CSE 344 SYSTEM PROGRAMMING HOMEWORK 2 REPORT

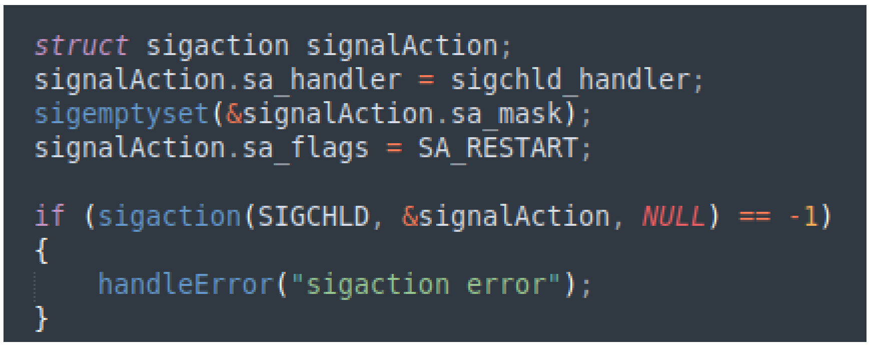
I'll start by explaining some of the things I implemented in the assignment: firstly, I get the size of the random array with read(STDIN_FILENO, sizeBuffer, COMMAND_SIZE);, then I convert it to integer with strtol() and use this number wherever necessary. It is also checked if this entry may be incorrect. Then the second child process will receive the command argument with command, read(STDIN_FILENO, COMMAND_SIZE); . At this point, I have to say that the second child process is only open to do addition and multiplication (because it seemed illogical and unnecessary to apply subtraction and division to the elements of a random array). For such requests, I expect multiply inputs for summation, summation and multiplication on the command line. Other inputs are overridden by the following code block. FIFO writes above a certain number are blocked by the operating system. Because every operating system has a limit to write to FIFO. In my system, this number was 16600, so I suppress an error for entries over 16000 (MAX_FIFO_WRITE).

* When the programme starts, it first waits for the command to be given to the second process and then the size of the random array as input from the user.

```
int isCommand(const char * command)

{

    if (strcmp(command, "multiply") != 0 && strcmp(command, "summation") != 0)

    {

        return FALSE;

    }

    return TRUE;

}
```

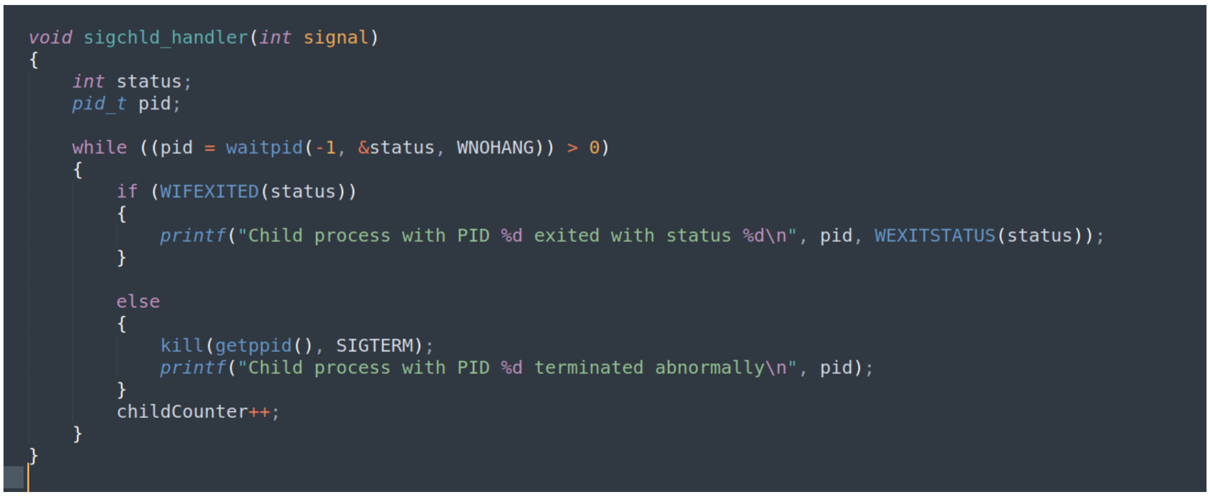
Then I create an array using malloc up to the size of the number I receive as input. I create random numbers between 1 and 10 in order to better control the outputs. In cases where a large array is created, I did not include 0 so that the multiplication result is not always 0.

```
void makeRandom(int * randomNumbers, int arraySize){

   if (randomNumbers == NULL)    {

      handleError("Memory allocation error!");

   }

   srand(time(NULL));

   for (int i = 0; i < arraySize; ++i)    {

      randomNumbers[i] = 1 + rand() % 10;

   }

}
```



```c
struct sigaction signalAction;
signalAction.sa_handler = sigchld_handler;
sigemptyset(&signalAction.sa_mask);
signalAction.sa_flags = SA_RESTART;

if (sigaction(SIGCHLD, &signalAction, NULL) == -1)
{
    handleError("sigaction error");
}
```

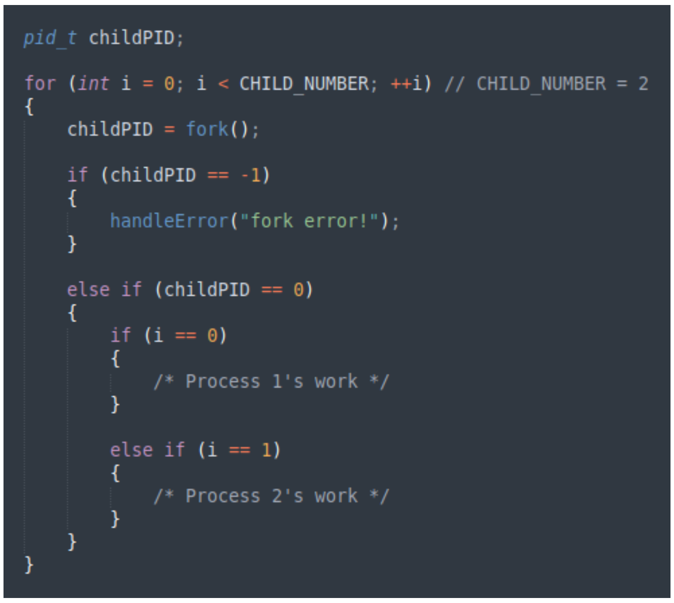
In the above code block, I provide the handling of the SIGCHILD signal with the sigaction structure. Here, the function sigchld_handler is assigned to the relevant part of the structure. Below is the code of the signal handler function.



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

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

        else
        {
            kill(getppid(), SIGTERM);
            printf("Child process with PID %d terminated abnormally\n", pid);
        }
        childCounter++;
    }
}
```

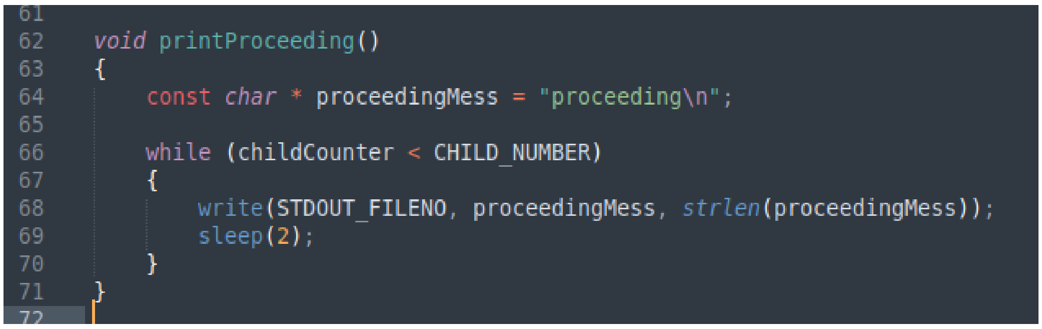
Here, the end state of the process is handled if it ends. If the process finished without any problems, I print the pid of the finished process and its completion status on the screen. If the process finished unexpectedly, I indicate this by printing the pid on the screen again. If there is a problem in the child processes and it is necessary to exit with exit. I end the parent process at the same time, get the id of the parent process with getppid() and end with the SIGTERM signal. Then I increment the childCounter variable, which I defined in the header file as 'extern', by one. This variable is used for the "proceeding" message written to the screen.

In the next step, after creating the two desired fifo files with mkfifo, I create two child processes with the fork() function. Below is a general structure of the code block with fork() calls.



```c
pid_t childPID;

for (int i = 0; i < CHILD_NUMBER; ++i) // CHILD_NUMBER = 2
{
    childPID = fork();

    if (childPID == -1)
    {
        handleError("fork error!");
    }

    else if (childPID == 0)
    {
        if (i == 0)
        {
            /* Process 1's work */
        }

        else if (i == 1)
        {
            /* Process 2's work */
        }
    }
}
```

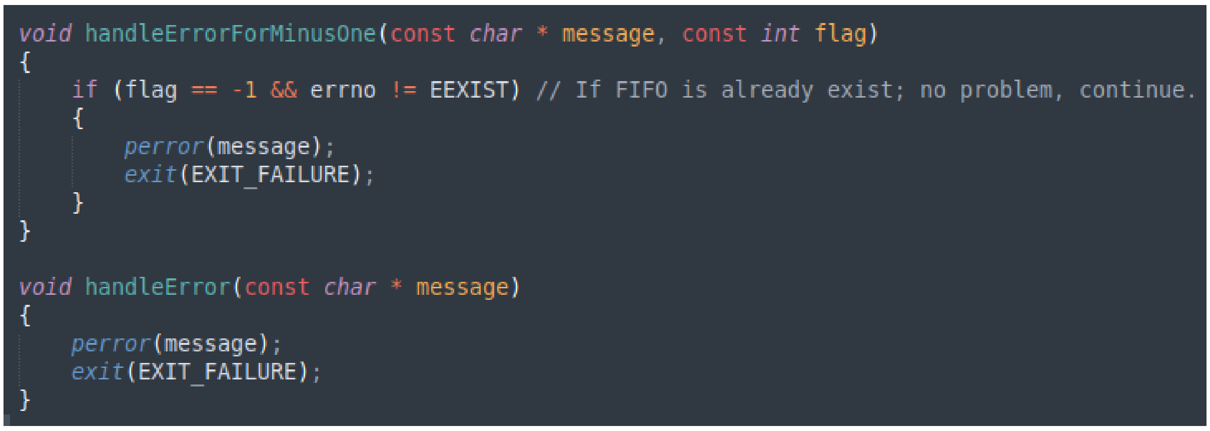
When I first started the assignment, I was opening the FIFO files with the O_RDWR flag, so that the program would not be blocked because both ends of the FIFOs were opened. However, in this case FIFO files would be no different from normal files. Therefore, I decided to open FIFO files with O_RDONLY or O_WRONLY flags when necessary and close them when I was done. But here I was facing the following problem; when I tried to write to FIFOs for the first time in the parent process before fork calls, the program was blocked there because the read end of the FIFO file was not open. In other words, another process had to open the read end so that the programme could continue. After realising this, I moved my first attempt to open the FIFOs in the parent to after the fork calls. Thus, I was able to open the read end as there would be different processes in the different program and I was able to use the FIFOs properly with a single program.



```c
void printProceeding()
{
    const char * proceedingMess = "proceeding\n";

    while (childCounter < CHILD_NUMBER)
    {
        write(STDOUT_FILENO, proceedingMess, strlen(proceedingMess));
        sleep(2);
    }
}
```

After all these operations, the function that will print the "proceeding" message on the screen comes. This function prints the desired message to the screen every 2 seconds while the other processes start working (during the 10 second sleep period) and continue to do their work. This process continues until the two child processes finish their work and call the signal handler. I put this condition inside the while loop with the childCounter++ line at the end of the sigchld_handler() function that I explained in the previous pages.

I use the two functions shown below to handle errors. I use one for system calls that return -1 in error conditions, and the other for error conditions other than system calls. Error checks were performed for all system calls. Also, as seen in the code, if by chance the FIFO files that are being created are already in the directory in which they are located, I do not throw an error for this case. I ignore the cases where errno takes the value EEXIST.

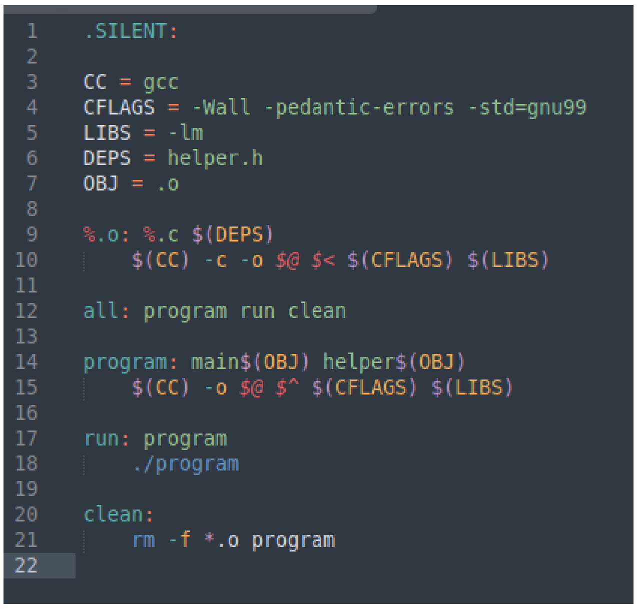


```
void handleErrorForMinusOne(const char * message, const int flag)
{
    if (flag == -1 && errno != EEXIST) // If FIFO is already exist; no problem, continue.
    {
        perror(message);
        exit(EXIT_FAILURE);
    }
}

void handleError(const char * message)
{
    perror(message);
    exit(EXIT_FAILURE);
}
```

At the end of the program, I delete the FIFO files I created with the unlink call, free the resources I dynamically allocated in memory and end the program completely with return EXIT_SUCCESS.

Both of the parts with extra points mentioned in the PDF have been realised as requested. The exit status of all processes is displayed on the screen (including the parent process) and zombie processes are prevented with the waitpid() function.

Make file ans usage:



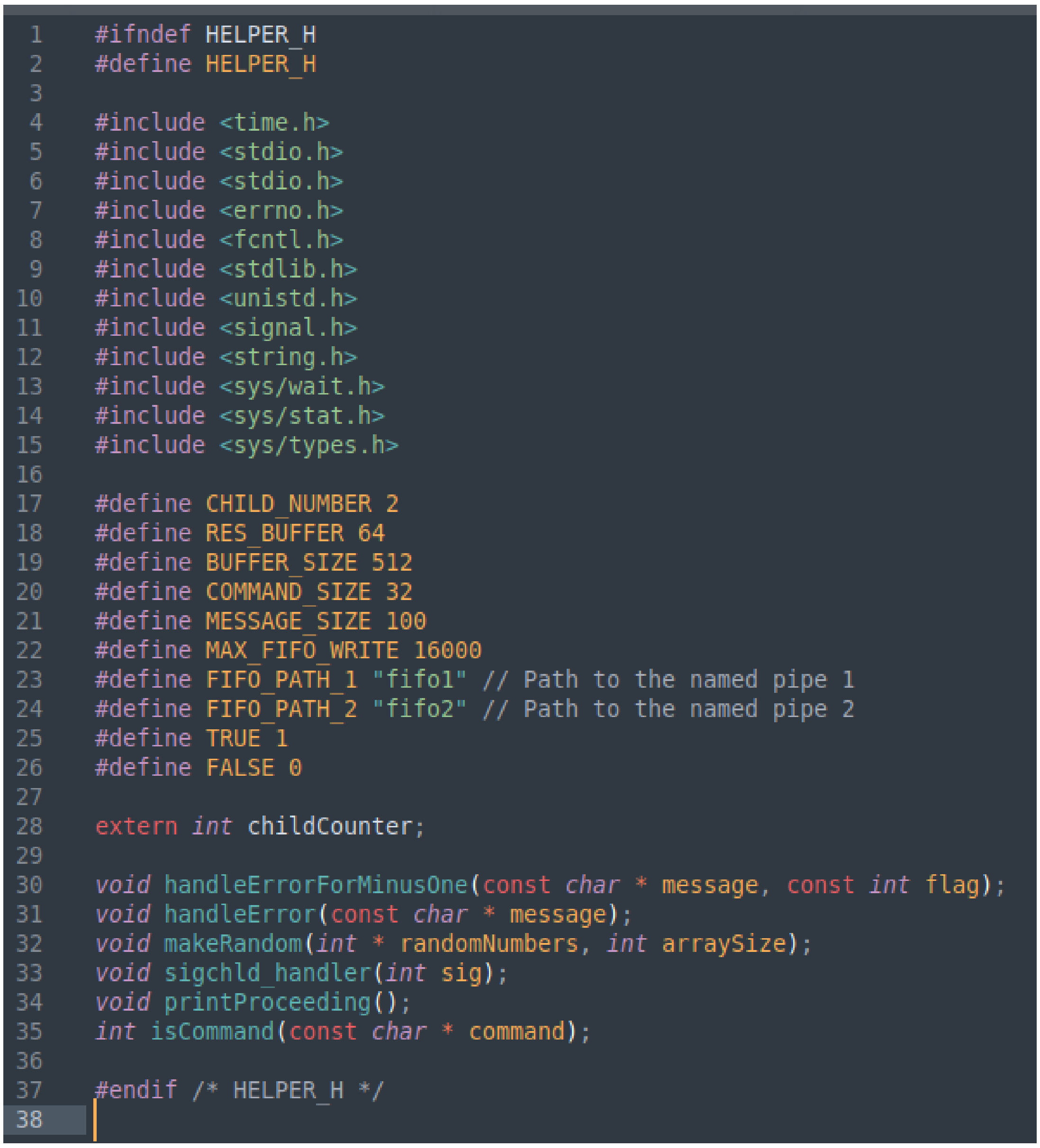
```
.SILENT:

CC = gcc
CFLAGS = -Wall -pedantic-errors -std=gnu99
LIBS = -lm
DEPS = helper.h
OBJ = .o

%.o: %.c $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS) $(LIBS)

all: program run clean

program: main$(OBJ) helper$(OBJ)
	$(CC) -o $@ $^ $(CFLAGS) $(LIBS)

run: program
	./program

clean:
	rm -f *.o program

```

make program -> compiles the program
make run -> runs the program
make clean -> clean the executable files and .o files

make -> compile and run at the same time

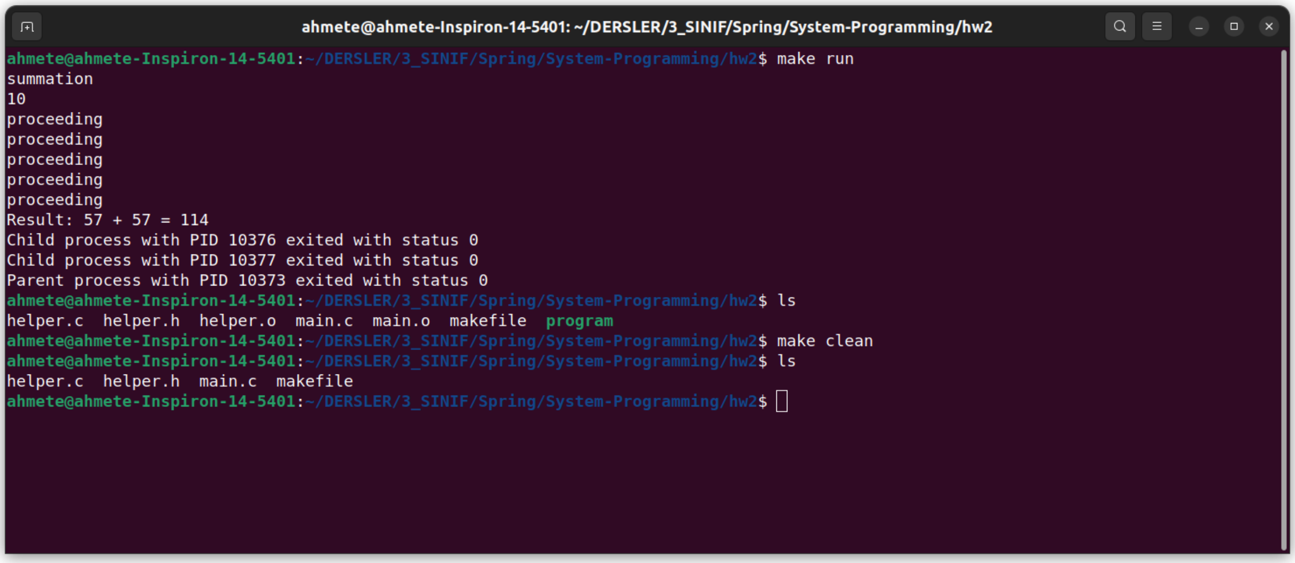
Header File:

Here you can see the functions I use, their arguments, the variables I use externally and some of my constants.

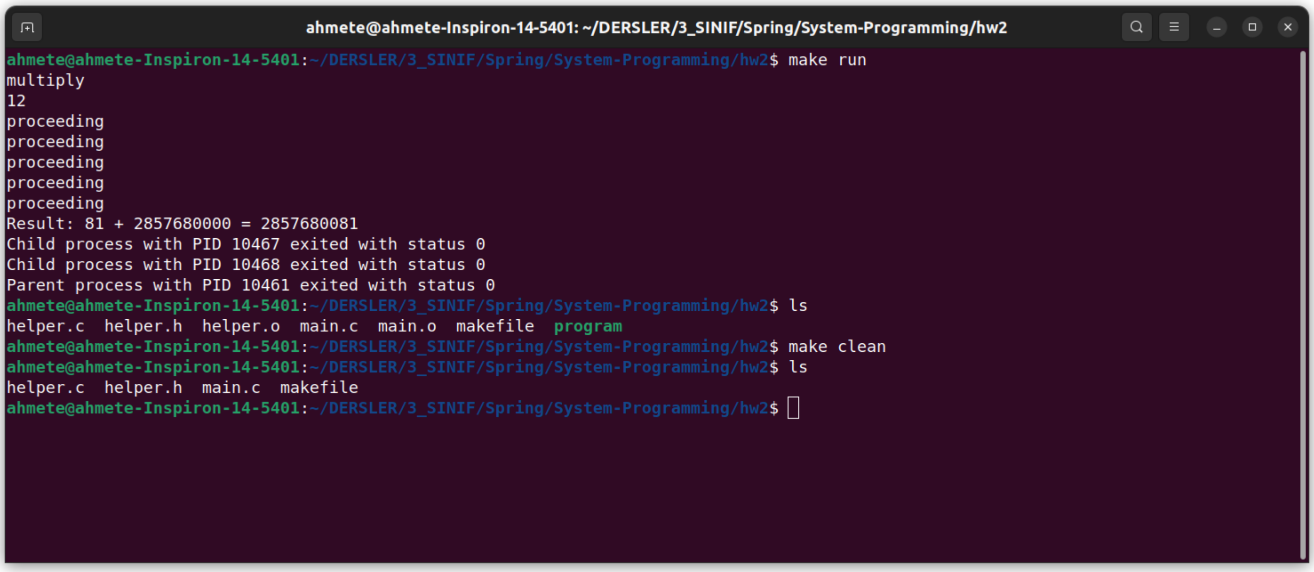


```c
#ifndef HELPER_H
#define HELPER_H

#include <time.h>
#include <stdio.h>
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <sys/types.h>

#define CHILD_NUMBER 2
#define RES_BUFFER 64
#define BUFFER_SIZE 512
#define COMMAND_SIZE 32
#define MESSAGE_SIZE 100
#define MAX_FIFO_WRITE 16000
#define FIFO_PATH_1 "fifo1" // Path to the named pipe 1
#define FIFO_PATH_2 "fifo2" // Path to the named pipe 2
#define TRUE 1
#define FALSE 0

extern int childCounter;

void handleErrorForMinusOne(const char * message, const int flag);
void handleError(const char * message);
void makeRandom(int * randomNumbers, int arraySize);
void sigchld_handler(int sig);
void printProceeding();
int isCommand(const char * command);

#endif /* HELPER_H */

```

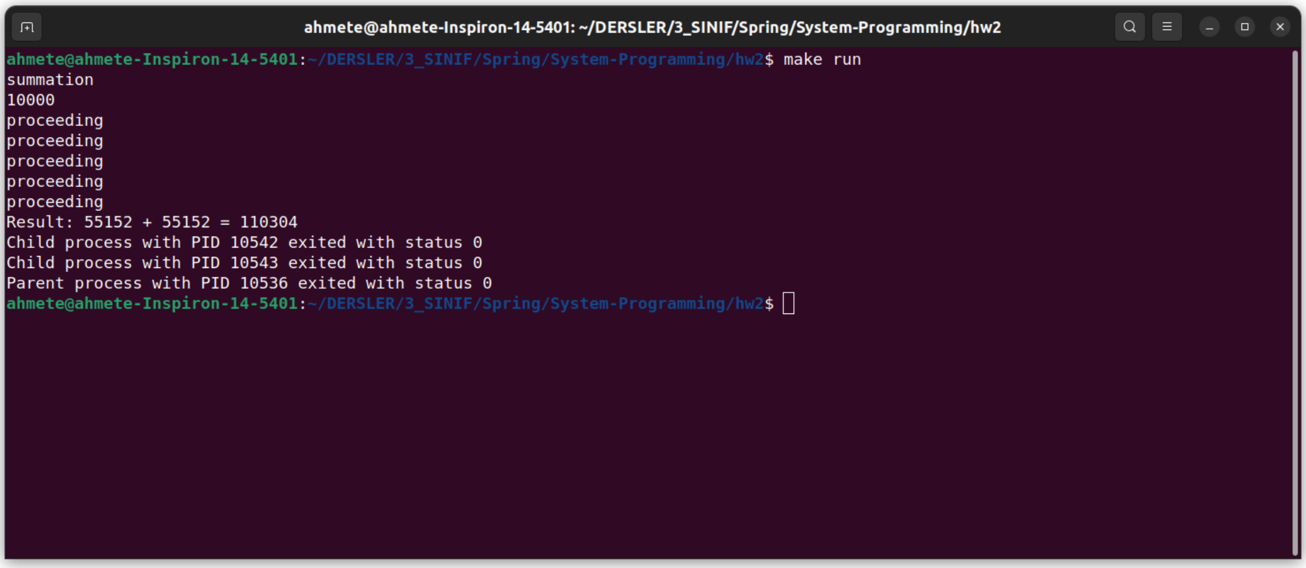
Sample outputs:



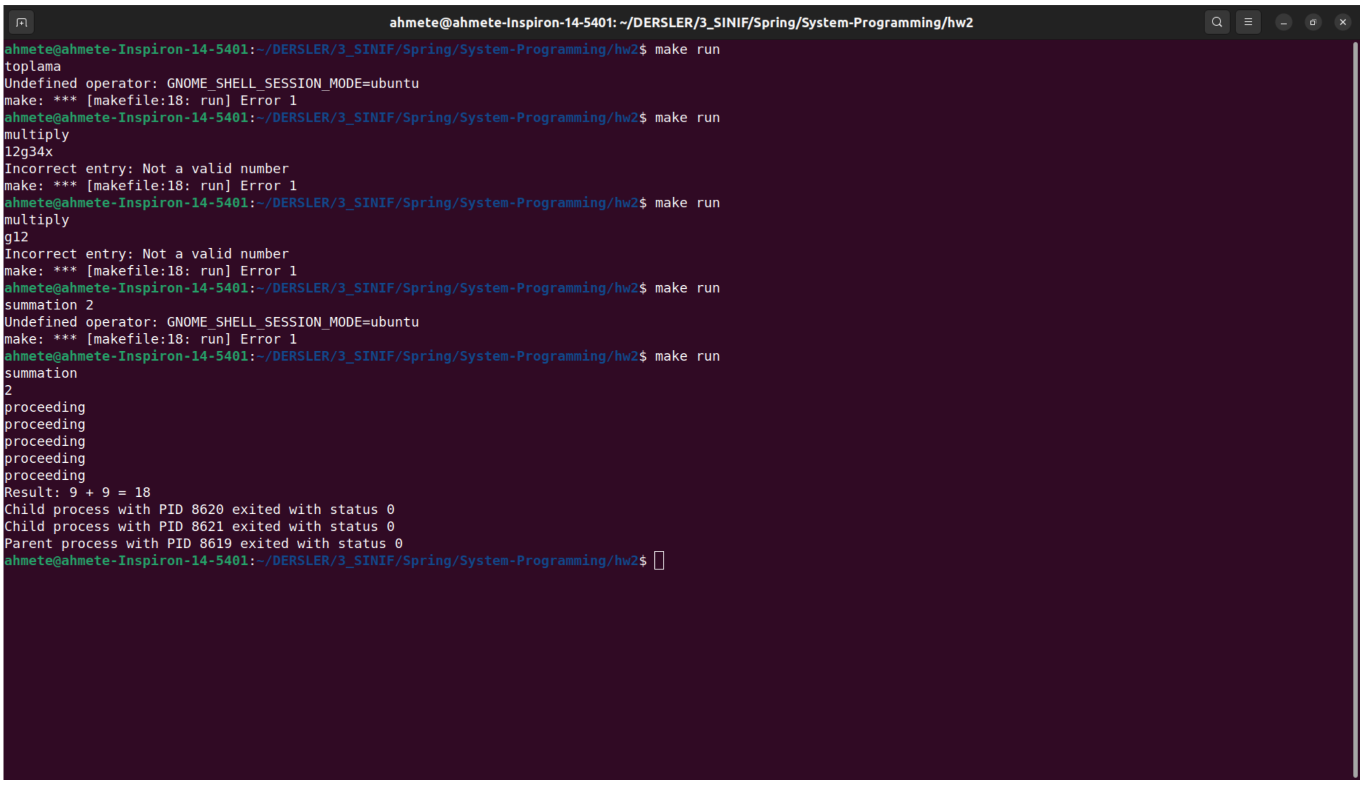
```
ahmete@ahmete-Inspiron-14-5401: ~/DERSLER/3_SINIF/Spring/System-Programming/hw2
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ make run
summation
10
proceeding
proceeding
proceeding
proceeding
proceeding
Result: 57 + 57 = 114
Child process with PID 10376 exited with status 0
Child process with PID 10377 exited with status 0
Parent process with PID 10373 exited with status 0
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ ls
helper.c  helper.h  helper.o  main.c  main.o  makefile  program
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ make clean
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ ls
helper.c  helper.h  main.c  makefile
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ 
```



```
ahmete@ahmete-Inspiron-14-5401: ~/DERSLER/3_SINIF/Spring/System-Programming/hw2
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ make run
multiply
12
proceeding
proceeding
proceeding
proceeding
proceeding
Result: 81 + 2857680000 = 2857680081
Child process with PID 10467 exited with status 0
Child process with PID 10468 exited with status 0
Parent process with PID 10461 exited with status 0
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ ls
helper.c  helper.h  helper.o  main.c  main.o  makefile  program
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ make clean
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ ls
helper.c  helper.h  main.c  makefile
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ 
```


ahmete@ahmete-Inspiron-14-5401: ~/DERSLER/3_SINIF/Spring/System-Programming/hw2

```
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ make run
summation
10000
proceeding
proceeding
proceeding
proceeding
proceeding
Result: 55152 + 55152 = 110304
Child process with PID 10542 exited with status 0
Child process with PID 10543 exited with status 0
Parent process with PID 10536 exited with status 0
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ 
```


ahmete@ahmete-Inspiron-14-5401: ~/DERSLER/3_SINIF/Spring/System-Programming/hw2

```
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ make run
toplama
Undefined operator: GNOME_SHELL_SESSION_MODE=ubuntu
make: *** [makefile:18: run] Error 1
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ make run
multiply
12g34x
Incorrect entry: Not a valid number
make: *** [makefile:18: run] Error 1
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ make run
multiply
g12
Incorrect entry: Not a valid number
make: *** [makefile:18: run] Error 1
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ make run
summation 2
Undefined operator: GNOME_SHELL_SESSION_MODE=ubuntu
make: *** [makefile:18: run] Error 1
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ make run
summation
2
proceeding
proceeding
proceeding
proceeding
proceeding
Result: 9 + 9 = 18
Child process with PID 8620 exited with status 0
Child process with PID 8621 exited with status 0
Parent process with PID 8619 exited with status 0
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw2$ 
```