Bounded Buffer

written by Nik Tsonev

Intro

there is a buffer of n slots where every buffer is able to store 1 unit of arbitrary data, and there are two processes running: producer and consumer.

Producer - inserts 1 unit of data into the some buffer i
Consumer - removes 1 unit of data from some buffer i

your task is to create the synchronisation.

note that we do not want the producer to insert data when the buffer is full and the consumer to take data when the buffer is empty

Solution

use a `mutex` m and two `condition variables` to signal when the buffer is empty and full

make producer lock the mutex and then while the buffer is full wait for the is_empty condition var that will be signaled from the consumer thread.

add a new element in buffer[count]
increment count and signal not_empty to the consumer
then unlock the mutex

for the consumer
lock the mutex
and while count == buffer_size wait for the cond variable not_empty
then decrement by one, signal that it is not full and unlock the mutex

```
#include <pthread.h>
```

```c
#include <stdlib.h>
#include <stdio.h>

#define BUFFER_SIZE 10 // arbitrary
int buffer[BUFFER_SIZE];
int count = 0;

pthread_mutex_t m;
pthread_cond_t not_empty;
pthread_cond_t not_full;

void* producer(void *arg) {
        (void) arg;
        while (1) {
                int item = rand() % 100;
                pthread_mutex_lock(&m);

                while (count == BUFFER_SIZE)
                        pthread_cond_wait(&not_full, &m);

                buffer[count] = item;
                count++;
                printf("produced %d\n", item);
                pthread_cond_signal(&not_empty);
                pthread_mutex_unlock(&m);

        }
        return NULL;
}

void* consumer(void *arg) {
        (void)arg;
        while (1) {
                pthread_mutex_lock(&m);

                while (count == 0)
                        pthread_cond_wait(&not_empty, &m);

                printf("got %d\n", buffer[--count]);
```

```
                pthread_cond_signal(&not_full);
                pthread_mutex_unlock(&m);
        }
        return NULL;
}
```