

**COMP2432 Operating Systems**

**Group 5**

**Appointment Organizer**

**Project Report**

DENG Chun Yung (21084115D)

JIANG Guanlin (21093962D)

LI Tong (21101988D)

LIU Yuzhou (21100602D)

NIYITEGEKA Berwa Aime Noel (21104645D)

# 

# 

# 1. INTRODUCTION

An **Appointment Organizer** (APO) is designed to help users to arrange their time to make sure the events can be arranged in that time. APO can let users add their events to the timetable to avoid the events having time crashes, and can also reschedule the events if the time crashes. An event with multiple users also can arrange for all the users automatically, users also can print their Appointment list and reject list to make sure those events are appointed.

*The Appointment Management App (APO) is a software application designed to help users schedule and manage appointments with ease. The app is designed to allow multiple users to schedule appointments based on each user's availability, using a variety of scheduling algorithms to ensure fairness and efficiency.*

*The APO app consists of several modules, including an input module, a scheduling module, and an output module. The input module allows users to enter appointment details, such as the date, time, duration, and callees involved, while the scheduling module is responsible for managing the scheduling and coordination of appointments between multiple users. The output module allows users to view details of scheduled appointments, and the report of the scheduling process.*

*The APO app makes use of various programming concepts and technologies, including C programming language, data structures such as linked lists, and interprocess communication mechanisms such as pipes. The app is designed to be user-friendly and efficient, providing a convenient and easy-to-use interface for scheduling and managing appointments.*

*In this report, we will provide a detailed overview of the APO app, including the design and implementation of each module, as well as the programming concepts and technologies used in the development of the app. We will also discuss the challenges faced during the development process, as well as the solutions and strategies employed to overcome these challenges. Finally, we will provide an evaluation of the app's performance and functionality, along with suggestions for future improvements and enhancements.*

# 2. SCOPE (What operating systems topics have been covered in this project?)

**System calls**

The system call is a routine that allows user programs to access services that require special permissions. System calls are used when a user application needs a special function or service which can only be performed by the operating system’s kernel. These are programming interfaces between the process and the operating system. We have multiple system calls in our program for services like file management, interprocess and communication process management.

These are some of the systems calls that we have used in our program:

* + fork(): The fork system call is used to start a new process, known as a child process, which runs concurrently with the process that makes the fork() call (parent process).

We built our application on the concept of multiprocessing. The application has a main process which creates a separate process for each user. Each user’s process will be responsible for managing their timetable, and the parent process is in charge of assigning the appointments.

* + pipe(): One of the core concepts behind our program is inter-process communication. There is constant communication between parent and child processes. For instance, when an appointment needs to be scheduled, the parent sends appointment details to all processes of concerned users. That communication between parent process and child process is restricted only to kernel mode; therefore, pipe() system call is used to get the access to that service.

**Scheduling**

* + Similar to CPU scheduling the processes, APO schedules appointments for users. It works with a similar working principle of admitting appointments in ready state, and then being scheduled one by one based on specified algorithms. In our case, the algorithms are FCFS, and priority-based algorithms.

FCFS (First-Come-First-Serve) is a scheduling algorithm used in the APO appointment management app to schedule appointments for multiple users. The algorithm works by processing appointment requests in the order in which they are received, with earlier requests being processed first. This approach is commonly used in systems where there is no need to prioritize one request over another, and where fairness in processing requests is important.

In the context of the APO, the FCFS scheduling algorithm is used to ensure that appointment requests are processed in the order in which they are received, without giving priority to any particular user or request. This means that users who submit their appointment requests earlier will have their appointments processed first, regardless of their priority or the urgency of their requests.

The use of FCFS in the APO is related to operating system scheduling in a number of ways. In particular, the FCFS scheduling algorithm is similar to the FSCS scheduling algorithm used by many operating systems to schedule processes. Like FCFS, OS’s FCFS schedules processes’ requests in the order in which they are received, with earlier requests being processed first.

Additionally, the FCFS scheduling algorithm used in the APO app is an example of a non-preemptive scheduling algorithm, meaning that once an appointment request is processed, the scheduler does not interrupt the current process to process a higher-priority request. This approach is similar to non-preemptive scheduling algorithms used by operating systems to manage process scheduling, where processes are allowed to run until they complete or voluntarily yield the CPU.

Overall, the use of FCFS in the APO appointment management app demonstrates how operating system scheduling concepts can be applied in other domains to manage resources and prioritize tasks. By leveraging the principles of FCFS scheduling, the APO app is able to provide a fair and efficient appointment scheduling experience for multiple users, helping to ensure that appointments are processed in a timely and orderly manner.

**IO redirection**

* + APO supports batch processing, which means that users can submit appointments in bulk rather than one at a time. To support this feature, APO uses I/O redirection to read appointment requests from a file instead of from the command line.

**Inter process communication**

* + In APO, interprocess communication is used to facilitate communication between the parent process and the child processes of the concerned users. Specifically, pipe() is used to create communication channels between processes for data transfer. When a new appointment is created, the parent process sends the appointment information to the appropriate child process using the communication channels. The child process then receives the appointment information and processes the appointment request accordingly. The child process uses a similar mechanism to communicate back to the parent process.

**Memory allocation algorithms**

The rescheduling module in APO uses the memory management algorithm of first fit to find available time slots that can accommodate a rejected appointment. Specifically, the module searches through a list of available time slots in order and selects the first one that is large enough to fit the appointment.

This approach is similar to the first fit memory management algorithm used by operating systems to allocate memory to programs. In the first fit algorithm, the operating system searches through a list of available memory blocks in order and selects the first one that is large enough to accommodate the program's memory requirements.

By using the first fit algorithm in the rescheduling module, APO can quickly and efficiently find available time slots that can accommodate rejected appointments. This approach is particularly useful in situations where there are many available time slots to choose from, as it allows APO to quickly identify potential matches without having to search through the entire list of available slots.

**File management**

The output module in APO is responsible for printing out a report about scheduled and rejected appointments and exporting the report to a file. To support this feature, the app relies heavily on the operating system's file management capabilities.

Specifically, file management functions such as fopen(), fwrite(), and fclose() are used to create, write to, and close output files. These functions allow APO to manipulate files in a way that is consistent with the operating system's file management standards, ensuring that APO can read and write files consistently across different operating systems and configurations.

# 3. CONCEPT (What are the algorithms behind?)

1. **FCFS scheduling**

One of the scheduling algorithms used in APO is First-Come, First-Served (FCFS) scheduling. This algorithm schedules appointments in the order in which they are received, with earlier appointments given priority over later appointments. FCFS scheduling is straightforward and easy to implement, but may not be ideal for situations where some appointments are more urgent or important than others.

First come first served algorithm:

* 1. The module receives an appointment request from a user and adds it to the appointment list.
  2. The module accesses the appointment list and retrieves the first appointment in the list.
  3. The module prompts users whether they are available at the requested time of the appointment.
  4. If all users are available, the module will notify all users to mark the appointment in their timetables.
  5. Otherwise, the module will notify the users that the appointment was not successful; hence no need to mark it in their timetable.
  6. The module continues to schedule appointments in the order they are received, until it reaches the end of the list.

1. **Priority-based scheduling**

Another scheduling algorithm used in APO is Priority-Based Scheduling. This algorithm assigns a priority level to each appointment request, based on factors such as urgency, importance, or the identity of the caller. Appointments with higher priority levels are given scheduling priority over appointments with lower priority levels.

Priority-based algorithm:

* 1. The module receives an appointment request from a user and adds it to the appointment list.
  2. The module accesses the appointment list and retrieves the first appointment in the list.
  3. The module prompts users whether they are available at the requested time of the appointment.

The time slot will be available, if there is no activity scheduled or the activity scheduled is of lower priority. If there are activities of lower activity, the users will notify the module about that.

* 1. If all users are available, the module will notify all users to mark the appointment in their timetables.
  2. The module will also send a cancellation message to all users involved in lower priority activities that were replaced by this appointment.
  3. if all users are not available, the module will notify the users that the appointment was not successful; hence no need to mark it in their timetable.
  4. The module continues to schedule appointments in the order they are received, until it reaches the end of the list.

1. **First-Fit allocation algorithm**

The appointment scheduling module with automatic rescheduling for rejected appointments is a feature of the appointment management app that allows users to easily reschedule appointments when they are rejected. When an appointment is rejected, the app will prompt all users involved in the appointment to provide their available time slots. The app will then use the First-Fit algorithm to find the common slot among the available time slots of all users. Once the common slot is identified, the app will automatically reschedule the appointment for that time slot and update the appointment schedules of all users involved. This feature helps users efficiently manage their appointments and reduces the need for manual rescheduling.

1. When an appointment is rejected by any user, the appointment management app will automatically prompt all users involved in the appointment to provide their available time slots from their timetables..
2. The app will then use the First-Fit algorithm to find the common slot among the available time slots of all users. The First-Fit algorithm starts at the earliest available time slot and checks if it can accommodate the duration of the appointment. If it can, the algorithm selects that time slot as the common slot. If it cannot, the algorithm moves to the next available time slot and repeats the process until it finds a common slot.
3. Once the common slot is identified, the app will automatically reschedule the appointment for that time slot.
4. The app will also notify all users involved in the appointment about the new appointment time and update their appointment schedules/timetables accordingly.

**4. YOUR OWN SCHEDULING ALGORITHM** (if any)

We do not implement our own scheduling algorithm.

**5. SOFTWARE STRUCTURE OF THE SYSTEM**

**5.1 Input module**

The Input Module is responsible for allowing users to add details of appointments to the scheduler. The module allows users to input various details about the appointment, including the date, time, duration, and callees involved. The user who initiates the appointment is referred to as the "caller" or the "user", while the other members involved in the appointment are referred to as "callees". The module supports both interactive and batch input modes, allowing users to enter appointment details through the command line or by reading from files.

To support batch input, the Input Module uses file I/O functions to read appointment details from files. The module reads input files line-by-line, parsing each line to extract the relevant appointment details.

**5.2 Input processing**

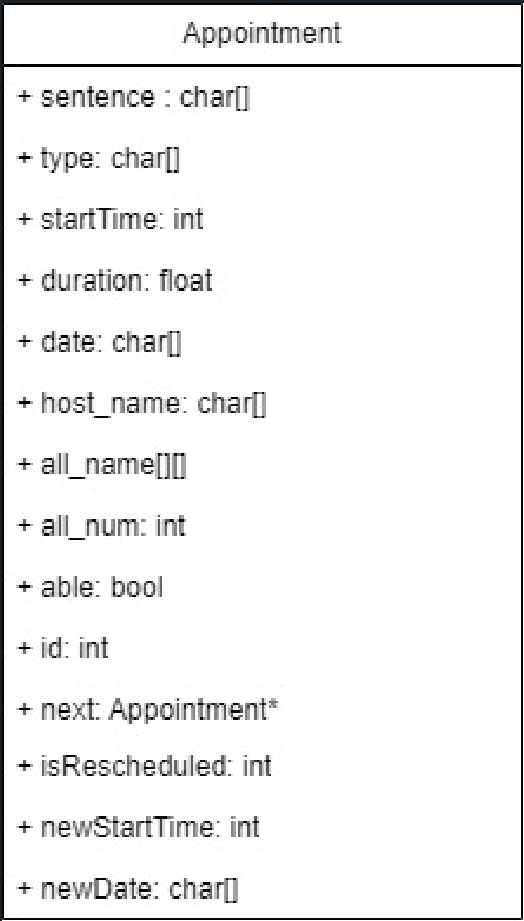
The input processing module is responsible for processing appointment details entered by users in the form of strings. The module takes a string input from the user, which contains appointment details such as the caller, callee, time, and duration. The module then uses string manipulation functions to break down the input string into multiple tokens, each representing a specific detail of the appointment.

To break down the input string into tokens, the input processing module may use string manipulation functions such as **strtok() or sscanf()**. These functions allow the module to split the input string into smaller chunks, based on a specified delimiter or format string.

Once the appointment details have been extracted from a string using the input processing module, the details are typically stored in a structure (struct) that encapsulates all of the relevant appointment information. This structure allows the appointment details to be easily accessed and manipulated by the other modules of the program, such as the scheduling module and the output module.

**5.2.1 Input store structure**

The appointment struct typically contains fields for each of the appointment details, such as the caller, callees, date, time, and duration. These fields are represented using different data types, depending on the needs of the program. Below is the diagram showing the structure:



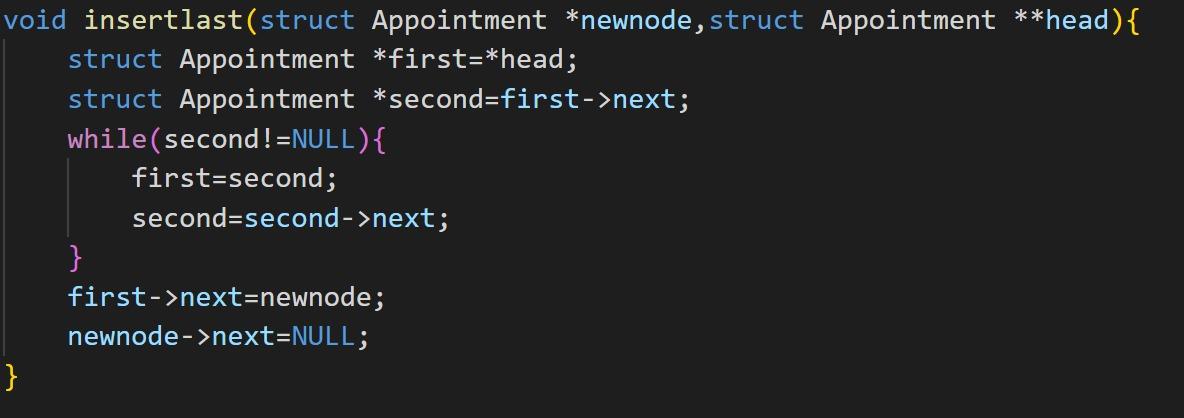
* Char[] “sentence”: this string stores the input appointment.
* Char[] “type”: this string stores the type of the input appointment. 
* Int “startTime”: stores the start time of an appointment as integer type. 
* Float “duration”: stores the duration of an appointment as float type. 
* Char[] “date”: this string stores the date of an appointment as string type. 
* Char[] “host\_name”: store the host of an appointment as string type. 
* Char[][] “all\_name”: this 2D array stores all the participants in an appointment, including the host.
* Int “all\_num”: stores the number of all the participants in an appointment, including the host.
* Bool “able”: this parameter represents whether the event was rejected or not. (This parameter is initialized to **True**)
* Int “id”: this parameter represents the ID of each appointment, which is easy to find in the subsequent use of the chain. (The first appointment comes in will be named as id **1**)
* Appointment\* “next”: This variable is of pointer type and points to the next appointment to facilitate the traversal of the linked list. (This parameter is initialized to NULL)
* Int isRescheduled: this parameter is used as a flag to find if the appointment has already been rescheduled or not, its initial value is 0. (0 means not rescheduled, 1 means rescheduled)
* Int newStartTime: stores the new start time of an appointment as integer type if this appointment has already been rescheduled.
* Char[] newDate: this string stores the new date of an appointment as string type if this appointment has already been rescheduled.

**5.2.2 Input structure creation**



As shown in the screenshot above, after receiving the Appointment, the function will create a node, store the corresponding fields into the corresponding variable names and initialize some variables(i.e. id, next) through the ***strtok()*** function. After that, a node pointer is returned.

**5.2.3 Node insert**

****

After calling the creation function, the main program will call this function to add the newly created node to the end of the chain.

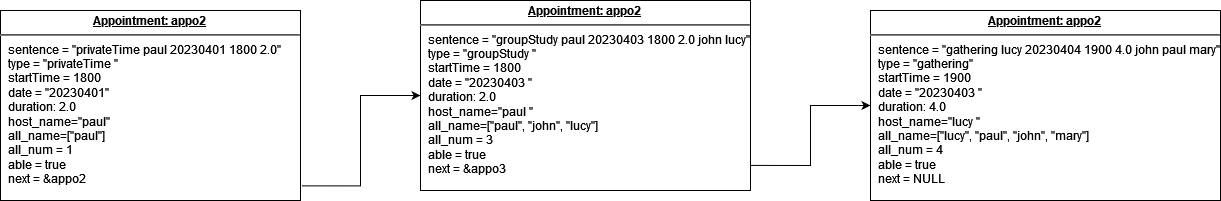
This function takes a node pointer variable and the address of the head node of the link list, finds the tail of this single-item chain table by traversing it, and adds the node to it. Later, we will join each node with the correct ID based on its position in the link list.

**5.3 Linked list**

After appointment details are extracted from a string and stored in a struct, the struct object is appended to **a linked list**. This linked list serves as a data structure for storing appointment details and allows the program to easily search and manipulate appointments as needed.

To implement this data structure, as seen in the struct definition in the previous section, each object of Appointment has a pointer to a next appointment on the list. The first node in the list is referred to as the head of the list, and it is the most important node of the list.

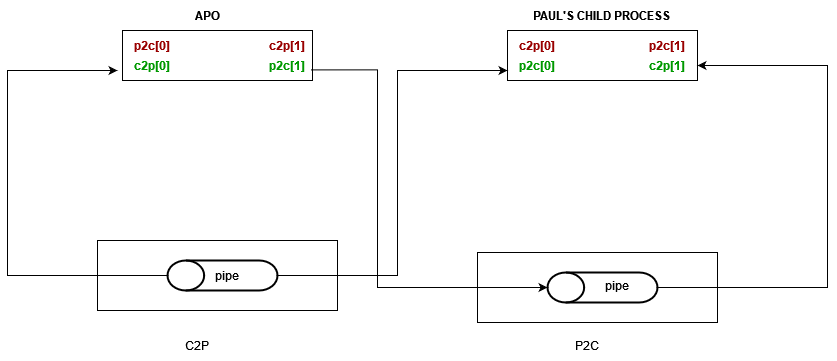
The linked list plays a critical role in the APO, as it allows the program to manage and schedule appointments efficiently. For instance, the scheduling module uses the linked list to iterate through appointment details and send the details to appropriate user processes, while the output module uses the linked list to print out and export details of scheduled appointments.



**5.4 Pipe communication protocols(synchronization)**

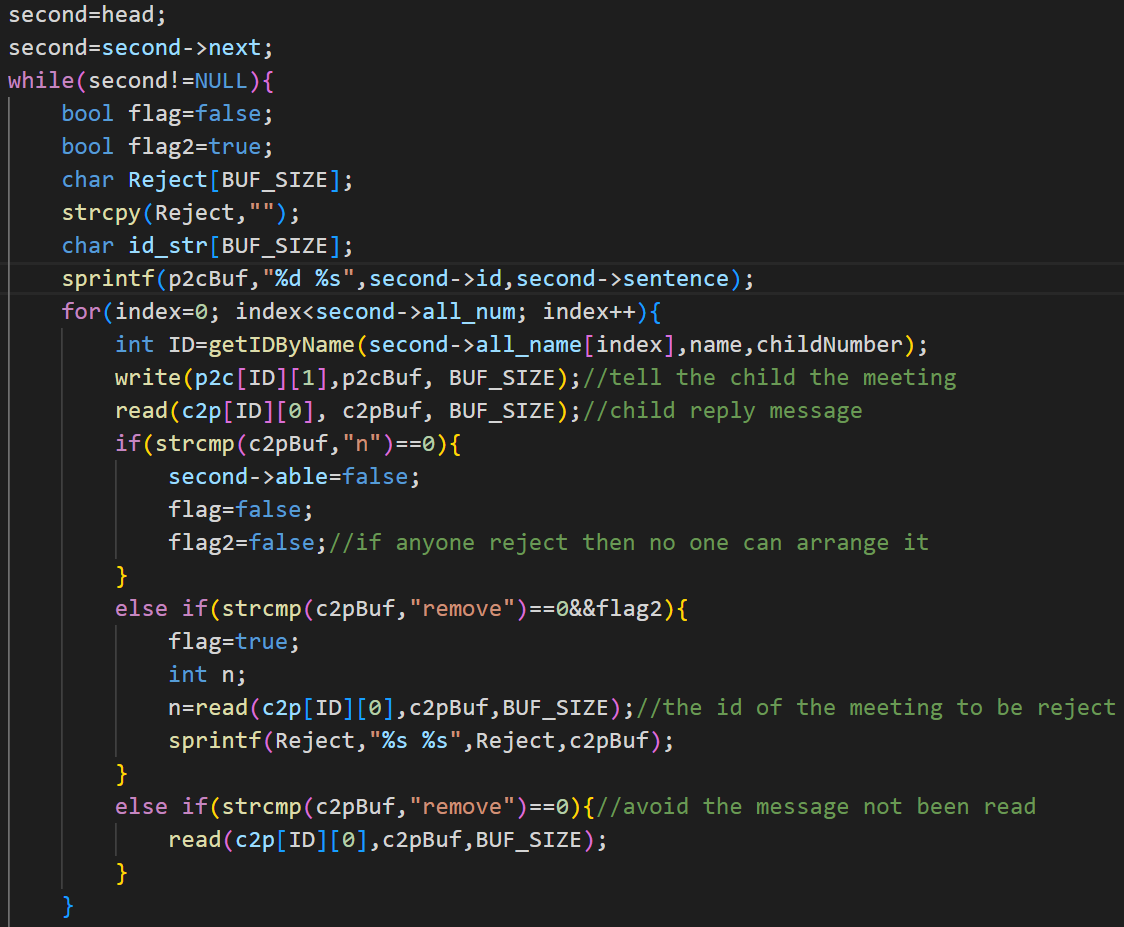
On scheduling command, the parent will loop over the linked list dispatching appointment details to concerned user processes. This communication is achieved by the use of system call, pipe(). With the use of pipes, the application can establish bidirectional communication between parent process and child process. These pipes serve as the communication channel for passing the message from parent to child, and message from child to parent.

Implementation of pipes for interprocess communication followed the standard implementation of having two pipes for duplex communication between child and parent. Let’s assume that a channel is needed for communication between process P, and process C. One pipe, fd1, will be used for communication from P to C, and pipe, fd2, will be used for communication from C to P.

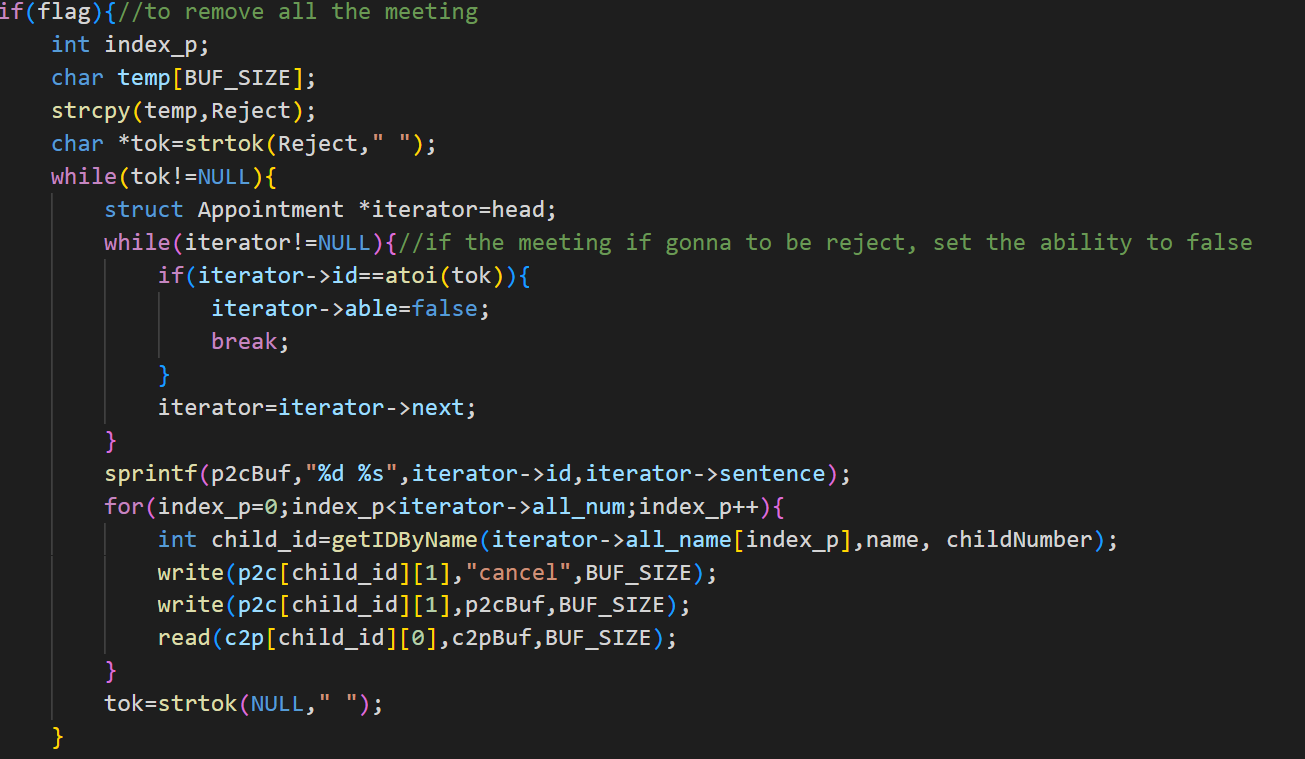


**5.5 Scheduling protocols**

**[PARENT]**

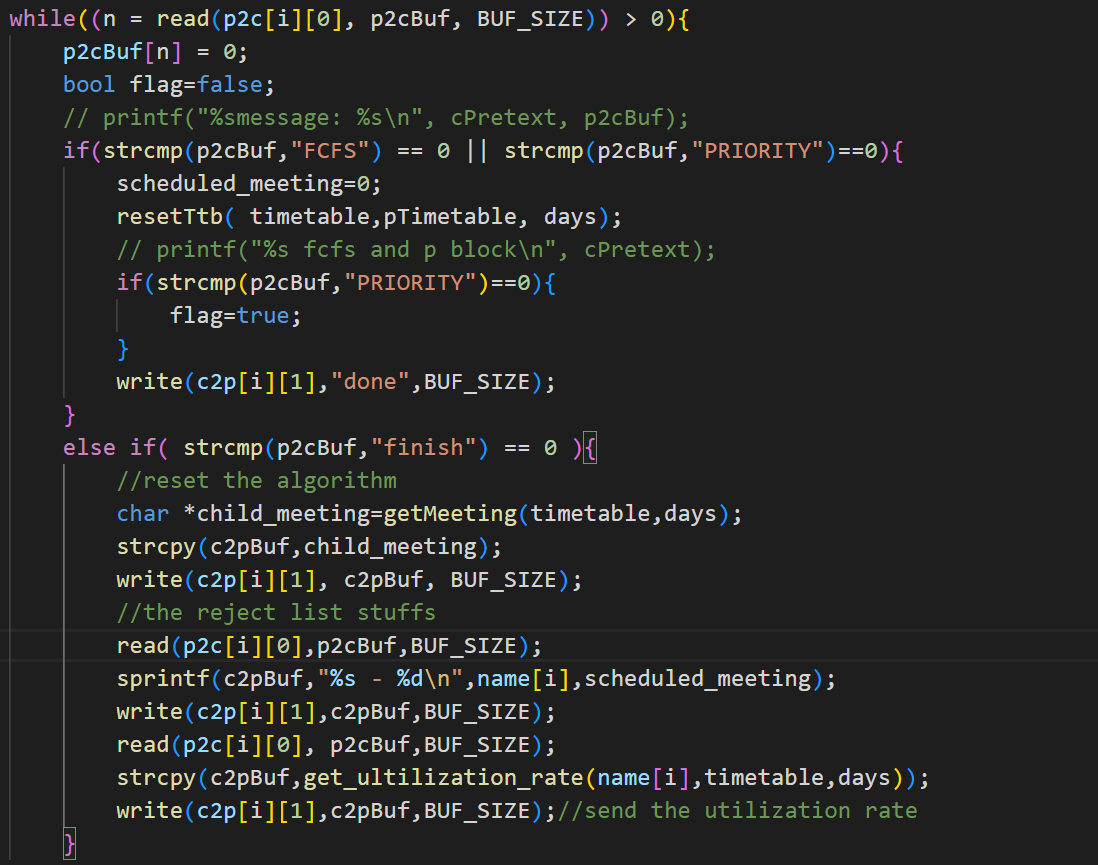


For both of the algorithms, we will iterate the linkedlist, and send the sentence stored in each structure(the information of the Appointment).As the screenshot below shows(The priority part). For the PRIORITY algorithm, if the child wants to remove the meeting which conflicts with the meeting that the parent wants to schedule,it will send a reject list, which contains the ID of the meeting to be removed. We have 2 flags here, flag2 is aimed to detect whether we need to ask the child to put the meeting into the reject list (called Reject), because when someone said that the meeting cannot be scheduled, then there is no way that the meeting can be scheduled.(There must be a higher priority in the meeting). The last else is for removing the message inside of the pipe. For FCFS, the child will only send “y” or “n”, indicating conflict or not.(The picture is above)

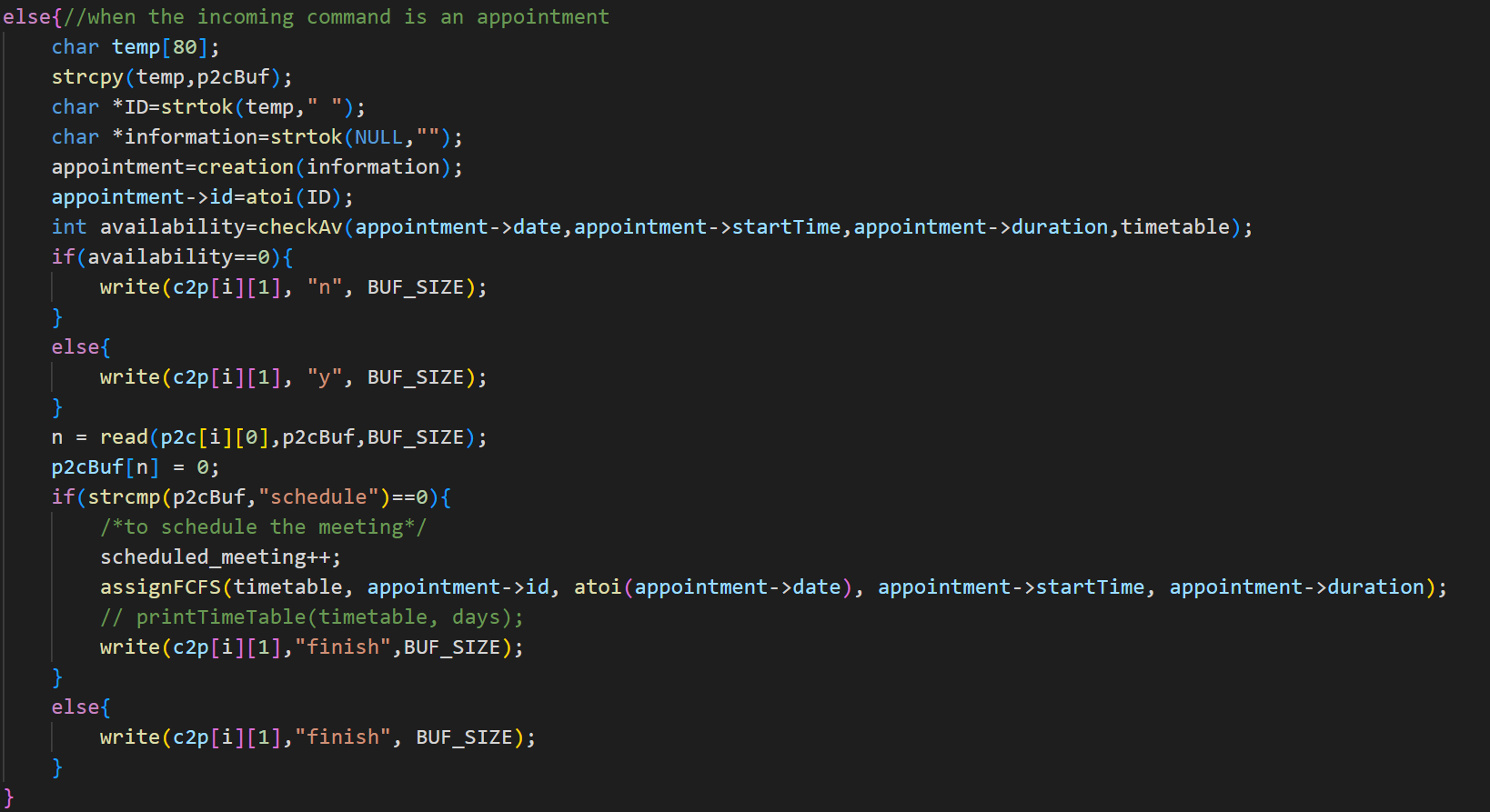


(PRIORITY)When the flag is true, that means we want to schedule the appointment, then we will go through the reject list, according to the id, find the appointment in the list, and inform each child related to the appointment to cancel it.(As the picture above shows)

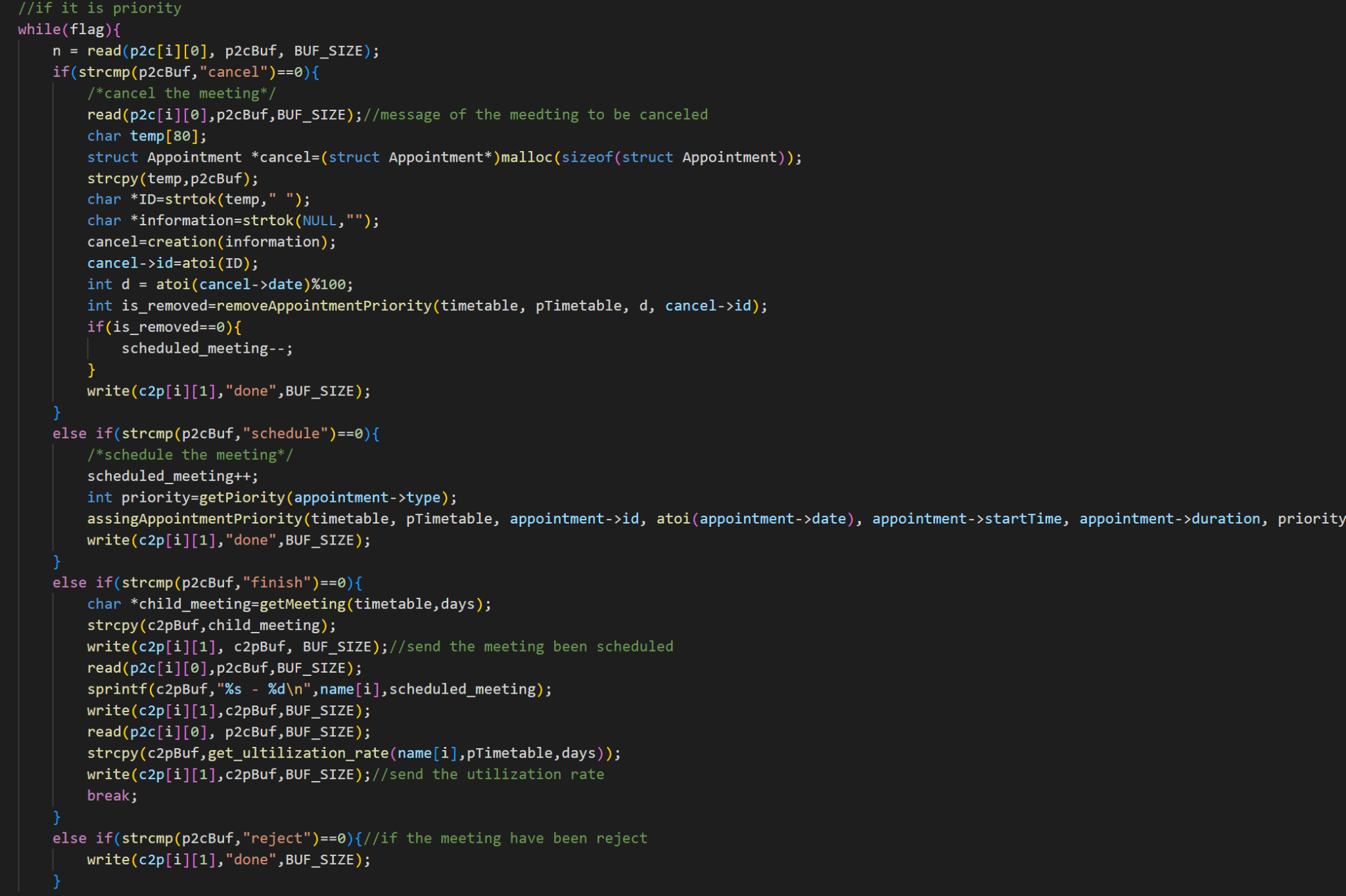
**[CHILD]**



The above chart is to determine the mode to go. If it is PRIORITY, set the flag to true, go to the while loop. And after all the information is sent, which means finish, the child will send all the information to the parent(Utilization of time table,and how many meetings it has).



The above part shows how the child determines whether the child needs to reject the meeting or not**.** If the meeting needs to be rejected by the child, the child sends “n” to parent, if the child is available**,** the child sends “y” to parent.

****

If it is the priority mode, we will ready to receive four kinds of information send from the parent, “cancel”, “Schedule”, “finish”, “reject”, if a meeting is going to be cancel, it will remove the meeting from the time table, or the meeting is going to be scheduled, send the reject list to parent. If you want to reject, just send “n” to the parent.

The above image shows how we are going to receive the information . The only difference is that we will use the PRIORITY time table and the normal time table to check whether the meeting is in conflict. We use a PRIORITY number to detect that(shows in the following image on the left hand side).

**Rescheduling protocol**

When rescheduling mode is activated, the rejected appointment will be automatically rescheduled if there are available common slots to accommodate the appointment.

Here are the pseudocodes for the rescheduling module:

1. When an appointment is rejected:
   * Calculate the number of required time slots for that appointment; for example, duration of 2.5 hrs will take 3 slots[ceil() function was used].
   * Initialize a timetable filled with zeros to be used as an availability map.
   * Prompt all users involved in the appointment to provide their available time slots.
   * Use the availability slots provided by each user to fill the availability map; i.e; if a user is available during a given time slot, the value of that slot in the time table will be incremented by one.
   * Do that for all users.
2. Use the First-Fit algorithm to find the common slot:
   * Initialize a start time variable to the earliest start time among the available time slots of all users.
   * Initialize a counter, c, for contagious free slots.
   * For each time slot in the availability map:
     + Check if all users are available in that time slot; i.e. the value in the availability equal to the number of users associated with the appointment.
     + If yes, increment the counter c
     + If no, set c = 0
     + Move to next slot in the map
     + If c = number of users involved in appointment, record the slotNo and break loop.
     + Loop all slots in the map.
   * If a common slot is found, notify all users involved of the common slot and ask them to schedule the appointment in that slot.
   * If no common slot is found, also notify them that no common slot was found.
3. Update the appointment:

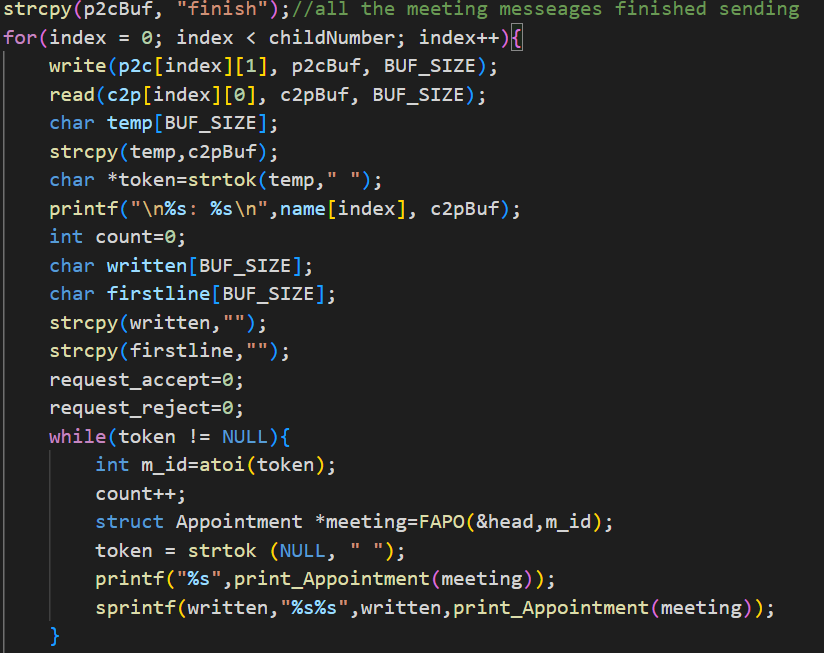
Since all appointments are stored on the list, if the appointment is successfully rescheduled, the details will need to be updated.

* + Update the reschedule flag for the appointment; i.e. appointment->isRescheduled = 1;
  + Updated the fields of newDate and newStart to those of the found common slot.
  + Update the appointment’s flag of successful scheduling; i.e; appointment->able = true.

**5.6 Output module**

On the output module part, we create the file pointers and use the system calls in C - open which help us to open files and use the writing mode, reading mode, or append mode to do some file operations.

**5.6.1 Get appointments by ID**



The above code shows how the parent process obtains the ID of the appointment to be output from the child process and calls related functions to convert it into the correct output format.

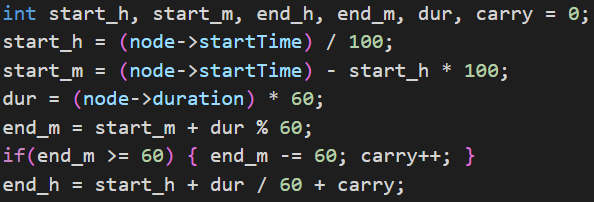
At the start part, the parent will send “finish” to every child (at this moment after the child gets the message from the pipe, it will send back a string consisting of all the confirmed appointment ID.) and then read the feedback from the pipe. Then use ***strtok()*** to get the meeting id one by one, later, use the ***FAPO()*** function to get the node by the meeting ID and then put the node into the function ***print\_Appointment()*** to get the correct output format.

**5.6.2 Transferm output sentence**

****

**The core function used when generate output sentences**

After the program completes the steps in 5.6.1, it will input the appointments found one after another into the above function. The function will take a pointer to a structure and return a sentence to be output and written to a file by calling the information stored in it.



**Part of the code in the print\_appointment function**

As required by the project, this part of the code shows how the function converts the time information (start time, duration) stored in the link table node into the format needed for the output (start time, end time).

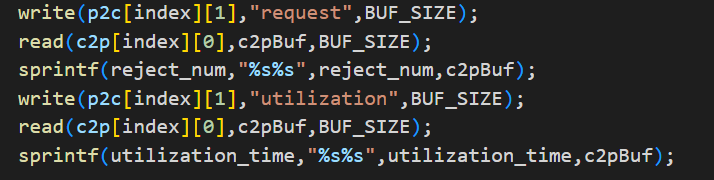




**Two formats of output (first for rescheduled and the second for normal)**

After preparing all the information to be output, the function will use the sprintf() function to synthesize all the information to be output into a single sentence and return this sentence.

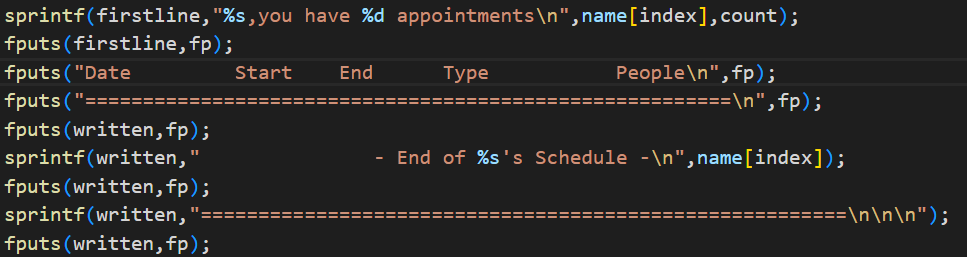
**5.6.3 Write in report file and output**

****

**Pipe communication to get the utilization**

When writing a report file, after the child process has calculated the usage rate, the parent process will obtain the data through pipeline communication and convert it into the correct format, and finally write it into the file (the performance sentences are also stored in the reject file).

The child program will also send back a string consisting of an appointment id which has already been confirmed, and then it will go through the whole link list and return all the nodes it needs and then write them into the report file after changing them into the right format.

****

**The basic format of rejected files written in**

**5.6.4 Write in the reject file and output**

The implementation logic of this part is very similar to part 5.6.3.



**The basic format of rejected files written in**

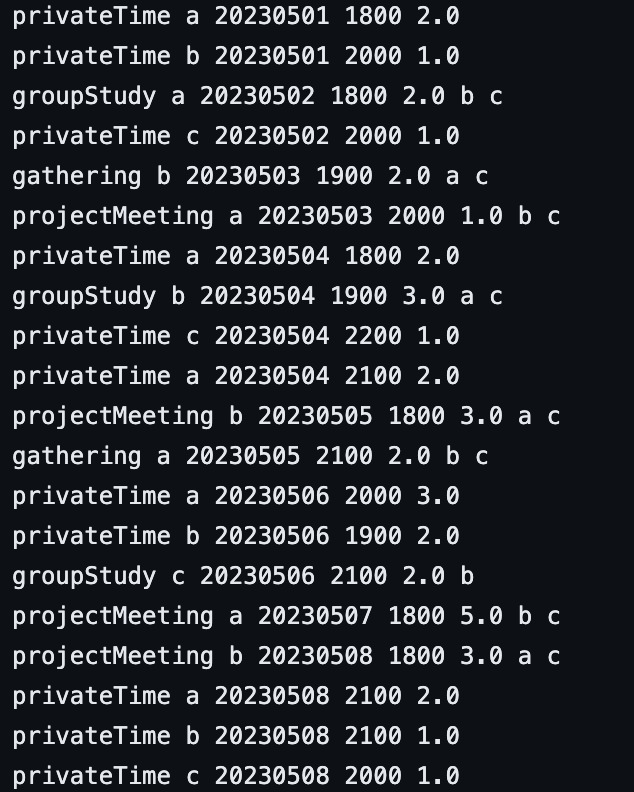
The picture above shows how this code writes information to a file. In this code, the correct format is successfully written to the file by alternating the ***fputs()*** and ***sprintf()*** functions.

Logically speaking, the program will traverse each node in the linked list and check the variable "able", when the value of the variable is false, it means that the appointment has been rejected, it will be selected and written to the rejected file.

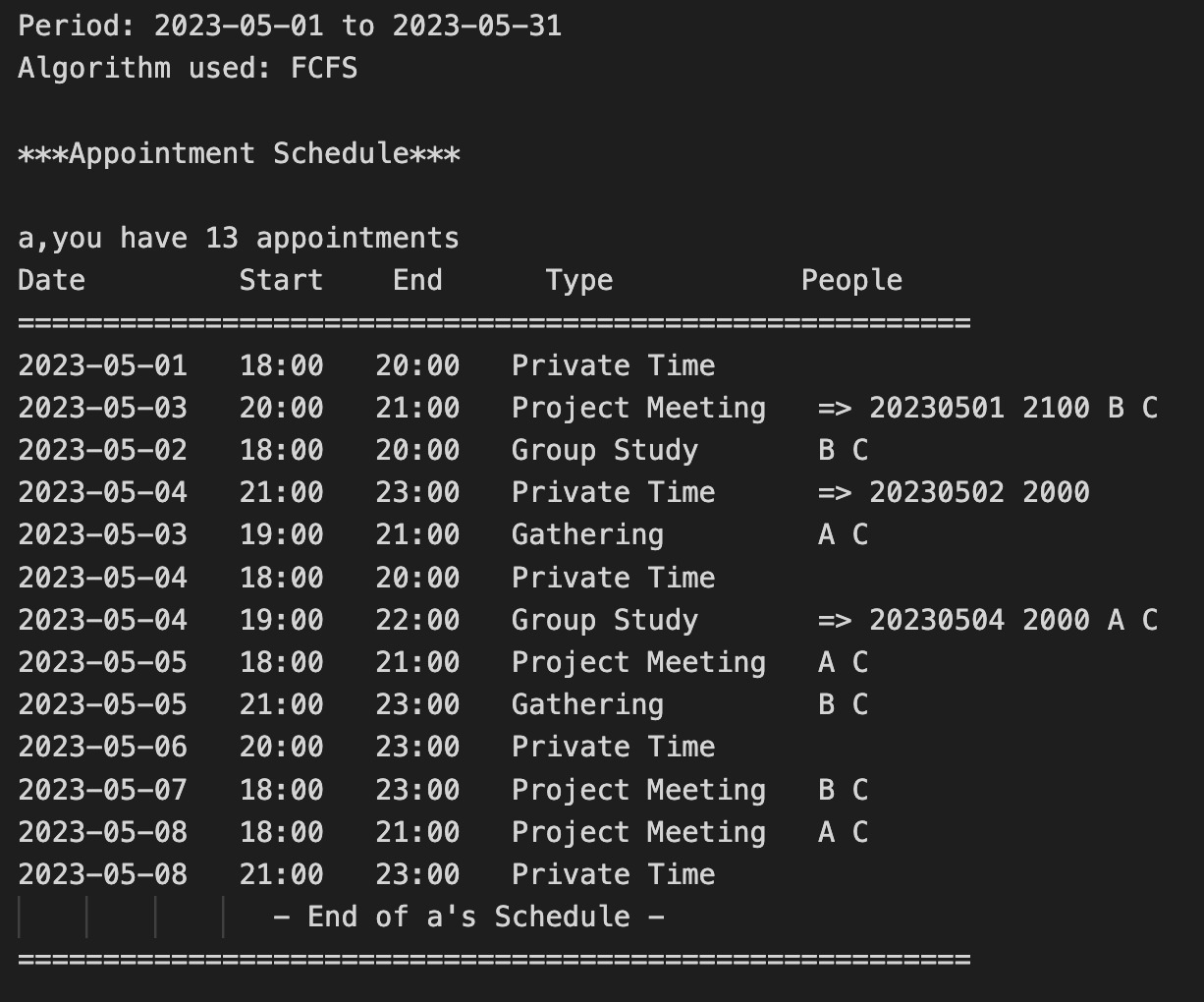
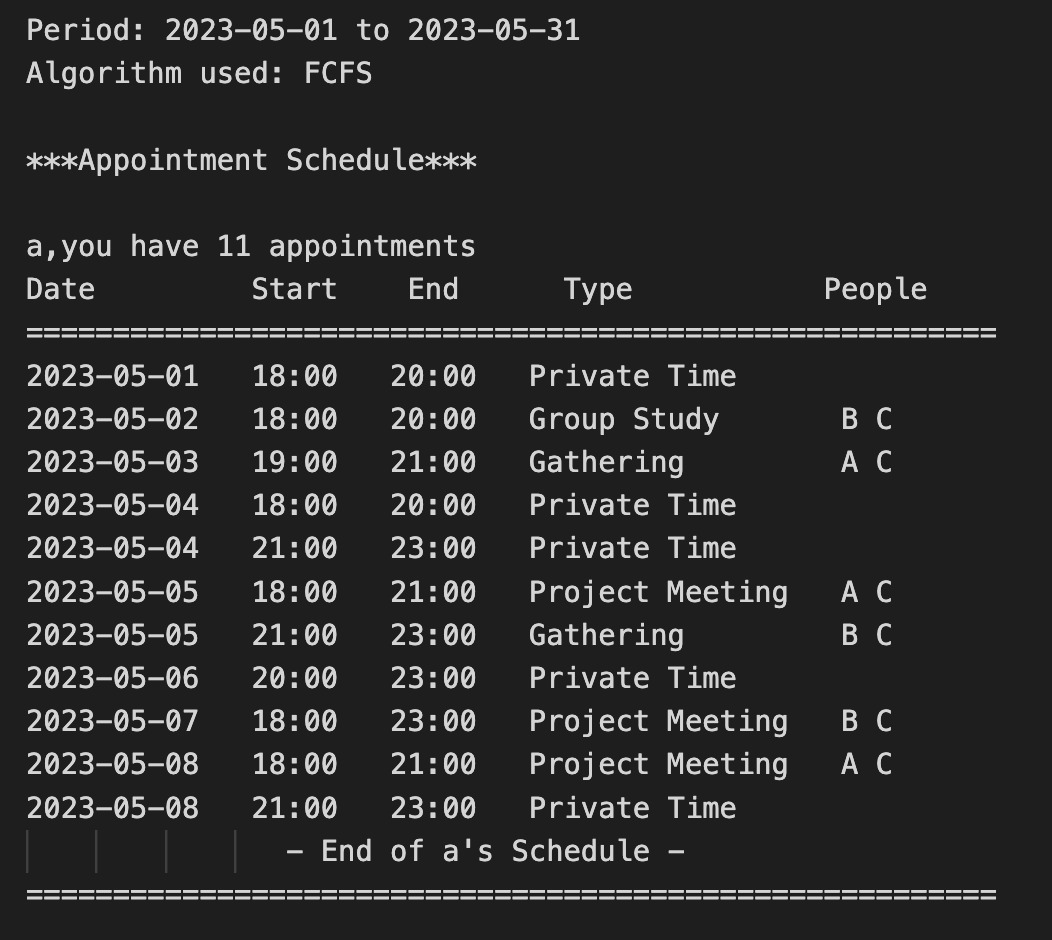
**6. TESTING CASES**

We calculate the timetable using the same data set which used to run the program. The output data files should be the same as the result we calculated. Also, we use some invalid input (for example: less than 3 users) to test the program.

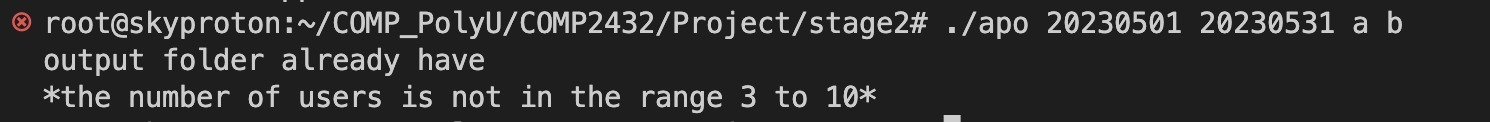
[ Testing Data Samples we made ]

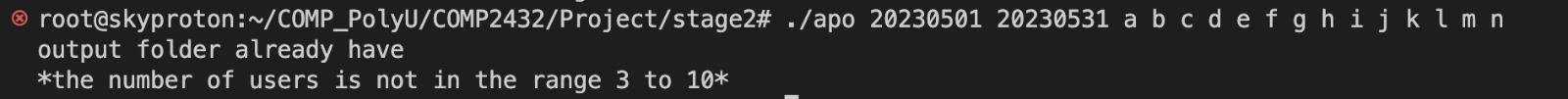


[ Part of the output by using our testing samples (use FCFS schedule & FCFS reschedule) ]



[ Testing the number of users (2 users & 14 users) ]

****

****

**7. PERFORMANCE ANALYSIS**

*(Discuss and analyze the algorithm used in the program. Why is it better than the others?)*

We use the First Come First Serve Scheduling algorithm and Priority Scheduling algorithm in scheduling the appointments, and the first fit algorithm for scheduling the appointments into the timetable. These algorithms were chosen based on the advantages they provide to the users. For instance if the users are looking for simplicity and fairness in scheduling their appointments, FCFS will be a good algorithm to use. On the other hand, if flexibility and customization is required, priority scheduling will be a good fit for the role.

**Pros**

1. **First Come First Served**

* Simplicity: FCFS is a simple scheduling algorithm that is easy to implement and understand. It involves scheduling appointments in the order in which they are received, which makes it easy to manage and operate.
* Fairness: FCFS ensures that appointments are scheduled in the order they are received, which makes it a fair scheduling algorithm as all appointments are treated equally.
* Predictability: FCFS scheduling algorithm is a non-preemptive algorithm which implies that once the appointment is scheduled, it will not be canceled in future under no circumstances, like if priority is considered; therefore it provides a predictable schedule as it schedules appointments in the order they are received.

1. Priority scheduling
   * Flexibility: The priority scheduling algorithm allows the system administrator to assign different priorities to different appointments based on their importance. This helps in managing critical appointments effectively.
   * Efficiency: Priority scheduling algorithm ensures that high priority appointments are scheduled first, which can save time and resources by ensuring that the most important appointments are handled first. For instance, it will prevent the scenario where important appointments like project meetings cannot be scheduled due to the user’s timetable being crowded with less important activities like gatherings and parties.
   * Customization: Priority scheduling algorithm can be customized to meet the specific needs of the appointment management system. This provides flexibility in scheduling appointments based on unique criteria. The function for evaluating whether an appointment has priority over another can be customized based on the user's needs.

**Cons**

Despite the benefits of those algorithms, they have limitations.

1. **First come first served**

* FCFS scheduling may not be ideal for situations where some appointments are more urgent or important than others. For example, if a user needs to schedule an urgent medical appointment, FCFS scheduling may not prioritize this appointment appropriately, potentially leading to delays or other issues.

1. **Priority scheduling**

* Priority-Based Scheduling may not always reflect the true priorities or preferences of users. Users may have different priorities or preferences for scheduling appointments, and it can be difficult to accurately capture these priorities in a single priority level or ranking. For instance, Mary might value family gathering or friend’s gathering over project meetings, whereas Paul regards project meetings as top priority appointments.

This can be overcome by each user setting their own priorities, but with current implementation of this system, the priorities and preference of activities are assumed to be universal across all users.

Both FCFS and Priority-Based Scheduling may not take into account other factors that could affect the scheduling of appointments, such as the availability of specific resources or the complexity of the appointment. For instance, a gathering can get a higher priority depending on who is attending the gathering.

**8. PROGRAM SETUP AND EXECUTION**

**8.1 Compile the program**

Compile the G05\_APO.c file & get the G05\_APO executable file (use -lm because we use the math library in the C file)

gcc G05\_APO.c -o G05\_APO -lm

**8.2 For normal input**

./G05\_APO 20230401 20230430 john mary lucy paul

[ ! 20230401 -> Start Date, 20230430 -> End Date, john mary lucy paul -> Users (!user can replace it) ]

**8.3 For batch file input**

> import G05\_tests.dat

[ ! G05\_tests.dat -> the batch file which includes all the test inputs (!user can replace it) ]

**8.4 Use 4 types of commands to schedule an appointment**

8.4.1 Example (privateTime):

> privateTime paul 20230401 1800 2.0

[ ! paul -> host user, 20230401 1800 -> start time, 2.0 -> time usage ]

8.4.2 Example (projectMeeting):

> projectMeeting john 20230402 1900 2.0 paul mary

[ ! lucy -> host user, 20230402 1900 -> start time, 2.0 -> time usage, paul mary -> other user who join this event ]

8.4.3 Example (groupStudy):

> groupStudy paul 20230403 1800 2.0 john lucy

[ ! lucy -> host user, 20230403 1800 -> start time, 2.0 -> time usage, john lucy -> other user who join this event ]

8.4.4 Example (gathering):

> gathering lucy 20230404 1900 4.0 john paul mary

[ ! lucy -> host user, 20230404 1900 -> start time, 4.0 -> time usage, john paul mary -> other user who join this event ]

**8.5 Use reschedule command to automatically reschedule the appointment if rejected**

If the user want to open the automatically reschedule the appointment function, the user need to input this command

> rescheduling

**8.6 Use printSchd to export the report file (after initial the executable file)**

8.6.1 Example: use FCFS (print the schedules using the FCFS algorithms)

> printSchd FCFS

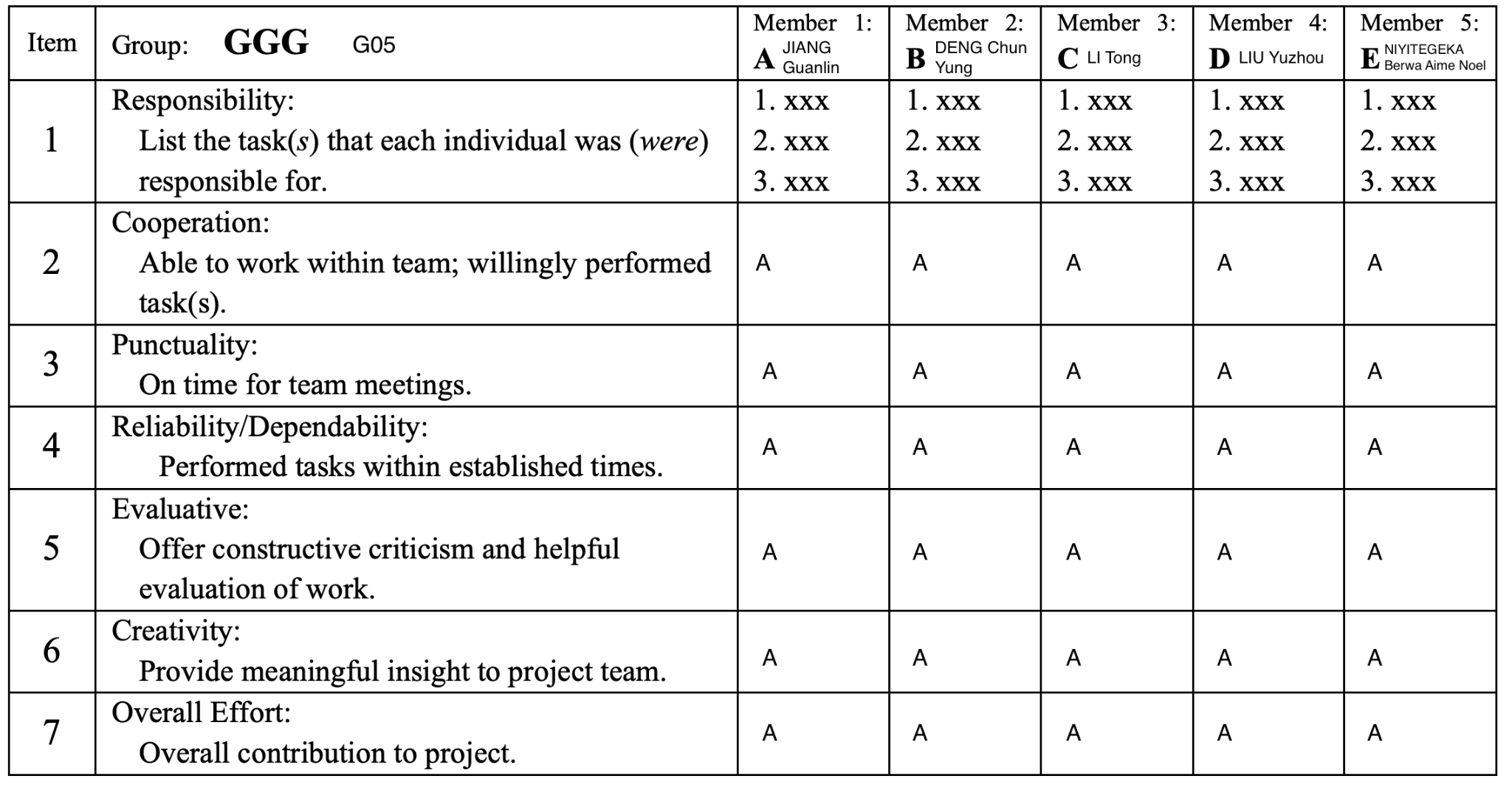
8.6.2 Example: use PRIORITY (print the schedules using the PRIORITY algorithms)

> printSchd PRIORITY

8.6.3 Example: use ALL (print the schedules using all the algorithms)

> printSchd ALL

9. Contribution of individual member



1. Responsibility

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Member 1 | Member 2 | Member 3 | Member 4 | Member 5 |
| Output module  FCFS Scheduling  Append all the code  Input module  Report  PPT | Build the timetable  Pipeline  Fork  Input module  Report | Append all the code  Priority Scheduling  Pipeline  Report  PPT | Output module  Append all the code  LinkedList Building  Data transfer  Report | Append all the code  Reschedule  Priority Scheduling  Pipeline  Report |

**10. Appendix**

**Source Code:** GitHub Link ⇒ https://github.com/COMP2432-G05/Appointment-Organizer.git

Period: 2023-05-01 to 2023-05-31

Algorithm used: FCFS

\*\*\*Appointment Schedule\*\*\*

a,you have 11 appointments

Date Start End Type People

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

2023-05-01 18:00 20:00 Private Time

2023-05-02 18:00 20:00 Group Study B C

2023-05-03 19:00 21:00 Gathering A C

2023-05-04 18:00 20:00 Private Time

2023-05-04 21:00 23:00 Private Time

2023-05-05 18:00 21:00 Project Meeting A C

2023-05-05 21:00 23:00 Gathering B C

2023-05-06 20:00 23:00 Private Time

2023-05-07 18:00 23:00 Project Meeting B C

2023-05-08 18:00 21:00 Project Meeting A C

2023-05-08 21:00 23:00 Private Time

- End of a's Schedule -

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

b,you have 10 appointments

Date Start End Type People

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

2023-05-01 20:00 21:00 Private Time

2023-05-02 18:00 20:00 Group Study B C

2023-05-03 19:00 21:00 Gathering A C

2023-05-05 18:00 21:00 Project Meeting A C

2023-05-05 21:00 23:00 Gathering B C

2023-05-06 19:00 21:00 Private Time

2023-05-06 21:00 23:00 Group Study B

2023-05-07 18:00 23:00 Project Meeting B C

2023-05-08 18:00 21:00 Project Meeting A C

2023-05-08 21:00 22:00 Private Time

- End of b's Schedule -

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

c,you have 9 appointments

Date Start End Type People

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

2023-05-02 18:00 20:00 Group Study B C

2023-05-02 20:00 21:00 Private Time

2023-05-03 19:00 21:00 Gathering A C

2023-05-04 22:00 23:00 Private Time

2023-05-05 18:00 21:00 Project Meeting A C

2023-05-05 21:00 23:00 Gathering B C

2023-05-06 21:00 23:00 Group Study B

2023-05-07 18:00 23:00 Project Meeting B C

2023-05-08 18:00 21:00 Project Meeting A C

- End of c's Schedule -

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

Period: 2023-05-01 to 2023-05-31

Algorithm used: PRIORITY

\*\*\*Appointment Schedule\*\*\*

a,you have 10 appointments

Date Start End Type People

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

2023-05-01 18:00 20:00 Private Time

2023-05-02 18:00 20:00 Group Study B C

2023-05-03 20:00 21:00 Project Meeting B C

2023-05-04 18:00 20:00 Private Time

2023-05-04 21:00 23:00 Private Time

2023-05-05 18:00 21:00 Project Meeting A C

2023-05-05 21:00 23:00 Gathering B C

2023-05-06 20:00 23:00 Private Time

2023-05-07 18:00 23:00 Project Meeting B C

2023-05-08 21:00 23:00 Private Time

- End of a's Schedule -

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

b,you have 9 appointments

Date Start End Type People

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

2023-05-01 20:00 21:00 Private Time

2023-05-02 18:00 20:00 Group Study B C

2023-05-03 20:00 21:00 Project Meeting B C

2023-05-05 18:00 21:00 Project Meeting A C

2023-05-05 21:00 23:00 Gathering B C

2023-05-06 19:00 21:00 Private Time

2023-05-06 21:00 23:00 Group Study B

2023-05-07 18:00 23:00 Project Meeting B C

2023-05-08 21:00 22:00 Private Time

- End of b's Schedule -

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

c,you have 9 appointments

Date Start End Type People

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

2023-05-02 18:00 20:00 Group Study B C

2023-05-02 20:00 21:00 Private Time

2023-05-03 20:00 21:00 Project Meeting B C

2023-05-04 22:00 23:00 Private Time

2023-05-05 18:00 21:00 Project Meeting A C

2023-05-05 21:00 23:00 Gathering B C

2023-05-06 21:00 23:00 Group Study B

2023-05-07 18:00 23:00 Project Meeting B C

2023-05-08 20:00 21:00 Private Time

- End of c's Schedule -

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

\*\*\* Rejected List(FCFS)\*\*\*

Altogether there are 3 appointments rejected.

1. projectMeeting a 20230503 2000 1.0 b c

2. groupStudy b 20230504 1900 3.0 a c

3. privateTime c 20230508 2000 1.0

\*\*\* Performance \*\*\*

Total Number of Requests Received: 20.0(100%)

Number of Requests Accepted: 17.0(85.0%)

Number of Requests Accepted: 3.0(15.0%)

Number of Requests Accepted by Individual:

a - 11

b - 10

c - 9

Utilization of Time Slot:

a - 18.1%

b - 14.8%

c - 13.5%

\*\*\* Rejected List(PRIORITY)\*\*\*

Altogether there are 3 appointments rejected.

1. gathering b 20230503 1900 2.0 a c

2. groupStudy b 20230504 1900 3.0 a c

3. projectMeeting b 20230508 1800 3.0 a c

\*\*\* Performance \*\*\*

Total Number of Requests Received: 20.0(100%)

Number of Requests Accepted: 17.0(85.0%)

Number of Requests Accepted: 3.0(15.0%)

Number of Requests Accepted by Individual:

a - 10

b - 9

c - 9

Utilization of Time Slot:

a - 12.9%

b - 8.4%

c - 7.7%