CSE 344

Homework #2 Report

Berru Lafcı

1901042681

How to run:

- It compiles with **make**.
- Runs with **./src <number>**
- Cleans with **make clean**.
- After the run, it waits for the command name which is “multiply”.

```
berrulafci@fedora:~/System-Programming/HW2$ make
gcc -Wall -Wextra -Werror -c src.c -o src.o
gcc -Wall -Wextra -Werror -o src src.o
berrulafci@fedora:~/System-Programming/HW2$ ./src 5
The initial random array is: 98 7 26 29 41
Write the command (multiply):
```

Logic of the code:

Signal Handlers:

1. **sigchld_handler()**:
    - This function is invoked when the parent process receives the SIGCHLD signal, indicating that a child process has terminated. It waits for child processes to terminate using **waitpid()** in a loop. By using *WNOHANG*, the parent process can continue its execution even if no child processes have terminated yet.
    - For each terminated child, it increments **child_count**, which keeps track of the number of terminated child processes.
    - If **waitpid()** returns an error and the error is not due to no child processes being available (**ECHILD**), it prints an error message using **perror()**.
2. **sigint_handler()**:
    - This function is called when the parent process receives the SIGINT signal (Ctrl+C).
    - It unlinks the FIFOs named "fifo1" and "fifo2" using **unlink()** to finish the program cleanly.
    - It sends the SIGKILL signal to kill all child processes using **kill()**. If any error occurs during this operation, it prints an error message using **perror()**.
    - Finally, it exits the program.

Main Function:

1. **Signal Handling**:
    - Sets up signal handlers for SIGCHLD and SIGINT using **sigaction()**.
    - **SA_RESTART** ensures that the parent process continues its execution smoothly after handling **SIGCHLD** signals without being affected by interrupted system calls.
2. The program expects one command-line argument, which is the length of the array. It parses this argument using **atoi()** to convert the string argument to an integer.

3. Creates an array of random integers of the specified length using **malloc()** to allocate memory and **rand()** to generate random numbers.

4. Asks the user to input a command. In this case, it expects the user to input "multiply". If the user writes command other than “multiply”, it catches that in child2 and gives **perror().** Then it doesn’t calculate the multiplication.

```
if (strcmp(command, "multiply") == 0) {
    for (int i = 0; i < arr_len; i++) {
        result2 *= arr2[i];
    }
}
else {
    perror("Invalid command");
    return 1;
}
```

5. Creates two FIFOs named "fifo1" and "fifo2" using **mkfifo()**.

6. Forks two child processes using **fork()**.

7. **Parent Process Operations**:

    - The parent process writes data (the array of random integers and the command) into FIFOs using write().

    - It handles the SIGCHLD signal using the **sigchld_handler()** function. This function waits for child processes to terminate using **waitpid()** within a loop. By waiting for child processes, the parent process ensures synchronization and prevents it from exiting badly.

```
while ((pid = waitpid(-1, &status, WNOHANG)) > 0) {
    printf("Child process with PID %d terminated.\n", pid);
    child_count += 1;
}
```

8. Upon termination of all processes, unlinks the FIFOs using **unlink()** to clean up the resources.

Child Processes:

1. **Child 1**:

    - Opens fifo1 with *Read Only* mode and reads the array through FIFO "fifo1".

```
int fd1 = open(fifo1, O_RDONLY);
if(fd1 == -1) {
    perror("open1");
    return 1;
}

int arr1[arr_len];

if(read(fd1, arr1, arr_len * sizeof(int)) == -1) {
    perror("read1");
    return 1;
}
```

    - Computes the sum of the array elements.

    - Opens fifo2 with *Write Only* mode and writes the sum through FIFO "fifo2".

```c
// Send the sum to the second FIFO
int fd2 = open(fifo2, O_WRONLY);
if(fd2 == -1) {
    perror("open2");
    return 1;
}

if(write(fd2, &sum, sizeof(int)) == -1) {
    perror("write2");
    return 1;
}
```

2. **Child 2**:

    - Reads the array, command and sum through FIFO "fifo2". I put sleep(5) in here because of synchronization problem. The reading order is like this because in parent, I first wrote command and array then in child1 I wrote the sum.

```c
char command[10];
if(read(fd2, command, 10) == -1) {
    perror("read2");
    return 1;
}

int arr2[arr_len];
if(read(fd2, arr2, arr_len * sizeof(int)) == -1) {
    perror("read2");
    return 1;
}

sleep(5);
int result_from_child1;
if(read(fd2, &result_from_child1, sizeof(int)) == -1) {
    perror("read2");
    return 1;
}
```

    - Computes the multiplication of the array elements.
    - Sums the old result and the multiplication result and prints it.

Error Handling:

- Throughout the code, errors in system calls (e.g., **fork()**, **open()**, **write()**, **read()**, **mkfifo()**) are handled using **perror()**.

How I achieved the synchronization:

Between child and parent:

First, I opened FIFOs as read for random number and command in child1 and child2. Then I opened as write for these in the parent. Thus, the synchronization problem was solved because I opened the FIFOs as write in one process and as read in the other.

Between child1 to fifo2 (sum):

Since I am going to write to fifo2 from child1, I first created child2 with a fork and made fifo2 read in it for sum. Then, I created child1 with fork and made fifo2 write sum in it. Thus, the synchronization for writing to fifo2 from child1 was achieved.

When I changed the forking order of child1 and child2, sum write did not work and was blocked. Because it waited for it to be opened as read in another process, but this could not happen because I opened it as write and it was blocked.

Bonus Parts:

1-Implement zombie protection method:

By properly waiting for child processes to terminate and promptly handling their termination in the SIGCHLD handler, the parent process prevents them from becoming zombies. Zombie processes are those that have terminated but whose exit status has not yet been collected by their parent process. In this code, the parent process collects the exit status of its child processes using waitpid(), ensuring that they do not remain in a zombie state.

In SIGCHLD handler:

```
while ((pid = waitpid(-1, &status, WNOHANG)) > 0) {
    printf("Child process with PID %d terminated.\n", pid);
    child_count += 1;
}
```

In child:

```
exit(EXIT_SUCCESS);
```

In parent:

```
// Parent process
while (child_count < 2) {
    printf("Proceeding...\n");
    sleep(2);
}

printf("All child processes terminated. Exiting.\n");
```

2-Print the exit statues of all processes:

I did this part in SIGCHLD handler for child processes.

```
while ((pid = waitpid(-1, &status, WNOHANG)) > 0) {
    printf("Child process with PID %d terminated.\n", pid);
    child_count += 1;
}
```

```
SIGCHLD received.
Child process with PID 12543 terminated.
child_count: 1
```

```
SIGCHLD received.
Child process with PID 12541 terminated.
```

And for parent, I did it in the end of the code.

```c
// Parent process
while (child_count < 2) {
    printf("Proceeding...\n");
    sleep(2);
}

printf("All child processes terminated. Exiting.\n");
```

```
All child processes terminated. Exiting.
berrulafci@fedora:~/System-Programming/HW2$
```

Example runs:

Successful run:

```
berrulafci@fedora:~/System-Programming/HW2$ ./src 7
The initial random array is: 24 52 6 39 80 15 50
Write the command (multiply):
multiply
Parent pid: 12539
Child pid: 12541
Child pid: 12543
The command and the array are sent to the second FIFO
The array is sent to the first FIFO
Proceeding...
The random number array is received from the first FIFO
The sum is sent to the second FIFO
The sum of the array is: 266
SIGCHLD received.
Child process with PID 12543 terminated.
child_count: 1
Proceeding...
Proceeding...
Proceeding...
The random number array is received from the second FIFO
The command is received from the second FIFO
The result from the first child is received from the second FIFO: 266
The result of the command (multiply) is: 342050816
The final result is: 342051082
SIGCHLD received.
Child process with PID 12541 terminated.
child_count: 2
All child processes terminated. Exiting.
berrulafci@fedora:~/System-Programming/HW2$
```

SIGINT received after one child process terminated:

```
berrulafci@fedora:~/System-Programming/HW2$ make
gcc -Wall -Wextra -Werror -c src.c -o src.o
gcc -Wall -Wextra -Werror -o src src.o
berrulafci@fedora:~/System-Programming/HW2$ ./src 5
The initial random array is: 98 7 26 29 41
Write the command (multiply):
multiply
Child pid: 11658
Parent pid: 11654
Child pid: 11659
The command and the array are sent to the second FIFO
The array is sent to the first FIFO
Proceeding...
The random number array is received from the first FIFO
The sum is sent to the second FIFO
The sum of the array is: 201
SIGCHLD received.
Child process with PID 11659 terminated.
child_count: 1
Proceeding...
^CSIGINT received. Terminating all processes.
SIGINT received. Terminating all processes.
Killed
```

SIGINT received after two child process created:

```
berrulafci@fedora:~/System-Programming/HW2$ ./src 6
The initial random array is: 70 55 66 93 6 63
Write the command (multiply):
multiply
Parent pid: 11688
Child pid: 11692
Child pid: 11693
^CSIGINT received. Terminating all processes.
SIGINT received. Terminating all processes.
SIGINT received. Terminating all processes.
Killed
berrulafci@fedora:~/System-Programming/HW2$
```

SIGINT received before child processes:

```
berrulafci@fedora:~/System-Programming/HW2$ ./src 6
The initial random array is: 77 7 88 19 11 46
Write the command (multiply):
^CSIGINT received. Terminating all processes.
Killed
berrulafci@fedora:~/System-Programming/HW2$
```

Wrong command name:

```
berrulafci@fedora:~/System-Programming/HW2$ ./src 7
The initial random array is: 49 89 96 33 99 24 78
Write the command (multiply):
aaaa
Parent pid: 12394
Child pid: 12403
Child pid: 12404
The command and the array are sent to the second FIFO
The array is sent to the first FIFO
Proceeding...
The random number array is received from the first FIFO
The sum is sent to the second FIFO
The sum of the array is: 468
SIGCHLD received.
Child process with PID 12404 terminated.
child_count: 1
Proceeding...
Proceeding...
Proceeding...
The random number array is received from the second FIFO
The command is received from the second FIFO
The result from the first child is received from the second FIFO: 468
Invalid command: Success
SIGCHLD received.
Child process with PID 12403 terminated.
child_count: 2
All child processes terminated. Exiting.
berrulafci@fedora:~/System-Programming/HW2$
```