**Instruction to compile and generate executable file:**

---

1. Download project3Pawan.c.
2. In terminal, type in 'gcc ./project3Project.c' which should compile the code and automatically generate ./a.out executable file.
3. Now run ‘./a.out’ to see the output.
4. Please do not forget to put “process.txt” file that contains input information in the same directory as the c file. Alternatively, change can be made inside the c code itself. Just change the directory and name of the file you wish to use as input. Please note that the format of the text content should be in the format (tab delimited) as given below. Please omit “process  priority  burst  arrival time” or any other header part for the best result.

```
P1	8	15	0
P2	3	20	0
P3	4	20	20
P4	4	20	25
P5	5	5	45
P6	5	15	55
```

**Output:**

---

```
nawap@nawap-Opp-Satellite-E45t-A:~/Desktop/CProgram$ ./a.out

Completion of Process: P1 at 15s

Context Switch (Higher Priority) at: 20s Prempted: P2  New_Process:P3

Context Switch (Round-Robin) at: 30s Prempted: P3  New_Process:P4

Context Switch (Round-Robin) at: 40s Prempted: P4  New_Process:P3

Context Switch (Higher Priority) at: 45s Prempted: P3  New_Process:P5

Completion of Process: P5 at 50s

Context Switch (Higher Priority) at: 55s Prempted: P4  New_Process:P6

Completion of Process: P6 at 70s

Completion of Process: P3 at 75s

Completion of Process: P4 at 80s

Completion of Process: P2 at 95s

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

Process    Turn_Around_Time     Wait_time
  P1            15                0

  P5            5                 0

  P6            15                0

  P3            55                35

  P4            55                35

  P2            95                75
```

**General Discussion On How I Got my Code to work**

---

1. **Background:** Frankly speaking, this project took me a while to complete. I have been working on this project since April 23rd and just today morning (April 29th) I finally completed it. I spent at least 5-6 hours everyday and yesterday I spent my entire day working on it.

2. **Objective and Design of the Project:** This project is basically simulation of how priority based CPU scheduling works. However that's not all; it has a little tweak. When the process with same priority as running process arrives, Round-Robin has to be implemented. That means, running process has to be context switched if it has run for certain time quantum which in our case is 10 units.
The first thing I want to mention on my design is 'struct Process'. It had all the members such as id, burst_time, arrival_time and priority of a process. When the process arrives, this struct Process was generated and was passed to certain function where it is concatenated to the singly linked list which we call "ready queue". This linked list, I formulated it in the order of decreasing priority so that when we access the head, we can get hand the process with highest priority. We take the head of the queue to our run state and also check if there is any higher priority or same priority process in the queue. If there is higher priority process, it is brought in immediately and the process in run_state is preempted. Otherwise, if the priority is the same, we let the process in the run-state keep running until it runs 10 times at least. Then we prempt it to bring the process with the same priority from the queue.

**3. General Steps on how the code was formulated:**

1) Implement a loop that increments an integer time variable t by 1 from t=0 to t = 95

2) The body of your loop will check for arrival of a process at time t by reading the Process Table from a file. You read the file and check for any processes at Arrival = t.

3) Inside the loop, if a new process has arrived at time t:
   a) Insert the new process into the ready queue (implement the queue as a singly linked list).

   b) If no process is in the run state, insert the new process into the run state. Otherwise, if the new process P_n has a higher priority than the process P_r in the run state, preempt the process in the run state (place it back in the ready queue) and assign the new process to the run state. Output the following for the context switch: time t, P_n, P_r

4) Inside the loop, increment the turnaround time for each process, and, depending on the processes' state, increment the wait time counter for each process.

5) When you finally terminate the loop:
Output the total turnaround and wait times for each process P_i: P_i, turnaround time, wait time

4. **Some Challenges:**

Main challenge that I faced was in reading from the file. It could not read it once and store it somewhere. I had to read file 95 times and see if any line has the arrival time that equaled the current time. Somehow I accomplish that part. Then when I was writing code; creating the singly linked list got me confused for sometime. I had to put entire process struct in the linked list which is what I did. I did not quite get the correct answer, so I put pointer to the struct Process in the linked list. That worked pretty well. However it didn't give me entirely correct answer. I got correct answer only up to 55 units of time. Then somehow, I had two P6 processes. Basically then one of my P6 was prempting my other P6 since they had the same priority. I checked my entire implementation more than 20 times, but could not get the answer I was desperately hoping for. Doubting I was doing everything wrong, I started fresh and got wrong answers again. That lasted for about 5 days.

It was just today morning I found that my implementation has been correct all the way but I had a issue scanning from the file. For some reason, my last line was scanned twice thereby resulting in creation of two P6. I put in break statement which broke the loop as soon as it saw EOF. That solved the problem and I finally got the correct answer.