22/6/23

Round Robin

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

struct process {
    int burst_time;
    int remaining_time;
    int process_ID;
};
void roundRobinScheduling(struct Process *processes, ... quantum) { int i;
    int *waitingtime = (int *) malloc(
        numProcesses * sizeof(int));
    int *turnaroundtime = (int *) malloc(
        numProcesses * sizeof(int));
    int *burstTime_copy = (int*) malloc(numProcesses *
        size of (int));
    for (i=0; i<numProcesses; i++) {
        burstTime copy [i] = processes [i].burstTime;
        waiting Time [i]=0;
        turnaroundTime [i]=0;
    }
    int currentTime=0;
    int remainingProcesses = numProcesses;
while (remainingprocesses >0) {
    for (i=0; i<numProcesses; i++) {
        if (burstTime copy [i] >0) {
        if (burstTime copy [i] <= Timequantum) {
        currentTime += burstTime copy [i];
        burstTime copy [i]=0;
```

```
            waitingTime[i] = currentTime - processes[i].burstTime;
            turnaroundtime[i] = currenttime;
            remainingProcesses--;
        } else {
            currentTime += timeQuantum;
            burstTimeCopy[i] -= timeQuantum;
        }
    }
}

printf("process \t burst time \t waiting time \t
        turnaroundtime \n");
    for (i=0; i<numProcesses; i++) {
printf("%d \t\t %d \t\t %d \t\t %d \n",
    processes[i].processID, processes[i].burstTime,
        waitingTime[i], turnaroundTime[i]);
    }
free(waitingTime);
free(turnaroundTime);
free(burstTimecopy);
}

int main() { int i;
        int numProcesses, timeQuantum;
printf("enter no. of processes\n");
scanf("%d", &numProcesses);
struct Process *processes = (struct process*) malloc
        (numProcesses * sizeof(struct Process));
printf("enter burst time for each process:\n");
    for (i=0; i<numprocesses; i++) {
        printf("process %d", i+1);
        scanf("%d", &processes[i].burstTime);
        processes[i].processID = i+1;
    processes[i].remainingTime = processes[i].
                burstTime; }
```

```
    printf ("enter time quantum");
    scanf ("%d", &timeQuantum);
roundRobinScheduling (processes, numProcesses,
                      timeQuantum);

    free (processes);
    return 0;
}
```

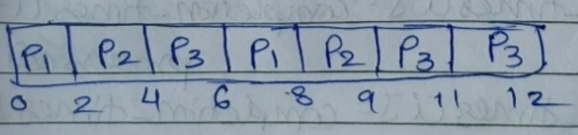
OUTPUT-

```
enter no. of processes: 4
enter burst time for each process:
process 1: 3
process 2: 4
process 3: 5
process 4: 6
enter time quantum: 2
```

process	burst time	waiting time	turnaround time
1	3	6	9
2	4	7	11
3	5	11	16
4	6	12	18

execution time: 10.245s

graph ⟹



(Non-Preemptive) Priority

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

struct Process {
    int burst_time;
    int priority;
    int waiting_time;
};

void priorityScheduling(struct Process* processes,
                        int n) { int i,j;
    for (i=0; i<n-1; i++) {
        for (j=0; j<n-i-1; j++) {
            if (processes[j].priority > processes[j+1].
                                        priority) {
                struct process temp = processes[j];
                processes[j] = processes[j+1];
                processes[j+1] = temp;
            }}}
int completion_time = (int) malloc(n* sizeof(int));
int turnaround_time = (int) malloc(n* sizeof(int));
    completion_times[0] = processes[0].burst_time;
    turnaround_times[0] = completion_times[0];
    for (i=1; i<n; i++) {
        completion_times[i] = completion_times[i-1] +
                              processes[i].burst_time;
        turnaround_times[i] = completion_times[i] -
                              processes[i].burst_time;
        processes[i].waiting_time = turnaround_times[i];
}
```

```
printf("\n process \t burst time \t priority \t waiting
                                        time\n");
    for (i=0; i<n; i++) {
printf("%d \t %d \t \t %d \t \t %d\n", i+1,
    processes[i].burst_time, processes[i].priority,
        processes[i].waiting_time); }
    free (completion_times);
    free (turnaround_times);
}

int main() { int n, i;
    printf("enter no. of processes:");
    scanf("%d", &n);
struct Process processes = (struct Process) malloc
                        (n * sizeof (struct Process));
printf("enter burst time & priority for each
                            process:\n");
    for (i=0; i<n; i++) {
        printf("process %d:\n", i+1);
        printf("burst-time:");
        scanf("%d", &processes[i].burst_time);
        printf("priority:");
        scanf("%d", &processes[i].priority);
        processes[i].waiting_time = 0;
    }
priority scheduling (processes, n);
free (processes);
return 0;
}
```

OUTPUT-

enter no. of processes : 3
enter burst time & priority for each process :
process 1:
burst time: 12
priority: 1
process 2:
burst time: 24
priority: 3
process 3:
burst time: 48
priority: 4

process	burst time	priority	waiting time
1	12	1	0
2	24	3	12
3	48	4	36

execution time: 28.9948

graph →