GTU CSE344 – SYSTEM PROGRAMMING HW2 REPORT

REŞİT AYDIN – 200104004019

1. Program execution details

- Execute “make = X” or “make all args=X” command to compile and run the program. (X is a positive integer)
- Execute “make clean” to clean up non-used files. (.out files, fifos).
- Makefile contains **-std=gnu11** flag in the compile section for stability and portability.

2. Example Input/Outputs

2.1

```
ra@ra-Aspire-F5-573G:~/Desktop/hw2$ make args=2
gcc main.c -std=gnu11
./a.out 2

Random array: 7 46
PARENT PROCESS is creating FIFOs...
PARENT PROCESS is creating child processes using fork...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
**** Process_id of the exited child: 17553 ****
**** Exit status of the child was 0 ****
Proceeding...
--------------- RESULT --------------- : 375
**** Process_id of the exited child: 17554 ****
**** Exit status of the child was 0 ****
Proceeding...
The program has completed successfully. Exiting...
ra@ra-Aspire-F5-573G:~/Desktop/hw2$
```

This command created an array of size 2 with random integers then proceeded with other operations to calculate the final result.

- FIFOs were created successfully, and data/command transmission occurred

without errors.

- The array was initialized with ten elements: [7, 46].
- These values were read by the FIFOs.
- In the first FIFO, the sum of all elements was calculated as 53.
- This result value was written to the second FIFO.
- In the second FIFO, the multiplication of all elements was calculated as 322.
- The printed value of the screen is 375.
- Child processes completed their tasks successfully and exited without errors.
- The parent process correctly managed the counter value and printed exit

statuses for each child process.

2.2

```
ra@ra-Aspire-F5-573G:~/Desktop/hw2$ make args=5
gcc main.c -std=gnu11
./a.out 5

Random array: 24 31 18 14 52
PARENT PROCESS is creating FIFOs...
PARENT PROCESS is creating child processes using fork...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
**** Process_id of the exited child: 17677 ****
**** Exit status of the child was 0 ****
Proceeding...
Proceeding...
-------------- RESULT -------------- : 9749515
**** Process_id of the exited child: 17678 ****
**** Exit status of the child was 0 ****
Proceeding...
The program has completed successfully. Exiting...
ra@ra-Aspire-F5-573G:~/Desktop/hw2$
```

This command created an array of size 5 with random integers then proceeded with other operations to calculate the final result.
Same steps apply for also this and other sizes of execution.

3. Error Handling

3.1

If FIFOs cannot be created or if there are errors in data/command transmission, appropriate error messages are being displayed.

```
if(mkfifo(fifo1, 0666) == -1){    // create first fifo
    if(errno != EEXIST){
        fprintf(stderr, "[%ld]: failed to create named pipe %s\n", (long)getpid(), "fifo1");
        exit(1);
    }
}

if(mkfifo(fifo2, 0666) == -1){    // create second fifo
    if(errno != EEXIST){
        fprintf(stderr, "[%ld]: failed to create named pipe %s\n", (long)getpid(), "fifo2");
        exit(1);
    }
}
```

3.2

If child processes fail to complete their tasks successfully or encounter unexpected errors, error messages indicating the failure re being displayed.

```
if(fifo1_fd == -1){
    fprintf(stderr, "[%ld]: failed to open named pipe %s\n", (long)getpid(), "fifo1");
        exit(1);
}

sleep(10);

int received_data[array_size];

// printf("CHILD-1 PROCESS is reading data from FIFO1...\n");
ssize_t bytes_read = read(fifo1_fd, received_data, array_size * sizeof(int));
if (bytes_read == -1) {
    fprintf(stderr, "[%ld]: Error reading from named pipe %s\n", (long)getpid(), fifo1);
    close(fifo1_fd);
    exit(1);
}
```

3.3

If the counter value is not managed correctly or if exit statuses are not printed for each child process, error messages are being displayed indicating the issue.

```c
while (1) {
    sleep(2);
    printf("Proceeding...\n");
    fflush(stdout); // Flush the stdout buffer to ensure the message is printed immediately

    // Exit the loop when all child processes have terminated
    if (child_counter == 2) {
        printf("The program has completed successfully. Exiting...\n");
        return 0;
    }

    // Check if more child processes have terminated than expected
    if (child_counter > 2) {
        fprintf(stderr, "Error: More child processes have terminated than expected.\n");
        return 1;
    }
```

```c
while ((exited_pid = waitpid(-1, &status, WNOHANG)) > 0){
        printf("**** Process_id of the exited child: %d ****\n", exited_pid);
    if (WIFEXITED(status)) {
        printf("**** Exit status of the child was %d ****\n", WEXITSTATUS(status));
    } else {
        fprintf(stderr, "Error: Child %d terminated abnormally.\n", exited_pid);
    }
    child_counter++;
}
```

4. Bonus Section

4.1

Implementing a zombie protection method

```c
// Signal handler function for SIGCHLD
void sigchld_handler() {

    int status = 0;
    int exited_pid = 0;
    while ((exited_pid = waitpid(-1, &status, WNOHANG)) > 0){
            printf("**** Process_id of the exited child: %d ****\n", exited_pid);
        if (WIFEXITED(status)) {
            printf("**** Exit status of the child was %d ****\n", WEXITSTATUS(status));
        } else {
            fprintf(stderr, "Error: Child %d terminated abnormally.\n", exited_pid);
        }
        child_counter++;
    }
}
```

This signal handler waits for child processes to terminate and increments the child_counter variable each time a child process exits. By doing so, it ensures that no zombie processes are left behind, as terminated child processes are promptly reaped by the parent process.

4.2

Printing the exit statuses of all processes

```c
if (WIFEXITED(status)) {
    printf("**** Exit status of the child was %d ****\n", WEXITSTATUS(status));
} else {
```

```
**** Process_id of the exited child: 17677 ****
**** Exit status of the child was 0 ****
Proceeding...
Proceeding...
-------------- RESULT -------------- : 9749515
**** Process_id of the exited child: 17678 ****
**** Exit status of the child was 0 ****
```