*current\_thread = NULL;

El planificador usa las siguientes operaciones que se han definido como parte de esta implementación EDF:

La función schedule\_next() usa la lista de hilos registrados para obtener el hilo más urgente (el hilo/tarea con plazo de finalización más próximo), si la lista RQ no estaba vacía y el hilo más urgente es distinto del hilo actual (el hilo en ejecución actualmente), imprime “Activate:” seguido del id del hilo más urgente y con la función posix\_appsched\_actions\_addactivate() se cambia el estado de hilo/tarea más urgente a ACTIVE. Si el hilo/tarea current\_thread es distinto de NULL y su estado no es BLOCKED, se imprime “Suspend:” seguido del id del hilo current\_thread; entonces, con la función posix\_appsched\_actions\_addsuspend() se suspende el hilo/tarea a la que apunta current\_thread. Finalmente, se hace current\_thread igual al hilo/tarea más urgente obtenido de la lista RQ.

La función add\_to\_list\_of\_threads() agrega un nuevo hilo o tarea a la lista de hilos a planificar. Con la función pthread\_getappschedparam() se obtienen en la struct edf\_sched\_param param, los parámetros de planificación, se configuran señales relacionadas con el uso de un temporizador; y se asigna un temporizador llamando a la función timer\_create(). Después de algunos ajustes más, relacionados con el temporizador asignado, se imprime el mensaje “ Add new thread ” junto con el id del hilo/tarea agregado y el periodo del mismo.

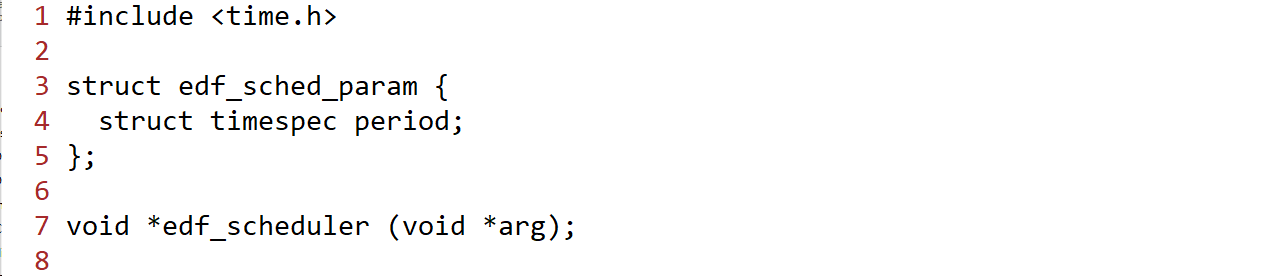
La función eliminate\_from\_list\_of\_threads() elimina un hilo terminado de la lista de hilos a planificar RQ.

La función make\_ready() cambia el estado de un hilo a ACTIVE.

La función make\_blocked() cambia el estado de un hilo a BLOCKED.

La función make\_timed() cambia el estado de un hilo a TIMED.

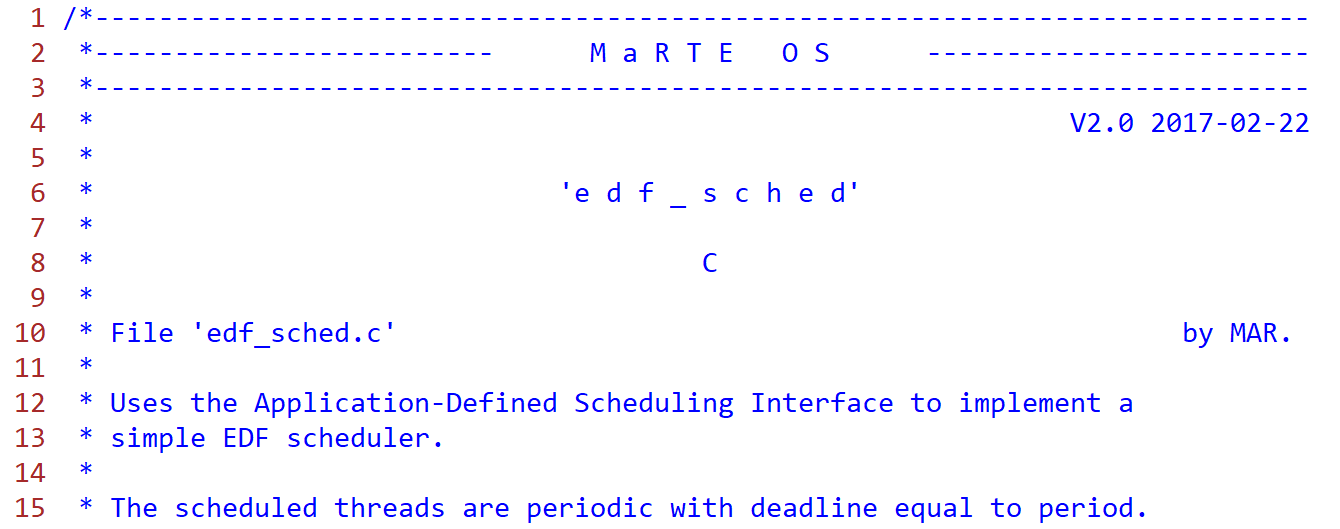
El código del archivo examples/appsched/edf\_sched.h es el siguiente:

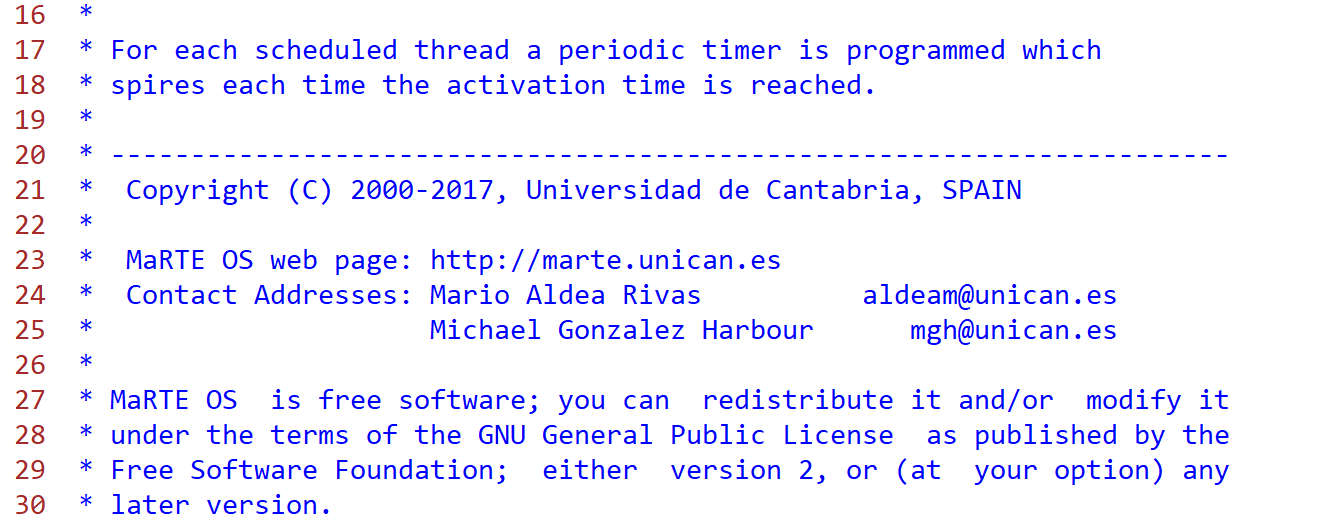


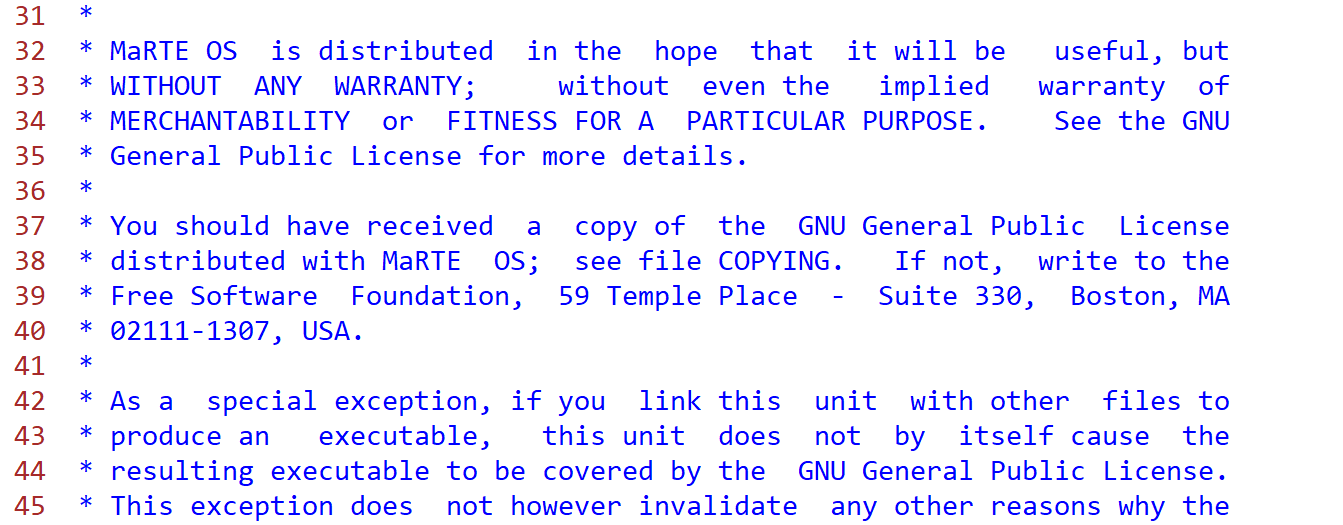
Archivo examples/appsched/edf\_sched.h

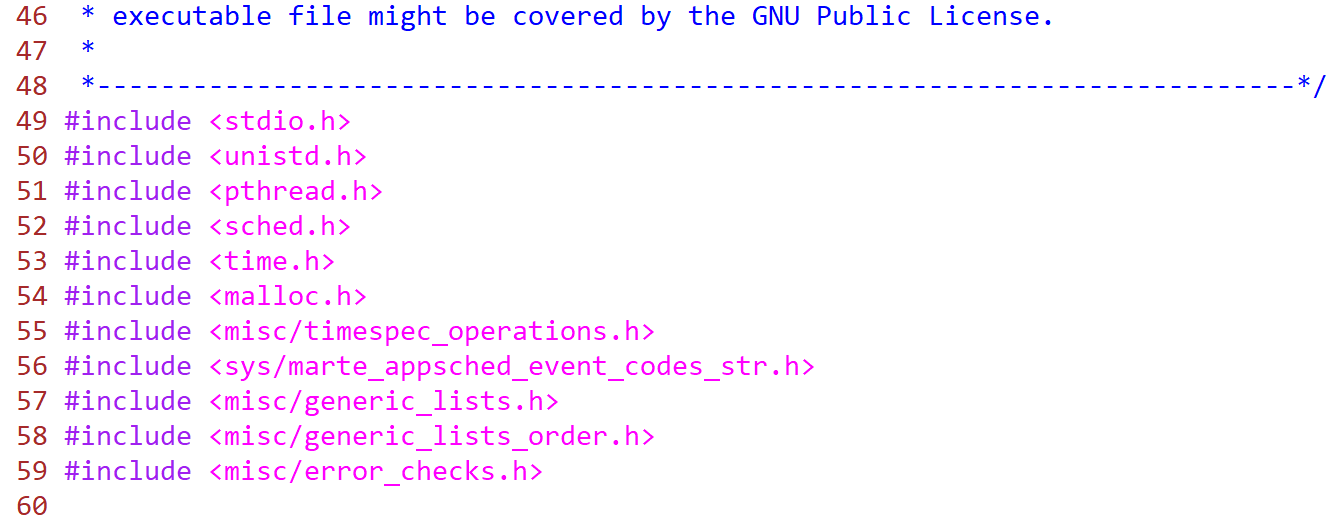
En el archivo examples/appsched/edf\_sched.c se usa la interface (de MaRTE OS) de scheduling definida en aplicación para implementar un scheduler EDF simple. Los hilos planificados son periódicos con plazo de finalización igual al periodo. Para cada hilo planificado se programa un temporizador periódico el cual expira cada vez que se alcanza el tiempo de activación.

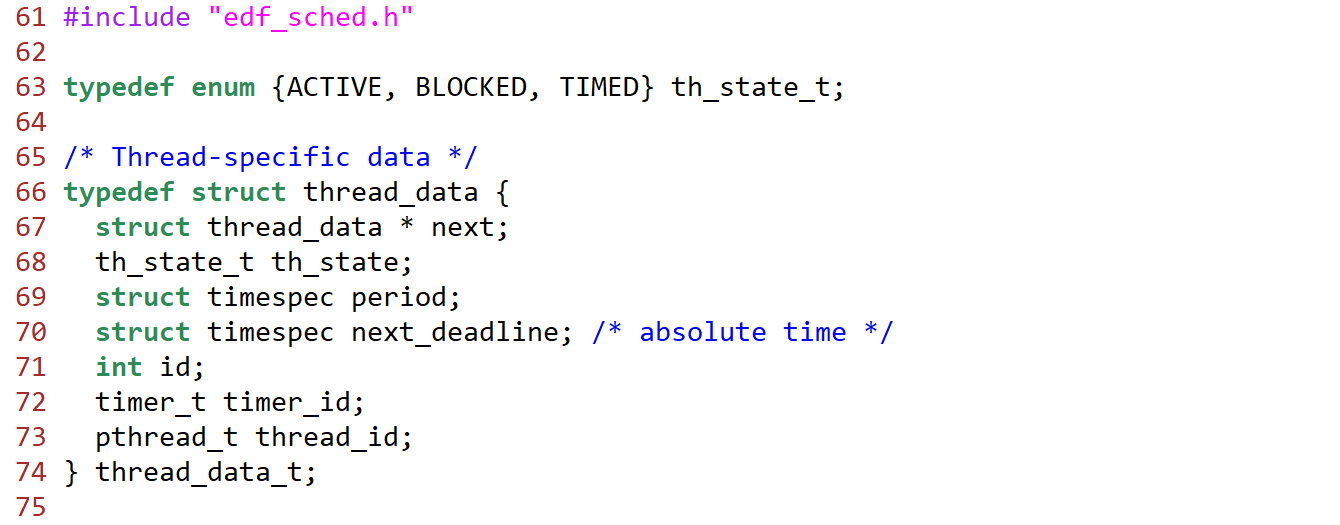
El código del archivo examples/appsched/edf\_sched.c es el siguiente:

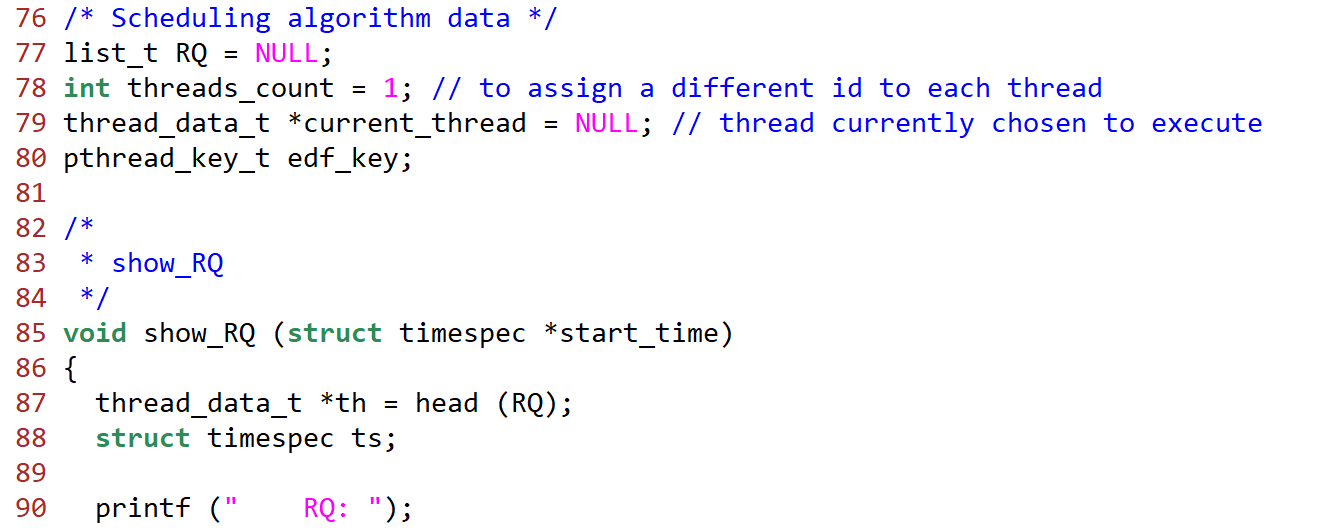


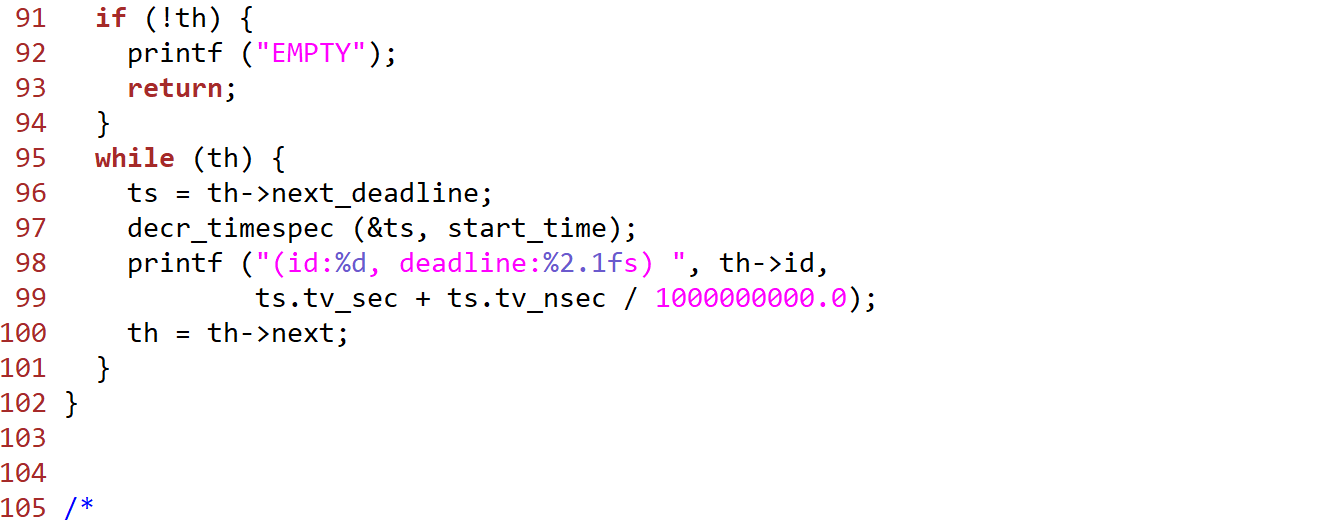


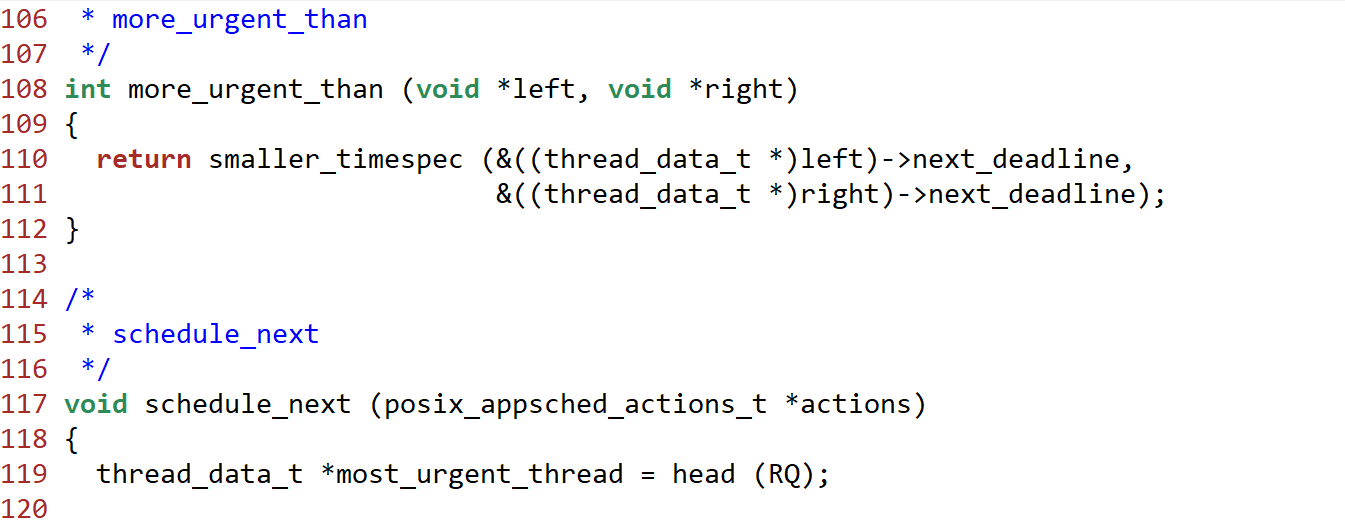


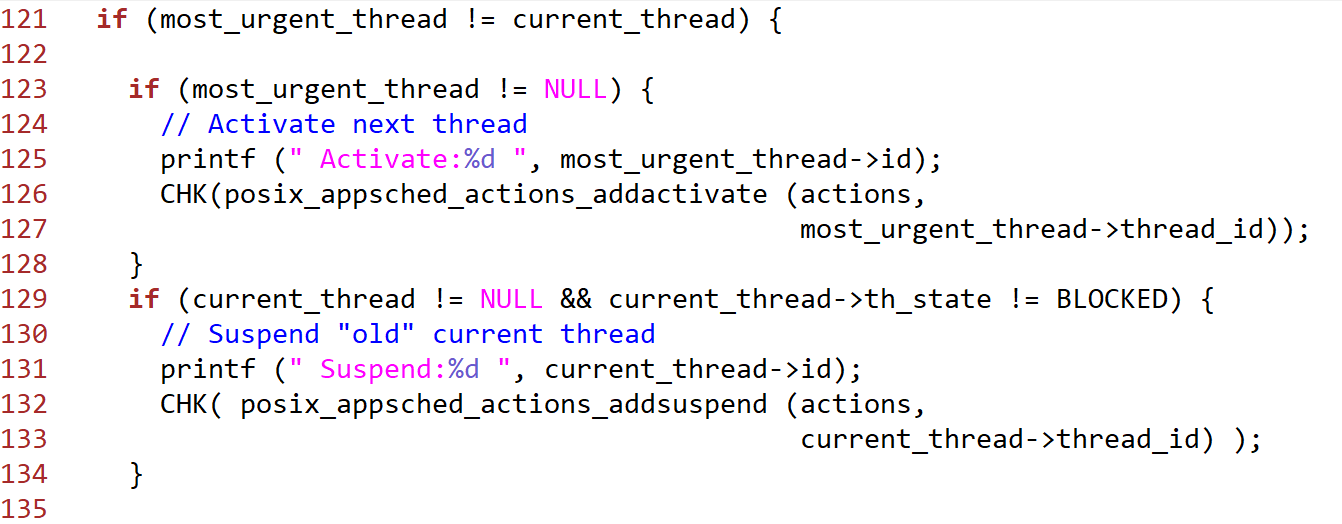


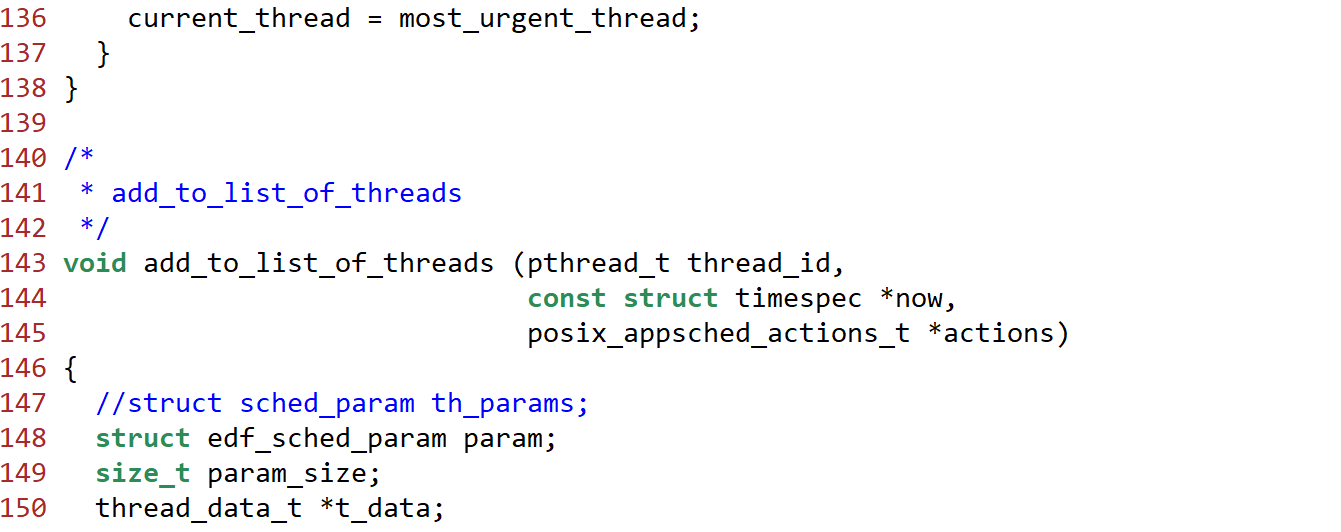


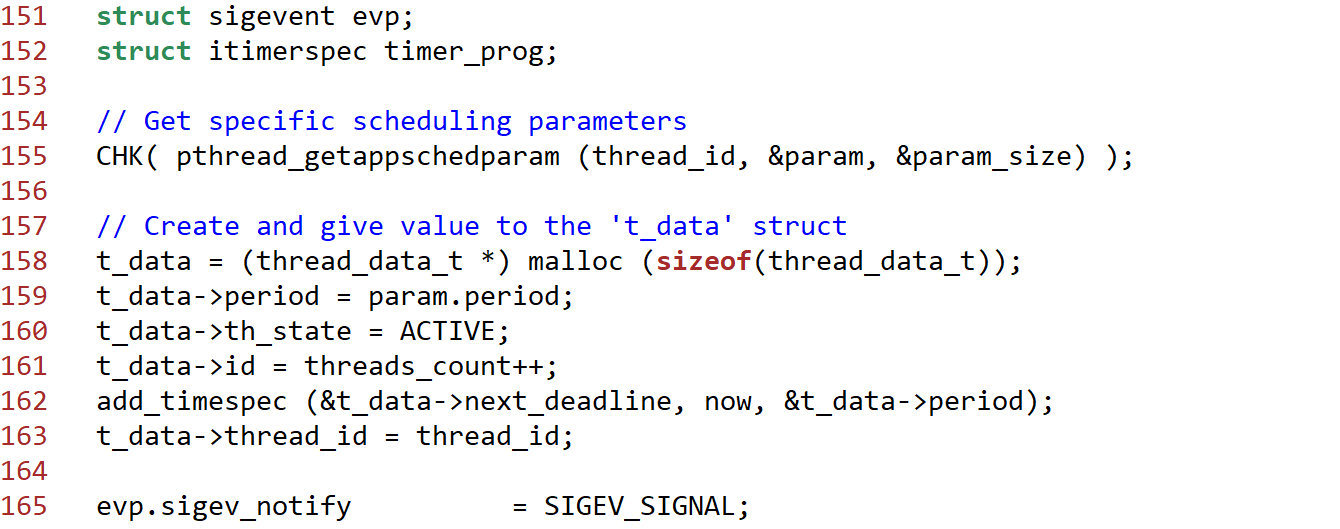


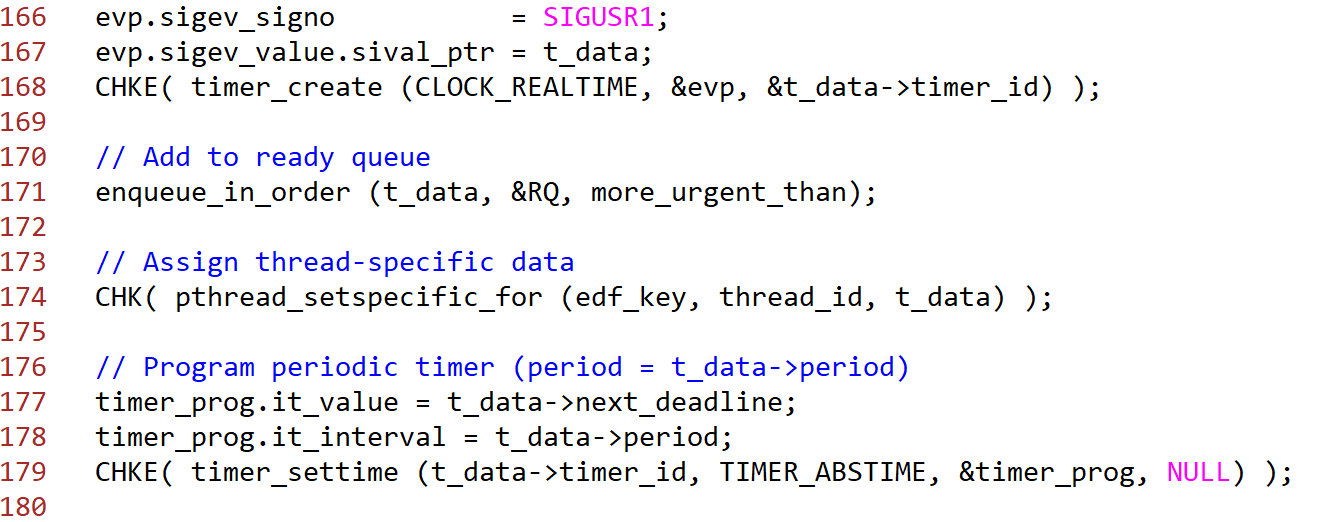


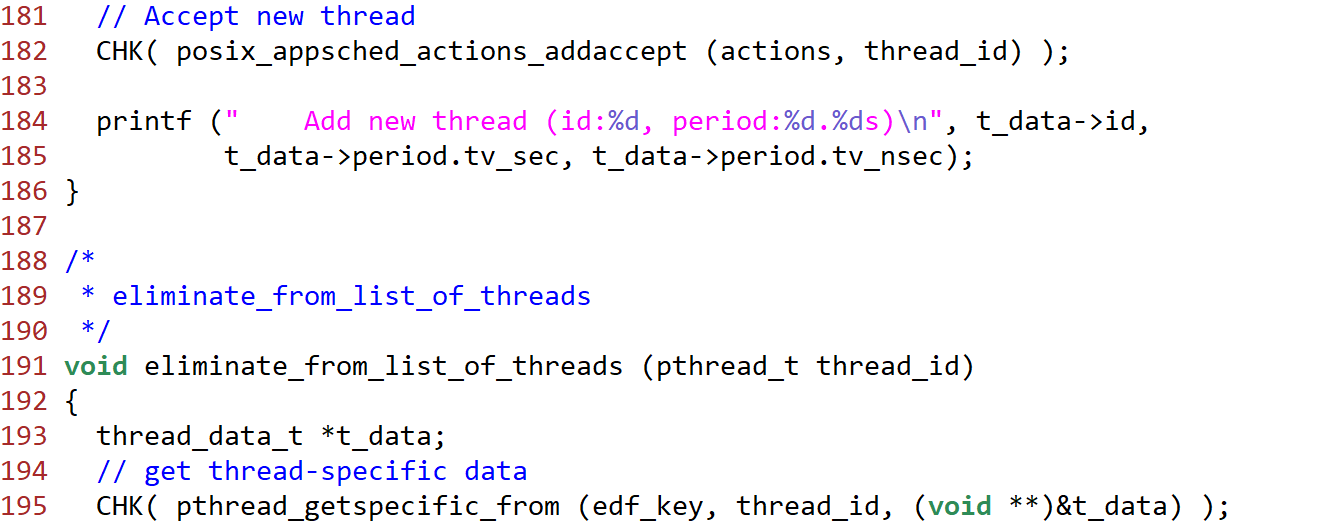


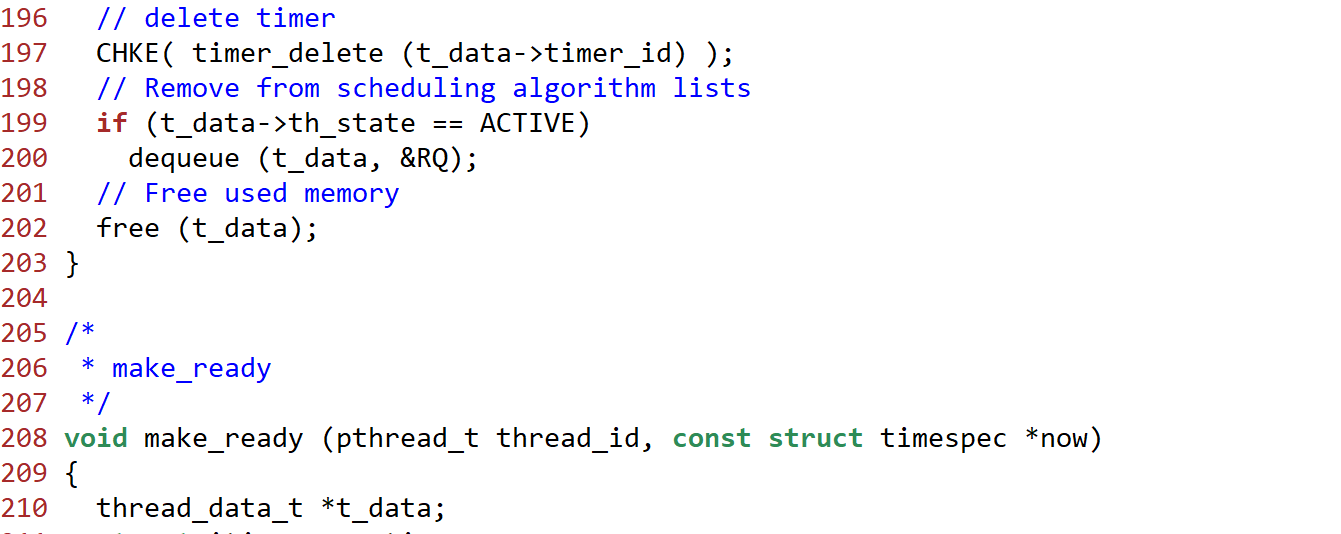


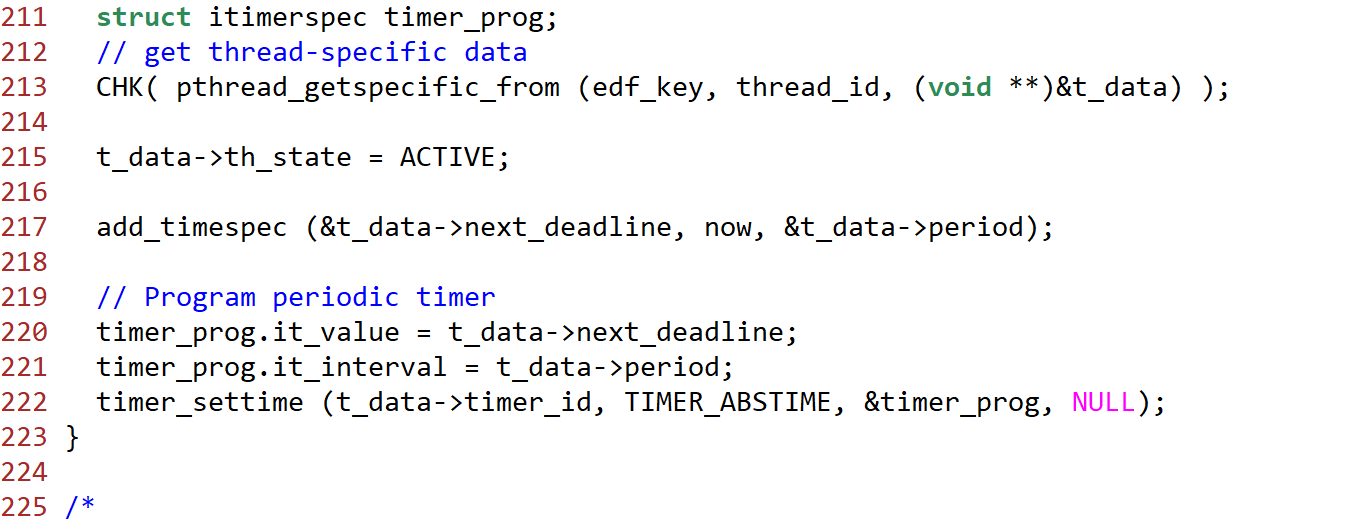


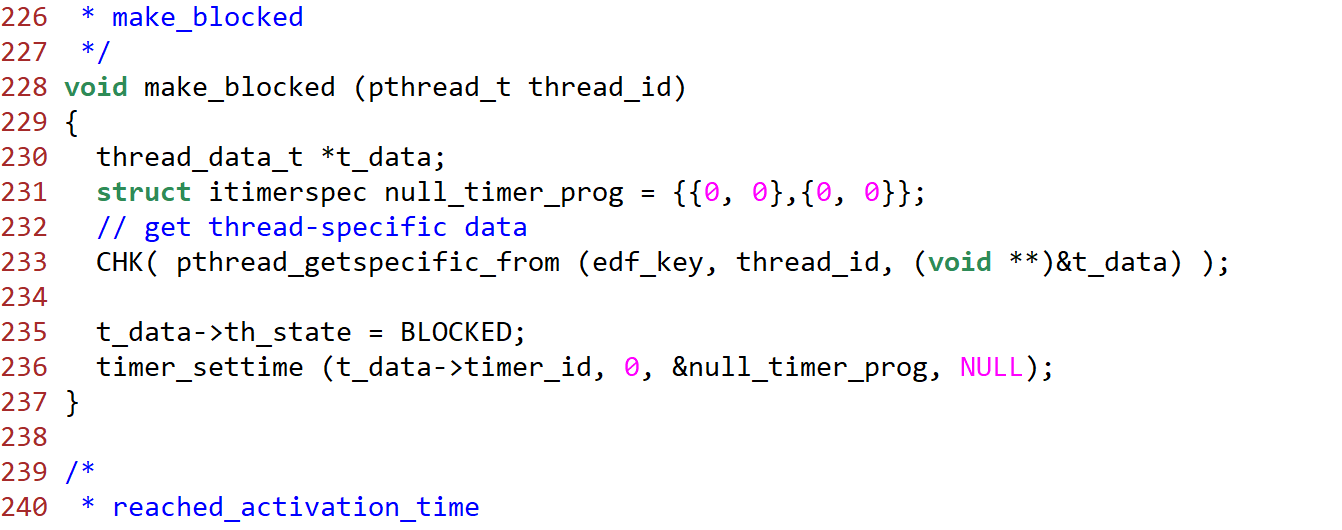


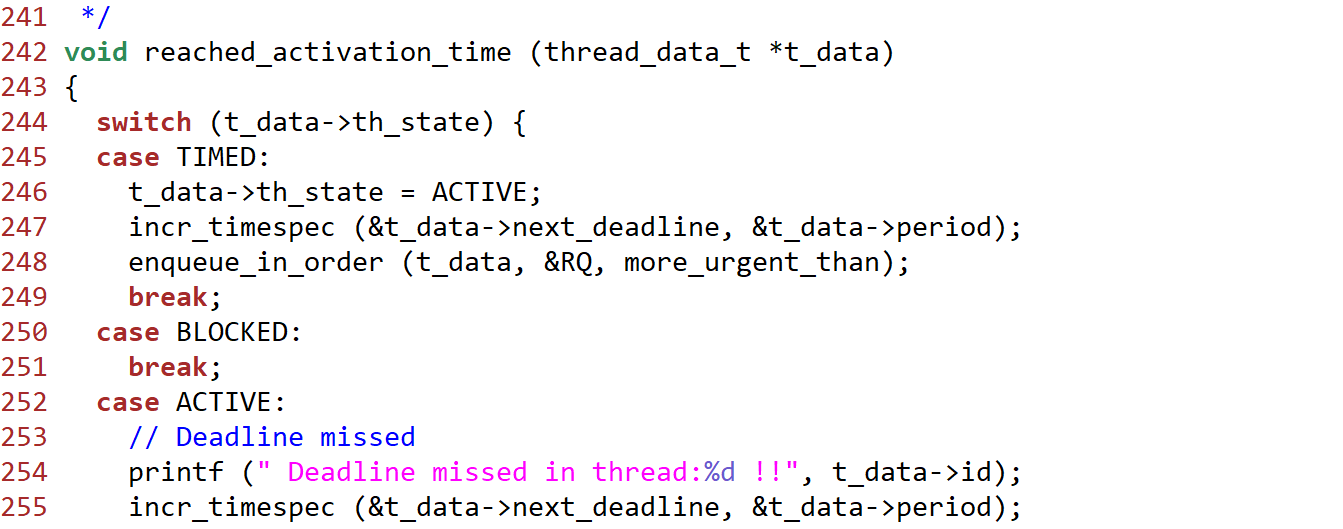


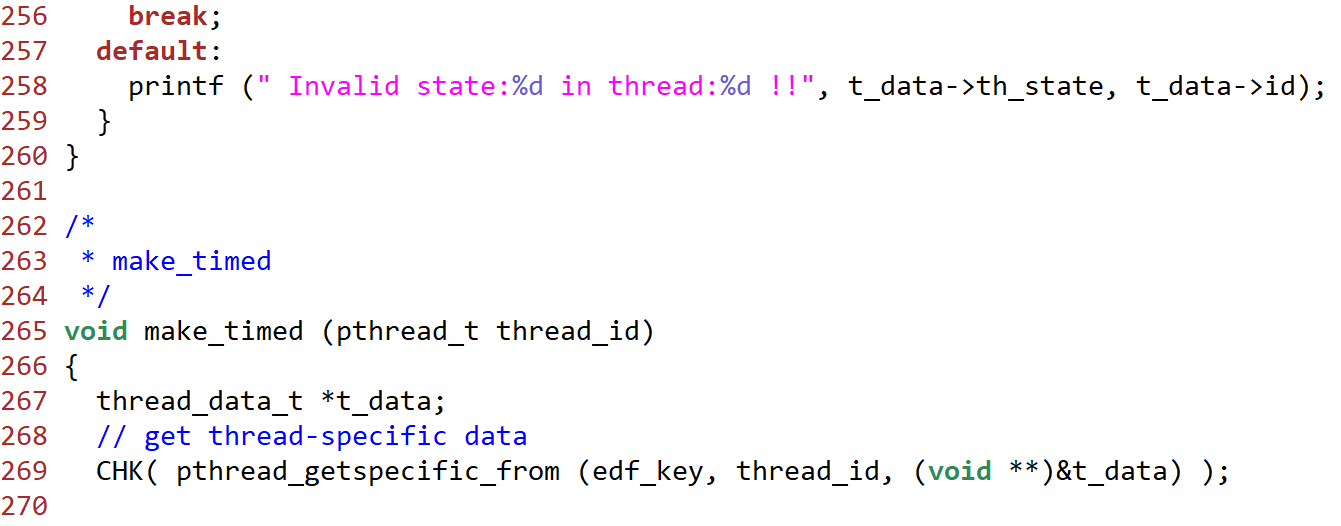


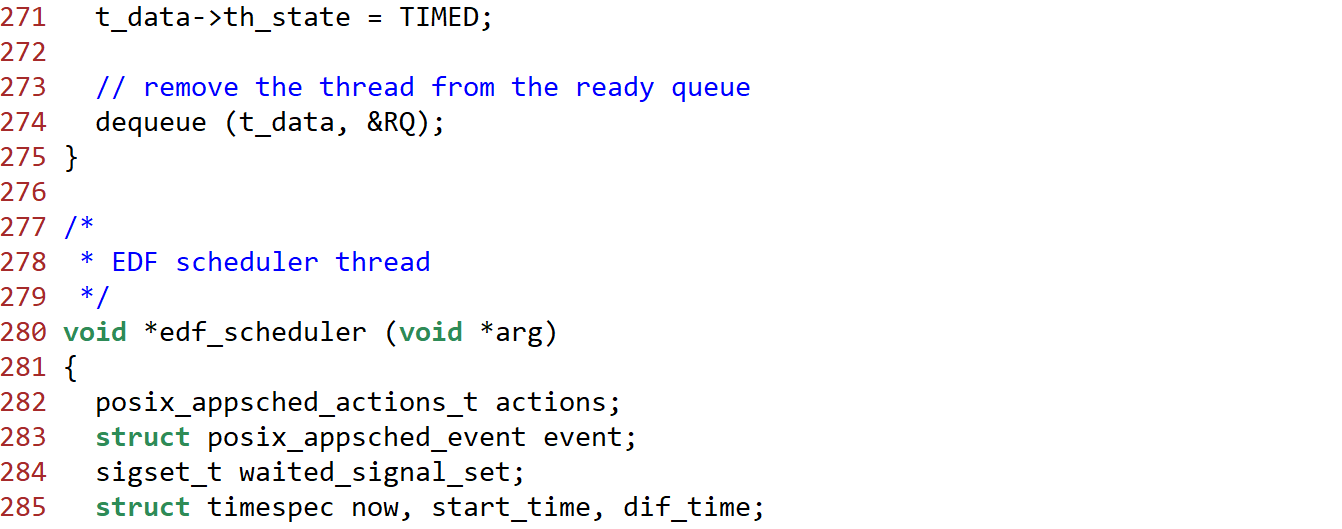


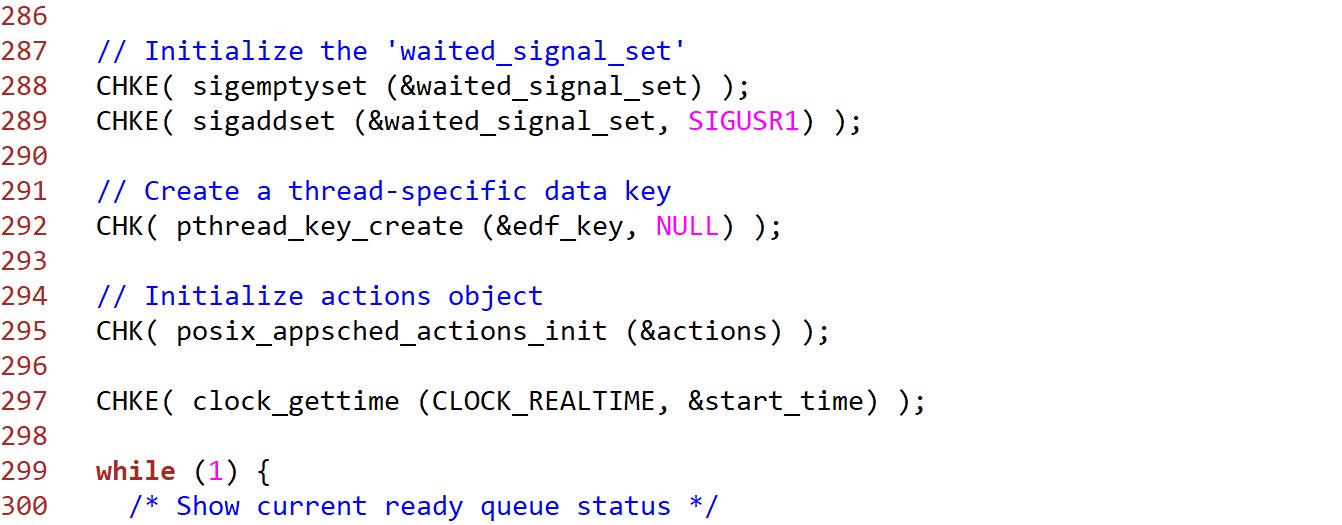


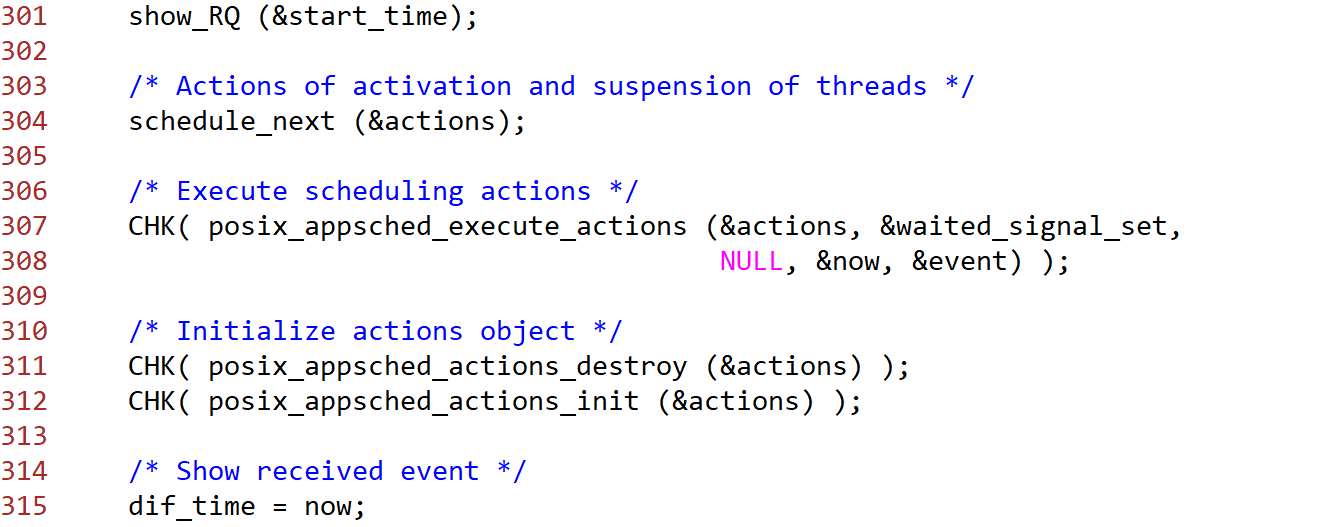


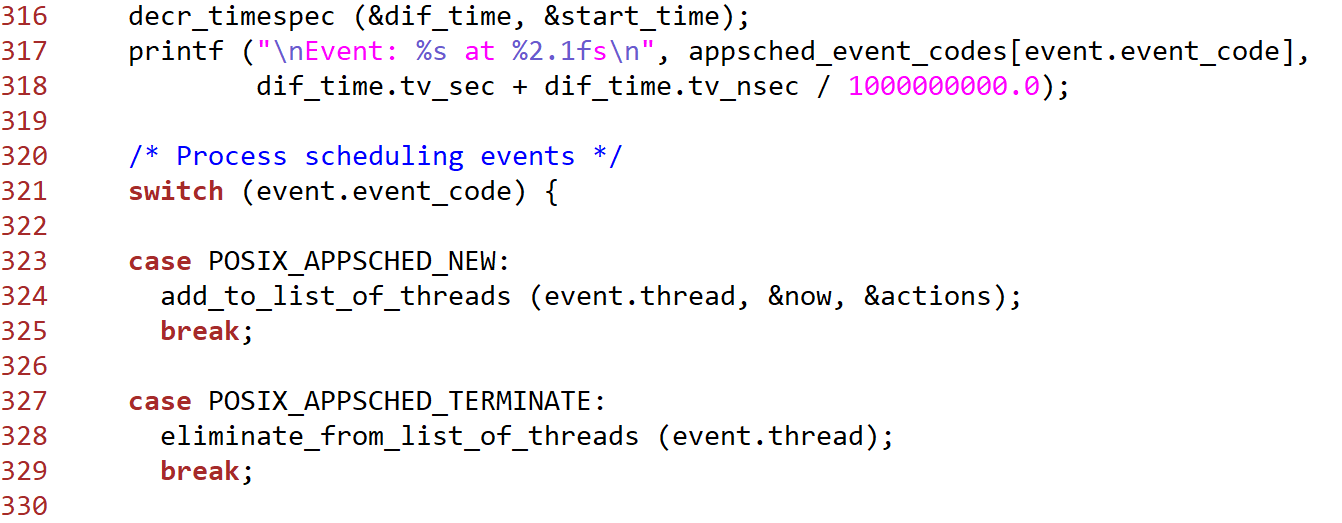


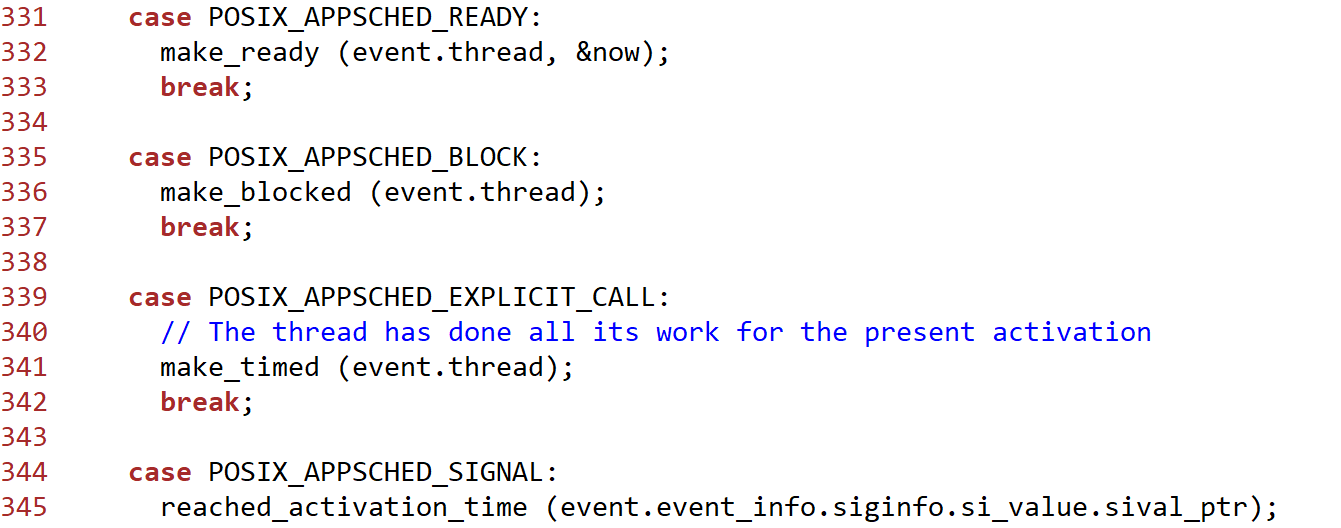








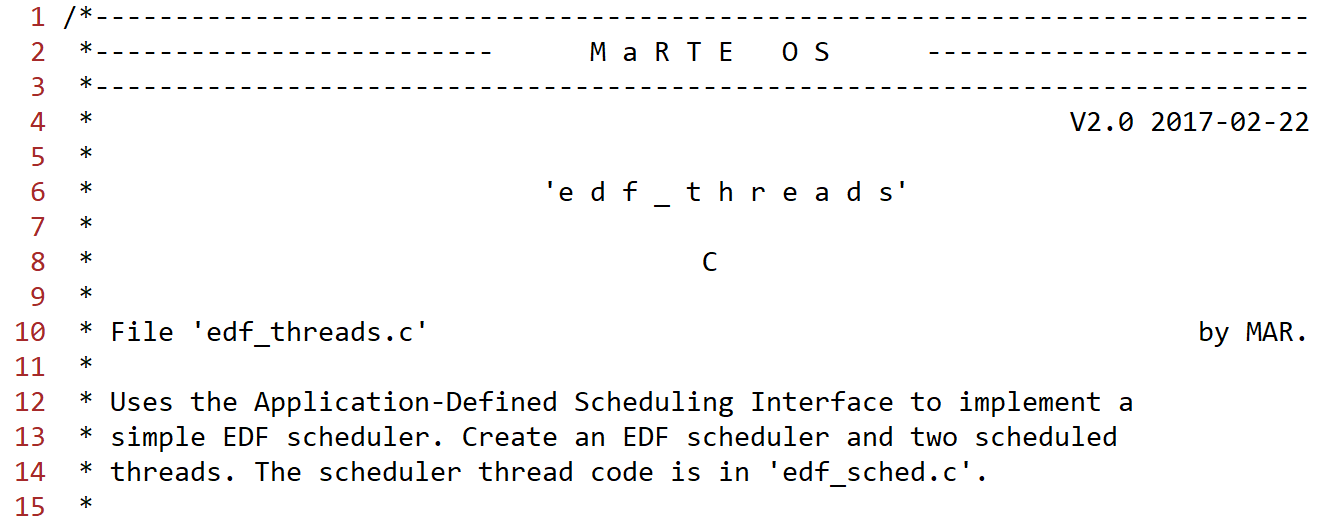


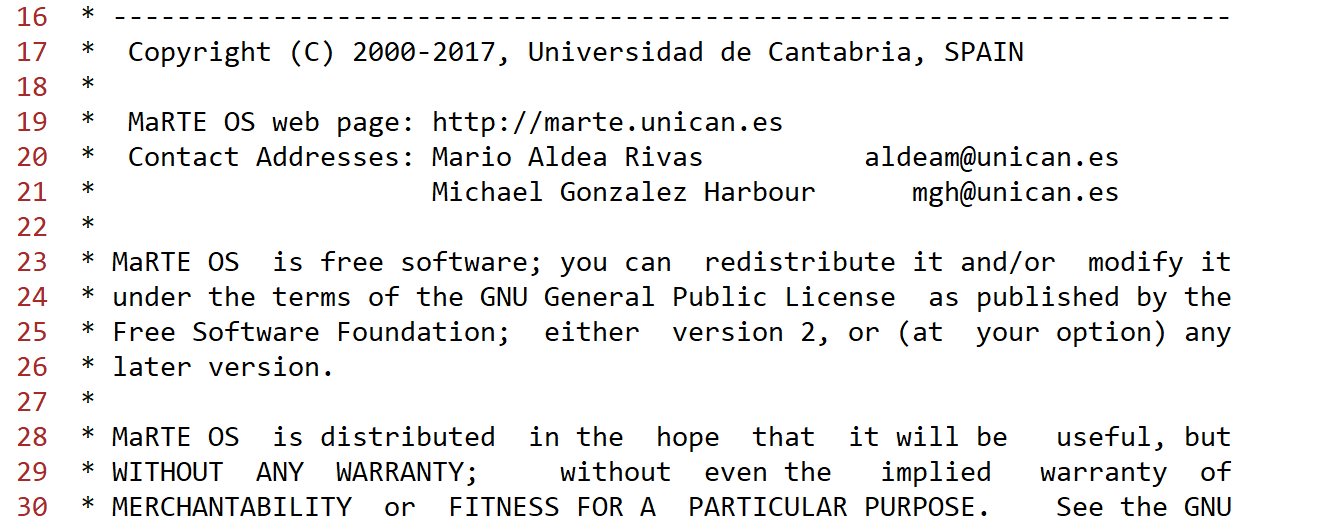


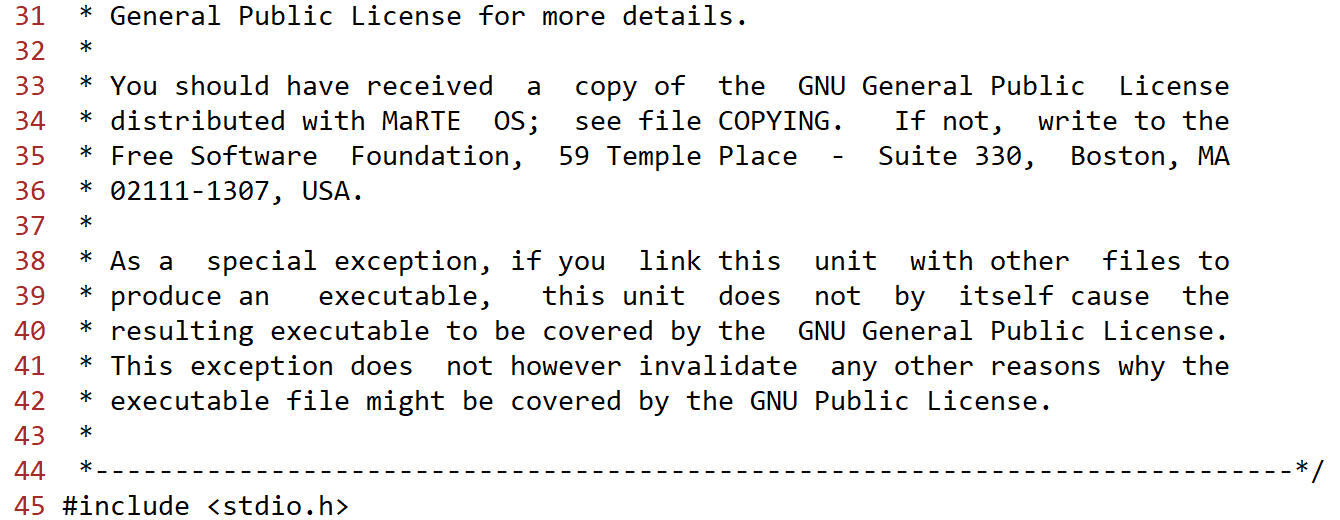


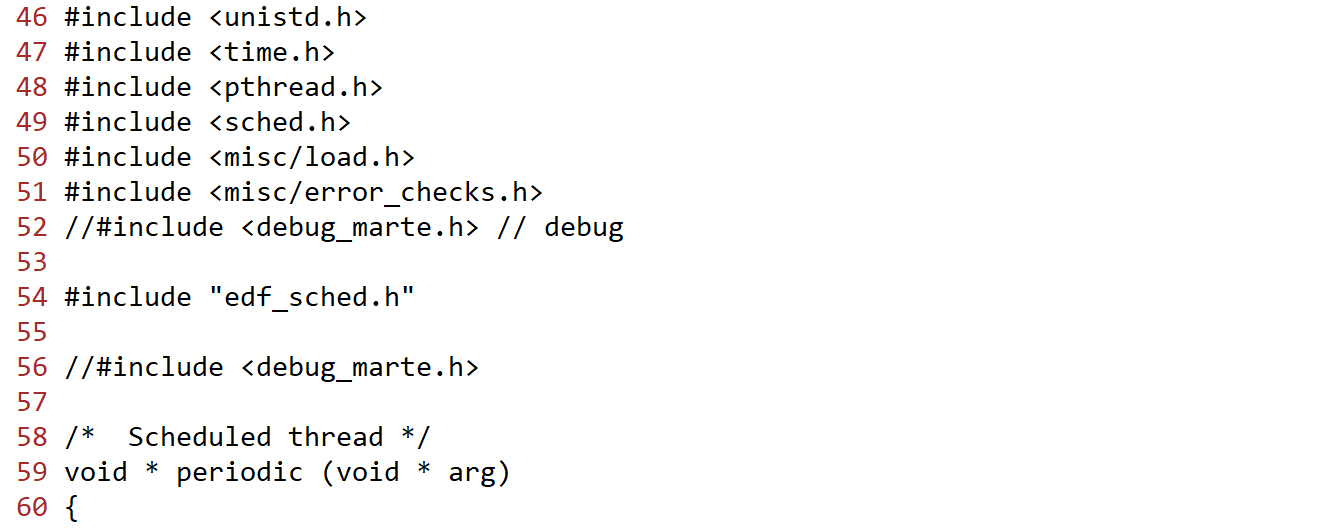
Archivo examples/appsched/edf\_sched.c

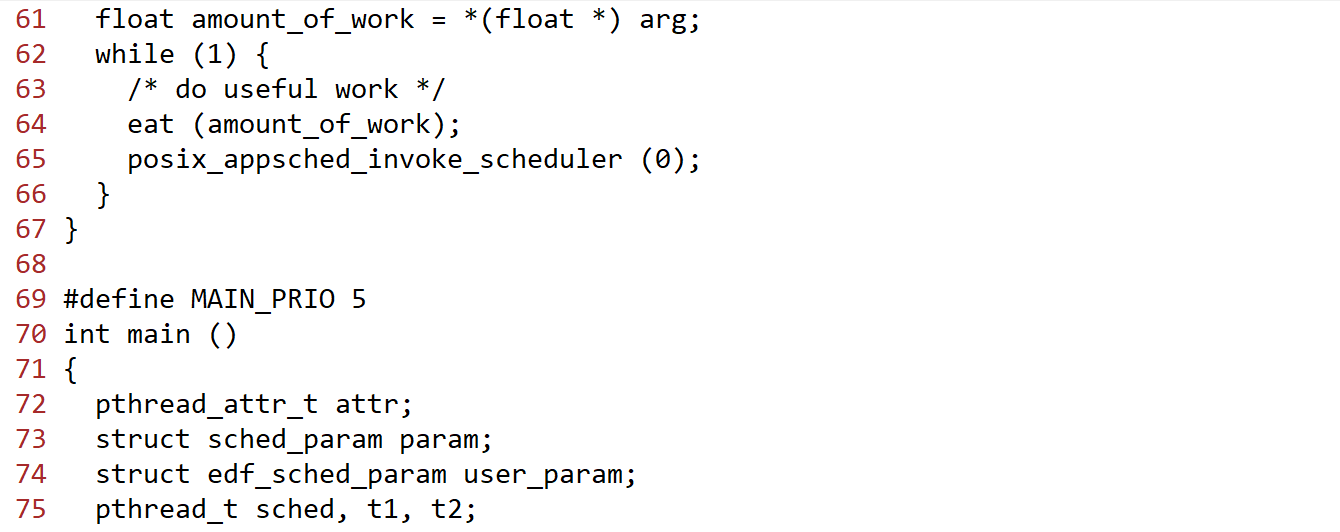
En el archivo examples/appsched/edf\_threads.c se usa la interface de Scheduling definida en Aplicación (de MaRTE OS) para implementar un scheduler EDF simple. Crea un scheduler EDF y dos hilos planificados. El código del hilo scheduler está en ‘edf\_sched.c’.

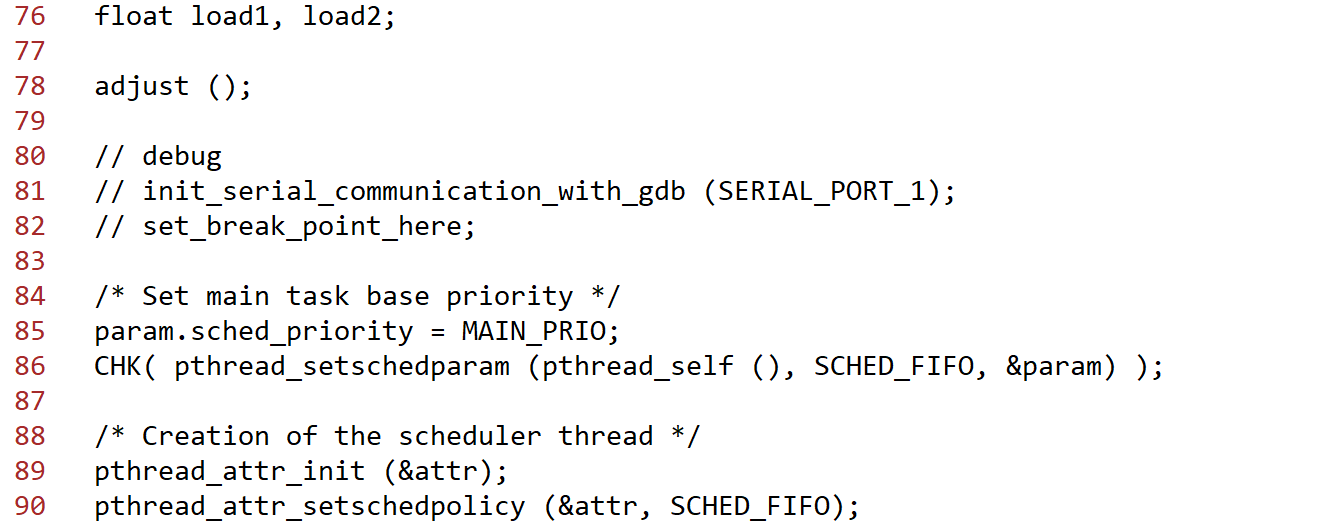


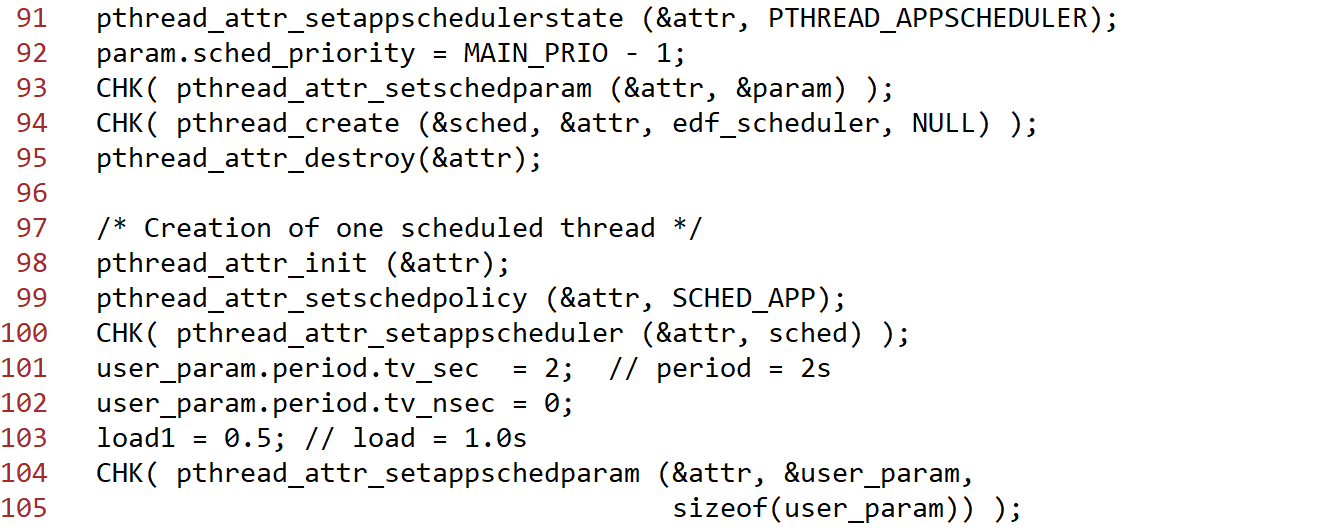


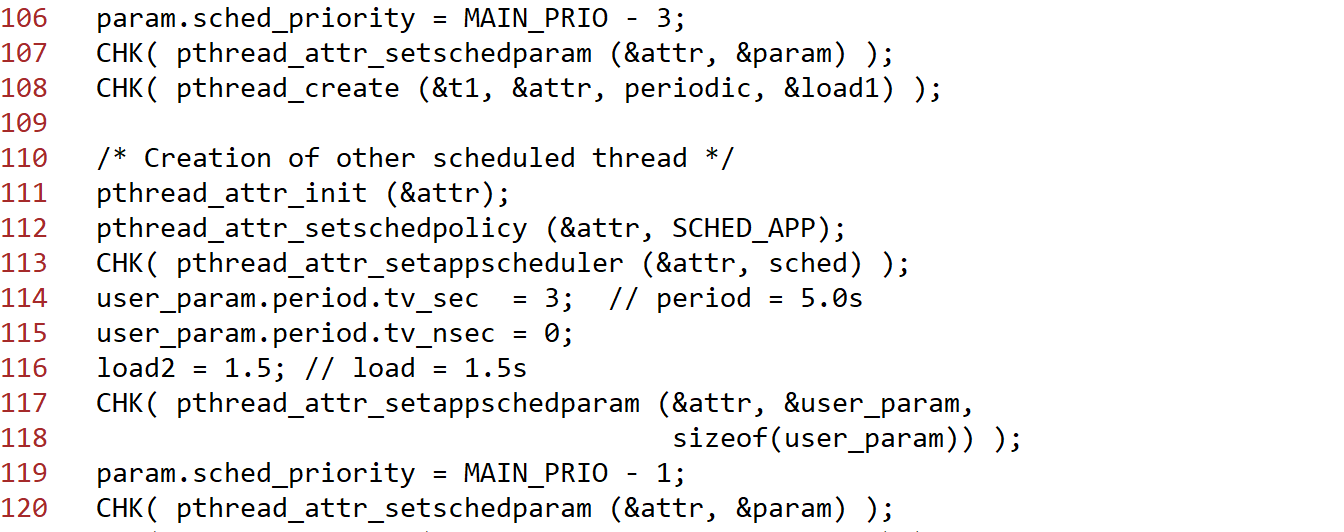


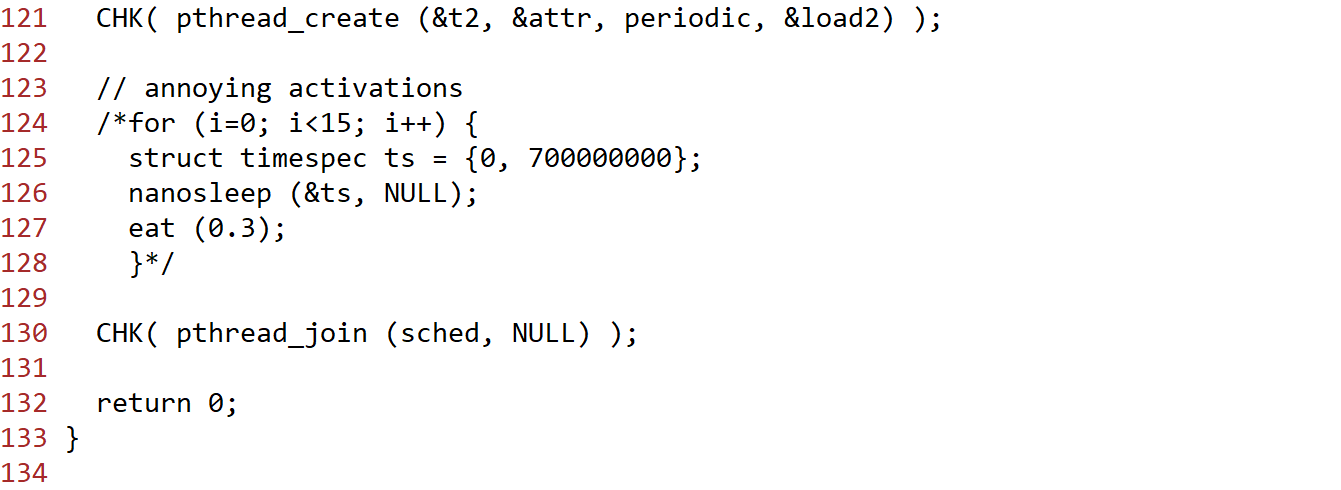






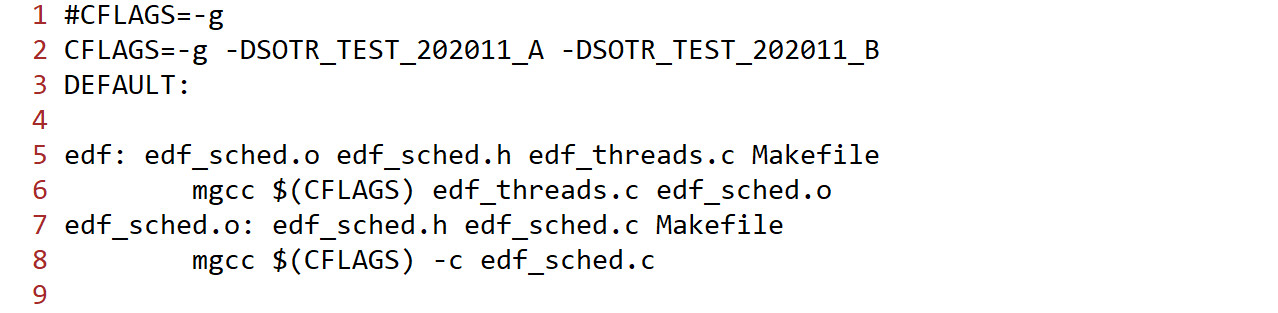






Archivo examples/appsched/edf\_threads.c

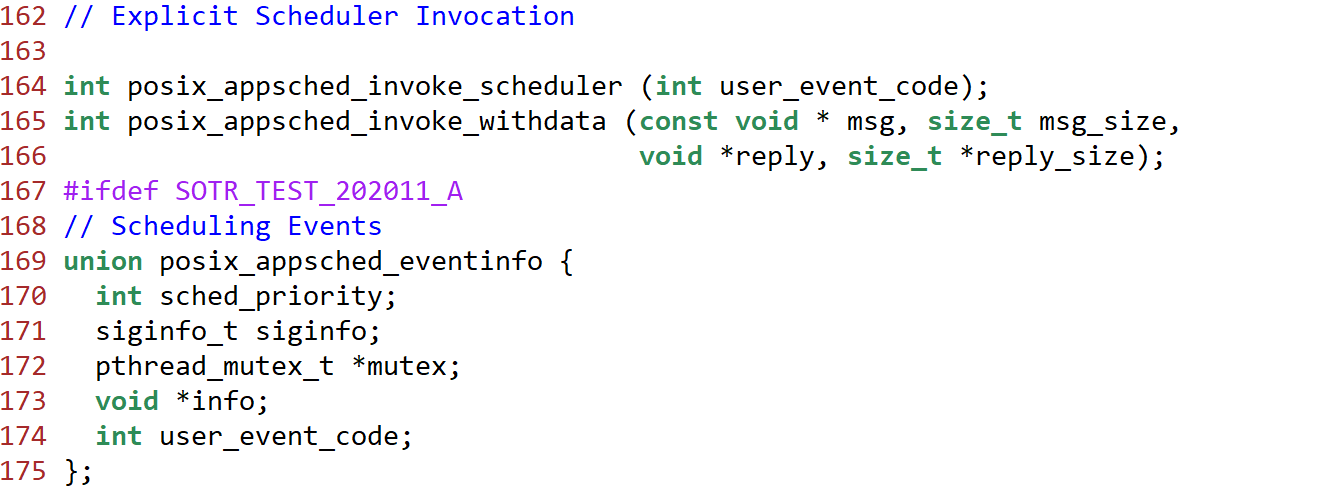
Para construir el programa ejecutable, podemos utilizar el siguiente archivo make

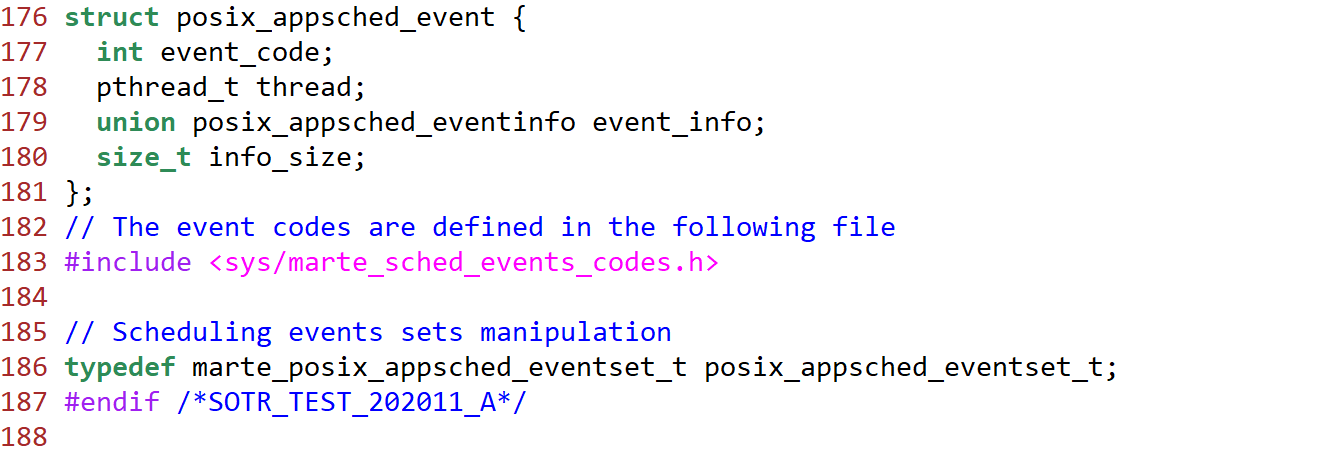


Archivo Makefile para construir el objetivo edf

Las macros SOTR\_TEST\_202011\_A y SOTR\_TEST\_202011\_B definidas en la variable CFLAGS del archivo make anterior se utilizan en los archivos x86\_arch/include/sched.h y x86\_arch/include/pthread.h como se muestra a continuación.

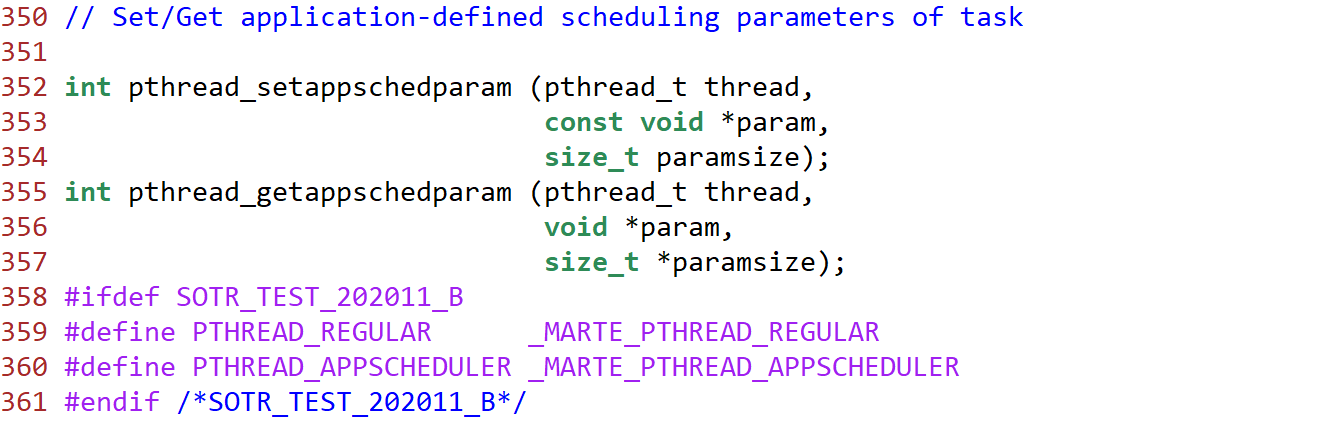
El siguiente fragmento del archivo x86\_arch/include/sched.h muestra la utilización de la macro SOTR\_TEST\_202011\_A





Fragmento del archivo x86\_arch/include/sched.h

El siguiente fragmento del archivo x86\_arch/include/pthread.h muestra la utilización de la macro SOTR\_TEST\_202011\_B



Fragmento del archivo x86\_arch/include/pthread.h

REFERENCIAS

[Aldea Rivas M.] Aldea Rivas, M. (2002). Planificación de Tareas en Sistemas Operativos de Tiempo Real Estricto para Aplicaciones Empotradas (Tesis Doctoral). Universidad de Cantabria, Facultad de Ciencias. Departamento de Electrónica y Computadores. Santander, noviembre de 2002.

[13720667.pdf] Altium TASKING POSIX Implementation.

[Altium, BV, 2015] TASKING POSIX Implementation.

[Burns, Welling, 2009] Alan Burns, Andy Welling, Real- Time Systems and Programming Languages, Pearson, 2009.

[Dietrich y Walker] Dietrich, S., y Walker, D., “The evolution of Real-Time Linux”, <http://www.cse.nd.edu/courses/cse60463/www/amatta2.pdf>, 2015.

[Labrosse, Jean J] Micro C/OS-II The Real Time Kernel (2/a edición). CRC Press, 2002.

[Liu, Layland] Liu, C.L., Layland, J.W., “Scheduling Algorithm for Multi-programming in a Hard Real-Time Environment,” J. ACM, Vol. 20, pp. 40-61, 1973.

[Wang] K.C. Wang, Design and Implementation of the MTX Operating System. Springer International Publishing, 2015.