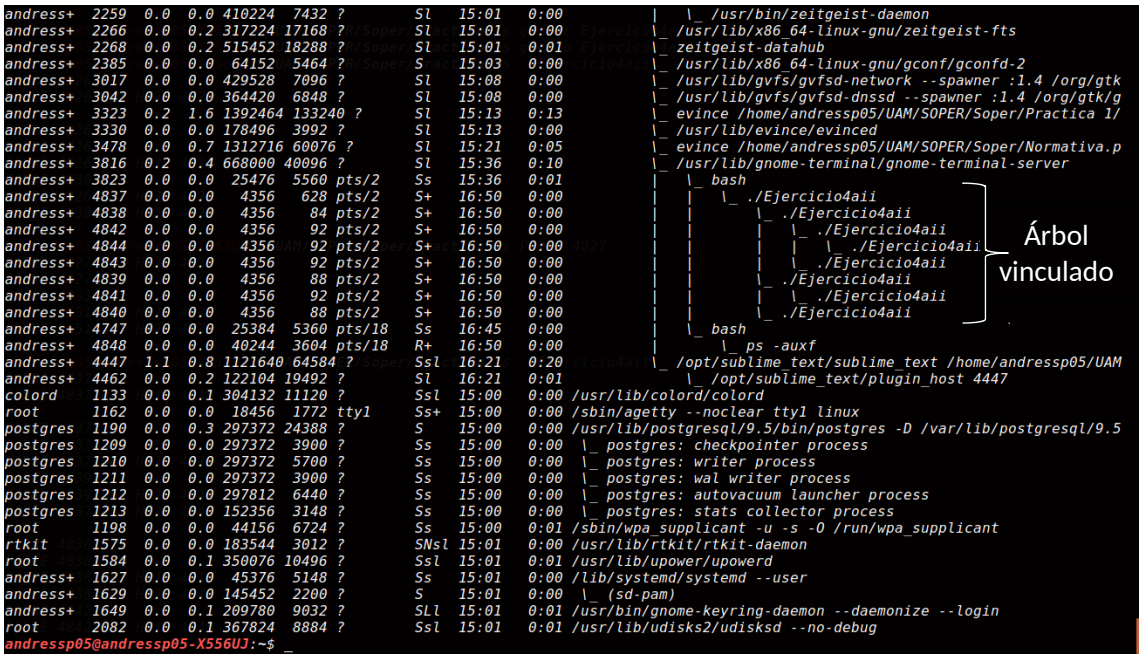
SISTEMAS OPERATIVOS: Informe práctica 1

Ejercicio 4:

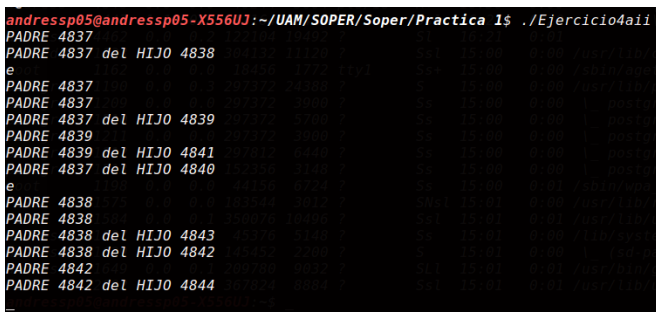
En este ejercicio se pedía analizar el árbol de procesos resultante. Como se puede observar en la figura 1, el proceso inicial crea un hijo en la primera iteración del bucle. En la siguiente iteración, ambos crean un hijo propio. Finalmente, en la tercera iteración, los cuatro procesos generan un hijo cada uno. El árbol resultante tiene una raíz con tres hijos, el tercero es una hoja, el segundo tiene un hijo y el primero tiene dos hijos y un 'nieto'.



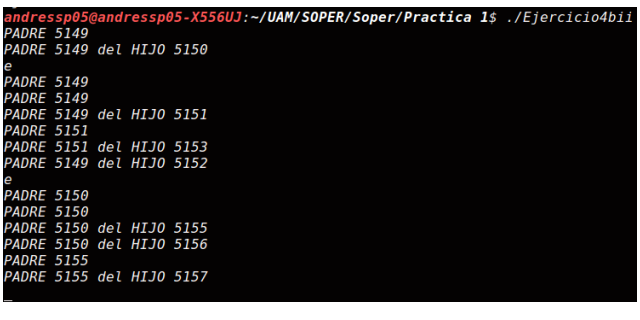

```
andress+  2259  0.0  0.0 410224  7432 ?        Sl   15:01   0:00          |   \_ /usr/bin/zeitgeist-daemon
andress+  2266  0.0  0.2 317224 17168 ?        Sl   15:01   0:00          \_ /usr/lib/x86_64-linux-gnu/zeitgeist-fts
andress+  2268  0.0  0.2 515452 18288 ?        Sl   15:01   0:01          \_ zeitgeist-datahub
andress+  2385  0.0  0.0  64152  5464 ?        S    15:03   0:00          \_ /usr/lib/x86_64-linux-gnu/gconf/gconfd-2
andress+  3017  0.0  0.0 429528  7096 ?        Sl   15:08   0:00          \_ /usr/lib/gvfs/gvfsd-network --spawner :1.4 /org/gtk
andress+  3042  0.0  0.0 364420  6848 ?        Sl   15:08   0:00          \_ /usr/lib/gvfs/gvfsd-dnssd --spawner :1.4 /org/gtk/g
andress+  3323  0.2  1.6 1392464 133240 ?      Sl   15:13   0:13          \_ evince /home/andressp05/UAM/SOPER/Soper/Practica 1/
andress+  3330  0.0  0.0 178496  3992 ?        Sl   15:13   0:00          \_ /usr/lib/evince/evinced
andress+  3478  0.0  0.7 1312716 60076 ?       Sl   15:21   0:05          \_ evince /home/andressp05/UAM/SOPER/Soper/Normativa.p
andress+  3816  0.2  0.4 668000 40096 ?        Sl   15:36   0:10          \_ /usr/lib/gnome-terminal/gnome-terminal-server
andress+  3823  0.0  0.0  25476  5560 pts/2    Ss   15:36   0:01          |   \_ bash
andress+  4837  0.0  0.0   4356   628 pts/2    S+   16:50   0:00          |   |   \_ ./Ejercicio4aii
andress+  4838  0.0  0.0   4356    84 pts/2    S+   16:50   0:00          |   |       \_ ./Ejercicio4aii
andress+  4842  0.0  0.0   4356    92 pts/2    S+   16:50   0:00          |   |       |   \_ ./Ejercicio4aii
andress+  4844  0.0  0.0   4356    92 pts/2    S+   16:50   0:00          |   |       |   |   \_ ./Ejercicio4aii
andress+  4843  0.0  0.0   4356    92 pts/2    S+   16:50   0:00          |   |       |   \_ ./Ejercicio4aii
andress+  4839  0.0  0.0   4356    88 pts/2    S+   16:50   0:00          |   |       \_ ./Ejercicio4aii
andress+  4841  0.0  0.0   4356    92 pts/2    S+   16:50   0:00          |   |       |   \_ ./Ejercicio4aii
andress+  4840  0.0  0.0   4356    88 pts/2    S+   16:50   0:00          |   |       \_ ./Ejercicio4aii
andress+  4747  0.0  0.0  25384  5360 pts/18   Ss   16:45   0:00          |   \_ bash
andress+  4848  0.0  0.0  40244  3604 pts/18   R+   16:50   0:00          |       \_ ps -auxf
andress+  4447  1.1  0.8 1121640 64584 ?       Ssl  16:21   0:20          \_ /opt/sublime_text/sublime_text /home/andressp05/UAM
andress+  4462  0.0  0.2 122104 19492 ?        Sl   16:21   0:01              \_ /opt/sublime_text/plugin_host 4447
colord    1133  0.0  0.1 304132 11120 ?        Ssl  15:00   0:00 /usr/lib/colord/colord
root      1162  0.0  0.0  18456  1772 tty1     Ss+  15:00   0:00 /sbin/agetty --noclear tty1 linux
postgres  1190  0.0  0.3 297372 24388 ?        S    15:00   0:00 /usr/lib/postgresql/9.5/bin/postgres -D /var/lib/postgresql/9.5
postgres  1209  0.0  0.0 297372  3900 ?        Ss   15:00   0:00  \_ postgres: checkpointer process
postgres  1210  0.0  0.0 297372  5700 ?        Ss   15:00   0:00  \_ postgres: writer process
postgres  1211  0.0  0.0 297372  3900 ?        Ss   15:00   0:00  \_ postgres: wal writer process
postgres  1212  0.0  0.0 297812  6440 ?        Ss   15:00   0:00  \_ postgres: autovacuum launcher process
postgres  1213  0.0  0.0 152356  3148 ?        Ss   15:00   0:00  \_ postgres: stats collector process
root      1198  0.0  0.0  44156  6724 ?        Ss   15:00   0:01 /sbin/wpa_supplicant -u -s -O /run/wpa_supplicant
rtkit     1575  0.0  0.0 183544  3012 ?        SNsl 15:01   0:00 /usr/lib/rtkit/rtkit-daemon
root      1584  0.0  0.1 350076 10496 ?        Ssl  15:01   0:01 /usr/lib/upower/upowerd
andress+  1627  0.0  0.0  45376  5148 ?        Ss   15:01   0:00 /lib/systemd/systemd --user
andress+  1629  0.0  0.0 145452  2200 ?        S    15:01   0:00  \_ (sd-pam)
andress+  1649  0.0  0.1 209780  9032 ?        SLl  15:01   0:01 /usr/bin/gnome-keyring-daemon --daemonize --login
root      2082  0.0  0.1 367824  8884 ?        Ssl  15:01   0:01 /usr/lib/udisks2/udisksd --no-debug
andressp05@andressp05-X556UJ:~$ _
```

Los ficheros ejercicio4ai.c y ejercicio4bi.c contienen los códigos dados en el enunciado. En el fichero ejercicio4aii.c y ejercicio4bii.c, hemos añadido las modificaciones pedidas, incluyendo una llamada a la función getpid() y otra a la función getppid() para obtener el pid de un hijo y el pid de su padre y posteriormente, imprimirlos por pantalla. Además, hemos añadido una llamada a la función getchar() para ejecutar el comando pstree en otra terminal aparte y poder observar con detenimiento el árbol resultante. El código fuente del segundo apartado difiere del primero en que se realiza una llamada a la función wait antes de terminar. Esto provoca que un padre no termina hasta que uno de sus hijos acaba. Sin embargo, todavía pueden quedar procesos huérfanos, dado que, en cuanto acaba uno de los tres hijos que tiene el proceso inicial, el padre acaba sin esperar a los otros dos, los cuales quedan huérfanos. En el primero, dado que no se realiza ningún wait, es muy probable que queden procesos huérfanos.

A continuación, se muestran las salidas de los ejecutables ejercicio4aii y ejercicio4bii:



```
andressp05@andressp05-X556UJ:~/UAM/SOPER/Soper/Practica 1$ ./Ejercicio4aii
PADRE 4837
PADRE 4837 del HIJO 4838
e
PADRE 4837
PADRE 4837
PADRE 4837 del HIJO 4839
PADRE 4839
PADRE 4839 del HIJO 4841
PADRE 4837 del HIJO 4840
e
PADRE 4838
PADRE 4838
PADRE 4838 del HIJO 4843
PADRE 4838 del HIJO 4842
PADRE 4842
PADRE 4842 del HIJO 4844
```



```
andressp05@andressp05-X556UJ:~/UAM/SOPER/Soper/Practica 1$ ./Ejercicio4bii
PADRE 5149
PADRE 5149 del HIJO 5150
e
PADRE 5149
PADRE 5149
PADRE 5149 del HIJO 5151
PADRE 5151
PADRE 5151 del HIJO 5153
PADRE 5149 del HIJO 5152
e
PADRE 5150
PADRE 5150
PADRE 5150 del HIJO 5155
PADRE 5150 del HIJO 5156
PADRE 5155
PADRE 5155 del HIJO 5157
```

Andrés Salas Peña
Antonio Martín Masuda

Grupo 2202
Pareja 3

Ejercicio 5:

En este ejercicio se pedía modificar el anterior para que los procesos se generasen de forma secuencial en el primer apartado y en el segundo, que el padre genere un conjunto de hijos. En ambos casos, el cambio introducido fue cambiar el bloque donde se realiza la llamada a la función fork() y se comprueba que ha funcionado correctamente. En el primero, este bloque únicamente la ejecuta el hijo, es decir, si el retorno del fork() vale cero. En el segundo, este bloque únicamente la ejecuta el padre, es decir, si el retorno del fork() es un número positivo no nulo. En ambos casos, se realiza una llamada a la función wait(NULL) para que cada proceso padre espere a sus hijos. Además, se ha añadido la sentencia getchar() para poder observar el árbol generado por cada programa y comprobar que no hay procesos huérfanos.

Árbol vinculado al ejercicio5a.c:

```
homerjr:~/workspace $ ps -auxf
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0   4056   188 ?        Ss+  08:45   0:00 /mnt/shared/sbin/micro-inetd 22 /mnt/shared/
root         269  0.0  0.0  19380  2180 ?        Ss   08:45   0:00 /mnt/shared/sbin/dropbear -i -s -m -R
ubuntu       270  0.0  0.1 1258336 55784 ?       Sl   08:45   0:02  \_ vfs-worker {"pingInterval":5000,"nodePat
ubuntu       726  0.0  0.0 123740  2552 pts/0    Ss+  08:45   0:00      \_ /mnt/shared/sbin/tmux -u2 -L cloud92
ubuntu      1853  0.0  0.0 123740  2560 pts/2    Ss+  09:30   0:00      \_ /mnt/shared/sbin/tmux -u2 -L cloud92
ubuntu       729  0.0  0.0 132676  3480 ?        Ss   08:45   0:00 /mnt/shared/sbin/tmux -u2 -L cloud92.2 new -
ubuntu       730  0.0  0.0  11264  2600 pts/1    Ss   08:45   0:00  \_ bash -c export ISOUTPUTPANE=0;bash -l
ubuntu       731  0.0  0.0  29084 12836 pts/1    S    08:45   0:00  |   \_ bash -l
ubuntu      1844  0.0  0.0   4200   676 pts/1    S+   09:29   0:00  |       \_ ./ejercicio5a
ubuntu      1845  0.0  0.0   4200    88 pts/1    S+   09:29   0:00  |           \_ ./ejercicio5a
ubuntu      1846  0.0  0.0   4200    92 pts/1    S+   09:29   0:00  |               \_ ./ejercicio5a
ubuntu      1847  0.0  0.0   4204    92 pts/1    S+   09:29   0:00  |                   \_ ./ejercicio5a
ubuntu      1855  0.0  0.0  11264  2632 pts/3    Ss   09:30   0:00  \_ bash -c export ISOUTPUTPANE=0;bash -l
ubuntu      1856  3.0  0.0  29040 12720 pts/3    S    09:30   0:00      \_ bash -l
ubuntu      2318  0.0  0.0  17252  2392 pts/3    R+   09:30   0:00          \_ ps -auxf
homerjr:~/workspace $
```

Árbol vinculado al ejercio5b.c:

```
homerjr:~/workspace $ ps -auxf
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0   4056   188 ?        Ss+  08:45   0:00 /mnt/shared/sbin/micro-inetd 22 /mnt/shared/sb
root         269  0.0  0.0  19380  2180 ?        Ss   08:45   0:00 /mnt/shared/sbin/dropbear -i -s -m -R
ubuntu       270  0.0  0.0 1252192 47504 ?       Sl   08:45   0:02  \_ vfs-worker {"pingInterval":5000,"nodePath"
ubuntu       726  0.0  0.0 123740  2552 pts/0    Ss+  08:45   0:00      \_ /mnt/shared/sbin/tmux -u2 -L cloud92.2
ubuntu      1853  0.0  0.0 123740  2560 pts/2    Ss+  09:30   0:00      \_ /mnt/shared/sbin/tmux -u2 -L cloud92.2
ubuntu       729  0.0  0.0 132676  3480 ?        Ss   08:45   0:00 /mnt/shared/sbin/tmux -u2 -L cloud92.2 new -s
ubuntu       730  0.0  0.0  11264  2600 pts/1    Ss   08:45   0:00  \_ bash -c export ISOUTPUTPANE=0;bash -l
ubuntu       731  0.0  0.0  29084 12836 pts/1    S    08:45   0:00  |   \_ bash -l
ubuntu      2432  0.0  0.0   4200   776 pts/1    S+   09:33   0:00  |       \_ ./ejercicio5b
ubuntu      2433  0.0  0.0   4204    88 pts/1    S+   09:33   0:00  |           \_ ./ejercicio5b
ubuntu      2434  0.0  0.0   4204    92 pts/1    S+   09:33   0:00  |           \_ ./ejercicio5b
ubuntu      2435  0.0  0.0   4204    92 pts/1    S+   09:33   0:00  |           \_ ./ejercicio5b
ubuntu      1855  0.0  0.0  11264  2632 pts/3    Ss   09:30   0:00  \_ bash -c export ISOUTPUTPANE=0;bash -l
ubuntu      1856  0.1  0.0  29040 12720 pts/3    S    09:30   0:00      \_ bash -l
ubuntu      2437  0.0  0.0  17252  2384 pts/3    R+   09:33   0:00          \_ ps -auxf
homerjr:~/workspace $
```

Salidas de ambos ejercicios:

```
homerjr:~/workspace/soper/Practica 1 $ ./ejercicio5a
PID PADRE: 731   PID HIJO: 1844
PID PADRE 1844
PID PADRE 1844
PID PADRE: 1844  PID HIJO: 1845
PID PADRE: 1844  PID HIJO: 1845
PID PADRE 1845
PID PADRE: 1845  PID HIJO: 1846
PID PADRE: 1845  PID HIJO: 1846
PID PADRE: 1846  PID HIJO: 1847
```

```
homerjr:~/workspace/soper/Practica 1 $ ./ejercicio5b
PADRE 0: PID 2432
PADRE 0: PID 2433
HIJO 1: PID PADRE: 2432  PID HIJO: 2433
HIJO 2: PID PADRE: 2432  PID HIJO: 2433
PADRE 1: PID 2434
PADRE 1: PID 2432
HIJO 2: PID PADRE: 2432  PID HIJO: 2434
PADRE 2: PID 2432
PADRE 2: PID 2435
```

Ejercicio 6:

En este ejercicio se pedía comprobar el uso de la memoria y los datos después de hacer una llamada a la función fork(). Como se puede observar en la salida, al reservar memoria dinámica para una cadena de caracteres en el proceso padre, también hay que liberarla en el hijo. Sin embargo, al pedir desde el proceso hijo al usuario que introduzca su nombre e insertarlo en esta cadena reservada, el padre no tiene acceso al valor introducido. Esto se debe a que al realizar el fork, la memoria se copia. De esta manera, cada proceso tiene una zona de memoria distinta y el proceso hijo inserta el dato en su zona de memoria y no en la del padre.

Salida del ejercicio: se ha usado la herramienta valgrind para comprobar que, como se esperaba, hay que liberar la cadena reservada en ambos procesos.

```
homerjr:~/workspace/soper/Practica 1 $ valgrind --leak-check=full ./ejercicio6
==2464== Memcheck, a memory error detector
==2464== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2464== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2464== Command: ./ejercicio6
==2464==
Introduce una cadena
Antonio

Antonio

==2465==
==2465== HEAP SUMMARY:
==2465==     in use at exit: 0 bytes in 0 blocks
==2465==   total heap usage: 1 allocs, 1 frees, 80 bytes allocated
==2465==
==2465== All heap blocks were freed -- no leaks are possible
==2465==
==2465== For counts of detected and suppressed errors, rerun with: -v
==2465== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==2464==
==2464== HEAP SUMMARY:
==2464==     in use at exit: 0 bytes in 0 blocks
==2464==   total heap usage: 1 allocs, 1 frees, 80 bytes allocated
==2464==
==2464== All heap blocks were freed -- no leaks are possible
==2464==
==2464== For counts of detected and suppressed errors, rerun with: -v
==2464== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
homerjr:~/workspace/soper/Practica 1 $
```

Ejercicio 8:

En este ejercicio se pedía realizar un programa en el que un proceso padre crease una cierta cantidad de procesos hijo según el número de argumentos de entrada, de forma que cada uno ejecutase un comando distinto introducido por el usuario. Primero, se comprueba que el número de argumentos es como mínimo tres: el nombre del ejecutable, los comandos a ejecutar y –l, -lp, -v o –vp para utilizar las funciones execl, execlp, execv o execvp respectivamente. Para que funcione correctamente, se realiza un fork() en cada iteración del bucle. El proceso padre espera a que finalice su hijo para poder pasar a la siguiente iteración. El hijo también realiza esta sentencia, pero al no tener ningún hijo, no surte ningún efecto y continúa ejecutándose. En el proceso hijo, se comprueba el último argumento de entrada. Si no es ninguno de los mencionados anteriormente, se devuelve un mensaje de error y termina su ejecución. En caso contrario, se ejecuta la llamada a la función correspondiente. En caso de que esta se ejecute correctamente, el proceso finaliza y si no, se imprime un mensaje de error y termina con exit(EXIT_FAILURE). Este bucle se realiza hasta que no quedan más comandos por ejecutarse.

Salida del ejercicio:

```
homerjr:~/workspace/soper/Practica 1 $ ./ejercicio8 /bin/ls /bin/df /usr/bin/du -l
Ej4i.c   Enunciado.pdf    ejercicio4ai.o   ejercicio4bi    ejercicio4bii.c  ejercicio5a.o  ejercicio6    ejercicio8.c  ejercicio9.o
Ej4ii.c  G2202_P03_1.tgz  ejercicio4aii    ejercicio4bi.c  ejercicio4bii.o  ejercicio5b    ejercicio6.c  ejercicio8.o  makefile
Ej5a.c   ejercicio4ai     ejercicio4aii.c  ejercicio4bi.o  ejercicio5a      ejercicio5b.c  ejercicio6.o  ejercicio9
Ej5b.c   ejercicio4ai.c   ejercicio4aii.o  ejercicio4bii   ejercicio5a.c    ejercicio5b.o  ejercicio8    ejercicio9.c
Filesystem                1K-blocks      Used Available Use% Mounted on
none                        2234668    605076   1494476  29% /
tmpfs                      26797620         0  26797620   0% /dev
tmpfs                      26797620         0  26797620   0% /sys/fs/cgroup
/dev/mapper/volg1-lvdata 1238412656 748027788 490368484  61% /nix
shm                           65536         0     65536   0% /dev/shm
1356    .
homerjr:~/workspace/soper/Practica 1 $
```

Ejercicio 9:

En este ejercicio se pedía implementar un programa que crease cuatro hijos, cada uno realiza una operación aritmética distinta con unos operandos pasados por el padre a través de una tubería y el resultado se devuelve a través de otra tubería. Al principio, el proceso padre pide al usuario que introduzca dos números enteros. Posteriormente, se ejecuta un bucle en el que primero se resetean dos buffers donde se escribirá la información. Esto es necesario porque si no, cabe el riesgo de que se impriman caracteres indeseados. Después, se crean dos tuberías que serán unidireccionales. El padre escribe los operandos en una de ellas y espera a que finalice de ejecutarse el hijo. Dicho hijo leerá los operandos y ejecutará la operación aritmética correspondiente (suma, resta, multiplicación y división en ese orden). Si el número de bytes leídos es cero, se imprimirá un mensaje de error y retorna con exit(EXIT_FAILURE). Si la lectura ha sido correcta, escribe en la otra tubería la información de su identificador de proceso, los operandos leídos, la operación realizada y el resultado de la misma. En caso de la división, también se comprueba que el segundo operando no sea nulo. En caso de serlo, simplemente se escribe en la tubería que no se puede dividir por cero. Finalmente, el padre lee lo que ha escrito el hijo y si la lectura ha sido correcta, lo imprime por pantalla y si no, se imprime un mensaje de error.

En un primer momento, se decidió realizar la implementación del ejercicio con una tubería bidimensional, a nivel de codificación la única modificación respecto a las dos tuberías unidimensionales es que no se tiene que cerrar ninguno de los dos extremos de la tubería ya que se usan ambos. Después, tras comentárselo al profesor, se modificó a esta versión final de dos tuberías unidimensionales.

Salida del ejercicio:

```
homerjr:~/workspace/soper/Practica 1 $ ./ejercicio9
Introduce el primer numero: 5
Introduce el segundo numero: 6
Datos enviados a través de la tubería por el proceso PID=2507. Operando 1: 5. Operando 2: 6. Suma: 11.
Datos enviados a través de la tubería por el proceso PID=2508. Operando 1: 5. Operando 2: 6. Resta: -1.
Datos enviados a través de la tubería por el proceso PID=2509. Operando 1: 5. Operando 2: 6. Multiplicacion: 30.
Datos enviados a través de la tubería por el proceso PID=2510. Operando 1: 5. Operando 2: 6. Division: 0.
homerjr:~/workspace/soper/Practica 1 $
```