

# Comunicación entre Procesos: SEÑALES

LABORATORIO DE SISTEMAS DE OPERACIÓN I  
(ci-3825)

Prof. Yudith Cardinale

Enero-marzo 2012

# Interrupciones

- Las interrupciones pueden ser:
  - de hardware: señales electrónicas producidas por dispositivos de entrada/salida
  - de software:
    - llamadas al sistema
    - Errores: enviados al kernel hacia el proceso que produjo el error
    - Señales programadas: enviadas entre procesos usuarios
- Las señales programadas se pueden utilizar:
  - como mecanismo de comunicación entre procesos.
  - para programar comportamientos de los procesos ante ciertas señales.

# Interrupciones

- `kill -l` : lista todas las señales del sistema.

1) SIGHUP	2) SIGINT	3) SIGQUIT	4) SIGILL	5) SIGTRAP
6) SIGABRT	7) SIGBUS	8) SIGFPE	9) SIGKILL	10) SIGUSR1
11) SIGSEGV	12) SIGUSR2	13) SIGPIPE	14) SIGALRM	15) SIGTERM
16) SIGSTKFLT	17) SIGCHLD	18) SIGCONT	19) SIGSTOP	20) SIGTSTP
21) SIGTTIN	22) SIGTTOU	23) SIGURG	24) SIGXCPU	25) SIGXFSZ
26) SIGVTALRM	27) SIGPROF	28) SIGWINCH	29) SIGIO	30) SIGPWR
31) SIGSYS	34) SIGRTMIN	35) SIGRTMIN+1	36) SIGRTMIN+2	

- `sstty -a`: lista algunas señales que pueden enviarse por teclado

`intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = M-^?; eol2 = M-^?;`

`swtch = M-^?; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W;`

- Todas las señales tienen asociado: un nro. entero, un nombre y su propio manejador.
- El manejador es la rutina del sistema de operación que atiende la señal correspondiente.

# Interrupciones

- Algunos de estos manejadores se pueden sobre-escribir en un programa en C, con llamadas al sistema para manejo de señales.
- Las acciones por defecto de los manejadores de señales son:
  - Terminar el proceso y generar un *core (dump)*
  - Terminar el proceso sin generar un *core (quit)*
  - Ignorar y descartar la señal
  - Suspender el proceso
  - Reanudar un proceso
- Se pueden enviar señales a los procesos *foreground* "desde afuera" (teclado):
  - CTRL-C (*stop*)
  - CTRL-Z (*suspend*)
  - CTRL-S (*suspend*) CTRL-Q (*start*)

# La señal Alarm()

- **unsigned int alarm(unsigned int count)**
  - Permite que un proceso se envíe a si mismo una señal de alarma (SIGALRM), luego de *count* segundos
  - Si `count = 0`, se cancelan todas las alarmas pendientes
  - La llamada de un **alarm** anula las llamadas anteriores que aún no han enviado la señal
  - Retorna cuántos segundos restan para que la señal sea enviada.
  - El manejador por defecto imprime el mensaje "Alarm Clock" y termina el proceso.

- Ejemplo

```
#include <stdio.h>
#include <signal.h>
main() {
    alarm(3);
    alarm(6);
    printf("Ciclo eterno...\n" );
    while (1);
    printf("Esta linea nunca se ejecutara\n");
}
```

```
Resultado:
$> ./alarma
Ciclo eterno...
Alarm Clock
$>
```

# Instalación de Manejadores

- Instalación de un nuevo manejador

`void * signal(int sigCode, void (*func)(int))`

- Permite instalar un nuevo manejador de señales para la señal sigCode
- sigCode: especifica la señal que será reprogramada
- func: Puede ser:
  - SIG\_IGN: ignora la señal
  - SIG\_DFL: usará el manejador por defecto
  - Dirección de la función que se ejecutará cuando llegue la señal sigCode, reemplazando al manejador por defecto
- Retorna -1 si ocurre un error o la dirección del manejador anterior
- SIGKILL y SIGSTP, no pueden ser reprogramadas
- Los hijos heredan las señales "seteadas" del padre, pero si hacen exec se pierden los manejadores instalados, pero permanecen los mandatos de ignorar (SIG\_IGN)

# Instalación de Manejadores

- Suspende un proceso hasta que reciba una señal

`int pause(void)`

- Suspende al proceso que realiza la llamada
- El proceso se reanuda cuando recibe una señal (cualquiera)
- Generalmente es usada para esperar por una señal SIGALRM
- No retorna nada

# Instalación de Manejadores

- **Ejemplo 1:**

```
#include <stdio.h>
```

```
#include <signal.h>
```

```
/****** Rutina principal *****/
```

```
int main() {
```

```
    void (*oldHandler)(); //para guardar la direccion del manejador por defecto
```

```
    printf("Puedo ser asesinado con Ctrl-C...\n");
```

```
    sleep(3);
```

```
    oldHandler = signal(SIGINT, SIG_IGN); // se ignora Ctrl-C
```

```
    printf("Estoy protegido contra el Ctrl-C...\n");
```

```
    sleep(6);
```

```
    signal(SIGINT, oldHandler); // se restaura el manejador por defecto de Ctrl-C
```

```
    printf("Puedo ser asesinado con Ctrl-C...de nuevo\n");
```

```
    while (1);
```

```
}
```

# Instalación de Manejadores

- Ejemplo 2:

```
#include <stdio.h>
#include <signal.h>
int alarmFlag = 0;
void alarmHandler();

int main() {
    signal(SIGALRM, alarmHandler); // se instala el nuevo manejador
    alarm(3);
    printf("Ciclando ...\n");
    while(!alarmFlag)
        pause();                //Espera hasta que llegue una senal
    printf("Finaliza el ciclo cuando se recibe la senal SIGALRM ...\n");
}
/***** Manejador de la senal SIGALRM *****/
void alarmHandler() {
    printf("Llego la senal SIGALRM...\n");
    printf("Ahora por defecto no se mata al proceso...\n");
    alarmFlag=1;
}
```

# Comunicación por señales entre procesos

- Transmitir señales entre procesos:

`int kill(pid_t pid, int sigCode)`

- Envía la señal *sigCode* al proceso *pid*
- Condiciones para el kill:
  - Los procesos que envían y reciben la señal tienen el mismo propietario
  - El proceso que envía (kill) es de un super-usuario
- Características:
  - si `pid=0`, la señal es enviada a todos los procesos en el grupo del proceso enviador.
  - si `pid= -1` y el enviador es super-usuario, la señal es enviada a todos los procesos incluyendo el propio enviador
  - si `pid= -1` y el enviador no es super-usuario, la señal es enviada a todos los procesos del mismo propietario del enviador sin incluir al proceso enviador
  - si `pid` es negativo  $\neq -1$ , la señal es enviada a todos los procesos en el grupo del proceso (equivalente a si `pid=0`)

# Comunicación por señales entre procesos

- Cuando un proceso hijo termina, envía la señal SIGCHLD al proceso padre
- El proceso Padre puede instalar un manejador SIGCHLD para no bloquearse en el wait de sus hijos.
- Normalmente las señales SIGUSR1 y SIGUSR2 son usadas por los programadores para comunicación entre procesos padre e hijos.

# Comunicación por señales entre procesos

```
#include <stdio.h>
#include <signal.h>
int delay;
void childHandler();
/***** Rutina principal *****/
int main(int argc, char * argv[]) {
    int pid;
    if (argc < 3) {
        printf("Error en argumentos.\n");
        printf("Usage: %s delay command \n", argv[0]);
        exit(0);
    }
    signal(SIGCHLD, childHandler); // se instala el nuevo manejador
    pid = fork();
    if (pid == 0) {
        execvp(argv[2],&argv[2]);
        perror("Error en exec");
    }
    else {
        delay=atoi(argv[1]);
        sleep(delay);
        printf("El hijo %d se excedio en tiempo...debe morir.\n",pid);
        kill(pid,SIGINT); //mata al hijo
    }
}
```

# Comunicación por señales entre procesos

```
/** Manejador de la señal SIGCHLD */
```

```
void childHandler() {  
    int childPid, childStatus;  
  
    childPid = wait(&childStatus);  
  
    printf("El hijo %d termino en menos de %d segundos...\n", childPid, delay);  
    exit(0);  
}
```

# Comunicación por señales entre procesos

- Ejercicios
  - Compile, ejecute y analice los programas: alarma1.c, alarma2.c, alarma3.c, manejador1.c, manejador2.c, comunic1.c y comunic2.c
  - Use los archivos (<http://www ldc usb ve/~yudith/docencia/ci-3825-taller/TomarTiempo.c>) TomarTiempo.c y (<http://www ldc usb ve/~yudith/docencia/ci-3825-taller/TomarTiempo.h>) TomarTiempo.h para crear un programa demonio que cada vez que reciba una señal SIGUSR1 imprima los segundos que han transcurrido desde la última interrupción (recuerde la instrucción pause()). Para probar la funcionalidad de su demonio, enviaremos la señal SIGUSR1 desde la consola con kill -USR1 daemon\_pid.
  - Modifique el primer programa, para que el proceso padre ejecute el wait, imprima los segundos desde que comenzó hasta que su hijo finalizó y el valor de retorno del wait, cuando reciba la señal SIGCHLD. Haga que su hijo tenga algún trabajo que hacer.