FATİH DOĞAÇ

CSE344 – SYSTEM PROGRAMMING

HOMEWORK #2

1901042654

1. Introduction

This code block's main task is to take an integer from user and create an array of that size of integer. Then pass this array to the child processes for those purposes:

Child Process 1:
Open the first FIFO and read random numbers to perform a summation operation.
Write the result to the second FIFO.

Child Process 2:
Open the second FIFO and read the command to perform the multiplication operation. (It should receive the "multiply" command from the parent process and
use a string compare method. Finally, it should perform the operation)
Print the sum of the results of all child processes on the screen.

And **parent process** takes care of these jobs:

- Create a program that takes an integer argument.
- Create two FIFOs (named pipes).
- Send an array of random numbers to the first FIFO and the second FIFO. Send a
- command to the second FIFO (requesting a multiplication operation).
- Use the fork() system call to create two child processes and assign each to a FIFO.
- All child processes sleep for 10 seconds, execute their tasks, and then exit.
- Set a signal handler for SIGCHLD in the parent process to handle child process
- termination.
- Enter a loop, printing a message containing "proceeding" every two seconds.
- The signal handler should call waitpid() to reap the terminated child process, print out the process ID of the exited child, and increment a counter.
- When the counter reaches the number of children originally spawned, the
- program exits.

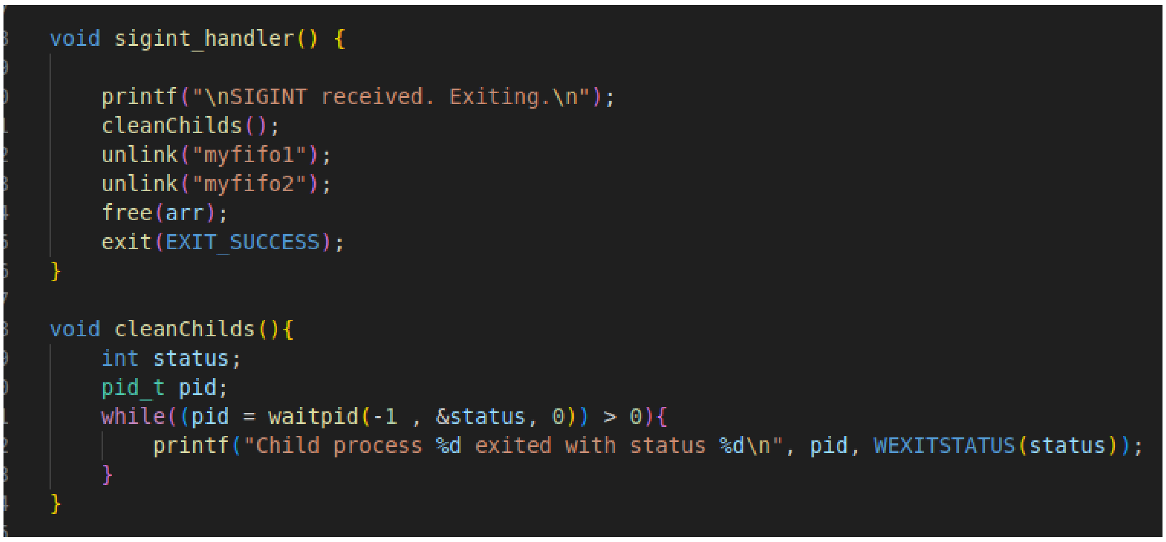
Error Scenarios:

- If FIFOs cannot be created or if there are errors in data/command transmission, appropriate error messages should be displayed.
- If child processes fail to complete their tasks successfully or encounter unexpected errors, error messages indicating the failure should be displayed.
- If the counter value is not managed correctly or if exit statuses are not printed for each child process, error messages should be displayed indicating the issue.

2. Signal Handling:

Handled signals are SIGINT and SIGCHLD.

SIGINT happens when the program gets interrupted by user or OS. To handle that case, the program has a SIGINT handler.



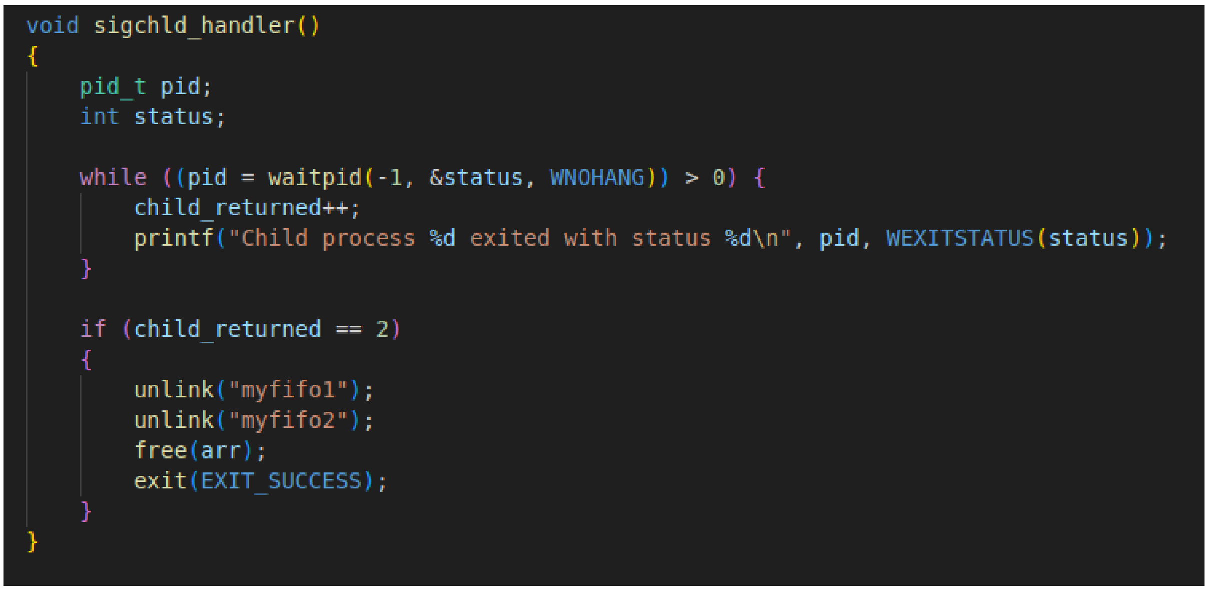
```c
void sigint_handler() {

    printf("\nSIGINT received. Exiting.\n");
    cleanChilds();
    unlink("myfifo1");
    unlink("myfifo2");
    free(arr);
    exit(EXIT_SUCCESS);
}

void cleanChilds(){
    int status;
    pid_t pid;
    while((pid = waitpid(-1 , &status, 0)) > 0){
        printf("Child process %d exited with status %d\n", pid, WEXITSTATUS(status));
    }
}
```

When SIGINT caught, the program informs the user and starts to clean child processes. Meaning by cleaning childs is waiting for them to finish naturally to prevent orphan and zombie childs.

Here the dynamic array that parent uses is global to be able to free it when the signals came.

While cleaning up the child processes, the program informs the user about which child exited with which status. Then it destroys the FIFOs, frees the dynamic array and ends the program.

SIGCHLD happens when a child finished its job. It informs the parent process with this signal. To handle this signal, the program has a SIGCHLD handler.



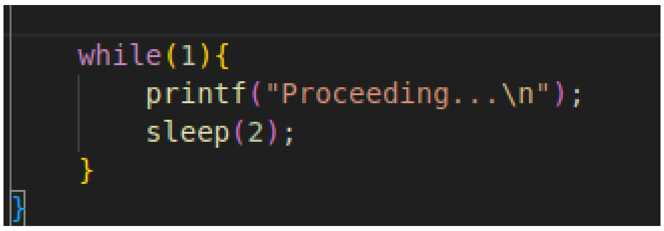
```c
void sigchld_handler()
{
    pid_t pid;
    int status;

    while ((pid = waitpid(-1, &status, WNOHANG)) > 0) {
        child_returned++;
        printf("Child process %d exited with status %d\n", pid, WEXITSTATUS(status));
    }

    if (child_returned == 2)
    {
        unlink("myfifo1");
        unlink("myfifo2");
        free(arr);
        exit(EXIT_SUCCESS);
    }
}
```

It waits for childs to finish their jobs. Then the program counts the child number and informs the user about which child exited with which status. If the counted child number is equal to the total number of childs, it means the program is finished. Destroy the FIFOs and exit.

The childs_returned variable is a volatile atomic variable (integer). And the reason why is volatile keyword prevents the compiler from applying any optimizations on objects that can change in ways that cannot be determined by compiler.

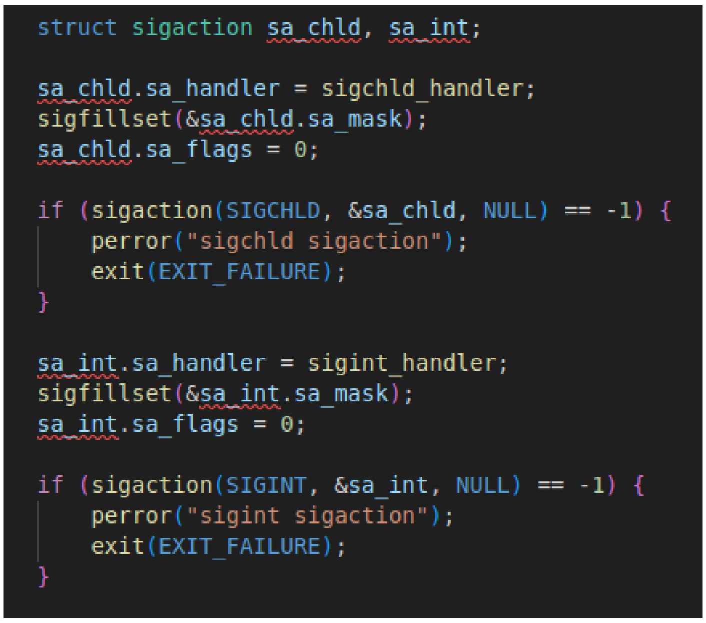
This exit ends the parent process. So the parent process trusts this handler for this job:

- Enter a loop, printing a message containing "proceeding" every two seconds.



```c
    while(1){
        printf("Proceeding...\n");
        sleep(2);
    }
}
```

Those signal handlers are initialized in parent process at the beginning of the code.

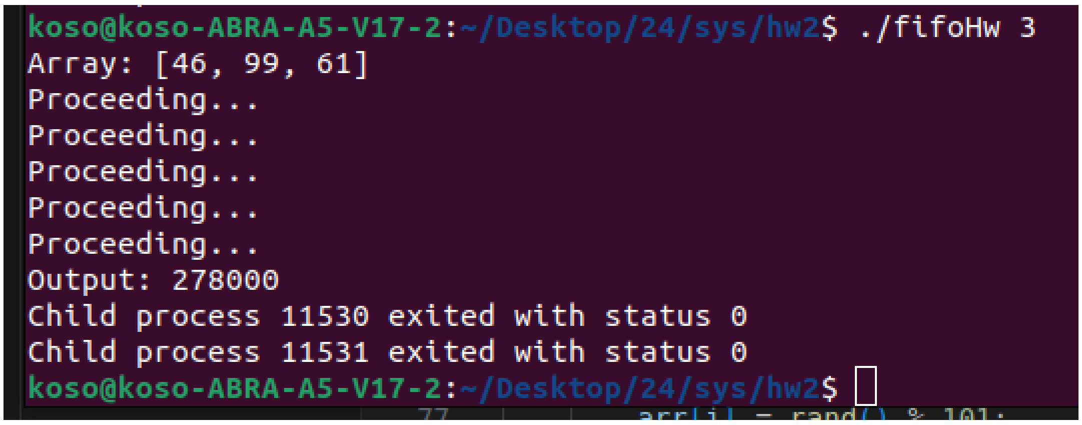


```c
struct sigaction sa_chld, sa_int;

sa_chld.sa_handler = sigchld_handler;
sigfillset(&sa_chld.sa_mask);
sa_chld.sa_flags = 0;

if (sigaction(SIGCHLD, &sa_chld, NULL) == -1) {
    perror("sigchld sigaction");
    exit(EXIT_FAILURE);
}

sa_int.sa_handler = sigint_handler;
sigfillset(&sa_int.sa_mask);
sa_int.sa_flags = 0;

if (sigaction(SIGINT, &sa_int, NULL) == -1) {
    perror("sigint sigaction");
    exit(EXIT_FAILURE);
}
```

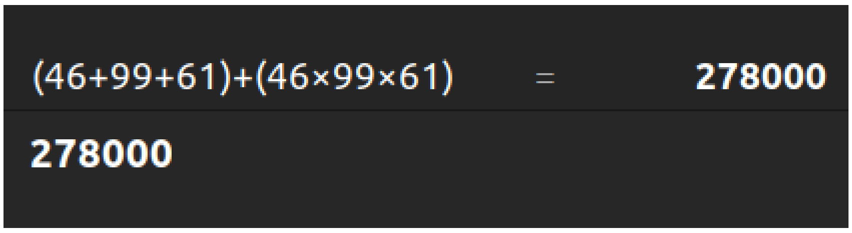
In the sa_chld and sa_int struct object, first the program sets the handler and then fills in the signals to be masked with sigfillset(). And this sigactions don’t need flags.

3. Test Scenarios

a) Happy path scenario



```
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw2$ ./fifoHw 3
Array: [46, 99, 61]
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Output: 278000
Child process 11530 exited with status 0
Child process 11531 exited with status 0
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw2$
```

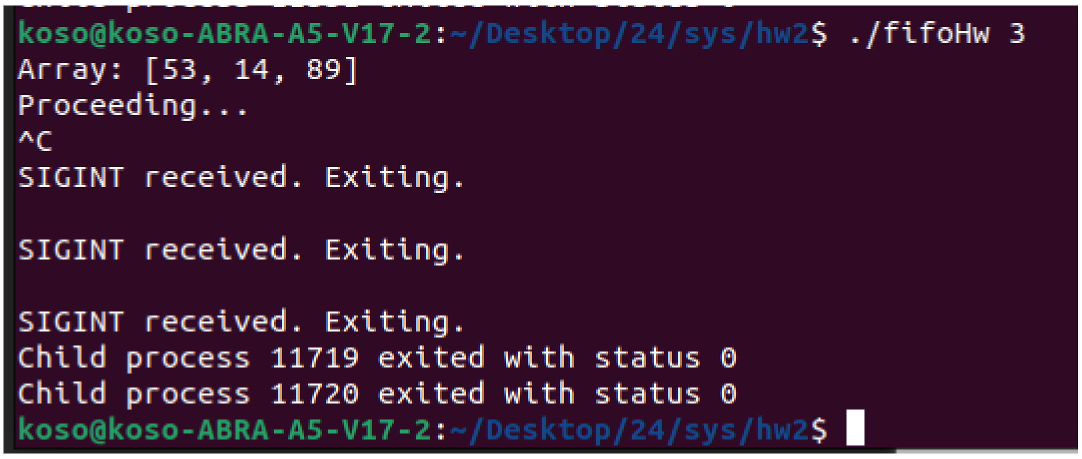


(46+99+61)+(46×99×61) = 278000

278000

Childs exited with success
Output is correct
Parent wrote Proceeding… every two seconds

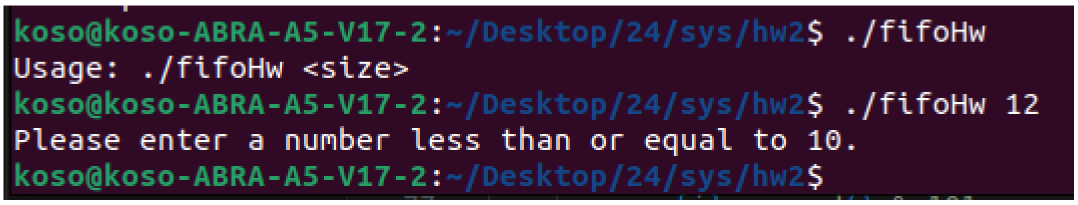
b) Interrupt Scenario



```
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw2$ ./fifoHw 3
Array: [53, 14, 89]
Proceeding...
^C
SIGINT received. Exiting.

SIGINT received. Exiting.

SIGINT received. Exiting.
Child process 11719 exited with status 0
Child process 11720 exited with status 0
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw2$ 
```

Childs exited gracefully
3 exiting messages for 2 children and 1 parent

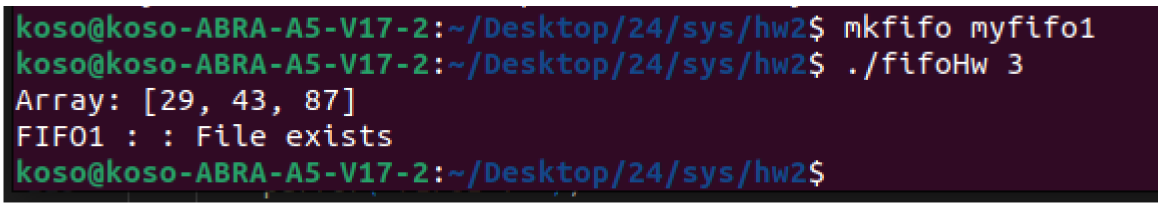
c) Bad Input Scenario



```
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw2$ ./fifoHw
Usage: ./fifoHw <size>
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw2$ ./fifoHw 12
Please enter a number less than or equal to 10.
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw2$
```

The max input number is 10 because if randomly chooses 10 numbers around 100, the result might be so big the program gets an overflow error.

d) FIFO exists Scenario



```
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw2$ mkfifo myfifo1
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw2$ ./fifoHw 3
Array: [29, 43, 87]
FIFO1 : : File exists
koso@koso-ABRA-A5-V17-2:~/Desktop/24/sys/hw2$
```