Rate Monotonic Scheduling

```
#include <stdio.h>
#include <math.h>
void main(){
  int n;
  float e[20], p[20];
  int i;
  float ut, u, x, y;
  printf("\n Enter no. of processes:");
  scanf("%d", &n);
  for (i=0; i<n; i++) {
    printf("\n Enter execution time for P%d:",
                                    (i+1));
    scanf("%f", &e[i]);
    printf("\n Enter period for P%d:", (i+1));
    scanf("%f", &p[i]);
  }
  for (i=0; i<n; i++) {
      x = e[i]/p[i];
      ut += x; }
    y = (float)n;
    y = y* (pow(2.0, 1/y)-1);
      u = y;
    if (ut<u) {
    printf("\n As %f <%f", ut, u);
    printf("\n System is schedulable:");
  }
  else
    printf("\n Not sure...");
  getch();
}
```

OUTPUT -

Enter no. of processes : 3
Enter execution time for P1 : 1
Enter period for P1 : 20
Enter execution time for P2 : 2
Enter period for P2 : 5
Enter execution time for P3 : 2
Enter period for P3 : 10

As 0.650000 < 0.779763,
System is schedulable.

Earliest Deadline First Scheduling

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
    int deadline;
    int execution;
    int execution_copy;
} Task;
int min (Task *tasks, int n);
void update_execution_copy (Task *tasks, int n);
void execute_task ( Task *tasks, int task_id,
                    int timer);
int main() {
    int n, timer = 0;
    float cpu_utilization;
    printf ("enter no. of tasks: \n");
    scanf ("%d", &n);
    Task *tasks = malloc (n * sizeof (Task));
    for (int i=0; i<n; i++) {
        printf ("enter task %d parameters: \n", i+1);
        printf ("execution time: ");
        scanf ("%d", &tasks[i].execution);
        printf ("deadline time: ");
        scanf ("%d", &tasks[i].deadline);
        tasks[i].execution_copy = 0;
    }
cpu_utilization = 0;
    for (int i=0; i<n; i++) {
        cpu_utilization += (float) tasks[i].
    execution / (float) tasks[i].deadline; }
    printf ("CPU utilization = %f \n", cpu_utilization);
```

```
if (cpu_utilization < 1)
    printf ("tasks can be scheduled: \n");
else
    printf ("schedule is not feasible: \n");
while (1) {
    int active_task_id = min(tasks, n);
    if (active_task_id == -1) {
        printf ("%.d Idle \n", timer);
    } else {
    execute_task (tasks, active_task_id,
    timer);
    if (tasks [active_task_id].executionCopy
                                    == 0) {
    update_execution_copy (tasks, active_task_
                                    id);
    }}
    timer ++;
int all_completed = 1;
    for (int i=0; i<n; i++) {
        if (tasks [i].execution_copy > 0) {
            all_completed = 0;
            break;
        }}
    free (tasks);
    return 0;
}
int min (Task *tasks, int n) {
    int min_deadline = - INT_MAX_;
    int task_id = -1;
    for (int i=0; i<n; i++) {
    if (tasks [i].execution_copy > 0 &&
        tasks [i].deadline < min_deadline) {
```

```
        min_deadline = tasks[i].deadline;
        task_id = i;
    }
}
return task_id;
}
void update_execution_copy(Task *tasks, int n){
    tasks[n].execution_copy = tasks[n].
                                execution; }
void execute_task(Task *tasks, int task_id,
                  int timer){
    tasks[tasks_id].execution_copy--;
    printf("%d %d Task %d\n", timer,
                  task_id+1); }
```

OUTPUT-

```
Enter no. of tasks: 3
Enter task 1 parameters:
execution time: 1
deadline time: 3
Enter task 2 parameters: 4
execution time: 1
deadline time: 4
Enter task 3 parameters:
execution time: 2
deadline time: 8

CPU utilization: 0.833333
Tasks can be scheduled.
```

Proportional Scheduling

```
#include <stdio.h>
int main() {
    int i, n, T;
    printf("\n Enter total val of share T:");
    scanf("%d", &T);
    printf("\n Enter no. of processes: ");
    scanf("%d", &n);
    int N[n];
    double NT = 0;
    for (i=0; i<n; i++) {
        printf("enter share of P%d:", i+1);
        scanf("%d", &N[i]);
        NT += (double) N[i] / T;
    } if (NT <= 1)
        printf("\n Proportional share schedule is
                possible.\n");
        else
        printf("\n Proportional share schedule is
                not possible\n");
    }
```

OUTPUT-

```
Enter total val of share T: 5
Enter no. of processes: 3
Enter share of P1: 2
Enter share of P2: 1
Enter share of P3: 0
```

Proportional share schedule is possible.

Multilevel Queue Scheduling

```
#include <stdio.h>

void swap (int *a, int *b) {
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

int main () {
    int n, p[n], art[n], burt[n], type[n];
    printf ("enter number of processors:");
    scanf ("%d", &n);
    for (int i=0; i<n; i++) {
        p[i] = i+1;
        printf ("enter burst time of processor %d", i+1);
        scanf ("%d", &burt[i]);
        printf ("enter arrival time of processor %d", p[i]);
        printf ("enter type %d", p[i]);
    }
    for (int i=0; i<n; i++) {
        for (int j=i+1; j<n; j++) {
            swap(&art[j], &burt[j+1]);
            if (art[j] > art[j+1]) {
                swap(&art[j], &art[j+1]);
                swap(&p[j], &p[j+1]);
                swap(&burt[j], &burt[j+1]);
                swap(&type[j], type[j+1]);
            }
        }
    }
}
```

```
for (int i=0; i<n; i++){
    for (int j=i+1; j<n; j++){
        if (art[j]==art[j+1] && type[j]<
                                type[j+1]){
            swap(&art[j], &art[j+1]);
            swap(&p[j], &p[j+1]);
            swap(&burt[j], &burt[j+1]);
            swap(&type[j], &type[j+1]);
        }
    }
}

int wt[n], tat[n], totwt, tottat;
    wt[0] = 0;
    tottat = 0;
    totwt = 0;
    for (int i=1; i<n; i++){
        wt[i] = wt[i-1] + burt[i-1] - art[i];
        totwt += wt[i];
    }
    for (int i=0; i<n; i++){
        tat[i] = wt[i] + burt[i];
        tottat += tat[i];
    }
    for (int i=0; i<n; i++){
    printf("processor %d, type %d, arrival time %d,
        burst time %d, waiting time %d, tat %d",
            p[i], type[i], art[i], burt[i], wt[i], tat[i]);
    }
    printf("average wt = %f, average tat = %f",
        (float)(totwt)/n, (float)(tottat)/n);
}
```

SPARK
Date Page

OUTPUT-

for p(2) Enter type (user 0, os 1), arrival time, burst time.

1
0
7

for p(3) Enter type (user 0, os 1) arrival time, burst time.

1
4
6

processor	type	arrival time	bt	wt	tat
2	1	0	7	0	7
1	0	0	5	7	12
3	1	4	6	8	14

avg. waiting time = 5
avg. turn around time = 11.