HW2 REPORT

```
serdar@serdar-pc:~/Desktop/ASD$ gcc -o main main.c
serdar@serdar-pc:~/Desktop/ASD$ ./main
Enter the size of the array: 10
created fifo1
created fifo2
The array is: [ 10 8 10 8 1 7 2 2 3 10 ]
CHILD 2: opening FIFO2 to read the array
CHILD 1: open FIFO1 to read the rray
CHILD 2: succesfully read integers from FIFO2
CHILD 2: reading the command from FIFO2
CHILD 2: succesfully read command from FIFO2, command is: multiply
CHILD 2: reading the sum from FIFO2
Proceeding...
CHILD 1: succesfully read integers from FIFO1 and the sum is: 61
CHILD 1: opening FIFO2 to write the sum
CHILD 1: succesfully wrote the sum into FIFO2
CHILD 2: succesfully read the sum from FIFO2
sum: 61
mult: 5376000

Child process with PID 9931 exited with status 0
Child process with PID 9930 exited with status 0
serdar@serdar-pc:~/Desktop/ASD$
```

Steps:

1) Parent: creates fifos

2) Parent: generates and prints random array (range: [1, 10])

3) Parent: writes arrays into fifos

3) Child 2: opens fifo2, reads array

4) Child 1: opens fifo1, reads the array

5) Child 2: reads command from fifo2 and prints it

6) Child 2: multiplies the integers of the array

7) Child 2: reopens the read end of fifo2 to read the sum which child process 1 writes into fifo2

8) Child 1: after reading the array, adds the integers of the array

9) Child 1: writes the sum to the write end of fifo2

10) Child 2: After reading the sum, prints sum and multiplication result

11) Parent: waits for child processes to end, if it takes forever then the program is terminated with message of ”Timeout Reached”. In this case fifos will be closed and unlinked, array will be freed. Zombie processes will be killed.

Example of entering different command than multiply:

```
serdar@serdar-pc:~/Desktop/ASD$ gcc -o main main.c
serdar@serdar-pc:~/Desktop/ASD$ ./main
Enter the size of the array: 10
created fifo1
created fifo2
The array is: [ 9 8 10 2 6 5 2 5 1 4 ]
CHILD 2: opening FIFO2 to read the array
CHILD 1: open FIFO1 to read the array
CHILD 2: succesfully read integers from FIFO2
CHILD 2: reading the command from FIFO2
CHILD 2: succesfully read command from FIFO2, command is: multiplyy
Proceeding...
Command is invalid, exiting child 2.
CHILD 1: succesfully read integers from FIFO1 and the sum is: 52
CHILD 1: opening FIFO2 to write the sum
Child process with PID 9420 exited with status 1
Proceeding...
Proceeding...
Proceeding...

Timeout reached. Terminating the program.
Child process with PID 9421 was terminated by signal 9
serdar@serdar-pc:~/Desktop/ASD$
```

When an invalid command is entered, the program terminates child 2.
If we look at the last part, child 1 opens the write-end of FIFO2 but this causes a deadlock because child 2 is terminated and the process of opening the read-end of FIFO2 won‘t be happening ever again. So, child 1 will wait for the read-end of FIFO2 to open forever, and it won‘t terminate. To solve that problem, parent process prints ”proceeding... ” every 2 seconds to show that it is waiting for child processes to terminate and if it takes too long this means something is failing. After waiting some time the program terminates with a message of ”Timeout reached”. Fifos will be closed and unlinked; arrays located in heap will be freed before the termination. If there are zombie processes, they will be terminated with a kill signal and the status will be printed. This timeout only catches zombie processes. Lets prove:

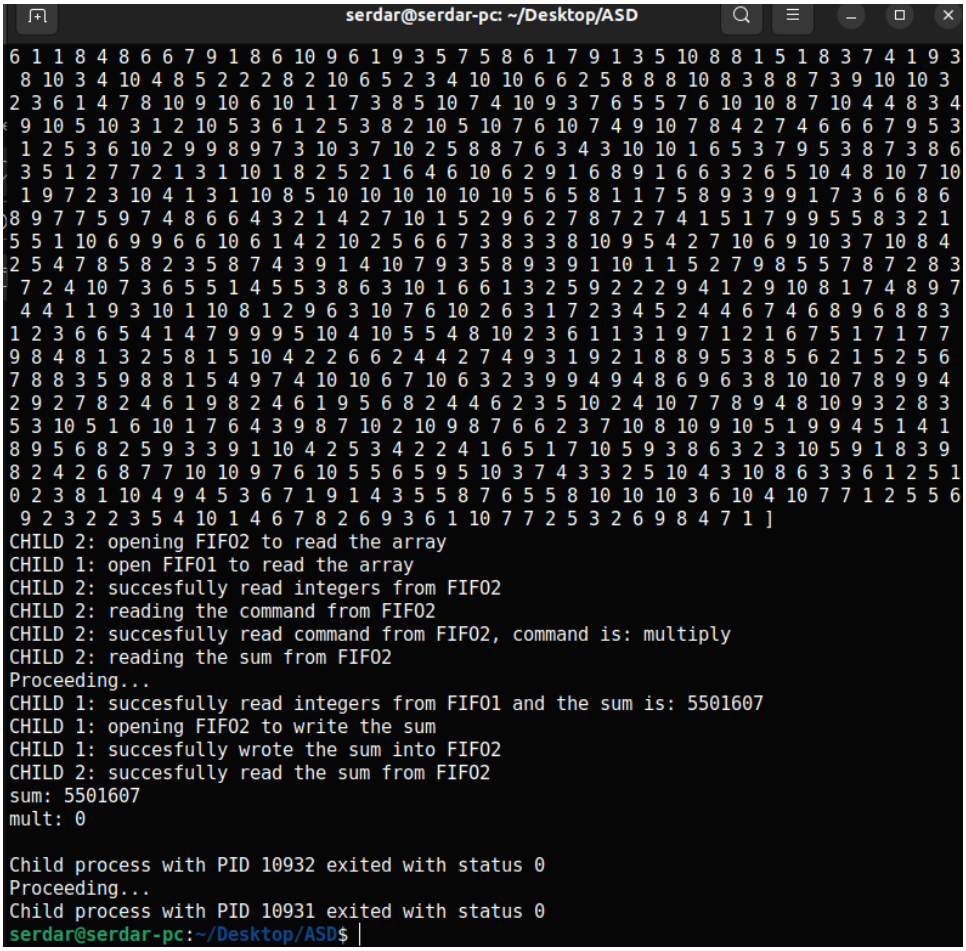
Proof: If timeout range is set to 0 but there is no zombie process

```
Enter the size of the array: 10
created fifo1
created fifo2
The array is: [ 4 2 6 5 6 3 3 2 1 8 ]
CHILD 2: opening FIFO2 to read the array
CHILD 1: open FIFO1 to read the array
CHILD 2: succesfully read integers from FIFO2
CHILD 2: reading the command from FIFO2
CHILD 2: succesfully read command from FIFO2, command is: multiply
CHILD 2: reading the sum from FIFO2
Proceeding...
CHILD 1: succesfully read integers from FIFO1 and the sum is: 40
CHILD 1: opening FIFO2 to write the sum
CHILD 1: succesfully wrote the sum into FIFO2
CHILD 2: succesfully read the sum from FIFO2
sum: 40
mult: 207360

Child process with PID 7700 exited with status 0
Child process with PID 7701 exited with status 0

Timeout reached. Terminating the program.
serdar@serdar-pc:~/Desktop/ASD$
```

Proof: If timeout range is set to 0 and there is a zombie process

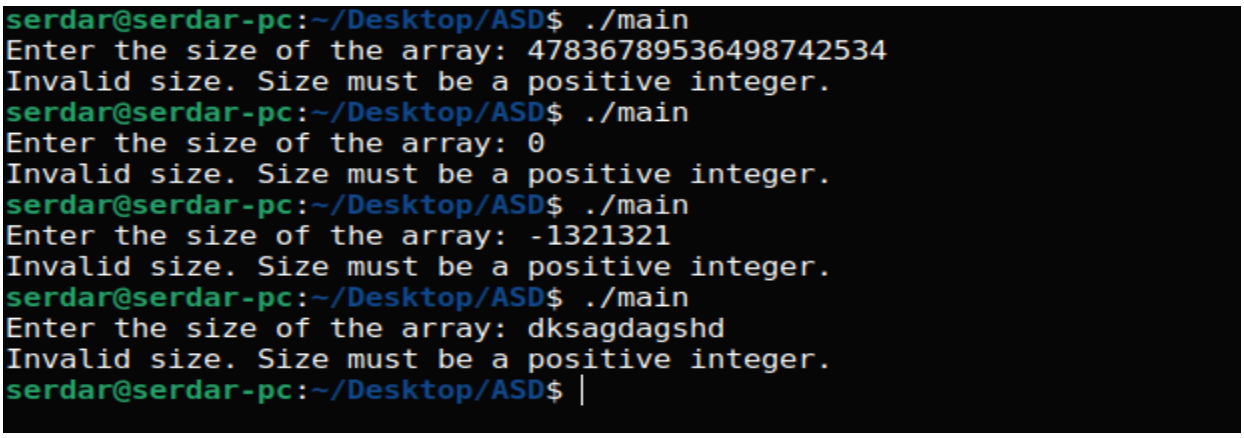
```
serdar@serdar-pc:~/Desktop/ASD$ ./main
Enter the size of the array: 10
created fifo1
created fifo2
The array is: [ 10 5 8 9 8 4 10 9 10 10 ]
CHILD 2: opening FIFO2 to read the array
CHILD 1: open FIFO1 to read the array
CHILD 2: succesfully read integers from FIFO2
CHILD 2: reading the command from FIFO2
CHILD 2: succesfully read command from FIFO2, command is: multiplyy
Command is invalid, exiting child 2.
Proceeding...
CHILD 1: succesfully read integers from FIFO1 and the sum is: 83
CHILD 1: opening FIFO2 to write the sum
Child process with PID 8451 exited with status 1

Timeout reached. Terminating the program.
Child process with PID 8453 was terminated by signal 9
serdar@serdar-pc:~/Desktop/ASD$
```

Example of entering a very big sized array: 1.000.000 integers



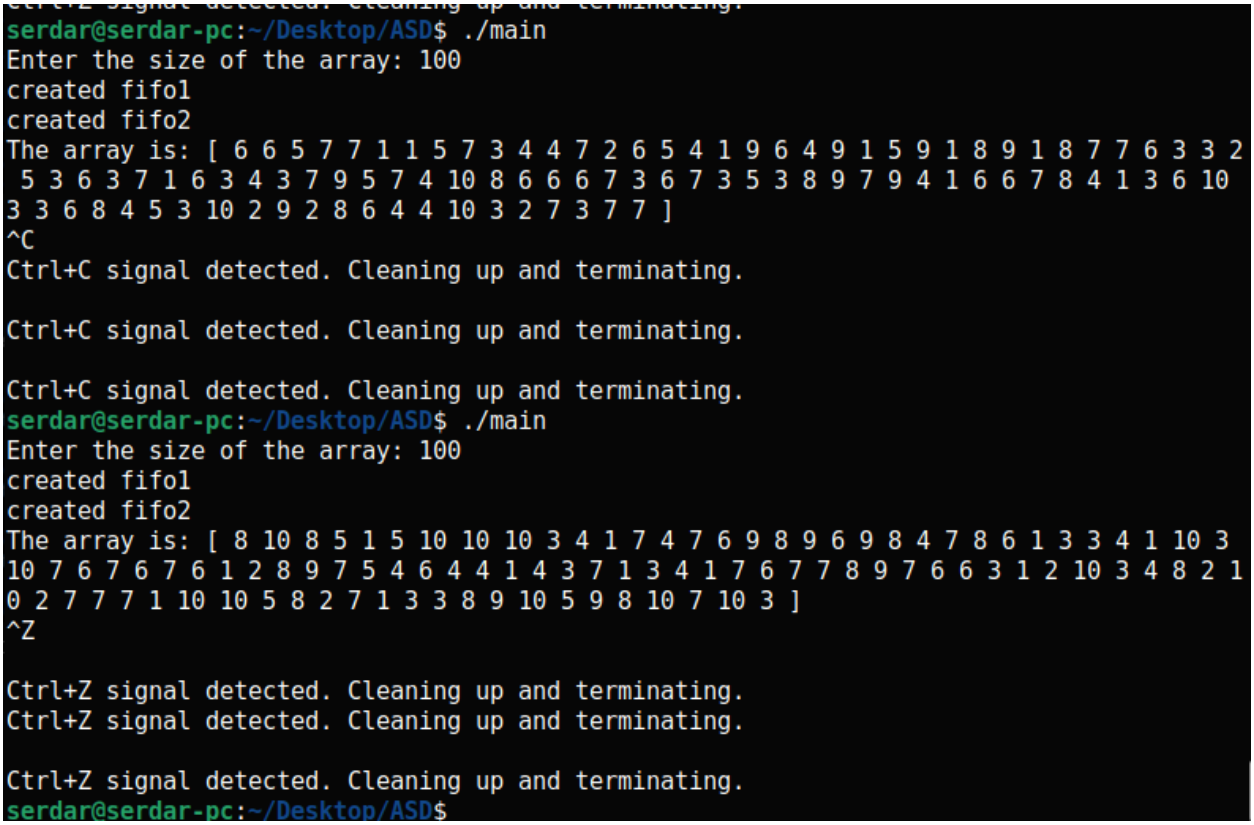
Here the multiplication result is shown as 0 because the result is too big to be an integer.

Example of entering invalid array sizes



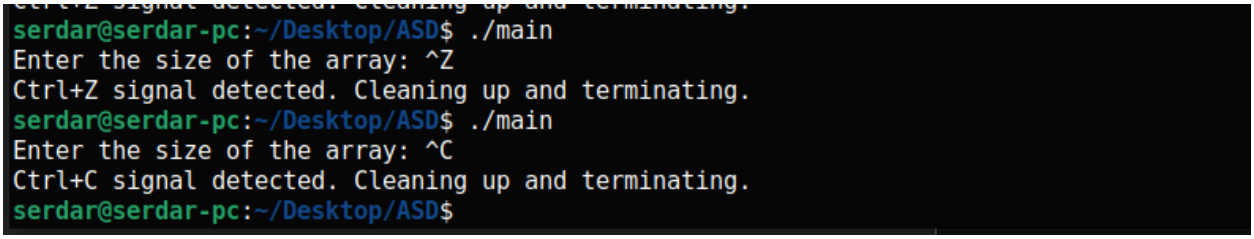
The program only accepts positive integers as array size

Example of ctrl+z/ ctrl+c after child processes are created with fork



As you can see all parent and child processes are terminated after SIGTSTP or SIGINT signals. The signal handler I implemented prints these messages and unlinks fifos.

Example of ctrl+z/ ctrl+c before child processes are created with fork



Compiling:

make: compiles and runs the program

make clean: cleans unnecessary files