

MiniTP: Shell y procesos.

Materia: Sistemas Operativos y Redes I

Comisión: 1, turno noche.

Alumno: Tomás Menegol.

Carrera: Licenciatura y Tecnicatura en sistemas.

2) Capturas de pantalla, estadosDeUnProceso.c corriendo:

```
tomasm@DeskTom: ~
0[ | 1.4%] 4[ | 0.7%]
1[ | 1.1%] 5[ | 100.0%]
2[ | 2.1%] 6[ | 0.0%]
3[ | 1.4%] 7[ | 0.0%]
Mem[ | 579M/4.75G] Tasks: 40, 29 thr; 2 running
Swp[ | 0K/2.00G] Load average: 0.12 0.07 0.04
Uptime: 01:09:16

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command (merged)
1 root 20 0 163M 12796 8480 S 1.8 0.3 1:20.34 /lib/systemd/systemd --system --deserialize 21
2 root 20 0 2476 1432 1320 S 0.0 0.0 0:00.04 init-systemd(ub)/init
7 root 20 0 2476 4 0 S 0.0 0.0 0:00.00 init|plan9 --control-socket 6 --log-level 4 --server-fd 7 --pipe-fd 9 --lo
8 root 20 0 2476 4 0 S 0.0 0.0 0:00.00 init|plan9 --control-socket 6 --log-level 4 --server-fd 7 --pipe-fd 9
9 root 20 0 2476 1432 1320 S 0.0 0.0 0:00.00 Interop/|init
4097 root 20 0 2480 112 0 S 0.0 0.0 0:00.00 SessionLeader/|init
4088 root 20 0 2496 116 0 S 0.0 0.0 0:01.71 Relay(409)/|init
409 tomasm 20 0 6308 5316 3428 S 0.0 0.1 0:00.49 /usr/bin/bash|-bash
16780 tomasm 20 0 2644 992 900 R 99.7 0.0 0:05.21 /home/tomasm/SOR1/estadosDeUnProceso ./estadosDeUnProceso
410 root 20 0 7524 5008 4084 S 0.0 0.1 0:00.04 /bin/login -f
476 tomasm 20 0 6068 4996 3356 S 0.0 0.1 0:00.05 /usr/bin/bash|-bash
10505 root 20 0 2480 116 0 S 0.0 0.0 0:00.00 SessionLeader/|init
10506 root 20 0 2496 120 0 S 0.0 0.0 0:00.50 Relay(10511)/|init
10511 tomasm 20 0 6084 5060 3336 S 0.0 0.1 0:00.11 /usr/bin/bash|-bash
10586 tomasm 20 0 5980 4736 3304 R 0.0 0.1 0:07.35 /usr/bin/htop
```

estadosDeUnProceso.c esperando I/O:

```
tomasm@DeskTom: ~
0[ | 1.3%] 4[ | 0.0%]
1[ | 0.0%] 5[ | 0.0%]
2[ | 0.7%] 6[ | 0.0%]
3[ | 1.3%] 7[ | 1.3%]
Mem[ | 586M/4.75G] Tasks: 40, 29 thr; 1 running
Swp[ | 0K/2.00G] Load average: 0.17 0.08 0.04
Uptime: 01:09:28

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command (merged)
1 root 20 0 163M 12796 8480 S 2.0 0.3 1:20.57 /lib/systemd/systemd --system --deserialize 21
2 root 20 0 2476 1432 1320 S 0.0 0.0 0:00.04 init-systemd(ub)/init
7 root 20 0 2476 4 0 S 0.0 0.0 0:00.00 init|plan9 --control-socket 6 --log-level 4 --server-fd 7 --pipe-fd 9 --lo
8 root 20 0 2476 4 0 S 0.0 0.0 0:00.00 init|plan9 --control-socket 6 --log-level 4 --server-fd 7 --pipe-fd 9
9 root 20 0 2476 1432 1320 S 0.0 0.0 0:00.00 Interop/|init
4097 root 20 0 2480 112 0 S 0.0 0.0 0:00.00 SessionLeader/|init
4088 root 20 0 2496 116 0 S 0.0 0.0 0:01.71 Relay(409)/|init
409 tomasm 20 0 6308 5316 3428 S 0.0 0.1 0:00.49 /usr/bin/bash|-bash
16780 tomasm 20 0 2776 992 900 S 0.0 0.0 0:09.70 /home/tomasm/SOR1/estadosDeUnProceso ./estadosDeUnProceso
410 root 20 0 7524 5008 4084 S 0.0 0.1 0:00.04 /bin/login -f
476 tomasm 20 0 6068 4996 3356 S 0.0 0.1 0:00.05 /usr/bin/bash|-bash
10505 root 20 0 2480 116 0 S 0.0 0.0 0:00.00 SessionLeader/|init
10506 root 20 0 2496 120 0 S 0.0 0.0 0:00.50 Relay(10511)/|init
10511 tomasm 20 0 6084 5060 3336 S 0.0 0.1 0:00.11 /usr/bin/bash|-bash
10586 tomasm 20 0 5980 4736 3304 R 0.0 0.1 0:07.41 /usr/bin/htop
42 root 19 -1 47800 14720 13696 S 0.0 0.3 0:00.43 /lib/systemd/systemd-journald
63 root 20 0 21960 5908 4576 S 0.0 0.1 0:00.85 /lib/systemd/systemd-udev
74 root 20 0 4492 196 44 S 0.0 0.0 0:00.00 snapfuse /var/lib/snapd/snaps/bare_5.snap /snap/bare/5 -o ro,nodev,allow_oth
75 root 20 0 4856 1884 1248 S 0.0 0.0 0:02.71 snapfuse /var/lib/snapd/snaps/core22_864.snap /snap/core22/864 -o ro,nodev,al
```

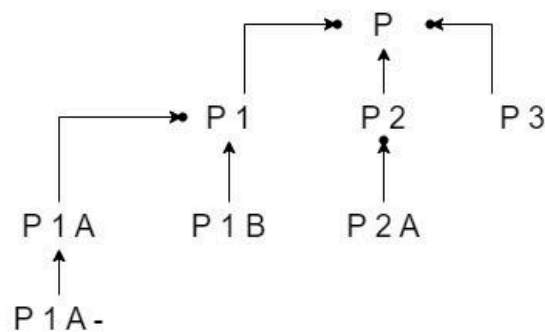
3) El output del programa dado es el siguiente:

```
tomasm@DeskTom: ~/SOR1$ ./procesosYFork
Soy un proceso!
Soy un proceso!
Soy un proceso!
Soy un proceso!
Soy un proceso!
Soy un proceso!
Soy un proceso!
Soy un proceso!
Soy un proceso!
Soy un proceso!
Soy un proceso!
Soy un proceso!
Soy un proceso!
Soy un proceso!
Soy un proceso!
tomasm@DeskTom: ~/SOR1$ Soy un proceso!
Soy un proceso!
```

Podemos ver 2 cosas importantes, la primera es que hay 14 prints en consola, y la segunda es que cuando se llega al final, el programa no "corta". Esto último es debido a que el código proporcionado no realiza un `wait()` para los procesos hijo. En cuanto a los prints, estos pueden ser explicados si recordamos que los mismos se disparan cada vez que:

1. Un proceso padre genera un proceso hijo.
2. Cada vez que un proceso hijo "nace".

El recorrido sería el siguiente: Tenemos un proceso inicial, que realiza una primera iteración donde crea un hijo. En este punto, al proceso inicial le quedan 2 iteraciones más que realizar (2 hijos más por generar). Como su proceso hijo es una copia idéntica, este nuevo proceso ahora también tiene 2 hijos restantes que generar. Posterior, el padre vuelve a iterar, ahora le queda un solo hijo más, con lo que el nuevo proceso recién generado por el padre original, también tiene una sola iteración restante. Ahora bien, el primer hijo de todos realiza una iteración por su cuenta, quedándose así con un solo hijo restante (si recordamos este proceso fue creado con 2 hijos restantes). El proceso resultado de esto, también tiene un solo hijo restante. Así sucesivamente se van limitando la cantidad de hijos posteriores que tendrá el siguiente proceso hasta llegar al siguiente esquema:



4) Por default, el programa anterior tarda entre 25 a 26 segundos en ejecutarse. Sin embargo, cuando se divide el programa en 5 procesos distintos, este tarda sólo 6 segundos, mostrando una clara mejora en el tiempo de ejecución. Esto resalta la importancia de poder dividir los trabajos entre varios procesos, ya que esto nos *permite* cierto nivel de paralelismo (que no significa que lo haya, esto lo decide el SO y su scheduler) y concurrencia si habláramos de procesos ejecutándose con hyperthreading. De hecho podemos ver como se divide la carga entre los procesadores cuando el primer código se ejecutaba sobre un solo núcleo:

```
tomasm@DeskTom: ~  
  
0[ | 0.7% 4[ | 0.7%  
1[ | 0.0% 5[ | 0.0%  
2[ | 0.7% 6[ | 100.0%  
3[ | 3.3% 7[ | 0.0%  
Mem[ | 564M/4.75G Tasks: 40, 29 thr; 2 running  
Swp[ | 0K/2.00G Load average: 0.20 0.17 0.10  
Uptime: 03:55:24  
  
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command (merged)  
1 root 20 0 163M 12804 8480 S 1.3 0.3 4:20.85 /lib/systemd/systemd --system --deseriali  
2 root 20 0 2476 1432 1320 S 0.0 0.0 0:00.06 init-systemd(Ub)/init  
7 root 20 0 2476 4 0 S 0.0 0.0 0:00.00 init|plan9 --control-socket 6 --log  
8 root 20 0 2476 4 0 S 0.0 0.0 0:00.00 init|plan9 --control-socket 6 --  
9 root 20 0 2476 1432 1320 S 0.0 0.0 0:00.00 Interop|/init  
407 root 20 0 2480 112 0 S 0.0 0.0 0:00.00 SessionLeader|/init  
408 root 20 0 2496 116 0 S 0.0 0.0 0:02.45 Relay(409)|/init  
409 tomas 20 0 6308 5316 3428 S 0.0 0.1 0:00.85 /usr/bin/bash|-bash  
55678 tomas 20 0 2644 1112 1024 R 100.0 0.0 0:04.24 /home/tomasm/SOR1/ejecutab  
410 root 20 0 7524 5008 4084 S 0.0 0.1 0:00.04 /bin/login -f  
476 tomas 20 0 6068 4996 3356 S 0.0 0.1 0:00.05 /usr/bin/bash|-bash
```

Pero el código con múltiples child process se ejecuta en 5 procesadores (como mínimo uno virtual):

```
tomasm@DeskTom: ~  
  
0[ ||| 8.1% → 4[ | 100.0%  
→ 1[ | 100.0% 5[ ||| 7.5%  
2[ ||| 6.4% → 6[ | 100.0%  
→ 3[ | 100.0% 7[ | 100.0%  
Mem[ | 573M/4.75G Tasks: 45, 29 thr; 6 running  
Swp[ | 0K/2.00G Load average: 0.49 0.16 0.11  
Uptime: 04:22:50  
  
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command (merged)  
1 root 20 0 163M 12804 8480 S 2.6 0.3 4:48.46 /lib/systemd/systemd --system --deseriali  
2 root 20 0 2476 1432 1320 S 0.0 0.0 0:00.06 init-systemd(Ub)/init  
7 root 20 0 2476 4 0 S 0.0 0.0 0:00.00 init|plan9 --control-socket 6 --log  
8 root 20 0 2476 4 0 S 0.0 0.0 0:00.00 init|plan9 --control-socket 6 --  
9 root 20 0 2476 1432 1320 S 0.0 0.0 0:00.00 Interop|/init  
407 root 20 0 2480 112 0 S 0.0 0.0 0:00.00 SessionLeader|/init  
408 root 20 0 2496 116 0 S 0.0 0.0 0:02.79 Relay(409)|/init  
409 tomas 20 0 6308 5316 3428 S 0.0 0.1 0:00.91 /usr/bin/bash|-bash  
62019 tomas 20 0 2644 1084 992 S 0.0 0.0 0:00.00 /home/tomasm/SOR1/ejecutab  
62020 tomas 20 0 2776 92 0 R 97.5 0.0 0:04.96 /home/tomasm/SOR1/ejecu
```

De aquí podemos hacer hincapié en la importancia de la cantidad de núcleos que se tienen disponibles para repartirse tareas, ya que cuantos más de estos tengamos, más carga podremos dividir entre ellos.