8) 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 {
    proces *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(proces *p) {
    int priority = p->priority;
    queues[priority]->rear++;
    queues[priority]->queue[queues[priority]
```

PAGE NO:
DATE:

```
#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 et;
    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 procs[n], tmp;
    for(int i=0; i<n; i++){
        procs[i].id=i+1;
        procs[i].vis=0;
        printf("Enter Burst time, Arrival Time and
            Queue of P%d: ", i+1);
        scanf("%d %d %d", &procs[i].bt, &procs[i].at,
            &procs[i].qt);
    }
```

```
for (int i=0; i<n; i++){
  for(int j= i+1; j<n; j++){
    if (procs[i].at > procs[j].at ||
      (procs[i].at == procs[j].at &&
        procs[i].pt)){
      temp = procs[i];
      procs[i]= procs[j];
      procs[j]= temp;
    }
  }
}

int compTime= procs[0].bt, minBT= INT_MAX,
  minBI, procCount= 0;
procs[0].vis= 1;
while (procCount<n-1){
  for(int j=0; i<n; i++){
    if(procs[i].at<= compTime && procs[i].
      vis==0){
      if(procs[i].pt < minBT){
        minBT = procs[i].pt;
        minBT= i;
      }
    }
  }

procs[minBT].vis= 1;
compTime += proc[minBT].bt;
minBT = INT_MAX;
temp= Procs[ProcCount+1];
Procs[ProcCount+1]=Procs[minBI];
Procs[minBI]= temp;
Procs(count++;
}
```

```
printf("\n process \t WT \t TAT \t RT \n");
for(int i=0; i<n; i++) {
    if (procs[i].at < curTime) {
        procs[i].st = curTime;
    } else {
        procs[i].st = procs[i].at;
    }
    procs[i].et = procs[i].st + procs[i].bt;
    curTime += procs[i].bt;
    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 for Queue :
4   0   1
3   0   1
5   0   2
5   10  1
```

PAGE NO.
DATE

Avg WT: 3.25
Avg TAT: 7.5
Avg RT: 3.25
Throughput: 1.25

2)

5/6/24