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

ОТЧЁТ

по лабораторной работе №9

по дисциплине «Организация процессов и программирования в среде Linux»

Тема: ОБМЕН ДАННЫМИ ЧЕРЕЗ РАЗДЕЛЯЕМУЮ ПАМЯТЬ

Студент гр. 9308 _____ Соболев М.С.
Преподаватель _____ Разумовский Г.В.

Санкт-Петербург,
2022

Оглавление

1. Введение

1.1. Введение

Тема работы: Обмен данными через разделяемую память.

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

1.2. Порядок выполнения работы

1. Написать 3 программы, которые запускаются в произвольном порядке и построчно записывают свои индивидуальные данные в один файл через определённый 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, размещённых в разделяемой памяти.

2. Откомпилировать 3 программы и запустить их на разных терминалах с различными входными параметрами.

3. Написать две программы, которые работают параллельно и обмениваются массивом целых чисел через две общие разделяемые области. Через первую область первая программа передаёт массив второй программе. Через вторую область вторая программа возвращает первой программе массив, каждый элемент которого уменьшен на 1. Обе программы должны вывести получаемую последовательность чисел. Синхронизация работы программ должна осуществляться с помощью общих переменных, размещённых в разделяемой памяти.

4. Откомпилировать 2 программы и запустить их на разных терминалах.

Выбранные задания: 1, 2.

1.3. Содержание отчёта

Отчёт по лабораторной работе должен содержать:

1. Цель и задания.
2. Тексты программ.
3. Скриншоты работы каждой программы.

2. Тексты программ

2.1. executable_0.cpp

```
/*
 * ./executable_0 interval_time number_of_strings
 *
 * interval_time
 *      Interval time for every loop (cycle), i.e. how many times we will wait after start new iteration. Integer number in the range [-1; +inf].
 * number_of_strings
 *      Number of loops (cycles), i.e. how many times program will write strings in the file. Integer number in the range [0; +inf].
 *
 */

#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include <unistd.h>
#include <sys/shm.h>

using namespace std;

typedef struct // struct for lamport algorithm
{
        bool choosing[3]; // array w/ variables, which indicates process is BUSY w/ choosing
        int number[3]; // array w/ variables w/ token (= priority* = number in queue for access the file) numbers
        // * -- word "priority" in this program also means "j" number in loop
} MrLamportIsBaker;

int main (int argc, char *argv[])
{
        // ---------- PREPARING ----------

        if (/*argv[1] == nullptr || */argv[2] == nullptr)
        {
```

```
		cout << "Syntax error. Not enough arguments, must be 2: \"./executable_0 interval_time number_of_strings\"!";
		exit(-1);
	}
	if (atoi(argv[1]) < 1)
	{
		cout << "Syntax error. Interval time to write file argument must be in range [1; +inf)!";
		exit(-1);
	}
	if (atoi(argv[2]) < 1)
	{
		cout << "Syntax error. Number of strings to write file argument must be in range [1; +inf)!";
		exit(-1);
	}

	int program_id = 0; // program id/number
	int interval_time = atoi(argv[1]); // interval time to wait before next start
	int number_of_strings = atoi(argv[2]); // number of strings to write in the file
	int key = 190; // key number for shared memory segment
	string filename = "shared_file.txt"; // name of the file to write strings

	bool shared_mem_seg_owner; // is this process is owner of the shared memory segment (to free it at the end)
	int shared_mem_seg_ptr; // pointer to the shared memory segment
	int i = 0; // for loop
	int j = 0; // for loop
	int k = 0; // for loop
	int local_token = -1; // local token number
	string local_string = "Written by program number " + to_string(program_id) + "\n"; // string to write in file
	MrLamportIsBaker* shared_mem_seg_this_process; // lamport algorithm additional variables

	cout << "---------- PROGRAM NUMBER " << program_id << " ----------\n";
	cout << "---------- OUTPUT FILENAME IS " << filename << " ----------\n";
	cout << "---------- INTERVAL TIME/NUMBER OF STRINGS IS " << interval_time << "/" << number_of_strings << " ----------\n";
	cout << "---------- KEY IS " << (key == IPC_PRIVATE ? "IPC_PRIVATE = " + to_string(key) : to_string(key)) << " ----------\n";

	// ---------- CREATING/OPENING SHARED MEMORY SEGMENT ----------
```

```
	shared_mem_seg_ptr = shmget(key, sizeof(MrLamportIsBaker), 0666 | IPC_CREAT | IPC_EXCL);
	/*
	 * 0400 -- allowed to read to the user who owns shared memory;
	 * 0200 -- write allowed to the user who owns shared memory;
	 * 0040 -- Reading is allowed for users included in that the same group as the owner of the shared memory;
	 * 0020 -- write allowed to users who are members of the same the same group as the owner of the shared memory;
	 * 0004 -- all other users are allowed to read;
	 * 0002 -- all other users are allowed to write;
	 */

	if (shared_mem_seg_ptr != -1)
	{
		shared_mem_seg_owner = true;
		cout << "---------- SHARED MEMORY SEGMENT HAS BEEN CREATED ----------\n\n";
	}
	else
	{
		shared_mem_seg_ptr = shmget(key, sizeof(MrLamportIsBaker), 0666 | IPC_CREAT);
		if(shared_mem_seg_ptr == -1)
		{
			cout << "---------- SHARED MEMORY SEGMENT HAS NOT BEEN OPENED ----------\n\n";
			exit(-1);
		}
		else
		{
			cout << "---------- SHARED MEMORY SEGMENT HAS BEEN OPENED ----------\n\n";
		}
	}
	/*
	 * https://man7.org/linux/man-pages/man2/shmget.2.html
	 * https://www.opennet.ru/man.shtml?topic=shmget&category=2&russian=0
	 * int shmget(key_t key, int size, int shmflg);
	 * on success, a valid shared memory identifier is returned
	 * on error, "-1" is returned, and "errno" is set to indicate the error
	 */
```

```
	// ---------- SHARED MEMORY SEGMENT ATTACH TO THE PROGRAM MEMORY (UNITE THEM) ----------

	shared_mem_seg_this_process = (MrLamportIsBaker*)shmat(shared_mem_seg_ptr, 0, 0);
	/*
	 * https://www.opennet.ru/man.shtml?topic=shmat&category=2&russian=0
	 * https://ru.manpages.org/shmat/2
	 * The "shmat" function attaches the shared memory segment with id = "shmid"
	 * to the address space of the calling process
	 */

	// ---------- LAMPORTH'S ALGORITHM ----------

	for (i = 0; i < number_of_strings; i++) // loop w/ number of strings to write in file = "-num" flag
	{
		// https://www.javatpoint.com/lamports-bakery-algorithm
		// all "entering" ("choosing") variables are initialized to false,
		// and n integer variables "numbers" ("number") are all initialized to 0
		// the value of integer "number" variables is used to form token numbers
		sleep(interval_time); // sleep w/ file write interval = "-time" flag
		shared_mem_seg_this_process->choosing[program_id] = true; // set choosing[???] to true to make other processes aware that it is choosing a token number
		local_token = -1;

		for (k = 0; k < 3; k++)
		{
			// when a process wishes to enter a critical section,
			// it chooses a greater token number than any earlier number
			if (shared_mem_seg_this_process->number[k] > local_token)
			{
				local_token = shared_mem_seg_this_process->number[k]; // choosing maximal token number
			}
		}
```

```
shared_mem_seg_this_process->number[program_id] = local_token + 1; // choosing greater token number
shared_mem_seg_this_process->choosing[program_id] = false; // sets choosing[???] to false after writing token number

// waiting for other processes
for(j = 0; j < 3; j++) // process enters a loop to evaluate the status of other processes
{
	// process "i" waits until some other process "j" is choosing its token number
	while(shared_mem_seg_this_process->choosing[j] == true)
	{}

	// process "i" then waits until all processes with
	// smaller token numbers or the same token number
	// but with higher priority (here -- id or "j") are served fast
	while((shared_mem_seg_this_process->number[j] != 0)
	&& ((shared_mem_seg_this_process->number[j] < shared_mem_seg_this_process->number[program_id])
	|| ((shared_mem_seg_this_process->number[j] == shared_mem_seg_this_process->number[program_id])
	&& (j < program_id))))
	{}
}

cout << "---------- OPEN OUTPUT FILE \"" << filename << "\" BY PROCESS №" << program_id << " BEGIN ----------\n";
ofstream local_file(filename, ios_base::app); // http://cppstudio.com/post/446/
cout << "---------- OPEN OUTPUT FILE \"" << filename << "\" BY PROCESS №" << program_id << " END ----------\n\n";

cout << "---------- WRITE STRING №" << i << " BY PROCESS №" << program_id << " BEGIN ----------\n";
local_file << local_string;
cout << "---------- WRITE STRING №" << i << " BY PROCESS №" << program_id << " END ----------\n\n";
```

```
                cout << "---------- CLOSE OUTPUT FILE \"" << filename << "\" BY PROCESS №" << program_id
<< " BEGIN ----------\n";
                local_file.close();
                cout << "---------- CLOSE OUTPUT FILE \"" << filename << "\" BY PROCESS №" << program_id
<< " END ----------\n\n";

                // when the process has finished with its critical section execution,
                // it resets its number variable to 0
                shared_mem_seg_this_process->number[program_id] = 0;
        }

        // ---------- SHARED MEMORY SEGMENT DETACH FROM THE PROGRAM MEMORY (SEPARATE
THEM) ----------

        shmdt((void*)shared_mem_seg_this_process);
        /*
         * https://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/shm/shmdt.html
         * shmdt(shm_ptr);
         *
         * system call "shmdt" is used to detach a shared memory;
         * after a shared memory is detached, it cannot be used in process;
         * but it is still there and can be re-attached back to a adress space of process,
         * perhaps at a different address;
         * "shared_mem_seg_this_process" -- argument of the call to "shmdt", the shared memory address returned by
"shmat";
         */

        // ---------- CLEANING & TERMINATING ----------

        if(shared_mem_seg_owner == true)
        {
                // https://en.cppreference.com/w/cpp/types/NULL
                shmctl(shared_mem_seg_ptr, IPC_RMID, NULL);
                cout << "\n---------- SHARED MEMORY SEGMENT HAS BEEN CLOSED ----------\n";
        }
        /*
         * https://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/shm/shmdt.html
```

```
         * shmctl(shm_id, IPC_RMID, NULL);
         *
         * to remove a shared memory, use "shmctl" function;
         * "shared_mem_seg_ptr" is the shared memory ID;
         * "IPC_RMID" indicates this is a remove operation;
         * if you want to use it again, you should use "shmget" followed by "shmat";
         */


        return 0;
}
```

2.2. executable_1.cpp

```
/*
 * ./executable_1 interval_time number_of_strings
 *
 * interval_time
 *      Interval time for every loop (cycle), i.e. how many times we will wait after start new iteration. Integer number in the range [-1; +inf].
 * number_of_strings
 *      Number of loops (cycles), i.e. how many times program will write strings in the file. Integer number in the range [0; +inf].
 *
 */

#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include <unistd.h>
#include <sys/shm.h>

using namespace std;

typedef struct // struct for lamport algorithm
{
        bool choosing[3]; // array w/ variables, which indicates process is BUSY w/ choosing
        int number[3]; // array w/ variables w/ token (= priority* = number in queue for access the file) numbers
        // * -- word "priority" in this program also means "j" number in loop
} MrLamportIsBaker;

int main (int argc, char *argv[])
{
        // ---------- PREPARING ----------

        if (/*argv[1] == nullptr || */argv[2] == nullptr)
        {
                cout << "Syntax error. Not enough arguments, must be 2: \"./executable_1 interval_time number_of_strings\"!";
                exit(-1);
```

```
	}
	if (atoi(argv[1]) < 1)
	{
		cout << "Syntax error. Interval time to write file argument must be in range [1; +inf)!";
		exit(-1);
	}
	if (atoi(argv[2]) < 1)
	{
		cout << "Syntax error. Number of strings to write file argument must be in range [1; +inf)!";
		exit(-1);
	}

	int program_id = 1; // program id/number
	int interval_time = atoi(argv[1]); // interval time to wait before next start
	int number_of_strings = atoi(argv[2]); // number of strings to write in the file
	int key = 190; // key number for shared memory segment
	string filename = "shared_file.txt"; // name of the file to write strings

	bool shared_mem_seg_owner; // is this process is owner of the shared memory segment (to free it at the end)
	int shared_mem_seg_ptr; // pointer to the shared memory segment
	int i = 0; // for loop
	int j = 0; // for loop
	int k = 0; // for loop
	int local_token = -1; // local token number
	string local_string = "Written by program number " + to_string(program_id) + "\n"; // string to write in file
	MrLamportIsBaker* shared_mem_seg_this_process; // lamport algorithm additional variables

	cout << "---------- PROGRAM NUMBER " << program_id << " ----------\n";
	cout << "---------- OUTPUT FILENAME IS " << filename << " ----------\n";
	cout << "---------- INTERVAL TIME/NUMBER OF STRINGS IS " << interval_time << "/" << number_of_strings << " ----------\n";
	cout << "---------- KEY IS " << (key == IPC_PRIVATE ? "IPC_PRIVATE = " + to_string(key) : to_string(key)) << " ----------\n";

	// ---------- CREATING/OPENING SHARED MEMORY SEGMENT ----------

	shared_mem_seg_ptr = shmget(key, sizeof(MrLamportIsBaker), 0666 | IPC_CREAT | IPC_EXCL);
```

```
	/*
	 * 0400 -- allowed to read to the user who owns shared memory;
	 * 0200 -- write allowed to the user who owns shared memory;
	 * 0040 -- Reading is allowed for users included in that the same group as the owner of the shared memory;
	 * 0020 -- write allowed to users who are members of the same the same group as the owner of the shared memory;
	 * 0004 -- all other users are allowed to read;
	 * 0002 -- all other users are allowed to write;
	 */


	if (shared_mem_seg_ptr != -1)
	{
		shared_mem_seg_owner = true;
		cout << "---------- SHARED MEMORY SEGMENT HAS BEEN CREATED ----------\n\n";
	}
	else
	{
		shared_mem_seg_ptr = shmget(key, sizeof(MrLamportIsBaker), 0666 | IPC_CREAT);
		if(shared_mem_seg_ptr == -1)
		{
			cout << "---------- SHARED MEMORY SEGMENT HAS NOT BEEN OPENED ----------\n\n";
			exit(-1);
		}
		else
		{
			cout << "---------- SHARED MEMORY SEGMENT HAS BEEN OPENED ----------\n\n";
		}
	}
	/*
	 * https://man7.org/linux/man-pages/man2/shmget.2.html
	 * https://www.opennet.ru/man.shtml?topic=shmget&category=2&russian=0
	 * int shmget(key_t key, int size, int shmflg);
	 * on success, a valid shared memory identifier is returned
	 * on error, "-1" is returned, and "errno" is set to indicate the error
	 */
```

```
	// ---------- SHARED MEMORY SEGMENT ATTACH TO THE PROGRAM MEMORY (UNITE THEM) ----------

	shared_mem_seg_this_process = (MrLamportIsBaker*)shmat(shared_mem_seg_ptr, 0, 0);
	/*
	 * https://www.opennet.ru/man.shtml?topic=shmat&category=2&russian=0
	 * https://ru.manpages.org/shmat/2
	 * The "shmat" function attaches the shared memory segment with id = "shmid"
	 * to the address space of the calling process
	 */

	// ---------- LAMPORTH'S ALGORITHM ----------

	for (i = 0; i < number_of_strings; i++) // loop w/ number of strings to write in file = "-num" flag
	{
		// https://www.javatpoint.com/lamports-bakery-algorithm
		// all "entering" ("choosing") variables are initialized to false,
		// and n integer variables "numbers" ("number") are all initialized to 0
		// the value of integer "number" variables is used to form token numbers
		sleep(interval_time); // sleep w/ file write interval = "-time" flag
		shared_mem_seg_this_process->choosing[program_id] = true; // set choosing[???] to true to make other processes aware that it is choosing a token number
		local_token = -1;

		for (k = 0; k < 3; k++)
		{
			// when a process wishes to enter a critical section,
			// it chooses a greater token number than any earlier number
			if (shared_mem_seg_this_process->number[k] > local_token)
			{
				local_token = shared_mem_seg_this_process->number[k]; // choosing maximal token number
			}
		}

		shared_mem_seg_this_process->number[program_id] = local_token + 1; // choosing greater token number
```

```
shared_mem_seg_this_process->choosing[program_id] = false; // sets choosing[???] to false after writing token number

// waiting for other processes
for(j = 0; j < 3; j++) // process enters a loop to evaluate the status of other processes
{
	// process "i" waits until some other process "j" is choosing its token number
	while(shared_mem_seg_this_process->choosing[j] == true)
	{}

	// process "i" then waits until all processes with
	// smaller token numbers or the same token number
	// but with higher priority (here -- id or "j") are served fast
	while((shared_mem_seg_this_process->number[j] != 0)
	&& ((shared_mem_seg_this_process->number[j] < shared_mem_seg_this_process->number[program_id])
	|| ((shared_mem_seg_this_process->number[j] == shared_mem_seg_this_process->number[program_id])
	&& (j < program_id))))
	{}
}

cout << "---------- OPEN OUTPUT FILE \"" << filename << "\" BY PROCESS №" << program_id << " BEGIN ----------\n";
ofstream local_file(filename, ios_base::app); // http://cppstudio.com/post/446/
cout << "---------- OPEN OUTPUT FILE \"" << filename << "\" BY PROCESS №" << program_id << " END ----------\n\n";

cout << "---------- WRITE STRING №" << i << " BY PROCESS №" << program_id << " BEGIN ----------\n";
local_file << local_string;
cout << "---------- WRITE STRING №" << i << " BY PROCESS №" << program_id << " END ----------\n\n";

cout << "---------- CLOSE OUTPUT FILE \"" << filename << "\" BY PROCESS №" << program_id << " BEGIN ----------\n";
local_file.close();
```

```
                cout << "---------- CLOSE OUTPUT FILE \"" << filename << "\" BY PROCESS №" << program_id
<< " END ----------\n\n";

                // when the process has finished with its critical section execution,
                // it resets its number variable to 0
                shared_mem_seg_this_process->number[program_id] = 0;
        }

        // ---------- SHARED MEMORY SEGMENT DETACH FROM THE PROGRAM MEMORY (SEPARATE
THEM) ----------

        shmdt((void*)shared_mem_seg_this_process);
        /*
         * https://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/shm/shmdt.html
         * shmdt(shm_ptr);
         *
         * system call "shmdt" is used to detach a shared memory;
         * after a shared memory is detached, it cannot be used in process;
         * but it is still there and can be re-attached back to a adress space of process,
         * perhaps at a different address;
         * "shared_mem_seg_this_process" -- argument of the call to "shmdt", the shared memory address returned by
"shmat";
         */

        // ---------- CLEANING & TERMINATING ----------

        if(shared_mem_seg_owner == true)
        {
                // https://en.cppreference.com/w/cpp/types/NULL
                shmctl(shared_mem_seg_ptr, IPC_RMID, NULL);
                cout << "\n---------- SHARED MEMORY SEGMENT HAS BEEN CLOSED ----------\n";
        }
        /*
         * https://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/shm/shmdt.html
         * shmctl(shm_id, IPC_RMID, NULL);
         *
         * to remove a shared memory, use "shmctl" function;
```

```
         * "shared_mem_seg_ptr" is the shared memory ID;
         * "IPC_RMID" indicates this is a remove operation;
         * if you want to use it again, you should use "shmget" followed by "shmat";
         */


        return 0;
}
```

2.3. executable_2.cpp

```
/*
 * ./executable_2 interval_time number_of_strings
 *
 * interval_time
 *      Interval time for every loop (cycle), i.e. how many times we will wait after start new iteration. Integer number in the range [-1; +inf].
 * number_of_strings
 *      Number of loops (cycles), i.e. how many times program will write strings in the file. Integer number in the range [0; +inf].
 *
 */

#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include <unistd.h>
#include <sys/shm.h>

using namespace std;

typedef struct // struct for lamport algorithm
{
        bool choosing[3]; // array w/ variables, which indicates process is BUSY w/ choosing
        int number[3]; // array w/ variables w/ token (= priority* = number in queue for access the file) numbers
        // * -- word "priority" in this program also means "j" number in loop
} MrLamportIsBaker;

int main (int argc, char *argv[])
{
        // ---------- PREPARING ----------

        if (/*argv[1] == nullptr || */argv[2] == nullptr)
        {
                cout << "Syntax error. Not enough arguments, must be 2: \"./executable_2 interval_time number_of_strings\"!";
                exit(-1);
```

```
	}
	if (atoi(argv[1]) < 1)
	{
		cout << "Syntax error. Interval time to write file argument must be in range [1; +inf)!";
		exit(-1);
	}
	if (atoi(argv[2]) < 1)
	{
		cout << "Syntax error. Number of strings to write file argument must be in range [1; +inf)!";
		exit(-1);
	}

	int program_id = 2; // program id/number
	int interval_time = atoi(argv[1]); // interval time to wait before next start
	int number_of_strings = atoi(argv[2]); // number of strings to write in the file
	int key = 190; // key number for shared memory segment
	string filename = "shared_file.txt"; // name of the file to write strings

	bool shared_mem_seg_owner; // is this process is owner of the shared memory segment (to free it at the end)
	int shared_mem_seg_ptr; // pointer to the shared memory segment
	int i = 0; // for loop
	int j = 0; // for loop
	int k = 0; // for loop
	int local_token = -1; // local token number
	string local_string = "Written by program number " + to_string(program_id) + "\n"; // string to write in file
	MrLamportIsBaker* shared_mem_seg_this_process; // lamport algorithm additional variables

	cout << "---------- PROGRAM NUMBER " << program_id << " ----------\n";
	cout << "---------- OUTPUT FILENAME IS " << filename << " ----------\n";
	cout << "---------- INTERVAL TIME/NUMBER OF STRINGS IS " << interval_time << "/" << number_of_strings << " ----------\n";
	cout << "---------- KEY IS " << (key == IPC_PRIVATE ? "IPC_PRIVATE = " + to_string(key) : to_string(key)) << " ----------\n";

	// ---------- CREATING/OPENING SHARED MEMORY SEGMENT ----------

	shared_mem_seg_ptr = shmget(key, sizeof(MrLamportIsBaker), 0666 | IPC_CREAT | IPC_EXCL);
```

```
	/*
	 * 0400 -- allowed to read to the user who owns shared memory;
	 * 0200 -- write allowed to the user who owns shared memory;
	 * 0040 -- Reading is allowed for users included in that the same group as the owner of the shared memory;
	 * 0020 -- write allowed to users who are members of the same the same group as the owner of the shared memory;
	 * 0004 -- all other users are allowed to read;
	 * 0002 -- all other users are allowed to write;
	 */


	if (shared_mem_seg_ptr != -1)
	{
		shared_mem_seg_owner = true;
		cout << "---------- SHARED MEMORY SEGMENT HAS BEEN CREATED ----------\n\n";
	}
	else
	{
		shared_mem_seg_ptr = shmget(key, sizeof(MrLamportIsBaker), 0666 | IPC_CREAT);
		if(shared_mem_seg_ptr == -1)
		{
			cout << "---------- SHARED MEMORY SEGMENT HAS NOT BEEN OPENED ----------\n\n";
			exit(-1);
		}
		else
		{
			cout << "---------- SHARED MEMORY SEGMENT HAS BEEN OPENED ----------\n\n";
		}
	}
	/*
	 * https://man7.org/linux/man-pages/man2/shmget.2.html
	 * https://www.opennet.ru/man.shtml?topic=shmget&category=2&russian=0
	 * int shmget(key_t key, int size, int shmflg);
	 * on success, a valid shared memory identifier is returned
	 * on error, "-1" is returned, and "errno" is set to indicate the error
	 */
```

```
	// ---------- SHARED MEMORY SEGMENT ATTACH TO THE PROGRAM MEMORY (UNITE THEM) ----------

	shared_mem_seg_this_process = (MrLamportIsBaker*)shmat(shared_mem_seg_ptr, 0, 0);
	/*
	 * https://www.opennet.ru/man.shtml?topic=shmat&category=2&russian=0
	 * https://ru.manpages.org/shmat/2
	 * The "shmat" function attaches the shared memory segment with id = "shmid"
	 * to the address space of the calling process
	 */

	// ---------- LAMPORTH'S ALGORITHM ----------

	for (i = 0; i < number_of_strings; i++) // loop w/ number of strings to write in file = "-num" flag
	{
		// https://www.javatpoint.com/lamports-bakery-algorithm
		// all "entering" ("choosing") variables are initialized to false,
		// and n integer variables "numbers" ("number") are all initialized to 0
		// the value of integer "number" variables is used to form token numbers
		sleep(interval_time); // sleep w/ file write interval = "-time" flag
		shared_mem_seg_this_process->choosing[program_id] = true; // set choosing[???] to true to make other processes aware that it is choosing a token number
		local_token = -1;

		for (k = 0; k < 3; k++)
		{
			// when a process wishes to enter a critical section,
			// it chooses a greater token number than any earlier number
			if (shared_mem_seg_this_process->number[k] > local_token)
			{
				local_token = shared_mem_seg_this_process->number[k]; // choosing maximal token number
			}
		}

		shared_mem_seg_this_process->number[program_id] = local_token + 1; // choosing greater token number
```

```
shared_mem_seg_this_process->choosing[program_id] = false; // sets choosing[???] to false after writing token number

// waiting for other processes
for(j = 0; j < 3; j++) // process enters a loop to evaluate the status of other processes
{
	// process "i" waits until some other process "j" is choosing its token number
	while(shared_mem_seg_this_process->choosing[j] == true)
	{}

	// process "i" then waits until all processes with
	// smaller token numbers or the same token number
	// but with higher priority (here -- id or "j") are served fast
	while((shared_mem_seg_this_process->number[j] != 0)
	&& ((shared_mem_seg_this_process->number[j] < shared_mem_seg_this_process->number[program_id])
	|| ((shared_mem_seg_this_process->number[j] == shared_mem_seg_this_process->number[program_id])
	&& (j < program_id))))
	{}
}

cout << "---------- OPEN OUTPUT FILE \"" << filename << "\" BY PROCESS №" << program_id << " BEGIN ----------\n";
ofstream local_file(filename, ios_base::app); // http://cppstudio.com/post/446/
cout << "---------- OPEN OUTPUT FILE \"" << filename << "\" BY PROCESS №" << program_id << " END ----------\n\n";

cout << "---------- WRITE STRING №" << i << " BY PROCESS №" << program_id << " BEGIN ----------\n";
local_file << local_string;
cout << "---------- WRITE STRING №" << i << " BY PROCESS №" << program_id << " END ----------\n\n";

cout << "---------- CLOSE OUTPUT FILE \"" << filename << "\" BY PROCESS №" << program_id << " BEGIN ----------\n";
local_file.close();
```

```
                cout << "---------- CLOSE OUTPUT FILE \"" << filename << "\" BY PROCESS №" << program_id
<< " END ----------\n\n";

                // when the process has finished with its critical section execution,
                // it resets its number variable to 0
                shared_mem_seg_this_process->number[program_id] = 0;
        }

        // ---------- SHARED MEMORY SEGMENT DETACH FROM THE PROGRAM MEMORY (SEPARATE
THEM) ----------

        shmdt((void*)shared_mem_seg_this_process);
        /*
         * https://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/shm/shmdt.html
         * shmdt(shm_ptr);
         *
         * system call "shmdt" is used to detach a shared memory;
         * after a shared memory is detached, it cannot be used in process;
         * but it is still there and can be re-attached back to a adress space of process,
         * perhaps at a different address;
         * "shared_mem_seg_this_process" -- argument of the call to "shmdt", the shared memory address returned by
"shmat";
         */

        // ---------- CLEANING & TERMINATING ----------

        if(shared_mem_seg_owner == true)
        {
                // https://en.cppreference.com/w/cpp/types/NULL
                shmctl(shared_mem_seg_ptr, IPC_RMID, NULL);
                cout << "\n---------- SHARED MEMORY SEGMENT HAS BEEN CLOSED ----------\n";
        }
        /*
         * https://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/shm/shmdt.html
         * shmctl(shm_id, IPC_RMID, NULL);
         *
         * to remove a shared memory, use "shmctl" function;
```

```
         * "shared_mem_seg_ptr" is the shared memory ID;
         * "IPC_RMID" indicates this is a remove operation;
         * if you want to use it again, you should use "shmget" followed by "shmat";
         */


        return 0;
}
```

3. Скриншоты работы каждой программы

Программы запускаются последовательно согласно их нумерации в названии, то есть сначала «executable_0», затем «executable_1», а в конце «executable_2». Нумерация программ начинается с 0, а не с 1. Это связано с прямым соответствием номеров (идентификаторов) программ с номерами ячеек массивов «bool choosing» и «int number» для контроля доступа к общим ресурсам.

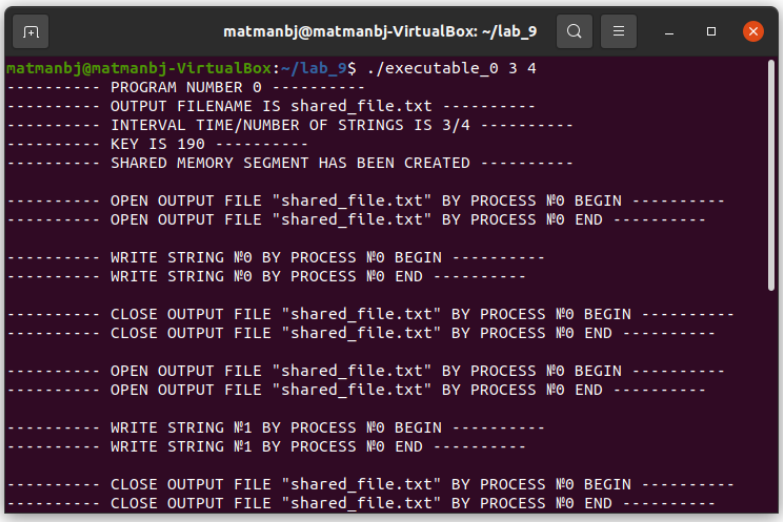

Рисунок 1. Запуск программы «executable_0» с параметром интервала «3» и с параметром количества записываемых строк «4»

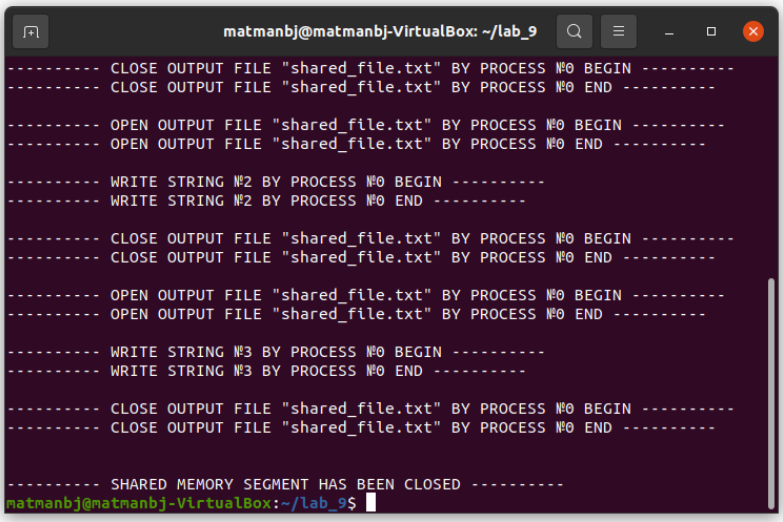

Рисунок 2. Запуск программы «executable_0» с параметром интервала «3» и с параметром количества записываемых строк «4»

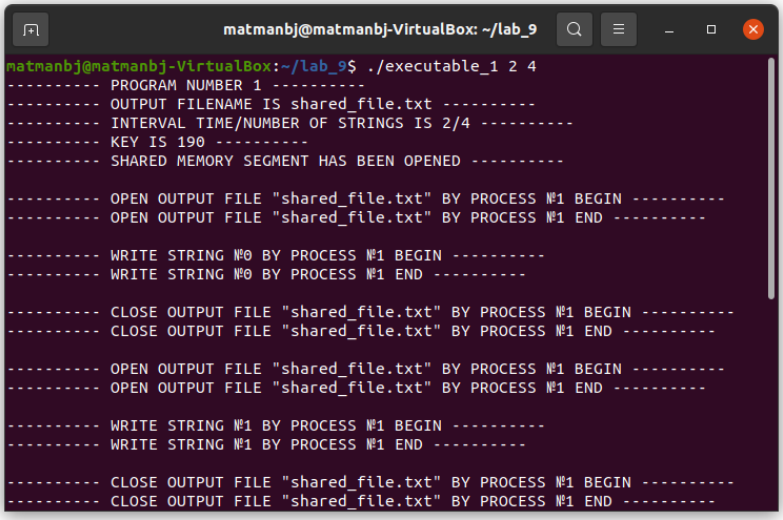

Рисунок 3. Запуск программы «executable_1» с параметром интервала «2» и с параметром количества записываемых строк «4»

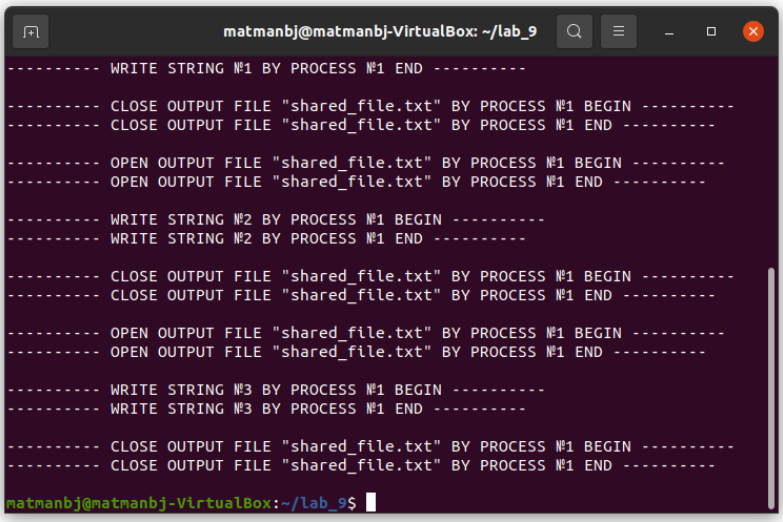

Рисунок 4. Запуск программы «executable_1» с параметром интервала «2» и с параметром количества записываемых строк «4»

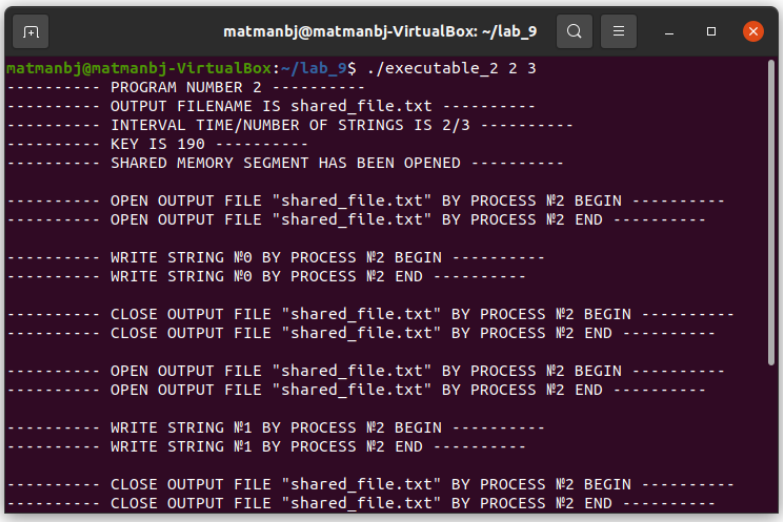

Рисунок 5. Запуск программы «executable_2» с параметром интервала «2» и с параметром количества записываемых строк «3»

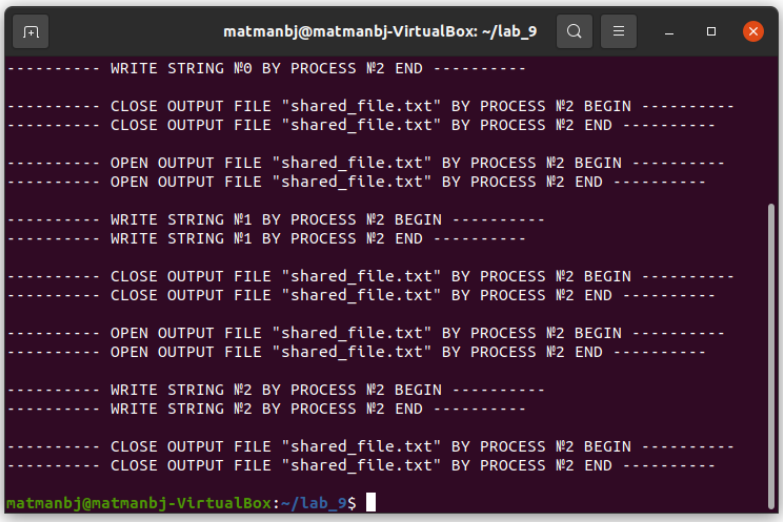

Рисунок 6. Запуск программы «executable_2» с параметром интервала «2» и с параметром количества записываемых строк «3»

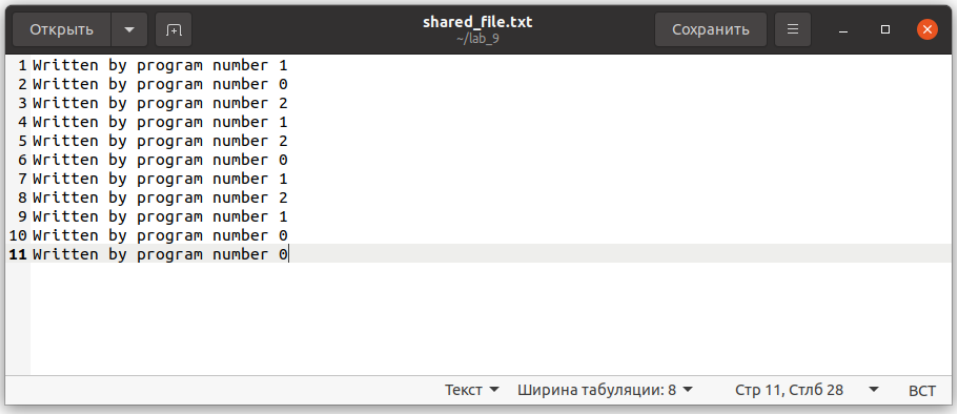

Рисунок 7. Выходной файл «shared_file.txt»

4. Вывод

В ходе выполнения лабораторной работы №9 «Обмен данными через разделяемую память» были изучены системные функции, отвечающие за выделение разделяемой памяти или подключение к ней («shmget»), за присоединение выделенного или подключённого сегмента к адресному пространству текущего процесса («shmat»), за отсоединение выделенного или подключённого сегмента от адресного пространства текущего процесса («shmdt») и за освобождение разделяемой памяти («shmctl»). Во время работы процессы синхронизировались с помощью сегмента разделяемой памяти, синхронизируясь по алгоритму Лампорта и поочерёдно записывая данные в файл. Таким образом и было произведено знакомство с организацией разделяемой памяти и системными функциями, обеспечивающими обмен данными между процессами.

5. Список использованных источников

1. Онлайн-курс «Организация процессов и программирование в среде Linux» в LMS Moodle [сайт]. URL: https://vec.etu.ru/moodle/course/view.php?id=9703.

2. Разумовский Г.В. Организация процессов и программирование в среде Linux: учебно-методическое пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2018. 40с.