

**Gebze Technical University**

---

# **CSE344 SYSTEM PROGRAMMING**

**HOMEWORK-1**

**2023-24**

---

**Lecturer: Erkan Zergeroglu**

**Emre YILMAZ 1901042606**

## General Notes

1. Syscall functions such as `open()`, `read()`, `write()`, and `create()` were used for file handling operations such as opening, closing, reading, and writing.
2. The `printf` function was generally used for output printing with the message "by permission of the T.A." Only the `read()` syscall function was used for taking input.
3. Functions from the `string.h` library, such as `sprintf` for string manipulation, were used by permission of the T.A.
4. Dynamic memory allocation was used with `malloc`, and error handling was implemented. Memory was carefully managed, freed at the end of processes, and appropriate logs were written.
5. To create an example student record file, assistance was sought from classmates who generate large files. I used sample student record files obtained from them solely to test the code I wrote.
6. In the assignment, there is an example record file named "emre.txt." You can either continue to work with this file or create a new one. It's up to you.
7. It is assumed that the maximum record length will be 100 characters, and memory is allocated accordingly.
8. It is assumed that there will be a maximum of 1000 lines in the file, and memory is allocated accordingly.
9. The created record file and log file are located in the same directory as `main.c` and `Makefile`.

## Introduction

The Student Grade Management System with Process Creation is a program designed to manage student grades stored in a file using the C programming language. This system allows users to perform various operations such as adding new student grades, searching for existing student grades, sorting grades, and displaying all grades stored in the file. Additionally, it provides functionalities for logging the completion of each task and displaying usage instructions.

The core functionalities of the system are implemented using process creation, utilizing the `fork()` system call to manage file operations concurrently. This approach ensures efficient handling of file manipulations and enhances system performance.

The system is designed to be user-friendly, providing clear commands for each operation and error handling mechanisms to gracefully manage unexpected scenarios. By utilizing this system, users can effectively manage student grades, facilitating efficient record keeping and retrieval of information.ng for a more flexible and extensible design as new functionalities are introduced to the system.

# Architecture

## 0.1 File Architecture

The directory where `main.c` exists contains a folder named `src`. This `src` folder has subfolders for each operation such as `Sort`, `Search`, `Display`, etc. Each operation folder contains a header file (`.h`) and an implementation file (`.c` file). Additionally, the main directory (where `main.c` is located) contains a `makefile`, input text files, and a log file.









Name
 <code>src</code>
 <code>CSE 344-HW1.pdf</code>
 <code>emre.txt</code>
 <code>grades.txt</code>
 <code>grades_5mb.txt</code>
 <code>log.txt</code>
 <code>main.c</code>
 <code>makefile</code>

Fig-1: Main Directory

## 0.2 How to Compile and Run

To compile and run the program, navigate to the parent directory (the directory where `main.c` is located) and type the command `make`. This command will compile the program. After compilation, you can execute the program by typing the appropriate commands.

```
emre@ubuntu:~/Downloads/cse344-system-programming-main$ make
gcc -Wall -Wextra -g -c main.c
gcc -Wall -Wextra -g -o main main.o searchStudent.o sort.o display.o addStudentGrade.o gtuStudentGrades.o log.o
./main
Welcome to GTU Student Management System
█
```

Fig-1: How to Run

# 1 Main Function Algorithm

1. **Infinite Loop:** The main function operates within an infinite loop, continuously waiting for commands.
2. **Command Parsing:** When a command is received, it is parsed by splitting it into arguments separated by whitespace. If the name contains more than one word (like Tarik Emir KALDIRIM), it's expected to be enclosed within quotation marks, and the function handles this accordingly.
3. **Command Execution:** After parsing the arguments, the command is checked. If it's not the "EXIT" command or null, a fork is created, generating a child process. This child process then executes the appropriate function for the command.
4. **Handling Errors:** If the command is invalid, the child process terminates, and control returns to the parent process to await a new command. Similarly, when the command execution is successful, the child process terminates, and control returns to the parent process to await a new command.
5. **Logging:** Throughout the process, log messages are generated for both errors and successful executions.

The function continuously checks if the arguments are entered correctly, ensuring proper handling of arguments both within and outside quotation marks. Additionally, when a student name consists of multiple parts (such as "Tarik Emir Kaldirim"), the function expects both the first and last names to be enclosed within quotation marks and handles this scenario appropriately.

## 2 gtuStudentGrades Command

### 2.1 gtuStudentGrades Command with No Arguments

This command does not take any parameters. It provides usage information on how the system is used.

This command does not contain any arguments. When invoked, it calls the usage() function from the gtuStudentGrades header file. This function forks the process, creating a child process, while the parent process waits for the child process to complete its task of providing information on how the system is to be used.

```
enre@ubuntu:~/Downloads/cse344-system-programming-main$ make
gcc -Wall -Wextra -g -c main.c
gcc -Wall -Wextra -g -o main main.o searchStudent.o sort.o display.o addStudentGrade.o gtuStudentGrades.o log.o
./main
Welcome to GTU Student Management System
gtuStudentGrades
Usage: [command] [arguments]
Commands:
    "gtuStudentGrades": Print usage of the system
        Example using: gtuStudentGrades

    gtuStudentGrades "student.txt": Create student.txt file.
        Example using: gtuStudentGrades "test.txt"

    addStudentGrade "Name Surname" "Grade" "grades.txt": Add a new student's grade.
        For example, addStudentGrade Enre Vilnaz FF text.txt

        If a user has more than one name, his/her name and surname must be inside quote signs
        For example, addStudentGrade "Mehmet Ali Birand" "FF" "text.txt"

    searchStudent "Name Surname" "grades.txt": Search for a student's grade.
        For example, searchStudent Enre Vilnaz text.txt

        If a user has more than one name, his/her name and surname must be inside quote signs
        For example, addStudentGrade "Mehmet Ali Birand" "FF" "text.txt"

    sortAll "grades.txt": Sort students according to grade or name
        Example using: sortAll "grades.txt"

    showAll "grades.txt": Show all students with grades
        Example using: showAll "grades.txt"

    listGrades "grades.txt": Display the first five student grades.
        Example using: listGrades "grades.txt"

    listSome "numEntries" "pageNumber" "grades.txt": Display student grades based on page number and number of entries per page.
        Example using: listSome 5 2 "grades.txt"

NOTE: Your inputs can be taken either within quotation marks or without quotation marks. It's up to you. However, if a student's name consists of more than one word, both their first name and last name
should be within quotation marks.
```

Fig-1: gtuStudentGrades Command with No Arguments

### 2.2 gtuStudentGrades Command with Argument

This command takes a file name and creates the file.

In the main function, the parent process forks to create a child process. If this command is entered with an argument, the gtuStudentGrades function from the gtuStudentGrades header file is called. The child process executes this command.

This command, which shares the same name as the previous one, takes one argument. This argument specifies the filename that the command will open. When opening the file, certain flags are used. By using the O\_WRONLY | O\_CREAT | O\_EXCL flags together, the process attempts to open the file, and if a file with the same name already exists, it gracefully terminates by providing

controlled error messages. If the process completes successfully, it creates the record txt file, it writes the necessary log message and terminates, returning to the parent process.

```
emre@ubuntu:~/Downloads/cse344-system-programming-main$ make
gcc -Wall -Wextra -g -c src/FileCreation/gtuStudentGrades.c
gcc -Wall -Wextra -g -o main main.o searchStudent.o sort.o display.o addStudentGrade.o gtuStudentGrades.o log.o
./main
Welcome to GTU Student Management System
gtuStudentGrades cse344.txt
File has been created successfully cse344.txt:
```

Fig-1: gtuStudentGrades Command with Argument

### 3 addStudentGrade Command

This command takes the name, surname, and grade of the student and adds them as a single line to the data.

#### 3.1 Implementation

To execute this command, after creating a child process with fork in the main function, we call the `addStudentGrade` function from the `addStudentGrade` header file located in the `src/Add/` directory.

This function takes three parameters: the filename, the name of the student to be added, and their grade. The letters of the grade string are converted to uppercase. The target file is opened using the `open()` syscall function with the `O_WRONLY | O_APPEND` flags.

If the file does not exist, it gracefully prints an error message and creates a log message before terminating the child process. If the file exists, the `write()` syscall function is used to append the string in the format "Name Surname, Grade" as a new line in the file.

Upon successful completion, the necessary log message is created, and the child process is terminated.

An important point to mention regarding this function is that if the student's name consists of multiple words, the name and surname should be provided within quotation marks: `addStudentGrade "Resat Nuri Guntekin" FF "emre.txt"`.

```
int fd = open(filename, O_WRONLY | O_APPEND); // Open the file in write-only mode and append mode
if (fd == -1)
{
    perror("open");
    char logMessage[100];
    sprintf(logMessage, "Error opening file: %s", filename);
    logToFile(logMessage);
    exit(EXIT_FAILURE);
}
```

Fig-5: Opening file for record adding

```
emre@ubuntu:~/Downloads/cse344-system-programming-main$ make  
./main  
Welcome to GTU Student Management System  
addStudentGrade "emre yilmaz" "aa" "nofile.txt"  
Adding student grade: emre yilmaz, aa to file nofile.txt  
open: No such file or directory
```

### 3.2 Screenshots

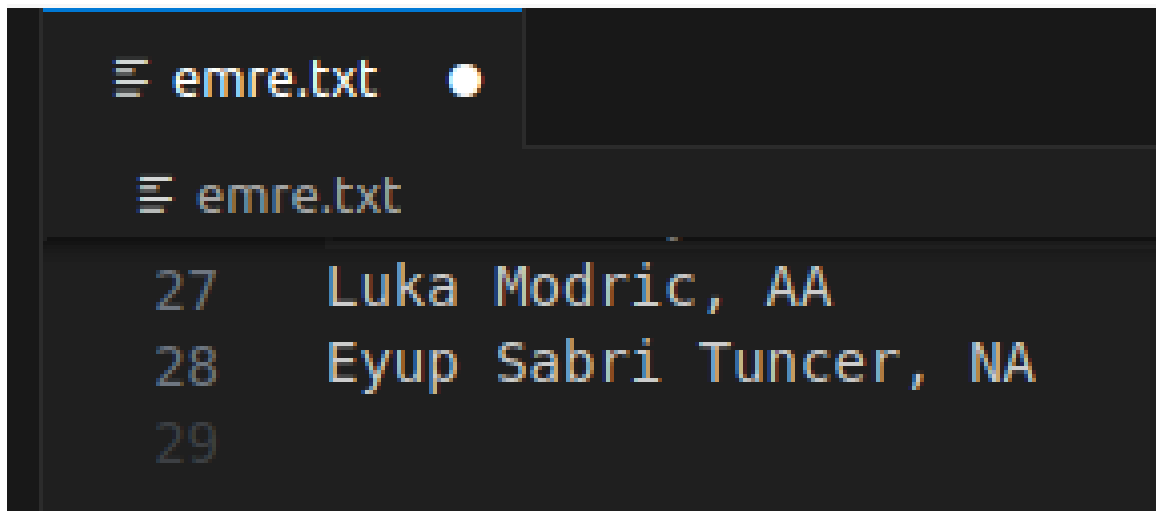
Fig-6: Try to non-existed file and getting error

```
emre@ubuntu:~/Downloads/cse344-system-programming-main$ make  
./main  
Welcome to GTU Student Management System  
addStudentGrade Luka Modric AA emre.txt  
Adding student grade: Luka Modric, AA to file emre.txt
```

Fig-7: Adding a record with single name example (Without quotes)

```
emre@ubuntu:~/Downloads/cse344-system-programming-main$ make  
./main  
Welcome to GTU Student Management System  
addStudentGrade "Eyup Sabri Tuncer" "NA" "emre.txt"  
Adding student grade: Eyup Sabri Tuncer, NA to file emre.txt  
█
```

Fig-8: Adding a record with multiple name example (With quotes)



```
emre.txt  
emre.txt  
27 Luka Modric, AA  
28 Eyup Sabri Tuncer, NA  
29
```

Fig-9: Text file after add operation



## 4 searchStudent Command

This command takes a name and surname, scans the data, and displays the name and grade of the corresponding student if a match is found.

To execute this command, after creating a child process with fork in the main function, we call the `searchStudent` function from the `searchStudent` header file located in the `src/Search/` directory.

This function takes two parameters: the target name and the filename. The file is opened using the `open()` syscall function with the `O_RDONLY` flag.

In case of any error during file operations, the appropriate error message is displayed, a log message is created, and the process is terminated.

If the file is successfully opened, a loop is created to read the file and find the target student. Here are the steps of this loop:

1. File Reading: The file is divided into chunks of 1024 bytes using the read function, and these chunks are read into a memory block called buffer.
2. Line-by-Line Search: Each chunk (buffer) is searched for the target. Each line is traversed, attempting to find a line matching the target name.
3. Data Parsing: The line is split into name and grade, separated by a comma. The `strtok` function is used to parse the name and grade.
4. Match Verification: If the parsed name matches the target, the corresponding record is printed to the console, and the loop exits.
5. The file pointer (fd) is moved back to the position of the last read byte, preparing for the next read operation.

If an error occurs while reading bytes from the file, the appropriate error message is displayed, a log message is created, and the process is terminated. If the search is successful, the necessary log message is created, and the process is terminated.

The search operation is NOT case-sensitive. Even if a record is written in uppercase, you can still find it by searching in lowercase.

An important point to mention regarding this function is that if the student's name consists of multiple words, the name and surname should be provided within quotation marks: `searchStudent "Resat Nuri Guntekin" "emre.txt"`.

```

emre@ubuntu:~/Downloads/cse344-system-programming-main$ make
./main
Welcome to GTU Student Management System
searchStudent "emre yilmaz" "emre.txt"
Searching for student: emre yilmaz in the file emre.txt
The record has been found: Name: emre yilmaz, Grade: FF

```

Fig-10: Successful search

```

emre@ubuntu:~/Downloads/cse344-system-programming-main$ make
./main
Welcome to GTU Student Management System
searchStudent christopher anderson emre.txt
Searching for student: christopher anderson in the file emre.txt
The record has been found: Name: Christopher Anderson, Grade: BA

```

Fig-10: Successful search

```

emre@ubuntu:~/Downloads/cse344-system-programming-main$ make
gcc -Wall -Wextra -g -c src/Search/searchStudent.c
gcc -Wall -Wextra -g -o main main.o searchStudent.o sort.o display.o addStudentGrade.o gtuStudentGrades.o log.o
./main
Welcome to GTU Student Management System
searchStudent "Burak Ceylan" "emre.txt"
Searching for student: Burak Ceylan in the file emre.txt
The record has not been found

```

Fig-10: Failed search

## 5 Displaying Commands

### 5.1 showAll Command

This command prints all records from the student list along with their names and grades.

### 5.2 listGrades Command

This command prints the first 5 records from the student list along with their names and grades.

### 5.3 listSome Command

This command has 2 parameters: **numEntry** and **pageNumber**. It assumes that student data consists of pages, each containing any number of entries defined by **numEntry**, and prints the page corresponding to the given **pageNumber**.

### 5.4 Implementation and Screenshots

The **showAll**, **listSome**, and **listGrades** commands all invoke the **showAll** function from the **display** header located in the **src/Display** directory. The function handles which command is called within it. It has three parameters: **filename**, **numEntries**, and **pageNumber**. If **numEntries=5** and **pageNumber=-1** are passed, it means that the **listSome** command is executed. If **numEntries** and **pageNumber** are not -1, it indicates that the **listSome** function is executed.

Firstly, the file is opened with the **O\_RDONLY** flag. If an error occurs during file opening, the appropriate error message is displayed, a log message is created, and the process is terminated. If the file is successfully opened, similar to the **searchStudent** function, we enter a loop to start reading the file. Within the loop, we check the conditions mentioned above to determine which command (**showAll**, **listGrades**, or **listSome**) is executed, and perform print operations accordingly.

If there is an error while reading the file, an error message is displayed, a log message is created, and the process is terminated. If the command is successfully executed, a log message is created, and the process is terminated.

Here are the basic steps of the loop implementation that forms the algorithm:

1. File Reading: The file is read in chunks of 1024 bytes using the **read** function, and each chunk is stored in a buffer.
2. Line Processing: Each line in the buffer is processed individually to handle special conditions or pagination requirements.
3. Newline Identification: The **strchr** function is used to find the newline character (**n**) in each line, indicating the end of the line.
4. Printing or Skipping: Based on certain conditions such as special cases or pagination, the line may be printed or skipped. A **printFlag** variable is used to control printing.
5. Pointer and Counter Updates: After processing each line, pointers and counters are updated to move to the next line and track the progress through the file.

```

emre@ubuntu:~/Downloads/cse344-system-programming-main$ make
gcc -Wall -Wextra -g -c src/Display/display.c
gcc -Wall -Wextra -g -o main main.o searchStudent.o sort.o display.o addStudentGrade.o gtuStudentGrades.o log.o
./main
Welcome to GTU Student Management System
showAll "emre.txt"
Displaying all student grades
Record 0: Matthew Williams, BB
Record 1: Patricia Robinson, AA
Record 2: Daniel Moore, VF
Record 3: Anthony Clark, CB
Record 4: emre yilmaz, FF
Record 5: uysa akgul, CC
Record 6: eren cakar, FF
Record 7: doga nalci, FF
Record 8: eray ozkan, BB
Record 9: erkan zer, NA
Record 10: Matthew Williams, BB
Record 11: Patricia Robinson, AA
Record 12: Daniel Moore, VF
Record 13: Anthony Clark, CB
Record 14: Anthony Lewis, VF
Record 15: Matthew Lee, FF
Record 16: Anthony Gonzalez, NA
Record 17: Betty Anderson, NA
Record 18: Christopher Anderson, BA
Record 19: Mary Perez, FF
Record 20: Barbara Hernandez, VF
Record 21: Robert Davis, BA
Record 22: Matthew Brown, DD
Record 23: cengiz selam aban, CC
Record 24: yakup genc, DC
Record 25: enes fol aysu, AA
Record 26: Luka Modric, AA
Record 27: Eyup Sabri Tuncer, NA

```

Fig-11: showAll Command Successful Execution

```

emre@ubuntu:~/Downloads/cse344-system-programming-main$ make
gcc -Wall -Wextra -g -c main.c
gcc -Wall -Wextra -g -o main main.o searchStudent.o sort.o display.o addStudentGrade.o gtuStudentGrades.o log.o
./main
Welcome to GTU Student Management System
listSome 2 3 "emre.txt"
Displaying all student grades
Record 4: emre yilmaz, FF
Record 5: uysa akgul, CC

```

Fig-12: listSome Command Successful Execution

```
emre@ubuntu:~/Downloads/cse344-system-programming-main$ make  
./main  
Welcome to GTU Student Management System  
listSome 5 7 grades.txt  
Displaying all student grades  
Record 30: Sarah Taylor, DC  
Record 31: Christopher Robinson, NA  
Record 32: Jessica Williams, AA  
Record 33: Sandra Perez, BB  
Record 34: Michael Jackson, BB  
█
```

Fig-12: listSome Command Successful Execution

```
emre@ubuntu:~/Downloads/cse344-system-programming-main$ make  
gcc -Wall -Wextra -g -c src/Display/display.c  
gcc -Wall -Wextra -g -o main main.o searchStudent.o sort.o display.o addStudentGrade.o gtuStudentGrades.o log.o  
./main  
Welcome to GTU Student Management System  
listGrades "emre.txt"  
Displaying all student grades  
Record 0: Matthew Williams, BB  
Record 1: Patricia Robinson, AA  
Record 2: Daniel Moore, VF  
Record 3: Anthony Clark, CB  
Record 4: emre yilmaz, FF  
█
```

Fig-13: listGrades Command Successful Execution

## 6 sortAll Command

The `sortAll` command allows us to sort and display all students in the file either by name or by grade, in ascending or descending order. This function is executed by the child process after forking in the main function. The function is located in the `sort.h` file within the `/src/Sort` directory.

### 6.1 Implementation and Screenshots

Firstly, it's worth mentioning that we have an enum type to sort the grades. The highest grade is represented by "AA", and the lowest grade by "NA".

When the function is executed, it first opens the target file using the `O_RDONLY` mode with the `read()` syscall. In case of an error, the appropriate message is displayed, a log message is created, and the process is terminated. Then, sorting options are obtained from the user. We determine whether to sort by name or by grade, and whether to sort in ascending or descending order. This input is also obtained using the `read()` syscall.

After obtaining the inputs, similar to other command implementations, the file is traversed with a loop. Strings are stored in a dynamically managed char array. Then, a simple Bubble Sort algorithm is used to sort the string array according to the user's selection. Since there are many string operations in this function, dynamic memory allocation is frequently used and carefully managed. You can see the details in the source file. Finally, the sorted string array is printed, and the function and process are terminated. Necessary log files are created as well.

Let me summarize the algorithm in 5 steps:

1. The code defines a function called `findGrade` that takes a grade as input and returns the corresponding `LetterGrade` enum value. It uses `strcmp` to compare the grade with predefined values and returns the appropriate enum value.
2. The code defines a function called `sortAll` that takes a filename as input. It opens the file in `read-only` mode and handles any errors that occur during the file opening process. It then prompts the user to enter the sorting option and order, reads the input from the user, and stores the values in variables.
3. The code reads the contents of the file and stores each line in a string array called `lines`. It uses a buffer to read the file in chunks and processes each chunk to extract individual lines. It keeps track of the total bytes read from the file to ensure that it reads the file correctly.
4. Depending on the sorting option chosen by the user, the code sorts the `lines` array either by name or by grade. It uses nested loops and string comparison functions to compare and swap the lines in the array based on the chosen sorting option.
5. Finally, the code prints the sorted lines based on the sorting order chosen by the user. It iterates over the `lines` array and prints each line along with a record number. It also logs a success message to a log file and frees the dynamically allocated memory before exiting the program.

```
emre@ubuntu:~/Downloads/cse344-system-programming-main$ make
./main
Welcome to GTU Student Management System
sortAll "emre.txt"
Enter the sorting option:
1. Sort by name
2. Sort by grade
1
Enter the sorting order:
1. Ascending
2. Descending
1
Displaying all student grades
Record 1: Anthony Clark, CB
Record 2: Anthony Clark, CB
Record 3: Anthony Gonzalez, NA
Record 4: Anthony Lewis, VF
Record 5: Barbara Hernandez, VF
Record 6: Betty Anderson, NA
Record 7: Christopher Anderson, BA
Record 8: Daniel Moore, VF
Record 9: Daniel Moore, VF
Record 10: Eyup Sabri Tuncer, NA
Record 11: Luka Modric, AA
Record 12: Mary Perez, FF
Record 13: Matthew Brown, DD
Record 14: Matthew Lee, FF
Record 15: Matthew Williams, BB
Record 16: Matthew Williams, BB
Record 17: Patricia Robinson, AA
Record 18: Patricia Robinson, AA
Record 19: Robert Davis, BA
Record 20: cengiz selam aban, CC
Record 21: dogan nalcı, FF
Record 22: emre yilmaz, FF
Record 23: enes fol aysu, AA
Record 24: eray ozkan, BB
Record 25: eren cakar, FF
Record 26: erkan zer, NA
Record 27: uysa akgul, CC
Record 28: yakup genc, DC
```

Fig-14: Name based ascending order sort execution

```
emre@ubuntu:~/Downloads/cse344-system-programming-main$ make
./main
Welcome to GTU Student Management System
sortAll "emre.txt"
Enter the sorting option:
1. Sort by name
2. Sort by grade
1
Enter the sorting order:
1. Ascending
2. Descending
1
Displaying all student grades
Record 1: Anthony Clark, CB
Record 2: Anthony Clark, CB
Record 3: Anthony Gonzalez, NA
Record 4: Anthony Lewis, VF
Record 5: Barbara Hernandez, VF
Record 6: Betty Anderson, NA
Record 7: Christopher Anderson, BA
Record 8: Daniel Moore, VF
Record 9: Daniel Moore, VF
Record 10: Eyup Sabri Tuncer, NA
Record 11: Luka Modric, AA
Record 12: Mary Perez, FF
Record 13: Matthew Brown, DD
Record 14: Matthew Lee, FF
Record 15: Matthew Williams, BB
Record 16: Matthew Williams, BB
Record 17: Patricia Robinson, AA
Record 18: Patricia Robinson, AA
Record 19: Robert Davis, BA
Record 20: cengiz selam aban, CC
Record 21: dogan nalcı, FF
Record 22: emre yilmaz, FF
Record 23: enes fol aysu, AA
Record 24: eray ozkan, BB
Record 25: eren cakar, FF
Record 26: erkan zer, NA
Record 27: uysa akgul, CC
Record 28: yakup genc, DC
█
```

Fig-15: Name based descending order sort execution



```

emre@ubuntu:~/Downloads/cse344-system-programming-main$ make
gcc -Wall -Wextra -g -c src/Sort/sort.c
gcc -Wall -Wextra -g -o main main.o searchStudent.o sort.o display.o addStudentGrade.o gtuStudentGrades.o log.o
./main
Welcome to GTU Student Management System
sortAll emre.txt
Enter the sorting option:
1. Sort by name
2. Sort by grade
2
Enter the sorting order:
1. Ascending
2. Descending
1
Displaying all student grades
Record 1: erkan zer, NA
Record 2: Anthony Gonzalez, NA
Record 3: Betty Anderson, NA
Record 4: Eyup Sabri Tuncer, NA
Record 5: Daniel Moore, VF
Record 6: Daniel Moore, VF
Record 7: Anthony Lewis, VF
Record 8: Barbara Hernandez, VF
Record 9: emre yilmaz, FF
Record 10: eren cakar, FF
Record 11: dogan nalcı, FF
Record 12: Matthew Lee, FF
Record 13: Mary Perez, FF
Record 14: Matthew Brown, DD
Record 15: yakup genc, DC
Record 16: uysa akgul, CC
Record 17: cengiz selam aban, CC
Record 18: Anthony Clark, CB
Record 19: Anthony Clark, CB
Record 20: Matthew Williams, BB
Record 21: eray ozkan, BB
Record 22: Matthew Williams, BB
Record 23: Christopher Anderson, BA
Record 24: Robert Davis, BA
Record 25: Patricia Robinson, AA
Record 26: Patricia Robinson, AA
Record 27: enes fol aysu, AA
Record 28: Luka Modric, AA

```

Fig-15: Grade based ascending order sort execution

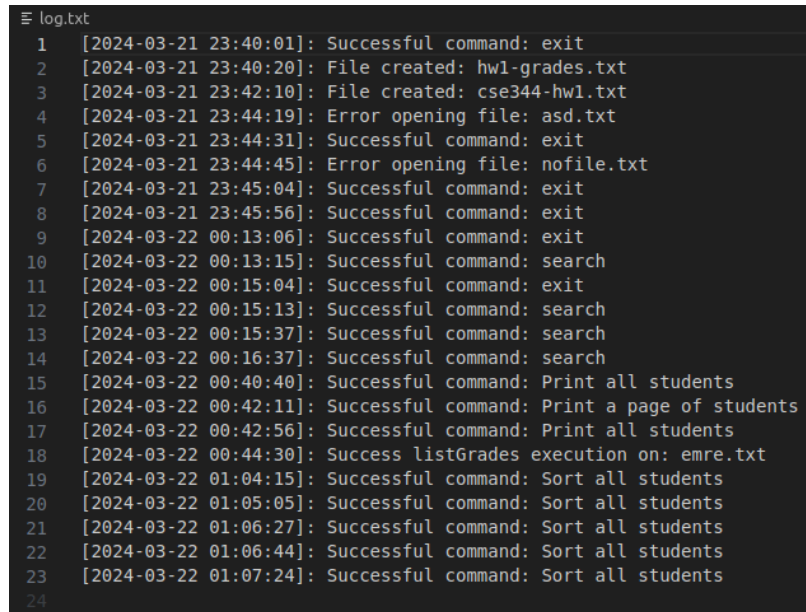
```
emre@ubuntu:~/Downloads/cse344-system-programming-main$ make
./main
Welcome to GTU Student Management System
sortAll "emre.txt"
Enter the sorting option:
1. Sort by name
2. Sort by grade
2
Enter the sorting order:
1. Ascending
2. Descending
2
Displaying all student grades
Record 1: Luka Modric, AA
Record 2: enes fol aysu, AA
Record 3: Patricia Robinson, AA
Record 4: Patricia Robinson, AA
Record 5: Robert Davis, BA
Record 6: Christopher Anderson, BA
Record 7: Matthew Williams, BB
Record 8: eray ozkan, BB
Record 9: Matthew Williams, BB
Record 10: Anthony Clark, CB
Record 11: Anthony Clark, CB
Record 12: cengiz selam aban, CC
Record 13: uysa akgul, CC
Record 14: yakup genc, DC
Record 15: Matthew Brown, DD
Record 16: Mary Perez, FF
Record 17: Matthew Lee, FF
Record 18: dogan nalcı, FF
Record 19: eren cakar, FF
Record 20: emre yilmaz, FF
Record 21: Barbara Hernandez, VF
Record 22: Anthony Lewis, VF
Record 23: Daniel Moore, VF
Record 24: Daniel Moore, VF
Record 25: Eyup Sabri Tuncer, NA
Record 26: Betty Anderson, NA
Record 27: Anthony Gonzalez, NA
Record 28: erkan zer, NA
```

Fig-15: Grade based descending order sort execution

## 7 Log Command

This command is frequently called from within both the child and parent processes. It takes one string parameter. Whenever there is an error or successful outcome in any function of any process, this Log function is called. It is located in the `/src/Log` directory.

When called, it first forks to create a child process and attempts to open the target file. The target file, `"log.txt"` is located in the same directory as `main.c`. It is opened with the `O_WRONLY | O_APPEND | O_CREAT` flags. If the file does not exist, it is created. The current date and time are retrieved from the relevant library. Along with the string parameter received, a line is written to the log file using the `write()` syscall function. Finally, the process is terminated, and the parent process resumes.



```
1 [2024-03-21 23:40:01]: Successful command: exit
2 [2024-03-21 23:40:20]: File created: hw1-grades.txt
3 [2024-03-21 23:42:10]: File created: cse344-hw1.txt
4 [2024-03-21 23:44:19]: Error opening file: asd.txt
5 [2024-03-21 23:44:31]: Successful command: exit
6 [2024-03-21 23:44:45]: Error opening file: nofile.txt
7 [2024-03-21 23:45:04]: Successful command: exit
8 [2024-03-21 23:45:56]: Successful command: exit
9 [2024-03-22 00:13:06]: Successful command: exit
10 [2024-03-22 00:13:15]: Successful command: search
11 [2024-03-22 00:15:04]: Successful command: exit
12 [2024-03-22 00:15:13]: Successful command: search
13 [2024-03-22 00:15:37]: Successful command: search
14 [2024-03-22 00:16:37]: Successful command: search
15 [2024-03-22 00:40:40]: Successful command: Print all students
16 [2024-03-22 00:42:11]: Successful command: Print a page of students
17 [2024-03-22 00:42:56]: Successful command: Print all students
18 [2024-03-22 00:44:30]: Success listGrades execution on: emre.txt
19 [2024-03-22 01:04:15]: Successful command: Sort all students
20 [2024-03-22 01:05:05]: Successful command: Sort all students
21 [2024-03-22 01:06:27]: Successful command: Sort all students
22 [2024-03-22 01:06:44]: Successful command: Sort all students
23 [2024-03-22 01:07:24]: Successful command: Sort all students
24
```

Fig-16: Log File