main.c

Share | Run

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main() {
    pid_t pid;

    // Create a new process
    pid = fork();

    if (pid < 0) {
        // Fork failed
        perror("Fork failed");
        return 1;
    } else if (pid == 0) {
        // Child process
        printf("Child Process:\n");
        printf("Process ID (PID): %d\n", getpid());
        printf("Parent Process ID (PPID): %d\n", getppid());
    } else {
        // Parent process
        printf("Parent Process:\n");
        printf("Process ID (PID): %d\n", getpid());
        printf("Child Process ID: %d\n", pid);
    }

    return 0;
}
```

Output | Clear

```
Parent Process:
Process ID (PID): 7561
Child Process ID: 7562
Child Process:
Process ID (PID): 7562
Parent Process ID (PPID): 7561


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

main.c

Share | Run

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#define BUFFER_SIZE 1024
int main() {
    int source_fd, dest_fd;
    ssize_t bytes_read, bytes_written;
    char buffer[BUFFER_SIZE];
    source_fd = open("source.txt", O_RDONLY);
    if (source_fd < 0) {
        perror("Error opening source file");
        exit(EXIT_FAILURE); }
    dest_fd = open("destination.txt", O_WRONLY | O_CREAT | O_TRUNC,
        0644);
    if (dest_fd < 0) {
        perror("Error opening destination file");
        close(source_fd);
        exit(EXIT_FAILURE);}
    while ((bytes_read = read(source_fd, buffer, BUFFER_SIZE)) > 0)
        {
        bytes_written = write(dest_fd, buffer, bytes_read);
        if (bytes_written != bytes_read) {
            perror("Error writing to destination file");
            close(source_fd);
            close(dest_fd);
            exit(EXIT_FAILURE);}}
    close(source_fd);
    close(dest_fd);
printf("File copied successfully.\n");
    return 0;}
```

Output | Clear

```
Error opening source file: No such file or directory


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

Programiz
C Online Compiler

main.c

Share Run

```c
#include <stdio.h>
struct Process {
    int pid;
    int burstTime;
    int waitingTime;
    int turnaroundTime;
};
void calculateTimes(struct Process p[], int n) {
    p[0].waitingTime = 0;
    p[0].turnaroundTime = p[0].burstTime;
    for (int i = 1; i < n; i++) {
        p[i].waitingTime = p[i - 1].waitingTime + p[i - 1].burstTime;
        p[i].turnaroundTime = p[i].waitingTime + p[i].burstTime;
    }
}
void printSchedule(struct Process p[], int n) {
    printf("PID\tBurst\tWaiting\tTurnaround\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\n", p[i].pid, p[i].burstTime, p[i].waitingTime, p[i]
            .turnaroundTime);
    }
}
int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    struct Process p[n];
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter burst time for Process %d: ", p[i].pid);
        scanf("%d", &p[i].burstTime);
    }

    calculateTimes(p, n);
    printSchedule(p, n);

    return 0;
}
```

Output

```
Enter number of processes: 4
Enter burst time for Process 1: 12
Enter burst time for Process 2:

14
Enter burst time for Process 3: 15
Enter burst time for Process 4: 16
PID Burst   Waiting Turnaround
1   12  0   12
2   14  12  26
3   15  26  41
4   16  41  57


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

Programiz
C Online Compiler

Programiz PRO >

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]; // burst time and process IDs
    for (i = 0; i < n; i++) {
        printf("Enter burst time for process %d: ", i+1);
        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;
            }
        }
    }

int wt[n], tat[n];
    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];
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]);
 return 0;
}
```

Output

Clear

```
Enter number of processes: 4
Enter burst time for Process 1: 12
Enter burst time for Process 2:

14
Enter burst time for Process 3: 15
Enter burst time for Process 4: 16
PID Burst   Waiting Turnaround
1   12  0   12
2   14  12  26
3   15  26  41
4   16  41  57


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

Online Compiler

main.c

Share Run

```c
#include <stdio.h>
struct Process {
    int id, bt, pr, wt, tat;
};
int main() {
    int n, i, j;
    printf("Enter number of processes: ");
    scanf("%d", &n);
struct Process p[n], temp;
for(i = 0; i < n; i++) {
        p[i].id = i+1;
        printf("Enter burst time and priority for P%d: ", i+1);
        scanf("%d %d", &p[i].bt, &p[i].pr);
    }
    for(i = 0; i < n-1; i++)
        for(j = i+1; j < n; j++)
            if(p[j].pr > p[i].pr) {
                temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
p[0].wt = 0;
    for(i = 1; i < n; i++)
        p[i].wt = p[i-1].wt + p[i-1].bt;
for(i = 0; i < n; i++)
        p[i].tat = p[i].wt + p[i].bt;
printf("\nProcess\tBT\tPR\tWT\tTAT\n");
    for(i = 0; i < n; i++)
        printf("P%d\t%d\t%d\t%d\t%d\n", p[i].id, p[i].bt, p[i].pr, p[i].wt, p[i]
            .tat);

    return 0;
}
```

Output

```
Enter number of processes: 4
Enter burst time for Process 1: 12
Enter burst time for Process 2:

14
Enter burst time for Process 3: 15
Enter burst time for Process 4: 16
PID Burst   Waiting Turnaround
1   12  0   12
2   14  12  26
3   15  26  41
4   16  41  57


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