```c
/* Tratamiento de senales con hilos */

#define _POSIX_C_SOURCE 199506L
#define _REENTRANT

#include <string.h>

#include <unistd.h>
#include <pthread.h>
#include <stdio.h>
#include <time.h>
#include <signal.h>
#include <errno.h>
#include <math.h>

#define NTH 4

struct timespec tim = {0, 200000000L}; /* 200 milisegundos */

/* Rutinas de hilo */

void *hilo(void *p);
void *hilo2(void *p);
void *erroneo(void *p);

/* Manejadores */

void manej(int signo, siginfo_t *datos, void *extra);
void errorseg(int signo);

pthread_t otroth;

void main(int argc, char *argv[])
{
  int res;
  int i;
  int j;
  int cnt;
  int isenal;
  void *result;
  pthread_t vth[NTH];
  sigset_t set;
  struct sigaction accion;

  /* Programacion de los manejadores */

  sigemptyset(&set);
  for(i=SIGRTMIN; i<=SIGRTMAX; i++)
  {
    sigaddset(&set, i);             /* Conjunto con todas las S.T.R. */

    accion.sa_sigaction = manej;    /* Todas tienen el mismo manejador */
    sigemptyset(&(accion.sa_mask));
    accion.sa_flags = SA_SIGINFO;
    sigaction(i, &accion, NULL);
  }

  /* Preparacion de la mascara del hilo "main": No ve ninguna senal
     de tiempo real       */

  pthread_sigmask(SIG_BLOCK, &set, NULL);

  /* Primera prueba: kill */
```



```c
  /* Arrancan NTH threads */

  printf("\nPrimera prueba: senales generadas con kill\n\n");

  for(i = 0; i < NTH; i++) pthread_create(vth + i, NULL, hilo, (void *)i);

  for(i=0; i < NTH; i++)
  {
    /* El proceso se autosenala */

    printf("main enviando senal %d\n", i+SIGRTMIN);

    kill(getpid(), i+SIGRTMIN);
    nanosleep(&tim, NULL);       /* Espera un tiempo */
  }

  /* Ahora cancela los hilos */

  for(i=0; i < NTH; i++)
  {
    printf("main cancelando al hilo %d\n", i);
    pthread_cancel(vth[i]);
    pthread_join(vth[i], NULL);
  }

  nanosleep(&tim, NULL);

  /* Segunda prueba: Senales generadas con pthread_kill */

  printf("\nSegunda prueba: senales generadas con pthread_kill\n\n");

  for(i=0; i < NTH; i++) pthread_create(vth + i, NULL, hilo, (void *)i);

  nanosleep(&tim, NULL);

  for(i=0; i<NTH; i++)
  {
    /* Senales a los diferentes threads */

    printf("Main enviando senal %d a los %d threads\n", i + SIGRTMIN, NTH);
    for(j=0; j<NTH; j++)
    {
      if(res = pthread_kill(vth[j], i+SIGRTMIN))
        printf("Main; error %d en pthread_kill\n", res);
    }
    nanosleep(&tim, NULL);  /* Espera un tiempo */
  }

  /* Ahora cancela los hilos */

  for(i=0; i < NTH; i++)
  {
    printf("cancelando el hilo %d\n", i);
    pthread_cancel(vth[i]);
    pthread_join(vth[i], NULL);
  }
  nanosleep(&tim, NULL);

  /* Tercera prueba: Senales capturadas con manejador */

  printf("\nTercera prueba: Senales capturadas con manejador\n\n");

  for(i=0; i < NTH; i++) pthread_create(vth + i, NULL, hilo2, (void *)i);
```

```
  nanosleep(&tim, NULL);

  i = 0; cnt = 1;

  for(i=0; i<NTH; i++)
  {
    /* Senales dirigidas a los diferentes threads */

    printf("Main enviando senal %d\n", i + SIGRTMIN);
    for(j=0; j<NTH; j++)
    {
      if(res = pthread_kill(vth[j], i+SIGRTMIN))
        printf("Main; error %d en pthread_kill\n", res);

      nanosleep(&tim, NULL);  /* Espera un tiempo */
    }
  }

  printf("\nCuarta prueba: Error de segmentacion\n\n");

  pthread_create(&otroth, NULL, erroneo, (void *)0);

  /* Fin del programa */

  for(i=0; i<20; i++) nanosleep(&tim, NULL);
  printf("Soy main; voy a acabar\n");
}

/* Manejador */

void manej(int signo, siginfo_t *datos, void *extrA)
{
  printf("El manejador recibe la senal %d\n", signo);
}

/* Rutina del hilo que utiliza sigwaitinfo */

void *hilo(void *arg)
{
  int ident = (int)arg;     /* Identificador de 0 a NTH-1 */
  sigset_t set;
  siginfo_t sigin;
  int res;

  /* Todas las senales necesarias estan ya bloqueadas */

  sigemptyset(&set);
  sigaddset(&set, ident + SIGRTMIN);

  /* El hilo solo espera senales */

  printf("El hilo %d va a esperar senales %d\n", ident, ident + SIGRTMIN);

  while(1)
  {
    if((res = sigwaitinfo(&set, NULL)) != -1)
         printf("El hilo %d ha recibido la senal prevista %d\n", ident, res);
    else printf("El hilo %d ha recibido senal no prevista\n");
    pthread_testcancel();
  }

  return NULL;
}
```



```
/* Rutina del hilo para utilizar manejador */

void *hilo2(void *arg)
{
  int ident = (int)arg;     /* Identificador de 0 a NTH-1 */
  sigset_t set;
  siginfo_t sigin;
  int res;
  int i;

  /* Estan bloqueadas todas las senales de T.R. que se utilizan */
  /* Se desbloquea la apropiada para utilizar el manejador */

  sigemptyset(&set);
  sigaddset(&set, ident+SIGRTMIN);
  pthread_sigmask(SIG_UNBLOCK, &set, NULL);

  printf("El hilo %d va a admitir senales %d\n", ident, ident + SIGRTMIN);

  while(1)
  {
    if(nanosleep(&tim, NULL) == -1)
      printf("nanosleep de %d interrumpido: %s\n", ident, strerror(errno));

    pthread_testcancel();
  }

  return NULL;
}

/* Manejador del error de segmentacion */

void errorseg(int signo)
{
  /* pthread_self podria no funcionar; esto es
     un manejador. En Solaris 2.6  SI funciona  */

  printf("Detectado error de segmentacion!\n");
  if(pthread_equal(otroth, pthread_self()))
   printf("El thread erroneo es el culpable.\n");

  exit(1);  /* Error irrecuperable */
}

/* Thread que genera el error de segmentacion */

void *erroneo(void *p1)
{
  float f;
  struct sigaction accion;
  float *p;

  sigset_t set;

  /* Desbloquea la senal de error de segmentacion
     (para este hilo)                                */

  sigemptyset(&set);
  sigaddset(&set, SIGSEGV);
  pthread_sigmask(SIG_UNBLOCK, &set, NULL);

  /* Programa la accion a realizar */

  accion.sa_handler = errorseg;
```

```
  accion.sa_flags = 0;
  sigemptyset(&(accion.sa_mask));
  sigaction(SIGSEGV, &accion, NULL);

  p = NULL;

  /* ERROR */

  f = *p;

  return NULL;
}
```