

FREE AI Code Generator Gener... Online C Compiler - Programiz +

programiz.com/c-programming/online-compiler/

BLACK FRIDAY SALE Save 60% on PRO: Sale Ends Soon! Claim My Discount

Sale ends in 00d : 03hrs : 18mins : 05s

Programiz PRO >

Programiz
C Online Compiler

Premium Coding Courses by Programiz

Programiz PRO

main.c

Process ID: 17897
ParentProcess ID: 17897
==== Code Execution Successful ===

Output

Clear

Run

Share

Run

Output

main.c

```
1 #include<stdio.h>
2 #include<unistd.h>
3 int main()
4 {
5 printf("Process ID: %d\n", getpid()); printf("ParentProcess ID: %d\n"
6         ,getpid());
6 return 0;
7 }
```

NIFTY -0.01%

Search

10:11 28-11-2024 ENG IN

FREE AI Code Generator: Gener... | #include <stdio.h> #include <s... | Online C Compiler - Programiz

programiz.com/c-programming/online-compiler/

BLACK FRIDAY SALE Save 60% on PRO! Sale Ends Soon! Claim My Discount

Sale ends in 00d : 03hrs : 09mins : 50s

Programiz PRO

main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     FILE *fptr1, *fptr2; char
6     filename[100], c;
7     printf("Enter the filename to open for reading \n");
8     scanf("%s", filename);
9     fptr1 = fopen(filename, "r"); if
10    (fptr1 == NULL)
11    {
12        printf("Cannot open file %s \n", filename);
13        exit(0);
14    }
15    printf("Enter the filename to open for writing \n");
16    scanf("%s", filename);
17    fptr2 = fopen(filename, "w"); if
18    (fptr2 == NULL)
19    {
20        printf("Cannot open file %s \n", filename);
21        exit(0);
22    }
23    c = fgetc(fptr1);
24    while (c != EOF)
25    {
26        fputc(c, fptr2); c
27        = fgetc(fptr1);
28    }
29    printf("\nContents copied to %s", filename);
30    fclose(fptr1);
31    fclose(fptr2);
32    return 0;
33 }
```

Output

Enter the filename to open for reading
source.txt

Cannot open file source.txt

== Code Execution Successful ==

Programiz PRO

Premium Coding Courses by Programiz

Learn More

24°C Mostly cloudy

Search

10:20 28-11-2024 ENG IN

FREE AI Code Generator Gener... | #include <stdio.h> #include <s... | Online C Compiler - Programiz

programiz.com/c-programming/online-compiler/

BLACK FRIDAY SALE Save 60% on PRO: Sale Ends Soon! Claim My Discount

Sale ends in 00d 02hrs : 56mins : 53s

Programiz PRO

main.c

```
1 #include <stdio.h>
2 struct Process {
3     int id, burstTime, waitTime, turnAroundTime;
4 };
5 int main() {
6     int n;
7     printf("Enter number of processes: ");
8     scanf("%d", &n);
9
10    struct Process processes[n];
11    for (int i = 0; i < n; i++) {
12        processes[i].id = i + 1;
13        printf("Enter burst time for Process %d: ", processes[i].id);
14        scanf("%d", &processes[i].burstTime);
15    }
16    for (int i = 0; i < n; i++) {
17        processes[i].waitTime = (i == 0) ? 0 : processes[i - 1].waitTime + processes[i - 1].burstTime;
18        processes[i].turnAroundTime = processes[i].waitTime + processes[i].burstTime;
19    }
20    printf("\nID\tBT\tWT\tTAT\n");
21    float totalWT = 0, totalTAT = 0;
22    for (int i = 0; i < n; i++) {
23        printf("%d\t%d\t%d\t%d\n", processes[i].id, processes[i].burstTime, processes[i].waitTime,
24               processes[i].turnAroundTime);
25        totalWT += processes[i].waitTime;
26        totalTAT += processes[i].turnAroundTime;
27    }
28    printf("\nAvg WT: %.2f, Avg TAT: %.2f\n", totalWT/n, totalTAT/n);
29    return 0;
}
```

Run

Output

```
Enter number of processes: 3
Enter burst time for Process 1: 24
Enter burst time for Process 2: 3
Enter burst time for Process 3: 3

ID BT WT TAT
1 24 0 24
2 3 24 27
3 3 27 30

Avg WT: 17.00, Avg TAT: 27.00

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

Programiz PRO

Premium Courses by Programiz

Learn More

24°C Mostly cloudy

Search

10:34 28-11-2024 ENG IN

FREE AI Code Generator: Gener... | #include <stdio.h> #include <s... | Online C Compiler - Programiz

programmiz.com/c-programming/online-compiler/

BLACK FRIDAY SALE Save 60% on PRO: Sale Ends Soon! Claim My Discount

Premium Coding Courses by Programiz

Programiz PRO

main.c

```
1 #include <stdio.h>
2 struct Process {
3     int id, burstTime, waitTime, turnAroundTime;
4 };
5 void sortProcesses(struct Process p[], int n) {
6     for (int i = 0; i < n - 1; i++) {
7         for (int j = 0; j < n - i - 1; j++) {
8             if (p[j].burstTime > p[j + 1].burstTime) {
9                 struct Process temp = p[j];
10                p[j] = p[j + 1];
11                p[j + 1] = temp;
12            }
13        }
14    }
15    int main() {
16        int n;
17        printf("Enter number of processes: ");
18        scanf("%d", &n);
19        struct Process processes[n];
20        for (int i = 0; i < n; i++) {
21            processes[i].id = i + 1;
22            printf("Enter burst time for Process %d: ", processes[i].id);
23            scanf("%d", &processes[i].burstTime);
24        }
25        sortProcesses(processes, n);
26        for (int i = 0; i < n; i++) {
27            processes[i].waitTime = (i == 0) ? 0 : processes[i - 1].turnAroundTime - processes[i - 1].burstTime;
28            processes[i].turnAroundTime = processes[i].waitTime + processes[i].burstTime;
29        }
30        printf("\nID\tBT\tWT\tTAT");
31        float totalWT = 0, totalTAT = 0;
32        for (int i = 0; i < n; i++) {
33            printf("\t%d\t%d\t%d\t%d\n", processes[i].id, processes[i].burstTime, processes[i].waitTime, processes[i].turnAroundTime);
34            totalWT += processes[i].waitTime;
35            totalTAT += processes[i].turnAroundTime;
36        }
37        printf("\nAvg WT: %.2f, Avg TAT: %.2f\n", totalWT / n, totalTAT / n);
38    }
}
```

Output

```
Enter number of processes: 3
Enter burst time for Process 1: 45
Enter burst time for Process 2: 32
Enter burst time for Process 3: 16
ID BT WT TAT
3 45 0 45
2 32 16 48
1 16 0 16
Avg WT: 21.33, Avg TAT: 32.33
== Code Execution Successful ==
```

Programiz PRO

Premium Courses by Programiz

Learn More

24°C Mostly cloudy

Search

10:38 28-11-2024 ENG IN

FREE AI Code Generator Gener... | #include <stdio.h> #include <s... | Online C Compiler - Programiz

programmiz.com/c-programming/online-compiler/

BLACK FRIDAY SALE Save 60% on PRO: Sale Ends Soon! Claim My Discount

Sale ends in 00d: 02hrs : 43mins : 51s

Programiz PRO

Programiz

C Online Compiler

main.c

```
1 #include <stdio.h>
2 struct Process {
3     int id, burstTime, waitTime, turnAroundTime, priority;
4 };
5 void sortProcesses(struct Process p[], int n) {
6     for (int i = 0; i < n - 1; i++) {
7         for (int j = 0; j < n - i - 1; j++) {
8             if (p[j].priority > p[j + 1].priority) {
9                 struct Process temp = p[j];
10                p[j] = p[j + 1];
11                p[j + 1] = temp;
12            }
13        }
14    }
15    int n;
16    printf("Enter number of processes: ");
17    scanf("%d", &n);
18    struct Process processes[n];
19    for (int i = 0; i < n; i++) {
20        processes[i].id = i + 1;
21        printf("Enter burst time for Process %d: ", processes[i].id);
22        scanf("%d", &processes[i].burstTime);
23        printf("Enter priority for Process %d (lower number = higher priority): ", processes[i].id);
24        scanf("%d", &processes[i].priority);
25    }
26    sortProcesses(processes, n);
27    for (int i = 0; i < n; i++) {
28        processes[i].waitTime = (i == 0) ? 0 : processes[i - 1].waitTime + processes[i - 1].burstTime;
29        processes[i].turnAroundTime = processes[i].waitTime + processes[i].burstTime;
30    }
31    printf("\nID\tBT\tP\tWT\tTAT");
32    float totalWT = 0, totalTAT = 0;
33    for (int i = 0; i < n; i++) {
34        printf("\n%d\t%d\t%d\t%d\t%d", processes[i].id, processes[i].burstTime, processes[i].priority, processes[i].waitTime, processes[i].turnAroundTime);
35        totalWT += processes[i].waitTime;
36        totalTAT += processes[i].turnAroundTime;
37    }
38    printf("\nAvg WT: %.2f, Avg TAT: %.2f\n", totalWT / n, totalTAT / n);
39    return 0;
40 }
```

Output

```
Enter number of processes: 3
Enter burst time for Process 1: 24
Enter priority for Process 1 (lower number = higher priority): 3
Enter burst time for Process 2: 32
Enter priority for Process 2 (lower number = higher priority): 1
Enter burst time for Process 3: 23
Enter priority for Process 3 (lower number = higher priority): 2
ID BT P WT TAT
2 32 1 0 32
3 23 2 32 55
1 24 3 55 79
Avg WT: 29.00, Avg TAT: 55.33
*** Code Execution Successful ***
```

Programiz PRO

Premium Courses by Programiz

Learn More

24°C Mostly cloudy

Search

10:46 28-11-2024 ENG IN

Sign in | FREE AI Code Generator: Generate | c compiler - Search | Online C Compiler - Programiz | 6 Construct a C program to implement FCFS scheduling algorithm | +

https://www.programiz.com/c-programming/online-compiler/

BLACK FRIDAY SALE Save 60% on PRO: Sale Ends Soon! Claim My Discount

Sale ends in 00d : 20hrs : 38mins : 59s

Programiz PRO >

Programiz
C Online Compiler

Premium Coding Courses by Programiz

Programiz PRO

main.c

```
1 #include <stdio.h>
2 struct Process {
3     int id;
4     int burst_time;
5     int arrival_time;
6     int priority;
7     int remaining_time;
8     int completion_time;
9     int waiting_time;
10    int turnaround_time;
11 };
12 void calculateTimes(struct Process proc[], int n) {
13     int time = 0, completed = 0, min_priority, shortest;
14     int is_completed[n];
15     for (int i = 0; i < n; i++) {
16         is_completed[i] = 0;
17         proc[i].remaining_time = proc[i].burst_time;
18     }
19     while (completed != n) {
20         min_priority = 10000;
21         shortest = -1;
22         for (int i = 0; i < n; i++) {
23             if (proc[i].arrival_time <= time && !is_completed[i] &&
24                 proc[i].priority < min_priority) {

```

Output

```
Enter the number of processes: 3
Enter arrival time, burst time, and priority for process 1: 0 3 3
Enter arrival time, burst time, and priority for process 2: 1 4 2
Enter arrival time, burst time, and priority for process 3: 2 6 4
ID Arrival Burst Priority Completion Waiting Turnaround
1 0 3 3 7 4 7
2 1 4 2 5 0 4
3 2 6 4 13 5 11

== Code Execution Successful ==
```

Programiz PRO

Premium Courses by Programiz

Learn More

80°F Mostly cloudy

Search

16:50 28-11-2024

Sign in | FREE AI Code Generator: Generate | c compiler - Search | Online C Compiler - Programiz | 6 Construct a C program to implement Round Robin Scheduling | +

https://www.programiz.com/c-programming/online-compiler/

BLACK FRIDAY SALE Save 60% on PRO: Sale Ends Soon! [Claim My Discount](#)

Sale ends in 00d : 20hrs : 38mins : 45s

Programiz PRO >

Programiz
C Online Compiler

Premium Coding Courses by Programiz

Programiz PRO

main.c

```
proc[i].priority < min_priority) {  
    min_priority = proc[i].priority;  
    shortest = i;  
}  
  
if (shortest != -1) {  
    proc[shortest].remaining_time--;  
    time++;  
    if (proc[shortest].remaining_time == 0) {  
        proc[shortest].completion_time = time;  
        proc[shortest].turnaround_time = proc[shortest]  
            .completion_time - proc[shortest].arrival_time;  
        proc[shortest].waiting_time = proc[shortest]  
            .turnaround_time - proc[shortest].burst_time;  
        is_completed[shortest] = 1;  
        completed++;  
    }  
} else {  
    time++;  
}  
}  
  
void printTable(struct Process proc[], int n) {  
    printf("ID\tArrival\tBurst\tPriority\tCompletion\tWaiting\tTurnaround\n");  
    for (int i = 0; i < n; i++) {  
        printf("%d\t", proc[i].id);  
        printf("%d\t", proc[i].arrival_time);  
        printf("%d\t", proc[i].burst_time);  
        printf("%d\t", proc[i].priority);  
        printf("%d\t", proc[i].completion_time);  
        printf("%d\t", proc[i].waiting_time);  
        printf("%d\n", proc[i].turnaround_time);  
    }  
}
```

Output

Clear

Enter the number of processes: 3
Enter arrival time, burst time, and priority for process 1: 0 3 3
Enter arrival time, burst time, and priority for process 2: 1 4 2
Enter arrival time, burst time, and priority for process 3: 2 6 4
ID Arrival Burst Priority Completion Waiting Turnaround
1 0 3 3 7 4 7
2 1 4 2 5 0 4
3 2 6 4 13 5 11
==== Code Execution Successful ===

Programiz PRO

Premium Courses by Programiz

Learn More

80°F Mostly cloudy

Search

16:51 28-11-2024

Sign in | FREE AI Code Generator: Generate | c compiler - Search | Online C Compiler - Programiz | 6 Construct a C program to implement FCFS scheduling algorithm | +

https://www.programiz.com/c-programming/online-compiler/

BLACK FRIDAY SALE Save 60% on PRO: Sale Ends Soon! [Claim My Discount](#)

Sale ends in 00d : 20hrs : 38mins : 40s

Programiz PRO >

Programiz
C Online Compiler

Premium Coding Courses by Programiz

Programiz PRO

main.c

```
round\n");
45+ for (int i = 0; i < n; i++) {
46     printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n", proc[i].id,
47           proc[i].arrival_time, proc[i].burst_time, proc[i]
48           .priority, proc[i].completion_time, proc[i].waiting_time
49           , proc[i].turnaround_time);
50 }
51 int main() {
52     int n;
53     printf("Enter the number of processes: ");
54     scanf("%d", &n);
55     struct Process proc[n];
56     for (int i = 0; i < n; i++) {
57         proc[i].id = i + 1;
58         printf("Enter arrival time, burst time, and priority for
59         process %d: ", i + 1);
60         scanf("%d %d", &proc[i].arrival_time, &proc[i].burst_time
61             , &proc[i].priority);
62     }
63     calculateTimes(proc, n);
64     printTable(proc, n);
65     return 0;
66 }
```

Run

Output

Clear

```
Enter the number of processes: 3
Enter arrival time, burst time, and priority for process 1: 0 3 3
Enter arrival time, burst time, and priority for process 2: 1 4 2
Enter arrival time, burst time, and priority for process 3: 2 6 4
ID Arrival Burst Priority Completion Waiting Turnaround
1 0 3 3 7 4 7
2 1 4 2 5 0 4
3 2 6 4 13 5 11
==== Code Execution Successful ===
```

Programiz PRO

Premium Courses by Programiz

Learn More

80°F Mostly cloudy

Search

16:51 28-11-2024

```
1 #include <stdio.h>
2 struct Process {
3     int id, burstTime, waitingTime, turnaroundTime;
4 };
5 void calculateTimes(struct Process proc[], int n) {
6     proc[0].waitingTime = 0;
7     for (int i = 1; i < n; i++) {
8         proc[i].waitingTime = proc[i - 1].waitingTime + proc[i - 1].burstTime;
9         proc[i].turnaroundTime = proc[i].waitingTime + proc[i].burstTime;
10    }
11 }
12 void sortProcesses(struct Process proc[], int n) {
13     struct Process temp;
14     for (int i = 0; i < n - 1; i++) {
15         for (int j = 0; j < n - i - 1; j++) {
16             if (proc[j].burstTime > proc[j + 1].burstTime) {
17                 temp = proc[j];
18                 proc[j] = proc[j + 1];
19                 proc[j + 1] = temp;
20             }
21         }
22     }
23     void displayResults(struct Process proc[], int n) {
24         float totalWait = 0, totalTurnaround = 0;
25         printf("\nID\tBurst\tWait\tTurnaround\n");
26         for (int i = 0; i < n; i++) {
27             totalWait += proc[i].waitingTime;
28             totalTurnaround += proc[i].turnaroundTime;
29             printf("%d\t%d\t%d\t%d\n", proc[i].id, proc[i].burstTime, proc[i].waitingTime, proc[i].turnaroundTime);
30         }
31         printf("\nAvg Wait: %.2f, Avg Turnaround: %.2f\n", totalWait / n, totalTurnaround / n);
32     }
33 }
```

```
20     }
21 }
22 void displayResults(struct Process proc[], int n) {
23     float totalWait = 0, totalTurnaround = 0;
24     printf("\nID\tBurst\tWait\tTurnaround\n");
25     for (int i = 0; i < n; i++) {
26         totalWait += proc[i].waitingTime;
27         totalTurnaround += proc[i].turnaroundTime;
28         printf("%d\t%d\t%d\t%d\n", proc[i].id, proc[i].burstTime, proc[i].waitingTime, proc[i].turnaroundTime);
29     }
30     printf("\nAvg Wait: %.2f, Avg Turnaround: %.2f\n", totalWait / n, totalTurnaround / n);
31 }
32 int main() {
33     struct Process proc[10];
34     int n;
35
36     printf("Enter number of processes: ");
37     scanf("%d", &n);
38     for (int i = 0; i < n; i++) {
39         proc[i].id = i + 1;
40         printf("Enter Burst Time for Process %d: ", proc[i].id);
41         scanf("%d", &proc[i].burstTime);
42     }
43     sortProcesses(proc, n);
44     calculateTimes(proc, n);
45     displayResults(proc, n);
46     return 0;
47 }
```



Output

```
Enter number of processes: 3
Enter Burst Time for Process 1: 24
Enter Burst Time for Process 2: 6
Enter Burst Time for Process 3: 3
```

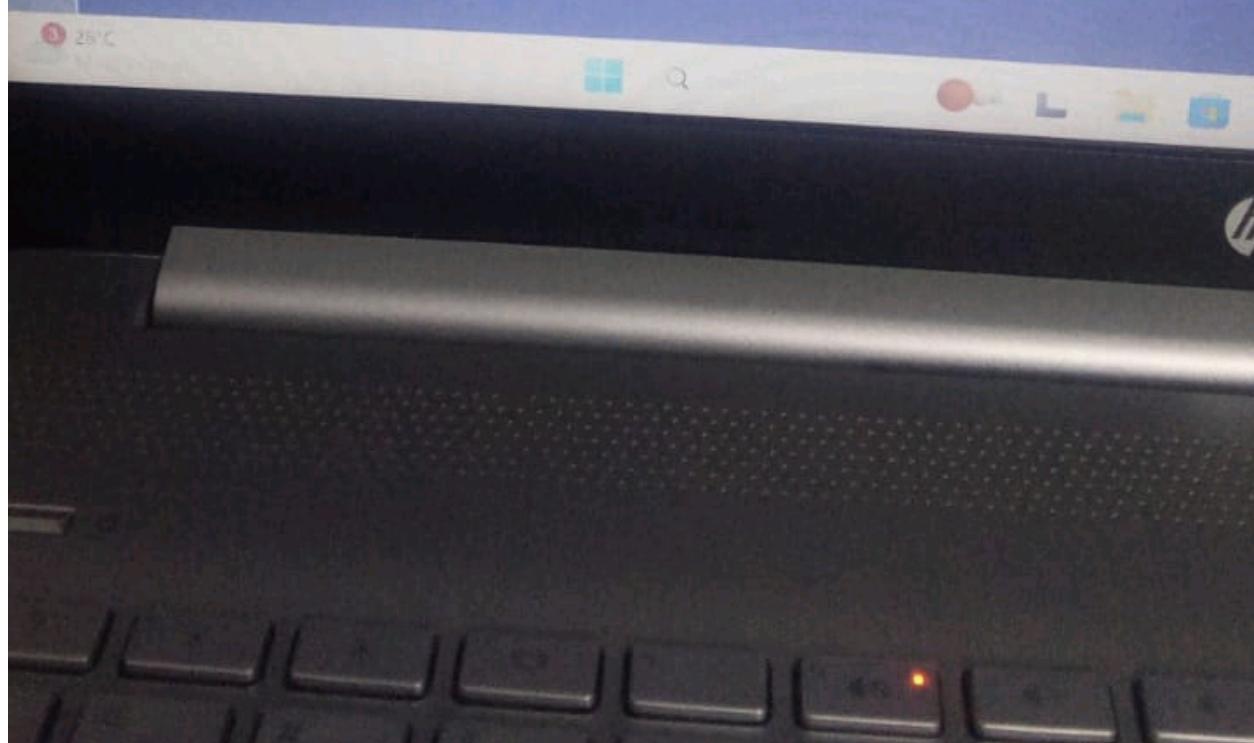
ID	Burst	Wait	Turnaround
3	3	0	0
2	6	3	9
1	24	9	33

Avg Wait: 4.00, Avg Turnaround: 14.00

```
==== Code Execution Successful ====
==== Session Ended. Please Run the code again ===
```

```
1 #include <stdio.h>
2 struct Process {
3     int id;
4     int burstTime;
5     int remainingTime;
6 };
7 void roundRobin(struct Process proc[], int n, int timeQuantum) {
8     int time = 0;
9     int completed = 0;
10    printf("\nProcess Execution Order:\n");
11    while (completed < n) {
12        for (int i = 0; i < n; i++) {
13            if (proc[i].remainingTime > 0) {
14                if (proc[i].remainingTime > timeQuantum) {
15                    time += timeQuantum;
16                    proc[i].remainingTime -= timeQuantum;
17                } else {
18                    time += proc[i].remainingTime;
19                    printf("P%d ", proc[i].id);
20                    proc[i].remainingTime = 0;
21                    completed++;
22                }
23            }
24        }
25    }
26    printf("\nTotal time taken: %d\n", time);
27 }
28 int main() {
```

```
19                     printf("P%d ", proc[i].id);
20                     proc[i].remainingTime = 0;
21                     completed++;
22                 }
23             }
24         }
25     }
26     printf("\nTotal time taken: %d\n", time);
27 }
28 int main() {
29     struct Process proc[10];
30     int n, timeQuantum;
31     printf("Enter number of processes: ");
32     scanf("%d", &n);
33     for (int i = 0; i < n; i++) {
34         proc[i].id = i + 1;
35         printf("Enter Burst Time for Process %d: ", proc[i].id);
36         scanf("%d", &proc[i].burstTime);
37         proc[i].remainingTime = proc[i].burstTime;
38     }
39     printf("Enter Time Quantum: ");
40     scanf("%d", &timeQuantum);
41     roundRobin(proc, n, timeQuantum);
42     return 0;
43 }
```



Run

Output

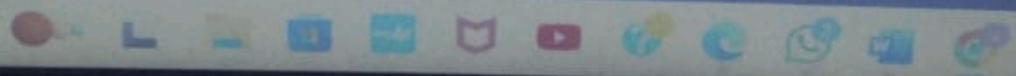
```
Enter number of processes: 3
Enter Burst Time for Process 1: 4
Enter Burst Time for Process 2:
6
Enter Burst Time for Process 3: 5
Enter Time Quantum: 34455
```

Process Execution Order:

P1 P2 P3

Total time taken: 15

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



Online C Compiler - Programiz

programiz.com/c-programming/online-compiler/

BLACK FRIDAY SALE Save 60% on PRO: Sale Ends Soon! [Claim My Discount](#)

Sale ends in 00d : 13hrs : 32mins : 02s

Programiz C Online Compiler

VIVARA SELECTION Oportunidade única para aumentar sua coleção.

JOIAS QUE ENCANTAM BLACK FRIDAY CONFIRA Programiz PRO >

main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/ipc.h>
4 #include <sys/shm.h>
5 #include <string.h>
6 #include <unistd.h>
7
8 #define SHM_SIZE 1024
9
10 int main() {
11     int shmid;
12     char *str;
13     shmid = shmget(IPC_PRIVATE, SHM_SIZE, IPC_CREAT | 0666);
14     str = (char *)shmat(shmid, NULL, 0);
15     strcpy(str, "Hello from the first process!");
16     sleep(2);
17     printf("Data read from shared memory: %s\n", str);
18     shmdt(str);
19     shmctl(shmid, IPC_RMID, NULL);
20 }
21 }
```

Output

Data read from shared memory: Hello from the first process!

== Code Execution Successful ==

79°F Mostly cloudy Search

23:57 27-11-2024 ENG IN

Sign in

Online C Compiler - Programiz

https://www.programiz.com/c-programming/online-compiler/

Programiz

C Online Compiler

Premium Coding Courses by Programiz

Programiz PRO

main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <sys/ipc.h>
5 #include <sys/msg.h>
6 #include <unistd.h>
7 #define MAX_TEXT 512
8 struct message {
9     long msg_type;
10    char text[MAX_TEXT];
11 };
12 int main() {
13     key_t key = ftok("progfile", 65);
14     int msgid = msgget(key, 0666 | IPC_CREAT);
15     struct message msg;
16     msg.msg_type = 1;
17     strcpy(msg.text, "Hello from Process 1!");
18     msgsnd(msgid, &msg, sizeof(msg), 0);
19     printf("Process 1: Sent message: %s\n", msg.text);
20     msgrcv(msgid, &msg, sizeof(msg), 1, 0);
21     printf("Process 2: Received message: %s\n", msg.text);
22     msgctl(msgid, IPC_RMID, NULL);
23     return 0;
24 }
```

Run

Output

Clear

Process 1: Sent message: Hello from Process 1!
Process 2: Received message: Hello from Process 1!
== Code Execution Successful ==

80°F Mostly cloudy

Search

16:17 28-11-2024

OS_LAB MANUAL[1].pdf | 42zry8jpd - C - OneCompiler | 42zrycz63 - C - OneCompiler +

https://onecompiler.com/c/42zrycz63

OneCompiler

Main.c + 42zrycz63

NEW C RUN ▶

```
1 #include <stdio.h>
2 #include <pthread.h>
3 void* threadFunction(void* arg)
4 {
5     char* message = (char*)arg; printf("%s\n",message);
6     return NULL;
7 }
8 int main() {
9     pthread_t thread1, thread2;
10    char* message1 = "Hello from Thread 1!";char*
11    message2 = "Hello from Thread 2!";
12    pthread_create(&thread1, NULL, threadFunction, (void*)message1);
13    pthread_create(&thread2, NULL, threadFunction, (void*)message2);
14    pthread_join(thread1, NULL);
15    pthread_join(thread2, NULL);
16    return 0;
17 }
```

STDIN
Input for the program (Optional)

Output:
Hello from Thread 1!
Hello from Thread 2!



Main.c

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4
5 #define NUM_PHILOSOPHERS 5
6
7 pthread_mutex_t forks[NUM_PHILOSOPHERS];
8 void* philosopher(void* num) {
9     int id = *(int*)num;
10    while (1) {
11        printf("Philosopher %d is thinking.\n", id);
12        usleep(1000);
13        pthread_mutex_lock(&forks[id]);
14        pthread_mutex_lock(&forks[(id + 1) % NUM_PHILOSOPHERS]);
15        printf("Philosopher %d is eating.\n", id);
16        usleep(2000);
17        pthread_mutex_unlock(&forks[(id + 1) % NUM_PHILOSOPHERS]);
18        pthread_mutex_unlock(&forks[id]);
19    }
20    return NULL;
21 }
22 int main() {
23     pthread_t threads[NUM_PHILOSOPHERS];
24     int philosopher_ids[NUM_PHILOSOPHERS];
25     for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
26         pthread_mutex_init(&forks[i], NULL);
27     }
28     for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
29         philosopher_ids[i] = i;
30         pthread_create(&threads[i], NULL, philosopher, &philosopher_ids[i]);
31     }
32     for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
33         pthread_join(threads[i], NULL);
34     }
35     for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
36         pthread_mutex_destroy(&forks[i]);
37     }
38     return 0;
39 }
```

42zs3j8q2

NEW

C ▾

RUN ▶



STDIN

Input for the program (Optional)

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

main.c



Clear

Output

```
First Fit Allocation:  
Process 1 allocated to Block 2  
Process 2 allocated to Block 5  
Process 3 allocated to Block 2  
Process 4 not allocated  
Best Fit Allocation:  
Process 1 allocated to Block 4  
Process 2 not allocated  
Process 3 allocated to Block 2  
Process 4 not allocated  
Worst Fit Allocation:  
Process 1 not allocated  
Process 2 not allocated  
Process 3 allocated to Block 3  
Process 4 not allocated
```

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

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3 void firstFit(int blockSize[], int m, int processSize[], int n) {  
4     int allocation[n];  
5     for (int i = 0; i < n; i++) {  
6         allocation[i] = -1;  
7     }  
8     for (int i = 0; i < n; i++) {  
9         for (int j = 0; j < m; j++) {  
10            if (blockSize[j] >= processSize[i]) {  
11                allocation[i] = j;  
12                blockSize[j] -= processSize[i];  
13                break;  
14            }  
15        }  
16    }  
17    printf("First Fit Allocation:\n");  
18    for (int i = 0; i < n; i++) {  
19        if (allocation[i] != -1) {  
20            printf("Process %d allocated to Block %d\n", i + 1,  
21                   allocation[i] + 1);  
22        } else {  
23            printf("Process %d not allocated\n", i + 1);  
24        }
```

main.c



Run

Clear

```
23     }
24 }
25 }
26 void bestFit(int blockSize[], int m, int processSize[], int n) {
27     int allocation[n];
28     for (int i = 0; i < n; i++) {
29         allocation[i] = -1;
30     }
31     for (int i = 0; i < n; i++) {
32         int bestIdx = -1;
33         for (int j = 0; j < m; j++) {
34             if (blockSize[j] >= processSize[i]) {
35                 if (bestIdx == -1 || blockSize[bestIdx] > blockSize[j])
36                     )
37                     bestIdx = j;
38                 }
39             }
40             if (bestIdx != -1) {
41                 allocation[i] = bestIdx;
42                 blockSize[bestIdx] -= processSize[i];
43             }
44 }
```

Output

```
First Fit Allocation:
Process 1 allocated to Block 2
Process 2 allocated to Block 5
Process 3 allocated to Block 2
Process 4 not allocated
Best Fit Allocation:
Process 1 allocated to Block 4
Process 2 not allocated
Process 3 allocated to Block 2
Process 4 not allocated
Worst Fit Allocation:
Process 1 not allocated
Process 2 not allocated
Process 3 allocated to Block 3
Process 4 not allocated
```

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

```
main.c
45     printf("Best Fit Allocation:\n");
46     for (int i = 0; i < n; i++) {
47         if (allocation[i] != -1) {
48             printf("Process %d allocated to Block %d\n", i + 1,
49                   allocation[i] + 1);
50         } else {
51             printf("Process %d not allocated\n", i + 1);
52         }
53     }
54     void worstFit(int blockSize[], int m, int processSize[], int n) {
55         int allocation[n];
56         for (int i = 0; i < n; i++) {
57             allocation[i] = -1;
58         }
59         for (int i = 0; i < n; i++) {
60             int worstIdx = -1;
61             for (int j = 0; j < m; j++) {
62                 if (blockSize[j] >= processSize[i]) {
63                     if (worstIdx == -1 || blockSize[worstIdx] <
64                         blockSize[j]) {
```

Run

Output

Clear

```
First Fit Allocation:
Process 1 allocated to Block 2
Process 2 allocated to Block 5
Process 3 allocated to Block 2
Process 4 not allocated
Best Fit Allocation:
Process 1 allocated to Block 4
Process 2 not allocated
Process 3 allocated to Block 2
Process 4 not allocated
Worst Fit Allocation:
Process 1 not allocated
Process 2 not allocated
Process 3 allocated to Block 3
Process 4 not allocated
```

--- Code Execution Successful ---

main.c



```
64         blockSize[j]) {
65             worstIdx = j;
66         }
67     }
68     if (worstIdx != -1) {
69         allocation[i] = worstIdx;
70         blockSize[worstIdx] -= processSize[i];
71     }
72 }
73 printf("Worst Fit Allocation:\n");
74 for (int i = 0; i < n; i++) {
75     if (allocation[i] != -1) {
76         printf("Process %d allocated to Block %d\n",
77               i + 1,
78               allocation[i] + 1);
79     } else {
80         printf("Process %d not allocated\n",
81               i + 1);
82 }
83 int main() {
84     int blockSize[] = {100, 500, 200, 300, 600};
85     int processSize[] = {212, 417, 112, 426};
```

Output

```
First Fit Allocation:
Process 1 allocated to Block 2
Process 2 allocated to Block 5
Process 3 allocated to Block 2
Process 4 not allocated
Best Fit Allocation:
Process 1 allocated to Block 4
Process 2 not allocated
Process 3 allocated to Block 2
Process 4 not allocated
Worst Fit Allocation:
Process 1 not allocated
Process 2 not allocated
Process 3 allocated to Block 3
Process 4 not allocated
```

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

Clear

main.c



Share

Run

Clear

```
71     }
72 }
73 printf("Worst Fit Allocation:\n");
74 for (int i = 0; i < n; i++) {
75     if (allocation[i] != -1) {
76         printf("Process %d allocated to Block %d\n", i + 1,
77             allocation[i] + 1);
78     } else {
79         printf("Process %d not allocated\n", i + 1);
80     }
81 }
82 int main() {
83     int blockSize[] = {100, 500, 200, 300, 600};
84     int processSize[] = {212, 417, 112, 426};
85     int m = sizeof(blockSize) / sizeof(blockSize[0]);
86     int n = sizeof(processSize) / sizeof(processSize[0]);
87     firstFit(blockSize, m, processSize, n);
88     bestFit(blockSize, m, processSize, n);
89     worstFit(blockSize, m, processSize, n);
90     return 0;
91 }
92 }
```

Output

```
* First Fit Allocation:
Process 1 allocated to Block 2
Process 2 allocated to Block 5
Process 3 allocated to Block 2
Process 4 not allocated
Best Fit Allocation:
Process 1 allocated to Block 4
Process 2 not allocated
Process 3 allocated to Block 2
Process 4 not allocated
Worst Fit Allocation:
Process 1 not allocated
Process 2 not allocated
Process 3 allocated to Block 3
Process 4 not allocated
```

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



```
main.c

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <fcntl.h>
4 #include <unistd.h>
5 #define BUFFER_SIZE 4096
6 void copy(){
7 const char *sourcefile= "C:/Users/itssk/OneDrive/Desktop/sasi.txt";
8 const char *destination_file="C:/Users/itssk/OneDrive/Desktop/sk.txt";int
9 source_fd = open(sourcefile, O_RDONLY );
10 int dest_fd = open(destination_file, O_WRONLY | O_CREAT |O_TRUNC, 0666);
11 char buffer[BUFFER_SIZE]; ssize_t
12 bytesRead, bytesWritten;
13 while ((bytesRead = read(source_fd, buffer, BUFFER_SIZE)) > 0) {
14 bytesWritten = write(dest_fd, buffer, bytesRead);
15 }
16 close(source_fd);
17 close(dest_fd);
18 printf("File copied successfully.\n");
19 }
20 void create()
21 {
22 char path[100];
23 FILE *fp; fp=fopen("C:/Users/itssk/OneDrive/Desktop/sasi.txt","w");
24 printf("file created successfully");
25 }
26 int main(){
27 int n;
28 printf("1. Create \t2. Copy \t3. Delete\nEnter your choice: ");
29 scanf("%d",&n);
30 switch(n){
31 case 1:
32 create(); break;
33 case 2:
34 copy();
35 break;
36 case 3:
37 remove("C:/Users/itssk/OneDrive/Desktop/sasi.txt");
38 printf("Deleted successfully");
39 }}
```

Copy

Delete

Enter your choice: 2
File copied successfully.

*** Code Execution Successful ***

Run

Output

Programs C Online Compiler

Programz PRO >

main.c

Output

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <dirent.h>
5 #define MAX_DIRS 10
6 #define MAX_FILES 10
7 #define DIR_NAME_LENGTH 50
8 #define FILE_NAME_LENGTH 50
9 typedef struct {
10     char files[MAX_FILES][FILE_NAME_LENGTH];
11     int file_count;
12 } SubDirectory;
13 typedef struct {
14     char name[DIR_NAME_LENGTH];
15     SubDirectory subdirs[MAX_DIRS];
16     int subdir_count;
17 } MainDirectory;
18 void create_directory(MainDirectory *main_dir, const char *dir_name) {
19     if (main_dir->subdir_count < MAX_DIRS) {
20         strcpy(main_dir->subdirs[main_dir->subdir_count].files[0], dir_name);
21         main_dir->subdirs[main_dir->subdir_count].file_count = 0;
22         main_dir->subdir_count++;
23         printf("Directory '%s' created successfully.\n", dir_name);
24     } else {
25         printf("Maximum number of directories reached.\n");
26     }
27 }
28 void add_file(MainDirectory *main_dir, const char *dir_name, const char *file_name) {
29     for (int i = 0; i < main_dir->subdir_count; i++) {
30         if (strcmp(main_dir->subdirs[i].files[0], dir_name) == 0) {
31             if (main_dir->subdirs[i].file_count < MAX_FILES) {
32                 strcpy(main_dir->subdirs[i].files[main_dir->subdirs[i].file_count + 1], file_name);
33                 main_dir->subdirs[i].file_count++;
34                 printf("File '%s' added to directory '%s'.\n", file_name, dir_name);
35             } else {
36                 printf("Maximum number of files in directory '%s' reached.\n", dir_name);
37             }
38         }
39     }
40 }
41 printf("Directory '%s' not found.\n", dir_name);
42 }
43 void display_structure(MainDirectory *main_dir) {
44     printf("Directory Structure:\n");
45     for (int i = 0; i < main_dir->subdir_count; i++) {
46         printf("Directory: %s\n", main_dir->subdirs[i].files[0]);
47         for (int j = 1; j <= main_dir->subdirs[i].file_count; j++) {
48             printf("  File: %s\n", main_dir->subdirs[i].files[j]);
49         }
50     }
51 }
52 int main() {
53     MainDirectory main_dir = { .subdir_count = 0 };
54     create_directory(&main_dir, "Documents");
55     add_file(&main_dir, "Documents", "Resume.docx");
56     add_file(&main_dir, "Documents", "CoverLetter.docx");
57     add_file(&main_dir, "Pictures", "Vacation.jpg");
58     display_structure(&main_dir);
59 }
```

Directory 'Documents' created successfully.
Directory 'Pictures' created successfully.
File 'Resume.docx' added to directory 'Documents'.
File 'CoverLetter.docx' added to directory 'Documents'.
File 'Vacation.jpg' added to directory 'Pictures'.
Directory Structure:
Directory: Documents
File: Resume.docx
File: CoverLetter.docx
Directory: Pictures
File: Vacation.jpg

--- Code Execution Successful ---

```
main.c | 10  char file[MAX_FILES][FILE_NAME_LENGTH];
11  int file_count;
12 } SubDirectory;
13 typedef struct {
14  char name[DIR_NAME_LENGTH];
15  SubDirectory subdirs[MAX_DIRS];
16  int subdir_count;
17 } MainDirectory;
18 void create_directory(MainDirectory *main_dir, const char *dir_name) {
19  if (main_dir->subdir_count < MAX_DIRS) {
20      strcpy(main_dir->subdirs[main_dir->subdir_count].files[0], dir_name);
21      main_dir->subdirs[main_dir->subdir_count].file_count = 0;
22      main_dir->subdir_count++;
23      printf("Directory '%s' created successfully.\n", dir_name);
24  } else {
25      printf("Maximum number of directories reached.\n");
26  }
27 }
28 void add_file(MainDirectory *main_dir, const char *dir_name, const char *file_name) {
29  for (int i = 0; i < main_dir->subdir_count; i++) {
30      if (strcmp(main_dir->subdirs[i].files[0], dir_name) == 0) {
31          if (main_dir->subdirs[i].file_count < MAXFILES) {
32              strcpy(main_dir->subdirs[i].files[main_dir->subdirs[i].file_count + 1], file_name);
33              main_dir->subdirs[i].file_count++;
34              printf("File '%s' added to directory '%s'.\n", file_name, dir_name);
35          } else {
36              printf("Maximum number of files in directory '%s' reached.\n", dir_name);
37          }
38      }
39  }
40 }
41 printf("Directory '%s' not found.\n", dir_name);
42 }
43 void display_structure(MainDirectory *main_dir) {
44  printf("Directory Structure:\n");
45  for (int i = 0; i < main_dir->subdir_count; i++) {
46      printf("Directory: %s\n", main_dir->subdirs[i].files[0]);
47      for (int j = 1; j <= main_dir->subdirs[i].file_count; j++) {
48          printf(" File: %s\n", main_dir->subdirs[i].files[j]);
49      }
50  }
51 }
52 int main() {
53  MainDirectory mainDir = { .subdir_count = 0 };
54  create_directory(&mainDir, "Documents");
55  create_directory(&mainDir, "Pictures");
56  add_file(&mainDir, "Documents", "Resume.docx");
57  add_file(&mainDir, "Documents", "CoverLetter.docx");
58  add_file(&mainDir, "Pictures", "Vacation.jpg");
59  display_structure(&mainDir);
60  return 0;
61 }
```

Output

```
▲ Directory 'Documents' created successfully.
  Directory 'Pictures' created successfully.
  File 'Resume.docx' added to directory 'Documents'.
  File 'CoverLetter.docx' added to directory 'Documents'.
  File 'Vacation.jpg' added to directory 'Pictures'.
  Directory Structure:
    Directory: Documents
      File: Resume.docx
      File: CoverLetter.docx
    Directory: Pictures
      File: Vacation.jpg

*** Code Execution Successful ***
```

programiz.com/c-programming/online-compiler/

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #define MAX_NAME_LENGTH 100
5 #define FILE_NAME "employees.dat"
6 typedef struct {
7     int id;
8     char name[MAX_NAME_LENGTH];
9     float salary;
10 }
11 Employee;
12 void addEmployee() {
13     FILE *file = fopen(FILE_NAME, "ab");
14     Employee emp;
15     printf("Enter ID, Name, Salary: ");
16     scanf("%d %s %f", &emp.id, emp.name, &emp.salary);
17     fwrite(&emp, sizeof(Employee), 1, file);
18     fclose(file);
19 }
20 void displayEmployees() {
21     FILE *file = fopen(FILE_NAME, "rb");
22     Employee emp;
23     printf("ID\tName\tSalary\n");
24     while (fread(&emp, sizeof(Employee), 1, file)) {
25         printf("%d\t%s\t%.2f\n", emp.id, emp.name, emp.salary);
26     }
27     fclose(file);
28 }
29 int main() {
30     int choice;
31     do {
32         printf("\n1. Add Employee\n2. Display Employees\n3. Exit\n");
33         printf("Choose an option: ");
34         scanf("%d", &choice);
35         switch (choice) {
36             case 1: addEmployee(); break;
37             case 2: displayEmployees(); break;
38             case 3: exit(0);
39             default: printf("Invalid choice!\n");
40         }
41     } while (!);
42     return 0;
43 }
```

1. Add Employee
2. Display Employees
3. Exit

Choose an option: 3

--- Code Execution Successful ---



Search

ENG
IN20:02
02-12-2024

```
1 #include <stdio.h>
2 #define P 5
3 #define R 3
4 int available[R] = {3, 3, 2};
5 int maximum[P][R] = {
6     {7, 5, 3},
7     {3, 2, 2},
8     {9, 0, 2},
9     {2, 2, 2},
10    {4, 3, 3}
11 };
12 int allocation[P][R] = {
13     {0, 1, 0},
14     {2, 0, 0},
15     {3, 0, 2},
16     {2, 1, 1},
17     {0, 0, 2}
18 };
19 int is_safe() {
20     int work[R];
21     int finish[P] = {0};
22     for (int i = 0; i < R; i++) work[i] = available[i];
23     for (int count = 0; count < P;) {
24         int found = 0;
25         for (int i = 0; i < P; i++) {
26             if (!finish[i]) {
27                 int j;
28                 for (j = 0; j < R; j++)
29                     if (maximum[i][j] - allocation[i][j] > work[j]) break;
30                 if (j == R) {
31                     for (int k = 0; k < R; k++) work[k] += allocation[i][k];
32                     finish[i] = 1; count++; found = 1;
33                 }
34             }
35         }
36         if (!found) return 0;
37     }
38     return 1;
39 }
40 int request_resources(int process_num, int request[]) {
41     for (int i = 0; i < R; i++) {
42         if (request[i] > available[i] || request[i] > maximum[process_num][i] - allocation[process_num][i])
```

```
Enter process number (0 to 4): 2
Enter resource request (e.g., 0 1 0): 1
```

```
0
```

```
1
```

```
Request denied. System is not in safe state.
```

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

```
35     }
36     if (!found) return 0;
37   }
38   return 1;
39 }

40 int request_resources(int process_num, int request[]) {
41   for (int i = 0; i < R; i++) {
42     if (request[i] > available[i] || request[i] > maximum[process_num][i] - allocation[process_num][i])
43       return 0;
44   }
45   for (int i = 0; i < R; i++) {
46     available[i] -= request[i];
47     allocation[process_num][i] += request[i];
48   }
49   if (is_safe()) return 1;
50   for (int i = 0; i < R; i++) {
51     available[i] += request[i];
52     allocation[process_num][i] -= request[i];
53   }
54   return 0;
55 }

56 int main() {
57   int process_num, request[R];
58   printf("Enter process number (0 to 4): ");
59   scanf("%d", &process_num);
60   printf("Enter resource request (e.g., 0 1 0): ");
61   for (int i = 0; i < R; i++) scanf("%d", &request[i]);
62   if (request_resources(process_num, request))
63     printf("Request granted.\n");
64   else
65     printf("Request denied. System is not in safe state.\n");
66
67   return 0;
68 }
```

main.c



Run

Clear

Output

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <semaphore.h>
5 #include <unistd.h>
6 #define BUFFER_SIZE 5
7 int buffer[BUFFER_SIZE], in = 0, out = 0;
8 sem_t empty, full;
9 void* producer(void* arg) {
10     for (int i = 0; i < 10; i++) {
11         sem_wait(&empty);
12         buffer[in] = i;
13         printf("Produced: %d\n", i);
14         in = (in + 1) % BUFFER_SIZE;
15         sem_post(&full);
16         sleep(1);
17     }
18     return NULL;
19 }
20 void* consumer(void* arg) {
21     for (int i = 0; i < 10; i++) {
22         sem_wait(&full);
23         int item = buffer[out];
24         printf("Consumed: %d\n", item);
25         out = (out + 1) % BUFFER_SIZE;
26         sem_post(&empty);
27         sleep(1);
28     }
29     return NULL;
30 }
31 int main() {
32     pthread_t prod, cons;
33     sem_init(&empty, 0, BUFFER_SIZE);
```

```
produced:1
Consumed:1
produced:2
Consumed:2
produced:3
consumed:3
produced:4
consumed:4
produced:5
consumed:5
produced:6
consumed:6
produced:7
consumed:7
produced:8
consumed:8
produced:9
consumed:9
produced:10
consumed:10
```

```
== Session Ended. Please Run the code again ==
```

main.c



Run

Output

Clear

```
11     sem_wait(&empty);
12     buffer[in] = i;
13     printf("Produced: %d\n", i);
14     in = (in + 1) % BUFFER_SIZE;
15     sem_post(&full);
16     sleep(1);
17 }
18 return NULL;
19 }

20 void* consumer(void* arg) {
21     for (int i = 0; i < 10; i++) {
22         sem_wait(&full);
23         int item = buffer[out];
24         printf("Consumed: %d\n", item);
25         out = (out + 1) % BUFFER_SIZE;
26         sem_post(&empty);
27         sleep(1);
28     }
29     return NULL;
30 }

31 int main() {
32     pthread_t prod, cons;
33     sem_init(&empty, 0, BUFFER_SIZE);
34     sem_init(&full, 0, 0);
35     pthread_create(&prod, NULL, producer, NULL);
36     pthread_create(&cons, NULL, consumer, NULL);
37     pthread_join(prod, NULL);
38     pthread_join(cons, NULL);
39     sem_destroy(&empty);
40     sem_destroy(&full);
41     return 0;
42 }
43 }
```

```
produced:1
consumed:1
produced:2
consumed:2
produced:3
consumed:3
produced:4
consumed:4
produced:5
consumed:5
produced:6
consumed:6
produced:7
consumed:7
produced:8
consumed:8
produced:9
consumed:9
produced:10
consumed:10
```

--- Session Ended. Please Run the code again ---

OS_LAB MANUAL[1].pdf x 42zuxvb3u - C - OneCompiler x +

https://onecompiler.com/c/42zuxvb3u

OneCompiler

Main.c + 42zuxvb3u

NEW C RUN ▶

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <semaphore.h>
4 #include <unistd.h>
5 sem_t mutex, writeBlock;
6 int data = 0, readersCount = 0;
7 void *reader(void *arg) {
8    for (int i = 0; i < 5; i++) {
9        sem_wait(&mutex);
10       readersCount++;
11       if (readersCount == 1) sem_wait(&writeBlock);
12       sem_post(&mutex);
13       printf("Reader reads data: %d\n", data);
14       usleep(100);
15       sem_wait(&mutex);
16       readersCount--;
17       if (readersCount == 0) sem_post(&writeBlock);
18       sem_post(&mutex);
19       usleep(100);
20    }
21    return NULL;
22 }
23 void *writer(void *arg) {
24    for (int i = 0; i < 5; i++) {
25        sem_wait(&writeBlock);
26        data++;
27        printf("Writer writes data: %d\n", data);
28        sem_post(&writeBlock);
29        usleep(100);
30    }
31    return NULL;
32 }
33 int main() {
34    pthread_t readers[3], writerThread;
35    sem_init(&mutex, 0, 1);
36    sem_init(&writeBlock, 0, 1);
37    for (int i = 0; i < 3; i++) pthread_create(&readers[i], NULL, reader, NULL);
38    pthread_create(&writerThread, NULL, writer, NULL);
39    for (int i = 0; i < 3; i++) pthread_join(readers[i], NULL);
40    pthread_join(writerThread, NULL);
41    sem_destroy(&mutex);
42    sem_destroy(&writeBlock);
43 }
44 }
```

STDIN

Input for the program (Optional)

Output:

Reader reads data: 0
Reader reads data: 0
Reader reads data: 0
Writer writes data: 1
Reader reads data: 1
Reader reads data: 1
Reader reads data: 1
Writer writes data: 2
Reader reads data: 2
Reader reads data: 2
Reader reads data: 2
Writer writes data: 3
Reader reads data: 3
Reader reads data: 3
Reader reads data: 3
Writer writes data: 4
Reader reads data: 4
Reader reads data: 4
Reader reads data: 4
Writer writes data: 5

28°C Mostly cloudy

Search

20:19 ENG IN 02-12-2024 PRE

OS_LAB MANUAL[1].pdf x 42zuxvb3u - C - OneCompiler x +

https://onecompiler.com/c/42zuxvb3u

OneCompiler

Main.c + 42zuxvb3u

```
1 #include <stdio.h>
2 #include <pthread.h>
3 int counter = 0;
4 pthread_mutex_t mutex;
5 void *threadFunction(void *arg)
6 {
7     int i;
8     for (i = 0; i < 1000000; ++i)
9     {
10    return NULL;
11 }
12 int main()
13 {
14     pthread_mutex_init(&mutex, NULL);
15     pthread_t thread1, thread2;
16     pthread_create(&thread1, NULL, threadFunction, NULL);
17     pthread_create(&thread2, NULL, threadFunction, NULL);
18     pthread_join(thread1, NULL);
19     pthread_join(thread2, NULL);
20     pthread_mutex_destroy(&mutex);
21     printf("Final counter value: %d\n", counter);
22     return 0;
23 }
```

STDIN
20000000

Output:
Final counter value: 0

28°C Mostly cloudy Search

20:26 02-12-2024 ENG IN

OS_LAB MANUAL[1].pdf x 42zuxvb3u - C - OneCompiler x +

https://onecompiler.com/c/42zuxvb3u

OneCompiler

Main.c + 42zuxvb3u

```
1 #include <stdio.h>
2 #include <pthread.h>
3 int counter = 0;
4 pthread_mutex_t mutex;
5 void *threadFunction(void *arg)
6 {
7     int i;
8     for (i = 0; i < 1000000; ++i)
9     {
10    return NULL;
11 }
12 int main()
13 {
14     pthread_mutex_init(&mutex, NULL);
15     pthread_t thread1, thread2;
16     pthread_create(&thread1, NULL, threadFunction, NULL);
17     pthread_create(&thread2, NULL, threadFunction, NULL);
18     pthread_join(thread1, NULL);
19     pthread_join(thread2, NULL);
20     pthread_mutex_destroy(&mutex);
21     printf("Final counter value: %d\n", counter);
22     return 0;
23 }
```

STDIN
20000000

Output:
Final counter value: 0

28°C Mostly cloudy Search ENG IN 20:26 02-12-2024