Operating systems

Lab Assignment 5

Create C programs for the different scheduling algorithms.

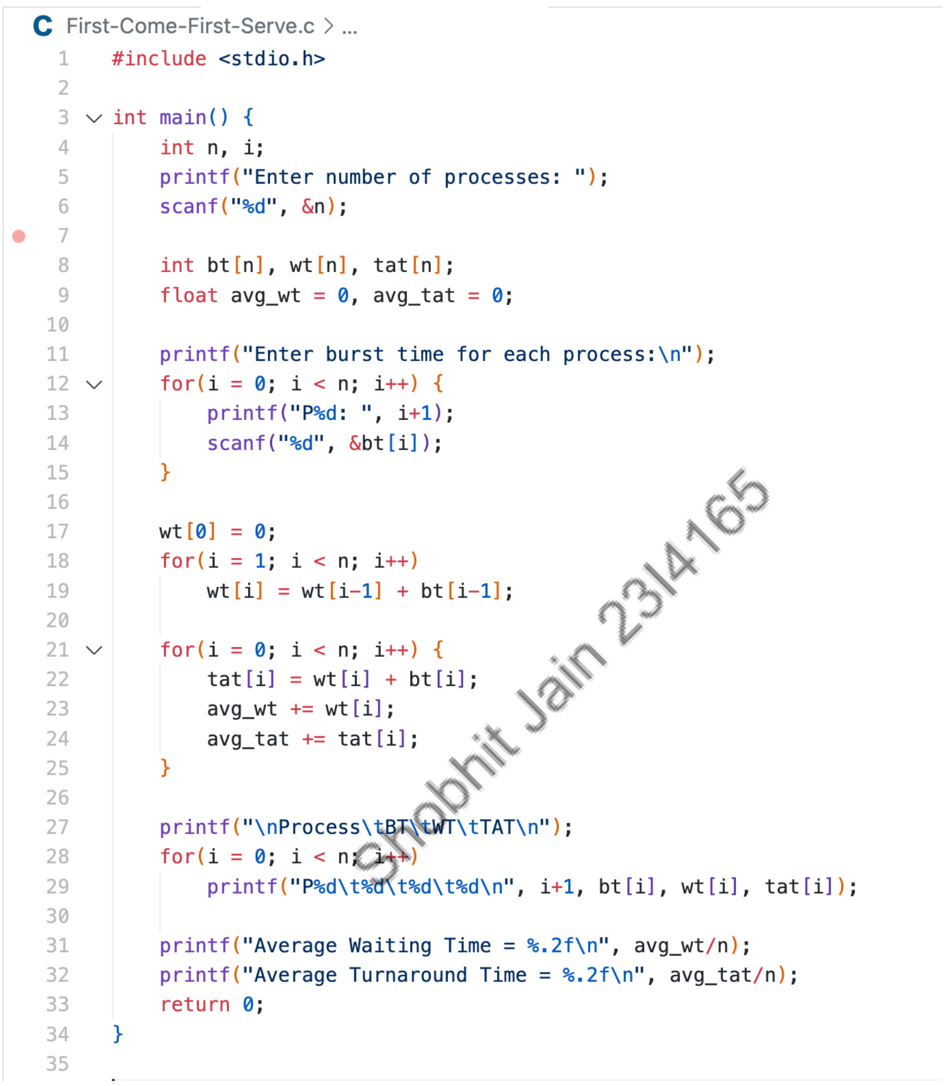
CPU Scheduling is the process of deciding which process will use the CPU next when multiple processes are ready to execute. It is a key function of the operating system that helps in allocating CPU time efficiently among all running processes.

It has many types:

1. **First-Come, First-Served (FCFS)**
   - Processes are executed in the order they arrive.
   - Simple but can lead to long waiting times.
2. **Shortest Job Next (SJN) / Shortest Job First (SJF)**
   - Executes the process with the shortest burst time next.
   - Reduces average waiting time but may cause starvation.
3. **Priority Scheduling**
   - Each process is assigned a priority; the CPU executes the process with the highest priority.
   - May lead to starvation of lower-priority processes.
4. **Round Robin (RR)**
   - Each process is assigned a fixed time slice (quantum) in a cyclic order.
   - Good for time-sharing systems; ensures fairness.
5. **Multilevel Queue Scheduling**
   - Processes are divided into different queues based on priority or type. Each queue has its own scheduling algorithm.
6. **Multilevel Feedback Queue Scheduling**
   - Similar to multilevel queue, but processes can move between queues based on their behavior and requirements.

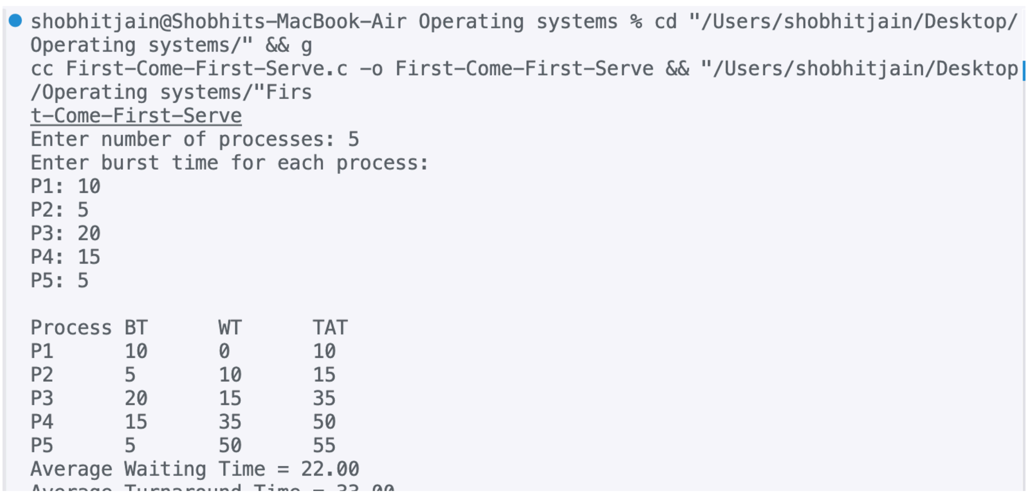
---

1. First Come First Serve (FCFS) : First Come First Serve (FCFS) is the simplest type of CPU scheduling algorithm. It executes processes in the exact order in which they arrive in the ready queue, similar to a queue at a ticket counter. The process that arrives first gets the CPU first. It is non- preemptive, meaning once a process starts executing, it runs till completion without interruption. FCFS is easy to implement but can lead to issues like convoy effect, where short jobs wait behind longer ones, increasing average waiting time.



First-Come-First-Serve.c > ...

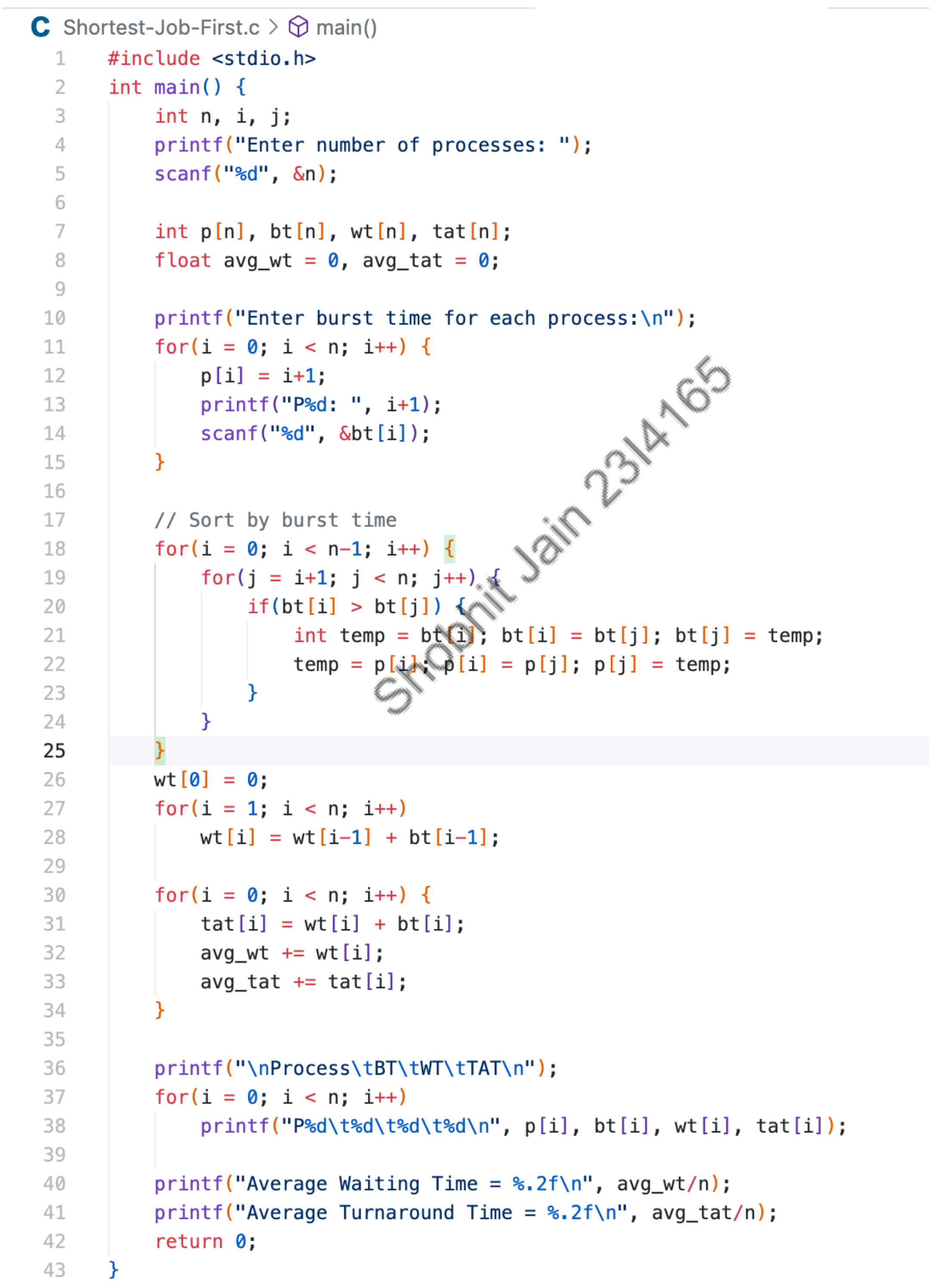
```c
#include <stdio.h>

int main() {
    int n, i;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    int bt[n], wt[n], tat[n];
    float avg_wt = 0, avg_tat = 0;

    printf("Enter burst time for each process:\n");
    for(i = 0; i < n; i++) {
        printf("P%d: ", i+1);
        scanf("%d", &bt[i]);
    }

    wt[0] = 0;
    for(i = 1; i < n; i++)
        wt[i] = wt[i-1] + bt[i-1];

    for(i = 0; i < n; i++) {
        tat[i] = wt[i] + bt[i];
        avg_wt += wt[i];
        avg_tat += tat[i];
    }

    printf("\nProcess\tBT\tWT\tTAT\n");
    for(i = 0; i < n; i++)
        printf("P%d\t%d\t%d\t%d\n", i+1, bt[i], wt[i], tat[i]);

    printf("Average Waiting Time = %.2f\n", avg_wt/n);
    printf("Average Turnaround Time = %.2f\n", avg_tat/n);
    return 0;
}
```



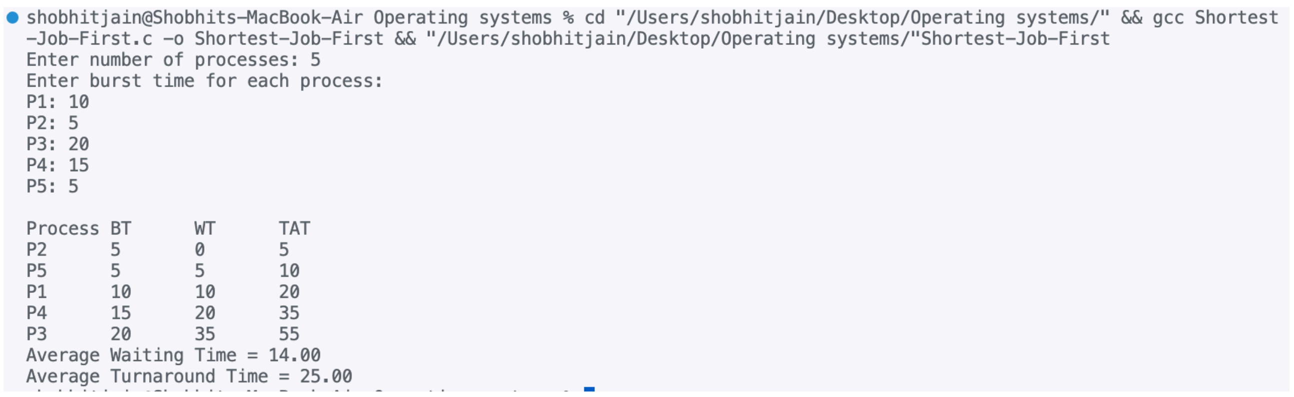
```
shobhitjain@Shobhits-MacBook-Air Operating systems % cd "/Users/shobhitjain/Desktop/
Operating systems/" && g
cc First-Come-First-Serve.c -o First-Come-First-Serve && "/Users/shobhitjain/Desktop
/Operating systems/"Firs
t-Come-First-Serve
Enter number of processes: 5
Enter burst time for each process:
P1: 10
P2: 5
P3: 20
P4: 15
P5: 5

Process BT      WT      TAT
P1      10      0       10
P2      5       10      15
P3      20      15      35
P4      15      35      50
P5      5       50      55
Average Waiting Time = 22.00
Average Turnaround Time = 33.00
```

---

2. Shortest Job First (SJF) : Shortest Job First (SJF) scheduling selects the process with the smallest burst time (execution time) from the ready queue. It is non-preemptive, meaning once the CPU is assigned, the process runs to completion. SJF is optimal in terms of minimizing average waiting time, but it requires prior knowledge of how long a process will take. If burst time predictions are accurate, this algorithm is very efficient. However, it can cause starvation for longer jobs if shorter ones keep arriving.



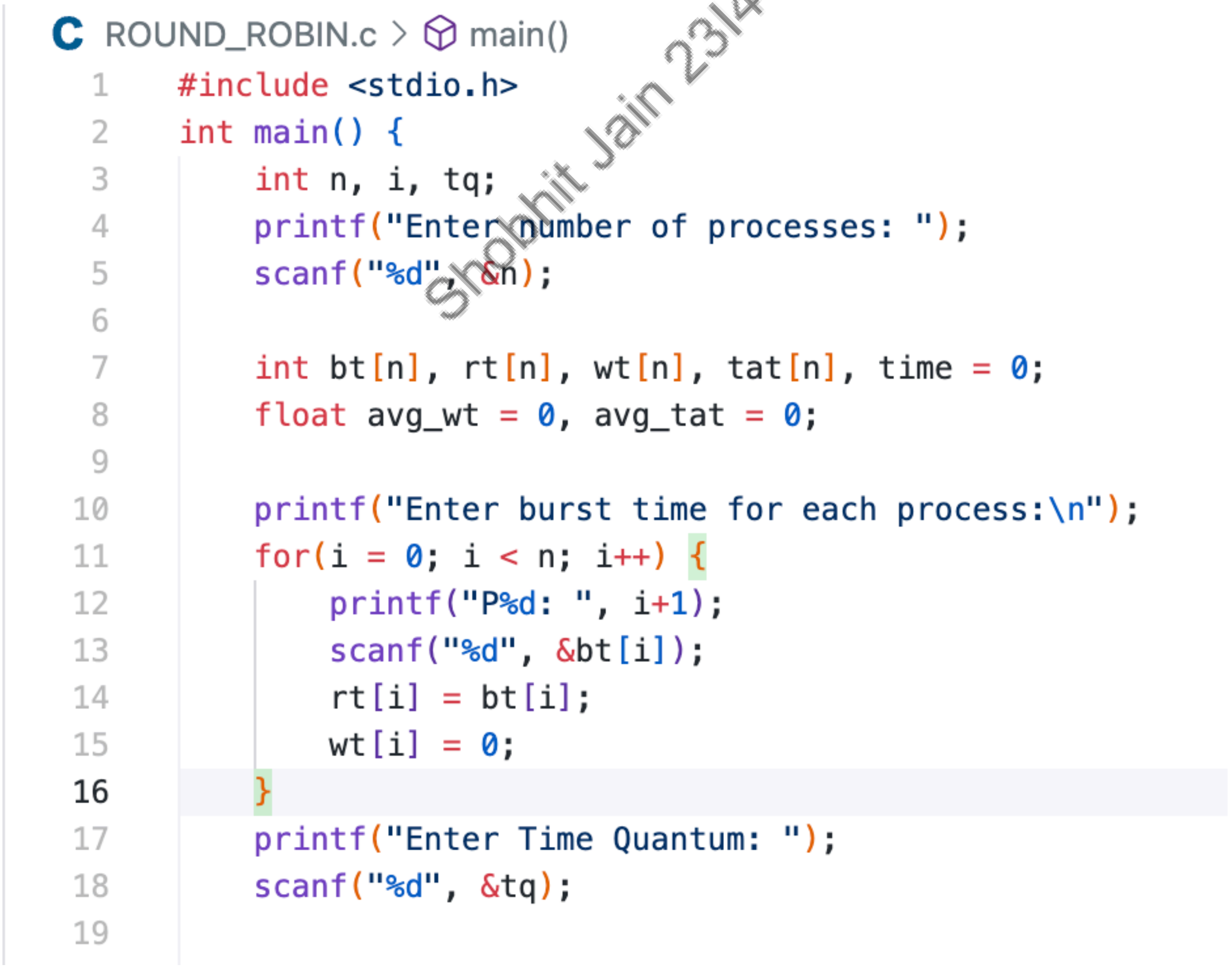
```c
#include <stdio.h>
int main() {
    int n, i, j;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    int p[n], bt[n], wt[n], tat[n];
    float avg_wt = 0, avg_tat = 0;

    printf("Enter burst time for each process:\n");
    for(i = 0; i < n; i++) {
        p[i] = i+1;
        printf("P%d: ", i+1);
        scanf("%d", &bt[i]);
    }

    // Sort by burst time
    for(i = 0; i < n-1; i++) {
        for(j = i+1; j < n; j++) {
            if(bt[i] > bt[j]) {
                int temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;
                temp = p[i]; p[i] = p[j]; p[j] = temp;
            }
        }
    }
    wt[0] = 0;
    for(i = 1; i < n; i++)
        wt[i] = wt[i-1] + bt[i-1];

    for(i = 0; i < n; i++) {
        tat[i] = wt[i] + bt[i];
        avg_wt += wt[i];
        avg_tat += tat[i];
    }

    printf("\nProcess\tBT\tWT\tTAT\n");
    for(i = 0; i < n; i++)
        printf("P%d\t%d\t%d\t%d\n", p[i], bt[i], wt[i], tat[i]);

    printf("Average Waiting Time = %.2f\n", avg_wt/n);
    printf("Average Turnaround Time = %.2f\n", avg_tat/n);
    return 0;
}
```



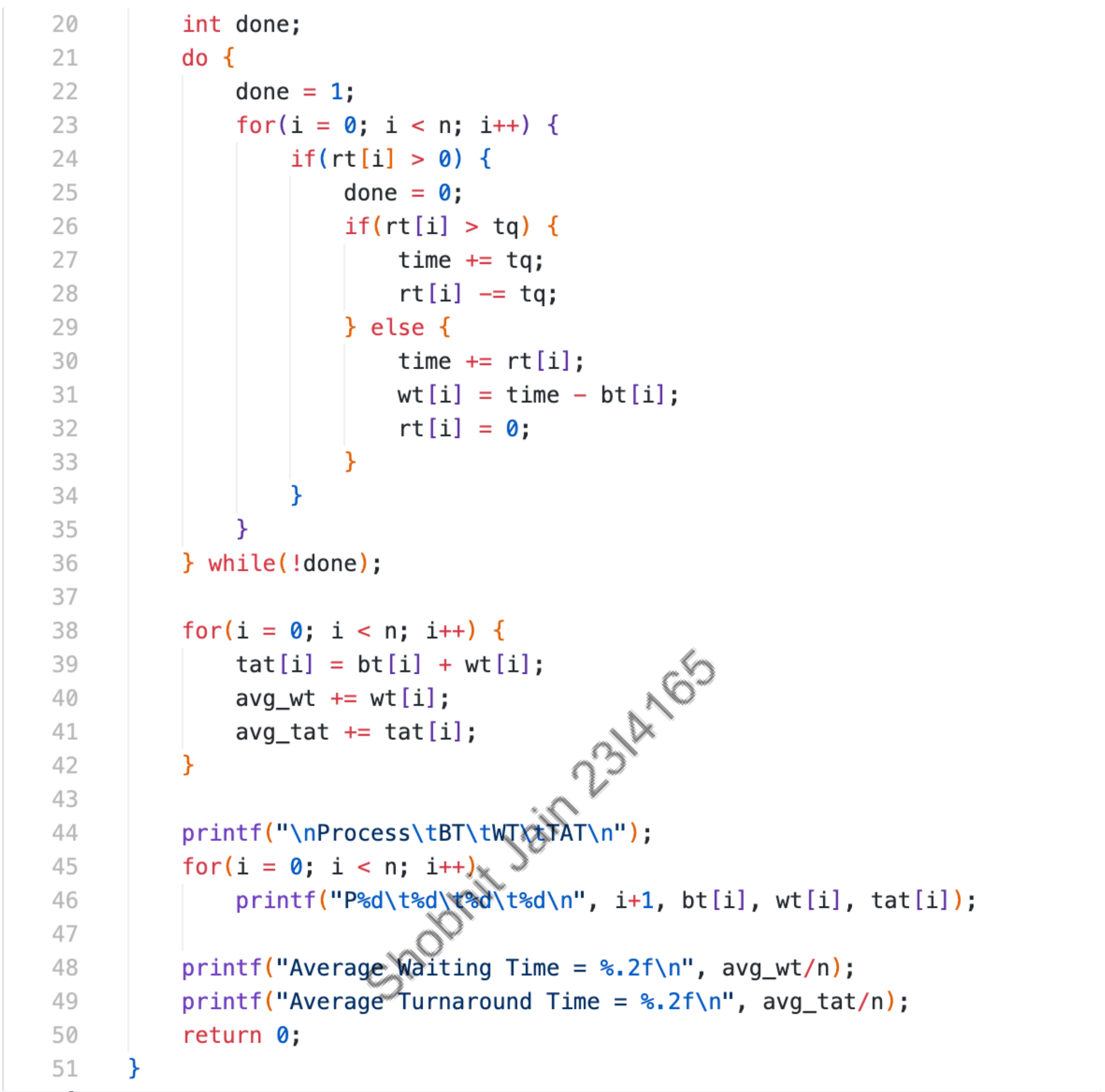
```
shobhitjain@Shobhits-MacBook-Air Operating systems % cd "/Users/shobhitjain/Desktop/Operating systems/" && gcc Shortest
-Job-First.c -o Shortest-Job-First && "/Users/shobhitjain/Desktop/Operating systems/"Shortest-Job-First
Enter number of processes: 5
Enter burst time for each process:
P1: 10
P2: 5
P3: 20
P4: 15
P5: 5

Process BT      WT      TAT
P2      5       0       5
P5      5       5       10
P1      10      10      20
P4      15      20      35
P3      20      35      55
Average Waiting Time = 14.00
Average Turnaround Time = 25.00
```

---

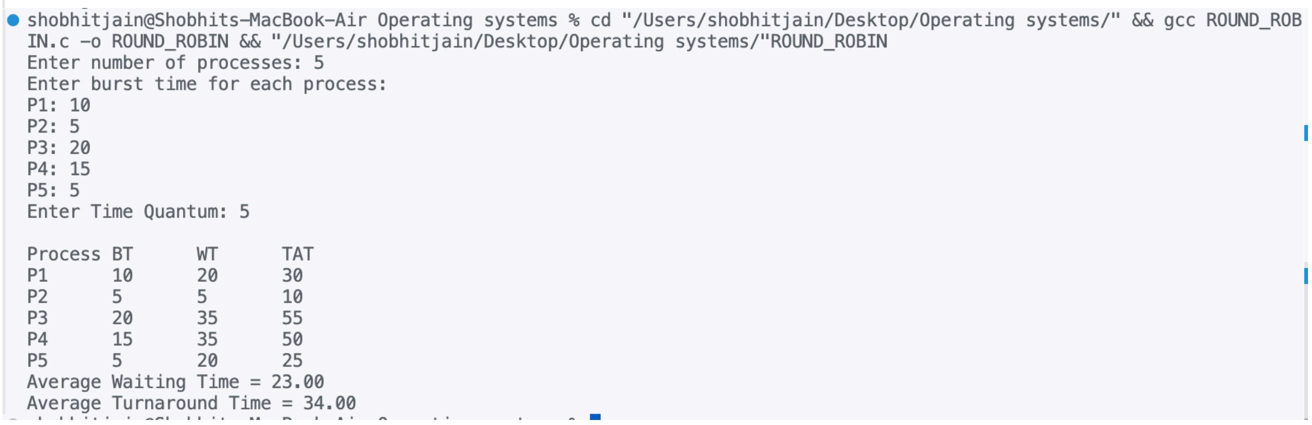
3. Round Robin Scheduling : Round Robin Scheduling is a preemptive scheduling algorithm designed for time-sharing systems. Each process is assigned a fixed time quantum (e.g., 4ms), and the CPU cycles through the processes in the ready queue, giving each one a turn. If a process doesn't finish in its time slice, it is moved to the end of the queue. This ensures fairness and prevents any process from monopolizing the CPU. However, if the time quantum is too small, it can lead to high context switching overhead.



```c
#include <stdio.h>
int main() {
    int n, i, tq;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    int bt[n], rt[n], wt[n], tat[n], time = 0;
    float avg_wt = 0, avg_tat = 0;

    printf("Enter burst time for each process:\n");
    for(i = 0; i < n; i++) {
        printf("P%d: ", i+1);
        scanf("%d", &bt[i]);
        rt[i] = bt[i];
        wt[i] = 0;
    }
    printf("Enter Time Quantum: ");
    scanf("%d", &tq);

```



```c
    int done;
    do {
        done = 1;
        for(i = 0; i < n; i++) {
            if(rt[i] > 0) {
                done = 0;
                if(rt[i] > tq) {
                    time += tq;
                    rt[i] -= tq;
                } else {
                    time += rt[i];
                    wt[i] = time - bt[i];
                    rt[i] = 0;
                }
            }
        }
    } while(!done);

    for(i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        avg_wt += wt[i];
        avg_tat += tat[i];
    }

    printf("\nProcess\tBT\tWT\tTAT\n");
    for(i = 0; i < n; i++)
        printf("P%d\t%d\t%d\t%d\n", i+1, bt[i], wt[i], tat[i]);

    printf("Average Waiting Time = %.2f\n", avg_wt/n);
    printf("Average Turnaround Time = %.2f\n", avg_tat/n);
    return 0;
}
```



```
shobhitjain@Shobhits-MacBook-Air Operating systems % cd "/Users/shobhitjain/Desktop/Operating systems/" && gcc ROUND_ROB
IN.c -o ROUND_ROBIN && "/Users/shobhitjain/Desktop/Operating systems/"ROUND_ROBIN
Enter number of processes: 5
Enter burst time for each process:
P1: 10
P2: 5
P3: 20
P4: 15
P5: 5
Enter Time Quantum: 5

Process BT      WT      TAT
P1      10      20      30
P2      5       5       10
P3      20      35      55
P4      15      35      50
P5      5       20      25
Average Waiting Time = 23.00
Average Turnaround Time = 34.00
```