COMP2432 Operating Systems

Multi-Process Task Scheduler Project

LIU Le 15103617D
YU Jing 16098537D
DING Dashan 17082316D
LIU Fengming 15104126D

April 9, 2019
The Hong Kong Polytechnic University

Contents

1 Introduction

This project aims to familiarize ourselves with one of the operating system's essential tasks that is scheduling, as well as sharpen our own programming skills and understand some of the most important system calls (fork(), pipe(), etc.) of the Linux operating system, by implementing a scheduler that schedules students activities, which is analogous to an OS's scheduling task, using these system calls.

2 Scope

The main areas of operating systems that are covered in this project are:

(1) Multiprogramming (multiple processes)

(2) Inter-process Communication

(3) Synchronization

(4) Scheduling

(5) Protection

For multiprogramming and inter-process communication (IPC), our program creates several processes that communicate with each other with the `pipe()` system call of the standard C library.

For synchronization, we orchestrate the rhythms of different processes with a polling/acknowledgement synchronization mechanism: each time a process sends out a message to other processes, the other process responds with a acknowledgement message to notify the sender of receipt of the message before the sender can continue eecution. This mechanism takes its inspiration from the "ACK" mechanism of the TCP protocol.

The biggest part of the project is the scheduling kernel, which resembles the scheduling of tasks by operating system kernels, but differs in significant ways: for a general-purpose modern operating system, the scheduler runs in a "on-the-fly" basis, meaning that the tasks are not pre-known, and jobs are submitted at random time points as demanded, the whole process is indeterministic; however, full information of the tasks to be scheduled in this project are pre-known before the execution of the program, and the whole process is deterministic given that the inputs are unchanged. These characteristics make the scheduling tasks in this project more like batch systems in the early days of computers, where a batch of job is loaded in one go into the machine before the machine is started and execution begins. It is not exactly the same as batch systems though, since early batch systems do not attempt to schedule the tasks at all but instead simply execute them sequentially, first come first served.

3 Concepts & Algorithms

Due to time limitation, we implemented only 3 algorithms. Two of them are what were taught in the lectures, namely, round robin (RR) and priority scheduling (PR); the other one was designed by us, and we proudly introduce the "deadline fighter algorithm" (the DF algorithm), we will explain where it got its funny yet appropriate name. Please note that throughout this documentation, we will use the term **event** as a generic term to refer to an object to be scheduled, i.e. a project, assignment, revision, or activity. In

addition, this project assumes that the minimum unit of time that can be handled is 1 hour, meaning all event duration, start time, end time, are integer numbers. An hour is also called a **time slot**.

3.1 Round Robin

The RR algorithm works as follows in our implementation: The initial queue is formed according to the sequence of input, i.e. an event A that is input earlier that an event B, will be in front of B in the initial queue. We assume that the time quantum is 1 hour. Once the scheduling starts, the current hour is set to the beginning of the time period entered, and the algorithm continuously tests the front of the queue to see:
If the event is a project/assignment, if its deadline has passed, discard it from the queue. Else, schedule it at the current hour and put it at the back of the queue, if it has not been finished;
If the event is a revision/activity, if its deadline has passed, discard it from the queue. Else, schedule it at the current slot if the current slot matches the start time of the event (provided the event is **legal**. See section 4.3 for definition of **legal**). If the current time slot does not match its start and its deadline has not passed, put it at the back of the queue and continue.

3.2 Priority

As is mentioned in section 2, full information of the events to be scheduled is known prior to the scheduling, therefore, the priority scheduling algorithm is somewhat different from that in modern general-purpose OS. The priority algorithm of our implementation works as follows:
Sort the events by priority. According to the project specification, priority Project > Assignment > Revision > Activity. For events with the same priority, events that belong to more advanced subjects come before subjects of lower levels, for example, level 4 subjects come before level 3 subjects, level 3 subjects come before level 2 subjects, and so on. In case of same type events from subjects of same level, i.e. COMP4000 project and COMP4110 project, the order is unimportant. Then, the scheduler schedules the events from top priority ones to lower ones sequentially. If in the course of an event, its deadline is reached (the event cannot be fully finished), the event is discarded from the queue and the scheduler continues to schedule the next one.

3.3 The Deadline Fighter

With the aim to achieve the best possible performance (to gain as much as possible from projects/assignments, as well as try to fit in as many as possible revisions and activities), we set out to design this "deadline fighter algorithm". The DF algorithm works as follows in our implementation:
We first calculate the total "benefits" of each individual event, which will be useful in the scheduling process later. Level 1 subject assignments have a base benefit of 20, level 1 subject projects have a base benefit of 30, since projects usually account for more percent of marks of the subject; revisions have a base benefit of 15, and activities have a fixed benefit of 5, since activities possess the lowest priority according to the specification of the project. With each level (of subject) higher than 1, 3 additional units of benefit are added to the total benefit for projects, assignments and revisions. For example, a level 3 subject assignment would have `20 + (3 - 1) * 3 = 26` units of total benefit; a level 4 subject project would have `30 + (4 - 1) * 3 = 39` units of total benefit, similar for revisions.

Then, we calculate the **unit benefit yield** for each of the project, which is calculated by dividing the total benefit with the duration of the event. For example, an event with 39 units of benefit and a duration of 26 hours, will yield a benefit of `39 / 26 = 1.5` units per hour. The **unit benefit yield** represents how much, on average, benefit the event yields for each hour of the event duration. Events with higher **unit benefit yield** means that the event is more "economically efficient", since more benefit can be gained for the same amount of time spent.

We then divide the events into 3 groups, the project/assignment group, the revision group, and the activity group, and sort the events within each group according to their **unit benefit yield** in descending order. In case of same unit benefit yield, the order is unimportant. Then, the scheduling process begins from the event with the highest unit benefit yield to lower ones sequentially in the project/assignment group. Afterwards, the algorithm moves to the revision group and then the activity group, in that order. The order is set in this way to be in accordance with the priority set by the project specification. We now look at how the algorithm schedules each event. For each project/assignment, the algorithm schedules the event from its deadline and moves forward in time consecutively (scan hour by hour). If an empty time slot is encountered, assign that time slot to the event. If a time slot is occupied, skip it, and move to its previous slot. The scheduling of this event stops when there is no more time slot available before its deadline, or the full duration of the event has been fulfilled. Then, the scheduler moves on to the next event.

After the project/assignment group finishes scheduling, the scheduler moves on to the revision group. Revisions, unlike projects/assignments, have to be done in one go, and have to happen at that specific time point. If these criteria cannot be satisfied, reject the event as a whole. We designed the algorithm to first complete as many projects/assignments as possible, then try to schedule as many revisions as possible. Therefore, we start from the first revision, and check if any of its required time slots has been occupied. If no, just schedule the revision there. If some of its time slots have been occupied by projects/assignments, we search forward in time to find other available time slots to re-schedule those time slots occupied them the projects/assignments. If enough slots is found searching forward in time, re-schedule these time slots to happen at earlier time slots, and schedule the revision here. If not enough is found, reject the revision. If some of its time slots are occupied by other revisions, reject the revision (since this revision is not allowed to replace previously scheduled revisions who have higher unit benefit yields). The scheduler handles other revisions alike.

After the revision group finished scheduling, the scheduler moves on to the activity group. The process is similar to the revision group, only different in that activities are not allowed to replace other activities as well as revisions.

The reason why this algorithm is called the "deadline fighter algorithm" is now evident. We try to gain as much as possible benefit from projects/assignments by scheduling them from top unit benefit yield to lower ones, also try to schedule them as late as possible (finish just before deadline) to ensure that it poses minimum interferences to other projects/assignments. If it does interfere with other projects/assignments, we search for available time slots earlier in time to finish the interfered event as much as possible. Although the name suggests a somewhat negative image, this algorithm mathematically guarantees that max benefit is extracted from projects and assignments.

We also thought about extracting as much benefit as possible from revisions and activities as well, but since revisions/activities have to be done in one go and have to happen at particular time points, the problem is clearly NP-complete [1] because it is essentially the Knapsack problem [2]. The key difference in scheduling projects/assignments and revisions/activities is the atomicity of revisions/activities, which is the case with the Knapsack problem [2]. Therefore, we decided to use the greedy approach as it is now,

which is to schedule according to unit benefit yield, and events with lower unit benefit yields cannot replace higher unit benefit ones, to avoid the super-polynomial (higher than polynomial) overhead to optimize this problem. The reason why the greedy approach fails to strictly optimize the general Knapsack problem is because an item with smaller total value but higher per-unit value may be efficient in terms of space, but will probably not maximize value since an item with lower per-unit benefit and higher total value may be able to fit in the remaining space (before placing in the higher per-unit benefit item) which is otherwise wasted. Similarly, an event with higher unit yield but smaller total benefit may prevent an event with lower unit yield but higher total benefit from being scheduled, therefore not able to reach maximum benefit in all occasions. However, in this particular application, the greedy approach yields good enough performance. The performance difference between the NP-complete approach and the greedy approach enlarges as the variance of the items increases. Simply put, the greedy approach performance deteriorates when there are items who differ significantly in total value and size, in our case, total benefit and event duration. However, for our application, the maximum duration of a revision/activity is 4 and minimum is 1, which is not a large range, and their total benefits differ only in a rather narrow neighborhood. Consequently, the greedy approach will perform fairly good in comparison with the NP-complete approach in our application.

4 Implementation

4.1 Program Structure

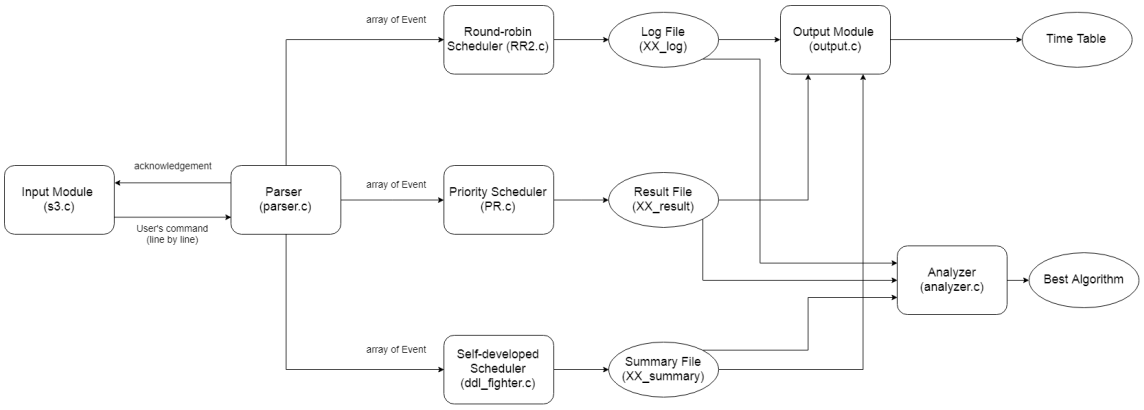


Figure 1: Program Structure

The program consists of seven modules (files):

Input
This module firstly forks a child process which is the Parser (see next paragraph). After that, this modules takes the commands input by the user and sends them to Parser line by line through pipes. After each command is received, the Parsesr sends an "ACK" message acknowledging its receipt, as described before.

Parser
This module receives the commands sent from the Input module via pipes and parses the unstructured commands into structured data, i.e. the Parser transforms the commands into variables inside an `Event` structure, or corresponding actions (run or exit commands). The addPeriod command is transformed to variables storing the information about the period and the starting and ending time of a day. The commands specifying events (i.e. addProject, addAssignment, addRevision, addActivity) are transformed

into elements in an array of `Event` structures. The "run" commands triggers the invocation of the schedulers accordingly (see section 8 for details). The Parser keeps track of the number of events input by the user, but does not check whether the event is actually legal. This check is done by the individual schedulers.

Round-Robin Scheduler
This module performs the Round-robin scheduling. It first transforms the array of Events into a linked list that represents a ready queue and stores the information about the period and the starting/ending time into variables. These work is done by the function RR_invoker(), which is a wrapper of the real scheduler. The real scheduler is the function Round_Robin(). This function runs the Round-robin algorithm. It records the by-hour usage of the period into the file RR_result, the accepts/rejects into the log file, and the summary report of the algorithm into the report file.

Priority Scheduler
This scheduler performs scheduling according to the priority of the processes, similar to that in operating systems. Similar to round robin, it first transforms the array of Events into a sorted linked list by PR_invoker() and Sort_By_Priority(). And then the function named Priority() takes in the linked list and some related information and runs the algorithm. The scheduler records the by-hour usage of the period into the file PR_result, the accepts/rejects into the log file, and the summary report of the algorithm into the report file.

The Deadline Fighter Scheduler
This module is the deadline fighter algorithm scheduler. The function `fight_ddl()` in the deadline fighter scheduler runs the actual algorithm, other functions in the `ddl_fighter.c` supports it. Loging, reporting and result file behavior of this module is the same as the other schedulers.

Output
Each of the three schedulers mentioned above generates 3 files when they are run, namely `XX_result, S3_XX.log` and `S3_report_XX.dat`. The `XX_result` files record the individual schedulers' schedule result information that are intended for the output module (which is this module) to read and generate the time table.

Analyzer
This module takes in the result files (i.e.RR_result, ddl_fighter_result, PR_result), analyzes the performances of the algorithms, and outputs a file named "analyzer_summary.txt" in the output directory showing the total benefits of each algorithm and stating the best algorithm. **Please note that the analyzer analyze according to the XX_result files. Therefore, if result files from previous runs were not cleared, the analyzer will still take them into account, even if the algorithm might not have been invoked in the current run. To analyze only the algorithms that have been invoked in the current run, clear the result files left over by other schedulers in previous runs.**

4.2 Program Output

This section describes each of the files output by our program.

Our programs will output five types of files in "output" folder, which are log, summary (report), result, timetable of each algorithm, as well as an analyzer summary file.

Logs
Naming: S3_xxx.log, where "xxx" is the algorithm used, same below.

The corresponding log file will be output after running the scheduler of the algorithm. This file displays the details of each event with their assigned ID, state (i.e. "Accepted" or "Rejected"), and percentage of completion. In addition, the file lists all illegal events (see section 4.3 for details).

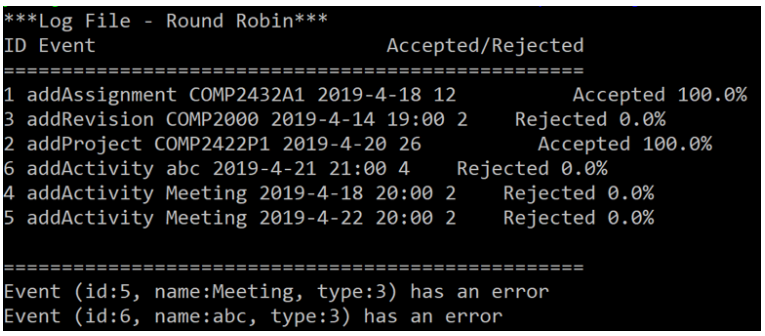


Figure 2: Example Log

Summary Report

Naming: S3_report_XXX.dat

The corresponding summary file will be output by each scheduler after running them. It is the summary of the scheduling result of the algorithms, showing the total number of events, the number of events that have been accepted or rejected, and the number of time slots used.

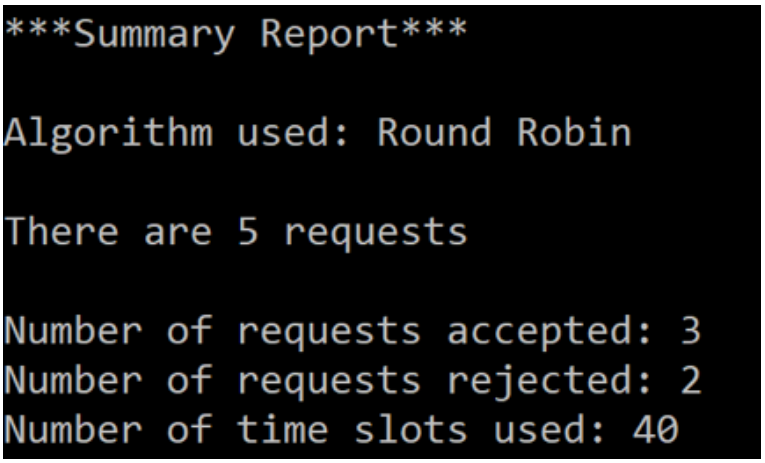


Figure 3: Example Summary Report

Result Files

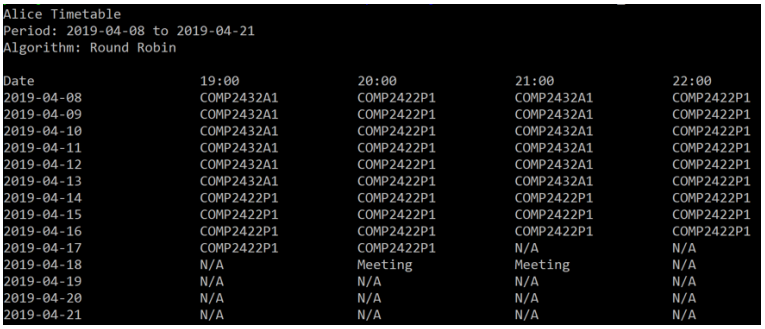
Naming: XXX_result

The corresponding result file will be output after running the scheduler of the algorithm. It lists out the assigned tasks in all time slots. Each line shows the detail information of each assignment (i.e. assigned date, assigned time, event ID, eventname, event type, and event duration). This non-required file is generated to provide necessary information for the output module to generate timetable as well as for the analyzer module to analyze performances.

Timetables

Naming: XXX_timetable.txt

The corresponding timetable file will be output after running the scheduler of the algorithm. It is a timetable that arrange tasks based on the algorithm, displaying the date, time slots and the events assigned. "N/A" means the time slot has NO event scheduled.



Alice Timetable
Period: 2019-04-08 to 2019-04-21
Algorithm: Round Robin

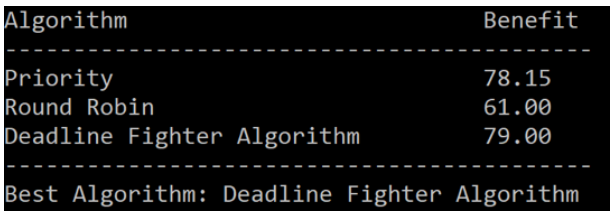
Date	19:00	20:00	21:00	22:00
2019-04-08	COMP2432A1	COMP2422P1	COMP2432A1	COMP2422P1
2019-04-09	COMP2432A1	COMP2422P1	COMP2432A1	COMP2422P1
2019-04-10	COMP2432A1	COMP2422P1	COMP2432A1	COMP2422P1
2019-04-11	COMP2432A1	COMP2422P1	COMP2432A1	COMP2422P1
2019-04-12	COMP2432A1	COMP2422P1	COMP2432A1	COMP2422P1
2019-04-13	COMP2432A1	COMP2422P1	COMP2432A1	COMP2422P1
2019-04-14	COMP2422P1	COMP2422P1	COMP2422P1	COMP2422P1
2019-04-15	COMP2422P1	COMP2422P1	COMP2422P1	COMP2422P1
2019-04-16	COMP2422P1	COMP2422P1	COMP2422P1	COMP2422P1
2019-04-17	COMP2422P1	COMP2422P1	N/A	N/A
2019-04-18	N/A	Meeting	Meeting	N/A
2019-04-19	N/A	N/A	N/A	N/A
2019-04-20	N/A	N/A	N/A	N/A
2019-04-21	N/A	N/A	N/A	N/A

Figure 4: Example Timetable

Analyzer Summary File

Naming: analyzer_summary

The analyzer summary file will be output after running the analyzer. It lists the results of performance analysis, which are benefits (calculated scores) of all algorithms, and the best algorithm among them.



```
Algorithm                              Benefit
-----------------------------------------------
Priority                               78.15
Round Robin                            61.00
Deadline Fighter Algorithm             79.00
-----------------------------------------------
Best Algorithm: Deadline Fighter Algorithm
```

Figure 5: Example Analyzer Summary

4.3 Error Handling

The program does not check for syntactic errors, but will print out events that are **illegal** to its log file. An event is **illegal** if its deadline is not within the period entered (for all types of events, e.g., a project with a deadline of 2019-04-30 is illegal since the deadline is not within the time period entered, an activity scheduled on 2019-04-28 is illegal for the same reason), or for revisions/activities, the event period exceeds the active hours of the day (a revision that happens at 12:00 midnight is **illegal**, which is outside the 19:00 - 23:00 designated period). Negative durations, invalid dates (e.g. 2019-15-90), are not checked.

5 Testing

Test cases of the program were strategically constructed to ensure the program is bug-free in extreme and boundary cases, which are situations where programs are most likely to have problems, and indeed, test cases helped us discover bugs. All the test case files are batch input files that can be directly added to

the program using the "addBatch" command described in section 8. **The test case files are stored under the `test_cases` directory under the root directory of the project.** The files are named as numbers. In this section, we describe a few of the test cases. For the full test case details, please refer to directory `test_cases`.

Case 0:
A realistic situation.

Case 1:
Mixed projects/assignments/revisions/activities. This is the basic case provided in the project specification but with one additional illegal event (for definition of "illegal", refer to section 4.3) to test that the algorithms can correctly reject illegal events.

Case 2:
More illegal events.

Case 3:
A random set of events.

Case 4:
An extremely long project.

Case 5:
A big amount of activities (10 of them).

Case 6:
A big amount of activities (10 of them) with an extremely long project.

6 Performance Evaluation

As mentioend, we assume that the benefit that can be extracted by partially completing projects/assignments increase linearly with respect to the percentage of completion, and that is one of the basic assumptions of our algorithm. We evaluate the performances of the algorithms based on the total benefit scheme described in section 3.3. Upon extensive testing, the DF algorithm developed by us, as has been justified in section 3.3, outperformed the other two in all other test cases apart from test case 6.

The only exception is due to a rare combination of the nature of the algorithms and a set of unusual events. The DF algorithm schedules projects/assignments first (before revisions/activities) as is demanded by the project specification, but round-robin does not adhere to the order of priority. In addition, the unique nature of this test case is "a big amount of activities (10 of them) with an extremely long project", in which case the unit benefit gain is extremely low for the project, actually making it a very bad deal (for a student); however although activities have only 5 units of total benefit each, they are usually very short and thus achieves much higher unit benefit gain. In summary, the combination of these two factors contributed to this rare exception. However, this test case is only constructed to test the correctness of the program and is not a realistic shceduling situation since the project is way too long (100 hours) and activities are way too many. Therefore, the superiority of the DF algorithm still stands in almost all situations and our design remains valid.

7 Limitation

Due to the strained time limit, limitations exist for the program. The major ones are as follows.

Firstly, the period has to be entered ONLY at the first user input. Multiple entries of period or entering period at any other statements other than the first statement will cause an error. (It is okay if the addPeriod statement is the first statement in a batch file, in which case the addBatch statement can be the first user input.)

Secondly, all events have to be added before any algorithm is run. Once an algorithm has been run (whichever offered by our program), no further insertion of event is accepted. To add another event after an algorithm has been run, the only way is to restart the program and re-enter.

Thirdly, as assumed by the project specification, the "period" of the scheduler is (tentatively) from April 8 to 21 (14 days). And for each day, it starts from 19:00 and ends at 23:00 (4 hours).

Fourthly, the scheduler supports 999 events at most, and a command has a maximum length of 200 characters.

8 Compiling & Using

Compile the program with (on Apollo):

```
$ gcc -std=c99 *.c
```

Run the program with:

```
$ ./a.out
```

To add a batch to the scheduler, use:

```
addBatch [batch_file_path]
```

For example, `addBatch batch.txt` will add all events included in the file `batch.txt` to the program. To add a period/project/assignment/revision/activity, use (as specified in the program specification):

```
addPeriod 2019-04-01 2019-04-31 19:00 23:00
addAssignment COMP2432A1 2019-04-18 12
addProject COMP2422P1 2019-04-20 26
addRevision COMP2000 2019-04-14 19:00 2
addActivity Meeting 2019-04-18 20:00 2
```

To run the Round Robin algorithm:

```
run rr
```

To run the Priority algorithm:

```
run pr
```

To run the Deadline Fighter algorithm:

```
run ddl
```

To run all available algorithms:

```
run all
```

To analyze:

```
analyze
```

Tha analyzer inspects the result files to calculate the total benefit obtained by each algorithm and selects the best.

To exit:

```
exitS3
```

Since the program stores its result/log/report files in the `output` directory under the root directory of the program. Please make sure to create an output folder first before running the program, otherwise the program will have error for not being able to find the output directory.

9 Work Division

Note that the order of appearance in this section does NOT correspond to the degree of contribution to the project.

9.1 LIU Le

LIU Le (15103617D) was responsible for writing the entire `parser.c, parser.h, ddl_fighter.c` and `ddl_fighter.h` files. The "deadline fighter" algorithm was conceived, designed and implemented by LIU Le. LIU Fengming offered suggestions in implementing the "deadline fighter" algorithm. LIU Le helped debugging the `s3.c` and `PR.c` files and was heavily involved in structuring the modules of the program, designing implementation details, as well as testing of the entire program (constructing test cases). Report sections 1, 2, 3, part of 4.1, 4.3, part of 5, entire 6, 7 and 8 were written by LIU Le.

9.2 LIU Fengming

LIU Fengming (15104126D) wrote the Round-Robin algorithm, created test cases, wrote part of the report, offered suggestions to the implementation of the DF algorithm.

9.3 YU Jing

YU Jing (16098537D) was responsible for writing the input, output and analyzer module, including the basic structure of s3.c program (implementing fork() and pipe()), as well as creating some test cases and writing part of the report.

9.4 DING Dashan

DING Dashan (17082316D) was responsible for implementing the priority algorithm (PR) and producing the result file. In addition, DING Dashan participated in testing the algorithm using the test cases and writing the report.

10 Appendix

This appendix contains the source codes (7 .c files) as well as header files (5 .h files). Please find example outputs (summary reports, logs, analyzer reports) in section 4.2.

Source Files

s3.c

```c
#include "s3.h"

/* prototype */
void getInput(char *instr);
void cmdToChild(int fd_toC[][2], char *instr);
void toChild(int fd_toC[][2], char *instr);
void test(int fd_toC[][2], int i);//to be deleted
char report_filename[100];
void analyzer();
float scoring(char *filename);

/* global variable */
int fd_toC[CHILD_NUM][2], fd_toP[CHILD_NUM][2];

/* main */
int main(int argc, char *argv[]) {
        int pid, i;

        // create pipes
        for (i = 0; i < CHILD_NUM; i++) {
                if (pipe(fd_toC[i]) < 0 || pipe(fd_toP[i]) < 0) {
                        printf("Pipe creation error\n");
                        exit(1);
                }
        }

        // create child processes
        for (i = 0; i < CHILD_NUM; i++) {
                pid = fork();
                if (pid < 0) {
```

```
31                      printf("Fork failed\n");
32                      exit(1);
33              }
34
35              if (pid == 0) { // child process
36                      close(fd_toC[i][1]);
37                      close(fd_toP[i][0]);
38
39                      int n=0;
40                      char str[100];
41                      bool parsed = false;
42                      while ((n = read(fd_toC[i][0],str,BUF_SIZE)) >
                            0) {
43                              str[n] = '\n'; str[n+1] = 0;
44                              write(fd_toP[i][1],"O",1); /* ACK
                                    message */
45                              if (strncmp(str,"run",3) == 0 && parsed
                                    == false) {
46                                      parsed = true;
47                                      parse();
48                              }
49                              if (strcmp(str,"run ddl\n") == 0) {
50                                      create_scheduler(DDL_FIGHTER);
51                                      continue;
52                              }
53                              else if (strcmp(str,"run rr\n") == 0) {
54                                      create_scheduler(RR);
55                                      continue;
56                              }
57                              else if (strcmp(str,"run pr\n") == 0) {
58                                      create_scheduler(PR);
59                                      continue;
60                              }
61                              else if (strcmp(str,"run all\n") == 0) {
62                                      create_scheduler(ALL);
63                                      continue;
64                              }
65                              else if (strcmp(str, "analyze\n") == 0)
                                    {
66                                      analyzer();
67                                      continue;
68                              }
69                              else if (strcmp(str,"exitS3\n") == 0) {
70                                      printf("Parser Exited!\n");
71                                      exit(0);
72                              }
```

```
73                                  /* Increment event_counter only if NOT
                                       run command */
74                                  strcpy(command[event_counter++],str);
75                          }
76
77                          close(fd_toC[i][0]);
78                          close(fd_toP[i][1]);
79                          exit(0);
80                  }
81          }
82
83          if (pid > 0) { // parent process
84                  for (i = 0; i < CHILD_NUM; i++) {
85                          close(fd_toC[i][0]);
86                          close(fd_toP[i][1]);
87                  }
88                  printf("\t~~WELCOME TO S3~~\n");
89                  char instr[BUF_SIZE];
90
91                  while (1) {
92                          getInput(instr);
93                          cmdToChild(fd_toC, instr);
94                          if (strcmp(instr, "exitS3") == 0) break;
95                  }
96
97                  printf("Bye-bye!\n");
98                  for (i = 0; i < CHILD_NUM; i++) {
99                          close(fd_toC[i][1]);
100                         close(fd_toP[i][0]);
101                 }
102         }
103
104         // wait for all child processes
105         for (i = 0; i < CHILD_NUM; i++)
106                 wait(NULL);
107
108         return 0;
109 }
110
111 /* input function: scan input command */
112 void getInput(char *instr) {
113         printf("Please enter:\n> ");
114         scanf("%[^\n]", instr); // scan the whole input line
115         getchar();
116 }
117
```

```
118 void sync() {
119         char temp[10];
120         for (int i = 0; i < CHILD_NUM; ++i)
121                 read(fd_toP[i][0],temp,1);
122 }
123
124 /* cmdToChild function: pass all inputed command to children */
125 void cmdToChild(int fd_toC[][2], char *instr) {
126         if (strncmp(instr, "addBatch", 8) != 0) {
127                 toChild(fd_toC, instr);
128                 sync();
129         }
130
131         else { // if the command is "addBatch ...", read file
132                 FILE *fp;
133                 char *filename = (char*) malloc(strlen(instr)-9+1);
134                 strcpy(filename, instr+9);
135                 fp = fopen(filename, "r");
136                 if (fp == NULL) {
137                         printf("Cannot open the file!\n");
138                         exit(1);
139                 }
140
141                 while(fscanf(fp, "%[^\n]\n", instr) != EOF) {
142                 //while( fgets (instr, BUF_SIZE, fp) != NULL ) {
143                         toChild(fd_toC, instr);
144                         int i = 0;
145                         sync();
146                 }
147                 fclose(fp);
148                 free(filename);
149         }
150 }
151
152 /* toChild function: pass a command to all children */
153 void toChild(int fd_toC[][2], char *instr) {
154         for (int i = 0; i < CHILD_NUM; i++)
155                 write(fd_toC[i][1], instr, strlen(instr));
156 }
157
158
159 //to be deleted
160 void test(int fd_toC[][2], int i) {
161         int n;
162         char buf[BUF_SIZE];
163
```

```
164         while ((n = read(fd_toC[i][0], buf, BUF_SIZE)) > 0) { // read
                from pipe
165                 buf[n] = 0;
166                 printf("<Child %d> message [%s] received of %d bytes\n",
                        getpid(), buf, n);
167                 if (buf[0] == 'e') break;
168         }
169 }
```

parser.c

```
1  /*
2   * Author: LIU Le 15103617d
3   * Date: 2019/3/27
4   */
5
6  // my headers
7  #include "parser.h"
8  #include "RR.h"
9  // macros
10
11 // prototypes
12
13 //global variables
14 struct Event events[1000]; // support at most 1000 events
15 int event_counter = 0;
16 char command[1000][200];
17 int period_start_date;
18 int period_end_date;
19 int period_start_time;
20 int period_end_time;
21
22 int split(char* input, char* output, int* start) {
23         char* ptr = output;
24         while (*(input + *start) != ' ' && *(input + *start) != '\n')
25                 *ptr++ = *(input + (*start)++);
26         *ptr = '\0';
27         if (*(input + *start -1) == '\n')
28                 return -1;
29         else
30                 return 0;
31 }
32
33 bool is_digit(char a) {
34         return (a >= '0' && a <= '9') ? true : false;
35 }
```

```
36
37 int parse_level(char* name) {
38         int n = strlen(name); int i = 0;
39         while (!is_digit(name[i++]));
40         return name[i-1]-'0';
41 }
42
43 void print_event(int i) {
44         printf("------------------\n");
45         printf("Event #%d\n", i);
46         printf("Name: %s\n",events[i].name);
47         printf("Type: %d\n",events[i].type);
48         printf("Date: %d\n",events[i].date);
49         printf("Time: %d\n",events[i].time);
50         printf("Duration: %d\n",events[i].duration);
51         printf("Unit_Benefit: %f\n",events[i].unit_benefit);
52         printf("------------------\n");
53 }
54
55 void parse_date(char* temp, int* dest) {
56         char date_temp[9]; int end=0;
57         for (int j = 0; j < strlen(temp); ++j)
58                 {
59                         if (temp[j] == '-')
60                                 continue;
61                         date_temp[end++] = temp[j];
62                 }
63         date_temp[end] = 0;
64         *dest = (int)atoi(date_temp);
65 }
66
67 // functions
68 void parse() {
69
70         // printf("Parsing!\n");
71         event_counter--;
72         for (int i = 1; i <= event_counter; ++i)
73                 events[i].id = i;
74         for (int i = 0; i <= event_counter; ++i)
75         {
76                 int a = 0;
77                 int* start = &a;
78                 char temp[100];
79                 split(command[i],temp,start);
80                 (*start)++;
81                 if (i == 0) {
```

```
82                      if (!(strcmp(temp,"addPeriod") == 0)) {
83                              printf("Must add period first. Exit.\n")
                                    ;
84                              exit(1);
85                      }
86                      split(command[i],temp,start);
87                      parse_date(temp,&(period_start_date));
88                      (*start)++;
89                      split(command[i],temp,start);
90                      parse_date(temp,&(period_end_date));
91                      (*start)++;
92                      // handle time
93                      split(command[i],temp,start);
94                      period_start_time = (temp[0] - '0') * 10 + (temp
                            [1] - '0');
95                      (*start)++;
96                      // handle time
97                      split(command[i],temp,start);
98                      period_end_time = (temp[0] - '0') * 10 + (temp
                            [1] - '0');
99                      (*start)++;
100             }
101
102             if (strcmp(temp,"addAssignment") == 0 || strcmp(temp,"
                    addProject") == 0) {
103                     if (strcmp(temp,"addAssignment") == 0)
104                             events[i].type = ASSIGNMENT_TYPE;
105                     else
106                             events[i].type = PROJECT_TYPE;
107                     // handle name
108                     split(command[i],temp,start);
109                     strcpy(events[i].name,temp);
110                     (*start)++;
111                     // handle date
112                     split(command[i],temp,start);
113                     parse_date(temp,&(events[i].date));
114                     (*start)++;
115                     // handle duration
116                     split(command[i],temp,start);
117                     events[i].duration = (int)atoi(temp);
118                     events[i].time = DAY_END;
119             }
120             else if (strcmp(temp,"addRevision") == 0 || strcmp(temp,
                    "addActivity") == 0) {
121                     if (strcmp(temp,"addRevision") == 0)
122                             events[i].type = REVISION_TYPE;
```

```
123                         else
124                                 events[i].type = ACTIVITY_TYPE;
125                         // handle name
126                         split(command[i],temp,start);
127                         strcpy(events[i].name,temp);
128                         (*start)++;
129                         // handle date
130                         split(command[i],temp,start);
131                         parse_date(temp,&(events[i].date));
132                         (*start)++;
133                         // handle time
134                         split(command[i],temp,start);
135                         events[i].time = (temp[0] - '0') * 10 + (temp[1]
                                - '0');
136                         (*start)++;
137                         // handle duration
138                         split(command[i],temp,start);
139                         events[i].duration = (int)atoi(temp);
140                 }
141         }
142
143 }
144
145 /* parser is responsible for creating scheduler upon command */
146 void create_scheduler(int option) {
147
148         if (option == DDL_FIGHTER) {
149                 fight_ddl();
150                 output("./output/ddl_fighter_result","Deadline Fighter
                        Algorithm","./output/ddl_fighter_timetable");
151         }
152         else if (option == RR) {
153                 RR_invoker(events, event_counter, 1, period_start_date,
                        period_end_date, period_start_time, period_end_time);
154                 output("./output/RR_result","Round Robin","./output/
                        RR_timetable");
155         }
156         else if (option == PR) {
157         PR_invoker(events, event_counter, period_start_date,
                period_end_date, period_start_time, period_end_time);
158         output("./output/PR_result","Priority","./output/PR_timetable");
159         }
160         else if (option == ALL) {
161                 RR_invoker(events, event_counter, 1, period_start_date,
                        period_end_date, period_start_time, period_end_time);
162                 output("./output/RR_result","Round Robin","./timetable/
```

```
                    RR_timetable");
163                 PR_invoker(events, event_counter, period_start_date,
                        period_end_date, period_start_time, period_end_time);
164         output("./output/PR_result","Priority","./timetable/PR_timetable
                ");
165                 fight_ddl();
166                 output("./output/ddl_fighter_result","Deadline Fighter
                        Algorithm","./timetable/ddl_fighter_timetable");
167         }
168
169
170         wait(NULL);
171 }
```

RR.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "parser.h"
4  #include "RR.h"
5
6  /*
7  Author: LIU Fengming
8  Student ID: 145104126D
9  Email: 15104126D@connect.polyu.hk
10
11 Notes:
12          (1) cur_time is of form YYYYMMDDhh, which exactly records the
                time slot of concern of each iteration in the scheduling
13                  the valid valus of hh are (start_time, start_time+1,
                        ..., end_time-1)
14          (2) Method to deal with the deadline of Project and Activity:
15                          <1> When the finishing time of the current time
                                quantum exceeds the deadline,
16                                  the time slots of the day of deadline
                                        are all depleted and the current time
                                         is tuned to the starting time of the
                                         day next the day of deadline
17                          <2> the rest of the duration is also maintained
                                to ensure consistency
18
19 */
20
21 int print_slots_alloc(struct Event* head, int cur_time, int
       slots_elapsed, int start_time, int end_time, FILE* sch_result) {
22          int i, temp=cur_time;
```

```
23          for (i=0;i<slots_elapsed;i++) {
24                  if (temp%100>=end_time) { // deal with overflow
25                          temp = (temp/100 + 1)*100 + start_time;
26                  }
27                  fprintf(sch_result, "%d %d %d %s %d %d\n", temp/100,
                        temp%100, head->id, head->name, head->type, head->
                        duration);
28                  temp++;
29          }
30          return temp;
31  }
32
33  void Round_Robin(int q, struct Event* head, struct Event* tail, int
        start_date, int end_date, int start_time, int end_time, FILE*
        sch_result, FILE* log_file, FILE* summary, int total_requests, int
        pro_ass_count) {
34          int cur_time = start_date*100 + start_time, situation=0,
                slots_elapsed=0, accepted_events=0, total_slots=0, flag=0;
35          struct Event* temp=NULL;
36          char* operations[] = {"addProject", "addAssignment", "
                addRevision", "addActivity"};
37
38          while (tail!=NULL && cur_time<(end_date*100+end_time)) {
39                  situation = 0;
40
41                  /* Revision or Activity */
42                  if (head->type==2 || head->type==3) {
43                          if (cur_time == head->date*100 + head->time) {
                                // it's the right time
44                                  if (end_time - cur_time%100 >= head->
                                        duration) { // the rest of the day is
                                         sufficient for the Event, Accept
45                                          cur_time = print_slots_alloc(
                                                head, cur_time, head->
                                                duration, start_time,
                                                end_time, sch_result);
46                                          accepted_events++;
47                                          total_slots = total_slots + head
                                                ->duration;
48                                          fprintf(log_file, "%d %s %s %d-%
                                                d-%d %d:00 %d    Accepted
                                                100.0%%\n", head->id,
                                                operations[head->type], head
                                                ->name, head->date/10000, (
                                                head->date/100)%100, head->
                                                date%100, head->time, head->
```

```
                                        duration);
49                                  } else{ // the rest of the day is not
                                        sufficient of the Event, Reject
50                                      fprintf(log_file, "%d %s %s %d-%
                                            d-%d %d:00 %d    Rejected
                                            0.0%%\n", head->id,
                                            operations[head->type], head
                                            ->name, head->date/10000, (
                                            head->date/100)%100, head->
                                            date%100, head->time, head->
                                            duration);
51                                  }
52                                  situation = 1;
53                              } else if (cur_time > head->date*100 + head->
                                    time) { // the right time has passed, Reject
54                                  fprintf(log_file, "%d %s %s %d-%d-%d %d
                                        :00 %d    Rejected 0.0%%\n", head->id
                                        , operations[head->type], head->name,
                                         head->date/10000, (head->date/100)
                                        %100, head->date%100, head->time,
                                        head->duration);
55                                  situation = 1;
56                              } else { // the right time is in the future,
                                    return the Event back to the queue for the
                                    future
57                                  if (pro_ass_count==0) { // only
                                        Revisions and Activities are left in
                                        the queue
58                                      cur_time++; // push the time for
                                            one hour
59                                      if (cur_time%100>=end_time) { //
                                            deal with overflow
60                                          cur_time = (cur_time/100
                                                + 1)*100 +
                                               start_time;
61                                      }
62
63                                  } else { // round the Event to the back
                                        of the queue and continue RR
64                                      tail->next = head;
65                                      tail = tail->next;
66                                      head = head->next;
67                                      tail->next = NULL;
68                                  }
69                              }
70                  }
```

```
71
72                  /* Project or Assignment */
73                  else {
74                          head->rest_t = head->rest_t - q;
75                          cur_time = cur_time + q; // try to do
76
77                          if (cur_time/100 > head->date) { // the Event
                                occupies time slots beyond the deadline
78                                  slots_elapsed = q - (cur_time%100 -
                                        start_time); // get the valid part of
                                         the quantum
79                                  head->percent = head->percent + (float)
                                        slots_elapsed/head->duration; //
                                        maintain the percent
80                                  head->rest_t = head->rest_t + q -
                                        slots_elapsed; // remove the
                                        excessive part of completion
81                                  cur_time = print_slots_alloc(head,
                                        cur_time-q, slots_elapsed, start_time
                                        , end_time, sch_result);
82                                  accepted_events++;
83                                  pro_ass_count--;
84                                  total_slots = total_slots +
                                        slots_elapsed;
85                                  fprintf(log_file, "%d %s %s %d-%d-%d %d
                                                Accepted %.1f%%\n", head->id
                                        , operations[head->type], head->name,
                                         head->date/10000, (head->date/100)
                                        %100, head->date%100, head->duration,
                                         head->percent*100);
86                                  situation = 1;
87                          } else if (head->rest_t>0) { // the Event
                                consumes the allocated quantum and the
                                quantum is within the deadline, but the Event
                                 has not been completed yet
88                                  cur_time = print_slots_alloc(head,
                                        cur_time-q, q, start_time, end_time,
                                        sch_result);
89                                  total_slots = total_slots + q;
90                                  head->percent = head->percent + (float)q
                                        /head->duration;
91                                  tail->next = head;
92                                  tail = tail->next;
93                                  head = head->next;
94                                  tail->next = NULL;
```

```
95                          } else { // the Event has been completed
96                                  cur_time = print_slots_alloc(head,
                                        cur_time-q, q+head->rest_t,
                                        start_time, end_time, sch_result);
97                                  accepted_events++;
98                                  pro_ass_count--;
99                                  total_slots = total_slots + q + head->
                                        rest_t;
100                                 fprintf(log_file, "%d %s %s %d-%d-%d %d
                                                Accepted 100.0%%\n", head->
                                        id, operations[head->type], head->
                                        name, head->date/10000, (head->date
                                        /100)%100, head->date%100, head->
                                        duration);
101                                 situation = 1;
102                         }
103                 }
104
105                 /* Throw away the rejected or finished Event */
106                 if (situation==1) { // the current event should be
                        removed from the queue
107                         if (head->next==NULL) { // the Event is the last
                                one
108                                 head = NULL;
109                                 fprintf(summary, "\nNumber of requests
                                        accepted: %d\n", accepted_events);
110                                 fprintf(summary, "Number of requests
                                        rejected: %d\n", total_requests-
                                        accepted_events);
111                                 fprintf(summary, "Number of time slots
                                        used: %d\n", total_slots);
112                                 return;
113                         } else {
114                                 head = head->next;
115                         }
116                 }
117         }
118
119         /* Clear the remaining rejected events */
120         while (head!=NULL) {
121                 fprintf(log_file, "%d %s %s %d-%d-%d", head->id,
                        operations[head->type], head->name, head->date/10000,
                         (head->date/100)%100, head->date%100);
122                 if (head->type==2 || head->type==3) {
123                         fprintf(log_file, " %d:00 %d    Rejected 0.0%%\n
                                ", head->time, head->duration);
```

```
124                 } else {
125                         if (head->percent>0) {
126                                 accepted_events++;
127                                 fprintf(log_file, " %d          Accepted
                                        %.1f%%\n", head->duration, head->
                                    percent*100);
128                         } else {
129                                 fprintf(log_file, " %d          Rejected
                                        0.0%%\n", head->duration);
130                         }
131                 }
132
133                 head = head->next;
134         }
135
136         fprintf(summary, "\nNumber of requests accepted: %d\n",
               accepted_events);
137         fprintf(summary, "Number of requests rejected: %d\n",
               total_requests-accepted_events);
138         fprintf(summary, "Number of time slots used: %d\n", total_slots)
               ;
139 }
140
141 void RR_invoker(struct Event events[1000], int event_counter, int q, int
        period_start_date, int period_end_date, int period_start_time, int
    period_end_time) {
142         struct Event* head = NULL;
143         struct Event* tail = NULL;
144         int i = 0, pro_ass_count = 0;
145         FILE *sch_result = fopen("./output/RR_result", "w"), *log_file =
                fopen("./output/S3_RR.log", "w"), *summary = fopen("./output
               /S3_report_RR.dat", "w");
146
147         head = &events[1];
148         for (i=1;i<=event_counter;i++) {
149                 if (i<event_counter) {
150                         events[i].next = &events[i+1];
151                 } else {
152                         events[event_counter].next = NULL;
153                 }
154                 events[i].rest_t = events[i].duration;
155                 if (events[i].type==0 || events[i].type==1) {
156                         pro_ass_count++;
157                 }
158         }
159         tail = &events[event_counter];
```

```
160
161         fprintf(log_file, "***Log File - Round Robin***\n");
162         fprintf(log_file, "ID Event                              Accepted/
               Rejected\n");
163         fprintf(log_file, "
               ================================================\n");
164         fprintf(summary, "***Summary Report***\n");
165         fprintf(summary, "\nAlgorithm used: Round Robin\n");
166         fprintf(summary, "\nThere are %d requests\n", event_counter);
167
168         Round_Robin(q, head, tail, period_start_date, period_end_date,
               period_start_time, period_end_time, sch_result, log_file,
               summary, event_counter, pro_ass_count);
169
170         fprintf(log_file, "\n
               ================================================\n");
171         for (i=1;i<=event_counter;i++) {
172                 if (events[i].date<period_start_date || events[i].date>
                    period_end_date) {
173                         fprintf(log_file, "Event (id:%d, name:%s, type:%
                            d) has an error\n", events[i].id, events[i].
                            name, events[i].type);
174                 } else {
175                         if (events[i].type==2 || events[i].type==3) {
176                                 if (events[i].time<period_start_time ||
                                    events[i].time>=period_end_time ||
                                    events[i].duration>(period_end_time-
                                    events[i].time)) {
177                                         fprintf(log_file, "Event (id:%d,
                                             name:%s, type:%d) has an
                                            error\n", events[i].id,
                                            events[i].name, events[i].
                                            type);
178                                 }
179                         }
180                 }
181         }
182
183         fclose(sch_result);
184         fclose(log_file);
185         fclose(summary);
186 }
```

PR.c

```
1 /*
```

```
 * Author: DING Dashan 17082316d
 * Date: 2019/3/30
 */
#include "PR.h"

// Priority: Project > Assignment > Revision > Activity

struct Event * Sort_By_Priority(struct Event* head, int length);

void Priority(struct Event* head, int start_date, int end_date, int
    start_time, int end_time,int length, FILE* sch_result, FILE* log_file
    , FILE* summary){
     /*No exemption for this version*/
     int cur_time = start_date*100 + start_time;
     int total_slots = (end_time-start_time)*(end_date-start_date+1);
     int slot, accept = 0;
     int *slots = (int *)malloc(total_slots * (sizeof(int)));
     for (int i = 0; i < total_slots; i++){
         slots[i] = 0;
     }
     char* operations[] = {"addProject", "addAssignment", "addRevision",
        "addActivity"};
     struct Event* cur = head;
     while (cur_time<(end_date*100+end_time) && cur!=NULL){
         /* Revision or Activity */
         if (cur->type == 2 || cur->type == 3){
             //if (cur_time <= cur->date*100 + cur->time) { // the date
                and time > current time
              if (cur->time + cur->duration > end_time || cur->date >
                 end_date || cur->date < start_date) { // the Revision
                  or Activity can not be finished in one go at the
                 current day
                                     // printf("Event (id: %d, name:
                                        %s, type: %d) has been
                                        rejected\n", cur->id, cur->
                                        name, cur->type);
                  fprintf(log_file, "%d %s %s %d-%d-%d %d:00 %d
                     REJECTED 0.0%%\n", cur->id, operations[cur->type
                     ], cur->name, cur->date/10000, (cur->date/100)
                     %100, cur->date%100, cur->time, cur->duration);
                              }
                              else {
                  int ifreject = 0;
                  slot = cur->time - start_time + 4 * (cur->date -
                     start_date);
                  for (int i = slot; i < slot+cur->duration; i++){
```

```
34                          if (slots[i] == 1){
35                              ifreject = 1;
36                          }
37                      }
38                      if (ifreject == 1){
39                          // printf("Event (id: %d, name: %s, type: %d)
                               has been rejected\n", cur->id, cur->name, cur
                               ->type);
40                          fprintf(log_file, "%d %s %s %d-%d-%d %d:00 %d
                                 REJECTED 0.0%%\n", cur->id, operations[cur
                               ->type], cur->name, cur->date/10000, (cur->
                               date/100)%100, cur->date%100, cur->time, cur
                               ->duration);
41                      }
42                      else{
43                          cur_time = cur->date * 100 + cur->time;
44                          slot = cur->time - start_time + 4 * (cur->date -
                                start_date);
45                          for (int i = slot; i < slot+cur->duration; i++){
46                              slots[i] = 1;
47                          }
48                          // printf("Event (id: %d, name: %s, type: %d)
                               has been accepted\n", cur->id, cur->name, cur
                               ->type);
49                          fprintf(log_file, "%d %s %s %d-%d-%d %d:00 %d
                                 ACCEPTED 100.0%%\n", cur->id, operations[
                               cur->type], cur->name, cur->date/10000, (cur
                               ->date/100)%100, cur->date%100, cur->time,
                               cur->duration);
50                          accept++;
51                          for (int i = 0; i < cur->duration; i++){
52                              fprintf(sch_result, "%d %d %d %s %d %d\n",
                                    cur_time/100, cur_time%100, cur->id, cur
                                    ->name, cur->type, cur->duration);
53                              cur_time++;
54                          }
55                      }
56                  }
57              /*}
58              else {
59                              // printf("Event (id: %d, name: %s, type
                                   : %d) has been rejected\n", cur->id,
                                   cur->name, cur->type);
60                              fprintf(log_file, "%d %s %s %d-%d-%d %d
                                   :00 %d    REJECTED 0.0%%\n", cur->id,
                                    operations[cur->type], cur->name,
```

```
                                    cur->date/10000, (cur->date/100)%100,
                                     cur->date%100, cur->time, cur->
                                    duration);
61                      }*/
62          }
63          else {
64              // See if the remaining time is enough for it
65              int cur_date = cur_time/100;
66              int rem_date = end_date-cur_date;
67              int today_end = cur_date*100 + 23;
68              int rem_time = rem_date*(end_time - start_time) + today_end
                    - cur_time;
69              int time_to_ddl = (cur->date - cur_date)*(end_time -
                    start_time) + today_end - cur_time;
70              if (cur->date < cur_date) {
71                  fprintf(log_file, "%d %s %s %d-%d-%d %d
                        REJECTED 0.0%%\n", cur->id, operations[cur->type],
                        cur->name, cur->date/10000, (cur->date/100)%100, cur
                        ->date%100, cur->duration);
72              }
73              else {
74                  if (rem_time >= cur->duration){
75                      if (time_to_ddl >= cur->duration) {
76                          //cur_time = cur_time + 100*(cur->rest_t/(
                                end_time-start_time)) + (start_time + cur->
                                rest_t%(end_time-start_time));
77                          // printf("Event (id: %d, name: %s, type: %d)
                                has been accepted and has completed\n", cur->
                                id, cur->name, cur->type);
78                          fprintf(log_file, "%d %s %s %d-%d-%d %d
                                        ACCEPTED 100.0%%\n", cur->id,
                                operations[cur->type], cur->name, cur->date
                                /10000, (cur->date/100)%100, cur->date%100,
                                cur->duration);
79                          accept++;
80                          // the Event has been completed
81                          slot = cur_time % cur_date - start_time + 4 * (
                                cur_date - start_date);
82                          for (int i = 0; i < cur->duration; i++){
83                              fprintf(sch_result, "%d %d %d %s %d %d\n",
                                    cur_time/100, cur_time%100, cur->id, cur
                                    ->name, cur->type, cur->duration);
84                              cur_time++;
85                              if (cur_time%100 >= end_time){
86                                  cur_time += 100;
87                                  cur_time = cur_time/100 * 100 +
```

```
                                start_time;
88                          }
89                      }
90                      //cur_time += cur->duration;
91
92                      for (int i = slot; i < slot+cur->duration; i++){
93                          slots[i] = 1;
94                      }
95                      /*int overflow = (cur_time % cur_date) -
                            end_time;
96                      if (overflow > 0){
97                          int day_spent = overflow / (end_time -
                                start_time);
98                          int remainder = overflow % (end_time -
                                start_time);
99                          cur_time = (cur_date + 1 + day_spent) * 100
                                + start_time + remainder;
100                     }*/
101                 }
102                 else { // it fails to finish before the ddl
103                     cur->percent = (float)time_to_ddl/(float)cur->
                            duration * 100;
104                     // printf("Event (id: %d, name: %s, type: %d)
                            has been accepted and only finished %f%%\n",
                            cur->id, cur->name, cur->type, cur->percent);
105                     fprintf(log_file, "%d %s %s %d-%d-%d %d
                                    ACCEPTED %.1f%%\n", cur->id,
                            operations[cur->type], cur->name, cur->date
                            /10000, (cur->date/100)%100, cur->date%100,
                            cur->duration, cur->percent);
106                     accept++;
107                     //cur_time = cur->date * 100 + 100 + start_time;
108                     for (int i = 0; i < time_to_ddl; i++){
109                         fprintf(sch_result, "%d %d %d %s %d %d\n",
                                cur_time/100, cur_time%100, cur->id, cur
                                ->name, cur->type, cur->duration);
110                         cur_time++;
111                         if (cur_time%100 >= end_time){
112                             cur_time += 100;
113                             cur_time = cur_time/100 * 100 +
                                    start_time;
114                         }
115                     }
116                 }
117             }
118             else
```

```
119                 { // it fails to finish before end date
120                     cur->percent = (float)rem_time/(float)cur->duration
                            * 100;
121                     // printf("Event (id: %d, name: %s, type: %d) has
                            been accepted but has not completed\n", cur->id,
                            cur->name, cur->type);
122                     fprintf(log_file, "%d %s %s %d-%d-%d %d
                            ACCEPTED %.1f%%\n", cur->id, operations[cur->type
                            ], cur->name, cur->date/10000, (cur->date/100)
                            %100, cur->date%100, cur->duration, cur->percent)
                            ;
123                     for (int i = 0; i < rem_time; i++){
124                         fprintf(sch_result, "%d %d %d %s %d %d\n",
                                cur_time/100, cur_time%100, cur->id, cur->
                                name, cur->type, cur->duration);
125                         cur_time++;
126                         if (cur_time%100 >= end_time){
127                             cur_time += 100;
128                             cur_time = cur_time/100 * 100 + start_time;
129                         }
130                     }
131                     accept++;
132                     slot = cur_time % cur_date - start_time + 4 * (
                            cur_date - start_date);
133                 }
134             }
135                 }
136         if (cur->next == NULL) {
137             cur = NULL;
138             break;
139         }
140         else {
141             cur = cur->next;
142         }
143     }
144     int slots_used = 0;
145     for (int i = 0; i < total_slots; i++){
146         slots_used += slots[i];
147     }
148     fprintf(summary, "\nNumber of requests accepted: %d\n", accept);
149     fprintf(summary, "Number of requests rejected: %d\n", length-accept)
            ;
150         fprintf(summary, "Number of time slots used: %d\n", slots_used);
151      /* Clear the remaining rejected events */
152         while (cur!=NULL) {
153                 // printf("Event (id: %d, name: %s, type: %d) has been
```

```
                    rejected\n", cur->id, cur->name, cur->type);
154                 fprintf(log_file, "%d %s %s %d-%d-%d", cur->id,
                        operations[cur->type], cur->name, cur->date/10000, (
                        cur->date/100)%100, cur->date%100);
155                 if (cur->type == 2 || cur->type == 3){
156             fprintf(log_file, " %d:00 %d    REJECTED 0.0%%\n", cur->time
                    , cur->duration);
157                 }
158                 else {
159                         fprintf(log_file, " %d          REJECTED 0.0%%\n
                            ", cur->duration);
160                 }
161                 cur = cur->next;
162         }
163     return;
164 }
165
166 struct Event * Sort_By_Priority(struct Event* head, int length){
167     // We duplicate the linked list, or else the original will change
168     if (length == 0 || length == 1 || head == NULL) return head;
169     struct Event* current, *lastNode=NULL, *newHead;
170     for (int i = 0; i < 4; i++){
171         current = head;
172         for (int j = 0; j < length; j++){
173             if (current->type == i){
174                 struct Event *newNode = (struct Event *)malloc(sizeof(
                        struct Event));
175                 memcpy(newNode, current, sizeof(struct Event));
176                 newNode->next = NULL;
177                 if (lastNode != NULL){
178                     lastNode->next = newNode;
179                 }
180                 else{
181                     newHead = newNode;
182                 }
183                 lastNode = newNode;
184             }
185             current = current->next;
186         }
187     }
188     return newHead;
189 }
190
191 void PR_invoker(struct Event events[1000], int length, int
    period_start_date, int period_end_date, int period_start_time, int
    period_end_time){
```

```
192     struct Event* head = NULL;
193         FILE *sch_result = fopen("./output/PR_result", "w"), *log_file =
                fopen("./output/S3_PR.log", "w"), *summary = fopen("./output
               /S3_report_PR.dat", "w");
194         head = &events[1];
195         for (int i = 1; i <= length; i++) {
196                 if (i < length) {
197                         events[i].next = &events[i+1];
198                 }
199                 else {
200                         events[length].next = NULL;
201                 }
202                 events[i].rest_t = events[i].duration;
203         }
204         fprintf(log_file, "***Log File - Priority***\n");
205         fprintf(log_file, "ID    Name
                                                  Status\n");
206         fprintf(log_file, "
               ======================================================\n");
207         fprintf(summary, "***Summary Report***\n");
208         fprintf(summary, "\nAlgorithm used: Priority\n");
209         fprintf(summary, "\nThere are %d requests\n", length);
210         head = Sort_By_Priority(head, length);
211         Priority(head, period_start_date, period_end_date,
               period_start_time, period_end_time, length, sch_result,
               log_file, summary);
212         fprintf(log_file, "\n
               ======================================================\n");
213         fprintf(log_file,"Errors (if any):\n");
214     int i;
215         for (i = 1; i <= length; i++) {
216                 if (events[i].date < period_start_date || events[i].date
                        > period_end_date) {
217                         fprintf(log_file, "Event #%d contains an error\n
                               ", events[i].id);
218                 } else {
219                         if (events[i].type == 2 || events[i].type == 3)
                               {
220                                 if (events[i].time < period_start_time
                                       || events[i].time >= period_end_time
                                       || events[i].duration > (
                                       period_end_time-events[i].time)) {
221                                         fprintf(log_file, "Event #%d
                                               contains an error\n", events[
                                               i].id);
222                                 }
```

```
                        }
                }
        }
    fclose(sch_result);
        fclose(log_file);
        fclose(summary);
        return;
}
```

ddl_fighter.c

```c
/*
 * Author: LIU Le 15103617d
 * Date: 2019/3/27
 */

// header files
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <stdbool.h>
#include <sys/wait.h>
#include <string.h>
#include <math.h>

// my headers
#include "ddl_fighter.h"

// macros
#define REJECT 0
#define OVERLAP 1
#define NO_OVERLAP 2

// global variables
int* schedule;
int total_hours;
int rejected[1000];
int number_of_reject = 0;

// prototypes
int get_ddl(struct Event e);
void generate_summary();
void generate_log();
void generate_intermediate_timetable();
void generate_report();
```

```
36
37 int compareTo(const void* a, const void* b) {
38         struct Event* structA = (struct Event*) a;
39         struct Event* structB = (struct Event*) b;
40         return (structA->unit_benefit >= structB->unit_benefit) ? -1 :
               1;
41 }
42
43 int get_date_difference(int date1, int date2) {
44         int month1 = (date1 % 10000) / 100;
45         int month2 = (date2 % 10000) / 100;
46         int day1 = date1 % 100;
47         int day2 = date2 % 100;
48         return (month2 - month1 - 1) * 30 + (30 - day1 + 1) + day2;
49 }
50
51 /* return the number of hours between two time points */
52 int get_hour_diff(int date1, int date2, int time1, int time2) {
53         int hours = (get_date_difference(date1, date2) - 1) *
               HOUR_PER_DAY;
54         hours = hours - (time1- DAY_START) + time2 - DAY_START;
55         return hours;
56 }
57
58 /* Test if an event is within the time bound */
59 bool is_legal(struct Event e) {
60         if (e.type == REVISION_TYPE || e.type == ACTIVITY_TYPE) {
61                 if (e.time < DAY_START || (e.time + e.duration) >
                       DAY_END)
62                 return false;
63                 int ddl = get_ddl(e);
64                 if ( ddl >= total_hours || (ddl - e.duration) < 0)
65                         return false;
66         }
67         else {
68                 if (e.date > period_end_date || e.date <
                       period_start_date)
69                         return false;
70         }
71         return true;
72 }
73
74 /* Get the corresponding hour in schedule[] the event ends */
75 int get_ddl(struct Event e) {
76         if (e.type == ASSIGNMENT_TYPE || e.type == PROJECT_TYPE)
77                 return get_hour_diff(period_start_date,e.date,
```

```
                    period_start_time,DAY_END) - 1;
78          else
79                  return get_hour_diff(period_start_date,e.date,
                        period_start_time,e.time + e.duration) - 1;
80  }
81
82  /* initialize stuff */
83  void init() {
84          for (int i = 1; i <= event_counter; ++i)
85          {
86                  events[i].status = ACCEPTED;
87                  int level = parse_level(events[i].name);
88                  float benefit = 0;
89                  if (events[i].type == PROJECT_TYPE)
90                          benefit = PROJECT_BASE + (level - 1) *
                                LEVEL_UP_POINT;
91                  else if (events[i].type == ASSIGNMENT_TYPE)
92                          benefit = ASSIGNMENT_BASE + (level - 1) *
                                LEVEL_UP_POINT;
93                  else if (events[i].type == REVISION_TYPE)
94                          benefit = REVISION_BASE + (level - 1) *
                                LEVEL_UP_POINT;
95                  else
96                          benefit = ACTIVITY_BASE;
97                  events[i].rest_t = events[i].duration;
98                  events[i].unit_benefit = benefit / events[i].duration;
99          }
100         total_hours = get_hour_diff(period_start_date,period_end_date,
                period_start_time,period_end_time);
101         schedule = (int*) malloc(sizeof(int) * total_hours);
102         for (int i = 0; i < total_hours; ++i)
103                 schedule[i] = 0;
104 }
105
106 /* check if the revision/activity has overlaps with already scheduled
        events */
107 int has_overlap(struct Event e) {
108         int ddl = get_ddl(e);
109         bool overlap = false;
110         for (int i = 0; i < e.duration; ++i) {
111                 if (schedule[ddl-i] == 0)
112                         continue;
113                 if (events[schedule[ddl-i]].type == ACTIVITY_TYPE ||
                        events[schedule[ddl-i]].type == REVISION_TYPE)
114                         return REJECT;
115                 overlap = true;
```

```
116         }
117         if (overlap == true)
118                 return OVERLAP;
119         return NO_OVERLAP;
120 }
121
122 void swap(int* ar, int a, int b) {
123         int temp = *(ar+a);
124         *(ar+a) = *(ar+b);
125         *(ar+b) = temp;
126 }
127
128 bool has_enough_slot(int index, int duration) {
129         int count = 0;
130         while (index-- >= 0) {
131                 if (schedule[index+1] != 0)
132                         continue;
133                 /* slots occupied by activities/revision is excluded */
134                 if (events[schedule[index+1]].type == ACTIVITY_TYPE ||
                        events[schedule[index+1]].type == REVISION_TYPE)
135                         continue;
136                 if (++count == duration)
137                         return true;
138         }
139         return false;
140 }
141
142 void print_schedule() {
143         int first_day_hours = period_start_time - DAY_START;
144         for (int i = 0; i < total_hours; ++i) {
145                 if ((i + first_day_hours) % HOUR_PER_DAY == 0)
146                         printf("|<%d> ", (i + first_day_hours) /
                                HOUR_PER_DAY + 1);
147                 printf("%d ", schedule[i]);
148         }
149         printf("\n");
150 }
151
152 void print_result() {
153         print_schedule();
154         printf("Rejected: ");
155         for (int i = 0; i < number_of_reject; ++i)
156                 printf("%d ", rejected[i]);
157 }
158
159 bool is_error(struct Event e) {
```

```
160         if (e.date < period_start_date || e.date > period_end_date)
161                 return true;
162         if (e.time > DAY_END || e.time < DAY_START)
163                 return true;
164         if (e.type == ACTIVITY_TYPE || e.type == REVISION_TYPE)
165                 return !is_legal(e);
166         return false;
167 }
168
169 void fight_ddl() {
170         init();
171
172         qsort(events+1, event_counter, sizeof(events[0]), compareTo);
173
174         // printf("Deadline fighter scheduling...\n");
175         /* handle project and assignment first */
176         for (int i = 1; i <= event_counter; ++i)
177         {
178                 if (events[i].type == ACTIVITY_TYPE || events[i].type ==
                        REVISION_TYPE)
179                         continue;
180
181                 int ddl = get_ddl(events[i]);
182                 /* search forward from deadline for empty spot */
183                 while(ddl >= 0 && events[i].rest_t > 0) {
184                         if (schedule[ddl--] != 0)
185                                 continue;
186                         schedule[ddl+1] = i;
187                         (events[i].rest_t)--;
188                 }
189         }
190
191         /* then handle revision */
192         for (int i = 1; i <= event_counter; ++i)
193         {
194                 if (events[i].type != REVISION_TYPE)
195                         continue;
196                 /* if does not fit in time period, reject directly */
197                 if (!is_legal(events[i])) {
198                         rejected[number_of_reject++] = i;
199                         continue;
200                 }
201                 /* test if overlap with other event occurs */
202                 int ddl = get_ddl(events[i]);
203                 int result = has_overlap(events[i]);
204                 if (result == OVERLAP) {
```

```
205                     if (!has_enough_slot(ddl, events[i].duration))
206                             rejected[number_of_reject++] = i;
207                     else
208                     {
209                             int index = ddl;
210                             while (index-- >= 0) {
211                                     if (events[i].rest_t == 0)
212                                             break;
213                                     if (schedule[index+1] != 0)
214                                             continue;
215                                     if (events[schedule[index+1]].
                                        type == REVISION_TYPE ||
                                        events[schedule[index+1]].
                                        type == ACTIVITY_TYPE)
216                                             continue;
217                                     swap(schedule,index+1,ddl);
218                                     schedule[ddl--] = i;
219                                     (events[i].rest_t)--;
220                             }
221                     }
222             }
223             else if (result == REJECT) {
224                     rejected[number_of_reject++] = i;
225             }
226             /* no overlap, just put the activity there */
227             else {
228                     while ((events[i].rest_t)-- > 0)
229                             schedule[ddl--] = i;
230             }
231     }
232
233     /* then handle activity */
234     for (int i = 1; i <= event_counter; ++i)
235     {
236             if (events[i].type != ACTIVITY_TYPE)
237                     continue;
238             /* if does not fit in time period, reject directly */
239             if (!is_legal(events[i])) {
240                     rejected[number_of_reject++] = i;
241                     continue;
242             }
243             /* test if overlap with other event occurs */
244             int ddl = get_ddl(events[i]);
245             int result = has_overlap(events[i]);
246             if (result == OVERLAP) {
247                     if (!has_enough_slot(ddl, events[i].duration))
```

```
248                                     rejected[number_of_reject++] = i;
249                             else
250                             {
251                                     int index = ddl;
252                                     while (index-- >= 0) {
253                                             if (events[i].rest_t == 0)
254                                                     break;
255                                             if (schedule[index+1] != 0)
256                                                     continue;
257                                             if (events[schedule[index+1]].
                                                    type == REVISION_TYPE ||
                                                    events[schedule[index+1]].
                                                    type == ACTIVITY_TYPE)
258                                                     continue;
259                                             swap(schedule,index+1,ddl);
260                                             schedule[ddl--] = i;
261                                             (events[i].rest_t)--;
262                                     }
263                             }
264                     }
265                     else if (result == REJECT) {
266                             rejected[number_of_reject++] = i;
267                     }
268                     /* no overlap, just put the activity there */
269                     else {
270                             while ((events[i].rest_t)-- > 0)
271                                     schedule[ddl--] = i;
272                     }
273             }
274
275             for (int i = 0; i < number_of_reject; ++i)
276                     events[rejected[i]].status = REJECTED;
277
278             // print_result();
279             generate_summary();
280             generate_log();
281             generate_intermediate_timetable();
282
283             free(schedule);
284
285             // printf("\nScheduling Complete!\n");
286     }
287
288     void generate_log() {
289             char dict[2][20] = {"ACCEPTED","REJECTED"};
290             FILE *log = fopen("./output/S3_ddl_fighter.log", "w");
```

```
291         fprintf(log,"***** Log - Deadline Fighter *****\n\n");
292         fprintf(log, "ID    Name
                                                        Status\n");
293         fprintf(log, "
               ================================================================\
               n");
294         char temp[100];
295         for (int i = 1; i <= event_counter; ++i) {
296                 strcpy(temp,command[events[i].id]);
297                 temp[strlen(temp)-1] = 0;
298                 if (events[i].rest_t < 0)
299                         events[i].rest_t = 0;
300                 fprintf(log, "%d\t%s\t\t\t%s\t%0.1f%%\n",events[i].id,
                       temp,dict[events[i].status],(events[i].duration -
                       events[i].rest_t) * 100 / (float)(events[i].duration)
                       );
301         }
302         fprintf(log, "\n
               ================================================================\
               n");
303         fprintf(log,"Errors (if any):\n");
304         for (int i = 1; i <= event_counter; ++i)
305         {
306                 if (is_error(events[i]) == true)
307                         fprintf(log,"Event #%d contains error.\n",
                            events[i].id);
308         }
309         fclose(log);
310 }
311
312 void generate_intermediate_timetable() {
313         FILE *file = fopen("./output/ddl_fighter_result", "w");
314         int current_time = -1;
315         int current_date;
316         for (int i = 0; i < total_hours; ++i)
317         {
318                 current_date = i / HOUR_PER_DAY + period_start_date;
319                 current_time = (current_time+1) % HOUR_PER_DAY;
320                 int index = schedule[i];
321                 if (index == 0) continue;
322                 fprintf(file, "%d %d %d %s %d %d\n", current_date,
                       DAY_START + current_time,events[index].id,events[
                       index].name,events[index].type,events[index].duration
                       );
323         }
324         fclose(file);
```

```
}

void generate_summary() {
        FILE *summary = fopen("./output/S3_report_ddl_fighter.dat", "w")
            ;
        fprintf(summary, "***** Summary Report *****\n");
        fprintf(summary, "\nAlgorithm: Deadline Fighter Algorithm\n");
        fprintf(summary, "\nNumber of requests: %d\n",event_counter);
        fprintf(summary, "Number of rejected: %d, [ ", number_of_reject)
            ;
        for (int i = 0; i < number_of_reject; ++i)
                fprintf(summary, "%d ", events[rejected[i]].id);
        fprintf(summary, "]\n\n");

        int hours_used = 0;
        for (int i = 0; i < total_hours; ++i)
                if (schedule[i] != 0) hours_used++;
        fprintf(summary, "Hours used: %d/%d\n",hours_used,total_hours);
        fclose(summary);
}

/*
int main() {
}
*/
```

output.c

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

#include "parser.h"

/* prototypes */
void toDateFormat(char *newstr, char *oldstr);
void output(char *summary_file, char *algorithm, char *timetable_file);

/* convert date format, such as "20190408" to "2019-04-08" */
void toDateFormat(char *newstr, char *oldstr) {
        strncpy(newstr, oldstr, 4);
    newstr[4]='-';
    strncpy(newstr+5, oldstr+4, 2);
    newstr[7]='-';
```

```
19      strncpy(newstr+8, oldstr+6, 2);
20      newstr[10]='\0';
21  }
22
23  /* Print formatted timetable */
24  void output(char *summary_file, char *algorithm, char *timetable_file) {
25          int i, j, k;
26          int date, time;
27          char startStr1[9], endStr1[9], lastDate[9];
28          char startStr2[11], endStr2[11], dateStr[11];
29          char timeStr[6];
30          char lineStr[80], path[50];
31          char *token;
32          FILE *timetable = fopen(timetable_file, "w");
33          FILE *fp = fopen(summary_file, "r");
34          if (fp == NULL) {
35                  printf("Cannot open the file!\n");
36                  exit(1);
37          }
38
39          fprintf(timetable, "Alice Timetable\n");
40          sprintf(startStr1, "%d", period_start_date);
41          toDateFormat(startStr2, startStr1);
42          sprintf(endStr1, "%d", period_end_date);
43          toDateFormat(endStr2, endStr1);
44          fprintf(timetable, "Period: %s to %s\nAlgorithm: %s\n\n%-25s",
                startStr2, endStr2, algorithm, "Date");
45          for (i=DAY_START; i<DAY_START+HOUR_PER_DAY; i++) {
46                  sprintf(timeStr, "%d", i);
47                  strcpy(timeStr+2, ":00");
48                  fprintf(timetable, "%-20s", timeStr);
49          }
50
51          time = DAY_START;
52          fprintf(timetable, "\n%-25s", startStr2);
53          strcpy(lastDate, startStr1);
54          while(fscanf(fp, "%[^\n]\n", lineStr) != EOF) {
55                  token = strtok(lineStr, " ");
56                  k=0;
57                  if (strcmp(token,lastDate)!=0) { // different date
58                          while (time < DAY_END) {
59                                  fprintf(timetable, "%-20s", "N/A");
60                                  time++;
61                          }
62                          while ((lastDate[6]-'0')*10+(lastDate[7]-'0')+1
                               < (token[6]-'0')*10+(token[7]-'0')) {
```

```
63                                  date = atoi(lastDate)+1;
64                                  sprintf(lastDate, "%d", date);
65                                  toDateFormat(dateStr, lastDate);
66                                  fprintf(timetable, "\n%-25s%-20s%-20s
                                        %-20s%-20s", dateStr,"N/A","N/A","N/A
                                        ","N/A");
67                          }
68                          toDateFormat(dateStr, token);
69                          fprintf(timetable, "\n%-25s", dateStr);
70                          time = DAY_START;
71                  }
72                  strcpy(lastDate, token);
73                  while( token != NULL ) {
74                          token = strtok(NULL, " ");
75                          k++;
76                          switch (k) {
77                                  case 1:
78                                          while (time < atoi(token)) { //
                                                different time
79                                                  fprintf(timetable, "%-20
                                                        s", "N/A");
80                                                  time++;
81                                          }
82                                          break;
83                                  case 3:
84                                          fprintf(timetable, "%-20s",
                                                token);
85                                          time++;
86                                          break;
87                          }
88                  }
89          }
90
91          while (strcmp(endStr1,lastDate)!=0) { // until period end date
92                  while (time < DAY_END) {
93                          fprintf(timetable, "%-20s", "N/A");
94                          time++;
95                  }
96                  date = atoi(lastDate)+1;
97                  sprintf(lastDate, "%d", date);
98                  toDateFormat(dateStr, lastDate);
99                  fprintf(timetable, "\n%-25s", dateStr);
100                 time = DAY_START;
101         }
102         while (time < DAY_END) { // until DAY_END time
103                 fprintf(timetable, "%-20s", "N/A");
```

```
104                 time++;
105         }
106         fprintf(timetable, "\n");
107
108         fclose(fp);
109         fclose(timetable);
110 }
```

analyzer.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5  #include <string.h>
6  #include "parser.h"
7
8  void analyzer();
9  float scoring(char *filename);
10
11
12 void analyzer() {
13         char *pr_filename = "./output/PR_result";
14         char *rr_filename = "./output/RR_result";
15         char *ddl_filename = "./output/ddl_fighter_result";
16         FILE *fp = fopen("./output/analyzer_summary.txt", "w");
17
18         float pr_benefit = scoring(pr_filename);
19         float rr_benefit = scoring(rr_filename);
20         float ddl_benefit = scoring(ddl_filename);
21
22         fprintf(fp, "%-35s%-35s\n
               ----------------------------------------\n", "Algorithm",
               "Benefit");
23         if (pr_benefit >= 0)
24                 fprintf(fp, "%-35s%-35.2f\n", "Priority", pr_benefit);
25         if (rr_benefit >= 0)
26                 fprintf(fp, "%-35s%-35.2f\n", "Round Robin", rr_benefit)
                       ;
27         if (ddl_benefit >= 0)
28                 fprintf(fp, "%-35s%-35.2f\n", "Deadline Fighter
                       Algorithm", ddl_benefit);
29         fprintf(fp, "----------------------------------------\n");
30
31         if (pr_benefit < 0 && rr_benefit < 0 && ddl_benefit < 0) {
32                 fclose(fp);
```

```
33                  return;
34          }
35          if (ddl_benefit >= pr_benefit && ddl_benefit >= rr_benefit)
36                  fprintf(fp, "Best Algorithm: Deadline Fighter Algorithm\
                       n");
37          else if (rr_benefit >= pr_benefit && rr_benefit >= ddl_benefit)
38                  fprintf(fp, "Best Algorithm: Round Robin\n");
39          else
40                  fprintf(fp, "Best Algorithm: Priority\n");
41
42          fclose(fp);
43  }
44
45  float scoring(char *filename) {
46          FILE *fp = fopen(filename, "r");
47          if (fp == NULL)
48                  return -1.0;
49
50          int k;
51          int type, duration, level;
52          float unit_benefit;
53          float benefit = 0;
54          char name[30];
55          char lineStr[80];
56          char *token;
57          while(fscanf(fp, "%[^\n]\n", lineStr) != EOF) {
58                  token = strtok(lineStr, " ");
59                  k=0;
60                  while( token != NULL ) {
61                          token = strtok(NULL, " ");
62                          k++;
63                          switch (k) {
64                                  case 3: //name
65                                          strcpy(name, token);
66                                          break;
67                                  case 4: //type
68                                          type = atoi(token);
69                                          break;
70                                  case 5: //duration
71                                          duration = atoi(token);
72                                          break;
73                          }
74                  }
75                  level = parse_level(name);
76
77                  if (type == PROJECT_TYPE)
```

```
78                          unit_benefit = PROJECT_BASE + (level - 1) *
                               LEVEL_UP_POINT;
79                  else if (type == ASSIGNMENT_TYPE)
80                          unit_benefit = ASSIGNMENT_BASE + (level - 1) *
                               LEVEL_UP_POINT;
81                  else if (type == REVISION_TYPE)
82                          unit_benefit = REVISION_BASE + (level - 1) *
                               LEVEL_UP_POINT;
83                  else
84                          unit_benefit = ACTIVITY_BASE;
85                  unit_benefit /= duration;
86                  benefit += unit_benefit;
87          }
88
89          fclose(fp);
90          return benefit;
91  }
92  }
```

Header Files

s3.h

```
1
2   #ifndef S3_H
3   #define S3_H
4
5   #define CHILD_NUM 1 //number of child
6   #define BUF_SIZE 100 //length of a buf
7
8   extern int fd_toC[CHILD_NUM][2];
9   extern int fd_toP[CHILD_NUM][2];
10
11  #include <stdio.h>
12  #include <stdlib.h>
13  #include <unistd.h>
14  #include <sys/wait.h>
15  #include <string.h>
16  #include <regex.h>
17
18  // our headers
19  #include "parser.h"
20  #include "s3.h"
21
22  void getInput(char *instr);
23  void cmdToChild(int fd_toC[][2], char *instr);
24  void toChild(int fd_toC[][2], char *instr);
```

```
25 void test(int fd_toC[][2], int i);//to be deleted
26 char report_filename[100];
27 void analyzer();
28 float scoring(char *filename);
29
30 #endif
```

parser.h

```
1
2  #ifndef PARSER_H
3  #define PARSER_H
4
5  // macros
6  #define PROJECT_TYPE 0
7  #define ASSIGNMENT_TYPE 1
8  #define REVISION_TYPE 2
9  #define ACTIVITY_TYPE 3
10 #define DDL_FIGHTER 0
11 #define RR 1
12 #define PR 2
13 #define ALL 3
14
15
16 /*
17 Project base point: 30
18 Assignment base point: 20
19 Revision base point: 10
20 Activity base point: 20
21 Each level up add 3
22 */
23
24 #define PROJECT_BASE 30
25 #define ASSIGNMENT_BASE 20
26 #define REVISION_BASE 15
27 #define ACTIVITY_BASE 5
28 #define LEVEL_UP_POINT 3
29 #define DAY_START 19
30 #define DAY_END 23
31 #define HOUR_PER_DAY (DAY_END - DAY_START)
32
33 // structs
34 struct Event {
35         int id;
36         int type; // 0: Project, 1: Assignment, 2: Revision, 3: Activity
```

```
        char name[30];
        int date; // format: YYYYMMDD
        int time; // format: hh (0<=hh<=23), -1 represents invalid
        int ddl;
        int duration;
        int rest_t; // the remaining hours
        float percent; // -1 represents in valid
        float unit_benefit;
        int status;
        struct Event* next;
};

// shared variables
extern struct Event events[1000]; // support at most 1000 events
extern int event_counter;
extern char command[1000][200];
extern int period_start_date;
extern int period_end_date;
extern int period_start_time;
extern int period_end_time;
extern char report_filename[100];

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <stdbool.h>
#include <sys/wait.h>
#include <string.h>
#include "ddl_fighter.h"
#include "RR.h"
#include "PR.h"

// prototypes
void parse();
void print_event();
void create_scheduler(int option);
int parse_level(char* name);
bool is_digit(char a);
void output(char *summary_file, char *algorithm, char *timetable_file);

#endif
```

RR.h

```
#ifndef RR_H
```

```
2  #define RR_H
3  #include "parser.h"
4  #include "ddl_fighter.h"
5
6  int print_slots_alloc(struct Event* head, int cur_time, int
       slots_elapsed, int start_time, int end_time, FILE* sch_result);
7  void Round_Robin(int q, struct Event* head, struct Event* tail, int
       start_date, int end_date, int start_time, int end_time, FILE*
       sch_result, FILE* log_file, FILE* summary, int total_requests, int
       pro_ass_count);
8  void RR_invoker(struct Event events[1000], int event_counter, int q, int
        period_start_date, int period_end_date, int period_start_time, int
       period_end_time);
9
10 #endif
```

PR.h

```
1  #ifndef PR_H
2  #define PR_H
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <unistd.h>
6  #include <string.h>
7  #include "parser.h"
8  void Priority(struct Event* head,  int start_date, int end_date, int
       start_time, int end_time,int length, FILE* sch_result, FILE* log_file
       , FILE* summary);
9  void PR_invoker(struct Event events[1000], int length, int
       period_start_date, int period_end_date, int period_start_time, int
       period_end_time);
10
11 #endif
```

ddl_fighter.h

```
1  #ifndef DDL_FIGHTER_H
2  #define DDL_FIGHTER_H
3
4  // my headers
5  #include "parser.h"
6
7  #define ACCEPTED 0
8  #define REJECTED 1
9
10 void fight_ddl();
11 bool is_error(struct Event e);
```

```
12
13 #endif
```

References

[1] Np-completeness. [Online]. Available: https://en.wikipedia.org/wiki/NP-completeness

[2] Knapsack problem. [Online]. Available: https://en.wikipedia.org/wiki/Knapsack_problem