

 Share

Run

Output

Clear

```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 int main() {
5     pid_t pid = fork(); // Create a new process
6
7     if (pid == -1) {
8         perror("Fork failed");
9         return 1;
10    }
11
12    if (pid == 0) {
13        // Child process
14        printf("Child Process ID: %d\n", getpid());
15        printf("Parent Process ID: %d\n", getppid());
16    } else {
17        // Parent process
18        printf("Parent Process ID: %d\n", getpid());
19    }
20
21    return 0;
22 }
```

```
Parent Process ID: 44982
Child Process ID: 44983
Parent Process ID: 44982
```



with



HP AI LAPTOPS

with Intel® Core™ Ultra 5 processor

Walk-in to exchange at HP World

Effective price ₹73 999.00 ₹56 999.00*

*T&C apply.



HP 14/15 AI PC

main.c



Run

Output

Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void copyFile(const char *source, const char *destination) {
5     FILE *src = fopen(source, "r");
6     FILE *dest = fopen(destination, "w");
7
8     if (src == NULL || dest == NULL) {
9         perror("File opening failed");
10        exit(1);
11    }
12
13    char ch;
14    while ((ch = fgetc(src)) != EOF) {
15        fputc(ch, dest);
16    }
17
18    fclose(src);
19    fclose(dest);
20 }
21
22 int main() {
23     copyFile("source.txt", "destination.txt");
24     printf("File copied successfully.\n");
25     return 0;
26 }
27
```

File opening failed: Permission denied

=== Code Exited With Errors ===



main.c



Run

Output

Clear

```
1 #include <stdio.h>
2
3 void FCFS(int burstTime[], int n) {
4     int waitingTime[n], turnaroundTime[n];
5     int totalWT = 0, totalTAT = 0;
6
7     waitingTime[0] = 0;
8     for (int i = 1; i < n; i++) {
9         waitingTime[i] = burstTime[i - 1] + waitingTime[i - 1];
10    }
11
12    for (int i = 0; i < n; i++) {
13        turnaroundTime[i] = burstTime[i] + waitingTime[i];
14        totalWT += waitingTime[i];
15        totalTAT += turnaroundTime[i];
16    }
17
18    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
19
20    for (int i = 0; i < n; i++) {
21        printf("%d\t%d\t\t%d\t\t%d\n", i+1, burstTime[i],
22            waitingTime[i], turnaroundTime[i]);
23    }
24
25    printf("\nAverage Waiting Time: %.2f\n", (float)totalWT/n);
26    printf("Average Turnaround Time: %.2f\n", (float)totalTAT/n);
27 }
```

Process	Burst Time	Waiting Time	Turnaround Time
1	6	0	6
2	8	6	14
3	7	14	21

Average Waiting Time: 6.67
Average Turnaround Time: 13.67

=== Code Execution Successful ===

main.c



Share

Run

Output

Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void SJF(int burstTime[], int n) {
5     int waitingTime[n], turnaroundTime[n];
6     int totalWT = 0, totalTAT = 0;
7
8     // Sorting the burst time in increasing order
9     for (int i = 0; i < n-1; i++) {
10         for (int j = i+1; j < n; j++) {
11             if (burstTime[i] > burstTime[j]) {
12                 int temp = burstTime[i];
13                 burstTime[i] = burstTime[j];
14                 burstTime[j] = temp;
15             }
16         }
17     }
18
19     waitingTime[0] = 0;
20     for (int i = 1; i < n; i++) {
21         waitingTime[i] = burstTime[i - 1] + waitingTime[i - 1];
22     }
23
24     for (int i = 0; i < n; i++) {
25         turnaroundTime[i] = burstTime[i] + waitingTime[i];
26         totalWT += waitingTime[i];
27         totalTAT += turnaroundTime[i];
28     }
29 }
```

Process	Burst Time	Waiting Time	Turnaround Time
1	6	0	6
2	7	6	13
3	8	13	21

Average Waiting Time: 6.33

Average Turnaround Time: 13.33

=== Code Execution Successful ===

main.c



Share

Run

Output

Clear

```
1 #include <stdio.h>
2
3 void priorityScheduling(int burstTime[], int priority[], int n) {
4     int waitingTime[n], turnaroundTime[n];
5     int totalWT = 0, totalTAT = 0;
6
7     // Sorting processes based on priority
8     for (int i = 0; i < n-1; i++) {
9         for (int j = i+1; j < n; j++) {
10             if (priority[i] > priority[j]) {
11                 int temp = priority[i];
12                 priority[i] = priority[j];
13                 priority[j] = temp;
14
15                 temp = burstTime[i];
16                 burstTime[i] = burstTime[j];
17                 burstTime[j] = temp;
18             }
19         }
20     }
21
22     waitingTime[0] = 0;
23     for (int i = 1; i < n; i++) {
24         waitingTime[i] = burstTime[i - 1] + waitingTime[i - 1];
25     }
26
27     for (int i = 0; i < n; i++) {
28         turnaroundTime[i] = burstTime[i] + waitingTime[i];
29         totalWT += waitingTime[i];
30         totalTAT += turnaroundTime[i];
31     }
32
33     printf("Average Waiting Time: %.2f\n", (float)totalWT/n);
34     printf("Average Turnaround Time: %.2f\n", (float)totalTAT/n);
35 }
```

Process	Burst Time	Priority	Waiting Time	Turnaround Time
1	8	1	0	8
2	6	2	8	14
3	7	3	14	21

Average Waiting Time: 7.33
Average Turnaround Time: 14.33

=== Code Execution Successful ===

main.c



Run

Output

Clear

```
1 #include <stdio.h>
2
3 void preemptivePriorityScheduling(int burstTime[], int priority[]
    , int n) {
4     int waitingTime[n], turnaroundTime[n];
5     int totalWT = 0, totalTAT = 0;
6
7     // Preemptive priority scheduling logic
8     // Sorting processes based on priority
9     for (int i = 0; i < n-1; i++) {
10         for (int j = i+1; j < n; j++) {
11             if (priority[i] > priority[j]) {
12                 int temp = priority[i];
13                 priority[i] = priority[j];
14                 priority[j] = temp;
15
16                 temp = burstTime[i];
17                 burstTime[i] = burstTime[j];
18                 burstTime[j] = temp;
19             }
20         }
21     }
22
23     waitingTime[0] = 0;
24     for (int i = 1; i < n; i++) {
25         waitingTime[i] = burstTime[i - 1] + waitingTime[i - 1];
26     }
27
28     // Calculating total waiting time and total turnaround time
```

Process	Burst Time	Priority	Waiting Time	Turnaround Time
1	8	1	0	8
2	6	2	8	14
3	7	3	14	21

Average Waiting Time: 7.33
Average Turnaround Time: 14.33

=== Code Execution Successful ===

main.c



Share

Run

Output

Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void SJF(int burstTime[], int n) {
5     int waitingTime[n], turnaroundTime[n];
6     int totalWT = 0, totalTAT = 0;
7
8     // Sorting the burst time in increasing order
9     for (int i = 0; i < n-1; i++) {
10         for (int j = i+1; j < n; j++) {
11             if (burstTime[i] > burstTime[j]) {
12                 int temp = burstTime[i];
13                 burstTime[i] = burstTime[j];
14                 burstTime[j] = temp;
15             }
16         }
17     }
18
19     waitingTime[0] = 0;
20     for (int i = 1; i < n; i++) {
21         waitingTime[i] = burstTime[i - 1] + waitingTime[i - 1];
22     }
23
24     for (int i = 0; i < n; i++) {
25         turnaroundTime[i] = burstTime[i] + waitingTime[i];
26         totalWT += waitingTime[i];
27         totalTAT += turnaroundTime[i];
28     }
29 }
```

Process	Burst Time	Waiting Time	Turnaround Time
1	6	0	6
2	7	6	13
3	8	13	21

Average Waiting Time: 6.33
Average Turnaround Time: 13.33

=== Code Execution Successful ===

main.c



Share

Run

Output

Clear

```
1 #include <stdio.h>
2
3 void roundRobin(int burstTime[], int n, int quantum) {
4     int remainingBurstTime[n];
5     for (int i = 0; i < n; i++) {
6         remainingBurstTime[i] = burstTime[i];
7     }
8
9     int time = 0;
10    while (1) {
11        int done = 1;
12        for (int i = 0; i < n; i++) {
13            if (remainingBurstTime[i] > 0) {
14                done = 0;
15                if (remainingBurstTime[i] > quantum) {
16                    remainingBurstTime[i] -= quantum;
17                    time += quantum;
18                } else {
19                    time += remainingBurstTime[i];
20                    remainingBurstTime[i] = 0;
21                }
22                printf("Process %d executed at time %d\n", i+1,
23                    time);
24            }
25        }
26        if (done) break;
27    }
28 }
```

```
Process 1 executed at time 3
Process 2 executed at time 6
Process 3 executed at time 9
Process 1 executed at time 12
Process 2 executed at time 15
Process 3 executed at time 18
Process 2 executed at time 20
Process 3 executed at time 21
```

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


main.c



Share

Run

Output

Clear

```
1 #include <stdio.h>
2 #include <sys/ipc.h>
3 #include <sys/shm.h>
4
5 int main() {
6     key_t key = ftok("shmfile", 65);
7     int shmid = shmget(key, 1024, 0666 | IPC_CREAT);
8     char *str = (char*) shmat(shmid, (void*)0, 0);
9
10    printf("Write Data: ");
11    gets(str);
12
13    printf("Data written in memory: %s\n", str);
14
15    shmdt(str);
16
17    return 0;
18 }
19
```

```
Process 1 executed at time 3
Process 2 executed at time 6
Process 3 executed at time 9
Process 1 executed at time 12
Process 2 executed at time 15
Process 3 executed at time 18
Process 2 executed at time 20
Process 3 executed at time 21
```

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

main.c

Share

Run

```
1 #include <stdio.h>
2 #include <sys/ipc.h>
3 #include <sys/msg.h>
4 #include <string.h>
5
6 struct message {
7     long mtype;
8     char mtext[100];
9 };
10
11 int main() {
12     key_t key = ftok("msgqueue", 65);
13     int msgid = msgget(key, 0666 | IPC_CREAT);
14
15     struct message msg;
16     msg.mtype = 1;
17
18     printf("Enter message: ");
19     fgets(msg.mtext, sizeof(msg.mtext), stdin);
20
21     msgsnd(msgid, &msg, sizeof(msg), 0);
22     printf("Message sent: %s\n", msg.mtext);
23
24     return 0;
25 }
26
```

Output

Clear

Enter message:

main.c



Share

Run

Output

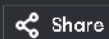
Clear

```
1 #include <stdio.h>
2 #include <pthread.h>
3
4 void* printMessage(void* arg) {
5     printf("Hello from thread!\n");
6     return NULL;
7 }
8
9 int main() {
10     pthread_t thread;
11
12     pthread_create(&thread, NULL, printMessage, NULL);
13     pthread_join(thread, NULL);
14
15     return 0;
16 }
17
```

Hello from thread!

=== Code Execution Successful ===

main.c



Run

Output

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4
5 #define N 5
6
7 pthread_mutex_t mutex;
8 pthread_mutex_t chopsticks[N];
9
10 void* philosopher(void* num) {
11     int id = *(int*)num;
12
13     while (1) {
14         printf("Philosopher %d is thinking\n", id);
15         usleep(1000); // Thinking
16
17         pthread_mutex_lock(&chopsticks[id]);
18         pthread_mutex_lock(&chopsticks[(id + 1) % N]);
19
20         printf("Philosopher %d is eating\n", id);
21         usleep(1000); // Eating
22
23         pthread_mutex_unlock(&chopsticks[id]);
24         pthread_mutex_unlock(&chopsticks[(id + 1) % N]);
25     }
26 }
27
28 int main() {
29     pthread_t t[5];
```

```
Philosopher 0 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 1 is thinking
Philosopher 4 is thinking
Philosopher 0 is eating
Philosopher 2 is eating
Philosopher 0 is thinking
Philosopher 4 is eating
Philosopher 2 is thinking
Philosopher 1 is eating
Philosopher 4 is thinking
Philosopher 1 is thinking
Philosopher 3 is eating
Philosopher 0 is eating
Philosopher 0 is thinking
Philosopher 2 is eating
Philosopher 4 is eating
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 3 is eating
Philosopher 1 is eating
Philosopher 2 is thinking
Philosopher 1 is thinking
Philosopher 3 is thinking
Philosopher 0 is eating
Philosopher 2 is eating
Philosopher 4 is eating
Philosopher 0 is thinking
```

main.c



Share

Run

Output

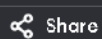
Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void worstFit(int blockSize[], int m, int processSize[], int n) {
5     int allocation[n];
6     for (int i = 0; i < n; i++) {
7         allocation[i] = -1;
8     }
9
10    for (int i = 0; i < n; i++) {
11        int worstIdx = -1;
12        for (int j = 0; j < m; j++) {
13            if (blockSize[j] >= processSize[i]) {
14                if (worstIdx == -1 || blockSize[j] >
15                    blockSize[worstIdx]) {
16                    worstIdx = j;
17                }
18            }
19
20            if (worstIdx != -1) {
21                allocation[i] = worstIdx;
22                blockSize[worstIdx] -= processSize[i];
23            }
24        }
25
26        for (int i = 0; i < n; i++) {
27            if (allocation[i] != -1) {
28                printf("Process %d allocated to block %d\n", i+1, allocation[i]+1);
29            }
30        }
31    }
32}
```

```
Process 1 allocated to block 5
Process 2 allocated to block 2
Process 3 allocated to block 5
Process 4 not allocated
```

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

main.c



Run

Output

```
1 #include <stdio.h>
2 #include <dirent.h>
3
4 int main() {
5     struct dirent *entry;
6     DIR *dp = opendir(".");
7
8     if (dp == NULL) {
9         perror("opendir");
10        return 1;
11    }
12
13    while ((entry = readdir(dp)) != NULL) {
14        printf("%s\n", entry->d_name);
15    }
16
17    closedir(dp);
18    return 0;
19 }
20
```

```
.
..
.bash_logout
.bashrc
.profile
```

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

main.c

Share

Run

```
1 #include <stdio.h>
2 #include <dirent.h>
3 #include <string.h>
4
5 void listDirectory(char *path) {
6     struct dirent *entry;
7     DIR *dp = opendir(path);
8
9     if (dp == NULL) {
10         perror("opendir");
11         return;
12     }
13
14     while ((entry = readdir(dp)) != NULL) {
15         printf("%s/%s\n", path, entry->d_name);
16     }
17
18     closedir(dp);
19 }
20
21 int main() {
22     char *dirPath = ".";
23     listDirectory(dirPath);
24     return 0;
25 }
26
```

Output

```
./
../
../.bash_logout
../.bashrc
../.profile

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

main.c



Share

Run

Output

```
1 #include <stdio.h>
2
3 struct Employee {
4     int empId;
5     char name[30];
6 };
7
8 int main() {
9     FILE *fp = fopen("employee.dat", "rb+");
10    if (fp == NULL) {
11        printf("File could not be opened\n");
12        return 1;
13    }
14
15    struct Employee emp = {101, "John Doe"};
16    fseek(fp, 0, SEEK_SET);
17    fwrite(&emp, sizeof(struct Employee), 1, fp);
18
19    fseek(fp, 0, SEEK_SET);
20    fread(&emp, sizeof(struct Employee), 1, fp);
21
22    printf("Employee ID: %d\n", emp.empId);
23    printf("Employee Name: %s\n", emp.name);
24
25    fclose(fp);
26    return 0;
27 }
28
```

File could not be opened

=== Code Exited With Errors ===