ASSIGNMENT 8 - MULTILEVEL QUEUE SCHEDULING

22000862 – R.K.J.P.KASHMIRA

- ## The Problem

  Write a program to simulate a multilevel queue scheduling algorithm with 4 queues. Each queue must be assigned a priority, with q0 having the highest priority and q3 having the lowest priority. The following scheduling algorithms should be used for each queue: ● q0 - Round Robin (RR)

  ● q1 - Shortest Job First (SJF)

  ● q2 - Shortest Job First (SJF)

  ● q3 - First-In-First-Out (FIFO) Each queue should be given a time quantum of 20 seconds, and the CPU should switch between queues after every 20 seconds. The user should be prompted to enter the number of processes along with their priority associated with each queue.

  Task: 1. Implement the scheduling algorithm in a programming language of your choice.

  2. Test the program with different numbers of processes and different priorities. 3. Analyze the results and compare the performance of each scheduling algorithm using following criteria.

  a. Waiting time

  b. Turnaround time

- ## The Solution I made….

  Round Robin :- Round Robin is a scheduling algorithm that assigns a fixed time slice to each process in a cyclic manner.It ensures fairness by allowing each process to take turns using the CPU, preventing any single process from monopolizing resources for too long.

  Shortest Job First (SJF) :- Shortest Job First (SJF) is a scheduling algorithm that prioritizes processes based on their expected burst time,selecting the one with the shortest time first for execution.

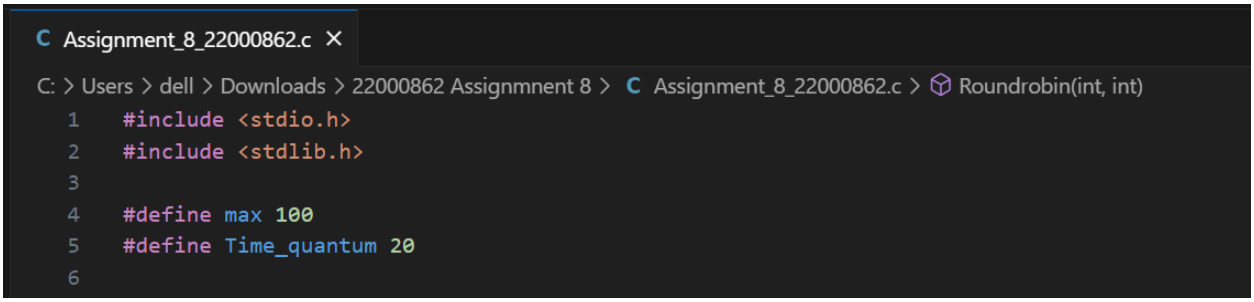
  First-In-First-Out (FIFO) :- FIFO (First-In-First-Out) is a queuing method where the first item added is the first to be processed

In this case the program uses three different scheduling algorithms for processes in different priority queues.first one is Round robin the second and third ones are SJF's(Shortest Job First) and fourth one is an traditional First in First out queue.

- # Implimentation.....

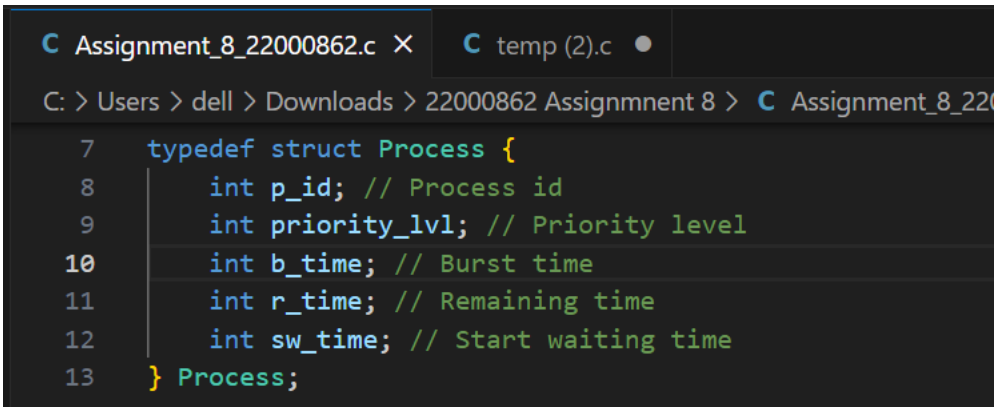
I utilized the C programming language to develop this program.

1. Header Files and Constants



I included the necessary header files (*stdio.h* and *stdlib.h*) and defined constants such as the maximum number of processes (*max*) and time quantum (*Time_quantum*) for the program's round robin algorithm.

2. Process Struct

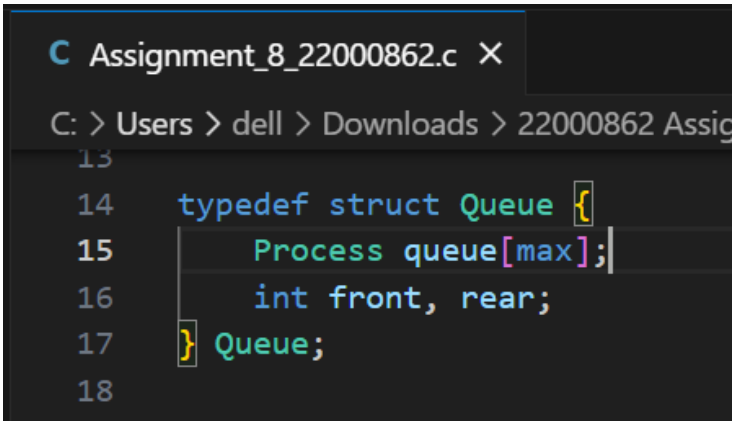


Here,I'm defining the Process structure.It includes the following attributes:

- p_id : Process id, which uniquely identifies a process.
- priority_lvl : Priority level,indicating the priority of the process in a priority scheduling algorithm.
- b_time : Burst time,representing the time required for the process to complete its execution.
- r_time : Remaining time,representing the time remaining for the process to complete its execution.
- sw_time : Represents the start waiting time of the process.

This structure allows us to store and manipulate information about processes.

3.Queue Struct



I'm defining another structure, Queue, to represent a queue of processes.It contains an array of Process structures and two integer variables,front and rear,to keep track of the queue's front and rear.

```
Queue queues[4]; //Queues
```

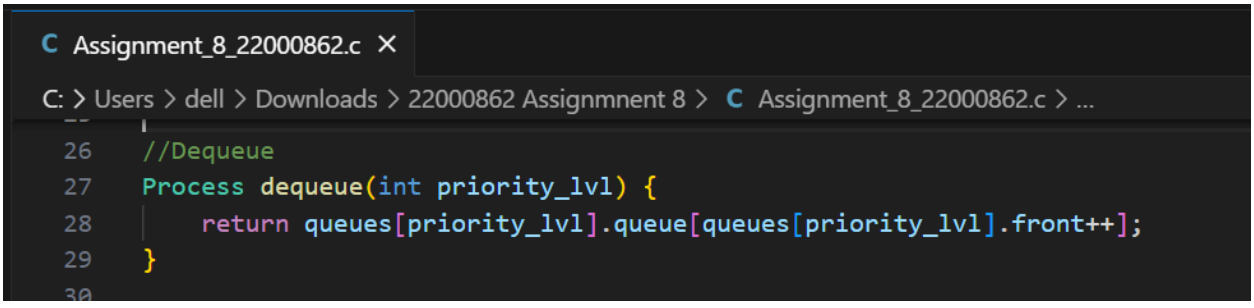
Here,I declare an array of Queue structures named queues.It will hold four queues,each corresponding to a different priority level

4.Enqueue

```
//Enqueue
void enqueue(int priority_lvl, Process process) {
    queues[priority_lvl].queue[queues[priority_lvl].rear++] = process;
}
```

This function enqueue adds a process to the queue corresponding to its priority level.It takes the priority level and the process to be added as parameters and places the process at the rear of the relavant queue.

4.Denqueue



I implement a function dequeue that removes and returns a process from the front of the queue.

5. Round Robin Function

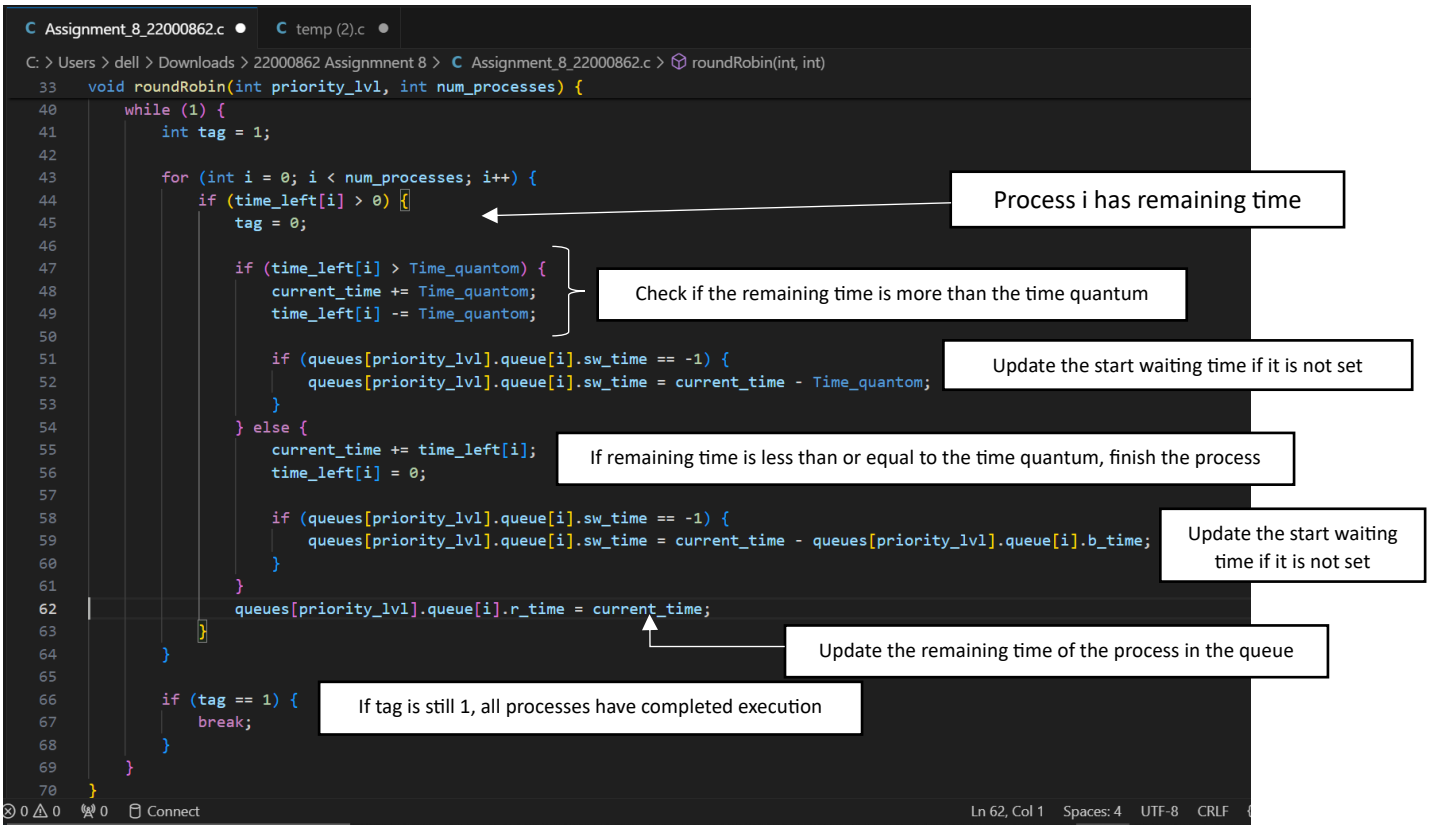
5.1. Initialize time_left Array :- This array is used to keep track of the remaining time for each process in the current priority level queue.

```
30
31  //Round Robin
32  void Roundrobin(int priority_lvl, int num_processes) {
33      int time_left[max];
34      for (int i = 0; i < num_processes; i++) {
35          time_left[i] = queues[priority_lvl].queue[i].r_time;
36      }
```

5.2 Set up current time :- Initialize current_time to 0,which represents the overall time elapsed during the execution of processes.

```
38      int current_time = 0;
```

5.3 While Loop :- The main loop continues until all processes have completed their execution.The loop checks if any process still has time left (time_left[i] > 0).If at least one process has time left, tag is set to 0,indicating that the execution is not finished.



6. Shortest Job First(SJF) Function

6.1 Sort the processes based on Burst Time :- This part implements a simple Bubble Sort algorithm to arrange processes in ascending order based on their burst times. The idea is to prioritize the process with the shortest burst time first.

Users > dell > Downloads > 22000862 Assignmnent 8 > C Assignment_8_22000862.c > ...

```c
//Shortest job First
void Shortestjobfirst(int priority_lvl, int num_processes) {
    for (int i = 0; i < num_processes - 1; i++) {
        for (int j = 0; j < num_processes - i - 1; j++) {
            if (queues[priority_lvl].queue[j].b_time > queues[priority_lvl].queue[j + 1].b_time) {
                Process temp = queues[priority_lvl].queue[j];
                queues[priority_lvl].queue[j] = queues[priority_lvl].queue[j + 1];
                queues[priority_lvl].queue[j + 1] = temp;
            }
        }
    }
```

Swap the processes if the burst time of the current process is greater than the next one

6.2 Calculate Remaining Time for each process :- After sorting the processes, this part of the function calculates the remaining time for each process.The *'r_time*' (remaining time) attribute of each process in the queue is updated based on the cumulative burst times of the previously processed processes.The *'current_time*' variable keeps track of the total time spent so far.

C Assignment_8_22000862.c ●

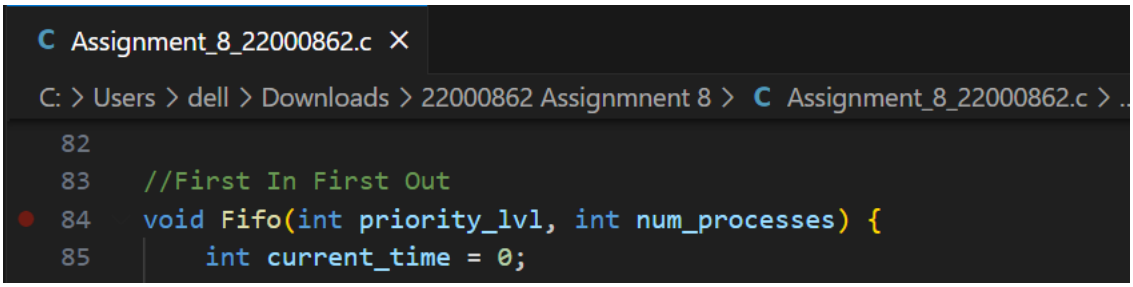
C: > Users > dell > Downloads > 22000862 Assignmnent 8 > C Assignment_8_22000862.c > main()

```c
73   void shortestJobFirst(int priority_lvl, int num_processes) {
83       int current_time = 0;
84       for (int i = 0; i < num_processes; i++) {
85           queues[priority_lvl].queue[i].r_time = current_time; // Calculating remaining time
86           current_time += queues[priority_lvl].queue[i].b_time;
87           queues[priority_lvl].queue[i].sw_time = current_time - queues[priority_lvl].queue[i].b_time;
88       }
89   }
```

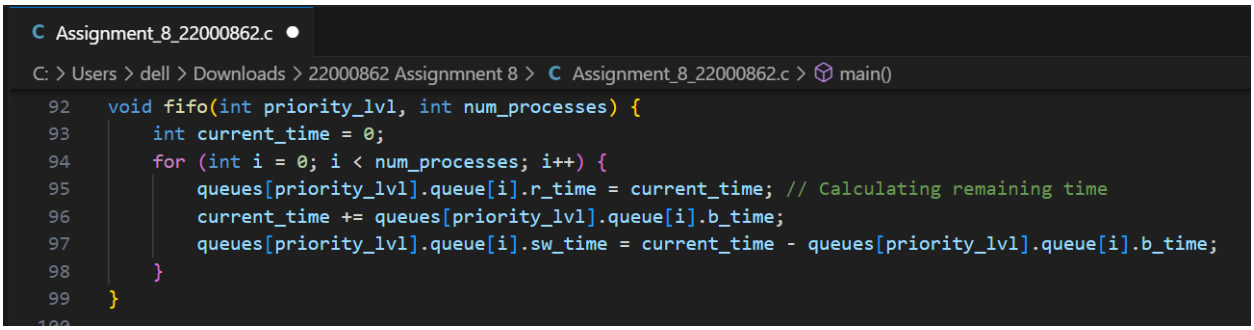
In summary, The shortestJobFirst function sorts the processes based on their burst times in ascending order.After sorting,it calculates the remaining time for each process,assuming that the processes are executed in the order determined by their burst times (shortest job first).This function is essential for implementing the Shortest Job First scheduling algorithm,where the process with the shortest burst time is given the highest priority for execution.

7. First In First Out(FIFO) Function

7.1 Initialize current_time to 0 :- This variable,current_time,keeps track of the current time during the execution of processes. It is initially set to 0.



7.2 Calculate Remaining Time for each process :- This loop iterates through each process in the specified priority level queue.For each process,it calculates the remaining time (r_time) based on the current time of the process.After updating the remaining time for the current process, current_time is then incremented by the burst time of the current process. This ensures that the next process's start time is correctly calculated.



In summary,the Fifo function assigns the remaining time to each process based on the order in which they are stored in the queue.It does not involve any reordering or prioritization based on burst times or other criteria.It Works as traditional Queue.The processes are processed in the order they are present in the queue,following the "First In,First Out" principle.

8. The Main Function

8.1 Get the number of processes from the user :- This part asks the user to enter the number of processes he/she want to input.The entered value is stored in the variable "num_processes".

```
ssignment_8_22000862.c ×
Users > dell > Downloads > 22000862 Assignmnent 8 > C Assignment_8_22000862.c > ...

int main() {
    int num_processes;
    printf("How many Processes do you want to Enter : ");
    scanf("%d", &num_processes);
    printf("\n");
```

8.2 Initialize the front and rear of all queues to 0 :- In this part I initialized the front and rear pointers of all four priority level queues to 0.By setting both front and rear to 0, it signifies that each queue is empty at the beginning, and processes can be enqueued (added to the queue) starting from index 0.

```
ssignment_8_22000862.c ×
Users > dell > Downloads > 22000862 Assignmnent 8 > C Assignment_8_22000862.c > main()

    for (int i = 0; i < 4; i++) {
        queues[i].front = 0;
        queues[i].rear = 0;
    }
```

8.3 Getting Input details for each process :- This loop collects details for each process from the user,such as priority level and burst time.It creates a Process structure,initializes the process ID and enqueues it into the appropriate priority level queue using the enqueue function.

```
C Assignment_8_22000862.c ●   C temp (2).c ●
C: > Users > dell > Downloads > 22000862 Assignmnent 8 > C Assignment_8_22000862.c > fifo(int, int)
100  int main() {
110      //Getting details
111      printf("<<<-----------------------Enter Details----------------------->>>\n\n");
112      for (int i = 0; i < num_processes; i++) {
113          Process p;
114          printf("Enter details for process %d --->> \n\n", i + 1);
115          p.p_id = i + 1;
116          printf("+ Priority Level (0 to 3): ");
117          scanf("%d", &p.priority_lvl);
118          printf("\n");
119          printf("+ Burst Time: ");
120          scanf("%d", &p.b_time);
121          printf("\n");
122
123          p.r_time = p.b_time;
124          p.sw_time = -1;
125          enqueue(p.priority_lvl, p);
126      }
127
```

Initializing remaining time and starting waiting time

current_time:Tracks the current time during the scheduling process.Initialized to 0 and updated as processes are executed.

switch_time:Marks the time for switching between priority levels.Used to determine when to move to the next priority level in the scheduling process.

```
int current_time = 0;
int switch_time = 0;
```

8.4 Execute Scheduling Algorithms for Each Queue :-

8.4.1 Priority Level Loop :- This loop iterates through each priority level (0 to 3) to execute the corresponding scheduling algorithm for processes in that priority level.

```
119     for (int priority_lvl = 0; priority_lvl < 4; priority_lvl++) {
```

8.4.2 Calculating the Number of Processes in the Queue :-

The variable num_processes_in_queue is calculated by subtracting the front index from the rear index. It represents the number of processes in the current priority level queue.

```
120         int num_processes_in_queue = queues[priority_lvl].rear - queues[priority_lvl].front;
```

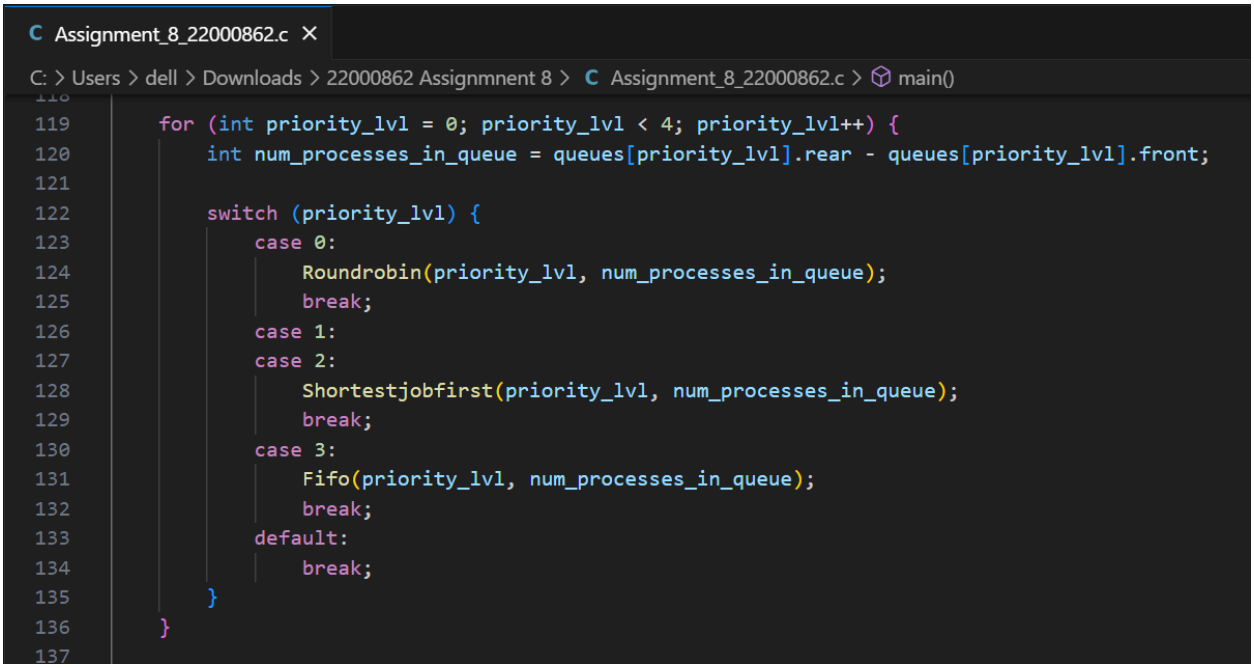
8.4.3 Switch Statement for Scheduling Algorithms :- A switch statement is used to determine the scheduling algorithm to execute based on the priority level.

Case 0 (Round Robin): If the priority level is 0, it calls the Roundrobin function with the current priority level and the number of processes in the queue.

Case 1 and Case 2 (Shortest Job First): If the priority level is 1 or 2, it calls the Shortestjobfirst(SJF) function with the current priority level and the number of processes in the queue.

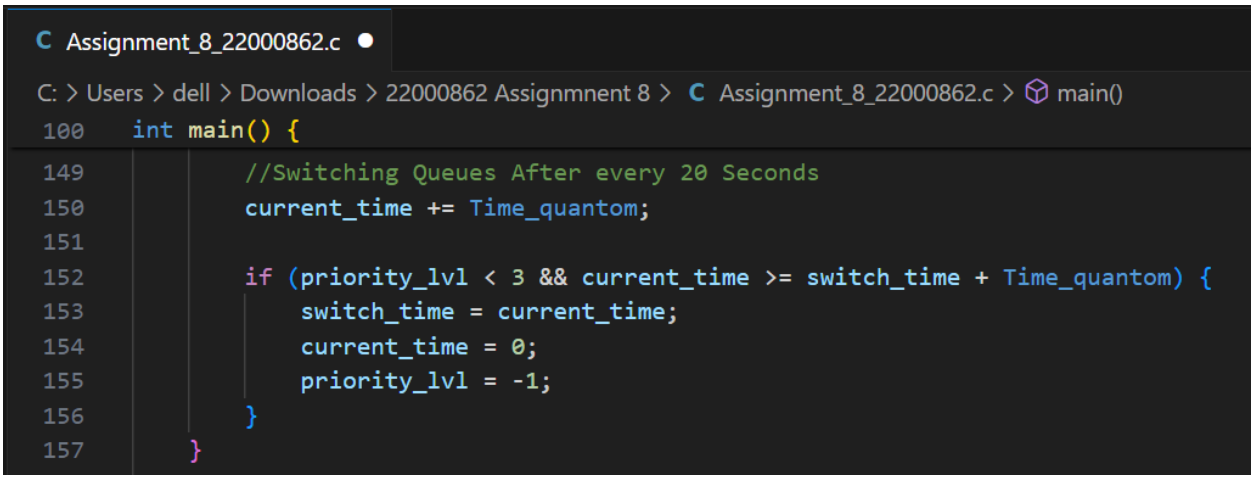
Case 3 (First In First Out): If the priority level is 3, it calls the Fifo function with the current priority level and the number of processes in the queue.

Default Case: There is a default case with a break statement.



8.4.4 Switching Queues After Every 20 Seconds :-

- Increments current_time by the time quantum (Time_quantom) after each scheduling iteration.
- Checks if it's time to switch to the next priority level based on the condition current_time >= switch_time + Time_quantom.
- If the condition is met, it updates switch_time to the current time and resets current_time to 0. Additionally, it sets priority_lvl to -1 (the loop increments it to 0, effectively making it 0 in the next iteration).



This part of the code essentially manages the scheduling process for each priority level, invoking the appropriate scheduling algorithm, and controlling the time progression and

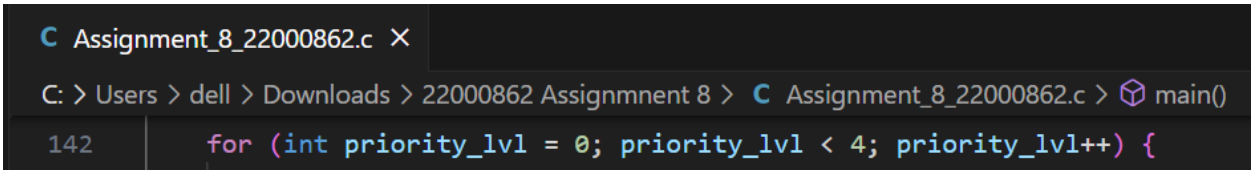
switching between priority levels. The *'switch_time*' mechanism ensures that the program switches to the next priority level after a specific time quantum.

8.5 Printing Part :-

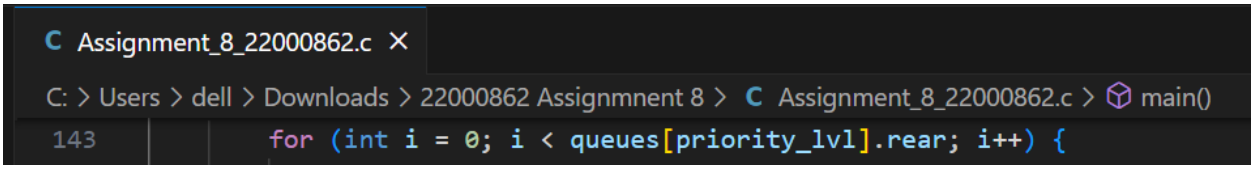
```
printf("-------------------------------------------------------------------------------------\n");
printf("\n| Process ID\t| Priority level\t| Burst Time\t| Waiting Time\t| Turnaround Time |\n");
printf("-------------------------------------------------------------------------------------\n");
```

This section prints the header for the table that displays the attributes of each process,including Process ID,Priority Level,Burst Time,Waiting Time and Turnaround Time.

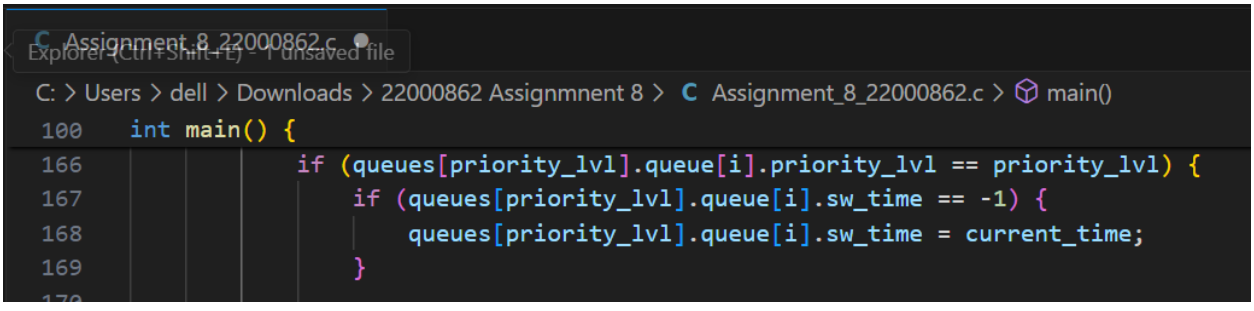
8.5.1. Outer Loop for Each Priority Level :- The outer loop iterates through each priority level (0 to 3).



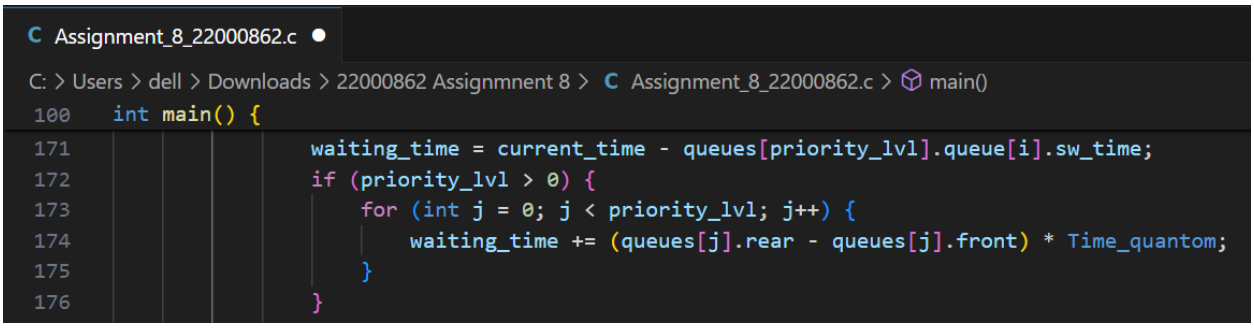
8.5.2. Inner Loop for Each Process in the Queue :- The inner loop iterates through each process in the current priority level queue.



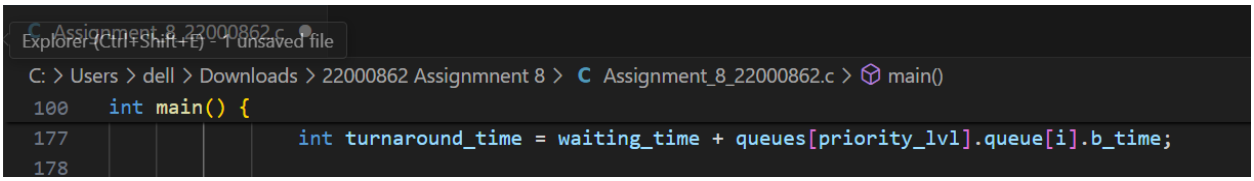
8.5.3. Checking and Updating Start Waiting Time :- Checks if the process in the queue belongs to the current priority level.If the start waiting time (sw_time) is -1,it means the process has not started waiting yet,so it sets sw_time to the current time.



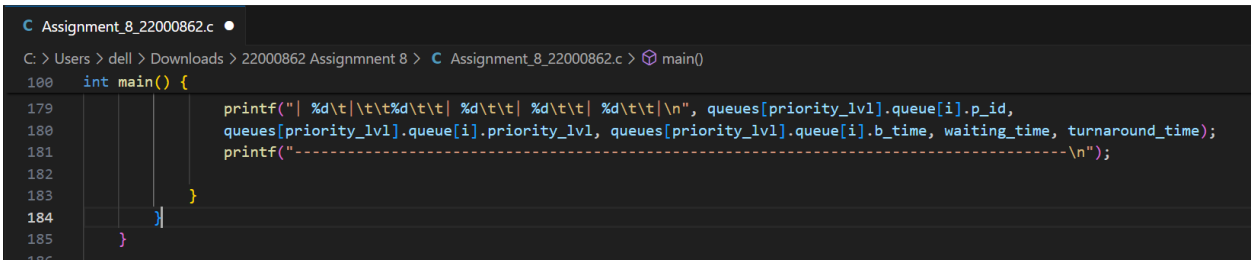
8.5.4. Calculating Waiting Time :- Calculates the waiting time for the process based on the difference between the current time and the start waiting time.If the priority level is greater than 0,it adjusts the waiting time by adding the waiting time of the processes from the lower priority levels.This adjustment is based on the assumption that lower priority processes have waited for the entire time quantum.



8.5.5. Calculating Turnaround Time :- Calculates the turnaround time by adding the waiting time to the burst time of the process.

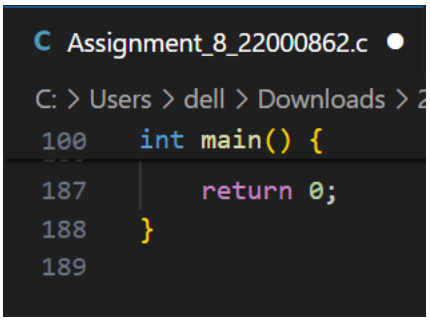


8.5.6. Printing Process :- It prints the details of each process using printf statements.The format includes the process ID,priority level,burst time,waiting time and turnaround time.This loop prints the details of each process in the form of a table,organized by priority level.



8.5 Return Part :-

The return 0; statement indicates a successful execution of the program



"In summary, this program facilitates user input for process details, organizes them into priority-level queues, and employs scheduling algorithms such as

Round Robin, Shortest Job First, and First In First Out for different priority levels. Each queue is given a time quantum of 20 seconds, and the CPU switches between queues after every 20 seconds. The scheduling results are presented in a structured table format, showcasing essential information for each process, including Process ID, Priority level, Burst Time, Waiting Time, and Turnaround Time. The program concludes with a return 0, indicating successful completion."

These Are Some Test Cases I made on my Code

Case 01 : -

```
PS C:\Users\dell> cd "c:\Users\dell\Downloads\22000862 Assignmnent 8\" ; if ($?) { gcc Assignm
_8_22000862 }
How many Processes do you want to Enter : 2

<<<-----------------------Enter Details----------------------->>>

Enter details for process 1 --->>

+ Priority Level (0 to 3): 0

+ Burst Time: 12

Enter details for process 2 --->>

+ Priority Level (0 to 3): 2

+ Burst Time: 23

-------------------------------------------------------------------------------------

| Process ID    | Priority level        | Burst Time    | Waiting Time  | Turnaround Time |
-------------------------------------------------------------------------------------
| 1     |               0               | 12            | 80            | 92            |
-------------------------------------------------------------------------------------
| 2     |               2               | 23            | 100           | 123           |
-------------------------------------------------------------------------------------
PS C:\Users\dell\Downloads\22000862 Assignmnent 8>
```

Case 02 : -

```
PS C:\Users\dell\Downloads\22000862 Assignmnent 8> cd "c:\Users\dell\Downloads\22000862 Assignmnent 8\" ; if ($?) { gcc Assignment_8_22000862.c -o Assignment_8_22
000862 } ; if ($?) { .\Assignment_8_22000862 }
How many Processes do you want to Enter : 4

<<<----------------------Enter Details---------------------->>>

Enter details for process 1 --->>

+ Priority Level (0 to 3): 0

+ Burst Time: 23

Enter details for process 2 --->>

+ Priority Level (0 to 3): 1

+ Burst Time: 23

Enter details for process 3 --->>

+ Priority Level (0 to 3): 3

+ Burst Time: 23

Enter details for process 4 --->>

+ Priority Level (0 to 3): 2

+ Burst Time: 23
```

```
-------------------------------------------------------------------------------------

| Process ID    | Priority level        | Burst Time    | Waiting Time  | Turnaround Time |
-------------------------------------------------------------------------------------
| 1     |               0               | 23            | 80            | 103           |
-------------------------------------------------------------------------------------
| 2     |               1               | 23            | 100           | 123           |
-------------------------------------------------------------------------------------
| 4     |               2               | 23            | 120           | 143           |
-------------------------------------------------------------------------------------
| 3     |               3               | 23            | 140           | 163           |
-------------------------------------------------------------------------------------
PS C:\Users\dell\Downloads\22000862 Assignmnent 8>
⚠ 0   0   Connect
```

In the provided output for the entered processes, each process has been assigned a priority level (0 to 3) and a burst time of 23. The output includes waiting time and turnaround time for each process. Let's analyze the results:

Process 1 (Priority Level 0):

- Burst Time: 23
- Waiting Time: 80
- Turnaround Time: 103

Process 2 (Priority Level 1):

- Burst Time: 23
- Waiting Time: 100
- Turnaround Time: 123

Process 4 (Priority Level 2):

- Burst Time: 23
- Waiting Time: 120
- Turnaround Time: 143

Process 3 (Priority Level 3):

- Burst Time: 23
- Waiting Time: 140
- Turnaround Time: 163

Analysis:

Waiting Time: This represents the total time each process had to wait before it could be executed. In this case, waiting time increases with priority level, as expected in a multilevel queue where higher priority processes are given preference.

Turnaround Time: This is the total time taken by the process from the submission to its completion. Again, turnaround time increases with priority level.

Comparison:

- Priority Level 0 (Process 1) has the lowest waiting and turnaround time.
- Priority Level 3 (Process 3) has the highest waiting and turnaround time.
- Priority Level 1 (Process 2) and Priority Level 2 (Process 4) fall in between.

- **Name**
  R.K.Janith Prabash Kahmira
- **Registration Number**
  2022/CS/086
- **Index Number**
  22000862