COMP2432

PROJECT REPORT

Study Scheduling System (S3)

Group 20

Cheung Ka Ho 17047931D

Ho Kin Ling 17056079D

Ngan Ting Cheuk 17061434D

So Hiu Tung 17057557D

Yan Ho Wang 17057122D

1. Introduction

This project mainly implements a study scheduling system. This system aims to help university students to manage their everyday schedule automatically. These days, university students often have a lot of assignments, projects and other extracurricular activities in one semester, especially during the semester’s end. Most of the students may not have a good schedule for the activities and tasks that have been mentioned above. As a result, they may have insufficient time to finish their works. They could not handle their tasks in a much more efficient and effective way. Thus, the project introduces a system that could help students to generate their everyday schedule according to a specific scheduling algorithm. Students no longer need to worry about how to plan their schedule.

2. Scope

In this project, several operating system topics have been covered.

One of the topics would be the process creation in Linux/Unix. By using the function fork(), we are going to make use of this function to create some child processes in the project. It is an important technique in our project because we would like to split the whole system into different small processes. These different small processes will share the same resources (e.g. pipe) and execute concurrently. Thus, we would like to implement this feature in our project.

Another topic would be interprocess communication. Communication between different processes is essential. It allows them to transfer data and share the resources. The message passing mechanism is achieved through pipes. By making use of system call pipe(), we could use an unnamed pipe for interprocess communication.

Process synchronization is covered in this project, we want to ensure the orderly execution of read() / write() operations to the shared resources (i.e. pipe) of the communicating processes in the program.

CPU scheduling is also a main part of the project. There are several scheduling algorithms in CPU scheduling. These kinds of scheduling algorithm could be used in the study scheduling system. Different scheduling algorithm which has their own specific feature could achieve different scheduling effect. We would like to implement some of the scheduling algorithms for producing a timetable that could let university students have a better schedule in their studies.

3. Concept

As this project is mainly focusing on developing a study scheduling system, several CPU scheduling algorithms would be possible and effective for implementing in our system.

First Come First Serve (FCFS) is one of the CPU scheduling algorithms. It is a simple scheduling algorithm that its name has indicated the principle of it. In CPU, the processes which come first should be handled first according to the arrival time. When applying in our study scheduling system, the processes should be some tasks such as assignments, projects, revisions. These tasks should be handled one by one according to the date of release. However, the downside of this algorithm is that it has a convoy effect, especially when all the tasks come at the same time.

Earliest Deadline First (EDF) would be also an effective scheduling algorithm for our system. The processes will declare their deadlines when they arrive. Therefore, the process with the earliest deadline will be executed first. It is similar to the situation of our project. Assignments and projects usually have their own deadlines. Most of the students handle them according to different deadlines. By implementing this scheduling algorithm, students could follow the timetable to finish their tasks efficiently.

4. Software Structure

In our software, there are three main components: Input module, Scheduling module and Output & Analyzer module. The whole software is based on process programming, which makes use of the fork(), pipe(), read(), write() system calls.

Input module is the parent process, it generates two more child processes for Scheduling module and Output & Analyzer module by calling fork(). Pipe() is used for communication between Input module and Scheduling module, Scheduling module and Output & Analyzer module.

Data flow: user input → input module → scheduling module → output & analyzing modules → output files

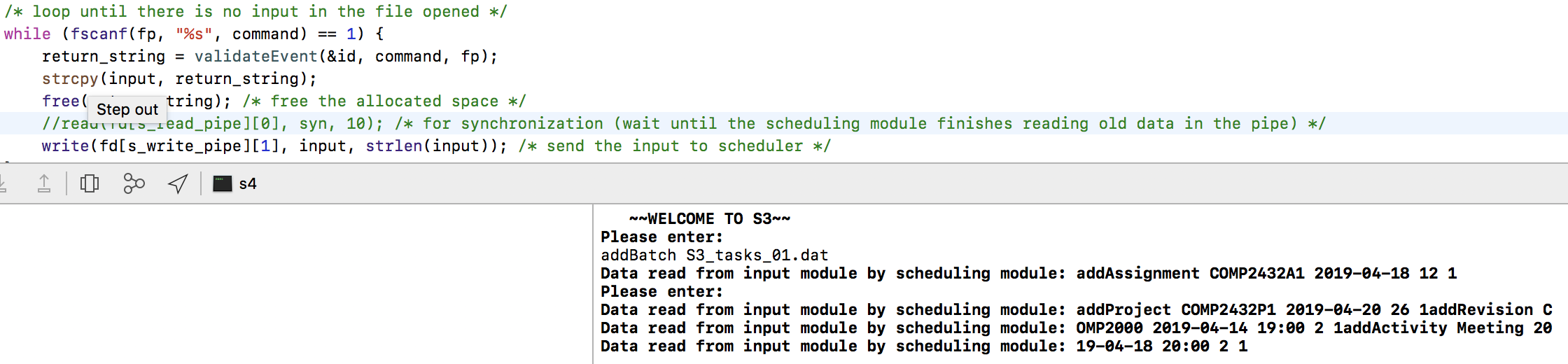
Implementation of process synchronization

Input module writes a set of data to Scheduling module and Scheduling module write a set of data to Output & Analyzer module when they finish processing that set data (ready to send data) and receives (read) notification.

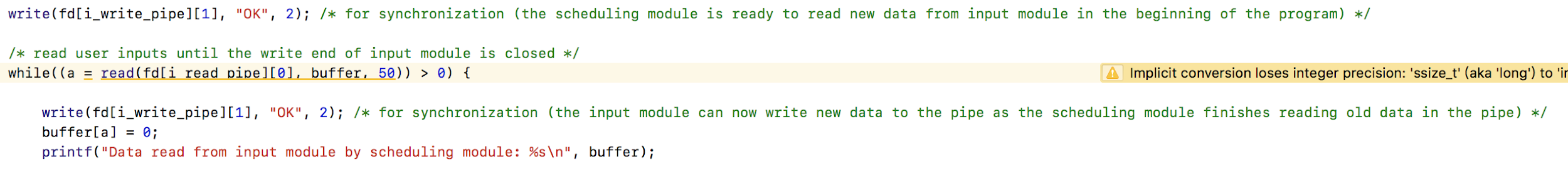
Scheduling module send (write) notification to Input module and Output & Analyzer module send (write) notification to Scheduling module when the program starts or they finish reading old data (ready to read a new set of data).

In doing so, mixing of data from multiple writes in a pipe can be avoided as data sender can only write a new set of data to the pipe when the data receiver finish reading the old set of data and send a notification to the data sender.

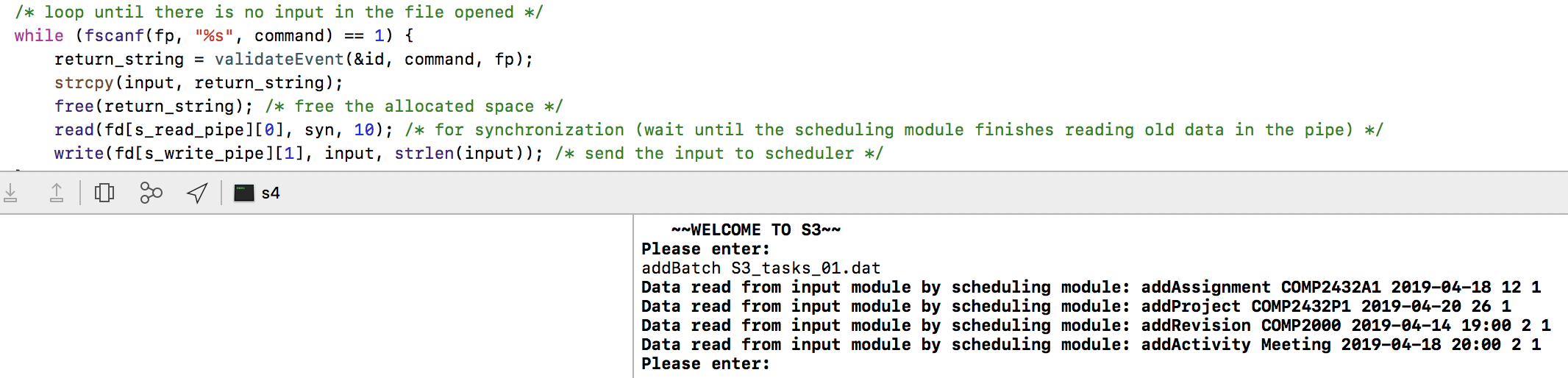
For example, when there is no synchronization of reading/writing between Input module and Scheduling module and the input module can write at any time (the “read” instruction for reading notification becomes a comment), multiple writes may be done before the scheduling module starts to read. When the scheduling module reads from the pipe, it may read a string mixed from data from multiple writes (e.g. “COMP2000 2019-04-14 19:00 2 1 addActivity Meeting 20”) which is not excepted and hence cannot handle the data properly.



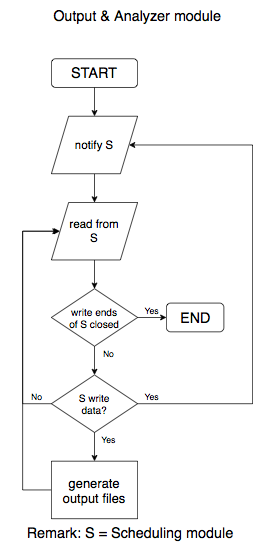
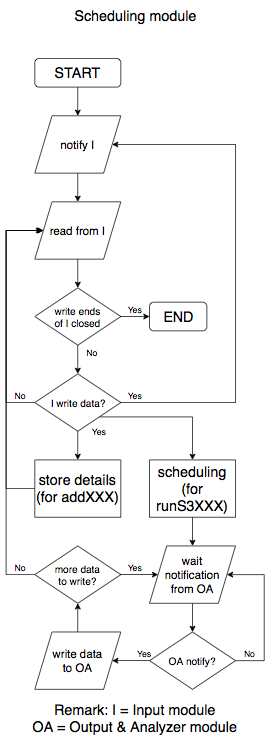
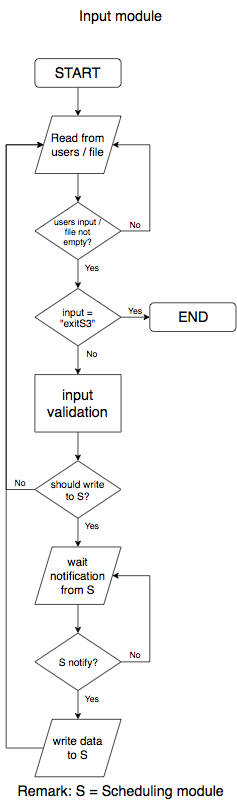
Code for reading and printing input from input module in the scheduling module



When there is synchronization (wait for (read) notification before writing data), the scheduling module can now read data from one and only one write in one read and hence can handle the data properly



System flowcharts:



Input module:

Input: Inputs from users

Output: a string concatenating a input and a character for indicating validity (for addPeriod, addAssignment, addProject, addRevision, addActivity) / a string concatenating a input (for runS3)

Input module first read the first string of user input to determine what type of command the input is (e.g. addPeriod, runS3, etc).

If the type of input is exitS3, all the processes will end.

If the type of input is unknown, an error message will be printed.

If the type of input is addPeriod, function validatePeriod is called to read other parameters and validate this input.

If the type of input is addAssignment / addProject / addRevision / addActivity, function validate Event is called to read other parameters and validate this input.

If the type of input is addBatch, function validateEvent will be called once each request in the file to read parameters and validate the requests.

In our program, the maximum range of period that can be handled is from 2019-04-08 to 2019-04-21 from 19:00 to 23:00. If the input date or time is out of this range, the input will be treated as invalid. Also, if the duration is smaller than or equal to 0, the input will also be treated as invalid by setting an indicator (a character which can be ‘0’ which represents invalid / ‘1’ which represents valid).

For both validatePeriod and validateEvent, a string concatenating the input and the character for indicating validity will be returned.

(format: ”[user\_input] [indicator]”)

If the type of input is runS3, a string concatenating the input will be created.

(format: ”[user\_input]”)

Except for unknown type of input, “exitS3”, invalid addPeriod and invalid algorithm for runS3 , input module will write the string generated to the scheduling module when scheduling module is reading to read.

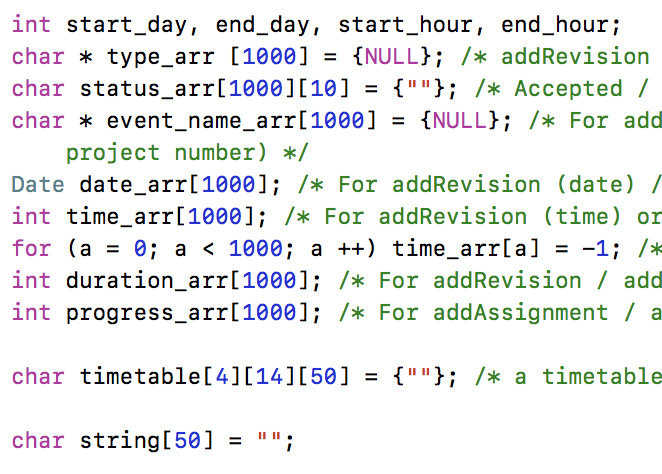
Scheduling module:

Input: a string concatenating a user input and a character for indicating validity (for addPeriod, addAssignment, addProject, addRevision, addActivity) / a string concatenating a input (for runS3)

Output: scheduled timetable stored in a multidimensional array, status array, progress array, array of requests and user input

Scheduling module first read the user inputs in the form of string in the pipe and turn them into the following data structure for scheduling.

Data structure of the Scheduling module:



int start\_day: the start day specified in the addPeriod instruction.

int end\_day: the end day specified in addPeriod instruction.

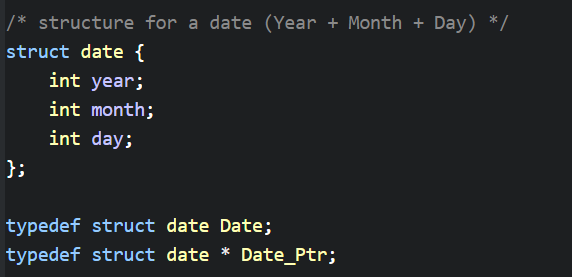
int start\_hour: the start hour specified in the addPeriod instruction.

int end\_hour: the end hour specified in the addPeriod instruction.

char \* type\_arr [1000]: the type of instruction i.e. addRevision / addActivity / addAssignment / addProject.

char status\_arr[1000][10]: It can be “Accepted”, “Rejected”, “Invalid” or “”, “Accepted” means the event can be scheduled successfully, “Rejected” means the event cannot be scheduled, however, if the user addPeriod again, the event is possible to become “Accepted”. “Invalid” means the events can never be scheduled forever because the program cannot handle an event that is out of the assumed maximum period, e.g. out of the period of 2019-04-08 and 2019-04-21, duration of the event smaller than or equal to 0 etc… The status of the invalid request will be set to “Invalid” if the indicator in the string piped from input module is 0 (indicate invalid request).

char \* event\_name\_arr[1000]: For addRevision (subject code) / addActivity (name of the event) / addAssignment (subject code with assignment number) / addProject (subject code with project number)

Date date\_arr[1000]: For addRevision (date) / addActivity (date) / addAssignment (due date) / addProject (due date)

Date

int time\_arr[1000]: the starting time for addRevision or addActivity, -1 means not applicable i.e. addAssignment, addProject.

int duration\_arr[1000]: the duration of the event.

int progress\_arr[1000]: the percentage of completion of the event.

char timetable[4][14][50]: the final timetable after the events are scheduled

char string[50]: a buffer for Scheduling module to store the parameters in the string sent from Input module temporarily before passing to the data structures.

There are two algorithms used in the scheduling module:

1. First-come-first-serve(FCFS) and 2. Earliest deadline first(EDF).

The scheduling module will call FCFS or EDF for scheduling the events according to the user input. Then, the functions will update the data stored, such as the status and progress of each event. Lastly, the updated data will be piped to output & analyzer module for printing and data analysis when the output & analyzer module is ready to read.

Output & Analyzer module:

Input: scheduled timetable stored in a multidimensional array, status array, progress array, array of requests and user input.

Output: log file, summary report with timetable.

The name of the Algorithm is read from the input which is piped by scheduling module.

It stored in char a\_name[5]. The number of the request also is piped by scheduling module and stored in int id. To print out a scheduled timetable. firstly, a timetable with 2019-04-08 to 2019-04-21 19:00 to 22:00 will be printed out. Then the name of activities will show on the correct time slot. If the user did not add the period, that time slot will be an "X" signal. Otherwise, if that period is added, but no activities are assigned, there will be an “N/A” signal. The summary report contains the number of requests, what algorithm the user use and the status of them. the status of requests are stored in char status\_arr[1000]. And int acc and int rej represent the number of requests accepted and the number of requests rejected respectively. Finally, the log file contains the information of the request, for example, the ID of the request and the status. For “Invalid” request, the status of it will also be printed as “Rejected”. The input of the user will be fully stored in the event column of the log file.

5. Testing Cases

In our testing cases, we mainly focus on the input with logic errors since the input is assumed to be syntax-error-free.

Instructions for testing:

1. addPeriod [start date] [end date] [start time] [end time]

For addPeriod, we tested the program with valid cases that all four parameters are within maximum range, which is from 2019-04-08 19:00 to 2019-04-21 23:00 and the program indicated those periods as available and the whole instruction is accepted.

Also, we have tried to test the program with error values that any of [start date] [end date] [start time] or [end time] greater than the maximum period. Such kind of instructions are rejected by the system and are recorded in the error log file.

Valid examples:

addPeriod 2019-04-12 2019-04-13 19:00 20:00

addPeriod 2019-04-08 2019-04-21 19:00 23:00

Invalid examples:

addPeriod 2019-04-06 2019-04-07 19:00 20:00

addPeriod 2019-04-08 2019-04-09 13:00 15:00

addPeriod 2019-04-22 2019-04-23 20:00 21:00

addPeriod 2019-04-07 2019-04-22 19:00 23:00

1. addAssignment [subject code with assignment number] [due date] [duration]

For addAssignment, we assume that the assignment is still required to be completed even the [due date] is past and it will be regarded as late submission. We tested the system with valid values first, such that [due date] is within the [start date] and [end date] specified in the “addPeriod” instruction, and the [duration] is greater than 0. The system accepted such instruction and schedule the assignment properly.

Then, we tested it with invalid values such that [due date] is beyond the [start date] and [end date] specified in the “addPeriod” instruction, or the [duration] is smaller than or equal to 0.

Valid examples:

addAssignment COMP2322A1 2019-04-08 1

addAssignment COMP2432A2 2019-04-21 20

Invalid examples:

addAssignment COMP2322A1 2019-04-07 12

addAssignment COMP2422A2 2019-04-08 0

1. addProject [subject code with project number] [due date] [duration]

For addProject, we assume that the project is still required to be completed even the [due date] is past and it will be regarded as late submissions. We tested the system with valid values first, such that due date is within the [start date] and [end date] specified in the “addPeriod” instruction, and the duration is greater than 0. The system accepted such instruction and schedule the project properly.

Then, we tested it with invalid values such that [due date] is beyond the [start date] and [end date] specified in the “addPeriod” instruction, or the [duration] is smaller than or equal to 0.

Valid examples:

addProject COMP2322P1 2019-04-08 1

addProject COMP2432P2 2019-04-21 20

Invalid examples:

addProject COMP2322P1 2019-04-07 12

addProject COMP2422P2 2019-04-08 0

1. addRevision [subject code] [date and time] [duration]

For addRevision, we tested the system with valid values first, such that due [date and time] is within the [start date], [end date], [start time] and [end time] specified in the “addPeriod” instruction, and the duration is greater than 0. The system accepted such instruction and schedule the revision properly.

Then, we tested it with invalid values such that [date and time] is beyond the [start date], [end date], [start time] and [end time] specified in the “addPeriod” instruction, or the [duration] is smaller than or equal to 0.

Valid examples:

addRevision COMP2000 2019-04-08 2

addRevision COMP2432 2019-04-21 1

Invalid examples:

addRevision COMP2000 2019-04-07 2

addRevision COMP2432 2019-04-15 0

addRevision COMP2000 2019-04-12 -1

1. addActivity [name of the event] [date and time of the event] [duration]

For addActivity, we tested the system with valid values first, such that due [date and time of the event] is within the [start date], [end date], [start time] and [end time] specified in the “addPeriod” instruction, and the duration is greater than 0. The system accepted such instruction and schedule the activity properly.

Then, we tested it with invalid values such that [date and time of the event] is beyond the [start date], [end date], [start time] and [end time] specified in the “addPeriod” instruction, or the [duration] is smaller than or equal to 0.

Valid examples:

addActivity Meeting 2019-04-08 2

addActivity Exercise 2019-04-21 1

Invalid examples:

addActivity Meeting 2019-04-07 2

addActivity Exercise 2019-04-15 0

addActivity Competition 2019-04-12 -1

Also, we have tested that our system can handle multiple “addPeriod” requests from the user and the timeslots are added cumulatively.

Consider the following instruction set as an example:

addPeriod 2019-04-10 2019-04-13 19:00 21:00

addRevision COMP2000 2019-04-15 19:00 2

addPeriod 2019-04-08 2019-04-21 19:00 23:00

runS3 FCFS S3\_report\_fcfs\_01.dat

In this example, the second instruction “addRevision 2019-04-15 12” is originally rejected by the system since the data and time specified is out of the range specified in the first “addPeriod” instruction. When the user input the third instruction, “addPeriod 2019-04-08 2019-04-21 19:00 23:00”, since some timeslots are not available in instruction 1, the system will automatically make the missing timeslots in instruction 1 to become available and hence the second instruction “addRevision 2019-04-15 12” is range of available period now. If the timeslots specified in second and later instructions has been available, the timeslots will remain unchanged.

6. Performance Analysis

First Come First Serve (FCFS) and Earliest Deadline First (EDF) are the scheduling algorithms that have been implemented in the scheduling module.

The advantage of FCFS is that it could be implemented in an easy way. But, the disadvantage is that FCFS may have a convoy effect. The convoy effect means when the processes arrive almost at the same time, the latest arrived process need to wait for a long period of time. It is because the previous processes may take a long period of time to execute. This situation is quite similar to a traffic jam or queue up with a lot of people.

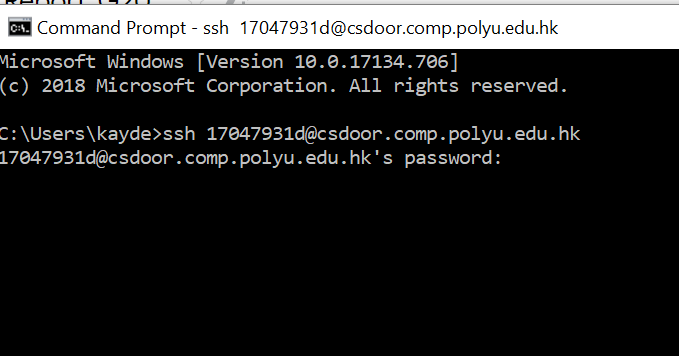
In this study scheduling system, the drawback of implementing FCFS is that some of the tasks may miss the deadlines. Since FCFS have to handle the tasks which come first, other tasks may not have enough time to finish. Students could only submit a partially completed assignment or project. Regarding the usage of the time slot, FCFS could fill up most of the time slot in the best scenario.

The advantage of EDF is that it considers the deadlines of each task first. Assuming the usage of the time slot is below 100%, EDF could guarantee no task misses the deadline. In the system of this project, EDF will first assign the slots to projects and assignments, since revisions and activities’ start time has not arrived, although they may have earlier deadlines. EDF will utilize the empty time slots and assign them to more urgent jobs. When an activity wants to use a timeslot, EDF will check if the current project or assignment deadline can be fulfilled after assigning the time slots to the activity. The EDF scheduling behavior ensures jobs with higher priority, its performance will not be affected by jobs with lower priority. However, the weakness of this scheduling method is that whenever there are 2 high prioritized jobs with a same or similar deadline, one of them or the one with later deadline may have very low completeness as the time slots are assigned to another job.

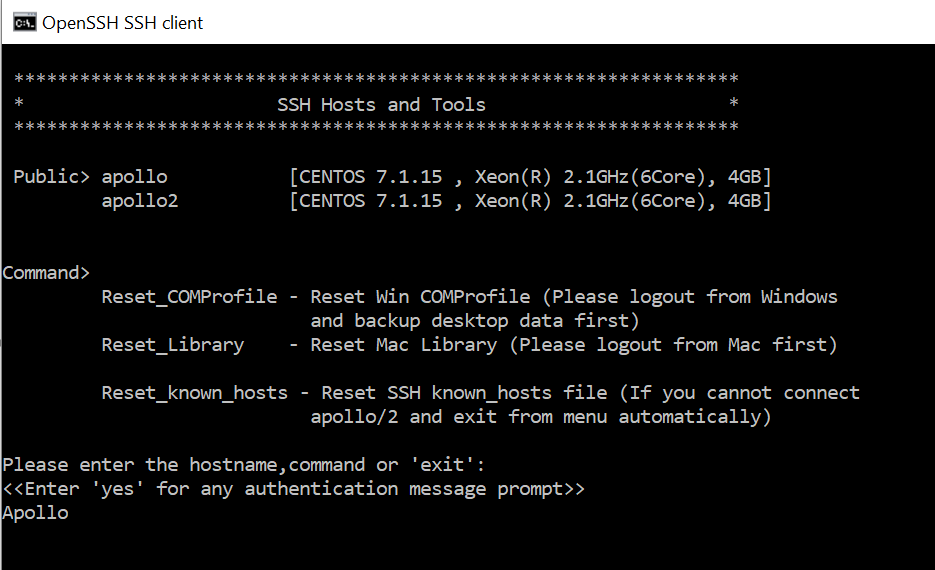
7. Program Set Up and Execution

This S3 system should be compiled and executed under the “Apollo” Linux server of Computing department. The compiler we are using is "gcc".

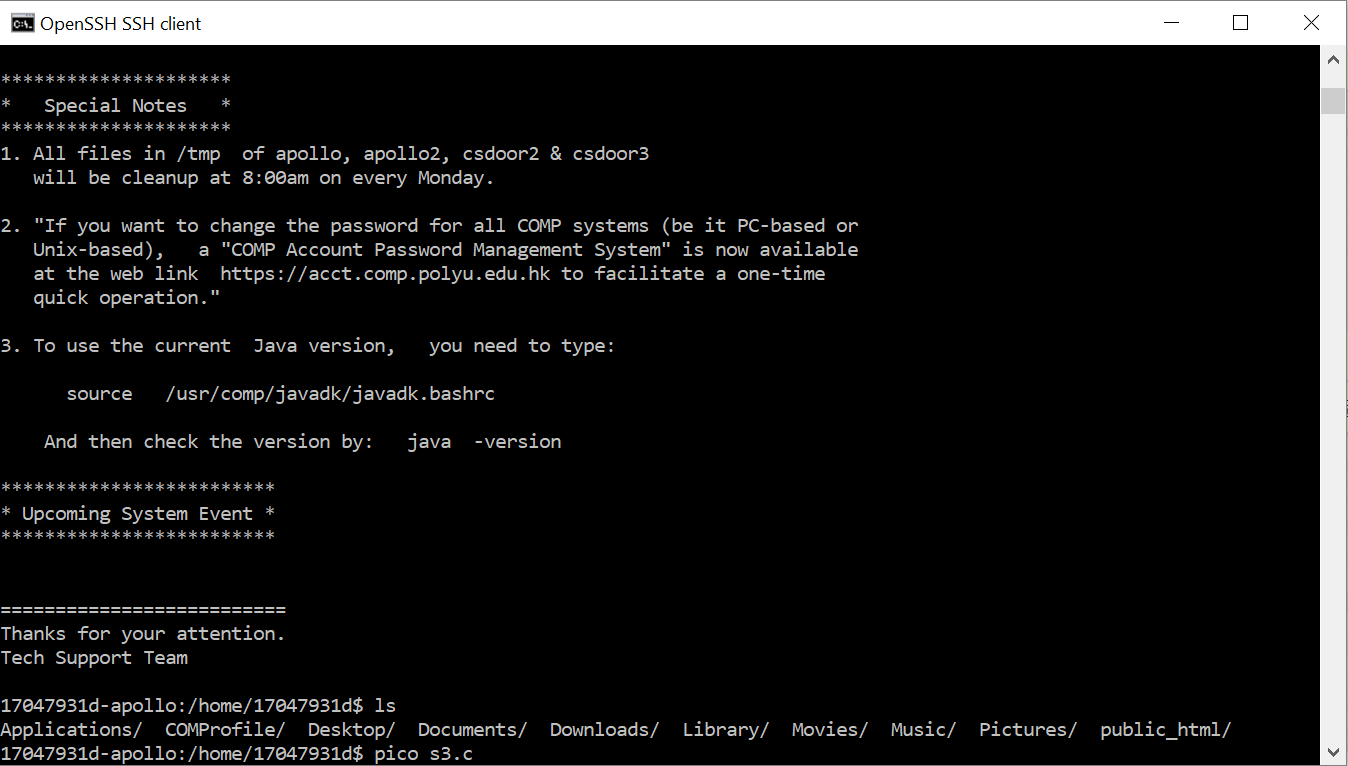
First, open “cmd” and connect to “Apollo” using ssh client by typing “ssh <Your studentID>@csdoor.comp.polyu.edu.hk”.



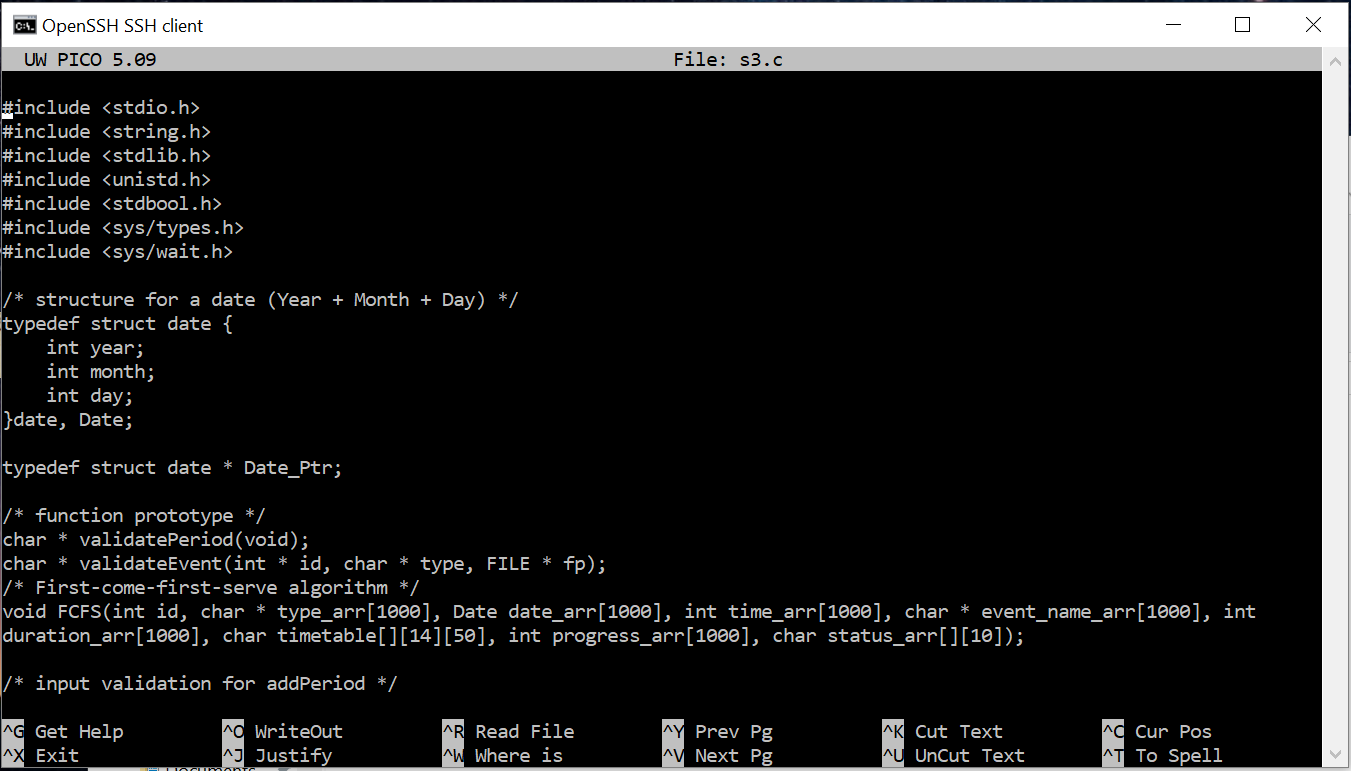
After logging in by typing the password, type “Apollo” to connect to “Apollo” Linux server.



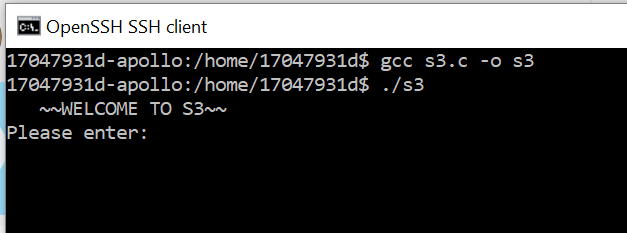
Then, type “pico s3.c” to create a C source file.



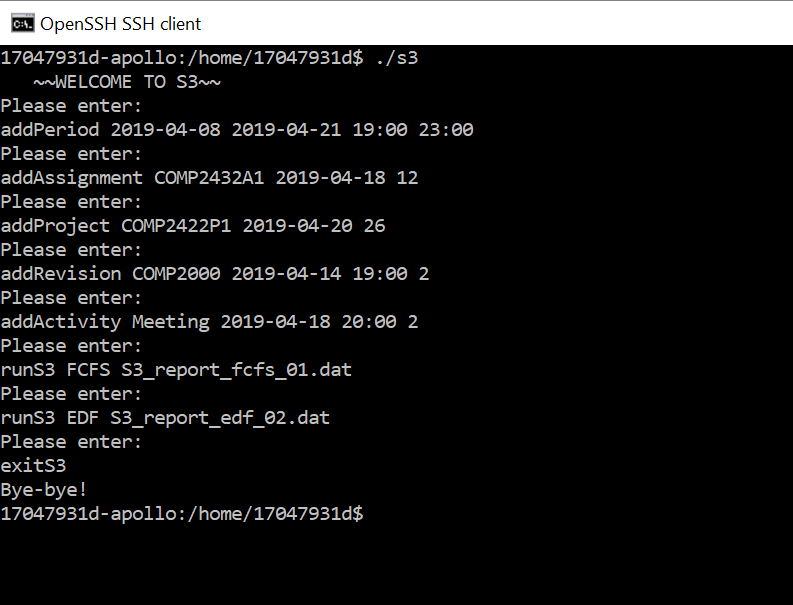
Copy the source code of s3 system into s3.c and exit.



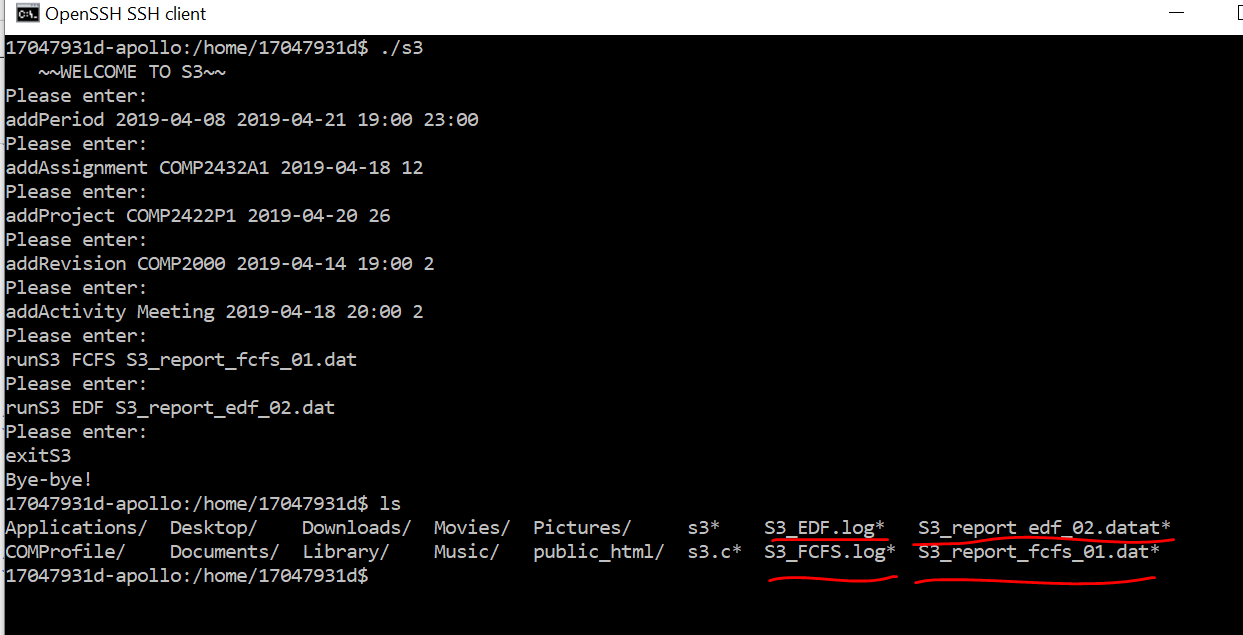
compile s3.c by typing “gcc s3.c -o s3”. Run s3 by typing “./s3”



Then, start to type in instructions in the system as follow:

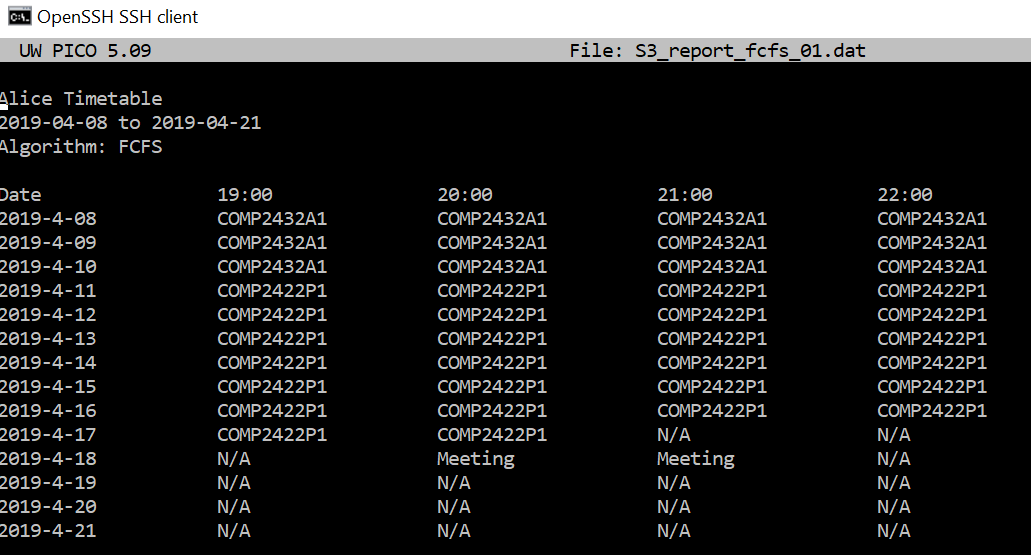


You can find the output timetable, analysis report and log files (.dat & .log) generated by the system by typing “ls”.

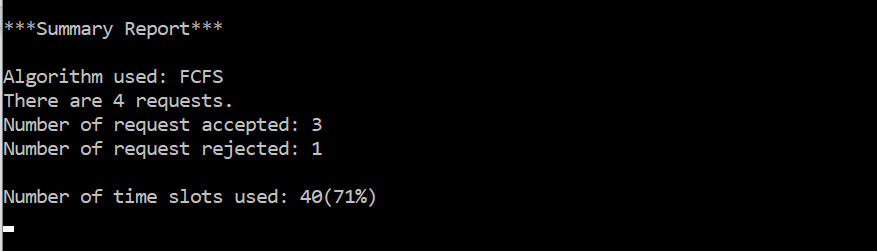


*Appendix I*

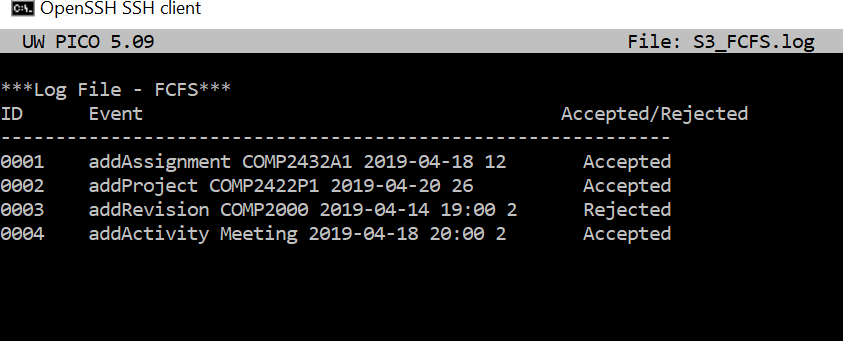
**Timetable(FCFS)**

****

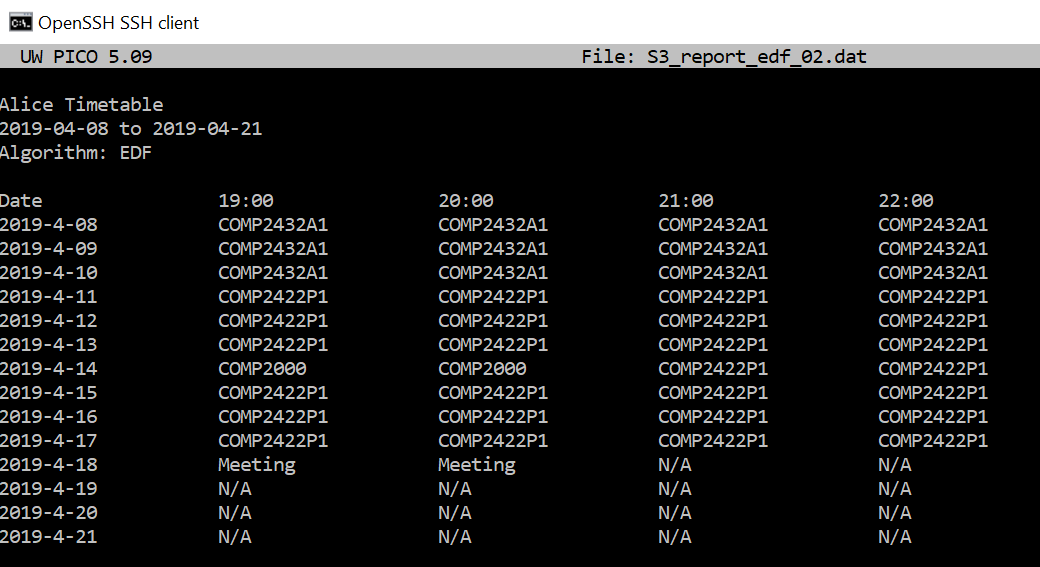
**Report(FCFS)**

****

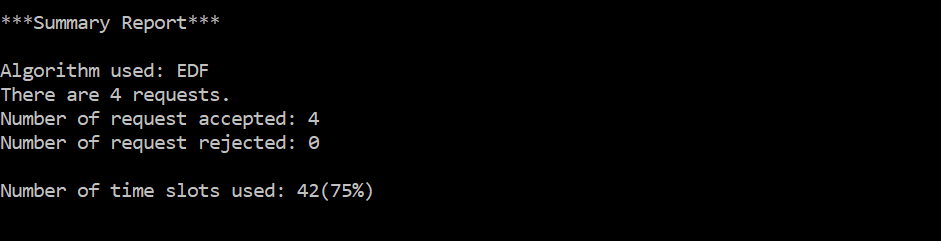
**Log file(FCFS)**

****

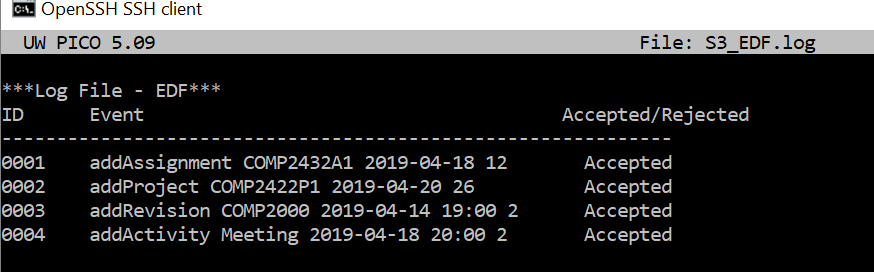
**Timetable(EDF)**

****

**Report(EDF)**

****

**Log file(EDF)**

****

**Source code:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <stdbool.h>

#include <sys/types.h>

#include <sys/wait.h>

/\* structure for a date (Year + Month + Day) \*/

typedef struct date {

int year;

int month;

int day;

}date, Date;

typedef struct date \* Date\_Ptr;

/\* function prototype \*/

char \* validatePeriod(void);

char \* validateEvent(int \* id, char \* type, FILE \* fp);

/\* First-come-first-serve algorithm \*/

void FCFS(int id, char \* type\_arr[1000], Date date\_arr[1000], int time\_arr[1000], char \* event\_name\_arr[1000], int duration\_arr[1000], char timetable[][14][50], int progress\_arr[1000], char status\_arr[][10]);

/\* input validation for addPeriod \*/

char \* validatePeriod(void) {

char valid[2] = "1"; /\* a character to indicate whether the input is valid or not (0: invalid, 1: valid) \*/

Date start\_date;

Date end\_date;

int start\_time\_hour;

int start\_time\_minute;

int end\_time\_hour;

int end\_time\_minute;

char parameter [20] = "";

char string[50] = "addPeriod"; /\* initialize the string to be copied to return\_string \*/

char \* return\_string; /\* string to be returned \*/

/\* reading and checking start date \*/

scanf("%s", parameter);

strcat(string, " ");

strcat(string, parameter); /\* concatenate the start date to the string \*/

start\_date.year = atoi(strtok(parameter, "-"));

/\* invalid if the starting year of period to be added is out of assumed range (2019) \*/

if (strcmp(valid, "1") == 0 && start\_date.year != 2019) {

printf("Invalid starting year added. (Valid year: 2019)\n");

strcpy(valid, "0");

}

start\_date.month = atoi(strtok(NULL, "-"));

/\* invalid if the starting month of period to be added is out of assumed range (4) \*/

if (strcmp(valid, "1") == 0 && start\_date.month != 4) {

printf("Invalid starting month added (Valid month: 4)\n");

strcpy(valid, "0");

}

start\_date.day = atoi(strtok(NULL, "-"));

/\* invalid if the starting day of period to be added is out of assumed range (8 - 21) \*/

if (strcmp(valid, "1") == 0 && (start\_date.day < 8 || start\_date.day > 21)) {

printf("Invalid starting day added (Valid day: 8 - 21)\n");

strcpy(valid, "0");

}

/\* reading and checking end date \*/

scanf("%s", parameter);

strcat(string, " ");

strcat(string, parameter); /\* concatenate the end date to the string \*/

end\_date.year = atoi(strtok(parameter, "-"));

/\* invalid if the ending year of period to be added is out of assumed range (2019) \*/

if (strcmp(valid, "1") == 0 && end\_date.year != 2019) {

printf("Invalid ending year added. (Valid year: 2019)\n");

strcpy(valid, "0");

}

end\_date.month = atoi(strtok(NULL, "-"));

/\* invalid if the ending month of period to be added is out of assumed range (4) \*/

if (strcmp(valid, "1") == 0 && end\_date.month != 4) {

printf("Invalid ending month added (Valid month: 4)\n");

strcpy(valid, "0");

}

end\_date.day = atoi(strtok(NULL, "-"));

/\* invalid if the ending day of period to be added is out of assumed range (start\_date.day - 21) \*/

if (strcmp(valid, "1") == 0 && (end\_date.day < start\_date.day || start\_date.day > 21)) {

printf("Invalid ending day added (Valid day: %d - 21)\n", start\_date.day);

strcpy(valid, "0");

}

/\* reading and checking start time \*/

scanf("%s", parameter);

strcat(string, " ");

strcat(string, parameter); /\* concatenate the start time to the string \*/

start\_time\_hour = atoi(strtok(parameter, ":"));

/\* invalid if the starting hour of period to be added is out of assumed range (19 - 22) \*/

if (strcmp(valid, "1") == 0 && (start\_time\_hour < 19 || start\_time\_hour > 22)) {

printf("Invalid starting hour added (Valid hour: 19 - 22)\n");

strcpy(valid, "0");

}

start\_time\_minute = atoi(strtok(NULL, ":"));

/\* invalid if the starting minute of period to be added is out of assumed range (0) \*/

if (strcmp(valid, "1") == 0 && start\_time\_minute != 0) {

printf("Invalid starting minute added (Valid minute: 00)\n");

strcpy(valid, "0");

}

/\* reading and checking end time \*/

scanf("%s", parameter);

strcat(string, " ");

strcat(string, parameter); /\* concatenate the end time to the string \*/

end\_time\_hour = atoi(strtok(parameter, ":"));

/\* invalid if the endinging hour of period to be added is out of assumed range (start\_time\_hour - 23) \*/

if (strcmp(valid, "1") == 0 && (end\_time\_hour <= start\_time\_hour || end\_time\_hour > 23)) {

printf("Invalid ending hour added (Valid hour: %d - 23)\n", start\_time\_hour + 1);

strcpy(valid, "0");

}

end\_time\_minute = atoi(strtok(NULL, ":"));

/\* invalid if the ending minute of period to be added is out of assumed range (0) \*/

if (strcmp(valid, "1") == 0 && end\_time\_minute != 0) {

printf("Invalid ending minute added (Valid minute: 00)\n");

strcpy(valid, "0");

}

strcat(string, " ");

strcat(string, valid); /\* concatenate the indicator (0 / 1) to the string to indicate whether the input is valid or not \*/

/\* return the string as an input to scheduler \*/

/\* format: [use input] [indicator] \*/

return\_string = malloc((strlen(string) + 1) \* sizeof(char));

strcpy(return\_string, string);

return return\_string;

}

/\* input validation for addRevision & addActivity & addAssignment & addProject \*/

char \* validateEvent(int \* id, char \* type, FILE \* fp) {

char valid[2] = "1"; /\* a character to indicate whether the input is valid or not (0: invalid, 1: valid) \*/

Date date;

int time\_hour;

int time\_minute;

int duration;

char parameter [50] = "";

char string[50] = "";

strcpy(string, type); /\* initialize the string to be copied to return\_string \*/

char \* return\_string; /\* string to be returned \*/

/\* reading (subject code with assignment or project number for addAssignment and addProject respectively) / (subject code or name of event for addRevision and addActivity

respectively) \*/

fscanf(fp, "%s", parameter);

strcat(string, " ");

strcat(string, parameter); /\* concatenate the event name to the string \*/

/\* reading and checking (due date for addAssignment & addProject) / (date for addRevision & addActivity) \*/

fscanf(fp, "%s", parameter);

strcat(string, " ");

strcat(string, parameter); /\* concatenate the date to the string \*/

date.year = atoi(strtok(parameter, "-"));

/\* invalid if the year is out of assumed range (2019) \*/

if (date.year != 2019) {

printf("ID: %d, Invalid year. (Valid year: 2019)\n", \*id + 1);

strcpy(valid, "0");

}

date.month = atoi(strtok(NULL, "-"));

/\* invalid if the month is out of assumed range (4) \*/

if (strcmp(valid, "1") == 0 && date.month != 4) {

printf("ID: %d, Invalid month. (Valid month: 4)\n", \*id + 1);

strcpy(valid, "0");

}

date.day = atoi(strtok(NULL, "-"));

/\* invalid if the day is out of assumed range (8 - 21) \*/

if (strcmp(valid, "1") == 0 && (date.day < 8 || date.day > 21)) {

printf("ID: %d, Invalid day. (Valid day: 8 - 21)\n", \*id + 1);

strcpy(valid, "0");

}

/\* reading and checking time for addRevision & addActivity \*/

if (strcmp(type, "addRevision") == 0 || strcmp(type, "addActivity") == 0) {

fscanf(fp, "%s", parameter);

strcat(string, " ");

strcat(string, parameter); /\* concatenate the time to the string \*/

time\_hour = atoi(strtok(parameter, ":"));

/\* invalid if the hour is out of assumed range (19 - 22) \*/

if (strcmp(valid, "1") == 0 && (time\_hour < 19 || time\_hour > 22)) {

printf("ID: %d, Invalid hour (Valid hour: 19 - 22)\n", \*id + 1);

strcpy(valid, "0");

}

time\_minute = atoi(strtok(NULL, ":"));

/\* invalid if the minute is out of assumed range (0) \*/

if (strcmp(valid, "1") == 0 && time\_minute != 0) {

printf("ID: %d, Invalid minute (Valid minute: 00)\n", \*id + 1);

strcpy(valid, "0");

}

}

/\* reading and checking duration \*/

fscanf(fp, "%s", parameter);

strcat(string, " ");

strcat(string, parameter); /\* concatenate the duration to the string \*/

duration = atoi(parameter);

/\* print error message if the duration is invalid \*/

if (strcmp(valid, "1") == 0 && duration <= 0) {

printf("ID: %d, Invalid duration. (Valid duration: > 0)\n", \*id + 1);

strcpy(valid, "0");

}

(\*id) ++; /\* increment the id \*/

strcat(string, " ");

strcat(string, valid); /\* concatenate the indicator (0 / 1) to the string to indicate whether the input is valid or not \*/

/\* return the string as an input to scheduler \*/

/\* format: [use input] [indicator] \*/

return\_string = malloc((strlen(string) + 1) \* sizeof(char));

strcpy(return\_string, string);

return return\_string;

}

/\* First-come-first-serve algorithm \*/

void FCFS(int id, char \* type\_arr[1000] ,Date date\_arr[1000], int time\_arr[1000], char \* event\_name\_arr[1000], int duration\_arr[1000], char timetable[][14][50], int progress\_arr[1000], char status\_arr[][10]){

int count=0; /\* duration of each task \*/

int i,x,y; /\* for loop counters \*/

for (i=0;i<id;i++){

if (strcmp(status\_arr[i], "") == 0) {

/\* Reject the request if the date and time is out of the period range \*/

if (strcmp(type\_arr[i],"addRevision")==0 || strcmp(type\_arr[i],"addActivity")==0){

if (strcmp(timetable[time\_arr[i] - 19][date\_arr[i].day - 8], "N/A") !=0) {

strcpy(status\_arr[i],"Rejected");

progress\_arr[i] = 0;

}

}

if (status\_arr[i] == NULL || strcmp(status\_arr[i],"Rejected") != 0) {

count=duration\_arr[i];

for (x=0;x<14;x++){

for (y=0;y<4;y++){

if (strcmp(timetable[y][x],"N/A")==0 && count>0){ /\* if the timeslot is available \*/

if(strcmp(type\_arr[i],"addRevision")!=0 && strcmp(type\_arr[i],"addActivity")!=0){

strcpy(timetable[y][x],event\_name\_arr[i]); /\* assign task for the timeslot \*/

count--; /\* duration of task by reduced by 1 after scheduled for a timeslot \*/

if (count==0){

progress\_arr[i]=100; /\* update progress record in main scheduler \*/

strcpy(status\_arr[i],"Accepted"); /\* update status record in main scheduler \*/

}

}

else if(x==(date\_arr[i].day-8) && y>=(time\_arr[i]-19) && y<=(time\_arr[i]-19+duration\_arr[i])){

strcpy(timetable[y][x],event\_name\_arr[i]); /\* assign task for the timeslot \*/

count--; /\* duration of task by reduced by 1 after scheduled for a timeslot \*/

if (count==0){

progress\_arr[i]=100; /\* update progress record in main scheduler \*/

strcpy(status\_arr[i],"Accepted"); /\* update status record in main scheduler \*/

}

}

}

}

}

if (count>0){ /\* if the task is not fully scheduled\*/

if (count<duration\_arr[i]) {

if (strcmp(type\_arr[i],"addRevision")!=0 && strcmp(type\_arr[i],"addActivity")!=0){

float prog = (float)(duration\_arr[i]-count) / (float)duration\_arr[i] \*100; /\*calculate percentage of completion \*/

progress\_arr[i]=(int)prog; /\* update progress record in main scheduler \*/

strcpy(status\_arr[i],"Accepted"); /\* update status record in main scheduler \*/

}

else {

progress\_arr[i]=0; /\* Progress = 0 if task cannot be scheduled \*/

strcpy(status\_arr[i],"Rejected"); /\* update status to "Rejected" in main scheduler if the task cannot be scheduled \*/

}

}

else {

progress\_arr[i]=0; /\* Progress = 0 if task cannot be scheduled \*/

strcpy(status\_arr[i],"Rejected"); /\* update status to "Rejected" in main scheduler if the task cannot be scheduled \*/

}

}

}

}

}

}

int main(int argc, const char \* argv[]) {

int a, b, temp;

int id = 0;

int parent\_id = getpid();

int pid;

int fd[4][2]; /\* (2 \* 2) pipes for communication between 2 pairs of modules (1.input module & scheduling module 2. scheduling module & output&analyzer module) \*/

char command [20] = "";

char buffer[50] = "";

char syn[10] = ""; /\* only used for synchronization \*/

/\* create pipes \*/

for (a = 0; a < 4; a ++) {

if (pipe(fd[a]) < 0) {

printf("Pipe creation error\n");

exit(1);

}

}

/\* create 2 children (Parent: input module, 1st child: scheduling module, 2st child: output&analyzer module) \*/

for(a = 0; a < 2; a ++){

if (getpid() == parent\_id) pid = fork();

}

if (pid < 0) {

printf("Fork Failed\n");

exit(1);

}

/\* child process \*/

else if (pid == 0) {

pid = getpid() - parent\_id;

/\* 1st child: schduling module \*/

if (pid == 1) {

int i\_read\_pipe = 1; /\* fd[1] is the pipe for scheduling module to read from input module \*/

int i\_write\_pipe = 0; /\* fd[0] is the pipe for scheduling module to write to input module \*/

int oa\_read\_pipe = 2; /\* fd[2] is the pipe for scheduling module to read from output&analyzer module \*/

int oa\_write\_pipe = 3; /\* fd[3] is the pipe for scheduling module to write to output&analyzer module \*/

/\* close unused pipe ends \*/

for (a = 0; a < 4; a ++) {

if (a != i\_read\_pipe && a != i\_write\_pipe && a != oa\_read\_pipe && a != oa\_write\_pipe) {

close(fd[a][0]);

close(fd[a][1]);

}

else if (a == i\_read\_pipe || a == oa\_read\_pipe) close(fd[a][1]);

else close(fd[a][0]);

}

int start\_day, end\_day, start\_hour, end\_hour;

char \* type\_arr [1000] = {NULL}; /\* addRevision / addActivity / addAssignment / addProject \*/

char status\_arr[1000][10] = {""}; /\* Accepted / Rejected \*/

char \* event\_name\_arr[1000] = {NULL}; /\* For addRevision (subject code) / addActivity (name of the event) / addAssignment (subject code with assignment number) / addProject (subject code with project number) \*/

Date date\_arr[1000]; /\* For addRevision (date) / addActivity (date) / addAssignment (due date) / addProject (due date) \*/

int time\_arr[1000]; /\* For addRevision (time) or addActivity (time) \*/

for (a = 0; a < 1000; a ++) time\_arr[a] = -1; /\* -1 means not applicable e.g. addAssignment & addProject \*/

int duration\_arr[1000]; /\* For addRevision / addActivity / addAssignment / addProject \*/

int progress\_arr[1000]; /\* For addAssignment / addProject (% of completion) \*/

char timetable[4][14][50] = {""}; /\* a timetable for events in 19:00 to 23:00 ([4]) from 2019-04-08 to 2019-04-21 ([14]) ("" means that that period is not added) \*/

char string[50] = "";

char request[1000][50] = {""}; /\* request list to be sent to output & analyzer module when "runS3" \*/

/\*EDF\*/

date edf\_start\_date;

date edf\_end\_date;

int edf\_period\_days;

int edf\_period\_day\_hrs;

int edf\_start\_hr;

int edf\_end\_hr;

/\*EDF functions\*/

Date str2date(char \*date1) {

char \*token;

char \*saveptr;

Date temp;

temp.year = atoi(strtok\_r(date1, "-", &saveptr));

temp.month = atoi(strtok\_r(NULL, "-", &saveptr));

temp.day = atoi(strtok\_r(NULL, "-", &saveptr));

return temp;

}

void date2str(Date date1,char \*str) {

char month[3];

char day[3];

if(date1.month<10){

sprintf(month, "0%d", date1.month);

}

else{

sprintf(month, "%d", date1.month);

}

if(date1.day<10){

sprintf(day, "0%d", date1.day);

}

else{

sprintf(day, "%d", date1.day);

}

sprintf(str, "%d-%s-%s", date1.year, month, day);

}

int str2hr(char\* time1){

char \*token;

char \*saveptr;

int temp = -1;

temp = atoi(strtok\_r(time1, ":", &saveptr));

return temp;

}

Date date\_tmr(Date date1){

int max\_days\_arr[12] = {31,28,31,30,31,30,31,31,30,31,30,31};

Date temp = date1;

if(date1.day != max\_days\_arr[date1.month-1]){

temp.day +=1;

return temp;

}

else{

if(date1.month != 12){

temp.month+=1;

temp.day=1;

return temp;

}

else{

temp.year+=1;

temp.month=1;

temp.day=1;

return temp;

}

}

}

int date\_cmp(Date date1, Date date2){

if(date1.year > date2.year){

return 1;

}

else if(date1.year < date2.year){

return -1;

}

else{

if(date1.month > date2.month){

return 1;

}

else if(date1.month < date2.month){

return -1;

}

else{

if(date1.day > date2.day){

return 1;

}

else if(date1.day < date2.day){

return -1;

}

else{

return 0;

}

}

}

}

bool date\_inrange(Date date1, Date date2, Date date3){

if(date\_cmp(date2, date1)>=0 && date\_cmp(date2, date3)<=0){

return true;

}

else{

return false;

}

}

int date\_days(Date date1, Date date2){

int i = 0;

int j =0;

int days = 0;

Date temp1;

Date temp2;

if(date\_cmp(date1, date2)==1){

temp2 = date1;

temp1 = date2;

}

else if(date\_cmp(date1, date2)==-1){

temp1 = date1;

temp2 = date2;

}

else{

return 1;

}

if(temp1.year==2019 || temp2.year==2019 ){

if(temp1.month!=temp2.month){

for(i=temp1.month+1;i<temp2.month;i++){

if(i<=7){

if(i!=2){

if(i%2==1){

days+=31;

}

else{

days+=30;

}

}

else{

days+=28;

}

}

else{

if(i%2==1){

days+=30;

}

else{

days+=31;

}

}

}

i = temp1.month;

if(i<=7){

if(i!=2){

if(i%2==1){

days+=31-temp1.day+1;

}

else{

days+=30-temp1.day+1;

}

}

else{

days+=28-temp1.day+1;

}

}

else{

if(i%2==1){

days+=30-temp1.day+1;

}

else{

days+=31-temp1.day+1;

}

}

days += temp2.day;

}

else{

days += temp2.day-temp1.day+1;

}

}

else{

return -1;

}

return days;

}

/\* Event entry to be scheduled \*/

typedef struct edf\_entry {

int id; /\* ID of the event name \*/

char \*name; /\* Name of the activity \*/

Date edf\_date; /\* Date of deadline or activity \*/

int edf\_time; /\* Time start hr:0-24 \*/

int edf\_cost; /\* Time cost for the activity in hrs \*/

int type\_piority; /\* Indicate the type of activity from 1 to 4, e.g. Project = 1 \*/

} edf\_entry;

/\* List of event entries \*/

typedef struct edf\_entry\_list {

edf\_entry entry;

struct edf\_entry\_list \* next\_entry;

} edf\_entry\_list;

edf\_entry\_list\* entry\_head = NULL;

edf\_entry\_list\* entry\_tail = NULL;

/\* Time slot element for timetable list \*/

typedef struct edf\_timeslot {

char \*name;

Date edf\_date;

int edf\_time;

} edf\_timeslot;

/\* A list of time slots to represent the timetable (slots by slots) \*/

typedef struct edf\_timetable\_list{

edf\_timeslot timeslot;

struct edf\_timetable\_list \* next\_slot;

} edf\_timetable\_list;

edf\_timetable\_list\* timetable\_head = NULL;

edf\_timetable\_list\* timetable\_tail = NULL;

int insert\_entry(int id, char\* insrt\_name, Date insrt\_date, int insrt\_time, int insrt\_cost, int insrt\_type){

edf\_entry\_list\* temp;

temp = malloc(sizeof(edf\_entry\_list));

temp->entry.id = id;

temp->entry.name = insrt\_name;

temp->entry.edf\_date = insrt\_date;

temp->entry.edf\_time = insrt\_time;

temp->entry.edf\_cost = insrt\_cost;

temp->entry.type\_piority = insrt\_type;

temp->next\_entry = NULL;

edf\_entry\_list\* entry\_list\_ptr;

if(entry\_head==NULL){

entry\_head = temp;

entry\_tail = temp;

}

else{

edf\_entry\_list\* entry\_list\_prev = NULL;

entry\_list\_ptr = entry\_head;

while(true){

/\*Date of new entry is later than current\*/ /\*checked\*/

if(date\_cmp(entry\_list\_ptr->entry.edf\_date, temp->entry.edf\_date) == -1){

if(entry\_list\_ptr->next\_entry!=NULL){

entry\_list\_prev = entry\_list\_ptr;

entry\_list\_ptr = entry\_list\_ptr->next\_entry;

}

else{

entry\_list\_ptr->next\_entry = temp;

entry\_tail = temp;

return 1;

}

}

/\*Date of new entry is eariler than current\*/ /\*checked\*/

else if(date\_cmp(entry\_list\_ptr->entry.edf\_date, temp->entry.edf\_date) == 1){

if(entry\_list\_prev != NULL){

entry\_list\_prev->next\_entry = temp;

temp->next\_entry = entry\_list\_ptr;

}

else{

temp->next\_entry = entry\_head;

entry\_head = temp;

}

return 1;

}

/\*Date of new entry is identical\*/

else{

/\*Date of new entry is later than current time\*/ /\*checked\*/

if(temp->entry.edf\_time > entry\_list\_ptr->entry.edf\_time){

if(entry\_list\_ptr->next\_entry!=NULL){

entry\_list\_prev = entry\_list\_ptr;

entry\_list\_ptr = entry\_list\_ptr->next\_entry;

}

else{

entry\_list\_ptr->next\_entry = temp;

entry\_tail = temp;

return 1;

}

}

/\*Date of new entry is earlier than current time\*/

else if(temp->entry.edf\_time < entry\_list\_ptr->entry.edf\_time){

if(entry\_list\_prev != NULL){

entry\_list\_prev->next\_entry = temp;

temp->next\_entry = entry\_list\_ptr;

}

else{

temp->next\_entry = entry\_head;

entry\_head = temp;

}

return 1;

}

/\*Date of new entry is same to current time\*/

else{

/\*New entry has higher priority\*/ /\*checked\*/

if(entry\_list\_ptr->entry.type\_piority < temp->entry.type\_piority){

if(entry\_list\_prev != NULL){

entry\_list\_prev->next\_entry = temp;

temp->next\_entry = entry\_list\_ptr;

}

else{

temp->next\_entry = entry\_head;

entry\_head = temp;

}

return 1;

}

/\*New entry has lower priority\*/ /\*checked\*/

else if(entry\_list\_ptr->entry.type\_piority > temp->entry.type\_piority){

if(entry\_list\_ptr->next\_entry!=NULL){

entry\_list\_prev = entry\_list\_ptr;

entry\_list\_ptr = entry\_list\_ptr->next\_entry;

}

else{

entry\_list\_ptr->next\_entry = temp;

entry\_tail = temp;

return 1;

}

}

/\*New entry has same priority\*/

else{

if(entry\_list\_ptr->next\_entry!=NULL){

entry\_list\_prev = entry\_list\_ptr;

entry\_list\_ptr = entry\_list\_ptr->next\_entry;

}

else{

entry\_list\_ptr->next\_entry = temp;

entry\_tail = temp;

return 1;

}

}

}

}

}

}

}

void print\_entries(){

char temp\_date\_str[20];

edf\_entry\_list\* temp\_ptr = NULL;

if(entry\_head != NULL){

temp\_ptr = entry\_head;

date2str(temp\_ptr->entry.edf\_date, temp\_date\_str);

printf("ID Entry name Deadline/Date Time Cost Priority\n");

printf("==================================================\n");

printf("%3d %-13s%-16s%-7d%-7d%-8d\n", temp\_ptr->entry.id, temp\_ptr->entry.name, temp\_date\_str, temp\_ptr->entry.edf\_time,temp\_ptr->entry.edf\_cost,temp\_ptr->entry.type\_piority);

}

else{return;}

while(temp\_ptr->next\_entry!=NULL){

temp\_ptr = temp\_ptr->next\_entry;

date2str(temp\_ptr->entry.edf\_date, temp\_date\_str);

printf("%3d %-13s%-16s%-7d%-7d%-8d\n", temp\_ptr->entry.id, temp\_ptr->entry.name, temp\_date\_str, temp\_ptr->entry.edf\_time,temp\_ptr->entry.edf\_cost,temp\_ptr->entry.type\_piority);

}

printf("\n");

}

void print\_slots(){

char temp\_date\_str[20];

edf\_timetable\_list\* temp\_ptr = NULL;

if(timetable\_head != NULL){

temp\_ptr = timetable\_head;

date2str(temp\_ptr->timeslot.edf\_date, temp\_date\_str);

printf("EDF Schedule\n");

printf("==================================================\n");

printf("%-16s%-2d:00 %-17s\n", temp\_date\_str, temp\_ptr->timeslot.edf\_time,temp\_ptr->timeslot.name);

}

else{return;}

while(temp\_ptr->next\_slot!=NULL){

temp\_ptr = temp\_ptr->next\_slot;

date2str(temp\_ptr->timeslot.edf\_date, temp\_date\_str);

printf("%-16s%-2d:00 %-17s\n", temp\_date\_str, temp\_ptr->timeslot.edf\_time,temp\_ptr->timeslot.name);

}

printf("\n");

}

int add\_time\_slot(char\* insrt\_name, Date insrt\_date, int insrt\_time){

edf\_timetable\_list\* temp;

temp = malloc(sizeof(edf\_timetable\_list));

temp->timeslot.name = insrt\_name;

temp->timeslot.edf\_date = insrt\_date;

temp->timeslot.edf\_time = insrt\_time;

temp->next\_slot = NULL;

if(timetable\_head==NULL){

timetable\_head = temp;

timetable\_tail = temp;

return 1;

}

else{

timetable\_tail->next\_slot = temp;

timetable\_tail = temp;

return 1;

}

}

void addPeriod(char\* parameters[]){

int i = 0;

int j = 0;

edf\_start\_date = str2date(parameters[0]);

edf\_end\_date = str2date(parameters[1]);

edf\_start\_hr = str2hr(parameters[2]);

edf\_end\_hr = str2hr(parameters[3]);

edf\_period\_days = date\_days(edf\_start\_date, edf\_end\_date);

edf\_period\_day\_hrs = edf\_end\_hr - edf\_start\_hr;

Date temp\_date = edf\_start\_date;

int temp\_time = edf\_start\_hr;

for(i=1; i<= edf\_period\_days; i++){

for(j=1; j<=edf\_period\_day\_hrs; j++){

add\_time\_slot("N/A", temp\_date, temp\_time);

/\*char temp\_str3[20];\*/

/\*date2str((\*timetable\_tail).timeslot.edf\_date, temp\_str3);\*/

/\*printf("%s %s %d\n", (\*timetable\_tail).timeslot.name, temp\_str, (\*timetable\_tail).timeslot.edf\_time);\*/

temp\_time++;

}

temp\_time = edf\_start\_hr;

temp\_date = date\_tmr(temp\_date);

}

return;

}

void addProject(int id, char\* parameters[]){

char \*name;

name = malloc(sizeof(char)\*20);

strcpy(name,parameters[0]);

Date temp\_date = str2date(parameters[1]);

int temp\_time = 24;

int temp\_cost = atoi(parameters[2]);

int type = 1;

insert\_entry(id, name, temp\_date, temp\_time, temp\_cost, type);

}

void addAssignment(int id, char\* parameters[]){

char \*name;

name = malloc(sizeof(char)\*20);

strcpy(name,parameters[0]);

Date temp\_date = str2date(parameters[1]);

int temp\_time = 24;

int temp\_cost = atoi(parameters[2]);

int type = 2;

insert\_entry(id, name, temp\_date, temp\_time, temp\_cost, type);

}

void addRevision(int id, char\* parameters[]){

char \*name;

name = malloc(sizeof(char)\*20);

strcpy(name,parameters[0]);

Date temp\_date = str2date(parameters[1]);

int temp\_time = str2hr(parameters[2]);

int temp\_cost = atoi(parameters[3]);

int type = 3;

insert\_entry(id, name, temp\_date, temp\_time, temp\_cost, type);

}

void addActivity(int id, char\* parameters[]){

char \*name;

name = malloc(sizeof(char)\*20);

strcpy(name,parameters[0]);

Date temp\_date = str2date(parameters[1]);

int temp\_time = str2hr(parameters[2]);

int temp\_cost = atoi(parameters[3]);

int type = 4;

insert\_entry(id, name, temp\_date, temp\_time, temp\_cost, type);

}

void EDF\_RunS3(){

edf\_entry\_list\* current\_PA = NULL;

int PA\_r\_cost = 0;

edf\_entry\_list\* current\_RA = NULL;

int RA\_r\_cost = 0;

edf\_timetable\_list\* current\_slot = NULL;

int i = 1;

int total\_slots = edf\_period\_days\*edf\_period\_day\_hrs;

void getNextPA(edf\_entry\_list\*\* current){

edf\_entry\_list\* temp = \*current;

int found = 0;

while(found==0&&temp->next\_entry!=NULL){

temp = temp->next\_entry;

if(temp->entry.type\_piority<3){

found=1;

}

}

if(found==0){

\*current = NULL;

}

else{

\*current = temp;

}

}

void getNextRA(edf\_entry\_list\*\* current){

edf\_entry\_list\* temp = \*current;

int found = 0;

while(found==0&&temp->next\_entry!=NULL){

temp = temp->next\_entry;

if(temp!=NULL&&temp->entry.type\_piority>2){

found=1;

}

}

if(found==0){

\*current = NULL;

}

else{

\*current = temp;

}

return;

}

void assign\_slot(edf\_timetable\_list\* location,char \*name){

location->timeslot.name = name;

}

bool validate\_RA(){

if((date\_cmp(current\_slot->timeslot.edf\_date,current\_RA->entry.edf\_date)==0 && current\_RA->entry.edf\_time>current\_slot->timeslot.edf\_time)

|| date\_cmp(current\_RA->entry.edf\_date,current\_slot->timeslot.edf\_date)==1){

if(date\_inrange(edf\_start\_date, current\_RA->entry.edf\_date, edf\_end\_date)){

if(current\_RA->entry.edf\_time>=edf\_start\_hr && current\_RA->entry.edf\_time+current\_RA->entry.edf\_cost<=edf\_end\_hr){

return true;

}

}

}

return false;

}

bool validate\_PA(){

/\*PA on same day but not last slot OR Later than today\*/

if( ((date\_cmp(current\_PA->entry.edf\_date, current\_slot->timeslot.edf\_date)==0 && current\_slot->timeslot.edf\_time!=edf\_end\_hr-1) ||

date\_cmp(current\_PA->entry.edf\_date, current\_slot->timeslot.edf\_date)==1)

/\*AND NOT out of period range\*/

&& date\_inrange(edf\_start\_date, current\_PA->entry.edf\_date, edf\_end\_date)

){

return true;

}

else{

return false;

}

}

void time\_check(){

/\*printf("Activity starts now.\n");\*/

int days\_til\_dead = (date\_days(current\_PA->entry.edf\_date, current\_slot->timeslot.edf\_date)-1);

/\*printf("Activity encountered: days til dead is: %d\n",days\_til\_dead);\*/

int cost\_left = (days\_til\_dead\*edf\_period\_day\_hrs) + (edf\_end\_hr -(current\_slot->timeslot.edf\_time));

/\*printf("Activity encountered: cost left is: %d\n",cost\_left);\*/

/\*Still have time to finish P/A\*/

if((cost\_left-(current\_RA->entry.edf\_cost)) >= PA\_r\_cost){

/\*printf("u can join current activity\n");\*/

assign\_slot(current\_slot, current\_RA->entry.name);

RA\_r\_cost-=1;

strcpy(status\_arr[current\_RA->entry.id],"Accepted");

/\*A R/A is finished\*/

if(RA\_r\_cost<=0){

/\*Get next until it is not on the same slot\*/

progress\_arr[current\_RA->entry.id] = 100;

do{

getNextRA(&current\_RA);

if(current\_RA!=NULL && !validate\_RA()){

strcpy(status\_arr[current\_RA->entry.id],"Rejected");

progress\_arr[current\_RA->entry.id] = 0;

/\*printf("RA ID %d: is rejected (can't validate)\n", current\_RA->entry.id);\*/

}

}while(current\_RA!=NULL && !validate\_RA());

if(current\_RA!=NULL){

RA\_r\_cost = current\_RA->entry.edf\_cost;

}

}

}

else{

/\*printf("u don't have time to join current activity\n");\*/

strcpy(status\_arr[current\_RA->entry.id],"Rejected");

progress\_arr[current\_RA->entry.id] = 0;

getNextRA(&current\_RA);

if(current\_RA!=NULL){

RA\_r\_cost = current\_RA->entry.edf\_cost;

if(date\_cmp(current\_slot->timeslot.edf\_date,current\_RA->entry.edf\_date)==0 && current\_RA->entry.edf\_time<=current\_slot->timeslot.edf\_time){

time\_check();

}

}

}

}

void assign\_PA(){

assign\_slot(current\_slot, current\_PA->entry.name);

if(PA\_r\_cost==current\_PA->entry.edf\_cost){

strcpy(status\_arr[current\_PA->entry.id],"Accepted");

}

PA\_r\_cost-=1;

/\*A P/A is finished\*/

if(PA\_r\_cost<=0 || (date\_cmp(current\_PA->entry.edf\_date,current\_slot->timeslot.edf\_date)==0 && current\_slot->timeslot.edf\_time==edf\_end\_hr-1)){

/\*printf("Current PA finished, finding next PA.\n");\*/

float temp = 1;

temp = ((temp\*current\_PA->entry.edf\_cost-PA\_r\_cost)/current\_PA->entry.edf\_cost)\*100;

progress\_arr[current\_PA->entry.id] = (int)temp;

/\*printf("%d has completed\n", (int) temp);\*/

do{

getNextPA(&current\_PA);

if(current\_PA!=NULL && !validate\_PA()){

strcpy(status\_arr[current\_PA->entry.id],"Rejected");

progress\_arr[current\_PA->entry.id] = 0;

/\*printf("PA ID%d : is rejected\n", current\_PA->entry.id);\*/

}

}while(current\_PA!=NULL && !validate\_PA());

if(current\_PA!=NULL){

PA\_r\_cost = current\_PA->entry.edf\_cost;

}

}

}

/\* Check the presence of entries and period\*/

if(entry\_head == NULL || timetable\_head==NULL){

printf("Entries or period missing.\n");

return;

}

else{

/\*Get the first PA / RA\*/

edf\_entry\_list\* temp = entry\_head;

int found = 0;

while(found==0&&temp!=NULL){

if(temp->entry.type\_piority<3){

current\_PA = temp;

found=1;

}

temp = temp->next\_entry;

}

while((current\_PA!=NULL)

&&

!date\_inrange(edf\_start\_date,current\_PA->entry.edf\_date,edf\_end\_date)

){

strcpy(status\_arr[current\_PA->entry.id],"Rejected");

progress\_arr[current\_PA->entry.id] = 0;

/\*printf("start PA ID%d : is rejected\n", current\_PA->entry.id);\*/

getNextPA(&current\_PA);

}

found = 0;

temp = entry\_head;

while(found==0&&temp!=NULL){

if(temp->entry.type\_piority>2){

current\_RA = temp;

found=1;

}

temp = temp->next\_entry;

}

while((current\_RA!=NULL)

&&

(

(!date\_inrange(edf\_start\_date,current\_RA->entry.edf\_date, edf\_end\_date))

||

((current\_RA->entry.edf\_time<edf\_start\_hr)||(current\_RA->entry.edf\_time+current\_RA->entry.edf\_cost>edf\_end\_hr))

)

){

strcpy(status\_arr[current\_RA->entry.id],"Rejected");

progress\_arr[current\_RA->entry.id] = 0;

/\*printf("start RA ID%d : is rejected\n", current\_RA->entry.id);\*/

getNextRA(&current\_RA);

}

/\*There is at least 1 Project or assignment\*/

current\_slot = timetable\_head;

/\*Initializing cost of current entry\*/

if(current\_PA!=NULL){

PA\_r\_cost = current\_PA->entry.edf\_cost;

}

/\*There is at least 1 revision / activity\*/

if(current\_RA!=NULL){

RA\_r\_cost = current\_RA->entry.edf\_cost;

}

}

/\* Assign until out of slots\*/

while(i<=total\_slots){

/\*printf("i=%d;\n",i);\*/

/\*There's still at least 1 PA / 1 RA\*/

if(current\_PA!=NULL&&current\_RA!=NULL){

/\*2 entries on same date\*/

if(date\_cmp(current\_slot->timeslot.edf\_date,current\_RA->entry.edf\_date)==0){

/\*Activity start on this slot or in progress\*/

if((current\_slot->timeslot.edf\_time==current\_RA->entry.edf\_time)||(RA\_r\_cost<current\_RA->entry.edf\_cost)){

/\* RA in progress\*/

if(RA\_r\_cost<(current\_RA->entry.edf\_cost)){

/\*printf("Activity in progress.\n");\*/

assign\_slot(current\_slot, current\_RA->entry.name);

RA\_r\_cost-=1;

/\*A R/A is finished\*/

if(RA\_r\_cost<=0){

strcpy(status\_arr[current\_RA->entry.id],"Accepted");

progress\_arr[current\_RA->entry.id] = 100;

do{

getNextRA(&current\_RA);

if(current\_RA!=NULL && !validate\_RA()){

strcpy(status\_arr[current\_RA->entry.id],"Rejected");

progress\_arr[current\_RA->entry.id] = 0;

/\*printf("RA ID %d: is rejected (can't validate)\n", current\_RA->entry.id);\*/

}

}while(current\_RA!=NULL && !validate\_RA());

if(current\_RA!=NULL){

RA\_r\_cost = current\_RA->entry.edf\_cost;

}

}

}

/\* RA starts at this slot\*/

else{

time\_check();

}

}

/\*Activity not on this slot\*/

else{

assign\_PA();

}

}

/\*Not on same date\*/

else{

assign\_PA();

}

}

else if(current\_PA!=NULL&&current\_RA==NULL){

assign\_PA();

}

else if(current\_PA==NULL&&current\_RA!=NULL){

/\*printf("No more PA and start scheduling RAs\n");\*/

/\*While RA is expired\*/

if(date\_cmp(current\_slot->timeslot.edf\_date,current\_RA->entry.edf\_date)==0){

/\*time is same

if( (current\_slot->timeslot.edf\_time==current\_RA->entry.edf\_time)||(RA\_r\_cost < current\_RA->entry.edf\_cost)){

/\* Scheduler accepted adding RA\*/

assign\_slot(current\_slot, current\_RA->entry.name);

RA\_r\_cost-=1;

/\*A R/A is finished\*/

if(RA\_r\_cost<=0){

strcpy(status\_arr[current\_RA->entry.id],"Accepted");

progress\_arr[current\_RA->entry.id] = 100;

do{

getNextRA(&current\_RA);

if(current\_RA!=NULL && !validate\_RA()){

strcpy(status\_arr[current\_RA->entry.id],"Rejected");

progress\_arr[current\_RA->entry.id] = 0;

}

}while(current\_RA!=NULL && !validate\_RA());

if(current\_RA!=NULL){

RA\_r\_cost = current\_RA->entry.edf\_cost;

}

}

}

}

/\* Both ptr is null and no more entries need to be scheduled; \*/

else{

break;

}

current\_slot = current\_slot->next\_slot;

i++;

}

}

void print\_timetable\_edf(){

int i, j = 0;

for(i=0; i < 14; i++){

for(j=0; j <4; j++){

/\*printf("Timetable print : Day=%d Slot=%d is %s\n",i,j,timetable[j][i]);\*/

}

}

}

/\*char timetable2[4][14][50] = {""};\*/

/\*!!!Only for 14 days 4hrs timetables!!!\*/

void import\_EDF\_2\_timetable(){

edf\_timetable\_list\* temp\_ptr;

if(timetable\_head!=NULL){

temp\_ptr = timetable\_head;

}

else{

printf("EDF period haven't added.\n");

}

int i, j = 0;

for(i=0; i < 14; i++){

for(j=0; j <4; j++){

if(temp\_ptr!=NULL && strcmp(temp\_ptr->timeslot.name,"N/A")!=0){

strncpy(timetable[j][i], temp\_ptr->timeslot.name, sizeof(timetable[j][i])-1);

/\*printf("i=%d j=%d imported name : %s\n", i,j,timetable[j][i]);\*/

}

else{

strcpy(timetable[j][i],"N/A");

}

temp\_ptr = temp\_ptr->next\_slot;

}

}

}

void command\_handler(int id, char\* input){

char \*command;

char \*parameters[5];

char \*token;

char \*saveptr = NULL;

int i = 0;

/\*printf("command handler running\n");\*/

/\*printf("Input is %s\n", input);\*/

command = strtok\_r(input, " ", &saveptr); /\*Extract a token\*/

/\*printf("Command is %s\n", command);\*/

token = strtok\_r(NULL, " ", &saveptr);

while(token != NULL && i < 5){

/\*printf("token %d is %s\n", i,token);\*/

parameters[i] = token;

token = strtok\_r(NULL, " ", &saveptr);

i++;

}

/\*printf("Parameter read done.\n", token);\*/

i--;

/\*printf("i is now %d.\n", i);\*/

if(strcmp(command,"runS3")!=0){

if(atoi(parameters[i]) == 0){

/\*printf("Scheduler received invalid message from pipe! Nothing need to do.\n");\*/

return;

}

}

/\*printf("The input is valid\n");\*/

/\*Add period\*/

if(strcmp(command,"addPeriod")==0){

/\*printf("Running add period.\n");\*/

timetable\_head = NULL;

timetable\_tail = NULL;

addPeriod(parameters);

}

/\*Add project\*/

else if(strcmp(command,"addProject")==0){

addProject(id, parameters);

}

/\*Add assignment\*/

else if(strcmp(command,"addAssignment")==0){

addAssignment(id, parameters);

}

/\*Add revision\*/

else if(strcmp(command,"addRevision")==0){

addRevision(id, parameters);

}

/\*Add activity\*/

else if(strcmp(command,"addActivity")==0){

addActivity(id, parameters);

}

/\* Start scheduling\*/

else if(strcmp(command,"runS3")==0){

if(strcmp(parameters[0],"EDF")==0){

/\*printf("EDF command handler: Running S3 EDF.\n");\*/

/\*print\_entries();\*/

EDF\_RunS3();

/\*print\_slots();\*/

import\_EDF\_2\_timetable();

/\*print\_timetable\_edf();\*/

}

/\*

else{

printf("EDF Scheduler received runS3 from pipe! But nothing need to do.\n");

}

\*/

}

/\*

else if(strcmp(command,"exitS3")==0){

exit(0);

}

\*/

else{

printf("EDF module unable to resolve this command.\n");

}

return;

}

/\*EDF\*/

write(fd[i\_write\_pipe][1], "OK", 2); /\* for synchronization (the scheduling module is ready to read new data from input module in the beginning of the program) \*/

/\* read user inputs until the write end of input module is closed \*/

while((a = read(fd[i\_read\_pipe][0], buffer, 50)) > 0) {

write(fd[i\_write\_pipe][1], "OK", 2); /\* for synchronization (the input module can now write new data to the pipe as the scheduling module finishes reading old data in the pipe) \*/

buffer[a] = 0;

/\* read the first 5 characters to determine to type of command \*/

for (b = 0; b < 5; b ++) command[b] = buffer[b];

command[b] = 0;

char edf\_input[50];

strcpy(edf\_input, buffer);

/\*printf("EDF received command: %s\n", edf\_input);\*/

if(strcmp(command, "runS3") != 0){

command\_handler(id, edf\_input);

}

/\* addPeriod \*/

if (strcmp(command, "addPe") == 0) {

/\* saving the starting day \*/

for (a = 0; a < 2; a ++) string[a] = buffer[a + 18];

string[a] = 0;

start\_day = atoi(string);

/\* saving the ending day \*/

for (a = 0; a < 2; a ++) string[a] = buffer[a + 29];

string[a] = 0;

end\_day = atoi(string);

/\* saving the starting hour \*/

for (a = 0; a < 2; a ++) string[a] = buffer[a + 32];

string[a] = 0;

start\_hour = atoi(string);

/\* saving the ending hour \*/

for (a = 0; a < 2; a ++) string[a] = buffer[a + 38];

string[a] = 0;

end\_hour = atoi(string);

/\* add the timeslots or period to the timetable \*/

for (a = start\_day - 8; a <= end\_day - 8; a ++) {

for (b = start\_hour - 19; b < end\_hour - 19; b++) {

strcpy(timetable[b][a], "N/A"); /\* N/A means that that period is added but with no assignment \*/

}

}

}

/\* addRevision / addActivity / addAssignment / addProject \*/

else if (strcmp(command, "addRe") == 0 || strcmp(command, "addAc") == 0 || strcmp (command, "addAs") == 0 || strcmp (command, "addPr") == 0) {

/\* saving the type into the type\_arr \*/

if (strcmp (command, "addRe") == 0) {

type\_arr[id] = malloc((strlen("addRevision") + 1) \* sizeof(char));

strcpy(type\_arr[id], "addRevision");

}

else if (strcmp (command, "addAc") == 0) {

type\_arr[id] = malloc((strlen("addActivity") + 1) \* sizeof(char));

strcpy(type\_arr[id], "addActivity");

}

else if (strcmp (command, "addAs") == 0) {

type\_arr[id] = malloc((strlen("addAssignment") + 1) \* sizeof(char));

strcpy(type\_arr[id], "addAssignment");

}

else if (strcmp (command, "addPr") == 0) {

type\_arr[id] = malloc((strlen("addProject") + 1) \* sizeof(char));

strcpy(type\_arr[id], "addProject");

}

/\* calculate the length of the string for the duration \*/

temp = 1;

b = a - 4;

while (buffer[b --] != ' ') temp++;

/\* saving the duration into the duration\_arr \*/

for (b = 0; b < temp; b ++) string[b] = buffer[a - 2 - temp + b];

string[b] = 0;

duration\_arr[id] = atoi(string);

/\* addRevision / addActivity \*/

if (strcmp (command, "addRe") == 0 || strcmp (command, "addAc") == 0) {

/\* saving the time into the time\_arr \*/

for (b = 0; b < 2; b ++) string[b] = buffer[a - 8 - temp + b];

string[b] = 0;

time\_arr[id] = atoi(string);

/\* saving the day into the date\_arr \*/

for (b = 0; b < 2; b ++) string[b] = buffer[a - 11 - temp + b];

string[b] = 0;

date\_arr[id].day = atoi(string);

/\* saving the month into the date\_arr \*/

for (b = 0; b < 2; b ++) string[b] = buffer[a - 14 - temp + b];

string[b] = 0;

date\_arr[id].month = atoi(string);

/\* saving the year into the date\_arr \*/

for (b = 0; b < 4; b ++) string[b] = buffer[a - 19 - temp + b];

string[b] = 0;

date\_arr[id].year = atoi(string);

/\* saving the event name into the event\_name\_arr \*/

for (b = 0; b <= a - 33 - temp; b ++) string[b] = buffer[b + 12];

string[b] = 0;

event\_name\_arr[id] = malloc((strlen(string) + 1) \* sizeof(char));

strcpy(event\_name\_arr[id], string);

}

else {

/\* saving the time into the time\_arr \*/

for (b = 0; b < 2; b ++) string[b] = buffer [a - 5 - temp + b];

string[b] = 0;

date\_arr[id].day = atoi(string);

/\* saving the month into the date\_arr \*/

for (b = 0; b < 2; b ++) string[b] = buffer[a - 8 - temp + b];

string[b] = 0;

date\_arr[id].month = atoi(string);

/\* saving the year into the date\_arr \*/

for (b = 0; b < 4; b ++) string[b] = buffer[a - 13 - temp + b];

string[b] = 0;

date\_arr[id].year = atoi(string);

/\* saving the event name into the event\_name\_arr \*/

if (strcmp (command, "addAs") == 0) for (b = 0; b <= a - 29 - temp; b ++) string[b] = buffer[b + 14];

else for (b = 0; b <= a - 26 - temp; b ++) string[b] = buffer[b + 11];

string[b] = 0;

event\_name\_arr[id] = malloc((strlen(string) + 1) \* sizeof(char));

strcpy(event\_name\_arr[id], string);

}

/\* the status of the invalid input is set to "Invalid" \*/

if (buffer[a - 1] == '0') {

strcpy(status\_arr[id], "Invalid");

progress\_arr[id] = 0;

}

buffer[a - 2] = 0;

strcpy(request[id], buffer); /\* copy the event to the request list \*/

id ++; /\* increment the id \*/

}

/\* runS3 \*/

else if (strcmp(command, "runS3") == 0) {

/\* reading the algorithm \*/

for (a = 0; a < 3; a ++) string[a] = buffer[a + 6];

string[a] = 0;

if (strcmp(string, "FCF") == 0) {

FCFS(id,type\_arr,date\_arr,time\_arr,event\_name\_arr,duration\_arr,timetable,progress\_arr,status\_arr);

}

else if (strcmp(string, "EDF") == 0) {

command\_handler(id, edf\_input);

}

read(fd[oa\_read\_pipe][0], syn, 10); /\* for synchronization (wait until the output & analyzer module finishes reading old data in the pipe) \*/

write(fd[oa\_write\_pipe][1], buffer, strlen(buffer)); /\* send the user input as a string to the output & analyzer module \*/

read(fd[oa\_read\_pipe][0], syn, 10); /\* for synchronization (wait until the output & analyzer module finishes reading old data in the pipe) \*/

write(fd[oa\_write\_pipe][1], &id, sizeof(id));

read(fd[oa\_read\_pipe][0], syn, 10); /\* for synchronization (wait until the output & analyzer module finishes reading old data in the pipe) \*/

write(fd[oa\_write\_pipe][1], request, 1000\*50\*sizeof(char));

read(fd[oa\_read\_pipe][0], syn, 10); /\* for synchronization (wait until the output & analyzer module finishes reading old data in the pipe) \*/

write(fd[oa\_write\_pipe][1], status\_arr, 1000\*10\*sizeof(char));

read(fd[oa\_read\_pipe][0], syn, 10); /\* for synchronization (wait until the output & analyzer module finishes reading old data in the pipe) \*/

write(fd[oa\_write\_pipe][1], progress\_arr, 1000\*sizeof(int));

read(fd[oa\_read\_pipe][0], syn, 10); /\* for synchronization (wait until the output & analyzer module finishes reading old data in the pipe) \*/

write(fd[oa\_write\_pipe][1], timetable, 4\*14\*50\*sizeof(char)); /\* send the time table to the output & analyzer module \*/

/\* reset the status of valid requests for next scheduling \*/

for (a = 0; a < 1000; a ++) {

if (strcmp(status\_arr[a],"Invalid") != 0) {

strcpy(status\_arr[a],"");

}

}

/\* reset the timetable for next scheduling \*/

for (a = 0; a < 14; a ++) {

for (b = 0; b < 4; b ++) {

if (strcmp(timetable[b][a], "") != 0) strcpy(timetable[b][a], "N/A");

}

}

}

}

/\* close all the pipes ends when the input module stops writing (process ends) \*/

close(fd[i\_read\_pipe][0]);

close(fd[i\_write\_pipe][1]);

close(fd[oa\_read\_pipe][0]);

close(fd[oa\_write\_pipe][1]);

/\* free all allocated space \*/

for (a = 0; a < 1000; a ++) {

if (type\_arr[id] != NULL) free(type\_arr[id]);

if (event\_name\_arr[id] != NULL) free(event\_name\_arr[id]);

}

}

/\* end of 1st child 1st child (Parent: input module, 1st child: scheduling module) \*/

/\* 2nd child (Parent: input module, 2st child: output&analyzer module) \*/

if (pid==2){

char filename[50]="";

char \* filename2 = "";

char a\_name[5]="";

int NA=0; /\*counting time slots used (how many N/A)\*/

int acc=0;/\*number of accepted\*/

int rej=0;/\*number of rejected\*/

int X=0; /\*counting time slots used (how many unused period)\*/

int c=0; /\*counting status\*/

int id=0;

int count=6;

int s\_read\_pipe = 3; /\* fd[3] is the pipe for output&analyzer module to read from scheduling module \*/

int s\_write\_pipe = 2; /\* fd[2] is the pipe for output&analyzer module to write to scheduling module \*/

FILE \* fp1;

FILE \* fp2;

/\* close unused pipe ends \*/

for (a = 0; a < 4; a ++) {

if (a != s\_read\_pipe && a != s\_write\_pipe) {

close(fd[a][0]);

close(fd[a][1]);

}

else if (a == s\_read\_pipe) close(fd[a][1]);

else close(fd[a][0]);

}

char timetable[4][14][50] = {""};

char request[1000][50] = {""};

char status\_arr[1000][10] = {""};

int progress\_arr[1000];

write(fd[s\_write\_pipe][1], "OK", 2); /\* for synchronization (the output & analyzer module is ready to read new data from scheduling module in the beginning of the program) \*/

/\* read user inputs until the write end of scheduling module is closed \*/

while((a = read(fd[s\_read\_pipe][0], buffer, 50)) > 0 ){

write(fd[s\_write\_pipe][1], "OK", 2); /\* for synchronization (the scheduling module can now write the time to the pipe as the output & analyzer module finishes reading old data in the pipe) \*/

read(fd[s\_read\_pipe][0], &id, sizeof(id));

write(fd[s\_write\_pipe][1], "OK", 2); /\* for synchronization (the scheduling module can now write the time to the pipe as the output & analyzer module finishes reading old data in the pipe) \*/

read(fd[s\_read\_pipe][0], request, 1000\*50\*sizeof(char));

write(fd[s\_write\_pipe][1], "OK", 2); /\* for synchronization (the scheduling module can now write the time to the pipe as the output & analyzer module finishes reading old data in the pipe) \*/

read(fd[s\_read\_pipe][0], status\_arr, 1000\*10\*sizeof(char));

write(fd[s\_write\_pipe][1], "OK", 2); /\* for synchronization (the scheduling module can now write the time to the pipe as the output & analyzer module finishes reading old data in the pipe) \*/

read(fd[s\_read\_pipe][0], progress\_arr, 1000\*sizeof(int));

write(fd[s\_write\_pipe][1], "OK", 2); /\* for synchronization (the scheduling module can now write the time to the pipe as the output & analyzer module finishes reading old data in the pipe) \*/

read(fd[s\_read\_pipe][0],timetable,4\*14\*50\*sizeof(char));

write(fd[s\_write\_pipe][1], "OK", 2); /\* for synchronization (the scheduling module can now write the time to the pipe as the output & analyzer module finishes reading old data in the pipe) \*/

/\* Initializing \*/

NA=0; /\*counting time slots used (how many N/A)\*/

acc=0;/\*number of accepted\*/

rej=0;/\*number of rejected\*/

X=0; /\*counting time slots used (how many unused period)\*/

c=0; /\*counting status\*/

memset(a\_name, 0, sizeof(a\_name));

count=6;

while(buffer[count]!=' '){

a\_name[count-6]=buffer[count];

count++;

}

count++;

a = 0;

while(buffer[count]!=' '){

filename[a]=buffer[count];

count++;

a++;

}

filename[a] = 0;

if(strcmp(a\_name,"FCFS")==0){

filename2="S3\_FCFS.log";

}

else{

filename2="S3\_EDF.log";

}

fp1= fopen(filename,"w+");

fprintf(fp1,"Alice Timetable\r\n");

fprintf(fp1,"2019-04-08 to 2019-04-21\r\n");

fprintf(fp1,"Algorithm: ");

fprintf(fp1,"%s",a\_name);

fprintf(fp1,"\r\n\r\n");

fprintf(fp1,"%-20s","Date");

fprintf(fp1,"%-20s","19:00");

fprintf(fp1,"%-20s","20:00");

fprintf(fp1,"%-20s","21:00");

fprintf(fp1,"%-20s\r\n","22:00");

for (a = 0; a < 14; a ++) {

for (b = 0; b < 5; b ++) {

if(b==0){

if(a<2){

fprintf(fp1, "%s%-12d", "2019-4-0",a + 8);

}

else{

fprintf(fp1, "%s%-13d", "2019-4-",a + 8);

}

}

else{

if (strcmp(timetable[b-1][a], "") != 0) {

fprintf(fp1, "%-20s", timetable[b-1][a]);

}

else{

fprintf(fp1,"%-20s","X");

}

}

}

fprintf(fp1,"\r\n");

}

fprintf(fp1,"\r\n\*\*\*Summary Report\*\*\*\r\n\r\n");

fprintf(fp1,"Algorithm used: ");

fprintf(fp1,"%s",a\_name);

fprintf(fp1,"\r\nThere are %d requests.\r\n",id);

while(c<id){

if(strcmp(status\_arr[c],"Accepted")==0){acc++;}

else if(strcmp(status\_arr[c],"Rejected")==0){rej++;}

c++;

}

fprintf(fp1,"Number of request accepted: %d\r\n",acc);

fprintf(fp1,"Number of request rejected: %d\r\n\r\n",rej);

for (a = 0; a < 14; a ++) {

for (b = 0; b < 4; b ++) {

if(strcmp(timetable[b][a],"N/A")==0){NA++;}

else if(strcmp(timetable[b][a],"")==0){X++;}

}

}

if (X == 56) { /\* avoid division by zero error \*/

fprintf(fp1,"Number of time slots used: %d",4\*14-X-NA);

}

else {

fprintf(fp1,"Number of time slots used: %d (%d%%)",4\*14-X-NA,((4\*14-X-NA)\*100/(4\*14-X)\*100)/100);

}

fclose(fp1);

fp2= fopen(filename2,"w+");

fprintf(fp2,"\*\*\*Log File - %s\*\*\*\r\n",a\_name);

fprintf(fp2,"%-8s","ID");

fprintf(fp2,"%-43s","Event");

fprintf(fp2,"%-s\r\n","Accepted/Rejected");

fprintf(fp2,"-------------------------------------------------------------\r\n");

for(count=0;count<id;count++){

fprintf(fp2,"000%-5d",count+1);

if(progress\_arr[count]!=100&&progress\_arr[count]!=0){

fprintf(fp2,"%s(%-d",request[count],progress\_arr[count]);

fprintf(fp2,"%-7s","%)");

}

else{fprintf(fp2,"%-45s",request[count]);}

fprintf(fp2,"%-s\r\n",status\_arr[count]);

}

fclose(fp2);

}

/\* close all the pipe ends when the scheduling module stops writing (process ends) \*/

close(fd[s\_read\_pipe][0]);

close(fd[s\_write\_pipe][1]);

}

/\* end of 2nd child (Parent: input module, 2st child: output&analyzer module) \*/

}

/\* Parent process: Input module \*/

else {

int s\_read\_pipe = 0; /\* fd[0] is the pipe for input module to read from scheduling module \*/

int s\_write\_pipe = 1; /\* fd[1] is the pipe for input module to write to scheduling module \*/

/\* close unused pipe ends \*/

for (a = 0; a < 4; a ++) {

if (a != s\_read\_pipe && a != s\_write\_pipe) {

close(fd[a][0]);

close(fd[a][1]);

}

else if (a == s\_read\_pipe) close(fd[a][1]);

else close(fd[a][0]);

}

char input[50] = ""; /\* the string to be sent to the scheduler (format: [user input] [indicator]) (indicator: '0': invalid / '1': valid) \*/

char \* return\_string;

char algorithm[50];

char file\_name[50];

printf(" ~~WELCOME TO S3~~\n"); /\* start the program properly \*/

while (strcmp(command, "exitS3") != 0) {

/\* prompt until the user enter "exitS3" \*/

printf("Please enter:\n");

scanf("%s", command); /\* scanning the first string in an input line \*/

/\* call different functions according to the first string in the input line \*/

if (strcmp(command, "addPeriod") == 0) {

return\_string = validatePeriod();

strcpy(input, return\_string);

free(return\_string); /\* free the allocated space \*/

/\* send the input to scheduler only if the input for addPeriod is valid \*/

if (input[44] == '1') {

read(fd[s\_read\_pipe][0], syn, 10); /\* for synchronization (wait until the scheduling module finishes reading old data in the pipe) \*/

write(fd[s\_write\_pipe][1], input, strlen(input));

}

}

else if (strcmp(command, "addRevision") == 0 || strcmp(command, "addActivity") == 0 || strcmp(command, "addAssignment") == 0 || strcmp(command, "addProject") == 0) {

return\_string = validateEvent(&id, command, stdin);

strcpy(input, return\_string);

free(return\_string); /\* free the allocated space \*/

read(fd[s\_read\_pipe][0], syn, 10); /\* for synchronization (wait until the scheduling module finishes reading old data in the pipe) \*/

write(fd[s\_write\_pipe][1], input, strlen(input)); /\* send the input to scheduler \*/

}

else if (strcmp(command, "addBatch") == 0) {

scanf("%s", file\_name);

FILE \*fp;

fp = fopen(file\_name, "r"); /\* open a file \*/

/\* error in opening file \*/

if (fp == NULL)

{

printf("Error in opening input file\n");

exit(1);

}

/\* loop until there is no input in the file opened \*/

while (fscanf(fp, "%s", command) == 1) {

return\_string = validateEvent(&id, command, fp);

strcpy(input, return\_string);

free(return\_string); /\* free the allocated space \*/

read(fd[s\_read\_pipe][0], syn, 10); /\* for synchronization (wait until the scheduling module finishes reading old data in the pipe) \*/

write(fd[s\_write\_pipe][1], input, strlen(input)); /\* send the input to scheduler \*/

}

fclose(fp); /\* close the file \*/

}

else if (strcmp(command, "runS3") == 0) {

strcpy(input, command);

strcat(input, " ");

scanf("%s", algorithm); /\* scan algorithm \*/

if (strcmp(algorithm, "FCFS") != 0 && strcmp(algorithm, "EDF") != 0) {

printf("Invalid algorithm (Valid: FCFS / EDF)\n");

}

strcat(input, algorithm); /\* concatenate the algorithm to the input \*/

strcat(input, " ");

scanf("%s", file\_name); /\* scan filename \*/

strcat(input, file\_name); /\* concatenate the filename to the input \*/

if (strcmp(algorithm, "FCFS") == 0 || strcmp(algorithm, "EDF") == 0) {

read(fd[s\_read\_pipe][0], syn, 10); /\* for synchronization (wait until the scheduling module finishes reading old data in the pipe) \*/

write(fd[s\_write\_pipe][1], input, strlen(input)); /\* send the input to scheduler \*/

}

}

else if (strcmp(command, "exitS3") == 0) {

continue;

}

else {

printf("invalid command\n");

}

}

printf("Bye-bye!\n"); /\* end the program properly \*/

/\* close all the pipe ends when the input module stops writing (process ends) \*/

close(fd[s\_read\_pipe][0]);

close(fd[s\_write\_pipe][1]);

while(wait(NULL)>0);

exit(0);

}

/\* end of parent process (input module) \*/

}

*Appendix II*

Contribution of Work:

|  |  |
| --- | --- |
| Cheung Ka Ho | Scheduling module |
| Ho Kin Ling | Output & Analyzer Module |
| Ngan Ting Cheuk | Input Module |
| So Hiu Tung | Documentation |
| Yan Ho Wang | Scheduling module |