```c
/*
   Windows 10
   Program Dev c ++
   The C Programming Language
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <time.h>
void *add_item();
void *remove_item();
void *append_buffer();
void *remove_buffer();
int i;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int PRODUCERS, CONSUMERS, BUFFER_SIZE, REQUEST;
int buffer[100000]; //Buffer Size
int tail = 0, head = 0, request = 0, success = 0;
clock_t timer, timer1;

//Define the main function
int main (int argc, char*argv)
{
        printf("PRODUCERS:\n") ;
        scanf("%d", &PRODUCERS);
        printf("Consumer:\n");
        scanf("%d", &CONSUMERS);
        printf("Buffer:\n");
        scanf("%d", &BUFFER_SIZE);
        printf("Request:\n");
        scanf("%d", &REQUEST);

    timer = clock(); //time of cpu run

    pthread_t thread_producer[PRODUCERS];
    pthread_t thread_consumer[CONSUMERS];

    for( i=0; i<PRODUCERS; i++){

        pthread_create(&thread_producer[i], NULL, append_buffer, NULL);
}
    for(i=0; i<CONSUMERS; i++){
        pthread_create(&thread_consumer[i], NULL, remove_buffer, NULL);
}
    for(i=0; i<CONSUMERS; i++){
        pthread_join(thread_consumer[i], NULL);
}
    for(i=0; i<PRODUCERS; i++){
        pthread_join(thread_producer[i], NULL);
}
    timer1 = clock(); //
    float elapsed = ((float)(timer1 - timer) / CLOCKS_PER_SEC); //Declare a translator to
hold the elapsed numeric value.

```

```c
    printf("\n");
    printf("# buff %d %d %d %d\n", PRODUCERS , CONSUMERS , BUFFER_SIZE , REQUEST);
    printf("Producers %d, Consumers %d\n", PRODUCERS, CONSUMERS);
    printf("Buffer size %d\n", BUFFER_SIZE);
    printf("Requests %d\n\n", request);
    printf("Successfully consumed %d requests (%.1f%%)\n", success, (float)success * 100
/ request);
    printf("Elapsed Time %.2f s\n", elapsed);
    printf("Throughput %.2f successful requests/s\n", (float)(success) / elapsed);

    exit(EXIT_SUCCESS);
}

// function Add item
void *add_item()
 {
    buffer[head++] = 1;
    head = head % BUFFER_SIZE;
    printf("Append  Head %d Tail %d Buff %d\n", head , tail , buffer[head]);
}
// function Remove item
void *remove_item()
{
    buffer[tail++] = 0;
    tail = tail % BUFFER_SIZE;
    printf("Remove  Head %d Tail %d Buff %d\n", head , tail , buffer[tail]);
}
// function Append in buffer item
void *append_buffer()
{
    printf("Append thread number %ld\n", pthread_self());

    while(request<REQUEST) {
        if(!pthread_mutex_trylock(&mutex) && request<REQUEST) {
            if(buffer[head] == 0) {
                add_item();
                request++;
                printf(" + thread %ld append success\n", pthread_self());
            }
            else {
            printf("Buffer overflow\n");
            }
            pthread_mutex_unlock(&mutex);
        }
    }
    pthread_exit(NULL);
}


// function Remove in buffer item
void *remove_buffer()
{
    printf("Remove thread number %ld\n", pthread_self());

    while(success<REQUEST) {
        if(!pthread_mutex_trylock(&mutex) && success<REQUEST) {
            if(buffer[tail] == 1) {
                remove_item();
                success++;
                printf(" - thread %ld remove success\n", pthread_self());
            }
            else {
            printf("Buffer underflow\n");
            }
            pthread_mutex_unlock(&mutex);
        }
    }
    pthread_exit(NULL);
}

```