

Práctico 1:

Shell, procesos e IPC

Sistemas Operativos

2017

Ejercicio 1 Dado el siguiente programa C:

```
1 #include <stdio.h>
2
3 int g=3, z=4;
4
5 int f(int * x, int y)
6 {
7     static int c=5;
8
9     return *(x+1)+y+(c++); //(1)
10 }
11
12 int main(void)
13 {
14     int a[5]={1,2,3,4,5};
15     int r = f(a,a[0]) + f(&g,a[g]);
16     printf("r=%d\n",r);
17     return r;
18 }
```

- Determine el valor final de las variables **r** y **c**. ¿Por qué la segunda invocación a **f** (línea 15) retorna 14?
- Dibuje el layout de la memoria del proceso indicando dónde están almacenadas todas las variables en el punto (1) de la primera invocación a **f**.
- Utilice el comando **objdump** para visualizar los segmentos del programa y la tabla de símbolos.

Ejercicio 2 Listar los procesos que están en estado de ejecución (running). Ver el comando `ps`.

Ejercicio 3 Redirigir la salida del comando `ls -l` al archivo `ls.out`.

Ejercicio 4 Escribir un comando que permita filtrar threads del sistema. Ayuda: el comando `ps -ax` lista todos los procesos (y threads) corriendo en el sistema. La salida de `ps -ax` es un archivo de texto donde cada línea tiene 5 columnas. La última es el nombre del comando (system thread). Los nombres de los threads aparecen entre corchetes (ej: `[hd-audio0]`).

El comando `awk` permite ejecutar programas (scripts) `awk`. Por ejemplo, `awk '{ print $1}' file.txt` lista la primera columna del archivo (columnas separadas por tabs). Una expresión regular en `awk` tiene la forma `var /E/A`, donde `var` es una variable (ej: `$2`), `E` es la expresión regular y `A` es la acción a ejecutar si hace matching.

Ejercicio 5 Dado el siguiente programa (ej5.c):

```
int main()
{
    write(1, "Hola\n", 5);
    write(2, "mundo\n", 6);
    return 0;
}
```

Compilarlo y ejecutarlo, redirigiendo la salida estándar (1) al archivo `f1` y la salida de errores estándar (2) al archivo `f2`.

Luego concatenar en el archivo `f1` el contenido de `f2`.

Ejercicio 6 Correr el comando `vi carta.txt` como un proceso de fondo (background). Pasarlo al frente (foreground). Durante la edición suspenderlo (Ctrl-z). Luego volver a la edición. Para editar en `vi` pulsar `i` (insert), para salir `ESC :x` (graba y sale) o `:q!` (sale sin grabar).

Ejercicio 7 Explicar el comportamiento y en qué orden se lanzan los procesos en los siguientes comandos:

- `grep Hola ej5.c ; echo OK`
- `grep chau ej5.c ; echo OK`
- `grep -q Hola ej5.c && echo OK`
- `grep chau ej5.c || echo OK`

Ejercicio 8 Dado el siguiente shell (bash) script, describir su propósito.

```
#!/bin/bash

l=0
let t=0
for i in $*
do
    l='wc -l $i | awk '{ print $1}''
    echo $l
    let t=t+$l
done

echo "Total: _$t , _args :$#"

```

Ejercicio 9 La biblioteca estándar de C ofrece la función `int system(const char *string)`, la cual crea un subproceso generado por el comando pasado en su argumento.

Definir `system()` en términos de las llamadas al sistema `fork()`, `exec()` y `wait()` (ver páginas del manual)¹.

Ejercicio 10 Escribir un programa C `mypipe.c` que reciba 3 argumentos en la línea de comandos y emule los siguientes comandos de shell: `cmd1 | cmd2`, `cmd1 < input_file, cmd > output_file`, `cmd >> output_file`, `cmd1 ; cmd2`, `cmd1 || cmd2` y `cmd1 && cmd2`.

Ejercicio 11 Hacer un programa C que capture la señal `SIGTERM`. El manejador de la señal deberá preguntar al usuario si realmente desea finalizar el proceso. En caso de respuesta afirmativa finalizar con un estado de salida -1.

Ejercicio 12 Implementar un programa en C que comunique al proceso padre con un proceso hijo por medio de un pipe (ver `pipe()` syscall). El proceso padre deberá enviarle un string y el hijo deberá responderle con el string invertido. Finalmente el proceso padre deberá mostrar el identificador de proceso (pid) del hijo y la cadena recibida.

Ejercicio 13 Idem al anterior pero haciendo dos programas que se comuniquen por medio de un FIFO (named pipe). Ayuda: ver el comando `mkfifo`.

¹Usar el comando *man*.

Ejercicio 14 El siguiente programa assembly realiza una llamada al sistema *exit(42)* en Linux, compilarlo y correrlo.

```
; file: exit_42.asm
; compile with:
; nasm -f elf exit_42.asm
; ld -s -nostdlib -o exit_42 exit_42.o
;
; try it using:
; ./exit_42 ; echo $?

BITS 32
GLOBAL _start
SECTION .text
_start:
    mov eax, 1    ; set syscall number 1 (EXIT)
    mov ebx, 42   ; argument of EXIT
    int 0x80      ; do the syscall
```

- a) Hacer un programa equivalente en C, luego compilarlo y comparar sus tamaños.
- b) Dar una explicación a la diferencia de tamaños entre las dos versiones (C y assembly).