**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра вычислительной техники**

отчет

**по лабораторной работе №10**

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

Темы: СИНХРОНИЗАЦИЯ ПРОЦЕССОВ С ПОМОЩЬЮ СЕМАФОРОВ

|  |  |  |
| --- | --- | --- |
| Студент гр. 8306 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Пеунов В.В. |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Разумовский Г.В. |

Санкт-Петербург,

2021

**Цель работы**

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

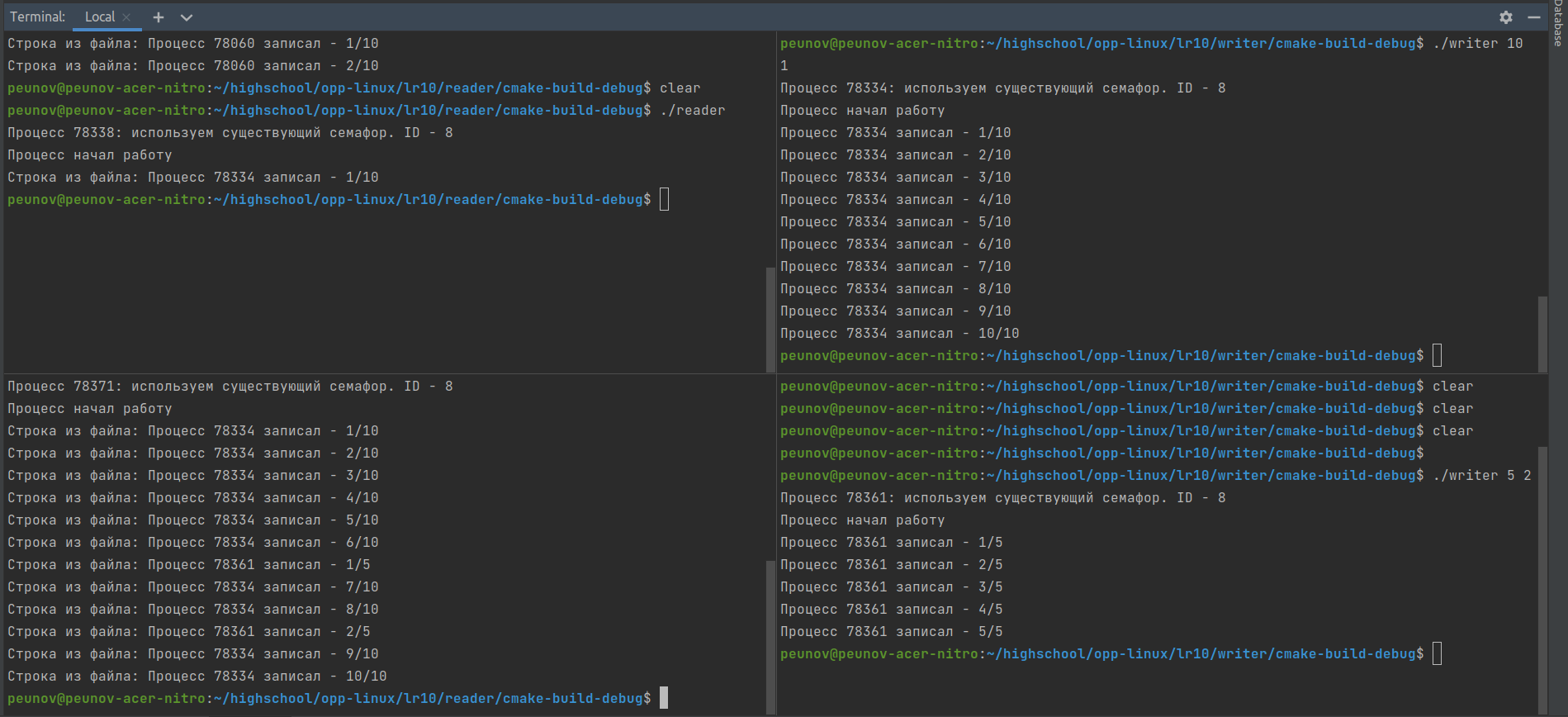
**Задание на лабораторную работу**

1. Написать две программы, экземпляры которых запускаются параллельно и с различной частотой обращаются к общему файлу. Каждый процесс из первой группы (Писатель) пополняет файл определенной строкой символов и выводит ее на экран вместе с именем программы. Процессы второй группы (Читатели) считывают весь файл и выводят его на экран. Писатели имеют приоритет перед Читателями. Пока один Писатель записывает строку в файл, другим Писателям и всем Читателям запрещено обращение к файлу. Читатели могут одновременно читать файл, если нет Писателей, готовых к записи в файл. Писатель заканчивает работу, после того как выполнит N-кратную запись строки в файл. Читатель заканчивает работу после прочтения текущего содержимого файла. Синхронизация процессов должна выполняться с помощью семафоров.

Откомпилировать программы Читатель и Писатель. Запустить на разных терминалах несколько Писателей и Читателей.

**Описание работы**

Скриншот работы программы представлен на рисунке 1.

Рисунок 1.

**Вывод:** в ходе лабораторной работы осуществлено знакомство с организацией семафоров, системными функциями, обеспечивающими управление семафорами, и их использованием для решения задач взаимоисключения и синхронизации

**Приложение A. Писатель**

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/time.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <signal.h>

#include <iostream>

#include <fstream>

using namespace std;

/\*

\* 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.

\*

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

\* \*/

#define COUNT\_PROCESSES 3

const char \*file\_path = "/home/peunov/highschool/opp-linux/lr9/output/result.txt";

typedef struct \_memory {

bool choosing[COUNT\_PROCESSES];

int number[COUNT\_PROCESSES];

int count;

} memory;

memory\* shared\_memory;

int program\_id;

void lock(memory\*, int);

void unlock(memory\*, int);

int get\_max(int\* array) {

int max = array[0];

for (size\_t j = 0; j < COUNT\_PROCESSES; ++j) {

if (array[j] > max) {

max = array[j];

}

}

return max;

}

bool more(int\* array, int i, int j){

return (array[j] < array[i] || (array[i] == array[j] && j < i));

}

void lock(memory\* m, int process) {

// Алгоритм пекарни Лампорта

m->choosing[process] = true;

m->number[process] = get\_max(m->number) + 1;

m->choosing[process] = false;

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

while (m->choosing[j]);

while (m->number[j] != 0 && more(m->number, process, j));

}

}

void unlock(memory\* m, int process) {

m->number[process] = 0;

}

void output\_in\_file(){

time\_t current\_time = time(nullptr);

ofstream output;

output.open(file\_path, ios::app);

output << "Программа №" << program\_id << ": PID - " << getpid() << ", время -" << ctime(&current\_time);

output.close();

}

void signal\_handler(int sig){

if (sig == SIGALRM) {

cout << "Программа №" << program\_id << " ожидает освобождения ресурса" << endl;

lock(shared\_memory, program\_id);

output\_in\_file();

unlock(shared\_memory, program\_id);

cout << "Программа №" << program\_id << " освободила ресурс" << endl;

}

}

void set\_timer(int period){

struct itimerval timer;

timerclear(&timer.it\_interval);

timerclear(&timer.it\_value);

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

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

setitimer(ITIMER\_REAL, &timer, nullptr);

}

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

if(argc != 5){

cout << "Ошибка! Недостаточно аргументов для запуска" << endl;

cout << "Правильный порядок аргументов: номер программы, ключ, количество запусков, период" << endl;

}

// получаем аргументы

program\_id = atoi(argv[1]);

int key = atoi(argv[2]);

int count\_starts = atoi(argv[3]);

int period = atoi(argv[4]);

//устанавливаем сигнал

signal(SIGALRM, signal\_handler);

int shared\_memory\_id = shmget(key, sizeof(memory), (0666 | IPC\_CREAT));

cout << "Программа №" << program\_id << " получила id: " << shared\_memory\_id << " разделяемой памяти" << endl;

void\* shared\_memory\_address = shmat(shared\_memory\_id,nullptr,0);

cout << "Программа №" << program\_id << " присоединила разделенную память" << endl;

shared\_memory = (memory\*)shared\_memory\_address;

//ожидаем открытия всех программ

shared\_memory->count += 1;

cout << "Ожидаем остальных......." << endl;

while (shared\_memory->count != 3);

cout << "........................дождались" << endl;

// установка таймера

set\_timer(period);

for(int i = 0; i < count\_starts; i++){

pause();

}

shmdt(shared\_memory\_address);

cout << "Программа №" << program\_id << " отсоединила разделяемую память" << endl;

exit(EXIT\_SUCCESS);

}#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <unistd.h>

#include <fstream>

using namespace std;

const char\* file\_path = "/home/peunov/highschool/opp-linux/lr10/files/output.txt";

#define SEM\_KEY 183

//0 - мьютекс файла

struct sembuf file\_mute = {0, -1, 0};

struct sembuf file\_unmute = {0, 1, 0};

//1 - семафор количества процессов

struct sembuf process\_increase = {1, 1, 0};

struct sembuf process\_decrease = {1, -1, 0};

//2 - семафор количества писателей желающих писать

struct sembuf writers\_wont\_write\_increase = {2, 1, 0};

struct sembuf writers\_wont\_write\_decrease = {2, -1, 0};

struct sembuf writers\_wont\_write\_is\_zero = {2, 0, 0};

//3 - семафор количества читателей

struct sembuf active\_readers\_increase = {3, 1, 0};

struct sembuf active\_readers\_decrease = {3, -1, 0};

struct sembuf active\_readers\_is\_zero = {3, 0, 0};

void create\_file(){

ofstream output;

output.open(file\_path, ios::out);

output.close();

}

void print\_in\_file(int pid, int i, int count){

ofstream output;

output.open(file\_path, ios::app);

output << "Процесс " << pid << " записал - " << i << "/" << count << endl;

cout << "Процесс " << pid << " записал - " << i << "/" << count << endl;

output.close();

}

void close\_semaphore(int semaphore\_id){

if(semctl(semaphore\_id, 1, GETVAL, 0 ) == 0){

semctl(semaphore\_id,IPC\_RMID,0);

printf("Семафор уничтожены\n");

}

}

int semaphore\_init(int pid){

int semaphore\_id = semget(SEM\_KEY,4,IPC\_CREAT|IPC\_EXCL|0666);

if(semaphore\_id > 0){

create\_file();

cout << "Процесс " << pid << ": создал семафор. ID - " << semaphore\_id << endl;

semop(semaphore\_id, &file\_unmute, 1); // размьючиваем файл

} else {

semaphore\_id = semget(SEM\_KEY,4,IPC\_CREAT);

cout << "Процесс " << pid << ": используем существующий семафор. ID - " << semaphore\_id << endl;

}

semop(semaphore\_id, &process\_increase, 1);

return semaphore\_id;

}

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

if(argc != 3){

printf("Недостаточное количество аргументов: ./name <количество операций>");

}

// парсим аргументы

int number\_of\_writing = atoi(argv[1]);

int delay = atoi(argv[2]);

// получаем номер процесса

int pid = getpid();

// инициализируем семафоры

int semaphore\_id = semaphore\_init(pid);

cout << "Процесс начал работу" << endl;

for(int i = 0; i < number\_of\_writing; i++){

semop(semaphore\_id, &writers\_wont\_write\_increase, 1); // повышает количество писателей желающих начать писать

semop(semaphore\_id, &active\_readers\_is\_zero, 1); // ждет всех читателей

semop(semaphore\_id, &file\_mute, 1); // мьютит файл

print\_in\_file(pid, i+1, number\_of\_writing); // пишет в файл

semop(semaphore\_id, &writers\_wont\_write\_decrease, 1); //понижает количество писателей желающих писать

semop(semaphore\_id, &file\_unmute, 1); //размьючивает файл

sleep(delay);

}

semop(semaphore\_id, &process\_decrease, 1);

close\_semaphore(semaphore\_id);

return 0;

}

**Приложение Б. Писатель**

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <unistd.h>

#include <fstream>

using namespace std;

const char\* file\_path = "/home/peunov/highschool/opp-linux/lr10/files/output.txt";

#define SEM\_KEY 183

//0 - мьютекс файла

struct sembuf file\_mute = {0, -1, 0};

struct sembuf file\_unmute = {0, 1, 0};

//1 - семафор количества процессов

struct sembuf process\_increase = {1, 1, 0};

struct sembuf process\_decrease = {1, -1, 0};

//2 - семафор количества писателей желающих писать

struct sembuf writers\_wont\_write\_increase = {2, 1, 0};

struct sembuf writers\_wont\_write\_decrease = {2, -1, 0};

struct sembuf writers\_wont\_write\_is\_zero = {2, 0, 0};

//3 - семафор количества читателей

struct sembuf active\_readers\_increase = {3, 1, 0};

struct sembuf active\_readers\_decrease = {3, -1, 0};

struct sembuf active\_readers\_is\_zero = {3, 0, 0};

void create\_file(){

ofstream output;

output.open(file\_path, ios::out);

output.close();

}

void read\_file(){

string buffer;

ifstream file(file\_path);

while(getline(file, buffer)){

cout << "Строка из файла: " << buffer << endl;

}

file.close();

}

int semaphore\_init(int pid){

int semaphore\_id = semget(SEM\_KEY,4,IPC\_CREAT|IPC\_EXCL|0666);

if(semaphore\_id > 0){

create\_file();

cout << "Процесс " << pid << ": создал семафор. ID - " << semaphore\_id << endl;

semop(semaphore\_id, &file\_unmute, 1); // размьючиваем файл

} else {

semaphore\_id = semget(SEM\_KEY,4,IPC\_CREAT);

cout << "Процесс " << pid << ": используем существующий семафор. ID - " << semaphore\_id << endl;

}

semop(semaphore\_id, &process\_increase, 1);

return semaphore\_id;

}

void close\_semaphore(int semaphore\_id){

if(semctl(semaphore\_id, 1, GETVAL, 0 ) == 0){

semctl(semaphore\_id,IPC\_RMID,0);

printf("Семафор уничтожены\n");

}

}

int main(){

// получаем номер процесса

int pid = getpid();

// инициализируем семафоры

int semaphore\_id = semaphore\_init(pid);

cout << "Процесс начал работу" << endl;

semop(semaphore\_id, &writers\_wont\_write\_is\_zero, 1);

semop(semaphore\_id, &active\_readers\_increase, 1);

read\_file();

semop(semaphore\_id, &active\_readers\_decrease, 1);

semop(semaphore\_id, &process\_decrease,1);

close\_semaphore(semaphore\_id);

return 0;

}