**МИНОБРНАУКИ РОССИИ**

**САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра ВТ**

**ОТЧЕТ**

**по лабораторной работе №10**

**по дисциплине *«Организация процессов и программирование в среде Linux»***

**Тема: «СИНХРОНИЗАЦИЯ ПРОЦЕССОВ**

**С ПОМОЩЬЮ СЕМАФОРОВ»**

Cтудентка гр. 7306 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Мищенко А.В.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Разумовский Г.В.

Санкт–Петербург

2020

**Цель работы:**

Знакомство с организацией семафоров, системными функциями, обеспечивающими управление семафорами, и их использованием для решения задач взаимоисключения и синхронизации.

**Задание:**

Написать две программы (Поставщик и Потребитель), которые работают с циклическим буфером ограниченного размера, расположенным в разделяемой памяти. Доступ к буферу и синхронизация работы Поставщика и Потребителя должны быть реализованы с помощью семафоров. 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 Поставщик должен пассивно ждать, 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.

**Ход работы:**

В ходе выполнения работы были написаны две программы: main.cpp - Поставщик и client.cpp - Потребитель. Их исходный код представлен в Приложении 1.

Программа Поставщик посимвольно считывает текст из файла и передает считанный символ в разделяемую память. Программа Потребитель считывает этот символ из разделяемой памяти и выводит его на экран. Для синхронизации работы Поставщика и Потребителя используется множественный семафор, состоящий из трех числовых семафоров. На рисунках 1 и 2 представлен пример работы программы.



Рисунок 1. Запуск Поставщика

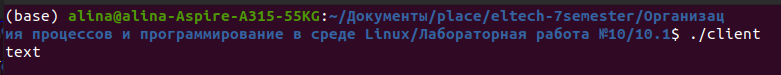


Рисунок 2. Запуск Потребителя

**Вывод:**

В ходе лабораторной работы я познакомилась с использованием множественных семафоров.

**Приложения**

Приложение 1.1. Код поставщика

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <stdlib.h>

#include <assert.h>

#include <iostream>

#include <unistd.h>

#include <fstream>

#include <stdio.h>

using namespace std;

const int SEM\_WRITE = 0,

SEM\_READ = 1,

SEM\_END = 2;

int main()

{

int shmid,

semid,

\*buff;

short buffSize;

key\_t key = 150;

buffSize = 5;

//запрос на разделяемый сегмент памяти

if ((shmid = shmget(key, 128, IPC\_CREAT | 0666)) == -1)

perror("shmget returned -1");

//включение разделяемой памяти в пространство процесса

buff = (int\*)shmat(shmid, NULL, 0);

//создаем множественный семафор(с количеством простых семафоров равным 3)

if ((semid = semget(key, 3, IPC\_CREAT | 0666)) == -1)

perror("semget returned -1");

//объединение для передачи в функцию smctl

union semaphEmun

{

int val;

struct semid\_ds \*buf;

ushort array [2];

} setup;

short values[3] = {5, 0, 0};

semctl(semid, 3, SETALL, values);

//создаем списки команд для работы с множественным семафором

struct sembuf operations[3];

//инициализируем список команд для записи

operations[SEM\_WRITE].sem\_num = SEM\_WRITE; //индекс простого семафора

operations[SEM\_WRITE].sem\_op = -1; //уменьшение на 1

//инициализируем список команд для чтения

operations[SEM\_READ].sem\_num = SEM\_READ;

operations[SEM\_READ].sem\_op = 1; //увеличение на 1 - открываем семафор на чтение

//инициализируем список команд для определения завершения работы

operations[SEM\_END].sem\_num = SEM\_END;

operations[SEM\_END].sem\_op = -1; //уменьшение на 1 - включаем в очередь к семафору

ifstream file;

file.open("file.txt", ios::app);

char c;

int writeIndex = 0;

while(!file.eof())

{

//считываем символ из файла

file.get(c);

//выполняем операцию семафора для записи (1 - количество выполняемых операций, отсчет от первого элемента operations)

semop(semid, operations, 1);

//устанавливаем в разделяемую память символ

buff[writeIndex] = c;

//увеличиваем индекс следующего символа

writeIndex++;

//выполняем операцию семафора для чтения (1 - количество выполняемых операций, отсчет от второго элемента operations)

semop(semid, operations + 1, 1);

}

//устанавливаем спец символ конца файла

buff[writeIndex] = EOF;

//выполняем операцию семафора определения завершения работы

semop(semid, operations + 2, 1);

//закрываем файл и уничтожаем разделяемую память

file.close();

semctl(semid, IPC\_RMID, 0); // Удаляем множество семафоров

shmdt((void\*)buff); // Исключаем сегмент разделяемой памяти из адресного пространства текущего процесса

shmctl(shmid, IPC\_RMID, 0);

}

Приложение 1.2. Код Потребителя

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <stdlib.h>

#include <assert.h>

#include <iostream>

#include <unistd.h>

#include <fstream>

#include <stdio.h>

const int SEM\_WRITE = 0,

SEM\_READ = 1,

SEM\_END = 2;

using namespace std;

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

{

key\_t key = 150;

int shmid,

semid,

buffSize,

\*buff;

buffSize = 5;

//запрос на разделяемый сегмент памяти

if ((shmid = shmget(key, 128, 0666)) == -1)

perror("shmget");

//включение разделяемой памяти в пространство процесса

buff = (int\*)shmat(shmid, NULL, 0);

//открываем множественный семафор(с количеством простых семафоров равным 3)

if ((semid = semget(key, 3, 0666)) == -1)

perror("semget");

//создаем списки команд для работы с множественным семафором

struct sembuf operations[3];

//инициализируем список команд для записи

operations[SEM\_WRITE].sem\_num = SEM\_WRITE;

operations[SEM\_WRITE].sem\_op = 1; //увеличиваем на 1 - открываем семафор на запись

//инициадизируем список команд для чтения

operations[SEM\_READ].sem\_num = SEM\_READ;

operations[SEM\_READ].sem\_op = -1; //уменьшаем на 1 - включаем в очередь к семафору

//инициализируем список команд для определения завершения работы

operations[SEM\_END].sem\_num = SEM\_END;

operations[SEM\_END].sem\_op = 1; //увеличиваем на 1 - открываем семафор на завершение работы

int readIndex = 0;

while(buff[readIndex] != EOF)

{

//выполняем операцию для семафора чтения

semop(semid, operations + 1, 1);

printf("%c", buff[readIndex]);

readIndex++;

//выполняем операцию для семафора записи

semop(semid, operations, 1);

}

//выполняем операцию для семафора завершения работы и тем самым

//позволяем писателю завершить работу

semop(semid, operations + 2, 1);

semctl(semid, IPC\_RMID, 0); // Удаляем множество семафоров

shmdt((void\*)buff); // Исключаем сегмент разделяемой памяти из адресного пространства текущего процесса

shmctl(shmid, IPC\_RMID, 0);

return 0;

}