

main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <semaphore.h>
5 #include <unistd.h>
6
7 #define NUM_READERS 5 // Number of reader threads
8 #define NUM_WRITERS 2 // Number of writer threads
9
10 // Shared resource
11 int shared_resource = 0;
12
13 // Semaphores
14 sem_t mutex, write_mutex, read_count_mutex;
15 int read_count = 0; // Number of readers currently reading
16
17 // Reader function
18 void* reader(void* arg) {
19     while (1) {
20         // Start reading
```

Share

Run

Output

Clear

```
Reader 1: Reading shared resource = 0
Reader 4: Reading shared resource = 0
Reader 3: Reading shared resource = 0
Reader 2: Reading shared resource = 0
Reader 5: Reading shared resource = 0
Writer 1: Writing shared resource = 1
Writer 2: Writing shared resource = 2
Reader 1: Reading shared resource = 2
Reader 4: Reading shared resource = 2
Reader 2: Reading shared resource = 2
Reader 3: Reading shared resource = 2
Reader 5: Reading shared resource = 2
Writer 1: Writing shared resource = 3
Reader 1: Reading shared resource = 3
Reader 4: Reading shared resource = 3
Reader 2: Reading shared resource = 3
Reader 3: Reading shared resource = 3
Reader 5: Reading shared resource = 3
Writer 2: Writing shared resource = 4
Reader 4: Reading shared resource = 4
```

main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <unistd.h>
5
6 #define BUFFER_SIZE 5 // Define buffer size
7 #define PRODUCER_COUNT 1 // Number of producer threads
8 #define CONSUMER_COUNT 1 // Number of consumer threads
9
10 // Shared buffer and variables
11 int buffer[BUFFER_SIZE];
12 int in = 0; // Index for inserting items
13 int out = 0; // Index for removing items
14
15 // Mutex lock for synchronization
16 pthread_mutex_t mutex;
17
18 // Producer function
19 void* producer(void* arg) {
20     int item;
```

Run

Output

Produced: 83 at index 0  
Consumed: 83 from index 0  
Produced: 86 at index 1  
Produced: 77 at index 2  
Consumed: 86 from index 1  
Produced: 15 at index 3  
Consumed: 77 from index 2  
Produced: 93 at index 4  
Produced: 35 at index 0  
Consumed: 15 from index 3  
Produced: 86 at index 1  
Consumed: 93 from index 4  
Produced: 92 at index 2  
Consumed: 35 from index 0  
Produced: 49 at index 3  
Produced: 21 at index 4  
Consumed: 86 from index 1  
Produced: 62 at index 0  
Consumed: 92 from index 2  
Produced: 27 at index 1

Clear

main.c

 Share

Run

## Output

Clear

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <semaphore.h>
5  #include <unistd.h>
6
7  #define BUFFER_SIZE 5 // Define buffer size
8  #define PRODUCER_COUNT 1 // Number of producer threads
9  #define CONSUMER_COUNT 1 // Number of consumer threads
10
11 // Shared buffer and variables
12 int buffer[BUFFER_SIZE];
13 int in = 0; // Index for inserting items
14 int out = 0; // Index for removing items
15
16 // Semaphores
17 sem_t empty, full, mutex;
18
19 // Producer function
20 void* producer(void* arg) {

```

```
Produced: 83 at index 0
Consumed: 83 from index 0
Produced: 86 at index 1
Consumed: 86 from index 1
Produced: 77 at index 2
Produced: 15 at index 3
Consumed: 77 from index 2
Produced: 93 at index 4
Produced: 35 at index 0
Consumed: 15 from index 3
Produced: 86 at index 1
Produced: 92 at index 2
Consumed: 93 from index 4
Produced: 49 at index 3
```



main.c

```
1 #include <stdio.h>
2 #include <stdbool.h>
3
4 #define P 5 // Number of processes
5 #define R 3 // Number of resource types
6
7 // Function to calculate if the system is in a safe state
8 bool is_safe(int available[], int max[][R], int allot[][R],
9             int need[][R]) {
10     int work[R];
11     bool finish[P] = {false};
12     int safeSeq[P];
13     int count = 0;
14
15     // Initialize work[] as available[]
16     for (int i = 0; i < R; i++) {
17         work[i] = available[i];
18     }
19     // Try to find a safe sequence
```

Run

Output

A module you have imported isn't available at the moment. It will be available soon.

Clear



main.c



Share

Run

Output

Clear

```
1 #include <stdio.h>
2 #include <string.h>
3
4 #define FILENAME "day 4 programs.pdf"
5
6 // Define the employee structure
7 typedef struct {
8     int id;
9     char name[50];
10    float salary;
11 } Employee;
12
13 // Function to add a new employee record to the file
14 void add_employee(FILE *file) {
15     Employee emp;
16
17     // Get employee details
18     printf("Enter employee ID: ");
19     scanf("%d", &emp.id);
20     getchar(); // Consume newline character left by scanf
```

Unable to open file.

=== Code Exited With Errors ===





main.c



Share

Run

Output

Clear

```
140     subdir_name[strcspn(subdir_name, "\n")] = '\0'
        ; // Remove newline character
141     printf("Enter filename to delete: ");
142     fgets(filename, sizeof(filename), stdin);
143     filename[strcspn(filename, "\n")] = '\0'; //
        Remove newline character
144     delete_file_from_subdir(&dir, subdir_name,
        filename);
145     break;
146 case 4:
147     printf("Enter subdirectory name to display: "
        );
148     fgets(subdir_name, sizeof(subdir_name), stdin
        );
149     subdir_name[strcspn(subdir_name, "\n")] = '\0'
        ; // Remove newline character
150     display_subdir(&dir, subdir_name);
151     break;
152 case 5:
153     printf("Exiting the program \n");
```

--- Two-Level Directory Management ---

1. Add subdirectory
2. Add file to subdirectory
3. Delete file from subdirectory
4. Display subdirectory contents
5. Exit

Enter your choice: 4

Enter subdirectory name to display: ghy

Subdirectory 'ghy' not found.

--- Two-Level Directory Management ---

1. Add subdirectory
2. Add file to subdirectory
3. Delete file from subdirectory
4. Display subdirectory contents
5. Exit

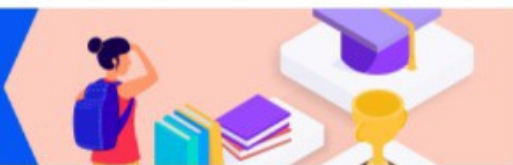
Enter your choice: 3

Enter subdirectory name to delete file from: xyz

Enter filename to delete: sdg







main.c



Share

Run

Output

Clear

```
1 #include <stdio.h>
2 #include <string.h>
3
4 #define MAX_FILES 10
5 #define FILENAME_LENGTH 50
6
7 // Structure to represent the directory
8 typedef struct {
9     char filenames[MAX_FILES][FILENAME_LENGTH];
10    int file_count;
11 } Directory;
12
13 // Function to initialize the directory
14 void init_directory(Directory* dir) {
15     dir->file_count = 0;
16 }
17
18 // Function to add a file to the directory
19 void add_file(Directory* dir, const char* filename) {
20     if (dir->file_count < MAX_FILES) {
```

```
--- Directory Management ---
1. Add file
2. Delete file
3. Display directory
4. Exit
Enter your choice: 2
Enter file name to delete: xyz
File 'xyz' not found in the directory.

--- Directory Management ---
1. Add file
2. Delete file
3. Display directory
4. Exit
Enter your choice: |
```



main.c



Share

Run

Output

Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX_BLOCKS 10
5
6 // Memory block structure
7 typedef struct {
8     int size; // Size of the block
9     int is_free; // 1 if the block is free, 0 if it is
        allocated
10 } MemoryBlock;
11
12 // Memory pool
13 MemoryBlock memory[MAX_BLOCKS];
14
15 // Initialize memory pool
16 void initialize_memory() {
17     for (int i = 0; i < MAX_BLOCKS; i++) {
18         memory[i].size = rand() % 100 + 1; // Assign random
            sizes

```

Memory Pool:

Block #	Size	Status
1	84	Free
2	87	Free
3	78	Free
4	16	Free
5	94	Free
6	36	Free
7	87	Free
8	93	Free
9	50	Free
10	22	Free

Memory allocation requests:  
First Fit: Allocated block 1 of size 84

Memory Pool:

Block #	Size	Status
1	84	Allocated
2	87	Free



main.c

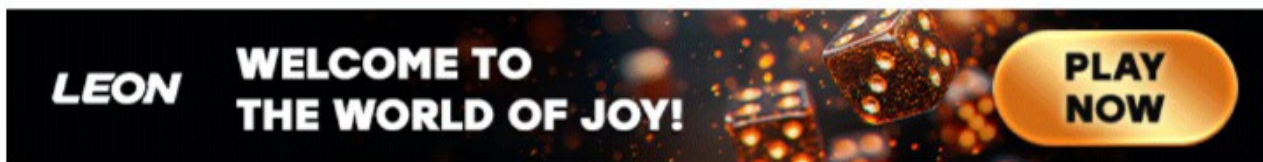
```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5
6 #define NUM_PHILOSOPHERS 5
7
8 pthread_mutex_t forks[NUM_PHILOSOPHERS]; // Mutex for each
   fork
9
10 void* philosopher(void* num) {
11     int id = *((int*)num);
12     int left = id;
13     int right = (id + 1) % NUM_PHILOSOPHERS;
14
15     while(1) {
16         // Thinking
17         printf("Philosopher %d is thinking\n", id);
18         sleep(1);
19     }
```

Run

Output

Philosopher 0 is thinking  
Philosopher 1 is thinking  
Philosopher 3 is thinking  
Philosopher 2 is thinking  
Philosopher 4 is thinking  
Philosopher 0 picked up left fork 0  
Philosopher 0 picked up right fork 1  
Philosopher 0 is eating  
Philosopher 3 picked up left fork 3  
Philosopher 3 picked up right fork 4  
Philosopher 3 is eating  
Philosopher 2 picked up left fork 2

Clear



main.c

```
1  #include <stdio.h>
2  #include <pthread.h>
3
4  void* print_message(void* msg) {
5      printf("%s\n", (char*)msg);
6      return NULL;
7  }
8
9  int main() {
10     pthread_t thread1, thread2;
11     const char* msg1 = "Hello from Thread 1!";
12     const char* msg2 = "Hello from Thread 2!";
13
14     // Create two threads
15     pthread_create(&thread1, NULL, print_message, (void*)msg1);
16     pthread_create(&thread2, NULL, print_message, (void*)msg2);
17
18     // Wait for both threads to finish
19     pthread_join(thread1, NULL);
20     pthread_join(thread2, NULL);
}
```

Run

Output

Clear

```
^Hello from Thread 1!
Hello from Thread 2!

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