ROUND ROBIN

Compiled by: Alina Liaquat

BSCS-5TH-A

What is Gantt Chart?

In operating system scheduling, a Gantt chart is a visual representation of a process schedule, showing which process is being executed by the CPU at any given time. This helps visualize a process's sequence, duration, and potential idle time. It clearly shows the order in which processes are executed, when the CPU is busy, and when it is idle. **Analysis**: Gantt charts are used to analyze the performance of different scheduling algorithms, helping to calculate metrics like average turnaround time and waiting time for processes.

Important terms to know:

What is Arrival Time (AT)?

The time when a process arrives in the ready queue (when it enters the system)

What is Burst Time (BT)?

The total time required by a process on the CPU for execution

What is Completion Time (CT)?

The time at which a process finishes its execution.

What is Turnaround Time (TAT)?

Total time spent by the process in the system from arrival to completion

TAT = Completion Time - Arrival Time

What is Waiting Time (WT)?

Total time a process spends waiting in the ready queue (not executing)

WT = Turnaround Time - Burst Time

Response Time (RT):

Time from arrival until the process gets the CPU for the first time.

RT = Time of first CPU allocation - Arrival Time

What is Round Robin Scheduling in OS?

The Round robin scheduling algorithm is one of the CPU scheduling algorithms in which every process gets a fixed amount of **time quantum** to execute the process.

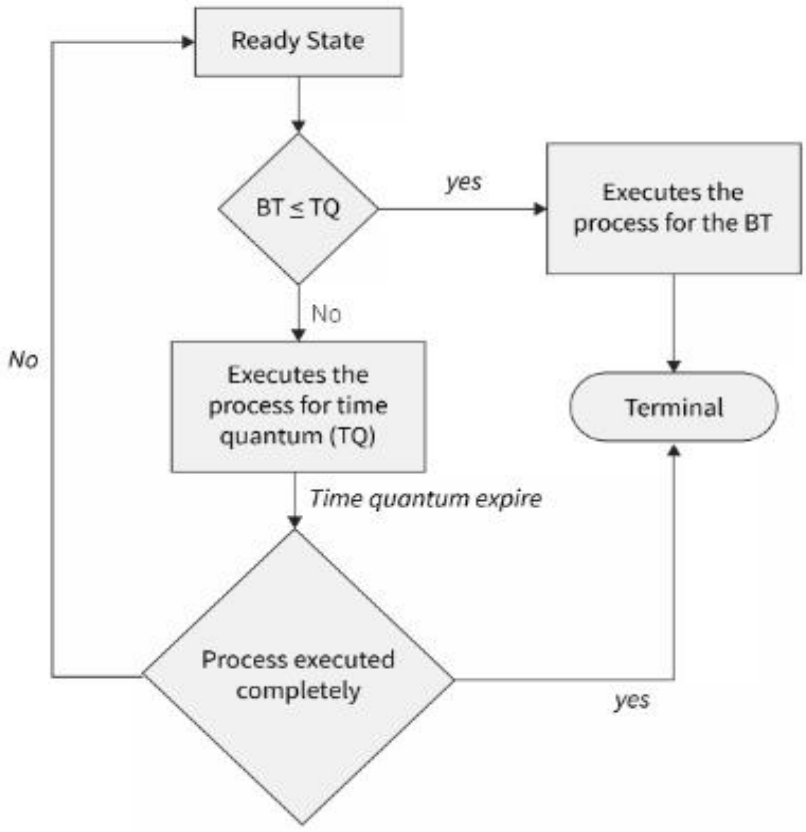
In this algorithm, every process gets executed cyclically. This means that processes that have their **burst time remaining** after the expiration of the time quantum are sent back to the ready state and wait for their next turn to complete the execution until it **terminates**. This processing is done in **FIFO** order which suggests that processes are executed on a first-come, first-serve basis.

The CPU time quantum is the time period defined in the system.

How does the Round Robin Algorithm Work?

1. All the processes are added to the ready queue.
2. At first, The burst time of every process is compared to the time quantum of the CPU.
3. If the burst time of the process is **less than or equal** to the time quantum in the round-robin scheduling algorithm, the process is executed to its burst time.
4. If the burst time of the process is **greater than** the time quantum, the process is executed up to the time quantum (TQ).
5. When the time quantum expires, it checks if the process is executed completely or not.
6. On completion, the process terminates. Otherwise, it goes back again to the ready state.

Consider the below flow diagram for a better understanding of Round Robin scheduling algorithm:



Example:

Given:

- Time Quantum (TQ) = 4 units
- 3 Processes with Arrival Time and Burst Time:

Process	Arrival Time (AT)	Burst Time (BT)
P1	0	5
P2	1	3
P3	2	8

Step-by-step execution:

Time 0:

Only P1 is in the ready queue (because P2 and P3 have not arrived yet).

P1's burst time = 5, Time Quantum (TQ) = 4.

Since burst time is greater than TQ, P1 runs on the CPU for 4 units.

Time 4:

P1's remaining burst time = 5 - 4 = 1 unit.

Now P2 (arrived at time 1) and P3 (arrived at time 2) have also arrived and joined the ready queue.

P1 is sent to the end of the ready queue.

Time 4 to 7:

P2's burst time = 3, which is less than the time quantum (4), so P2 runs for its full burst time.

P2 finishes at time 7.

Time 7 to 11:

P3's burst time = 8, TQ = 4, so P3 runs for 4 units.

Remaining burst time for P3 = 8 - 4 = 4.

Time 11 to 12:

P1 returns to the CPU and runs its remaining 1 unit

P1 finishes at time 12.

Time 12 to 16:

P3 runs its remaining 4 units and completes at time 16

Implementation of Round Robin Algorithm in C++

```cpp
#include <iostream>     // for standard input/output streams (cout)
#include <queue>        // STL (Standard Template Library) queue
#include <vector>       // STL vector container for dynamic arrays
#include <string>       // for using string class
#include <iomanip>      // for formatting output (setw, left)
using namespace std;

// Process class represents a single process in the scheduler
class Process {
public:
    string name;
    int burstTime;
    int remainingTime;
    int waitingTime = 0;
    int turnaroundTime = 0;

    // Constructor to initialize a process with name and burst time
    Process(string n, int bt) {
        name = n;
        burstTime = bt;
        remainingTime = bt;  // Initialize remaining time as burst time
    }
};

 class RoundRobinScheduler {
private:
    int timeQuantum;
    queue<int> readyQueue;
    vector<Process> processes;

    struct GanttEntry {
        string name;
        int start;
        int end;
    };

    vector<GanttEntry> gantt;

public:
    RoundRobinScheduler(int tq) {
        timeQuantum = tq;
    }

    void addProcess(string name, int burst) {
        processes.emplace_back(name, burst);
        readyQueue.push(processes.size() - 1);
    }

    void runScheduler() {
        int currentTime = 0;

        while (!readyQueue.empty()) {
            int index = readyQueue.front();
            readyQueue.pop();
```

```
            Process &p = processes[index];

            int start = currentTime;

            if (p.remainingTime > timeQuantum) {
                p.remainingTime -= timeQuantum;
                currentTime += timeQuantum;
                readyQueue.push(index);
            } else {
                currentTime += p.remainingTime;
                p.waitingTime = currentTime - p.burstTime;
                p.remainingTime = 0;
            }

            // Save Gantt Chart Entry
            gantt.push_back({p.name, start, currentTime});
        }

        // Turnaround Times
        for (auto &p : processes)
            p.turnaroundTime = p.waitingTime + p.burstTime;
    }

    void printGanttChart() {
        cout << "\n=== GANTT CHART (TABLE FORMAT) ===\n\n";

        cout << left << setw(15) << "Process"
             << setw(10) << "Start"
             << setw(10) << "End" << "\n";

        cout << string(35, '-') << "\n";

        for (auto &g : gantt) {
            cout << left << setw(15) << g.name
                 << setw(10) << g.start
                 << setw(10) << g.end
                 << "\n";
        }

        cout << "\n";
    }

    void printProcessTable() {
        cout << "=== PROCESS TABLE ===\n\n";

        cout << left << setw(15) << "Process"
             << setw(10) << "Burst"
             << setw(10) << "Waiting"
             << setw(12) << "Turnaround" << "\n";

        cout << string(50, '-') << "\n";

        for (auto &p : processes) {
            cout << left << setw(15) << p.name
                 << setw(10) << p.burstTime
```

```
                << setw(10) << p.waitingTime
                << setw(12) << p.turnaroundTime
                << "\n";
        }
    }
};

int main()
{
    RoundRobinScheduler scheduler(2);

    scheduler.addProcess("Chrome", 5);
    scheduler.addProcess("VSCode", 3);
    scheduler.addProcess("Terminal", 8);
    scheduler.addProcess("Spotify", 6);
    scheduler.addProcess("Explorer", 4);

    cout << "\n=== ROUND ROBIN CPU SCHEDULER (TQ = 2) ===\n";

    scheduler.runScheduler();
    scheduler.printGanttChart();
    scheduler.printProcessTable();

    return 0;
}
```

Round Robin CPU Scheduler in C++ (Ubuntu Guide)

1. Introduction

This manual guides you through:

- Creating a C++ source file
- Writing the Round Robin Scheduler code
- Compiling the program using g++
- Running the final executable
- Understanding the output

2.System Requirements

Before starting, make sure your Ubuntu system has:

- Ubuntu OS (any LTS version recommended)
- Terminal (default GNOME Terminal is fine)
- G++ compiler
- A text editor (nano or VS Code)

To check if g++ is installed, run:

```
g++ --version
```

If it shows a version, you are ready.

If not, install it using:

```
sudo apt update

sudo apt install g++
```

3. Creating the Project Directory

It is recommended to store your project in a separate folder

Step 1: Open the Terminal

Press:

```
CTRL + ALT + T
```

Step 2: Create a directory

```
mkdir round_robin_scheduler
```

Step 3: Enter the directory

```
cd round_robin_scheduler
```

4. Creating the C++ Source File

Step 1: Create a new file

Use nano:

```
nano scheduler.cpp
```

This opens an empty file inside the terminal.

Nano is a simple, user-friendly, and lightweight command-line text editor commonly found in Linux and Unix-like operating systems. It is often the default text editor in many distributions, including Ubuntu

Step 2: Paste the C++ Code

Paste your Round Robin C++ OOP code inside this file.

Step 3: Save the file

Press:

```
CTRL + O

ENTER

CTRL + X
```

This saves and exits nano

5. Compiling the Program

Step 1: Compile using g++

Inside the same folder, run:

```
g++ scheduler.cpp -o scheduler
```

Explanation:

- scheduler.cpp → your source code
- -o scheduler → output file/executable name

Step 2: Check if executable was created

Run:

```
ls
```

You should see:

```
scheduler

scheduler.cpp
```

6. Running the Program

Step 1: Execute the compiled program

```
./scheduler
```

This will run your Round Robin CPU Scheduler

Step 2: Observe the output

The output will show:

- **Gantt Chart (Table Format)**
- **Process Table**

  Burst Time, Waiting Time, Turnaround Time

What is this Gantt Chart showing?

The Gantt Chart is a **timeline representation** of how each process got CPU time slices (time quantum = 2) in the Round Robin scheduling

Column	Meaning
Process	The name of the process that was running in this time slice
Start	The time at which the process started running this slice
End	The time at which the process stopped running this slice

Time Quantum = 2

Each process gets **up to 2 units of CPU time** per turn in the queue.

If the process still needs more time after that, it goes back to the **end of the queue** and waits for its next turn.

Process	Start	End	Duration (End - Start)	Explanation
Chrome	0	2	2	Chrome runs first for 2 units
VSCode	2	4	2	Then VSCode runs for 2 units
Terminal	4	6	2	Terminal runs for 2 units
Spotify	6	8	2	Spotify runs for 2 units
Explorer	8	10	2	Explorer runs for 2 units
Chrome	10	12	2	Chrome gets next 2 units
VSCode	12	13	1	VSCode needs only 1 unit now
Terminal	13	15	2	Terminal runs again 2 units
Spotify	15	17	2	Spotify runs again 2 units
Explorer	17	19	2	Explorer runs again 2 units
Chrome	19	20	1	Chrome finishes with 1 unit
Terminal	20	22	2	Terminal runs again 2 units
Spotify	22	24	2	Spotify runs again 2 units
Terminal	24	26	2	Terminal finishes last 2 units

What the Gantt Chart tells us about process execution:

- Each process executes in a **fair, cyclic order**: Chrome → VSCode → Terminal → Spotify → Explorer → repeat
- No process gets the CPU all at once; instead, they run in **time slices of 2 units each**
- Some processes finish earlier (like VSCode in its second turn with only 1 unit left)
- Terminal runs several times because it has the longest burst time (8 units)
- The total time shown ends at 26 units, which is when the last process (Terminal) finishes