

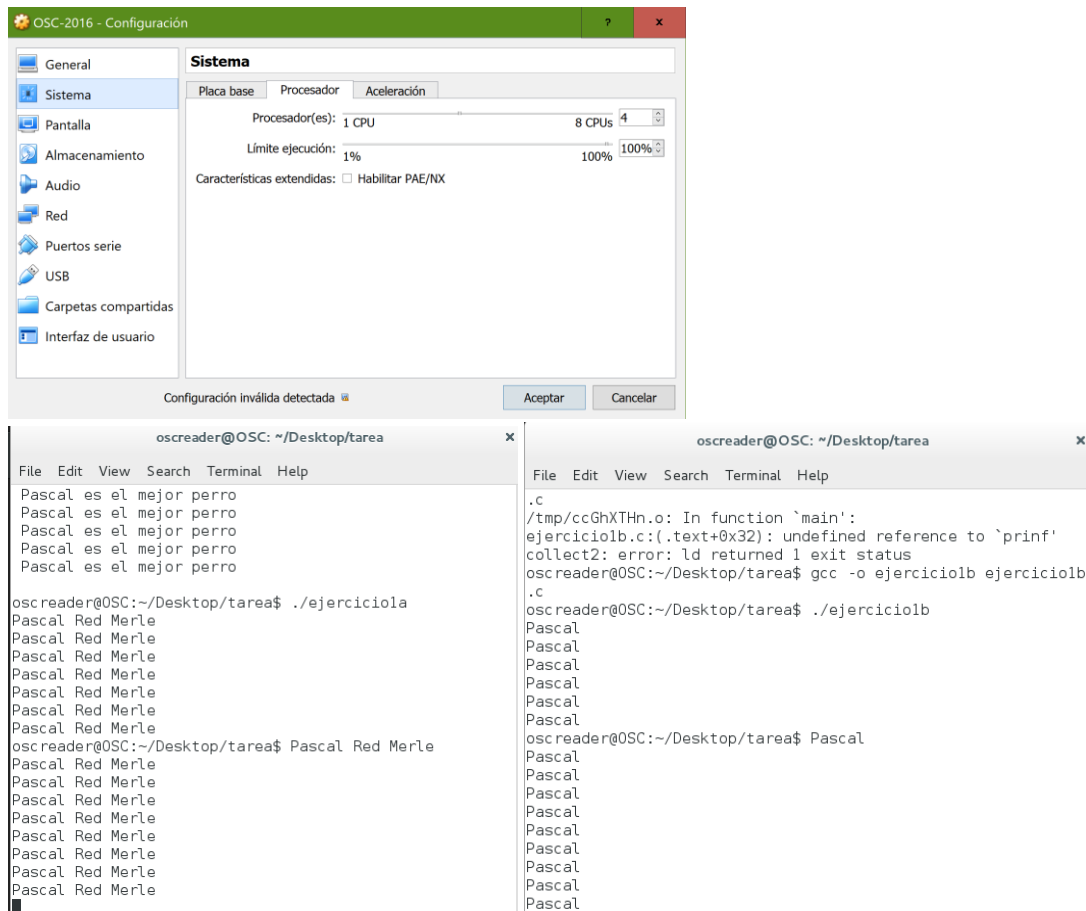
Universidad del Valle de Guatemala

Sistemas Operativos

Ana lucia Diaz Leppe #151378

Laboratorio 3

Ejercicio 1:



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

En el primer programa se utiliza 2 elevado a la n, es decir 2^4 por lo tanto se imprime 16 veces. En el segundo caso se ejecutan 2^4-1 . Si sumamos

todos los niveles del árbol anterior para $i = 0$ a $n-1$, obtenemos $2n - 1$. Por lo tanto, habrá $2n - 1$ procesos secundarios.

- **Explique cómo se crea cada proceso y qué sucede después.**

Un sistema fork llama procesos de generación como hojas de un árbol binario de crecimiento. Como llamamos el fork cuatro veces se reprodujo $2^4 = 16$ procesos. Todos estos 16 procesos forman la hoja de hijos del árbol binario. Por lo tanto, Pascal Red Merle se imprimió esa cantidad de veces. En el segundo caso, se ejecutaron $2^4 - 1$ debido a que: se crea un hijo en el proceso que se genera el primer fork, esto da en consecuencia 2 diferentes hijos del proceso creados por el segundo fork y esto pasara consecutivamente. Por lo tanto tenemos el caso de $2^4 - 1$.

Ejercicio 2

```
El numero de segundos en trabajar fue 0.007623
oscreader@OSC:~/Desktop/tarea$ ./ejercicio2

El numero de segundos en trabajar fue 0.007739
oscreader@OSC:~/Desktop/tarea$ ./ejercicio2

El numero de segundos en trabajar fue 0.006937
oscreader@OSC:~/Desktop/tarea$ ./ejercicio2

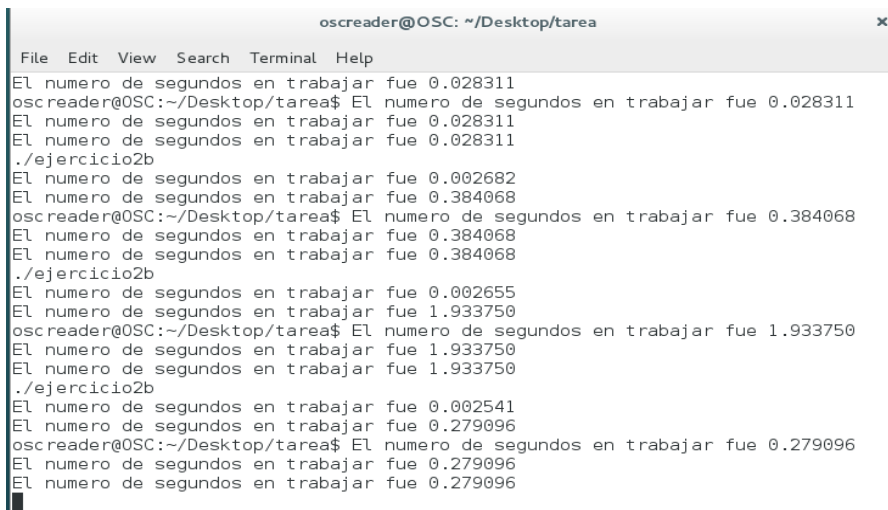
El numero de segundos en trabajar fue 0.007007
oscreader@OSC:~/Desktop/tarea$ ./ejercicio2

El numero de segundos en trabajar fue 0.008116
oscreader@OSC:~/Desktop/tarea$ ./ejercicio2

El numero de segundos en trabajar fue 0.009119
oscreader@OSC:~/Desktop/tarea$ ./ejercicio2

El numero de segundos en trabajar fue 0.035760
oscreader@OSC:~/Desktop/tarea$ ./ejercicio2

El numero de segundos en trabajar fue 0.007025
oscreader@OSC:~/Desktop/tarea$
```



```
oscreader@OSC: ~/Desktop/tarea
File Edit View Search Terminal Help
El numero de segundos en trabajar fue 0.028311
oscreader@OSC:~/Desktop/tarea$ El numero de segundos en trabajar fue 0.028311
El numero de segundos en trabajar fue 0.028311
El numero de segundos en trabajar fue 0.028311
./ejercicio2b
El numero de segundos en trabajar fue 0.002682
El numero de segundos en trabajar fue 0.384068
oscreader@OSC:~/Desktop/tarea$ El numero de segundos en trabajar fue 0.384068
El numero de segundos en trabajar fue 0.384068
El numero de segundos en trabajar fue 0.384068
./ejercicio2b
El numero de segundos en trabajar fue 0.002655
El numero de segundos en trabajar fue 1.933750
oscreader@OSC:~/Desktop/tarea$ El numero de segundos en trabajar fue 1.933750
El numero de segundos en trabajar fue 1.933750
El numero de segundos en trabajar fue 1.933750
./ejercicio2b
El numero de segundos en trabajar fue 0.002541
El numero de segundos en trabajar fue 0.279096
oscreader@OSC:~/Desktop/tarea$ El numero de segundos en trabajar fue 0.279096
El numero de segundos en trabajar fue 0.279096
El numero de segundos en trabajar fue 0.279096
```

- **¿Cuál, en general, toma tiempos más largos?**

Se tarda mas el de los procesos, es decir la opción B.

- **¿Qué causa la diferencia de tiempo, o por qué se tarda más el que se tarda más?**
El proceso padre al crear hijos puede darse 2 posibilidades en términos de ejecución. El proceso padre se ejecuta con los procesos hijos de manera concurrente. El proceso padre espera a que todos los procesos hijos terminen, Es debido a esto que se puede llegar a tomar mas tiempo. Además utilizamos un wait(NULL) el cual espera hasta que el proceso del hijo ya haya terminado.

Ejercicio 3

The screenshot shows a terminal window with two panes. The left pane shows the output of the command `dpkg-query -f='${Package} ${Version} ${Architecture}\n'` for the package `sysstat`. The right pane shows the output of the `top` command, displaying system statistics and a list of running processes.

```

oscreader@OSC: ~/Desktop/tarea
File Edit View Search Terminal Help
upgraded.
Need to get 290 kB of archives.
After this operation, 1,390 kB of additional disk space will be used.
Get:1 http://ftp.us.debian.org/debian/ jessie/main sysstat i386 11.0.1-1 [290 kB]
Fetched 290 kB in 0s (355 kB/s)
Preconfiguring packages ...
Selecting previously unselected package sysstat.
(Reading database ... 154305 files and directories currently installed.)
Preparing to unpack .../sysstat_11.0.1-1_i386.deb ...
Unpacking sysstat (11.0.1-1) ...
Processing triggers for man-db (2.7.0.2-5) ...
Processing triggers for systemd (215-17+deb8u2) ...
Setting up sysstat (11.0.1-1) ...
Creating config file /etc/default/sysstat with new version
update-alternatives: using /usr/bin/sar.sysstat to provide /usr/bin/sar (sar) in auto mode
Processing triggers for systemd (215-17+deb8u2) ...
oscreader@OSC:~/Desktop/tarea$

oscreader@OSC: ~
File Edit View Search Terminal Help
04:50:12 PM 1000 3202 1.00 84.00 pidstat
04:50:12 PM UID PID cswch/s nvcswch/s Command
04:50:13 PM 0 3 0.99 0.00 ksoftirqd/0
04:50:13 PM 0 7 5.94 0.00 rcu_sched
04:50:13 PM 0 170 86.14 0.00 kworker/0:2
04:50:13 PM 0 776 57.43 95.05 Xorg
04:50:13 PM 0 947 0.99 0.00 kworker/3:3
04:50:13 PM 1000 1119 99.01 52.48 gnome-shell
04:50:13 PM 1000 1341 92.08 0.00 gnome-terminal-
04:50:13 PM 0 1816 0.99 0.00 kworker/2:2
04:50:13 PM 0 1845 0.99 0.00 kworker/1:0
04:50:13 PM 0 3178 0.99 0.00 kworker/0:0
04:50:13 PM 1000 3202 0.99 83.17 pidstat

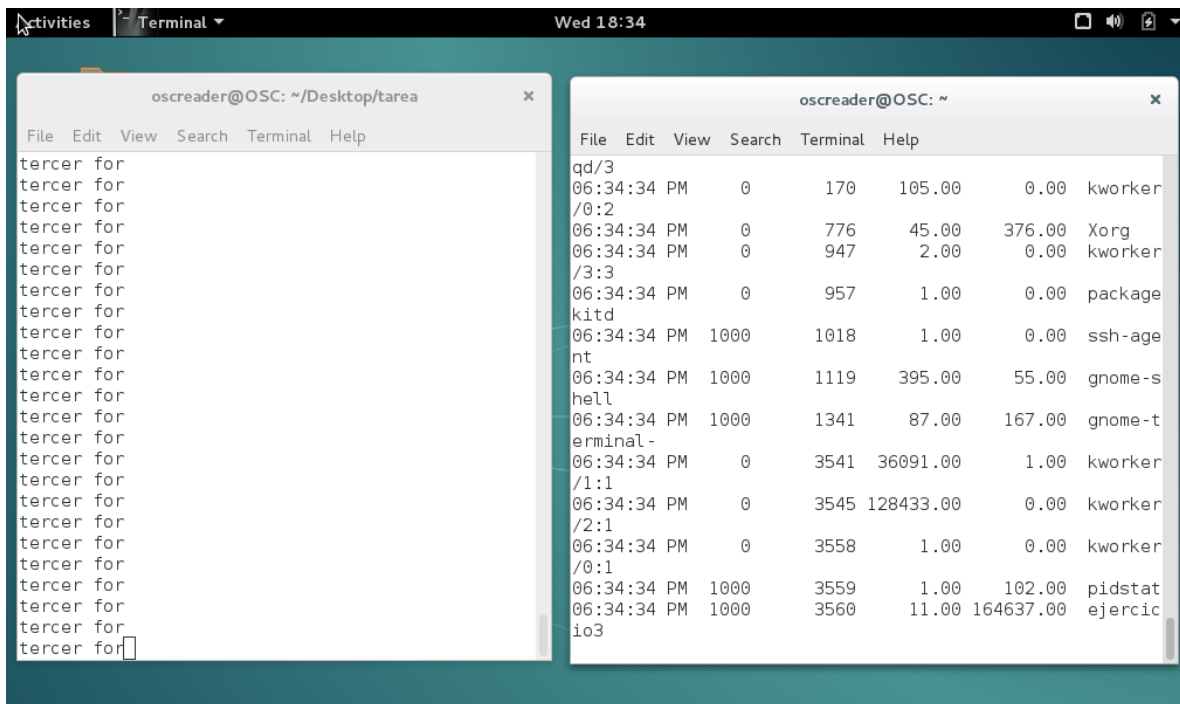
```

- **¿Qué tipo de cambios de contexto incrementa notablemente en cada caso, y por qué?**
Gnome-shell al abrir una nueva pantalla.
Xorg puede ser afectado por sus dispositivos de entrada como la aceleración del ratón, botones de entrada o teclados.


```
oscreader@OSC: ~/Desktop/tarea
File Edit View Search Terminal Help
tercer for
tercer for
tercer for
tercer for
tercer for
tercer for
tercer for
tercer for
tercer for
tercer for
tercer for
tercer for
tercer for
tercer for
tercer for
tercer for
tercer for
tercer for
tercer for
tercer for

oscreader@OSC: ~
File Edit View Search Terminal Help
Average: 1000 3543 0.04 0.08 pidstat
Average: 1000 3544 12.80 187557.63 ejercici
Average: 0 3545 49856.56 0.00 kworker/
2:1
Average: 0 3546 0.15 0.00 kworker/
0:1
oscreader@OSC:~$ pidstat -w 28 1
Linux 3.16.0-4-686-pae (OSC) 02/13/2019 _i686_ (
4 CPU)
06:25:45 PM UID PID cswch/s nvcswch/s Command
06:26:13 PM 0 1 0.14 0.00 systemd
06:26:13 PM 0 3 1.89 0.00 ksoftirq
d/0
06:26:13 PM 0 6 129.28 0.25 kworker/
u8:0
06:26:13 PM 0 7 103.50 0.00 rcu_sche
d
06:26:13 PM 0 9 0.04 0.00 migratio
n/0
06:26:13 PM 0 10 0.25 0.00 watchdog
/0
06:26:13 PM 0 11 0.25 0.00 watchdog
```

- **¿Qué diferencia hay en el número y tipo de cambios de contexto de entre programas?**
Uno esta mostrando una mayor cantidad de cambios que el otro. Esto se puede deber al hecho que uno lo esta dividiendo en procesos de padres e hijos mientras que el otro no. Otros de los cambios a resaltar es que uno tiene muchos mas datos que el otro, y como tuvo una mayor cantidad de segundos que el otro, tardo más tiempo en ejecutar.
- **¿A qué puede atribuir los cambios de contexto voluntarios realizados por sus programas?**
El número total de contextos voluntarios cambia la tarea hecha por segundo. Se produce un cambio de contexto voluntario cuando una tarea se bloquea porque requiere un recurso que no esta disponible.
- **¿A qué puede atribuir los cambios de contexto involuntarios realizados por sus programas?**
Al igual que el anterior, el numero total de contextos no voluntarios cambia la tarea hecha por segundo. Un cambio de contexto involuntario toma lugar cuando una tarea se ejecuta durante la duración de su tiempo. Corta y luego se ve obligado a renunciar al procesador.
- **¿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?**
Porque el resultado del ultimo proceso el cual solo muestra el proceso inicializando.



- **¿Qué efecto percibe sobre el número de cambios de contexto de cada tipo?**
Estos nunca terminan y no se mantienen constantes durante su ejecución, y bajan cuando termina la ejecución.

Ejercicio 4

- **¿Qué significa la Z y a qué se debe?**
Es un proceso Zombie que termina pero no es cosechado por su padre.

File	Edit	View	Search	Terminal	Help
16241726	16241727	16241728	16241729	16241730	16241731
16241732	16241733	16241734	16241735	16241736	16241737
16241738	16241739	16241740	16241741	16241742	16241743
16241744	16241745	16241746	16241747	16241748	16241749
16241750	16241751	16241752	16241753	16241754	16241755
16241756	16241757	16241758	16241759	16241760	16241761
16241762	16241763	16241764	16241765	16241766	16241767
16241768	16241769	16241770	16241771	16241772	16241773
16241774	16241775	16241776	16241777	16241778	16241779
16241780	16241781	16241782	16241783	16241784	16241785
16241786	16241787	16241788	16241789	16241790	16241791
16241792	16241793	16241794	16241795	16241796	16241797
16241798	16241799	16241800	16241801	16241802	16241803
16241804	16241805	16241806	16241807	16241808	16241809
16241810	16241811	16241812	16241813	16241814	16241815
16241816	16241817	16241818	16241819	16241820	16241821
16241822	16241823	16241824	16241825	16241826	16241827
16241828	16241829	16241830	16241831	16241832	16241833
16241834	16241835	16241836	16241837	16241838	16241839
16241840	16241841	16241842	16241843	16241844	16241845
16241846	16241847	16241848	16241849	16241850	16241851
16241852	16241853	16241854	16241855	16241856	16241857
16241858	16241859	16241860	16241861	16241862	16241863
16241864	16241865	16241866	16241867	16241868	16241869
16241870	16241871	16241872	16241873	16241874	16241875
16241876	16241877	16241878	16241879	16241880	16241881
16241882	16241883	16241884	16241885	16241886	16241887
16241888	16241889	16241890	16241891	16241892	16241893
16241894	16241895	16241896	16241897	16241898	16241899
16241900	16241901	16241902	16241903	16241904	16241905
16241906	16241907	16241908	16241909	16241910	16241911
16241912	16241913	16241914	16241915	16241916	16241917
16241918	16241919	16241920	16241921	16241922	16241923
16241924	16241925	16241926	16241927	16241928	16241929
16241930	16241931	16241932	16241933	16241934	16241935
16241936	16241937	16241938	16241939	16241940	16241941
16241942	16241943	16241944	16241945	16241946	16241947
16241948	16241949	16241950	16241951	16241952	16241953
16241954	16241955	16241956	16241957	16241958	16241959
16241960	16241961	16241962	16241963	16241964	16241965
16241966	16241967	16241968	16241969	16241970	16241971
16241972	16241973	16241974	16241975	16241976	16241977
16241978	16241979	16241980	16241981	16241982	16241983
16241984	16241985	16241986	16241987	16241988	16241989
16241990	16241991	16241992	16241993	16241994	16241995
16241996	16241997	16241998	16241999	16242000	16242001

- **¿Qué sucede en la ventana donde ejecutó su programa?**
Se elimino el padre por lo que el huerfano fue adoptado por el init process.

- **¿Quién es el padre del proceso que quedó huérfano?**

Es adoptado por el init process como un padre.

Ejercicio 5

```

hola
(Parent)b:      hola
shared memory has: hola
Created new shared mem obj: hola
(child)a:      hola
(Parent)b: hola
Process: Shared Memory: Wrote 1023 bytes
Process: Shared Memory: Wrote 1023 bytes
Process: Shared Memory: Wrote 1023 bytes
Process: Shared Memory: Wrote 1023 bytes
Process: Shared Memory: Wrote 1023 bytes
Process: Wrote 5 times
Process: Complete
(parent)a: hola

```

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

La comunicación entre procesos a través de la memoria compartida es un concepto donde uno o mas procesos pueden acceder a la memoria común. Y la comunicación se realiza a través de esta memoria compartida donde los cambios realizados por un proceso pueden ver por otro proceso. El servidor a diferencia, leerá desde un archivo de entrada. El servidor escribe estos datos en un mensaje y existirá un cliente que leerá los datos del canal IPC y nuevamente requiere que los datos se copien desde el búfer IPC del kernel al búfer del cliente.

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

Ya que asigna el objeto utilizando el file descriptor con una llamada a la función del mmap. Este ayuda a identificar un archivo abierto dentro de un proceso mientras usa recursos de entrada/salida como sockets o tuberías de red. Esto además no se utiliza porque se mapeara la memoria compartida en un puntero.

- **¿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*).**
- *pipe* (|) se usa para pasar stdout a stdin de subshell. Pero, el comando *less* siempre debe tener un argumento de nombre de archivo como *less file1.txt*. Por lo tanto se usa *ls -l / root | less*. *Less* aceptara un un nombre de archivo , pero si no se da ningún archivo , se lee desde la entrada estándar.
Si es posible, esto se llegaría a realizar por medio de hacer un programa un “hijo” y otro como un “padre ”. El maestro crea N procesos secundarios, con un conducto desde la salida estándar de cada hijo al proceso maestro. Cada proceso hijo ejecuta un binario esclavo separado. Cada esclavo espera una señal antes de escribir en la el pipe, luego sale el padre y espera hasta que no haya mas pipe abiertas. Es decir, el hijo solo espera la señal luego escribe en la salida estándar y sale.
- **¿Cómo puede asegurarse de que ya se ha abierto un espacio de memoria compartida con un nombre determinado? Investigue y explique *errno*.**
Shm_open se basa en *stmpfs*, generalmente montado bajo */dev/shms*. Lo que hace *shm_open* es convertir el nombre del objeto en una ruta de archivo anteponiéndolo con el punto de montaje del sistema de archivos *tmpfs*. Mientras el objeto de shared memory no este desvinculado, estará visible en el sistema de archivos y todo lo que debe hacer es emitir un simple comando *ls*. Para saber que proceso tiene asignado los objetos de memoria el comando *ls*of servirá de guía.
Errno.h es un archivo de cabecera en la biblioteca estándar. En ella se definen las macros que presentan un informe de error a través de códigos de error. El cual establece mediante llamadas al sistema y algunas funciones de la biblioteca en caso de un error para indicar que algo salió mal.
- **¿Qué pasa si se ejecuta *shm_unlink* cuando hay procesos que todavía están usando la memoria compartida?**
Shm_unlik() eliminara el nombre del objeto de memoria compartida nombrado por la cadena señalada por su nombre. Si existe una o mas referencias al objeto de memoria compartida cuando el objeto esta desvinculado, el nombre se eliminara antes de que *shm_unlink()* regrese, pero la eliminación de los contenidos del objeto de memoria pospondrá hasta que todas las referencias abiertas y de mapas al objeto de memoria se hayan removido.
- **¿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 almacena la dirección de una variable. Por lo tanto, el espacio de memoria para un puntero depende del ancho de la dirección de esa computadora. Si tiene un procesador de 32 bits un puntero ocupara 32 bits o 4 bytes. Dado que un puntero es también un tipo de variable, también tiene su propia dirección donde la variable del puntero se guarda en la memoria.

- **Imagine que una ejecución de su programa sufre un error que termina la ejecución prematuramente, dejando el espacio de memoria compartido 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”?**

Una de las maneras usar ipcs para ver en donde se encuentra el shared memory y luego eliminarlo con ipcrm.

- **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.**

Aproximadamente menos de 15 segundos. Usleep se encarga de suspender la ejecución por intervalos de microsegundos. Es decir, se encarga de suspender la ejecución del hilo de llamada user microsegundos. Usleep puede alargarse ligeramente por cualquier actividad del sistema o por el tiempo dedicado a procesar la llamada . Retorna 0 si funciona y si tiene error retorna -1.