




```
main.c
1 #include<stdio.h>
2 #include<stdlib.h>
3 struct Employee { int empId; char empName[50]; float empSalary; };
4 int main() {
5     FILE *filePtr = fopen("employee.dat", "rb+");
6     if (!filePtr) filePtr = fopen("employee.dat", "wb+");
7     struct Employee emp; int choice;
8     do {
9         printf("\n1.Add 2.Display 3.Update 4.Exit: ");
10        scanf("%d", &choice);
11        if (choice == 1) { scanf("%d %s %f", &emp.empId, emp.empName, &emp.empSalary); fseek(filePtr, (emp.empId - 1) * sizeof(emp), SEEK_SET); fwrite(&emp, sizeof(emp), 1, filePtr); }
12        else if (choice == 2) { scanf("%d", &emp.empId); fseek(filePtr, (emp.empId - 1) * sizeof(emp), SEEK_SET); fread(&emp, sizeof(emp), 1, filePtr); printf("%d %s %.2f\n", emp.empId, emp.empName, emp.empSalary); }
13        else if (choice == 3) { scanf("%d %s %f", &emp.empId, emp.empName, &emp.empSalary); fseek(filePtr, (emp.empId - 1) * sizeof(emp), SEEK_SET); fwrite(&emp, sizeof(emp), 1, filePtr); }
14    } while (choice != 4);
15    fclose(filePtr);
16    return 0;
17 }
18
```

Output

1.Add 2.Display 3.Update 4.Exit:

main.c	   Share	Run	Output
<pre>1 #include&lt;stdio.h&gt; 2 #include&lt;stdlib.h&gt; 3 #include&lt;unistd.h&gt; 4 #include&lt;pthread.h&gt; 5 void *myThreadFun(void *vargp) 6 { 7     sleep(1); 8     printf("Printing from Thread\n"); 9     return NULL; 10 } 11 int main() 12 { 13     pthread_t thread_id; 14     printf("Before Thread\n"); 15     pthread_create(&amp;thread_id, NULL, myThreadFun, NULL); 16     pthread_join(thread_id, NULL); 17     printf("After Thread\n"); 18     exit(0); 19 }</pre>			<pre>Before Thread Printing from Thread After Thread  === Code Execution Successful ===</pre>

main.c



Share

Run

Output

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<pthread.h>
4 #include<semaphore.h>
5 #include<unistd.h>
6 sem_t room;
7 sem_t chopstick[5];
8 void * philosopher(void *);
9 void eat(int);
10 int main()
11 {
12     int i,a[5];
13     pthread_t tid[5];
14     sem_init(&room,0,4);
15     for(i=0;i<5;i++)
16         sem_init(&chopstick[i],0,1);
17     for(i=0;i<5;i++){
18         a[i]=i;
19         pthread_create(&tid[i],NULL,philosopher,(void *)&a[i]);
20     }
21     for(i=0;i<5;i++)
22         pthread_join(tid[i],NULL);
23 }
24 void * philosopher(void * num)
25 {
26     int phil=*(int *)num;
27     sem_wait(&room);
28     printf("\nPhilosopher %d has entered room",phil);
29     sem_wait(&chopstick[phil]);
30     sem_wait(&chopstick[(phil+1)%5]);
31     eat(phil);
32     sleep(2);
33     printf("\nPhilosopher %d has finished eating",phil);
34     sem_post(&chopstick[(phil+1)%5]);
35     sem_post(&chopstick[phil]);
36     sem_post(&room);
37 }
38 void eat(int phil)
39 {
40     printf("\nPhilosopher %d is eating",phil);
41 }
```

Philosopher 0 has entered room  
Philosopher 0 is eating  
Philosopher 1 has entered room  
Philosopher 2 has entered room  
Philosopher 2 is eating  
Philosopher 3 has entered room  
Philosopher 0 has finished eating  
Philosopher 4 has entered room  
Philosopher 4 is eating  
Philosopher 2 has finished eating  
Philosopher 1 is eating  
Philosopher 4 has finished eating  
Philosopher 3 is eating  
Philosopher 1 has finished eating  
Philosopher 3 has finished eating

=== Code Execution Successful ===

main.c

Share

Run

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 int mutex = 1, full = 0, empty = 3, x = 0;
4 int wait(int s) {
5     return (--s);
6 }
7 int signal(int s) {
8     return (++s);
9 }
10 void producer() {
11     mutex = wait(mutex);
12     full = signal(full);
13     empty = wait(empty);
14     x++;
15     printf("\nProducer produces item %d", x);
16     mutex = signal(mutex);
17 }
18 void consumer() {
19     mutex = wait(mutex);
20     full = wait(full);
21     empty = signal(empty);
22     printf("\nConsumer consumes item %d", x);
23     x--;
24     mutex = signal(mutex);
25 }
26 int main() {
27     int n;
28     printf("\n1. Producer\n2. Consumer\n3. Exit");
29     while (1) {
30         printf("\nEnter your choice: ");
31         scanf("%d", &n);
32         switch (n) {
33             case 1:
34                 if ((mutex == 1) && (empty != 0))
35                     producer();
36                 else
37                     printf("Buffer is full!!");
38                 break;
39             case 2:
40                 if ((mutex == 1) && (full != 0))
41                     consumer();
42                 else
43                     printf("Buffer is empty!!");
44                 break;
45             case 3:
46                 exit(0);
47         }
48     }
49 }
```

Output

1. Producer  
2. Consumer  
3. Exit  
Enter your choice:



main.c

Share

Run

1 #include <stdio.h>  
2 #include <stdlib.h>  
3 #include <string.h>  
4 int main() {  
5 char mainDirectory[] = "C:/Users/itsak/OneDrive/Desktop";  
6 char subDirectory[] = "00";  
7 char fileName[] = "example.txt";  
8 char filePath[300];  
9 char mainDirPath[400];  
10 sprintf(mainDirPath, sizeof(mainDirPath), "%s/%s/", mainDirectory, subDirectory);  
11 sprintf(filePath, sizeof(filePath), "%s%s", mainDirPath, fileName);  
12 FILE \*file = fopen(filePath, "w");  
13 if (file == NULL) {  
14 printf("Error creating file.\n");  
15 return 1;  
16 }  
17 fprintf(file, "This is an example file content.");  
18 printf("File created successfully: %s\n", filePath);  
19 }

Output

Clear

Error creating file.  
  
--- Code Exited With Errors ---

```

26         break;
27     }
28 }
29 if (bytesRead < 0) {
30     perror("Error reading from source file");
31 }
32 close(source_fd);
33 close(dest_fd);
34 printf("File copied successfully.\n");
35 }
36 void create() {
37     const char *file_path = "C:/Users/itsik/OneDrive/Desktop/ssi.txt";
38     FILE *fp = fopen(file_path, "w");
39     if (fp == NULL) {
40         perror("Error creating file");
41         return;
42     }
43     fprintf(fp, ""); // Create an empty file
44     fclose(fp);
45     printf("File created successfully.\n");
46 }
47 int main() {
48     int n;
49     printf("1. Create 2. Copy 3. Delete\nEnter your choice: ");
50     scanf("%d", &n);
51     switch (n) {
52     case 1:
53         create();
54         break;
55     case 2:
56         copy();
57         break;
58     case 3:
59         if (remove("C:/Users/itsik/OneDrive/Desktop/ssi.txt") == 0) {
60             printf("File deleted successfully.\n");
61         } else {
62             perror("Error deleting file");
63         }
64         break;
65     default:
66         printf("Invalid choice.\n");
67         break;
68     }
69     return 0;
70 }
71

```

1. Create 2. Copy 3. Delete  
Enter your choice:

```

1 #include <stdio.h>
2 #include <pthread.h>
3 #include <semaphore.h>
4 sem_t mutex, writeBlock;
5 int data = 0, readersCount = 0;
6 void *reader(void *arg) {
7     for (int i = 0; i < 10; i++) {
8         sem_wait(&mutex);
9         readersCount++;
10        if (readersCount == 1) {
11            sem_wait(&writeBlock);
12        }
13        sem_post(&mutex);
14        printf("Reader reads data: %d\n", data);
15        sem_wait(&mutex); // Lock the mutex
16        readersCount--;
17        if (readersCount == 0) {
18            sem_post(&writeBlock); // Allow writers if no readers
19        }
20        sem_post(&mutex); // Unlock the mutex
21    }
22    return NULL;
23 }
24 void *writer(void *arg) {
25     for (int i = 0; i < 10; i++) {
26         sem_wait(&writeBlock); // Lock writeBlock semaphore for writing
27         data++;
28         printf("Writer writes data: %d\n", data);
29         sem_post(&writeBlock); // Unlock writeBlock semaphore
30     }
31     return NULL;
32 }
33 int main() {
34     pthread_t readers, writers;
35     sem_init(&mutex, 0, 1);
36     sem_init(&writeBlock, 0, 1);
37     pthread_create(&readers, NULL, reader, NULL);
38     pthread_create(&writers, NULL, writer, NULL);
39     pthread_join(readers, NULL);
40     pthread_join(writers, NULL);
41     sem_destroy(&mutex);
42     sem_destroy(&writeBlock);
43     return 0;
44 }
45

```

```

Reader reads data: 0
Reader reads data: 0
Reader reads data: 0
Reader reads data: 0
Reader reads data: 0
Reader reads data: 0
Reader reads data: 0
Reader reads data: 0
Reader reads data: 0
Reader reads data: 0
Writer writes data: 1
Writer writes data: 2
Writer writes data: 3
Writer writes data: 4
Writer writes data: 5
Writer writes data: 6
Writer writes data: 7
Writer writes data: 8
Writer writes data: 9
Writer writes data: 10

```

--- Code Execution Successful ---



main.c

Share

Run

Output

Clear

```
13 int file_count;
14 };
15 struct TwoLevelDir {
16     struct Directory dirs[MAX_DIRS];
17     int dir_count;
18 };
19 void addFile(struct TwoLevelDir *dir, char *dir_name, char *file_name, int size) {
20     int i, found = 0;
21     for (i = 0; i < dir->dir_count; i++) {
22         if (strcmp(dir->dirs[i].name, dir_name) == 0) {
23             if (dir->dirs[i].file_count < MAX_FILES_PER_DIR) {
24                 struct File new_file;
25                 strcpy(new_file.name, file_name);
26                 new_file.size = size;
27                 dir->dirs[i].files[dir->dirs[i].file_count++] = new_file;
28                 printf("File '%s' added to directory '%s'\n", file_name, dir_name);
29             } else {
30                 printf("Directory '%s' is full, cannot add more files\n", dir_name);
31             }
32             found = 1;
33             break;
34         }
35     }
36     if (!found) {
37         printf("Directory '%s' not found\n", dir_name);
38     }
39 }
40 void displayContents(struct TwoLevelDir *dir) {
41     int i, j;
42     for (i = 0; i < dir->dir_count; i++) {
43         printf("Directory: %s\n", dir->dirs[i].name);
44         printf("Files:\n");
45         for (j = 0; j < dir->dirs[i].file_count; j++) {
46             printf("    %s - Size: %d\n", dir->dirs[i].files[j].name, dir->dirs[i].files[j].size);
47         }
48     }
49 }
50 int main() {
51     struct TwoLevelDir root_dir;
52     root_dir.dir_count = 0;
53     addFile(&root_dir, "Documents", "Report.docx", 2048);
54     addFile(&root_dir, "Documents", "Presentation.pptx", 4096);
55     addFile(&root_dir, "Images", "Photo.jpg", 1024);
56     displayContents(&root_dir);
57     return 0;
58 }
```

Directory 'Documents' not found  
Directory 'Documents' not found  
Directory 'Images' not found  
  
--- Code Execution Successful ---

main.c

Share

Run

```
1 #include <stdio.h>
2 #include <pthread.h>
3 int counter = 0;
4 pthread_mutex_t mutex;
5 void* threadFunction(void *arg) {
6     for (int i = 0; i < 1000000; ++i) {
7         pthread_mutex_lock(&mutex);
8         counter++;
9         pthread_mutex_unlock(&mutex);
10    }
11    return NULL;
12 }
13 int main() {
14     pthread_mutex_init(&mutex, NULL);
15
16     pthread_t thread1, thread2;
17     pthread_create(&thread1, NULL, threadFunction, NULL);
18     pthread_create(&thread2, NULL, threadFunction, NULL);
19
20     pthread_join(thread1, NULL);
21     pthread_join(thread2, NULL);
22
23     pthread_mutex_destroy(&mutex);
24
25     printf("Final counter value: %d\n", counter);
26     return 0;
27 }
28
```

Output

Final counter value: 2000000

=== Code Execution Successful ===