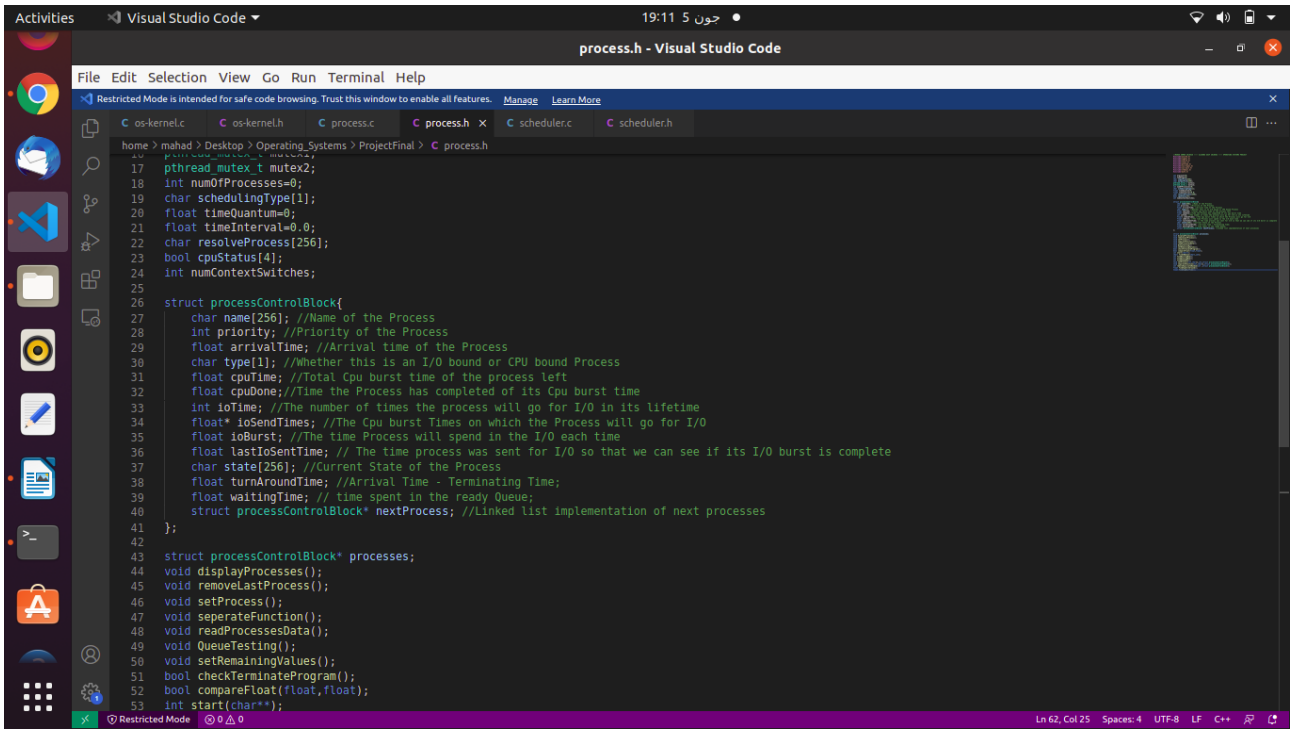
SPARK KERNEL – OPERATING SYSTEMS PROJECT

Mahad Ahmed – 20i0426
Alishba Asif – 20i0582

As per our requirement in the course we have created a spark kernel to replicate the working of a real operating system and how it handles different processes. Our Spark kernel works for 1 CPU and can operate on a range of numerours processes with different properties. Our spark Kernel can handle 3 different scheduling algorithms – First Come First Serve , Round Robin, and Priority Scheduling, much like a real Operating system. We have implemented this using C language and available PTHREAD library.

Step 1: Initialising PCB and Queues, Command Line Inputs

We have made our PCB linked list base which will hold different attributes required to run our scheduling algorithm.



We then moved on to make our Ready Queues. We needed two Queues for this project. One would be a normal FCFS ready Queue used for FCFS and Round Robin, while another would be a priority based ready queue for Priority Scheduling. Our I/O queue is also formed as a normal FCFS Queue. Command line Inputs are read by our Program and saved in variables.

Step 2: File Reading And Initializing PCB

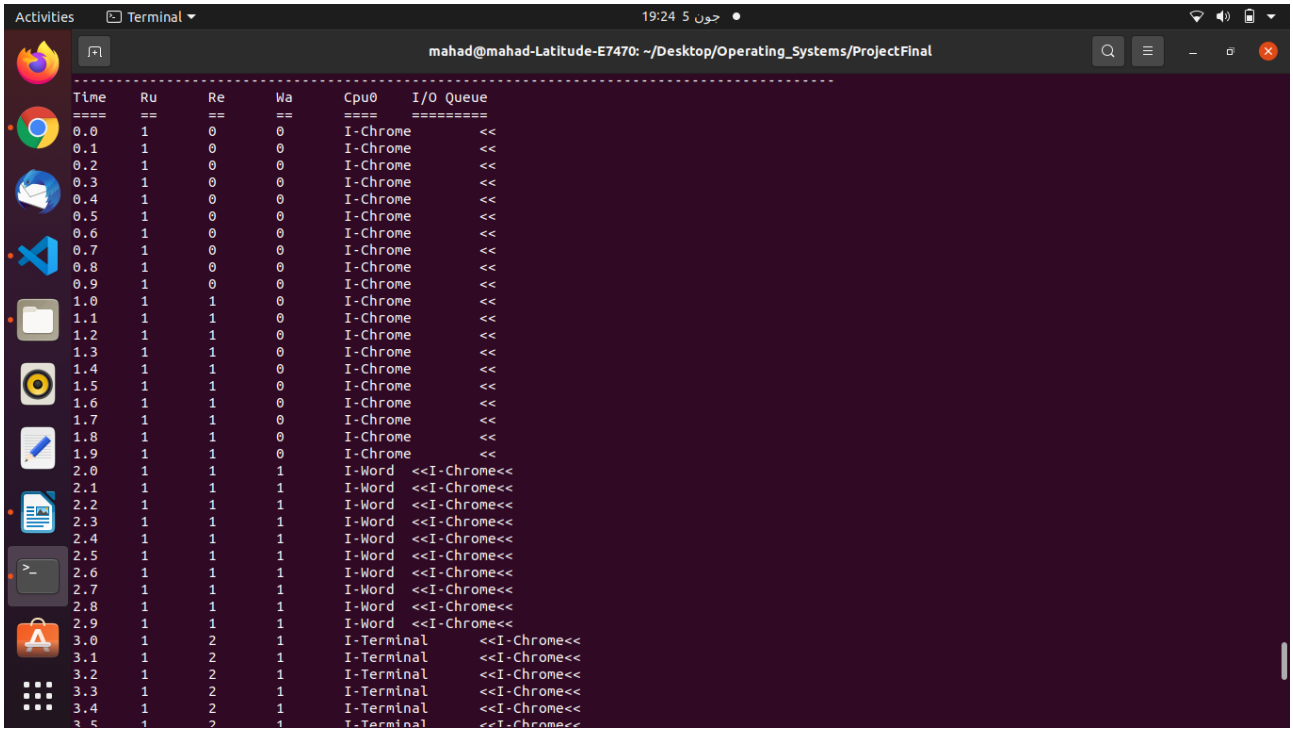
We used FILE* __ datatype to read our file in C. We used a self made parser function to parse a line from the file and assigned the attributes to the processes accordingly. Our Reader also reads the 2nd format file we were given. The attributes that are left out are assigned randomly ourselves.

Step3: Initializing threads and Working of Scheduler

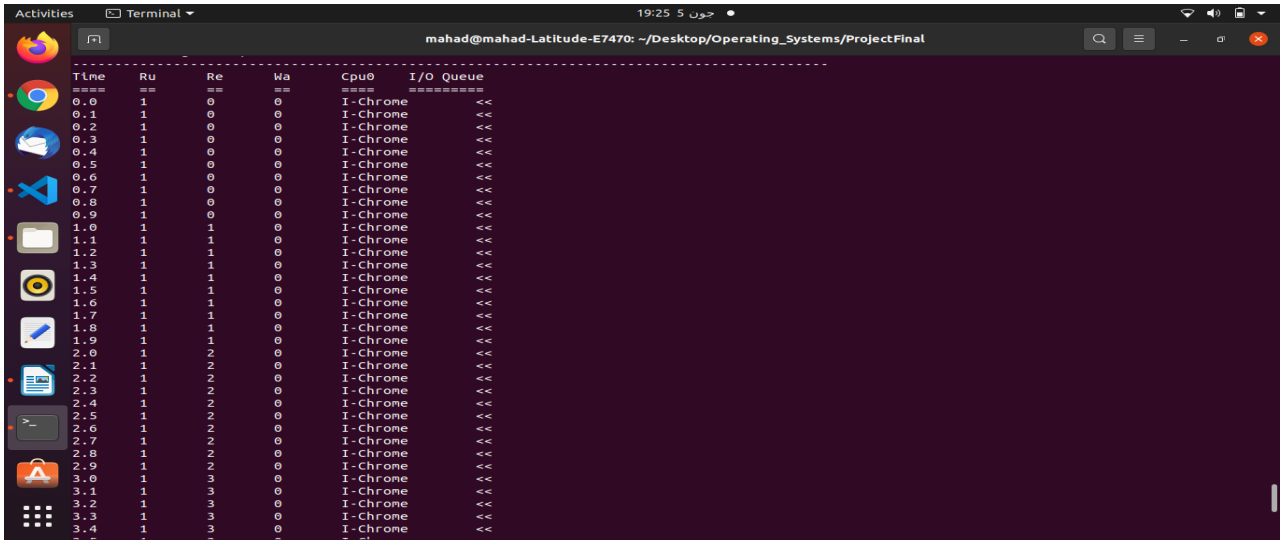
Our Main function creates a Controller thread which further creates CPU threads. First Wake_up is called to start the running where a process is received which has its arrival time and sends it to ready Queue. Scheduler function then reads from the ready Queue and context switches the process giving the timeslice. In FCFS and Priority the timeslice is -1 so the process keeps running without any timeslice break condition whereas in Round Robin it will break when it has run for its timeslice. Once it has run for this interval, a synchronize function is called where if there are more CPUs to run in this interval this CPU must wait until the rest have run, then the last CPU will increment time and give a signal to the rest of the CPUs to also continue. We were unable to complete the implementation of multiple CPUs but our code works on 1 CPU. After returning from synchronise it will return back to context_switch() and run for its time slice. In Priority Scheduling a force_preempt schedule can change process if a more important one arrives / returns from I/O. Similarly a process can be taken off the I/O if it needs to Yield or Terminate. The Controller thread keeps all this running till all Processes have terminated. While this is going on we continuously collect data and keep track of processes.

Step4: Calculations and Output

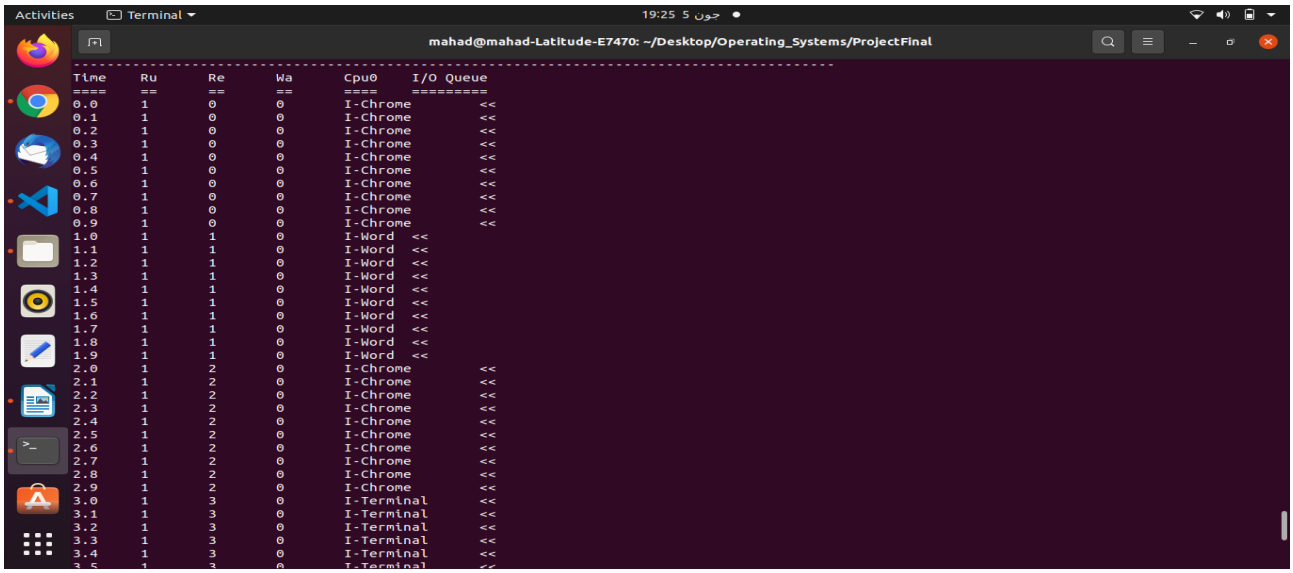
Our results are outputted both to console and to the file in the same format as given in the Question PDF. We also calculate number of Running, Ready, and Waiting Processes simply by checking state of each process in the program at each time interval and counting them. At the end our processes hold their data for total turnaround time and waiting time which we then send to a function and calculate average for the scheduling algorithm.



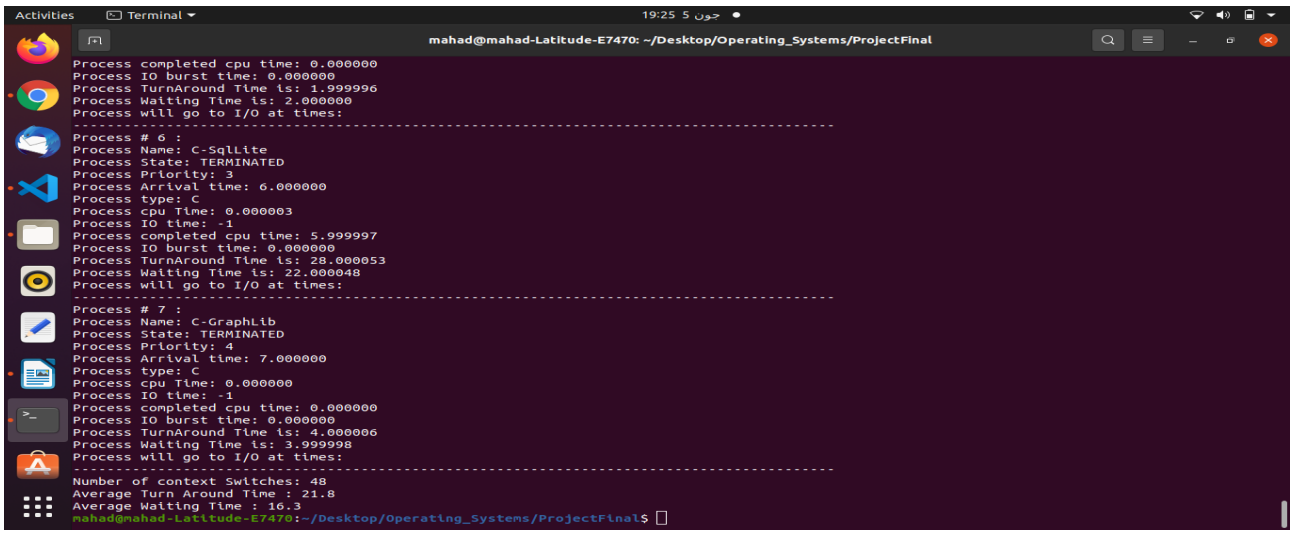
Example Output of Priority Scheduling



Example Output of FCFS



Example Output of RR with timeslice 1



At the bottom we can see results of calculations