LABORATORIO 2

Ejercicio 1:

a. 4 forks: 16 resultados.
   fork en un for de 4 iteraciones: 63 resultados.
b. En el caso de los 4 forks porque cada uno crea un proceso hijo y esos procesos a su vez crean otro proceso al siguiente fork, por lo que en este caso la cantidad de procesos será 2 a la n donde n será la cantidad de forks. en el caso del ciclo sucede que en cada iteración se crea un proceso hijo que se reiniciará donde se encuentre la iteración, en consecuencia se crean muchísimos subprocesos que aumentan enormemente cuantas más iteraciones se hagan.

Ejercicio 2:

Tiempos de corrida 1:

1. 0.000074s
2. 0.000054s
3. 0.000064s
4. 0.000049s
5. 0.000050s

Tiempos de corrida 2:

1. 0.000030 s -0.000272 s -0.000267 s -0.000292 s
2. 0.000029 s -0.000270 s -0.000243 s -0.000226 s
3. 0.000028 s -0.000240 s -0.000219 s -0.000194 s
4. 0.000031 s -0.000340 s -0.000317 s -0.000242 s
5. 0.000031 s -0.000343 s -0.000316 s -0.000249 s

Los segundos tiempos de corrida son mayores dado que en el hay procesos a la espera de que otros terminen antes de poder proceder.

ejer2.c
ejer3.c

Ejercicio 5:

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

•¿Por qué no se debe usar el file descriptor de la memoria compartida producido por otra instancia para realizar el mmap?
Porque el addres puede ser variar y en dado caso el mmap no funciona

•¿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).
Es posible, parecido al funcionamiento de este ejercicio, el pipe funciona como medio de comunicación, por lo que un programa puede dejar escrito en el pipe el output deseado y el otro lo recogerá de ahí.

•¿Cómo puede asegurarse de que ya se ha abierto un espacio de memoria compartida con un nombre determinado? Investigue y explique errno.
Usando un objeto de memoria compartida. luego con shm_open() y un if se puede verificar si ya existe dependiendo del valor que devuelva, siendo -1 por si no existe.

•¿Qué pasa si se ejecuta shm_unlink cuando hay procesos que todavía están usando la memoria compartida?

Esos procesos se detienen pues, con eso se está desvinculando la memoría compartida y ya no podrán acceder a ella.
•¿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.
basta con hacer un llamado al puntero ahí donde se necesite la localización que guarda.

•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”?
haciendo uin llamado a shm_unlink y matando los procesos que aun queden.

•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.