

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

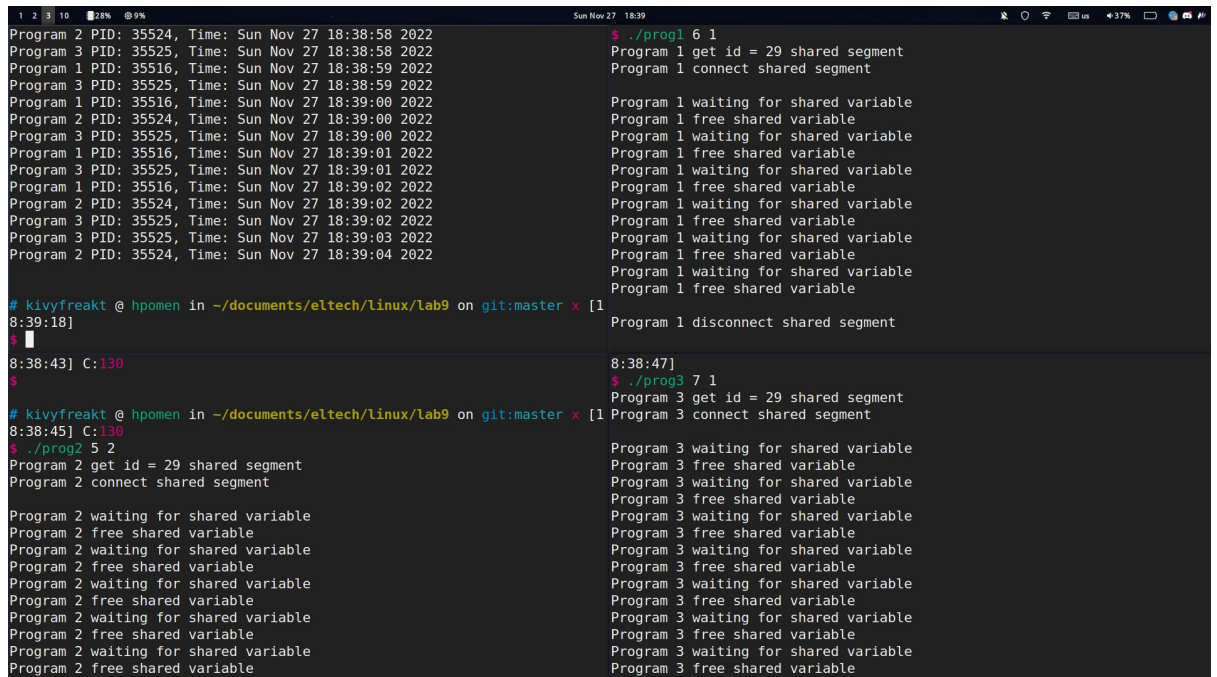
Целью лабораторной работы является знакомство с организацией разделяемой памяти и системными функциями, обеспечивающими обмен данными между процессами.

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.



```
1 2 3 10 28% @ 9% Sun Nov 27 18:39
Program 2 PID: 35524, Time: Sun Nov 27 18:38:58 2022
Program 3 PID: 35525, Time: Sun Nov 27 18:38:58 2022
Program 1 PID: 35516, Time: Sun Nov 27 18:38:59 2022
Program 3 PID: 35525, Time: Sun Nov 27 18:38:59 2022
Program 1 PID: 35516, Time: Sun Nov 27 18:39:00 2022
Program 2 PID: 35524, Time: Sun Nov 27 18:39:00 2022
Program 3 PID: 35525, Time: Sun Nov 27 18:39:00 2022
Program 1 PID: 35516, Time: Sun Nov 27 18:39:01 2022
Program 3 PID: 35525, Time: Sun Nov 27 18:39:01 2022
Program 1 PID: 35516, Time: Sun Nov 27 18:39:02 2022
Program 2 PID: 35524, Time: Sun Nov 27 18:39:02 2022
Program 3 PID: 35525, Time: Sun Nov 27 18:39:02 2022
Program 3 PID: 35525, Time: Sun Nov 27 18:39:03 2022
Program 2 PID: 35524, Time: Sun Nov 27 18:39:04 2022

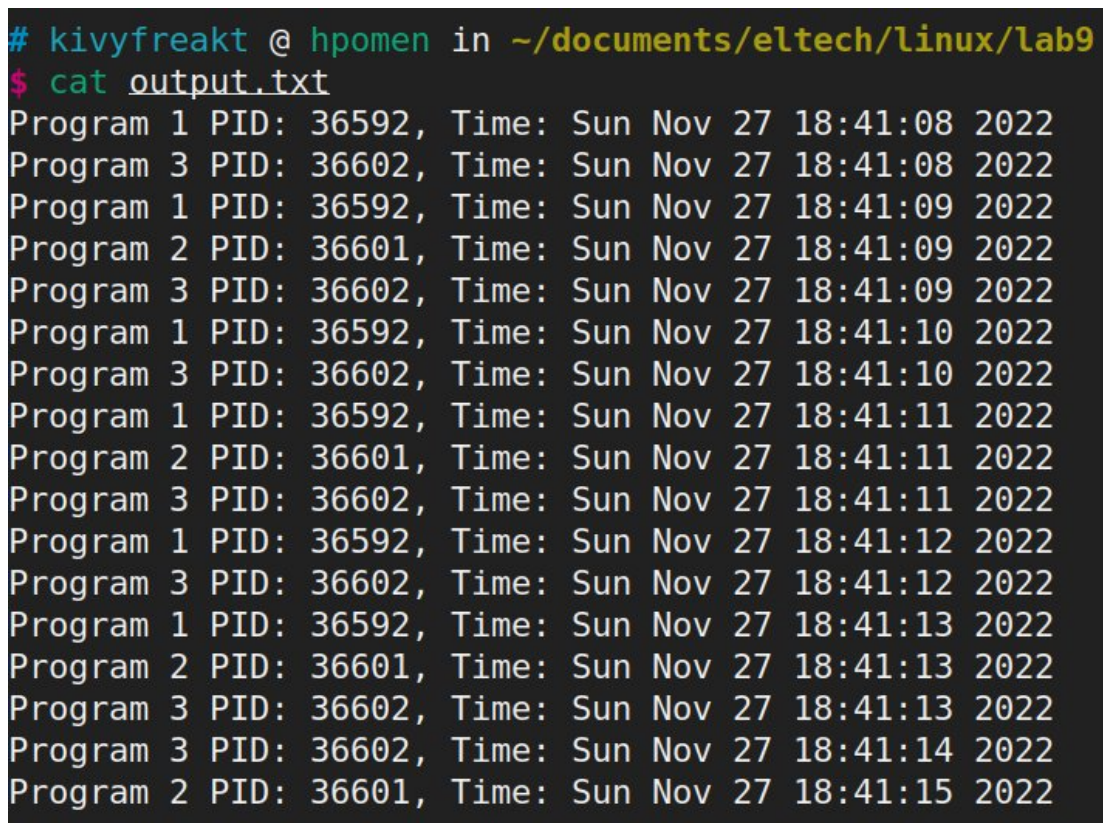
# kivyfreakt @ hpomen in ~/documents/eltech/linux/lab9 on git:master x [1
8:39:18]
$
8:38:43] C:130
$
# kivyfreakt @ hpomen in ~/documents/eltech/linux/lab9 on git:master x [1
8:38:45] C:130
$ ./prog2 5 2
Program 2 get id = 29 shared segment
Program 2 connect shared segment
Program 2 waiting for shared variable
Program 2 free shared variable
Program 2 waiting for shared variable
Program 2 free shared variable
Program 2 waiting for shared variable
Program 2 free shared variable
Program 2 waiting for shared variable
Program 2 free shared variable
Program 2 waiting for shared variable
Program 2 free shared variable
Program 2 free shared variable

$ ./prog1 6 1
Program 1 get id = 29 shared segment
Program 1 connect shared segment
Program 1 waiting for shared variable
Program 1 free shared variable
Program 1 waiting for shared variable
Program 1 free shared variable
Program 1 waiting for shared variable
Program 1 free shared variable
Program 1 waiting for shared variable
Program 1 free shared variable
Program 1 waiting for shared variable
Program 1 free shared variable
Program 1 free shared variable
Program 1 disconnect shared segment

8:38:47]
$ ./prog3 7 1
Program 3 get id = 29 shared segment
Program 3 connect shared segment
Program 3 waiting for shared variable
Program 3 free shared variable
Program 3 waiting for shared variable
Program 3 free shared variable
Program 3 waiting for shared variable
Program 3 free shared variable
Program 3 waiting for shared variable
Program 3 free shared variable
Program 3 waiting for shared variable
Program 3 free shared variable
Program 3 waiting for shared variable
Program 3 free shared variable
Program 3 free shared variable
```

Рисунок 1

Данные выходного файла представлены на рис. 4.



```
# kivyfreakt @ hpomen in ~/documents/eltech/linux/lab9
$ cat output.txt
Program 1 PID: 36592, Time: Sun Nov 27 18:41:08 2022
Program 3 PID: 36602, Time: Sun Nov 27 18:41:08 2022
Program 1 PID: 36592, Time: Sun Nov 27 18:41:09 2022
Program 2 PID: 36601, Time: Sun Nov 27 18:41:09 2022
Program 3 PID: 36602, Time: Sun Nov 27 18:41:09 2022
Program 1 PID: 36592, Time: Sun Nov 27 18:41:10 2022
Program 3 PID: 36602, Time: Sun Nov 27 18:41:10 2022
Program 1 PID: 36592, Time: Sun Nov 27 18:41:11 2022
Program 2 PID: 36601, Time: Sun Nov 27 18:41:11 2022
Program 3 PID: 36602, Time: Sun Nov 27 18:41:11 2022
Program 1 PID: 36592, Time: Sun Nov 27 18:41:12 2022
Program 3 PID: 36602, Time: Sun Nov 27 18:41:12 2022
Program 1 PID: 36592, Time: Sun Nov 27 18:41:13 2022
Program 2 PID: 36601, Time: Sun Nov 27 18:41:13 2022
Program 3 PID: 36602, Time: Sun Nov 27 18:41:13 2022
Program 3 PID: 36602, Time: Sun Nov 27 18:41:14 2022
Program 2 PID: 36601, Time: Sun Nov 27 18:41:15 2022
```

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 билета
    shared_var->waiting_ticket[process] = 1;

    // стадия получения значения для билета
    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.