```c
/* Problema 22/12/08 */

#define _POSIX_C_SOURCE 199506L

#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>
#include <time.h>
#include <pthread.h>

#include "problema.h"

/* Constantes simbolicas */


/* Pulsadores del grafico (no se pedia en el enunciado) */

struct puls
{
    int fila;                /* Fila y columna en el grafico */
    int col;
    int ipunto;           /* Punto de carga/descarga asociado */
    int incsenal;        /* Incremento del num. de senal respecto de SIGRTMIN */
    elemento_t id;     /* Elemento grafico */
};

#define N_PULS (N_PUNTOS*2)

struct puls pulsadores[N_PULS] =
{
   {
      3, 0, 0, 0, -1
   },
   {
      3, 1, 1, 0, -1
   },
   {
      3, 2, 2, 0, -1
   },
   {
      3, 12, 0, 1, -1
   },
   {
      3, 13, 1, 1, -1
   },
   {
      3, 14, 2, 1, -1
   }
};

/* Variables compartidas: peticion de carga y descarga */

struct pet {                /* Tipo para describir la peticion */
  int ind_carga;           /* Punto de carga */
  int ind_descarga;     /* Punto de descarga */
  int pet_pendiente;    /* Indicador de peticion pendiente de aceptar */
};

pthread_mutex_t  acceso_peticion      =    PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t     peticion_pendiente =    PTHREAD_COND_INITIALIZER;
pthread_cond_t     peticion_aceptada  =    PTHREAD_COND_INITIALIZER;

struct pet ultima_pet = {-1, -1, 0};   /* Peticion */
int fin = 0;

/* Mutex para el pasillo */

pthread_mutex_t   acceso_pasillo = PTHREAD_MUTEX_INITIALIZER;
```



```c
int              contador_movimientos = 0;

/* Manejador  de t. real */

void manej(int signo, siginfo_t *info, void *extra)
{
    printf("Salta el manejador manej para %d", signo);
}

/* Manejador no de t. real */

void manej2(int signo)
{
    printf("Salta el manejador manej2 para %d", signo);
}


/* Funciones de arranque de hilos */

void *hilo_control(void *p);
void *hilo_peticiones(void *p);

/* main */

int main(void)
{
   pthread_t hpets;                    /* Hilo de peticiones */
   pthread_t hcontrol[N_VEH];   /* Hilos de control de vehiculo */
   sigset_t parada;                     /* Senales de parada */
   void *rhilo;
   int i;
   sigset_t todas;
   char nom[50];
   struct sigaction accion;
   int res;

   fprintf(stdout, "practica3: arrancando graficos\n");

   /* Cambia mapa y despues inicializa graficos
       (no se pedia en el enunciado)                        */

   cambia_escala_dib(ESCALA_DIB);
   cambia_mapa((int *)mapa, ANCHO, ALTO);
   ini_grafi(1);

   fprintf(stdout, "graficos inicializados\n");

   /* Senales utilizadas:
       SIGTERM, SIGINT, SIGRTMIN, SIGRTMIN+1
       Se bloquean para tratamiento sincrono */

   accion.sa_sigaction = manej;
   sigemptyset(&accion.sa_mask);
   accion.sa_flags = SA_SIGINFO;
   sigaction(SIGRTMIN, &accion, NULL);
   sigaction(SIGRTMIN+1, &accion, NULL);

   accion.sa_handler = manej2;
   accion.sa_flags = 0;
   sigaction(SIGTERM, &accion, NULL);
   sigaction(SIGINT, &accion, NULL);

   sigemptyset(&todas);
   sigaddset(&todas, SIGRTMIN);
   sigaddset(&todas, SIGRTMIN+1);
   sigaddset(&todas, SIGTERM);
   sigaddset(&todas, SIGINT);
   sigprocmask(SIG_BLOCK, &todas, NULL);
```

```
   /* Senales de parada */

   sigemptyset(&parada);
   sigaddset(&parada, SIGTERM);
   sigaddset(&parada, SIGINT);

   /* Creacion de pulsadores (no se pedia en el enunciado) */

   for(i=0; i<N_PULS; i++)
   {
      sprintf(nom, "P%d", pulsadores[i].ipunto);
      pulsadores[i].id = crear_pulsador_codigo(pulsadores[i].fila,
                                               pulsadores[i].col,
                                               SIGRTMIN+pulsadores[i].incsenal,
                                               nom, pulsadores[i].ipunto);
   }

  /* Arrancar hilo de pulsadores */

  pthread_create(&hpets, NULL, hilo_peticiones, NULL);

  /* Arrancar hilos de control de vehiculos */

  for(i=0; i<N_VEH; i++)
    pthread_create(hcontrol+i, NULL, hilo_control, (void *)i);

   /* Esperar senal de fin */

   sigwaitinfo(&parada, NULL);

   printf("parando aplicacion\n");
   pthread_mutex_lock(&acceso_peticion);
   fin = 1;                                                          /* Indicar fin */
   pthread_cond_broadcast(&peticion_pendiente);   /* Desbloquear vehiculos */
   pthread_cond_signal(&peticion_aceptada);
   kill(getpid(), SIGRTMIN);
   kill(getpid(), SIGRTMIN+1);
   pthread_mutex_unlock(&acceso_peticion);

  /* Esperar terminacion de hilos */

  printf("esperando hilos\n");
  for(i=0; i<N_VEH; i++)
     pthread_join(hcontrol[i], &rhilo);
  pthread_join(hpets, &rhilo);

   /* Cerrar graficos */

   printf("parando graficos\n");
   fin_grafi();
   printf("acabando\n");

   return 0;
}


/* Hilo de control de vehiculo; el argumento
    indica el vehiculo controlado                        */

void *hilo_control(void *p)
{
   int i = (int)p;
   int origen, destino;
   elemento_t idveh;

    /* Crear grafico (no se pedia en el enunciado) */

    idveh = crear_vehiculo(FILA_VUELTA, COL_APARC0+i);
```



```
   printf("vehiculo %d arrancado\n", i);


   while(!fin)
   {
      /* Esperando a que haya peticiones pendientes (o fin) */

      printf("vehiculo %d: Esperando peticion\n", i);

      pthread_mutex_lock(&acceso_peticion);
      while(!ultima_pet.pet_pendiente && !fin)
      {
          pthread_cond_wait(&peticion_pendiente, &acceso_peticion);
      }

      if(!fin)
      {
         origen = ultima_pet.ind_carga;
         destino = ultima_pet.ind_descarga;
         ultima_pet.pet_pendiente = 0;
         pthread_cond_signal(&peticion_aceptada);
      }
      pthread_mutex_unlock(&acceso_peticion);

      if(fin) continue;   /* Saltar todo lo que queda de la iteracion */

      printf("vehiculo %d: Carga en %d y descarga en %d\n", i, origen, destino);

      /* Ir a carga */
      pthread_mutex_lock(&acceso_pasillo);
      mueve(idveh, FILA_IDA, COL_APARC0+i);
      mueve(idveh, FILA_IDA, COL_ESPERA1);
      pthread_mutex_unlock(&acceso_pasillo);
      mueve(idveh, FILA_IDA, COL_CARGA0+origen);
      carga(idveh);
      mueve(idveh, FILA_IDA, COL_CARGA0);
      mueve(idveh, FILA_VUELTA, COL_CARGA0);

      printf("vehiculo %d: Cargado\n", i);

      /* Ir a descarga */
      mueve(idveh, FILA_VUELTA, COL_ESPERA1);

      pthread_mutex_lock(&acceso_pasillo);
      mueve(idveh, FILA_VUELTA, COL_ESPERA1+1);
      mueve(idveh, FILA_IDA, COL_ESPERA1+1);
      mueve(idveh, FILA_IDA, COL_ESPERA2);
      pthread_mutex_unlock(&acceso_pasillo);

      mueve(idveh, FILA_IDA, COL_DESCARGA0+destino);
      descarga(idveh);

      printf("vehiculo %d: Descargado\n", i);

      /* Vuelta */
      mueve(idveh, FILA_IDA, COL_DESCARGA0+N_PUNTOS-1);
      mueve(idveh, FILA_VUELTA, COL_DESCARGA0+N_PUNTOS-1);
      mueve(idveh, FILA_VUELTA, COL_ESPERA2);

      pthread_mutex_lock(&acceso_pasillo);
      mueve(idveh, FILA_VUELTA, COL_ESPERA2-1);
      mueve(idveh, FILA_IDA, COL_ESPERA2-1);
      mueve(idveh, FILA_IDA, COL_APARC0+i);
      mueve(idveh, FILA_VUELTA, COL_APARC0+i);
      pthread_mutex_unlock(&acceso_pasillo);

   }
```

```
    printf("vehiculo %d: acabando\n", i);

    return NULL;
}

/* Hilo para recibir senales de pulsadores y transformarlas
    en peticiones                                                    */

void *hilo_peticiones(void *p)
{
    sigset_t sig_orig;
    sigset_t sig_destino;
    siginfo_t valor;
    int origen;
    int destino;

    printf("hilo de peticiones arrancado\n");

    /* Hay que esperar SIG_PULS */
    /* Los de origen envian SIGRTMIN, y los
        de destino, SIGRTMIN+1, como se especifica
        en el enunciado */

    sigemptyset(&sig_orig);
    sigaddset(&sig_orig, SIGRTMIN);

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

    /* Bucle principal */

    while(!fin)
    {
       printf("hilo_pulsadores: Esperando pulsador\n");

       /* Esperar algun pulsador (o fin) */

       while(sigwaitinfo(&sig_orig, &valor) == -1);
       if(fin) continue;
       origen = valor.si_value.sival_int;

       printf("hilo_pulsadores: Origen %d\n", origen);

       while(sigwaitinfo(&sig_destino, &valor) == -1);
       if(fin) continue;
       destino = valor.si_value.sival_int;

       printf("hilo_pulsadores: Destino %d\n", destino);

       /* Activar peticion en datos compartidos */
       /* Vale pthread_cond_signal porque solo sirve para uno */

       pthread_mutex_lock(&acceso_peticion);
       ultima_pet.pet_pendiente = 1;
       ultima_pet.ind_carga = origen;
       ultima_pet.ind_descarga = destino;
       pthread_cond_signal(&peticion_pendiente);
       pthread_mutex_unlock(&acceso_peticion);

       printf("hilo_pulsadores: Enviada peticion\n");

        /* Esperar a que se acepte la peticion antes de
            reconocer mas pulsaciones                              */

       pthread_mutex_lock(&acceso_peticion);
       while(ultima_pet.pet_pendiente && !fin)
       {
           pthread_cond_wait(&peticion_aceptada, &acceso_peticion);
       }
```



```
       pthread_mutex_unlock(&acceso_peticion);

       printf("hilo_pulsadores: Peticion aceptada\n");

    }

    printf("hilo_pulsadores: acabando\n");

    return NULL;

}


/* Funciones de carga y descarga (no se pedian) */

void carga(int veh)
{
   sleep(1);
   poner_cont(veh, 1);
}

void descarga(int veh)
{
    sleep(1);
   poner_cont(veh, 0);
}
```

```c
/* Cabecera para el problema de 22/12 */

#ifndef __ROBOT_H__
#define __ROBOT_H__

/* Acceso a interfaz grafica */

#include "api_grafi.h"

/* Macros del enunciado */

#define N_VEH 3
#define N_PUNTOS 3

#define FILA_IDA 1
#define FILA_VUELTA 2
#define COL_APARC0  6
#define COL_CARGA0  0
#define COL_DESCARGA0 12
#define COL_ESPERA1 3
#define COL_ESPERA2 11
#define COL_APARC0   6

/* Funciones de carga y descarga */

void carga(int veh);
void descarga(int veh);

/* Definiciones relacionadas con los graficos, sin
    relacion con el enunciado */

/* Mapa:
    C: camino
    P: Pared
    A: Zona de aparcamiento */

#define P PARED                 /* Definido en api_grafi.h */
#define C PASILLO               /*   "              "           "     */

#define ESCALA_DIB 30

#define ALTO 4
#define ANCHO 15

int mapa[ALTO][ANCHO] =
                {
                    {C, C, C, P, P, P, P, P, P, P, P, P, C, C, C},
                    {C, C, C, C, C, C, C, C, C, C, C, C, C, C, C},
                    {C, C, C, C, C, P, C, C, C, P, C, C, C, C, C},
                    {P, P, P, P, P, P, P, P, P, P, P, P, P, P, P},
                } ;


#endif
```