Московский авиационный институт
(Национальный исследовательский университет)
Факультет "Информационные технологии и прикладная математика"
Кафедра "Вычислительная математика и программирование"

Лабораторная работа №3 по курсу “Операционные системы”

Студент: Слетюрин Кирилл Сергеевич

Группа: М8О-208Б-22

Преподаватель: Миронов Евгений Сергеевич

Вариант: 5

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2023

Содержание

1 Репозиторий

https://github.com/kirill483/OS

2 Цель работы

Приобретение практических навыков в:

- Освоении принципов работы с файловыми системами
- Обеспечении обмена данных между процессами посредством технологии «File mapping»

3 Задание

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

4 Описание работы программы

Задание аналогично первой лабораторной работе.

В ходе выполнения лабораторной работы я использовала следующие системные вызовы:

- fork() - создание нового процесса
- sem_open() - создание/открытие семафора
- sem_post() - увеличивание значения семафора и разблокировка ожидающих потоков
- sem_wait() - уменьшение значения семафора. Если 0, то вызывающий поток блокируется
- sem_close() - закрытие семафора
- shm_open() - создание/открытие разделяемой памяти POSIX
- shm_unlink() - закрытие разделяемой памяти
- ftruncate() - уменьшение длины файла до указанной
- mmap() - отражение файла или устройства в памяти
- munmap() - снятие отражения
- execlp() - запуск файла на исполнение

5 Исходный код

lab3.cpp

```
#include "lab3.hpp"

int ParentRoutine(const char *pathToChild) {
    std::string fileName1;
    std::string fileName2;

    getline(std::cin, fileName1);
    getline(std::cin, fileName2);

    sem_t* semptr1 = OpenSemaphore(EVEN_SEMAPHORE_NAME, 0);
    sem_t* semptr2 = OpenSemaphore(ODD_SEMAPHORE_NAME, 0);

    int shared_memory_fd1 = OpenSharedMemory(SHARED_MEMORY_NAME_1,
    MAP_SIZE);
    char *memptr1 = MapSharedMemory(MAP_SIZE, shared_memory_fd1);

    int shared_memory_fd2 = OpenSharedMemory(SHARED_MEMORY_NAME_2,
    MAP_SIZE);
    char *memptr2 = MapSharedMemory(MAP_SIZE, shared_memory_fd2);

    pid_t pid[2] {-1, -1};
    pid[0] = createChildProcess();
    pid[1] = createChildProcess();

    if (pid[0] == 0) {//child 1
        if (execl(pathToChild, "child", EVEN_SEMAPHORE_NAME,
    SHARED_MEMORY_NAME_1, fileName1.c_str(), nullptr) == -1) {
            perror("Error with execl");
            exit(EXIT_FAILURE);
        }

    } else
    if (pid[1] == 0) {//child 2
        if (execl(pathToChild, "child", ODD_SEMAPHORE_NAME,
    SHARED_MEMORY_NAME_2, fileName2.c_str(), nullptr) == -1) {
            perror("Error with execl");
            exit(EXIT_FAILURE);
        }

    } else {//parent
        std::string str;
        while (getline(std::cin, str)) {
            std::chrono::milliseconds delay(10);
            std::this_thread::sleep_for(delay);

            if (str.size() % 2 == 0) {
                strcpy(memptr1, str.c_str());
                sem_post(semptr1);
            } else {
                strcpy(memptr2, str.c_str());
                sem_post(semptr2);
            }
            str.clear();
        }
        std::chrono::milliseconds delay(10);
        std::this_thread::sleep_for(delay);
```

```
53
54          strcpy(memptr1, "\0");
55          sem_post(semptr1);
56          strcpy(memptr2, "\0");
57          sem_post(semptr2);
58
59          int status;
60          waitpid(pid[0], &status, 0);
61          waitpid(pid[1], &status, 0);
62
63          sem_close(semptr1);
64          sem_unlink(EVEN_SEMAPHORE_NAME);
65          shm_unlink(SHARED_MEMORY_NAME_1);
66          munmap(memptr1, MAP_SIZE);
67          close(shared_memory_fd1);
68
69          sem_close(semptr2);
70          sem_unlink(ODD_SEMAPHORE_NAME);
71          shm_unlink(SHARED_MEMORY_NAME_2);
72          munmap(memptr2, MAP_SIZE);
73          close(shared_memory_fd2);
74      }
75      return 0;
76  }
```

utils.c

```
1  #include "utils.hpp"
2  #include <sys/mman.h>
3  #include <sys/stat.h>
4  #include <fcntl.h>
5
6  pid_t createChildProcess() {
7      pid_t pid = fork();
8      if (pid == -1) {
9          perror("Couldn't create child process");
10         exit(EXIT_FAILURE);
11     }
12     return pid;
13 }
14
15 sem_t* OpenSemaphore(const char *name, int value) {
16     sem_t *semptr = sem_open(name, O_CREAT, S_IRUSR | S_IWUSR,
       value);
17     if (semptr == SEM_FAILED){
18         perror("Couldn't open the semaphore");
19         exit(EXIT_FAILURE);
20     }
21     return semptr;
22 }
23
24 int OpenSharedMemory(const char *name, const int size) {
25     int sh_fd = shm_open(name, O_CREAT | O_RDWR, S_IRUSR | S_IWUSR
       );
26     if (sh_fd == -1) {
27         perror("Couldn't create memory shared object");
28         exit(EXIT_FAILURE);
29     }
30     if (ftruncate(sh_fd, size) == -1) {
31         perror("Couldn't truncate a file");
```

```
        exit(EXIT_FAILURE);
    }
    return sh_fd;
}

char* MapSharedMemory(const int size, int fd) {
    char *memptr = (char*)mmap(nullptr, size, PROT_READ |
    PROT_WRITE, MAP_SHARED, fd, 0);
    if (memptr == MAP_FAILED) {
        perror("Error with file mapping");
        exit(EXIT_FAILURE);
    }
    return memptr;
}
```

child.cpp

```
#include "utils.hpp"

int main(int argc, char* argv[]) {
    (void)argc;

    sem_t *semptr = OpenSemaphore(argv[1], 0);
    int shared_memory_fd = OpenSharedMemory(argv[2], MAP_SIZE);
    char *memptr = MapSharedMemory(MAP_SIZE, shared_memory_fd);
    std::ofstream fout(argv[3],  std::ios::trunc);
    fout.close();

    while (true) {
        sem_wait(semptr);
        std::string_view st(memptr);
        std::string str = {st.begin(), st.end()};
        if (str == "\0") {
            break;
        }
        std::reverse(str.begin(), str.end());
        std::ofstream fout(argv[3],  std::ios::app);
        if (!fout.is_open()) {
            perror("Couldn't open the file");
            exit(EXIT_FAILURE);
        }
        fout << str << std::endl;
        fout.close();
    }
    sem_close(semptr);
    sem_unlink(argv[1]);
    shm_unlink(argv[2]);
    munmap(memptr, MAP_SIZE);
    close(shared_memory_fd);
    exit(EXIT_SUCCESS);
}
```

main.c

```
#include "lab3.hpp"

int main() {
    const char path[] = "/mnt/c/Users/Kirill/OS/lab3/build/child";
    ParentRoutine(path);
    exit(EXIT_SUCCESS);
}
```

6 Тесты

```
#include <gtest/gtest.h>

#include <filesystem>
#include <memory>
#include <vector>

#include <lab3.hpp>
#include <utils.hpp>

namespace fs = std::filesystem;

void testingProgram(const std::vector<std::string> &input, const std::vector<std::string> &expectedOutput1, const std::vector<std::string> &expectedOutput2) {
    const char *fileWithInput = "input.txt";
    const char *fileWithOutput1 = "output1.txt";
    const char *fileWithOutput2 = "output2.txt";

    std::stringstream inFile(fileWithInput);
    inFile << fileWithOutput1 << std::endl;
    inFile << fileWithOutput2 << std::endl;
    for (std::string line : input) {
        inFile << line << std::endl;
    }
    //inFile.close();

    //std::ifstream inFile1(fileWithInput);
/*
    if (!inFile1.is_open()) {
        perror("Couldn't open the file");
        exit(EXIT_FAILURE);
    }
*/
    std::streambuf* oldInBuf = std::cin.rdbuf(inFile.rdbuf());

    ASSERT_TRUE(fs::exists("/mnt/c/Users/Kirill/OS/lab3/build/child"));

    ParentRoutine("/mnt/c/Users/Kirill/OS/lab3/build/child");
    std::cin.rdbuf(oldInBuf);

    auto outFile1 = std::ifstream(fileWithOutput1);
    auto outFile2 = std::ifstream(fileWithOutput2);
    if (!outFile1.is_open()) {
        perror("Couldn't open the file");
        exit(EXIT_FAILURE);
    }
    for (const std::string &line : expectedOutput1) {
        std::string result;
        getline(outFile1, result);
        EXPECT_EQ(result, line);
    }
    outFile1.close();
    std::remove(fileWithOutput1);

    if (!outFile2.is_open()) {
        perror("Couldn't open the file");
```

```
55          exit(EXIT_FAILURE);
56      }
57      for (const std::string &line : expectedOutput2) {
58          std::string result;
59          getline(outFile2, result);
60          EXPECT_EQ(result, line);
61      }
62      outFile2.close();
63      std::remove(fileWithOutput2);
64  }
65
66  TEST(thirdLabTests, emptyTest) {
67      std::vector<std::string> input = {
68          ""
69      };
70
71      std::vector<std::string> expectedOutput1 = {};
72
73      std::vector<std::string> expectedOutput2 = {};
74
75      testingProgram(input, expectedOutput1, expectedOutput2);
76  }
77
78  TEST(thirdLabTests, firstSimpleTest) {
79      std::vector<std::string> input = {
80          "01",
81          "02",
82          "001",
83          "002",
84          ""
85      };
86
87      std::vector<std::string> expectedOutput1 = {
88          "10",
89          "20"
90      };
91
92      std::vector<std::string> expectedOutput2 = {
93          "100",
94          "200"
95      };
96
97      testingProgram(input, expectedOutput1, expectedOutput2);
98  }
99
100 TEST(thirdLabTests, secondSimpleTest) {
101     std::vector<std::string> input = {
102             "This test has only",
103             "one output file.",
104             ""
105     };
106
107     std::vector<std::string> expectedOutput1 = {
108             "ylno sah tset sihT",
109             ".elif tuptuo eno"
110     };
111
112     std::vector<std::string> expectedOutput2 = {};
113
```

```
114
115     testingProgram(input, expectedOutput1, expectedOutput2);
116 }
117
118 TEST(thirdLabTests, thirdSimpleTest) {
119     std::vector<std::string> input = {
120             "42",
121             "5",
122             "1000000000",
123             "60",
124             ""
125     };
126
127     std::vector<std::string> expectedOutput1 = {
128             "24",
129             "06"
130     };
131
132     std::vector<std::string> expectedOutput2 = {
133             "5",
134             "0000000001"
135     };
136
137
138     testingProgram(input, expectedOutput1, expectedOutput2);
139 }
140
141 int main(int argc, char *argv[]) {
142     testing::InitGoogleTest(&argc, argv);
143     return RUN_ALL_TESTS();
144 }
```

7 Демонстрация работы программы

```
kirill@DESKTOP-7E05ERB:/mnt/c/Users/Kirill/OS/lab3/build$ ./main
file.txt
22
6
kirill@DESKTOP-7E05ERB:/mnt/c/Users/Kirill/OS/lab3/build$ cat file.txt
22
6
```

8 Запуск тестов

```
kirill@DESKTOP-7E05ERB:/mnt/c/Users/Kirill/OS/lab3/build$ ./Test
[==========] Running 4 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 4 tests from thirdLabTests
[ RUN      ] thirdLabTests.emptyTest
[       OK ] thirdLabTests.emptyTest (25 ms)
[ RUN      ] thirdLabTests.firstSimpleTest
[       OK ] thirdLabTests.firstSimpleTest (67 ms)
[ RUN      ] thirdLabTests.secondSimpleTest
[       OK ] thirdLabTests.secondSimpleTest (46 ms)
[ RUN      ] thirdLabTests.thirdSimpleTest
[       OK ] thirdLabTests.thirdSimpleTest (65 ms)
[----------] 4 tests from thirdLabTests (203 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 1 test suite ran. (203 ms total)
[  PASSED  ] 4 tests.
```

9 Выводы

В результате выполнения данной лабораторной работы была написана программа на языке C++, осуществляющая работу с процессами и взаимодействие между ними через системные сигналы и отображаемые файлы. Приобретены практические навыки в освоении принципов работы с файловыми системами и обеспечении обмена данных между процессами посредством технологии «File mapping».