

Temporizadores POSIX

Uso de *Señales*, *Temporizadores* y *Threads* para ejecutar tareas periódicas en POSIX

1

Tareas Periódicas en POSIX

- POSIX no implementa un mecanismo directo para especificar tareas periódicas como en [ADA95](#) o [RT-Java](#).
- Pueden implementarse utilizando : threads, señales y temporizadores.
 - El **Thread** especifica el código de la tarea a realizar
 - El **Temporizador** establece un reloj contador que especifica el periodo de ejecución
 - La **Señal** permite notificar y activar el thread con la tarea periódica cuando expira el temporizador

Temporizadores

- Un temporizador es un registro contador asociado a un reloj.
 - Una vez creado se inicializa con el tiempo a contar
 - Cada pulso de reloj decreenta el contador del temporizador.
 - Al llegar a 0 se notifica un evento (Señal) al *proceso* que lo creó y inicializa de nuevo su valor
 - Están asociados al manejo de señales
 - Permiten ejecutar tareas con retardos relativos y absolutos precisos
 - Permiten ejecutar tareas periódicas

Temporizadores POSIX

- Cada temporizador se identifica mediante un valor de tipo *timer_t*, (entero) definido en la cabecera *<time.h>*
- El valor de espera se especifica mediante una estructura de tipo *itimerspec*, cabecera *<sys/time.h>*

```
struct itimerspec {  
    struct timespec it_interval;    /* periodo */  
    struct timespec it_value;      /* valor inicial*/  
};
```

- *it_value*: valor inicial del temporizador (0 lo desactiva)
- *it_interval*: nuevo valor tras expiración (0 un solo evento)

```
struct timespec {  
    time_t tv_sec; /* seconds */  
    long tv_nsec; /* and nanoseconds */  
};
```

Temporizadores

- Antes de usar un temporizador este debe ser creado mediante la siguiente función:

```
int timer_create (clockid_t clock_id, struct sigevent *evp, timer_t *timerid)
```

- Devuelve 0 si el temporizador pudo crearse y -1 en caso contrario
- Esta función crea un temporizador asociado al reloj `clock_id` (`CLOCK_REALTIME`).
- El identificador del temporizador se devuelve en `*timer_id`.
- `*evp` indica el tipo de notificación que se produce al expirar el temporizador (se trata de un manejador de señal). Si `evp` es `NULL` se asigna a la señal `SIGALARM`

```
struct sigevent {  
    int sigev_notify /* notification type */  
    int sigev_signo; /* signal number */  
    union sigval sigev_value; /* signal value */  
};
```

- `sigev_notify` ->
 - `SIGEV_SIGNAL` : se asocia una señal
 - `SIGEV_NONE`: no se notifica nada

Temporizadores

- Una vez creado el temporizador debe ser activado especificando si debe ser absoluto o relativo y el valor inicial. Para ello se dispone de la función:

```
int timer_settime ( timer_t timerid, int flag,  
                   const struct itimerspec *value,  
                   struct itimerspec *ovalue);
```

- El tipo de temporización se selecciona con el parámetro `flag`.
 - `TIMER_RELTIME`: temporizador relativo a la llamada a la función
 - `TIMER_ABSTIME`: temporizador absoluto (Época UNIX)

```
struct itimerspec {  
    struct timespec it_interval;  
    struct timespec it_value;  
};
```

- El valor inicial del temporizador se indica en `value.it_value`
- El funcionamiento se repite periódicamente si `value.it_interval > 0`.
- En `*ovalue` se devuelve el valor que quedaba de la temporización anterior. Puede ser `NULL` si no deseamos obtener este valor

Temporizadores

- Valor que queda para que expire el temporizador:

```
int timer_gettime ( timer_t timerid, struct itimerspec *value);
```

- Devuelve en `value.it_value` el tiempo restante de temporización
- El temporizador se destruye una vez que no sea necesario mediante la función:

```
int timer_delete ( timer_t timerid);
```

Ejemplo

Ejecución temporizada de una tarea en POSIX

- Un temporizador envía una señal cuando expira
- La tarea se asigna al manejador de la señal

:compilar con librerías: `-lposix4 -lpthread`

:compilar en **linux** con librerías: `-lrt -lpthread`

```

#define REENTRANT
#include <pthread.h>
#include <signal.h>
#include <sched.h>
#include <time.h>
#include <sys/time.h>

#define SIGNAL SIGRTMAX

struct timespec RETARDO = { 10, 0L}; // retardo de 10s (ejecución del programa)

struct timespec VAL_TEMPORIZADOR = { 5, 0L}; // retardo temporizador de 5s
struct timespec REINICIO_TEMPORIZADOR = { 0, 0L}; // reinicio temporizador 0 (1 ciclo)

void ManejadorSig ( int signo, siginfo_t *info, void *context);

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

    timer_t temporizador; // id temporizador
    struct sigevent evp; // evento asociado al temporizador
    struct itimerspec param_temp; // parámetros del temporizador

    struct timespec retardo_pend; // retardo pendiente nanosleep

    // Continúa ->

```

SITR: Relojes de Tiempo Real

9

```

// -> Continúa

```

```

// bloquea la señal: pthread_sigmask()
// los threads creados posteriormente heredan la máscara
sigemptyset(&sigset); // crea una máscara vacía
sigaddset(&sigset, SIGNAL); // añade la señal
pthread_sigmask(SIG_BLOCK, &sigset, NULL);
printf("Señal # %d bloqueada por el proceso: %d\n",
        SIGNAL, getpid());

```

```

// Asigna un manejador (Atrapa la señal)
act.sa_sigaction = ManejadorSig;
sigemptyset(&act.sa_mask); // máscara vacía
act.sa_flags = SA_SIGINFO;
if( sigaction(SIGNAL, &act, NULL) < 0) {
    perror("Sigaction ha fallado");
    exit(-1);
}
printf("Señal %d atrapada por el proceso: %d\n", SIGNAL, getpid());

```

```

// Desbloquea la señal: pthread_sigmask()
sigemptyset(&sigset); // crea una máscara vacía
sigaddset(&sigset, SIGNAL); // añade la señal
pthread_sigmask(SIG_UNBLOCK, &sigset, NULL);

printf("Señal # %d Desbloqueada por el proceso: %d. Espero una señal.\n",
        SIGNAL, getpid());

```

```

// Continúa ->

```

10



```

// Crea el temporizador
evp.sigev_notify = SIGEV_SIGNAL;           // tipo de notificación
evp.sigev_signo = SIGNUM;                 // número de la señal
evp.sigev_value.sival_int = 1; // valor pasado al manejador
if (timer_create (CLOCK_REALTIME, &evp, &temporizador)!=0)
{ perror("Error creando el temporizador"); exit(-1); }

printf("Temporizador (señal # %d) Creado en el thread # %d\n", sig, pthread_self());

// Configura el temporizador (5 sec)
param_temp.it_value=VAL_TEMPORIZADOR;
// nuevo valor cuando expira el temporizador 0 (solo un evento)
param_temp.it_interval=REINICIO_TEMPORIZADOR;

if ( timer_settime (temporizador, TIMER_RELTIME, &param_temp, NULL)!=0)
{ perror("Error configurando el temporizador"); exit(-1); }
printf("Temporizador activado en el proceso: %d\n", getpid());

// espera un rato a que espire el temporizador
if( nanosleep (&RETARDO, &retardo_pend)!=0)
{ // Cuando llega la señal al proceso el thread se reactiva antes de terminar el retardo
printf("Retardo pendiente: %d.%09ld sec\n",
retardo_pend.tv_sec, retardo_pend.tv_nsec);
nanosleep(&retardo_pend, NULL); // espera un poco mas
}

printf("Termina el Programa.\n");
exit(0);
}

```



Ejemplo: Temporizador

```

void ManejadorSig( int signo, siginfo_t *info, void *context)
{
    printf("Soy el manejador de la señal #, 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" );
}

```

```

[/u0/sitr/sitr001/signals]temporizador
Señal # 45 bloqueada por el proceso: 27203
Señal # 45 atrapada por el proceso: 27203
Señal # 45 Desbloqueada por el proceso: 27203 Espero una señal
Temporizador activado en el proceso: 27203
Soy el manejador de la señal # 45 Valor: 1 Code: (-3) SI_TIMER
Retardo pendiente: 5.009354335 sec
Temporizador Eliminado en el proceso: 27203
Termina el Programa.
[/u0/sitr/sitr001/signals]

```

Ejemplo: Tareas Periódicas en POSIX

Esquema de activación periódico:

- Crea e inicializa el temporizador con el valor del *periodo*
- El Temporizador envía una señal cuando expira
- Cuando expira se vuelve a inicializar al mismo valor inicial (*periodo*) iniciando la cuenta de nuevo
- La tarea se asigna al manejador de la señal

13

```
#define _REENTRANT
#include <pthread.h>
#include <signal.h>
#include <sched.h>
#include <time.h>
#include <sys/time.h>

#define SIGNUM SIGRTMAX

struct timespec RETARDO = { 10, 0L}; // retardo de 10s (ejecución del programa)
struct timespec PERIODO = {1, 0L}; // retardo temporizador de y valor de reinicio

void ManejadorSig ( int signo, siginfo_t *info, void *context);

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

    timer_t temporizador; // id temporizador
    struct sigevent evp; // evento asociado al temporizador
    struct itimerspec param_temp; // parámetros del temporizador

    struct timespec retardo_pend; // retardo pendiente nanosleep
    int cont;

    // Continúa ->
```

SITR: Relojes de Tiempo Real

14

```
// -> Continúa
```

```
// bloquea la señal: pthread_sigmask()
// los threads creados posteriormente heredan la máscara
sigemptyset(&sigset); // crea una máscara vacía
sigaddset(&sigset, SIGNUM); // añade la señal
pthread_sigmask(SIG_BLOCK, &sigset, NULL);
printf("Señal # %d bloqueada por el proceso: %d\n",
      SIGNUM, getpid());
```

```
// Asigna un manejador (Atrapa la señal)
act.sa_sigaction = ManejadorSig;
sigemptyset(&(act.sa_mask)); // máscara vacía
act.sa_flags = SA_SIGINFO;
if (sigaction(SIGNUM, &act, NULL) < 0) {
    perror("Sigaction ha fallado");
    exit(-1);
}
printf("Señal %d atrapada por el proceso: %d\n", SIGNUM, getpid());
```

```
// Desbloquea la señal: pthread_sigmask()
sigemptyset(&sigset); // crea una máscara vacía
sigaddset(&sigset, SIGNUM); // añade la señal
pthread_sigmask(SIG_UNBLOCK, &sigset, NULL);

printf("Señal # %d Desbloqueada por el proceso: %d. Espero una señal.\n",
      SIGNUM, getpid());
```

```
// Continúa ->
```

15

```
// continúa ->
```

```
// Crea el temporizador
evp.sigev_notify = SIGEV_SIGNAL; // tipo de notificación
evp.sigev_signo = SIGNUM; // número de la señal
evp.sigev_value.sival_int = 1; // valor pasado al manejador
if (timer_create(CLOCK_REALTIME, &evp, &temporizador) != 0)
{ perror("Error creando el temporizador"); exit(-1); }

printf("Temporizador (señal # %d) Creado en el thread # %d\n", sig, pthread_self());
```

```
// Configura el temporizador (periodo)
param_temp.it_value = PERIODO;
// nuevo valor cuando expira el temporizador (periodo)
param_temp.it_interval = PERIODO;

if (timer_settime(temporizador, TIMER_RELTIME, &param_temp, NULL) != 0)
{ perror("Error configurando el temporizador"); exit(-1); }
printf("Temporizador activado en el proceso: %d\n", getpid());
```

```
// continúa ->
```

SITR: Relojes de Tiempo Real

16

```

// espera del proceso de 10s efectivos (el proceso se despierta con cada señal)
retardo_pend=RETARDO; cont=0;
while(retardo_pend.tv_sec>0)
{
    if( nanosleep (&retardo_pend, &retardo_pend)==-1)
    {
        // Cuando llega la señal el thread se reactiva antes de terminar el retardo
        printf("-----Ciclo (%d) Retardo pendiente: %d.%09ld sec\n",
            cont++, retardo_pend.tv_sec, retardo_pend.tv_nsec);
    }
}

```

```

if (timer_delete (temporizador)!=0)
    perror("Error eliminando el temporizador");
else
    printf("Temporizador Eliminado en el proceso: %d\n", getpid());

```

```

printf("Termina el Programa.\n");
exit(0);
}

```

```

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

```

SITR: Relojes de Tiempo Real

17

Ejemplo: Tarea Periódica

```

[/u0/sitr/sitr001/signals]periodic1
Señal # 45 bloqueada por el proceso: 28159
Señal # 45 atrapada por el proceso: 28159
Señal # 45 Desbloqueada por el proceso: 28159 Espero una señal
Temporizador activado en el proceso: 28159
Soy el manejador de la señal # 45 Valor: 1 Code: (-3) SI_TIMER
-----Ciclo (0) Retardo pendiente: 9.003533397 sec
Soy el manejador de la señal # 45 Valor: 1 Code: (-3) SI_TIMER
-----Ciclo (1) Retardo pendiente: 7.993786532 sec
Soy el manejador de la señal # 45 Valor: 1 Code: (-3) SI_TIMER
-----Ciclo (2) Retardo pendiente: 6.993956666 sec
Soy el manejador de la señal # 45 Valor: 1 Code: (-3) SI_TIMER
-----Ciclo (3) Retardo pendiente: 5.994055808 sec
Soy el manejador de la señal # 45 Valor: 1 Code: (-3) SI_TIMER
-----Ciclo (4) Retardo pendiente: 4.994389047 sec
Soy el manejador de la señal # 45 Valor: 1 Code: (-3) SI_TIMER
-----Ciclo (5) Retardo pendiente: 3.994580547 sec
Soy el manejador de la señal # 45 Valor: 1 Code: (-3) SI_TIMER
-----Ciclo (6) Retardo pendiente: 2.994763782 sec
Soy el manejador de la señal # 45 Valor: 1 Code: (-3) SI_TIMER
-----Ciclo (7) Retardo pendiente: 1.994958033 sec
Soy el manejador de la señal # 45 Valor: 1 Code: (-3) SI_TIMER
-----Ciclo (8) Retardo pendiente: 0.995177752 sec
Temporizador Eliminado en el proceso: 28159
Termina el Programa
[/u0/sitr/sitr001/signals]

```

SITR: Relojes de Tiempo Real

18