**МИНОБРНАУКИ РОССИИ**

**САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра ВТ**

**ОТЧЕТ**

**по лабораторной работе №9**

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

**Тема: «ОБМЕН ДАННЫМИ ЧЕРЕЗ**

**РАЗДЕЛЯЕМУЮ ПАМЯТЬ»**

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

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

Санкт–Петербург

2020

**Цель работы:**

Знакомство с организацией разделяемой памяти и системными функциями, обеспечивающими обмен данными между процессами.

**Задание:**

Написать 3 программы, которые запускаются в произвольном порядке и построчно записывают свои индивидуальные данные в один файл через определенный промежуток времени. Пока не закончит писать строку одна программа, другие две не должны обращаться к файлу. Частота записи данных в файл и количество записываемых строк определяются входными параметрами, задаваемыми при запуске каждой программы. При завершении работы одной из программ другие должны продолжить свою работу. Синхронизация работы программ должна осуществляться с помощью общих переменных, размещенных в разделяемой памяти.

**Ход работы:**

В ходе выполнения работы были написаны три программы: exec1.cpp, exec2.cpp и exec3.cpp. Их исходный код представлен в Приложении 1.

При запуске кажого приложения в качестве аргументов нужно передать количество строк, которые запишет каждая из программы и период записи. Каждая программа записывает строки в файл file.txt и для синхронизации их работы используется алгоритм Булочной. На рисунках 1 - 3 представлены запуски приложений, а на рисунке 4 - содержимое файла file.txt. Сначала запускается prog3, prog2 и prog1

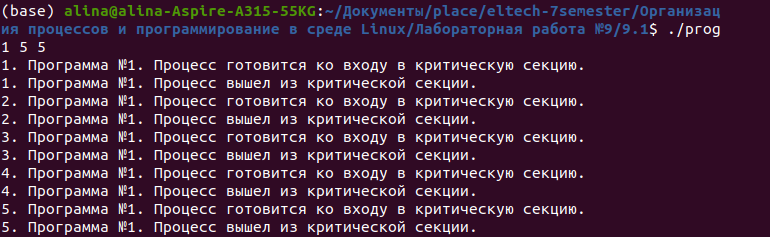


Рисунок 1. Запуск первого приложения

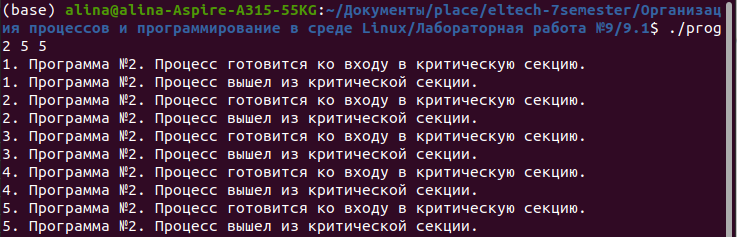


Рисунок 2. Запуск второго приложения

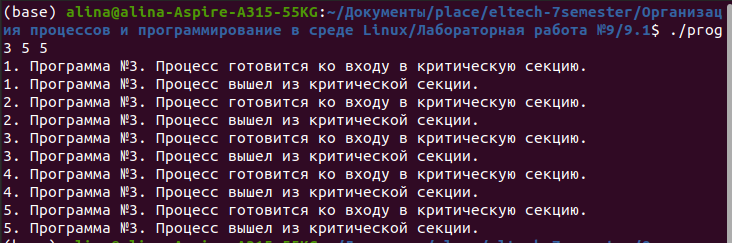


Рисунок 3. Запуск третьего приложения

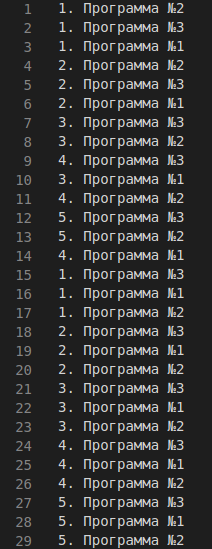


Рисунок 4. Содержимое файла file.txt

**Вывод:**

В ходе лабораторной работы я познакомилась с организацией разделяемой памяти и системными функциями, обеспечивающими обмен данными между процессами.

**Приложения**

Приложение 1.1. Файл exec1.cpp

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <fstream>

#include <assert.h>

const int KEY = 128,

NUM\_PROCESSES = 3,

NUM = 0;

struct shared {

int choosing[NUM\_PROCESSES];

int number[NUM\_PROCESSES];

int stop;

};

std::ofstream file;

//Алгоритм Булочной

void lock();

shared\* shared\_var;

/\* argv[1] - количество строк, argv[2] - период записи \*/

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

int id\_shm;

void\* shmaddr = NULL; /\* Указатель на виртуальный адрес \*/

int count\_str, period, counter;

assert(argc == 3);

count\_str = atoi(argv[1]);

period = atoi(argv[2]);

//запрос на разделяемый сегмент памяти

if ((id\_shm = shmget((key\_t)KEY, sizeof(shared), (0666 | IPC\_CREAT))) == -1) {

perror("shmget");

return -1;

}

//включение разделяемой памяти в пространство процесса

//0 - чтение и запись

if ((int\*)(shmaddr = shmat(id\_shm, NULL, 0)) == (int\*)-1) {

perror("shmat");

}

shared\_var = (shared\*)shmaddr;

counter = 0;

while (counter < count\_str) {

lock();

printf("%d. Программа №%d. Процесс готовится ко входу в критическую секцию.\n", counter + 1, NUM + 1);

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

if(!file)

{

perror("open file");

exit(EXIT\_FAILURE);

}

file << counter + 1 << ". Программа №" << NUM + 1 << std::endl;

file.close();

shared\_var->number[NUM] = 0;

printf("%d. Программа №%d. Процесс вышел из критической секции.\n",counter + 1, NUM+1 );

counter++;

sleep(period);

}

(shared\_var->stop)++;

shared\_var->number[NUM] = 0;

if (shared\_var->stop == NUM\_PROCESSES) {

//отсоединение сегмента

if (shmdt(shmaddr) == -1)

{

perror("shmdt");

}

//удаление сегмента

if (shmctl(id\_shm, IPC\_RMID, NULL) == -1)

{

perror("shmctl+IPC\_RMID");

}

}

return EXIT\_SUCCESS;

}

int max(int a, int b)

{

int res = (a > b) ? a : b;

return res;

}

bool less(int a, int b, int c, int d)

{

if(a < c)

return true;

if (a == c && b < d)

return true;

return false;

}

//Алгоритм булочной

void lock() {

shared\_var->choosing[NUM] = 1;

shared\_var->number[NUM] = 1 + max(shared\_var->number[0],

max(shared\_var->number[1], shared\_var->number[2]));

shared\_var->choosing[NUM] = 0;

for (int j = 0; j < NUM\_PROCESSES; ++j) {

//ждем пока поток j получит свой номер

while (shared\_var->choosing[j]);

//ждем пока все потоки с меньшим номером или с таким же номером, но с более высоким приоритетом, закончат свою работу

while (shared\_var->number[j] != 0 && less(shared\_var->number[j], j, shared\_var->number[NUM],NUM));

}

/\* Критическая секция \*/

}

Приложение 1.2. exec2.cpp

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <fstream>

const int KEY = 128,

NUM\_PROCESS = 3,

NUM = 1;

struct shared {

int choosing[NUM\_PROCESS];

int number[NUM\_PROCESS];

int stop;

};

std::ofstream file;

//Алгоритм Булочной

void lock();

shared\* shared\_var;

/\* argv[1] - количество строк, argv[2] - период записи \*/

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

int id\_shm = -1;

void\* shmaddr = NULL; /\* Указатель на виртуальный адрес \*/

int count\_str, period, counter;

if (argc != 3) {

exit(EXIT\_FAILURE);

} else {

count\_str = atoi(argv[1]);

period = atoi(argv[2]);

}

//запрос на разделяемый сегмент памяти

if ((id\_shm = shmget((key\_t)KEY, sizeof(shared), (0666 | IPC\_CREAT))) == -1) {

perror("req\_shm -> shmget");

return -1;

}

//включение разделяемой памяти в пространство процесса

//0 - чтение и запись

if ((int\*)(shmaddr = shmat(id\_shm, NULL, 0)) == (int\*)-1) {

perror("shmat");

}

shared\_var = (shared\*)shmaddr;

counter = 0;

while (counter < count\_str) {

lock();

printf("%d. Программа №%d. Процесс готовится ко входу в критическую секцию.\n", counter + 1, NUM + 1);

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

if(!file)

{

perror("open file");

exit(EXIT\_FAILURE);

}

file << counter + 1 << ". Программа №" << NUM + 1 << std::endl;

file.close();

shared\_var->number[NUM] = 0;

printf("%d. Программа №%d. Процесс вышел из критической секции.\n",counter + 1, NUM+1 );

counter++;

sleep(period);

}

(shared\_var->stop)++;

shared\_var->number[NUM] = 0;

if (shared\_var->stop == NUM\_PROCESS) {

if (shmdt(shmaddr) == -1)

{

perror("und\_shm -> shmdt");

}

if (shmctl(id\_shm, IPC\_RMID, NULL) == -1)

{

perror("dest\_shm -> shmctl+IPC\_RMID");

}

}

return EXIT\_SUCCESS;

}

int max(int a, int b)

{

int res = (a > b) ? a : b;

return res;

}

bool less(int a, int b, int c, int d)

{

if(a < c)

return true;

if (a == c && b < d)

return true;

return false;

}

void lock() {

shared\_var->choosing[NUM] = 1;

shared\_var->number[NUM] = 1 + max(shared\_var->number[0],

max(shared\_var->number[1], shared\_var->number[2]));

shared\_var->choosing[NUM] = 0;

for (int j = 0; j < NUM\_PROCESS; ++j) {

while (shared\_var->choosing[j]);

while (shared\_var->number[j] != 0 && less(shared\_var->number[j], j, shared\_var->number[NUM],NUM));

}

/\* Критическая секция \*/

}

Приложение 1.3. exec3.cpp

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <fstream>

const int KEY = 128,

NUM\_PROCESS = 3,

NUM = 2;

struct shared {

int choosing[NUM\_PROCESS];

int number[NUM\_PROCESS];

int stop;

};

std::ofstream file;

//Алгоритм Булочной

void lock();

shared\* shared\_var;

/\* argv[1] - количество строк, argv[2] - период записи \*/

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

int id\_shm = -1;

void\* shmaddr = NULL; /\* Указатель на виртуальный адрес \*/

int count\_str, period, counter;

if (argc != 3) {

exit(EXIT\_FAILURE);

} else {

count\_str = atoi(argv[1]);

period = atoi(argv[2]);

}

//запрос на разделяемый сегмент памяти

if ((id\_shm = shmget((key\_t)KEY, sizeof(shared), (0666 | IPC\_CREAT))) == -1) {

perror("req\_shm -> shmget");

return -1;

}

//включение разделяемой памяти в пространство процесса

//0 - чтение и запись

if ((int\*)(shmaddr = shmat(id\_shm, NULL, 0)) == (int\*)-1) {

perror("shmat");

}

shared\_var = (shared\*)shmaddr;

counter = 0;

while (counter < count\_str) {

lock();

printf("%d. Программа №%d. Процесс готовится ко входу в критическую секцию.\n", counter + 1, NUM + 1);

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

if(!file)

{

perror("open file");

exit(EXIT\_FAILURE);

}

file << counter + 1 << ". Программа №" << NUM + 1 << std::endl;

file.close();

shared\_var->number[NUM] = 0;

printf("%d. Программа №%d. Процесс вышел из критической секции.\n",counter + 1, NUM+1 );

counter++;

sleep(period);

}

(shared\_var->stop)++;

shared\_var->number[NUM] = 0;

if (shared\_var->stop == NUM\_PROCESS) {

if (shmdt(shmaddr) == -1)

{

perror("und\_shm -> shmdt");

}

if (shmctl(id\_shm, IPC\_RMID, NULL) == -1)

{

perror("dest\_shm -> shmctl+IPC\_RMID");

}

}

return EXIT\_SUCCESS;

}

int max(int a, int b)

{

int res = (a > b) ? a : b;

return res;

}

bool less(int a, int b, int c, int d)

{

if(a < c)

return true;

if (a == c && b < d)

return true;

return false;

}

void lock() {

shared\_var->choosing[NUM] = 1;

shared\_var->number[NUM] = 1 + max(shared\_var->number[0],

max(shared\_var->number[1], shared\_var->number[2]));

shared\_var->choosing[NUM] = 0;

for (int j = 0; j < NUM\_PROCESS; ++j)

{

while (shared\_var->choosing[j]);

while (shared\_var->number[j] != 0 && less(shared\_var->number[j], j, shared\_var->number[NUM],NUM));

}

/\* Критическая секция \*/

}