```
#define _XOPEN_SOURCE 600
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <stdlib.h>  // for rand()

#define BUFFER_SIZE 5  // Size of the shared buffer

int buffer[BUFFER_SIZE]; // Shared buffer
int in = 0, out = 0;     // Indices for producer and consumer

// Declare semaphores and mutex
sem_t empty;  // Counts empty buffer slots
sem_t full;   // Counts filled buffer slots
pthread_mutex_t mutex; // Protects shared buffer access

// Producer function
void* producer(void* arg) {
    int id = *((int*)arg);
    int item;

    while (1) {
        item = rand() % 100;  // Produce an item
        sem_wait(&empty);     // Wait if buffer is full
        pthread_mutex_lock(&mutex); // Enter critical section

        buffer[in] = item;
        printf("Producer %d produced item %d at position %d\n", id, item, in);
        in = (in + 1) % BUFFER_SIZE;
```

```
        pthread_mutex_unlock(&mutex); // Leave critical section
        sem_post(&full);   // Signal that buffer has more filled slots

        sleep(1); // Simulate time delay
    }
    return NULL;
}

// Consumer function
void* consumer(void* arg) {
    int id = *((int*)arg);
    int item;

    while (1) {
        sem_wait(&full);      // Wait if buffer is empty
        pthread_mutex_lock(&mutex); // Enter critical section

        item = buffer[out];
        printf("\tConsumer %d consumed item %d from position %d\n", id, item, out);
        out = (out + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex); // Leave critical section
        sem_post(&empty);   // Signal that buffer has more empty slots

        sleep(2); // Simulate time delay
    }
    return NULL;
}

int main() {
    pthread_t prod[3], cons[2];
```

```
    int p_id[3] = {1, 2, 3};
    int c_id[2] = {1, 2};

    // Initialize semaphores and mutex
    sem_init(&empty, 0, BUFFER_SIZE); // Initially all buffer slots are empty
    sem_init(&full, 0, 0);           // Initially buffer has no items
    pthread_mutex_init(&mutex, NULL);

    // Create producer threads
    for (int i = 0; i < 3; i++) {
        pthread_create(&prod[i], NULL, producer, &p_id[i]);
    }

    // Create consumer threads
    for (int i = 0; i < 2; i++) {
        pthread_create(&cons[i], NULL, consumer, &c_id[i]);
    }

    // Join threads (infinite loop, so typically won't reach)
    for (int i = 0; i < 3; i++)
        pthread_join(prod[i], NULL);
    for (int i = 0; i < 2; i++)
        pthread_join(cons[i], NULL);

    // Cleanup
    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```