Introduction

My program consists of 1parent process and 2 child processes. Parent process: The parent process generates random numbers and sends them to the child processes through two different FIFOs (named pipes).

FIFO1: I use FIFO1 to send random numbers from the main process to child1.

FIFO2: I use FIFO2 to send the total result from child1 and the multiplication command from the main process to the second child process. In child2, I multiply the random numbers and add them to the sum received from child1, then print the result to the screen.

Parent Process:

1)Create integer argument

```
int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        fprintf(stderr, "Usage: %s <array_size>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
```

Here, we are taking the argument entered by the user using **argc** and **argv[]**. If the user has not entered the correct number of arguments, we print an error message indicating the usage and terminate the program with failure.

2) Creating two FIFOs (named pipes):

```
if (mkfifo(FIFO1, 0666) == -1 || mkfifo(FIFO2, 0666) == -1)
{
    perror("mkfifo");
    exit(EXIT_FAILURE);
}
```

In this step, we use the **mkfifo()** function to create two FIFOs. If the FIFOs cannot be created successfully, we print an error message and terminate the program

4)Creating two child processes and assigning each to a FIFO:

```
pid_t pid1, pid2;
pid1 = fork();
if (pid1 == -1)
{
    perror("fork");
    exit(EXIT_FAILURE);
}
if (pid1 == 0)
{
    child_process1();
}
else
{
    pid2 = fork();
    if (pid2 == 0)
    {
        child_process2();
    }
    else if (pid2 == -1)
    {
        perror("fork");
        exit(EXIT_FAILURE);
    }
```

In this step, I create two child processes using the **fork()** function. I then assign each child process to the respective FIFO.

5) SIGCHLD signal and The signal handler should call waitpid()

```
while ((id = waitpid(-1, &status, WNOHANG)) > 0)
{
    if (WIFEXITED(status))
    {
        printf("Child with PID %d exited, status=%d\n", id, WEXITSTATUS(status));
    }
    else if (WIFSIGNALED(status))
    {
        printf("Child with PID %d killed by signal %d\n", id, WTERMSIG(status));
    }
```

In this step, upon receiving the SIGCHLD signal, I clean up the terminated child processes by calling the waitpid() function.

7) "proceeding" every two seconds and the program exits.

```
while (count_childexit < 2)
{
    printf("Proceeding...\n");
    sleep(2);
}
```

In this step, we ensure the continuation of operations within the loop while waiting for all child processes to complete. Once all child processes have finished, we proceed to terminate the program.

Child Process 1:

```
void child_process1()
{
    int fd1 = open(FIFO1, O_RDONLY);
    if (fd1 == -1)
```

I have defined a function named child_process1(). I open FIFO1 in read mode and perform addition by reading random numbers. Then, I write the result to FIFO2.

Child Process 2:

```
void child_process2()
{
    int fd2 = open(FIFO2, O_RDONLY);
    if (fd2 == -1)
    {
        perror("Failed to open FIFO2");
        exit(EXIT_FAILURE);
    }
```

I have defined a function named child_process2(). After opening in read mode, I first read the "multiply" command. Then, I read random numbers and multiply them with each other. After that, I had saved the summation result from child1 via FIFO2. I add it to the multiplication result and print the final result to the screen.

zombie protection method and Print the exit statuses of all processes for an additional

```c
void child_handler(int sig)
{
    pid_t id;
    int status;
    while ((id = waitpid(-1, &status, WNOHANG)) > 0)
    {
        if (WIFEXITED(status))
        {
            printf("Child with PID %d exited, status=%d\n", id, WEXITSTATUS(status));
        }
        else if (WIFSIGNALED(status))
        {
            printf("Child with PID %d killed by signal %d\n", id, WTERMSIG(status));
        }
        count_childexit++;
    }
}
```

```c
struct sigaction sa;
sigemptyset(&sa.sa_mask);
sa.sa_flags = 0;
sa.sa_handler = child_handler;

sigaction(SIGCHLD, &sa, NULL);
```

A zombie process is a process that has terminated but whose exit status has not yet been collected by its parent process. A child process may send a termination notification, but the parent process may not have waited for or collected its termination status yet. To address this issue, I used the SIGCHLD signal method. By receiving the SIGCHLD signal, the parent process can retrieve the status of the child process and fully terminate it, preventing zombie processes.

Error:

If FIFOs cannot be created,

```c
if (mkfifo(FIFO1, 0666) == -1 || mkfifo(FIFO2, 0666) == -1)
{
    perror("mkfifo");
    exit(EXIT_FAILURE);
}
```

If child processes fail to complete their tasks successfully or encounter unexpected errors,

```c
if (write(fd2, &sum, sizeof(int)) == -1)
{
    perror("Error writing to FIFO2");
    exit(EXIT_FAILURE);
}
```

```c
int fd1 = open(FIFO1, O_RDONLY);
if (fd1 == -1)
{
    perror("Failed to open FIFO1");
    exit(EXIT_FAILURE);
}
sleep(10);
int random_numbers[arraysize];
if (read(fd1, random_numbers, arraysize * (sizeof(int))) == -1)
{
    perror("Error reading from FIFO1");
    exit(EXIT_FAILURE);
}
```

If the counter value is not managed correctly or exit statuses are not printed for each child process,

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

Synchronization and Timing:

If child processes attempt to access FIFOs immediately, the FIFOs may not be ready yet, resulting in an error. By using sleep(10), the child processes are delayed by 10 seconds, allowing sufficient time for the FIFOs to become ready.

Using sleep(10) ensures that the child processes start simultaneously and allows the first child process to complete the addition operation and write the result to FIFO2. This provides enough time for the second child process to read both the multiplication command and the addition result from FIFO2 and perform its operation.

OUTPUT

```
arife@LAPTOP-SRROEDP3:~/210104004294_arife_yurtseven$ make
----------------------------------------
Compiling...
arife@LAPTOP-SRROEDP3:~/210104004294_arife_yurtseven$ ./test 5
Random numbers: 0 3 5 4 5
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
CHILD1 SUM 17
CHILD2 PRODUCT: 0
RESULT: 17
Child with PID 549 exited, status=0
Child with PID 550 exited, status=0
```

```
arife@LAPTOP-SRROEDP3:~/210104004294_arife_yurtseven$ ./test 4
Random numbers: 7 8 2 7
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
CHILD1 SUM 24
CHILD2 PRODUCT: 784
RESULT: 808
Child with PID 710 exited, status=0
Child with PID 711 exited, status=0
arife@LAPTOP-SRROEDP3:~/210104004294_arife_yurtseven$ make clean
----------------------------------------
Removing compiled files...
arife@LAPTOP-SRROEDP3:~/210104004294_arife_yurtseven$
```

Mkfifo: if fifo files already exist, program gives error.