VIT

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Name :- Akshat Agarwal

Registration No. :- 23BKT0003

Faculty :- Naveen Kumar J

Course Name :- Operating System Theory

Course Code :- BCSE303L

Digital Assignment

(Topic : CPU Scheduling Algorithms : Round Robin Scheduling Algorithm)

Question Statement :

Digital Assessment - 1

- ❑ To design and develop an interactive animation tool that visually demonstrates the working of a specific Operating System (OS) algorithm. This tool will serve as a teaching aid to help students and educators understand how the algorithm works step by step. Choose any one algorithms from the given list:
  - ❑ **CPU Scheduling Algorithms**: FCFS, SJF, Priority Scheduling, Round Robin, hybrid.
  - ❑ **Deadlock Algorithms**: Banker's Algorithm, Deadlock Detection.
  - ❑ **Page Replacement Algorithms**: FIFO, LRU, Optimal Page Replacement.
  - ❑ **Disk Scheduling Algorithms**: FCFS, SSTF, SCAN, C-SCAN.
  - ❑ **Memory Allocation**: First Fit, Best Fit, Worst Fit.

Digital Assessment - 1

Functionality (10)	❑ The tool must accept user input for required parameters (e.g., process burst time, arrival time, priority, page requests, etc.). ❑ It should animate the working of the algorithm step by step, showing states like queues, memory frames, disk head movement, allocation tables, etc. respective to the algorithms ❑ The animation must clearly show transitions and results (e.g., waiting time, turnaround time, page faults, etc.)
User Interface (10)	❑ The tool must have a simple, intuitive interface. ❑ Clear labels and instructions for input and output. ❑ Visuals must be clear and easy to understand.
Document and Demonstration (10)	❑ A short report (2-4 pages) describing: ❑ The algorithm with an explanation of its working. ❑ How the tool is designed and how to use it. ❑ Screenshots of the tool in action. ❑ Challenges faced and how you solved them. ❑ Source Code Complete

Round Robin CPU Scheduling Algorithm

1. Explanation of the Algorithm

Definition:

Round Robin (RR) is a preemptive CPU scheduling algorithm used by operating systems to manage multiple processes.

Each process is assigned a fixed time slot called a time quantum.

Processes are scheduled in a circular queue (FIFO order). If a process doesn’t finish within its time quantum, it is preempted and sent to the end of the queue, and the next process is scheduled.

This repeats until all processes are finished.

Key Properties:

- Equal CPU allocation (fairness)
- Starvation-free
- Simple to implement
- Highly dependent on the chosen time quantum value
- Context switches occur at each quantum expiration

Working Principle:

New processes join the end of the ready queue.

Each process executes for one quantum.

If the burst time is less than or equal to quantum, it completes.

If burst time exceeds quantum, it’s preempted and re-queued.

Repeat until the queue is empty.

Example Workflow:

Suppose three processes with burst times:

P1: 6

P2: 3

P3: 7

Time quantum = 2

Time	Process Executed	State
0–2	P1	P2,P3 wait
2–4	P2	P3,P1 requeued
4–6	P3	P1,P2 wait
...	...	(continue until done)

After each quantum, update the remaining time and queue order. After this We Calculate turnaround/waiting times.

2. Tool Design and Usage

User Inputs:

- Number of processes
- Burst times for each process
- Arrival times (for advanced simulation)
- Time quantum

Functionality:

- Add processes and adjust burst/arrival times
- Show processes in ready queue
- Step-by-step animation of CPU execution:
- Highlight active process
- Update/remain times and queues after each quantum
- Represent context switches

Display results:

- Waiting time,
- Turnaround time
- Order of execution (Gantt chart)

Sample Interface Features:

- Simple input form for process data
- Visual queue to show process order
- Animated Gantt chart of execution timeline
- Labels for all transitions and results

3. Screenshots (conceptual, as per tool in development)

Example Screenshot 1:

Inputs form: Enter burst time, arrival time, quantum time then "Add Process"

Sample Input :

```
PS C:\Users\Lenovo\OneDrive\Desktop\Vit 5th Sem\Vit Lab\os_cmd\Digital Assignment> ./code1.exe

====== Round Robin Scheduling ======
1. Input Processes
2. Run Round Robin Scheduling
3. Display Final Results
4. Exit
Enter choice: 1

Enter number of processes: 3
Enter Time Quantum: 2
Enter burst time of Process P1: 5
Enter burst time of Process P2: 4
Enter burst time of Process P3: 2
```

Example Screenshot 2:

Ready queue visual: Shows current order and remaining times

```
====== Round Robin Scheduling ======
1. Input Processes
2. Run Round Robin Scheduling
3. Display Final Results
4. Exit
Enter choice: 2

--- Step by Step Gantt Chart ---
Interval        | Executed Process | Ready Queue (during execution)
-------------------------------------------------------------
  0 -   2       |   P1             | P2 P3
  2 -   4       |   P2             | P3 P1
  4 -   6       |   P3             | P1 P2
  6 -   8       |   P1             | P2
  8 -  10       |   P2             | P1
 10 -  11       |   P1             | Empty
```

Example Screenshot 3:

Animated step: CPU executing a process, requeueing after quantum time

```
====== Round Robin Scheduling ======
1. Input Processes
2. Run Round Robin Scheduling
3. Display Final Results
4. Exit
Enter choice: 3

--- Final Results ---
PID     Burst   Waiting Turnaround
P1          5         6         11
P2          4         6         10
P3          2         4          6

Average Waiting Time    = 5.33
Average Turnaround Time= 9.00
```

```
====== Round Robin Scheduling ======
1. Input Processes
2. Run Round Robin Scheduling
3. Display Final Results
4. Exit
Enter choice: 4
```

4. Challenges Faced

- Designing a clear and interactive animation that accurately visualizes all queue transitions and context switches.
- Achieving a simple, intuitive user interface with easy-to-follow stepwise execution.
- Correctly computing waiting and turnaround times, especially with different arrival times.
- Communicating results in a visually clear manner (graphical and numerical).
- Ensuring code modularity for easy updates and debugging.

5. Source Code (Sample – Python-like pseudocode for educational illustration)

Code :

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

#define MAX 100

typedef struct {
    int pid;
    int burst;
    int remaining;
    int waiting;
    int turnaround;
} Process;

Process proc[MAX];
int n, tq;

/* Circular queue implementation */
int q[MAX];
int qfront = 0, qrear = 0;

void enqueue(int x) {
    int next = (qrear + 1) % MAX;
    if (next == qfront) {
        printf("Queue overflow!\n");
        exit(1);
    }
    q[qrear] = x;
    qrear = next;
}

int dequeue() {
    if (qfront == qrear) return -1; // empty
    int val = q[qfront];
    qfront = (qfront + 1) % MAX;
    return val;
}

bool qempty() {
    return qfront == qrear;
}

void printReadyQueue() {
    if (qempty()) {
        printf("Empty");
        return;
    }
```

```c
    int i = qfront;
    bool first = true;
    while (i != qrear) {
        if (!first) printf(" ");
        printf("P%d", proc[q[i]].pid);
        first = false;
        i = (i + 1) % MAX;
    }
}

/* Input */
void inputProcesses() {
    printf("\nEnter number of processes: ");
    if (scanf("%d", &n) != 1) exit(0);
    printf("Enter Time Quantum: ");
    if (scanf("%d", &tq) != 1) exit(0);

    for (int i = 0; i < n; i++) {
        proc[i].pid = i + 1;
        printf("Enter burst time of Process P%d: ", proc[i].pid);
        if (scanf("%d", &proc[i].burst) != 1) exit(0);
        proc[i].remaining = proc[i].burst;
        proc[i].waiting = 0;
        proc[i].turnaround = 0;
    }
}

/* Round Robin: prints step-by-step Gantt chart + ready queue */
void roundRobin() {
    int time = 0, completed = 0;

    /* reset queue */
    qfront = qrear = 0;

    /* initially enqueue all processes (arrival = 0 assumption) */
    for (int i = 0; i < n; i++) enqueue(i);

    printf("\n--- Step by Step Gantt Chart ---\n");
    printf("Interval\t| Executed Process | Ready Queue (during execution)\n");
    printf("--------------------------------------------------------------\n");

    while (completed < n) {
        int idx = dequeue();
        if (idx == -1) { /* shouldn't happen if completed < n, but safeguard */
            printf("No process in ready queue but not all completed. Exiting.\n");
            break;
        }

        if (proc[idx].remaining <= 0) {
            /* If this process had 0 remaining (edge), skip */
            continue;
        }

        int exec = (proc[idx].remaining > tq) ? tq : proc[idx].remaining;
        int start = time;
```

```c
        time += exec;
        proc[idx].remaining -= exec;

        /* Print this slice and the ready queue (which currently contains other waiting processes) */
        printf("%3d - %3d\t|   P%-3d           | ", start, time, proc[idx].pid);
        printReadyQueue();
        printf("\n");

        if (proc[idx].remaining > 0) {
            /* not finished -> re-enqueue at rear */
            enqueue(idx);
        } else {
            /* finished */
            completed++;
            proc[idx].turnaround = time;                 // arrival=0 => turnaround = finish time
            proc[idx].waiting = proc[idx].turnaround - proc[idx].burst;
        }
    }
}

/* Final results */
void displayResults() {
    float totalWT = 0, totalTAT = 0;

    printf("\n--- Final Results ---\n");
    printf("PID\tBurst\tWaiting\tTurnaround\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%5d\t%7d\t%10d\n",
               proc[i].pid, proc[i].burst, proc[i].waiting, proc[i].turnaround);
        totalWT += proc[i].waiting;
        totalTAT += proc[i].turnaround;
    }

    printf("\nAverage Waiting Time    = %.2f\n", totalWT / n);
    printf("Average Turnaround Time= %.2f\n", totalTAT / n);
}

int main() {
    int choice;
    while (1) {
        printf("\n====== Round Robin Scheduling ======\n");
        printf("1. Input Processes\n");
        printf("2. Run Round Robin Scheduling\n");
        printf("3. Display Final Results\n");
        printf("4. Exit\n");
        printf("Enter choice: ");
        if (scanf("%d", &choice) != 1) break;

        switch (choice) {
            case 1:
                inputProcesses();
                break;
            case 2:
```

```
                roundRobin();
                break;
            case 3:
                displayResults();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice!\n");
        }
    }
    return 0;
}
```

Web References

1. Tutorialspoint. Operating System - Round Robin Scheduling. Available at: https://www.tutorialspoint.com/operating_system/os_process_scheduling.htm
2. StudyTonight. Round Robin CPU Scheduling. Available at: https://www.studytonight.com/operating-system/round-robin-scheduling

---

Open-Source Projects / Visualizers

GitHub – CPU Scheduling Visualizer (includes Round Robin).
https://github.com/Ashutosh102/CPU-Scheduling-Visualizer