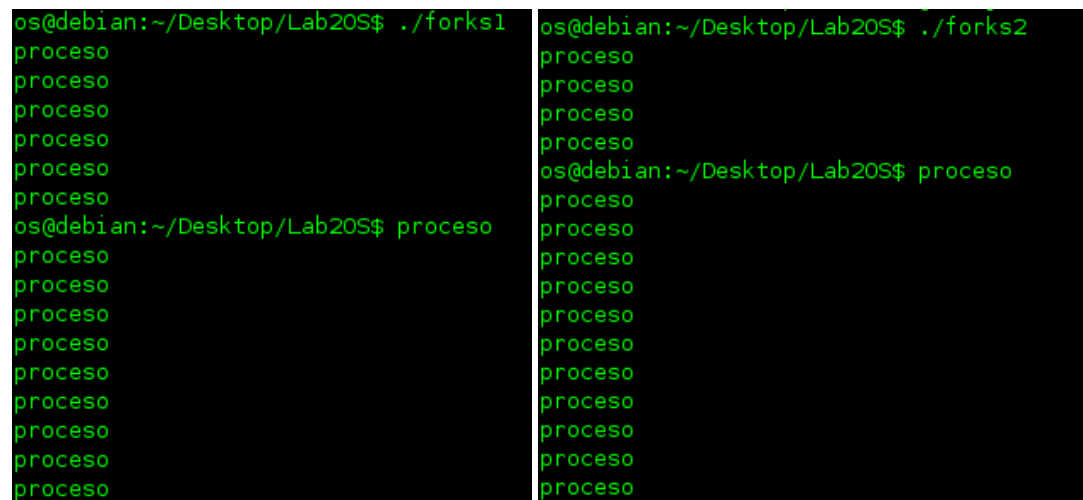


## Laboratorio 2

### Ejercicio 1



```
os@debian:~/Desktop/Lab20S$ ./forks1
proceso
proceso
proceso
proceso
proceso
proceso
os@debian:~/Desktop/Lab20S$ proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso

os@debian:~/Desktop/Lab20S$ ./forks2
proceso
proceso
proceso
proceso
proceso
os@debian:~/Desktop/Lab20S$ proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
proceso
```

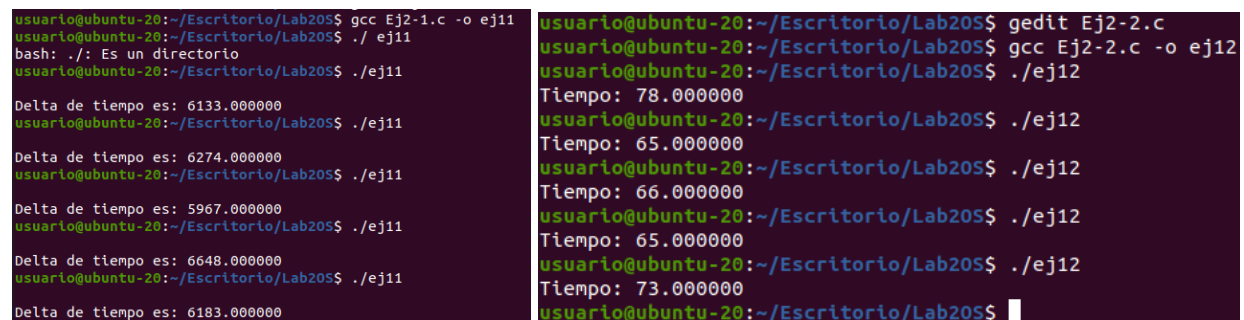
¿Cuántos procesos se crean en cada uno de los programas?

16 procesos en ambos

¿Por qué hay tantos procesos en ambos programas cuando uno tiene cuatro llamadas fork() y el otro sólo tiene una?

Al ejecutar el fork cuatro veces, se crean procesos hijos 4 veces. La cantidad de procesos será  $2^4$ . Es decir 16. Ambas maneras de hacerlo son iguales ya que virtualmente solo se repite el fork 4 veces. Si el ciclo for se repite 4 veces, solo se va a ejecutar lo que esta adentro, es decir el fork, 4 veces. Eso hace que sea lo mismo a ejecutarlo 4 veces y por lo tanto se crean la misma cantidad de procesos.

### Ejercicio 2



```
usuario@ubuntu-20:~/Escritorio/Lab20S$ gcc Ej2-1.c -o ej11
usuario@ubuntu-20:~/Escritorio/Lab20S$ ./ej11
bash: ./: Es un directorio
usuario@ubuntu-20:~/Escritorio/Lab20S$ ./ej11
Delta de tiempo es: 6133.000000
usuario@ubuntu-20:~/Escritorio/Lab20S$ ./ej11
Delta de tiempo es: 6274.000000
usuario@ubuntu-20:~/Escritorio/Lab20S$ ./ej11
Delta de tiempo es: 5967.000000
usuario@ubuntu-20:~/Escritorio/Lab20S$ ./ej11
Delta de tiempo es: 6648.000000
usuario@ubuntu-20:~/Escritorio/Lab20S$ ./ej11
Delta de tiempo es: 6183.000000

usuario@ubuntu-20:~/Escritorio/Lab20S$ gedit Ej2-2.c
usuario@ubuntu-20:~/Escritorio/Lab20S$ gcc Ej2-2.c -o ej12
usuario@ubuntu-20:~/Escritorio/Lab20S$ ./ej12
Tiempo: 78.000000
usuario@ubuntu-20:~/Escritorio/Lab20S$ ./ej12
Tiempo: 65.000000
usuario@ubuntu-20:~/Escritorio/Lab20S$ ./ej12
Tiempo: 66.000000
usuario@ubuntu-20:~/Escritorio/Lab20S$ ./ej12
Tiempo: 65.000000
usuario@ubuntu-20:~/Escritorio/Lab20S$ ./ej12
Tiempo: 73.000000
usuario@ubuntu-20:~/Escritorio/Lab20S$
```

¿Cuál, en general, toma tiempos más largos?

El primero, en donde no se usa concurrencia

¿Qué causa la diferencia de tiempo, o por qué se tarda más el que se tarda más?

Al usar concurrencia, se pueden llevar a cabo diferentes procesos de forma simultánea. Esto hace que vaya más rápido al primero, en donde se realizan los fors de manera secuencial.

### Ejercicio 3

```
usuario@ubuntu-20: ~/Escritorio/Lab2OS
Media: 1000 1972 0,22 0,00 evolution-alarm
Media: 1000 1994 0,20 0,00 gsd-wacom
Media: 1000 1999 0,20 0,00 gsd-xsettings
Media: 1000 2227 57,57 8,71 gnome-terminal-
Media: 1000 2271 0,23 0,02 update-notifier
Media: 0 2468 2,79 0,00 kworker/1:0-mm_per
Media: 0 2507 4,71 0,00 kworker/0:0-mm_per
Media: 0 2552 50,93 0,01 kworker/u6:2-event
Media: 0 2578 14,57 0,02 kworker/u6:0-event
Media: 0 2739 4,73 0,02 kworker/u6:3-event
Media: 0 3766 0,20 0,00 packagekitd
Media: 1000 3983 0,04 0,00 bash
Media: 1000 3990 0,99 48,64 pidstat
usuario@ubuntu-20:~/Escritorio/Lab2OS$ pidstat -w 47 1

usuario@ubuntu-20: ~/Escritorio
usuario@ubuntu-20:~/Escritorio$ ./Ej12
01:50:07 0 10 1,00 0,00 ksoftirqd/0
01:50:07 0 11 9,00 0,00 rcu_sched
01:50:07 0 40 7,00 0,00 kworker/2:1-nm_percpu_wq
01:50:07 0 428 2,00 0,00 irq/18-vmwgfx
01:50:07 126 802 1,00 0,00 ntpd
01:50:07 1000 1005 16,00 1,00 Xorg
01:50:07 1000 1037 15,00 5,00 gnome-shell
01:50:07 1000 2227 7,00 5,00 gnome-terminal-
01:50:07 0 2468 2,00 0,00 kworker/1:0-events
01:50:07 0 2507 3,00 0,00 kworker/0:0-events
01:50:07 0 2552 6,00 0,00 kworker/u6:2-events_unbound
01:50:07 0 2578 13,00 0,00 kworker/u6:0-events_power_efficient
01:50:07 1000 3990 1,00 0,00 pidstat
```

¿Qué tipo de cambios de contexto incrementa notablemente en cada caso, y por qué?

Al mover las ventanas se observan cambios de contexto no voluntarios y cuando se escribe se notan cambios voluntarios. Esto es porque al mover la ventana se hacen cambios de contexto para realizar los cambios. Al escribir, se hacen llamadas al sistema para solucionar las interrupciones del teclado.

¿Qué diferencia hay en el número y tipo de cambios de contexto de entre programas?

Cuando no se hace uso de fork, hay mas cambios de contexto involuntarios. Cuando se usan forks, hay menos diferencia entre la cantidad de cambios voluntarios e involuntarios. Esto significa que cuando se usa el fork, hay mas voluntarios y menos involuntarios.

¿A qué puede atribuir los cambios de contexto voluntarios realizados por sus programas?

Se pueden atribuir a llamadas al sistema como los forks.

¿A qué puede atribuir los cambios de contexto involuntarios realizados por sus programas?

Se pueden atribuir a un proceso ocupando tiempo o recursos y otros procesos se ponen en espera.

¿Por qué el reporte de cambios de contexto para su programa con fork()s muestra cuatro procesos, uno de los cuales reporta cero cambios de contexto?

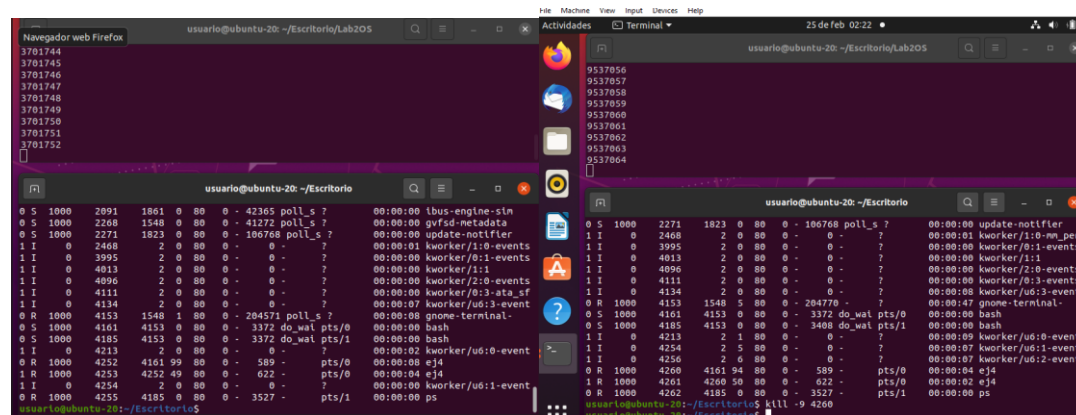
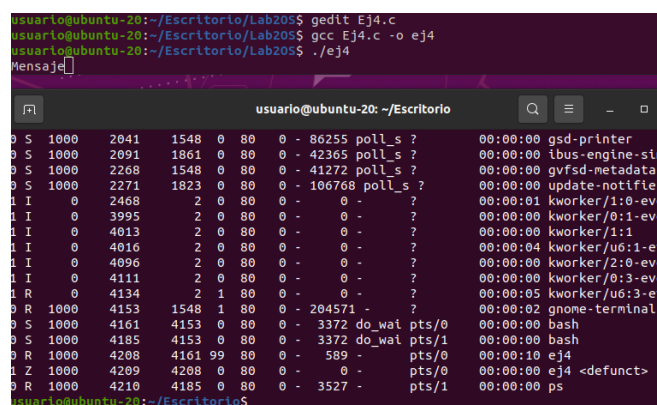
Cada proceso corresponde a los forks realizados en el programa. El cero corresponde al primer fork.

¿Qué efecto percibe sobre el número de cambios de contexto de cada tipo?

Quando se se hace alguna interrupción o hay un evento simultaneo, hay mas cambios no voluntarios.

## Ejercicio 4

```
usuario@ubuntu-20:~/Escritorio/Lab20$ gedit Ej4.c
usuario@ubuntu-20:~/Escritorio/Lab20$ gcc Ej4.c -o ej4
usuario@ubuntu-20:~/Escritorio/Lab20$ ./ej4
Mensaje
```



¿Qué significa la Z y a qué se debe?

La z significa proceso zombie o defunct. Este es un proceso terminado pero que no ha sido recogido por el padre. En este caso, el proceso padre ejecuta un while infinito que evita que finalice. El hijo finaliza después de escribir el mensaje.

Anote los números de proceso de tanto el padre como el hijo.

El padre 4252 y 4253 el hijo

¿Qué sucede en la ventana donde ejecutó su programa? y

El programa siguió corriendo hasta terminar el conteo. Luego de esto el programa finalizo a diferencia de cuando se corrió sin el kill. Esto se debe a que solo se a mato el proceso padre que tenia el while infinito.

¿Quién es el padre del proceso que quedó huérfano?

El proceso systemd

## Ejercicio 5

```
usuario@ubuntu-20:~/Escritorio/Lab20S$ gcc Ej5.c -o ej5
usuario@ubuntu-20:~/Escritorio/Lab20S$ ./ej5
I am a
I am b
b: Created new shared mem obj 4
b: Ptr created with value 0xb7535000
b: Initialized shared mem obj
Shared mem obj already created
Recieved shm fd: -1
Ptr created with value 0xb5798000

Shared memory has: bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
Shared memory has: bbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
usuario@ubuntu-20:~/Escritorio/Lab20S$
```

¿Qué diferencia hay entre realizar comunicación usando memoria compartida en lugar de usando un archivo de texto común y corriente?

La memoria compartida es usada por programas ejecutados con diferentes procesadores y evita copias redundantes. Esto hace que ambos se puedan comunicar entre si sin tener que duplicar datos o tener mas instancias. En un txt se tendría información de memoria duplicada.

¿Por qué no se debe usar el file descriptor de la memoria compartida producido por otra instancia para realizar el mmap?

Porque para cada instancia los file descriptors son diferentes. Esto ocasiona que no se refiera al mismo espacio en memoria si el file descriptor se produce por otra instancia.

¿Es posible enviar el output de un programa ejecutado con exec a otro proceso por medio de un pipe? Investigue y explique cómo funciona este mecanismo en la terminal (e.g., la ejecución de ls | less).

Si es posible. Si ambos procesos que se desean comunicar tienen el mismo proceso padre se pueden crear cadenas de información de una o dos vías. Con ls / les, ls devuelve como standard output un listado de elementos en el directorio y les lo recibe como input y permite verlo línea por línea.

¿Cómo puede asegurarse de que ya se ha abierto un espacio de memoria compartida con un nombre determinado? Investigue y explique errno.

Esto se se puede llevar a cabo una operación sobre el espacio. Si este no existe da un error. Si ya existe se espera -1. Errno es el número de error que se utiliza para denotar un error y las condiciones en las que se encontraba el proceso al tener ese error. Puede ser domain error, range error o ilegal multi-byte carácter sequence.

¿Qué pasa si se ejecuta shm unlink cuando hay procesos que todavía están usando la memoria compartida?

Se podría seguir usando el espacio ya que shm\_unlink solo hace que la región ya no pueda ser mapeada.

¿Cómo puede referirse al contenido de un espacio en memoria al que apunta un puntero? Observe que su programa deberá tener alguna forma de saber hasta dónde ha escrito su otra instancia en la memoria compartida para no escribir sobre ello.

El puntero se inicializa y luego se le puede asignar un valor que representa el contenido de la memoria a donde apunta. Luego se puede referir a la dirección del puntero para referirse al contenido.

Imagine que una ejecución de su programa sufre un error que termina la ejecución prematuramente, dejando el espacio de memoria compartida abierto y provocando que nuevas ejecuciones se queden esperando el file descriptor del espacio de memoria compartida. ¿Cómo puede liberar el espacio de memoria compartida “manualmente”?

Munmap es una llamada al sistema que elimina asignaciones para un rango de direcciones especificadas y hace que otras referencias generen referencias no válidas. Si se tiene éxito, munmap devuelve 0 o -1 si falla.

Observe que el programa que ejecute dos instancias de ipc.c debe cuidar que una instancia no termine mucho antes que la otra para evitar que ambas instancias abran y cierren su propio espacio de memoria compartida. ¿Aproximadamente cuánto tiempo toma la realización de un fork()? Investigue y aplique usleep.

Depende de la capacidad de la computadora que se use, pero generalmente toma milisegundos. Por esto, un usleep de 1500 es mas que suficiente para que las dos instancias no se crucen.