18. Write a program for solving the producer consumer problem with the following scenario: The producer should produce data only when the buffer is not full. Data can only be consumed by the consumer if and only if the memory buffer is not empty.

Test Case:

Buffer Size: 3

Consume an item in the beginning and show that the buffer is EMPTY

Produce 4 items and show that the buffer is FULL

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define BUFFER\_SIZE 3

int buffer[BUFFER\_SIZE]; // the shared buffer

int count = 0; // the number of items in the buffer

int in = 0; // the index of the next free slot in the buffer

int out = 0; // the index of the next item to consume from the buffer

sem\_t empty; // counting semaphore for empty slots in the buffer

sem\_t full; // counting semaphore for full slots in the buffer

pthread\_mutex\_t mutex; // mutual exclusion lock for accessing the buffer

void\* producer(void\* arg) {

int data = 1;

while (1) {

sem\_wait(&empty); // wait for an empty slot in the buffer

pthread\_mutex\_lock(&mutex); // acquire the lock

if (count < BUFFER\_SIZE) {

buffer[in] = data;

printf("Produced item %d at index %d\n", data, in);

in = (in + 1) % BUFFER\_SIZE;

count++;

data++;

if (count == BUFFER\_SIZE) {

printf("The buffer is FULL\n");

}

}

pthread\_mutex\_unlock(&mutex); // release the lock

sem\_post(&full); // signal that a slot in the buffer is now full

}

return NULL;

}

void\* consumer(void\* arg) {

while (1) {

sem\_wait(&full); // wait for a full slot in the buffer

pthread\_mutex\_lock(&mutex); // acquire the lock

if (count > 0) {

int data = buffer[out];

printf("Consumed item %d from index %d\n", data, out);

out = (out + 1) % BUFFER\_SIZE;

count--;

if (count == 0) {

printf("The buffer is EMPTY\n");

}

}

pthread\_mutex\_unlock(&mutex); // release the lock

sem\_post(&empty); // signal that a slot in the buffer is now empty

}

return NULL;

}

int main() {

// initialize semaphores and mutex

sem\_init(&empty, 0, BUFFER\_SIZE);

sem\_init(&full, 0, 0);

pthread\_mutex\_init(&mutex, NULL);

// create producer and consumer threads

pthread\_t prod\_tid, cons\_tid;

pthread\_create(&prod\_tid, NULL, producer, NULL);

pthread\_create(&cons\_tid, NULL, consumer, NULL);

// wait for threads to finish (which they never will)

pthread\_join(prod\_tid, NULL);

pthread\_join(cons\_tid, NULL);

// clean up

pthread\_mutex\_destroy(&mutex);

sem\_destroy(&full);

sem\_destroy(&empty);

return 0;

}

