CSE 344 - Homework 2 Report

Eren ÇAKAR
1901042656

April 2024

1 Introduction

This report presents the design and implementation details of the IPC communication demonstration system. The application is executed with a positive integer and executes some operations on them in child processes.

2 System Structure

2.1 Main Process

The main processes is the parent process of the two child processes. The main function of the parent process consists of the following sequence of parts:

1. Assert the terminal execution is valid.
2. Assert the terminal execution parameters are valid.
3. Connect child process termination signal handler.
4. Create two pipes.
5. Write a random array of integers to both pipes.
6. Write desired (multiplication) operation in second pipe.
7. Create first process.
8. Create second process.
9. Print "Proceeding" every two seconds while waiting for child processes to terminate.
10. Terminate.

The parent process terminates in failure of any of these steps and prints out the error.

The first child process is responsible with summation of received integers. The sequence of actions it takes is the following:

1. Sleep for 10 seconds.
2. Read an integer from first pipe.
3. Read as many integers as the first read integer value.
4. Sum the read integers.
5. Write the result to second pipe.

Failure in any step results in termination with printing the error. The child processes assumes the first sent integer is he size of the array of integers to be received. Which is agreed upon by the array sending function definition.

The second child process is responsible for computing a result based on the integers received from the first child process and a command string. The sequence of actions it takes is outlined below:

1. Sleep for 10 seconds.
2. Read an integer from the second pipe, representing the size of the array to be received.
3. Read the integers from the second pipe.
4. Read a command string from the second pipe.
5. Check the command string; if it is "multiply", multiply all integers in the array.
6. If the command is not "multiply", print an error message and terminate.
7. Read an additional integer from the second pipe.
8. Add this integer to the computed result.
9. Print the final result to the standard output.

Similar to the first child process, any failure during these steps results in termination with an error message printed.

This child process expects to receive an array size, followed by the corresponding array of integers, and finally, a command string indicating the operation to perform on the array. Which does not have synchronization issues because the parent process forks only after sending the arrays to the pipes.

3 Implementation

In this section, the implementation details of the processes and functions are explored.

3.1 Functions

The program consists of several functions responsible for various tasks such as sending arrays and commands through FIFOs, handling signals, and checking for multiplication commands.

3.1.1 `send_array`

This function sends an array consisting of the first `array_size` positive integers through the given FIFO. The fact that we send only positive integers is important implementation wise which will be mentioned again in following subsections.

```
send_array(fifo, array_size):
    open FIFO for writing
    write array_size to FIFO
    write each element of the array to FIFO
    close FIFO
```

3.1.2 `send_command`

This function sends a command through a FIFO.

```
send_command(fifo, command, command_length):
    open FIFO for writing
    write command to FIFO
    close FIFO
```

3.1.3 `sigchld_handler`

This signal handler function is invoked upon receiving a SIGCHLD signal, indicating the termination of a child process.

```
sigchld_handler(signum):
    while there are terminated child processes:
        get status of terminated child process
        increment child counter
```

3.1.4 `sigint_handler`

This signal handler function is invoked upon receiving a SIGINT signal (Ctrl+C), handling the unlinking of the FIFOs.

```
sigint_handler(signum):
    perform cleanup tasks (unlink FIFOs)
    exit program with failure status
```

3.1.5 `check_string` and `is_multiplication`

These functions are used to check if a given string represents a multiplication command.

3.2 Child Processes

3.2.1 First Child Process

This process calculates the negative sum of the array of integers received from the parent process and sends the result to the second child process through the second FIFO.

```
First Child Process:
    open FIFO1 for reading
    sleep for 10 seconds
    read array size from FIFO1
    read array elements from FIFO1
    calculate negative sum of array elements
    close FIFO1
    open FIFO2 for writing
    send negative sum to Second Child Process through FIFO2
    close FIFO2
    exit successfully
```

Here opening `FIFO1` for reading before sleeping is done to let process successfully write and not be blocked.

3.2.2 Second Child Process

This process calculates the product of an array of integers received from the parent process, subtracts the negative sum received from the first child process, which ultimately equals to adding the sum, and prints the final result.

```
Second Child Process:
    open FIFO2 for reading
    sleep for 10 seconds
    read array size from FIFO2
    read array elements from FIFO2
    read command from FIFO2
    check if command is multiplication
    calculate product of array elements
    read negative sum from FIFO2
    subtract negative sum from product
```

```
print final result
close FIFO2
exit successfully
```

Here we make use of the fact that the sum is sent from the first child process as negative by checking if the received integer is negative or not, which in the case it is we know for certain it is the integer that the first child process sent and not the parent process.

Here opening `FIFO2` for reading before sleeping is done to let process successfully write and not be blocked.

3.3 Parent Process

The parent process orchestrates the execution of child processes, sending data and commands through FIFOs.

4 Examples

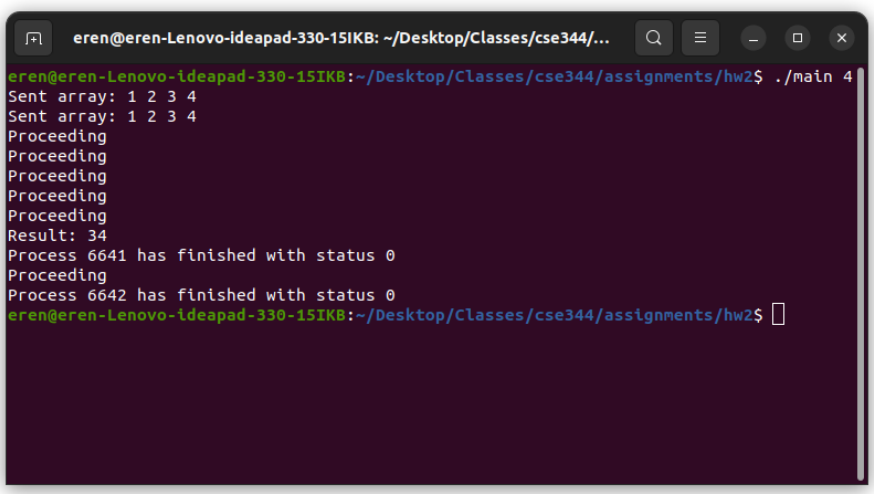

Figure 1: Proper execution

Here the "random" array of integers `1, 2, 3, 4` are sent. Which have a sum of **10** and product of **24**, therefore the result should be and is **34**.

An interesting detail is that parent process prints "Proceeding" in between child process terminations. This is due to the signal interrupt received from the first child, which causes interruption on parent process's sleep and makes it continue on the next iteration of the while loop. But after termination of second

child this does not occur since the printing is done before sleeping in the while loop.

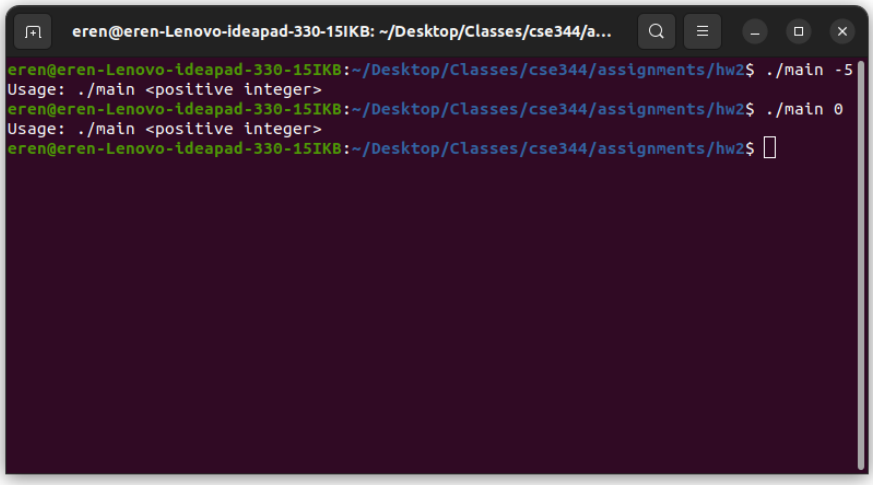

Figure 2: Invalid argument

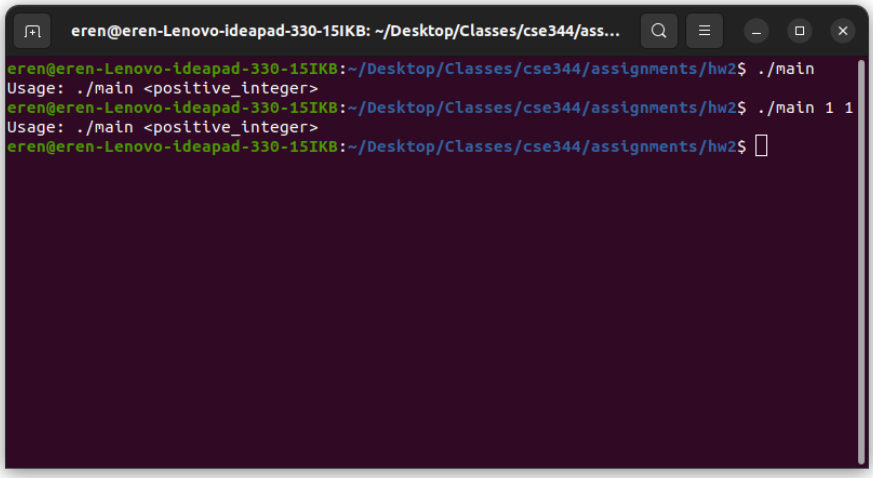

Figure 3: Invalid number of arguments

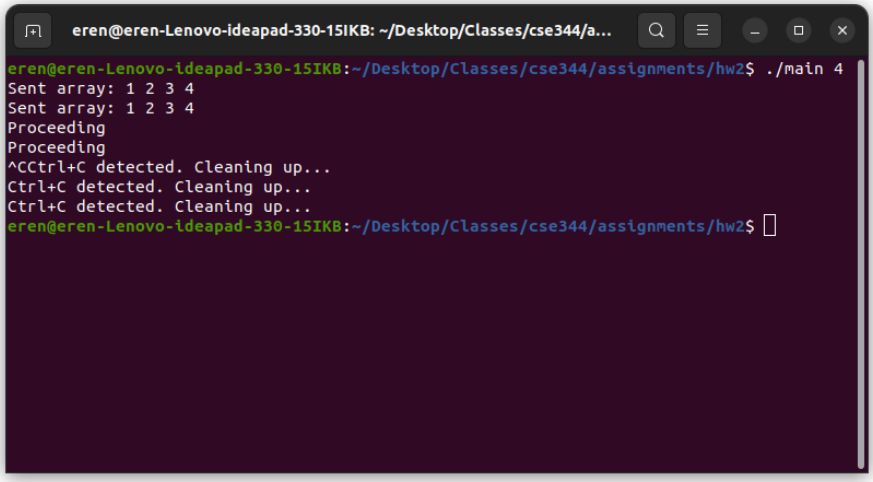

Figure 4: User interrupt

It is seen in Figure 2 and 3 the errors related to an invalid terminal argument. And in Figure 4 `SIGINT` handling is showcased.