main.c

Run

Output

Clear

```c
#include <stdio.h>

void fcfs(int processes[], int n, int burst_time[]) {
    int waiting_time[n], turnaround_time[n];
    waiting_time[0] = 0;

    for (int i = 1; i < n; i++) {
        waiting_time[i] = waiting_time[i - 1] + burst_time[i - 1];
    }

    for (int i = 0; i < n; i++) {
        turnaround_time[i] = waiting_time[i] + burst_time[i];
    }

    float avg_waiting_time = 0, avg_turnaround_time = 0;
    for (int i = 0; i < n; i++) {
        avg_waiting_time += waiting_time[i];
        avg_turnaround_time += turnaround_time[i];
    }

```

```
/tmp/Qs8FW59sKW.o
Enter the number of processes: 5
Enter burst time for each process:
Process 1: 7
Process 2: 8
Process 3: 4
Process 4: 2
Process 5: 6
Process Burst Time  Waiting Time    Turnaround Time
1   7       0       7
2   8       7       15
3   4       15      19
4   2       19      21
5   6       21      27
Average Waiting Time: 12.40
Average Turnaround Time: 17.80
```

```c
#include <stdio.h>

void sjn(int n, int bt[]) {
    int wt[n], tat[n], total_wt = 0, total_tat = 0;

    // Initialize an array to keep track of whether a process is
        completed
    int completed[n];
    for (int i = 0; i < n; i++) {
        completed[i] = 0;
    }

    int time = 0, min_bt_index;

    while (1) {
        min_bt_index = -1;

        // Find the process with the smallest remaining burst time
        for (int i = 0; i < n; i++) {
            if (!completed[i] && (min_bt_index == -1 || bt[i] <
```

```
/tmp/jh3lCpKBh1.o
Enter number of processes: 2
Enter burst times for each process:
Process 1: 9
Process 2: 7
Process Burst Time  Waiting Time    Turnaround Time
1   9       7       16
2   7       0       7
Average Waiting Time: 3.50
Average Turnaround Time: 11.50
```

```c
#include <stdio.h>

void rr(int n, int bt[], int quantum) {
    int rem_bt[n];
    for (int i = 0; i < n; i++) {
        rem_bt[i] = bt[i];
    }

    int wt[n], tat[n], total_wt = 0, total_tat = 0;
    int time = 0;

    while (1) {
        int done = 1;

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

                if (rem_bt[i] > quantum) {
                    time += quantum;
```

```
/tmp/jh3lCpKBh1.o
Enter number of processes: 5
Enter burst times for each process:
Process 1: 6
Process 2: 2
Process 3: 1
Process 4: 4
Process 5: 8
Enter time quantum: 12
Process Burst Time  Waiting Time    Turnaround Time
1   6       0       6
2   2       6       8
3   1       8       9
4   4       9       13
5   8       13      21
Average Waiting Time: 7.20
Average Turnaround Time: 11.40
```

```c
#include <stdio.h>
#include <unistd.h>

int main() {
    int pipefd[2]; // Pipe file descriptors
    char buffer[20];
    pipe(pipefd); // Create a pipe

    if (fork() == 0) { // Child process
        close(pipefd[0]); // Close read end
        write(pipefd[1], "Hello, saran!", 14);
        close(pipefd[1]);
    } else { // Parent process
        close(pipefd[1]); // Close write end
        read(pipefd[0], buffer, sizeof(buffer));
        printf("Received message in parent: %s\n", buffer);
        close(pipefd[0]);
    }
    return 0;
}
```

```
/tmp/jh3lCpKBh1.o
Received message in parent: Hello, saran!
```

main.c

Run

Output

Clear

```c
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

struct msg_buffer {
    long msg_type;
    char msg_text[100];
};

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

    key = ftok("progfile", 65); // Generate unique key
    msg_id = msgget(key, 0666 | IPC_CREAT); // Create message queue

    message.msg_type = 1;
    strcpy(message.msg_text, "Hello, message queue!");

    msgsnd(msg_id, &message, sizeof(message), 0); // Send message

```

```
/tmp/T4xeoOkvXT.o
Received message: Hello, message queue!
```

main.c

Run

Output

```c
#include <stdio.h>
#include <stdbool.h>

#define MAX_PROCESSES 5
#define MAX_RESOURCES 3

int available[MAX_RESOURCES];
int max[MAX_PROCESSES][MAX_RESOURCES];
int allocation[MAX_PROCESSES][MAX_RESOURCES];
int need[MAX_PROCESSES][MAX_RESOURCES];

bool isSafeState(int process, int request[]) {
    int work[MAX_RESOURCES];
    for (int i = 0; i < MAX_RESOURCES; i++) {
        work[i] = available[i];
    }

    int finish[MAX_PROCESSES] = {0};
    bool canExecute = true;

    // Try to allocate the requested resources temporarily
    for (int i = 0; i < MAX_RESOURCES; i++) {
```

```
78
9
0
98
66
78
90
86
67
56
Enter the process number (0 to 3) making a request: 7
Enter the request for each resource type:
8

1
3
4
56
78
9
7
Request granted! System is in safe state.
```