ROUND ROBIN ALGORITHM

CSE316 BY- ASHUTOSH KUMAR

AGENDA

INTRODUCTION

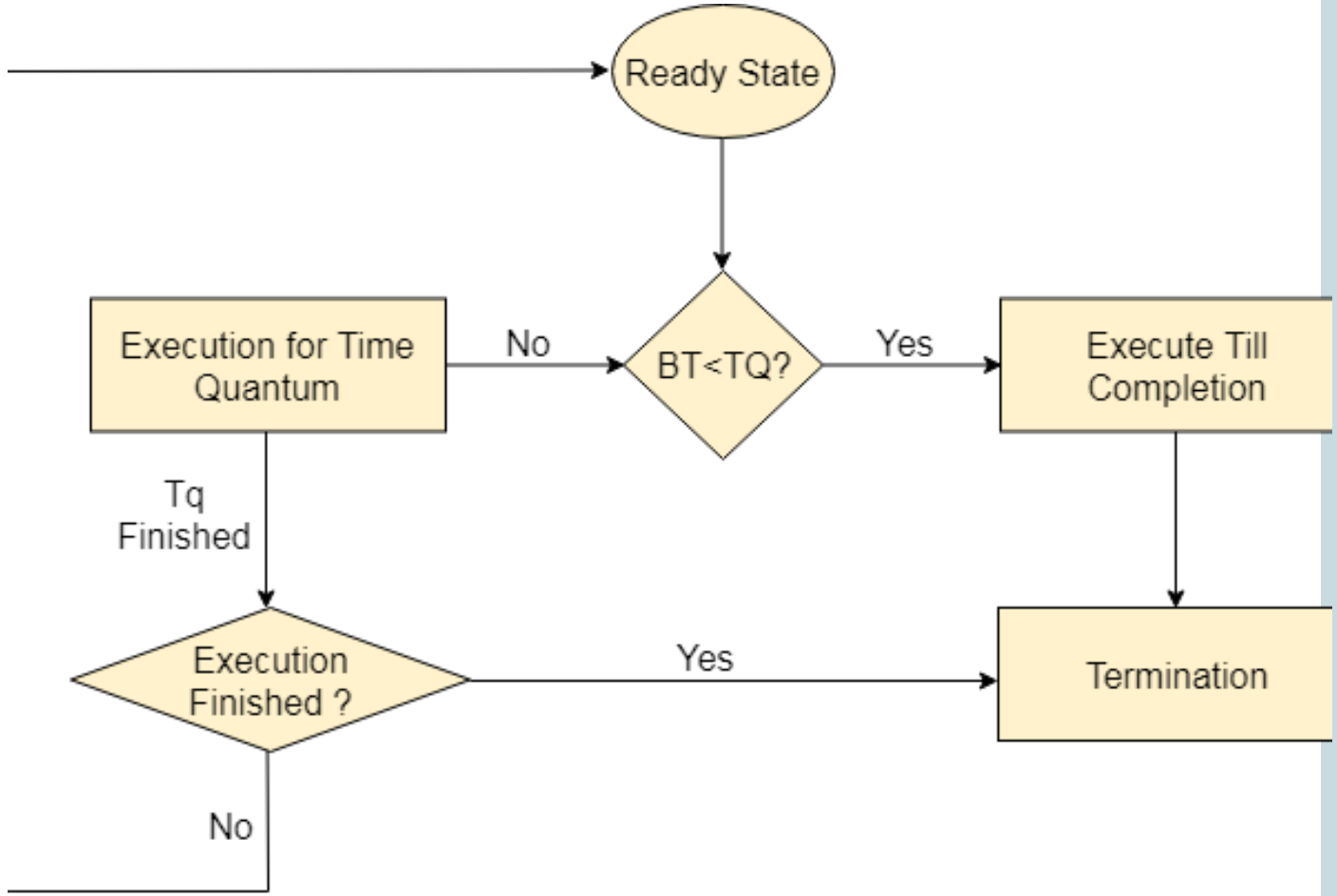
CHARACTERISTICS

ADVANTAGES

DISADVANTAGES

WORKING OF ROUND ROBIN

CODE IMPLEMENTATION



INTRODUCTION

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way. It is basically the preemptive version of First come First Serve CPU Scheduling algorithm.

- Round Robin CPU Algorithm generally focuses on Time Sharing technique.
- The period of time for which a process or job is allowed to run in a pre-emptive method is called time **quantum**.
- Each process or job present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will **end** else the process will go back to the **waiting table** and wait for its next turn to complete the execution.

CHARACTERISTICS

•It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.

•One of the most commonly used technique in CPU scheduling as a core.

•It is preemptive as processes are assigned CPU only for a fixed slice of time at most.

•The disadvantage of it is more overhead of context switching.

ADVANTAGES OF ROUND ROBIN CPU SCHEDULING ALGORITHM:

- There is fairness since every process gets equal share of CPU.
- The newly created process is added to end of ready queue.
- A round-robin scheduler generally employs time-sharing, giving each job a time slot or quantum.
- While performing a round-robin scheduling, a particular time quantum is allotted to different jobs.
- Each process get a chance to reschedule after a particular quantum time in this scheduling.

DISADVANTAGES OF ROUND ROBIN CPU SCHEDULING ALGORITHM:

- There is Larger waiting time and Response time.
- There is Low throughput.
- There is Context Switches.
- Gantt chart seems to come too big (if quantum time is less for scheduling. For Example:1 ms for big scheduling.)
- Time consuming scheduling for small quantum.

EXAMPLE

Time Quantum=2ms

PROCESS	ARRIVAL TIME	BURST TIME
P1	0ms	5ms
P2	1ms	4ms
P3	2ms	2ms
P4	4ms	1ms

The Round Robin CPU Scheduling Algorithm will work on the basis of steps as mentioned below:
At time = 0,

•The execution begins with process P1, which has burst time 5.
•Here, every process executes for 2 milliseconds (**Time Quantum Period**). P2 and P3 are still in the waiting queue.

Time Instance	Process	Arrival Time	Ready Queue	Running Queue	Execution Time	Initial Burst Time	Remaining Burst Time
0-2ms	P1	0ms	P2, P3	P1	2ms	5ms	3ms

EXAMPLE

Time Quantum=2ms

PROCESS	ARRIVAL TIME	BURST TIME
P1	0ms	5ms
P2	1ms	4ms
P3	2ms	2ms
P4	4ms	1ms

At time = 2,

•The processes P2 and P3 arrives in the ready queue and P2 starts executing for **TQ** period

Time Instance	Process	Arrival Time	Ready Queue	Running Queue	Execution Time	Initial Burst Time	Remaining Burst Time
2-4ms	P1	0ms	P3, P1	P2	0ms	3ms	3ms
	P2	1ms			2ms	4ms	2ms

At time = 4,

•The process P4 arrives in the **ready queue**,

•Then P3 executes for **TQ** period.

Time Instance	Process	Arrival Time	Ready Queue	Running Queue	Execution Time	Initial Burst Time	Remaining Burst Time
4-6ms	P1	0ms	P1, P4, P2	P3	0ms	3ms	3ms
	P2	1ms			0ms	2ms	2ms
	~~P3~~	~~2ms~~			~~2ms~~	~~2ms~~	~~0ms~~

EXAMPLE

At time = 6,

•Process P3 completes its execution

•Process P1 starts executing for **TQ** period as it is next in the b.

Time Instance	Process	Arrival Time	Ready Queue	Running Queue	Execution Time	Initial Burst Time	Remaining Burst Time
6-8ms	P1	0ms	P4, P2	P1	2ms	3ms	1ms
	P2	1ms			0ms	2ms	2ms

At time = 8,

•Process P4 starts executing, it will not execute for **Time Quantum period** as it has burst time = 1

•Hence, it will execute for only 1ms.

Time Instance	Process	Arrival Time	Ready Queue	Running Queue	Execution Time	Initial Burst Time	Remaining Burst Time
8-9ms	P1	0ms	P2, P1	P4	0ms	3ms	1ms
	P2	1ms			0ms	2ms	2ms
	~~P4~~	~~4ms~~			1ms	1ms	0ms

EXAMPLE

At time = 9,

•Process P4 completes its execution

•Process P2 starts executing for **TQ** period as it is next in the **ready queue**

Time Instance	Process	Arrival Time	Ready Queue	Running Queue	Execution Time	Initial Burst Time	Remaining Burst Time
9-11ms	P1	0ms	P1	P2	0ms	3ms	1ms
	~~P2~~	~~1ms~~			~~2ms~~	~~2ms~~	~~0ms~~

At time = 11,

•Process P2 completes its execution.

•Process P1 starts executing, it will execute for 1ms only

Time Instance	Process	Arrival Time	Ready Queue	Running Queue	Execution Time	Initial Burst Time	Remaining Burst Time
11-12ms	~~P1~~	~~0ms~~		~~P1~~	~~1ms~~	~~1ms~~	~~0ms~~

EXAMPLE

At time = 12,

•Process P1 completes its execution.

•The overall execution of the processes will be as shown below:

Time Instance	Process	Arrival Time	Ready Queue	Running Queue	Execution Time	Initial Burst Time	Remaining Burst Time
0-2ms	**P1**	**0ms**	**P2, P3**	**P1**	**2ms**	**5ms**	**3ms**
2-4ms	P1	0ms	P3, P1	P2	0ms	3ms	3ms
	P2	**1ms**			**2ms**	**4ms**	**2ms**
4-6ms	P1	0ms	P1, P4, P2	P3	0ms	3ms	3ms
	P2	1ms			0ms	2ms	2ms
	~~P3~~	~~2ms~~			~~2ms~~	~~2ms~~	~~0ms~~
6-8ms	**P1**	**0ms**	P4, P2	P1	**2ms**	**3ms**	**1ms**
	P2	1ms			0ms	2ms	2ms
8-9ms	P1	0ms	P2, P1	P4	0ms	3ms	1ms
	P2	1ms			0ms	2ms	2ms
	~~P4~~	~~4ms~~			**1ms**	**1ms**	**0ms**
9-11ms	P1	0ms	P1	P2	0ms	3ms	1ms
	~~P2~~	~~1ms~~			~~2ms~~	~~2ms~~	~~0ms~~
11-12ms	~~P1~~	~~0ms~~		~~P1~~	~~1ms~~	~~1ms~~	~~0ms~~

EXAMPLE

Gantt chart will be as following below:

How to compute below times in Round Robin using a program?

- **Completion Time:** Time at which process completes its execution.
- **Turn Around Time:** Time Difference between completion time and arrival time. **Turn Around Time = Completion Time – Arrival Time**
- **Waiting Time(W.T):** Time Difference between turn around time and burst time.

Waiting Time = Turn Around Time – Burst Time

Processes	AT	BT	CT	TAT	WT
P1	0	5	12	12-0 = 12	12-5 = 7
P2	1	4	11	11-1 = 10	10-4 = 6
P3	2	2	6	6-2 = 4	4-2 = 2
P4	4	1	9	9-4 = 5	5-1 = 4

- **Average Turn around time** = (12 + 10 + 4 + 5)/4 = 31/4 = 7.7
- **Average waiting time** = (7 + 6 + 2 + 4)/4 = 19/4 = 4.7

ALGORITHM TO IMPLEMENT ROUND ROBIN SCHEDULING

1. Define the maximum number of processes.
2. Define a process structure with the following fields:
    1. name
    2. arrival_time
    3. burst_time
    4. completion_time
    5. turnaround_time
    6. waiting_time
    7. remaining_time
3. Define the main function that does the following:
    - Declare variables for filename, time slice, array of processes, number of processes, current time, completed processes, total turnaround time, and total waiting time.
    - Prompt the user for the filename and time slice.
    - Open the input file.
    - Read the list of processes from the file and store them in the array of processes.
    - Simulate the execution of the processes using the Round Robin scheduling algorithm:

- While the number of completed processes is less than the total number of processes:
  - Loop through the array of processes.
  - For each process with remaining time:
    - If the remaining time is less than or equal to the time slice:
      - Update the current time to reflect the completion of the process.
      - Calculate the completion time, turnaround time, and waiting time for the process.
      - Increment the number of completed processes.
      - Update the total turnaround time and total waiting time.
      - Set the remaining time of the process to zero.
    - If the remaining time is greater than the time slice:
      - Update the current time to reflect the passage of the time slice.
      - Reduce the remaining time of the process by the time slice.
- Print the results:
  - For each process, print the arrival time, burst time, completion time, turnaround time, and waiting time.
  - Print the average turnaround time and average waiting time for all processes.

4. End the main function.

CODE

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

#define MAX_PROCESS 100

struct process {
    char name[10];
    int arrival_time;
    int burst_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int remaining_time;
};

int main() {
    char filename[20];
    int time_slice;
    struct process processes[MAX_PROCESS];
    int num_processes = 0;
    int current_time = 0;
    int completed_processes = 0;
    int i, j;
    int total_turnaround_time = 0, total_waiting_time = 0;

    // Prompt user for filename and time slice
    printf("Enter the filename: ");
    scanf("%s", filename);
    printf("Enter the time slice: ");
    scanf("%d", &time_slice);

    // Open input file
    FILE *fp = fopen(filename, "r");
    if (fp == NULL) {
        printf("Error opening file\n");
        return 1;
    }

    // Read input file and populate process array
    while (fscanf(fp, "%s %d %d", processes[num_processes].name,
&processes[num_processes].arrival_time,
&processes[num_processes].burst_time) != EOF) {
        processes[num_processes].remaining_time =
processes[num_processes].burst_time;
        num_processes++;
    }
    fclose(fp);

    // Simulate execution of processes
    while (completed_processes < num_processes) {
        for (i = 0; i < num_processes; i++) {
            if (processes[i].remaining_time > 0) {
                if (processes[i].remaining_time <= time_slice) {
                    current_time += processes[i].remaining_time;
                    processes[i].remaining_time = 0;
                    processes[i].completion_time = current_time;
                    processes[i].turnaround_time = processes[i].completion_time
- processes[i].arrival_time;
                    processes[i].waiting_time = processes[i].turnaround_time -
processes[i].burst_time;
                    completed_processes++;
                    total_turnaround_time += processes[i].turnaround_time;
                    total_waiting_time += processes[i].waiting_time;
                } else {
                    current_time += time_slice;
                    processes[i].remaining_time -= time_slice;
                }
            }
        }
    }

    // Print results
    printf("\nProcess\t\tArrival Time\tBurst
Time\tCompletion Time\tTurnaround Time\tWaiting
Time\n");
    for (i = 0; i < num_processes; i++) {
        printf("%s\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
processes[i].name, processes[i].arrival_time,
processes[i].burst_time, processes[i].completion_time,
processes[i].turnaround_time, processes[i].waiting_time);
    }
    printf("\nAverage Turnaround Time: %.2f\n", (float)
total_turnaround_time / num_processes);
    printf("Average Waiting Time: %.2f\n", (float)
total_waiting_time / num_processes);

    return 0;
}
```

SAMPLE OUTPUT

TEXT FILE- process.txt

P1	0	5
P2	1	4
P3	2	2
P4	4	1

Enter the filename: processes.txt
Enter the time slice: 2

Processes	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
P1	0	5	12	12	7
P2	1	4	11	10	6
P3	2	2	6	4	2
P4	4	1	9	5	4

Average Turnaround Time: 7.70
Average Waiting Time: 4.70

THANK YOU