Práctica 10 Planificando Procesos

# Creación de Procesos en MaRTE OS

En esta práctica se muestra la creación de procesos sobre un sistema operativo de tiempo real. En particular, se muestra cómo crear procesos periódicos haciendo uso de varios servicios disponibles sobre el sistema operativo MaRTE OS.

Funciones utilizadas

La función (incluir time.h)

int clock\_gettime (clockid\_t clock\_id, struct timespec \*tp);

permite obtener en la struct timespec cuya dirección se le pasa como segundo argumento, el tiempo transcurrido desde el inicio de la ejecución de la aplicación. El argumento clock\_id puede ser CLOCK\_MONOTONIC, CLOCK\_REALTIME, CLOCK\_THREAD\_CPUTIME\_ID. La struct timespec está declarada en el archivo de cabecera time.h

struct timespec {

time\_t tv\_sec; /\*\* segundos transcurridos\*/

int tv\_nsev;/\*\* nanosegundos transcurridos después de los tv\_sec segundos \*/

};

La función (incluir misc/timespec\_operations.h)

static inline void incr\_timespec (struct timespec \*t1, const struct timespec \*t2)

incrementa el tiempo almacenado en la estructura timespec a la que apunta t1, sumándole en tiempo almacenado en la estructura timespec a la que apunta t2.

La función (incluir pthread.h)

int pthread\_attr\_init (pthread\_attr\_t \*attr);

se utiliza para inicializar la estructura de atributos de thread de la cual se pasa su dirección al llamar a esta función.

La función (incluir pthread.h)

int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr,

void \*(\*start\_routine)(void \*), void \*arg);

crea un thread (un proceso ligero) a partir de la información que se le pasa en sus argumentos. En esta función, el parámetro thread su usa para pasar un entero por referencia (es un parámetro de salida), en el parámetro attr se debe pasar la dirección de una estructura pthread\_attr\_t previamente incializada con la función pthread\_attr\_init() mencionada antes. En el parámetro start\_routine, se debe pasar un apuntador a función de tipo void \* que recibe un apuntador a void, esto es, el nombre de una función void \*fn(void \*). En el parámetro arg se pasa un apuntador a void que en el código de la función fn se usa como argumento de la misma.

La función (incluir time.h)

int clock\_nanosleep(clockid\_t clock\_id, int FLAGS,

const struct timespec \*rqtp, struct timespec \*rmtp);

se utiliza para poner a dormir al thread que la llama durante el tiempo especificado en la estructura timespec a la que apunta rqtp. El cuarto parámetro rmtp puede ser NULL.

Utilización de las funciones anteriores

En esta sección, analizaremos primero el código que usaremos en la función main del primer ejercicio de esta práctica, y después el código de una función llamada periodic, que se utiliza como código de los hilos periódicos. En la función main() del presente ejercicio práctico, primero se crea un thread llamando a las funciones pthread\_attr\_init() y pthread\_create():

pthread\_t th;

pthread\_attr\_t attr;

int period1 = 1; /\* 1 second \*/

CHK( pthread\_attr\_init (&attr) );

CHK( pthread\_create (&th, &attr, periodic, &period1) );

Después se llamará a la función clock\_gettime() como sigue

struct timespec next\_activ\_dummy, my\_lag;

// …

if (clock\_gettime (CLOCK\_MONOTONIC, &next\_active\_dummy) )

printf(“Error en clock\_gettime”);

Con esto se obtiene en la estructura next\_active\_dummy, el tiempo en segundos y nanosegundos transcurrido desde que inició la ejecución de la aplicación –llamemos a este tiempo **tiempo inicial**. Después, se inicializarán los campos tv\_sec y tv\_nsec (segundos y nanosegundos) de la estructura my\_lag.

my\_lag.tv\_sec = 0; /\* 0 segundos \*/

my\_lag.tv\_nsec = 500000000; /\* 0.5 segundo \*/

llamemos a este tiempo, **tiempo de retardo**. En este punto, se hace un llamado a la función incr\_timespec() pasando &next\_activ\_dummy y &my\_lag como se indica a continuación

incr\_timespec(&next\_activ\_dummy, &my\_lag);

para obtener en la estructura next\_activ\_dummy el tiempo que corresponde a la suma **tiempo inicial** más **tiempo de retardo**.

Una vez hecho esto, se llama a la función clock\_nanosleep() como se indica aquí:

If ( clock\_nanosleep(CLOCK\_MONOTONE, TIMER\_ABSTIME,

&next\_activ\_dummy, NULL) )

printf(“Error in clock\_nanosleep”);

Con esto, el thread principal (el que corre la función main()) dormirá hasta el instante de tiempo **tiempo inicial** más **tiempo de retardo** (tiempo de retardo equivale a 0.5 s).

Después de que el thread principal despierta en el instante **tiempo inicial** más **tiempo de retardo**, se crea un segundo thread llamando nuevamente a las funciones pthread\_attr\_init() y pthread\_create():

int period2 = 2; /\* 2 seconds \*/

CHK( pthread\_attr\_init (&attr) );

CHK( pthread\_create (&th, &attr, periodic, &period2) );

Es decir, se han creado dos threads en los instantes de tiempo:

**tiempo inicial**,

y

**tiempo inicial** más **tiempo de retardo**.

Después de esto, en la función se llama a la función sleep() pasando como argumento 22, con lo cual se le permite a los dos threads creados que se ejecuten durante 22 segundos.

Ahora procedemos a analizar el código de la función periodic utilizada para los threads creados en main(). En la función

void \* periodic ( void \*arg)

Al inicio de esta función, se inicializa la estructura my\_period, (una estructura de tipo timespec)

my\_period.tv\_sec = \* (int \*)arg;

my\_period.tv\_nsec = 0;

Llamemos a este tiempo T, (en este ejemplo, para un thread T es 1s y para el otro T es 2s). Se inicializa la estructura next\_activation, (de tipo struct timespec) con el llamado a la función

if (clock\_gettime(CLOCK\_MONOTONIC, &next\_activation))

printf (“Error in clock\_gettime\n”);

Denominemos a este valor del tiempo como Ti. Después de esto, se tiene un ciclo while infinito

while (1) {

// . . .

}

Antes de este ciclo while, se tiene un comentario que indica que en ese ciclo se realizaría algún trabajo útil y se esperará hasta el siguiente periodo. Para lograr este efecto, dentro del ciclo while, al tiempo guardado en la estructura next\_activation se le suma el valor del periodo T con el llamado a la función

incr\_timespec (&next\_activation, &my\_period);

Ahora en la estructura next\_activation se tiene almacenado el valor Ti + T –el tiempo en el que deberá iniciar el próximo periodo. En este punto se incluiría el código para realizar el trabajo útil. En este ejercicio de ejemplo, se incluyeron las líneas

if (clock\_gettime(CLOCK\_MONOTONIC, &ahora))

printf(“Error in clock\_gettime\n”);

printf(“[Time:%2.1fs] ”, ahora.tv\_sec + ahora.tv\_nsec / 1000000000.0);

printf(“Thread with period %ds activation %d\n”,

my\_period.tv\_sec, activation\_count++);

Una vez realizado el “trabajo util”, se pone a dormir al thread hasta que se termine el periodo actual (es decir hasta Ti + T), con la siguiente sentencia if.

if ( clock\_nanosleep(CLOCK\_MONOTONIC, TIMER\_ABSTIME,

&next\_activation, NULL))

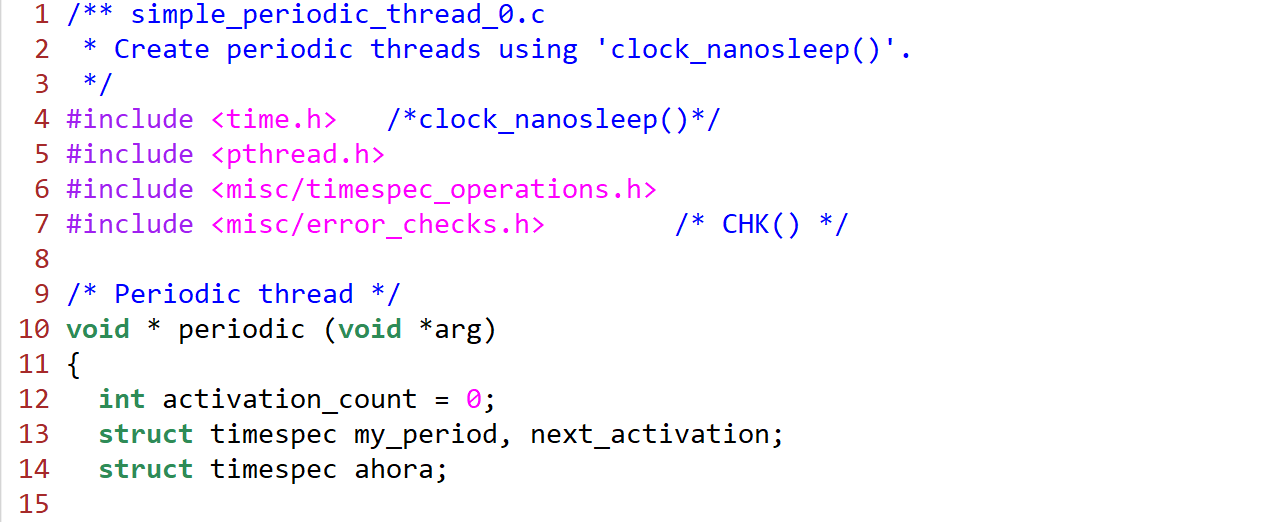
printf(“Error in clock\_nanosleep”);

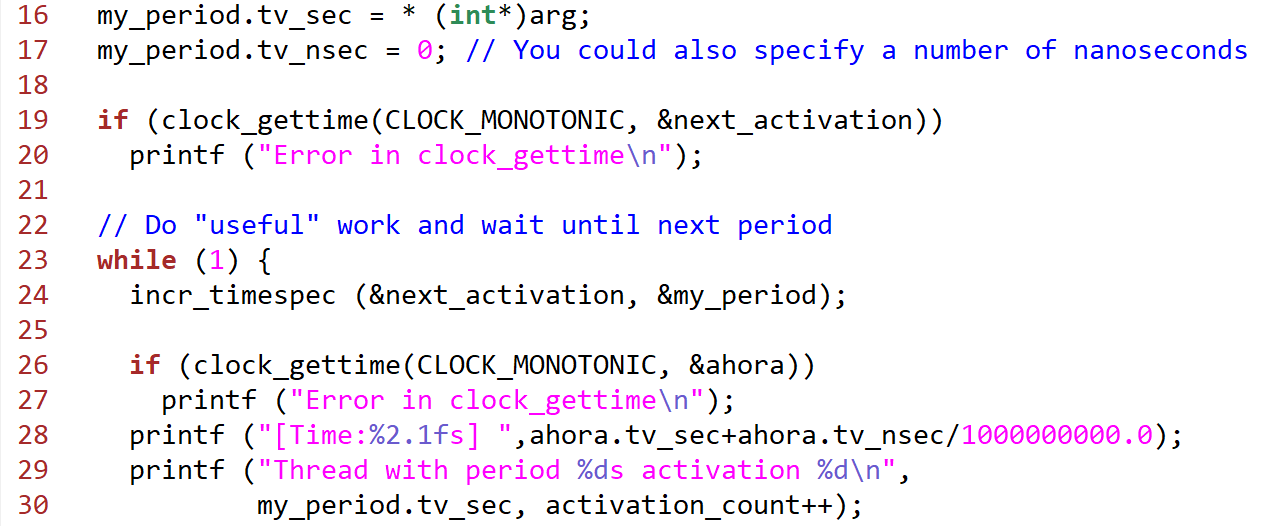
Esta es la última sentencia contenida en el ciclo while infinito.

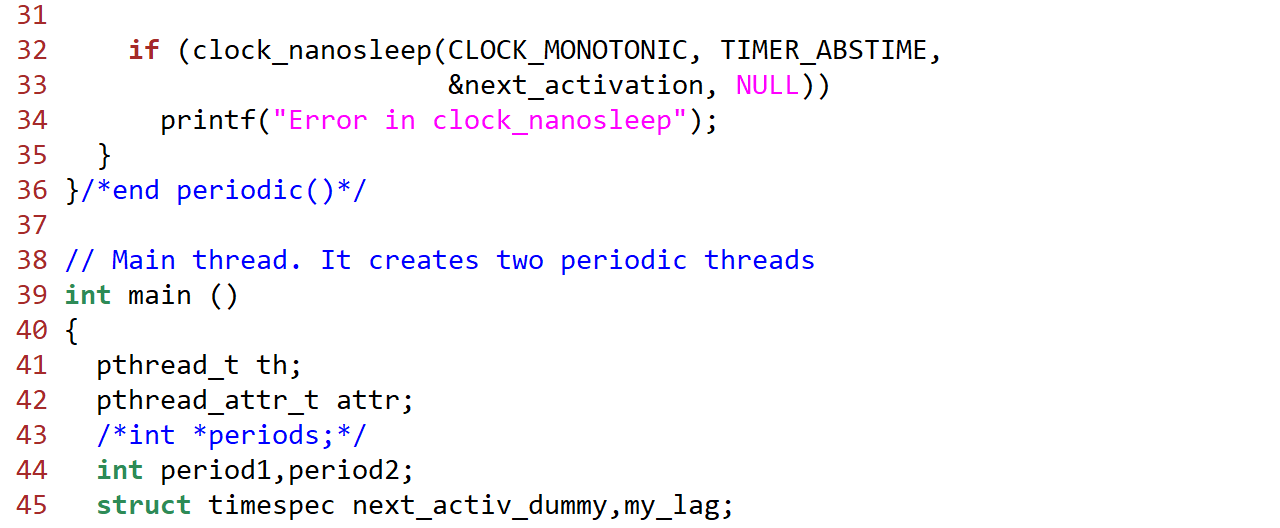
Con esta combinación de llamadas a las funciones incr\_timespec() y clock\_nanosleep() dentro de un ciclo while infinito, se logra que los threads creados se activen periódicamente.

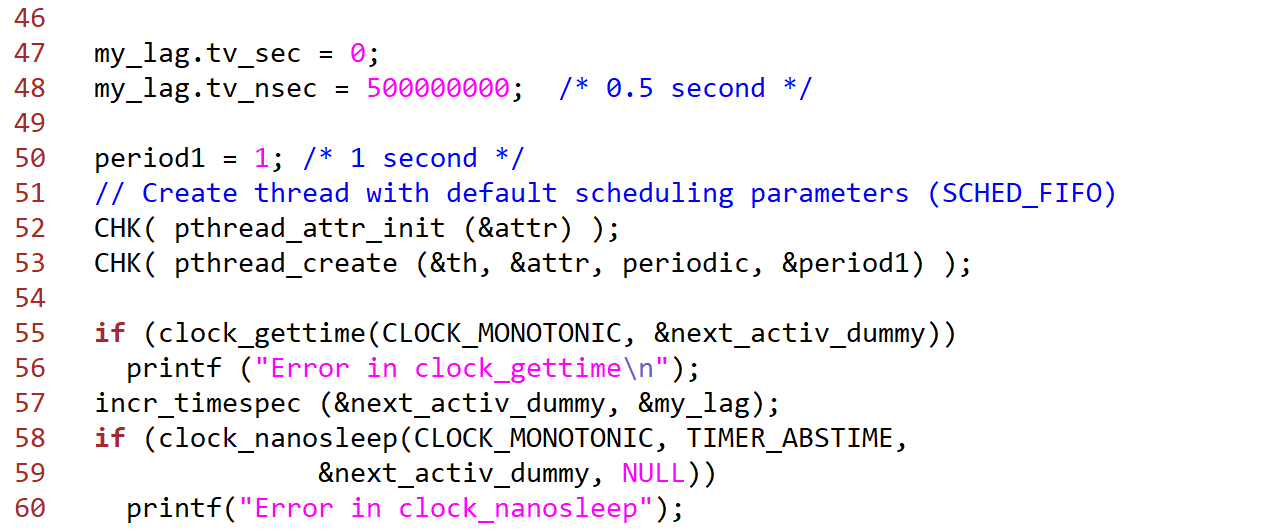
Archivo de ejemplo de creación hilos periódicos para construir sobre MaRTE OS

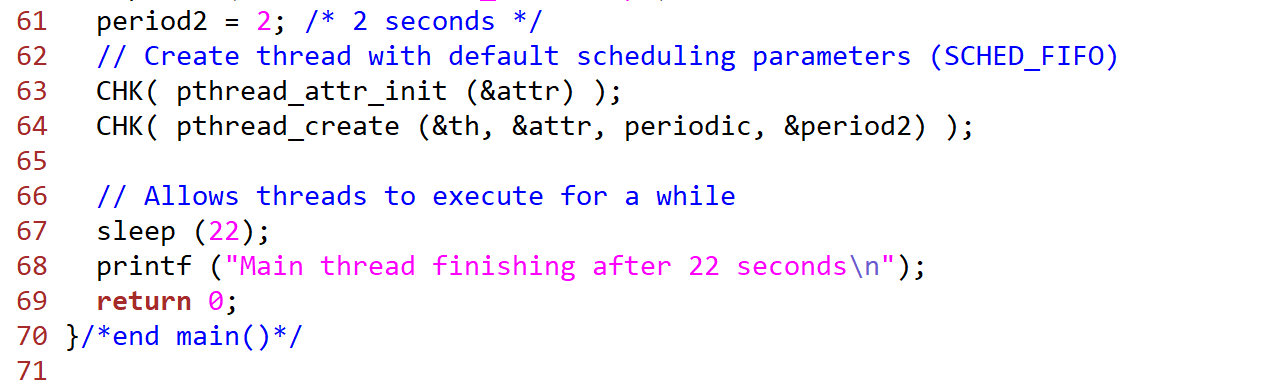
A continuación, se incluye el listado del archivo simple\_periodic\_thread\_0.c





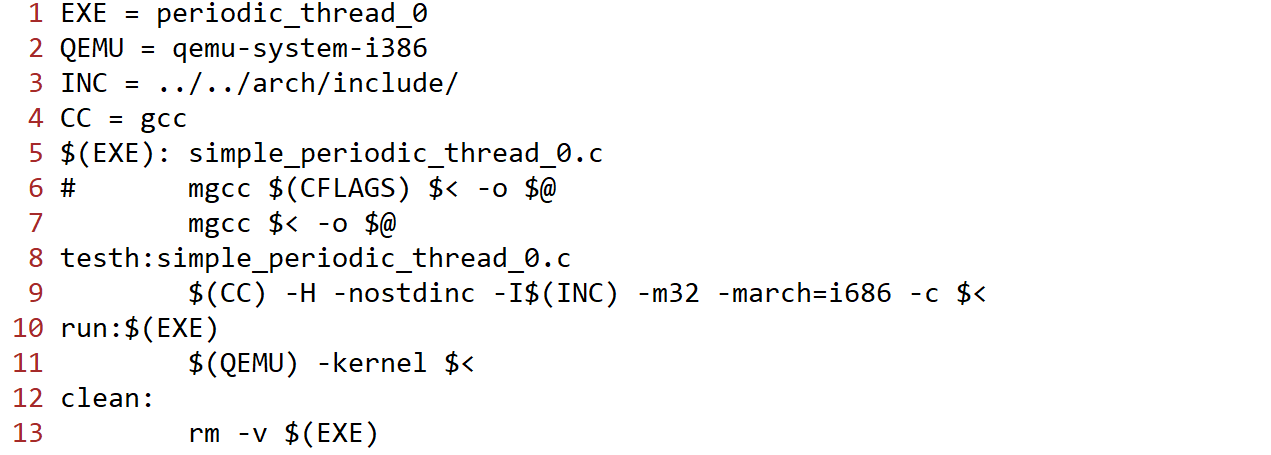






Archivo simple\_periodic\_thread\_0.c

El programa de aplicación se puede construir utilizando el siguiente archivo make

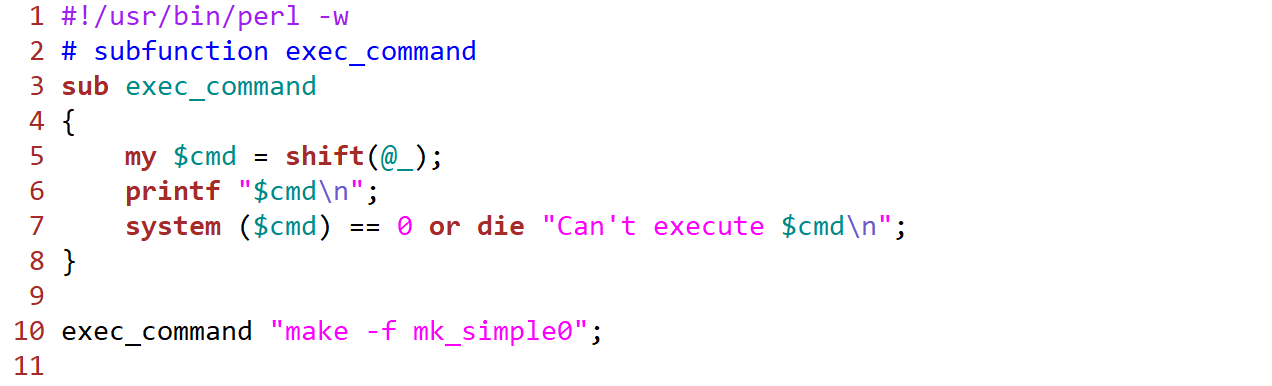


Archivo mk\_simple0

Para construir la aplicación periodic\_thread\_0 con el archivo make anterior, se debe teclear en la línea de comandos

$ make -f mk\_simple0

Alternativamente, y teniendo el archivo make anterior en el mismo directorio, se puede usar el siguiente script de perl para construir la aplicación.

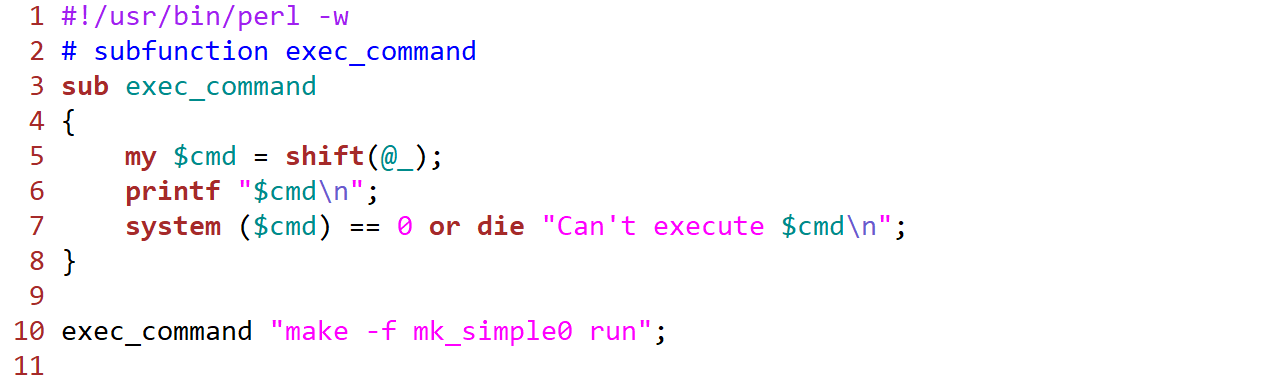


Archivo mksimple0

Para construir la aplicación periodic\_thread\_0 usando el script anterior, se debe teclear en la línea de comandos

$ ./mksimple0

Para correr la aplicación se puede usar el siguiente script de perl



Archivo runsimple0

Para ejecutar la aplicación periodic\_thread\_0 usando el script anterior se debe teclear en la línea de comandos

$ ./runsimple0

La siguiente figura muestra un ejemplo de la ejecución de periodic\_thread\_0 sobre

qemu-system-i386

