Experiment No. 5

Aim: Study & implementation of Round Robin (RR) algorithm

Objective: To understand & implement the Round Robin (RR) CPU scheduling algorithm & analyze it's performance in terms of waiting time & turnaround time.

Experiment No. 5

Aim: Study & implementation of Round Robin (RR) algorithm.

Objective: To understand & implement the Round Robin (RR) CPU scheduling Algorithm & analyze it's performance in terms of waiting time & turnaround time.

Theory: Process scheduling in operating systems:

Process scheduling is a fundamental concept in OS that determines the order in which processes are executed by the CPU. One of the most commonly used scheduling algorithms is Round Robin Algorithm.

Round Robin Scheduling Algorithm:

RR is a preemptive scheduling algorithm that assigns a fixed time quantum to each process in the ready state / queue. The CPU executes each process for the given time quantum, then moves to the next process in a cyclic manner.

Key features:

Time Quantum: A fixed time slice assigned to each process.

Preemptive: If a process does not complete within it's allocated time quantum, it is moved to the end of the queue.

Fair scheduling : Ensure that all processes get equal share of time slice.
Efficient for Time sharing systems : Frequently used in multi-user & interactive systems.

Key Parameters :
Arrival Time (AT) : Time at which a process arrives in the system.
Burst Time (BT) : Total execution time for which a process can run before terminating.
Time Quantum : (TQ) : The fixed time for which a process can run before switching.
Waiting Time (WT) : Total time a process spends waiting in the ready queue.
Turnaround Time (TAT) : Total time taken from arrival to completion.

Formulas Used : Turnaround Time (TAT) = Completion time (CT) - Arrival Time (AT)

Waiting Time (WT) = Turnaround Time (TAT) - Burst Time (BT).

Algorithm: START:

i) Input the number of processes
ii) Input the (AT) & (BT) for each process.
iii) Set a fixed time quantum.
iv) Place process in a queue based on (AT).
v) Execute each process for the given time quantum & move it to the end of queue if it is not completed.
vi) Continue until all processes are completed.
vii) Compute (TAT) & (WT)
viii) Compute Average (TAT) & Average (WT).
ix) Display Results
x) STOP.

Code C++ implementation:

```
#include<iostream>
#include<queque>
using namespace std;

struct Process {
        int id, arrival, burst, remaining,
            completion, turnaround, waiting;
        };

void roundRobin(Process processes[], int n,
                            int timeQuantum){
        queue <int> q;
        int time = 0, completed = 0.
```

```
for (int i=0; i<n; i++) {
        processes [i]. remaining = processes [i].
                                        burst;
    }
for (int i=0; i<n; i++) {
        if (processes [i]. arrival == 0) {
                q.push (i);
        }
}
while (!q.empty ()) {
        int index = q.front ();
        q.pop ();

if (processes [index]. remaining > time Quantum ) {
        time += time Quantum;
        processes [index]. remaining -= timeQuantum;
}else {
    time += processes [index].remaining;
    processes [index].remaining = 0;
    processes [index]. completion = time;
    completed ++;
}
for (int =0; i<n; i++) {
        if (processes [i]. arrival <= time && processes [i]
            .remaining > 0 && find (q.begin (), q.end(),
            i )== q.end ()) {
                    q.push().
```

```
    }
}
~~if (processes [i]. arrival <= time~~
if (process [index]. remaining > 0) {
        q. push (index);
}
for (int i = 0; i < n; i++) {
    processes [i]. turnaround = processes [i]. completion
                    - processes [i]. arrival;
    processes [i]. waiting = processes [i]. turnaround
                    - processes [i]. burst;
}
cout << "PID Arrival burst completion Turnaround time
waiting Time \n ");

for (int i=0; i<n; i++) {
        cout << processes [i]. id << "\t" << processes [i]
        . arrival << "\t" << processes [i]. burst << "\t"
    << processes [i]. completion << "\t" << processes [i] .
    turnaround << "\t" << processes [i]. waiting << "\n";
    }
}
int main () {
    Process processes [] = {{1, 0, 8}, {2, 1, 4},
                {3, 2, 9}, {4, 3, 5}};
    int n = 4;
```

Conclusion : The Round Robin scheduling Algorithm is efficient for time-sharing systems & ensures fairness among processes. The selection of an optimum time quantum is crucial for performance.

1/04/25


```
int timeQuantum = 3;
roundRobin(processes, n, timeQuantum);
return 0;
}.
```

Output :

PID	Arrival	Burst	Completion	Turnaround	Waiting
1	0	8	16	16	8
2	1	4	10	9	5
3	2	9	21	19	10
4	3	5	14	11	6

Conclusion: The Round Robin scheduling Algorithm is efficient for time-sharing systems & ensures fairness among processes. The selection of an optimum time quantum is crucial for performance.

1/04/25