# COMP2432 – Operating Systems

# Group Project (2017/2018 Semester 2)

Submission deadline: *April 7, 2018 (Week 11)*

Weighting: 15%

Project title: **Personal Organizer** (**PO**)

## Scenario

Every year, there are a number of freshmen coming to PolyU to start a different living style in the campus. Unlike in high school, they have to schedule their own timetable in order to achieve a good time management for studying and playing. In Department of Computing, Amy, Kate, Simon and Tom are all first year students and are good friends. They have formed a study group to help each other. However, they find that it is hard to squeeze out time for social gatherings since there are lots of activities and classes on campus, since they are taking different CARs. They usually mark all their appointments on their cell phones but it does not really help. This is because they have to check with each other for the availability. Knowing that you have studied the subject COMP2432 and have learned the algorithms of scheduling, they ask for your help to develop an application which can help them to schedule their appointments and gatherings *automatically*. More importantly, they can spend their leisure time more efficiently and effectively.

## Project Requirements

In this project, you have an opportunity to apply the theory of process scheduling that you have learnt from COMP2432 to a daily-life scenario and create an application called **"Personal Organizer" (PO)**. This project consists of four parts:

*Part 1*. Write a program that allows users to add details of appointment (date, time, duration, and/or callees) to the schedule. 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". This part of the program is referred to as the **Input Module**.

*Part 2*. **PO** checks whether the time slots are available. That is, not only the user's time slot, but also those of the callees', if any, would be ***checked***[1]. **PO** then sends messages to all callees to check for their availability and callees must reply the inquiry message no matter the time slot is available or not. If all parties are available, the

[1] *We assume that all callees would* ***accept*** *the appointment if there is a timeslot available according to his/her current schedule. Unavailable time slots should have been explicitly marked or blocked by callees in advance.*

appointment should then be ***confirmed***[2]. Otherwise, the appointment would be ***rejected*** or ***rescheduled***[3].

For example, consider that Simon calls for a gathering with the other three. **PO** checks not only for Simon's schedule but also for the schedules of the other three members (Amy, Kate and Tom). Only when all of them are available, the appointment would be recorded and confirmed. Otherwise, the appointment would be rejected and/or rescheduled.

Actually, rescheduling is one of bonus options. This appointment information should be reflected in the "appointment schedule" and "schedule report" generated (see **`printSchd`** and **`printReport`** below under **Implementation** section) by all the users involved.

*Part 3*. Extend the program developed in *Part 2* with a scheduler to generate a schedule for all appointments (i.e. timeslots engaged). The scheduler will implement several scheduling algorithms similar to those covered in lectures. This is called the **Scheduling Kernel**, within the **Scheduling Module**.

*Part 4*. Augment the program with the facilities to print appointment schedule for users and schedule report in *Part 2*. Those rejected and/or rescheduled appointments should all be included. This constitutes the **Output Module**.

*Part 5*. (*Bonus up to 5%*) Analyze the program with different scheduling methods on which you have tested. List out those advantages and disadvantages of the different algorithms which can be illustrated by the program implemented, that is, provide a comparison of those algorithms 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 or 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.

***** *Note that "gcc" or "cc" on Department's servers is the only compiler that can be used in this project.* *****

[2] *For implementation, actual human users are not involved. The system will answer the requests automatically, i.e. PO as the parent process is to check for the availability of callees (individual time tables are kept by child processes). If the time slots are available, PO will mark/reserve the time slots with the appointment and inform the child processes accordingly. As such, you can consider that there is an AI or "chatbot" acting on behalf of each user automatically in the system.*

[3] *Automatic rescheduling is an optional feature in the program. If such an automatic "rescheduling" feature is implemented, it would be considered towards the project 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 appropriate alternative time slot for the appointment among the available timeslots of the user and other callees.*

## Implementation

### User Interface

First, your program should provide a prompt to allow the user to input appointments or command for the organizer. There are several input methods to be accepted by the system. In addition, the system should show whether the appointment is accepted or rejected when the user requests for the schedule report to be printed. Below is an example for your reference.

```
./po amy kate simon tom
~~WELCOME TO PO~~
Please enter ->
addClass amy 20180401 1300 2
Please enter ->
addClass tom 20180411 0900 3
Please enter ->
addMeeting simon 20180410 1000 2 tom amy
Please enter ->
addMeeting amy 20180406 1000 3 kate
Please enter ->
addGathering tom 20180412 1500 2 simon kate
Please enter ->
addGathering kate 20180403 1200 1 amy simon tom
…
…
Please enter ->
printSchd simon fcfs SimonSchd.dat
Please enter ->
addClass simon 20180412 1500 2
Please enter ->
…
Please enter ->
printSchd simon pr SimonSchd_9.dat
printReport PO_Report_n.dat
…
Please enter ->
addBatch data_file_01.dat
…
…
Please enter ->
printReport PO_Report_m.dat
Please enter ->
printSchd amy fcfs AmySchd_2.dat
…
…
endProgram
-> Bye!
```

**Command Syntax and Usage**

| **To execute the program** | |
|---|---|
| Syntax | **`./po x1 x2 x3 …`**<br>**`e.g. ./po amy kate simon tom`** |
| Use | Enter [**`./po`**] to start the program where [**`x1 x2 x3`** …] are the users of the application. For example, [**`amy kate simon tom`**], they are the users in the program. At least there are three users and the maximum number is nine. System should return an error message if the number of users is not in the range, 3 to 9. You need to convert the names to the standard format (i.e., first letter capitalized regardless of the input string). |

| Command | **`addClass`** |
|---|---|
| Syntax | **`addClass xxx YYYYMMDD hhmm n`**<br>**`e.g. addClass simon 20180412 1500 2`** |
| Use | It is to add class session into a user's schedule. As in the example, it is to add a class session for [**`Simon`**] on [**`April 12, 2018`**] at [**`15:00`**]. And the duration is [**`2`**] hours.<br>[**`xxx`**] – Caller<br>[**`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).<br>[**`n`**] – Duration of the appointment in hours (single digit) |

| Command | **`addMeeting`** |
|---|---|
| Syntax | **`addMeeting xxx YYYYMMDD hhmm n xxx xxx …`**<br>**`e.g. addMeeting simon 20180410 1000 2 tom amy`** |
| Use | It is to set up a meeting and to invite callees. As in the example, Simon (caller) asks for a meeting session with [**`Tom and Amy`**] (callees) on [**`April 10, 2018`**] at [**`10:00`**] and the duration is [**`2`**] hours. |

| Command | **`addGathering`** |
|---|---|
| Syntax | **`addGathering xxx YYYYMMDD hhmm n xxx xxx …`**<br>**`e.g. addGathering kate 20180403 1200 1 amy simon tom`** |
| Use | It is to organize a gathering and invite callees. As in the example, it is to make a gathering session for [**`Kate`**] and invite [**`Amy, Simon and Tom`**] on [**`April 3, 2018`**] at [**`12:00`**]. And the duration is [**`1`**] hours. |

| Command | **`addBatch`** |
|---|---|
| Syntax | **`addBatch [`*`filename`*`]`**<br>**`e.g. addBatch data_file_01.dat`** |
| Use | That allows user to prepare a text file that contains multiple lines of requests |

| | |
|---|---|
| | ([`addClass`], [`addMeeting`] and [`addGathering`]) and then import that data file into the application. As in the example, it is to import a data file named [`data_file_01.dat`]. |

| Command | **`printSchd`** |
|---|---|
| Syntax | **`printSchd xxx [`***`fcfs/pr/…`***`] [`***`filename`***`]`**<br>**`e.g. printSchd simon pr SimonSchd_9.dat`** |
| Use | It is to print out a timetable (generated from a specific scheduling algorithm) of a caller to a named file. In the above example, it is to print out Simon's timetable to the file [`SimonSchd_9.dat`] with the scheduling algorithm "priority".<br>Depends on the number of algorithms that you have implemented in your application. Your program should be able to print timetables that generated by different scheduling algorithms, such as "first come first served", "priority", "shortest job first", etc. |

| Command | **`printReport`** |
|---|---|
| Syntax | **`printReport [`***`filename`***`]`**<br>**`e.g. printReport PO_Report_m.dat`** |
| Use | It is to print out all the accepted and rejected requests that generated by different algorithms. In the example, it is to print out the "Schedule Report" to a file named "PO_Report_m.dat".<br>In addition, some simple statistics figures should be provided (*see example below*) in the report. |

| 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 the examples.
2. There are at least three users and at most nine.
3. Schedule period is from April 1, 2018 to April 14, 2018 (two weeks' time). In addition, timeslot is one hour per unit and it is from 08:00 to 18:00, i.e. 10 hours per day. In other words, there are no half hour time slots.
4. In general, the algorithm "first come first served" is simply based on the input order. Once a timeslot is occupied by a former request, a later request for the same timeslot should be rejected.
5. For "priority", we assume that a "class" has the highest priority and then it is a "meeting". This is followed by a "personal matter" such as private tutoring. Finally, it is a "gathering". That means, a meeting may "displace" an already scheduled

gathering so that the caller/callee involved would attend the meeting and the gathering would be canceled.

6. 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 the 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 "**`|`**").
7. If **`pipe()`** and **`fork()`** system calls are both used properly in the program, the maximum possible mark is 100% plus up to 10% of bonus points. On the other hand, if both system calls are not used in the program, only a passing mark would be awarded. 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.

```
Personal Organizer
***Appointment Schedule***

Simon, you have 999 appointments.
Algorithm used: Priority
999 timeslots occupied.

Date          Start    End      Type         Remarks
=========================================================
2018-04-03    13:00    15:00    Class        -
2018-04-10    10:00    12:00    Gathering    Tom Amy
…
…

   - End -
=========================================================================
```

This example shows the schedule of the user “Simon” and is based on the “Priority” algorithm to arrange the activities. The column “Remarks” shows other parties who are involved in that particular activity.

```
Personal Organizer
***Schedule Report***

Algorithm used: First come first serv

***Accepted List***
There are 999 requests accepted.

Date          Start    End      Type         Remarks
=========================================================================
2018-04-01    10:00    12:00    Class        Amy
2018-04-03    13:00    15:00    Gathering    Kate Amy Simon Tom
…
…

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

***Rejected List***
There are 999 requests rejected.

Date          Start    End      Type         Remarks
=========================================================================
2018-04-10    10:00    12:00    Meeting      Kate Tom
2018-04-13    13:00    15:00    Gathering    Tom Simon Amy
…
…
=========================================

Total number of request:        999
Timeslot in use:                999 hours
Timeslot not in use:            999 hours
Utilization:                    999 %

   - End -
```

This example shows the “Schedule Report” which is based on the algorithm “First Come First Served” and lists out all the “accepted” and “rejected” requests. Furthermore, for each algorithm you have implemented in your application should have one individual report for that algorithm.

In “Remarks” column, it shows all parties who are involved in that activity. The one who is underlined is the “caller”, i.e. the one who made the request.

“Timeslot in use” is the total available timeslots of all users which have been assigned an activity. In this example, there are four users and the schedule period is from April 1 to 14 (14 days), and for each day, there are 10 hours available for each user. That is, total available timeslots are 4 x 14 x 10 = 560-hour timeslots.

## Error handling

Although the program does not need to check for the correctness of the values that users inputs (we simply assume that they are correct already), 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 9), PO should return a message to indicate that. In addition, for cases of timeslot and/or date that are out of range, you may simply ignore them and/or group those requests as “Incorrect Request”.

## 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 want to implement this project, i.e. 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 (What have you done to test the correctness of the program/application?)
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? On which Linux server would you like your project to be executed?)
9. *Appendix I* – source code, and, soft copies of "Appointment Schedule" and "Schedule Report".
10. *Appendix II* – **Contribution of Work**. You should indicate/describe each project group member's effort/workload for the project in details. Individual assessment would be based on this part mainly.

## Demonstration

The demonstration will be held in weekends of **Week 11 and Week 12** (tentatively). Each group is allocated 15 minutes to demonstrate the implementation of the scheduler. Please write down the *name*s and *student ID*s of your group members on the **COMP2432 Project Group Form** which should be posted outside **PQ821**.

Please confirm your group on or before **March 5, 2018**.

Otherwise, you will be assigned to a group randomly. You will be asked to sign up a demo time slot sheet later.

*Mark distribution*

The mark distribution of this project is as follows:

| | |
|---|---|
| Implementation: | 60 % |
| Documentation: | 25 % |
| Demonstration: | 15 % |
| Bonus: | 10 % |

***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.

## Submission

You must submit the following files (*zip them all in one*) to the Blackboard System:

1. Readme file (write down your project title and the name of each member in your group, together with all necessary information that may be required to run your program; name the file as Gxx_**`Readme.txt, where Gxx is your group number assigned`**).

2. Source code and output files:

   - Name the source code file as **`"PO.c".`**
   - Name the output files as follows:
     - Appointment Schedule as **`"SimonSchd_1.dat", "KateSchd_9.dat", etc`**. That means in running the application, the same schedule would be called to print more than one, so it is better to have a serious number to indicated the order of the printing.
     - Schedule Report as **`"PO_Report_1.dat", "PO_Report_n.dat", etc`**. Again, the report would be called to print more than once

3. Name the project report as **`"PO_Report_Gxx.doc"`**, where Gxx is your group number assigned.

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

Group these files in ONE zipped file.

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