Федеральное государственное автономное образовательное учреждение высшего образования
"Национальный исследовательский университет "Высшая школа экономики"
Московский институт электроники и математики
им. А. Н. Тихонова НИУ ВШЭ
Департамент компьютерной инженерии

Лабораторная работа № 9
по теме:
«Мьютексы. Условные переменные»

По дисциплине

«Операционные системы»

Группа: **БПМ 243**

Студент: **Николаев Степан Денисович**

Москва 2026

Цель работы

Изучить мьютексы и условные переменные pthread и применить их в задаче «производитель — потребитель» с кольцевым буфером.

Ход работы

Все программы собирал и запускал в виртуальной машине с Ubuntu в UTM под обычным пользователем marmos, компилятор gcc с ключами -Wall -Wextra, предупреждений нет.

В prodcons.c буфер — массив на 10 int с индексами head (откуда брать), tail (куда класть) и счётчиком count. Один поток-производитель кладёт числа от 1 до 30, три потока-потребителя забирают их. Буфер, индексы и счётчик защищены одним мьютексом.

Условных переменных две. not_full: производитель ждёт на ней, пока в буфере появится место (в тексте задания написано «не пуст», по смыслу здесь «не полон»). not_empty: на ней ждут потребители, пока появится элемент. Условия проверяются в цикле while, а не в if, потому что поток может проснуться, когда другой потребитель уже забрал элемент. Потребители после взятия элемента «обрабатывают» его через usleep (20, 40 и 60 мс), поэтому буфер успевает заполниться.

Завершение: после последнего элемента производитель под мьютексом ставит флаг done и вызывает pthread_cond_broadcast на not_empty, чтобы разбудить всех потребителей. Потребитель выходит, когда буфер пуст и done равен 1. Главный поток ждёт все потоки через pthread_join, печатает итоги и уничтожает мьютекс и обе условные переменные.

```
marmos@marmos-QEMU-Virtual-Machine:~/practice9$ cd ~/practice9
gcc -Wall -Wextra -pthread -o prodcons prodcons.c
./prodcons > run1.txt
head -n 22 run1.txt
Производитель: положил 1 (в буфере 1)
Производитель: положил 2 (в буфере 2)
Производитель: положил 3 (в буфере 3)
Производитель: положил 4 (в буфере 4)
Производитель: положил 5 (в буфере 5)
Производитель: положил 6 (в буфере 6)
Производитель: положил 7 (в буфере 7)
Производитель: положил 8 (в буфере 8)
Производитель: положил 9 (в буфере 9)
Производитель: положил 10 (в буфере 10)
Производитель: буфер полон, жду
  Потребитель 3: взял 1 (в буфере 9)
  Потребитель 1: взял 2 (в буфере 8)
  Потребитель 2: взял 3 (в буфере 7)
Производитель: положил 11 (в буфере 8)
Производитель: положил 12 (в буфере 9)
Производитель: положил 13 (в буфере 10)
Производитель: буфер полон, жду
  Потребитель 1: взял 4 (в буфере 9)
Производитель: положил 14 (в буфере 10)
Производитель: буфер полон, жду
  Потребитель 2: взял 5 (в буфере 9)
marmos@marmos-QEMU-Virtual-Machine:~/practice9$
```

Рис. 1. Компиляция и начало работы программы.

Производитель сразу заполнил буфер до 10 элементов и встал на not_full, потребители начали освобождать место.

```
marmos@marmos-QEMU-Virtual-Machine:~/practice9$ tail -n 12 run1.txt
grep -c положил run1.txt
grep -c взял run1.txt
grep -c 'буфер полон' run1.txt
  Потребитель 2: взял 23 (в буфере 7)
  Потребитель 1: взял 24 (в буфере 6)
  Потребитель 1: взял 25 (в буфере 5)
  Потребитель 2: взял 26 (в буфере 4)
  Потребитель 1: взял 27 (в буфере 3)
  Потребитель 3: взял 28 (в буфере 2)
  Потребитель 1: взял 29 (в буфере 1)
  Потребитель 2: взял 30 (в буфере 0)
Потребитель 1 обработал 15 элементов
Потребитель 2 обработал 9 элементов
Потребитель 3 обработал 6 элементов
Всего обработано 30 из 30, сумма 465 (ожидалась 465)
30
30
17
marmos@marmos-QEMU-Virtual-Machine:~/practice9$
```

Рис. 2. Конец работы и подсчёт записей в выводе.

Всего положено 30 элементов и взято 30, производитель ждал освобождения места 17 раз. Потребитель 1 обработал 15 элементов, 2 — 9, 3 — 6: чем быстрее поток, тем больше он забрал. Сумма взятых чисел 465 равна 1+2+...+30.

```
marmos@marmos-QEMU-Virtual-Machine:~/practice9$ for i in 1 2 3; do ./prodcons | tail -n 4; echo; done
Потребитель 1 обработал 16 элементов
Потребитель 2 обработал 8 элементов
Потребитель 3 обработал 6 элементов
Всего обработано 30 из 30, сумма 465 (ожидалась 465)

Потребитель 1 обработал 15 элементов
Потребитель 2 обработал 9 элементов
Потребитель 3 обработал 6 элементов
Всего обработано 30 из 30, сумма 465 (ожидалась 465)

Потребитель 1 обработал 16 элементов
Потребитель 2 обработал 8 элементов
Потребитель 3 обработал 6 элементов
Всего обработано 30 из 30, сумма 465 (ожидалась 465)

marmos@marmos-QEMU-Virtual-Machine:~/practice9$
```

Рис. 3. Три запуска подряд (последние строки).

Распределение между потребителями от запуска к запуску немного меняется (16, 8, 6 или 15, 9, 6), а итог всегда одинаковый: 30 элементов и сумма 465, ничего не потеряно и не обработано дважды.

Вывод. Реализовал задачу производитель — потребитель на кольцевом буфере с мьютексом и двумя условными переменными. Завершение через флаг done и broadcast работает без зависаний, все ресурсы освобождаются.

Приложение prodcons.c:

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

#define BUF_SIZE 10
#define ITEMS 30
#define CONSUMERS 3

static int buffer[BUF_SIZE];
static int head = 0;    // откуда читаем
static int tail = 0;    // куда пишем
static int count = 0;   // сколько элементов лежит в буфере
static int done = 0;    // производитель закончил

static pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
static pthread_cond_t not_full = PTHREAD_COND_INITIALIZER;
static pthread_cond_t not_empty = PTHREAD_COND_INITIALIZER;

static long consumed_sum[CONSUMERS];
static int consumed_cnt[CONSUMERS];

static void *producer(void *arg) {
    (void)arg;
    for (int i = 1; i <= ITEMS; i++) {
        pthread_mutex_lock(&mutex);
        while (count == BUF_SIZE) {
            printf("Производитель: буфер полон, жду\n");
            pthread_cond_wait(&not_full, &mutex);
        }
        buffer[tail] = i;
        tail = (tail + 1) % BUF_SIZE;
        count++;
        printf("Производитель: положил %d (в буфере %d)\n", i, count);
        pthread_cond_signal(&not_empty);
        pthread_mutex_unlock(&mutex);
    }
    // Сообщаем, что элементов больше не будет, и будим всех потребителей
    pthread_mutex_lock(&mutex);
    done = 1;
    pthread_cond_broadcast(&not_empty);
    pthread_mutex_unlock(&mutex);
    return NULL;
}

static void *consumer(void *arg) {
    int id = (int)(long)arg;
    for (;;) {
        pthread_mutex_lock(&mutex);
        while (count == 0 && !done)
            pthread_cond_wait(&not_empty, &mutex);
        if (count == 0 && done) {
            pthread_mutex_unlock(&mutex);
            break;
        }
        int item = buffer[head];
        head = (head + 1) % BUF_SIZE;
        count--;
        printf("  Потребитель %d: взял %d (в буфере %d)\n", id, item, count);
        pthread_cond_signal(&not_full);
        pthread_mutex_unlock(&mutex);

        consumed_sum[id - 1] += item;
        consumed_cnt[id - 1]++;
```

```
        usleep(20000 * id);   // имитация обработки, потребители разной скорости
    }
    return NULL;
}

int main() {
    pthread_t prod, cons[CONSUMERS];

    pthread_create(&prod, NULL, producer, NULL);
    for (long i = 1; i <= CONSUMERS; i++)
        pthread_create(&cons[i - 1], NULL, consumer, (void *)i);

    pthread_join(prod, NULL);
    for (int i = 0; i < CONSUMERS; i++)
        pthread_join(cons[i], NULL);

    long total = 0;
    int taken = 0;
    for (int i = 0; i < CONSUMERS; i++) {
        printf("Потребитель %d обработал %d элементов\n", i + 1, consumed_cnt[i]);
        total += consumed_sum[i];
        taken += consumed_cnt[i];
    }
    printf("Всего обработано %d из %d, сумма %ld (ожидалась %d)\n",
           taken, ITEMS, total, ITEMS * (ITEMS + 1) / 2);

    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&not_full);
    pthread_cond_destroy(&not_empty);
    return 0;
}
```