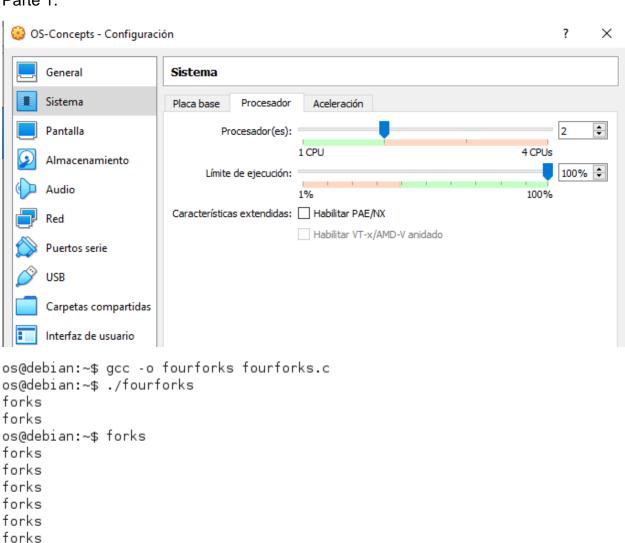
Universidad del Valle de Guatemala José Gabriel Block Staackmann carne no. 18935 Sistemas Operativos

HDT#2

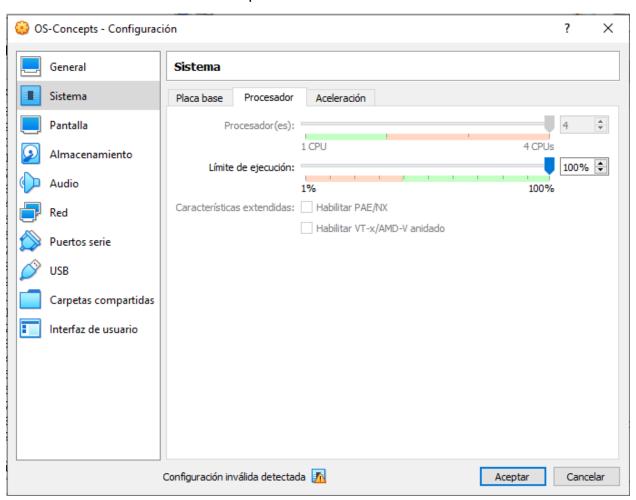
Parte 1.

forks forks forks forks forks forks



forksos@debian:~\$ gcc -o forforks forforks.c os@debian:~\$./forforks forks os@debian:~\$

- ¿Cuántos procesos se crean en cada uno de los programas? En el primero de los 4 forks se crean 16 y en el segundo del for, se crea uno.
- ¿Por qué hay tantos procesos en ambos programas cuando uno tiene cuatro llamadas fork()y el otro sólo tiene una?
 Porque al llamar cada función, hace 4 llamadas diferentes, porque son hechas al mismo tiempo, pero con el for hace la misma llamada 4 veces por eso ambas usan la misma cantidad de procesos.



Parte 2.

```
blockmann@blockmann-VirtualBox:~/Documents/Hdt2$ gcc -o clocklockmann@blockmann-VirtualBox:~/Documents/Hdt2$ ./clock1 6104.0000000 blockmann@blockmann-VirtualBox:~/Documents/Hdt2$ ./clock1 5971.0000000 blockmann@blockmann-VirtualBox:~/Documents/Hdt2$ ./clock1 5907.0000000 blockmann@blockmann-VirtualBox:~/Documents/Hdt2$ ./clock1 6749.0000000 blockmann@blockmann-VirtualBox:~/Documents/Hdt2$ ./clock1 6140.0000000 blockmann@blockmann-VirtualBox:~/Documents/Hdt2$ ./clock1 6037.0000000 blockmann@blockmann-VirtualBox:~/Documents/Hdt2$ ./clock1 6037.0000000 blockmann@blockmann-VirtualBox:~/Documents/Hdt2$
```

```
blockmann@blockmann-VirtualBox:~/Documents/Hdt2$ ./clock2 90.000000 blockmann@blockmann-VirtualBox:~/Documents/Hdt2$ ./clock2 120.000000 blockmann@blockmann-VirtualBox:~/Documents/Hdt2$ ./clock2 93.000000 blockmann@blockmann-VirtualBox:~/Documents/Hdt2$ ./clock2 119.000000 blockmann@blockmann-VirtualBox:~/Documents/Hdt2$
```

- ¿Cuál, en general, toma tiempos más largos?
 El primer programa tarda mucho más que el segundo.
- ¿Qué causa la diferencia de tiempo, o por qué se tarda más el que se tarda más?
 - El orden en el que se organizan y separan en diferentes procesos acelera el tiempo de tardanza de forma notoria.

Tipos de c. de contexto

- · Cambio "voluntario":
 - Proceso realiza llamada al sistema que implica esperar por evento
 - Transición de en ejecución a bloqueado
 - Ejemplos: leer del terminal o bajar un semáforo cerrado
 - Motivo: Eficiencia en el uso del procesador
- · Cambio "involuntario":
 - S.O. le quita la UCP al proceso
 - Transición de en ejecución a listo
 - Ejemplos: fin de rodaja de ejecución o pasa a listo proceso bloqueado de mayor prioridad
 - Motivo: Reparto del procesador

The following NEW packages will be installed:	10:13:28 PM	4 PID	cswch/s	nvcswch/s	Comr
sysstat	and				
O upgraded, 1 newly installed, O to remove and 2	10:13:29 PM	1 9	1.00	0.00	ksof
Need to get 284 kB of archives.	tirqd/1				
After this open ation, 1,151 kB of additional dis	10:13:29 PM	1 16	1.00	0.00	ksof
WARNING: The following packages cannot be authen	tirad/3				
sysstat	10:13:29 PM	4 31	71.00	0.00	kwor
Install these packages without verification [y/N	ker/1:1				
Err http://ftp.us.debian.org/debian/ squeeze/mai	10:13:29 PM	441	1.00	0.00	kwor
404 Not Found [IP: 64.50.233.100 80]	ker/2:2				
Get:1 http://archive.debian.org/debian/ squeeze/	10:13:29 PM	1 1001	3.00	0.00	Xorq
kB]	10:13:29 PM	1 1847	1.00	0.00	upda
Fetched 284 kB in 1s (222 kB/s)	te-notifier				'
Preconfiguring packages	10:13:29 PM	1 1854	1.00	0.00	gnor
Selecting previously deselected package sysstat.	e-screensav	/			
(Reading database 134149 files and directori	10:13:29 PM	1 2105	32.00	2.00	gnor
Unpacking sysstat (from/sysstat_9.0.6.1-2_i3	e-terminal				
Processing triggers for man-db	10:13:29 PM	1 2288	1.00	0.00	kwor
Setting up sysstat (9.0.6.1-2)	ker/3:1				
	10:13:29 PM	1 2338	1.00	0.00	flus
Creating config file /etc/default/sysstat with n	h-8:0				
update-alternatives: using /usr/bin/sar.sysstat	10:13:29 PM	1 2530	1.00	71.00	pids
auto mode	tat				•
os@debian:~\$	~				

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

Al teclear cosas o mover el mouse sobre otras, incrementan los pidstats y los Xorgs al usar un programa aumentaron los números de keyworkers.



- ¿Qué diferencia hay en el número y tipo de cambios de contexto de entre programas?
 - Al correr el programa concurrente de forks, se puede notar una disminución de los cambios de contexto involuntarios, y los voluntarios aumentan.
- ¿A qué puede atribuir los cambios de contexto voluntarios realizados por sus programas?
 - Es por los forks() y su forma de funcionar de manera secuencial.
- ¿A qué puede atribuir los cambios de contexto involuntarios realizados por sus programas?
 - Por los procesos del SO, al llevar un tiempo prolongado de corrida se dan los cambios de contexto involuntarios.
- ¿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 un fork() no espera a ser ejecutado, por lo tanto este se realiza rápido y por lo tanto se generan los procesos hijos.
- ¿Qué efecto percibe sobre el número de cambios de contexto de cada tipo?
 Aumentan cambios involuntarios.

Parte 4.

00:00:00 bash pts/l 1 S 0 2628 2 0 80 0 -0 ? ? 00:00:00 flush-8:0 1 S 0 2660 2 0 80 0.7 ? 00:00:01 kworker/3:1 1 S 2 0 80 0 2687 0 -0 ? ? 00:00:00 kworker/0:1 0 2689 2 0 80 ? 00:00:00 kworker/3:2 1 S 0 2744 2 0 80 0 -0 ? ? 00:00:00 kworker/3:0 0 R 1000 2761 2107 99 80 396 -pts/0 00:01:34 parte4 repadrepadrepadrepadrepadrepadrepadrep 1 Z 1000 2762 2761 0 80 0 -0 ? adrepadrepadrepadrepadrepadrepa^C 00:00:00 parte4 <defunct> pts/0 ps@debian:~\$ gcc -o parte4 perte4hdt2.c 0 R 1000 2764 2506 0 80 0 -928 os@debian:~\$./parte4 00:00:00 ps pts/l nijo os@debian:~\$

¿Qué significa la Z y a qué se debe?
 Z significa que es un proceso zombie, o sea que este a terminado su tarea, pero el producto de esta no ha sido usado de ninguna forma.

```
3999991
                                                         0 2772
                                                                    2 0 80
                                                                                       0 ?
                                                 1 S
3999992
                                                           00:00:01 kworker/3:0
3999993
                                                 0 T 1000 2773 2107 1 80
                                                                               0 -
                                                                                     396 -
3999994
                                                  pts/0
                                                         00:00:03 parte4
3999995
                                                 1 Z 1000 2774 2773 0 80
                                                                               0 -
                                                                                       0 ?
3999996
                                                         00:00:00 parte4 <defunct>
                                                  pts/0
3999997
                                                 1 S
                                                         0 2814
                                                                    2 0 80
                                                                                       0 ?
3999998
                                                           00:00:00 kworker/3:1
3999999
                                                 0 R 1000 2815 2506 0 80
                                                                                     928 -
^C
                                                  pts/l
                                                           00:00:00 ps
os@debian:~$
                                                 os@debian:~$
3999994
                                                               00:00:00 ps
                                                     pts/l
3999995
                                                    os@debian:~$ kill -9 <2814>
3999996
                                                    bash: syntax error near unexpected tol
3999997
                                                    os@debian:~$ kill -9 2814
3999998
                                                    bash: kill: (2814) - Operation not per
3999999
                                                    os@debian:~$ kill -9 2774
Killed
                                                    os@debian:~$ kill -9 2816
os@debian:~$
                                                    os@debian:∼$
```

```
00:00:00 kworker/3:1
0 R 1000
         2816 2107 99
                                     396 -
pts/0
         00:00:05 parte4
1 S 1000 2817
                2816 53
                                     397 -
pts/0
         00:00:02 parte4
0 R 1000 2818 2506
                      0
                                     928 -
pts/l
         00:00:00 ps
os@debian:~$ kill -9 <2814>
bash: syntax error near unexpected token `2814'
os@debian:~$ kill -9 2814
bash: kill: (2814) - Operation not permitted
os@debian:~$ kill -9 2774
os@debian:~$ kill -9 2816
```

- ¿Qué sucede en la ventana donde ejecutó su programa? En la Ventana que se usó el programa, se está contando de 0 a 3999999.
- ¿Quién es el padre del proceso que quedó huérfano?
 El padre es el 2816, y quedo huérfano el 2817.

Parte 5.

```
a: Objeto de la memoria compartida: 5
a: Memoria compartida: ej5 a 0x7f367eead000
a: padre: 18301
a: hijo: 0
Tiempo transcurrido: -574
a: La memoria compartida tiene: AAAAAAAAAAA
Tiempo transcurrido: 421
b: Objeto de la memoria compartida: 5
b: Memoria compartida: ej5 b 0x7f8344e3e000
b: padre: 18302
b: hijo: 0
Tiempo transcurrido: -1659
b: La memoria compartida tiene: BBBBBBBBBB
Tiempo transcurrido: 365
blockmann@blockmann-VirtualBox:~/Documents/Hdt2$
```

- ¿Qué diferencia hay entre realizar comunicación usando memoria compartida en lugar de usando un archivo de texto común y corriente?
 La memoria compartida suele ser más rápida con el paso de mensajes porque usa menos al sistema operativo porque no requiere de un protocolo ni preparación y manejo especial de la información. Por otro lado está la transmisión de mensajes send() y receive() que se hacen a través de un enlace de comunicación y se diferencia entre directa e indirecta.
- ¿Por qué no se debe usar el file descriptor de la memoria compartida producido por otra instancia para realizar el mmap?
 El file descriptor no puede ser utilizado más de una vez por instancias y además el file descriptor es necesario para que el mmap funcione adecuadamente.
- ¿Es posible enviar el output de un programa ejecutado con execl otro proceso por medio de un pipe? Investigue y explique cómo funciona este mecanismo en la terminal (e.g., la ejecución de ls | less).
 La ejecución ls busca el path en las ubicaciones posibles del directorio, una vez encontrado entonces se ejecuta execl y por un pipe se envía el resultado de la búsqueda.

- ¿Cómo puede asegurarse de que ya se ha abierto un espacio de memoria compartida con un nombre determinado? Investigue y explique errno.
 Errno es un int var que se define gracias a las llamadas al sistema que retornan un tipo de error o no. Con errno se utiliza shm_open que sirve para saber si el espacio en la memoria se generó correctamente.
- ¿Qué pasa si se ejecuta shm_unlink cuando hay procesos que todavía están usando la memoria compartida?
 Todos los procesos dejan de hacer el mapeo, o sea que deja de funcionar y por lo tanto se libera el espacio de memoria compartida.
- ¿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.
 Se podría con el buffer ya que muestra la información del puntero para poder escribir en la memoria compartida.
- 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"?
 Con munmap() ya que este elimina un mapeo en la memoria compartida.
 https://pubs.opengroup.org/onlinepubs/009695399/functions/munmap.html
- 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.
 - Toma alrededor de 800 tics de la función clock() microsegundos con buffer de 100. Usleep detiene la a un proceso por un cierto tiempo determinado.