```c
//Producer Consumer Problem
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define BUFFER_SIZE 10
#define NUM_PRODUCERS 2
#define NUM_CONSUMERS 2

// Buffer and related variables
int buffer[BUFFER_SIZE];
int in = 0;
int out = 0;

// Semaphores
sem_t empty; // Counts the number of empty slots in the buffer
sem_t full; // Counts the number of full slots in the buffer
sem_t mutex; // Mutex for mutual exclusion on buffer

void print_buffer()
{
    printf("Buffer: ");
    for (int i = 0; i < BUFFER_SIZE; ++i)
    {
        if (i == in && i == out)
        {
            printf("[%d*] ", buffer[i]); // Mark the current in and out indices
        }
        else if (i == in)
        {
            printf("[%d<] ", buffer[i]); // Mark the current in index
        }
        else if (i == out)
        {
            printf("[%d>] ", buffer[i]); // Mark the current out index
        }
        else
        {
            printf("[%d] ", buffer[i]); // Regular slot
        }
    }
    printf("\n");
}

void* producer(void* arg)
{
    int id = *((int*)arg);
    while (1)
    {
        sleep(rand() % 3); // Simulate variable time for producing an item
        int item = rand() % 100; // Produce a random item
```

```
54          sem_wait(&empty); // Wait for an empty slot
55          sem_wait(&mutex); // Enter critical section
56          // Put the item into the buffer
57          buffer[in] = item;
58          printf("Producer %d produced %d at index %d\n", id, item, in);
59          in = (in + 1) % BUFFER_SIZE;
60          print_buffer(); // Print buffer state
61          sem_post(&mutex); // Exit critical section
62          sem_post(&full); // Signal that buffer has new data
63      }
64      return NULL;
65  }
66  void* consumer(void* arg)
67  {
68      int id = *((int*)arg);
69      while (1)
70      {
71          sem_wait(&full); // Wait for a full slot
72          sem_wait(&mutex); // Enter critical section
73          // Take the item from the buffer
74          int item = buffer[out];
75          printf("Consumer %d consumed %d from index %d\n", id, item, out);
76          out = (out + 1) % BUFFER_SIZE;
77          print_buffer(); // Print buffer state
78          sem_post(&mutex); // Exit critical section
79          sem_post(&empty); // Signal that buffer has an empty slot
80          sleep(rand() % 3); // Simulate variable time for consuming an item
81      }
82      return NULL;
83  }
84  int main()
85  {
86      pthread_t producers[NUM_PRODUCERS];
87      pthread_t consumers[NUM_CONSUMERS];
88      int producer_ids[NUM_PRODUCERS];
89      int consumer_ids[NUM_CONSUMERS];
90
91      // Initialize semaphores
92      sem_init(&empty, 0, BUFFER_SIZE); // Buffer is initially empty
93      sem_init(&full, 0, 0); // No items to consume initially
94      sem_init(&mutex, 0, 1); // Mutex is initially unlocked
95
96      // Initialize buffer with -1 (indicating empty slots)
97      for (int i = 0; i < BUFFER_SIZE; ++i)
98      {
99          buffer[i] = -1;
100     }
101
102     // Create producer threads
103     for (int i = 0; i < NUM_PRODUCERS; ++i)
104     {
105         producer_ids[i] = i;
106         pthread_create(&producers[i], NULL, producer, &producer_ids[i]);
107     }
```

```
108
109     // Create consumer threads
110     for (int i = 0; i < NUM_CONSUMERS; ++i)
111     {
112         consumer_ids[i] = i;
113         pthread_create(&consumers[i], NULL, consumer, &consumer_ids[i]);
114     }
115
116     // Wait for threads to finish (they won't in this example)
117     for (int i = 0; i < NUM_PRODUCERS; ++i)
118     {
119         pthread_join(producers[i], NULL);
120     }
121     for (int i=0; i<NUM_CONSUMERS; ++i)
122     {
123         pthread_join(consumers[i], NULL);
124     }
125
126     // Destroy semaphores
127     sem_destroy(&empty);
128     sem_destroy(&full);
129     sem_destroy(&mutex);
130     return 0;
131 }
```