```c
/* Problema del 5/9/11 */

#define __POSIX_C_SOURCE 199506L
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <stdio.h>
#include <mqueue.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>


#include "problema.h"

void enviar_cola(char *ncola);


int main(int a1, char **a2)  {
   mqd_t cola;                          /* Cola de especificaciones */
   int timeout;                         /* sobretiempo en segundos */
   timer_t tciclo;                      /* Temporizador de ciclo */
   timer_t tduty;                       /* Temporizador de "duty cycle" */
   struct param datos = {1000, 50.}; /* Especificacion inicializada valor por defecto */
   int acabar;
   int timeout_pendiente;
   struct mq_attr atrib;
   sigset_t s;
   sigset_t sig_duty;
   sigset_t sig_cic_o_timeout;
   int pri;
   int res;
   int sig;
   int nueva_prog;
   struct timespec cero = {0,0};
   struct timespec ciclo;
   struct timespec ton;
   struct itimerspec prog_ciclo;
   struct itimerspec prog_duty;
   struct sigevent ev_ciclo;
   struct sigevent ev_duty;

   if(a1 < 3)
   {
     printf("%d son pocos argumentos\n", a1);
     exit(1);
   }

   /* Averiguar cuenta de sobretiempo */

   sscanf(a2[2], "%d", &timeout);
   printf("timeout de %d\n", timeout); fflush(stdout);
```

```c
/* Bloquear senales de temporizadores y sobretiempo */

sigemptyset(&s);
sigaddset(&s, SIGRTMIN);
sigaddset(&s, SIGRTMIN+1);
sigaddset(&s, SIGALRM);
sigprocmask(SIG_BLOCK, &s, NULL);

/* Conjunto para senales de ciclo o de sobretiempo */

sigemptyset(&sig_cic_o_timeout);
sigaddset(&sig_cic_o_timeout, SIGRTMIN);
sigaddset(&sig_cic_o_timeout, SIGALRM);

/* Conjunto para senal de "duty cycle" */

sigemptyset(&sig_duty);
sigaddset(&sig_duty, SIGRTMIN+1);

/* Crear temporizador de ciclo con senal SIGRTMIN */

ev_ciclo.sigev_signo = SIGRTMIN;
ev_ciclo.sigev_notify = SIGEV_SIGNAL;
ev_ciclo.sigev_value.sival_int = 0;
timer_create(CLOCK_REALTIME, &ev_ciclo, &tciclo);

/* Crear temporizador de duty cycle con senal SIGRTMIN+1 */

ev_duty.sigev_signo = SIGRTMIN+1;
ev_duty.sigev_notify = SIGEV_SIGNAL;
ev_duty.sigev_value.sival_int = 0;
timer_create(CLOCK_REALTIME, &ev_duty, &tduty);

prog_duty.it_interval = cero;

/* Crear y abrir cola de lectura de especificaciones */

atrib.mq_msgsize = sizeof(struct param);
atrib.mq_maxmsg = LCOLA;
mq_unlink(a2[1]);
cola = mq_open(a2[1], O_RDONLY  | O_NONBLOCK | O_CREAT, S_IRWXU, &atrib);
if(cola == -1)
{
  printf("problema al abrir cola\n");
  exit(1);
}

/* Prueba */
{
    pid_t p;
    p = fork();
    if(!p)
    {
      enviar_cola(a2[1]);
```

```
      exit(0);
    }
}
/* Fin prueba */

 /* Una programacion por ciclo; la primera es la programada por
    defecto */

acabar = 0;
timeout_pendiente = 0;
while(!acabar)
{
   long ms_on;

   /* Programar temporizadores */

   ciclo.tv_sec = datos.ciclo/1000;
   ciclo.tv_nsec = (datos.ciclo - ciclo.tv_sec*1000)*1e6;

   ms_on = datos.dcic*datos.ciclo*0.01;
   ton.tv_sec = ms_on/1000;
   ton.tv_nsec = (ms_on - ton.tv_sec*1000)*1e6;

   printf("ciclo de %d sg y %ld ms\n", (int)ciclo.tv_sec, ciclo.tv_nsec/1000000);
   fflush(stdout);
   printf("Tiempo activo: %d sg %ld ms\n", (int)ton.tv_sec, ton.tv_nsec/1000000);
   fflush(stdout);

   /* Preparacion de programacion de temporizadores */

   prog_ciclo.it_value = ciclo;
   prog_ciclo.it_interval = ciclo;
   prog_duty.it_value = ton;
   prog_duty.it_interval = cero;

   /* Programacion de temporizador de ciclo */

   timer_settime(tciclo, 0, &prog_ciclo, NULL);

   /* Activar sobretiempo */

   alarm(timeout);

   /* Iterar una vez por ciclo,
      mientras no haya que reprogramar y no sea hora de acabar */

   nueva_prog = 0;
   while(!nueva_prog && !acabar)
   {
      /* Activar senal y temporizador de sobretiempo */

      cambiar_senal(1);
      timer_settime(tduty, 0, &prog_duty, NULL);

      /* Esperar senal de "duty cycle" y desactivar senal */

      sigwaitinfo(&sig_duty, NULL);
      cambiar_senal(0);

      /* Esperar temporizador ciclico o bien SIGALRM */
      /* Actuar segun la senal */

      do
      {
        sig = sigwaitinfo(&sig_cic_o_timeout, NULL);
        if(sig == SIGALRM)
        {
          printf("Sobretiempo detectado\n"); fflush(stdout);
          timeout_pendiente = 1;          /* Hay que acabar el ciclo */
        }
        else
        {
          /* Senal del temporizador de ciclo: Fin de ciclo */

          if(timeout_pendiente == 1)
          {
            /* Si ya habia saltado el timeout, acabar */

            printf("Ejecutando sobretiempo pendiente\n"); fflush(stdout);
            acabar = 1;
          }
          else
          {
            /* Comprobar si hay otro mensaje de especificacion; si
               lo hay, hay que reprogramar temporizadores */

            res = mq_receive(cola, (char *)&datos, sizeof(datos), &pri);
            if(res != -1)
            {
              printf("Leyendo de cola\n"); fflush(stdout);

              alarm(timeout);          /* Reiniciar sobretiempo */
              nueva_prog = 1;
              printf("Recibo ciclo %ld, dcic %f\n",
                      datos.ciclo, datos.dcic);
              fflush(stdout);

            }
          }
        }
      } while(sig != SIGRTMIN);  /* Hay que esperar fin de ciclo */
   }
}

printf("Acabando\n");
mq_close(cola);
mq_unlink(a2[1]);
```

```
   exit(0);
}

/* Cambiar valor de la senal e imprimir medida de tiempo para comprobar */

void cambiar_senal(int valor)
{
    struct timespec instante;
    static struct timespec referencia = {0, -1};
    clock_gettime(CLOCK_REALTIME, &instante);
    if(referencia.tv_nsec == -1) referencia = instante;
    printf("Senal cambiada a %d en %f\n", valor,
            (instante.tv_sec - referencia.tv_sec) +
            (double)(instante.tv_nsec-referencia.tv_nsec)*1e-9);
    fflush(stdout);
}

/* Prueba */

void enviar_cola(char *ncola)
{
    mqd_t cola;
    struct param datos[] = { {2000, 25.}, {2000, 50.} };
    int n;
    int i;

    n = sizeof(datos)/sizeof(struct param);
    cola = mq_open(ncola, O_WRONLY, 0, NULL);
    sleep(3);
    for(i=0; i < n; i++)
    {
       printf("enviando %ld %f\n", datos[i].ciclo, datos[i].dcic);
       fflush(stdout);
       mq_send(cola, (char *)&datos[i], sizeof(struct param), 0);
       sleep(10);
    }

    printf("enviar_cola acabando\n");
}
```

```
/* Problema 5/9/11; cabecera problema.h */

#define LCOLA 10   /* Tamano de la cola */

struct param
{
   long ciclo;    /* Ciclo en milisegundos */
   float dcic;    /* Duty cycle en % del ciclo (50 es 50%) */
};

void cambiar_senal(int valor);  /* Cambiar salida a valor 0 o 1 */
```

```
timeout de 11
ciclo de 1 sg y 0 ms
Tiempo activo: 0 sg 500 ms
Senal cambiada a 1 en 0.000000
Senal cambiada a 0 en 0.501923
Senal cambiada a 1 en 1.001847
Senal cambiada a 0 en 1.503770
Senal cambiada a 1 en 2.001694
Senal cambiada a 0 en 2.503617
enviando 2000 25.000000
Senal cambiada a 1 en 3.001541
Senal cambiada a 0 en 3.503464
Leyendo de cola
Recibo ciclo 2000, dcic 25.000000
ciclo de 2 sg y 0 ms
Tiempo activo: 0 sg 500 ms
Senal cambiada a 1 en 4.001388
Senal cambiada a 0 en 4.503311
Senal cambiada a 1 en 6.003081
Senal cambiada a 0 en 6.505005
Senal cambiada a 1 en 8.002775
Senal cambiada a 0 en 8.504699
Senal cambiada a 1 en 10.002469
Senal cambiada a 0 en 10.504393
Senal cambiada a 1 en 12.003163
Senal cambiada a 0 en 12.505086
enviando 2000 50.000000
Leyendo de cola
Recibo ciclo 2000, dcic 50.000000
ciclo de 2 sg y 0 ms
Tiempo activo: 1 sg 0 ms
Senal cambiada a 1 en 14.002857
Senal cambiada a 0 en 15.004704
Senal cambiada a 1 en 16.004551
Senal cambiada a 0 en 17.006398
Senal cambiada a 1 en 18.004245
Senal cambiada a 0 en 19.006092
Senal cambiada a 1 en 20.003939
Senal cambiada a 0 en 21.005786
Senal cambiada a 1 en 22.004633
enviar_cola acabando
Senal cambiada a 0 en 23.006479
Senal cambiada a 1 en 24.004327
Senal cambiada a 0 en 25.006173
Sobretiempo detectado
Ejecutando sobretiempo pendiente
Acabando
```