Boletín 4: Señales y el comando `bjobs`

Ampliación de Sistemas Operativos

Dpto. Ingeniería y Tecnología de Computadores (DITEC)

Universidad de Murcia

Curso 2019/2020

Índice

1. Señales

Señales

Definición

- Una señal es una notificación a un proceso ante un evento[1]
  - Excepción hardware: Dirección de memoria inválida (`SIGSEGV`)
  - Evento software: Un proceso hijo ha terminado (`SIGCHLD`)
  - Notificación de E/S: Interrupción con CTRL+C (`SIGINT`)
- Un proceso puede recibir una señal enviada por otro o por el kernel
- Una señal es un entero ≥ 1 definida en `signal.h` como `SIGXXXX`
- Desde que una señal se genera hasta que se entrega está *pendiente*
- Una señal pendiente se entrega al proceso en la siguiente transición de modo kernel a modo usuario, es decir, cuando se completa una llamada al sistema o cuando se reanuda su ejecución (*timeslice*)
- Un proceso puede bloquear la entrega de una señal de manera indefinida mediante una *máscara de señales* que la deja *bloqueada*
- Referencia: `http://man7.org/linux/man-pages/man7/signal.7.html`

[1] `cat /proc/$$/status | grep ^Signal:` ⇒ SigPnd (per-thread pending signals), ShdPnd (process-wide pending signals), SigBlk (blocked signals), SigIgn (ignored signals), and SigCgt (caught signals)

Señales

Acciones por defecto

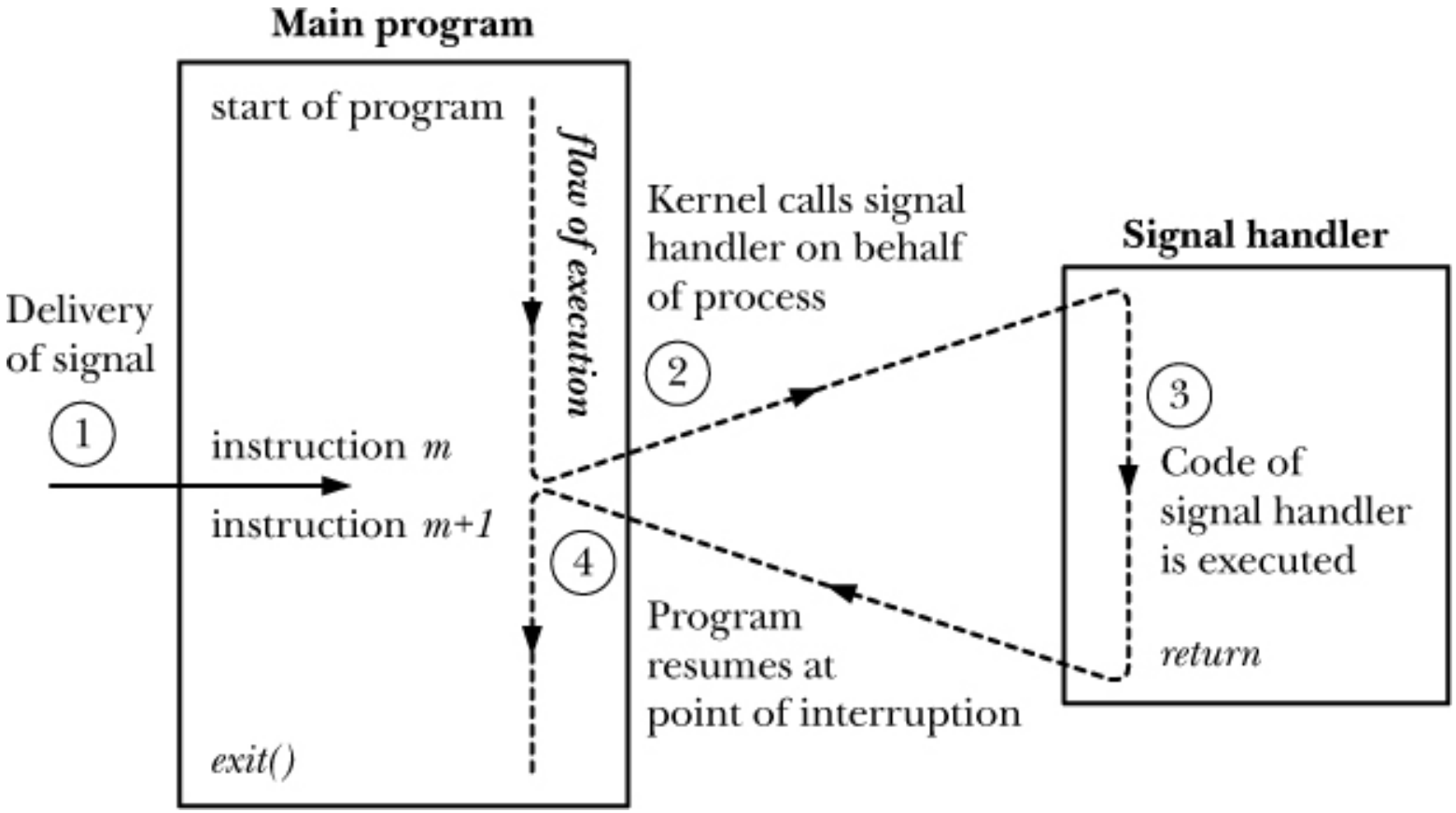
- Cuando un proceso recibe una señal, ejecuta la acción por defecto, que puede ser una de las siguientes:
  - La señal es ignorada
  - El proceso es matado (*killed*)
  - Se genera un fichero *core* y el proceso es matado
  - El proceso se detiene (*stopped*)
  - El proceso reanuda su ejecución (*resumed*)
- Un proceso puede cambiar la acción por defecto:
  - Ignorando la señal
  - Ejecutando un manejador de señales (*signal handler*)
  - Restaurando la acción por defecto

Señales

Acciones por defecto

Name	Signal number	Description	SUSv3	Default
SIGABRT	6	Abort process	•	core
SIGALRM	14	Real-time timer expired	•	term
SIGBUS	7 (SAMP=10)	Memory access error	•	core
SIGCHLD	17 (SA=20, MP=18)	Child terminated or stopped	•	ignore
SIGCONT	18 (SA=19, M=25, P=26)	Continue if stopped	•	cont
SIGEMT	undef (SAMP=7)	Hardware fault		term
SIGFPE	8	Arithmetic exception	•	core
SIGHUP	1	Hangup	•	term
SIGILL	4	Illegal instruction	•	core
SIGINT	2	Terminal interrupt	•	term
SIGIO / SIGPOLL	29 (SA=23, MP=22)	I/O possible	•	term
SIGKILL	9	Sure kill	•	term
SIGPIPE	13	Broken pipe	•	term
SIGPROF	27 (M=29, P=21)	Profiling timer expired	•	term
SIGPWR	30 (SA=29, MP=19)	Power about to fail		term
SIGQUIT	3	Terminal quit	•	core
SIGSEGV	11	Invalid memory reference	•	core
SIGSTKFLT	16 (SAM=undef, P=36)	Stack fault on coprocessor		term
SIGSTOP	19 (SA=17, M=23, P=24)	Sure stop	•	stop
SIGSYS	31 (SAMP=12)	Invalid system call	•	core
SIGTERM	15	Terminate process	•	term
SIGTRAP	5	Trace/breakpoint trap	•	core
SIGTSTP	20 (SA=18, M=24, P=25)	Terminal stop	•	stop
SIGTTIN	21 (M=26, P=27)	Terminal read from BG	•	stop
SIGTTOU	22 (M=27, P=28)	Terminal write from BG	•	stop
SIGURG	23 (SA=16, M=21, P=29)	Urgent data on socket	•	ignore
SIGUSR1	10 (SA=30, MP=16)	User-defined signal 1	•	term
SIGUSR2	12 (SA=31, MP=17)	User-defined signal 2	•	term
SIGVTALRM	26 (M=28, P=20)	Virtual timer expired	•	term
SIGWINCH	28 (M=20, P=23)	Terminal window size change		ignore
SIGXCPU	24 (M=30, P=33)	CPU time limit exceeded	•	core
SIGXFSZ	25 (M=31, P=34)	File size limit exceeded	•	core

- Un manejador de señales es una función definida por el usuario que realiza acciones apropiadas en respuesta a una señal concreta[2]



[2] La interrupción del *timer* puede reanudar la ejecución del kernel en cualquier momento. Por tanto, no es posible predecir cuando entrará en acción un manejador de señales cuando el proceso que lo instaló recibe la señal que provoca su ejecución.

2. Señales en `simplesh`

Señales en `simplesh`

- (0.25p) **EJERCICIO 1**: Tratamiento de las señales `SIGINT` y `SIGQUIT`:
  - `simplesh` debe **bloquear** la señal `SIGINT` (CTRL+C)
  - `simplesh` debe **ignorar** la señal `SIGQUIT` (CTRL+\)
- Para ello, únicamente se pueden usar llamadas al sistema POSIX o funciones de la biblioteca estándar de C (C11)
- Llamadas `POSIX` y/o `glibc` a considerar:
  - `sigemptyset()`
  - `sigaddset()`
  - `sigprocmask()`
  - `sigaction()`

3. Comandos en segundo plano

Comandos en segundo plano

- (1.75p) **EJERCICIO 2**: Identifica y soluciona las defiencias en la implementación de los comandos en segundo plano en `simplesh`

```
simplesh> sleep 1 &
/* Se muestra de nuevo el prompt inmediatamente */
simplesh> ls
simplesh> Makefile simplesh simplesh.c simplesh.o
/* No se muestra el prompt tras el listado de ficheros */
```

  - La ejecución de comandos en segundo plano no debe interferir con la ejecución de los comandos en primer plano en ningún caso
  - Cuando un comando en segundo plano se ejecuta o cuando termina, se debe enviar su `[PID]` a `stdout`
  - Cuando un comando en segundo plano termina, `simplesh` debe evitar que se convierta en un proceso *zombie*

```
simplesh> sleep 1 &
[PID]           /* Se muestra de nuevo el prompt inmediatamente */
simplesh> [PID] /* Transcurrido 1 segundo... */
/* Intro */
simplesh> ls
Makefile simplesh simplesh.c simplesh.o
```

- Para ello, únicamente se pueden usar llamadas al sistema POSIX o funciones de la biblioteca estándar de C (C11)
- Llamadas `POSIX` y/o `glibc` a considerar: `sigaction()` y `waitpid()`

Notas

Tratamiento de señales y comandos en segundo plano en `simplesh`

1. Bloquear las señales `SIGINT` y `SIGQUIT` antes de `parse_args()`
2. Modificar `run_cmd()` para que la ejecución de los comandos en segundo plano no interfiera con la de los comandos en primer plano
3. Modificar `run_cmd()` para que la ejecución de los comandos en segundo plano envíe `[PID]` a `stdout`
4. Implementar un manejador de señales para la señal `SIGCHLD`
   - El manejador tiene que evitar que los procesos creados para comandos en segundo plano se conviertan en procesos *zombies* al terminar
   - El manejador debe enviar `[PID]` a `stdout` SÓLO cuando termine la ejecución de un proceso creado para un comando en segundo plano
   - El manejador no debe interferir con los procesos en primer plano
   - El manejador sólo debe usar funciones reentrantes (*async-signal-safe*)
   - La implementación debe basarse en el esqueleto `handle_sigchld()` que aparece en *Reap zombie processes using a SIGCHLD handler*
   - En este boletín NO pueden usarse primitivas de sincronización[3]

[3] Para más detalles, véase la diapositiva 110 del Tema 1.

4. El **comando** `bjobs`

El comando bjobs

- (1.0p) **EJERCICIO 3**: Implementa el comando interno `bjobs`:

```
simplesh> sleep 5 &
[PID0]
simplesh> sleep 10 &
[PID1]
simplesh> bjobs     /* Antes de que transcurran 5 segundos... */
[PID0]
[PID1]
simplesh> [PID0]    /* Transcurridos 5 segundos */
/* Intro */
simplesh> bjobs     /* Transcurridos entre 5 y 10 segundos... */
[PID1]
simplesh> [PID1]    /* Transcurridos 10 segundos */
```

- Para ello, únicamente se pueden usar llamadas al sistema POSIX o funciones de la biblioteca estándar de C (C11)
- Llamadas `POSIX` y/o `glibc` a considerar:
  - `sigaction()`
  - `waitpid()`
  - `kill()`
- Para implementar `bjobs`, escribe la función `run_bjobs()` e inserta llamadas a la misma en `simplesh.c` donde sea necesario

El comando bjobs

- Si se proporciona la opción -h, bjobs debe enviar a stdout:

```
simplesh> bjobs -h
Uso: bjobs [-k] [-h]
     Opciones:
     -k Mata todos los procesos en segundo plano.
     -h Ayuda
```

- La opción -k mata todos los procesos en segundo plano:

```
simplesh> sleep 3 &
[PID0]
simplesh> sleep 3 &
[PID1]
simplesh> bjobs     /* Antes de que transcurran 3 segundos... */
[PID0]
[PID1]
simplesh> bjobs -k  /* Antes de que transcurran 3 segundos... */
simplesh> [PID0]
[PID1]

simplesh> bjobs     /* Antes de que transcurran 3 segundos... */
```

Notas

Implementación del comando `bjobs`

5. Procesar las opciones de `bjobs` con `getopt()` (`man 3 getopt`)
   - **Nota**: Antes de volver a usar `getopt()`, se debe inicializar `optind` a `1`
6. Enviar el `[PID]` de cada proceso en segundo plano *activo* a `stdout`
7. Con `-k`: Matar los procesos en segundo plano *activos*
8. Para resolver este ejercicio se necesita seguir la pista a los procesos en segundo plano *activos* en cada momento
   - **Nota**: Por simplicidad, asumiremos que el número máximo de comandos en segundo plano *activos* en cada momento es de 8