Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №6

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:

Для работы с именованными семафорами предусмотрены следующие функции:

sem_open() — открывает или создает новый семафор, инициализирует его (если она его создала) и возвращает дескриптор, который можно использовать в дальнейшем;

sem_post(sem) и sem_wait(sem) — соответственно инкрементируют и декрементируют значение семафора;

sem_getvalue() — возвращает текущее значение семафора;

sem_close() — закрывает семафор, ранее открытый вызывающим процессом;

sem_unlink() — удаляет имя семафора и делает его кандидатом на удаление; само удаление произойдет, когда семафор закроет все процессы.

Ход работы:

Код программы на C++:

```
#include <iostream>
#include <pthread.h>
#include <semaphore.h>
using namespace std;

int G_D=5;
int R_D=20;
sem_t sem_rec_data;
sem_t sem_post_data;
pthread_t server;
pthread_t client;
pthread_mutex_t mutx = PTHREAD_MUTEX_INITIALIZER;

struct Buffer{
  int data[50] = {0};
  int inputPoint = 0;
  int outputPoint = 0;
};

void printBuffer(Buffer * buffer){
  for(int value: buffer->data){

    if(value!=0)
      cout << value << " " ;
    else
      cout << value << " " ;
```

```
  }
}

static void* serverThread(void* buffer){
  Buffer * buf = (Buffer *) buffer;

  for(int j = 0 ; j < 10 ; j++){

    for(int i = 0 ; i < G_D ; i++){
      if(i<=0){
        sem_wait(&sem_rec_data);
      } else {
        sem_trywait(&sem_rec_data);
      }
      if(buf->inputPoint==50){
        buf->inputPoint=0;
      }

      buf->data[buf->inputPoint] = rand() % 9 + 1;
      buf->inputPoint++;
    }

    pthread_mutex_lock(&mutx);
    cout << "Server" << endl;
    printBuffer(buf);
    cout << endl;
    pthread_mutex_unlock(&mutx);
    // if(j==9){
    //     for(int i=0 ; i < G_D+10 ; i++){
    //   sem_post(&sem_post_data);
    // }


    for(int i=0 ; i < G_D ; i++){
      sem_post(&sem_post_data);
    }

    }
    for(int i=0 ; i < 10 ; i++){
      sem_post(&sem_post_data);
    }
  pthread_exit(NULL);
}

static void * clientThread(void* buffer){
  Buffer * buf = (Buffer *) buffer;

  for(int j = 0 ; j < 3 ; j++){
    for(int i = 0 ; i < R_D ; i++){
      sem_wait(&sem_post_data);
      if(buf->outputPoint==50){
```

```
      buf->outputPoint=0;
     }
     buf->data[buf->outputPoint] = 0;
     buf->outputPoint++;
    }

    pthread_mutex_lock(&mutx);
    cout << "Client" << endl;
    printBuffer(buf);
    cout << endl;
    pthread_mutex_unlock(&mutx);

    int currentValueRD;
    sem_getvalue(&sem_rec_data,&currentValueRD);
    if(currentValueRD < R_D){
     int sizeOfReqiuredData = R_D + currentValueRD;
     for(int i=0 ; i < sizeOfReqiuredData ; i++){
      sem_post(&sem_rec_data);
     }
    }


  }
  pthread_exit(NULL);
}

int main(int argc, char *argv[]) {
  srand(time(NULL));
  Buffer buffer;

  sem_init(&sem_rec_data,0,R_D);
  sem_init(&sem_post_data,0,0);
  pthread_create(&client,NULL,&clientThread,&buffer);
  pthread_create(&server,NULL,&serverThread,&buffer);

  pthread_join(server,NULL);
  pthread_join(client,NULL);
  sem_destroy(&sem_rec_data);
  sem_destroy(&sem_post_data);
  return 0;
}
```

```
Server
5 5 5 4 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Server
5 5 5 4 3 1 6 7 6 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Server
5 5 5 4 3 1 6 7 6 5 6 6 7 7 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Server
5 5 5 4 3 1 6 7 6 5 6 6 7 7 9 1 6 3 4 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Client
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Server
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 6 2 9 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Server
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 6 2 9 7 3 7 8 7 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Server
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 6 2 9 7 3 7 8 7 7 2 9 2 5 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Server
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 6 2 9 7 3 7 8 7 7 2 9 2 5 4 3 5 7 7 2 0 0 0 0 0 0 0 0 0 0
Client
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Server
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 3 7 6 8 0 0 0 0 0
Server
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 3 7 6 8 4 6 4 4 9
Client
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

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