

TP0-SO2  
Eric Alexander Szuka

a)

Al ingresar el comando **yes**, en la consola se empezó a imprimir **y** tras **y**. Para detener esto se puede pulsar **Ctrl+C**. Esta combinación lo que hace es enviarle la señal **SIGINT**.

Para hacerlo desde otro proceso bash por ejemplo, existen varios metodos.

Podemos hacerlo mediante el programa **top** o **htop**, y enviarle una señal específica usando **F9**

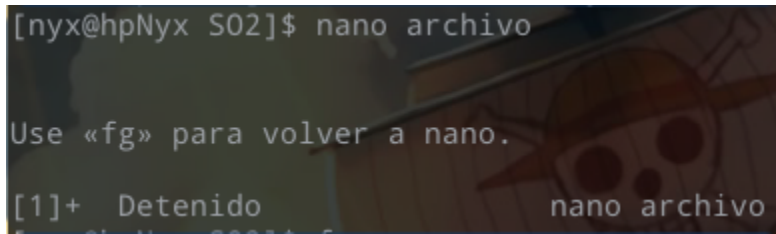
```
0[|||||] 35.8% 4[|||||] 37.6%
1[|||||] 21.3% 5[|||||] 44.4%
2[|||||] 43.8% 6[|||||] 43.3%
3[|||||] 38.4% 7[|||||] 30.7%
Mem[|||||] 8.40G/15.5G Tasks: 83, 473 thr, 195 kthr; 2 running
Swp[|||||] 0K/16.0G Load average: 3.37 2.79 2.42
Uptime: 07:40:32

Main I/O
Send signal: mand (merged)
0 Cancel dbus-broker-launch --scope user
1 SIGHUP dbus-broker --log 4 --controller 10 --machine-id 404f295544294bd2b563d8882518b40c --
2 SIGINT systemd-udevd
3 SIGQUIT wpa_supplicant -u -s -O /run/wpa_supplicant
4 SIGILL accounts-daemon
5 SIGTRAP pool-spawner|accounts-daemon
6 SIGABRT gmain|accounts-daemon
7 SIGIOT gdbus|accounts-daemon
8 SIGBUS polkitd --no-debug --log-level=notice
9 SIGFPE gmain|polkitd --no-debug --log-level=notice
10 SIGKILL pool-spawner|polkitd --no-debug --log-level=notice
11 SIGUSR1 gdbus|polkitd --no-debug --log-level=notice
12 SIGSEGV picom -f -i 0.9 -b -e 0.8 -I 0.05 -o 0.05
13 SIGUSR2 systemd-logind
14 SIGPIPE systemd-timesyncd
15 SIGALRM sd-resolve|systemd-timesyncd
16 SIGTERM lightdm
17 SIGSTKFLT Xorg :0 -seat seat0 -auth /run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch
18 SIGCHLD Xorg :0 -seat seat0 -auth /run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch
19 SIGCONT Xorg :0 -seat seat0 -auth /run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch
20 SIGSTOP Xorg :0 -seat seat0 -auth /run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch
21 SIGTSTP InputThread|Xorg :0 -seat seat0 -auth /run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch
22 SIGTTIN lightdm --session-child 13 20
23 SIGTTOU i3
24 SIGURG python3.13|/usr/bin/python /usr/bin/terminator
25 SIGXCPU gmain|python /usr/bin/terminator
26 SIGXFSZ pool-spawner|python /usr/bin/terminator
27 SIGVTALRM dconf worker|python /usr/bin/terminator
28 SIGPROF gdbus|python /usr/bin/terminator
29 SIGWINCH [pango] fontcon|python /usr/bin/terminator
30 SIGIO threaded-ml|python /usr/bin/terminator
31 SIGPOLL bash
32 SIGPWR yes
33 SIGSYS bash
34 SIGRTMIN bash
```

Otra forma seria obteniendo el PID del proceso y ejecutando, en otro bash, **kill -s SIGINT pid**

b)

Si ejecutamos un proceso y lo queremos suspender, la combinación de teclas es **Ctrl+Z** (**Ctrl+T** + **Ctrl+Z** si ejecutamos **nano bash**)

A screenshot of a terminal window with a dark background and a skull icon. The prompt is [nyx@hpNyx S02]\$ and the command nano archivo has been entered. Below the prompt, a message reads 'Use «fg» para volver a nano.' At the bottom, the text '[1]+ Detenido nano archivo' indicates the process is suspended.

```
[nyx@hpNyx S02]$ nano archivo  
  
Use «fg» para volver a nano.  
  
[1]+ Detenido nano archivo
```

Al igual que para detener el comando **yes**, aca podemos usar el programa **top** o **htop**; o utilizar **kill -s SIGSTOP**

Ahora, para reanudarlo podemos utilizar el comando **fg**.

c)

Para comprobar su funcionamiento, tenemos que ejecutar el programa y enviar las diferentes señales. Podemos comprobar que estas señales llegaron por medio de nuestro programa (usando el print cuando llega esa señal) o usando **strace -p \$(pidof signal\_handler)** (ambas formas tienen su screenshot correspondiente)

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

int stop_flag = 0;

void intHandler(int sig){
    printf("Se lanzo la señal INT\n");
}
void killHandler(int sig){
    printf("Se lanzo la señal KILL\n");
}
void stopHandler(int sig){
    printf("Se lanzo la señal STOP\n");
}
void usr1Handler(int sig){
    printf("Se lanzo la señal USR1\n");
    stop_flag = 1;
}

int main(int argc, char** argv)
{
    signal(SIGINT,intHandler);
    signal(SIGKILL,killHandler);
    signal(SIGSTOP,stopHandler);
    signal(SIGUSR1,usr1Handler);

    while(stop_flag==0){
        ;
    }

    return 0;
}
```

```
[nyx@hpNyx S02]$ nvim signal_handler.c
[nyx@hpNyx S02]$ gcc -o signal_handler signal_handler.c
[nyx@hpNyx S02]$ ./signal_handler
Se lanzo la señal INT

[2]+  Detenido                  ./signal_handler
[nyx@hpNyx S02]$ fg
./signal_handler
Terminado (killed)
[nyx@hpNyx S02]$ ./signal_handler
Se lanzo la señal USR1
[nyx@hpNyx S02]$
```

```
[nyx@hpNyx ~]$ kill -s SIGINT $(pidof signal_handler)
[nyx@hpNyx ~]$ kill -s SIGSTOP $(pidof signal_handler)
[nyx@hpNyx ~]$ kill -s SIGKILL $(pidof signal_handler)
[nyx@hpNyx ~]$ kill -s SIGUSR1 $(pidof signal_handler)
[nyx@hpNyx ~]$
```

```
[nyx@hpNyx S02]$ ./signal_handler
^CSe lanzo la señal INT
Terminado (killed)
[nyx@hpNyx S02]$
```

```
[nyx@hpNyx ~]$ sudo strace -p $(pidof signal_handler)
[sudo] contraseña para nyx:
strace: Process 11113 attached
--- SIGINT {si_signo=SIGINT, si_code=SI_KERNEL} ---
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x2), ...}) = 0
getrandom("\xe1\x2d\xe9\xc6\x31\xe0\x42\xf1", 8, GRND_NONBLOCK) = 8
brk(NULL) = 0x5757882d5000
brk(0x5757882f6000) = 0x5757882f6000
write(1, "Se lanzo la se\u00303\u0261a1 INT\n", 23) = 23
rt_sigreturn({mask=[]}) = 0
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
+++ killed by SIGKILL +++
[nyx@hpNyx ~]$
```

```
[nyx@hpNyx ~]$ kill -s SIGKILL $(pidof signal_handler)
[nyx@hpNyx ~]$
```

d)

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

```
int main(int argc, char *argv[])
{
    pid_t b = fork();
    if (b == 0) {
        printf("B is my name.\nMy PID:%d\nMy PPID:%d\n\n", getpid(), getppid());
        pid_t d = fork();
        if (d==0) {
            printf("D is my name.\nMy PID:%d\nMy PPID:%d\n\n", getpid(), getppid());
            sleep(10);
            return 0;
        }
        pid_t e = fork();
        if (e==0) {
            printf("E is my name.\nMy PID:%d\nMy PPID:%d\n\n", getpid(), getppid());
            sleep(10);
            return 0;
        }
        pid_t f = fork();
        if (f==0) {
            printf("F is my name.\nMy PID:%d\nMy PPID:%d\n\n", getpid(), getppid());
            sleep(10);
            return 0;
        }
        wait(NULL);
        return 0;
    }

    pid_t c = fork();
    if(c==0){
        printf("C is my name.\nMy PID:%d\nMy PPID:%d\n\n", getpid(), getppid());
        sleep(10);
        return 0;
    }
    printf("A is my name.\nMy PID:%d\nMy PPID:%d\n\n", getpid(), getppid());
    wait(NULL);
    return 0;
}
```

```
A is my name.  
My PID:2874  
My PPID:2609
```

```
B is my name.  
My PID:2875  
My PPID:2874
```

```
C is my name.  
My PID:2876  
My PPID:2874
```

```
D is my name.  
My PID:2877  
My PPID:2875
```

```
E is my name.  
My PID:2878  
My PPID:2875
```

```
F is my name.  
My PID:2879  
My PPID:2875
```

```
bash  
└─ forks2 | ./forks2  
    └─ forks2 | ./forks2  
        └─ forks2 | ./forks2  
            └─ forks2 | ./forks2  
                └─ forks2 | ./forks2
```

2-a: a y b)

```
[nyx@hpNyx Sistemas-Operativos-2]$ mkfifo pruebafifo
[nyx@hpNyx Sistemas-Operativos-2]$ ls -al pruebafifo
prw-r--r-- 1 nyx nyx 0 ago 22 15:02 pruebafifo
```

c)

```
[nyx@hpNyx Sistemas-Operativos-2]$ bc < pruebafifo
4
```

d)

```
[nyx@hpNyx Sistemas-Operativos-2]$ echo "2+2" > pruebafifo
```

2-b: a)

```
[nyx@hpNyx Sistemas-Operativos-2]$ sudo cat /etc/issue | awk '{ print toupper($0) }' >> /tmp/issue.out
[nyx@hpNyx Sistemas-Operativos-2]$ cat /tmp/issue.out
\${PRETTY_NAME} \R (\L)
```

b)

```
[nyx@hpNyx Sistemas-Operativos-2]$ ip addr show | grep -E '10\.230\.[0-9]{1,3}\.[0-9]{1,3}'
[nyx@hpNyx Sistemas-Operativos-2]$ ip addr show | grep -E '192\.168\.[0-9]{1,3}\.[0-9]{1,3}'
    inet 192.168.1.63/24 brd 192.168.1.255 scope global dynamic noprefixroute wlo1
```

c)

```
[nyx@hpNyx Sistemas-Operativos-2]$ ls /bin | sort | tail -n 3
zvbi-chains
zvbid
zvbi-ntsc-cc
```

d)

```
[nyx@hpNyx Sistemas-Operativos-2]$ ls -l /usr/bin | wc -l
4479
```

e)

```
[nyx@hpNyx Sistemas-Operativos-2]$ echo $(groups | wc -w) + $(users | wc -w) | bc
10
[nyx@hpNyx Sistemas-Operativos-2]$ groups | wc -w
9
[nyx@hpNyx Sistemas-Operativos-2]$ users | wc -w
1
```



2-c: a-b)

```
[nyx@hpNyx Sistemas-Operativos-2]$ mkfifo /tmp/pruebafifo2  
[nyx@hpNyx Sistemas-Operativos-2]$ ls -al /tmp/pruebafifo2  
prw-r--r-- 1 nyx nyx 0 ago 22 15:40 /tmp/pruebafifo2
```

c)

```
[nyx@hpNyx ~]$ echo Hola > /tmp/pruebafifo2  
[nyx@hpNyx ~]$
```

```
[nyx@hpNyx Sistemas-Operativos-2]$ cat /tmp/pruebafifo2  
Hola
```

Realice un programa en C utilizando un PIPE donde el proceso hijo le mande tareas a imprimir por salida estándar al proceso padre

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

int main(int argc, char *argv[])
{
    int fildes[2];
    const int BUFFER_SIZE = 128;
    char buffer[BUFFER_SIZE];
    ssize_t nbytes;

    if(pipe(fildes)==-1){
        printf("Error en la creacion del pipe\n");
        return -1;
    }
    switch (fork()) {
    case -1:
        //Error
        printf("Error en la creacion del hijo\n");
        break;
    case 0:
        //Hijo
        close(fildes[0]);
        write(fildes[1], "Mensaje del hijo\n", 17);
        close(fildes[1]);
        sleep(1);
        exit(0);
        break;
    default:
        //Padre
        close(fildes[1]);
        nbytes = read(fildes[0], buffer, BUFFER_SIZE);
        buffer[nbytes] = '\0';
        printf("%s", buffer);
        close(fildes[0]);
        break;
    }
    return 0;
}
```

**Realice dos programas en C para crear un FIFO en el cual se puedan comunicar un proceso cliente y un servidor. Para ello el proceso cliente le debe enviar la cadena “enviando datos al servidor!”. Si un proceso intenta leer datos de él, ¿en qué estado queda? ¿Cómo logro cambiarlo de estado?**

Si un proceso, en este caso el servidor, intenta leer el archivo FIFO, este queda “bloqueado” esperando a que algun proceso escriba algo en el FIFO. Cuando un proceso escribe algo en el FIFO (el cliente en nuestro caso), el proceso servidor se desbloquea y lee lo que hay en el archivo.

#### **Server.c**

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

```
int main(int argc, char *argv[])
```

```
{
    char buffer[128];
    char* path = "./fitito";
    mkfifo(path,0666);
```

```
    int fd = open(path, O_RDONLY);
```

```
    read(fd, buffer, 128);
    printf("%s\n",buffer);
    close(fd);
    unlink(path);
```

```
    return 0;
```

```
}
```

**Client.c**

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

```
int main(int argc, char *argv[])
{
    char* path = "./fitito";
    int fd = open(path, O_WRONLY);

    char* mensaje = "enviando datos al servidor!";
    write(fd, mensaje, 27);
    close(fd);

    return 0;
}
```

**-Si no hay mensaje alguno, el lector queda “bloqueado” hasta que el escritor haga algo**

[illegible]

#### Line-creator.c

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

```
int main(int argc, char *argv[])
{
    char buffer[128];
    char* path = "./cola";
    mkfifo(path,0666);
    return 0;
}
```

#### Line-reader.c

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

```
int main(int argc, char *argv[])
{
    char buffer[128];
    char* path = "./cola";

    while(1){
        int fd = open(path, O_RDONLY);
        read(fd, buffer, 128);
        printf("%s\n",buffer);
        close(fd);
    };

    return 0;
}
```

**Line-sender.c**

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

```
int main(int argc, char *argv[])
{
    char buffer[128];
    char* path = "./cola";

    int fd = open(path, O_WRONLY);
    char* mensaje = "enviando datos al servidor!";
    write(fd, mensaje, 27);
    close(fd);

    return 0;
}
```