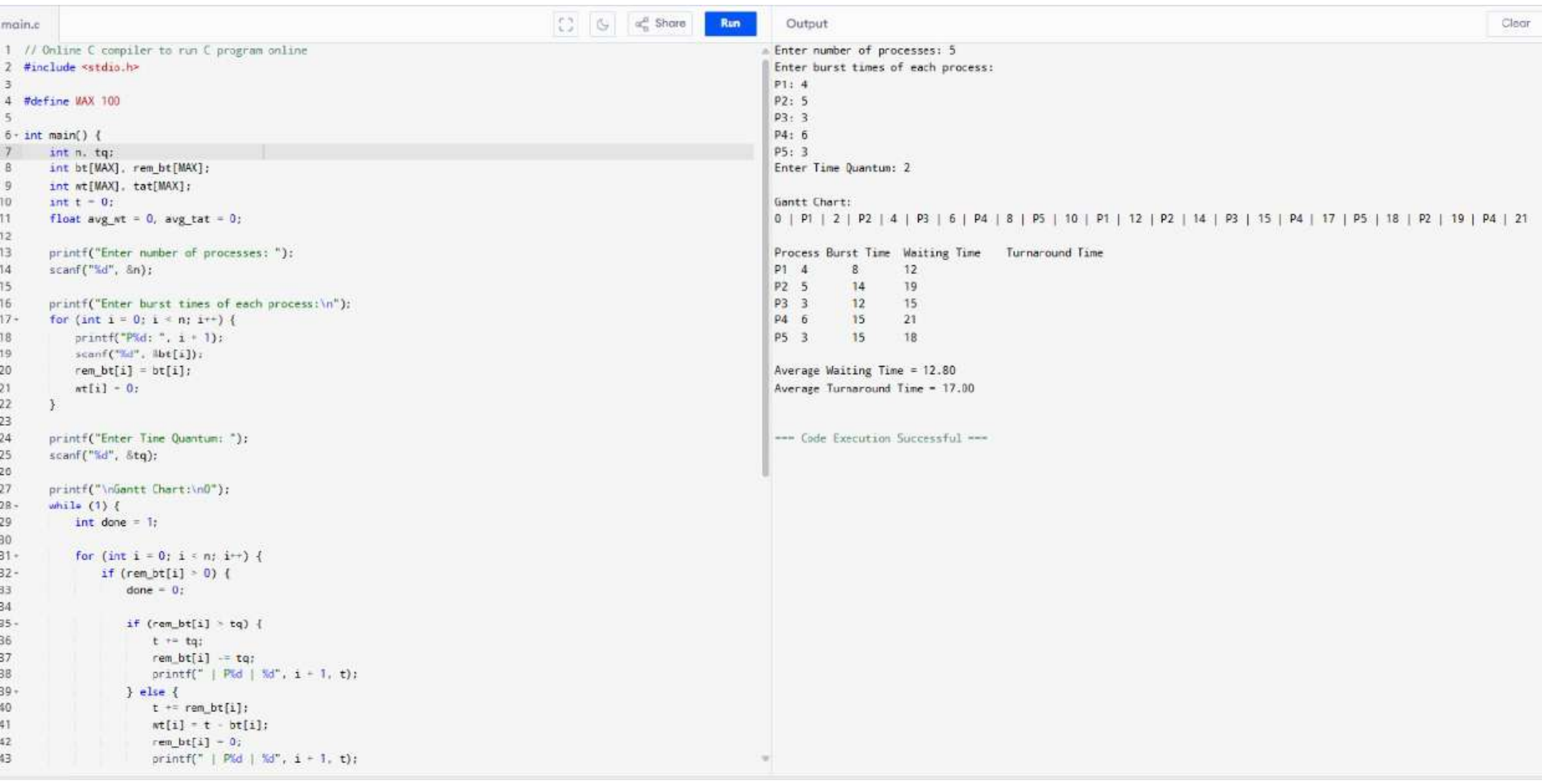
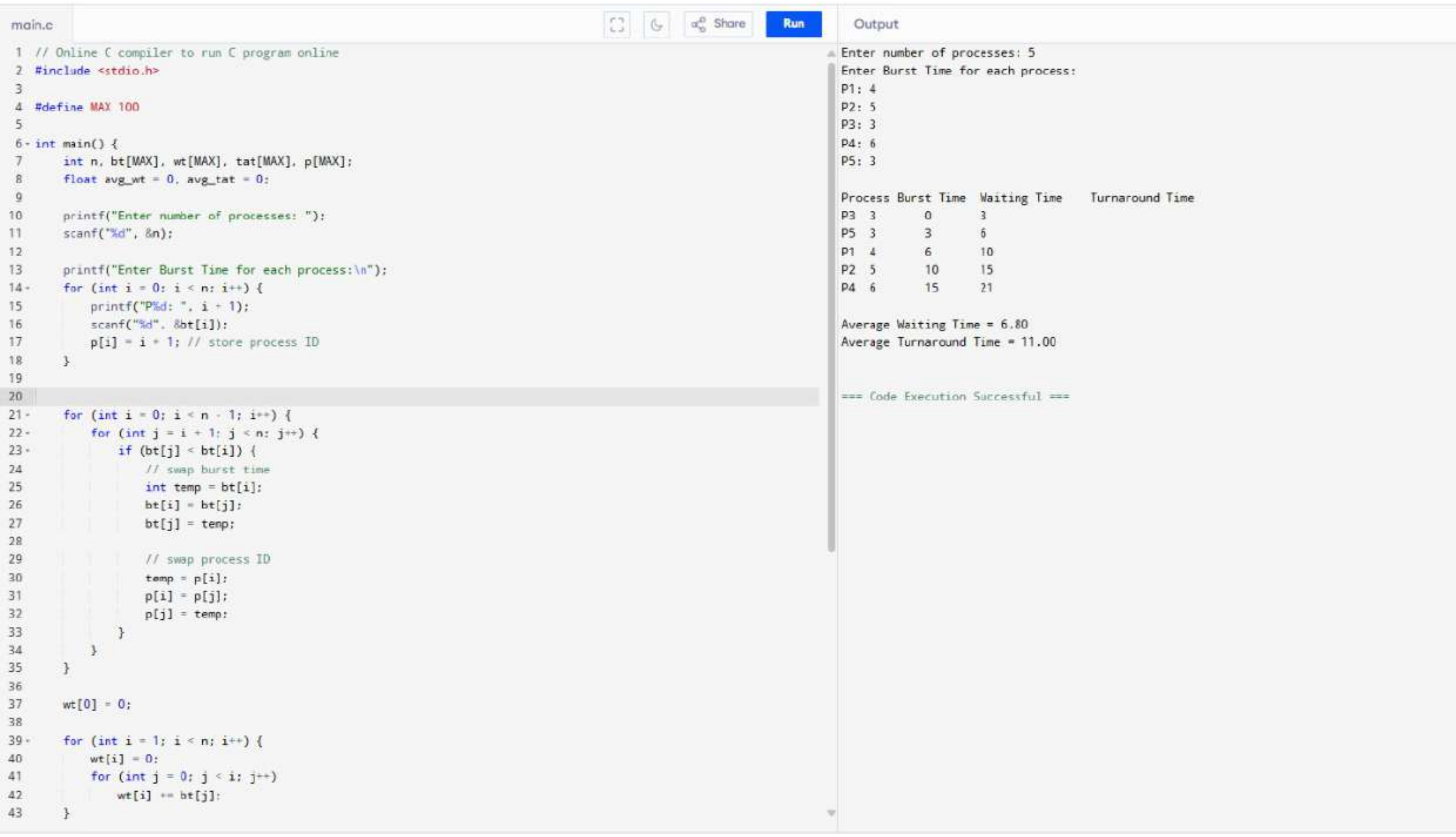


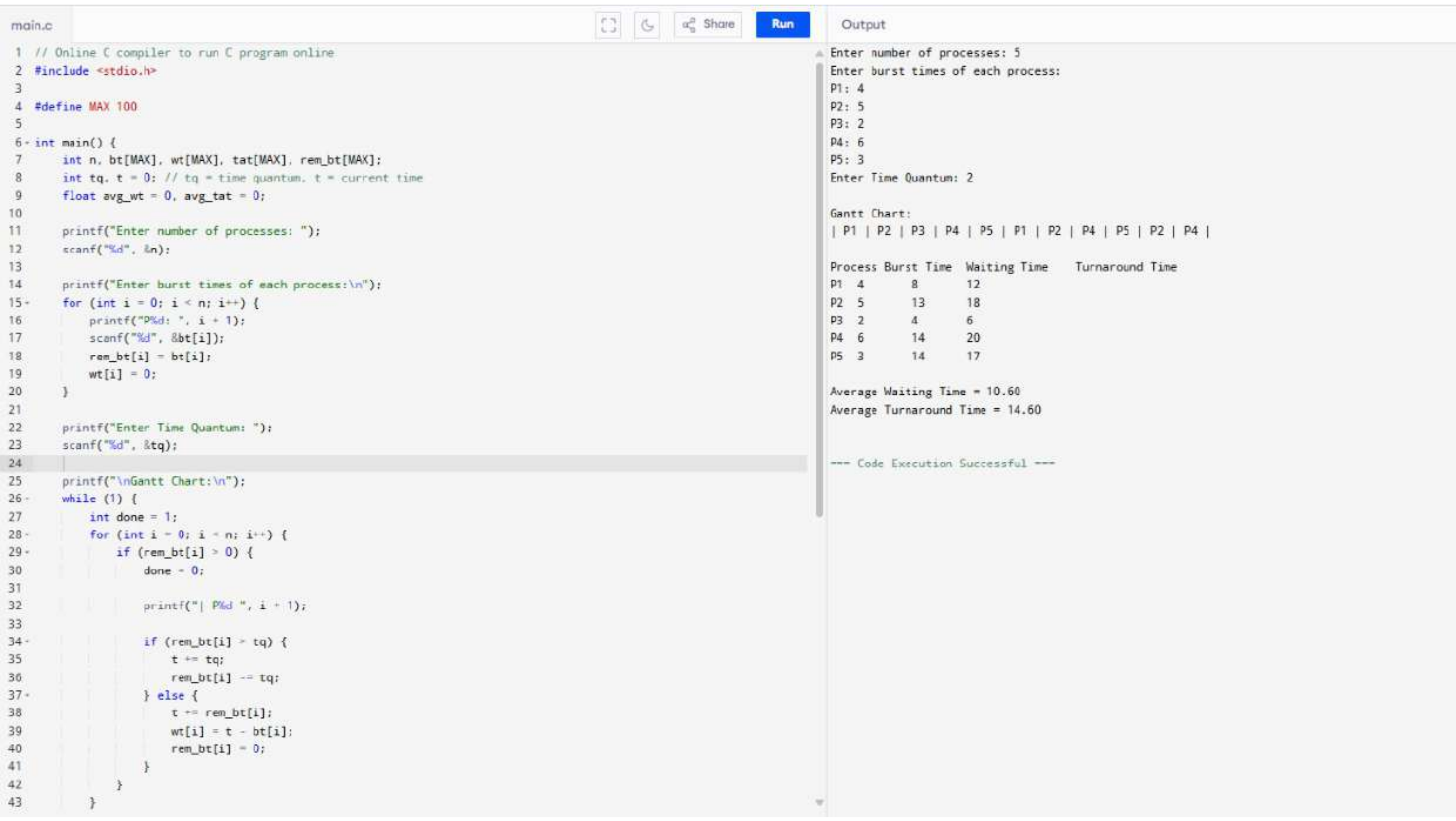
```c
// Online C compiler to run C program online
#include <stdio.h>

#define MAX 100

int main() {
    int n, tq;
    int bt[MAX], rem_bt[MAX];
    int wt[MAX], tat[MAX];
    int t = 0;
    float avg_wt = 0, avg_tat = 0;

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

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

    printf("Enter Time Quantum: ");
    scanf("%d", &tq);

    printf("\nGantt Chart:\n0");
    while (1) {
        int done = 1;

        for (int i = 0; i < n; i++) {
            if (rem_bt[i] > 0) {
                done = 0;

                if (rem_bt[i] > tq) {
                    t += tq;
                    rem_bt[i] -= tq;
                    printf(" | P%d | %d", i + 1, t);
                } else {
                    t += rem_bt[i];
                    wt[i] = t - bt[i];
                    rem_bt[i] = 0;
                    printf(" | P%d | %d", i + 1, t);
```

```
Enter number of processes: 5
Enter burst times of each process:
P1: 4
P2: 5
P3: 3
P4: 6
P5: 3
Enter Time Quantum: 2

Gantt Chart:
0 | P1 | 2 | P2 | 4 | P3 | 6 | P4 | 8 | P5 | 10 | P1 | 12 | P2 | 14 | P3 | 15 | P4 | 17 | P5 | 18 | P2 | 19 | P4 | 21

Process Burst Time  Waiting Time    Turnaround Time
P1  4       8       12
P2  5       14      19
P3  3       12      15
P4  6       15      21
P5  3       15      18

Average Waiting Time = 12.80
Average Turnaround Time = 17.00


=== Code Execution Successful ===
```



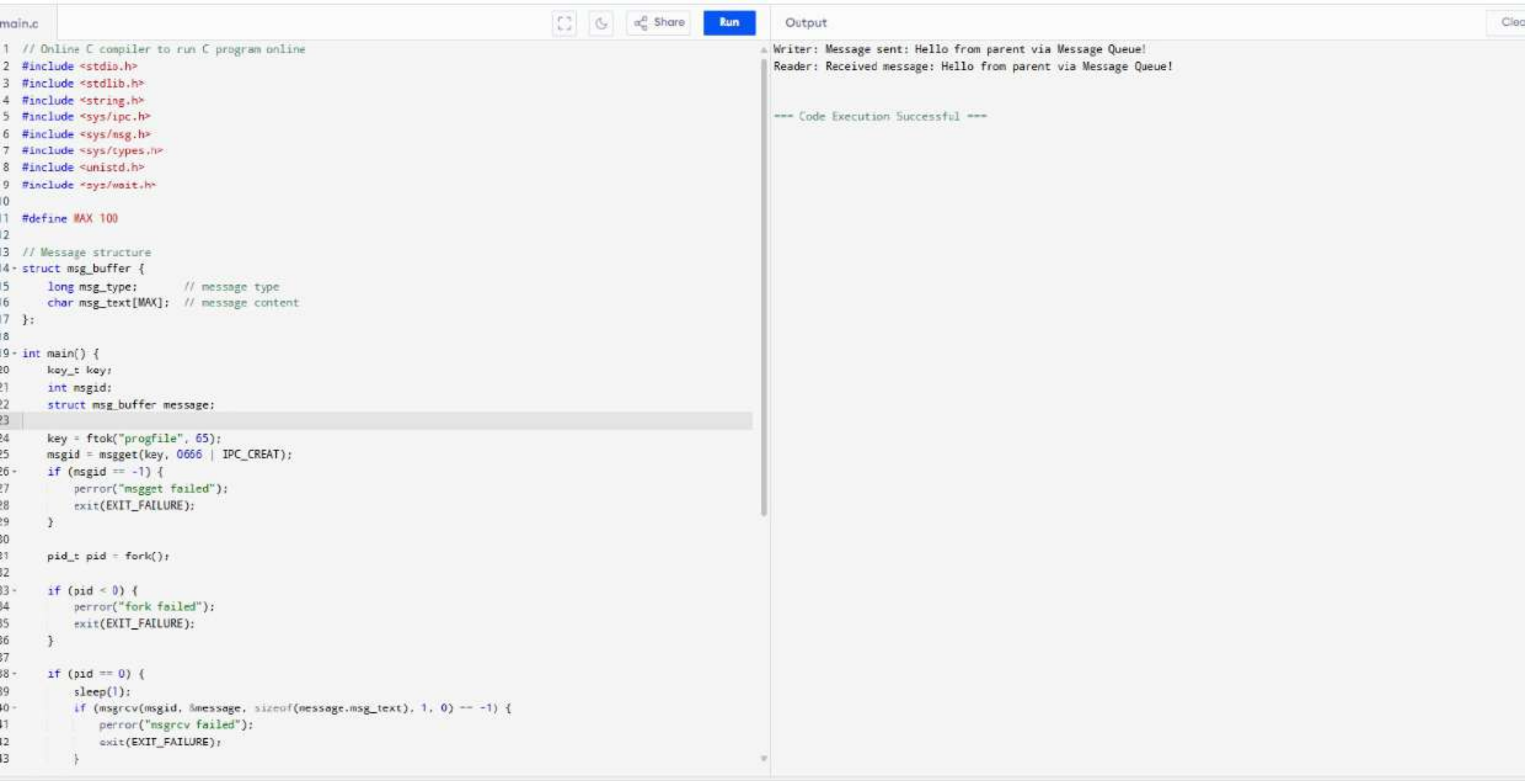
```c
// Online C compiler to run C program online
#include <stdio.h>

#define MAX 100

int main() {
    int n, bt[MAX], wt[MAX], tat[MAX], p[MAX];
    float avg_wt = 0, avg_tat = 0;

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

    printf("Enter Burst Time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("P%d: ", i + 1);
        scanf("%d", &bt[i]);
        p[i] = i + 1; // store process ID
    }


    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (bt[j] < bt[i]) {
                // swap burst time
                int temp = bt[i];
                bt[i] = bt[j];
                bt[j] = temp;

                // swap process ID
                temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }

    wt[0] = 0;

    for (int i = 1; i < n; i++) {
        wt[i] = 0;
        for (int j = 0; j < i; j++)
            wt[i] += bt[j];
    }
```

```
Enter number of processes: 5
Enter Burst Time for each process:
P1: 4
P2: 5
P3: 3
P4: 6
P5: 3

Process Burst Time  Waiting Time    Turnaround Time
P3  3       0       3
P5  3       3       6
P1  4       6       10
P2  5       10      15
P4  6       15      21

Average Waiting Time = 6.80
Average Turnaround Time = 11.00


=== Code Execution Successful ===
```



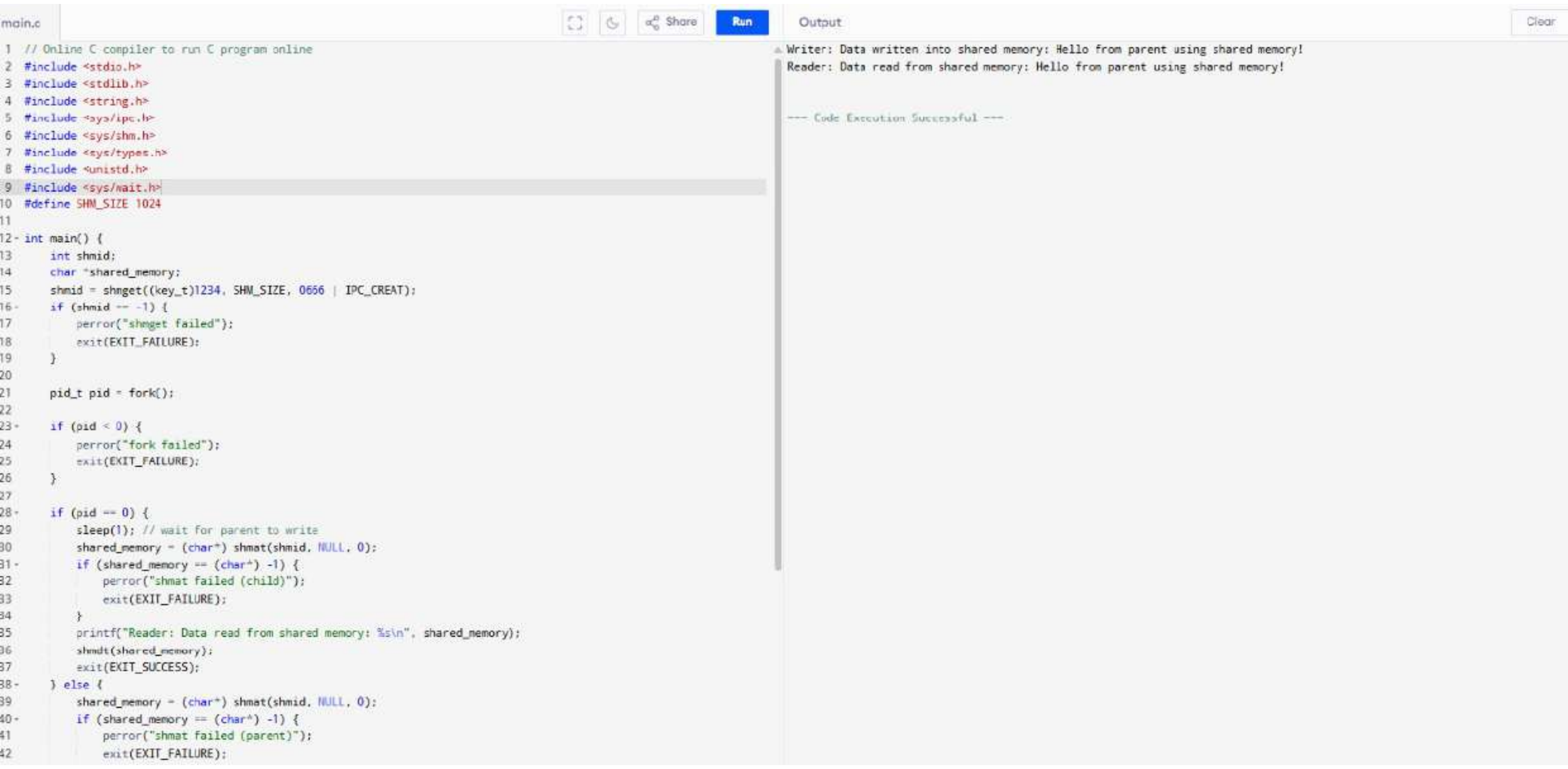
```c
// Online C compiler to run C program online
#include <stdio.h>

#define MAX 100

int main() {
    int n, bt[MAX], wt[MAX], tat[MAX], rem_bt[MAX];
    int tq, t = 0; // tq = time quantum, t = current time
    float avg_wt = 0, avg_tat = 0;

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

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

    printf("Enter Time Quantum: ");
    scanf("%d", &tq);

    printf("\nGantt Chart:\n");
    while (1) {
        int done = 1;
        for (int i = 0; i < n; i++) {
            if (rem_bt[i] > 0) {
                done = 0;

                printf("| P%d ", i + 1);

                if (rem_bt[i] > tq) {
                    t += tq;
                    rem_bt[i] -= tq;
                } else {
                    t += rem_bt[i];
                    wt[i] = t - bt[i];
                    rem_bt[i] = 0;
                }
            }
        }
```

```
Enter number of processes: 5
Enter burst times of each process:
P1: 4
P2: 5
P3: 2
P4: 6
P5: 3
Enter Time Quantum: 2

Gantt Chart:
| P1 | P2 | P3 | P4 | P5 | P1 | P2 | P4 | P5 | P2 | P4 |

Process Burst Time  Waiting Time    Turnaround Time
P1   4        8         12
P2   5        13        18
P3   2        4         6
P4   6        14        20
P5   3        14        17

Average Waiting Time = 10.60
Average Turnaround Time = 14.60


--- Code Execution Successful ---
```



```c
// Online C compiler to run C program online
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>

#define MAX 100

// Message structure
struct msg_buffer {
    long msg_type;       // message type
    char msg_text[MAX];  // message content
};

int main() {
    key_t key;
    int msgid;
    struct msg_buffer message;

    key = ftok("progfile", 65);
    msgid = msgget(key, 0666 | IPC_CREAT);
    if (msgid == -1) {
        perror("msgget failed");
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();

    if (pid < 0) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        sleep(1);
        if (msgrcv(msgid, &message, sizeof(message.msg_text), 1, 0) == -1) {
            perror("msgrcv failed");
            exit(EXIT_FAILURE);
        }
```

```
Writer: Message sent: Hello from parent via Message Queue!
Reader: Received message: Hello from parent via Message Queue!


=== Code Execution Successful ===
```



```c
// Online C compiler to run C program online
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#define SHM_SIZE 1024

int main() {
    int shmid;
    char *shared_memory;
    shmid = shmget((key_t)1234, SHM_SIZE, 0666 | IPC_CREAT);
    if (shmid == -1) {
        perror("shmget failed");
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();

    if (pid < 0) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        sleep(1); // wait for parent to write
        shared_memory = (char*) shmat(shmid, NULL, 0);
        if (shared_memory == (char*) -1) {
            perror("shmat failed (child)");
            exit(EXIT_FAILURE);
        }
        printf("Reader: Data read from shared memory: %s\n", shared_memory);
        shmdt(shared_memory);
        exit(EXIT_SUCCESS);
    } else {
        shared_memory = (char*) shmat(shmid, NULL, 0);
        if (shared_memory == (char*) -1) {
            perror("shmat failed (parent)");
            exit(EXIT_FAILURE);
```

```
Writer: Data written into shared memory: Hello from parent using shared memory!
Reader: Data read from shared memory: Hello from parent using shared memory!


--- Code Execution Successful ---
```