Universidad del Valle de Guatemala
Sistemas Operativos
Tomás Gálvez

Andrea María Cordón Mayén
**Carné:** 16076
**Fecha:** 31/01/2019

# Laboratorio 1

## Ejercicio 1:

Se creó el primer programa que se solicitaba para este ejercicio y después de ejecutarlo, este fue el resultado:

```
os@debian:~/Documents/SistemasOperativos/Laboratorios/Laboratorio1$ ./ejer
Hello World!2327
os@debian:~/Documents/SistemasOperativos/Laboratorios/Laboratorio1$ ./ejer
Hello World!2328
os@debian:~/Documents/SistemasOperativos/Laboratorios/Laboratorio1$ ./ejer
Hello World!2329
os@debian:~/Documents/SistemasOperativos/Laboratorios/Laboratorio1$ ./ejer
Hello World!2330
os@debian:~/Documents/SistemasOperativos/Laboratorios/Laboratorio1$ ./ejer
Hello World!2331
os@debian:~/Documents/SistemasOperativos/Laboratorios/Laboratorio1$ ./ejer
Hello World!2332
os@debian:~/Documents/SistemasOperativos/Laboratorios/Laboratorio1$ ./ejer
Hello World!2333
os@debian:~/Documents/SistemasOperativos/Laboratorios/Laboratorio1$ ./ejer
Hello World!2334
os@debian:~/Documents/SistemasOperativos/Laboratorios/Laboratorio1$ ./ejer
Hello World!2335
```

Posteriormente, se realizó el segundo programa de este ejercicio y se obtuvo lo siguiente después de ejecutarlo:

```
os@debian:~/Documents/SistemasOperativos/Laboratorios/Laboratorio1/Ejercicio1$ .
/ejercicio12
2247
Hello World!2247
os@debian:~/Documents/SistemasOperativos/Laboratorios/Laboratorio1/Ejercicio1$ H
ello World!2248
/ejercicio12
2249
Hello World!2249
os@debian:~/Documents/SistemasOperativos/Laboratorios/Laboratorio1/Ejercicio1$ H
ello World!2250
/ejercicio12
2251
Hello World!2251
os@debian:~/Documents/SistemasOperativos/Laboratorios/Laboratorio1/Ejercicio1$ H
ello World!2252
os@debian:~/Documents/SistemasOperativos/Laboratorios/Laboratorio1/Ejercicio1$
```

A continuación, se responden las preguntas de este ejercicio:

**Preguntas:**

1. **Para el primer programa ¿Por qué aparecen números diferentes cada vez?**
    a. Porque cada vez que se ejecuta el programa, se crea un nuevo proceso que se encarga de realizar lo que el programa indica.
2. **En el segundo programa ¿por qué aparecen dos números distintos a pesar de que estamos ejecutando un único programa?**
    a. Porque la llamada fork() crea un proceso padre y un proceso hijo, lo cual hace que cuando se ejecute el programa no solo se imprima el número del proceso del padre, sino que también el del hijo.
3. **¿Por qué el primer y el segundo número son iguales?**
    a. Porque el proceso hijo utiliza el mismo pc, los mismos registros del procesador y los mismos archivos que el proceso padre.
4. **Después de ejecutar el comando top ¿para qué sirve este proceso?**
    a. Nos muestra un listado de todos los procesos que se están ejecutando en el procesador en el momento. No solo nos los muestra, sino que también nos da el número del proceso, el usuario que lo está realizando, porcentajes de memoria y procesador y muchas otras cosas.

## Ejercicio 2:

Esto fue lo que resultó después del ejercicio 2:

```
execve("./ejercicio2", ["./ejercicio2", "copiar.txt", "copiado.txt"], [/* 35 var
s */]) = 0
brk(0)                                  = 0x9c79000
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
795000
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) = 0xb7785000
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\3\0\3\0\1\0\0\0\0n\1\0004\0\0\0"..., 51
2) = 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
763e000
mprotect(0xb777e000, 4096, PROT_NONE)   = 0
mmap2(0xb777f000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRI
TE, 3, 0x140) = 0xb777f000
mmap2(0xb7782000, 10600, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMO
US, -1, 0) = 0xb7782000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7
63d000
set_thread_area({entry_number:-1 -> 6, base_addr:0xb763d6c0, limit:1048575, seg_
32bit:1, contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, usea
ble:1}) = 0
mprotect(0xb777f000, 8192, PROT_READ)   = 0
mprotect(0xb77b3000, 4096, PROT_READ)   = 0
munmap(0xb7785000, 64828)               = 0
brk(0)                                  = 0x9c79000
brk(0x9c9a000)                          = 0x9c9a000
open("copiar.txt", O_RDONLY)            = 3
open("copiado.txt", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 4
fstat64(3, {st_mode=S_IFREG|0644, st_size=39, ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7
794000
read(3, "Este ejercicio fue un dolor de c"..., 4096) = 39
fstat64(4, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7
793000
read(3, "", 4096)                       = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7
792000
close(3)                                = 0
munmap(0xb7794000, 4096)                = 0
write(4, "Este ejercicio fue un dolor de c"..., 39) = 39
```

**Preguntas:**

1. **¿Por qué la primera llamada que aparece es execve?**
    a. Porque es la función que se dedica a ejecutar un programa específico a un archivo o a el nombre de un archivo específico.
2. **¿Qué significan los resultados?**
    a. Es un “resumen” de todo lo que realiza el proceso para llevar a cabo la acción que le fue solicitada. Como se puede observar en la imagen anterior, se puede ver el número del proceso, el momento cuando abre los archivos, el momento cuando lee el archivo fuente, el momento en cuando lo escribe en el nuevo archivo y el momento cuando cierra los archivos. Muestra muchas cosas más, pero esas son las más importantes.
3. **¿Por qué entre las llamadas realizadas por usted hay un read vacío?**
    a. Porque no solo se hace la lectura a los caracteres del documento, sino que también se lee el final de este. Ese es el read vacío, la lectura del final.
4. **Servicios provistos por el sistema operativo en el strace:**
    a. open(): Hace una llamada para abrir el archivo fuente que le solicitamos leer. También abre el archivo sobre el que le solicitamos escribir. Básicamente se encarga de abrir todo lo que deba para llevar a cabo la acción solicitada. Es importante que mencione que no solo abre los archivos, sino que también el espacio en la memoria dónde almacenará todo.
    b. write(): Escribe todo lo que le solicitamos que escribiera. Es una llamada que se dedica a escribir, pero no solo escribe lo que necesitamos en el archivo destino, sino que también lo que hayamos solicitado escribir en pantalla.
    c. close(): Cierra todo lo que la función open() haya abierto, ya sean archivos, espacios de memoria, etc.

## Ejercicio 3:

**Preguntas:**

1. **¿Qué ha modificado aquí, la interfaz de llamadas de sistema o el API?**
    a. La interfaz de llamadas del sistema ya que lo que hicimos fue modificarla para poder ejecutar una llamada creada personalmente. Es decir que modificamos la interfaz de llamadas para tener acceso a ella y poder hacer la llamada correspondiente a nuestra propia llamada.
2. **¿Por qué usamos el número de nuestra llamada de sistema en lugar de su nombre?**
    a. Porque como no se modificó el API, entonces el nombre de nuestra llamada no está registrada en él y no se puede utilizar para llamarla.
3. **¿Por qué las llamadas de sistema existentes como read o fork se pueden llamar por nombre?**
    a. Porque ya se encuentran incluidas en el API.