CSE 344 System Programming Homework 1 Report

First of all, I divided the assignment into three modules as ***main.c***, ***cmdHandler.c***, c***mdProcess.c*** and ***.h*** files. main.c contains my main function. This function implements a mechanism that expects a continuous input from the user. It also provides an option like *"Enter (q) for quit."* in case the user wants to exit the programme. After the input is received, a tokenisation process is performed. In this process, the first word of the input is queried as command and the rest as parameters. In other words, everything except the first parameter is subjected to a parse process with the " character. Only entries with the .txt extension are allowed in the file creation command. Otherwise, a warning message is given and re-entry is requested. After the parse process, all tokens are collected in an array named tokens and passed to the process function in the cmdProcess.c file as follows:

process(tokenCount, tokens, MAX_TOKENS);

*void process(int, char **, int);*

Here, I created an if else chain to understand which command is called. We call the desired function here according to the command entered. Of course, in the meantime, I check if the inputs are valid, for example:

```
472
473         else if (strcmp(tokens[0], command7) == 0) /* Handle command 7 (listSome) */
474         {
475             if (tokenCount == 4 && tokens[1] != NULL && tokens[2] != NULL && tokens[3] != NULL)
476             {
477                 int numFlag = 0;
478
479                 for (int i = 0; i < strlen(tokens[1]); ++i)
480                 {
481                     if (tokens[1][i] < 49 || tokens[1][i] > 57 || tokens[2][i] < 48 || tokens[2][i] > 57) /* ASCII character value for integer */
482                     {
483                         numFlag = 1;
484                         break;
485                     }
486                 }
487                 int pageNumberAmount = atoi(tokens[1]); //kontrol koy
488                 int pageNumberIndex = atoi(tokens[2]);
489
490                 if (pageNumberAmount < 1 || pageNumberIndex < 0 || numFlag != 0)
491                 {
492                     fprintf(stderr, "Error: Enter a valid number for page amount and page index.\n");
493                 }
494
495                 else
496                 {
497                     logToFile("User entered command: listSome\n");
498                     listSomeCMD(tokens[3], pageNumberAmount, pageNumberIndex);
499                 }
500             }
501
502             else if (tokenCount < 4)
503             {
504                 fprintf(stderr, "Too few arguments.\nUsage: listSome \"<filename>\" or listSome number number \"<filename>\"\n");
505             }
506
507             else
508             {
509                 fprintf(stderr, "Too few arguments.\nUsage: listSome \"<filename>\" or listSome number number \"<filename>\"\n");
510             }
511         }
512
```

I used const ***char * command1 = "gtuStudentGrades";*** at the top of the command query file, the ***strcmp()*** function in the string.h header file checks which command is entered. Then I check for error conditions such as token count and input format if necessary. If everything is correct, the function that processes the relevant command is called. I set the name of each function as command_nameCMD. Each command function first performs a ***fork()*** operation if the ***fork()*** system call runs without errors

else if (childPID == 0)

{

/ code */*

}

The parent process continues as follows :

```
255
256     else
257     {
258         int status;
259
260         waitpid(childPID, &status, 0); /* wait for the child to complete the process */
261
262         if (WIFEXITED(status) && WEXITSTATUS(status) != 0) /* to prevent possible zombie processes */
263         {
264             afterWaitErr("addStudentGradeCMD", WEXITSTATUS(status));
265         }
266     }
267 }
268
```

The codes written in the block are executed and the desired command is processed here. I will now give the functioning of each command, the source code where necessary, and sample outputs.

The commands I use and their descriptions:

1 – manGTUCMD() when user only type “gtuStudentGrades” command :

When this command is entered, all commands and their functions defined in the system are printed on the screen. This command opens a new process using ***fork()*** and performs the necessary operations there. The ***prepareLogFormat()*** function is called so that the result of the operation is suitable for writing to the log file, and the process calls the ***exit()*** function with the ***EXIT_SUCCESS*** flag and terminates the process. I had written this output in colour to make it more readable, but I converted it to normal colour in case of portability incompatibility. A sample I/O is below:

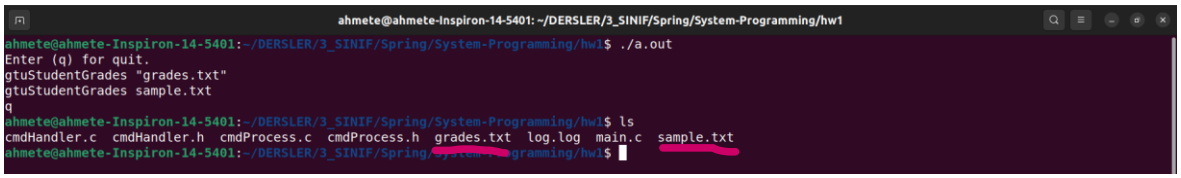
```
ahmete@ahmete-Inspiron-14-5401: ~/DERSLER/3_SINIF/Spring/System-Programming/hw1
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw1$ gcc *.c
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw1$ ./a.out
Enter (q) for quit.
gtuStudentGrades
gtuStudentGrades: lists all commands in the system.
gtuStudentGrades <filename>: opens a file on the system with the given file name.
addStudentGrade "name surname" "grade" <filename>: a student is added to the filename file with the given information.
searchStudent "name surname" <filename>: checks the filename file for the given name and prints detailed information about the name if it exists.
sortAll <filename>: sorts the lines in the file in the desired format and prints them on the screen.
showAll <filename>: lists all content of the filename.
listGrades <filename>: lists first five elements in the filename.
listSome page-amount page-index <filename>: divides the file into pages according to the given page-amount number and prints the information on the des
ired page to the screen with page-index.
q
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw1$
```

2 – gtuStudentGradesCMD when user type gtuStudentGrades with “example.txt” parameter :

When this command is entered, a new child process is created with ***fork()***. In this process, a file with the given name is opened in the following format:

int file = open(filename, O_CREAT | O_RDWR | O_APPEND, S_IRUSR | S_IWUSR);

Then the error conditions that may occur in file operations are discussed. The ***prepareLogFormat()*** function is called so that the result of the operation is suitable for writing to the log file, and the process calls the ***exit()*** function with the ***EXIT_SUCCESS*** flag and terminates the process. A sample I/O is below:



As you can see, the file name is accepted with or without the **"** signs and a file with that name is created. This is only valid for the file name in the whole system. The file can also be created with any extension. I did not feel the need to put a control such as only .txt.

3 – addStudentGradeCMD with parameter “name surname” “grade” “filename” :

When this command is entered with the given parameters, it is first checked whether the name entered is already in the file. A variable of type ***off_t*** is defined and this variable is incremented byte by byte in the file during the search. If the same name is found, this increment stops. In this way, we can now only change that line in the file. If the name is not already in the file, a new line is added at the end as usual. After the process is finished, the prepareLogFormat() function is called for the log file and the child process is finished with ***EXIT_SUCCESS***. When a text is to be written to the file, I use the " character again as a separator. So the format of each line in the file is as follows:

"name surname" "grade"

Sample I/O and code below:

```c
else if (childPID == 0)
{
    deleteEnter(filename);
    off_t byteCount = 0;

    char tokens[MAX_LINES][2][FBUFFER_SIZE];
    int lineIndex = 0, existingIndex = 0;

    readAndTokenizeFile(filename, tokens, &lineIndex);

    for (int i = 0; i < lineIndex; ++i)
    {
        if (strcmp(nameSurname, tokens[i][0]) == 0)
        {
            existingIndex = 1;
            break;
        }

        else
        {
            byteCount += strlen(tokens[i][0]) + strlen(tokens[i][1]) + 2;
        }
    }
```

```c
else
{
    lseek(file, byteCount, SEEK_SET); /* imleci güncellenecek satıra getiriyoruz. */

    ssize_t writtenBytes = write(file, merged, strlength);

    if (writtenBytes < strlength)
    {
        char message[MESSAGE_BUFFER];
        char * status = "Error: Write file error - ";

        strcpy(message, status);
        strcat(message, filename);
        strcat(message, "\n");

        handleError(message);
    }
    free(merged);
    prepareLogFormat("addStudentGrade update grade", nameSurname, grade, filename);

    lseek(file, 0, SEEK_END);
}
```

After writing to the index, the cursor is moved to the end of the file again.

```
ahmete@ahmete-Inspiron-14-5401: ~/DERSLER/3_SINIF/Spring/System-Programming/hw1
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw1$ ./a.out
Enter (q) for quit.
addStudentGrade "ahmet ozdemir""AA" "grades.txt"
addStudentGrade "esma sonmez" "BA" grades.txt
addStudentGrade "yasir sekerci" "CC" gra.txt
Open file error: gra.txt
: No such file or directory
addStudentGradeCMD: Child process failed with exit status 1
addStudentGrade "yasir sekerci" "FT" "grades.txt"
Only AA, BA, BB, CB, CC, DC, DD, FF, VF, NA format is acceptable for grade.
addStudentGrade "yasir sekerci" "CC" "grades.txt"
q
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw1$ cat grades.txt
"ahmet ozdemir" "AA"
"esma sonmez" "BA"
"yasir sekerci" "CC"
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw1$ ./a.out
Enter (q) for quit.
addStudentGrades "hakan yasar""VF""grades.txt"
addStudentGrades: command not found
addStudentGrade "esma sonmez" "AA" "grades.txt"
q
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw1$ cat grades.txt
"ahmet ozdemir" "AA"
"esma sonmez" "AA"
"yasir sekerci" "CC"
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw1$
```

4 – searchStudentCMD with parameter “name surname” “filename” :

This command searches the given student name in the file and if it exists, it prints the information on the screen, if not, it informs that it was not found. ***readAndTokeniseFile()*** function reads all lines in the file and assigns them to a 3D array named tokens. I made the size of this array fixed and I use this method in almost 3 or 4 functions. The first of the 2 indexes of tokens holds the name and surname, while the second index holds the grade.

in functions :

char tokens[MAX_LINES][2][FBUFFER_SIZE];

Int lineIndex = 0;

readAndTokenizeFile(filename, tokens, &lineIndex);

in header :

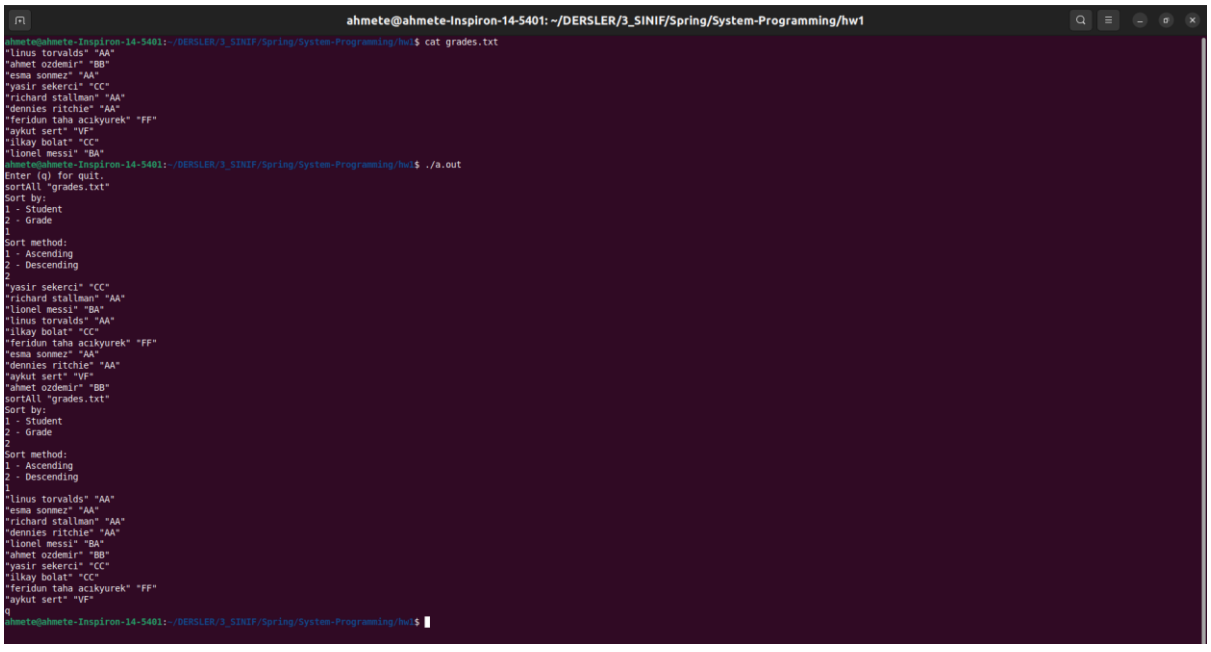
#define FBUFFER_SIZE 4096

#define MAX_LINES 100

Then prepareLogFormat() function is called for the log file and the function is terminated with EXIT_SUCCESS.

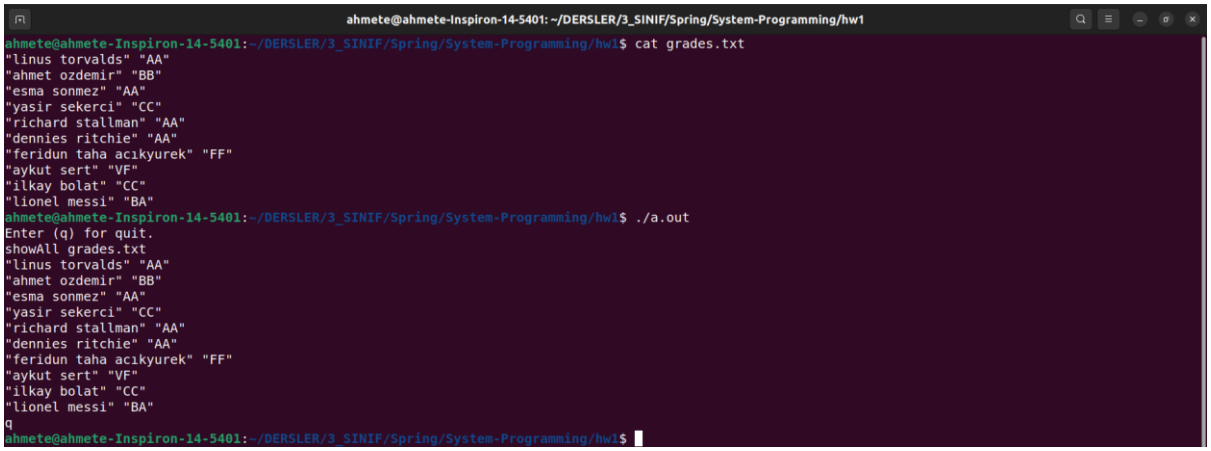
5 – sortAllCMD with parameter “filename” :

This command assigns all lines in the file to an array named tokens with the help of the function called ***readAndTokeniseFile()***, just like any other function. The user is then shown a menu where the user selects the sorting format and the items. According to the selected option, the inputs in the file are printed on the screen in sorted form. This command does not change the original file. It is only printed on the screen. Example I/O below:



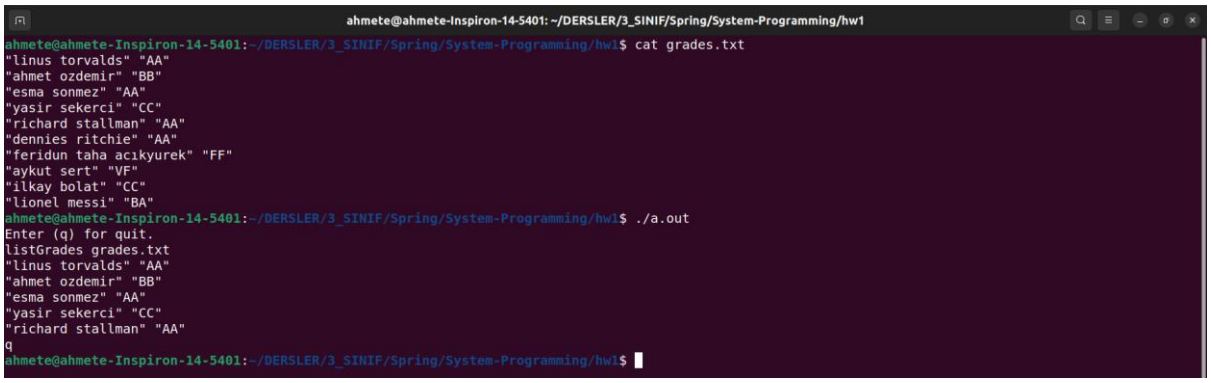
6 – showAllCMD with parameter “filename” :

This command prints all lines in the tokens array to the screen with a for loop up to ***lineIndex***. Sample I/O below:



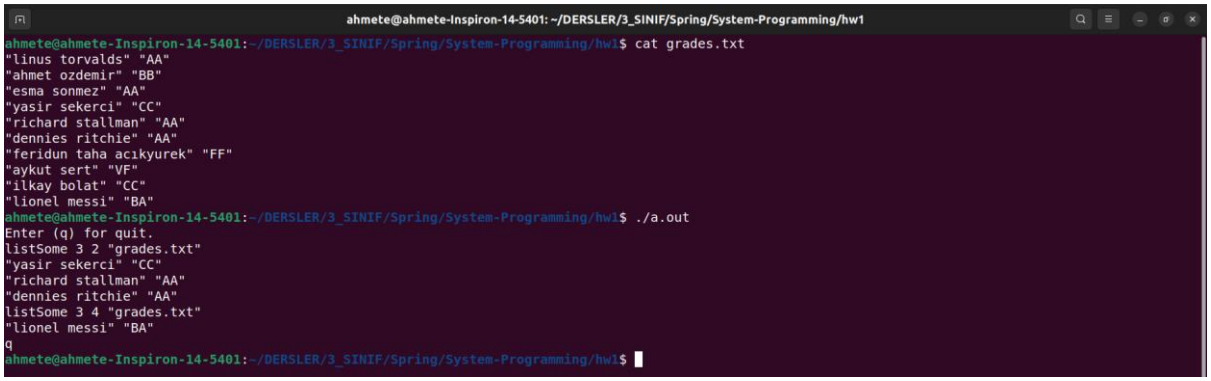
7 – listGradesCMD with parameter “filename” :

This command also prints the first 5 elements of the tokens array filled by the ***readAndTokeniseFile()*** function. Sample I/O is below:



8 – listSomeCMD with parameter page-amount, page-index and “filename” :

This function converts the tokens array filled by ***readAndTokeniseFile()*** into a book. The book is created as follows. page-amount determines the number of lines on each page. So, for a file of 100 lines, a book of 20 pages is created when page-amount is 5. The second parameter, page-index, determines which page will be printed on the screen. Sample I/O is below:



As you can see, although the number of rows is not exactly divisible by page-amount, the last remaining part is thrown to a page. The algorithm here is as follows:

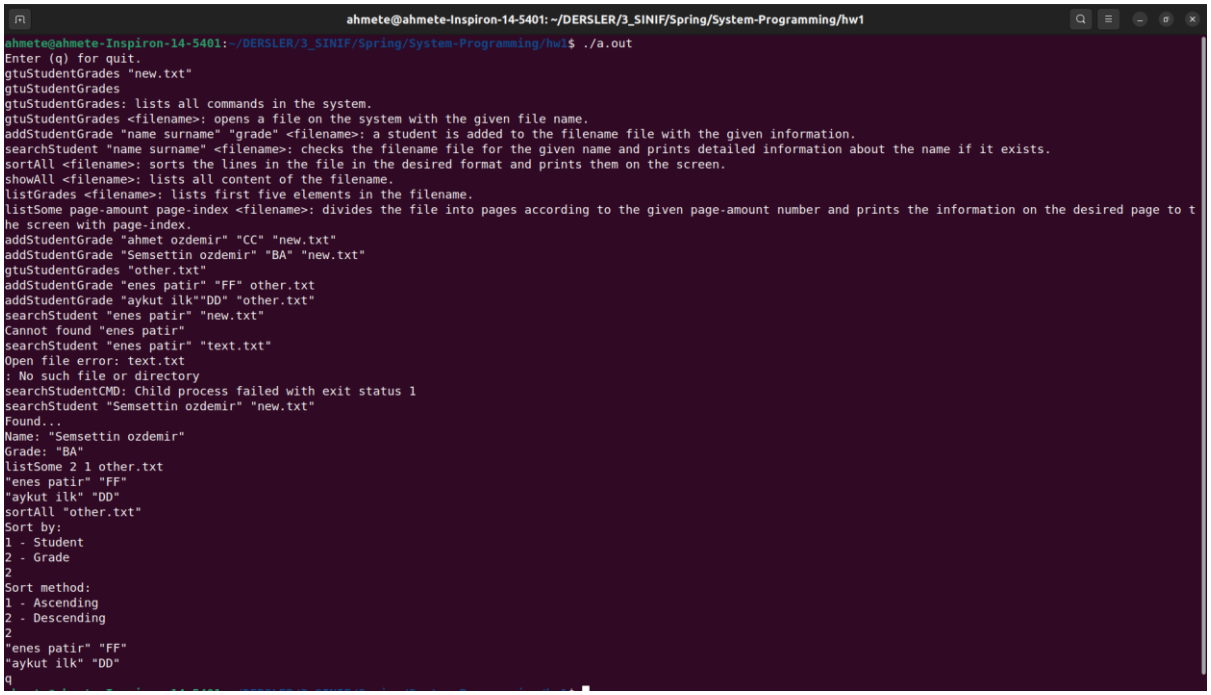
```
552
553     else if (childPID == 0)
554     {
555         char tokens[MAX_LINES][2][FBUFFER_SIZE];
556         int lineIndex = 0;
557 
558         int realIndexStart = (pageIndex - 1) * pageAmount;
559         int realIndexFinish = realIndexStart + pageAmount;
560 
561         readAndTokenizeFile(filename, tokens, &lineIndex);
562 
563         if (realIndexStart >= lineIndex)
564         {
565             char * message = "Error: Page index is out of range in the listSome command.\n";
566             write(STDOUT_FILENO, message, strlen(message));
567         }
568 
569         for (int i = realIndexStart; i < realIndexFinish && i < lineIndex; ++i)
570         {
571             if (tokens[i][0] != NULL && tokens[i][1] != NULL)
572             {…
577             }
578         }
579         char numStr1[5];
580         char numStr2[5];
581 
582         snprintf(numStr1, sizeof(numStr1), "%d", pageAmount);
583         snprintf(numStr2, sizeof(numStr2), "%d", pageIndex);
584 
585         prepareLogFormat("listSome", numStr1, numStr2, filename);
586         exit(EXIT_SUCCESS);
587     }
588
```

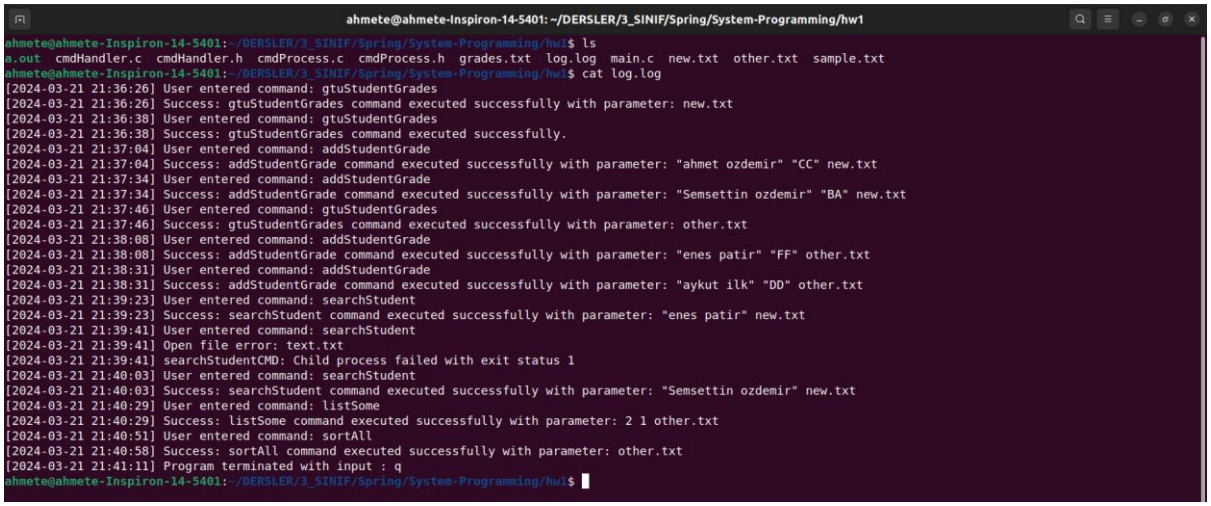
9 – logToFile() :

This function was requested from us, even though it is not a command, I opened a separate process using ***fork()*** for this function. The time of typing each command, the command information entered, the result of the command entered, and explanations for possible errors are written to a file called log.log. This file is defined as follows:

#define LOG_FILE "log.log"

This way I can access this name everywhere. Below are sample log outputs for some inputs.





Other functions and constants I use :

void readAndTokenizeFile(const char *, char tokens[MAX_LINES][2][FBUFFER_SIZE], int *);

This function is called by almost all processes. Its general function is this: it extracts all the information from the requested file and puts it in a certain format in the tokens array given as a parameter. It also equates the variable lineIndex, which is also given as a pointer, to the total number of lines. Thus, by using the size of the array, operations can now be done more easily.

void process(int, char **, int);

This function provides the general programme flow according to the parameter from the command line. It redirects to whichever process is called. In addition, before doing this, the command and parameters are subjected to a series of queries. Number of parameters, wrong type of arguments, etc.

void afterWaitErr(const char *, const int);

Function written to handle possible errors in the child while the parent process is waiting for its child. This function is also called every time fork() is called.

int searchForAdd(const char *, const char *, off_t *);

It checks if the name entered in the ***addStudentGradeCMD*** function already exists in the file. If the name already exists, it takes a variable called ***byteCount*** of type ***off_t*** as a parameter to find the index where the name is written.

I will not explain all functions in detail. Other functions are below:

Note : *I did not use the makefile to show the output in order to better see and specify the terminal output. There is also a makefile file in the file I delivered.*

```h
#ifndef CMD_HANDLER_H
#define CMD_HANDLER_H

#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

void manGTUCMD();
void gtuStudentGradesCMD(const char *);
void addStudentGradeCMD(const char *, const char *, char *);
void searchStudentCMD(const char *, const char *);
void sortAllCMD(const char *);
void showAllCMD(const char *);
void listGradesCMD(const char *);
void listSomeCMD(const char *, const int, const int);

#endif /* CMD_HANDLER_H */
```

```h
#ifndef CMD_PROCESS_H
#define CMD_PROCESS_H

#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <time.h>

#define LOG_FILE "log.log"
#define INPUT_BUFFER_SIZE 3
#define TIME_BUFFER 20
#define MESSAGE_BUFFER 200
#define INPUT_BUFFER 100
#define MAX_TOKENS 10
#define MAX_WORD_SIZE 50
#define FBUFFER_SIZE 4096
#define MAX_LINES 100
#define ERR_NUM 6
#define TRUE 1

void process(int, char **, int);
char * mergeStr(const char *, const char *);
void deleteEnter(char *);
void trimQuotes(char *);
void sortMenu(int *, int *);
void logToFile(const char *);
void handleError(const char *);
void readAndTokenizeFile(const char *, char tokens[MAX_LINES][2][FBUFFER_SIZE], int *);
void prepareLogFormat(const char *, const char *, const char *, const char *);
void afterWaitErr(const char *, const int);
int searchForAdd(const char *, const char *, off_t *);

#endif /* CMD_PROCESS_H */
```