COMP2432 Group Project (2022/2023 Semester 2)

Submission deadline:

First Stage: *March 3, 2023*

Second Stage: *March 31, 2023*

Weighting: 15%

Project title: **AP**pointment **O**rganizer (**APO**)

Scenario

Nowadays, people are always busy. There are dates, meetings, gatherings and many other events and activities that have to be made and arranged. In order to avoid conflicts, people need a good assistant application to record and arrange those appointments, e.g. using PDA or phone organizer. In the Department of Computing, students John, Lucy, Mary and Paul are good friends and have formed a study group. They always have gatherings and other activities on the campus. However, due to the condensed classes and time tables in this semester, the gatherings cannot be arranged properly in most of the time. They have to check with one another for the availability of timeslots when they want to confirm with an appointment. It is very time−consuming and inefficient to make appointments in this "manual" approach. Knowing that you have learnt process scheduling from the subject COMP2432; maybe you can give them a hand to simplify their appointment bookings and organization problems.

Furthermore, in future, you might want to further develop this APO and deploy within their mobile phones to carry out such scheduling and organizing functionalities, if you are interested.

Project Requirements

In this project, you will be given an opportunity to apply the theory of process scheduling you have learnt from COMP2432 to a daily-life scenario and produce the **AP**pointment **O**rganizer (**APO**). In general, **APO** may be designed with different modules with different functionalities. As a default arrangement, this project may be developed in five stages in the form of five parts (*or up to six parts*):

Part 1. Write a program that allows users to add details of appointment (*date, time, duration, and/or callees*) to the scheduler. Note that the one who initiates for the appointment is called the "**user**" or the "**caller**" and those other members involved in the appointment are called "**callees**", making use of the terms in procedure call within a program. This program is referred to as the **Input Module**. Bear in mind that this application should allow batch input, i.e. reading in from files for data entries. It is recommended to store all meeting requests in a log file for future audit purpose (and in your case, debugging purpose). By the way, the application is developed to be

used internally for the time being. If the result is satisfactory, you may consider promoting it to the public to use the application, e.g. publish it under Google Play.

Part 2. **APO** checks whether the timeslot(s) of the involved parties are available or reserved. That is, not only the user's timeslot(s), but also those of the callees', if any, would be checked[1]. **APO** sends *messages* to all callees to check for their availability and callees should reply the *message* no matter the timeslot(s) are available or not. If all parties are available, the appointment should be then confirmed[2]. Otherwise, the appointment would be rejected or rescheduled[3]. For example, consider the scenario that Paul calls for a meeting with Lucy and Mary. **APO** checks not only for Paul's schedule but also the schedules of Lucy and Mary. The appointment would be accepted and confirmed only when all of them are available. Otherwise, the appointment would be rejected or rescheduled. Here, "rescheduling" is a bonus option. This appointment information should be reflected within the time table generated (see **`printSchd`** below under **Implementation** section) by all the users involved.

Part 3. Extend the program developed in *Part 2* with a scheduler to generate a timetable for all appointments (i.e. timeslots engaged or available). The scheduler might implement several scheduling algorithms similar to those covered in lecture 6. (*In this project, you are asked to implement **at least two**. One of them should be the relatively straightforward **F**irst **C**ome **F**irst **S**erved*.) This is called the **Scheduling Kernel**, within the **Scheduling Module**. That is the module to include all the scheduling algorithms that you have implemented.

Part 4. Augment the program with the facilities to print appointment schedule for users in *Part 2*. Those rejected appointments should all be included and recorded in an output file, named "**`rejected.dat`**". This constitutes the **Output Module**.

---

[1] *It is assumed that all callees would accept the appointment if there is timeslot available according to his/her current schedule.*

[2] *For implementation, actual human users are not involved. The system will answer the requests automatically, i.e. APO as the parent process is to check the availability of caller and callees (individual time tables are kept by child processes). If the time slots are available, APO will mark/reserve the timeslots with the appointment and inform the child processes accordingly.*

[3] *If such an automatic "rescheduling" feature is added, it would be treated as bonus score of up to 5%. Automatic rescheduling may look a bit like choices in different memory allocation algorithms on FF, BF and WF (in lecture 7), in finding an alterative time slot for the appointment within the available timeslots of the user and other callees.*

Part 5. Analyze the program with different scheduling methods for which you have tested. List out those advantages and disadvantages which can be read from the program implemented.

Part 6. (*Bonus up to 5%*) Implement the automatic rescheduling algorithm(s) and discuss on the advantages and disadvantages of your rescheduling algorithm(s).

You must form a group of 4 to 5 persons for the project. The project must be implemented using **programming language C** and it must be successfully executed on **ANY ONE of Linux Servers** (*apollo or apollo2*) in the Department of Computing. You will have to specify to us on which Linux server your project would be executed for our testing.

***** *Note that "gcc" or "cc" is the compiler that can be used in this project.* *****

Implementation

User Interface

First of all, your program should provide a menu to allow the user to input appointments or command for the organizer. There are several input methods to be accepted by the system. The system should also show whether the appointment is accepted or rejected (print the reason if possible). Below is an example of some inputs and outputs as seen on a screen.

```
./apo 20230401 20230430 john mary lucy paul
~~WELCOME TO APO~~
Please enter appointment:
privateTime paul 20230401 1800 2.0
-> [Recorded]
Please enter appointment:
privateTime mary 20230402 2000 3.0
-> [Recorded]
Please enter appointment:
projectMeeting john 20230402 1900 2.0 paul mary
-> [Recorded]
Please enter appointment:
groupStudy paul 20230403 1800 2.0 john lucy
-> [Recorded]
Please enter appointment:
gathering lucy 20230404 1900 4.0 john paul mary
-> [Recorded]
…
…
Please enter appointment:
printSchd FCFS
-> [Exported file: Ggg_01_FCFS.txt]
…
…
Please enter appointment:
printSchd PRIORITY
-> [Exported file: Ggg_08_PRIORITY.txt]
Please enter appointment:
projectMeeting john 20230429 2000 2.0 lucy mary
-> [Recorded]
…
…
Please enter appointment:
printSchd ALL
-> [Exported file: Ggg_99_ALL.txt]
Please enter appointment:
endProgram
-> Bye!
```

Command Syntax and Usage

To execute the program	
Syntax	**./apo YYYYMMDD YYYYMMDD u1 u2 u3 …** **e.g. ./apo 20230401 20230430 john mary lucy paul**
Use	Enter [**./apo**] to start the program where [**YYYYMMDD**] x 2 and [**u1 u2 u3 …**] are **Start Date** and **End Date**, and the **Users** of the application respectively. For example, APO will manage requests starting from April 1, 2023 (**20230401**) and ending at April 30, 2023 (**20230430**). In addition, [**john mary lucy paul**], are the users in this application. ***Note***: This is the first instruction or command that should have been executed before the application could accept user requests and schedule those appointments. There are at least three users and the maximum number is ten. The system should return an error message if the number of users is not in the range 3 to 10. In addition, you need to convert the names to the standard format (*i.e., first letter capitalized regardless of the input string*) for all outputs. For the time being, APO simply manage the timeslots from 18:00 to 23:00 and it is one hour one timeslot. In the other words, there are five timeslots per day. Upon further development, APO may have half hour as one timeslot.

Command	**privateTime**
Syntax	**privateTime uuu YYYYMMDD hhmm n.n** **e.g. privateTime paul 20230401 1800 2.0**
Use	It is to add private time into a user's time table. As in the example, it is to add a private time session for [**Paul**] on [**April 1, 2023**] starting at [**18:00**] and the duration is [**2.0**] hours. [**uuu**] – Caller [**YYYYMMDD hhmm**] – Date and time of the event, **YYYY**:Year (4 digits), **MM**:Month (2 digits), **DD**:Day (2 digits), **hh**:Hour (2 digits) and **mm**:Minute (2 digits). [**n.n**] – Duration of the appointment in hours (fixed point of one decimal place)

Command	**projectMeeting**
Syntax	**projectMeeting uuu YYYY-MM-DD hh:mm n.n u1 u2 …** **e.g. projectMeeting john 20230402 1900 2 paul mary**
Use	It is to set up a meeting and to invite callees. As in the example, it is to create a project meeting session for [**John**] and invite [**Paul** and **Mary**] on [**April 2, 2023**] starting at [**19:00**] and the duration is [**2.0**] hours.

Command	**groupStudy**
Syntax	**groupStudy uuu YYYYMMDD hhmm n.n u1 u2 …** **e.g. groupStudy paul 20230403 1800 2.0 john lucy**
Use	It is to organize a group study and invite callees. As in the example, it is to make a group study session for [**Paul**] and invite [**John** and **Lucy**] on [**April 3, 2023**] starting at [**18:00**] and the duration is [**2.0**] hours.

Command	**gathering**
Syntax	**gathering uuu YYYYMMDD hhmm n.n u1 u2 …** **e.g. gathering lucy 20230404 1900 4.0 john paul mary**
Use	It is to organize a gathering and invite callees. As in the example, it is to make a gathering session for [**Lucy**] and invite [**John, Paul** and Mary] on [**April 4, 2023**] starting at [**19:00**] and the duration is [**4.0**] hours.

Command	**printSchd**
Syntax	**printSchd sssss / ALL** **e.g. printSchd FCFS** **[Exported file: Ggg_01_FCFS.txt]**
Use	It is to print the time table and the rejected list based on the algorithm indicated. In the example above, it is to print the time table and rejected list based on the algorithm [**FCFS**]. Then, the application prints out the report and exports it to the file [**Ggg_nn_sssss.txt**]. Where [**sssss**] – Scheduling algorithm [**ALL**] – All scheduling algorithms are included in this report. That is, if there are two algorithms used in the application, say, **FCFS** and **PRIORITY**, the [**printSchd ALL**] instruction will print out the report with the two algorithms. [**Ggg**] – Group Number [**nn**] – Sequence number (2 digits) of report printed, start from [**01**]

Command	**endProgram**
Syntax	**endProgram**
Use	This ends the program completely, upon collecting all the child processes and closing all the files.

To ease your work, we make the following assumptions:

1. Input formats simply follow those in the examples.
2. We will test the schedule for a period, from 1 to 31 May 2023 and the time is from 18:00 to 23:00. One time slot is an hour of time. That means there are 5 time slots a day. In addition, there is no appointment to be assigned on Sunday and Public Holiday.
3. There are at least three users and at most ten.
4. Usually, a "**`privateTime`**" entry has the highest priority and then it is a "**`projectMeeting`**". The next one is "**`groupStudy`**". The lowest one is "**`gathering`**". If the **`APO`** applies "**`priority`**" as the scheduling algorithm, that priority information for appointments would be considered. For example, a *project meeting* may "displace" an already scheduled *gathering* so that the caller/callee involved would attend the meeting and the gathering would be canceled (*or rescheduled*).
5. There are many implementation methods for the modules. The scheduler module may be implemented as a separate process, in the form of a child process created by the parent via **`fork()`** system call. The output module can also be a separate process. The scheduler may also be implemented as a separate program. If the parent is passing appointment details to a child scheduler, one should use the **`pipe()`** and associated **`write()`** / **`read()`** system calls. If the parent is passing information to a separate scheduler program, one could use the Unix shell "**`pipe`**" (denoted by "**`|`**").
6. If **`pipe()`** and **`fork()`** system calls are both used properly in the program, the maximum possible mark is 100% plus up to 5% of bonus points. On the other hand, if both system calls are not used in the program, only a passing mark would be awarded at best. Intermediate scoring will be given if only one system call is used, depending on the extent of usage.

Output Format

The "*appointment schedule*" and the "*rejected list*" may look like the following. Of course, you may have your own design for this report.

```
Period: 2023-04-01 to 2023-04-30
Algorithm used: FCFS

***Appointment Schedule***

. John, you have 999 appointments.
Date          Start   End     Type              People
================================================================================
2023-04-02    19:00   21:00   Project Meeting
2023-04-04    19:00   23:00   Gathering         Mary Paul
…
…
                          - End of John's Schedule -
================================================================================

  Paul, you have 999 appointments.
Date          Start   End     Type              People
================================================================================
2023-04-01    18:00   20:00   Private Time      -
2023-04-04    19:00   23:00   Gathering         John Mary
…
…
                          - End of Paul's Schedule -
================================================================================
…
…

…
…
                         - End of Lucy's Schedule -
================================================================================
…
…

…
…
                          - End of Mary's Schedule -
================================================================================
```

```
***Rejected List***

Altogether there are 999 appointments rejected.
================================================================================
 1. gathering lucy 20230414 1900 1.0 mary
 2. gathering john 20230424 1800 4.0 paul mary
 3.…
   …
                           - End of Rejected List -
================================================================================
```

```
*** Performance ***

Total Number of Requests Received: 999 (99.9%)
      Number of Requests Accepted: 999 (99.9%)
      Number of Requests Rejected: 999 (99.9%)



Number of Requests Accepted by Individual:

     John                - 99
     Lucy                - 99
     Mary                - 99
     Paul                - 99



Utilization of Time Slot:

     John                - 99.9%
     Lucy                - 99.9%
     Mary                - 99.9%
     Paul                - 99.9%
```

Error handling

Although the program does not need to check for the correctness of the values that users input, some other error handling features are required in the application. For example, if the number of users is out of range (*not between 3 and 10*), APO should return a message to indicate that.

Documentation

Apart from the above program implementation, you are also required to write a project report that consists of following parts:

1. Introduction (*Why do you have this project (the objective)*?)
2. Scope (*What operating systems topics have been covered in this project*?)
3. Concept (*What are the algorithms behind*?)
4. Your own scheduling algorithm (*if any*)
5. Software structure of your system
6. Testing cases (*Simply describe what you have done to test the correctness of the program. For the details of the tests, that should be attached in the section "**Appendix**".*)
7. Performance analysis (*Discuss and analyze the algorithm used in the program. Why it is better than the others*?)
8. Program set up and execution (*How to compile and execute your project*?)
9. Contribution of individual member – fill out the following table. For Item 1, fill in a task list assigned and done by the member. For Item 2 to 7, each box, fill in one of following letter grades:

   A – excellent
   B – good
   C – satisfactory
   D – marginal satisfactory
   F – fail

   According to your inputs, we will consider adjusting your individual grade. That means each member may receive a different grade eventually. If a member who has no contribution (*i.e. has done nothing*), that member might receive a ZERO finally.

Item	Group: **GGG**	Member 1: **A**	Member 2: **B**	Member 3: **C**	Member 4: **D**	Member 5: **E**
1	Responsibility: List the task(*s*) that each individual was (*were*) responsible for.	1. xxx 2. xxx 3. xxx	1. xxx 2. xxx 3. xxx	1. xxx 2. xxx 3. xxx	1. xxx 2. xxx 3. xxx	1. xxx 2. xxx 3. xxx
2	Cooperation: Able to work within team; willingly performed task(s).					
3	Punctuality: On time for team meetings.					
4	Reliability/Dependability: Performed tasks within established times.					
5	Evaluative: Offer constructive criticism and helpful evaluation of work.					
6	Creativity: Provide meaningful insight to project team.					
7	Overall Effort: Overall contribution to project.					

10. Appendix – source code, and, samples of "appointment schedule" and "rejected list".

Project Group

Please confirm your group on or before **February 10, 2023**. Send the member list to Alvin (alvin.mak@polyu.edu.hk).

Otherwise, you will be assigned to a group ***randomly***.

Demonstration

The demonstration will be held in weekend of **Week 11, *April 1, 2023***. Time is from 12:00pm to 6:00pm (*six hours*), please reserve your time on that day. You will receive a set of data and a worksheet to test your application. Follow the worksheet to execute your application properly. In the meantime, you are asked to take a video to record the execution. The length of the video should be about 3 to 5 minutes.

The data set will be released at noon and you have six hours to complete the demonstration. If you need to modify the source code, please also upload the revised source code file again to BlackBoard.

Mark distribution

The mark distribution of this project is as follows:

Implementation:	60 %
Documentation:	25 %
Demonstration:	15 %
Bonus:	5 %

Bonus

The bonus marks will be awarded for being able to propose and implement your own scheduling algorithm that performs better than AT LEAST ANY ONE of the well-known scheduling algorithms. In addition, proper use of the two system calls, **`pipe()`** and **`fork()`**, may have a certain proportion in the marking.

Late Submission Penalty – **10 %** per day.

Submission

There are two stages of the project.

For the **first stage**, you are asked to submit the followings on or before **March 3, 2023**:

1. A prototype of the application. It is a simple program of the execution of the application. Accept different number of users and assign one child process to each of them. That means if there are four users, you have to generate four child processes. Name the source code file as **`"Ggg_prototype.c"`**, where **`gg`** is your group number, e.g G05.

2. A project plan together with the screen shots of the execution of the prototype application. Name this document, **`"Ggg_projectPlan.docx".`**

For the **second stage**, details of submission are as follows:

1. Readme file (*write down the project title and the name of each member in your group, together with all necessary information that may be required to setup and run the application; name the file as* **`Ggg_Readme.txt`**).

2. Source code and output files

   - name the source code file as **`"Ggg_APO.c"`**
     `If there are more than one source code files, name them with a sequence (i.e. Ggg_APO_1.c, Ggg_APO_2, …)`
   - name the output files:
     - Log file of all input requests, **`"All_Requests.dat"`**.
     - Appointment Schedule (*according to the algorithm used*) as **`"Ggg_03_FCFS.txt", "Ggg_18_PRIORITY.txt", etc`**.
     - Rejected requests as **`"Ggg_rejected.dat"`**.

3. Name the project report as **`"Ggg_APOReport.docx"`**.

4. Record all inputs used in testing the program and save it to **`"Ggg_tests.dat"`**.

5. A presentation file (*Ggg_slides.pptx*) to induce the application.

6. A 3 to 5 minutes video (*in MP4 format and name it Ggg_video.mp4*) to demonstrate the application (*submit it after the demonstration*) together with outputs from the application and modified source code (*if applicable*).

Group these documents (*except Item 6*) in ONE zipped file and upload it to BlackBoard on or before the due date, **March 31, 2023**.

The video could be uploaded after the demonstration (**April 1, 2023**). In addition, if the size of the video is too large and cannot be uploaded to Blackboard properly, you may send us a link to download it.

- ꕥ End ꕥ -