

Lab 2

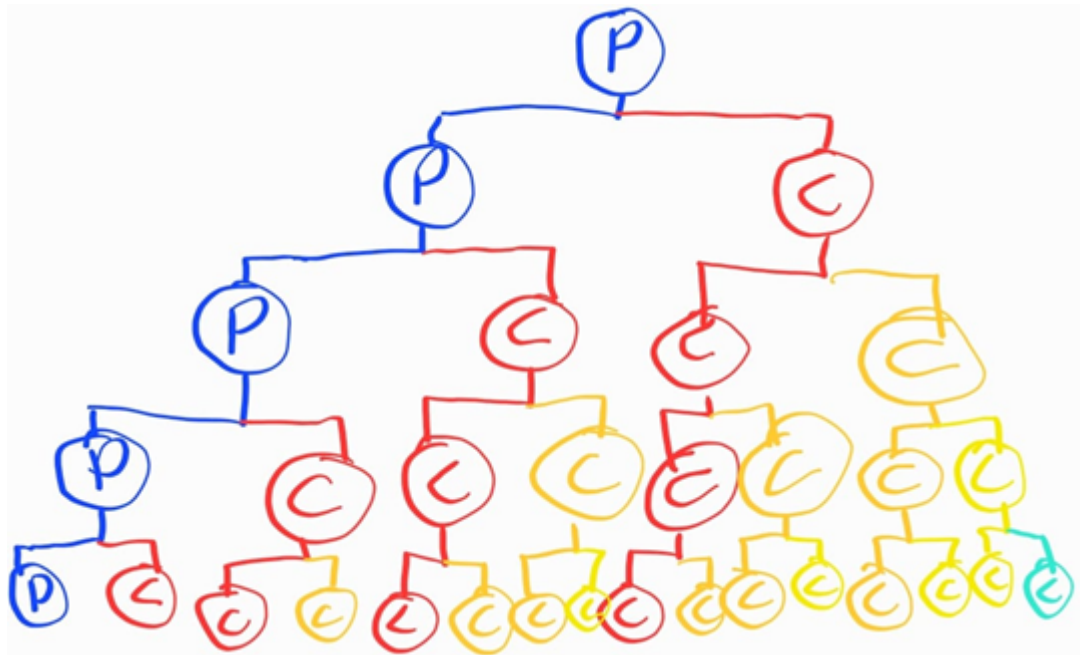
## 1. Ejercicio 1

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

Para ambos programas se crean 16 procesos.

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

Hay tantos procesos debido a que cada uno de los fork crea 1 proceso extra, uno padre y uno hijo. De manera que al momento de generar cada proceso lo que pasa es:



## 2. Ejercicio 2

Toma de tiempos

PRIMER PROGRAMA

```
diego@diego-VirtualBox: ~/s_o
File Edit View Search Terminal Help
diego@diego-VirtualBox:~/s_o$ gcc ex2_1.c -o ex2_1
diego@diego-VirtualBox:~/s_o$ ./ex2_1
7945.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_1
7066.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_1
7364.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_1
7145.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_1
7178.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_1
7180.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_1
7120.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_1
7152.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_1
7446.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_1
7360.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_1
7118.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_1
7240.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_1
7136.000000

diego@diego-VirtualBox:~/s_o$ ./ex2_1
7078.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_1
7113.000000
```

## SEGUNDO PROGRAMA

```
diego@diego-VirtualBox:~/s_o$ ./ex2_2
El resultado del proceso es: 158.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_2
El resultado del proceso es: 69.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_2
El resultado del proceso es: 172.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_2
El resultado del proceso es: 163.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_2
El resultado del proceso es: 177.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_2
El resultado del proceso es: 158.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_2
El resultado del proceso es: 226.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_2
El resultado del proceso es: 162.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_2
El resultado del proceso es: 172.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_2
El resultado del proceso es: 163.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_2
El resultado del proceso es: 251.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_2
El resultado del proceso es: 329.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_2
El resultado del proceso es: 135.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_2
El resultado del proceso es: 164.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_2
El resultado del proceso es: 169.000000
diego@diego-VirtualBox:~/s_o$ ./ex2_2
El resultado del proceso es: 161.000000
```

Número de corrida	Tiempo	Tiempo
01	7945	158
02	7066	69
03	7364	172
04	7145	163
05	7178	177
06	7180	158
07	7120	226

08	7152	162
09	7446	172
10	7360	163
11	7118	251
12	7240	329
13	7136	135
14	7078	164
15	7113	169

- a. ¿En general cuál programa toma los tiempos más largos?

El programa que toma los tiempos más largos es el concurrente, el segundo programa, y suele ser bastante más rápido.

- b. ¿Qué causa la diferencia de tiempos, o por qué se tarda más el que se tarda más?

El que se tarda más es el no concurrente, y es debido a que al no utilizar varios procesos, un solo proceso tiene que realizar los 3 for, mientras que en el otro programa, 3 procesos distintos se encargan de realizar los for's y lo realizan casi al mismo tiempo.

### 3. Ejercicio 3

- a. Investigación de los cambios de contexto: Los cambios de contexto son interrupciones de un proceso para hacer otro.

- Voluntarios: Es aquel en el que el proceso realiza una llamada para esperar por un evento.
- Involuntarios: Es aquel en el que el sistema operativo le quita la UCP al proceso.

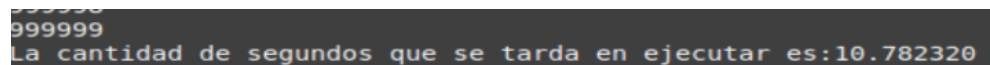
- b. ¿Qué tipo de cambios de contexto incrementa notablemente en cada caso, y por qué?

- Xorg: En el caso de xorg aparece un incremento de cambios de contexto voluntarios en la tarea realizada debido a que no es el único programa que se encarga de la interfaz gráfica, por ende, entra

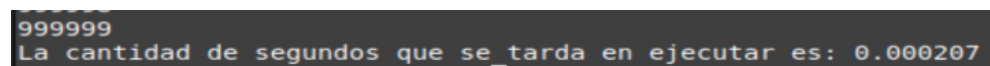
fácilmente en estado de bloqueo y por ello es que hace los switches cada vez que se escribe en la terminal. Este es el que se encarga de desplegar la interfaz gráfica, por ende mientras que no haya nada que sea un cambio. Otra razón por la cuál es que en el caso de Xorg es un contexto voluntario principalmente es debido a que se realiza un input por ende libera esos procesos para pasar a un estado de ready esperando el input.

- ii. Pidstat: Se mantiene constante durante toda la ejecución, y es un cambio involuntario el que realiza debido a que a pesar de que sigue en su intervalo de ejecución mientras estamos viendo todos los procesos siendo ejecutados, los demás procesos le exigen que devuelva el CPU.
- iii. CSD: Varía dependiendo de cómo es que cambia la interfaz gráfica a través del tiempo ya que puede hacer cambios de contexto voluntarios según como vaya cambiando la GUI. Este csd es el responsable de que cambien las decoraciones de las ventanas como tal y la GUI, se debe de destacar el hecho de que sí se mantiene constante la vista de la GUI dicho proceso no aparecerá. Además, igual que en el caso del Xorg pues se ejecuta un cambio de contexto voluntario dado que es una GUI.

El primer programa se tarda 10.782320 segundos en ejecutar y terminar todo.

A terminal window with a dark background. The first line shows a series of '9's. The second line displays the text 'La cantidad de segundos que se tarda en ejecutar es:10.782320'.

El segundo programa se tarda 0.000207 segundos en ejecutar y terminar todo.

A terminal window with a dark background. The first line shows a series of '9's. The second line displays the text 'La cantidad de segundos que se tarda en ejecutar es: 0.000207'.

Se debe de destacar que puede parecer que se tarden más, debido a los prints que se realizan en pantalla, pero realmente en ejecución esos son los tiempos.

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

Pues hay una gran diferencia entre las cantidades de cambios de contexto entre ambos programas debido a que en el caso del primer programa hay una gran cantidad de cambios de contexto de un solo proceso y son cambios de contexto involuntarios, mientras que en el segundo programa con fork's hay cambios de contexto tanto voluntarios como involuntarios, así mismo, no son

números tan grandes, y son 3 procesos diferentes, seguramente son debido a los fork's. Por ende se realiza un cambio de contexto mayor debido a que son más procesos.

Primer programa sin fork's:

Average:	1000	11607	1.00	0.00	cpptools-srv
Average:	1000	12144	0.08	0.25	pidstat
Average:	1000	12145	4.24	101166.61	ex3_1

Segundo programa, con fork's:

Average:	1000	12188	0.49	0.00	ex3_2
Average:	1000	12188	3605.42	8293.10	ex3_2
Average:	1000	12188	3129.06	14176.35	ex3_2
Average:	1000	12188	3827.09	3830.05	ex3_2

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

El cambio de contexto voluntario ocurre cuando una tarea se bloquea porque solicita un recurso que actualmente no está disponible.

En el primer programa esto ocurre casi en ningún momento porque no hay suficientes procesos que soliciten memoria o el recurso que pide. Ya que en el primer programa solo hay un proceso realizando el for.

En el segundo programa ocurren los cambios de contexto voluntarios en bastedad debido a que pues varios procesos hacen solicitudes de recursos iguales, y como son varios hilos pues suelen estar ocupados por otros procesos entonces se realiza un estado de bloqueo y se hace un cambio de proceso voluntario.

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

El cambio de contexto involuntario es cuando un proceso que está en su periodo de tiempo es obligado a liberar el procesador.

En el primer programa esto es debido a que hay otros procesos de mayor prioridad en el procesador que obligan al proceso a liberar su uso. Lo mismo sucede con el segundo programa, con la diferencia que en este caso no es solo un proceso sino que son varios los que liberan.

- f. ¿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?

Muestra cuatro procesos porque son 3 hilos que realizan el conteo, el hijo, nieto y bisnieto, así mismo uno de los procesos reporta 0 porque es el padre

que está esperando a que los procesos descendientes terminen de realizar su trabajo.

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

El efecto percibido es que en el caso de que sea un fork, varios procesos se estarán peleando por los mismos recursos, llevando a cabo cambios de contexto voluntarios.

Mientras que en el caso de los procesos involuntarios son más frecuentes debido a que siempre se darán dado que siempre aparecerá un proceso que tenga mayores permisos y que exija que los recursos se le sean liberados.

#### 4. Ejercicio 4

- a. ¿Qué significa la Z y a qué se debe?

diego@diego-VirtualBox: ~

FileEditViewSearchTerminalHelp

4	S	1000	14983	12311	0	80	0 - 604397 do_pol ?	00:00:00	Isolated Web Co
0	S	1000	15284	1138	0	80	0 - 97835 do_pol ?	00:00:00	gvfsd-network
0	S	1000	15302	1138	0	80	0 - 79836 do_pol ?	00:00:00	gvfsd-dnssd
4	S	1000	15445	12311	2	80	0 - 814422 do_pol ?	00:04:38	Isolated Web Co
0	S	1000	15710	921	0	80	0 - 1237780 futex_ ?	00:00:04	cpptools-srv
0	S	1000	15735	921	0	80	0 - 1237813 futex_ ?	00:00:01	cpptools-srv
1	I	0	18629	2	0	80	0 - 0 - ?	00:00:00	kworker/1:0-events
1	I	0	18670	2	0	80	0 - 0 - ?	00:00:00	kworker/3:2-events
1	I	0	18812	2	0	80	0 - 0 - ?	00:00:01	kworker/2:1-events
1	I	0	18866	2	0	80	0 - 0 - ?	00:00:00	kworker/3:0
4	S	1000	18882	12311	0	80	0 - 611145 do_pol ?	00:00:02	Isolated Servic
4	S	1000	18911	12311	0	80	0 - 599906 do_pol ?	00:00:00	Web Content
1	I	0	18930	2	0	80	0 - 0 - ?	00:00:00	kworker/0:2-events
4	S	1000	18931	12311	0	80	0 - 599906 do_pol ?	00:00:00	Web Content
1	I	0	18949	2	0	80	0 - 0 - ?	00:00:00	kworker/1:1-events
1	I	0	19199	2	0	80	0 - 0 - ?	00:00:00	kworker/0:1-events
1	I	0	19276	2	0	80	0 - 0 - ?	00:00:00	kworker/2:0
1	I	0	19317	2	0	80	0 - 0 - ?	00:00:00	kworker/u8:1-flush-8:0
1	I	0	19333	2	0	80	0 - 0 - ?	00:00:00	kworker/u8:3-events_unbound
4	S	1000	19414	12311	0	80	0 - 599906 do_pol ?	00:00:00	Web Content
1	R	0	19452	2	0	80	0 - 0 - ?	00:00:00	kworker/u8:0-events_unbound
0	S	1000	19511	13051	0	80	0 - 3386 do_wai pts/1	00:00:00	bash
0	S	1000	19538	921	0	80	0 - 1237780 futex_ ?	00:00:00	cpptools-srv
0	S	1000	19576	14400	0	80	0 - 9534619 ep_pol ?	00:00:00	code
1	I	0	19616	2	0	80	0 - 0 - ?	00:00:00	kworker/2:2-events
0	R	1000	19644	13076	99	80	0 - 660 - pts/0	00:00:18	ex4 1
1	Z	1000	19645	19644	0	80	0 - 0 - pts/0	00:00:00	ex4 1 <defunct>
4	R	1000	19649	19511	0	80	0 - 3800 - pts/1	00:00:00	ps

El comando ps sirve para monitorear los procesos que se realizan en la computadora. El status z indica que es un proceso zombie o aquel que no responde, que es aquel proceso que ya termino de ejecutarse, sin embargo, aun tiene entrada en el PCB dado que puede ser que su proceso padre haya terminado antes que el y no lo haya podido borrar de dicha tabla.

- b. Los números del padre y del hijo son: 22963 y 22964 respectivamente.

0 R	1000	21962	13076	90	80	0 - 660 -	pts/0	00:00:22	ex4 2
1 Z	1000	21963	21962	58	80	0 - 0 -	pts/0	00:00:14	ex4 2 <defunct>



- c. Esto sucede cuando se mata el proceso padre antes de que se termine de ejecutar el proceso hijo.

```

3999973 3999973
3999974 3999974
3999975 3999975
3999976 3999976
3999977 3999977
3999978 3999978
3999979 3999979
3999980 3999980
3999981 3999981
3999982 3999982
3999983 3999983
3999984 3999984
3999985 3999985
3999986 3999986
3999987 3999987
3999988 3999988
3999989 3999989
3999990 3999990
3999991 3999991
3999992 3999992
3999993 3999993
3999994 3999994
3999995 3999995
3999996 3999996
3999997 3999997
3999998 3999998
3999999 Killed

```

```

0 S 1000 15302 1138 0 80 0 - 79836 do_pol ? 00:00:00 gvfsd-dnssd
4 S 1000 15445 12311 3 80 0 - 831602 do_pol ? 00:07:01 Isolated Web Co
0 S 1000 15710 921 0 80 0 - 1237780 futex_ ? 00:00:04 cpptools-srv
0 S 1000 15735 921 0 80 0 - 1237813 futex_ ? 00:00:01 cpptools-srv
1 I 0 18670 2 0 80 0 - 0 - ? 00:00:01 kworker/3:2-events
1 I 0 18866 2 0 80 0 - 0 - ? 00:00:00 kworker/3:0
4 S 1000 18882 12311 0 80 0 - 611401 do_pol ? 00:00:06 Isolated Servic
1 I 0 18930 2 0 80 0 - 0 - ? 00:00:01 kworker/0:2-events
0 S 1000 19511 13051 0 80 0 - 3386 do_wai pts/1 00:00:00 bash
0 S 1000 19538 921 0 80 0 - 1237780 futex_ ? 00:00:01 cpptools-srv
0 S 1000 19576 14400 0 80 0 - 9534619 ep_pol ? 00:00:00 code
1 I 0 19616 2 0 80 0 - 0 - ? 00:00:00 kworker/2:2-events
1 I 0 19664 2 0 80 0 - 0 - ? 00:00:05 kworker/u8:1-events_unbound
1 I 0 20982 2 0 80 0 - 0 - ? 00:00:07 kworker/u8:0-events_unbound
1 I 0 20989 2 0 80 0 - 0 - ? 00:00:00 kworker/0:0-events
1 I 0 21116 2 0 80 0 - 0 - ? 00:00:00 kworker/1:1-events
4 S 0 21154 1 0 80 0 - 98524 - ? 00:00:00 fwupd
1 I 0 21426 2 0 80 0 - 0 - ? 00:00:00 kworker/1:0
1 I 0 21506 2 1 80 0 - 0 - ? 00:00:10 kworker/u8:3-events_unbound
1 I 0 21684 2 0 80 0 - 0 - ? 00:00:00 kworker/2:0-events
4 S 1000 21779 12311 0 80 0 - 599905 do_pol ? 00:00:00 Web Content
4 S 1000 21851 12311 0 80 0 - 599906 do_pol ? 00:00:00 Web Content
4 S 1000 21854 12311 0 80 0 - 599906 do_pol ? 00:00:00 Web Content
0 S 1000 21920 921 0 80 0 - 1237780 futex_ ? 00:00:00 cpptools-srv
1 I 0 21964 2 1 80 0 - 0 - ? 00:00:06 kworker/u8:2-events_unbound
0 R 1000 22056 13076 84 80 0 - 660 - pts/0 00:00:07 ex4_2
1 R 1000 22057 22056 59 80 0 - 693 - pts/0 00:00:05 ex4_2
4 R 1000 22059 19511 0 80 0 - 3800 - pts/1 00:00:00 ps
diego@diego-VirtualBox:~$ kill -9 22055

```



¿Qué sucede en la ventana donde ejecutó su programa? Lo que sucede es que no aparece nada, en el caso de que se mate el proceso aparece un mensaje de asesinato.

- d. ¿Quién es el padre del proceso que quedó huérfano? El padre del proceso que quedó huérfano es el proceso con id 921.

```

1 R 1000 22199 921 56 80 0 - 693 - pts/0 00:00:08 ex4_2
4 R 1000 22202 19511 0 80 0 - 3800 - pts/1 00:00:00 ps

```

Es decir el systemd, el systemd es básicamente el proceso init actualizado así que realmente quien termina siendo el padrastró del proceso es el proceso inicial.

```

4 S 0 916 759 0 80 0 - 41137 - ? 00:00:00 lightdm
4 S 1000 921 1 0 80 0 - 4321 ep_pol ? 00:00:00 systemd
5 S 1000 922 921 0 80 0 - 26393 - ? 00:00:00 (sd-pam)

```

## 5. Ejercicio 5

- a. Pipes: Es la conexión entre dos procesos.
- Anónimos: es una pipe que transmite información comúnmente entre padre e hijo, unidireccional.
  - Nombrados: es una pipe que puede ser bidireccional y no necesariamente entre padre e hijo.

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

La memoria compartida al permitir que puedan leer y escribir dicha información desde la memoria compartida, esto brinda mucha velocidad ya que no necesita de escribir y leer archivos. Mientras que en el caso de los archivos de texto implica que debe de leer y escribir desde el disco de manera que es más lento además de que permite una persistencia.

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

No debe de realizarse esto debido a que cada proceso que accede a la memoria posee un espacio propio de direcciones, por ende puede que el file descriptor no sea válido en el proceso. Esto es debido al identificador único que se genera, es por ello que cuando se llama la función mmap se debe dar el file descriptor del proceso actual no otro ya que esto generaría error.

- d. ¿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`).

Sí es posible enviar el output mediante una pipe, se podría crear una pipe antes de la llamada a `exec` y luego redirigir el output de manera que se podrían leer los datos del extremo de lectura.

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

Para asegurarse que ya se ha abierto se puede usar la función `shm_open` para abrir el objeto de memoria compartida. En caso de que esté abierto se abre y si no entonces se crea, además, se puede verificar lo que haya hecho la función mediante un `errno`. Así mismo el `errno` es una variable global de manera que esta variable mediante un entero representa un tipo de error.

- f. ¿Qué pasa si se ejecuta `shm_unlink` cuando hay procesos que todavía están usando la memoria compartida?

Dado que se utiliza para eliminar una región de memoria compartida nombrada, cuando se ejecuta mientras se están usando la memoria compartida entonces la región eliminará los archivos, de manera que ya no estará disponible para la prosiguiente ejecución, aunque los que en ese momento la estén usando sí pueden acceder a dicha memoria hasta que cierren la conexión.

- g. ¿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.

- h. 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”?

Mediante la función `shm_unlink` en la línea de comandos, dado que permite que se elimine el objeto de memoria compartida asociado a un particular nombre.

- i. 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`.