

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра вычислительной техники

ОТЧЕТ
по лабораторной работе № 10
по дисциплине
«Организация процессов и программирование в среде Linux»
ТЕМА: Синхронизация процессов с помощью семафоров

Студенты гр. 4306

Веретенников Л.М.

Преподаватель

Разумовский Г.В.

Санкт-Петербург
2017

Цель работы

Знакомство с организацией семафоров, системными функциями, обеспечивающими управление семафорами, и их использования для решения задач взаимного исключения и синхронизации.

Задание 1. Напишите две программы (Поставщик и Потребитель), которые работают с циклическим буфером ограниченного размера, расположенным в разделяемой памяти. Поставщик выделяет буфер и семафоры, читает по одному символу из файла и записывает его в буфер. Потребитель считывает по одному символу из буфера и выводит их на экран. Если буфер пустой, то Потребитель должен пассивно ждать, пока Поставщик не занесет туда хотя бы один символ. Если буфер полностью заполнен, то пассивно должен ждать Поставщик, пока Потребитель не извлечет из него по крайней мере один символ. Поставщик заканчивает свою работу, как только прочитает последний символ из файла и убедится, что Потребитель его прочитал. Потребитель заканчивает свою работу при отсутствии символов в буфере и завершении работы Поставщика.

Откомпилируете программы Поставщик и Потребитель. Запустите их на разных терминалах.

Текст программы producer.cpp:

```
#include <iostream>
#include <sys/shm.h>
#include <sys/sem.h>
#include <fstream>

using namespace std;

#define SIZE_BUFFER 10
#define MUTEX 0           //Контроллер ресурсов (семафор - 0)
#define FULL 1            //Контроллер заполненности (семафор - 1)
#define EMPTY 2          //Контроллер свободного места в буфере (семафор - 2)

union semun {
    int val;
    struct semid_ds *buf;
    unsigned short *array;
};
```

```

void semaphoreInit(int, int, int);
void semDecrement(int, int);
void semIncrement(int, int);

int main() {
    char* buffer;
    int shmid = shmget(1, SIZE_BUFFER * sizeof(char), 0666 | IPC_CREAT);
    buffer = static_cast<char *>(shmat(shmid, nullptr, 0));

    int semid = semget(1, 3, IPC_CREAT | 0666);
    semaphoreInit(semid, MUTEX, 1);
    semaphoreInit(semid, FULL, 0);
    semaphoreInit(semid, EMPTY, 10);

    char ch;
    int index = 0;
    FILE* file = fopen("input.txt", "r");
    while (ch = getc(file)) {
        semDecrement(semid, EMPTY);
        semDecrement(semid, MUTEX);
        buffer[index] = ch;
        semIncrement(semid, MUTEX);
        semIncrement(semid, FULL);
        index = (index + 1) % SIZE_BUFFER;
        if (ch == EOF)
            break;
        cout << "Producer: " << ch << endl;
    }

    shmdt(buffer);
    return 0;
}

void semaphoreInit(int semid, int semaphoreIndex, int value) {
    union semun un;
    un.val = value;
    semctl(semid, semaphoreIndex, SETVAL, un);
}

void semDecrement(int semid, int semaphoreIndex) {
    struct sembuf buf{};
    buf.sem_num = static_cast<unsigned short>(semaphoreIndex);
    buf.sem_op = -1;
    buf.sem_flg = 0;
    semop(semid, &buf, 1);
}

void semIncrement(int semid, int semaphoreIndex) {

```

```

    struct sembuf buf{};
    buf.sem_num = static_cast<unsigned short>(semaphoreIndex);
    buf.sem_op = 1;
    buf.sem_flg = 0;
    semop(semid, &buf, 1);
}

```

Текст программы consumer.cpp:

```

#include <iostream>
#include <sys/shm.h>
#include <sys/sem.h>

using namespace std;

#define SIZE_BUFFER 10
#define MUTEX 0           //Контроллер ресурсов (семафор - 0)
#define FULL 1            //Контроллер заполненности (семафор - 1)
#define EMPTY 2          //Контроллер свободного места в буфере (семафор
- 2)

void semDecrement(int, int);
void semIncrement(int, int);

int main() {
    char* buffer;
    int shmid = shmget(1, SIZE_BUFFER * sizeof(char), 0666 | IPC_CREAT);
    buffer = static_cast<char *>(shmat(shmid, nullptr, 0));

    int semid = semget(1, 3, IPC_CREAT | 0666);

    char ch;
    int index = 0;
    while (true) {
        semDecrement(semid, FULL);
        semDecrement(semid, MUTEX);
        ch = buffer[index];
        if (ch == EOF)
            break;
        semIncrement(semid, MUTEX);
        semIncrement(semid, EMPTY);
        index = (index + 1) % SIZE_BUFFER;
        cout << "Consumer: " << ch << endl;
    }

    shmdt(buffer);
    shmctl(shmid, IPC_RMID, nullptr);
    semctl(semid, IPC_RMID, 0);
    return 0;
}

```

```
}
```

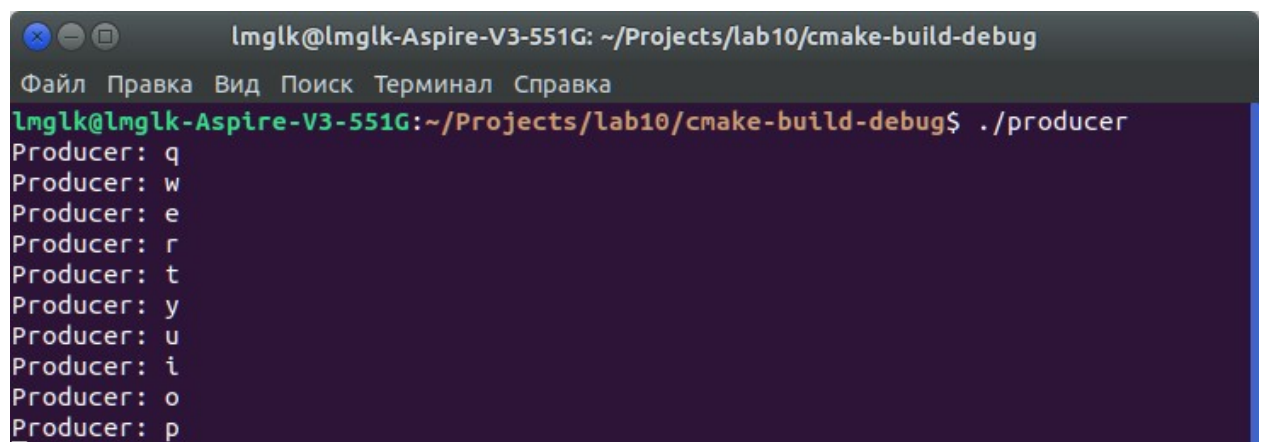
```
void semDecrement(int semid, int semaphoreIndex) {  
    struct sembuf buf{};  
    buf.sem_num = static_cast<unsigned short>(semaphoreIndex);  
    buf.sem_op = -1;  
    buf.sem_flg = 0;  
    semop(semid, &buf, 1);  
}
```

```
void semIncrement(int semid, int semaphoreIndex) {  
    struct sembuf buf{};  
    buf.sem_num = static_cast<unsigned short>(semaphoreIndex);  
    buf.sem_op = 1;  
    buf.sem_flg = 0;  
    semop(semid, &buf, 1);  
}
```

Содержимое файла input.txt:

qwertyuiopasdfghjkl

Запуск программы:



```
lmg1k@lmg1k-Aspire-V3-551G: ~/Projects/lab10/cmake-build-debug  
Файл Правка Вид Поиск Терминал Справка  
lmg1k@lmg1k-Aspire-V3-551G:~/Projects/lab10/cmake-build-debug$ ./producer  
Producer: q  
Producer: w  
Producer: e  
Producer: r  
Producer: t  
Producer: y  
Producer: u  
Producer: i  
Producer: o  
Producer: p
```

```
lmg1k@lmg1k-Aspire-V3-551G: ~/Projects/lab10/cmake-build-debug
Файл Правка Вид Поиск Терминал Справка
lmg1k@lmg1k-Aspire-V3-551G:~/Projects/lab10/cmake-build-debug$ ./consumer
Consumer: q
Consumer: w
Consumer: e
Consumer: r
Consumer: t
Consumer: y
Consumer: u
Consumer: i
Consumer: o
Consumer: p
Consumer: a
Consumer: s
Consumer: d
Consumer: f
Consumer: g
Consumer: h
Consumer: j
Consumer: k
Consumer: l
Consumer:
lmg1k@lmg1k-Aspire-V3-551G:~/Projects/lab10/cmake-build-debug$
```

```
lmg1k@lmg1k-Aspire-V3-551G: ~/Projects/lab10/cmake-build-debug
Файл Правка Вид Поиск Терминал Справка
lmg1k@lmg1k-Aspire-V3-551G:~/Projects/lab10/cmake-build-debug$ ./producer
Producer: q
Producer: w
Producer: e
Producer: r
Producer: t
Producer: y
Producer: u
Producer: i
Producer: o
Producer: p
Producer: a
Producer: s
Producer: d
Producer: f
Producer: g
Producer: h
Producer: j
Producer: k
Producer: l
Producer:
lmg1k@lmg1k-Aspire-V3-551G:~/Projects/lab10/cmake-build-debug$
```

Задание 2. Напишите две программы, экземпляры которых запускаются параллельно и с разной частотой обращаются к общему файлу. Каждый процесс из первой группы (Писатель) пополняет файл определенной

строкой символов и выводит ее на экран вместе с именем программы. Процессы второй группы (Читатели) считывают строки из файла и выводят их на экран. Пока один Писатель записывает строку в файл, другим Писателям и всем Читателям запрещено обращение к файлу. Если Писатели не пишут в файл, то разрешается одновременная работа всех Читателей. Писатель заканчивает работу после того как выполнит N-кратную запись строки в файл. Работа Читателя завершается, когда он прочитал весь файл.

Откомпилируйте программы Читатель и Писатель. Запустите на разных терминалах несколько Писателей и Читателей.

Текст программы writer.cpp:

```
#include <iostream>
#include <sys/sem.h>
#include <fstream>
#include <unistd.h>

#define MUTEX 0
#define FULL 1

using namespace std;

union semun {
    int val;
    struct semid_ds *buf;
    unsigned short *array;
    struct seminfo *__buf;
};

void semaphoreInit(int, int, int);
void semDecrement(int, int);
void semIncrement(int, int);

int main(int argc, char* argv[]) {
    if (argc < 3) {
        cout << "Ошибка. Программа должна принимать имя и количество строк
для записи" << endl;
        return 1;
    }

    string nameProgram = "Writer ";
    nameProgram += argv[1];
    int countRow = atoi(argv[2]);

    int semid = semget(2, 2, 0666 | IPC_CREAT);
    semaphoreInit(semid, MUTEX, 1);
    semaphoreInit(semid, FULL, 0);

    ofstream file("output.txt", ios_base::app);
    srand(static_cast<unsigned int>(time(nullptr)));
    for (int i = 0; i < countRow; i++) {
```

```

        sleep(static_cast<unsigned int>(rand() % 2));
        while(!semctl(semid, MUTEX, GETVAL, 0));
        while(semctl(semid, FULL, GETVAL, 0));
        semDecrement(semid, MUTEX);
        file << nameProgram << ": " << "Test string" << endl;
        cout << nameProgram << ": " << "Test string" << endl;
        semIncrement(semid, MUTEX);
    }
    file.close();

    return 0;
}

void semaphoreInit(int semid, int semaphoreIndex, int value) {
    union semun un;
    un.val = value;
    semctl(semid, semaphoreIndex, SETVAL, un);
}

void semDecrement(int semid, int semaphoreIndex) {
    struct sembuf buf{};
    buf.sem_num = static_cast<unsigned short>(semaphoreIndex);
    buf.sem_op = -1;
    buf.sem_flg = 0;
    semop(semid, &buf, 1);
}

void semIncrement(int semid, int semaphoreIndex) {
    struct sembuf buf{};
    buf.sem_num = static_cast<unsigned short>(semaphoreIndex);
    buf.sem_op = 1;
    buf.sem_flg = 0;
    semop(semid, &buf, 1);
}

```

Текст программы reader.cpp:

```

#include <iostream>
#include <sys/sem.h>
#include <fstream>
#include <unistd.h>

#define MUTEX 0
#define FULL 1

using namespace std;

void semDecrement(int, int);
void semIncrement(int, int);

int main(int argc, char* argv[]) {
    int semid = semget(2, 2, 0666 | IPC_CREAT);

    ifstream file("output.txt");
    string str;
    while(semctl(semid, MUTEX, GETVAL, 0));
    while(!file.eof()){

```


Writer 1: Test string
Writer 2: Test string
Writer 1: Test string
Writer 1: Test string
Writer 2: Test string
Writer 1: Test string
Writer 2: Test string
Writer 2: Test string
Writer 1: Test string
Writer 2: Test string
Writer 1: Test string
Writer 2: Test string
Writer 1: Test string
Writer 1: Test string
Writer 1: Test string
Writer 2: Test string
Writer 1: Test string
Writer 2: Test string
Writer 1: Test string
Writer 2: Test string
Writer 1: Test string
Writer 2: Test string
Writer 1: Test string
Writer 2: Test string
Writer 2: Test string
Writer 1: Test string
Writer 2: Test string
Writer 1: Test string
Writer 2: Test string
Writer 2: Test string
Writer 2: Test string

Вывод

В ходе выполнения лабораторной работы был ознакомлен с организацией семафоров, системными функциями, обеспечивающими управление семафорами, и их использования для решения задач взаимного исключения и синхронизации.