**Comp 2432 Operating System Project:**

**Project Meeting Scheduler**

**[Contributors]**

Sollal Fouilland 19086822D

Gaukhar Turgambekova 20065366D

Yang Ke 20100884D

**[Video]**

<https://photos.app.goo.gl/t9mHohntrAsX1Gju5>

**[Introduction]**

This project develops a scheduling operating system in C language using modular design and multi-processing. Users can enter data such as team information, and meeting details into the program and receive a schedule based on different scheduling algorithms like First-Come-First-Serve or Priority-based. Both single line input and file batch input are supported by this program.

**[Scope]**

* **Multi-processing**

This program uses a fork() system call to generate three child processes in the main program PMS.c. Each of the child processes handles a module’s work: Input Module, Output Module, and Schedule Module. The Input Module talks to the parent process about whether to process running based on user requests and then the parent process acts as the main control to the other two children and gives instructions to them. It also waits until all child processes exit before it exits to prevent any orphan child processes.

* **Inter-process Communication**

Pipe() is used to implement Inter-process communication. The communication topology is tree-like, which means the Input Module child talks to the parent based on what user input, and then the parent delivers a corresponding message to either the Output Module child or the Schedule Module child. Unnamed pipes are used to deliver information: multidimensional file descriptor arrays are implemented at the beginning and unnecessary ends are closed for each process.

* **Scheduling Algorithms**

There are two scheduling algorithms for users to choose from in this program: first-come-first-serve(FCFS) and Alphabetical.

For FCFS, if any time conflict happens between meetings, the first requested one gets to be accepted while the other one is rejected. A conflict is defined as the situation where any member of the team has a time clash between the requested meeting and a previously assigned meeting.

For the priority-based algorithm please refer to the Scheduling Algorithms part.

* **Process Synchronization**

The global data is kept uniform between processes by pthread.h. The address is locked every time a child tries to write to it and unlock it after change.

* **I/O**

I/O modules are called through user-level libraries like stdin in C language to read in user input and commands. Device management system calls are made to read in and write the scheduling results to files.

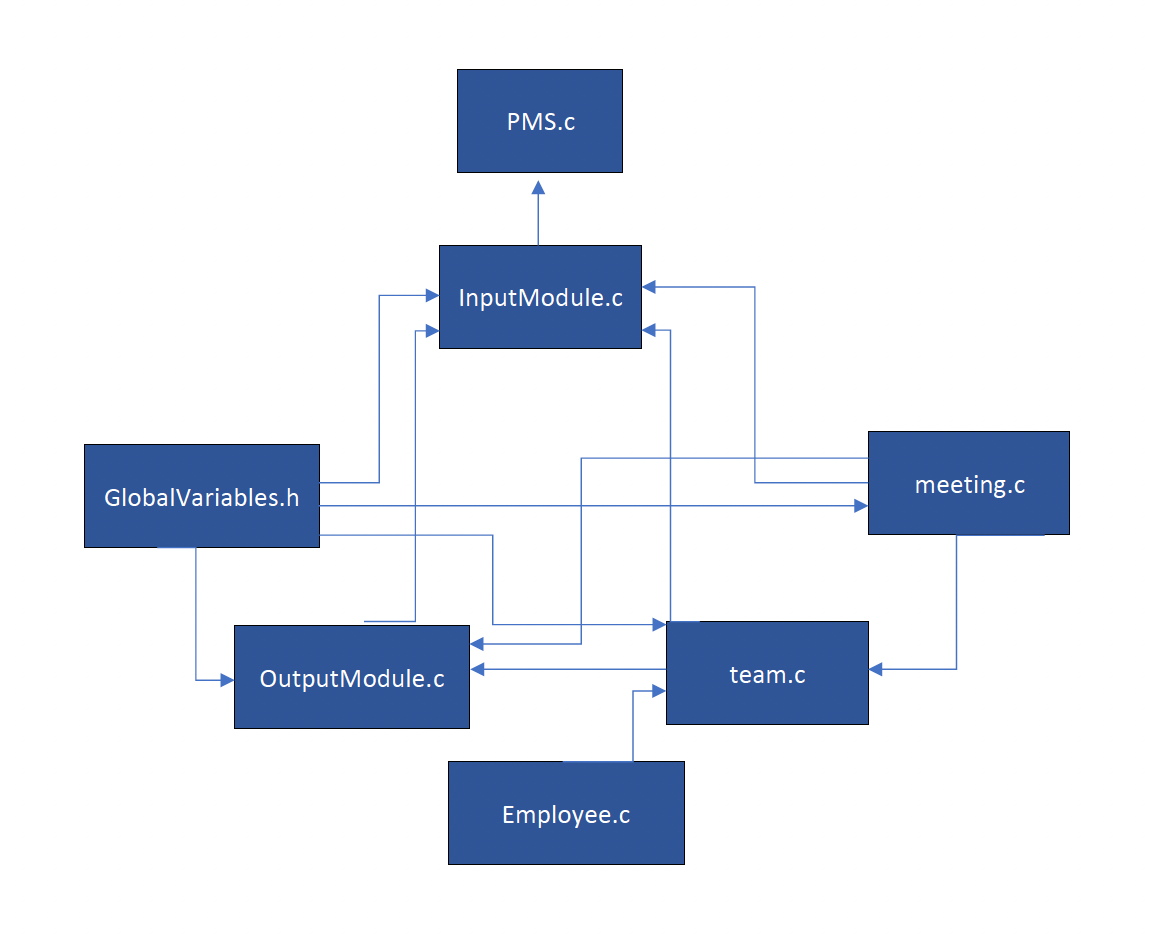
* **File System**

Various types of file operations like open, read and write are executed to support the processing of user input and program output. If any files are being requested by the user as input data, the FILE structure, fopen(), and fclose() functions are called in use.

* **Memory**

Shared memory space is allocated at the beginning of the program, which will act as the memory space for teams, meetings, and employee data.

**[Concept]**

****

**[Scheduling Algorithm]**

As we mentioned before, the first algorithm was first-come-first-serve which did not consider the priorities of the meetings while scheduling, it relies on an index variable stored in the meeting structure that is incremented on creation. We called the second algorithm alphabetical. So the second algorithm implemented considers the name of the teams taking part in the meetings, the teams that have an alphabetically higher name (A>Z) have the priority. The earliest meeting is considered first one by one we reject meetings with lower ranking which intersects with the timing of high-ranked meetings.

**[Software Structure]**

* **PMS.c**

The core program connects Input, Output, and Scheduling modules altogether. It is also where the global variables and defined.

* **InputModule.c**

The program allows two types of input: single line and file batch. The module includes input processing of a meeting or team creation and scheduling algorithms.

Functions used:

* int getStrInput(char \* buffer, int size, FILE \* stream)

Reading line by line from the stream file and saving to the buffer.

* int processTeamCreation(char \* input)

Processing of the input information for further team creation.

* int processMeetingRecording(char \* input)

Processing of the input information for further meeting creation.

* int processBatchInput(char \* fileName)

Processing of the FileName information for batch input usage.

* int processOption3a(char \* input)

Processing of the input for further FSFC or Alphabetical scheduling.

* int processScheduling(FILE \* inputStream)

Coordination between the main menu output and process options using the information provided in the inputStream file.

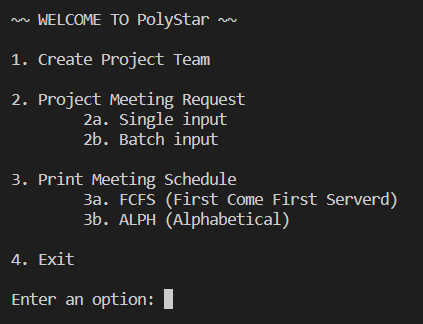
* **OutputModule.c**

This module includes the main user menu and printing of teams or meetings and formatting and prompts for the user input.

Functions used:

* void displayMainMenu ()

Listing all processing options to the user.



* void displayPompt()

Printing “Enter>” before every input for a better user experience.

* void displayDivider()

Printing the line for the same function as displayPompt().

* void printMeetings(struct Meeting\* head)

Printing the meeting from the list for a better coding experience and easier debugging.

* static void printTeams(struct Team\* tail)

Printing the teams from the linked list beginning from the given node tail.

* **SchedulingModule.c**

It includes scheduling algorithms and records of accepted and rejected meetings per employee.

* int generalSchedule(char \* algoName, struct tm start, struct tm end)

Implementation of selected scheduling algorithms considering the conflicts and priority. You can choose algoName, the beginning, and end of the meeting.

* **meeting.c**

This module is created for different manipulations with meetings and includes the structure meeting. It consists of the structure team, time of the meeting, its duration, status, and index.

Functions used:

* static void meetingToString (struct Meeting\* mtng, char outputString[128])

Preparing the meeting object to the format to put into the scheduling file.

* struct Meeting \* scheduleMeeting(struct Meeting\*\* head, char \* teamName, struct tm newTime, int newDuration, int creationIndex)

Scheduling a new meeting at the time period of newTime and newTime+newDuration and inserting the object into the doubly linked list which is in chronological order.

* bool CheckEventConflict(struct Meeting\* MeetingOne, struct Meeting\* MeetingTwo)

Checking the conflict between MeetingOne and MeetingTwo considering the common members between them and time clash.

* **team.c**

This module is created for different manipulations with teams and includes team meetings. It consists of team name, project name, manager, and members.

Functions used:

* static void teamToString(struct Team\* tm, char outputString[128])

Converting the meeting to a string format to put into a scheduling file.

* struct Team \* createTeam(struct Team\*\* head, char \* newTeamName, char \* newProjectName, struct Employee \*\* newMembers)

Creating new meetings using the parameters team name, project name, their manager, and members and putting it to the given double linked list.

* static struct Team\* findTeam(struct Team\* head, char \* tmName)

Finding the team from the given list and with the given name.

* **Employee.c**

This module includes the structure Employee (includes members’ names) and the function for adding the new members.

Functions used:

* struct Employee \* recordEmployee(struct Employee\*\* head, char \* newName)

Adding employee named newName to the given list.

* **GlobalVariables.h**

This module includes the declaration of the global variables used in other modules like the length of the team name or a maximum number of employees for a team. The variable are only define in PMS when the program is started. It is implemented for a better coding experience and a coherent flow of the program.

**[Testing Cases]**

For the testing the program, we implemented test\_input.dat file that has examples of teams and meetings creation. Error checking includes testing for time outside of working day (before 9am or after 6 pm) and for conflicting situation. The conflict happens when teams or the members who are in several teams do not have meetings’ time intersection. We made sure to test for conflicts that gave different results depending on the scheduling algorithm. The use of the batch file allowed us to test for large volumes of meetings and complex composition of teams. We checked the main conflict detection functions by making sure that we not only had conflicting meeting from the same team but also from different teams sharing a member. We checked that the meeting recording function worked on a large time period and for big volumes.

**[Performance Analysis]**

As it was mentioned before, for scheduling we implemented Alphabetical and First-Come-First-Serve algorithms. Because both of those algorithms are used to evaluate meeting in chronological order, they tended to place the same number of meetings. However, the Alphabetical priority scheduler has some useful applications depending on the use of the program. Teams can be named by their “rank” with teams overviewing projects of higher importance having alphabetically higher-ranked names. This makes the identification of the importance of the teams easier without having to rely on a hidden index or value. Overall it provides a more human-centered experience.

**[Program set up and execution]**

The execution process details:

Compile command: gcc G20\_PMS.c -o PMS -std=gnu99 -lpthread

Run command: ./PMS

We tested the program on apollo.

**[Individual member contribution]**

| **Item** | **Group20** | **Member A:**  **Sollal** | **Member B:**  **Gaukhar** | **Member C:**  **Yang** |
| --- | --- | --- | --- | --- |
| **1** | **Responsibility** | **Scheduling input, output functions** | | **Pipe and fork system** |
| **2** | **Cooperation** | **A** | **A** | **A** |
| **3** | **Punctuality** | **A** | **A** | **A** |
| **4** | **Reability/Dependability** | **A** | **A** | **A** |
| **5** | **Evaluative** | **A** | **A** | **A** |
| **6** | **Creativity** | **A** | **A** | **A** |
| **7** | **Overall Effort** | **A** | **A** | **A** |