GEBZE TECHNICAL UNIVERSITY
CSE344 – HW2 DOCUMENTATION

Student Name: Emre Oytun
Student Number: 200104004099

Important Notes:

1) In some error handling parts, I used “fprintf” to print errors to the stderr instead of “perror” because it was printing “Successful” after printing my error message after a logical check (not after a syscall), therefore errno was not set and this seemed confusing to print this after an error message so I prefered “fprintf” in this kind of errors like below.

In parent process (main function) :

```
int main(int argc, char* argv[]) {
    int result = 0;

    // Take integer number argument
    if (argc != 2) {
        fprintf(stderr, "Integer argument is missing \n");
        parent_exit(EXIT_FAILURE);
    }

    int n = 0;
    if (convert_str_to_int(argv[1], &n) == -1) {
        fprintf(stderr, "Given argument is not an integer \n");
        parent_exit(EXIT_FAILURE);
    }
```

In child2 (child2 function):

```
else {
    // handle wrong command as error
    fprintf(stderr, "Wrong command sent from parent process to fifo2 \n");
    close_fd_fifos();
    _exit(EXIT_FAILURE);
}
```

2) I printed the random generated numbers in parent process to the screen, so that checking if the program works correctly is easy.

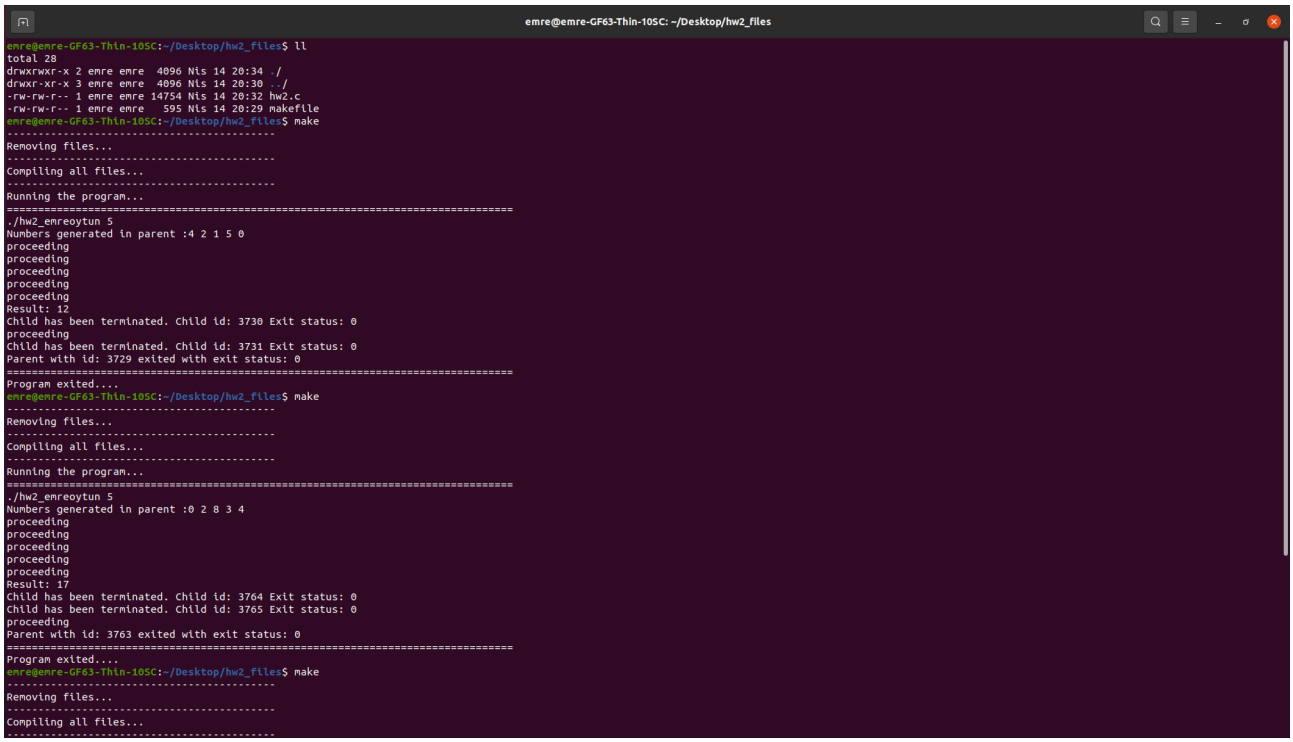
3) In homework PDF, it says we need to open both fifos and write to them before creating the child processes with fork. For this purpose, I complied with the POSIX standarts by opening a fifo firstly with O_RDONLY | O_NONBLOCK, then with O_WRONLY. This complies the POSIX standarts because in man page, it says it is sure that the fifo will be opened without delay successfully when we open a fifo with O_RDONLY | O_NONBLOCK. After that, opening it with O_WRONLY will be successful since read-end has been opened.
The code example is in the screenshot below:

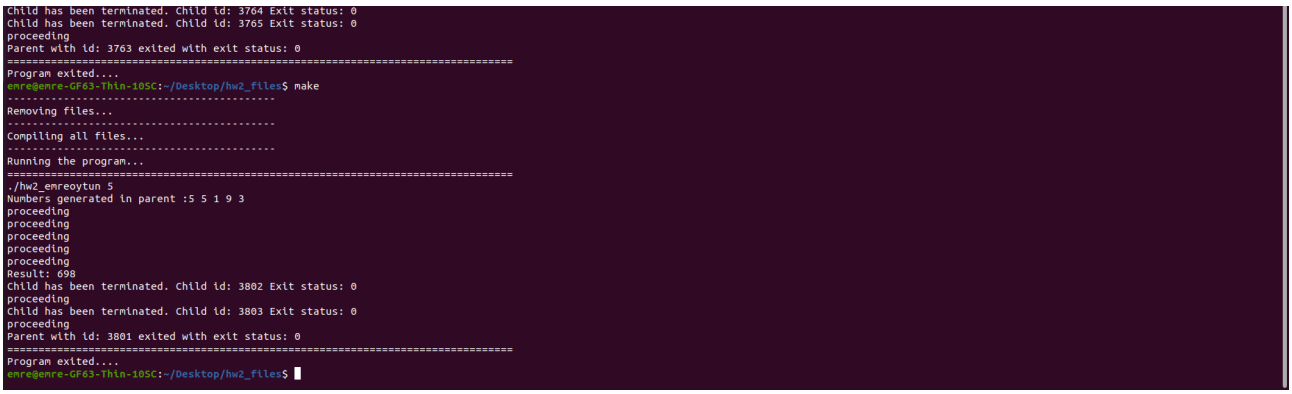
```
fd_fifo1_read = open(fifo1, O_RDONLY | O_NONBLOCK);
if (fd_fifo1_read == -1) {
    perror("Error in opening fifo1");
    unlink_fifos(fifo1, fifo2);
    parent_exit(EXIT_FAILURE);
}

fd_fifo1_write = open(fifo1, O_WRONLY);
if (fd_fifo1_write == -1) {
    perror("Error in opening fifo1");
    close_fd_fifos();
    unlink_fifos(fifo1, fifo2);
    parent_exit(EXIT_FAILURE);
}
```

4) I coded all bonus parts of the homework.
In normal program execution, it prints all children exit statuses and protects from zombie children since parent process waits them.
If parent process terminates with an error or if it is signalled with SIGINT by Ctrl + C or kill signal from terminal, it sends SIGTERM signal to the children processes and waits them to terminate so it prints all exit statuses and protects from zombie children.
If any child process terminates with an error or a signal like SIGINT or SIGKILL, it is detected in SIGCHLD signal handler of parent process and SIGTERM signal is sent to the other child. Parent process still waits for SIGCHLD signal from this child. Therefore, it prints all exit statuses and protects from zombie children.

Screenshots From Program Execution:





As it is seen, the program logic is working successfully. It prints the generated values in the parent for demonstration purposes. The parent goes into the loop for waiting the children processes wile printing “proceeding” in every 2 seconds. After 10 seconds, children start running and calculate the results. The final result in child2 is printed into the screen. Also, the exit statuses of all processes are printed into the screen and children processes are caught to signal handler of SIGCHLD signal.

If we examine the last execution, the generated values are [5, 5, 1, 9, 3]. Their sum is 23 and production is 675. Child2 sums these two results up, finds the result 698 and prints into the screen.

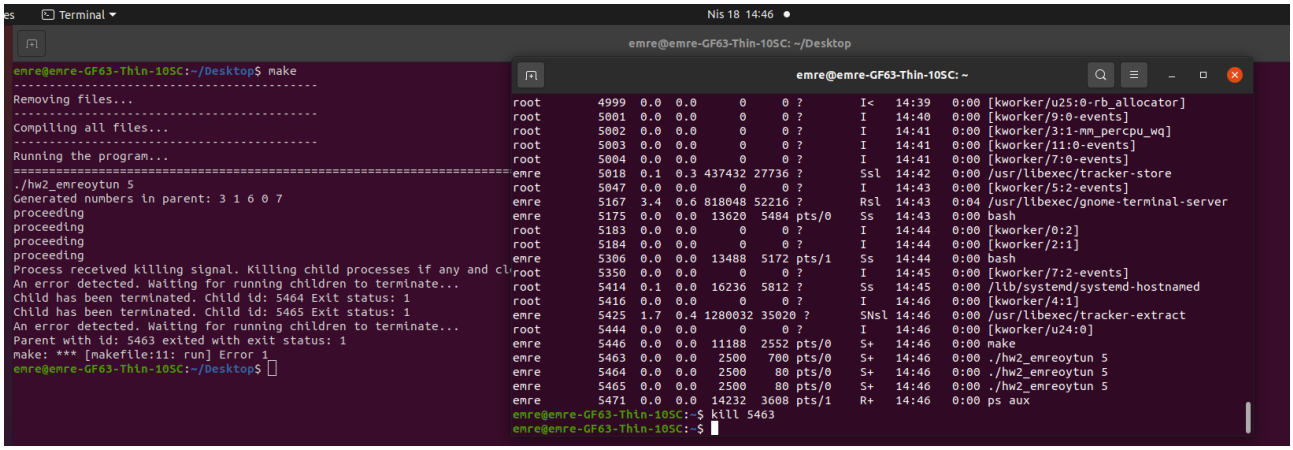
Screenshots From Program Abnormal Termination: You can find the screenshots I mentioned in “Important Notes 4th Part”.

1) Ctrl + C to parent process :

```
emre@emre-GF63-Thin-10SC:~/Desktop$ make
----------------------------------------
Removing files...
----------------------------------------
Compiling all files...
----------------------------------------
Running the program...
================================================================================
./hw2_emreoytun 5
Generated numbers in parent: 1 9 0 1 2
proceeding
proceeding
^CProcess received killing signal. Killing child processes if any and cleaning file descriptors
An error detected. Waiting for running children to terminate...
Child has been terminated. Child id: 5243 Exit status: 1
Child has been terminated. Child id: 5244 Exit status: 1
An error detected. Waiting for running children to terminate...
Parent with id: 5242 exited with exit status: 1
make: *** [makefile:11: run] Error 1

emre@emre-GF63-Thin-10SC:~/Desktop$
```

2) Kill signal to parent process :



3) Kill signal to child process :