

Laboratorio II
Sistemas Operativos
Maria Jose Castro Lemus #181202

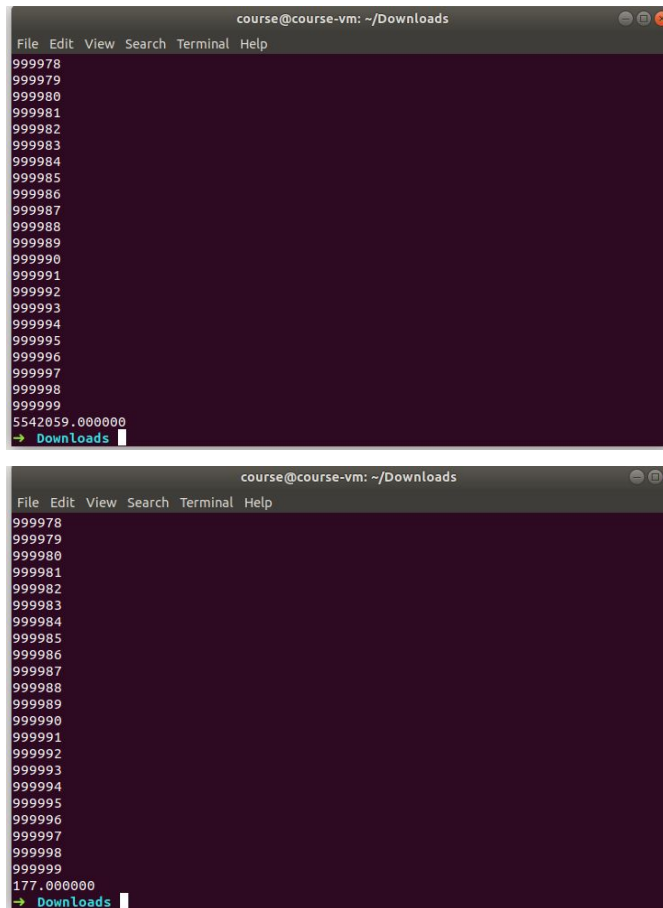
Ejercicio 1:

```
course@course-vm: ~/Downloads
File Edit View Search Terminal Help
gcc: fatal error: no input files
compilation terminated.
→ Downloads gcc -o ej1exe1A.c
gcc: fatal error: no input files
compilation terminated.
→ Downloads gcc -o ej1 exe1A.c
→ Downloads ./ej1
3750
3757
3759
3753
3754
3751
3755
3762
3756
3761
3758
3763
3766
3752
3767
3764
→ Downloads
```

```
course@course-vm: ~/Downloads
File Edit View Search Terminal Help
3763
3766
3752
3767
3764
→ Downloads gcc -o ej2 exe1B.c
→ Downloads ./ej2
3889
3892
3890
3896
3893
3900
3897
3902
3894
3898
3901
3891
3899
3895
3904
3905
→ Downloads
```

1. ¿Cuántos procesos se crean en cada uno de los programas?
 - a. El programa y los 4 forks consecutivos así como el 4 con 4 iteraciones, se crearon 16 procesos en total.
2. ¿Por qué hay tantos procesos en ambos programas cuando uno tiene cuatro llamadas fork() y el otro sólo tiene una?
 - a. Esto se debe a que el sol tiene una iteración de 4 veces, esto es lo mismo que ejecutar cuatro llamada si fork() como en el anterior.

Ejercicio 2:



```
course@course-vm: ~/Downloads
File Edit View Search Terminal Help
999978
999979
999980
999981
999982
999983
999984
999985
999986
999987
999988
999989
999990
999991
999992
999993
999994
999995
999996
999997
999998
999999
5542059.000000
→ Downloads

course@course-vm: ~/Downloads
File Edit View Search Terminal Help
999978
999979
999980
999981
999982
999983
999984
999985
999986
999987
999988
999989
999990
999991
999992
999993
999994
999995
999996
999997
999998
999999
177.000000
→ Downloads
```

1. ¿Cuál, en general, toma tiempos más largos?
 - a. El programa más lento es el sin forks
2. ¿Qué causa la diferencia de tiempo, o por qué se tarda más el que se tarda más?
 - a. Este se tarda mas porque o se ejecuta de forma concurrente, un proceso no tiene la capacidad de hacer a otro mientras se ejecuta. Ete tambien espera a sus procesos hijos para terminar su tarea.

Ejercicio 3:

The top screenshot shows the output of 'pidstat -w 1' at 02:23:22. The output is a table with columns: Time, UID, PID, cswch/s, nvcsch/s, and Command. The 'cat' process (PID 1000) is not yet running. The 'kworker' processes are running with low context switch rates.

The bottom screenshot shows the output of 'pidstat -w 1' at 02:29:07. The 'cat' process (PID 1000) is now running, and its context switch rate (cswch/s) has increased significantly to 0.99. The 'kworker' processes are still running, but their context switch rates have decreased.

1. ¿Qué tipo de cambios de contexto incrementa notablemente en cada caso, y por qué?
 - a. Al usar el teclado aumenta la candidata de cambio de contexto voluntario, ya que realizada una llamada al sistema y esto provoca una espera por un evento, en este caso leer lo que se escribió en el teclado. En el caso de los cambios de contexto no voluntarios aumentaron cuando se movió la interfaz, esto debe ser a que el sistema operativo removió el proceso del CPU y el uso de procesador se repartió.
2. ¿Qué diferencia hay en el número y tipo de cambios de contexto de entre programas?
 - a. Al comparar el programa con y sin forkes podemos ver los cambios de contexto voluntario incrementando de forma notable en el segundo, mientras que los involuntarios disminuyen.
3. ¿A qué puede atribuir los cambios de contexto voluntarios realizados por sus programas?
 - a. Los indicados por las llamadas a sistema `fork()` que implica esperar un evento. Esto quiere decir que entre más concurrencia, se incrementan los cambios de contexto.
4. ¿A qué puede atribuir los cambios de contexto involuntarios realizados por sus programas?

- a. Estos son los que el sistema debe realizar de forma obligatoria para desbloquear el proceso siguiente. Esto indica que deben esperar a que cada proceso termine para poder usar los recursos de CPU al proceso anterior y continuar con el resto .
5. ¿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?
 - a. En este caso el último no tiene que esperar a ningún otro proceso para ejecutarse y finalizar el programa de forma inmediata.
6. ¿Qué efecto percibe sobre el número de cambios de contexto de cada tipo?
 - a. Los cambios involuntarios son los que aumentan durante la ejecución y sin interrupciones como escribir o mover la interfaz.

Ejercicio 4:

```

course@course-vm: ~/Downloads
File Edit View Search Terminal Help
4 S 1000 2639 1 2 80 0 - 643820 poll_s tty1 00:01:00 firefox
4 S 1000 2718 2639 0 80 0 - 456208 poll_s tty1 00:00:05 Web Content
4 R 1000 2790 2639 3 80 0 - 573961 - tty1 00:01:13 Web Content
4 S 1000 2868 2639 6 80 0 - 616052 poll_s tty1 00:02:06 Web Content
0 S 1000 3030 1644 0 80 0 - 150436 poll_s tty1 00:00:00 update-notifier
0 S 1000 3045 1560 0 80 0 - 51127 poll_s ? 00:00:00 gvfsd-metac
0 S 1000 3068 1 0 80 0 - 168958 poll_s tty1 00:00:01 gedit
0 S 1000 3327 1560 0 80 0 - 207480 poll_s ? 00:00:04 nautilus
0 R 1000 3353 1560 1 80 0 - 201724 poll_s ? 00:00:30 gnome-terminal
0 S 1000 3361 3353 0 80 0 - 13633 wait_w pts/0 00:00:00 zsh
1 I 0 3548 2 0 80 0 - 0 - ? 00:00:00 kworker/1:7339
1 I 0 3827 2 0 80 0 - 0 - ? 00:00:00 kworker/2:7338
1 I 0 4144 2 0 80 0 - 0 - ? 00:00:03 kworker/u8:7329
1 I 0 4651 2 0 80 0 - 0 - ? 00:00:00 kworker/1:7341
1 I 0 4845 2 0 80 0 - 0 - ? 00:00:00 kworker/u8:7337
1 I 0 4860 2 0 80 0 - 0 - ? 00:00:00 kworker/0:7334
1 I 0 4861 2 0 80 0 - 0 - ? 00:00:00 kworker/0:7336
1 I 0 4880 2 0 80 0 - 0 - ? 00:00:00 kworker/3:7333
1 I 0 5660 2 0 80 0 - 0 - ? 00:00:00 kworker/2:7335
0 S 1000 6383 3353 0 80 0 - 13633 sigsus pts/1 00:00:00 zsh
1 I 0 7079 2 0 80 0 - 0 - ? 00:00:00 kworker/3:7343
1 I 0 7220 2 0 80 0 - 0 - ? 00:00:00 kworker/0:7344
4 R 1000 7526 6383 0 80 0 - 9006 - pts/1 00:00:00 ps
→ Downloads
→ Downloads

```

1. ¿Qué significa la Z y a qué se debe?
 - a. La segunda columna es el estado del proceso, en la cual Z significa Zombie, o sea que el proceso no responde.
2. ¿Qué sucede en la ventana donde ejecutó su programa?
 - a. El programa continua en ejecución, por lo tanto se imprimen todos los índices de los ciclos for
3. ¿Quién es el padre del proceso que quedó huérfano?
 - a. Systemd

Ejercicio 5:

```
course@course-vm: ~/Downloads
File Edit View Search Terminal Help
→ Downloads gcc -o ipc ipc.c
/tmp/ccY7iWoL.o: In function `main':
ipc.c:(.text+0xd2): undefined reference to `shm_open'
ipc.c:(.text+0xfb): undefined reference to `shm_open'
ipc.c:(.text+0x3ae): undefined reference to `shm_unlink'
collect2: error: ld returned 1 exit status
→ Downloads gcc -o ipc ipc.c -lrt
→ Downloads gcc -o ej5 ej5.c
gcc: error: ej5.c: No such file or directory
gcc: fatal error: no input files
compilation terminated.
→ Downloads gcc -o ej5 ej5.c
→ Downloads ./ej5
I am a
a: Create new shared mem obj 3
I am b
b: Share mem obj already created
a: Initialized shared mem obj
a: ptr created with value 0x7f3dae05a000
b: Received shm fd: -1
b: ptr created with value 0x7f58e2b87000
a: Shared memory has: aaaaaaaaaaaaaaaaaaaaaaaaaa
b: Shared memory has: aaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbb
→ Downloads
```

1. ¿Qué diferencia hay entre realizar comunicación usando memoria compartida en lugar de usando un archivo de texto común y corriente?
 - a. AL usar memoria compartida varios programas puede acceder a ella de forma simultanea para comunicarse entre ellos. En este caso de un archivo de texto es imposible, ademas que en este tipo de situaciones se requiere de una menor intervencion del sistema operativo
2. ¿Por qué no se debe usar el file descriptor de la memoria compartida producido por otra instancia para realizar el mmap?
 - a. Porque el mismo file description no puede ser utilizado más de una vez en una misma instancia para la memoria compartida. Por otro lado la función mmap requiere de él file description, producido para esa instancia específica para poder ser ejecutado de la forma esperada.
3. ¿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)?
 - a. Si es posible. El comando ls busca en las locations del path y al encontrarlo se ejecuta en exec() por medio de pipes y el output es enviado.
4. ¿Cómo puede asegurarse de que ya se ha abierto un espacio de memoria compartida con un nombre determinado? Investigue y explique erro. 4.
 - a. Con el uso de errno se tiene la función shm_open, la cual retorna -1 si el espacio de memoria no existe. Por lo que cuando se obtenga un 0 como

respuesta se podrá identificar el espacio de memoria compartida que se usó.

- b. Erro también es una variable de tipo int definida por la llamadas al sistema y algunas librerías de error para poder indicar que ocurrió y cual es el número de error.
5. ¿Qué pasa si se ejecuta shm_unlink cuando hay procesos que todavía están usando la memoria compartida?
 - a. Se libera un espacio de memoria compartida, se elimina el nombre del objeto de memoria compartida, por lo cual shm_open deja de funcionar. Antes de liberar o destruir el espacio se espera que todos los procesos eliminen su mapeo.
6. ¿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.
 - a. Usando el buffer creado, este indica la información del puntero necesaria para escribir en el espacio de memoria compartida.
7. 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”?
 - a. munmap() se usa para eliminar un espacio de memoria compartida referido en sus parámetros como direcciones y tamaño, por lo que sería necesario llamar a la función en caso de error.
8. 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.
 - a. En promedio la llamada a sistema fork() lleva 12.08 microsegs. La función usleep() suspende la ejecución de cierto proceso por intervalos de microsegundos y cuyo parámetro es la cantidad de microsegundos a pausar.

FUENTES DE CONSULTA

GitBook. (n.d.). ps · Comandos Linux. Retrieved from <https://didweb.gitbooks.io/comandos-linux/content/chapter1/procesos/ps.html>

Silberschatz, A., Galvin, P., & Gagne, G. (2013). Operating System Concepts. John Wiley & Sons, Inc.

Universidad Carlos III de Madrid. (2015). Introducción a la gestión de procesos. Retrieved from

http://ocw.uc3m.es/ingenieria-informatica/sistemas-operativos/material-de-clase-1/mt_t2_l3.pdf