LAB 13
Breeha Qasim

Listing1

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <pthread.h>
#include <string.h>

static pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;
static pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
static int avail = 0;

static void *threadFunc(void *arg) {
    int cnt = atoi((char *)arg);
    int s, j;

    for (j = 0; j < cnt; j++) {
        //sleep(1); // Simulate production delay

        s = pthread_mutex_lock(&mtx);

        avail++;
        s = pthread_cond_signal(&cond);

        s = pthread_mutex_unlock(&mtx);

    }

    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t tid;
    int s, j;
    int totRequired;

    int numConsumed;
    int done;
    time_t t;
```

```c
    t = time(NULL);

    totRequired = 0;
    for (j = 1; j < argc; j++) {
        totRequired += atoi(argv[j]);
        s = pthread_create(&tid, NULL, threadFunc, argv[j]);
    }

    numConsumed = 0;
    done = 0;

    for (;;) {
        s = pthread_mutex_lock(&mtx);

        while (avail == 0) {
            s = pthread_cond_wait(&cond, &mtx);
        }

        while (avail > 0) {
            numConsumed++;
            avail--;
            printf("T=%ld : numConsumed=%d\n", (long)(time(NULL) - t), numConsumed);

            done = numConsumed >= totRequired;
        }

        s = pthread_mutex_unlock(&mtx);

        if (done)
            break;
    }

    exit(EXIT_SUCCESS);
}
```

```
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab13$ nano listing1.c
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab13$ gcc listing1.c -o list1 -lpthread
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab13$ ./list1 1 2 3
T=0 : numConsumed=1
T=0 : numConsumed=2
T=0 : numConsumed=3
T=0 : numConsumed=4
T=0 : numConsumed=5
T=0 : numConsumed=6
```

Explanation:
Three producer threads are generated, each of which produces one, two, and three units, for a total of six units. As soon as these units are created, the main thread consumes them, acting as a consumer. The output shows the total amount of units consumed (numConsumed) and the elapsed time (T=0) on each line. The time stays at 0 seconds for all processes because production and consumption occur nearly instantaneously.

prod_consume.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <pthread.h>

#define BUFFER_SIZE 10

pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond_producer = PTHREAD_COND_INITIALIZER;
pthread_cond_t cond_consumer = PTHREAD_COND_INITIALIZER;

int buffer[BUFFER_SIZE]; //thread sharing an array of size
int avail = 0;
int produce_index = 0;
int consume_index = 0;

void *threadFunc(void *arg) {
    int cnt = atoi((char *)arg);
    int item;

    for (int i = 0; i < cnt; i++) {
        //sleep(1);
        pthread_mutex_lock(&mtx);

        while (avail == BUFFER_SIZE) {
            pthread_cond_wait(&cond_producer, &mtx);
        }

        item = rand() % 100;
        buffer[produce_index] = item;
        printf("Produced: %d at index %d\n", item, produce_index);
        produce_index = (produce_index + 1) % BUFFER_SIZE;

        avail++;

        pthread_cond_signal(&cond_consumer);
        pthread_mutex_unlock(&mtx);

    }

    return NULL;
}
```

```c
void *threadFunc2(void *arg) {
    int totRequired = atoi((char *)arg);
    int numConsumed = 0;
    int item;
    int done;

    for (;;) {
        pthread_mutex_lock(&mtx);


        while (avail == 0) {
            pthread_cond_wait(&cond_consumer, &mtx);
        }

        item = buffer[consume_index];
        printf("Consumed: %d from index %d\n", item, consume_index);
        consume_index = (consume_index + 1) % BUFFER_SIZE;

        avail--;
        numConsumed++;

        //printf("T=%ld : numConsumed=%d\n", (long)(time(NULL) - t), numConsumed);

        pthread_cond_signal(&cond_producer);
        pthread_mutex_unlock(&mtx);

        done = numConsumed >= totRequired;
        if (done)
            break;

        sleep(1);
    }

    return NULL;
}

int main(int argc, char *argv[]) {

    pthread_t producer_thread, consumer_thread;

    pthread_create(&producer_thread, NULL, threadFunc, argv[1]);
    pthread_create(&consumer_thread, NULL, threadFunc2, argv[2]);
```

```c
    pthread_join(producer_thread, NULL);
    pthread_join(consumer_thread, NULL);

    return 0;
}
```

```
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab13$ nano prod_consume.c
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab13$ gcc prod_consume.c -o prod -lpthread
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab13$ ./prod 5 5
Produced: 83 at index 0
Produced: 86 at index 1
Produced: 77 at index 2
Produced: 15 at index 3
Produced: 93 at index 4
Consumed: 83 from index 0
Consumed: 86 from index 1
Consumed: 77 from index 2
Consumed: 15 from index 3
Consumed: 93 from index 4
```

Explanation:
The producer creates 5 items, and the consumer consumes them, ensuring no overproduction or overconsumption. The circular buffer handles indices, and the interleaved output reflects the concurrent execution of threads.