

main.c

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5
6 void* threadFunction(void* arg) {
7     printf("Thread %ld is running\n", pthread_self());
8     pthread_exit(NULL);
9 }
10
11 int main() {
12     pthread_t thread1, thread2;
13     int res;
14
15     // (i) Create threads
16     res = pthread_create(&thread1, NULL, threadFunction, NULL);
17     if (res != 0) {
18         perror("Thread creation failed");
19         return 1;
20     }
```

Run

Output

Thread 137067147384512 is running
Thread 137067138991808 is running
Threads are not equal

=== Code Execution Successful ===

Clear

main.c

 Share

Run

Output

Clear

```

1  #include <stdio.h>
2  #include <pthread.h>
3  #include <semaphore.h>
4  #include <unistd.h>
5
6  #define N 5 // Number of philosophers
7  #define THINKING 0
8  #define HUNGRY 1
9  #define EATING 2
10
11  sem_t mutex;
12  sem_t S[N];
13  int state[N];
14  int phil[N] = {0, 1, 2, 3, 4};
15
16 void test(int philID) {
17     if (state[philID] == HUNGRY && state[(philID + 4) % N] !=
        EATING && state[(philID + 1) % N] != EATING) {
18         state[philID] = EATING;
19         printf("Philosopher %d is Eating\n", philID + 1);

```

```
Philosopher 1 is Thinking
Philosopher 2 is Thinking
Philosopher 3 is Thinking
Philosopher 4 is Thinking
Philosopher 5 is Thinking
Philosopher 1 is Hungry
Philosopher 1 is Eating
```

Google Classroom - Login x Day 3 Lab Programs x Worst Fit Algorithm C x Online C Compiler - Programiz x +

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

Programiz
C Online Compiler

Premium Coding Courses by Programiz

Programiz PRO

Programiz PRO >

main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 void grep(const char *filename, const char *searchTerm) {
6     FILE *file = fopen(filename, "r");
7     if (file == NULL) {
8         perror("Error opening file");
9         return;
10    }
11
12    char line[1024];
13    int lineNumber = 1;
14    while (fgets(line, sizeof(line), file)) {
15        if (strstr(line, searchTerm)) {
16            printf("%s:%d: %s", filename, lineNumber, line);
17        }
18        lineNumber++;
19    }
20}
```

Run

Output

Clear

Usage: /tmp/ub09VeE5Sd/main.o <search_term> <file>

=== Code Exited With Errors ===

32°C Haze

Search

ENG IN

15:48 17-02-2025



main.c

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

Run

Output

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

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

Clear



main.c

```
12 struct stat fileStat;  
13 DIR *dir;  
14 struct dirent *entry;  
15  
16 // Creating and opening a file  
17 fd = open("testfile.txt", O_CREAT | O_WRONLY, 0644);  
18 if (fd < 0) {  
19     perror("Error opening file");  
20     return 1;  
21 }  
22  
23 // Writing to the file  
24 write(fd, text, strlen(text));  
25 close(fd);  
26  
27 // Reopening file for reading  
28 fd = open("testfile.txt", O_RDONLY);  
29 if (fd < 0) {  
30     perror("Error opening file");  
31     return 1;  
}
```

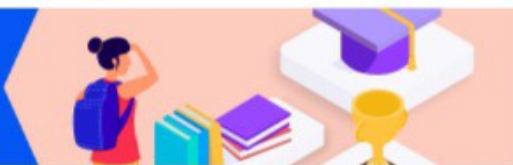
Run

Output

Error opening file: Permission denied

=== Code Exited With Errors ===

Clear



main.c



Run

Output

Clear

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <string.h>
5 #include <sys/stat.h>
6 #include <dirent.h>
7
8 int main() {
9     int fd;
10    char buffer[100];
11    char *text = "Hello, this is a test file.";
12    struct stat fileStat;
13    DIR *dir;
14    struct dirent *entry;
15
16    // Creating and opening a file
17    fd = open("testfile.txt", O_CREAT | O_WRONLY, 0644);
18    if (fd < 0) {
19        perror("Error opening file");
20        return 1;
21    }
```

Error opening file: Permission denied

=== Code Exited With Errors ===



main.c

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <string.h>
5
6 int main() {
7     int fd;
8     char buffer[100];
9     char *text = "Hello, this is a test file.";
10
11     // Creating and opening a file
12     fd = open("testfile.txt", O_CREAT | O_WRONLY, 0644);
13     if (fd < 0) {
14         perror("Error opening file");
15         return 1;
16     }
17
18     // Writing to the file
19     write(fd, text, strlen(text));
20     close(fd);
```

Run

Output

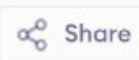
Error opening file: Permission denied

=== Code Exited With Errors ===

Clear



main.c



Run

Output

Clear

```
1 #include <stdio.h>
2
3 void firstFit(int blockSize[], int m, int processSize[], int n)
4 {
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        for (int j = 0; j < m; j++) {
12            if (blockSize[j] >= processSize[i]) {
13                allocation[i] = j;
14                blockSize[j] -= processSize[i];
15                break;
16            }
17        }
18    }
19 }
```

Process No.	Process Size	Block No.
1	212	2
2	417	5
3	112	2
4	426	Not Allocated

=== Code Execution Successful ===

main.c

```
1 #include <stdio.h>
2
3 void bestFit(int blockSize[], int m, int processSize[], int n)
4 {
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 bestIdx = -1;
12        for (int j = 0; j < m; j++) {
13            if (blockSize[j] >= processSize[i]) {
14                if (bestIdx == -1 || blockSize[j] <
15                    blockSize[bestIdx]) {
16                    bestIdx = j;
17                }
18            }
19        }
20    }
21 }
```

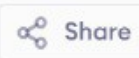
Output

Process No.	Process Size	Block No.
1	212	4
2	417	2
3	112	3
4	426	5

=== Code Execution Successful ===



main.c



Run

Output

Clear

```
1 #include <stdio.h>
2
3 void worstFit(int blockSize[], int m, int processSize[], int n)
4 {
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    }
21 }
```

Process No.	Process Size	Block No.
1	212	5
2	417	2
3	112	5
4	426	Not Allocated

=== Code Execution Successful ===