Multilevel Queue Scheduling

Purpose:

This code about multilevel queue algorithm in CPU scheduling. i use about it three scheduling policies. if sent round robin architecture, shortest job first architecture and first in first serve architecture.

Code Structure:

1.Data Structures:

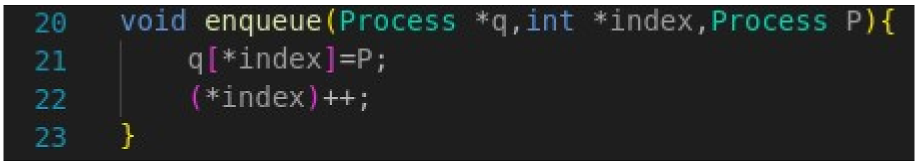
- Process structure : a process has Process ID, priority, burst time, remaining time, waiting time, turnaround time

- Processors structure : processors structure is queue data structure .It has maximum 100 size queue.

  - q0 - Round Robin (RR)
  - q1 - Shortest Job First (SJF)
  - q2 - Shortest Job First (SJF)
  - q3 - First-In-First-Out (FIFO)

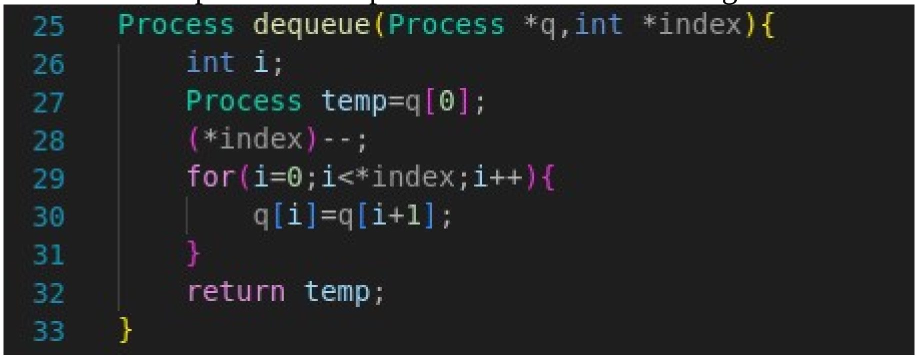
2.Function Definition:

- enqueue : add a process to the relevant processor queue



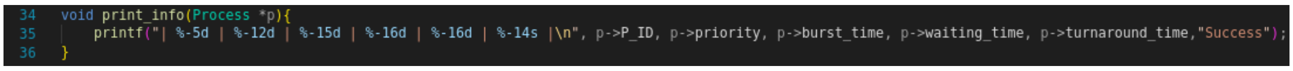
```
void enqueue(Process *q,int *index,Process P){
    q[*index]=P;
    (*index)++;
}
```

- dequeue : remove the process from processor after the executing



```
Process dequeue(Process *q,int *index){
    int i;
    Process temp=q[0];
    (*index)--;
    for(i=0;i<*index;i++){
        q[i]=q[i+1];
    }
    return temp;
}
```

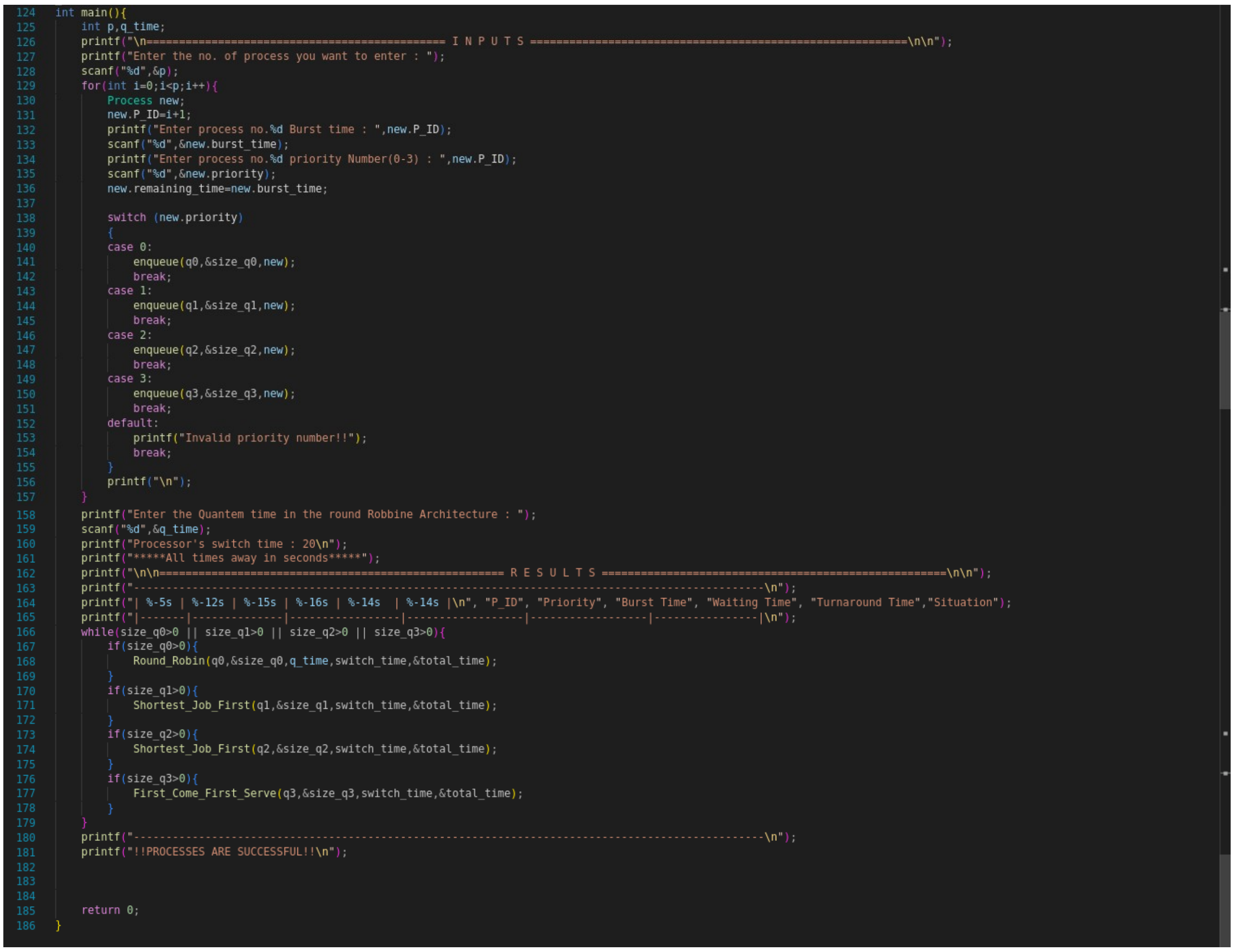
- print_info : print information each of process



```
void print_info(Process *p){
    printf("| %-5d | %-12d | %-15d | %-16d | %-16d | %-14s |\n", p->P_ID, p->priority, p->burst_time, p->waiting_time, p->turnaround_time,"Success");
}
```

- Round_Robin : Implements the Round Robin scheduling algorithm
- Shortest_Job_First : Implements the Shortest Job First scheduling algorithm.
- First_Come_First_Serve : Implements the First Come First Serve scheduling algorithm.

3.Main Function:

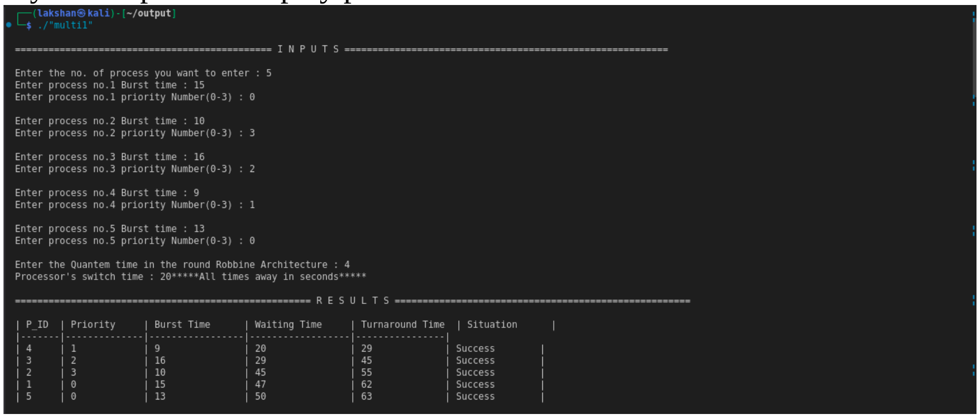


```c
int main(){
    int p,q_time;
    printf("\n============================================= I N P U T S ==========================================================\n\n");
    printf("Enter the no. of process you want to enter : ");
    scanf("%d",&p);
    for(int i=0;i<p;i++){
        Process new;
        new.P_ID=i+1;
        printf("Enter process no.%d Burst time : ",new.P_ID);
        scanf("%d",&new.burst_time);
        printf("Enter process no.%d priority Number(0-3) : ",new.P_ID);
        scanf("%d",&new.priority);
        new.remaining_time=new.burst_time;

        switch (new.priority)
        {
        case 0:
            enqueue(q0,&size_q0,new);
            break;
        case 1:
            enqueue(q1,&size_q1,new);
            break;
        case 2:
            enqueue(q2,&size_q2,new);
            break;
        case 3:
            enqueue(q3,&size_q3,new);
            break;
        default:
            printf("Invalid priority number!!");
            break;
        }
        printf("\n");
    }
    printf("Enter the Quantem time in the round Robbine Architecture : ");
    scanf("%d",&q_time);
    printf("Processor's switch time : 20\n");
    printf("*****All times away in seconds*****");
    printf("\n\n==================================================== R E S U L T S ====================================================\n\n");
    printf("--------------------------------------------------------------------------------------------------\n");
    printf("| %-5s | %-12s | %-15s | %-16s | %-14s  | %-14s |\n", "P_ID", "Priority", "Burst Time", "Waiting Time", "Turnaround Time","Situation");
    printf("|-------|-------------|----------------|-----------------|-----------------|---------------|\n");
    while(size_q0>0 || size_q1>0 || size_q2>0 || size_q3>0){
        if(size_q0>0){
            Round_Robin(q0,&size_q0,q_time,switch_time,&total_time);
        }
        if(size_q1>0){
            Shortest_Job_First(q1,&size_q1,switch_time,&total_time);
        }
        if(size_q2>0){
            Shortest_Job_First(q2,&size_q2,switch_time,&total_time);
        }
        if(size_q3>0){
            First_Come_First_Serve(q3,&size_q3,switch_time,&total_time);
        }
    }
    printf("--------------------------------------------------------------------------------------------------\n");
    printf("!!PROCESSES ARE SUCCESSFUL!!\n");



    return 0;
}
```

- Get user input for count of process, process burst time, process priority number and quantum time(for round robin architecture) .
- Enter process to respective processor using switch case and user input priority number.
- Till each of processor queue size equal 0 regenerate relevant processors.
- Lastly, print process properties as tabular format.

4.out put:

display user input and display process details after execution as tabular format.



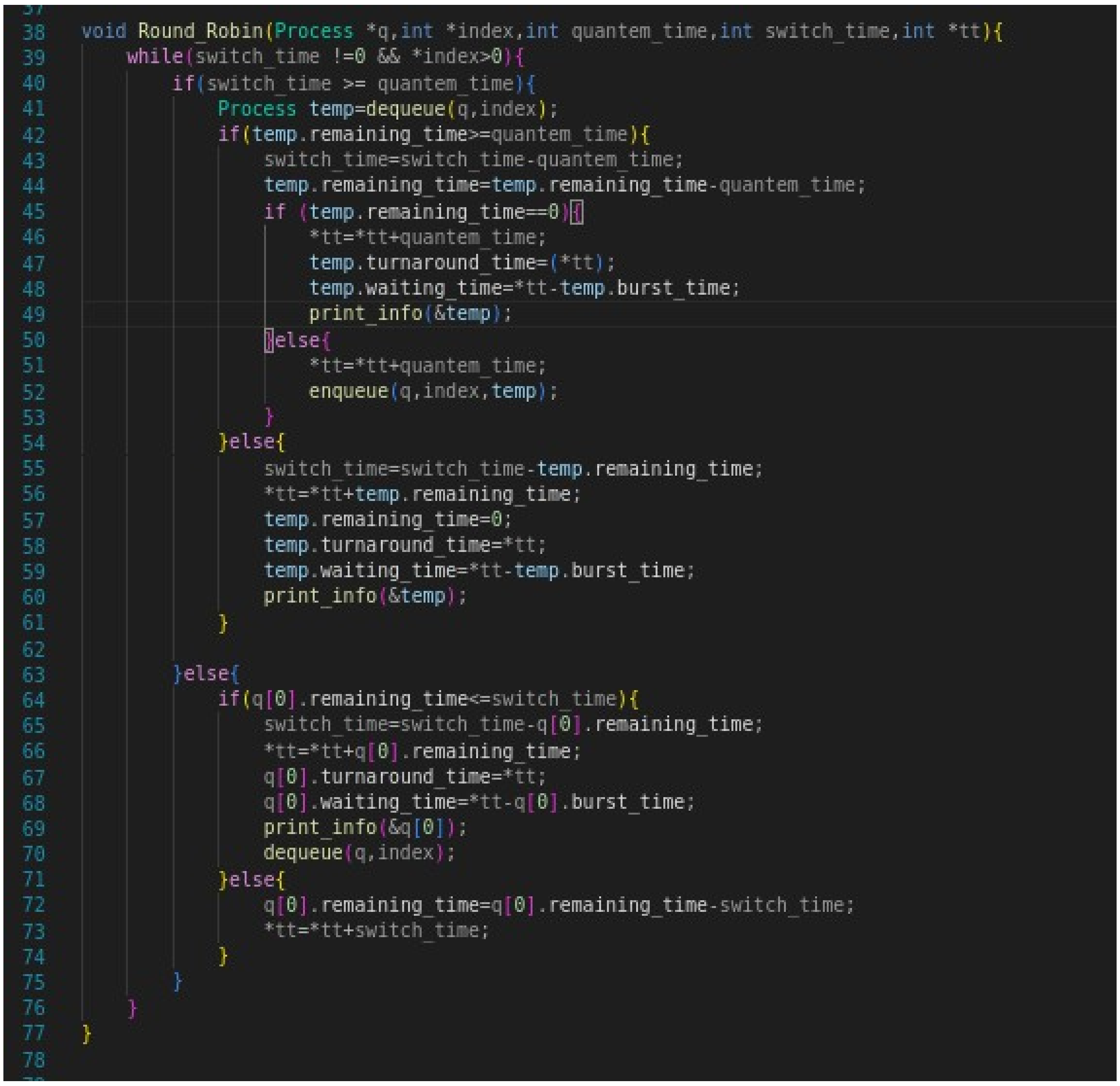
```
┌──(lakshan㉿kali)-[~/output]
└─$ ./"multi1"

============================================= I N P U T S ==========================================================

Enter the no. of process you want to enter : 5
Enter process no.1 Burst time : 15
Enter process no.1 priority Number(0-3) : 0

Enter process no.2 Burst time : 10
Enter process no.2 priority Number(0-3) : 3

Enter process no.3 Burst time : 16
Enter process no.3 priority Number(0-3) : 2

Enter process no.4 Burst time : 9
Enter process no.4 priority Number(0-3) : 1

Enter process no.5 Burst time : 13
Enter process no.5 priority Number(0-3) : 0

Enter the Quantem time in the round Robbine Architecture : 4
Processor's switch time : 20*****All times away in seconds*****

==================================================== R E S U L T S ====================================================

| P_ID  | Priority     | Burst Time      | Waiting Time     | Turnaround Time  | Situation      |
|-------|-------------|----------------|-----------------|-----------------|
| 4     | 1           | 9              | 20              | 29              | Success       |
| 3     | 2           | 16             | 29              | 45              | Success       |
| 2     | 3           | 10             | 45              | 55              | Success       |
| 1     | 0           | 15             | 47              | 62              | Success       |
| 5     | 0           | 13             | 50              | 63              | Success       |
```

Scheduling Algorithms:

- switch time-The amount of time that one process is running at a time

<u>1. Round Robin</u>

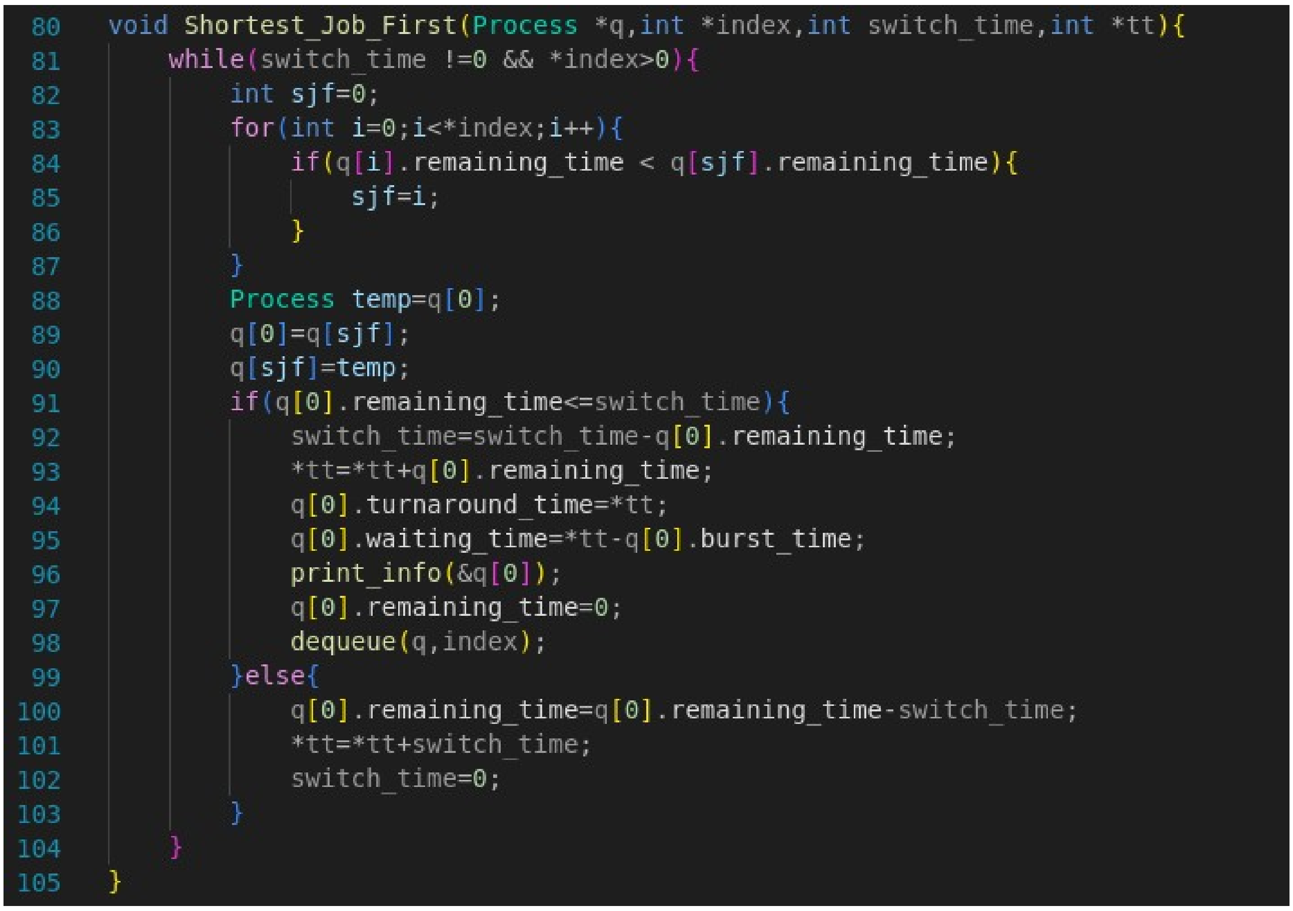
The while loop executes until the switch time equals 0 and the quantum time equals 0
If the switch time is greater than the quantum time, remove the first process from the queue, and if the remaining time is greater than the quantum time, the process will not complete, and the new process will be added to the queue, and the total time remaining will be calculated. If the time is equal to the quantum time, the process ends and the total time, turnaround time and waiting time are calculated.
If the quantum time is greater than the remaining time, the process ends before the quantum time is over and the necessary calculations are repeated.
If the quantum time is greater than the switch time, if the remaining time is less than the switch time, the process ends and the relevant calculations are performed.
If the remaining time is greater than the switch time, the process will not end and the process will be kept in the queue.



```c
void Round_Robin(Process *q,int *index,int quantem_time,int switch_time,int *tt){
    while(switch_time !=0 && *index>0){
        if(switch_time >= quantem_time){
            Process temp=dequeue(q,index);
            if(temp.remaining_time>=quantem_time){
                switch_time=switch_time-quantem_time;
                temp.remaining_time=temp.remaining_time-quantem_time;
                if (temp.remaining_time==0){
                    *tt=*tt+quantem_time;
                    temp.turnaround_time=(*tt);
                    temp.waiting_time=*tt-temp.burst_time;
                    print_info(&temp);
                }else{
                    *tt=*tt+quantem_time;
                    enqueue(q,index,temp);
                }
            }else{
                switch_time=switch_time-temp.remaining_time;
                *tt=*tt+temp.remaining_time;
                temp.remaining_time=0;
                temp.turnaround_time=*tt;
                temp.waiting_time=*tt-temp.burst_time;
                print_info(&temp);
            }

        }else{
            if(q[0].remaining_time<=switch_time){
                switch_time=switch_time-q[0].remaining_time;
                *tt=*tt+q[0].remaining_time;
                q[0].turnaround_time=*tt;
                q[0].waiting_time=*tt-q[0].burst_time;
                print_info(&q[0]);
                dequeue(q,index);
            }else{
                q[0].remaining_time=q[0].remaining_time-switch_time;
                *tt=*tt+switch_time;
            }
        }
    }
}

```

2.Shortest Job First

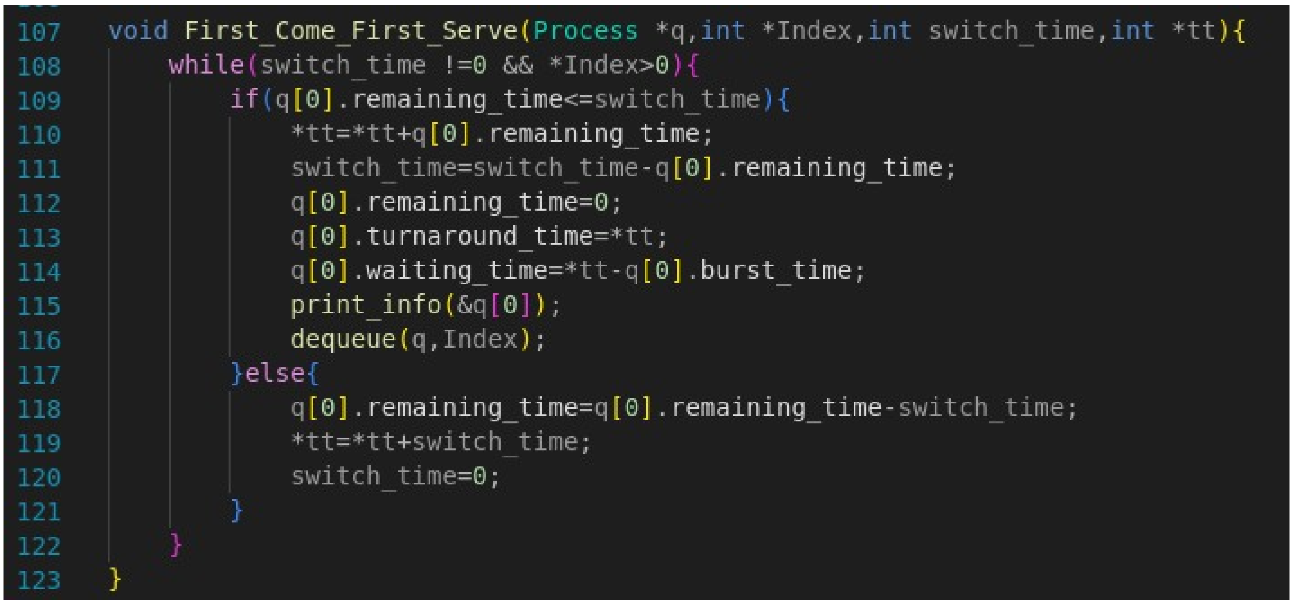
If the switch time is not equal to 0 and the index of the queue is greater than 0, the while loop will work. the process with the smallest remaining time in the queue is selected and the queue is started.
If the remaining time is less than the switch time, the process ends and the total time, turnaround time, and waiting time are calculated.
If the remaining time is greater than the switch time, the process will not finish and it will remain in the queue



```c
void Shortest_Job_First(Process *q,int *index,int switch_time,int *tt){
    while(switch_time !=0 && *index>0){
        int sjf=0;
        for(int i=0;i<*index;i++){
            if(q[i].remaining_time < q[sjf].remaining_time){
                sjf=i;
            }
        }
        Process temp=q[0];
        q[0]=q[sjf];
        q[sjf]=temp;
        if(q[0].remaining_time<=switch_time){
            switch_time=switch_time-q[0].remaining_time;
            *tt=*tt+q[0].remaining_time;
            q[0].turnaround_time=*tt;
            q[0].waiting_time=*tt-q[0].burst_time;
            print_info(&q[0]);
            q[0].remaining_time=0;
            dequeue(q,index);
        }else{
            q[0].remaining_time=q[0].remaining_time-switch_time;
            *tt=*tt+switch_time;
            switch_time=0;
        }
    }
}
```

3.First in First Serve

If the switch time is not equal to 0 and the index of the queue is greater than 0, the while loop will work.
If the remaining time of the first process in the queue is less than the switch time, the process ends and it is removed from the queue and the relevant times are calculated.
If the remaining time of the first process is less than the switch time, the process will not end and it will not be removed from the queue



```c
void First_Come_First_Serve(Process *q,int *Index,int switch_time,int *tt){
    while(switch_time !=0 && *Index>0){
        if(q[0].remaining_time<=switch_time){
            *tt=*tt+q[0].remaining_time;
            switch_time=switch_time-q[0].remaining_time;
            q[0].remaining_time=0;
            q[0].turnaround_time=*tt;
            q[0].waiting_time=*tt-q[0].burst_time;
            print_info(&q[0]);
            dequeue(q,Index);
        }else{
            q[0].remaining_time=q[0].remaining_time-switch_time;
            *tt=*tt+switch_time;
            switch_time=0;
        }
    }
}
```

Round-robin scheduling

Advantages:

1.Fairness: Round-robin is a fair scheduling algorithm as it allocates an equal amount of time to each process, regardless of its priority or size.
2.Responsiveness: Round-robin scheduling ensures that every process gets a chance to run regularly, which increases the responsiveness of the system.
3.Low latency: Round-robin scheduling has a low latency as processes get scheduled quickly, reducing the waiting time.
4.Preemptive: Round-robin is a preemptive scheduling algorithm. It means that a process can be interrupted by another process that has a higher priority, which ensures that no process monopolizes the CPU.

Disadvantages:

1.Overhead: The overhead of context switching is high in round-robin scheduling because the CPU has to switch between processes frequently, which can lead to performance degradation.
2.Long waiting times: If the time quantum is too long, the waiting time for a process can be very long, which can impact the overall performance of the system.
3.Inefficient use of resources: Round-robin scheduling can lead to inefficient use of resources as some processes may not need the entire time quantum allocated to them.
4.Starvation: Round-robin scheduling can also suffer from starvation, where a low-priority process may not get a chance to run if there are high-priority processes in the queue.

Shortest Job First (SJF)

Advantages:

1.It provides optimal average waiting time for all processes, leading to improved performance.
2.It minimizes average turnaround time, as shorter jobs are completed faster, which allows more processes to be executed.
3.It ensures fairness by giving priority to processes with shorter burst times, which can prevent longer processes from monopolizing resources.
4.It can lead to higher throughput as smaller processes are executed faster, freeing up resources for other processes.

Disadvantages:

1.It requires knowledge of the burst time of each process beforehand, which can be difficult to estimate accurately.
2.It can lead to starvation for longer processes, as they may have to wait a long time to be executed if many short processes are constantly arriving.
3.It may be difficult to implement in practice, as per-emptive algorithms may be more effective at handling real world scenarios.
4.It may lead to lower CPU utilization, as shorter processes may be executed faster, resulting in idle time for the CPU

First Come First Out (FCFS)

Advantages:

1.Simplicity: FCFS is easy to implement and understand. It does not require any complex data structures or algorithms.
2.Fairness: FCFS is fair to all processes because the process that arrives first gets executed first. There is no priority given to any process.
3.No starvation: Since FCFS follows a queue-based approach, there is no possibility of any process getting starved or left behind.

Disadvantages:

1.Convoy effect: FCFS scheduling suffers from the convoy effect, where a long process blocks the CPU, and all other small processes have to wait for their turn. This leads to poor performance and longer waiting times for smaller processes.
2.Poor utilization of resources: FCFS may not make efficient use of system resources. For instance, if a short process arrives after a long process, the short process has to wait for the long process to complete, even if there are free resources available.
3.No priority: FCFS does not take into account the priority of the processes. All processes are treated equally, and there is no way to give preference to important processes.