## INTRODUCTION

This project aims to simulate a scheduler system using C language. The scheduling algorithm prioritizes processes based on their priority levels and adjusts their execution according to predefined quantum values and process types. The program is tested on Ubuntu 22.04.

## IMPLEMENTATION

Two main data structures are used throughout the program: Process and Node. Node structure is used to create a priority queue, which is implemented by using linked lists. The process structure holds information about processes (arrival time, priority, last executed instruction, type, etc.).

Inputs are taken as files. In the beginning, the program reads process files and instruction files and then stores the necessary information. After that, it reads the definition file, which provides information about process types, arrival times, and priorities. The output is directly printed to the console.

The program enters a main simulation loop. The loop continues until the last process exits from the system. Inside the loop, the current time, the current process ID, and various variables to track execution details are initialized.

The algorithm starts with checking if the current process exceeded its time quantum or reached the exit status. Then, it updates the arrival time, increments the quantum count, and resets the execution time.

If the quantum count exceeds the predefined limit, the program performs type conversions. Unless it is an exit instruction, it reorders the ready queue (due to the mentioned updates, the order of the queue might be ruined). If it is an exit instruction, the program pops the process from the queue.

The program scans all active processes and adds them to the ready queue if their arrival time has been reached, and they have not been pushed onto the queue before. After the addition of new processes is done, the algorithm compares the current head of the queue to the head at the beginning of the iteration. If they are different, that means there has been a context switch and time is incremented by 10.

Instructions are executed by using the information stored in the process structure. The structure holds the last executed instruction index, by using the index the required time for the corresponding instruction is determined. Then the program update the process's execution time, total execution time, last executed instruction and advances the time.

Upon completion of the main loop, the program calculates the waiting and turnaround times for each process, providing a total average for both of them. Floating-point results are rounded to one decimal point.

## CONCLUSION

The hardest part was debugging. It is very hard to determine at which step the algorithm fails. I think using a priority queue was a very sensible choice, building the algorithm on it helped me to progress faster (at least in the beginning). The platinum processes create a lot of problems, most of the bugs are caused by platinum processes or the processes that are turning to platinum. In short, there were lots of edge cases and I spent most of the time debugging them.