Operating System (CS301)

Assignment - 6

U19CS012

Write a program for the simulation of

Basic Description

- ✓ Take "**n**" no. of process from user with **arrival_time** and **burst_time**.
- ✓ **arrival_time** and **burst_time** should be generated randomly and Compute *Completion Time*, *Turnaround (TAT) Time* and *Waiting Time*.
- ✓ Also show the Count of Context Switching in all of the algorithms.

1. Shortest Job First (SJF)

- ✓ Process which have the **Shortest Burst Time** are scheduled first.
- ✓ If two processes have same Burst Time then **FCFS** is used to break the tie.
- ✓ It is a **Non-Preemptive** scheduling algorithm.
- ✓ In non-preemptive scheduling, once the CPU cycle is allocated to process, the process **holds it** till it reaches a waiting state or terminated.

Code

```cpp
// Shortest Job First Algorithm - BHAGYA RANA [U19CS012]

#include <bits/stdc++.h>
using namespace std;

// "Process" Class
class Process
{
public:
    // Process ID
    int id;
    // Arrival Time: Time at which the process arrives in the ready queue.
    int arrivalTime;
    // Burst Time: Time required by a process for CPU execution.
    int burstTime;
    // Completion Time: Time at which process completes its execution.
    int completionTime;
    // Turn Around Time: Time Difference between completion time and arrival time.
```

```
    int turnaroundTime;
    // Waiting Time(W.T): Time Difference between turn around time and burst time.
    int waitingTime;

    // Constructor for Intializing
    Process(int id, int arrivalTime, int burstTime)
    {
        this->id = id;
        this->arrivalTime = arrivalTime;
        this->burstTime = burstTime;
    }
};

// Compare F(x) for Sorting p
bool compare(const Process &p1, const Process &p2);

// Function for Shortest Job First Scheduling
void shortestJobFirst(vector<Process> &p);

// F(x) to Generate Random Input
vector<Process> randomInputGenerator(int n);

// F(x) to Print the Output in Well Structured Format
void printOutput(vector<Process> &p);

// F(x) to Call the Algorithm
void solve(vector<Process> &p);

int main()
{
    cout << "--------------------SHORTEST JOB FIRST ALGORITHM--------------------\n";
    int n;
    cout << "Enter Number of Processes : ";
    cin >> n;
    char choice;
    cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";
    cout << "1 -> Random Input\n2 -> User Input\n\n";
    cout << "Enter Your Choice : ";
    cin >> choice;
    cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

    vector<Process> p;
    if (choice == '1')
    {
        p = randomInputGenerator(n);
        cout << "Randomly generated inputs are : " << endl;
        for (int i = 0; i < n; i++)
        {
            cout << "Arrival time and Burst time of Process " << p[i].id << " : " << p[i].arr
ivalTime << " " << p[i].burstTime << endl;
```

```
        }
    }
    else if (choice == '2')
    {
        for (int i = 0; i < n; i++)
        {
            int arriveTime, burstTime;
            cout << "Enter Arrival time and Burst time of Process " << i + 1 << " : ";
            cin >> arriveTime >> burstTime;
            p.push_back(Process(i + 1, arriveTime, burstTime));
        }
    }
    else
    {
        cout << "Incorrect Input Entered\n";
        return 0;
    }

    solve(p);

    return 0;
}

// Customized function for sorting
bool compare(const Process &p1, const Process &p2)
{
    if (p1.arrivalTime == p2.arrivalTime)
    {
        return p1.burstTime < p2.burstTime;
    }
    return p1.arrivalTime < p2.arrivalTime;
}

// Function for Shortest Job First Scheduling
void shortestJobFirst(vector<Process> &p)
{
    int temp, curr;

    // 1st process would complete 1st.
    p[0].completionTime = p[0].arrivalTime + p[0].burstTime;
    p[0].turnaroundTime = p[0].completionTime - p[0].arrivalTime;
    p[0].waitingTime = p[0].turnaroundTime - p[0].burstTime;

    // Now for other processes...
    for (int i = 1; i < p.size(); i++)
    {
        temp = p[i - 1].completionTime; // completion time

        // Burst Time: Time required by a process for CPU execution.
        int minBurst = p[i].burstTime;
```

```
        // Current index for swapping.
        curr = -1;

        while (curr == -1)
        {
            for (int j = i; j < p.size(); j++)
            {
                // If completion time >= arrive time =>[Implies] a process is in queue
                // if minBurst >= burstTime of curr process then, we need to swap.
                if (temp >= p[j].arrivalTime and minBurst >= p[j].burstTime)
                {
                    minBurst = p[j].burstTime;
                    curr = j;
                }
            }

            // If no such process found
            if (curr == -1)
                temp = p[i].arrivalTime;
        }

        // Time at which process completes its execution.= Completion Time
        p[curr].completionTime = temp + p[curr].burstTime;
        // Turn Around Time = Completion Time - Arrival Time
        p[curr].turnaroundTime = p[curr].completionTime - p[curr].arrivalTime;
        // Waiting Time = Turn Around Time - Burst Time
        p[curr].waitingTime = p[curr].turnaroundTime - p[curr].burstTime;

        // If process is found then swapping it with ith process.
        Process temp = p[i];
        p[i] = p[curr];
        p[curr] = temp;
    }
}

// Function to generate random input
vector<Process> randomInputGenerator(int n)
{
    unsigned seed = chrono::system_clock::now().time_since_epoch().count();
    default_random_engine generator(seed);
    uniform_int_distribution<int> d1(0, ((10 * n) + 1) / 2);
    uniform_int_distribution<int> d2(1, 10);

    vector<Process> p;

    for (int i = 0; i < n; i++)
    {
        p.push_back(Process(i + 1, d1(generator), d2(generator)));
        d1.reset();
```

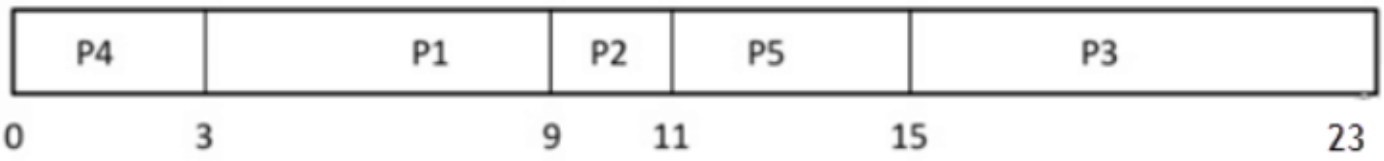
```
    }
    return p;
}

// Referrence : https://stackoverflow.com/questions/14765155/how-can-i-easily-format-my-data-table-in-c
template <typename T>
void printElement(T t)
{
    cout << left << setw(17) << setfill(' ') << t;
}

// Utility function to print Output
void printOutput(vector<Process> &p)
{
    int TotalWaiting = 0, TotalTurnAround = 0;
    cout << "\n----------------------------------SCHEDULED---PROCESS---DETAILS----------
------------------\n\n";

    printElement("Process ID");
    printElement("Arrival Time");
    printElement("Burst Time");
    printElement("Completion Time");
    printElement("Turn Around Time");
    printElement("Waiting Time");
    cout << endl;

    for (int i = 0; i < p.size(); i++)
    {
        cout << "     ";
        printElement(p[i].id);
        printElement(p[i].arrivalTime);
        printElement(p[i].burstTime);
        printElement(p[i].completionTime);
        printElement(p[i].turnaroundTime);
        printElement(p[i].waitingTime);

        cout << endl;

        TotalWaiting += p[i].waitingTime;
        TotalTurnAround += p[i].turnaroundTime;
    }

    cout << "\n----------------------------------SCHEDULED---PROCESS---SUMMARY----------
------------------\n\n";

    cout << "Average Waiting Time : " << (double)TotalWaiting / (double)p.size() << endl;
    cout << "Average Turn Around Time : " << (double)TotalTurnAround / (double)p.size() << en
dl;
    cout << "Context Switch : 0\n\n";
```

```cpp
}

// Function which will call other needed functions.
void solve(vector<Process> &p)
{
    sort(p.begin(), p.end(), compare);
    shortestJobFirst(p);
    printOutput(p);
}
```

Output [Non-Preemptive SJF]

Process Queue	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

```
--------------------SHORTEST JOB FIRST ALGORITHM--------------------
Enter Number of Processes : 5
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1 -> Random Input
2 -> User Input

Enter Your Choice : 2
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Enter Arrival time and Burst time of Process 1 : 2 6
Enter Arrival time and Burst time of Process 2 : 5 2
Enter Arrival time and Burst time of Process 3 : 1 8
Enter Arrival time and Burst time of Process 4 : 0 3
Enter Arrival time and Burst time of Process 5 : 4 4

---------------------------------SCHEDULED---PROCESS---DETAILS---------------------------------

Process ID       Arrival Time     Burst Time       Completion Time  Turn Around Time Waiting Time
     4                0                3                3                3                0
     1                2                6                9                7                1
     2                5                2                11               6                4
     5                4                4                15               11               7
     3                1                8                23               22               14

---------------------------------SCHEDULED---PROCESS---SUMMARY---------------------------------

Average Waiting Time : 5.2
Average Turn Around Time : 9.8
Context Switch : 0
```




```
Wait time
P4= 0-0=0
P1=  3-2=1
P2= 9-5=4
P5= 11-4=7
P3= 15-1=14
```

```
Average Waiting Time= 0+1+4+7+14/5 = 26/5 = 5.2
```

Random Input

```
--------------------SHORTEST JOB FIRST ALGORITHM--------------------
Enter Number of Processes : 6
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1 -> Random Input
2 -> User Input

Enter Your Choice : 1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Randomly generated inputs are :
Arrival time and Burst time of Process 1 : 11 2
Arrival time and Burst time of Process 2 : 22 10
Arrival time and Burst time of Process 3 : 24 9
Arrival time and Burst time of Process 4 : 1 4
Arrival time and Burst time of Process 5 : 8 9
Arrival time and Burst time of Process 6 : 11 6

---------------------------------SCHEDULED---PROCESS---DETAILS---------------------------------

Process ID       Arrival Time     Burst Time       Completion Time  Turn Around Time Waiting Time
     4                1                4                5                4                0
     5                8                9                17               9                0
     1                11               2                19               8                6
     6                11               6                25               14               8
     3                24               9                34               10               1
     2                22               10               44               22               12

---------------------------------SCHEDULED---PROCESS---SUMMARY---------------------------------

Average Waiting Time : 4.5
Average Turn Around Time : 11.1667
Context Switch : 0
```

2. Shortest Remaining Time First (SRTF) CPU scheduler.

- ✓ **Preemptive** mode of **Shortest Job First** is called as *Shortest Remaining Time First (SRTF).*
- ✓ In SRTF, the Execution of Process can be stopped after certain amount of time.

Code

```
// Shortest Remaining Time First Algorithm - BHAGYA RANA [U19CS012]
#include <bits/stdc++.h>
using namespace std;

// "Process" Class
class Process
{
public:
    // Process ID
    int id;
    // Arrival Time: Time at which the process arrives in the ready queue.
    int arrivalTime;
    // Burst Time: Time required by a process for CPU execution.
    int burstTime;
    // Completion Time: Time at which process completes its execution.
    int completionTime;
    // Turn Around Time: Time Difference between completion time and arrival time.
    int turnaroundTime;
    // Waiting Time(W.T): Time Difference between turn around time and burst time.
    int waitingTime;
    int remainingTime;

    // Constructor for Intializing
    Process(int id, int arrivalTime, int burstTime)
    {
        this->id = id;
        this->arrivalTime = arrivalTime;
        this->burstTime = burstTime;
        this->remainingTime = burstTime;
    }
};

// Compare F(x) for Sorting p
bool compare(const Process &p1, const Process &p2);

// Function for Shortest Remaining Time First Scheduling
void shortestRemainingTimeFirst(vector<Process> &p);

// F(x) to Generate Random Input
```

```cpp
vector<Process> randomInputGenerator(int n);

// F(x) to Print the Output in Well Structured Format
void printOutput(vector<Process> &p);

// F(x) to Call the Algorithm
void solve(vector<Process> &p);

int main()
{
    cout << "--------------------SHORTEST REMAINING TIME FIRST ALGORITHM--------------------
\n";

    int n;
    cout << "Enter Number of Processes : ";
    cin >> n;

    char choice;

    cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";
    cout << "1 -> Random Input\n2 -> User Input\n\n";
    cout << "Enter Your Choice : ";
    cin >> choice;
    cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n";

    vector<Process> p;
    if (choice == '1')
    {
        p = randomInputGenerator(n);
        cout << "Randomly generated inputs are : " << endl;
        for (int i = 0; i < n; i++)
        {
            cout << "Arrival time and Burst time of Process " << p[i].id << " : " << p[i].arr
ivalTime << " " << p[i].burstTime << endl;
        }
    }
    else if (choice == '2')
    {
        for (int i = 0; i < n; i++)
        {
            int arriveTime, burstTime;
            cout << "Enter Arrival time and Burst time of Process " << i + 1 << " : ";
            cin >> arriveTime >> burstTime;
            p.push_back(Process(i + 1, arriveTime, burstTime));
        }
    }
    else
    {
        cout << "Incorrect Input Entered\n";
        return 0;
```

```
    }

    solve(p);

    return 0;
}

// Customized function for sorting
bool compare(const Process &p1, const Process &p2)
{
    if (p1.arrivalTime == p2.arrivalTime)
        return p1.burstTime < p2.burstTime;
    return p1.arrivalTime < p2.arrivalTime;
}

// Function for Shortest Job First Scheduling
void shortestRemainingTimeFirst(vector<Process> &p)
{
    // Count No. of Context Switch
    int contextSwitch = 0, done = 0, currTime = 0, minRemainTime = INT_MAX, minProcess = 0;
    bool flag = false;

    // For Gantt Chart
    string time, processFlow;

    // Loop till all processes are completed.
    while (done != p.size())
    {
        time += to_string(currTime) + " ";
        if (currTime < 10)
            time += " ";

        bool flagTemp = false;

        //Checking every process to find process with minimum remianing time.

        for (int i = 0; i < p.size(); i++)
        {
            // Condition for minimum remaining time.
            if (p[i].arrivalTime <= currTime and p[i].remainingTime < minRemainTime and p[i].
remainingTime > 0)
            {
                // a new process is executed i.e. no context switch
                if (minRemainTime == INT_MAX)
                    flagTemp = true;

                minRemainTime = p[i].remainingTime;
                minProcess = i;
                flag = true;
```

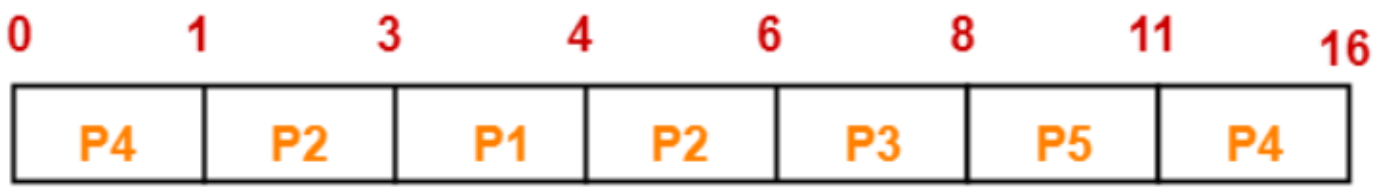
```
                if (flagTemp)
                    continue;

                // Increasing context switch count
                contextSwitch += 1;
            }
        }

        // No process in queue with minimum remaining time.
        if (!flag)
        {
            if (minRemainTime == INT_MAX)
                processFlow += "    ";
            else
                processFlow += "P" + to_string(p[minProcess].id) + " ";

            currTime++;
            continue;
        }

        processFlow += "P" + to_string(p[minProcess].id) + " ";

        //Decrementing remaining time of the feasible process.
        p[minProcess].remainingTime -= 1;
        // Update minRemainTime
        minRemainTime = p[minProcess].remainingTime;

        if (minRemainTime == 0)
        {
            // Process is completed therefore resetting.
            minRemainTime = INT_MAX;
            // Increase completed process count
            done += 1;

            flag = false;
            // Updating Data for a process
            p[minProcess].completionTime = currTime + 1;

            p[minProcess].waitingTime = p[minProcess].completionTime - p[minProcess].burstTim
e - p[minProcess].arrivalTime;

            if (p[minProcess].waitingTime < 0)
                p[minProcess].waitingTime = 0;

            p[minProcess].turnaroundTime = p[minProcess].waitingTime + p[minProcess].burstTim
e;
        }
        currTime += 1;
    }
```

```
    cout << "\n------------------------------GANTT---CHART-----------------------------\n";
    cout << "Time   : " + time + to_string(currTime) << endl;
    cout << "Process    : " + processFlow << endl;
    cout << "Total Number of Context Switching : " << contextSwitch << endl;
}

// Function to generate random input
vector<Process> randomInputGenerator(int n)
{
    unsigned seed = chrono::system_clock::now().time_since_epoch().count();
    default_random_engine generator(seed);
    uniform_int_distribution<int> d1(0, ((10 * n) + 1) / 2);
    uniform_int_distribution<int> d2(1, 10);

    vector<Process> p;

    for (int i = 0; i < n; i++)
    {
        p.push_back(Process(i + 1, d1(generator), d2(generator)));
        d1.reset();
    }
    return p;
}

// Referrence : https://stackoverflow.com/questions/14765155/how-can-i-easily-format-my-data-table-in-c
template <typename T>
void printElement(T t)
{
    cout << left << setw(17) << setfill(' ') << t;
}

// Utility function to print Output
void printOutput(vector<Process> &p)
{
    int TotalWaiting = 0, TotalTurnAround = 0;
    cout << "\n---------------------------------SCHEDULED---PROCESS---DETAILS---------------
------------------\n\n";

    printElement("Process ID");
    printElement("Arrival Time");
    printElement("Burst Time");
    printElement("Completion Time");
    printElement("Turn Around Time");
    printElement("Waiting Time");
    cout << endl;

    for (int i = 0; i < p.size(); i++)
    {
        cout << "     ";
```

```
        printElement(p[i].id);
        printElement(p[i].arrivalTime);
        printElement(p[i].burstTime);
        printElement(p[i].completionTime);
        printElement(p[i].turnaroundTime);
        printElement(p[i].waitingTime);

        cout << endl;

        TotalWaiting += p[i].waitingTime;
        TotalTurnAround += p[i].turnaroundTime;
    }

    cout << "\n----------------------------------SCHEDULED---PROCESS---SUMMARY---------------
------------------\n\n";

    cout << "Average Waiting Time : " << (double)TotalWaiting / (double)p.size() << endl;
    cout << "Average Turn Around Time : " << (double)TotalTurnAround / (double)p.size() << en
dl;
}

// Function which will call other needed functions.
void solve(vector<Process> &p)
{
    sort(p.begin(), p.end(), compare);
    shortestRemainingTimeFirst(p);
    printOutput(p);
}
```

Output

Process ID	Arrival Time	Burst Time
1	3	1
2	1	4
3	4	2
4	0	6
5	2	3



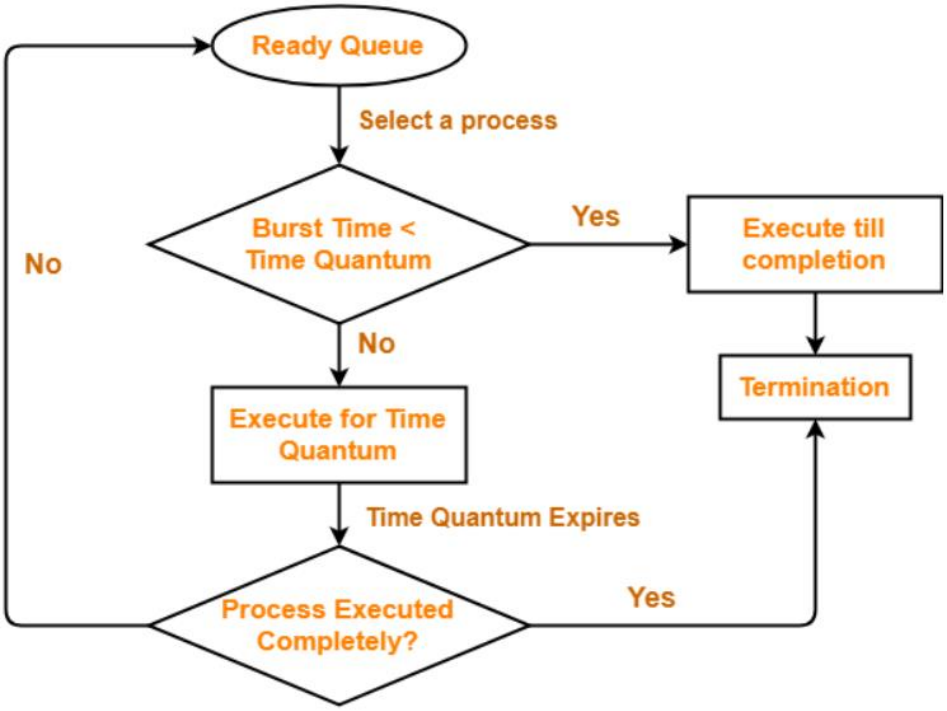
Gantt Chart

```
------------------SHORTEST REMAINING TIME FIRST ALGORITHM------------------
Enter Number of Processes : 5
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1 -> Random Input
2 -> User Input

Enter Your Choice : 2
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Enter Arrival time and Burst time of Process 1 : 3 1
Enter Arrival time and Burst time of Process 2 : 1 4
Enter Arrival time and Burst time of Process 3 : 4 2
Enter Arrival time and Burst time of Process 4 : 0 6
Enter Arrival time and Burst time of Process 5 : 2 3

------------------------------GANTT---CHART-----------------------------
Time    : 0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
Process : P4 P2 P2 P1 P2 P2 P3 P3 P5 P5 P5 P4 P4 P4 P4 P4
Total Number of Context Switching : 2

---------------------------------SCHEDULED---PROCESS---DETAILS---------------------------------

Process ID       Arrival Time     Burst Time       Completion Time  Turn Around Time Waiting Time
     4                0                6                16               16               10
     2                1                4                6                5                1
     5                2                3                11               9                6
     1                3                1                4                1                0
     3                4                2                8                4                2

---------------------------------SCHEDULED---PROCESS---SUMMARY---------------------------------

Average Waiting Time : 3.8
Average Turn Around Time : 7
```

Randomly Generated Input

```
------------------SHORTEST REMAINING TIME FIRST ALGORITHM------------------
Enter Number of Processes : 4
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1 -> Random Input
2 -> User Input

Enter Your Choice : 1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Randomly generated inputs are :
Arrival time and Burst time of Process 1 : 3 5
Arrival time and Burst time of Process 2 : 15 10
Arrival time and Burst time of Process 3 : 13 1
Arrival time and Burst time of Process 4 : 8 9

------------------------------GANTT---CHART-----------------------------
Time    : 0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
Process :                     P1 P1 P1 P1 P1 P4 P4 P4 P4 P4 P3 P4 P4 P4 P4 P2 P2 P2 P2 P2 P2 P2 P2 P2 P2
Total Number of Context Switching : 1

---------------------------------SCHEDULED---PROCESS---DETAILS---------------------------------

Process ID       Arrival Time     Burst Time       Completion Time  Turn Around Time Waiting Time
     1                3                5                8                5                0
     4                8                9                18               10               1
     3                13               1                14               1                0
     2                15               10               28               13               3

---------------------------------SCHEDULED---PROCESS---SUMMARY---------------------------------

Average Waiting Time : 1
Average Turn Around Time : 7.25
```

3. Round Robin Scheduling.



Round Robin Scheduling

- ✓ 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
- ✓ The disadvantage of it is **more overhead** of Context Switching.

Code

```
// Round Robin Algorithm - BHAGYA RANA [U19CS012]
#include <bits/stdc++.h>
using namespace std;

// "Process" Class
class Process
{
public:
    // Process ID
    int id;
    // Arrival Time: Time at which the process arrives in the ready queue.
    int arrivalTime;
    // Burst Time: Time required by a process for CPU execution.
    int burstTime;
    // Completion Time: Time at which process completes its execution.
```

```
    int completionTime;
    // Turn Around Time: Time Difference between completion time and arrival time.
    int turnaroundTime;
    // Waiting Time(W.T): Time Difference between turn around time and burst time.
    int waitingTime;
    int remainingTime;

    // Constructor for Intializing
    Process(int id, int arrivalTime, int burstTime)
    {
        this->id = id;
        this->arrivalTime = arrivalTime;
        this->burstTime = burstTime;
        this->remainingTime = burstTime;
    }
};

// Compare F(x) for Sorting p
bool compare(const Process &p1, const Process &p2);

// F(x) for Round Robin Scheduling
void roundRobin(vector<Process> &p, int interval);

// F(x) to Generate Random Input
vector<Process> randomInputGenerator(int n);

// F(x) to Print the Output in Well Structured Format
void printOutput(vector<Process> &p);

// F(x) to Call the Algorithm
void solve(vector<Process> &p, int interval);

int main()
{
    cout << "--------------------ROUND ROBIN ALGORITHM--------------------\n";

    int n;
    cout << "Enter Number of Processes : ";
    cin >> n;

    int interval;
    cout << "Enter Time Quantum for Round Robin : ";
    cin >> interval;

    char choice;
    cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";
    cout << "1 -> Random Input\n2 -> User Input\n\n";
    cout << "Enter Your Choice : ";
    cin >> choice;
    cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n";
```

```
    vector<Process> p;
    if (choice == '1')
    {
        p = randomInputGenerator(n);
        cout << "Randomly generated inputs are : " << endl;
        for (int i = 0; i < n; i++)
        {
            cout << "Arrival time and Burst time of Process " << p[i].id << " : " << p[i].arr
ivalTime << " " << p[i].burstTime << endl;
        }
    }
    else if (choice == '2')
    {
        for (int i = 0; i < n; i++)
        {
            int arriveTime, burstTime;
            cout << "Enter Arrival time and Burst time of Process " << i + 1 << " : ";
            cin >> arriveTime >> burstTime;
            p.push_back(Process(i + 1, arriveTime, burstTime));
        }
    }
    else
    {
        cout << "Incorrect Input Entered\n";
        return 0;
    }

    solve(p, interval);

    return 0;
}

// Customized function for sorting
bool compare(const Process &p1, const Process &p2)
{
    return p1.arrivalTime < p2.arrivalTime;
}

// F(x) for Round Robin Scheduling
void roundRobin(vector<Process> &p, int interval)
{
    // Count Context Switch
    int contextSwitch = 0;
    int currTime = 0;
    int prev = -1;

    // Queue to maintain processes.
    queue<int> q;
    string time = "";
```

```
    string processFlow = "";

    // While all processes are not done.
    while (true)
    {
        bool flag = true;
        for (int i = 0; i < p.size(); i++)
        {
            if (p[i].remainingTime != 0)
            {
                flag = false;
                break;
            }
        }
        // If all processes are covered.
        if (flag)
            break;

        // If queue is empty, it means that there are no process currently in waiting.
        // Therefore, jumping to the next process by incrementing currTime.
        if (q.empty())
        {
            int prevTime = currTime;
            for (int i = 0; i < p.size(); i++)
            {
                if (p[i].remainingTime != 0)
                {
                    currTime = p[i].arrivalTime;
                    q.push(i);
                    int j = i + 1;
                    while (j < p.size() and p[j].arrivalTime == currTime and p[j].remainingTi
me > 0)
                    {
                        q.push(j);
                        j++;
                    }
                    break;
                }
            }
            for (int k = prevTime; k < currTime; k++)
                processFlow += " ";
        }

        // Popping the process from queue.
        int current = q.front();
        q.pop();

        // Incrementing context switch if the previous process is not completed
        // and also it's not the same as current process
        if (prev != -1 and prev != current)
```

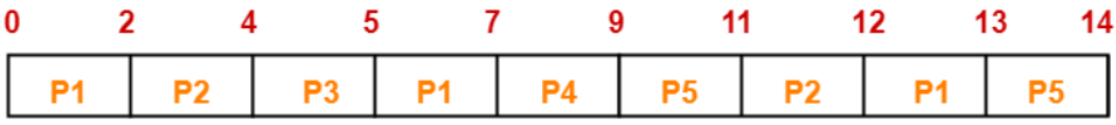
```
            contextSwitch++;

        // If remaining time is greater than interval then we will minus interval time
        // and also add processes to queue which arrived in that interval.
        if (p[current].remainingTime > interval)
        {
            p[current].remainingTime -= interval;
            for (int j = current + 1; j < p.size(); j++)
            {
                if (p[j].arrivalTime > currTime and p[j].arrivalTime <= currTime + interval)
                    q.push(j);
            }
            q.push(current);
            currTime += interval;
            prev = current;

            for (int k = 0; k < interval; k++)
                processFlow += "P" + to_string(p[current].id) + " ";
        }
        // If remaining time is lesser than interval then we will make remaining time ZERO
        // and also add processes to queue which arrived in that time while the process was r
unning.
        else
        {
            for (int j = current + 1; j < p.size(); j++)
            {
                if (p[j].arrivalTime > currTime and p[j].arrivalTime <= currTime + p[current]
.remainingTime)
                    q.push(j);
            }
            for (int k = 0; k < p[current].remainingTime; k++)
                processFlow += "P" + to_string(p[current].id) + " ";

            currTime += p[current].remainingTime;

            // Updating records for a process which is completed.
            p[current].remainingTime = 0;
            p[current].completionTime = currTime;
            p[current].turnaroundTime = currTime - p[current].arrivalTime;
            p[current].waitingTime = currTime - p[current].burstTime - p[current].arrivalTime
;
            prev = -1;
        }
    }
    for (int k = 0; k < currTime; k++)
    {
        time += (to_string(k) + " ");
        if (k < 10)
            time += " ";
    }
```

```
    cout << "\n------------------------------GANTT---CHART-----------------------------
\n\n";
    cout << "Time : " + time << endl;
    cout << "Process : " + processFlow << endl;
    cout << "Total Number of Context Switching : " << contextSwitch << endl;
}

// Function to generate random input
vector<Process> randomInputGenerator(int n)
{
    unsigned seed = chrono::system_clock::now().time_since_epoch().count();
    default_random_engine generator(seed);
    uniform_int_distribution<int> d1(0, ((10 * n) + 1) / 2);
    uniform_int_distribution<int> d2(1, 10);

    vector<Process> p;

    for (int i = 0; i < n; i++)
    {
        p.push_back(Process(i + 1, d1(generator), d2(generator)));
        d1.reset();
    }
    return p;
}

// Referrence : https://stackoverflow.com/questions/14765155/how-can-i-easily-format-my-data-
table-in-c
template <typename T>
void printElement(T t)
{
    cout << left << setw(17) << setfill(' ') << t;
}

// Utility function to print Output
void printOutput(vector<Process> &p)
{
    int TotalWaiting = 0, TotalTurnAround = 0;
    cout << "\n---------------------------------SCHEDULED---PROCESS---DETAILS--------------
-------------------\n\n";

    printElement("Process ID");
    printElement("Arrival Time");
    printElement("Burst Time");
    printElement("Completion Time");
    printElement("Turn Around Time");
    printElement("Waiting Time");
    cout << endl;

    for (int i = 0; i < p.size(); i++)
```

```
    {
        cout << "     ";
        printElement(p[i].id);
        printElement(p[i].arrivalTime);
        printElement(p[i].burstTime);
        printElement(p[i].completionTime);
        printElement(p[i].turnaroundTime);
        printElement(p[i].waitingTime);

        cout << endl;

        TotalWaiting += p[i].waitingTime;
        TotalTurnAround += p[i].turnaroundTime;
    }

    cout << "\n---------------------------------SCHEDULED---PROCESS---SUMMARY--------------
-------------------\n\n";

    cout << "Average Waiting Time : " << (double)TotalWaiting / (double)p.size() << endl;
    cout << "Average Turn Around Time : " << (double)TotalTurnAround / (double)p.size() << en
dl;
}

// Function which will call other needed functions.
void solve(vector<Process> &p, int interval)
{
    sort(p.begin(), p.end(), compare);
    roundRobin(p, interval);
    printOutput(p);
}
```

Output

Process ID	Arrival Time	Burst Time
1	0	5
2	1	3
3	2	1
4	3	1
5	4	3



Gantt Chart

Process Id	Exit time	Turn Around time	Waiting time
P1	13	13 – 0 = 13	13 – 5 = 8
P2	12	12 – 1 = 11	11 – 3 = 8
P3	5	5 – 2 = 3	3 – 1 = 2
P4	9	9 – 3 = 6	6 – 2 = 4
P5	14	14 – 4 = 10	10 – 3 = 7

Now,

- Average Turn Around time = (13 + 11 + 3 + 6 + 10) / 5 = 43 / 5 = 8.6 unit
- Average waiting time = (8 + 8 + 2 + 4 + 7) / 5 = 29 / 5 = 5.8 unit

```
------------------ROUND ROBIN ALGORITHM------------------
Enter Number of Processes : 5
Enter Time Quantum for Round Robin : 2          <-- Interval
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1 -> Random Input
2 -> User Input

Enter Your Choice : 2
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Enter Arrival time and Burst time of Process 1 : 0 5
Enter Arrival time and Burst time of Process 2 : 1 3
Enter Arrival time and Burst time of Process 3 : 2 1
Enter Arrival time and Burst time of Process 4 : 3 2
Enter Arrival time and Burst time of Process 5 : 4 3

----------------------------GANTT---CHART----------------------------

Time : 0  1  2  3  4  5  6  7  8  9  10 11 12 13
Process : P1 P1 P2 P2 P3 P1 P1 P4 P4 P5 P5 P2 P1 P5
Total Number of Context Switching : 4

---------------------------------SCHEDULED---PROCESS---DETAILS---------------------------------

Process ID       Arrival Time     Burst Time       Completion Time  Turn Around Time Waiting Time
     1                0                5                13               13               8
     2                1                3                12               11               8
     3                2                1                5                3                2
     4                3                2                9                6                4
     5                4                3                14               10               7

---------------------------------SCHEDULED---PROCESS---SUMMARY---------------------------------

Average Waiting Time : 5.8
Average Turn Around Time : 8.6
```

Randomly Generated Input

```
Enter Number of Processes : 5
Enter Time Quantum for Round Robin : 3
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1 -> Random Input
2 -> User Input

Enter Your Choice : 1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Randomly generated inputs are :
Arrival time and Burst time of Process 1 : 14 6
Arrival time and Burst time of Process 2 : 13 6
Arrival time and Burst time of Process 3 : 16 7
Arrival time and Burst time of Process 4 : 17 4
Arrival time and Burst time of Process 5 : 11 5

------------------------------GANTT---CHART------------------------------

Time : 0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
Process :            P5 P5 P5 P2 P2 P2 P1 P1 P1 P5 P5 P3 P3 P3 P4 P4 P4 P2 P2 P2 P1 P1 P1 P3 P3 P3 P4 P3
Total Number of Context Switching : 6   <-- Greater Context Switches

---------------------------------SCHEDULED---PROCESS---DETAILS---------------------------------

Process ID       Arrival Time     Burst Time       Completion Time  Turn Around Time Waiting Time
     5               11               5               22               11               6
     2               13               6               31               18               12
     1               14               6               34               20               14
     3               16               7               39               23               16
     4               17               4               38               21               17

---------------------------------SCHEDULED---PROCESS---SUMMARY---------------------------------

Average Waiting Time : 13
Average Turn Around Time : 18.6
```

Thus, we have Successfully Understood and **Implemented**

- ✓ Shortest Job First (SJF) [Non Preemptive]
- ✓ Shortest Remaining Time First (SRTF) [Preemptive SJF]
- ✓ Round Robin Scheduling

SUBMITTED BY:

U19CS012

BHAGYA VINOD RANA