

Boletín 5: Señales y el comando bjobs

Ampliación de Sistemas Operativos

Dpto. Ingeniería y Tecnología de Computadores (DITEC)

Universidad de Murcia

Curso 2017/2018

1 Señales

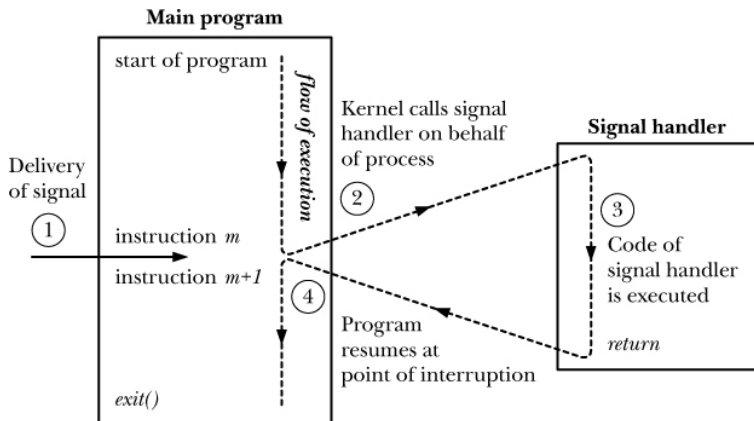
2 El comando bjobs

1. Señales

- Una señal es una notificación a un proceso cuando sucede un evento
 - 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 inmediatamente al proceso si se está ejecutando o tan pronto como reanude su ejecución en caso contrario
- No obstante, un proceso puede bloquear la entrega de una señal mediante una *máscara de señales* que la deja en estado *bloqueada*
- Referencia:
<http://man7.org/linux/man-pages/man7/signal.7.html>

- Cuando un proceso recibe una señal, ejecuta la acción por defecto:
 - 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

- Un manejador de señales es una función definida por el usuario que lleva a cabo acciones apropiadas en respuesta a una señal concreta



● Consulta y modificación de la acción por defecto:

```
1  #include <signal.h> /* POSIX */
2  int sigaction(int sig, const struct sigaction *act, struct sigaction *oldact);
3
4  struct sigaction {
5      void (*sa_handler)(int); /* Address of handler */
6      sigset_t sa_mask; /* Signals blocked during handler invocation */
7      int sa_flags; /* Flags controlling handler invocation */
8      void (*sa_restorer)(void); /* Not for application use */
9  };
```

- `sig` es la señal para la cual se realiza la consulta o modificación
- `sa_handler` se refiere al manejador de la señal o a las constantes `SIG_IGN` (ignorar la señal) o `SIG_DFL` (restaurar la acción por defecto)
- `sa_mask` define qué señales se bloquearán durante la ejecución del manejador de la señal además de las ya bloqueadas en la máscara de señales del proceso con `sigprocmask()` (incluida la señal `sig`)
- `sa_flags` es una máscara de bits con opciones que afectan a la ejecución del manejador de la señal, por ejemplo, `SA_RESETHAND` restablece la acción por defecto cuando se ejecuta el manejador

- Consulta y modificación de la máscara de señales:

```
1 #include <signal.h> /* POSIX */  
   int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
```

- Si set es NULL, sigprocmask() devuelve la máscara de señales del proceso en oldset
 - En caso contrario, sigprocmask() modifica la máscara de señales del proceso en función de how:
 - SIG_BLOCK añade set a la máscara de señales del proceso
 - SIG_UNBLOCK elimina set de la máscara de señales del proceso
 - SIG_SETMASK establece la máscara de señales del proceso a set
- sigset_t es un conjunto de señales gestionado con sigemptyset(), sigfillset(), sigaddset(), sigdelset() y sigismember()
- SIGKILL y SIGSTOP no se pueden capturar, bloquear o ignorar

Servicios POSIX para señales

● Código:

```
1  int main(int argc, char *argv[])
2  {
3      /* Block signal SIGSEGV */
4      sigset_t blocked_signals;
5      sigemptyset(&blocked_signals);
6      sigaddset(&blocked_signals, SIGSEGV);
7      if (sigprocmask(SIG_BLOCK, &blocked_signals, NULL) == -1) {
8          perror("sigprocmask");
9          exit(EXIT_FAILURE);
10     }
11
12     /* Establish signal handler for SIGINT */
13     struct sigaction sa;
14     memset(&sa, 0, sizeof(sa)); /* SIGSEGV!!! */
15     sa.sa_handler = signal_handler;
16     sigemptyset(&sa.sa_mask);
17     if (sigaction(SIGINT, &sa, NULL) == -1) {
18         perror("sigaction 1");
19         exit(EXIT_FAILURE);
20     }
21
22     while(1)      /* Loop forever, waiting for signals */
23     {
24         pause(); /* Block until a signal is caught */
25         printf("Caught SIGINT %d times\n", shc);
26     }
27
28     /* The program will never get here! */
29     return EXIT_SUCCESS;
30 }
```

• Código (cont.):

```
1  static int shc = 0; /* signal handler counter */  
3  static void signal_handler(int sig)  
4  {  
5      if (sig == SIGINT) shc++; /* SIGINT (^C): increase the counter */  
6  }
```



- (0.5p) **EJERCICIO 1:** Tratamiento de las señales SIGINT y SIGQUIT:
 - `simplesh` debe bloquear la señal SIGINT (CTRL+C)
 - `simplesh` debe bloquear 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()`

- (1.0p) **EJERCICIO 2:** Identifica y soluciona las deficiencias en la implementación de los comandos en segundo plano en `simplesh`

```
simplesh> sleep 1 &  
2 /* Se muestra de nuevo el prompt inmediatamente */  
simplesh> ls  
4 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*

```
1 simplesh> sleep 1 &  
[PID] /* Se muestra de nuevo el prompt inmediatamente */  
3 simplesh> [PID] /* Transcurrido 1 segundo... */  
/* Intro */  
5 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: `waitpid()` y `sigaction()`

2. El comando `bjobs`

El comando bjobs

- (0.5p) **EJERCICIO 3:** Implementa el comando interno bjobs:

```
1  simplesh> sleep 5 &
2  [PID0]
3  simplesh> sleep 10 &
4  [PID1]
5  simplesh> bjobs      /* Antes de que transcurran 5 segundos... */
6  [PID0]
7  [PID1]
8  simplesh> [PID0]     /* Transcurridos 5 segundos */
9                        /* Intro */
10 simplesh> bjobs      /* Transcurridos entre 5 y 10 segundos... */
11 [PID0]
12 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:
 - waitpid()
 - sigaction()
 - 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 el parámetro `-h`, bjobs debe enviar a stdout:

```
1  simplesh> bjobs -h
2  Uso: bjobs [-k] [-h]
   Opciones:
4     -k Mata todos los procesos en segundo plano
     -h help
```

- (0.5p) **Opcional:** El parámetro `-k` mata todos los procesos en segundo plano:

```
1  simplesh> sleep 10 &
   [PID0]
3  simplesh> sleep 10 &
   [PID1]
5  simplesh> bjobs      /* Antes de que transcurran 10 segundos... */
   [PID0]
   [PID1]
7  simplesh> bjobs -k   /* Antes de que transcurran 10 segundos... */
9  simplesh> [PID0]
   [PID1]
11 simplesh> bjobs      /* Antes de que transcurran 10 segundos... */
```

El comando bjobs

- Implementación:

- ❶ Bloquear las señales SIGINT y SIGQUIT antes de `parse_args()`
- ❷ Modificar `run_cmd()` para que la ejecución de los comandos en segundo plano no interfiera con la de los comandos en primer plano
- ❸ Modificar `run_cmd()` para que la ejecución de los comandos en segundo plano envíe `[PID]` a `stdout`
- ❹ Implementar un manejador de señales para la señal SIGCHLD
 - El manejador debe enviar `[PID]` a `stdout` SÓLO cuando termine la ejecución de un proceso creado para un comando en segundo plano
 - El manejador tiene que evitar que los procesos creados para comandos en segundo plano se conviertan en procesos *zombies* al terminar
 - **Referencia:** *Reap zombie processes using a SIGCHLD handler*
- ❺ Procesar los parámetros de `bjobs` con `getopt()` (`man 3 getopt`)
 - **Nota:** Antes de volver a usar `getopt()`, se debe inicializar `optind` a 1
- ❻ Implementar la función `run_bjobs()` e insertar llamadas a la misma igual que para las funciones `run_*`() de los boletines anteriores

Importante: Para resolver los ejercicios 2 y 3 se necesita seguir la pista a los procesos en segundo plano *activos* en cada momento