

Q) C code for implementing multi level queue process scheduling

```
#include <stdio.h>
#include <stdlib.h>
#define NUM_QUEUES 3
#define QUANTUM 3
```

```
typedef struct {
    int id;
    int priority;
    int burst-time;
    int remaining-time;
} process;
```

```
typedef struct {
    process *queue[100];
    int front;
    int rear;
} Queue;
Queue *queues[NUM_QUEUES];
```

```
void init_queues() {
    for (int i = 0; i < NUM_QUEUES; i++) {
        queues[i] = (Queue*) malloc(sizeof(Queue));
        queues[i] -> front = -1;
        queues[i] -> rear = -1;
    }
}
```

```
void enqueue(process *p) {
    int priority = p -> priority;
    queues[priority] -> rear++;
    queues[priority] -> queue[queues[priority] -> rear] = p;
}
```



```
#include <stdio.h>
```

```
#include <limits.h>
```

```
struct proc
```

```
int id;
```

```
int bt;
```

```
int at;
```

```
int qt;
```

```
int wt;
```

```
int tat;
```

```
int rt;
```

```
int st;
```

```
int ed;
```

```
int vis;
```

```
};
```

```
void main() {
```

```
int n, CurrTime=0;
```

```
float avgWT=0, avgTAT=0, avgRT=0, tP;
```

```
printf("Queue 1 is system process\n Queue  
2 is user process\n");
```

```
printf("Enter number of processes:");
```

```
scanf("%d", &n);
```

```
struct proc proc[n], tmp;
```

```
for (int i=0; i<n; i++) {
```

```
proc[i].id = i+1;
```

```
proc[i].vis = 0;
```

```
printf("Enter Burst Time, Arrival Time and
```

```
Queue of P%d: ", i+1);
```

```
scanf("%d %d %d", &proc[i].bt, &proc[i].at,
```

```
&proc[i].qt);
```

```
}
```



```

for (int i=0; i<n; i++) {
    for (int j=i+1; j<n; j++) {
        if (procs[i].wt > procs[j].wt ||
            (procs[i].wt == procs[j].wt &&
             procs[i].qt > procs[j].qt)) {
            tmp = procs[i];
            procs[i] = procs[j];
            procs[j] = tmp;
        }
    }
}

```

```

int compTime = procs[0].bt, minQT = INT_MAX;
minQT, procCount = 0;
procs[0].vis = 1;
while (procCount < n-1) {
    for (int i=0; i<n; i++) {
        if (procs[i].wt <= compTime && procs[i].vis == 0) {
            if (procs[i].qt < minQT) {
                minQT = procs[i].qt;
                minQT = i;
            }
        }
    }
}

```

```

procs[minQT].vis = 1;
compTime += procs[minQT].bt;
minQT = INT_MAX;
tmp = procs[procCount+1];
procs[procCount+1] = procs[minQT];
procs[minQT] = tmp;
procCount++;
}

```



```

printf("\n process \t wt \t TAT \t RT \n");
for(int i=0; i<n; i++) {
    if (procs[i].at < currTime) {
        procs[i].st = currTime;
    } else {
        procs[i].st = procs[i].at;
    }
    procs[i].et = procs[i].st + procs[i].bt;
    currTime = procs[i].et;
    procs[i].tat = procs[i].et - procs[i].at;
    procs[i].wt = procs[i].tat - procs[i].bt;
    procs[i].rt = procs[i].st - procs[i].at;
    printf("%d \t %d \t %d \t ", procs[i].id,
        procs[i].wt, procs[i].tat, procs[i].rt);
    avgWT += procs[i].wt;
    avgTAT += procs[i].tat;
    avgRT += procs[i].rt;
}

```

```

tp = (float) procs[n-1].et / n;
printf("Avg WT: %.2f \n", avgWT / n);
printf("Avg TAT: %.2f \n", avgTAT / n);
printf("Avg RT: %.2f \n", avgRT / n);
printf("Throughput: %.2f", tp);
}

```

output

Enter no. of process : 4

Enter BURST Time, ArrivalTime @ Queue :

4 0 1

3 0 1

5 0 2

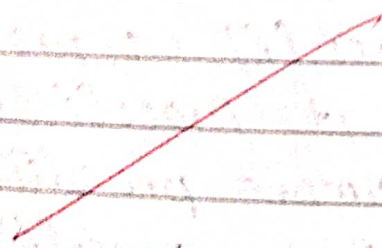
5 10 1

Avg WTI: 3.25

Avg TAT: 7.5

Avg RT: 3.25

Throughput: 7.25



[Signature]
5/6/25