Configuración de una Señal:

```
#include <signal.h>

main()
{
	struct sigaction act;   // manejador señal
	sigset_t sigset;  // conjunto de señales

	// Primero bloquea la señal
	sigemptyset(&sigset);   // crea una máscara vacía
	sigaddset(&sigset, SIGUSR1);  // añade la señal
	pthread_sigmask (SIG_BLOCK, &sigset, NULL); // bloqueo

	// Configura la señal
	act.sa_sigaction = ManejadorSig;    // manejador de la señal
	sigemptyset(&(act.sa_mask));        // máscara vacía
	act.sa_flags = SA_SIGINFO;          // Modo extendido

	if(sigaction(SIGUSR1, &act, NULL)<0)     // configuración
		perror("Sigaction fallado");

	// desboquea la señal
	sigemptyset(&sigset);
	sigaddset(&sigset, SIGUSR1);
	pthread_sigmask (SIG_UNBLOCK, &sigset, NULL); //desbloqueo

	while(1)
	{
		// espera indefinidamente la aparición de la señal
	}

}

/* Manejador de la Señal */
void ManejadorSig( int signo, siginfo_t *info, void *context)
{
	printf("Soy el manejador de la señal # %d Valor: %d ",
		info->si_signo, info->si_value.sival_int);

	printf("Code: (%d) ",info->si_code);
	if( info->si_code == SI_USER)
		printf("SI_USER \n" );
	else if( info->si_code == SI_TIMER )
		printf("SI_TIMER \n" );
	else if( info->si_code == SI_QUEUE )
		printf("SI_QUEUE \n" );
	else if( info->si_code == SI_ASYNCIO )
		printf("SI_ASYNCIO \n" );
	else if( info->si_code == SI_MESGQ )
		printf("SI_MESGQ \n" );
}
```

Ejemplo de espera de una señal:

```
#define _REENTRANT

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <sched.h>
#include <pthread.h>
#include <signal.h>
#include <time.h>
#include <sys/time.h>

#define SIGNUM          SIGRTMAX
#define ESPERA_DEF      20000L            // 20s

void ManejadorSig( int signo, siginfo_t *info, void *context);
void * ThreadSig(void *arg);
void TestArg(int argc, char *argv[], long *esp);
void wait_t(long ms);

/*****************************
      función main
 *****************************/
int main(int argc, char *argv[])
{
      sigset_t sigset;  // conjunto de señales
      pthread_t th_id;  // identificador para el thread
      pthread_attr_t attr;    // atributos de los threads
      int sig =  SIGNUM;
      int ires;
      long esp;

      // comprueba los parámetros
      TestArg(argc, argv, &esp);

      // bloquea la señal
      // los threads creados heredan la máscara
      sigemptyset(&sigset);
      sigaddset(&sigset, sig);
      pthread_sigmask(SIG_BLOCK, &sigset, NULL);

      //inicializa los parámetros de los threads por defecto
      pthread_attr_init (&attr);
      // crea el thread de gestión de la señal (hereda la máscara)
      ires=pthread_create (&th_id, &attr, ThreadSig, (void *)sig);

      wait_t(esp);  // espera un rato

      // termina el thread de señal
      if (pthread_cancel(th_id)!=0)
            perror("Cancelando thread");
      else
            printf("Thread de señal, # %d, ha terminado\n",th_id);

      wait_t(esp);  // espera un rato
      printf("Main thread, # %d termina\n", pthread_self());
}
```

// -→ Continua

// -→ Continuación

```
/*************************************************************
 Thread de gestión de la señal
 *************************************************************/
void * ThreadSig(void *arg)
{
      int sig;
      struct sigaction act;   // manejador señal
      sigset_t sigset;  // conjunto de señales
      int res;

      sig = (int)arg;


      act.sa_sigaction = ManejadorSig;    // manejador
      sigemptyset(&(act.sa_mask));  // sin bloqueos adicionales
      act.sa_flags = SA_SIGINFO;    // información extendida

      if(sigaction(SIGNUM, &act, NULL)<0)
      {
            perror("Sigaction fallado");
            pthread_exit(NULL);
      }

      printf(" Señal %d atrapada por el proceso: %d  thread: %d\n",
                  sig, getpid(), pthread_self());

      // permite la cancelación del thread
      pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, NULL);
      pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);

      // define el conjunto para la espera
      // Lee los bloqueos actuales
      sigemptyset(&sigset);
      sigaddset(&sigset, sig);      // desactiva la señal

      // Espera la señal o la cancelación del thread
      while(1)
      {
          // sigwait es un punto de cancelación
          res= sigwait(&sigset, &sig);

          if (res!=-1)
              printf(" Señal %d depositada en el thread: %d\n",
                        sig, pthread_self());
      }

}
```

// -→ Continua

// -→ Continuación

```
/*************************************************************
 Manejador de la Señal signo
 *************************************************************/
void ManejadorSig( int signo, siginfo_t *info, void *context)
{
      printf("Soy el manejador de la señal # %d Valor: %d ",
            info->si_signo, info->si_value.sival_int);
}


/*************************************************************
 Comprueba los parámetros y asigna su valor
 *************************************************************/
void TestArg(int argc, char *argv[], long *esp)
{
        if(argc > 2)
        {   printf("Utilizar %s [espera(ms)]\n", argv[0]);
            exit(0); }
        if(argc==2)  *esp = atol(argv[1]);
        else      *esp = ESPERA_DEF;      // espera por defecto
}


/*************************************************************
 Espera ms_ret milisegundos (no bloquea con threads de usuario)
 *************************************************************/
void wait_t(long ms_ret)
{
      timespec_t t1;
      double t_actual, t_final;

      clock_gettime(CLOCK_REALTIME, &t1);
      t_actual= t1.tv_sec + (double) (t1.tv_nsec) /1000000000L;
      t_final     = t_actual + (double)ms_ret/1000;

      do {
         sched_yield(); /* libera la CPU */
         clock_gettime(CLOCK_REALTIME, &t1);
         t_actual= t1.tv_sec + (double) (t1.tv_nsec) /1000000000L;
      }while (t_actual<t_final);
}
```

// Final del Programa

RESUMEN FUNCIONES DE MANEJO DE SEÑALES

FUNCIÓN	CABECERA	DESCRIPCIÓN
int ***kill****(pid_t pid, int sig);*	*<signal.h>*	Envía una señal a un proceso
int ***sigsend****(idtype_t idtype, id_t id, int sig);*		Envía una señal a un conjunto de procesos
int ***sigqueue****(pid_t pid, int signo, const union sigval value);*		Envía una señal a un proceso junto con un valor al manejador. La señal es encolable
int ***sigemptyset****(sigset_t *set);*	*<signal.h>*	Configura un conjunto de señales vacío (todas desactivas)
int ***sigfillset****(sigset_t *set);*		Configura un conjunto de señales con todas activadas
int ***sigaddset****(sigset_t *set, int signo);*		Añade una señal a un conjunto (activa)
int ***sigdelset****(sigset_t *set, int signo);*		Borra una señal de un conjunto (desactiva)
int ***sigprocmask****(int how, const sigset_t *set, sigset_t *oset);*	*<signal.h>*	Configura la mascara de bloqueo
int ***pthread_sigmask****(int how, const sigset_t *set, sigset_t *oset);*		Configura la mascara de bloqueo
int ***sigaction****(int sig, const struct sigaction *act, struct sigaction *oact);*		Configura el manejador de una señal
int ***sigsuspend****(const sigset_t *set);*	*<signal.h>*	Suspende un proceso o thread hasta que sea depositada una señal
int ***sigwait****(sigset_t *set);*		Suspende un proceso o thread hasta que sea depositada una señal. Devuelve el número de la señal.
int ***sigwait****(const sigset_t *set, int *sig); (-D_POSIX_PTHREAD_SEMANTICS)*		
int ***sigwaitinfo****(const sigset_t *set, siginfo_t *info);*		Suspende un proceso o thread hasta que sea depositada una señal de tiempo real. Devuelve una estructura con información adicional.
int ***sigtimedwait****(const sigset_t *set, siginfo_t *info, const struct timespec *timeout);*		Suspende un proceso o thread hasta que sea depositada una señal de tiempo real. Se especifica un intervalo de espera máximo. Devuelve una estructura con información adicional.