CLASSICAL PROCESS SYNCHRONIZATION

```
  #include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define BUFFER_SIZE 5
int buffer[BUFFER_SIZE];
int in = 0, out = 0;
sem_t empty, full;
pthread_mutex_t mutex;
void* producer(void* arg) {
    int item;
    for (int i = 1; i <= 10; i++) {
        item = i; // produce item
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[in] = item;
        printf("Producer produced: %d\n", item);
        in = (in + 1) % BUFFER_SIZE;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
        sleep(1);
    }
    return NULL;
}
void* consumer(void* arg) {
    int item;
    for (int i = 1; i <= 10; i++) {
        sem_wait(&full);            // wait if buffer is empty
        pthread_mutex_lock(&mutex);
        item = buffer[out];
        printf("Consumer consumed: %d\n", item);
        out = (out + 1) % BUFFER_SIZE;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
        sleep(2);
    }
    return NULL;
}
int main() {
    pthread_t prod, cons;
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    pthread_mutex_init(&mutex, NULL);
    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);
    pthread_join(prod, NULL);
    pthread_join(cons, NULL);
    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);
    return 0;
}
```