6-5-2024

Relojes y Temporizadores

Práctica 7

Julián Blanco González
UNIVERSIDAD DE EXTREMADURA

Contenido

Ejercicio 1

```c
#include <stdio.h>
#include <unistd.h>
int main()
{
    long ticks_per_second;
    // Consultar el número de ticks por segundo
    if ((ticks_per_second = sysconf(_SC_CLK_TCK)) == -1)
    {
        perror("Error al obtener el número de ticks por segundo");
        return 1;
    }
    printf("Número de ticks por segundo: %ld\n",ticks_per_second);
    return 0;
}
```

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$ ./ejercicio1
Número de ticks por segundo: 100
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$ ./ejercicio1
Número de ticks por segundo: 100
```

¿Cuántos ticks por segundo tiene tu sistema?

Tiene 100 ticks por segundo.

¿Cuál es la relación entre el número de tics por segundo y la resolución de tiempo del sistema?

a mayor número de ticks por segundo, mayor es la resolución del tiempo del sistema y viceversa, es decir, es una relación inversa.

¿Cuál es el impacto de cambiar el valor devuelto por sysconf(_SC_CLK_TCK) en un sistema POSIX?

Si los ticks fueran menos de 100, el reloj del sistema sería menos preciso, pero habría menos sobrecarga

Si los ticks fueran mayores de 100, el sistema operativo sería capaz de manejar eventos mejor, pero se sobrecargaría el sistema.

¿Está relacionado con el reloj de la CPU?

No, el número de ticks del reloj del sistema operativo. La CPU va por otro lado.

Ejercicio 2

```
 julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$ ./ejercicio2
CLOCK_REALTIME-0: 1714990743 segundos, 967689905 nanosegundos (1714990743.967)
TIEMPO EJECUCIÓN CON CLOCK_REALTIME-0: 0 segundos, 2678 nanosegundos
 CLOCK_MONOTONIC-1: 4090 segundos, 653546914 nanosegundos
 (      4090.653)
TIEMPO EJECUCIÓN CON CLOCK_MONOTONIC-1: 0 segundos, 2631 nanosegundos
 julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$ ./ejercicio2
CLOCK_REALTIME-0: 1714990745 segundos, 132132270 nanosegundos (1714990745.132)
TIEMPO EJECUCIÓN CON CLOCK_REALTIME-0: 0 segundos, 2640 nanosegundos
 CLOCK_MONOTONIC-1: 4091 segundos, 817989283 nanosegundos
 (      4091.817)
TIEMPO EJECUCIÓN CON CLOCK_MONOTONIC-1: 0 segundos, 2584 nanosegundos
```

Primera parte

¿Qué códigos están asignados a cada reloj?

Reloj Realtime:

Estructura timespec ts_realtimeSTART y ts_realtimeEND

```c
if (clock_gettime(CLOCK_REALTIME, &ts_realtimeSTART) == -1)
{
    perror("Error con CLOCK_REALTIME [1]");
    return 1;
}

if (clock_gettime(CLOCK_REALTIME, &ts_realtimeEND) == -1)
{
    perror("Error con CLOCK_REALTIME [2]");
    return 1;
}
```

Reloj Monotonic:

Estructura timespec ts_monotonic_START y ts_monotonic_END.

```c
if (clock_gettime(CLOCK_MONOTONIC, &ts_monotonic_START) == -1)
{
    perror("Error con CLOCK_MONOTONIC [1]");
    return 1;
}

if (clock_gettime(CLOCK_MONOTONIC, &ts_monotonic_END) == -1)
{
    perror("Error con CLOCK_MONOTONIC [2]");
    return 1;
}
```

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$ ./ejercicio2
CLOCK_REALTIME-0: 1715417830 segundos, 727037828 nanosegundos (1715417830.727)
TIEMPO EJECUCIÓN CON CLOCK_REALTIME-0: 0 segundos, 2616 nanosegundos
 CLOCK_MONOTONIC-1: 129 segundos, 124619607 nanosegundos
 (       129.124)
TIEMPO EJECUCIÓN CON CLOCK_MONOTONIC-1: 0 segundos, 2567 nanosegundos
```

¿Qué significa el primer tiempo obtenido por el CLOCK_REALTIME? Pásalo a días y estima la cantidad en años.

Es el tiempo que ha pasado desde el 1 de enero de 1970 (época de Unix) hasta que se ejecuta el programa.

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$ ./ejercicio2
CLOCK_REALTIME-0: 1715418327 segundos, 902652173 nanosegundos (1715418327.902)
Tiempo en días (CLOCK_REALTIME): 19854
Estimación de años (CLOCK_REALTIME): 54
TIEMPO EJECUCIÓN CON CLOCK_REALTIME-0: 0 segundos, 2723 nanosegundos
 CLOCK_MONOTONIC-1: 626 segundos, 300233976 nanosegundos
 (       626.300)
TIEMPO EJECUCIÓN CON CLOCK_MONOTONIC-1: 0 segundos, 2637 nanosegundos
 julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$
```

Días: 19584 días, Años: 54 años

¿Qué significa el primer tiempo obtenido por el CLOCK_MONOTONIC? Pásalo a horas

Tiempo que ha transcurrido desde algún punto en el pasado (tiempo de arranque del sistema…).

```
 julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$ ./ejercicio2
CLOCK_REALTIME-0: 1715418489 segundos, 763778983 nanosegundos (1715418489.763)
Tiempo en días (CLOCK_REALTIME): 19854
Estimación de años (CLOCK_REALTIME): 54
TIEMPO EJECUCIÓN CON CLOCK_REALTIME-0: 0 segundos, 2669 nanosegundos
 CLOCK_MONOTONIC-1: 788 segundos, 161360732 nanosegundos
 (       788.161)
Tiempo en horas (CLOCK_MONOTONIC): 0
TIEMPO EJECUCIÓN CON CLOCK_MONOTONIC-1: 0 segundos, 2639 nanosegundos
```

Compara los tiempos de ejecución obtenidos por cada reloj. ¿Son iguales? ¿Qué conclusiones extraes?

Clock_Realtime: 0 segundos, 2669 nanosegundos

Clock_Monotonic: 0 segundos, 2639 nanosegundos

Son tiempos casi iguales, es decir, que ambos pueden usarse para la ejecución de una tarea.

Segunda parte

Modifica el código para mostrar las resoluciones de CLOCK_REALTIME y CLOCK_MONOTONIC.

```
// Calcular resoluciones
struct timespec res_realtime, res_monotonic;
clock_getres(CLOCK_REALTIME, &res_realtime);
clock_getres(CLOCK_MONOTONIC, &res_monotonic);

// Mostrar los resultados
printf("Resolución de CLOCK_REALTIME: %ld segundos, %ld nanosegundos\n",
       res_realtime.tv_sec, res_realtime.tv_nsec);
printf("Resolución de CLOCK_MONOTONIC: %ld segundos, %ld nanosegundos\n",
       res_monotonic.tv_sec, res_monotonic.tv_nsec);
```

Resultados

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$ ./ejercico2_2
Resolución de CLOCK_REALTIME: 0 segundos, 1 nanosegundos
Resolución de CLOCK_MONOTONIC: 0 segundos, 1 nanosegundos
```

Tercera parte

Modifica el código para calcular el tiempo transcurrido por el proceso con CLOCK_PROCESS_CPUTIME_ID. Compara los tiempos obtenidos. ¿Se diferencia con REALTIME y MONOTONIC? ¿Por qué?

```
struct timespec ts_realtimeSTART, ts_realtimeEND, ts_monotonic_START,
ts_process_START, ts_process_END;
int aux = 1;

// Consultar el reloj CLOCK_PROCESS_CPUTIME_ID (START)
if (clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &ts_process_START) == -1)
{
    perror("Error con CLOCK_PROCESS_CPUTIME_ID [1]");
    return 1;
}

// Consultar el reloj CLOCK_PROCESS_CPUTIME_ID (END)
if (clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &ts_process_END) == -1)
{
    perror("Error con CLOCK_PROCESS_CPUTIME_ID [2]");
    return 1;
}
```

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$ ./ejercico2_2
Resolución de CLOCK_REALTIME: 1715419247 segundos, 129872234 nanosegundos
Resolución de CLOCK_MONOTONIC: 1545 segundos, 527453954 nanosegundos
Tiempo transcurrido por el proceso con CLOCK_PROCESS_CPUTIME_ID: 0 segundos, 3580 nanosegundos
CLOCK_REALTIME-0: 1715419247 segundos, 129867386 nanosegundos (1715419247.129)
Tiempo en días (CLOCK_REALTIME): 19854
Estimación de años (CLOCK_REALTIME): 54
TIEMPO EJECUCIÓN CON CLOCK_REALTIME-0: 0 segundos, 4848 nanosegundos
 CLOCK_MONOTONIC-1: 1545 segundos, 527449155 nanosegundos
 (      1545.527)
Tiempo en horas (CLOCK_MONOTONIC): 0
TIEMPO EJECUCIÓN CON CLOCK_MONOTONIC-1: 0 segundos, 4799 nanosegundos
```

Tiempo del proceso: 0 segundos, 3580 nanosegundos. El valor es menor porque este valor mide el tiempo de la CPU que usa el proceso y los relojes miden el tiempo real y el tiempo del reloj monotónico.

Cambia el bucle por un sleep(1). ¿Qué tiempo nos da CLOCK_PROCESS_CPUTIME_ID?

Para poder usar, hay que incluir la librería: #include <unistd.h>

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$ ./ejercico2_2
Resolución de CLOCK_REALTIME: 1715420017 segundos, 595957802 nanosegundos
Resolución de CLOCK_MONOTONIC: 2315 segundos, 993539541 nanosegundos
Tiempo transcurrido por el proceso con CLOCK_PROCESS_CPUTIME_ID: 0 segundos, 153118 nanosegundos
CLOCK_REALTIME-0: 1715420016 segundos, 501372636 nanosegundos (1715420016.501)
Tiempo en días (CLOCK_REALTIME): 19854
Estimación de años (CLOCK_REALTIME): 54
TIEMPO EJECUCIÓN CON CLOCK_REALTIME-0: 1 segundos, 94585166 nanosegundos
 CLOCK_MONOTONIC-1: 2314 segundos, 898954389 nanosegundos
 (      2314.898)
Tiempo en horas (CLOCK_MONOTONIC): 0
TIEMPO EJECUCIÓN CON CLOCK_MONOTONIC-1: 1 segundos, 94585152 nanosegundos
```

Tiempo proceso: 0 segundos, 153118 nanosegundos. Ha aumentado considerablemente el tiempo del proceso, ya que el bucle for es más rápido que un 1 segundo.

Calcula la resolución de CLOCK_PROCESS_CPUTIME_ID

```c
struct timespec ts_realtimeSTART, ts_realtimeEND, ts_monotonic_START, ts_monc
ts_process_START, ts_process_END, res_process;
int aux = 1;

// Consultar la resolución de CLOCK_PROCESS_CPUTIME_ID
if (clock_getres(CLOCK_PROCESS_CPUTIME_ID, &res_process) == -1)
{
    perror("Error al obtener la resolución de CLOCK_PROCESS_CPUTIME_ID");
    return 1;
}

// Mostrar la resolución obtenida
printf("Resolución de CLOCK_PROCESS_CPUTIME_ID: %ld segundos, %ld nanosegundos\n",
    res_process.tv_sec, res_process.tv_nsec);
```

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$ ./ejercico2_2
Resolución de CLOCK_PROCESS_CPUTIME_ID: 0 segundos, 1 nanosegundos
Resolución de CLOCK_REALTIME: 1715420435 segundos, 145404074 nanosegundos
Resolución de CLOCK_MONOTONIC: 2733 segundos, 542985799 nanosegundos
```

Cuarta parte

Modifica el código para calcular el tiempo transcurrido por el hilo del proceso con CLOCK_THREAD_CPUTIME_ID. Compara los tiempos obtenidos. ¿Se diferencia con REALTIME, MONOTONIC y PROCESS_CPUTIME? ¿Por qué?

```
struct timespec ts_realtimeSTART, ts_realtimeEND, ts_monotonic_START, ts_monoto
ts_process_START, ts_process_END, res_process, ts_thread_START, ts_thread_END;
int aux = 1;


 // Consultar el reloj CLOCK_THREAD_CPUTIME_ID (START)
if (clock_gettime(CLOCK_THREAD_CPUTIME_ID, &ts_thread_START) == -1)
{
    perror("Error con CLOCK_THREAD_CPUTIME_ID [1]");
    return 1;
}

// Consultar el reloj CLOCK_THREAD_CPUTIME_ID (END)
if (clock_gettime(CLOCK_THREAD_CPUTIME_ID, &ts_thread_END) == -1)
{
    perror("Error con CLOCK_THREAD_CPUTIME_ID [2]");
    return 1;
}

long segundos_hilo = ts_thread_END.tv_sec - ts_thread_START.tv_sec;
long nanosegundos_hilo = ts_thread_END.tv_nsec - ts_thread_START.tv_nsec;
if (nanosegundos_hilo < 0) {
    segundos_hilo -= 1;
    nanosegundos_hilo += NANOSECS_IN_SEC;
}

printf("Tiempo transcurrido por el hilo del proceso con CLOCK_THREAD_CPUTIME_ID: 
    segundos_hilo, nanosegundos_hilo);
```

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$ ./ejercico2_2
Resolución de CLOCK_PROCESS_CPUTIME_ID: 0 segundos, 1 nanosegundos
Tiempo transcurrido por el hilo del proceso con CLOCK_THREAD_CPUTIME_ID: 0 segundos, 143325 nanosegundos
Resolución de CLOCK_REALTIME: 1715420872 segundos, 885786879 nanosegundos
Resolución de CLOCK_MONOTONIC: 3171 segundos, 283368601 nanosegundos
Tiempo transcurrido por el proceso con CLOCK_PROCESS_CPUTIME_ID: 0 segundos, 143289 nanosegundos
CLOCK_REALTIME-0: 1715420872 segundos, 885642441 nanosegundos (1715420872.885)
Tiempo en días (CLOCK_REALTIME): 19854
Estimación de años (CLOCK_REALTIME): 54
TIEMPO EJECUCIÓN CON CLOCK_REALTIME-0: 0 segundos, 144438 nanosegundos
 CLOCK_MONOTONIC-1: 3171 segundos, 283224227 nanosegundos
 (      3171.283)
Tiempo en horas (CLOCK_MONOTONIC): 0
TIEMPO EJECUCIÓN CON CLOCK_MONOTONIC-1: 0 segundos, 144374 nanosegundos
```

Se diferencia con ambos relojes (Monotonic y Realtime), pero tiene un tiempo muy parecido al del proceso de la CPU, ya que ambos miden el tiempo de la CPU consumido por el proceso y el hilo.

Cambia el bucle por un sleep(1). ¿Qué tiempo nos da CLOCK_THREAD_CPUTIME_ID?

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$ ./ejercico2_2
Resolución de CLOCK_PROCESS_CPUTIME_ID: 0 segundos, 1 nanosegundos
Tiempo transcurrido por el hilo del proceso con CLOCK_THREAD_CPUTIME_ID: 0 segundos, 384085 nanosegundos
Resolución de CLOCK_REALTIME: 1715420948 segundos, 927589727 nanosegundos
Resolución de CLOCK_MONOTONIC: 3247 segundos, 325171464 nanosegundos
Tiempo transcurrido por el proceso con CLOCK_PROCESS_CPUTIME_ID: 0 segundos, 382868 nanosegundos
CLOCK_REALTIME-0: 1715420947 segundos, 922423177 nanosegundos (1715420947.922)
Tiempo en días (CLOCK_REALTIME): 19854
Estimación de años (CLOCK_REALTIME): 54
TIEMPO EJECUCIÓN CON CLOCK_REALTIME-0: 1 segundos, 5166550 nanosegundos
 CLOCK_MONOTONIC-1: 3246 segundos, 320004997 nanosegundos
 (      3246.320)
Tiempo en horas (CLOCK_MONOTONIC): 0
TIEMPO EJECUCIÓN CON CLOCK_MONOTONIC-1: 1 segundos, 5166467 nanosegundos
```

Ha aumentado casi el triple de tiempo.

Calcula la resolución de CLOCK_THREAD_CPUTIME_ID

```c
// Consultar la resolución de CLOCK_THREAD_CPUTIME_ID
if (clock_getres(CLOCK_THREAD_CPUTIME_ID, &res_thread) == -1)
{
    perror("Error al obtener la resolución de CLOCK_THREAD_CPUTIME_ID");
    return 1;
}

// Mostrar la resolución obtenida para CLOCK_THREAD_CPUTIME_ID
printf("Resolución de CLOCK_THREAD_CPUTIME_ID: %ld segundos, %ld nanosegundos\n",
       res_thread.tv_sec, res_thread.tv_nsec);
```

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$ ./ejercico2_2
Resolución de CLOCK_THREAD_CPUTIME_ID: 0 segundos, 1 nanosegundos
Resolución de CLOCK_PROCESS_CPUTIME_ID: 0 segundos, 1 nanosegundos
Tiempo transcurrido por el hilo del proceso con CLOCK_THREAD_CPUTIME_ID: 0 segundos, 219840 nanosegundos
Resolución de CLOCK_REALTIME: 1715421480 segundos, 68497216 nanosegundos
Resolución de CLOCK_MONOTONIC: 3778 segundos, 466078971 nanosegundos
Tiempo transcurrido por el proceso con CLOCK_PROCESS_CPUTIME_ID: 0 segundos, 218832 nanosegundos
CLOCK_REALTIME-0: 1715421479 segundos, 67330734 nanosegundos (1715421479.067)
Tiempo en días (CLOCK_REALTIME): 19854
Estimación de años (CLOCK_REALTIME): 54
TIEMPO EJECUCIÓN CON CLOCK_REALTIME-0: 1 segundos, 1166482 nanosegundos
 CLOCK_MONOTONIC-1: 3777 segundos, 464912485 nanosegundos
 (      3777.464)
Tiempo en horas (CLOCK_MONOTONIC): 1
TIEMPO EJECUCIÓN CON CLOCK_MONOTONIC-1: 1 segundos, 1166486 nanosegundos
```

Ejercicio 3

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <unistd.h>

int main(void)
{
    struct timespec retardo, retardo_pend;
    int res;

    //@TODO preparar un retardo relativo de 2.550 s con retardo
    retardo.tv_sec = 2;
    retardo.tv_nsec = 150E6; // 150 ms
    printf("Voy a esperar %ld.%09ld s ......\n",
           retardo.tv_sec, retardo.tv_nsec);

    //@TODO nanosleep
    if ((res = nanosleep(&retardo, &retardo_pend)) == -1)
        printf("He despertado por una señal, Retardo pendiente: %ld s %ld nsec\n", retardo_pend.tv_sec,
                                                retardo_pend.tv_nsec);
    else
        printf("He despertado y ya terminó.\n");

    exit(0);
}
```

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$ ./ejercicio3
Voy a esperar 2.150000000 s ......
He despertado y ya terminó.
```

Se crea una estructura timespec de retardo (tiempo de espera), retardo_pend (tiempo de espera si nanosleep se interrumpe).

Ahora, hay que asignar los segundos al campo de la estructura, así como los nanosegundos. Ahora, llamar a la función nanosleep, pasandolo como parámetros ambas estructuras timespec. Si la función falla (res = -1), se muestra por pantalla los valores de la estructura de retardo_pend; si no falla, se muestra un mensaje de que termina la espera (“despierta y termina”) al cabo del tiempo especificado en la estructura de retardo.

Ejercicio 4

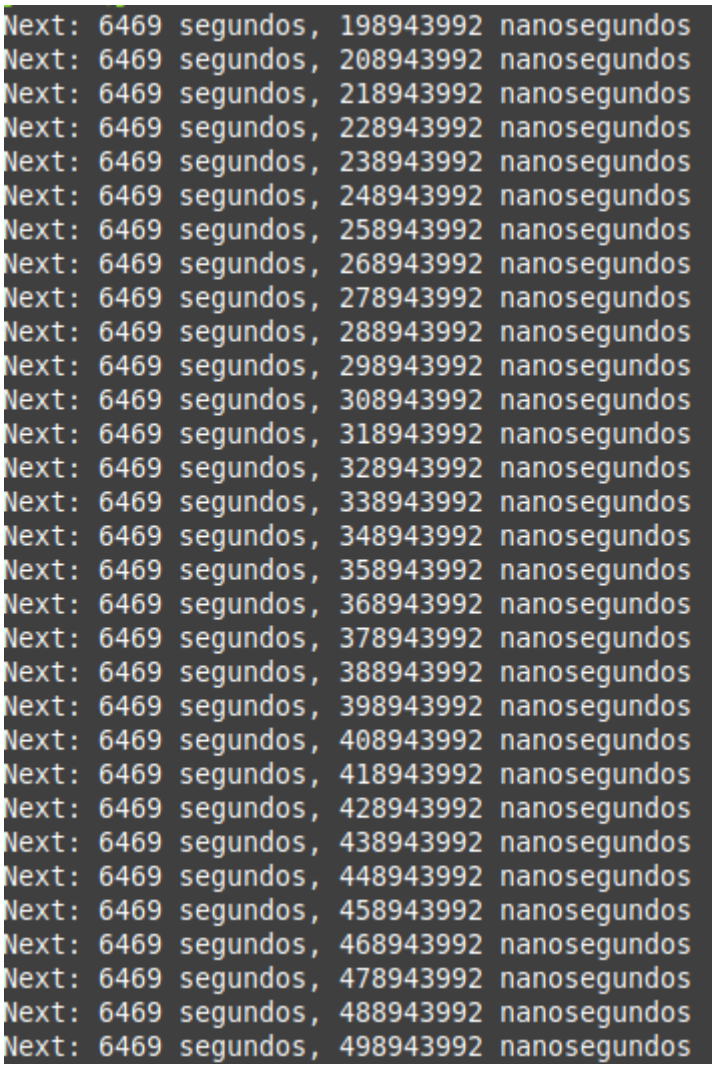
Analiza el código e imprime los valores de next. ¿Qué está pasando?

Al compilar da muchos errores. Lo primero es incluir las librerías de time.h (estructuras de timespec) y signal.h (constante TIMER_ABSTIME).

Luego, el incremento de next = next + period no puede hacerse, ya que no pueden sumarse 2 estructuras. Para ello, hay que sumar por campos, pero hay que tener en cuenta que hay que hacer el acarreo de los nanosegundos.

```c
#include <stdio.h>
#include <unistd.h>
//Librerías que faltaban
#include <time.h>
#include <signal.h>

int main()
{
    struct timespec next, period;

    if (clock_gettime(CLOCK_MONOTONIC, &next) != 0)
        return -1;

    period.tv_sec = 0;
    period.tv_nsec = 10.0E6; /* 10 ms */

    while (1)
    {
        if (clock_nanosleep(CLOCK_MONOTONIC,TIMER_ABSTIME, &next, 0) != 0)
            return -1;

        printf("Next: %ld segundos, %ld nanosegundos\n", next.tv_sec, next.tv_nsec);
        next.tv_nsec = next.tv_nsec + period.tv_nsec;
        next.tv_sec = next.tv_sec + (next.tv_nsec / 1000000000);
        next.tv_nsec = next.tv_nsec % 1000000000;
    }
    return 0;
}
```

Estructuras timespec para ver los tiempos de espera y los tiempos entre iteraciones del bucle. Hay un tiempo de espera de 10 milisegundos. Se hace un bucle infinito, se llama a la función clock_nanosleep con parámetros: reloj monotonic, time_abstime (retardo absoluto) y next (tiempo de espera hasta el siguiente momento). Por último, se calcula el siguiente tiempo de espera con la estructura period (10 ms).



Imprime el tiempo en el que ocurrirá la próxima ejecución en segundos y nanosegundos.

Ejercicio 5

```c
void timer_handler(int sig)
{
    static int count = 0;
    printf("Timer expired %d times\n", ++count);
}
int main()
{
    //@TODO Declarar el temporizador
    timer_t timerid;
    //@TODO Declarar el sigevent para notificar la expiración del temporizador
    struct sigevent sev;
    //@TODO Declarar el itimerspec para configurar el
    struct itimerspec its;
    // Configurar la acción de la señal
    signal(SIGRTMIN, timer_handler);
    //@TODO Configurar la estructura sigevent para usar una señal en tiempo real: SIGRTMIN
    sev.sigev_notify = SIGEV_SIGNAL;
    sev.sigev_signo = SIGRTMIN;
    /* Data passed with notification: Pointer value */
    sev.sigev_value.sival_ptr = &timerid;
    //@TODO Crear el temporizador con REALTIME
    if (timer_create(CLOCK_REALTIME, &sev, &timerid) == -1)
    {
        perror("timer_create");
        exit(EXIT_FAILURE);
    }
    //@TODO Configurar el temporizador para expirar después de 3 segundos y cada 3 segundos
    its.it_value.tv_sec = 3;
    its.it_value.tv_nsec = 0;
    its.it_interval.tv_sec = 3;
    its.it_interval.tv_nsec = 0;
    //@TODO Establecer el temporizador
    if (timer_settime(timerid, 0, &its, NULL) == -1)
    {
        perror("timer_settime");
        exit(EXIT_FAILURE);
    }
    // Pausa el programa y espera señales indefinidamente
    while (1)
    {
        // La función pause() suspende el programa hats que llegue una señal
        pause();
    }
    //@TODO Eliminar el temporizador
    timer_delete(timerid);
    return 0;
}
```

Primero, es un método que muestra por pantalla un mensaje con un contador cada vez que el temporizador termine.

Luego crea el temporizador, la configuración del temporizador (itimerspec) y un sigevent. Después se configura la sekal SIGRTMIN con el método, después se rellena los campos de la estructura sigevent para notificar la expiración del temporizador (SIGRTMIN).

Crea el temporizador (timer_create()) con la estructura sigevent. El temporizador una vez creado, se configura para que termine cada 3 segundos y vuelva a repetirse cada 3

segundos. Después establece el temporizador con timer_settime(). Luego, se crea un bucle infinito con la instrucción pause() en espera de señales.

Por último, con la función timer_delete, se elimina el temporizador una vez que acabe el programa.

¿Que hace la línea signal(SIGRTMIN, timer_handler)?

Usa la función SIGRTMIN con el método void_handler para que cada vez que el temporizador termine, salte esta función y se llame al método para mostrar el mensaje por pantalla.

¿Para qué se usa SIGRTMIN? Consulta las señales disponibles con el comando kill -l.

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Es la número 34. Es la señal en tiempo real mínima. Esta señal es útil cuando se necesita una comunicación rápida y eficiente entre procesos.

¿Qué está haciendo el programa?

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S8$ ./ejercicio5
Timer expired 1 times
Timer expired 2 times
Timer expired 3 times
Timer expired 4 times
Timer expired 5 times
Timer expired 6 times
Timer expired 7 times
Timer expired 8 times
Timer expired 9 times
Timer expired 10 times
Timer expired 11 times
Timer expired 12 times
Timer expired 13 times
Timer expired 14 times
Timer expired 15 times
Timer expired 16 times
Timer expired 17 times
Timer expired 18 times
Timer expired 19 times
^C
```

Cada 3 segundos, salta la señal de SIGRTMIN por la configuración de sigevent, por lo que, gracias a la función signal, se ejecuta el método void_handler, mostrando el mensaje por pantalla añadiendo 1 al contador cada vez que se llama a la función. Esto se produce gracias también a la función pause (espera de señales).

Bibliografía

Realización de la práctica

Pdf de la práctica: POSIX 3.pdf

Páginas externas

Sumar estructuras timespec

- c - How to properly add to timespec variables - Stack Overflow

Señal SIGRTMIN

- Signal (IPC) - Wikipedia