ASSIGNMENT 6
17/10/23

NAME : SHRESTH SONKAR
REGNO : 20214272
GROUP : CS5D
TOPIC : OS LAB
CODE : CS-15203

```c
// Q1a : FCFS

#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
#include <unistd.h>

void findWT(int n, int bt[], int wt[]) {
    wt[0] = 0;
    for (int i = 1; i < n; i++)
        wt[i] = wt[i - 1] + bt[i - 1];
}

void findTAT(int n, int bt[], int wt[], int tat[]) {
    for (int i = 0; i < n; i++)
        tat[i] = bt[i] + wt[i];
}

void findAvg(int n, int bt[]) {
    int wt[n], tat[n];
    float total_wt = 0, total_tat = 0;
    findWT(n, bt, wt);
    findTAT(n, bt, wt, tat);

    for (int i = 0; i < n; i++) {
        total_wt += wt[i];
        total_tat += tat[i];
    }

    printf("Average Waiting Time: %.2f\n", (total_wt /
n));
    printf("Average Turnaround Time: %.2f\n",
(total_tat / n));
}

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

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

```
    }

    findAvg(n, bt);
    return 0;
}
```

```c
// Q1b : SJF

#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
#include <unistd.h>

int findSJ(int n, int bt[], int at[], int done[]) {
    int shortest = -1;
    int shortestTime = INT_MAX;

    for (int i = 0; i < n; i++) {
        if (!done[i] && at[i] <= 0) {
            if (bt[i] < shortestTime) {
                shortestTime = bt[i];
                shortest = i;
            }
        }
    }

    return shortest;
}

void findTAT(int n, int bt[], int wt[], int tat[]) {
    for (int i = 0; i < n; i++)
        tat[i] = bt[i] + wt[i];
}

void findWT(int n, int bt[], int wt[], int at[]) {
    int done[n];
    int currT = 0;
    int remT = n;

    for (int i = 0; i < n; i++)
        done[i] = 0;

    while (remT > 0) {
        int shortest = findSJ(n, bt, at, done);

        if (shortest == -1) {
            currT++;
            continue;
        }

        wt[shortest] = currT - at[shortest];
```

```
            done[shortest] = 1;
            currT += bt[shortest];
            remT--;
        }
    }

    void findAvg(int n, int bt[], int at[]) {
        int wt[n], tat[n];
        float total_wt = 0, total_tat = 0;
        findWT(n, bt, wt, at);
        findTAT(n, bt, wt, tat);

        for (int i = 0; i < n; i++) {
            total_wt += wt[i];
            total_tat += tat[i];
        }

        printf("Average Waiting Time: %.2f\n", (total_wt /
    n));
        printf("Average Turnaround Time: %.2f\n",
    (total_tat / n));
    }

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

        int burstTime[n];
        int arrivalTime[n];

        printf("Enter burst time and arrival time for each
    process:\n");
        for (int i = 0; i < n; i++) {
            printf("Process %d: ", i + 1);
            scanf("%d %d", &burstTime[i], &arrivalTime[i]);
        }

        findAvg(n, burstTime, arrivalTime);
        return 0;
    }
```

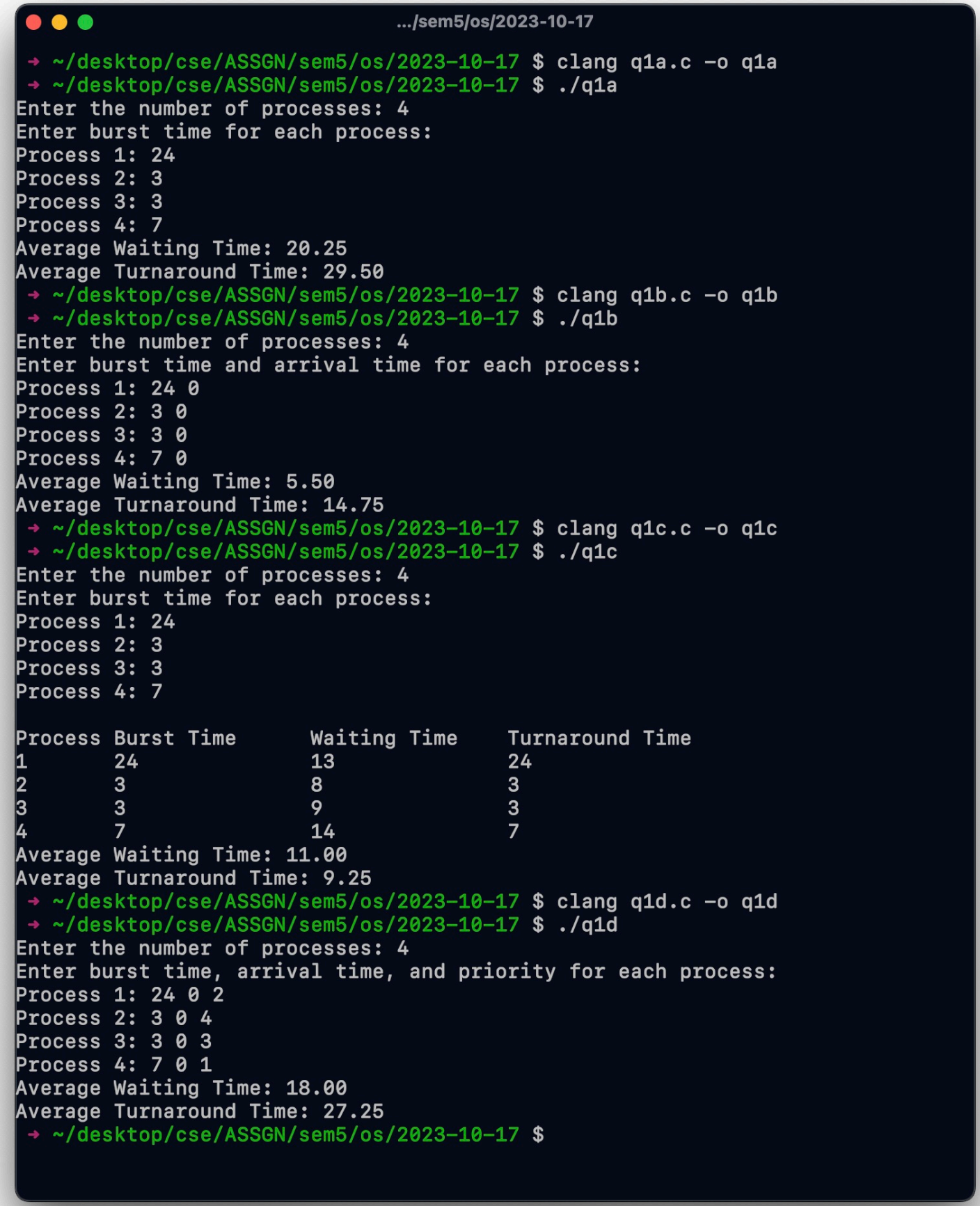
```c
// Q1c : Round Robin

#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
#include <unistd.h>

struct Process {
    int id;
    int bt;
    int rt;
};


void findTAT(struct Process processes[], int n, int
tat[]) {
    for (int i = 0; i < n; i++) {
        tat[i] = processes[i].bt;
    }
}

void findWT(struct Process processes[], int n, int
wt[]) {
    int currT = 0;
    int remT = n;

    while (remT > 0) {
        for (int i = 0; i < n; i++) {
            if (processes[i].rt > 0) {
                if (processes[i].rt <= 2) {
                    currT += processes[i].rt;
                    wt[i] = currT - processes[i].bt;
                    processes[i].rt = 0;
                    remT--;
                } else {
                    currT += 2;
                    processes[i].rt -= 2;
                }
            }
        }
    }
}

int main() {
    int n;
```

```c
    float total_wt = 0, total_tat = 0;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process processes[n];
    int tat[n];
    int wt[n];

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

    findWT(processes, n, wt);
    findTAT(processes, n, tat);

    printf("\nProcess\tBurst Time\tWaiting
Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", processes[i].id,
processes[i].bt, wt[i], tat[i]);
    }
    for (int i = 0; i < n; i++) {
        total_wt += wt[i];
        total_tat += tat[i];
    }

    printf("Average Waiting Time: %.2f\n", (total_wt /
n));
    printf("Average Turnaround Time: %.2f\n",
(total_tat / n));
    return 0;
}
```

```c
// Q1d : Priority

#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
#include <unistd.h>

int findHYPR(int n, int priority[], int at[], int
done[]) {
    int highest = -1;
    int highestPriority = INT_MAX;

    for (int i = 0; i < n; i++) {
        if (!done[i] && at[i] <= 0) {
            if (priority[i] < highestPriority) {
                highestPriority = priority[i];
                highest = i;
            }
        }
    }

    return highest;
}

void findTAT(int n, int bt[], int wt[], int tat[]) {
    for (int i = 0; i < n; i++)
        tat[i] = bt[i] + wt[i];
}

void findWT(int n, int bt[], int wt[], int at[], int
priority[]) {
    int done[n];
    int currentTime = 0;
    int rem = n;

    for (int i = 0; i < n; i++)
        done[i] = 0;

    while (rem > 0) {
        int highest = findHYPR(n, priority, at, done);
        if (highest == -1) {
            currentTime++;
            continue;
        }
```

```
        wt[highest] = currentTime - at[highest];
        done[highest] = 1;
        currentTime += bt[highest];
        rem--;
    }
}

void findAvg(int n, int bt[], int at[], int priority[])
{
    int wt[n], tat[n];
    float total_wt = 0, total_tat = 0;
    findWT(n, bt, wt, at, priority);
    findTAT(n, bt, wt, tat);


    for (int i = 0; i < n; i++) {
        total_wt += wt[i];
        total_tat += tat[i];
    }

    printf("Average Waiting Time: %.2f\n", (total_wt /
n));
    printf("Average Turnaround Time: %.2f\n",
(total_tat / n));
}

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

    int bt[n];
    int at[n];
    int priority[n];

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

    findAvg(n, bt, at, priority);
    return 0;
```

```
}
```


.../sem5/os/2023-10-17
→ ~/desktop/cse/ASSGN/sem5/os/2023-10-17 $ clang q1a.c -o q1a
→ ~/desktop/cse/ASSGN/sem5/os/2023-10-17 $ ./q1a
Enter the number of processes: 4
Enter burst time for each process:
Process 1: 24
Process 2: 3
Process 3: 3
Process 4: 7
Average Waiting Time: 20.25
Average Turnaround Time: 29.50
→ ~/desktop/cse/ASSGN/sem5/os/2023-10-17 $ clang q1b.c -o q1b
→ ~/desktop/cse/ASSGN/sem5/os/2023-10-17 $ ./q1b
Enter the number of processes: 4
Enter burst time and arrival time for each process:
Process 1: 24 0
Process 2: 3 0
Process 3: 3 0
Process 4: 7 0
Average Waiting Time: 5.50
Average Turnaround Time: 14.75
→ ~/desktop/cse/ASSGN/sem5/os/2023-10-17 $ clang q1c.c -o q1c
→ ~/desktop/cse/ASSGN/sem5/os/2023-10-17 $ ./q1c
Enter the number of processes: 4
Enter burst time for each process:
Process 1: 24
Process 2: 3
Process 3: 3
Process 4: 7
Process Burst Time Waiting Time Turnaround Time
1 24 13 24
2 3 8 3
3 3 9 3
4 7 14 7
Average Waiting Time: 11.00
Average Turnaround Time: 9.25
→ ~/desktop/cse/ASSGN/sem5/os/2023-10-17 $ clang q1d.c -o q1d
→ ~/desktop/cse/ASSGN/sem5/os/2023-10-17 $ ./q1d
Enter the number of processes: 4
Enter burst time, arrival time, and priority for each process:
Process 1: 24 0 2
Process 2: 3 0 4
Process 3: 3 0 3
Process 4: 7 0 1
Average Waiting Time: 18.00
Average Turnaround Time: 27.25
→ ~/desktop/cse/ASSGN/sem5/os/2023-10-17 $

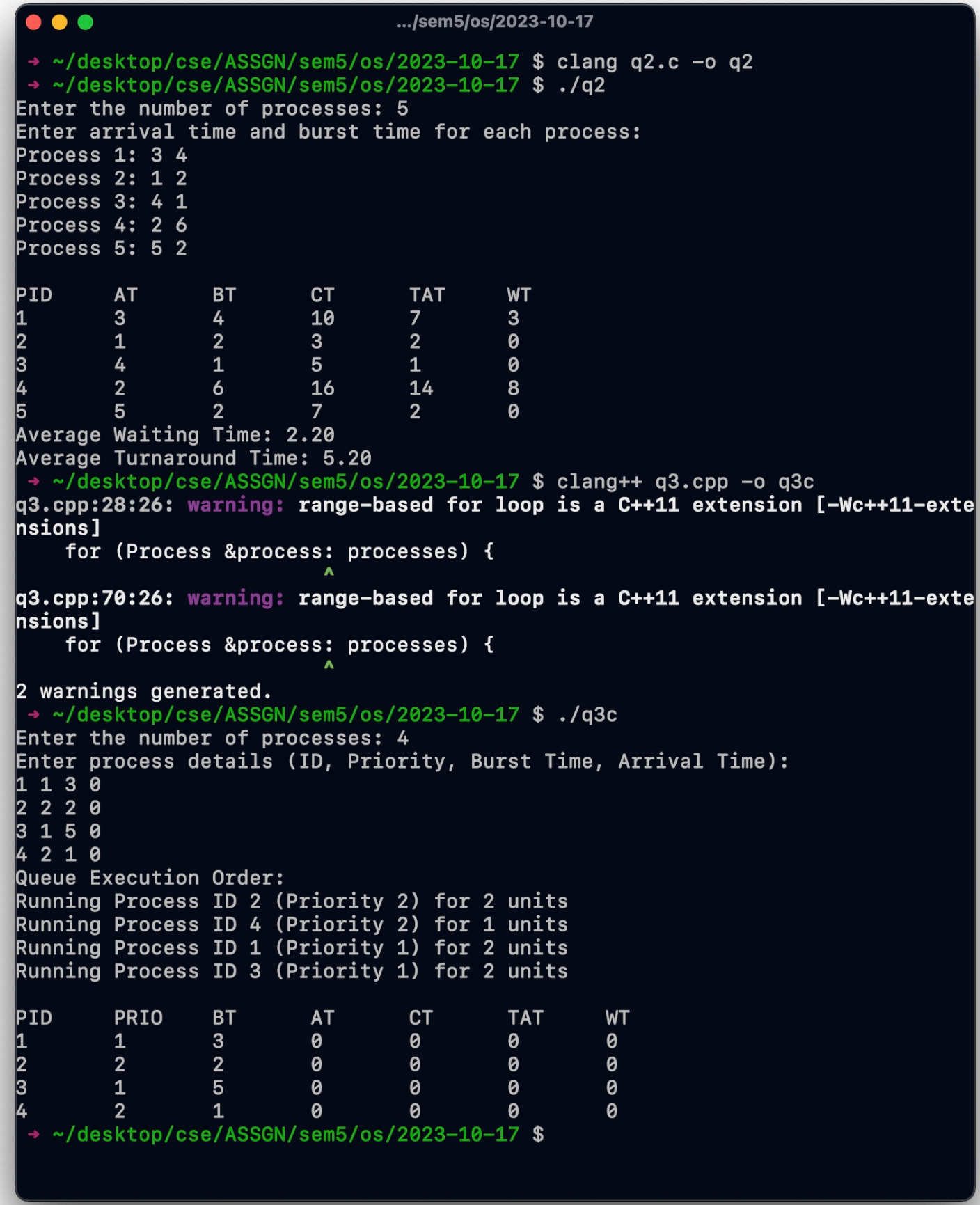
```c
// Q2 : SRTF

#include <stdio.h>
#include <limits.h>
#include <stdlib.h>

struct Process {
    int id;
    int at;
    int bt;
    int remT;
    int ct;
};

int findSJ(struct Process *processes, int n, int currT)
{
    int shortestJob = -1;
    int shortestTime = INT_MAX;

    for (int i = 0; i < n; i++) {
        if (processes[i].at <= currT &&
processes[i].remT < shortestTime &&
            processes[i].remT > 0) {
            shortestTime = processes[i].remT;
            shortestJob = i;
        }
    }

    return shortestJob;
}

void findTAT(struct Process *processes, int n, int
tat[]) {
    for (int i = 0; i < n; i++) {
        tat[i] = processes[i].ct - processes[i].at;
    }
}

void findWT(struct Process *processes, int n, int wt[])
{
    for (int i = 0; i < n; i++) {
        wt[i] = processes[i].ct - processes[i].at -
processes[i].bt;
    }
}
```

```c
int main() {
    int n;
    float total_wt = 0, total_tat = 0;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process *processes = (struct Process *)
malloc(n * sizeof(struct Process));
    int *tat = (int *) malloc(n * sizeof(int));
    int *wt = (int *) malloc(n * sizeof(int));

    printf("Enter arrival time and burst time for each
process:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d: ", i + 1);
        scanf("%d %d", &processes[i].at,
&processes[i].bt);
        processes[i].id = i + 1;
        processes[i].remT = processes[i].bt;
    }

    int currT = 0;
    int completed = 0;

    while (completed < n) {
        int shortestJob = findSJ(processes, n, currT);

        if (shortestJob == -1) {
            currT++;
        } else {
            processes[shortestJob].remT--;

            if (processes[shortestJob].remT == 0) {
                completed++;
                processes[shortestJob].ct = currT + 1;
            }

            currT++;
        }
    }

    findTAT(processes, n, tat);
    findWT(processes, n, wt);
```

```c
    printf("\nPID\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\t%d\n",
processes[i].id, processes[i].at, processes[i].bt,
               processes[i].ct, tat[i], wt[i]);
    }
    for (int i = 0; i < n; i++) {
        total_wt += wt[i];
        total_tat += tat[i];
    }

    printf("Average Waiting Time: %.2f\n", (total_wt /
n));
    printf("Average Turnaround Time: %.2f\n",
(total_tat / n));
    free(processes);
    free(tat);
    free(wt);
    return 0;
}
```

```cpp
// Q3 : MLVL Q

#include <iostream>
#include <vector>
#include <queue>

using namespace std;

struct Process {
    int id;
    int pr;
    int bt;
    int at;
    int ct;
    int wt;
    int tat;
};

struct Comparepr {
    bool operator()(const Process &p1, const Process
&p2) {
        return p1.pr > p2.pr;
    }
};

void multiLevelQueueScheduling(vector <Process>
&processes) {
    vector < priority_queue < Process, vector < Process
>, Comparepr > > queues(3);

    int currT = 0;

    for (Process &process: processes) {
        int pr = process.pr - 1;
        queues[pr].push(process);
    }

    cout << "Queue Execution Order:" << endl;

    for (int i = 2; i >= 0; i--) {
        while (!queues[i].empty()) {
            Process process = queues[i].top();
            queues[i].pop();

            int execT = min(process.bt, 2);
```

```
            process.bt -= execT;
            currT += execT;

            process.ct = currT;
            process.tat = process.ct - process.at;
            process.wt = process.tat - process.bt;

            cout << "Running Process ID " << process.id
<< " (Priority " << process.pr << ") for " << execT
                 << " units" << endl;
        }
    }
}

int main() {
    int n;

    cout << "Enter the number of processes: ";
    cin >> n;

    vector <Process> processes(n);

    cout << "Enter process details (ID, Priority, Burst
Time, Arrival Time):" << endl;
    for (int i = 0; i < n; i++) {
        cin >> processes[i].id >> processes[i].pr >>
processes[i].bt >> processes[i].at;
    }

    multiLevelQueueScheduling(processes);

    cout << "\nPID\tPRIO\tBT\tAT\tCT\tTAT\tWT" << endl;
    for (Process &process: processes) {
        cout << process.id << "\t" << process.pr <<
"\t" << process.bt << "\t" << process.at << "\t" <<
process.ct
             << "\t" << process.tat << "\t" <<
process.wt << endl;
    }

    return 0;
}
```


.../sem5/os/2023-10-17
→ ~/desktop/cse/ASSGN/sem5/os/2023-10-17 $ clang q2.c -o q2
→ ~/desktop/cse/ASSGN/sem5/os/2023-10-17 $ ./q2
Enter the number of processes: 5
Enter arrival time and burst time for each process:
Process 1: 3 4
Process 2: 1 2
Process 3: 4 1
Process 4: 2 6
Process 5: 5 2
PID AT BT CT TAT WT
1 3 4 10 7 3
2 1 2 3 2 0
3 4 1 5 1 0
4 2 6 16 14 8
5 5 2 7 2 0
Average Waiting Time: 2.20
Average Turnaround Time: 5.20
→ ~/desktop/cse/ASSGN/sem5/os/2023-10-17 $ clang++ q3.cpp -o q3c
q3.cpp:28:26: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
for (Process &process: processes) {
^
q3.cpp:70:26: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
for (Process &process: processes) {
^
2 warnings generated.
→ ~/desktop/cse/ASSGN/sem5/os/2023-10-17 $ ./q3c
Enter the number of processes: 4
Enter process details (ID, Priority, Burst Time, Arrival Time):
1 1 3 0
2 2 2 0
3 1 5 0
4 2 1 0
Queue Execution Order:
Running Process ID 2 (Priority 2) for 2 units
Running Process ID 4 (Priority 2) for 1 units
Running Process ID 1 (Priority 1) for 2 units
Running Process ID 3 (Priority 1) for 2 units
PID PRIO BT AT CT TAT WT
1 1 3 0 0 0 0
2 2 2 0 0 0 0
3 1 5 0 0 0 0
4 2 1 0 0 0 0
→ ~/desktop/cse/ASSGN/sem5/os/2023-10-17 $