**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной техники**

**ОТЧЕТ**

**по лабораторной работе №9**

**по дисциплине «Организация процессов и программирование**

**В среде Linux»**

**Тема: ОБМЕН ДАННЫМИ ЧЕРЕЗ РАЗДЕЛЯЕМУЮ ПАМЯТЬ**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9308 |  | Яловега Н.В. |
| Преподаватель |  | Разумовский Г.В. |

Санкт-Петербург

2022

**Цель работы**

Целью лабораторной работы является 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.

**Примеры выполнения программы**

Скриншоты выполнения программ представлены на рис 1.

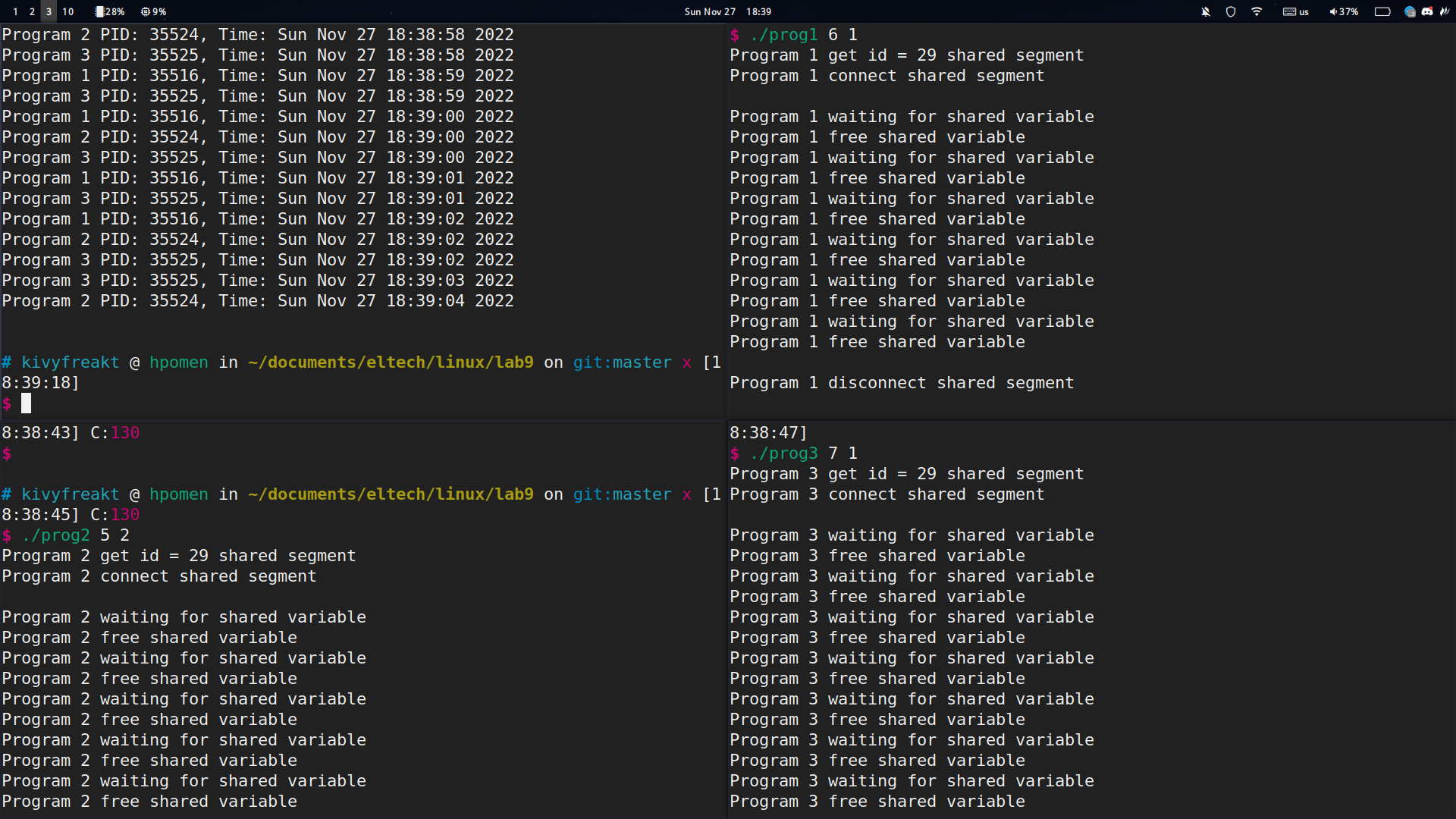


Рисунок 1

Данные выходного файла представлены на рис. 4.

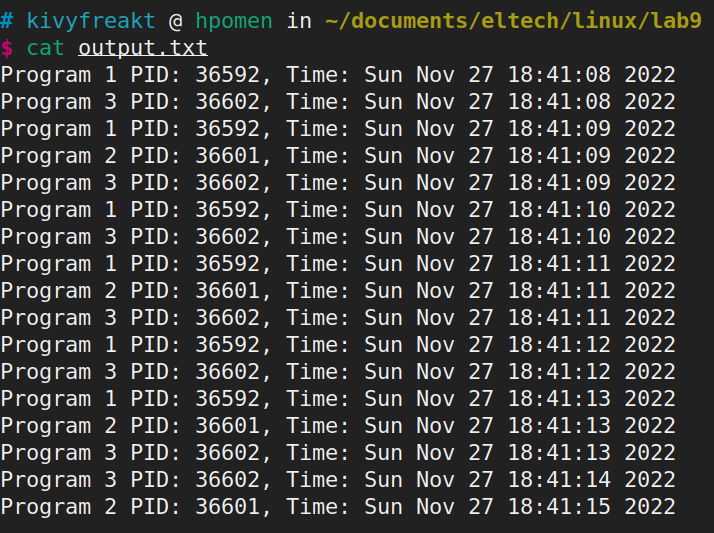


Рисунок 2

**Исходный код**

prog1.cpp

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/time.h>

#include <iostream>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <signal.h>

const int NUM\_PROCESS = 3;

const int NUM = 1;

struct shared

{

int waiting\_ticket[NUM\_PROCESS]; // какой процесс ждет билет

int values[NUM\_PROCESS]; // какой номер билета получил процесс

};

shared\* shared\_var;

// Алгоритм Лампорта (булочной)

void lock(shared\* shared\_var, int process)

{

// стадия ожидания билетика

shared\_var->waiting\_ticket[process] = 1;

// стадия получения значения для билетика

shared\_var->values[process] = 1 + std::max(shared\_var->values[0], std::max(shared\_var->values[1], shared\_var->values[2]));

shared\_var->waiting\_ticket[process] = 0;

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

{

if (i != process)

{

// Если процесс еще ждет билет, то ждем:

while (shared\_var->waiting\_ticket[i]);

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

while (shared\_var->values[i] > 0 // Если значение было проинициализировано

&& (shared\_var->values[i] < shared\_var->values[process]) // И значение билетика было раньше

|| (shared\_var->values[process] == shared\_var->values[i] && i < process)); // Или если получили в одно время, то по номеру

}

}

}

void unlock(shared\* shared\_var, int process)

{

shared\_var->values[process] = 0;

}

void signal\_handler(int sig)

{

if (sig == SIGALRM)

{

std::cout << "Program "<< NUM << " waiting for shared variable\n";

lock(shared\_var, NUM);

FILE \*file = fopen("output.txt", "a");

time\_t curTime = time(NULL);

fprintf(file, "Program %d PID: %d, Time: %s", NUM, getpid(), ctime(&curTime));

fclose(file);

unlock(shared\_var, NUM);

std::cout << "Program " << NUM << " free shared variable\n";

}

}

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

{

if(argc == 3)

{

int starts = atoi(argv[1]);

int period = atoi(argv[2]);

if (starts == 0)

return 0;

if (period == 0)

return 0;

signal(SIGALRM, signal\_handler);

struct itimerval timer\_value;

timerclear(&timer\_value.it\_interval);

timerclear(&timer\_value.it\_value);

timer\_value.it\_interval.tv\_sec = period;

timer\_value.it\_value.tv\_sec = period;

setitimer(ITIMER\_REAL, &timer\_value, NULL);

// создание разделяемого сегмента памяти

// 123 - ключ

// sizeof(shared) - размер сегмента

// 0666 - чтение и запись вообще всем

int shm\_id = shmget(123, sizeof(shared), (0666 | IPC\_CREAT));

if(shm\_id != -1)

std::cout << "Program " << NUM << " get id = " << shm\_id << " shared segment\n";

else

return -1;

// получение виртуального адреса, по которому сегмент был привязан к процессу

// shm\_id - идентификатор сегмента

// 0 или адрес, по которому присоединить

// 0 - флаги

void\* shm\_addr = shmat(shm\_id, 0, 0);

if(\*(int\*)shm\_addr != -1)

std::cout << "Program " << NUM << " connect shared segment\n\n";

else

return -1;

// получение данных из разделяемой памяти

shared\_var = (shared\*)shm\_addr;

for(int i = 0; i < starts; i++)

{

pause();

}

// Отсоединение сегмента

if(shmdt(shm\_addr) != -1)

std::cout << "\nProgram " << NUM << " disconnect shared segment\n";

}

else

std::cout << "Error: " << argv[0] << " starts period\n";

return 0;

}

**Вывод**

При выполнении лабораторной работы изучена организация разделяемой памяти и системные функции, обеспечивающие обмен данными между процессами; написаны три программы, построчно записывающие в выходной файл свои данные. Синхронизация работы программ выполнена с использованием алгоритма Лампорта и общих переменных, размещенных в разделяемой памяти.