

Ejercicio 1:

¿Por qué aparecen números diferentes cada vez?

Porque se crea un nuevo proceso cada vez que se ejecuta y ese proceso debe ser alocado en la CPU. Cada vez que esto ocurre, el CPU le asigna un id que cambia debido a que el proceso es nuevo.

```
os@debian:~$ cd Documents
os@debian:~/Documents$ gcc -o wtf hello_world.c
os@debian:~/Documents$ ./helloWorld
Hello World
3259
os@debian:~/Documents$ gcc -o forkExec fork.c
os@debian:~/Documents$ ./forkExec
3266
os@debian:~/Documents$ ./helloWorld
Hello World
3268
os@debian:~/Documents$ ./helloWorld
Hello World
3269
os@debian:~/Documents$ ./helloWorld
Hello World
3270
os@debian:~/Documents$ ./helloWorld
Hello World
3271
os@debian:~/Documents$ █
```

¿Por qué aparecen dos números distintos a pesar de que estamos ejecutando un único programa?

Porque se ejecutan dos veces el programa, por lo que se alocan dos procesos distintos. Uno es el padre y el otro es el hijo.

```
os@debian: ~/Documents
File Edit View Terminal Help
gcc -o forkExec fork.c ^C
os@debian:~/Documents$ gcc -o forkExec fork.c
os@debian:~/Documents$ ./forkExec
3402
os@debian:~/Documents$ Hello World
3403
./forkExec
3404
os@debian:~/Documents$ Hello World
3405
^C
os@debian:~/Documents$ ^C
os@debian:~/Documents$ gcc -o forkExec fork.c
os@debian:~/Documents$ ./forkExec
3412
os@debian:~/Documents$ Hello World
3413

os@debian:~/Documents$ gcc -o forkExec fork.c
os@debian:~/Documents$ ./forkExec
3426
os@debian:~/Documents$ Hello World
3427
```

¿Por qué el primer y el segundo números son iguales?

Porque ambos son del mismo proceso padre.

En la terminal, ejecute el comando `top`(que despliega el top de procesos en cuanto a consumo de CPU) y note cuál es el primer proceso en la lista (con identificador 1).
¿Para qué sirve este proceso?

Xorg es un servidor de display para Linux. Es un requisito para todas las aplicaciones que necesiten de GUI.

```
os@debian: ~/Documents
File Edit View Terminal Help
top - 21:55:28 up 4:44, 2 users, load average: 0.00, 0.00, 0.00
Tasks: 117 total, 1 running, 116 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.3%us, 2.0%sy, 0.0%ni, 96.7%id, 0.0%wa, 1.0%hi, 0.0%si, 0.0%st
Mem: 384644k total, 369044k used, 15600k free, 93476k buffers
Swap: 392184k total, 60k used, 392124k free, 156280k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 1951 root        20   0 51660 26m 7976 S   1.0   7.0   0:45.16 Xorg
2944 os          20   0 19824 9816 8124 S   0.7   2.6   0:01.30 metacity
   1 root        20   0 2036  708  628 S   0.0   0.2   0:01.44 init
   2 root        20   0   0   0   0 S   0.0   0.0   0:00.01 kthreadd
   3 root        RT   0   0   0   0 S   0.0   0.0   0:00.00 migration/0
   4 root        20   0   0   0   0 S   0.0   0.0   0:01.61 ksoftirqd/0
   5 root        RT   0   0   0   0 S   0.0   0.0   0:00.00 watchdog/0
   6 root        20   0   0   0   0 S   0.0   0.0   0:00.00 events/0
   7 root        20   0   0   0   0 S   0.0   0.0   0:00.00 cpuset
   8 root        20   0   0   0   0 S   0.0   0.0   0:00.00 khelper
   9 root        20   0   0   0   0 S   0.0   0.0   0:00.00 netns
  10 root        20   0   0   0   0 S   0.0   0.0   0:00.00 async/mgr
  11 root        20   0   0   0   0 S   0.0   0.0   0:00.00 pm
  12 root        20   0   0   0   0 S   0.0   0.0   0:00.00 sync_supers
  13 root        20   0   0   0   0 S   0.0   0.0   0:00.00 bdi-default
  14 root        20   0   0   0   0 S   0.0   0.0   0:00.00 kintegrityd/0
  15 root        20   0   0   0   0 S   0.0   0.0   0:00.04 kblockd/0
```

Ejercicio 2

```
os@debian: ~/Documents
File Edit View Terminal Help
Desktop Documents Downloads Music Pictures Public Templates Videos
os@debian:~$ cd Documents
os@debian:~/Documents$ strace ./ejercicioDos text1.txt text2.txt
execve("./ejercicioDos", ["/.ejercicioDos", "text1.txt", "text2.txt"], [/* 35 va
rs */]) = 0
brk(0) = 0x913c000
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7
71d000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=64828, ...}) = 0
mmap2(NULL, 64828, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb770d000
close(3) = 0
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
open("/lib/i686/cmov/libc.so.6", O_RDONLY) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\0n\1\0004\0\0\0"...
, 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1327556, ...}) = 0
mmap2(NULL, 1337704, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb
75c6000
mprotect(0xb7706000, 4096, PROT_NONE) = 0
mmap2(0xb7707000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWR
ITE, 3, 0x140) = 0xb7707000
```

```
os@debian: ~/Documents
File Edit View Terminal Help
write(1, "rgwe\n", 5rgwe
) = 5
write(1, "rg\n", 3rg
) = 3
write(1, "werg\n", 5werg
) = 5
write(1, "weg\n", 4weg
) = 4
write(1, "rw\n", 3rw
) = 3
write(1, "erge\n", 5erge
) = 5
write(1, "rg\n", 3rg
) = 3
write(1, "weg\n", 4weg
) = 4
read(3, "", 4096) = 0
write(4, "ergnweorgne\nrgwe\nrg\nwerg\nweg\nrw\n...", 44) = 44
close(4) = 0
munmap(0xb771a000, 4096) = 0
close(3) = 0
munmap(0xb771c000, 4096) = 0
exit_group(0) = ?
os@debian:~/Documents$
```

¿Por qué la primera llamada que aparece es execve?

Porque estamos ejecutan un programa referido por un pathname. El programa que ejecutamos da el pathname que execve necesita y luego recibe los argumentos del programa.

¿Qué significan los resultados (números que están luego del signo '=')?

Son valores de retorno de las llamadas al sistema. open() regresa valores positivos para decir que se realizó exitosamente, close() devuelve 0 y lseek devuelve los bits que se movieron.

¿Por qué entre las llamadas realizadas por usted hay un read vacío?

Porque el programa intenta leer el archivo destino que está vacío.

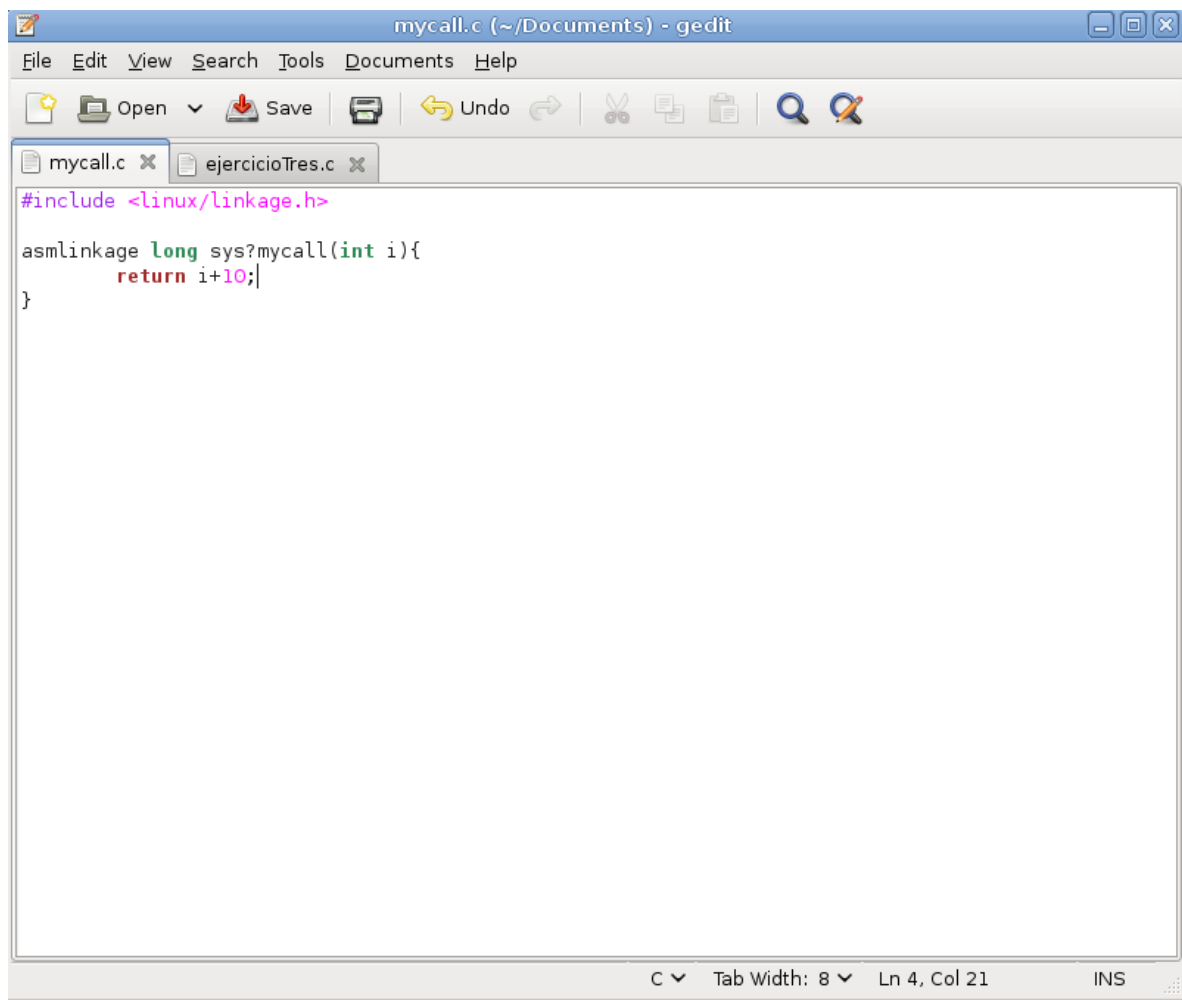
Identifique tres servicios distintos provistos por el sistema operativo en este strace. Liste y explique brevemente las llamadas a sistema que corresponden a los servicios identificados (puede incluir read, write, open o close que el sistema haga por usted, no los que usted haya producido directamente con su programa).

open(): Esta llamada al sistema permite abrir el archivo en el modo que se le haya sido especificado.

fstat(): Retorna información acerca de un archivo.

write(): Escribe hasta una cierta cantidad de bytes del buffer empezando en el buf hasta el archivo referido por el descriptor de archivos.

Ejercicio 3

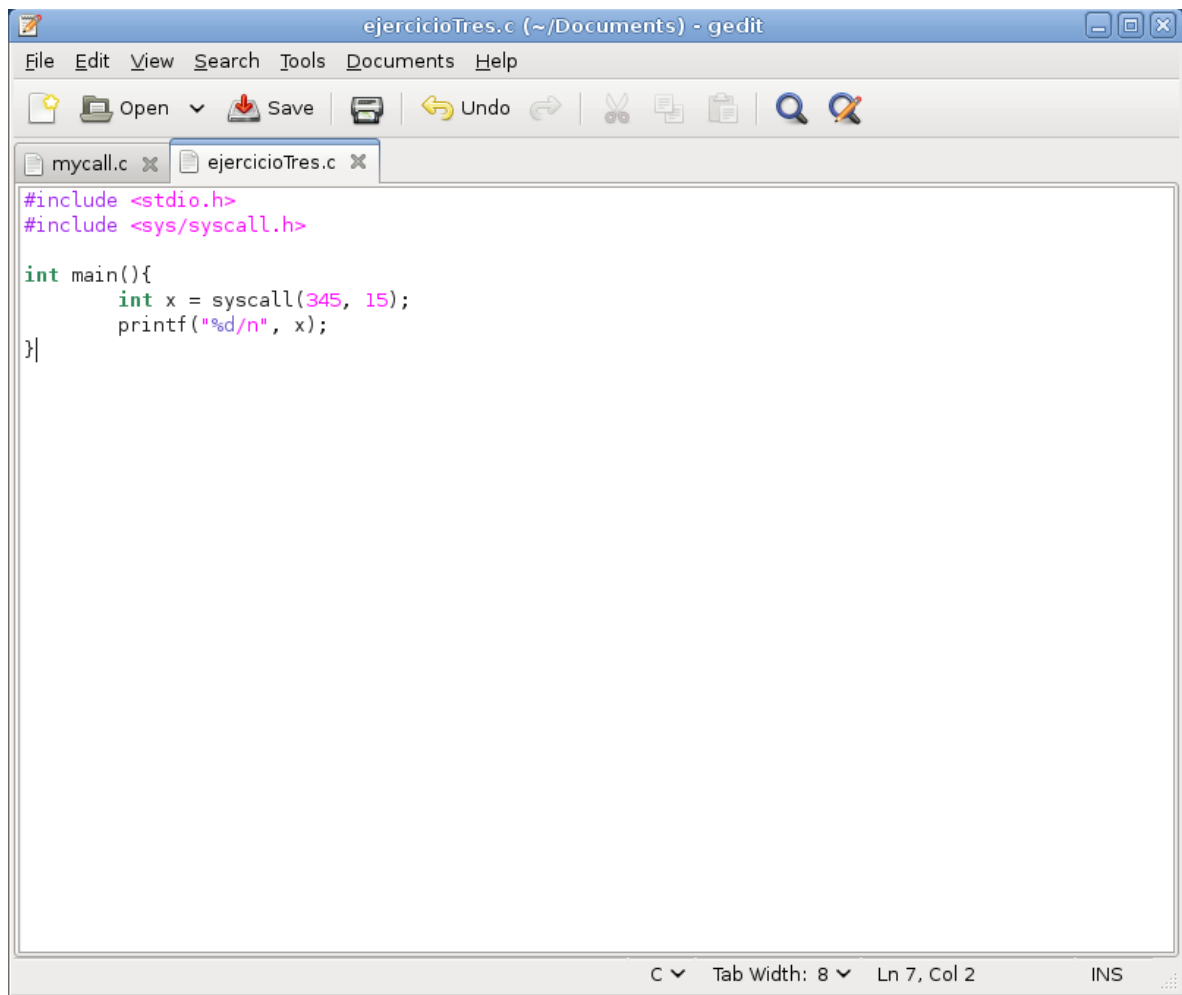


The image shows a screenshot of a gedit text editor window. The title bar at the top reads "mycall.c (~/.Documents) - gedit". Below the title bar is a menu bar with the following options: File, Edit, View, Search, Tools, Documents, and Help. Underneath the menu bar is a toolbar containing icons for Open, Save, Undo, and other standard editing functions. The editor has two tabs open: "mycall.c" and "ejercicioTres.c". The "mycall.c" tab is currently active, displaying the following C code:

```
#include <linux/linkage.h>

asm linkage long sys?mycall(int i){
    return i+10;
}
```

The code is color-coded: the include directive is in magenta, the function name and arguments are in green, and the return statement is in red. The status bar at the bottom of the window shows "C", "Tab Width: 8", "Ln 4, Col 21", and "INS".



The image shows a screenshot of a gedit text editor window. The title bar indicates the file is "ejercicioTres.c (~/.Documents) - gedit". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for Open, Save, Print, Undo, Redo, Cut, Copy, Paste, Find, and Replace. Two tabs are open: "mycall.c" and "ejercicioTres.c". The active tab, "ejercicioTres.c", contains the following C code:

```
#include <stdio.h>
#include <sys/syscall.h>

int main(){
    int x = syscall(345, 15);
    printf("%d/n", x);
}
```

The status bar at the bottom shows "C", "Tab Width: 8", "Ln 7, Col 2", and "INS".

```
os@debian: ~/Documents
File Edit View Terminal Help
./ejercicioTres: line 55: typedef: command not found
./ejercicioTres: line 56: typedef: command not found
./ejercicioTres: line 57: typedef: command not found
./ejercicioTres: line 62: __extension__: command not found
./ejercicioTres: line 63: __extension__: command not found
./ejercicioTres: line 71: __extension__: command not found
./ejercicioTres: line 72: __extension__: command not found
./ejercicioTres: line 78: __extension__: command not found
./ejercicioTres: line 79: __extension__: command not found
./ejercicioTres: line 80: __extension__: command not found
./ejercicioTres: line 81: __extension__: command not found
./ejercicioTres: line 82: __extension__: command not found
./ejercicioTres: line 83: __extension__: command not found
./ejercicioTres: line 84: __extension__: command not found
./ejercicioTres: line 85: __extension__: command not found
./ejercicioTres: line 86: __extension__: command not found
./ejercicioTres: line 87: __extension__: command not found
./ejercicioTres: line 88: syntax error near unexpected token `}'
./ejercicioTres: line 88: `__extension__ typedef struct { int __val[2]; } __fsid
_t;'
os@debian:~/Documents$ gcc -o ejercicioTres ejercicioTres.c
os@debian:~/Documents$ ./ejercicioTres
25
os@debian:~/Documents$
```

¿Qué ha modificado aquí, la interfaz de llamadas de sistema o el API? Justifique su respuesta.

El API, puesto que una nueva llamada al sistema forma parte del API del kernel.

¿Por qué usamos el número de nuestra llamada de sistema en lugar de su nombre?

Porque es el número especificado en la interface del lenguaje assembly de la llamada de sistema.

¿Por qué las llamadas de sistema existentes como read o fork se pueden llamar por nombre?

Porque tiene un wrapper encima del fork.