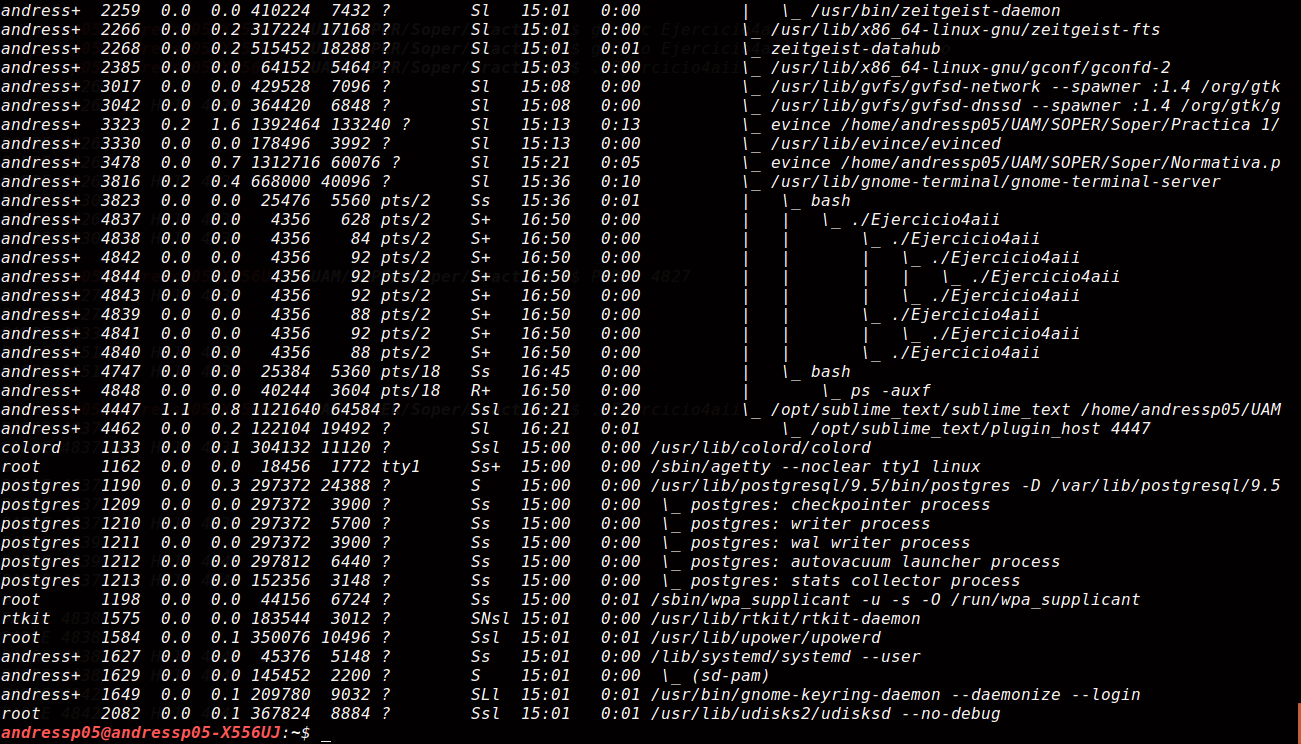
**SISTEMAS OPERATIVOS: Informe práctica 1**

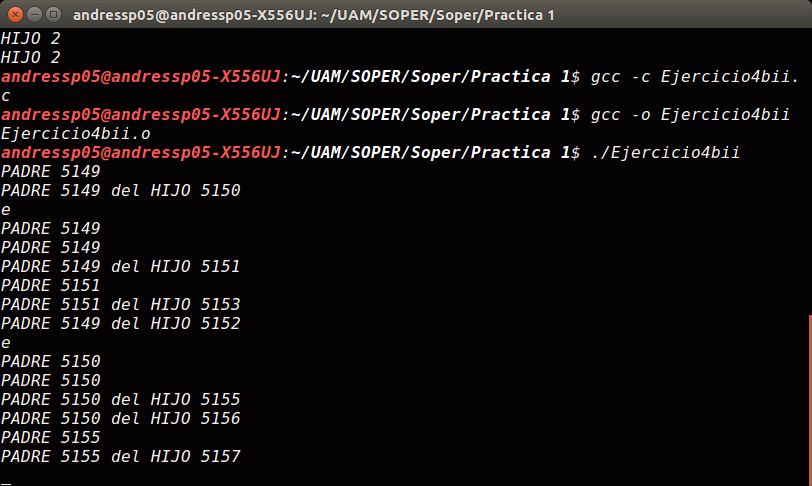
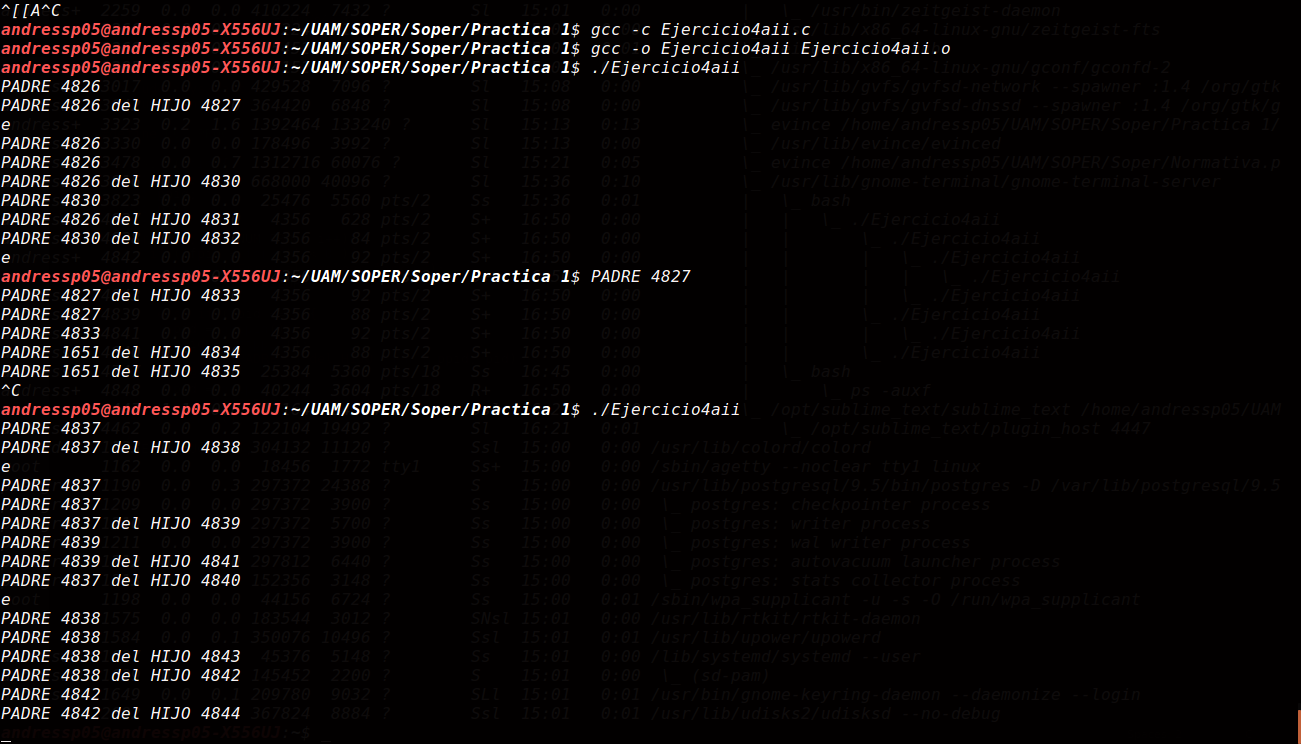
**Ejercicio 4:**

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’.



Árbol vinculado

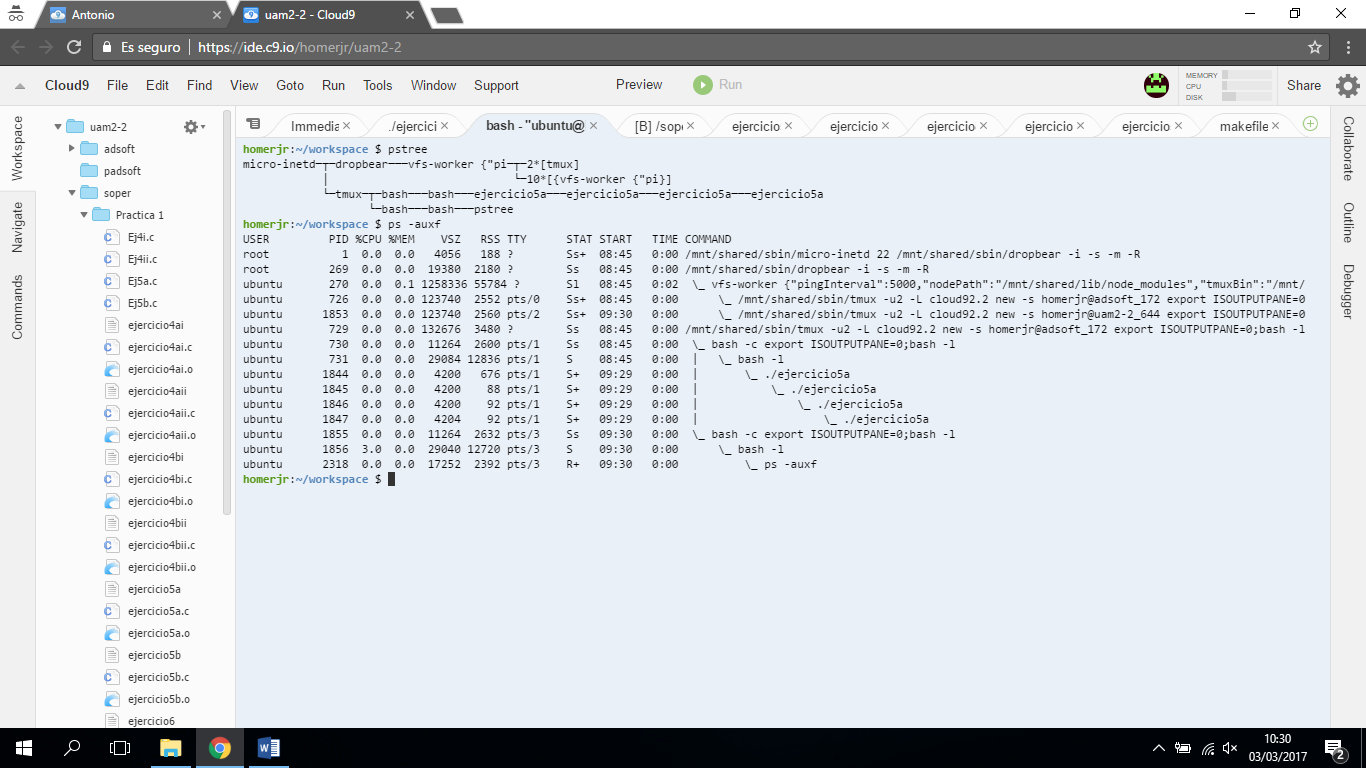
Los ficheros ejercicio4ai.c y ejercicio4bi.c contienen los códigos dados en el enunciado. En el fichero ejercicio4aii.c y ejercicio4bii.c, hemos añadido las modificaciones pedidas, incluyendo una llamada a la función getpid() y otra a la función getppid() para obtener el pid de un hijo y el pid de su 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:

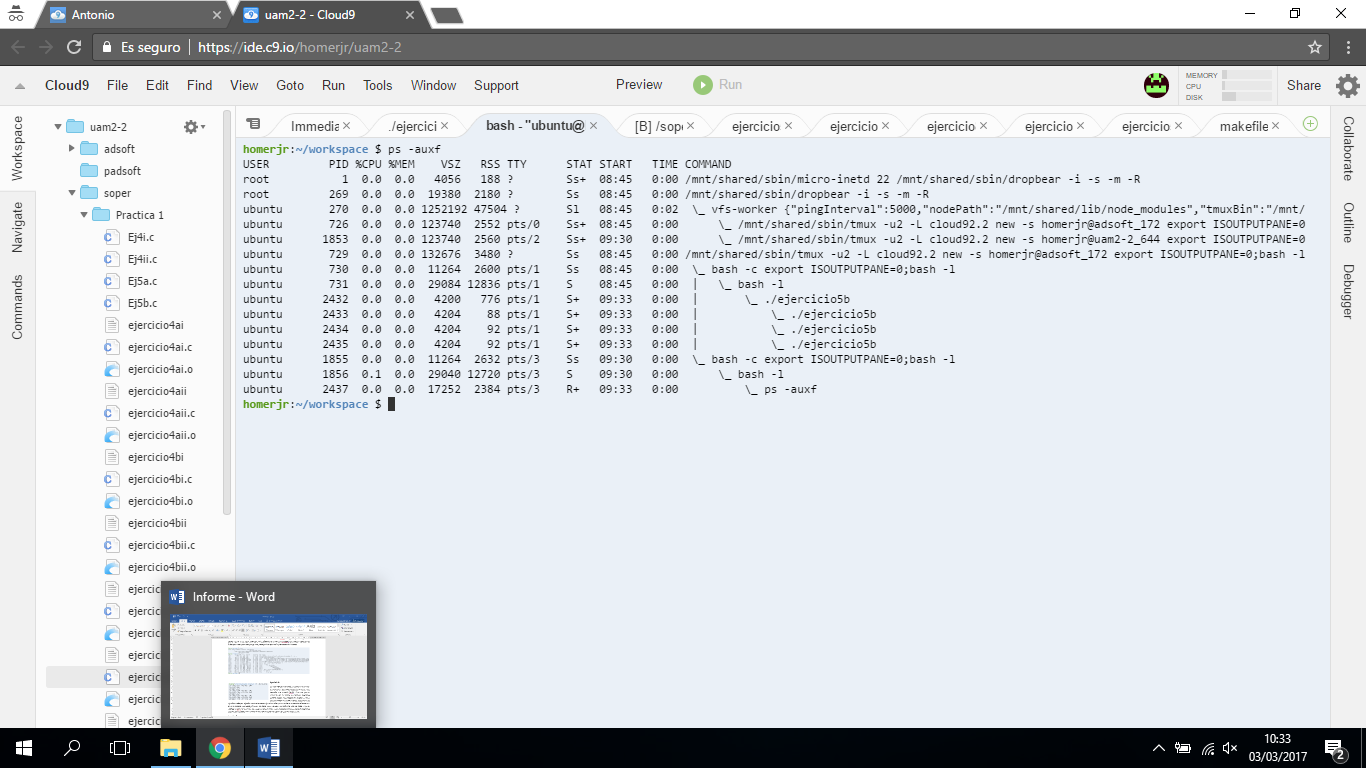
**Ejercicio 5:**

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.

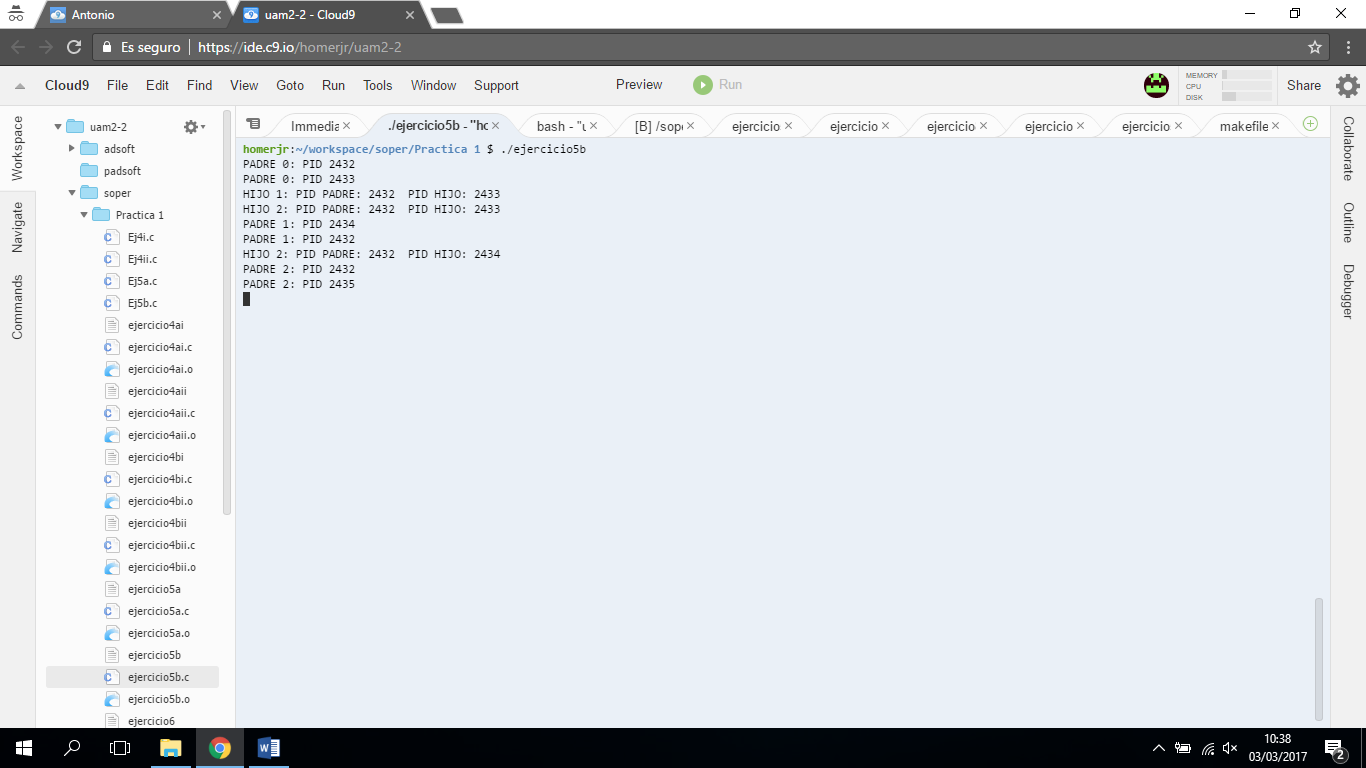
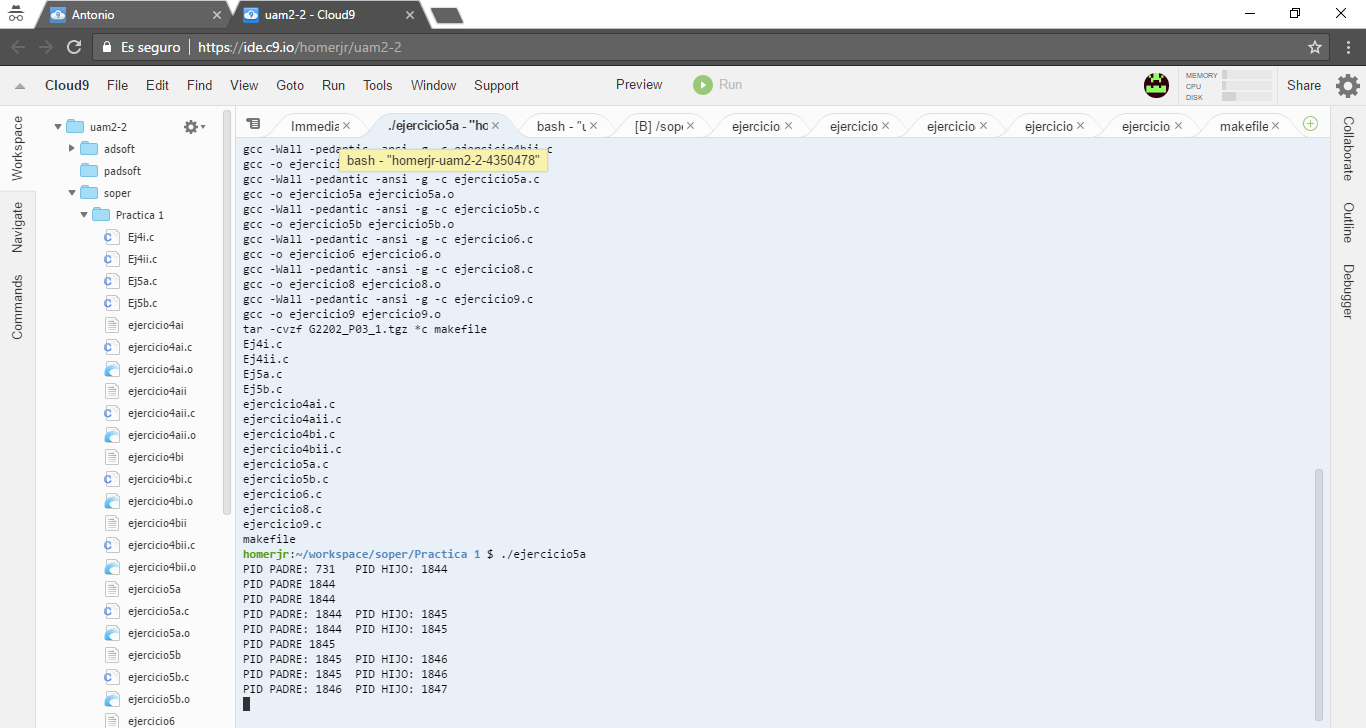
Árbol vinculado al ejercicio5a.c:



Árbol vinculado al ejercio5b.c:



Salidas de ambos ejercicios:

****

**Ejercicio 6:**

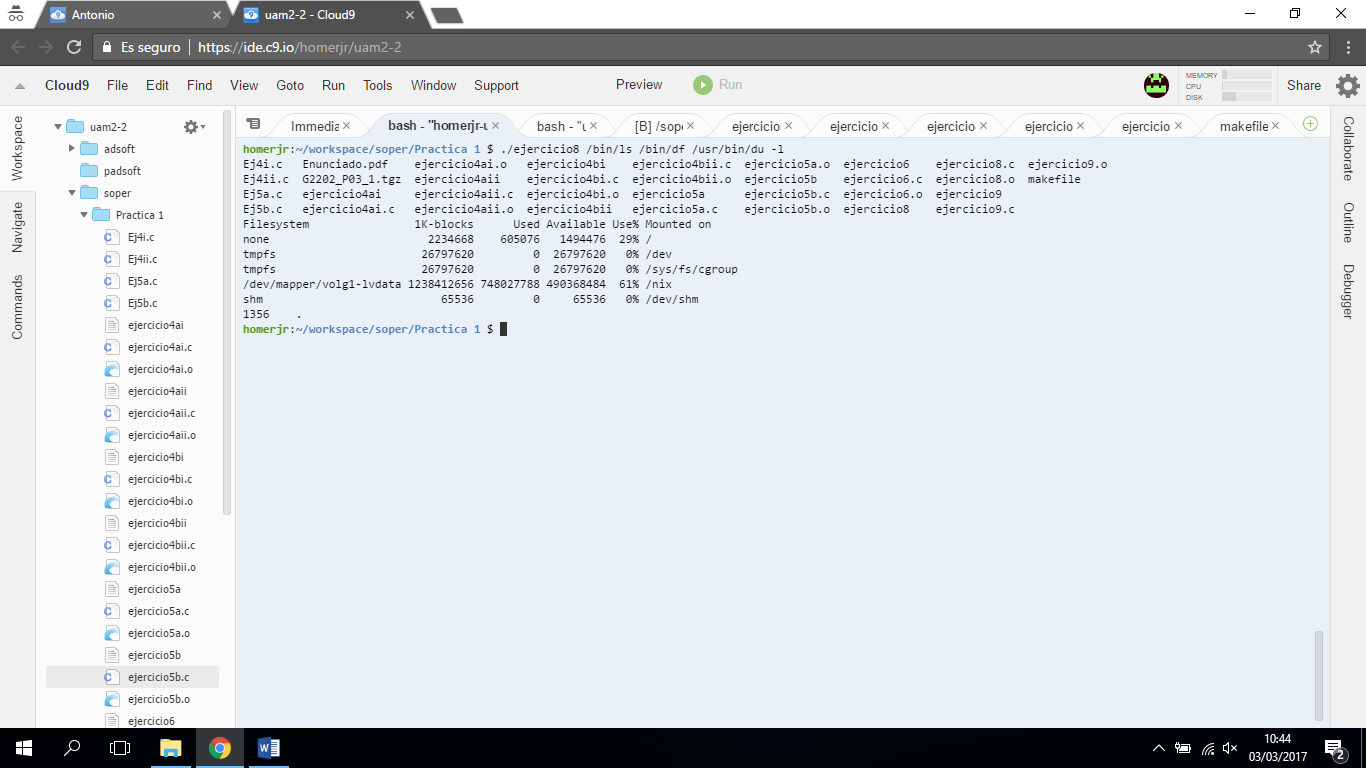
En este ejercicio se pedía comprobar el uso de la memoria y los datos después de hacer una llamada a la función fork(). Como se puede observar en la salida, al reservar memoria dinámica para una cadena de caracteres en el proceso padre, también hay que liberarla en el hijo. Sin embargo, al pedir desde 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.

**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.

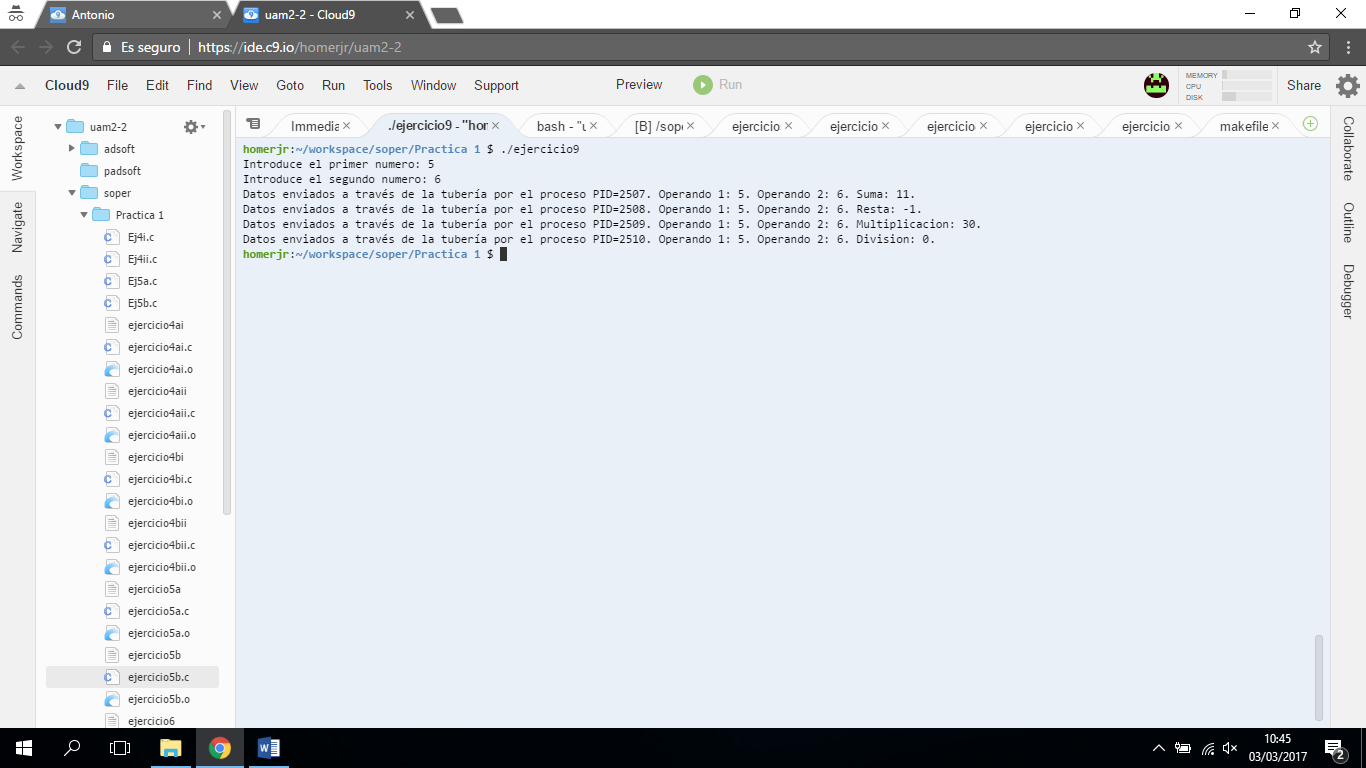
Salida del ejercicio:



**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: