CSE 344
System Programming
2nd Assignment
Report

Author

Mert Emir ŞEKER

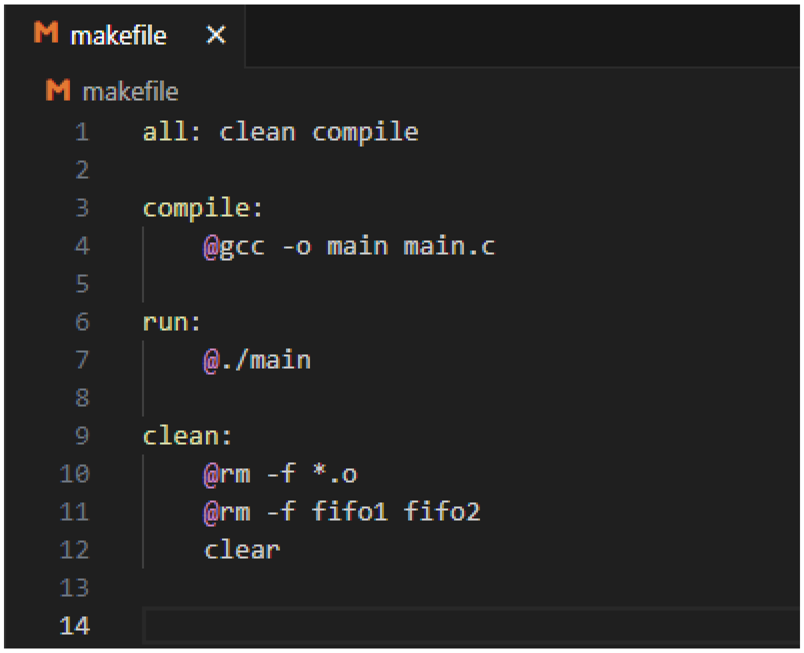
200104004085

Date

18.04.2024

Table Of Contents

Makefile:


makefile ×

makefile

```
all: clean compile

compile:
	@gcc -o main main.c

run:
	@./main

clean:
	@rm -f *.o
	@rm -f fifo1 fifo2
	clear

```

Makefile Commands:

All: Clears terminal, delete fifos and compiles program.

Compile: Compiles the code.

Clean: Clears terminal and delete fifos.

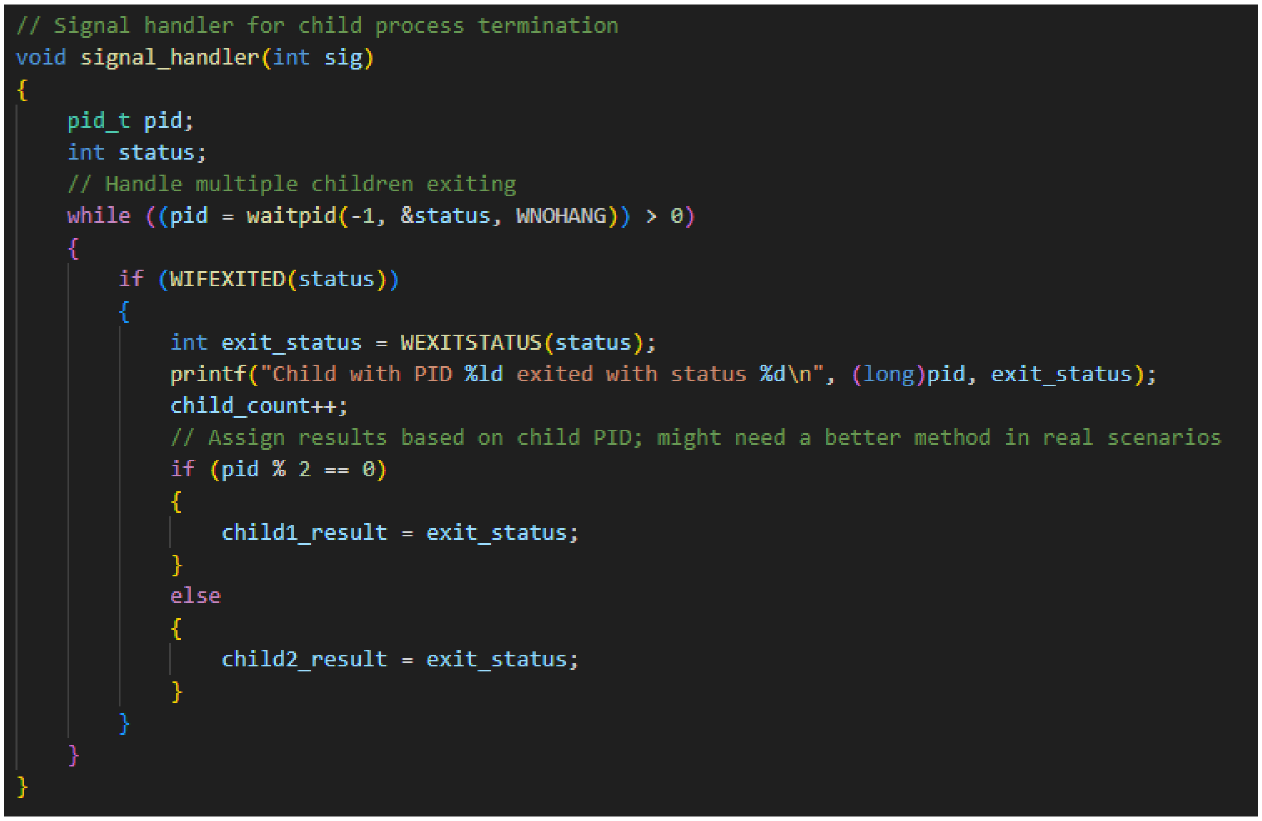
Run: Runs the program but our program runs with an argument so run command remain useless.

How to run the code?

Begin by compiling the code using the command **make all**. After compilation, execute the program by appending the array size to the program's name as an argument. For example, to run the program with an array size of 4, you would type **./main 4** in the terminal.

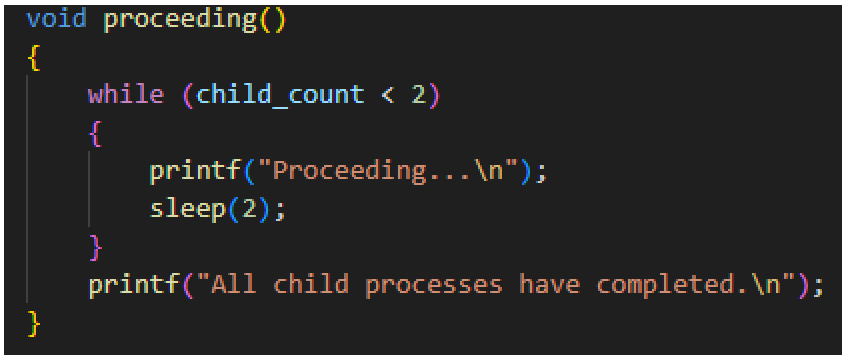
Code Explanation:

signal_handler: By catching the SIGCHLD signal, the signal_handler function is in charge of controlling the termination of child processes. The function checks for any terminated children using waitpid in a non-blocking fashion when a child process ends. For every kid who has been terminated, it collects and logs the exit status. Global variables that keep track of how many child processes have finished and their associated outcomes are then updated using this information. With this configuration, the parent process is guaranteed to stay responsive and effectively manage the outputs of its child processes.



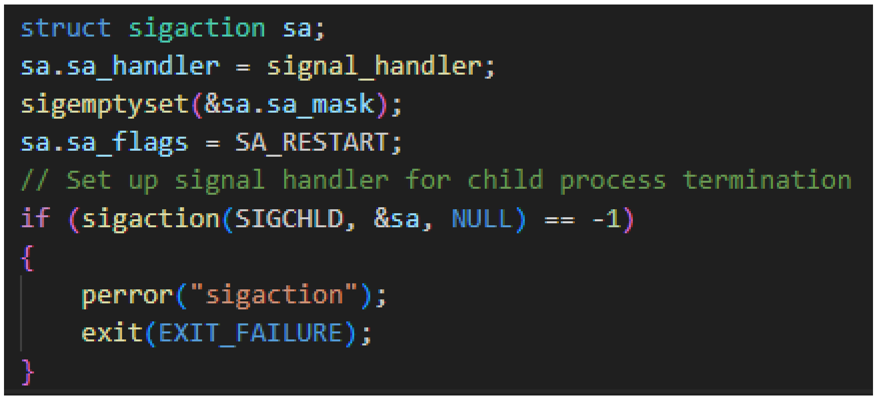
```c
// Signal handler for child process termination
void signal_handler(int sig)
{
    pid_t pid;
    int status;
    // Handle multiple children exiting
    while ((pid = waitpid(-1, &status, WNOHANG)) > 0)
    {
        if (WIFEXITED(status))
        {
            int exit_status = WEXITSTATUS(status);
            printf("Child with PID %ld exited with status %d\n", (long)pid, exit_status);
            child_count++;
            // Assign results based on child PID; might need a better method in real scenarios
            if (pid % 2 == 0)
            {
                child1_result = exit_status;
            }
            else
            {
                child2_result = exit_status;
            }
        }
    }
}
```

proceeding: The parent process is paused until this condition is satisfied by the Proceeding function, which checks if two child processes have finished. While it waits, "Proceeding..." is printed every two seconds. After both kids are done, it verifies that "All child processes have completed." and permits the parent process to carry on.



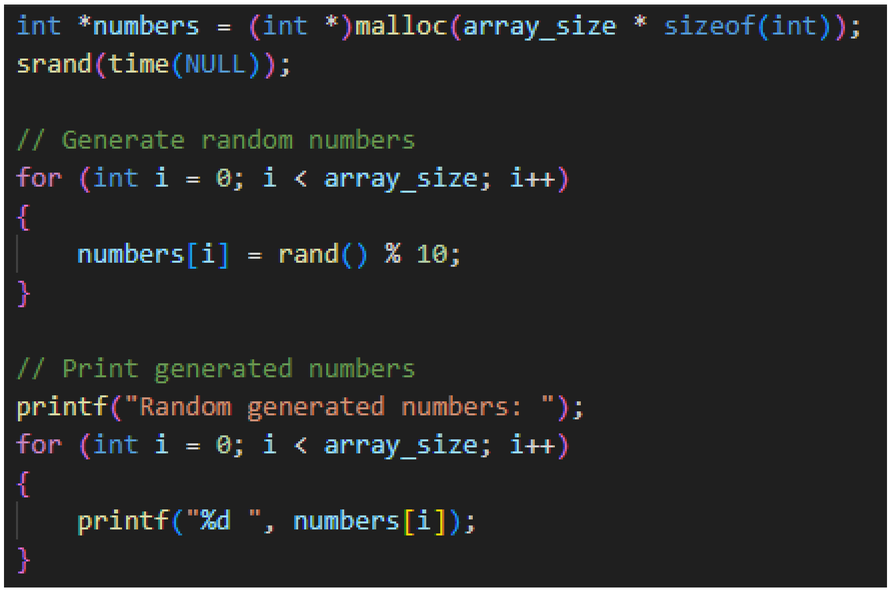
```c
void proceeding()
{
    while (child_count < 2)
    {
        printf("Proceeding...\n");
        sleep(2);
    }
    printf("All child processes have completed.\n");
}
```

Signal Handler Setup for Child Termination: The SIGCHLD signal is generated when child processes end, and this code snippet configures a signal handler for it. The sigaction structure is initialized, the signal handler function is assigned to handle SIGCHLD, any signals blocked during the handler are cleared with sigemptyset, and the SA_RESTART flag is set to guarantee that interrupted system calls are restarted. In the event that the signal handler setup is unsuccessful, the software produces an error message and ends with a failure status. By setting it up this way, signals from ending child processes are handled correctly by the parent process.



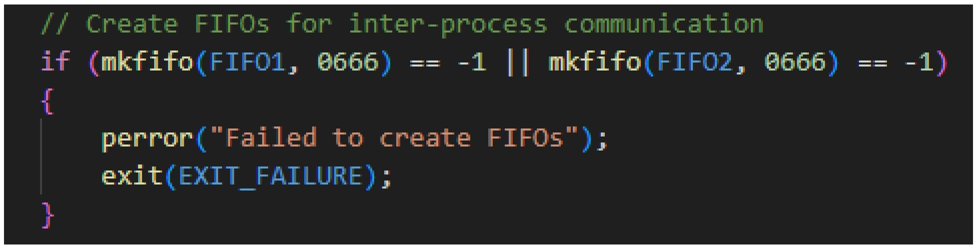
```c
struct sigaction sa;
sa.sa_handler = signal_handler;
sigemptyset(&sa.sa_mask);
sa.sa_flags = SA_RESTART;
// Set up signal handler for child process termination
if (sigaction(SIGCHLD, &sa, NULL) == -1)
{
    perror("sigaction");
    exit(EXIT_FAILURE);
}
```

Random Number Generator: Using a user-specified size (array_size), this code snippet dynamically allocates memory for an array of numbers. To guarantee diversity in the produced numbers, it initializes the random number generator with the current time. Next, a loop is used to randomly assign integers between 0 and 9 to the array. It outputs the numbers to the console after creating them. This part of the code generates and shows a collection of random integers that may be utilized for additional processing at a later time in the program.



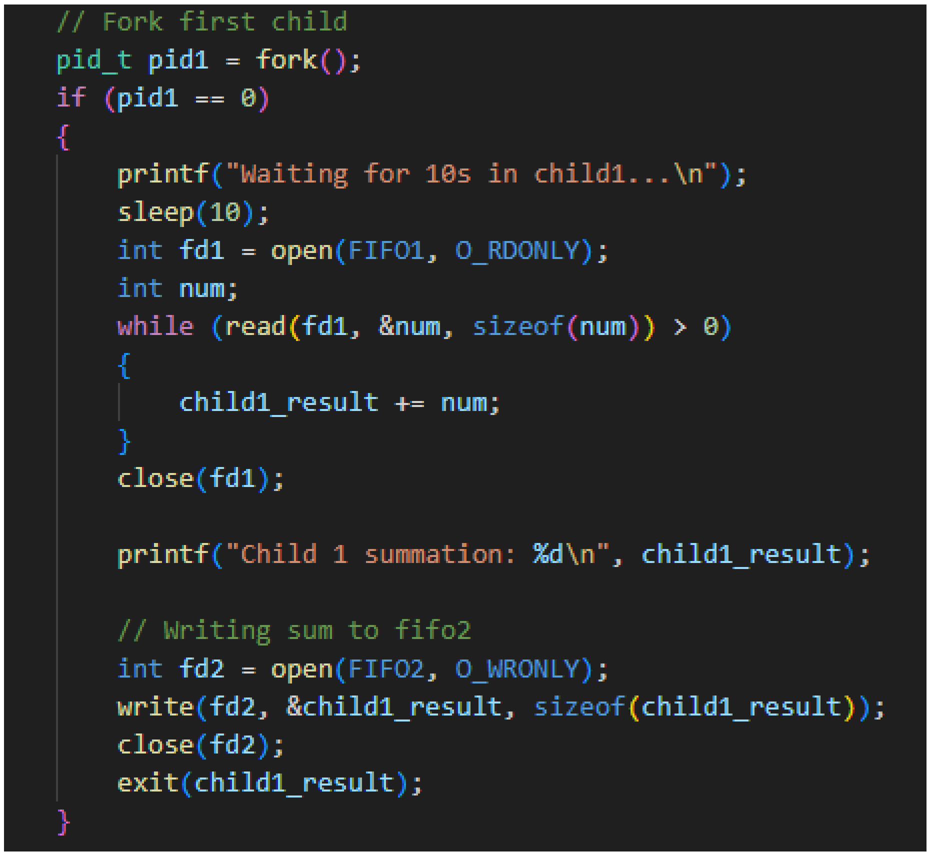
```c
int *numbers = (int *)malloc(array_size * sizeof(int));
srand(time(NULL));

// Generate random numbers
for (int i = 0; i < array_size; i++)
{
    numbers[i] = rand() % 10;
}

// Print generated numbers
printf("Random generated numbers: ");
for (int i = 0; i < array_size; i++)
{
    printf("%d ", numbers[i]);
}
```

Creating fifos: Using the file names FIFO1 and FIFO2, this section tries to build two named pipes (FIFOs) with read and write rights for all users. Should one of the FIFO construction attempts fail, the program will exit with a failure status and output an error message detailing the issue. Enabling inter-process communication between the parent process and its child processes requires this configuration.



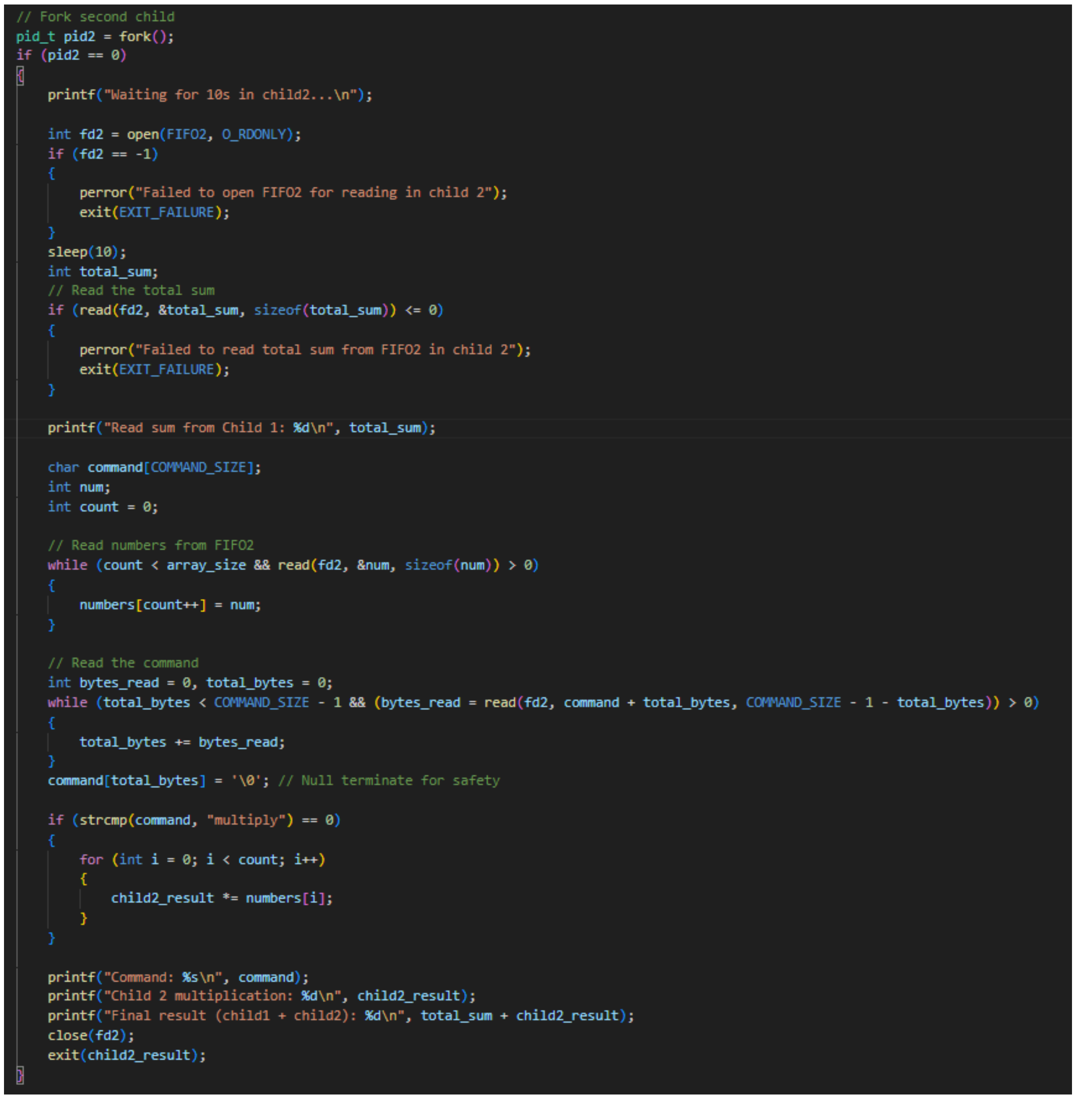
```c
// Create FIFOs for inter-process communication
if (mkfifo(FIFO1, 0666) == -1 || mkfifo(FIFO2, 0666) == -1)
{
    perror("Failed to create FIFOs");
    exit(EXIT_FAILURE);
}
```

Child 1: This portion of code uses fork() to create the first child process. If the fork is successful, a different code path is executed by the child process. It starts with a 10-second pause to simulate a delay. After that, the youngster opens FIFO1 for reading and reads numbers one at a time from it, adding them up to get the total.FIFO1 is closed once all the data has been read. The computed total is then produced and delivered to FIFO2 in case it is needed by another operation. The child process then departs, returning the total amount as its exit status, and the FIFO is closed. This procedure successfully demonstrates interprocess communication and fundamental process management by isolating a portion of the burden (number summing) to a different process.



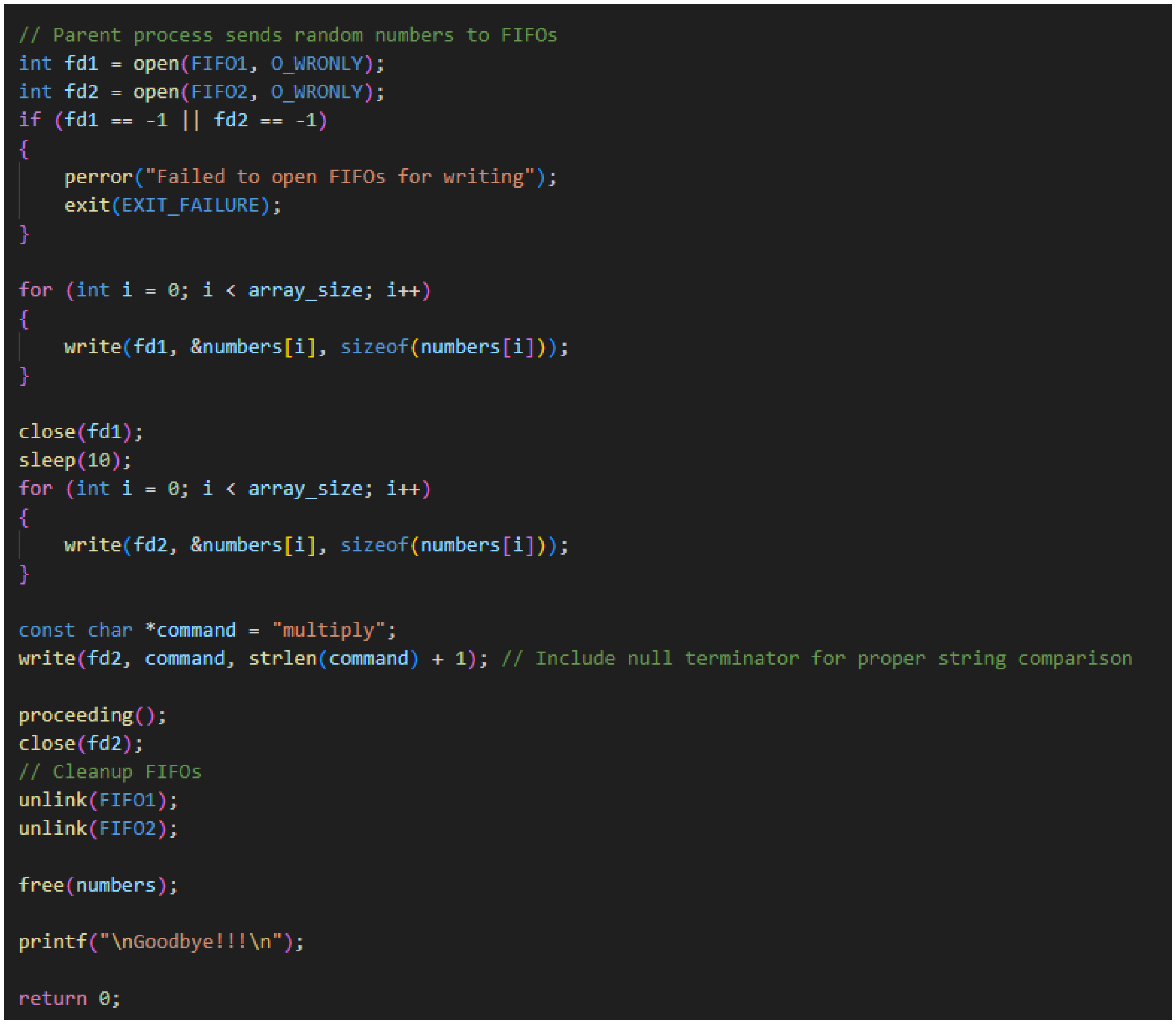
```c
// Fork first child
pid_t pid1 = fork();
if (pid1 == 0)
{
    printf("Waiting for 10s in child1...\n");
    sleep(10);
    int fd1 = open(FIFO1, O_RDONLY);
    int num;
    while (read(fd1, &num, sizeof(num)) > 0)
    {
        child1_result += num;
    }
    close(fd1);

    printf("Child 1 summation: %d\n", child1_result);

    // Writing sum to fifo2
    int fd2 = open(FIFO2, O_WRONLY);
    write(fd2, &child1_result, sizeof(child1_result));
    close(fd2);
    exit(child1_result);
}
```

Child 2: The second child process that fork() produced is managed by this code block. Similar to the first child, the child process starts after the fork by halting for 10 seconds to simulate a delay or wait for resources. After that, an attempt is made to read FIFO2. It produces an error message and quits with a failure status if it is unable to open the FIFO. The process waits once again, assuming that the FIFO opens properly, and then it reads the total sum that the first child process had already calculated and written. It starts reading more numbers from FIFO2 and outputs the obtained total for verification. In addition, the child process receives a command from FIFO2 that tells it what action to do next. When the instruction is "multiply," the method multiplies each integer that is received in order to calculate the result (child2_result).The ultimate combined result of both child processes is displayed, and the multiplication result and the command are printed. The child process departs, returning the multiplication result as its exit status, and the FIFO is closed. In this section, coordination and data transmission between processes are highlighted when the second child receives and analyzes data depending on information from the first child. This illustrates more inter-process communication.



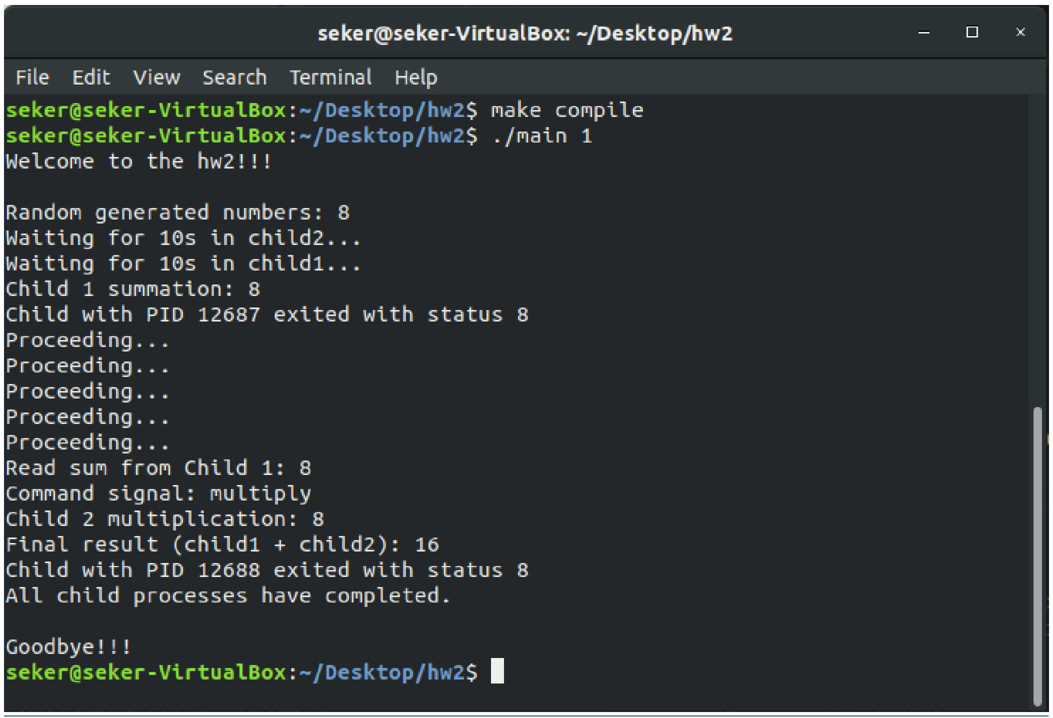
```c
// Fork second child
pid_t pid2 = fork();
if (pid2 == 0)
{
    printf("Waiting for 10s in child2...\n");

    int fd2 = open(FIFO2, O_RDONLY);
    if (fd2 == -1)
    {
        perror("Failed to open FIFO2 for reading in child 2");
        exit(EXIT_FAILURE);
    }
    sleep(10);
    int total_sum;
    // Read the total sum
    if (read(fd2, &total_sum, sizeof(total_sum)) <= 0)
    {
        perror("Failed to read total sum from FIFO2 in child 2");
        exit(EXIT_FAILURE);
    }

    printf("Read sum from Child 1: %d\n", total_sum);

    char command[COMMAND_SIZE];
    int num;
    int count = 0;

    // Read numbers from FIFO2
    while (count < array_size && read(fd2, &num, sizeof(num)) > 0)
    {
        numbers[count++] = num;
    }

    // Read the command
    int bytes_read = 0, total_bytes = 0;
    while (total_bytes < COMMAND_SIZE - 1 && (bytes_read = read(fd2, command + total_bytes, COMMAND_SIZE - 1 - total_bytes)) > 0)
    {
        total_bytes += bytes_read;
    }
    command[total_bytes] = '\0'; // Null terminate for safety

    if (strcmp(command, "multiply") == 0)
    {
        for (int i = 0; i < count; i++)
        {
            child2_result *= numbers[i];
        }
    }

    printf("Command: %s\n", command);
    printf("Child 2 multiplication: %d\n", child2_result);
    printf("Final result (child1 + child2): %d\n", total_sum + child2_result);
    close(fd2);
    exit(child2_result);
}
```

Parent Process: The parent process manages a number of crucial responsibilities related to inter-process synchronization and communication in this section of the program. When writing, it first opens two FIFOs, FIFO1 and FIFO2, making sure that errors are handled appropriately in case one of them cannot be opened. After that, it closes FIFO1, puts a string of random integers there, and then, after a little delay, repeats the same for FIFO2. The parent transmits the instruction "multiply" along with the numbers, making sure to include the null terminator so that the receiving child can process the data accurately. In order to synchronize the parent process with the completion of the child processes and make sure that all children have completed their responsibilities prior to the program moving forward, the continuing() function is called after the transmission. Lastly, the parent releases the memory allotted for the numbers, shuts FIFO2, cleans up by unlinking both FIFOs, and finishes the program with a closing message. The orderly startup, operation, and shutdown of all relevant processes is made possible by this flow, which guarantees that the parent retains control over resource management and process synchronization.

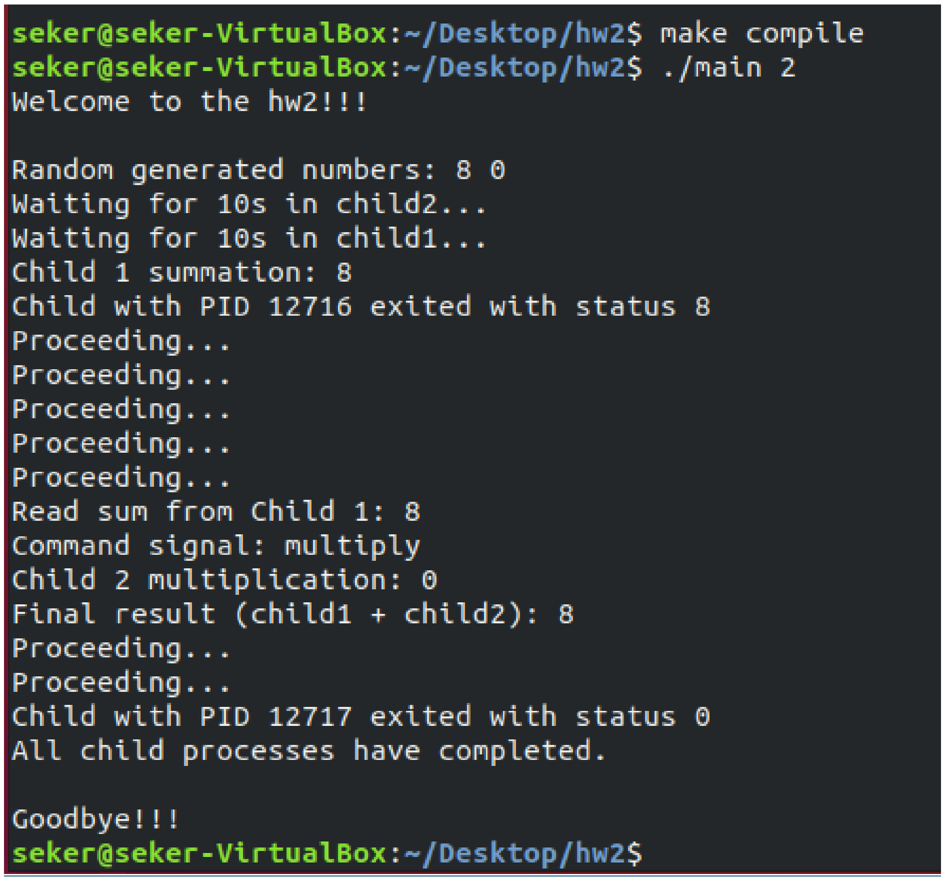


```c
// Parent process sends random numbers to FIFOs
int fd1 = open(FIFO1, O_WRONLY);
int fd2 = open(FIFO2, O_WRONLY);
if (fd1 == -1 || fd2 == -1)
{
    perror("Failed to open FIFOs for writing");
    exit(EXIT_FAILURE);
}

for (int i = 0; i < array_size; i++)
{
    write(fd1, &numbers[i], sizeof(numbers[i]));
}

close(fd1);
sleep(10);
for (int i = 0; i < array_size; i++)
{
    write(fd2, &numbers[i], sizeof(numbers[i]));
}

const char *command = "multiply";
write(fd2, command, strlen(command) + 1); // Include null terminator for proper string comparison

proceeding();
close(fd2);
// Cleanup FIFOs
unlink(FIFO1);
unlink(FIFO2);

free(numbers);

printf("\nGoodbye!!!\n");

return 0;
```

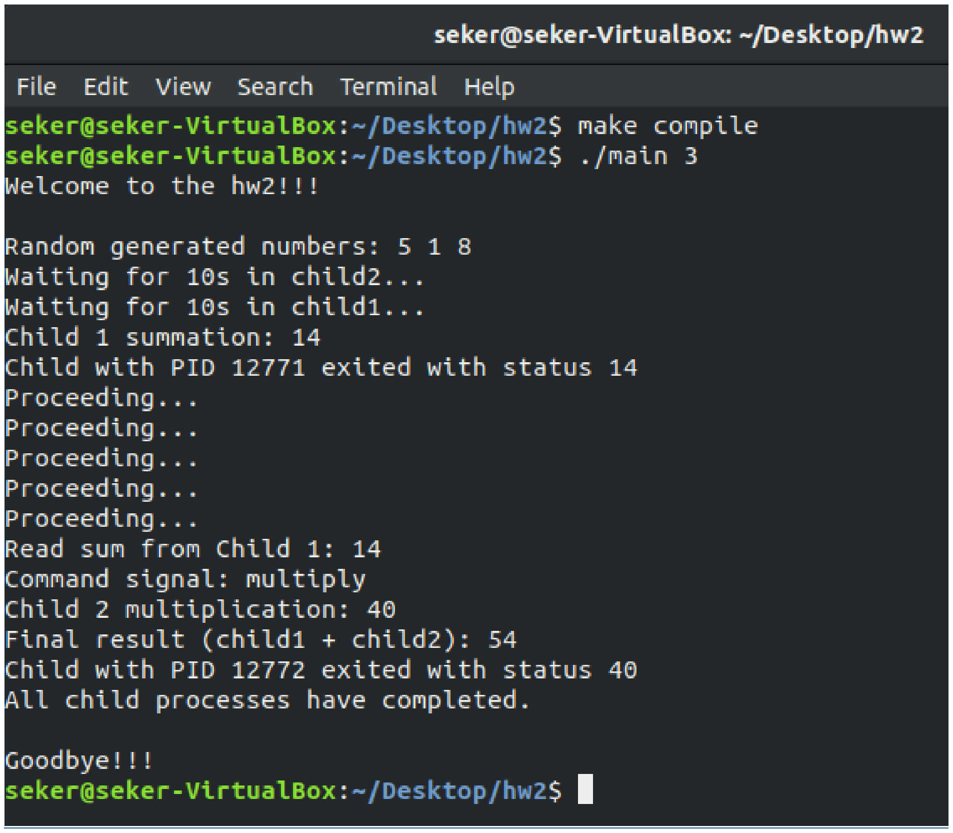
Example Outputs:


seker@seker-VirtualBox: ~/Desktop/hw2
File Edit View Search Terminal Help

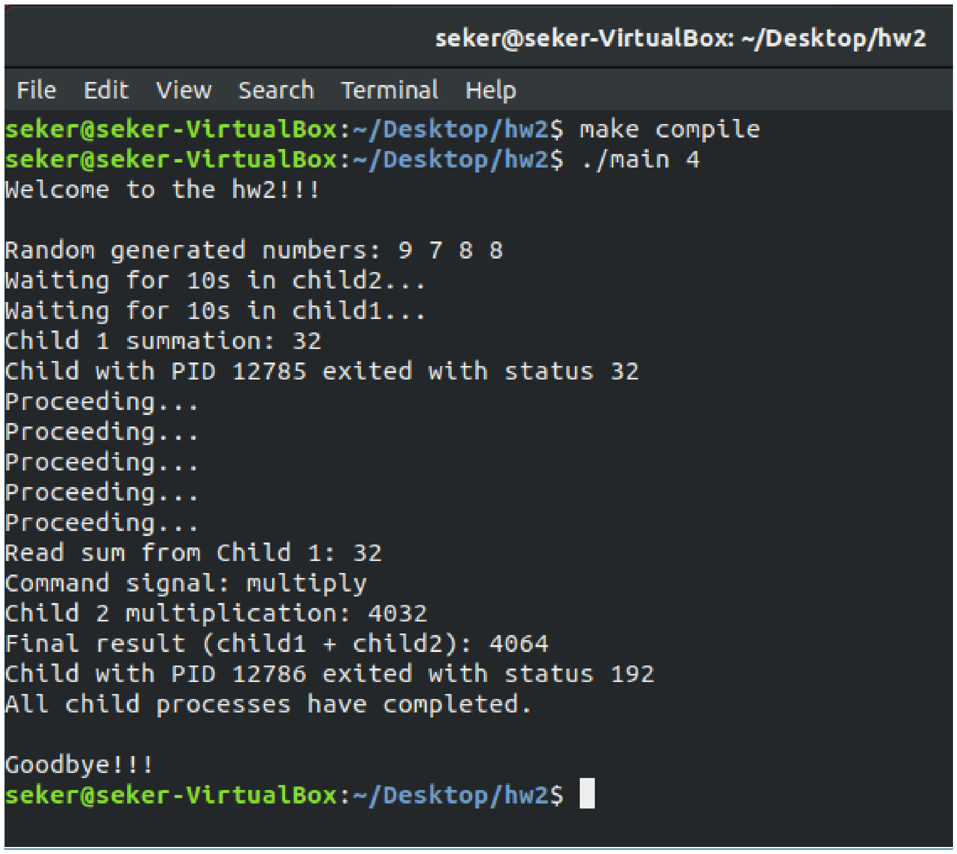
```
seker@seker-VirtualBox:~/Desktop/hw2$ make compile
seker@seker-VirtualBox:~/Desktop/hw2$ ./main 1
Welcome to the hw2!!!

Random generated numbers: 8
Waiting for 10s in child2...
Waiting for 10s in child1...
Child 1 summation: 8
Child with PID 12687 exited with status 8
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Read sum from Child 1: 8
Command signal: multiply
Child 2 multiplication: 8
Final result (child1 + child2): 16
Child with PID 12688 exited with status 8
All child processes have completed.

Goodbye!!!
seker@seker-VirtualBox:~/Desktop/hw2$
```



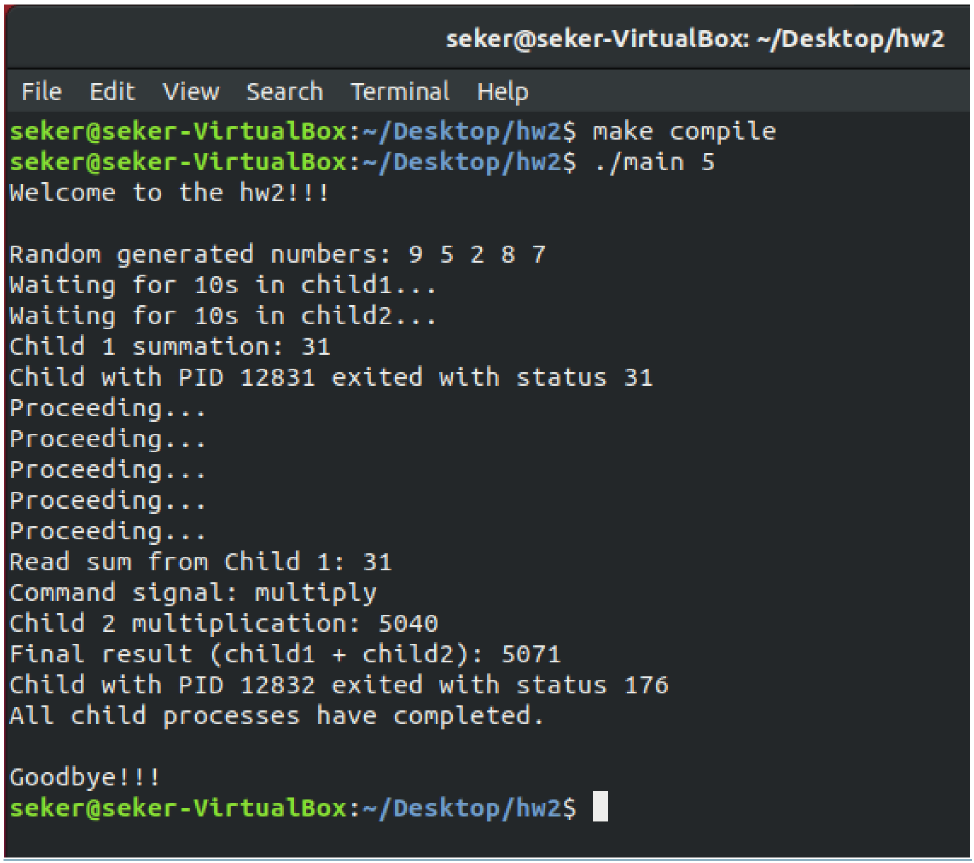
```
seker@seker-VirtualBox:~/Desktop/hw2$ make compile
seker@seker-VirtualBox:~/Desktop/hw2$ ./main 2
Welcome to the hw2!!!

Random generated numbers: 8 0
Waiting for 10s in child2...
Waiting for 10s in child1...
Child 1 summation: 8
Child with PID 12716 exited with status 8
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Read sum from Child 1: 8
Command signal: multiply
Child 2 multiplication: 0
Final result (child1 + child2): 8
Proceeding...
Proceeding...
Child with PID 12717 exited with status 0
All child processes have completed.

Goodbye!!!
seker@seker-VirtualBox:~/Desktop/hw2$
```


seker@seker-VirtualBox: ~/Desktop/hw2
File Edit View Search Terminal Help

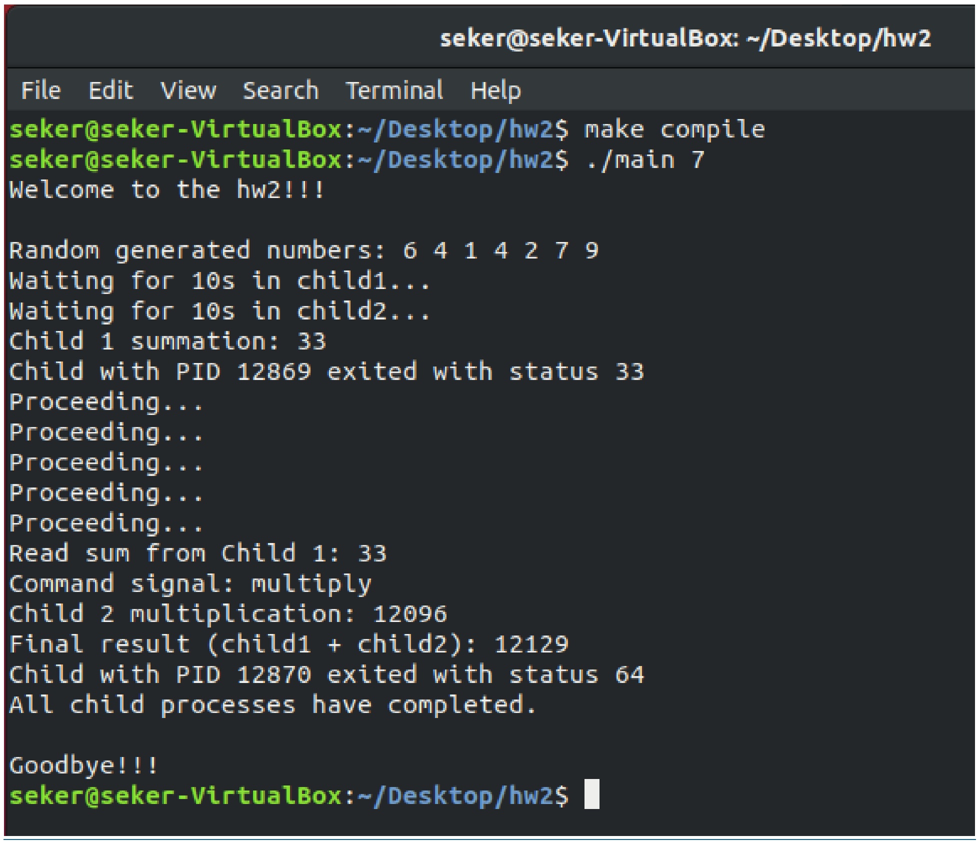
```
seker@seker-VirtualBox:~/Desktop/hw2$ make compile
seker@seker-VirtualBox:~/Desktop/hw2$ ./main 3
Welcome to the hw2!!!

Random generated numbers: 5 1 8
Waiting for 10s in child2...
Waiting for 10s in child1...
Child 1 summation: 14
Child with PID 12771 exited with status 14
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Read sum from Child 1: 14
Command signal: multiply
Child 2 multiplication: 40
Final result (child1 + child2): 54
Child with PID 12772 exited with status 40
All child processes have completed.

Goodbye!!!
seker@seker-VirtualBox:~/Desktop/hw2$
```


seker@seker-VirtualBox: ~/Desktop/hw2
File Edit View Search Terminal Help

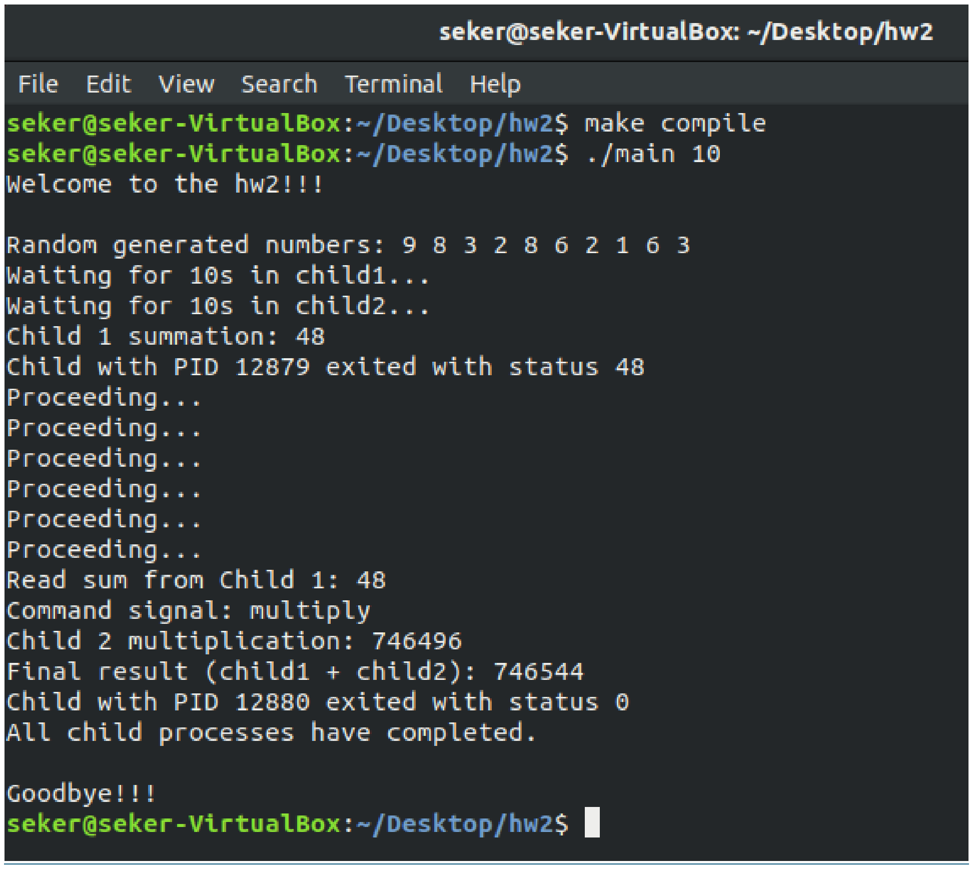
```
seker@seker-VirtualBox:~/Desktop/hw2$ make compile
seker@seker-VirtualBox:~/Desktop/hw2$ ./main 4
Welcome to the hw2!!!

Random generated numbers: 9 7 8 8
Waiting for 10s in child2...
Waiting for 10s in child1...
Child 1 summation: 32
Child with PID 12785 exited with status 32
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Read sum from Child 1: 32
Command signal: multiply
Child 2 multiplication: 4032
Final result (child1 + child2): 4064
Child with PID 12786 exited with status 192
All child processes have completed.

Goodbye!!!
seker@seker-VirtualBox:~/Desktop/hw2$
```


seker@seker-VirtualBox: ~/Desktop/hw2
File Edit View Search Terminal Help

```
seker@seker-VirtualBox:~/Desktop/hw2$ make compile
seker@seker-VirtualBox:~/Desktop/hw2$ ./main 5
Welcome to the hw2!!!

Random generated numbers: 9 5 2 8 7
Waiting for 10s in child1...
Waiting for 10s in child2...
Child 1 summation: 31
Child with PID 12831 exited with status 31
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Read sum from Child 1: 31
Command signal: multiply
Child 2 multiplication: 5040
Final result (child1 + child2): 5071
Child with PID 12832 exited with status 176
All child processes have completed.

Goodbye!!!
seker@seker-VirtualBox:~/Desktop/hw2$
```

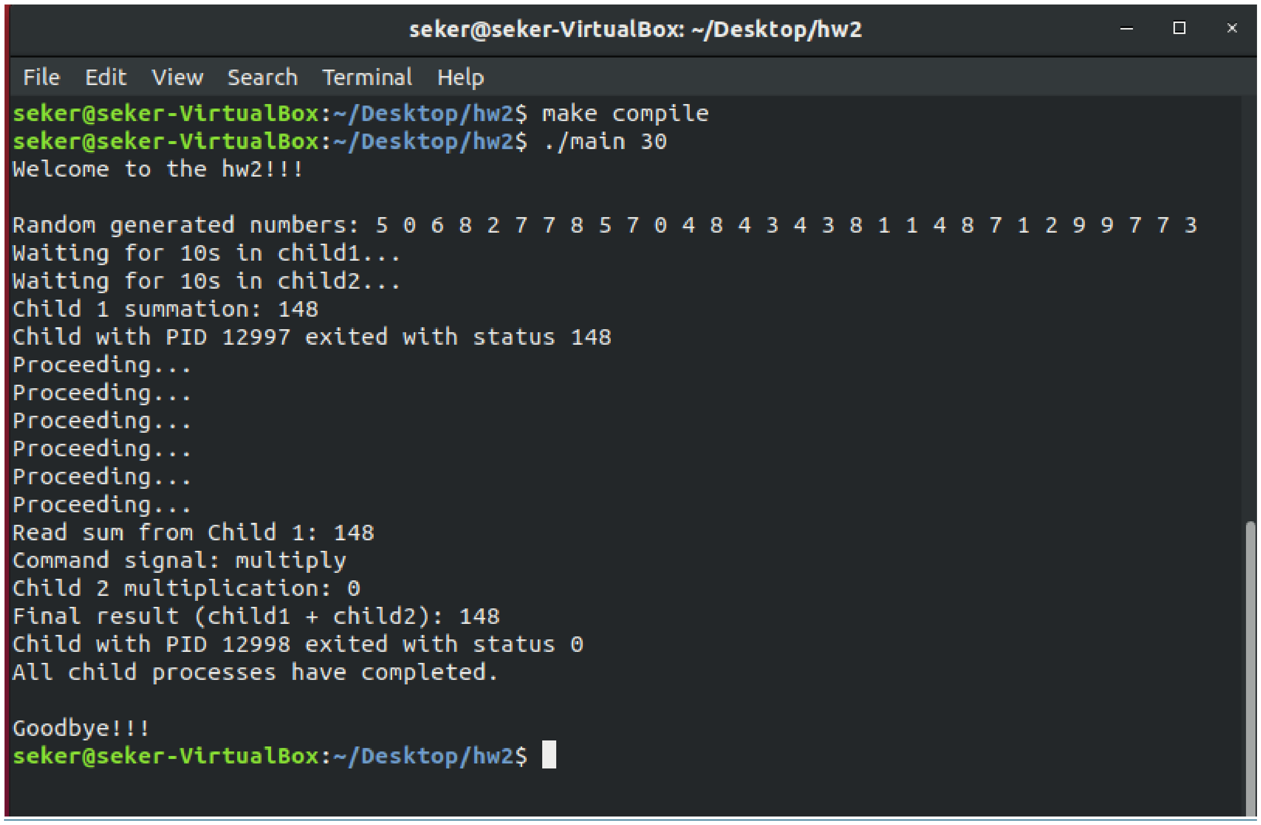

seker@seker-VirtualBox: ~/Desktop/hw2
File Edit View Search Terminal Help

```
seker@seker-VirtualBox:~/Desktop/hw2$ make compile
seker@seker-VirtualBox:~/Desktop/hw2$ ./main 7
Welcome to the hw2!!!

Random generated numbers: 6 4 1 4 2 7 9
Waiting for 10s in child1...
Waiting for 10s in child2...
Child 1 summation: 33
Child with PID 12869 exited with status 33
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Read sum from Child 1: 33
Command signal: multiply
Child 2 multiplication: 12096
Final result (child1 + child2): 12129
Child with PID 12870 exited with status 64
All child processes have completed.

Goodbye!!!
seker@seker-VirtualBox:~/Desktop/hw2$
```



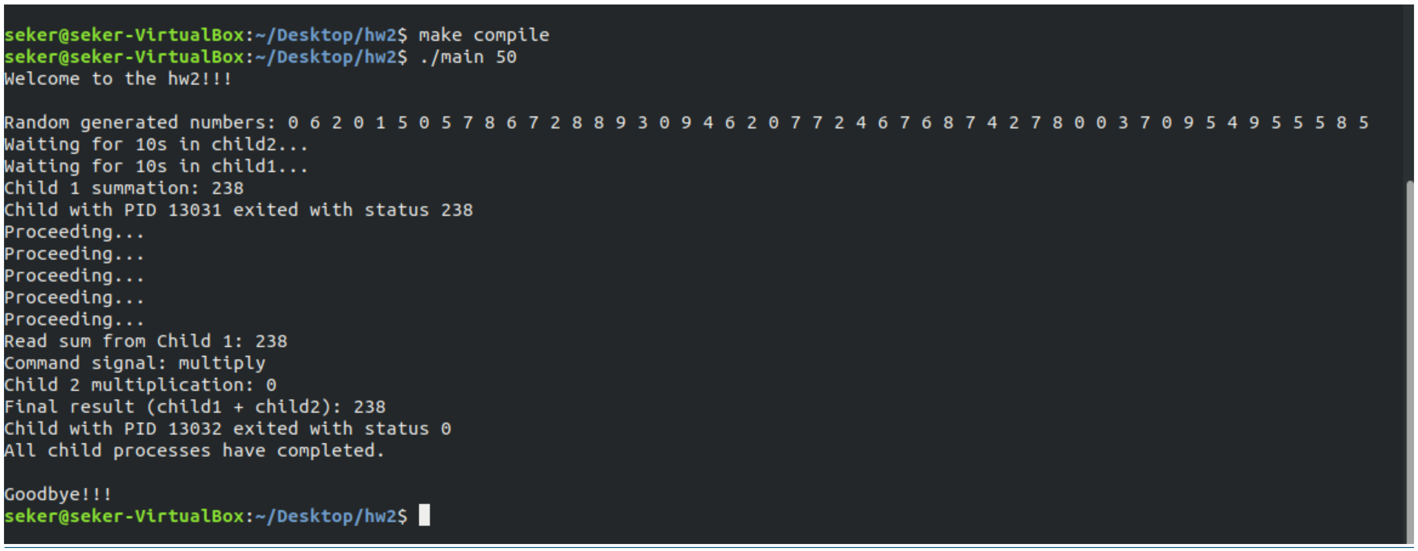
seker@seker-VirtualBox: ~/Desktop/hw2
File Edit View Search Terminal Help

```
seker@seker-VirtualBox:~/Desktop/hw2$ make compile
seker@seker-VirtualBox:~/Desktop/hw2$ ./main 10
Welcome to the hw2!!!

Random generated numbers: 9 8 3 2 8 6 2 1 6 3
Waiting for 10s in child1...
Waiting for 10s in child2...
Child 1 summation: 48
Child with PID 12879 exited with status 48
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Read sum from Child 1: 48
Command signal: multiply
Child 2 multiplication: 746496
Final result (child1 + child2): 746544
Child with PID 12880 exited with status 0
All child processes have completed.

Goodbye!!!
seker@seker-VirtualBox:~/Desktop/hw2$
```

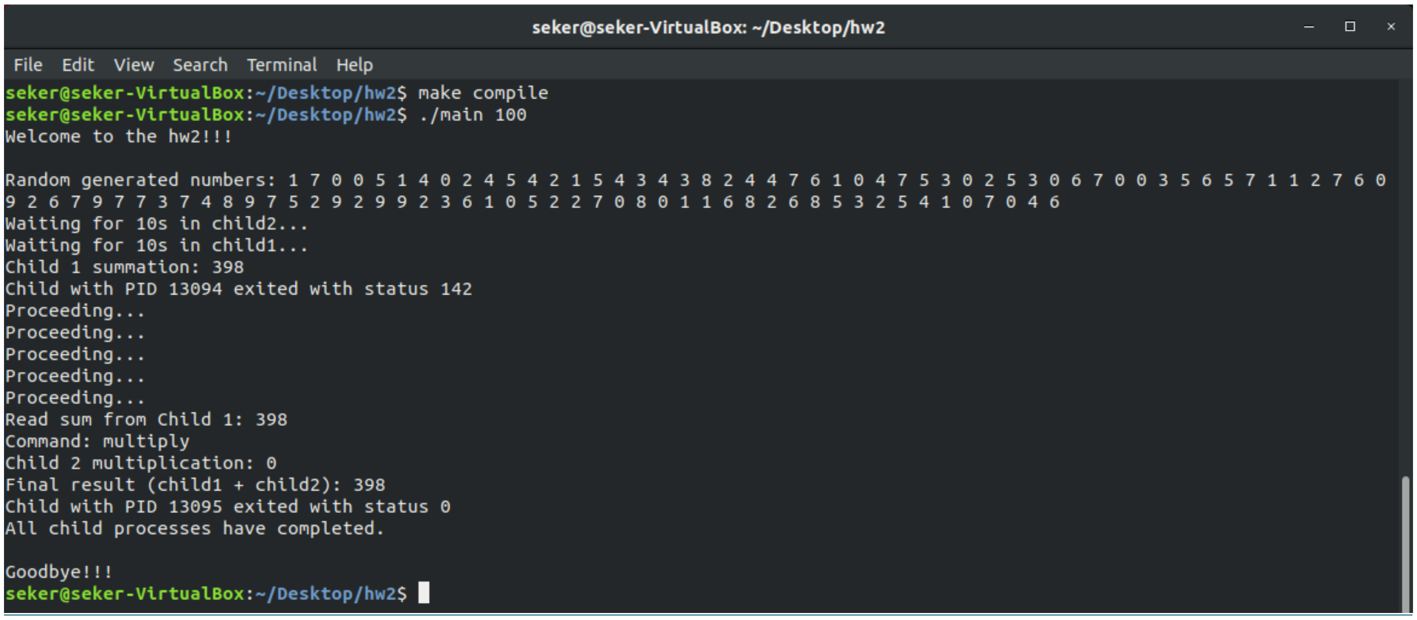


seker@seker-VirtualBox: ~/Desktop/hw2
File Edit View Search Terminal Help

```
seker@seker-VirtualBox:~/Desktop/hw2$ make compile
seker@seker-VirtualBox:~/Desktop/hw2$ ./main 30
Welcome to the hw2!!!

Random generated numbers: 5 0 6 8 2 7 7 8 5 7 0 4 8 4 3 4 3 8 1 1 4 8 7 1 2 9 9 7 7 3
Waiting for 10s in child1...
Waiting for 10s in child2...
Child 1 summation: 148
Child with PID 12997 exited with status 148
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Read sum from Child 1: 148
Command signal: multiply
Child 2 multiplication: 0
Final result (child1 + child2): 148
Child with PID 12998 exited with status 0
All child processes have completed.

Goodbye!!!
seker@seker-VirtualBox:~/Desktop/hw2$
```



```
seker@seker-VirtualBox:~/Desktop/hw2$ make compile
seker@seker-VirtualBox:~/Desktop/hw2$ ./main 50
Welcome to the hw2!!!

Random generated numbers: 0 6 2 0 1 5 0 5 7 8 6 7 2 8 8 9 3 0 9 4 6 2 0 7 7 2 4 6 7 6 8 7 4 2 7 8 0 0 3 7 0 9 5 4 9 5 5 5 8 5
Waiting for 10s in child2...
Waiting for 10s in child1...
Child 1 summation: 238
Child with PID 13031 exited with status 238
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Read sum from Child 1: 238
Command signal: multiply
Child 2 multiplication: 0
Final result (child1 + child2): 238
Child with PID 13032 exited with status 0
All child processes have completed.

Goodbye!!!
seker@seker-VirtualBox:~/Desktop/hw2$
```



```
seker@seker-VirtualBox:~/Desktop/hw2$ make compile
seker@seker-VirtualBox:~/Desktop/hw2$ ./main 100
Welcome to the hw2!!!

Random generated numbers: 1 7 0 0 5 1 4 0 2 4 5 4 2 1 5 4 3 4 3 8 2 4 4 7 6 1 0 4 7 5 3 0 2 5 3 0 6 7 0 0 3 5 6 5 7 1 1 2 7 6 0
9 2 6 7 9 7 7 3 7 4 8 9 7 5 2 9 2 9 9 2 3 6 1 0 5 2 2 7 0 8 0 1 1 6 8 2 6 8 5 3 2 5 4 1 0 7 0 4 6
Waiting for 10s in child2...
Waiting for 10s in child1...
Child 1 summation: 398
Child with PID 13094 exited with status 142
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Read sum from Child 1: 398
Command: multiply
Child 2 multiplication: 0
Final result (child1 + child2): 398
Child with PID 13095 exited with status 0
All child processes have completed.

Goodbye!!!
seker@seker-VirtualBox:~/Desktop/hw2$
```