GEBZE TECHNICAL UNIVERSITY

CSE 344
SYSTEM PROGRAMMING
Homework #2 Report

OGUZ MUTLU
1801042624

1.Problem Definition

This assignment requires creating a communication protocol between two child process spawned by a parent process using interprocess communication techniques for example FIFO(named pipes)

-Accept integer as an argument

-Create 2 FIFO communication

-Generate random integer array and generate some operation on it

- Summation operation in child1

-Multiplication operation in child2

-Set signal handler for SIGCHLD

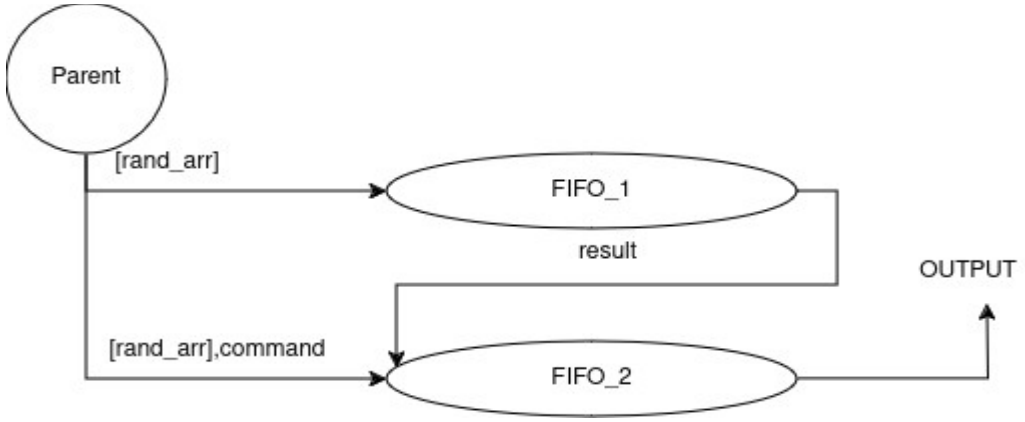
-use waitpid in signal handler and print process id

-Terminate correctly

Implement error handling for potential issues like fifo creation failures, reading writing syscall error or invalid args.

In summary system is considered successful if it can do some operation on random array, prints it and successfully terminated from parent and child processes.

2.Implementation



The homework flow is as follows. First of all, fifo files are created with mkfifo and after the necessary error checking operations are carried out, the fork operation is performed. During the fork process, while continuing in the parent process section (the default part of the switch case structure), a random array is created as much as the entered argument and the first fifo is opened wıth WRONLY. Here the fifo is blocked because, according to the man page of the fifo file, both ends must be open.

Since the flow continues in the child process, this time the fifo file was opened in the child process with RDONLY in order to receive the array coming from the parent.

When writing from the parent to the first fifo, the arrays are converted from integer to string. Since the number of child processes will then increase to two and two different processes will write to the same file, a comma is used if writing is done from the main process for parsing, and an exclamation is used if the process is child. For example;

*randArr[0],randArr[1],randArr[2], → written into **fifo1** from **parent***

*randArr[0],randArr[1],randArr[2],multiply, → written to **fifo2** from **parent***

*result! → written to **fifo2** from **child1***

Example Usage:

12,24,88,15,multiply,145! → 145 is sum result

145!12,24,88,15,multiply, → according to scheduling it can change

Since only the main process will write to the first fifo, it was parsed by commas and converted to integers with strtol. Then, these values were placed in an integer array in child1, an exclamation character was placed at the end of the meeting and result, and written to fifo2, and the child process was terminated with exit success.

After forking the second fifo, the "multiply" command was added to the string version of the random array created in the parent, and it was written to fifo2 with the same spelling as fifo1.

NOTE:

This is how I solved the synchronization problem on fifo two. If two different processes write the same type of data to the same file, I put a process-specific sign at the end(you can check example in above). If it was necessary for me to read from all these processes, I waited with read until all these marks were read. (I kept values such as delimeter_flag1, delimeter_flag2 and exited the read if they were all set to 1)

```
static void sigchldHandler(int sign){
    int status, savedErrno;
    int childpid;

    savedErrno = errno;
    while((childpid = waitpid(-1, &status, WNOHANG)) > 0){
        printf("handler: Reaped child %ld\n", (long) childpid);
        signalCnt++;
    }
    if (childpid == -1 && errno != ECHILD)
        perror("waitpid");
    if(sign == SIGINT){
        printf("SIGINT handled... Your operation will done after process done\n");
        unlink(FIRST_CHILD_FIFO);
        unlink(SEC_CHILD_FIFO);
    }

    errno = savedErrno;
}
```

The SIGCHLD signal is sent to a parent process whenever one of its children terminates. By default, this signal is ignored, but we can catch it by installing a signal handler.Within the signal handler, we can use wait()-waitpid (or similar) to **reap the zombie child**.This handler displays the process ID and wait status of each reaped child.

In this code there is a problem In order to see that multiple SIGCHLD signals are not queued while the handler is already invoked.

To handle the situation where multiple `SIGCHLD` signals may be received while the handler is already executing, I used a signal mask to block `SIGCHLD` signals while the handler is running. This prevents multiple invocations of the handler from overlapping. I did it with waitpid() and also with SIGINT handling we can avoid **orphan** situation of process, (parent dies before the child.)

3.Results and Test Cases

3.1 Invalid Argument Test

3.1.1. Negative value

```
oguz@oguz-ubuntu:~/Desktop/Spring2023/System Programming/System-Prog
ramming-/hw2_2024$ ./oguz_hw2 -5
Argument must be greater than 0.
```

3.1.2. No value

```
oguz@oguz-ubuntu:~/Desktop/Spring2023/System Programming/System-Prog
ramming-/hw2_2024$ ./oguz_hw2
Invalid argument number, must be 2.
```

3.2 Unit Test of Child1

```
oguz@oguz-ubuntu:~/Desktop/Spring2023/System Programming/System-Prog
ramming-/hw2_2024$ ./oguz_hw2 3
Random values generated (ends with comma): 13,5,44,
Entered Child Process 1 : 176822
Array successfully written to first fifo
13,5,44,
Sum (exlamation ended): 62!child exited
```

Generated array values → **13, 5, 44**

child pid = **176822**

sum = 44+13+5 = **62**

3.3 System Test With Different Argument

```
oguz@oguz-ubuntu:~/Desktop/Spring2023/System Programming/System-Programming-/hw2_2024$ ./oguz_hw2 5
Random values generated (ends with comma): 1,5,0,4,4,
Entered Child Process 1. (pid : 235839)
Array successfully written to first fifo
Sum (exlamation ended): 14!
Entered Child Process 2. (pid : 235845)
Successfully write to first child to second fifo file
Proceeding...
Multiplication result: 0
Addition result: 14
Total result: 14
handler: Reaped child 235839
handler: Reaped child 235845
All childs are dead... Terminating...Bye
oguz@oguz-ubuntu:~/Desktop/Spring2023/System Programming/System-Programming-/hw2_2024$
```

values = **1, 5, 0, 4, 4** **(arg = 5)**

sum result = **14** **(SUCCESSFULL)**

multiplication = **0** **(SUCCESSFULL)**

total result = **0** **(SUCCESSFULL)**

exited child pid → **(SUCCESSFULL)**

Test → **(SUCCESSFULL)**

```
oguz@oguz-ubuntu:~/Desktop/Spring2023/System Programming/System-Programming-/hw2_2024$ ./oguz_hw2 1
Random values generated (ends with comma): 9,
^CEntered Child Process 1. (pid : 235955)
Array successfully written to first fifo
Sum (exlamation ended): 9!
Entered Child Process 2. (pid : 235957)
Successfully write to first child to second fifo file
Proceeding...
Multiplication result: 9
Addition result: 9
Total result: 18
```

values = **9** **(arg = 1)**

sum result = **9** **(SUCCESSFULL)**

multiplication = **9** **(SUCCESSFULL)**

total result = **9** **(SUCCESSFULL)**

exited child pid → **(SUCCESSFULL)**

Test → **(SUCCESSFULL)**

```
oguz@oguz-ubuntu:~/Desktop/Spring2023/System Programming/System-Programming-/hw2_2024$ ./oguz_hw2 6
Random values generated (ends with comma): 2,5,0,8,1,5,
Entered Child Process 1. (pid : 239108)
Array successfully written to first fifo
Sum (exlamation ended): 21!
Entered Child Process 2. (pid : 239109)
Proceeding...
Successfully write to first child to second fifo file
Multiplication result: 0
Addition result: 21
Total result: 21
handler: Reaped child 239108
handler: Reaped child 239109
Proceeding...
All childs are dead... Terminating...Bye
```

values = **2,5,0,8,1,5** **(arg = 6)**

sum result = **21** **(SUCCESSFULL)**

multiplication = **0** **(SUCCESSFULL)**

total result = **21** **(SUCCESSFULL)**

exited child pid → **(SUCCESSFULL)**

Test → **(SUCCESSFULL)**

3.4 Signal Test

```
oguz@oguz-ubuntu:~/Desktop/Spring2023/System Programming/System-Programming-/hw2_2024$ ./oguz_hw2 1
Random values generated (ends with comma): 9,
^CEntered Child Process 1. (pid : 235955)
Array successfully written to first fifo
Sum (exlamation ended): 9!
```

SIGINT signal checked with externally when process is continue(in sleep in parent or child) I mask **SIGINT** and wait process termination successfully.

3.5 Memory Leak Check

I work my program like below and result shown in image

>> valgrind ./oguz_hw2 4

```
==239507==
==239507== HEAP SUMMARY:
==239507==     in use at exit: 0 bytes in 0 blocks
==239507==   total heap usage: 3 allocs, 3 frees, 1,052 bytes allocated
==239507==
==239507== All heap blocks were freed -- no leaks are possible
==239507==
==239507== For lists of detected and suppressed errors, rerun with: -s
==239507== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```