Master en Computación

Programación Concurrente

Bloque II: Programación concurrente en POSIX

Tema 7. Señales

7.1 Conceptos Básicos

Se trata de un mecanismo de intercambio de mensajes

Señal POSIX: mecanismo por el que un proceso o thread puede ser notificado de, o afectado por, un evento producido en el sistema

- **excepciones detectadas por hardware**
- **expiración de una alarma o temporizador**
- **finalización de una operación de I/O asíncrona**
- **llegada de un mensaje a una cola de mensajes**
- **mensaje enviado desde otro proceso o thread**
- **actividad del terminal, etc.**

Número de señal:

- **Entero positivo que identifica una señal. Existen también constantes predefinidas para dar nombre a las señales**

7.2 Señales Existentes

Algunas señales “no fiables”	
Nombre de Señal	Descripción
SIGABRT	Terminación anormal, p.e. con *abort*()
SIGALRM	Timeout, p.e. con *alarm*()
SIGFPE	Operación aritmética errónea, p.e. división por cero u overflow
SIGILL	Instrucción hardware inválida
SIGINT	Señal de atención interactiva (ctrl-C)
SIGKILL	Señal de terminación (no puede ser “cazada” ni ignorada)
SIGQUIT	Señal de terminación interactiva (terminales)
SIGSEGV	Referencia a memoria inválida
SIGTERM	Señal de terminación
SIGUSR1	Reservada para la aplicación
SIGUSR2	Reservada para la aplicación
SIGBUS	Acceso a una porción indefinida de un objeto de memoria (opcional)

Todas las señales anteriores son “no fiables”, pues se pueden perder si ya hay una señal del mismo número pendiente

Señales de Tiempo Real

Si se soporta la opción de señales de tiempo real, existen señales con números comprendidos entre `SIGRTMIN` y `SIGRTMAX`, que son señales de tiempo real, o “fiables”

Las señales de tiempo real:

- **no se pierden (se encolan)**
- **se aceptan en orden de prioridad (a menor número de señal mayor prioridad)**
- **tienen un campo adicional de información**
- **hay como mínimo 8**

7.3 Generación, Entrega y Aceptación

Cada thread tiene su propia *máscara de señales* que define las señales bloqueadas para ese thread

- **en un proceso monothread sólo existe la máscara de señales del thread principal**

Una señal se *genera*:

- **cuando el evento que causa la señal ocurre**

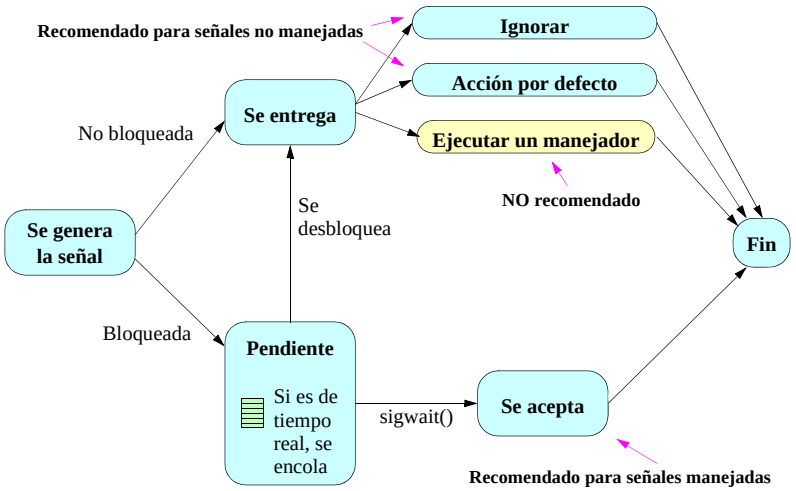
Una señal no bloqueada se *entrega*:

- **cuando la señal causa al proceso la acción asociada: ignorar, terminar el proceso o ejecutar un manejador**

Una señal bloqueada se *acepta*:

- **cuando se selecciona y devuelve por una función `sigwait()`**

Diagrama de funcionamiento



Señales Pendientes

Señal *pendiente:*

- **estado entre su generación y su entrega o aceptación**
- **observable cuando la señal está *bloqueada* (enmascarada)**
- **si ocurre un nuevo evento de una señal pendiente:**
  - **si es una señal "no fiable", el evento puede perderse**
  - **si es una señal "de tiempo real", y la opción `SA_SIGINFO` ha sido especificada (ver `sigaction()`), la señal se encola**

Acciones Asociadas a una Señal

Hay tres tipos de acciones que se pueden asociar a una señal:

- **`SIG_DFL`: acción por defecto (generalmente, terminar el proceso)**
- **`SIG_IGN`: ignorar la señal**
- ***puntero a función*: capturar (o "cazar") la señal, ejecutando un manejador de señal**
  - **al entregarse la señal, se ejecuta el manejador indicado**
  - **una vez finalizado el manejador, se continúa el proceso en el lugar interrumpido**
  - **Los manejadores sólo pueden llamar a funciones "seguras"**

7.4 Señales en Procesos Multi-Thread

Cada thread tiene su propia máscara de señales

- **define las señales bloqueadas para ese thread**
- **la máscara se hereda del thread padre, y se puede modificar**

Una señal se puede generar para un proceso o un thread

- **si el evento es a causa de un thread, se envía a ese thread**
  - **p.e. una excepción hardware**
- **si el evento es asíncrono al proceso, se envía al proceso**
  - **p.e. expiración de un temporizador**
- **las señales generadas en asociación a un `pid` se envían al proceso; en asociación a un `tid`, se envían al thread**

Señales en Procesos Multi-Thread (cont.)

Una señal generada para un thread

- **si no está bloqueada: se entrega**
- **si está bloqueada y la acción no es "ignorar": se queda pendiente hasta que se acepta con `sigwait()` o se desbloquea**
- **si está bloqueada y la acción es "ignorar": no especificado**

Una señal generada para un proceso:

- **si la acción no es ignorar, se envía a un único thread que esté esperando en un `sigwait()`, o a un thread que no tenga la señal bloqueada**
- **si no se entrega, la señal queda pendiente hasta que un thread llame a `sigwait()`, un thread desbloquee la señal, o la acción asociada se ponga a "ignorar"**

Uso Recomendado de Señales en Procesos Multithread

El thread principal enmascara todas las señales que se vayan a usar (salvo las destinadas a finalizar el proceso)

- **los demás threads heredan esta máscara y no la modifican**

No se usan "manejadores de señal"; en su lugar, se crean threads que aceptan señales con `sigwait()`. Ventajas:

- **se puede especificar la prioridad a la que se ejecuta la función manejadora**
- **la función manejadora se ejecuta en un contexto conocido y sin restricciones (el del thread)**
  - **por el contrario, el contexto en que se ejecutan los manejadores de señal no está definido por el estándar y desde ellos sólo se puede invocar funciones "seguras"**

7.5 Funciones para Configurar Señales

Manipular conjuntos de señales:

```
#include <signal.h>
int sigemptyset (sigset_t *set);
int sigfillset  (sigset_t *set);
int sigaddset   (sigset_t *set, int signo);
int sigdelset   (sigset_t *set, int signo);
int sigismember (const sigset_t *set,
                 int signo);
```

Estas funciones no afectan de forma directa al comportamiento del sistema

Cambio de la acción asociada a una señal

Examinar y cambiar la acción asociada a una señal:

```
int sigaction (int sig,
               const struct sigaction *act,
               struct sigaction *oact);
```

- **si `act` es `NULL`, la acción actual no se cambia**
- **si `oact` no es `NULL`, la acción actual se devuelve en `*oact`**

Estructura `sigaction`

```
struct sigaction {
  void (*) (int signo) sa_handler;
    // SIG_DFL, SIG_IGN, o puntero a función
    // manejadora de señal
  sigset_t sa_mask;
    // señales a ser bloqueadas mientras se ejecuta
    // el manejador (además de la señal manejada)
  int sa_flags;
    // si vale SA_SIGINFO la señal se encola y lleva
    // información asociada; el manejador es
    // sa_sigaction
  void (*)(int signo,siginfo_t *info,
           void *context) sa_sigaction;
    // puntero a función manejadora de señal
    // (SA_SIGINFO). context no está especificado por
    // el estándar
};
```

Configurar una señal de tiempo real

La función `sigaction()` nos permite configurar una señal para que tenga comportamiento de tiempo real:

- no se pierde (se encola)
- se acepta en orden de prioridad (a menor número de señal mayor prioridad)
- tiene un campo adicional de información
- señales comprendidas entre `SIGRTMIN` y `SIGRTMAX`

```
// configura la señal SIGRTMIN para que tenga
// comportamiento de tiempo real y no instala
// manejador (se utilizará sigwait)
struct sigaction sigact;
sigact.sa_handler=SIG_DFL;
sigact.sa_flags=SA_SIGINFO;
CHKE( sigaction(SIGRTMIN, &sigact , NULL) );
```


Información Asociada a una Señal de TR

Las señales de tiempo real pueden llevar información asociada
- representada mediante una estructura `siginfo_t`

```
typedef struct {
  int si_signo; // número de señal
  int si_code;  // causa de la señal:
    // SI_USER: enviada por kill()
    // SI_QUEUE: enviada por sigqueue()
    // SI_TIMER: expiración de un temporizador
    // SI_ASYNCIO: fin de operación de I/O asíncrona
    // SI_MESGQ: llegada de mensaje a cola vacía
  union sigval si_value; // valor asociado a la señal
} siginfo_t;
```

```
union sigval {
   int sival_int
   void *sival_ptr
};
```


Cambio de la máscara de señal

Examinar y cambiar señales bloqueadas:

```
int pthread_sigmask (int how,
                     const sigset_t *set,
                     sigset_t *oset);
```

- how indica la forma de operar:
  - `SIG_BLOCK`: señales bloqueadas = anteriores + nuevas
  - `SIG_UNBLOCK`: señales bloqueadas = anteriores - nuevas
  - `SIG_SETMASK`: señales bloqueadas = nuevas
- `set`: señales nuevas; si vale `NULL`, no se cambian
- `oset`: si no es `NULL`, devuelve señales bloqueadas anteriores
- existe una función relacionada (`sigprocmask()`) que sólo debe ser utilizada en procesos mono-thread

7.6 Funciones para Envío y Aceptación de Señales

Enviar una señal a un proceso:

```
#include <sys/types.h>
#include <signal.h>
int kill (pid_t pid, int sig);
int sigqueue (pid_t pid, int signo,
              const union sigval value);
```

Enviar una señal a un thread:

```
int pthread_kill(pthread_t thread, int sig);
```

Examinar señales pendientes:

```
int sigpending (sigset_t *set);
```

- Señales pendientes para el proceso + señales pendientes para el thread

Envío y Aceptación de Señales (cont.)

Esperar a que se ejecute un manejador de señal:

```
int pause (void);
int sigsuspend (const sigset_t *sigmask);
```

- las señales bloqueadas se cambian a `sigmask`, y vuelven al valor original cuando `sigsuspend()` retorna

Aceptar una señal bloqueada:

```
int sigwait (const sigset_t *set, int *sig);

int sigwaitinfo (const sigset_t *set,
                 siginfo_t *info);

int sigtimedwait (const sigset_t *set,
                  siginfo_t *info,
                  const struct timespec *timeout);
```


7.7 Ejemplo del Uso de Señales

Thread que acepta la señal `SIGINT`

```
// El thread acepta la señal SIGINT. Cuando la recibe escribe un
// mensaje en pantalla. A la cuarta señal recibida cambia la
// máscara para que la siguiente señal termine el proceso.


#include <signal.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <misc/error_checks.h>


// declaración del cuerpo de la función manejadora
void * sigwaiter (void * arg);
```

```
// Thread principal
int main () {
  sigset_t set;
  pthread_t th;
  // Coloca la máscara para él y para el hijo
  sigemptyset(&set);
  sigaddset(&set,SIGINT);
  CHK( pthread_sigmask(SIG_BLOCK, &set, NULL) );
  // crea la tarea manejadora de señales
  CHK( pthread_create(&th,NULL,sigwaiter,NULL) );
  // permite ejecutar a la tarea manejadora
  while (1) {
    printf ("Main thread executing \n");
    sleep(1);
  }
}
```



```
void * sigwaiter (void * arg) {
  sigset_t set;
  int counter = 0;
  int signo;
  // La mascara de señales se hereda del padre (SIGINT enmascarada)

  // crea conjunto de señales a esperar
  sigemptyset(&set);
  sigaddset(&set, SIGINT);

  while (1) {
    CHKE( sigwait(&set, &signo) );
    if (signo == SIGINT) {
      // Maneja la señal de terminación
      counter ++;

      if (counter < 4) {
        printf ("SIGINT %d recibida \n", counter);
      } else {
        printf ("Señal final cazada, la próxima mata el proceso\n");

        // desbloquea la señal
        CHK( pthread_sigmask(SIG_UNBLOCK, &set, NULL) );
      }
    }
  } // while
}
```


7.8 Ejemplo de Señales con Información

```
#include <signal.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include "misc/error_checks.h"


// información a enviar con la señal
typedef struct {
  char * msj;
  int num;
} user_info_t;
```

```
// thread que acepta la señal
void * sigwaiter (void * arg)
{
  sigset_t set;
  siginfo_t sig_info;

  // La mascara de señales se hereda del padre
  // Debe tener SIGRTMIN enmascarada

  // prepara el set de señales a esperar
  sigemptyset(&set);
  sigaddset(&set,SIGRTMIN);

  while (1) {
    // espera la señal
    CHKE( sigwaitinfo(&set, &sig_info) );
    printf ("Signal %d accepted with info: (%d,%s)\n",
       sig_info.si_signo,
       ((user_info_t *)(sig_info.si_value.sival_ptr))->num,
       ((user_info_t *)(sig_info.si_value.sival_ptr))->msj);
  }
}
```



```
// thread principal que envía la señal
int main () {
  sigset_t set;
  pthread_t th;
  user_info_t user_info;
  user_info.msj = "Message";
  union sigval user_sigval;
  user_sigval.sival_ptr = &user_info;
  int i;

  // configura la señal SIGRTMIN para que tenga comportamiento
  // de tiempo real y no instala manejador (se utilizará
  // sigwaitinfo para aceptarla)
  struct sigaction sigact;
  sigact.sa_handler=SIG_DFL;
  sigact.sa_flags=SA_SIGINFO;
  CHKE( sigaction(SIGRTMIN, &sigact , NULL) );

  // pone la mascara para él y para su hijo
  sigemptyset(&set);
  sigaddset(&set,SIGRTMIN);
  CHK( pthread_sigmask(SIG_BLOCK, &set, NULL) );
```



```
  // crea el thread sigwaiter
  CHK( pthread_create(&th,NULL,sigwaiter,NULL) );

  // envía tres veces la señal
  for(i=0; i<3; i++) {
    user_info.num = i;
    printf ("Main thread send signal %d with info: (%d,%s)\n",
       SIGRTMIN, user_info.num, user_info.msj);

    // envía la señal
    sigqueue(0, SIGRTMIN, user_sigval);
    sleep(5);
  }

  return 0;
}
```