UNIVERSITY OF ASIA PACIFIC

Course Title: Operating System Lab
Course Code: CSE 406

Date of Submission: 1st August,2025

Submitted by:

Name: Md.Sarjil Hasan

Reg: 22101168

Sec : D

Submitted to:

Atia Rahman Orthi

Lecturer

University of Asia Pacific

Problem Statement:

Round Robin CPU Scheduling Algorithm.

Process ID	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

Completion Time (CT)
Turnaround Time (TAT) = CT – Arrival Time(AT)
Waiting Time (WT) = TAT – Burst Time(BT)

The algorithm should also compute and display the average waiting time and average turnaround time.

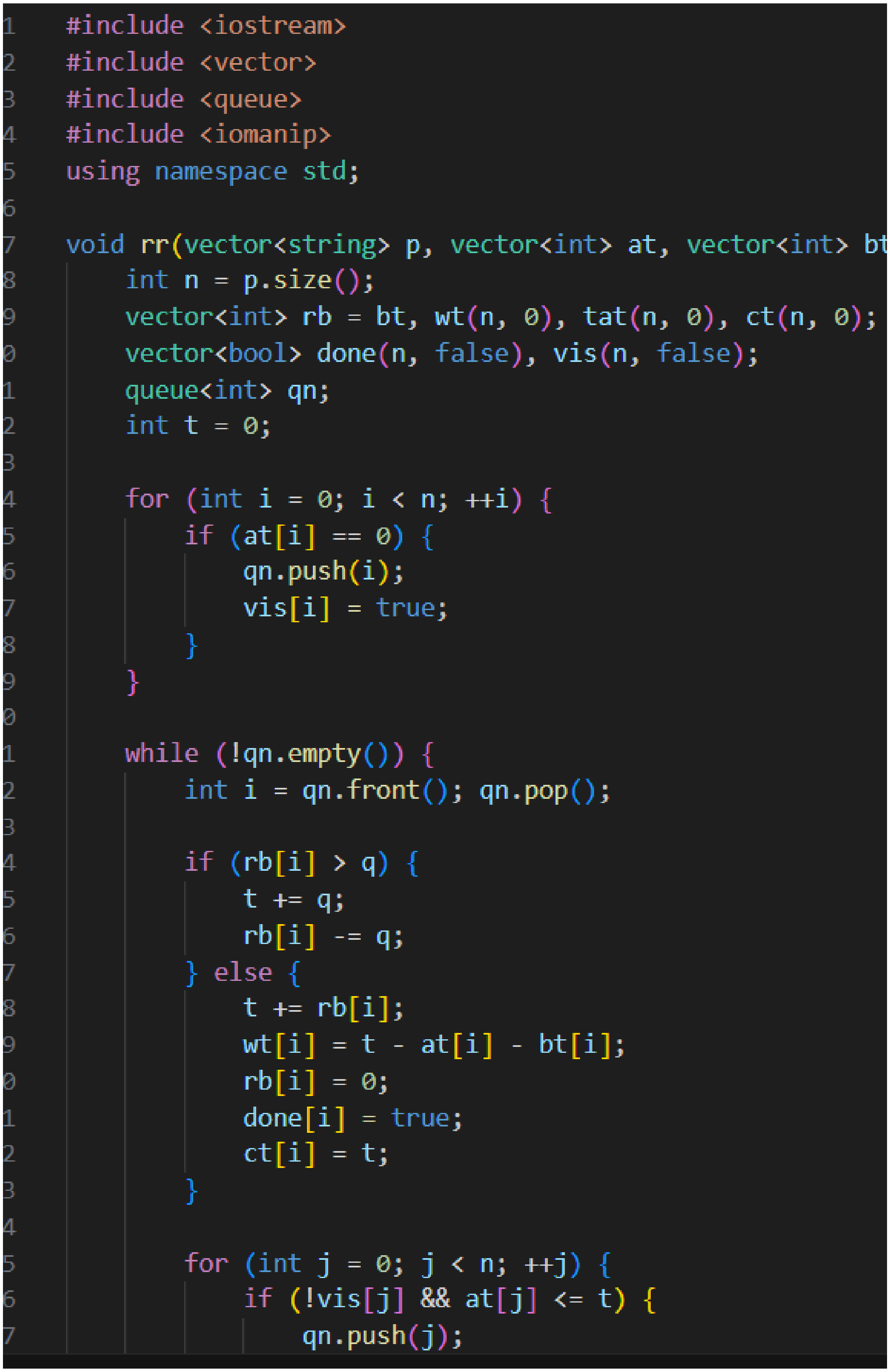
Objective:

Simulate the **Round Robin CPU scheduling algorithm** for a given set of processes with their **arrival times** and **burst times**, using a fixed **time quantum**.

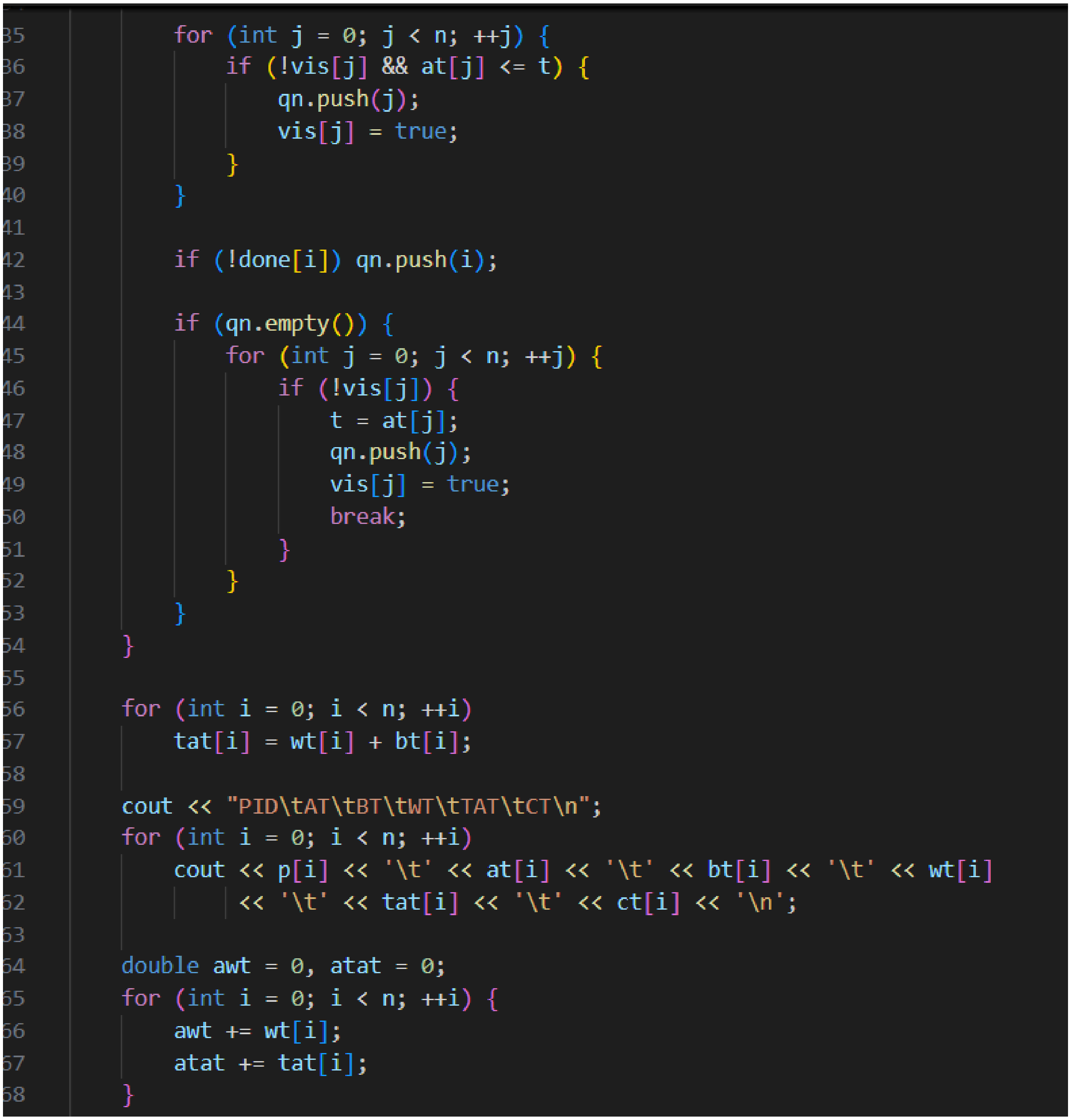
Algorithm Steps:

1. Start from the first process in the ready queue.
2. Give CPU for a time slice (quantum).
3. If the process completes in time, remove it from the queue.
4. If not, reduce remaining time and push it to the end of the queue.
5. Repeat until all processes complete.

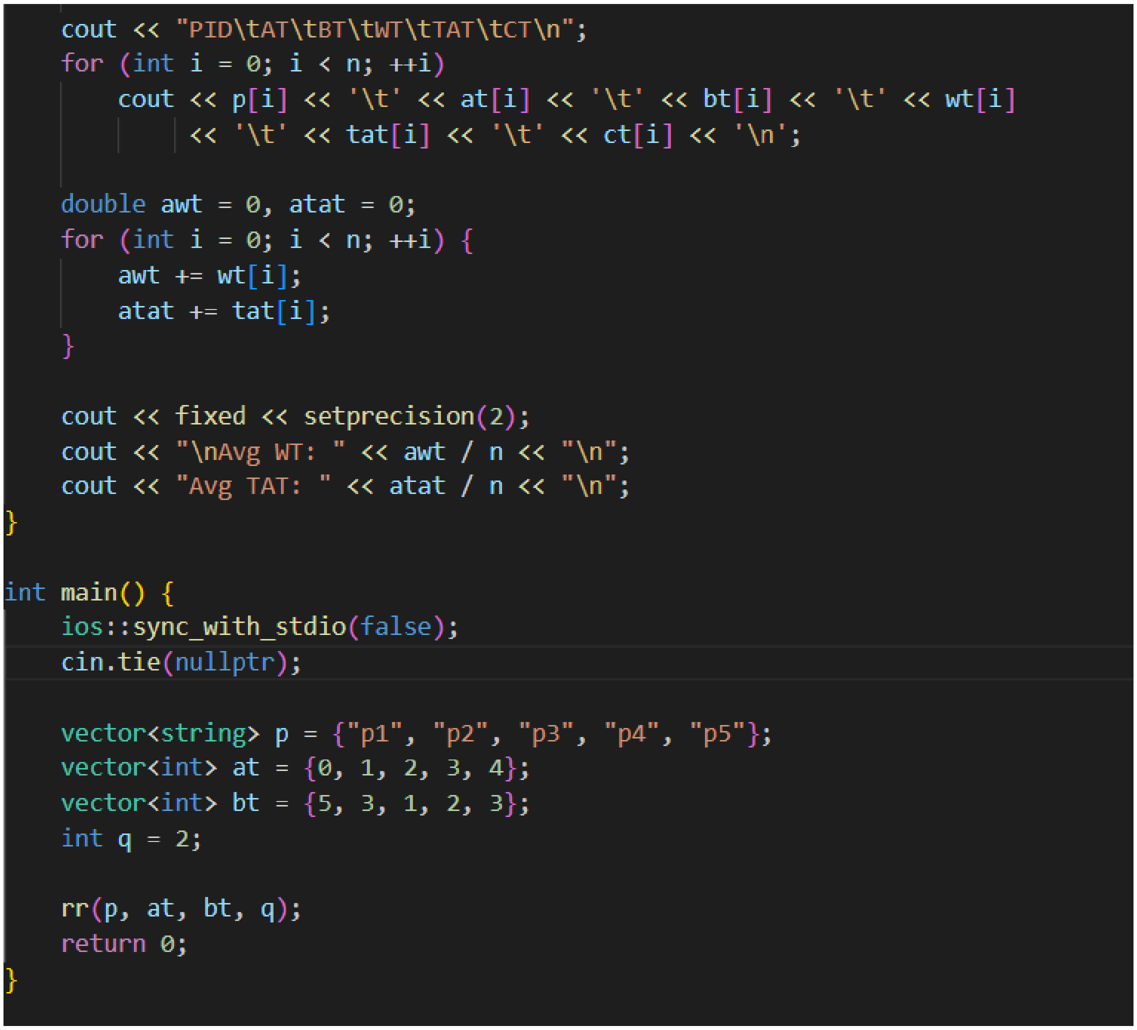
Code:



```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <iomanip>
using namespace std;

void rr(vector<string> p, vector<int> at, vector<int> bt
    int n = p.size();
    vector<int> rb = bt, wt(n, 0), tat(n, 0), ct(n, 0);
    vector<bool> done(n, false), vis(n, false);
    queue<int> qn;
    int t = 0;

    for (int i = 0; i < n; ++i) {
        if (at[i] == 0) {
            qn.push(i);
            vis[i] = true;
        }
    }

    while (!qn.empty()) {
        int i = qn.front(); qn.pop();

        if (rb[i] > q) {
            t += q;
            rb[i] -= q;
        } else {
            t += rb[i];
            wt[i] = t - at[i] - bt[i];
            rb[i] = 0;
            done[i] = true;
            ct[i] = t;
        }

        for (int j = 0; j < n; ++j) {
            if (!vis[j] && at[j] <= t) {
                qn.push(j);
```

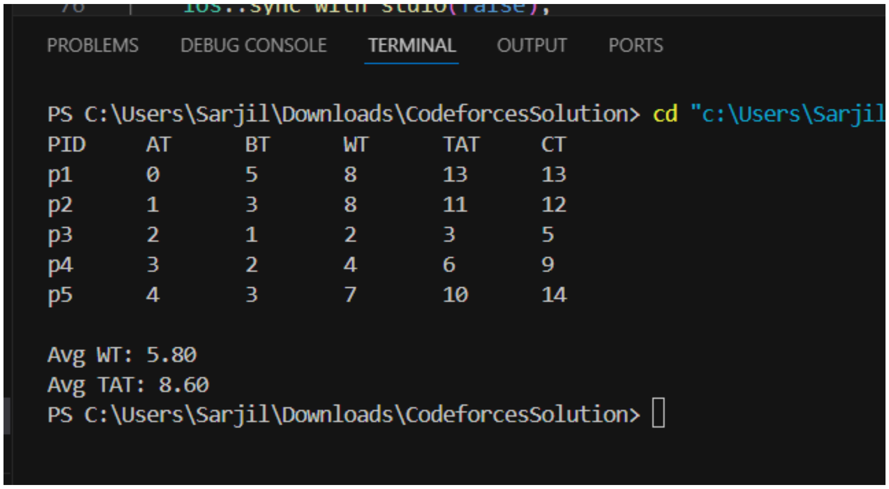


```cpp
        for (int j = 0; j < n; ++j) {
            if (!vis[j] && at[j] <= t) {
                qn.push(j);
                vis[j] = true;
            }
        }

        if (!done[i]) qn.push(i);

        if (qn.empty()) {
            for (int j = 0; j < n; ++j) {
                if (!vis[j]) {
                    t = at[j];
                    qn.push(j);
                    vis[j] = true;
                    break;
                }
            }
        }
    }

    for (int i = 0; i < n; ++i)
        tat[i] = wt[i] + bt[i];

    cout << "PID\tAT\tBT\tWT\tTAT\tCT\n";
    for (int i = 0; i < n; ++i)
        cout << p[i] << '\t' << at[i] << '\t' << bt[i] << '\t' << wt[i]
             << '\t' << tat[i] << '\t' << ct[i] << '\n';

    double awt = 0, atat = 0;
    for (int i = 0; i < n; ++i) {
        awt += wt[i];
        atat += tat[i];
    }
```



```cpp
    cout << "PID\tAT\tBT\tWT\tTAT\tCT\n";
    for (int i = 0; i < n; ++i)
        cout << p[i] << '\t' << at[i] << '\t' << bt[i] << '\t' << wt[i]
             << '\t' << tat[i] << '\t' << ct[i] << '\n';

    double awt = 0, atat = 0;
    for (int i = 0; i < n; ++i) {
        awt += wt[i];
        atat += tat[i];
    }

    cout << fixed << setprecision(2);
    cout << "\nAvg WT: " << awt / n << "\n";
    cout << "Avg TAT: " << atat / n << "\n";
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    vector<string> p = {"p1", "p2", "p3", "p4", "p5"};
    vector<int> at = {0, 1, 2, 3, 4};
    vector<int> bt = {5, 3, 1, 2, 3};
    int q = 2;

    rr(p, at, bt, q);
    return 0;
}
```

Output:



```
ios::sync_with_stdio(false);
```

```
PS C:\Users\Sarjil\Downloads\CodeforcesSolution> cd "c:\Users\Sarjil
PID     AT      BT      WT      TAT     CT
p1      0       5       8       13      13
p2      1       3       8       11      12
p3      2       1       2       3       5
p4      3       2       4       6       9
p5      4       3       7       10      14

Avg WT: 5.80
Avg TAT: 8.60
PS C:\Users\Sarjil\Downloads\CodeforcesSolution> 
```

Advantages:

1.Simple and easy to implement.
2.Provides fairness and avoids starvation.
3.Ideal for time-sharing systems.

Disadvantages:

1.Performance depends on time quantum.
2.Too small: too many context switches.
3.Too large: behaves like FCFS.

Conclusion:

In this lab, we successfully implemented the Round Robin Scheduling Algorithm. We calculated waiting time, turnaround time, and learned how the choice of time quantum affects CPU scheduling performance.

Github Link:
https://github.com/Sarjil-SarZzz/CSE406-LAB.git