«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И.Ульянова (Ленина)»
СПбГЭТУ «ЛЭТИ»

Кафедра вычислительной техники

Отчет по по лабораторной работе №9 по дисциплине «Организация процессов и программирование в среде Linux»
Тема: «ОБМЕН ДАННЫМИ ЧЕРЕЗ РАЗДЕЛЯЕМУЮ ПАМЯТЬ»

Студент гр.8306 _____ Слепов А.Э.

Преподаватель _____ Разумовский Г.В.

Санкт-Петербург

2021

Цель работы

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

Задание

1. 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.

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

Порядок выполнения работы

В работе запускаются 3 программы, которые используют разделяемый сегмент и алгоритм Лампорта для синхронизации работы. Компилируется одна программа, но при запуске аргументом требуется передать ее номер. Программы запрашивают разделяемый сегмент с ключом 123, затем присоединяют его к своему адресному пространству. Затем с помощью алгоритма Лампорта синхронизируют доступ к файлу. Когда программа выждала свой период, она пытается захватить разделяемый ресурс. Если ресурс свободен, то программа захватывает его и работает с общим файлом. Если ресурс занят другим процессом, программа находится в ожидании освобождения этого ресурса. Когда программа закончит писать в файл, она освобождает разделяемый ресурс.

Результаты запуска программ и выходной файл приведен на рисунке 1.

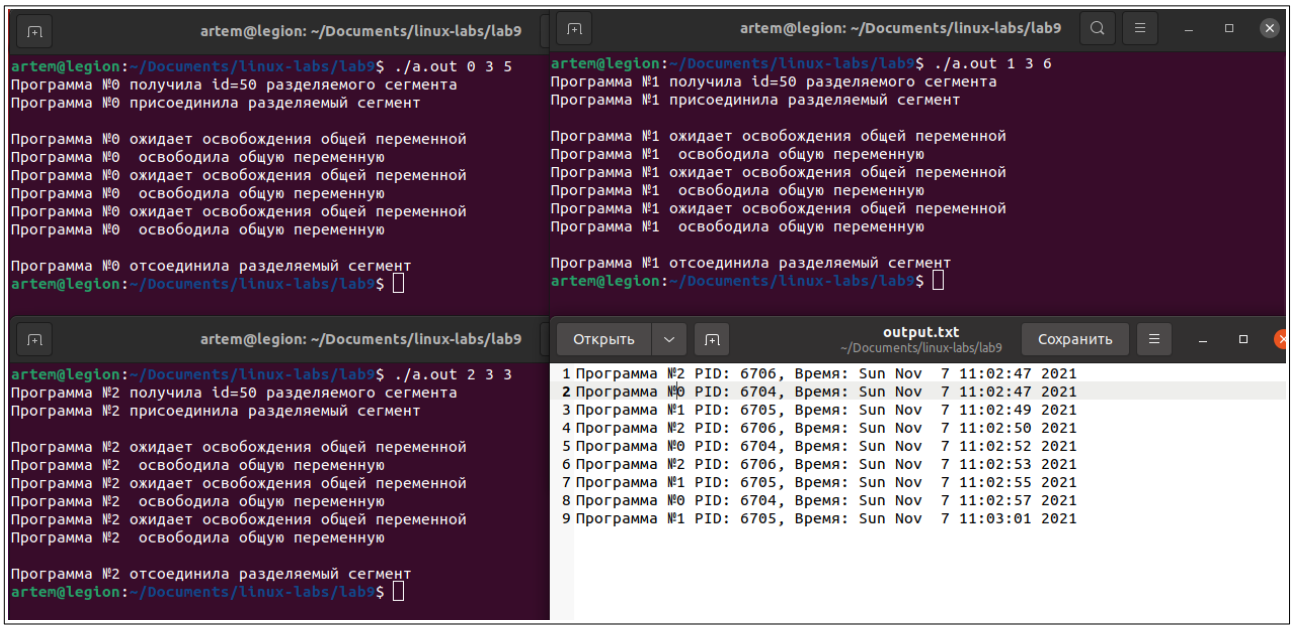

Рисунок 1. Результаты запуска программы

Выводы

В ходе работы были изучены механизмы организации разделяемой памяти и системные функции, обеспечивающие обмен данными между процессами в операционной системе Ubuntu. Также было закреплено на практике применение алгоритма Лампорта для синхронизации процессов.

ПРИЛОЖЕНИЕ А
Текст программы

```
##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>

#define NUM_PROCESS 3
#define MAX(a,b) (a>b)?a:b

typedef struct shared_data {
    int choosing[NUM_PROCESS];
    int number[NUM_PROCESS];
} shared;

void lock(shared*, int);
void unlock(shared*, int);

// Алгоритм Лампорта (булочной)
void lock(shared* shared_var, int process) {
    shared_var->choosing[process] = 1;
    shared_var->number[process] = 1 + MAX(shared_var->number[0], MAX(shared_var-
>number[1], shared_var->number[2]));
    shared_var->choosing[process] = 0;
    for (int i = 0; i < NUM_PROCESS; ++i) {
        if (i != process) {
            while (shared_var->choosing[i]);
            while (shared_var->number[i] != 0 && (shared_var->number[process] >
shared_var->number[i] ||
                (shared_var->number[process] == shared_var->number[i] && process
> i)));
        }
    }

}

void unlock(shared* shared_var, int process) {
    shared_var->number[process] = 0;
}

shared* sharedVar;
int progNum;

void signal_handler(int sig){
      if (sig == SIGALRM) {
            printf("Программа №%d ожидает освобождения общей переменной\n",
progNum);
        lock(sharedVar, progNum);

        FILE *file = fopen("output.txt", "a");
        time_t curTime = time(NULL);
        fprintf(file, "Программа №%d PID: %d, Время: %s",progNum, getpid(),
ctime(&curTime));
        fclose(file);

        unlock(sharedVar, progNum);
        printf("Программа №%d  освободила общую переменную\n", progNum);
```

```
    }
}

void main(int argc, char** argv){
      if(argc == 4){
            progNum = atoi(argv[1]);
            int numStarts = atoi(argv[2]);
            int period = atoi(argv[3]);
            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);


          int shmId = shmget(123, sizeof(shared), (0666 | IPC_CREAT));
          if(shmId != -1)
            printf("Программа №%d получила id=%d разделяемого сегмента\n",
progNum, shmId);
          else
            exit(EXIT_FAILURE);
          void* shmAddr = shmat(shmId,0,0);
          if(*(int*)shmAddr != -1)
            printf("Программа №%d присоединила разделяемый сегмент\n\n",
progNum);
          else
            exit(EXIT_FAILURE);

          sharedVar = (shared*)shmAddr;

          for(int i = 0; i < numStarts; i++){
            pause();
          }

          if(shmdt(shmAddr) != -1)
            printf("\nПрограмма №%d отсоединила разделяемый сегмент\n",
progNum);

      }
      else
            printf("Недостаточно аргументов для запуска: ./prog НОМЕР_ПРОГРАММЫ
КОЛ-ВО_ЗАПУСКОВ ПЕРИОД\n");
      exit(EXIT_SUCCESS);
}
```