main.c

Share Run

Output

Toggle light mode

```c
#include<stdio.h>
#include<unistd.h>
int main()
{
 printf("Process ID: %d\n", getpid() );
 printf("Parent Process ID: %d\n", getpid() );
 return 0;
}

```

```
Process ID: 38464
Parent Process ID: 38464


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

main.c

Share Run

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
int main(int argc, char *argv[]) {
    if (argc != 3) {
        perror("Usage: ./filecopy source destination");
        return 1;
    }
    int src = open(argv[1], O_RDONLY);
    int dest = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, 0644);
    char buffer[1024];
    ssize_t bytesRead;
    if (src < 0 || dest < 0) {
        perror("Error opening files");
        return 1;
    }
    while ((bytesRead = read(src, buffer, sizeof(buffer))) > 0) {
        write(dest, buffer, bytesRead);
    }
    close(src);
    close(dest);
    return 0;
}
```

Output

```
Usage: ./filecopy source destination: Success


=== Code Exited With Errors ===
```

main.c

Share Run Output

```c
#include <stdio.h>
typedef struct {
    int id, burst_time, waiting_time, turnaround_time;
} Process;
void findWaitingTime(Process proc[], int n) {
    proc[0].waiting_time = 0;
    for (int i = 1; i < n; i++)
        proc[i].waiting_time = proc[i - 1].waiting_time + proc[i - 1].burst_time;
}
void findTurnaroundTime(Process proc[], int n) {
    for (int i = 0; i < n; i++)
        proc[i].turnaround_time = proc[i].burst_time + proc[i].waiting_time;
}
void findavgTime(Process proc[], int n) {
    findWaitingTime(proc, n);
    findTurnaroundTime(proc, n);
    float total_wait = 0, total_turnaround = 0;
    for (int i = 0; i < n; i++) {
        total_wait += proc[i].waiting_time;
        total_turnaround += proc[i].turnaround_time;
    }
    printf("Average Waiting Time: %.2f\n", total_wait / n);
    printf("Average Turnaround Time: %.2f\n", total_turnaround / n);
}
int main() {
    Process proc[] = {{1, 5}, {2, 3}, {3, 8}};
    int n = sizeof(proc) / sizeof(proc[0]);
    findavgTime(proc, n);
    return 0;
}
```

```
Average Waiting Time: 4.33
Average Turnaround Time: 9.67


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

main.c

Share Run

```c
#include <stdio.h>
typedef struct {
    int id;
    int burst_time;
} Process;

void sortProcesses(Process processes[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (processes[j].burst_time > processes[j + 1].burst_time) {
                Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }
}
void scheduleProcesses(Process processes[], int n) {
    sortProcesses(processes, n);
    printf("Process Execution Order:\n");
    for (int i = 0; i < n; i++) {
        printf("Process ID: %d, Burst Time: %d\n", processes[i].id, processes[i].burst_time);
    }
}
int main() {
    Process processes[] = {{1, 6}, {2, 8}, {3, 7}, {4, 3}};
    int n = sizeof(processes) / sizeof(processes[0]);
    scheduleProcesses(processes, n);
    return 0;
}
```

Output

```
Process Execution Order:
Process ID: 4, Burst Time: 3
Process ID: 1, Burst Time: 6
Process ID: 3, Burst Time: 7
Process ID: 2, Burst Time: 8


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

main.c

Share Run

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_PROCESSES 10
typedef struct {
    int id;
    int priority;
} Process;
void schedule(Process processes[], int n) {
    int highestPriorityIndex = 0;
    for (int i = 1; i < n; i++) {
        if (processes[i].priority > processes[highestPriorityIndex].priority) {
            highestPriorityIndex = i;
        }
    }
    printf("Executing Process ID: %d with Priority: %d\n", processes[highestPriorityIndex].id,
        processes[highestPriorityIndex].priority);
}
int main() {
    Process processes[MAX_PROCESSES] = {{1, 3}, {2, 5}, {3, 1}, {4, 4}};
    int n = 4;
    schedule(processes, n);
    return 0;
}
```

Output

```
Executing Process ID: 2 with Priority: 5


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

```c
#include <stdio.h>
#include <stdlib.h>
struct Process {
    int id, bt, pt, wt, tat;
};
void findWaitingTime(struct Process proc[], int n) {
    int complete = 0, t = 0, min_pt = 9999, min_index;
    while (complete != n) {
        for (int i = 0; i < n; i++) {
            if (proc[i].bt > 0 && proc[i].pt < min_pt) {
                min_pt = proc[i].pt;
                min_index = i;
            }
        }
        t++;
        proc[min_index].bt--;
        if (proc[min_index].bt == 0) {
            proc[min_index].tat = t;
            proc[min_index].wt = t - proc[min_index].pt;
            complete++;
        }
        min_pt = 9999;
    }
}
void findTurnAroundTime(struct Process proc[], int n) {
    for (int i = 0; i < n; i++)
        proc[i].tat = proc[i].wt + proc[i].bt;
}
void priorityScheduling(struct Process proc[], int n) {
    findWaitingTime(proc, n);
    findTurnAroundTime(proc, n);
    printf("Process\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++)
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", proc[i].id, proc[i].bt, proc[i].pt, proc[i].wt, proc[i].tat);
}
int main() {
    struct Process proc[] = {{1, 10, 2}, {2, 5, 1}, {3, 8, 3}};
    int n = sizeof(proc) / sizeof(proc[0]);
    priorityScheduling(proc, n);
    return 0;
}
```

```
Process Burst Time  Priority    Waiting Time    Turnaround Time
1   0       2       13      13
2   0       1       4       4
3   0       3       20      20


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

main.c

Share Run

```c
#include <stdio.h>
int main() {
    int n, i, j, temp;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    int bt[n], p[n], wt[n], tat[n], total_wt = 0, total_tat = 0;
    printf("Enter burst times: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &bt[i]);
        p[i] = i + 1;
    }
    for (i = 0; i < n - 1; i++) {
        for (j = i + 1; j < n; j++) {
            if (bt[i] > bt[j]) {
                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];
        total_wt += wt[i];
        total_tat += tat[i];
    }
    printf("\nProcess\tBurst\tWait\tTurnaround\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("\nAvg Wait = %.2f, Avg Turnaround = %.2f\n",
           (float)total_wt / n, (float)total_tat / n);
    return 0;
}
```

Output

```
Enter number of processes: 3
Enter burst times: 3
6
4

Process Burst   Wait    Turnaround
P1  3   0   3
P3  4   3   7
P2  6   7   13

Avg Wait = 3.33, Avg Turnaround = 7.67


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

```c
#include <stdio.h>
int main() {
    int n, i, tq, time = 0, done = 0;
    printf("Enter number of processes and time quantum: ");
    scanf("%d %d", &n, &tq);
    int bt[n], rem_bt[n], wt[n], tat[n];
    printf("Enter burst times: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &bt[i]);
        rem_bt[i] = bt[i];
        wt[i] = 0;  }
    while (done < n) {
        for (i = 0; i < n; i++) {
            if (rem_bt[i] > 0) {
                if (rem_bt[i] <= tq) {
                    time += rem_bt[i];
                    wt[i] = time - bt[i];
                    rem_bt[i] = 0;
                    done++;
                } else {
                    rem_bt[i] -= tq;
                    time += tq;  }  } } }
    printf("\nProcess\tBurst\tWait\tTurnaround\n");
    for (i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        printf("P%d\t%d\t%d\t%d\n", i + 1, bt[i], wt[i], tat[i]); }
    float total_wt = 0, total_tat = 0;
    for (i = 0; i < n; i++) {
        total_wt += wt[i];
        total_tat += tat[i];
    }
    printf("\nAvg Wait = %.2f, Avg Turnaround = %.2f\n", total_wt / n, total_tat / n);
    return 0;}
```

```
Enter number of processes and time quantum: 4
5
Enter burst times: 23
32
3
45

Process Burst   Wait    Turnaround
P1  23  43  66
P2  32  56  88
P3  3   10  13
P4  45  58  103

Avg Wait = 41.75, Avg Turnaround = 67.50


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

main.c

Share Run

```c
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
#include <string.h>
int main() {
    key_t key = ftok("shmfile", 65);
    int shmid = shmget(key, 1024, 0666 | IPC_CREAT);
    char *str = (char *)shmat(shmid, (void *)0, 0);
    if (fork() == 0) {
        strcpy(str, "Hello from shared memory!");
        shmdt(str);
    } else {
        sleep(1);
        printf("Data read: %s\n", str);
        shmdt(str);
        shmctl(shmid, IPC_RMID, NULL);
    }
    return 0;
}
```

Output

```
Data read: Hello from shared memory!


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

main.c

Share Run

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
#include <unistd.h>
#define MSG_SIZE 100
struct msg_buffer {
    long msg_type;
    char msg_text[MSG_SIZE];
};
int main() {
    key_t key = ftok("progfile", 65);
    int msgid = msgget(key, 0666 | IPC_CREAT);
    struct msg_buffer message;
    message.msg_type = 1;
    strcpy(message.msg_text, "Hello, World!");
    msgsnd(msgid, &message, sizeof(message), 0);
    msgrcv(msgid, &message, sizeof(message), 1, 0);
    printf("Received: %s\n", message.msg_text);
    msgctl(msgid, IPC_RMID, NULL);
    return 0;
}

```

Output

```
Received: Hello, World!


=== Code Execution Successful
```