Pablo González Nalda
Universidad del País Vasco Euskal Herriko Unibertsitatea
GASTEIZKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE VITORIA-GASTEIZ
Depto. de *Lenguajes y Sistemas Informáticos* lsi.vc.ehu.eus/pablogn

Señales y tuberías

En esta actividad vamos a ver las características principales de las señales y de las tuberías.

8.1. Ejemplos de señales

En esta parte trataremos las señales más sencillas para observar su funcionamiento y para modificar el comportamiento por defecto.

8.1.1. Caso básico: de interrupción software en la FPU a señal SIGFPE

Señal de aviso de error en operaciones FPU (Unidad de coma flotante):

```c
/**
 * Ejemplo de recepcion de sennales
 */
#include <stdio.h>
#include <signal.h>
#include <time.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

void funerror(){
  printf("\nError en coma flotante\n");
    exit(5); // sin esto, intenta repetir la division infinitamente.
}
int main(){
  signal(SIGFPE,funerror);
    int b = 1;
    int c = 0;
    int d = b/c;
  printf("\nd es %d\n\n",d);
}
```

Fichero 8.1: `fpe.c`

Prueba el comportamiento del programa cuando no se cambia en la tabla de señales con la llamada al sistema `signal` la respuesta por defecto a la señal `SIGFPE` que ocurre al dividir por cero.

8.1.2. Señales de control de un proceso

Este programa, como dice su encabezado, crea dos funciones que imprimen sendos mensajes y *captura* (modificando la tabla de señales) la señal SIGINT enviada al proceso al hacer CTRL-C y comprueba también si se puede evitar o no la terminación del programa con SIGTERM Y SIGKILL.

```c
/**
$ gcc -o c ctrlc.c
$ ./c

^C                  si le damos a CTRL-C todas las veces que queramos
No paro

No paro             si le hacemos desde otra consola $ kill -SIGINT  23350
(o el PID de proceso que se vea en top)

No me mates         si le hacemos desde otra consola $ kill -SIGTERM 23350
Terminado (killed)  si le hacemos desde otra consola $ kill -SIGKILL 23350
 */
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

void f1(){
        printf("\nNo paro\n"); // ejecucio'n asi'ncrona
}
void f2(){
        printf("\nNo me mates\n");
}
int main(){
        signal(SIGINT,f1);
        signal(SIGTERM,f2);
        printf("\nSoy el proceso PID %d\n\n",getpid());
        while(1);               // ciclo infinito !!!!!
        printf("\nFIN\n\n"); // no se ejecuta nunca
}
```

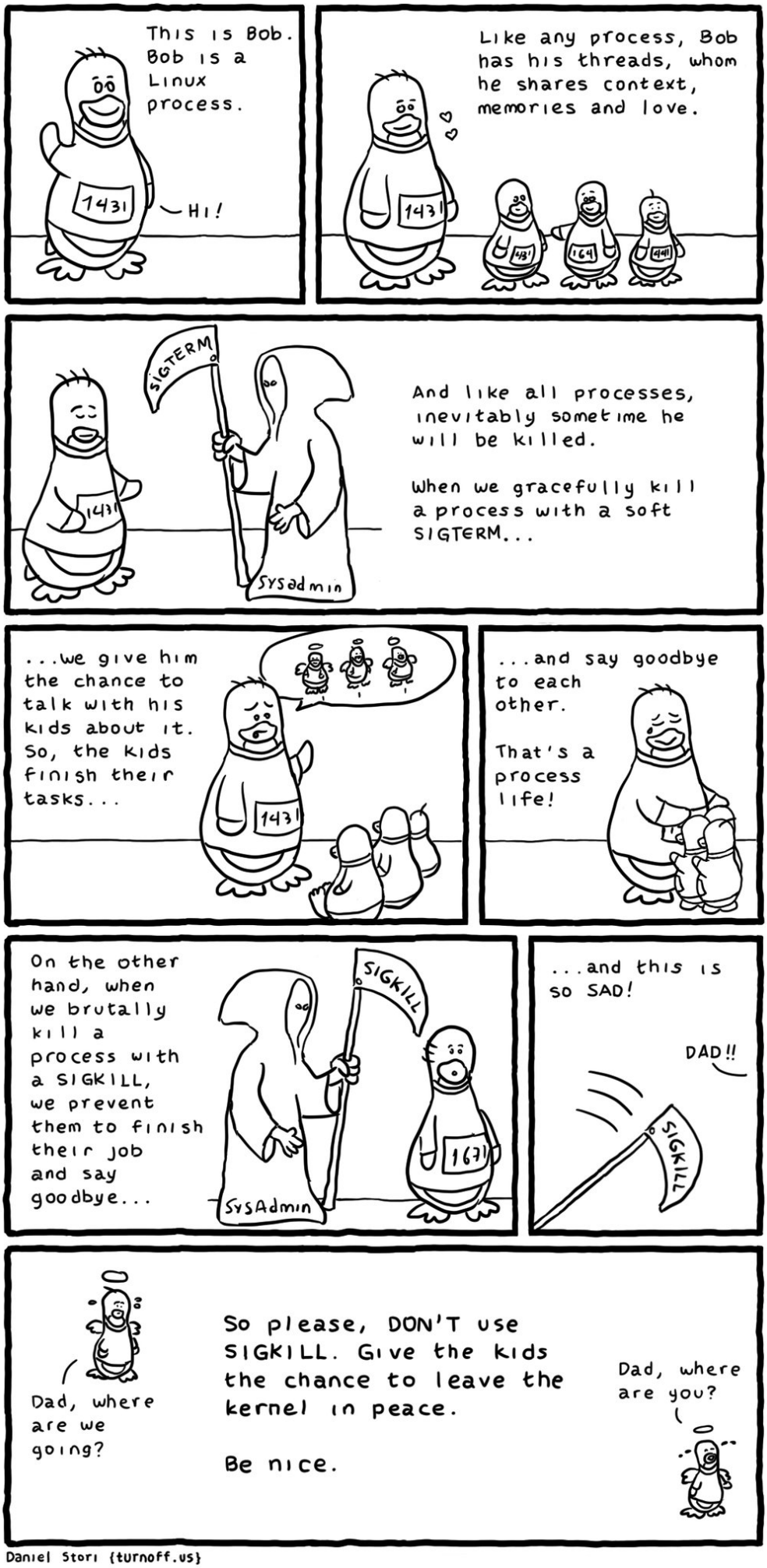
Fichero 8.2: `ctrlc.c`

8.1.3. Comportamiento básico de un programa con la señal `SIGALRM`

```c
#include <stdio.h>
#include <signal.h>
#include <time.h>
#include <stdlib.h>
#include <unistd.h>

void f(){
  printf("\nDespertamos\n");
}
int main(){
  signal(SIGALRM,f); // cambiamos la entrada SIGALRM de la tabla de
     sennales
  alarm(2); // dentro de 2s el SO nos enviara' SIGALRM
  pause();  // nos bloqueamos
  printf("\nFIN\n");
}
```

Fichero 8.3: `alarma.c`



Daniel Stori {turnoff.us}

https://turnoff.us/

8.1.4. Señales entre padre e hijo

Programa que hace `fork` para comprobar que un proceso padre puede enviar señales a un hijo y al revés como forma de señalización de eventos.

Lo único necesario es que pertenezcan al mismo usuario. La tabla de canales se hereda, para ello comentar la línea `signal(SIGALRM,fh);` que cambia la del hijo, porque sin ella escribirá `Padre` dos veces.

Tanto `sleep` como `pause` bloquean el proceso hasta que se reciba una señal (o se pase el tiempo determinado en `sleep`)

Crea ungráfico que describa en el tiempo el orden de las llamadas al sistema.

¿Por qué no espera 25 segundos en el `sleep` del hijo en la línea 21?

```
1  #include <stdio.h>
2  #include <signal.h>
3  #include <time.h>
4  #include <stdlib.h>
5  #include <unistd.h>
6  #include <sys/types.h>
7  #include <sys/wait.h>
8
9  void fp(){
10   printf("\nPadre\n");
11 }
12 void fh(){
13   printf("\nHijo\n");
14 }
15 int main(){
16   int p,r,idht;
17   signal(SIGALRM,fp);
18   p=fork();
19   if (p==0) {
20     signal(SIGALRM,fh);
21     sleep(25);
22     sleep(5);
23     kill(getppid(),SIGALRM);
24     exit(1);
25   }
26   sleep(1);
27   kill(p,SIGALRM);
28   idht=wait(&r);
29   printf("\nr %d PID del hijo terminado %d\n\n",r, idht);}
```

Fichero 8.4: `senales.c`

8.1.5. Señal a un hijo no capturada

El hijo muere al recibir `SIGALRM` sin capturarla. Como consecuencia, el padre recibe a través de `wait` el código numérico de la señal (14) que lo ha matado.

La variable `r` es pasada por referencia al `wait` y el padre recibe el número de la señal, o si el hijo termina en un `exit` el valor que el hijo ha pasado por `exit` multiplicado por 256.

Si es por una señal, recibe directamente el número de la señal (ver `man 7 signal` para la lista de las señales), y si es por `exit` lo recibe en el 2º byte menos significativo el valor devuelto por el `exit`. Eso es equivalente a multiplicar el número por 256, por lo que se divide entre 256 o se usa la macro `WEXITSTATUS(r)`).

```c
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(){
  int p,r, idht;
  p=fork();
  if (p==0) {
    sleep(25);
    exit(1); // fin del hijo
  }
  sleep(1); // padre
  kill(p,SIGALRM);
  idht=wait(&r);
  printf("\nr %d PID del hijo terminado %d\n\n",r, idht);
  if (WIFEXITED(r)) /* process exited normally */
    printf("child process exited with value %d\n", WEXITSTATUS(r));
  else if (WIFSIGNALED(r))  /* child exited on a signal */
    printf("child process exited due to signal %d\n", WTERMSIG(r));
  else if (WIFSTOPPED(r)) /* child was stopped */
    printf("child process was stopped by signal %d\n", WIFSTOPPED(r));
}
```

Fichero 8.5: `senalesB.c`

8.2. Ejemplos de tuberías

Las tuberías son ficheros especiales que actúan como colas FIFO de caracteres.

8.2.1. Tuberías con nombre (`mkfifo`)

Cuando se crean con la llamada al sistema `mkfifo` tienen nombre y se usan como cualquier fichero con `open`. Sólo necesitamos abrirlas usando su nombre para lectura o escritura. En este caso el programa genera 10 hijos con `fork`, que escriben su número `i` en la tubería y el padre los lee en grupos de la tubería.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

#define N 10
int main(){
    int x,i,n,r;
    char s[10];
    mkfifo("tub", 0666);
    for (i=0;i<N;i++) {
        if (fork()==0) {
            n=open("tub", O_WRONLY, 0666);
            sprintf (s,"%d", i); /* imprime el entero en la cadena s */
            sleep(1);
            write(n, s, 1);
            sleep(10); // con ctrl-z ps /proc/PID/fd/ se ve el uso de la
               tub W
            close(n);
            exit(1);
        }
    }
    n=open("tub", O_RDONLY, 0666);
    while ((x=read(n,s,10))>0){
        write(1,s,x);
        write(1,"-",1);
        fflush(stdout);
    }
    while(wait(&r)!=-1);
    close(n);
    printf("\n");
    unlink("tub");
}
```

Fichero 8.6: `mkfifo.c`

8.2.2. Tuberías sin nombre (`pipe`)

El siguiente programa es el más sencillo, crea una tubería sin nombre a la que sólo puede acceder por los canales cuyos números están almacenados en el vector `df` de forma que escribe por `df[1]` y lee por `df[0]`. Escribe una frase en la tubería y la vuelve a leer.

Para ver su efcto es interesante consultar tanto la tabla de canales como la TFA, con instrucciones como las que aparecen en los comentarios a la línea 12.

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <stdlib.h>
4  void main() {
5    int df[2], lg;
6    char buf[50];
7    pipe(df);
8    printf("\n Canal de lectura %d \nCanal de escritura %d\n",df[0],df[1]);
9    write(df[1],"Hola...",7);
10   lg=read(df[0],buf,50);
11   printf("\nBuf: %s \n",buf);
12   sleep(50);  // te permite hacer CTRL-Z
13                  // ps aux | grep basicpipe
14                  // ls -l /proc/514/fd
15                  // lsof|grep basicpipe
16   exit(0);
17 }
```

Fichero 8.7: `basicpipe.c`

En el siguiente caso el padre espera a que se ejecute el hijo y lee bloques de 40 caracteres de forma no bloqueante por la línea del `fcntl`, para que cuando no haya nada que leer siga en el `while`.

El hijo hereda la tabla de canales del padre por lo que puede acceder a la tubería sin nombre. Cierra la salida estándar para duplicar la escritura en la tubería en el canal 1. Por ello, el programa `uname` *recicla* la tabla de canales modificada, por lo que cuando escribe por la salida estándar se envía a la tubería y le llega al padre.

```c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>

void f1(){
  write(1,">",1);
}
int main(){
  int df[2],x,h,r;
  char s[200];
  pipe(df);
  fcntl(df[0], F_SETFL,fcntl(df[0], F_GETFL) |O_NONBLOCK);

  if((h=fork())!=0) {
    signal(SIGALRM, f1);
    alarm(1);
    pause();
    x = read (df[0], s, 40);
    while ( x > 0 ) {
      write(1, s, x);
      x = read (df[0], s, 40);
      write(1, "\n---\n", 5);
      printf("\nx %d\n",x);
    }
    printf("\nHaz: lsof | grep %d\n",getpid());
    h=wait(&r);
    sleep(5); // para ver el pipe con lsof|grep PID
  }
  else {
    close(1);
    dup(df[1]);
    execlp("uname","uname","-a",NULL);
    perror("uname");
  }
}
```

Fichero 8.8: `pipe.c`

8.3. Ejercicios con tuberías

En esta actividad vamos a escribir dos programas que consigan el mismo resultado, que es hacer las mismas operaciones que hace el intérprete de comandos o `bash` cuando se le escribe `cat fichero|wc -l`.

Cada versión del programa que escribimos debe crear una tubería, tener dos hijos y conectar la salida estándar del primero con la tubería y la entrada estándar del segundo también con la misma tubería. Una vez que cada hijo haya hecho una de las conexiones, el primer hijo debe ejecutar la parte `cat fichero` para escribir el fichero en la tubería y el segundo la parte `wc -l` para contar las líneas de lo que lee de la tubería. El padre debe esperar la finalización de todos los hijos y borrar la tubería si es con nombre.

Para cada ejercicio haced un gráfico que refleje la estructura explicada, y colocad la llamada al sistema que corresponda en cada línea.

Primero se debe hacer el ejercicio con tuberías **con nombre** (es más sencillo) y después **sin nombre**.