**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №5

по дисциплине

«Операционные системы»

Вариант 2

Студент

Группа АИ-20-1 \_\_\_\_\_\_\_\_\_\_ Глубоков Г.В.

Подпись, дата

Руководитель

к.т.н.,доц. Батищев Р. В.

Подпись, дата

Липецк 2022г.

# Задание кафедры:

Реализовать с помощью семафоров решение задачи «производитель-потребитель», которая сформулирована в следующем виде: написать программу-клиент – потребитель данных из кольцевого буфера и программу-сервер – генератор (производитель) данных в буфер. В кольцевом буфере можно выделить первую занятую, последнюю занятую, далее – первую свободную и затем – последнюю свободную позиции. Сервер генерирует заданное количество данных и после получения доступа к буферу помещает их в буфер, начиная с первой свободной позиции, если это возможно. В противном случае он ждет, когда освободится место в буфере. Клиент забирает заданное количество данных из буфера, освобождая соответствующие позиции в буфере, начиная с первой занятой, если это возможно. В противном случае клиент ждет, когда сервер поместит достаточное количество данных в буфер (количество данных M, генерируемых сервером, количество данных N, считываемых клиентом, размер буфера K заданы в табл. 5 прил.).

|  |  |
| --- | --- |
| № вар. | M, N,K |
| 2 | 5, 20, 50 |

# Теоретические сведения:

# Чтобы закрыть мьютекс, его нужно передать в вызов pthread\_mutex\_lock(). Если он в данный момент открыт, вызов сразу же возвращается. Если же мьютекс закрыт другим потоком, pthread\_mutex\_lock() блокируется, пока тот не откроется; в момент открытия данная функция снова блокирует мьютекс и возвращается. Если вызывающий поток сам закрыл мьютекс, переданный в функцию pthread\_mutex\_ lock(), тогда, если это стандартный тип мьютекса, тогда случится одно из двух, в зависимости от реализации: либо поток войдет в состояние взаимного блокирования, пытаясь закрыть мьютекс, которым он уже владеет, либо вызов завершится неудачей и вернет ошибку EDEADLK. В Linux по умолчанию происходит взаимное блокирование (другие возможные варианты будут описаны после рассмотрения разных типов мьютексов в подразделе 30.1.7). Функция pthread\_mutex\_unlock() открывает мьютекс, закрытый ранее вызывающим потоком. Если мьютекс уже открыт или был закрыт другим потоком, данная функция возвращает ошибкуХод работы:

## Код программы на C++:

## #include <pthread.h>

## #include <iostream>

## using namespace std;

## static char buff[20];

## static pthread\_mutex\_t mtx = PTHREAD\_MUTEX\_INITIALIZER;

## char gen\_random() {

## static const char alphanum[] ={'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'};

## int i=rand()%50;

## return alphanum[i];

## }

## static void \* threadFunc(void \*arg)

## {

## srand(time(0));

## 

## int i,j,s;

## int position=\*((int\*)arg);

## for(j=0;j<20;j++){

## s = pthread\_mutex\_lock(&mtx);

## 

## for(i=position\*5-5;i<position\*5;i++){

## char ch=gen\_random();

## buff[j] = ch;

## }

## s = pthread\_mutex\_unlock(&mtx);

## }

## return NULL;

## }

## static void \* WriteBuff(void \*arg)

## {

## int s;

## s = pthread\_mutex\_lock(&mtx);

## cout<<buff<<endl;

## s = pthread\_mutex\_unlock(&mtx);

## return NULL;

## }

## int main(int argc, char \*argv[])

## {

## for (int i=0;i<20;i++) {

## buff[i]='\0';

## }

## int position=rand()%3+1;

## pthread\_t r1, r2, w1, w2,w3,w4;

## int s, arr[4]={-1,-1,-1,-1},flag=0;

## for(;flag!=4;){

## if(arr[position]==-1){

## s = pthread\_create(&r1, NULL, threadFunc, &position);

## flag++;

## }

## else

## continue;

## position=rand()%4+1;

## if(arr[position]==-1){

## s = pthread\_create(&r2, NULL, threadFunc,&position);

## flag++;

## }

## 

## }

## s = pthread\_join(r1, NULL);

## s = pthread\_join(r2, NULL);

## s = pthread\_create(&w1, NULL, WriteBuff, NULL);

## s = pthread\_create(&w2, NULL, WriteBuff, NULL);

## s = pthread\_create(&w3, NULL, WriteBuff, NULL);

## s = pthread\_create(&w4, NULL, WriteBuff, NULL);

## s = pthread\_join(w1, NULL);

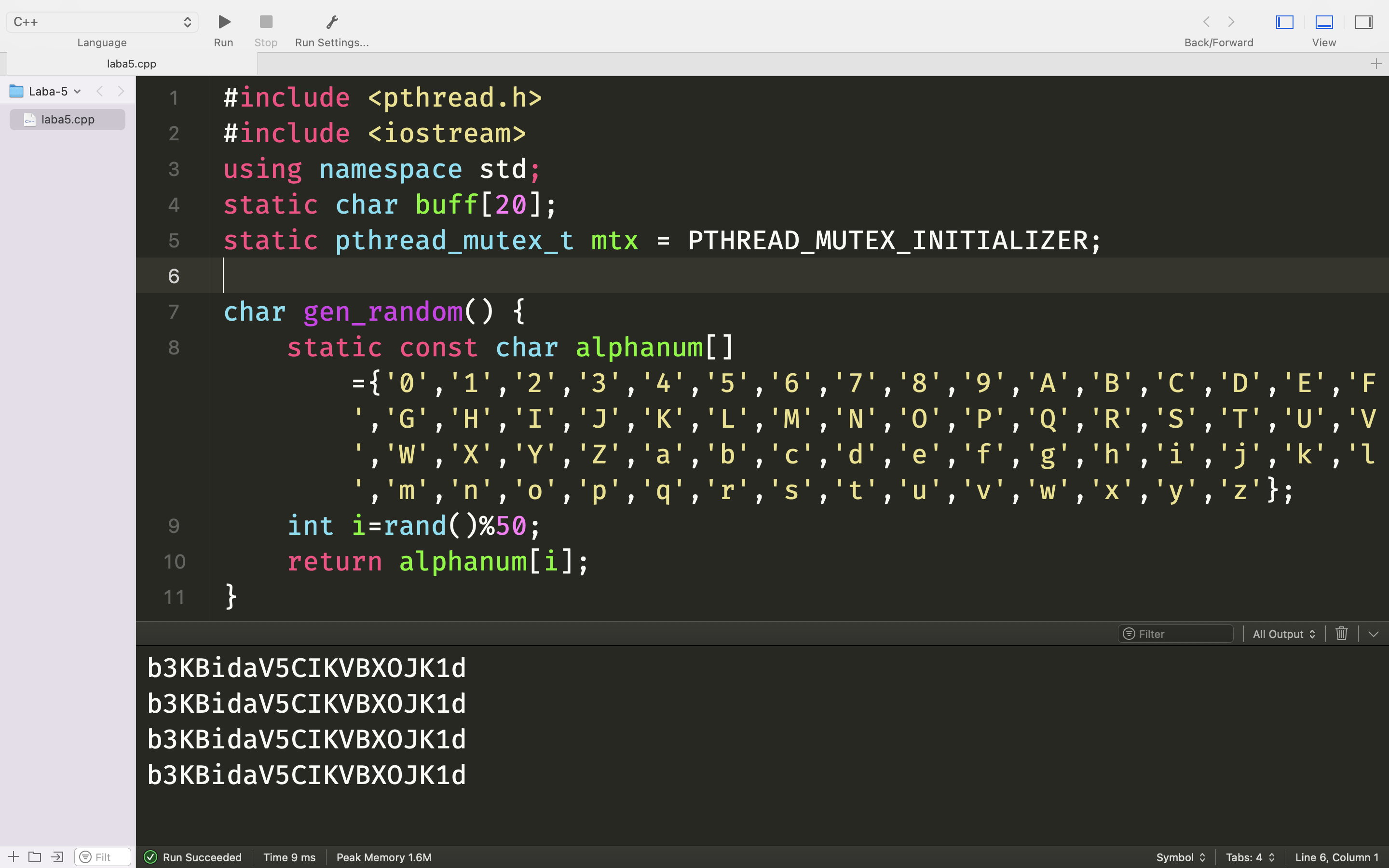
## s = pthread\_join(w2, NULL);

## s = pthread\_join(w3, NULL);

## s = pthread\_join(w4, NULL);

## }Пример выполнения:

На рисунке 1 показаны два файла: исходный и результирующий.



В ходе выполнения лабораторной работы были получены основные навыки работы с мьюиексами Linux API