MiniTP: Shell y procesos.

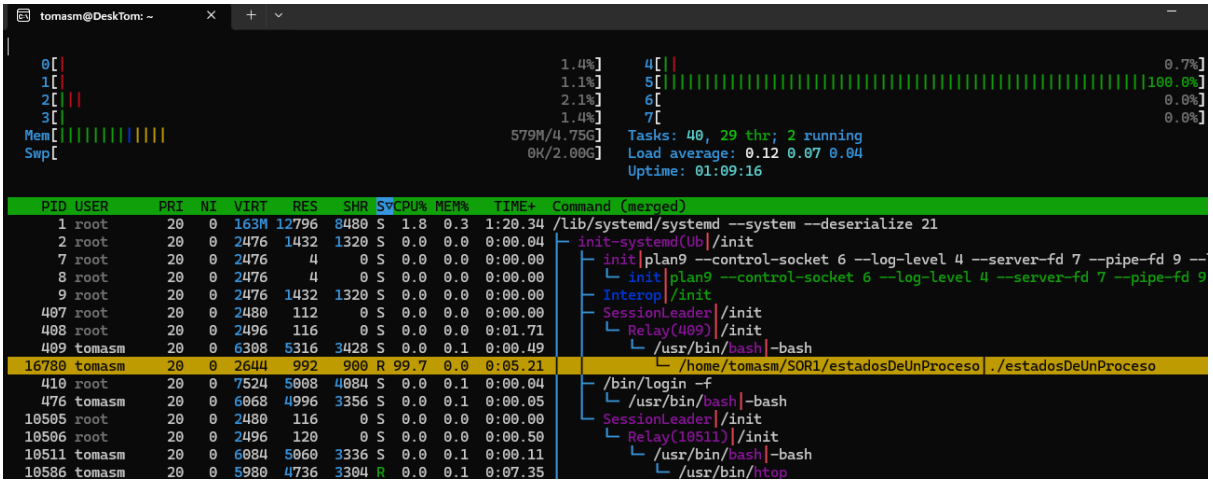
Materia: Sistemas Operativos y Redes I

Comisión: 1, turno noche.

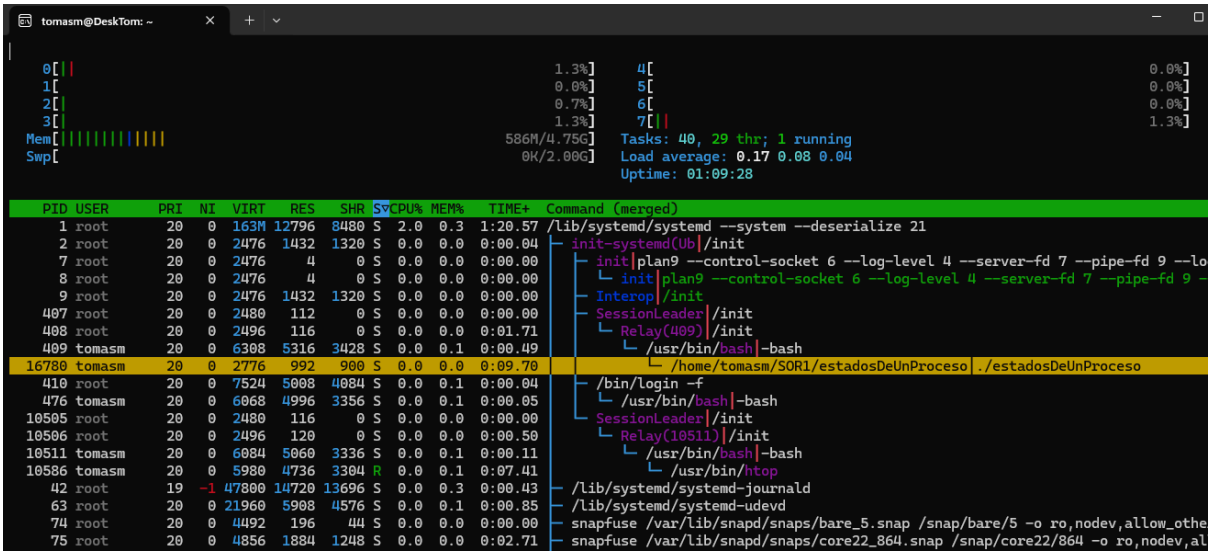
Alumno: Tomás Menegol.

Carrera: Licenciatura y Tecnicatura en sistemas.

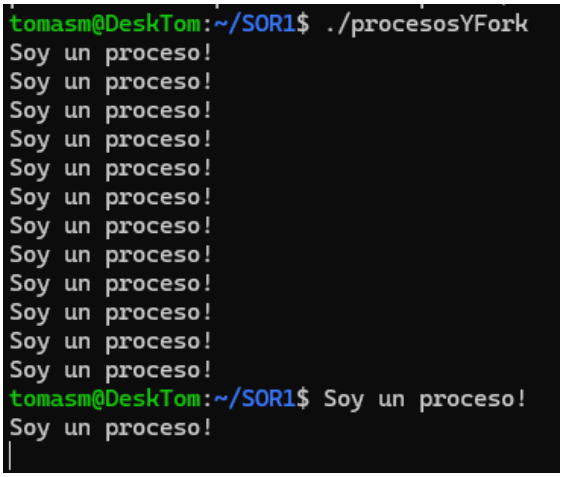
2) Capturas de pantalla, estadosDeUnProceso.c corriendo:



estadosDeUnProceso.c esperando I/O:

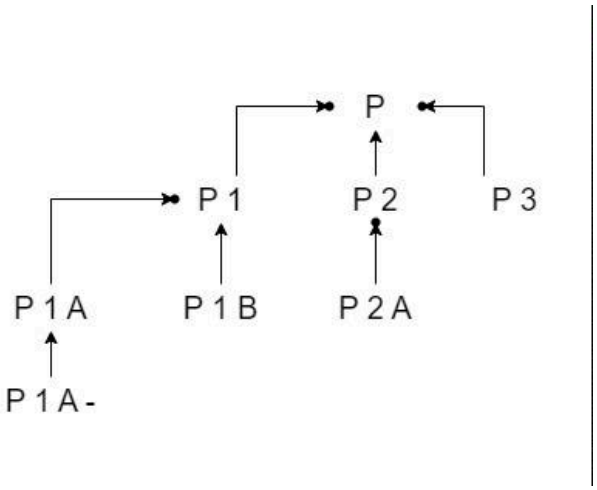


3) El output del programa dado es el siguiente:

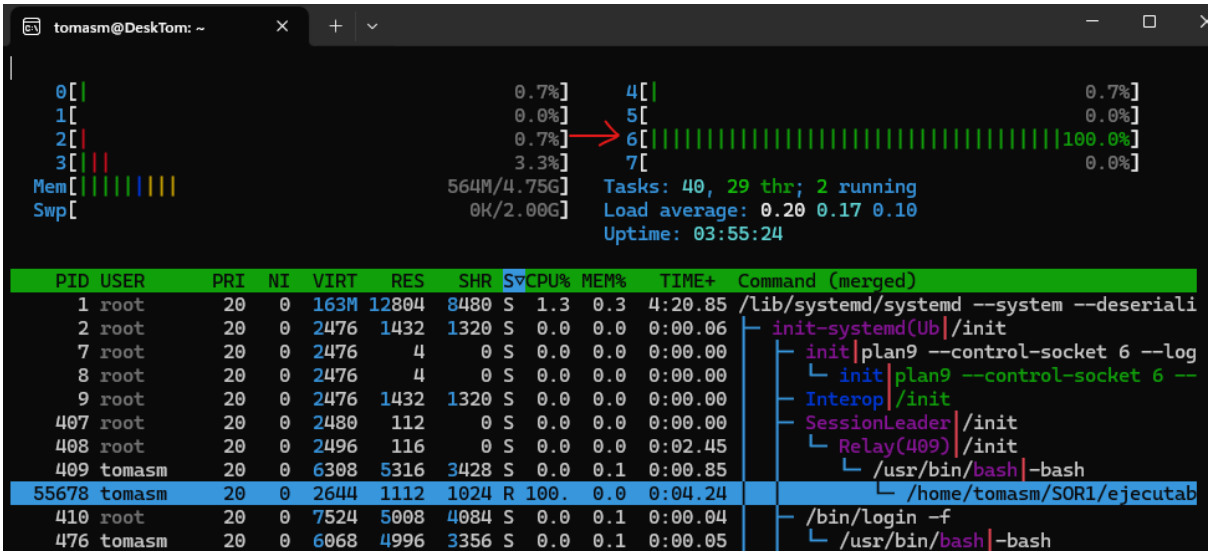


Podemos ver 2 cosas importantes, la primera es que hay 14 prints en consola, y la segunda es que cuando se llega al final, el programa no "corta". Esto último es debido a que el código proporcionado no realiza un wait() para los procesos hijo. En cuanto a los prints, estos pueden ser explicados si recordamos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

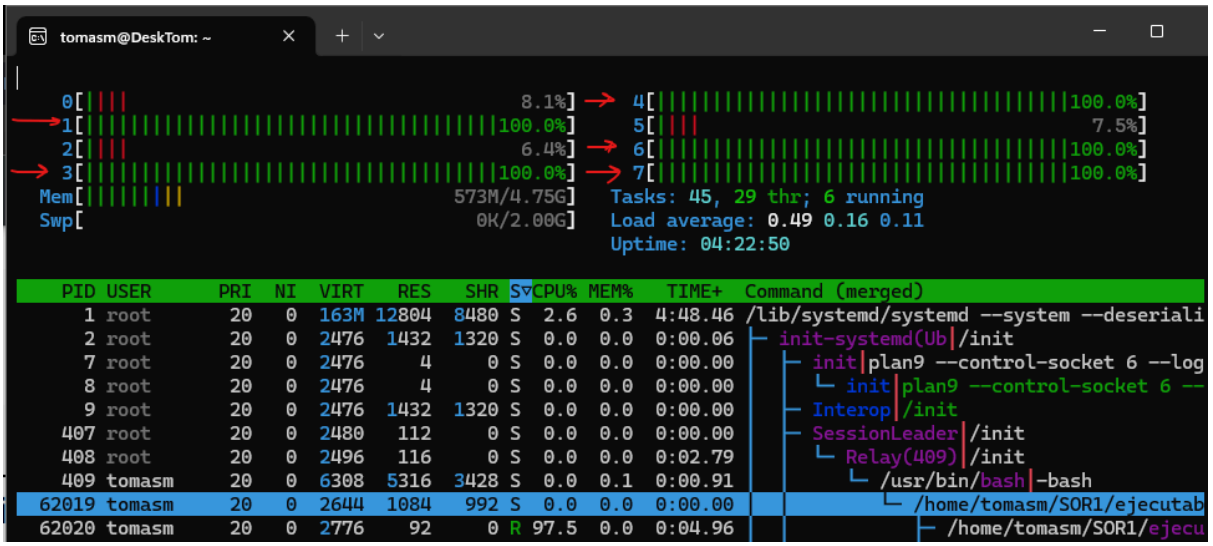
El recorrido sería el siguiente: Tenemos un proceso inicial, que realiza una primera iteración donde crea un hijo. En este punto, al proceso inicial le quedan 2 iteraciones más que realizar (2 hijos más por generar). Como su proceso hijo es una copia idéntica, este nuevo proceso ahora también tiene 2 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 hablaramos 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:



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



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.