Systems Programming

You can view all the diagrams from this link:

https://onedrive.live.com/?authkey=%21AOUkg15cHh7Tohg&id=9EDC91C9DA425EC8%2172388&cid=9EDC91C9DA425EC8

SAIF KHATATBEH_18110004

Table of Contents

I. Introduction: 3

II. Part 1- OS-Controlled Resources with System Design: 4

A. Sharing of OS-Controlled Resources on Different OS Components: 4

Process Component: 4

Socket Component: 4

Parallel Component: 4

Memory Component: 4

File Component: 4

Shell Component: 4

B. HTU Instructor’s portal requirements and design: 5

1. HTU requirements for the system (user requirements): 5

2. Requirements for HTU Instructor’s Portal (System Requirements): 6

C. Analyzation of the communication between the different subcomponents in the HTU Instructors Portal: 9

1. Input Subcomponents (File Input Processes): 9

2. Output Subcomponents (File Output Processes): 10

3. Client-Server Communication Subcomponents: 11

4. Shell Communication Subcomponents: 13

5. Concurrent (Threading) Component: 14

D. Evaluation of the Effectiveness of the design in terms of User Requirements: 15

III. Part 2- Implementation of Shell Components: 17

A. Implementation of Shell Scripts: 17

1. Inter-process Communications: 17

2. Managing file permissions: 17

3. Advanced regular expression searches: 17

B. Comparison between Shell Script and C framework: 18

C. Evaluation of the effectiveness of shell components in terms of user requirements: 20

IV. Part 3- Principles of Thread Management: 21

A. Implementation of Threads in Instructors Portal: 21

B. Discussion on the usefulness of Threads: 21

C. Other Parallelization Techniques than the Threads: 21

1. Dependency Analysis: 22

2. Loop Parallelization: 22

V. Part 4.1: User Interface and Socket Programming: 23
A. Implementation of Client-Server System Component: 23
B. Implementation of User Interface Menu: 23
C. Updating Communication Component to handle Concurrent Tasks: 23
VI. Part 4.2: Critical Evaluation of HTU’s Instructors Portal System: 24
VII. Bibliography 25
Appendix 1: 26
A. Admins Functionalities: 27
1. Login/Logout Functionality: 27
2. Saving Session when exiting from the portal: 28
3. Adding and Showing Instructors: 29
4. Initiating Backup Requests: 30
5. Show Files and Directories: 31
B. Instructors Functionality 33
1. Add Lectures 33
2. Adding Students to Lecture: 34
3. Showing Students Added to Lectures: 35
4. Adding Absence to students: 36

Figure 1: Input Components in HTU Instructors Program 9
Figure 2: Output Subcomponents for HTU Instructors portal 10
Figure 3: Shell Communications Subcomponents for HTU Instructors Portal 13
Figure 4: Threading Process for the login 15
Figure 5: Lectures File Data 18
Figure 6: Reading lectures data from shell 18
Figure 7: shell output 18
Figure 8: Reading lecture data from C 19
Figure 9: C output 19

I. Introduction:

I was assigned by my company's manager to partake in developing a state-of-the-art system using C language and shell scripting for HTU university, where they required an instructor's portal with an admin account to follow it up.

This report will follow and explain the journey and the processes that went through when developing the system. It will be split into 4 parts, where part 1 will talk about how OS-controlled resources are shared among processes of different purposes along with the design process of the system. Part 2 is for the shell scripting process. Part 3 is for optimizing the C code by implementing threading. Finally, part 4 is an introduction on a concept called socket programming and how it was implemented in the system.

II. Part 1- OS-Controlled Resources with System Design:

This section will highlight how OS-controlled resources are shared among processes of different purposes where it will touch upon the concept of threading, multiprocessing operating systems, and more. Then a detailed explanation of the system requirements provided by HTU to develop the design of the system which will contribute to developing the full system.

A. Sharing of OS-Controlled Resources on Different OS Components:

operating system (OS) is a program that manages the computer resources and shares them among different components (Hemmendinger, 2021). There are many different components in an operating system, but the main ones are:

Process Component:

The process component is the component responsible of handling different processes defined by the user.

Socket Component:

The socket component is the component responsible for handling the communication between the user (client) and server on the internet.

Parallel Component:

Parallel component is the component responsible for handling multiple of user request among multiple processors simultaneously without affecting the performance of one another.

Memory Component:

Memory component is the component responsible for allocating the users request and inputs to the memory for further usage.

File Component:

File component is the component responsible for handling files operations inside the operating system, from read, write, or append operations depending on the user requests.

Shell Component:

Shell component is the component responsible for handling shell scripts and shell files.

where these OS resources are shared among these components, these shared resources are usually classified into two different categories:

1. Private Resources: which are usually like data structures for ex. Hash tables
2. System Resources: which are components that provides capabilities to contribute to the improvement of the overall system performance. For ex. Files, databases, memory cache

The concept of an operating system is to manage the system resources along with sharing them among system components, which makes the operations done on the operating systems functional. The sharing process starts with the user requesting a task from the operating, this task is created as a process by the process component then based on the criteria of the task, it will pass the other system components until the task (purpose) is finished.

To further understand the sharing process, lets take task that the instructor's portal would perform in the operating system as an example, where the task is to add lectures to a file.

The task starts with the menu where the instructor selects the task to add lectures to a file called lectures.txt, where the operating system will create a process for the task using the Process Component. Then the user would enter the lecture's information, this information is then allocated to the memory through the Memory Component, and then this information are passed to the File Component where based on the task, the file component would know how to operate, and since the task is too to a file, the information that is allocated in the memory would be added the file using the file component.

B. HTU Instructor's portal requirements and design:

The main problem with HTU is that their instructors need a portal to handle their tasks from adding lectures to teach in the system, to adding students to managing their absences to viewing the lectures they teach, and the students registered to the lectures. Alongside this there must be an admin to manage the instructors from adding new instructors to handling system tasks.

1. HTU requirements for the system (user requirements):

Here are the user requirements for the instructors and admin as asked by HTU:

Instructors Portal Requirements:

The first requirement is that the instructor shall login to their account in the system to handle their operations. The login shall only take the instructor's ID (username) and their password to enter their respective portal.

The second requirement is that the instructor shall enter the lectures they want to teach in the system by defining the lecture ID, the name of the lecture, and finally the time for the lecture were after the lectures are entered to the system a message must pop to indicate that the lecture was added successfully.

The third requirement is that the instructor shall have the ability to add students to the lectures they teach by inserting the student's ID and name, and before adding the student the lectures that the instructor teach must be viewed first to allow the instructor to select which lecture, they want to their students to.

The fourth requirement is that the instructor shall have the ability to view the students they teach in the system in accordance with the lecture, where the instructor searches for the lecture by its ID then sees the students they teach along with their respective details, which are student ID and their name, the lecture's ID, and its instructors, and finally student absences in the lecture.

The fifth requirement is that the instructor shall have the ability to exit the system without being logged out were if they want to re-enter the portal, they should still be logged in.

The sixth and final requirement for the instructor portal is that the instructor must have the ability to logout of the system, where if the instructor re-enters the system while logged out, the login menu shall appear prompting the instructor to enter their ID and password.

Admin Portal Requirements:

The first requirement for the admins, like the instructors, they must login to the system. Where the admin must enter the username (rootUser) and the password (0000) to enter the respective portal. These admin data must be entered manually defined in the system.

The second requirement for the admins is that they shall have the ability add new instructors to the system, in other words, creating new users. Were the admin must enter the ID (username) and password of the instructors to the system. After the user has been added, a message must be prompted to the admin that the user was added successfully.

The third requirement for the admins is that they shall have the ability to view the users' details entered in the system, where the information that would be prompted to the admin is the ID (username) and password of the instructors in the system.

The fourth requirement for the admins is that they shall have the ability to view the files and directories in the system. Where there must be a menu with multiple viewing options for the files, which are viewing shell files, text files, executable files, all the files and directories in the system.

The most important requirement to the admin is the fifth requirement where the admin shall have the ability to back up all the important files in the system and save them to multiple zip files. The files that should be backed up are the system itself, the user data, the lecture data, the student's data, all the shell files used for managing and handling files in the system. The system

The sixth and seventh requirements are like the fifth and sixth for the instructors, where the admins must also can exit the portal and return to their logged in state unless they logout of the system.

2. Requirements for HTU Instructor's Portal (System Requirements):

The system requirements are the requirements needed to build the system based on what the user needs (user requirements). Here are the system requirements for HTU instructor's portal.

Login functionality

where the admin and the instructor shall login into their respective dashboard. The user enters their ID and password from the application, there must be a shell script that take these inputs and compare them with the user data saved in a file that holds all the user's data registered in the system. If the user is found or not found, the script shall send a response that must be read by the program and the program must act accordingly. If the user is found, then the user will login into their respective dashboard by reading the type of the user from the file that holds the user's data and if not found then the program will show an error message.

An extra feature to add to the system value is the system shall record every login made by the admins and the instructors and save them into a file that it details must not be shown on the menu. This is done using through the threading component and the file components by creating a child process first through forking the function then using the threading component to create and join threads then calling the shell component using exec to redirect to the shell component with the username, then the shell component would manage the file that has the login numbers of the users.

Logout functionality

where the admin and the instructor must have the ability to logout of the system by sending a response through a script to file that reads the responses sent by the program or shell scripts. Where the response would be a logout response where the user shall exit the program and when they reenter, they will be prompted to the login menu.

Save session when exiting the program where the admin and user must have the ability to exit the program without being logged out which will be done by simply exiting the application without sending a logout request as done in the logout functionality.

Registering instructors into the program

where the admin must have the ability to register the instructors into the system by using a shell script where the admin will enter the username and password of the instructor through the program where the program shall allocate these data into the memory using memory allocation techniques found in the C framework then save these data to file that holds all the user data.

Displaying Instructors data

where the admin must have the ability to view all the users that have been registered in the system which will be done by using a shell command that is responsible of displaying all the contents in the file. In addition to displaying all the user's data, the admin must also have the ability to view the details for a specific instructor by entering their username through the program and by using a shell command that can perform advance regular expression searches the specific user data will be displayed in the program.

Performing backups using a server

which is a the most important part of this program, where the admin must have the ability to initiate a backup request from a server by communicating with the server through TCP/UDP communications. Where the admin shall be prompted with the all the pervious backups performed and their dates, what are the files that the system will backup and finally the choice if the admin want to back up the system. If the admin where to accept performing a backup, then a backup request will be sent to the server via TCP/UDP, or what is most commonly known as socket programming in C, the server will read the request and then checks via shell script if the required files to perform a backup are present in the system, if they are then the backup request will be initiated and the files will be backed up via a shell script that creates a copy of the files and then compress them in a zip file then saves the zip file in the system. If the files are not present the backup will not be initiated. Whether the backup request was initiated or not the server sends a response to the client, the program, informing the status of the backup request whether it was successfully done or not.

Displaying files and directories in the system

where the admins must have the ability to view multiple files and directories presented in the system by using multiple of shell scripts that are specialized in performing searches depending on the file type, text files or executable files or shell files. In addition to this, the admins shall also display all the files and directories presented in the system using a shell script that helps in finding and displaying all the files in the system.

Adding Lectures to the system

where the instructors must have the ability add the lectures they want to teach in the system. This will be done by the instructor entering the lecture details (ID, Name, Time), then these data will be save in a structure that holds different data types then this structure will be allocated into the memory using memory allocation techniques in C programming. Finally, the lecture data will be saved into a file that holds the lectures data where the program will check if the file is present in the system, if it is it will simply add the data into the file, if not then it will create a new file and then saves the data.

Add students to a lecture

where the instructor must have the ability to add a student to lecture already present in the system by entering the lecture ID present in the system then entering the number of students, they want to add to the lecture then entering the students data (ID and name) and saving them in a structure then this structure will be allocated in the memory and save to file that holds all the lectures data that have student in.

Show all students inside a specific lecture

where the instructor must have the ability to view all the students in a specific lecture by entering the lecture's ID then the file that holds all the students with their saved lectures will be opened in the program then the data of the file will be compared with the username of the instructor and the lecture id in the program displayed onto the system.

Add absence to student in a lecture

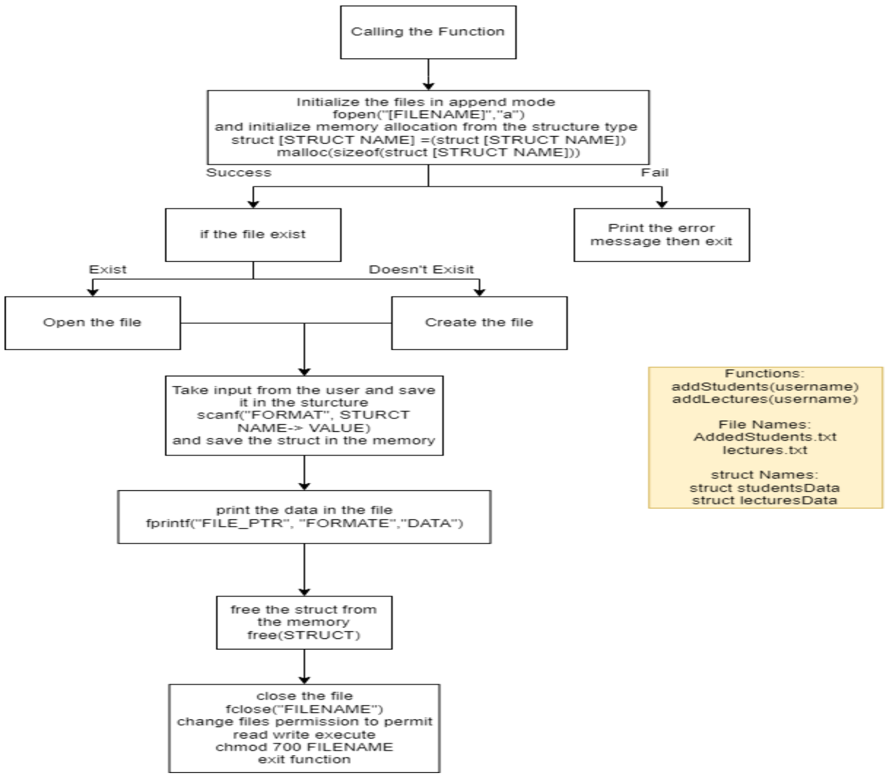
where the instructor shall have the ability to add an absence to the student in a specific lecture by entering the student and lecture id where then these data shall be sent to a shell script to be processed and compared then when the absence is added a response shall be sent to program informing that the absence was added successfully.

C. Analyzation of the communication between the different subcomponents in the HTU Instructors Portal:

A system this big would have multiple of subcomponents that communicate with each other in the system. The subcomponents in the HTU instructor's portal is categorized into 5 main subcomponents which are:

1. Input Subcomponents (File Input Processes):

Input subcomponents are components that takes the input from the C program and inputs them into a file, there are 2 functions that creates and inputs to a file.



Calling the Function
Initialize the files in append mode fopen("[FILENAME]","a") and initialize memory allocation from the structure type struct [STRUCT NAME] =(struct [STRUCT NAME]) malloc(sizeof(struct [STRUCT NAME]))
Success
Fail
if the file exist
Print the error message then exit
Exist
Doesn't Exisit
Open the file
Create the file
Take input from the user and save it in the sturcture scanf("FORMAT", STURCT NAME-> VALUE) and save the struct in the memory
print the data in the file fprintf("FILE_PTR", "FORMATE","DATA")
free the struct from the memory free(STRUCT)
close the file fclose("FILENAME") change files permission to permit read write execute chmod 700 FILENAME exit function
Functions: addStudents(username) addLectures(username)
File Names: AddedStudents.txt lectures.txt
struct Names: struct studentsData struct lecturesData

Figure 1: Input Components in HTU Instructors Program

As seen in figure 1, the process starts with calling the functions responsible for adding to a file, then allocate a block in the memory for a structure that will hold the data. Then a condition occurs to see if the initialization of a file was successful or not, if it was successful then check if the file exist, if not then prompt the user with an error, then exit the function. Then another condition will occur to check if the file exists, if it does exist then open the respective file, if not then create the file. The next step will be is to take the input from the user via the program and store it in the respective structure for which this structure has been already allocated in the memory. Then send the data stored in the structure to the respective file then remove the memory block allocated for the structure and close the file. Finally change the permission of the files added to the system to permit the user to either read, write, or execute these files. This process is as simple and efficient as it gets when trying to add data to a file without having to worry about crashes or a slow process, and even if there where crashes they can be detected easily through error message prompted to the user.

2. Output Subcomponents (File Output Processes):

Output subcomponents are components that take data from a file and outputs (display) them in the program. There are 6 functions that takes data from 6 different files, saves for processing and/or outputs them in the program.

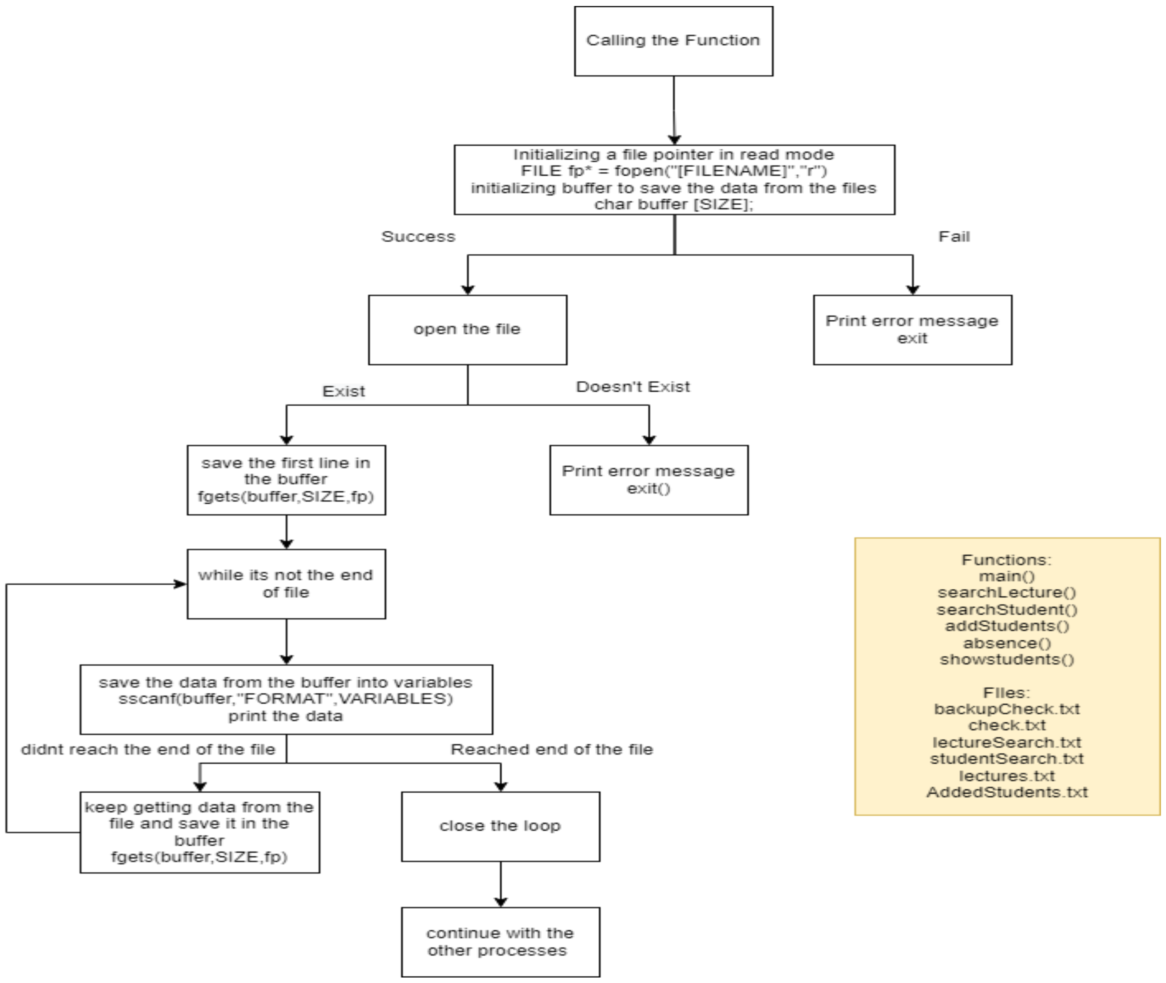


Figure 2: Output Subcomponents for HTU Instructors portal

As seen in figure 2, the process starts when a function is called from the main menu, whether it was the admin or the instructor menu, then a file pointer is initialized in read mode, meaning that its only allowed to read the file data without modifying them, along with a buffer with a defined size to save the data coming from files into the program. This step invokes a condition to see if the file was initialized successfully, if it wasn’t then prompt the user with an error message and exit the function, whereas if it was initialized successfully then try to open the file. For which it invokes another condition to check if the file exists or not. Unlike the input process where it creates the file if it’s not existent in the system, it crashes. So, to avoid that the condition suggests that if the file doesn’t exist then print an error message a exit the function without crashing the application. But if the file does exist then read the data in the file line by line and save the data in the buffer until the it reaches the end of the file then break the loop. The data saved in the buffer can be save in separate variables to be processed later within the application or can just be displayed in the program and then after its done the file must be closed.

This process of reading and outputting file data kind of like the inputting process, but instead of writing to the file, the data that already exist are read and saved in the program which is very helpful if these data required further processing. The main similarity between the input and the output components is the ability to handle crashes will especially if the file wasn’t initialized properly or if it didn’t exist.

3. Client-Server Communication Subcomponents:

Client-Server Communication Subcomponents are components that are responsible to initiate a connection between a client and a server through the internet allowing the sending and receiving of information between both client and server on the same port. The main function responsible for this is the backup function called clientSocket() in the program which will communicate with the backup server called serverC.c and initiate a backup request to backup certain important files in the system.

https://onedrive.live.com/?authkey=%21AOUkg15cHh7Tohg&cid=9EDC91C9DA425EC8&id=9EDC91C9DA425EC8%2172392&parId=9EDC91C9DA425EC8%2172388&o=OneUp

As seen in the diagram, the process starts with the server where the first step is to activate the server before commencing with any backup requests. The process in the server starts with defining a structure that will hold the connection data from the socket description to the address and its length which would be further used along the process. Then a port number of 5678 is defined to base all the communication coming from the clients to this port and connect to the server along with a thread id to allow the creation of threads. After defining the port number and the thread id comes the creation of the socket to help start the server. The socket is created with the IPV4 communication domain with the type of socket stream to get all the incoming client sockets in real-time and with the TCP IP protocol to allow communication through the internet. this invokes a condition that states if the socket wasn’t created successfully then prompt the server with an error message and close it, whereas if it was created successfully then proceed in binding the socket with the address family structure that contains the port number, communication domain needed (IPV4) and with an address. This invokes another condition to check if the address family structure was bonded with the socket, if not then prompt the server with an error message and close the server. However, if the binding was successful then proceed in listening to all the clients that are trying to connect to the server on the same port 5678. Which invokes another condition to see if the server is listening to incoming clients on the port, if not then it will prompt the server with an error message and exit the server. On the hand, if the server successfully listened to the incoming clients, then an infinite loop will be initialized for which it will allocate a block in the memory for all the incoming connections then try to accept this connection from the clients. Going to the client side after the server was open and the function clientSocket() was called in the main instructor menu, at first the address structure along with the port number of server 5678, host name (localhost) and the buffer to save the data from the server are all defined, then a socket is created for the client with the same configuration as the server. This will invoke a condition to check if the socket was created successfully, if not an error message will be prompted to the user and the function will be exited. However, If the socket was created successfully then add the required resources to the address family structure as it was did in the server along with host name as the server address then the client will send a request to connect to the server. This invokes to conditions one in the client side and one in the server side to see if the connection can be accepted by the server and to see if the client is connected or not. If the server did not accept the connection the allocated memory block for the connection will be freed and the loop will continue to listen to more connection in the server side, whereas in the client side an error message will be prompted to the user that connection failed, and function will be exited returning the user to the main menu. However, if the connection was accepted a new thread will be created and then detached to allow for multiple connection without having to wait for this thread to finish to join with the system which in turn calls the thread function called void *process(void *ptr) that takes the connection structure for further processing. While in the client side if the connection was accepted then the backup menu will be prompted to the user asking the user if they want to initiate a backup request, if they don't want to then a request to terminate the created thread will be sent to the server along with closing the client socket which in turn the thread created in the server will read this request and terminate the thread. If the user wants to backup then a request to backup is sent to the server which in turn is sent to the created thread that was created specifically for the requesting client, where this thread will then check the files that are needed to backup exists, if it doesn't then the thread will send an error message to the client that the backup process failed and the thread is terminated where the client will prompt the error message to the user and close the connection with server and exit the function. But if the files are found then a success message will be sent to the client then terminating the thread. While on the client side, the success message will be read, and the backup process is starts and one the backup finished the success message will be prompted to the user then terminated the connection with the server and exiting the function back to the main admin menu. While the server will keep listening for more connections and creating threads for every successful connection.

This whole process is the optimal solution for creating a communication between a client and a server because it avoids as much crashes as possible which may lead to unfavorable results and allows the communication between the server and multiple of admins (clients) since the admin doesn't have to wait for another admin to finish communicating with the server. This also the best solution when trying to backup a system because of the mentioned reasons and the way it handles the backup by searching for the files before initiating a backup is necessary to avoid incomplete backups which may lead to unsatisfactory results.

4. Shell Communication Subcomponents:

Shell Communication Subcomponents are subcomponents that communicates with the system by executing bash script files or bash scripts commands in general which is done by creating a system call in C using either execl() that allows the passing of inputs to a shell script file or command, and system() that only execute a command and returns a value where if the value is less than 0 then the system called failed. The diagram below shows the optimal way to create a system call using forking process, which is a system call that creates a new process called the child process that concurrently runs with the process that created the fork (parent process) (geeksforgeeks, 2019). There are 12 subcomponents in the system for shell communications where 3 of them doesn't take an input. Also, there are 6 shell files being executed in the program and some of these files create a check file for the responses created by the system call.

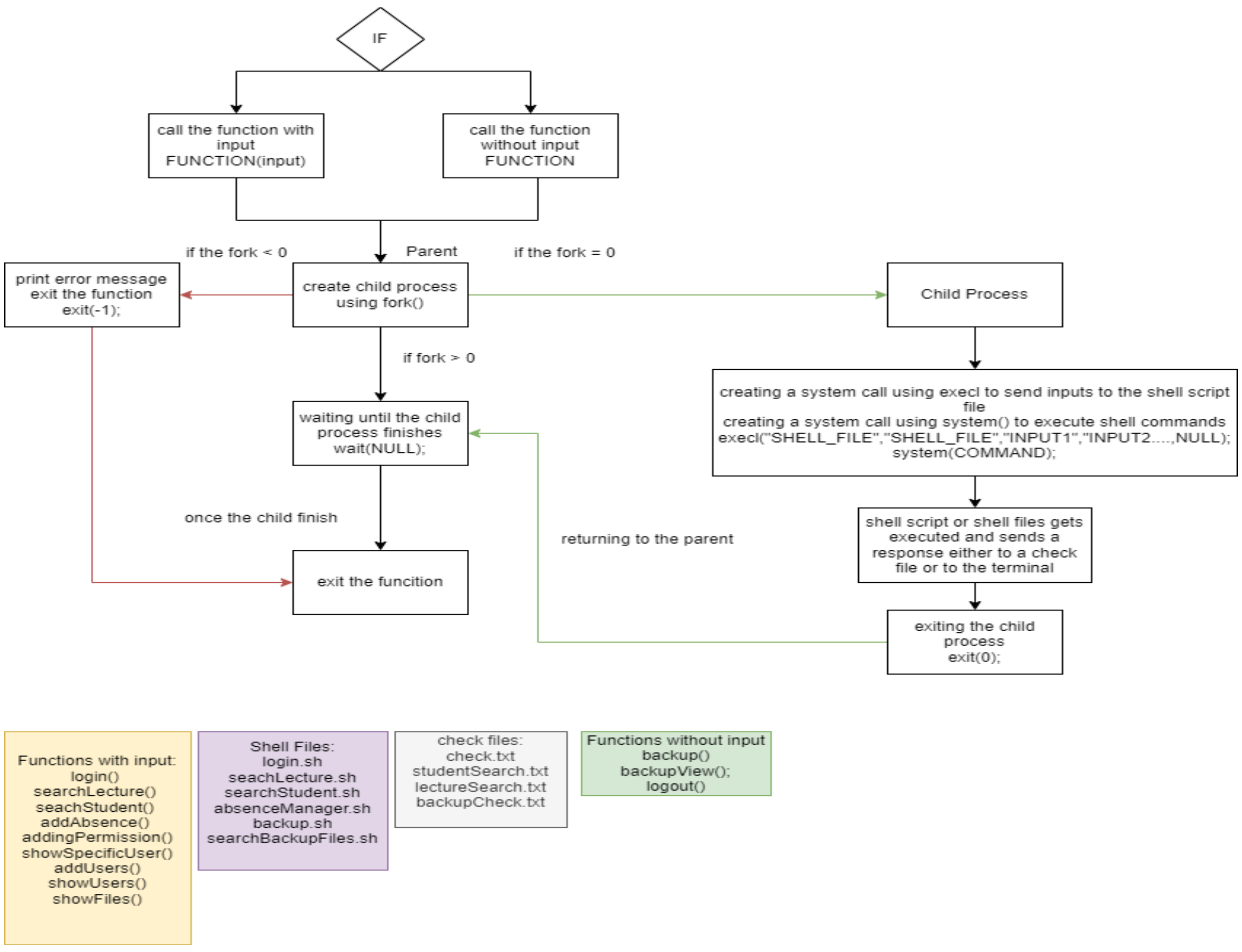

IF
call the function with input FUNCTION(input)
call the function without input FUNCTION
if the fork < 0
Parent
if the fork = 0
print error message exit the function exit(-1);
create child process using fork()
Child Process
if fork > 0
creating a system call using execl to send inputs to the shell script file
creating a system call using system() to execute shell commands
execl("SHELL_FILE","SHELL_FILE","INPUT1","INPUT2....,NULL);
system(COMMAND);
waiting until the child process finishes wait(NULL);
once the child finish
returning to the parent
shell script or shell files gets executed and sends a response either to a check file or to the terminal
exit the funcition
exiting the child process exit(0);
Functions with input: login() searchLecture() seachStudent() addAbsence() addingPermission() showSpecificUser() addUsers() showUsers() showFiles()
Shell Files: login.sh seachLecture.sh searchStudent.sh absenceManager.sh backup.sh searchBackupFiles.sh
check files: check.txt studentSearch.txt lectureSearch.txt backupCheck.txt
Functions without input backup() backupView(); logout()

Figure 3: Shell Communications Subcomponents for HTU Instructors Portal

As seen in figure 3, the process starts with calling a function whether it takes an input or not, then start creating the forking process, where it invokes three conditions. For the child process to be created the fork return value must be 0 if it was bigger then it moved to the parent and if it was smaller then the fork has failed. So this condition suggests that if the fork is less than 0 then prompt an error message to the user then exit the function. Also if the fork is bigger than 0 then wait until the child proccess is finished then exit the function, however if the fork equals 0 then start creating the child process where in this child process a system call execute shell files or shell scripts will be called, meaning either system() or execl(), if the system call was to execute a shell script file then the file must have the execute permission then the scripts in the file is executed and the response will either be prompted in the user terminal or to check file that holds the responses from executing a shell script file.

There are multiple ways to create system call, but I found that this way was the most optimal because with forking you ensure that there will be a response from the system call without affecting the other process. For example, if I wanted to execute a bash command that takes in data from the user and process it in C the forking ensures that the bash will be executed first, and the other process will wait until the bash gets executed to continue processing the data.

5. Concurrent (Threading) Component:

Concurrent component also known as the threading component is a component responsible for allowing the processing and execution of multiple of tasks from multiple of requests without any performance of efficiency loss. the threading component is going to be implemented on the user’s login functionality where when a user login a thread would be created and then the number of the login the user has would be recorded on a separate file that will be shown on the menu as shown in figure 4.

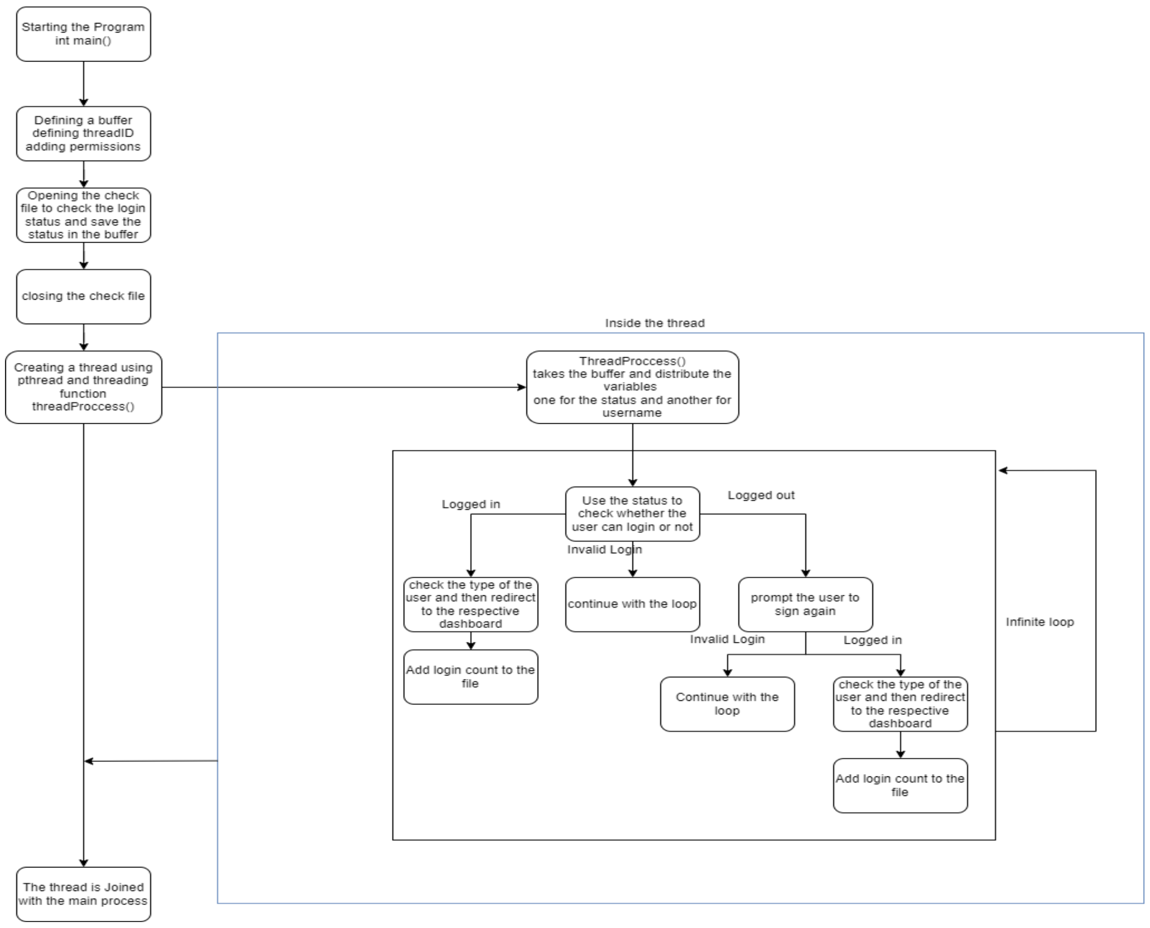

Starting the Program int main()
Defining a buffer defining threadID adding permissions
Opening the check file to check the login status and save the status in the buffer
closing the check file
Creating a thread using pthread and threading function threadProccess()
Inside the thread
ThreadProccess() takes the buffer and distribute the variables one for the status and another for username
Use the status to check whether the user can login or not
Logged in
Logged out
Invalid Login
check the type of the user and then redirect to the respective dashboard
continue with the loop
prompt the user to sign again
Add login count to the file
Invalid Login
Logged in
Continue with the loop
check the type of the user and then redirect to the respective dashboard
Add login count to the file
Infinite loop
The thread is Joined with the main process

Figure 4: Threading Process for the login

D. Evaluation of the Effectiveness of the design in terms of User Requirements:

After designing the main functionalities of the system and implementing them using C framework, I have found that the design is as effective as it can be now in terms of user requirements since it covers what HTU required and more the addition of ability that multiple clients can connect to the server, it opened many possibilities for the future to improve on the server and to it making it more than just a backup server along with opening the possibilities for the utilization of more than one admin.

The best thing about this design is that it embodies what HTU required and elevates on it through the implementation of many subcomponents that are all connected to each other from shell scripts that handles the rigorous work of data searches or comparison making the process much faster than if it was done in C. In addition to this, the design follows the optimal techniques when sending data to a file by using memory allocation techniques in C which would help when dealing with large number of data and avoid the hindrance of slowing down the application, along with the ability to handle crashes in the program by prompting error messages that helps in the identification of the problem occurred then exiting the process to avoid dealing with unfavorable data.

Seeing the effectiveness of the design in terms of user requirements, as effective as it is, there are so much room for improvement as this design allows for more and more improvements which makes more effective to be used by HTU since it can be scaled into much higher levels.

III. Part 2- Implementation of Shell Components:

This section of the report will cover the implementation of shell scripts in file format or as a command that will be used to handle multiple of function which consist of advance regular expression searches, inter-process communications, and managing file permissions that shall be called in the C program per the design implemented. Also, this section will compare between shared functionality between programming them in C or Shell. Finally, an Evaluation to see if the shell was useful in terms of HTU requirements.

A. Implementation of Shell Scripts:

As per the design created in the previous part, there will be some components that needs a shell script to function properly. There will be 6 shell scripts files, one to handle the login functionality, another to search for lectures in the file, another for search the students registered to a specific lecture, another for adding absences to the students registered in a lecture, another to search for the need files to backup, and finally a shell script file for creating backups. In addition to the script's files, some shell script commands will be used within the program to help in various task from searches to managing file permissions. As mentioned, there will be many shell components to handle multiple of tasks, these tasks are:

1. Inter-process Communications:

Inter-process communications means that an input is processed in a shell file and then creates an output and saves it to a file as per standards. An example of an inter-process communication that happens within the HTU system is the absence management where it will take the lecture and student id and send them to the shell script file where they will be responsible to find the required student then modify the absence data to add an absence to the required standard.

2. Managing file permissions:

Managing file permissions is giving the file certain permissions that would help the user to either read, write, or execute a certain file. File permission will be done using the shell command chmod, which is a command in shell that allows to modify file permissions per the user needs (Technology, 2021), where it will be mostly used to give the permission to read, write, and execute for the bash scripts files and other newly created files.

3. Advanced regular expression searches:

Advanced regular expression searches means to search for any file or within any file using certain requirements. Advanced searches will be mostly done using the grep command in shell which is used to search any text within a file or a certain string by giving it some requirements, for example grep "*a" file.txt means that print any text within file.txt that ends with an "a" (Gite, 2021). advanced searches will be mostly used to search for specific user's details in the system and some certain files with different formats.

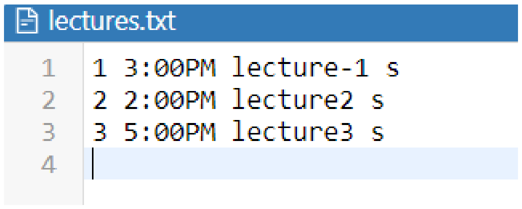
*Note: check the attached system for all the shell file components and script files.

B. Comparison between Shell Script and C framework:

After using shell scripts to implement certain components in the system like the login and file searches, along with implementing C components for file management, I have gained a better understanding on how shell scripts and C framework works and found a lot of similarities in terms of certain functionalities. One of which was the handling of files. I can safely say that I don’t favor a certain language because there are some things that are done in shell better than C and vice versa in terms of file management.

The functionality that I found that is more favorable in shell was how it reads and writes data into a file because it requires less code and functions than in C. to further showcase this difference here is an example where a file contains all the lectures data, the below diagram shows the difference in code between C and shell.

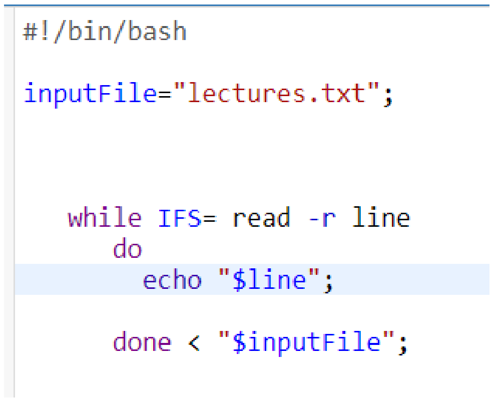
Here is the data inside the lectures.txt file that shall be read by the shell and the C program. Where the data in the file is formatted in this way: ID, Time, lecture Name, Instructor Name



```
lectures.txt
1 3:00PM lecture-1 s
2 2:00PM lecture2 s
3 5:00PM lecture3 s

```

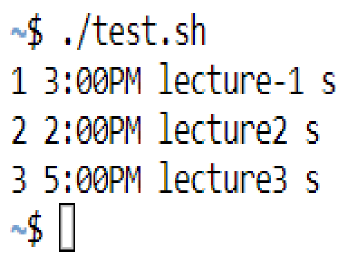
Figure 5: Lectures File Data

Here is how the shell file reads the data from the lecture file



```
#!/bin/bash

inputFile="lectures.txt";



   while IFS= read -r line
      do
        echo "$line";

      done < "$inputFile";
```

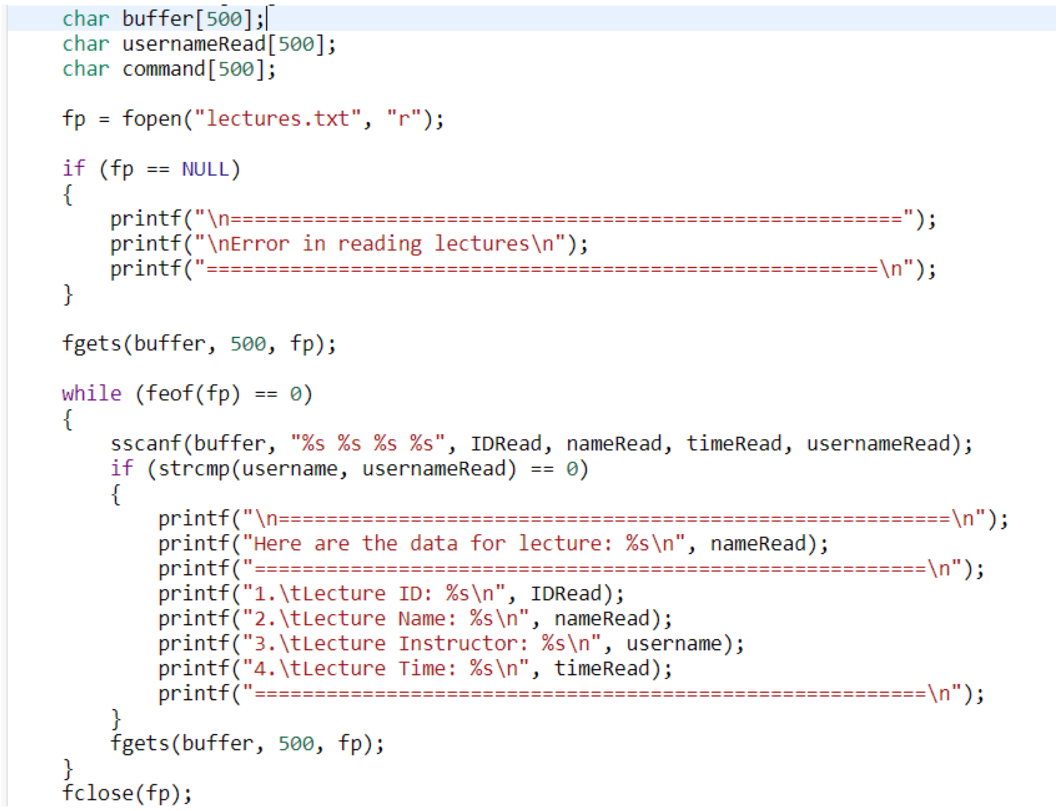
Figure 6: Reading lectures data from shell



```
~$ ./test.sh
1 3:00PM lecture-1 s
2 2:00PM lecture2 s
3 5:00PM lecture3 s
~$ 
```

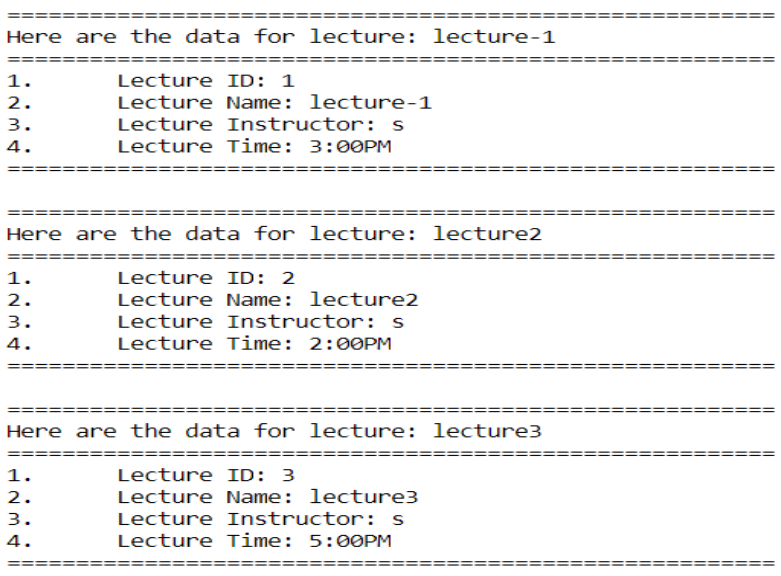
Figure 7: shell output

Here is how the C program reads data from the lecture file:



```
char buffer[500];
char usernameRead[500];
char command[500];

fp = fopen("lectures.txt", "r");

if (fp == NULL)
{
    printf("\n=========================================================");
    printf("\nError in reading lectures\n");
    printf("=========================================================\n");
}

fgets(buffer, 500, fp);

while (feof(fp) == 0)
{
    sscanf(buffer, "%s %s %s %s", IDRead, nameRead, timeRead, usernameRead);
    if (strcmp(username, usernameRead) == 0)
    {
        printf("\n=========================================================\n");
        printf("Here are the data for lecture: %s\n", nameRead);
        printf("=========================================================\n");
        printf("1.\tLecture ID: %s\n", IDRead);
        printf("2.\tLecture Name: %s\n", nameRead);
        printf("3.\tLecture Instructor: %s\n", username);
        printf("4.\tLecture Time: %s\n", timeRead);
        printf("=========================================================\n");
    }
    fgets(buffer, 500, fp);
}
fclose(fp);
```

Figure 8: Reading lecture data from C



```
========================================================
Here are the data for lecture: lecture-1
========================================================
1.      Lecture ID: 1
2.      Lecture Name: lecture-1
3.      Lecture Instructor: s
4.      Lecture Time: 3:00PM
========================================================

========================================================
Here are the data for lecture: lecture2
========================================================
1.      Lecture ID: 2
2.      Lecture Name: lecture2
3.      Lecture Instructor: s
4.      Lecture Time: 2:00PM
========================================================

========================================================
Here are the data for lecture: lecture3
========================================================
1.      Lecture ID: 3
2.      Lecture Name: lecture3
3.      Lecture Instructor: s
4.      Lecture Time: 5:00PM
========================================================
```

Figure 9: C output

As you can see from the figures above, putting aside the aesthetics to beatify the output, the C code is much larger than the shell code since there needs to be variable definitions and the way to open the file where you must specify the mode and make sure that it opened successfully, whereas in the shell there is just the input File definition and the buffer definition. This makes the shell more favorable to those who prefer a smaller, less storage and complex code.

However, in terms of error handling and restriction of opening the file, I have found that C is much more favorable than shell because you can specify which mode you want to open the file with, whether it was writing or reading, without having to change the permissions on the file like the shell if you want to specify the opening of the file mode. And as for error handling, I have that the C is much better in error handling when it comes to the file as it checks if the file exists or not before opening it which reduces the chance of crashes.

In conclusion, there is no better programming language when it comes to the C and shell in terms of file handling since it depends on what the user wants. If the user wants a less complex and less storage intensive way to open a file without having to worry about reading and writing modes, then shell is better option. However, if the user wants to specify which mode to be used when opening the file and wants better error handling without having to care about complexity then C is the far better option for them.

C. Evaluation of the effectiveness of shell components in terms of user requirements:

After implementing the shell components, I have found that they were the better approach when it comes to the some of the requirements for the HTU Portal system, which are the handling of the login, logout, absence, and searches for students and lectures, than C even though they can be implemented in C but can be very time intensive and would make the design less optimized as it is now. Also, there are some requirements that could only be implemented in shell, which where the backup, and file and directory searches, which proved how essential was shell to the system.

In terms of effectiveness, shell was very much effective when handling these requirements in terms of providing a well optimized code structure that wouldn't affect but improve on the performance of the program.

In conclusion, shell was very essential to HTU Instructors portal as it provided a much better way of handling data in files and directories than in C. The best examples to prove the effectiveness of shell were the absence management and creating backups, where the way it handled the absence by search for the student ID, lecture ID, and Instructor name in the file then modifying the absences to add an absence to that specific user was far better than if I would to implement it in C in terms of complexity and code optimization. As for the backup, there is no way that the C program can create a backup of the important files because this needs some kernel commands that can only be found in shell like directory making and file copying.

IV. Part 3- Principles of Thread Management:

This section of the report will cover the implementation of threads to distribute different tasks in the Instructors Portal program. There will be also a discussion about how the implementation of threads was useful to my project and if there are any other parallelization techniques that can be used instead of threading.

A. Implementation of Threads in Instructors Portal:

Threads are very important to allow parallelization of the program which helps improve its performance. In the Instructors Portal system threads are implemented at the login stage where after entering the login data, a thread is created to for checking the user type to redirect to the specified dashboard and adding how many times the user had logged in into the system to a log file that will not be accessed within the program and format of the file is username, NUMLOG.(number of logins).

*Note: check the Instructors Portal Program to see the full threading functionality.

B. Discussion on the usefulness of Threads:

After implementing threads to record the number of logins of a user whether it was an admin or an instructor, I think that I have gained more understanding of what threads are and how useful they can be. In this case, threads were somewhat useful, since the logging of the number of logins can be done easily without the need of threads. However, this is the case when a small number of users logins to the system, if there where many users that want to login, it may cause the system to be slow and may crash, that's when the threading is essential because it allows many users to login simultaneously without slowing down the system because it creates a separate thread for each user then after the user exits or logs out the resources from this thread is joined with the main system.

To sum things up, as the system is right now it doesn't need the threading functionality since there is not going to be as many users that may cause the slowing down of the application and the logging of the number of logins for each user can be done without the use of threads, but in the future threading will be necessary as the number of users will grow meaning the number of logins will grow as well.

C. Other Parallelization Techniques than the Threads:

Threading is a parallelization technique, which means it uses multiple cores and parallel process through using the resources in the system. Threads aren't the only parallelization that can be used, there are a multiple of parallelization techniques that help in parallel processing. The process of parallelization usually involves around partitioning complex program into tasks then mapping them to different processors, maintain communication and synchronization between the processors, and finally load balancing between the processors to reduce loads on the processors. Two of the most important parallelization techniques are (S. Prema, 2013):

1. Dependency Analysis:

Two statements can be only executed in parallel when there is no dependency between instructions. Therefore, these dependencies should be removed to make the code more parallelizable. Dependency analysis plays a major role in the parallelization process where this technique revolves around identifying dependency relation between statements through complex analysis (S. Prema, 2013).

2. Loop Parallelization:

Loop Parallelization technique is one of the most important parallelization techniques since 90% of the execution time is mostly due the presence of loops in codes. Loop Parallelization functions by distributing loop elements into chunks then assigning them to different processors reducing the latency of the loop (S. Prema, 2013).

V. Part 4.1: User Interface and Socket Programming:

This section of the report will talk about the implementation of a new concept called socket programming in C to handle TCP/UDP communications along with a user interface menu to combine all the components that have been implemented to ease the navigation between these components for the user. Also, this section will talk about the update that went to the communication component (Socket Programming) to allow it to handle concurrent communications among available threads.

A. Implementation of Client-Server System Component:

As per user requirements, the need to backup the system is extremely important. And that's where the implementation of a client-server system was implemented which allows the admin to backup all the important files by communicating with the server to initiate a backup request and wait until the backup request is confirmed to start with the backup.

*Note: check the Instructors Portal Program and ServerC for the socket programming implementation.

B. Implementation of User Interface Menu:

After implementing all the components necessary for HTU's Instructor's portal system, they need a way to help them access and use each component. That's where the implementation of a menu comes into place, where the menu was implemented to be displayed on the user's terminal through multiple of printing statements along with a lot of prompts to help the user navigate correctly through the menu and interface.

*Note: check the Instructors Portal Program for the full interface menu.

C. Updating Communication Component to handle Concurrent Tasks:

After implementing the communication component between the client and server to initiate and start backups of the important files presented in the system, there was a problem. The problem was that when a user initiates a backup, the other users must wait until the backup is finished which can prove to be a problem for a system that revolves on having multiple users. The solution was to handle the backups between users concurrently, meaning that multiple of users can initiate a backup request without having to wait for each user to finish with their request. This was implemented using threads where each user backup request is handled on a separate thread then joined with the main process when the user disconnects from the server.

*Note: check the ServerC program for the full implementation of threads.

VI. Part 4.2: Critical Evaluation of HTU's Instructors Portal System:

After implementing the full system components on HTU's Instructor's portal program in terms of what the user required. I have a greater understanding of the program and its functionalities.

In terms of functionalities, the system is fully functional, where HTU required functionalities that must be implemented on the system for both the admin and instructors' users, which are login and logout, adding instructors through the admin, adding lectures and students to these lectures through the instructors, backing up the important files and programs through the admin and many more functionalities to add on the user experience such as complex searches for admins and instructors. These functionalities have been tested thoroughly as seen in the screenshots in the appendix which show that the system's functionality have met the expectations and requirements of the user (HTU).

In terms of interface, a menu was implemented for the users (instructors and admins) to better navigated between the system components that have been used to implement the instructor's portal to execute multiple of tasks. This menu was implemented to be viewed on the terminal using multiple of print statements. This is as basic as it gets when implanting a menu and although its functional, its no way near what I have envisioned for the future, since this menu required the user to navigate through writing the choice number (meaning navigation through the keyboard) which makes the user experience insufferable, since user interface and experience goes hand in hand, which means that if the user interface is bad the experience of the user would be at its worst. The reason I have implemented the menu that way was to test if the system is fully functional and to give HTU a chance to perform end to end testing before using it on a wide scale. However, what I envisioned the menu for the future is to be implemented using graphical user interface components, which would have colors, logos, font styles and more, along with the navigation that would be through using the mouse to click item in the menu to be redirect to other submenus. Implementing this solution would enhance the user experience on multiple of levels.

In terms of handling concurrent tasks, the system is capable of handling multiple of tasks at the same time without having to wait for the task to be completed. This was implemented using the threading parallelization technique, where it was implemented in two places in the instructor's portal system. The first place was in the login functionality, where a thread is created when a user login to their dashboard and the number of logins is increased every time the user logins to their dashboard. The implementation of threads in the login functionality was unnecessary at this stage of the system since there aren't many users in the system and this functionality can be easily implemented without the use of thread. However, that doesn't say about the future where the implementation of threads would be necessary because the number of users on the next stages of the system would increase which means that threads would be necessary since they enhance the system performance and efficiency avoiding the system to be slow. The second place where the threads where implanted was in the socket component, specifically on the sever side, where the implementation of threads was very necessary and important for every stage in the system, to allow multiple of admins to connect to the server to perform multiple of backup requests and without threads this wouldn't be even possible to implement.

To Conclude, this system its still in its first stage and since we made sure that every component implemented in HTU’s Instructors Portal is fully functional and up to their and the competition standards, the next stages would have more functionality and more improvements to make sure that this system is out of the competitions reach and provided the instructors and admins with the best experience.

VII. Bibliography

geeksforgeeks, 2019. *fork() in C.* [Online]
Available at: https://www.geeksforgeeks.org/fork-system-call/
[Accessed 25 January 2022].

Gite, V., 2021. *How To Use grep Command In Linux / UNIX With Practical Examples.* [Online]
Available at: https://www.cyberciti.biz/faq/howto-use-grep-command-in-linux-unix/
[Accessed 25 January 2022].

Hemmendinger, D., 2021. *operating system.* [Online]
Available at: https://www.britannica.com/technology/operating-system
[Accessed 2 January 2022].

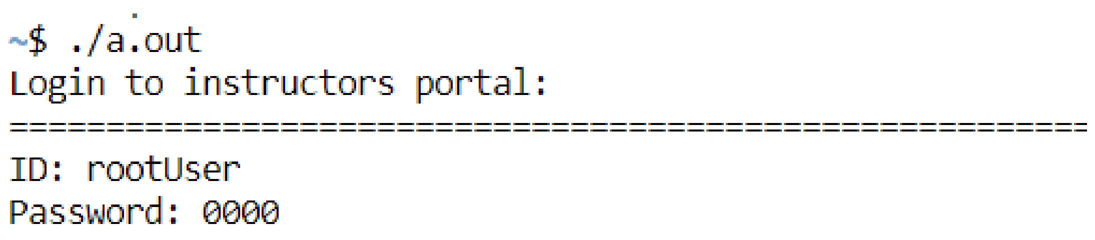
S. Prema, R. J., 2013. *Analysis of Parallelization Techniques and Tools.* [Online]
Available at: https://www.ripublication.com/irph/ijict_spl/17_ijictv3n5spl.pdf
[Accessed 1 February 2022].

Technology, B. U. I. S. &., 2021. *Change Mode (chmod) Command.* [Online]
Available at: https://www.bu.edu/tech/support/research/system-usage/using-scc/managing-files/chmod/
[Accessed 25 January 2022].

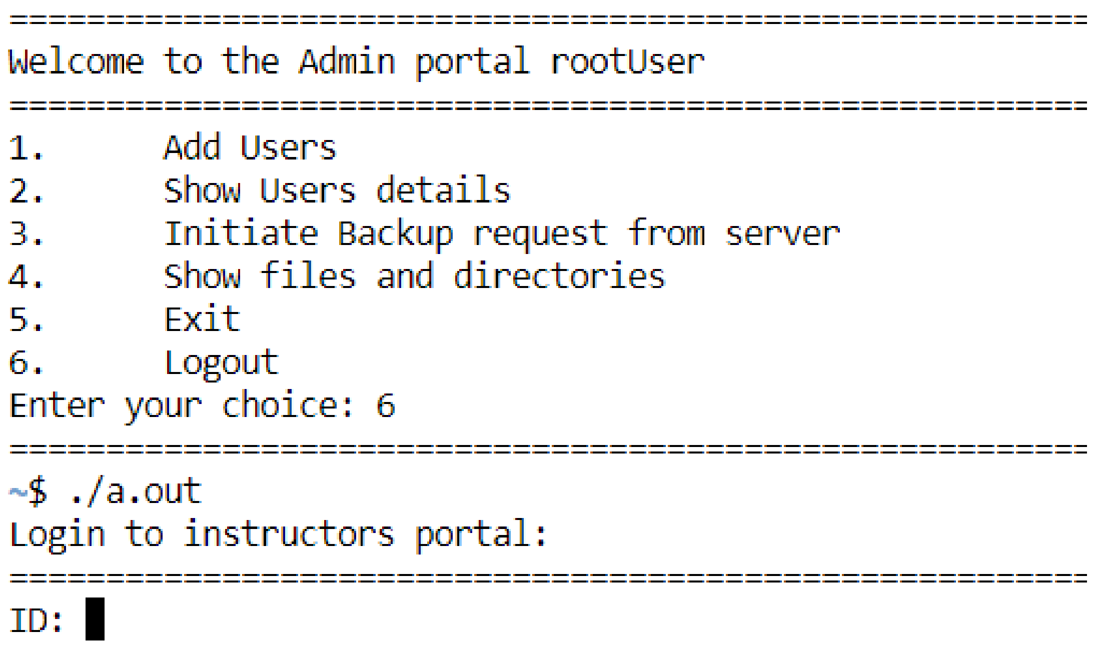
Appendix 1:
Screenshots for testing the functionality

A. Admins Functionalities:

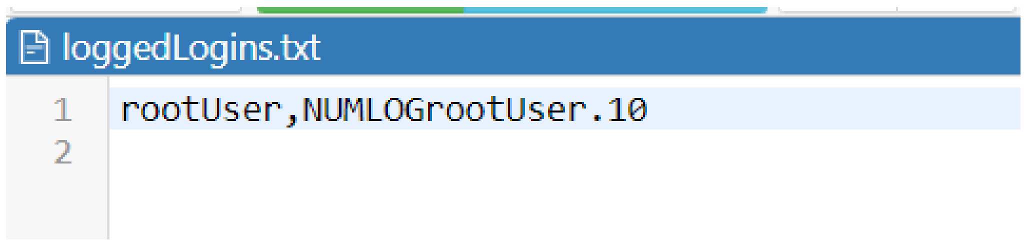
1. Login/Logout Functionality:



```
~$ ./a.out
Login to instructors portal:
========================================================
ID: rootUser
Password: 0000
```



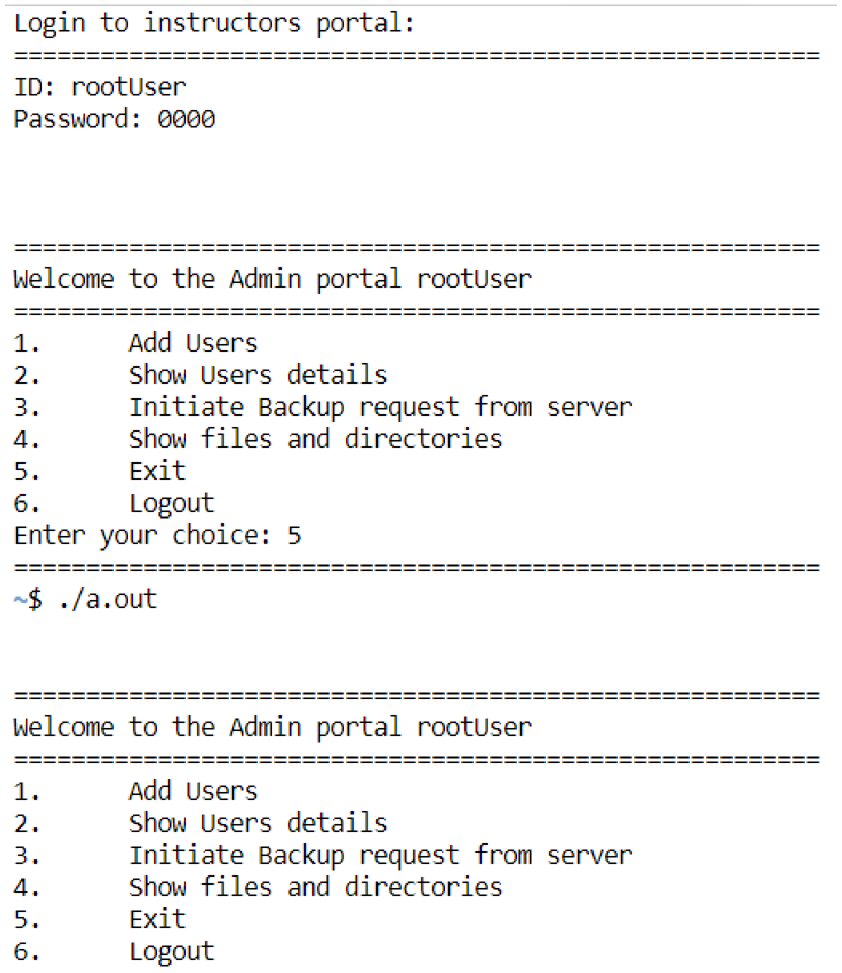
```
========================================================
Welcome to the Admin portal rootUser
========================================================
1.      Add Users
2.      Show Users details
3.      Initiate Backup request from server
4.      Show files and directories
5.      Exit
6.      Logout
Enter your choice: 6
========================================================
~$ ./a.out
Login to instructors portal:
========================================================
ID: 
```



loggedLogins.txt

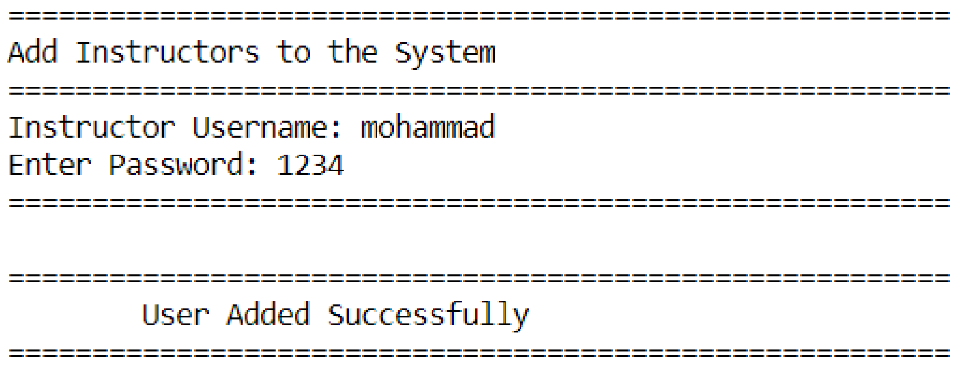
```
rootUser,NUMLOGrootUser.10
```

2. Saving Session when exiting from the portal:

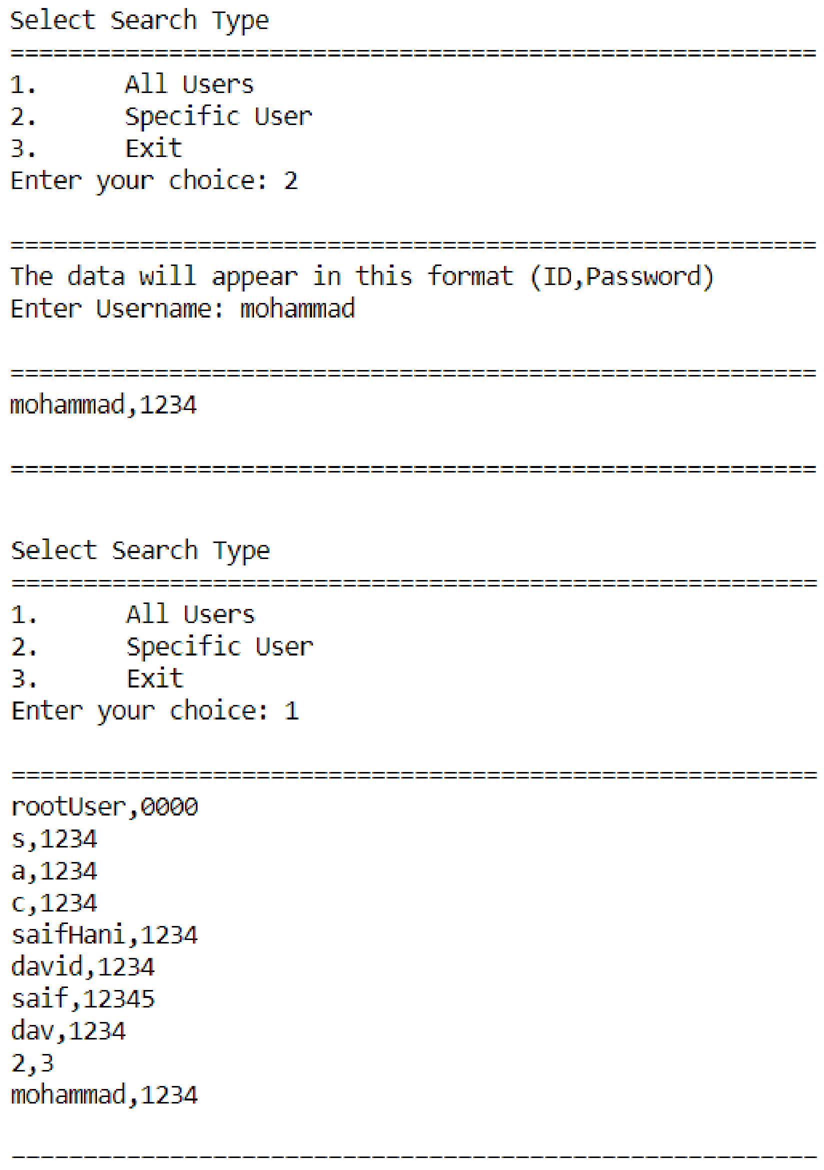


```
Login to instructors portal:
========================================================
ID: rootUser
Password: 0000



========================================================
Welcome to the Admin portal rootUser
========================================================
1.      Add Users
2.      Show Users details
3.      Initiate Backup request from server
4.      Show files and directories
5.      Exit
6.      Logout
Enter your choice: 5
========================================================
~$ ./a.out


========================================================
Welcome to the Admin portal rootUser
========================================================
1.      Add Users
2.      Show Users details
3.      Initiate Backup request from server
4.      Show files and directories
5.      Exit
6.      Logout
```

3. Adding and Showing Instructors:

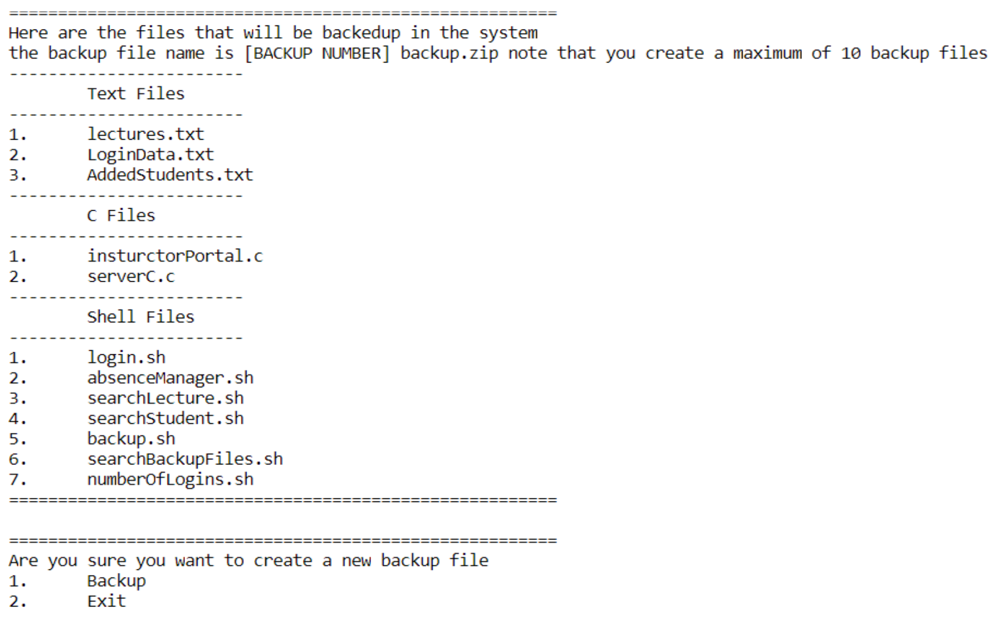


```
========================================================
Add Instructors to the System
========================================================
Instructor Username: mohammad
Enter Password: 1234
========================================================

========================================================
        User Added Successfully
========================================================
```

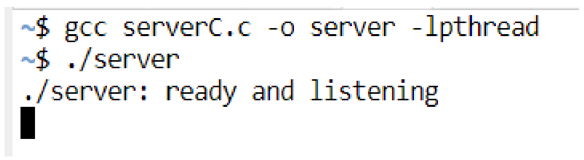


```
Select Search Type
========================================================
1.      All Users
2.      Specific User
3.      Exit
Enter your choice: 2

========================================================
The data will appear in this format (ID,Password)
Enter Username: mohammad

========================================================
mohammad,1234

========================================================


Select Search Type
========================================================
1.      All Users
2.      Specific User
3.      Exit
Enter your choice: 1

========================================================
rootUser,0000
s,1234
a,1234
c,1234
saifHani,1234
david,1234
saif,12345
dav,1234
2,3
mohammad,1234

--------------------------------------------------------
```

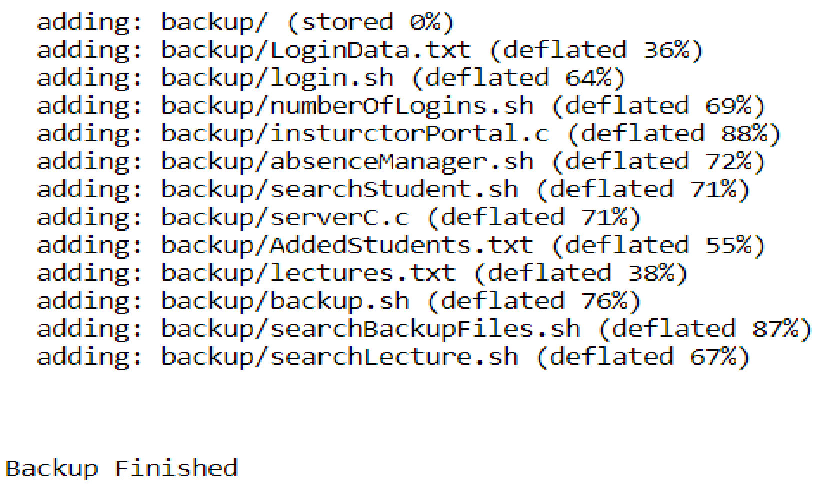
4. Initiating Backup Requests:



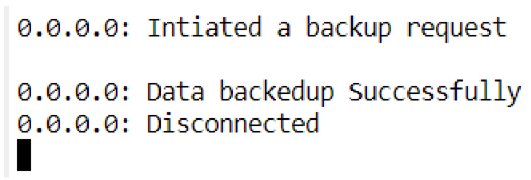
```
=========================================================
Here are the files that will be backedup in the system
the backup file name is [BACKUP NUMBER] backup.zip note that you create a maximum of 10 backup files
------------------------
        Text Files
------------------------
1.      lectures.txt
2.      LoginData.txt
3.      AddedStudents.txt
------------------------
        C Files
------------------------
1.      insturctorPortal.c
2.      serverC.c
------------------------
        Shell Files
------------------------
1.      login.sh
2.      absenceManager.sh
3.      searchLecture.sh
4.      searchStudent.sh
5.      backup.sh
6.      searchBackupFiles.sh
7.      numberOfLogins.sh
=========================================================

=========================================================
Are you sure you want to create a new backup file
1.      Backup
2.      Exit
```



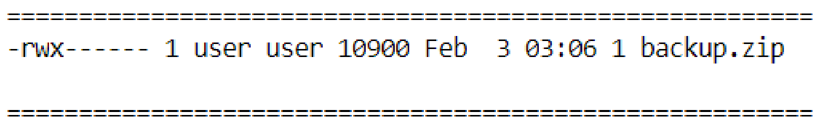
```
~$ gcc serverC.c -o server -lpthread
~$ ./server
./server: ready and listening
```



```
  adding: backup/ (stored 0%)
  adding: backup/LoginData.txt (deflated 36%)
  adding: backup/login.sh (deflated 64%)
  adding: backup/numberOfLogins.sh (deflated 69%)
  adding: backup/insturctorPortal.c (deflated 88%)
  adding: backup/absenceManager.sh (deflated 72%)
  adding: backup/searchStudent.sh (deflated 71%)
  adding: backup/serverC.c (deflated 71%)
  adding: backup/AddedStudents.txt (deflated 55%)
  adding: backup/lectures.txt (deflated 38%)
  adding: backup/backup.sh (deflated 76%)
  adding: backup/searchBackupFiles.sh (deflated 87%)
  adding: backup/searchLecture.sh (deflated 67%)


Backup Finished
```



```
0.0.0.0: Intiated a backup request

0.0.0.0: Data backedup Successfully
0.0.0.0: Disconnected
```



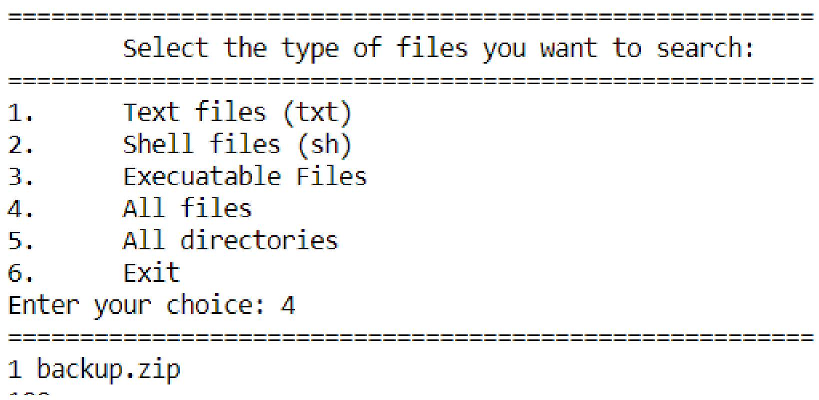
```
=========================================================
-rwx------ 1 user user 10900 Feb  3 03:06 1 backup.zip

=========================================================
```

5. Show Files and Directories:

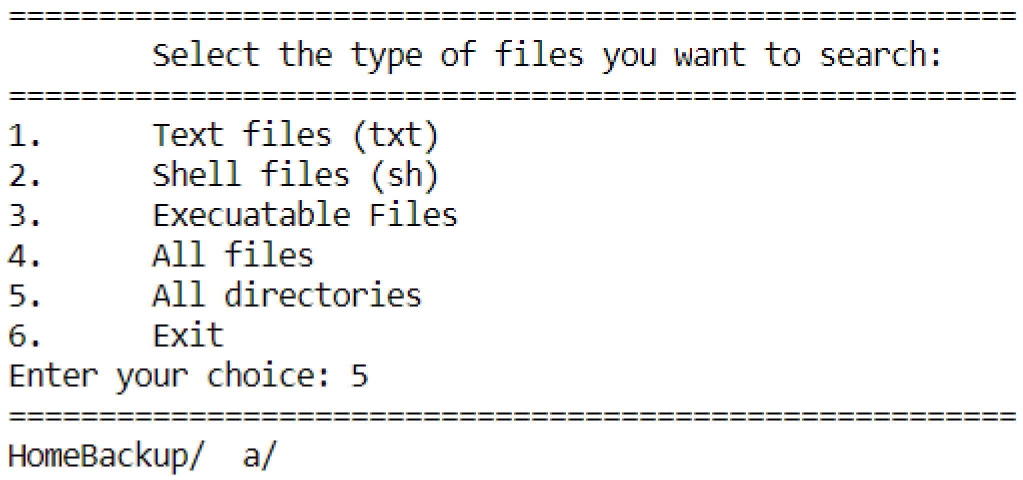
```
========================================================
        Select the type of files you want to search:
========================================================
1.      Text files (txt)
2.      Shell files (sh)
3.      Execuatable Files
4.      All files
5.      All directories
6.      Exit
Enter your choice: 1
========================================================
-rw-r--r-- 1 user user    21 Jan  6 12:30 2021-12-18-223738.txt
-rwx------ 1 user user   364 Jan 24 01:44 AddedStudents.txt
-rw-r--r-- 1 user user    98 Feb  3 03:01 LoginData.txt
-rwx------ 1 user user    36 Jan 23 18:55 available-guitars.txt
```

```
========================================================
        Select the type of files you want to search:
========================================================
1.      Text files (txt)
2.      Shell files (sh)
3.      Execuatable Files
4.      All files
5.      All directories
6.      Exit
Enter your choice: 2
========================================================
-rwxr-xr-x 1 user user   191 Dec 19 07:55 2021-12-19-081533.sh
-rwx------ 1 user user  1259 Jan 12 23:56 absenceManager.sh
-rwx------ 1 user user  2031 Jan 25 07:04 backup.sh
```

```
========================================================
        Select the type of files you want to search:
========================================================
1.      Text files (txt)
2.      Shell files (sh)
3.      Execuatable Files
4.      All files
5.      All directories
6.      Exit
Enter your choice: 3
========================================================
./studentSearch.txt
./client
./lectures.txt
```



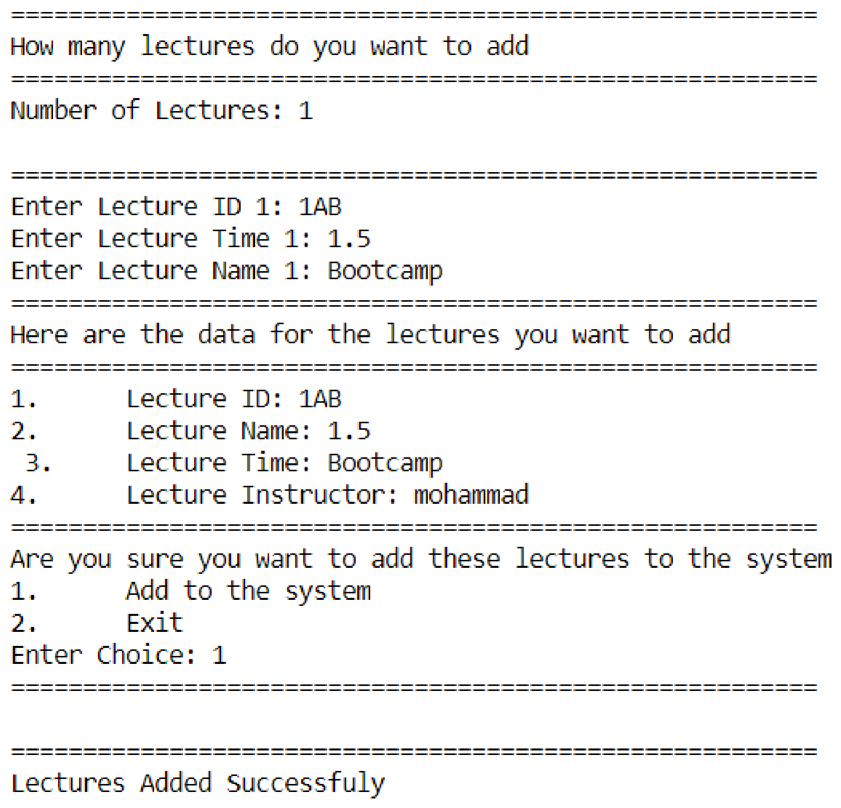
```
=========================================================
        Select the type of files you want to search:
=========================================================
1.      Text files (txt)
2.      Shell files (sh)
3.      Execuatable Files
4.      All files
5.      All directories
6.      Exit
Enter your choice: 4
=========================================================
1 backup.zip
```



```
=========================================================
        Select the type of files you want to search:
=========================================================
1.      Text files (txt)
2.      Shell files (sh)
3.      Execuatable Files
4.      All files
5.      All directories
6.      Exit
Enter your choice: 5
=========================================================
HomeBackup/  a/
```

B. Instructors Functionality

1. Add Lectures

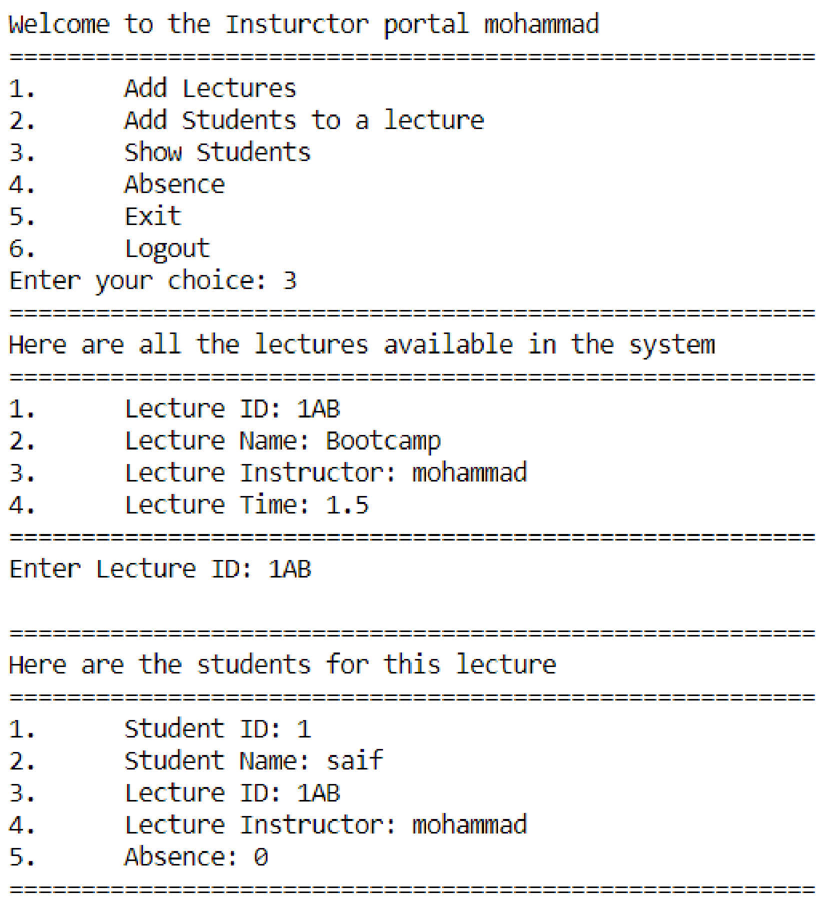


```
========================================================
How many lectures do you want to add
========================================================
Number of Lectures: 1

========================================================
Enter Lecture ID 1: 1AB
Enter Lecture Time 1: 1.5
Enter Lecture Name 1: Bootcamp
========================================================
Here are the data for the lectures you want to add
========================================================
1.      Lecture ID: 1AB
2.      Lecture Name: 1.5
 3.     Lecture Time: Bootcamp
4.      Lecture Instructor: mohammad
========================================================
Are you sure you want to add these lectures to the system
1.      Add to the system
2.      Exit
Enter Choice: 1
========================================================

========================================================
Lectures Added Successfuly
```

2. Adding Students to Lecture:

```
========================================================
Select Lecture ID: 1AB
========================================================



========================================================
How many Students do you want to the lecture
========================================================
Number of Students: 1

========================================================

Enter Student ID: 1
Enter Student Name: saif

========================================================
Here are the data for students you want to add
========================================================
1.      Student ID: 1
2.      Student Name: saif
3.      Lecture ID: 1AB
4.      Lecture Instructor: mohammad
========================================================
Are you sure you want to add these students to the system
1.      Add to the system
2.      Exit
Enter Choice: 1
saif Information Added:

========================================================
```

3. Showing Students Added to Lectures:



```
Welcome to the Insturctor portal mohammad
========================================================
1.      Add Lectures
2.      Add Students to a lecture
3.      Show Students
4.      Absence
5.      Exit
6.      Logout
Enter your choice: 3
========================================================
Here are all the lectures available in the system
========================================================
1.      Lecture ID: 1AB
2.      Lecture Name: Bootcamp
3.      Lecture Instructor: mohammad
4.      Lecture Time: 1.5
========================================================
Enter Lecture ID: 1AB

========================================================
Here are the students for this lecture
========================================================
1.      Student ID: 1
2.      Student Name: saif
3.      Lecture ID: 1AB
4.      Lecture Instructor: mohammad
5.      Absence: 0
========================================================
```

4. Adding Absence to students:

```
========================================================
Welcome to the Insturctor portal mohammad
========================================================
1.      Add Lectures
2.      Add Students to a lecture
3.      Show Students
4.      Absence
5.      Exit
6.      Logout
Enter your choice: 4
========================================================

========================================================
Here are the data for: saif
========================================================
1.      Student ID: 1
2.      Student Name: saif
3.      Lecture ID: 1AB
4.      Lecture Instructor: mohammad
5.      Absence: 0
========================================================
Enter the required data to add the absence
Enter Student ID: 1
Enter Lecture ID: 1AB

========================================================
Here are all the lectures available in the system
========================================================
1.      Lecture ID: 1AB
2.      Lecture Name: Bootcamp
3.      Lecture Instructor: mohammad
4.      Lecture Time: 1.5
========================================================
Enter Lecture ID: 1AB

========================================================
Here are the students for this lecture
========================================================
1.      Student ID: 1
2.      Student Name: saif
3.      Lecture ID: 1AB
4.      Lecture Instructor: mohammad
5.      Absence: 1
========================================================
```