**COMP2432 Group Project (2020/2021 Semester 2)**

**Room Booking Manager (RBM)**

Group Member

Sham Wing Chung (19050208D)

Lam Chi Yuen (19052423D)

Lam Kwan Lok (19051723D)

Wong Tsz Fung (19059855D)

**Introduction**

PolySME Business Center is a company aims at supporting small business companies by providing shared office and office facilities. To record the bookings made by their tenants, PolySME made use of a room booking system. However, the system has been in use for more than 10 years and suffered from drawbacks like lack of flexibility and poor utilization. Therefore, as a group of undergraduates working in PolySME as part-time, we propose to revise the system based on the scheduling methods acquired from the course COMP2432 Operating System. We hope that through a series of renovation, the company will be able generate more revenue on renting.

**Scope**

Before our plan to renovate PolySME’s booking system, we have been exposed to a range of topics regarding operating system. In the course COMP2432 Operating System, the fundamentals of operating system such as interprocess communication and programming (IPC), CPU scheduling and memory management are introduced. CPU scheduling, in particular, has provided us with the ideas to renovate the booking system installed in PolySME. In such chapter, various algorithms to schedule the processes are introduced. This includes, but not limited to, first-come-first-served (FCFS) and priority scheduling. These two algorithms form the basis of the system and this will be elaborated in the following sections.

**Concept**

The algorithms behind the new booking system are, as aforementioned, based on first-come-first-served and priority CPU scheduling. Unlike the usual preemptive implementation for algorithms like priority scheduling in most of the operating systems, since a booking should not be interrupted by another one, all these algorithms are non-preemptive, which means a booking cannot be broken down into small pieces and be assigned to the time slots non-continuously.

For the first-come-first-served scheduling in this system, the idea is simple – for the same time slot, the one who is able to make a booking the earliest will have his/her request accepted. Otherwise, it will be rejected. It should be known that for a booking to succeed, all its component should be available in that time slot. For instance, if tenant A wants to book both room A and a pair of devices, both the room and the pair of devices should be available in that period of time. Should one of the any components be occupied, the booking will be rejected.

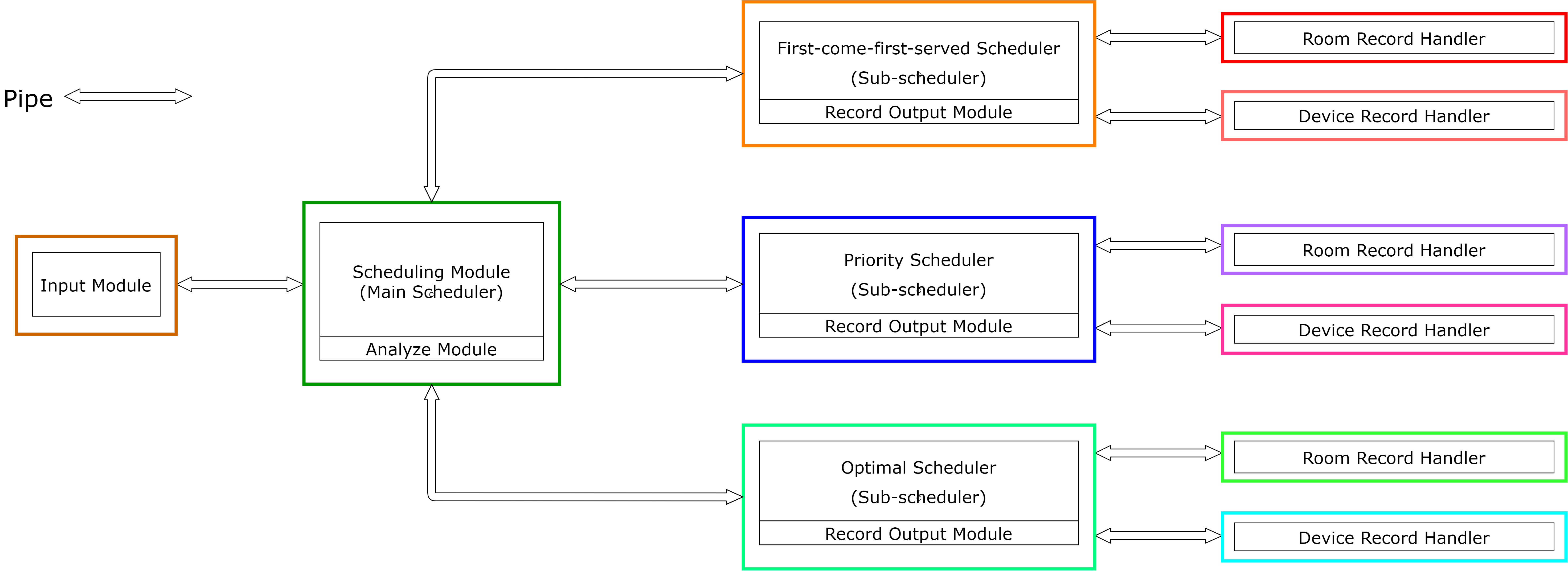
The priority scheduling in this system, on the other hand, is slightly more complicated than the FCFS scheduling. Apart from the time of making a booking, the priority of the booking, which is determined by the type of booking, should also be considered. Conference dwarfs all other types of booking and it is followed by presentation. Meeting ranked the third and device (a booking solely on a single device) has the lowest priority. The algorithm will attempt to replace an existing booking in the record if the priority is higher than the one stored. Again, all components should be available or has a higher priority when the system is to accept the booking.

**Optimal Scheduling Algorithm**

An optimal scheduling algorithm is specifically designed for the booking system. Such algorithm is a heavily modified version of the priority scheduling algorithm mentioned above. Instead of using the booking type as the criterion to determine the priority, we have used the length of the booking to determine the priority of booking, the longer the length, the higher the priority. In other words, long sessions will be first assigned to the time slots available. The idea behind this algorithm is simple – we deem performance of the booking as the utilization of the facilities. To optimize the utilization of the facilities, one of the ways is to assign longer session first, so that more time slots can be potentially booked. For instance, for tenant A booking a short session of 1 hour and tenant B booking a long session of 8 hours, and both bookings start at the same time, to maximize the utilization, we want to prioritize the long session, instead of the short session. As a result, tenant B’s booking will have a higher priority. The optimal scheduling algorithm provides a slight boos in performance without creating an algorithm from scratch.

**Program Structure**

To increase the efficiency of the process by increasing the level of multiprogramming, the booking system is divided into several modules implemented by creating child via fork() system call. For interprocess communication, pipe() is used together with the write() and read() system calls. The diagram below shows the structure of the whole booking system.

****

(Note: To simplify the diagram, only one bidirection pipe is shown between two processes. There are actually two unidirectional pipes between each pair of processes to prevent intermingling of pipe message. Each coloured box represent a distinct process/child created from fork().)

**Input Module**

After launching the booking system program, the driver section main() will fork() once, and the parent will be the input module. The input module is responsible for taking in and tokenizing user input. For example, if the input is addMeeting –tenant\_A 2021-05-16 12:00 2.0 9;, then it will break the string into addMeeting, -tenant\_A, 2021-05-16, 12:00, 2.0 and 9. After the tokenization, it will decide which categories the input fall into – it can be either an “addBatch” command or non “addBatch” command. If it an “addBatch” command, the input module will search for the corresponding file by using the parameter given and store the content of the file as strings. Otherwise, it will pass the tokenized string to the main scheduler for further processing. Error checking is also done while parsing. Should the booking has a device name error, it will be rejected.

**Scheduling Module (Main Scheduler)**

As mentioned, main will fork() once. The child created there will be the main scheduler. The main scheduler further fork() twice first to create two sub-schedulers as children, which are using first-come-first-served and priority algorithm respectively. As the parent, the main scheduler serves as a “switch” to filter and forward the information to sub-schedulers for arranging bookings. For example, if the user wants to print the bookings arranged by priority algorithm, the main scheduler will pass the message only to the priority scheduler. The main scheduler is also responsible for the parsing the tokenized strings into a structure called booking (see Table 1). Such structure is established to maintain a standard on information sharing and will record all the information related to the booking itself.

The main scheduler also includes the analyze module for performance evaluation. To retrieve the booking information from the child, a series of flags are used to communicate with the child (e.g. room record handler). After retrieving the information by using the dedicated pipe, the main scheduler carries out the necessary calculation. For example, utilization of a device is calculated by dividing the number of time slot occupied of that device by 168 (the total number of time slots (in hour) in 7 days. It is worth to mention that many other operations, such as printing the records, all use flags for communication between parents and their children.

|  |  |  |
| --- | --- | --- |
| Data Type | Field Name | Meaning of the Field |
| int | recordId | The record ID (input order) of a booking |
| int | day | The number of days between the booking day and 10-05-2021. (e.g. for a booking day of 14-05-2021, the value will be 4) |
| int | time | The time in hour (e.g. 12:00 will be 12) |
| int | length | The duration of the booking in hour |
| int | pplCount | The number of participants |
| char[2] | bookingType | The type of booking (e.g. meeting, presentation) |
| char[2] | tenantLetter | The letter representing who the tenant is |
| char[2][20] | device | The device booked in tandem with the room. This must be in pair |
| char | roomLetter | The letter for the room to be booked |

*Table 1: The structure of* struct booking.

**First-come-first-served Scheduler**

The main scheduler fork() for three times to create two children as sub-schedulers. One of which is the first-come-first-served schedulers. In the FCFS scheduler, it accepts and stores bookings in a FCFS manner, that is, the earlier the booking made, the more likely the booking will be accepted. The FCFS scheduler also fork() twice to create two handlers for booking arrangement. Apart from storing the bookings, the FCFS scheduler is also responsible for printing the booking records, with the help from the room record and device record handlers. When printing the records, the FCFS scheduler will first send all the records to the handlers to arrange the bookings in a FCFS fashion. If the booking is not rejected (judging from the handlers’ response), it will print the bookings out. It is mention-worthy that the analyze module also uses the printing function in the FCFS scheduler, since such module needs to know the statistics of the records arranged in a FCFS fashion.

**Priority Scheduler**

Another child of the main scheduler is the priority scheduler. In the priority scheduler, it accepts and stores bookings with respect to the priority of the booking, which is the booking type of the booking. The priority scheduler also fork() twice to create two handlers for booking arrangement. Apart from storing the bookings, the priority scheduler also needs to print the booking records, by using the two handlers created before. When printing the records, the scheduler will first send all the records to the handlers for processing. If the booking is not rejected (judging from the handlers’ response), it will print the bookings out. Much like the FCFS scheduler, the priority scheduler is used by the main scheduler for the analyze module.

**Optimal Scheduler**

The third child of the main scheduler is the optimal scheduler. In the optimal scheduler, it accepts and stores bookings according to our algorithm, which sets the priority of the booking by using its length (the longer the length, the higher the priority). Again, the optimal scheduler will fork() twice to create two handlers and is responsible for printing the records. The mechanism of optimal scheduler is largely the same as the FCFS and priority scheduler. It is also used by the main scheduler for the analyze module.

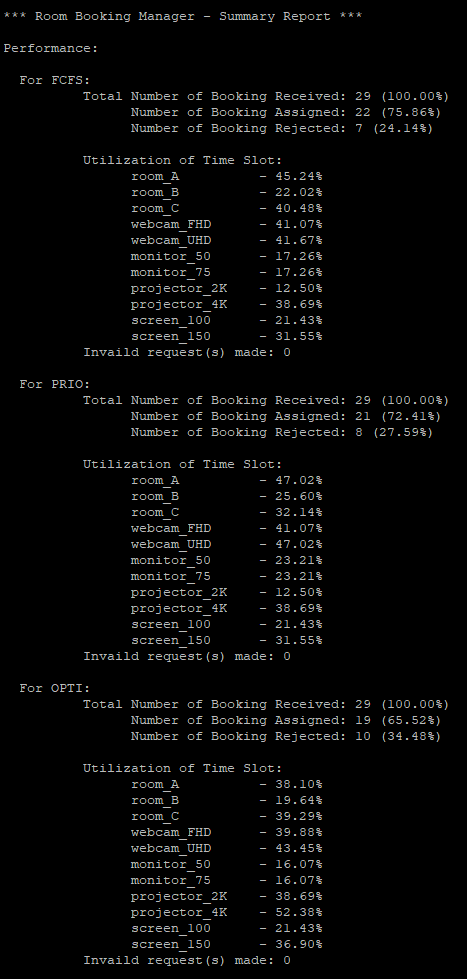
**Testing Cases**

Various testing cases are inputted into the program to test the validity of the program. The following table lists the type of possible input and the response of the program. To find out the input used for each case, please check in the appendix for corresponding case number.

|  |  |  |
| --- | --- | --- |
| Testing Cases | Case No. | Program Response |
| Four valid bookings with different day and time | 1 | The four records should be added without conflicts. |
| Three valid bookings with same day, time and number of participants | 2 | The three records should be added without conflicts, since each booking can have either room A, B or C. |
| Four valid bookings with same day, time, meeting type, number of participants | 3 | The first three records should be added without conflicts, since they can have either room A, B or C.  The last record will be rejected since there is not any slot. |
| Four valid bookings with same day, time, number of particpants. First three bookings are of “meeting” and the last booking is of “conference”. | 4 | **FCFS scheduling**  The first three records should be added without conflicts, since they can have either room A, B or C.  The fourth booking will be rejected. |
| **Priority scheduling**  The first three records should be added without conflicts, since they can have either room A, B or C.  The fourth booking will replace either one of the three existing records due to its higher priority.  The replaced record will be rejected |
| Three valid bookings with same day, time. The first booking is for booking device only. The remaining two is for booking a room with the same devices. The quantity of the device is two. | 5 | **FCFS scheduling**  The first booking for device should be added without conflicts.  The second booking for a room and device should also be added without conflicts, since there are more than one such devices (chosen by the first tenant).  The third booking will be rejected since there are not enough device. |
| **Priority scheduling**  The first booking for device should be added without conflicts.  The second booking for a room and device should also be added without conflicts, since there are more than one such devices (chosen by the first tenant).  Since device has a lower priority, the first booking will be replaced and rejected. The third booking will be added. |
| Two valid bookings with same day, room C, same booking type but different time. The first booking’s running overlapped with the second booking’s starting time. | 6 | The first booking for room C should be added without conflicts.  However, the second booking for room C will be rejected since the first tenant will be still using the room when the second tenant starts. |
| Two valid bookings with same day, room C but different time. The first booking’s running overlapped with the second booking’s starting time. The second booking also has a higher priority. | 7 | **FCFS scheduling**  The first booking for room C should be added without conflict.  The second booking will directly be rejected. |
| **Priority scheduling**  The first booking for room C should be added without conflicts at first.  However, as the second booking came in, the first booking will be rejected and the time slot it occupied should be emptied.  The second booking for room C should then be added without conflicts. |
| Three valid bookings for the same device with the same day, time and duration. The quantity of the device is two. | 8 | The first two bookings should be added without conflict.  However, for the last booking, since there are not enough device, it will be rejected. |
| One booking with invalid device name | 9 | The only booking should be ignored, but not marked as rejected. |
| A general test case for evaluation performance | 10 | / |

*Table 2: Testing Cases and Response.*

**Performance Analysis**



By using the test case 10, we can retrieve the performance evaluation above. Below is some analysis for the performance.

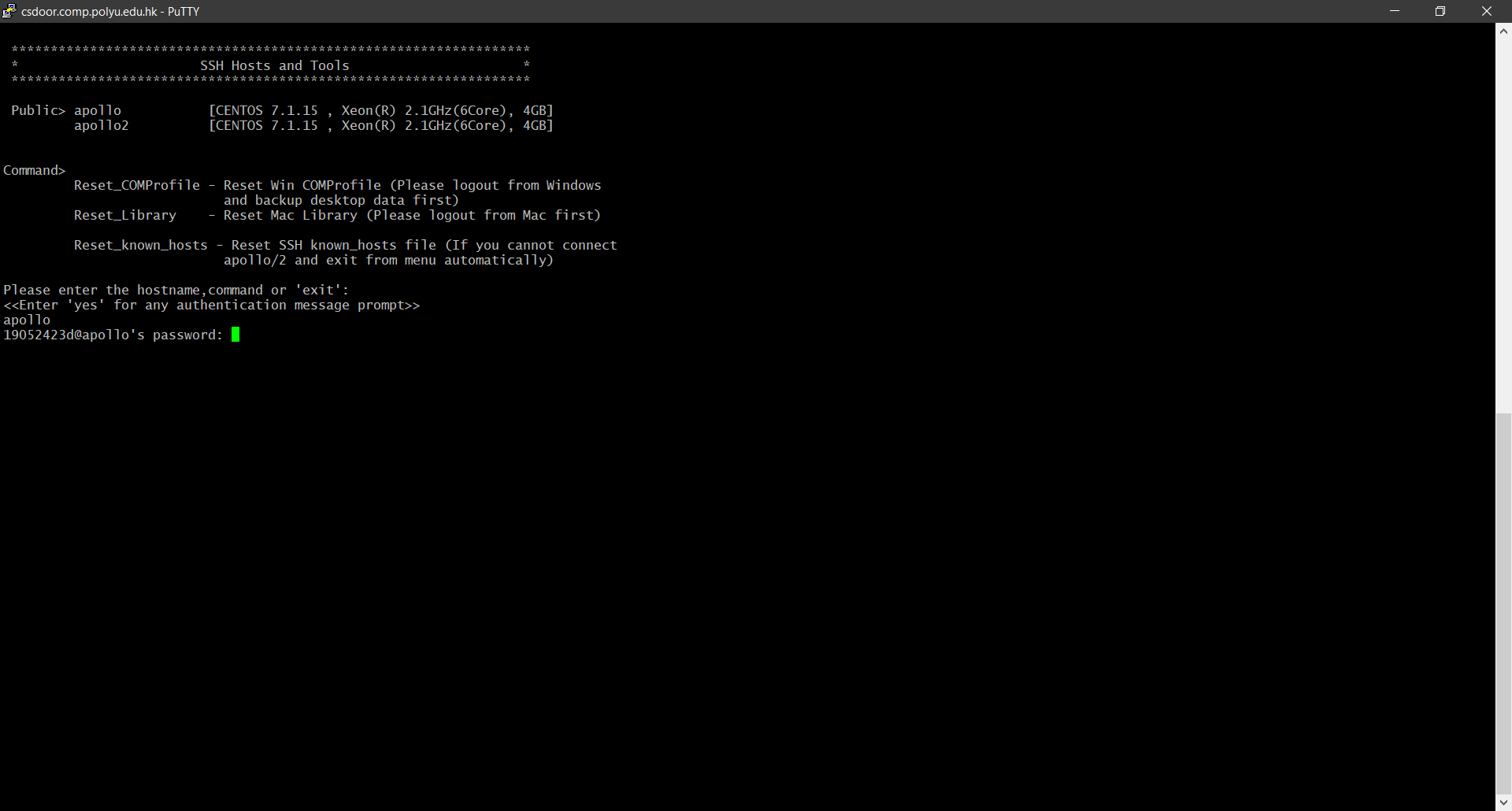
For FCFS scheduling, in total 22 out of 29 bookings are accepted, The average utilization for the three rooms is 35.89%, the highest among all. The average utilization for all the devices is 27.68%, the lowest among all. The reason behind FCFS scheduling having the lowest utilization for all the devices is that the bookings without device are mostly placed in front. As a result, the bookings (with devices) in the end of the batch file are mostly rejected.

For priority scheduling, in total 21 out of 29 bookings are accepted. The average utilization for three rooms is 34.92%, the second among all. The average utilization for all the devices is 29.84%, the second among all. Unlike FCFS scheduling, it is difficult to explain the performance of priority scheduling since the booking type is deemed “random” and plays an important role facilities allocation.

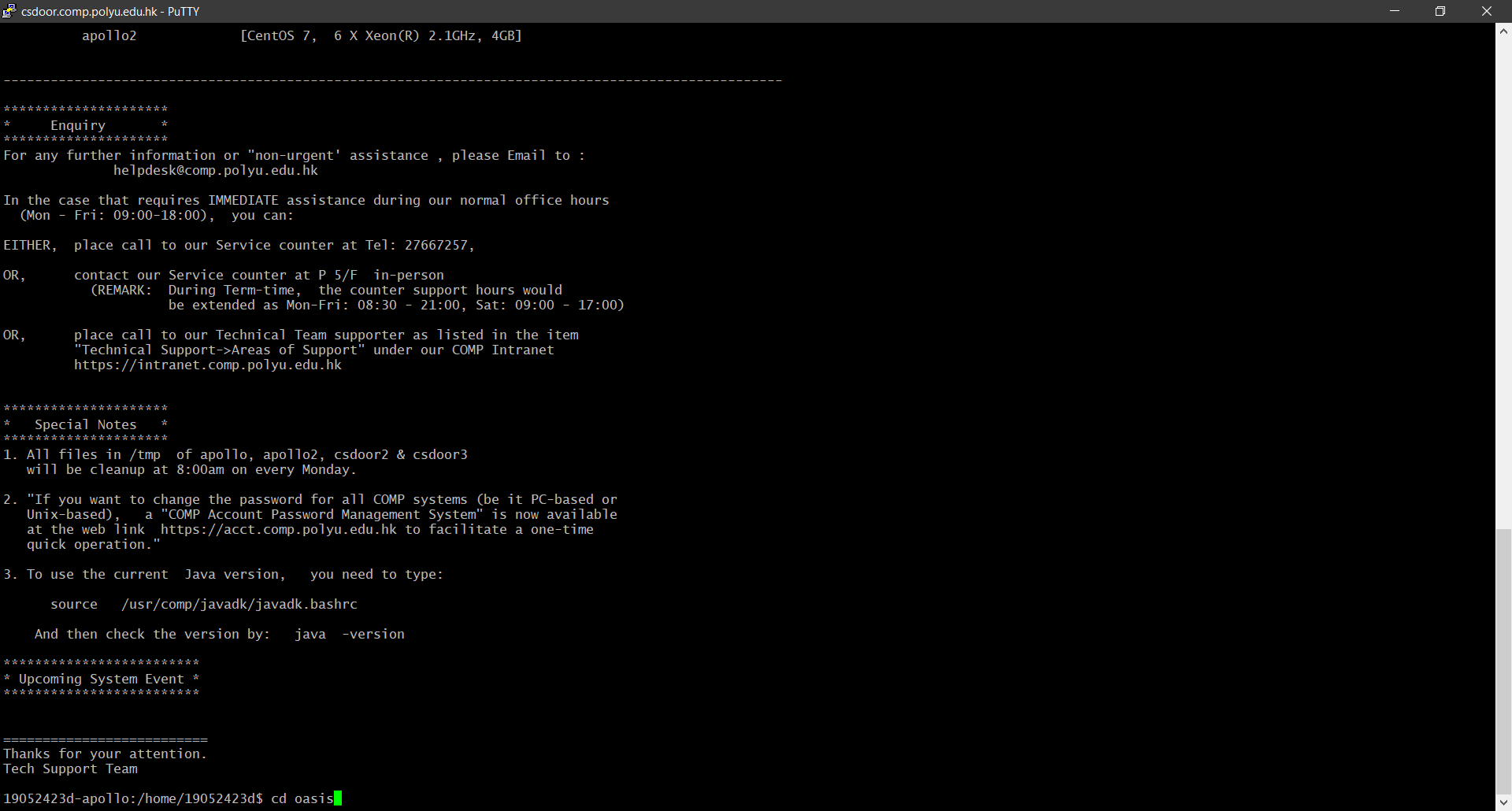
For optimal scheduling, in total 19 out of 29 bookings are accepted. The average utilization for three rooms is 32.34%, the lowest among all. The average utilization for all the devices is 33.11%, the highest among all. The reason behind priority scheduling having the highest average utilization for all the device is that we assign the longest session first. As a result, more time slots of the devices are occupied compared a shorter session. However, for the rooms, since devices are mostly not available, the rooms cannot be assigned together with the device. It, therefore, results in a relatively low performance.

**Program Set-up and Execution**

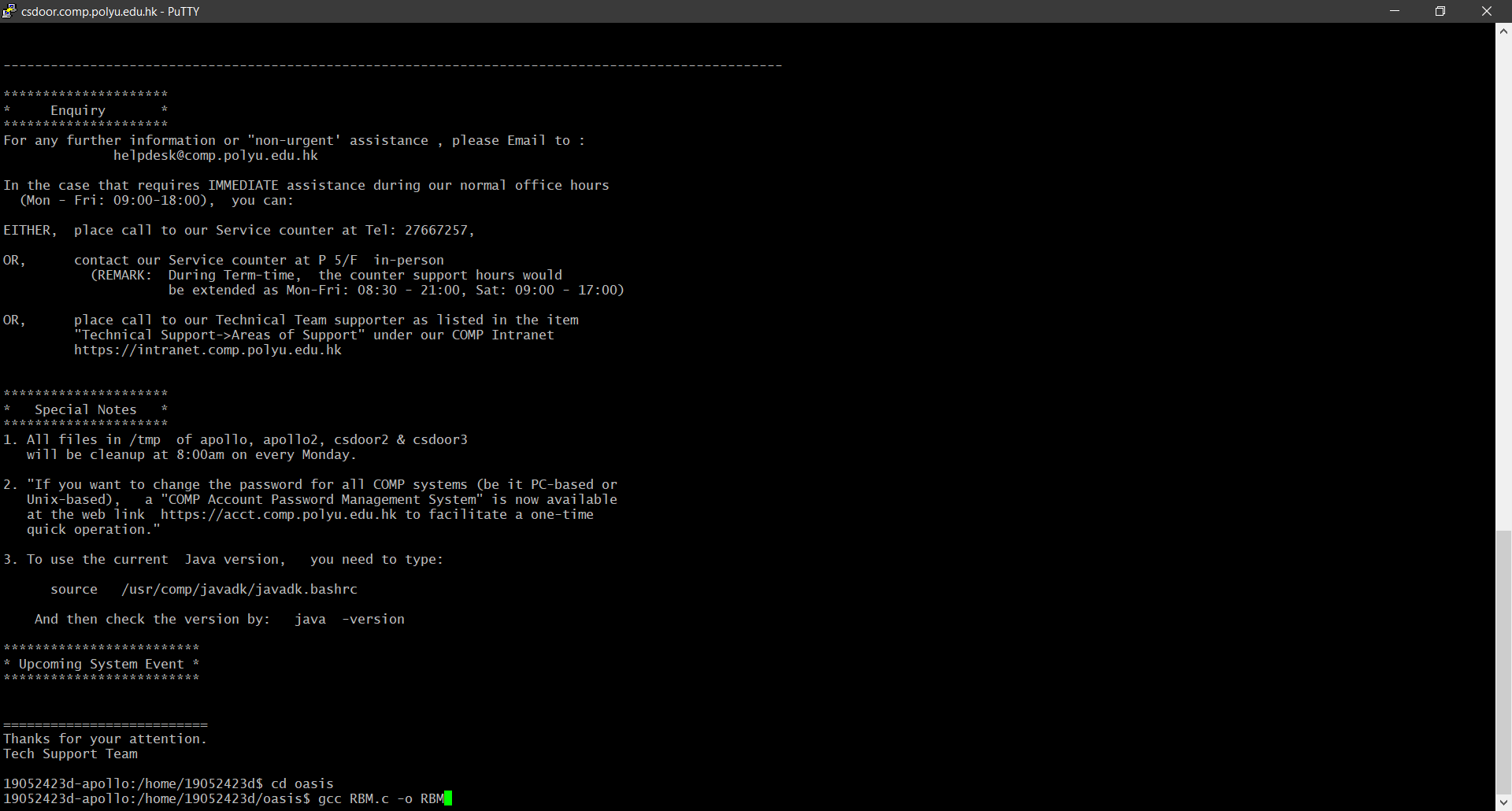
To compile the program, both Linux servers (apollo and apollo2) can be used. The .c source file can be compiled like most of the other .c source files, by typing gcc RBM.c -o RBM in the terminal. Such will generate an executable file. If one is to use a batch file (in .dat format) for inputting command, the batch file should be placed in the same directory as the generated executable file. The batch file should follow Linux convention of line change. If one is to use batch file edited in Windows environment, dos2unix command should be used to convert a DOS text file to Linux format. The following screenshots show an example of program set-up and execution.



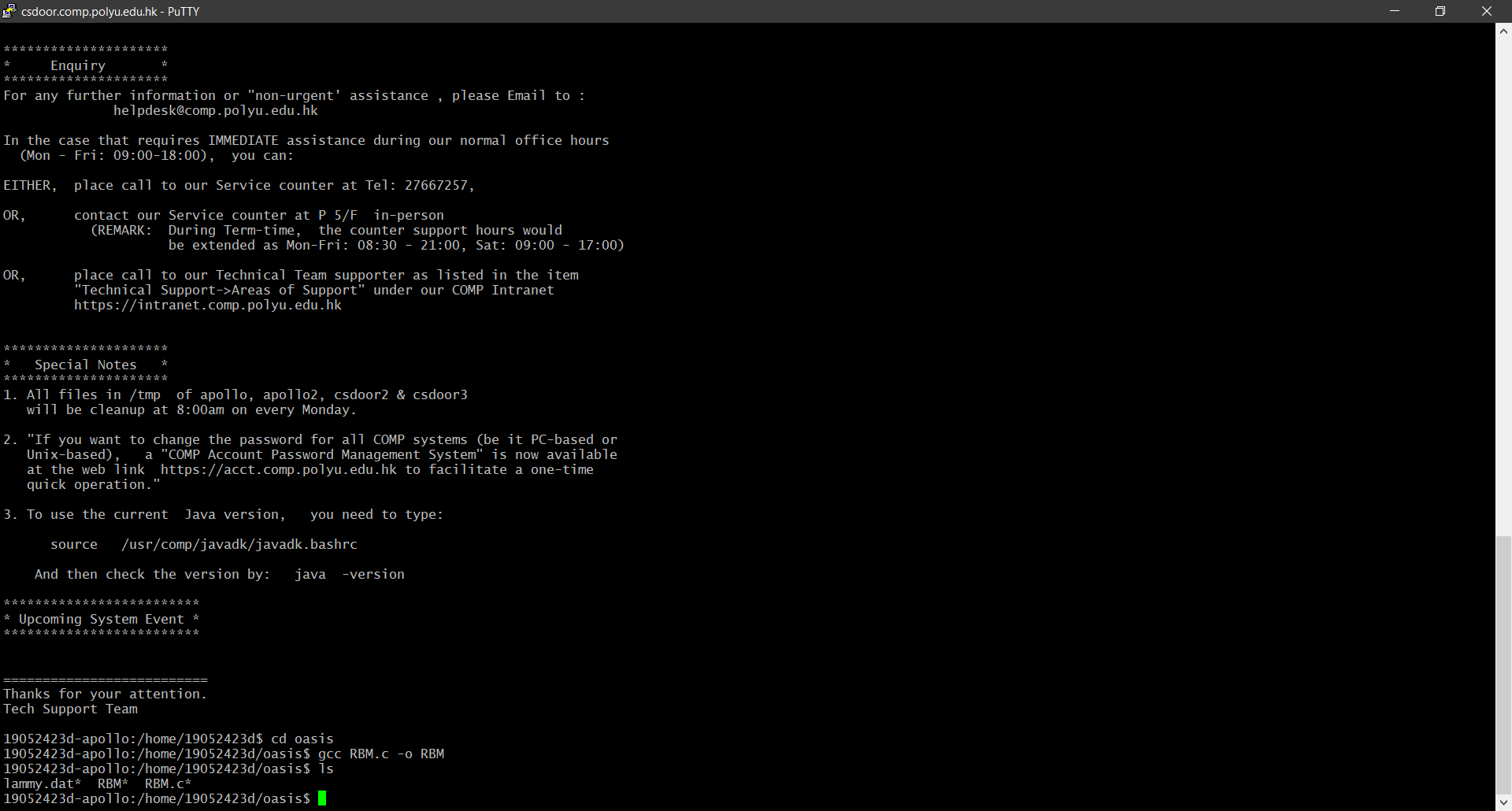
*Screenshot 1: Linux Server Login*



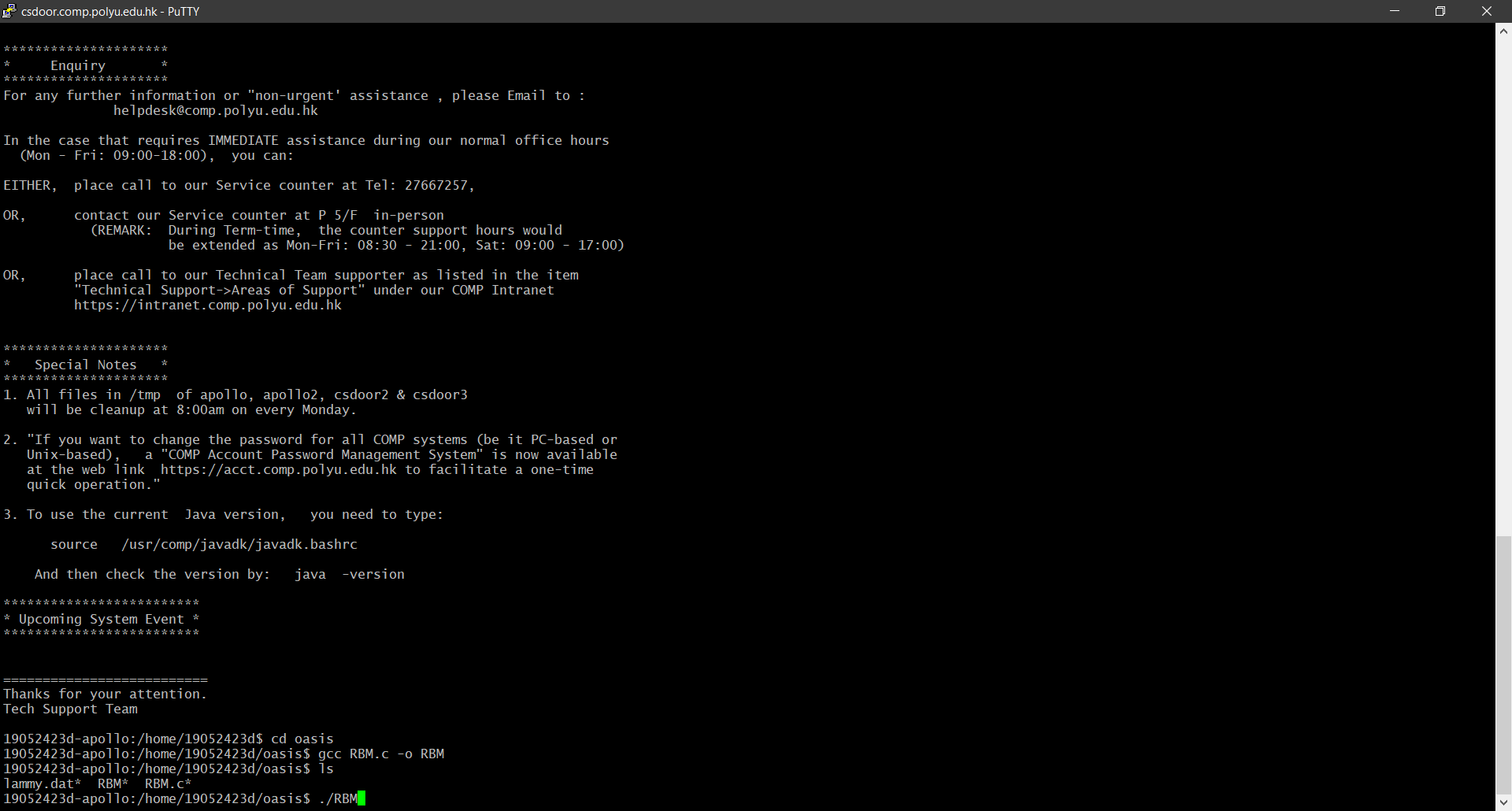
*Screenshot 2: Directory Selection*



*Screenshot 3: Source Code Compilation*



*Screenshot 4: Showing the Executable Program and Batch File*



*Screenshot 5: Launching the Program*

**Appendix**

**Test Cases**

|  |  |
| --- | --- |
| Case No. | Command Input |
| 1 | addMeeting –tenant\_B 2021-05-10 12:00 5.0 9;  addPresentation –tenant\_C 2021-05-10 12:00 2.0 12;  addConference –tenant\_D 2021-05-15 12:00 4.0 9;  bookDevice –tenant\_E 2021-05-15 12:00 3.0 2 monitor\_50; |
| 2 | addMeeting –tenant\_B 2021-05-10 12:00 5.0 9;  addMeeting –tenant\_C 2021-05-10 12:00 5.0 9;  addMeeting –tenant\_D 2021-05-10 12:00 5.0 9; |
| 3 | addMeeting –tenant\_B 2021-05-10 12:00 5.0 9;  addMeeting –tenant\_C 2021-05-10 12:00 5.0 9;  addMeeting –tenant\_D 2021-05-10 12:00 5.0 9;  addMeeting –tenant\_E 2021-05-10 12:00 5.0 9; |
| 4 | addMeeting –tenant\_B 2021-05-10 12:00 5.0 9;  addMeeting –tenant\_C 2021-05-10 12:00 5.0 9;  addMeeting –tenant\_D 2021-05-10 12:00 5.0 9;  addConference –tenant\_E 2021-05-10 12:00 5.0 9; |
| 5 | bookDevice –tenant\_C 2021-05-10 12:00 4.0 webcam\_FHD;  addMeeting –tenant\_B 2021-05-10 12:00 5.0 9 webcam\_FHD monitor\_50;  addMeeting –tenant\_C 2021-05-10 12:00 5.0 9 webcam\_FHD monitor\_50; |
| 6 | addMeeting –tenant\_B 2021-05-10 12:00 5.0 15;  addMeeting –tenant\_C 2021-05-10 14:00 5.0 15; |
| 7 | addMeeting –tenant\_B 2021-05-10 12:00 5.0 15;  addConference –tenant\_C 2021-05-10 14:00 5.0 15; |
| 8 | bookDevice –tenant\_C 2021-05-10 12:00 4.0 webcam\_FHD;  bookDevice –tenant\_D 2021-05-10 12:00 4.0 webcam\_FHD;  bookDevice –tenant\_E 2021-05-10 12:00 4.0 webcam\_FHD; |
| 9 | bookDevice –tenant\_C 2021-05-10 12:00 4.0 banana\_100g; |
| 10 | addMeeting -tenant\_A 2021-05-10 00:00 15.0 8 webcam\_UHD monitor\_50;  addMeeting -tenant\_C 2021-05-11 12:00 17.0 7 projector\_4K screen\_150;  addMeeting -tenant\_C 2021-05-12 02:00 22.0 2 projector\_2K screen\_150;  addMeeting -tenant\_C 2021-05-14 12:00 14.0 9 webcam\_FHD monitor\_50;  addPresentation -tenant\_A 2021-05-10 20:00 14.0 18 webcam\_FHD monitor\_75;  addConference -tenant\_C 2021-05-10 00:00 24.0 1 webcam\_UHD monitor\_75;  addConference -tenant\_D 2021-05-11 13:00 10.0 18 webcam\_UHD monitor\_50;  addPresentation -tenant\_A 2021-05-12 14:00 12.0 7 webcam\_FHD monitor\_75;  bookDevice -tenant\_A 2021-05-12 08:00 16.0 screen\_150;  bookDevice -tenant\_B 2021-05-10 00:00 20.0 screen\_100;  bookDevice -tenant\_E 2021-05-11 10:00 16.0 screen\_100;  addMeeting -tenant\_C 2021-05-11 02:00 22.0 20 projector\_2K screen\_150;  addPresentation -tenant\_D 2021-05-13 01:00 10.0 14 webcam\_UHD monitor\_50;  addPresentation -tenant\_D 2021-05-15 00:00 12.0 17 webcam\_FHD monitor\_50;  bookDevice -tenant\_B 2021-05-16 15:00 3.0 projector\_2K;  addMeeting -tenant\_E 2021-05-12 15:00 15.0 2 webcam\_FHD monitor\_75;  addConference -tenant\_C 2021-05-12 02:00 19.0 3 webcam\_UHD monitor\_50;  addPresentation -tenant\_D 2021-05-13 16:00 2.0 4 projector\_4K screen\_150;  addConference -tenant\_C 2021-05-14 08:00 18.0 8 projector\_2K screen\_150;  addMeeting -tenant\_C 2021-05-12 03:00 19.0 13 webcam\_FHD monitor\_75; addMeeting -tenant\_A 2021-05-14 00:00 10.0 3 projector\_2K screen\_100;  addPresentation -tenant\_E 2021-05-16 15:00 3.0 17 webcam\_FHD monitor\_75;  addConference -tenant\_B 2021-05-15 13:00 16.0 10 webcam\_UHD monitor\_50;  bookDevice -tenant\_A 2021-05-10 07:00 12.0 projector\_4K;  bookDevice -tenant\_C 2021-05-16 02:00 20.0 webcam\_UHD;  bookDevice -tenant\_B 2021-05-12 06:00 18.0 webcam\_FHD;  bookDevice -tenant\_E 2021-05-10 16:00 22.0 webcam\_FHD;  bookDevice -tenant\_A 2021-05-13 09:00 22.0 projector\_4K;  bookDevice -tenant\_B 2021-05-12 00:00 42.0 projector\_4K;  bookDevice -tenant\_C 2021-05-14 22:00 34.0 projector\_4K; |

**The Timetable for Test Case 10**

