

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: Былькова Кристина Алексеевна

Группа: М8О-208Б-22

Преподаватель: Миронов Евгений Сергеевич

Вариант: 16

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2023

Содержание

1	Репозиторий	3
2	Цель работы	3
3	Задание	3
4	Описание работы программы	3
5	Исходный код	4
6	Тесты	8
7	Консоль	11
8	Запуск тестов	11
9	Выводы	12

1 Репозиторий

https://github.com/kr1st1na0/OS_labs

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()` - увеличение 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

utils.hpp

```
1 #pragma once
2
3 #include <iostream>
4 #include <sstream>
5 #include <string>
6 #include <vector>
7 #include <cstring>
8 #include <stdlib.h>
9 #include <unistd.h>
10 #include <sys/types.h>
11 #include <sys/wait.h>
12 #include <sys/mman.h>
13 #include <ext/stdio_filebuf.h>
14
15 #include <fcntl.h>
16 #include <sys/stat.h>
17 #include <semaphore.h>
18
19 const int MAP_SIZE = 1024;
20
21 constexpr const char *SEMAPHORE_NAME_1 = "/semaphore_1";
22 constexpr const char *SHARED_MEMORY_NAME_1 = "/shared_memory_1";
23
24 constexpr const char *SEMAPHORE_NAME_2 = "/semaphore_2";
25 constexpr const char *SHARED_MEMORY_NAME_2 = "/shared_memory_2";
26
27 sem_t* OpenSemaphore(const char *name, int value);
28 int OpenSharedMemory(const char *name, const int size);
29 char* MapSharedMemory(const int size, int fd);
30 pid_t CreateChildProcess();
31 bool CheckString(const std::string_view str);
```

parent.hpp

```
1 #pragma once
2
3 #include "utils.hpp"
4
5 void ParentProcess(const char *pathToChild);
```

utils.cpp

```
1 #include "utils.hpp"
2
3 sem_t* OpenSemaphore(const char *name, int value) {
4     sem_t *semptr = sem_open(name, O_CREAT, S_IRUSR | S_IWUSR,
5     value);
6     if (semptr == SEM_FAILED){
7         perror("Couldn't open the semaphore");
8         exit(EXIT_FAILURE);
9     }
10    return semptr;
11 }
12
13 int OpenSharedMemory(const char *name, const int size) {
14     int sh_fd = shm_open(name, O_CREAT | O_RDWR, S_IRUSR | S_IWUSR
15 );
16 }
```

```

14     if (sh_fd == -1) {
15         perror("Couldn't create memory shared object");
16         exit(EXIT_FAILURE);
17     }
18     if (ftruncate(sh_fd, size) == -1) {
19         perror("Couldn't truncate a file");
20         exit(EXIT_FAILURE);
21     }
22     return sh_fd;
23 }
24
25 char* MapSharedMemory(const int size, int fd) {
26     char *memptr = (char*)mmap(nullptr, size, PROT_READ |
27     PROT_WRITE, MAP_SHARED, fd, 0);
28     if (memptr == MAP_FAILED) {
29         perror("Error with file mapping");
30         exit(EXIT_FAILURE);
31     }
32     return memptr;
33 }
34 pid_t CreateChildProcess() {
35     pid_t pid = fork();
36     if (pid == -1) {
37         perror("Couldn't create child process");
38         exit(EXIT_FAILURE);
39     }
40     return pid;
41 }
42
43 bool CheckString(const std::string_view str) {
44     if (str[str.size() - 1] == '.' || str[str.size() - 1] == ';')
45     {
46         return true;
47     }
48     return false;
49 }

```

child.cpp

```

1 #include "utils.hpp"
2
3 int main(int argc, char *argv[]) {
4     if (argc != 2) {
5         perror("Not enough arguments");
6         exit(EXIT_FAILURE);
7     }
8
9     const char *fileName = argv[1];
10    std::ofstream fout(fileName, std::ios::app);
11    if (!fout.is_open()) {
12        perror("Couldn't open the file");
13        exit(EXIT_FAILURE);
14    }
15
16    sem_t *semptr1 = OpenSemaphore(SEMAPHORE_NAME_1, 0);
17    int shared_memory_fd1 = OpenSharedMemory(SHARED_MEMORY_NAME_1,
18    MAP_SIZE);
19    char *memptr1 = MapSharedMemory(MAP_SIZE, shared_memory_fd1);

```

```

20 sem_t *semptr2 = OpenSemaphore(SEMAPHORE_NAME_2, 0);
21 int shared_memory_fd2 = OpenSharedMemory(SHARED_MEMORY_NAME_2,
MAP_SIZE);
22 char* memptr2 = MapSharedMemory(MAP_SIZE, shared_memory_fd2);
23
24 while (1) {
25     sem_wait(semptr1);
26     std::string_view str(memptr1);
27     if (str.empty()) {
28         sem_post(semptr2);
29         break;
30     }
31     if (CheckString(str)) {
32         fout << str << std::endl;
33     } else {
34         strcpy(memptr2, "ERROR_STRING");
35     }
36     sem_post(semptr2);
37 }
38
39 sem_close(semptr1);
40 sem_unlink(SEMAPHORE_NAME_1);
41 shm_unlink(SHARED_MEMORY_NAME_1);
42 munmap(memptr1, MAP_SIZE);
43 close(shared_memory_fd1);
44
45 sem_close(semptr2);
46 sem_unlink(SEMAPHORE_NAME_2);
47 shm_unlink(SHARED_MEMORY_NAME_2);
48 munmap(memptr2, MAP_SIZE);
49 close(shared_memory_fd2);
50
51 exit(EXIT_SUCCESS);
52 }

```

parent.cpp

```

1 #include "parent.hpp"
2 #include "utils.hpp"
3
4 void ParentProcess(const char *pathToChild) {
5
6     std::string fileName;
7     getline(std::cin, fileName);
8
9     sem_t *semptr1 = OpenSemaphore(SEMAPHORE_NAME_1, 0);
10    int shared_memory_fd1 = OpenSharedMemory(SHARED_MEMORY_NAME_1,
MAP_SIZE);
11    char *memptr1 = MapSharedMemory(MAP_SIZE, shared_memory_fd1);
12
13    sem_t *semptr2 = OpenSemaphore(SEMAPHORE_NAME_2, 0);
14    int shared_memory_fd2 = OpenSharedMemory(SHARED_MEMORY_NAME_2,
MAP_SIZE);
15    char* memptr2 = MapSharedMemory(MAP_SIZE, shared_memory_fd2);
16
17    std::string str;
18    std::vector<std::string> errorStrings;
19
20    int pid = CreateChildProcess();
21    if (pid != 0) { // Parent process

```

```

22     while(getline(std::cin, str)) {
23         strcpy(memptr1, str.c_str());
24         sem_post(sem_ptr1);
25
26         sem_wait(sem_ptr2);
27         if (strcmp(memptr2, "ERROR_STRING") == 0) {
28             errorStrings.push_back(str);
29             strcpy(memptr2, "");
30         }
31     }
32     strcpy(memptr1, "");
33     sem_post(sem_ptr1);
34 } else { // Child process
35     if (execlp(pathToChild, pathToChild, fileName.c_str(),
36 nullptr) == -1) { // to child.cpp
37         perror("Error with execlp");
38         exit(EXIT_FAILURE);
39     }
40
41     for (const std::string &err : errorStrings) {
42         std::cout << "ERROR with string: " << err << std::endl;
43     }
44
45     sem_close(sem_ptr1);
46     sem_unlink(SEMAPHORE_NAME_1);
47     shm_unlink(SHARED_MEMORY_NAME_1);
48     munmap(memptr1, MAP_SIZE);
49     close(shared_memory_fd1);
50
51     sem_close(sem_ptr2);
52     sem_unlink(SEMAPHORE_NAME_2);
53     shm_unlink(SHARED_MEMORY_NAME_2);
54     munmap(memptr2, MAP_SIZE);
55     close(shared_memory_fd2);
56 }

```

main.cpp

```

1 #include "parent.hpp"
2
3 int main() {
4     ParentProcess(getenv("PATH_TO_CHILD"));
5     // bash: export PATH_TO_CHILD="/home/kristinab/ubuntu_main/
6     OS_labs/build/lab3/child3"
7     // bash: printenv PATH_TO_CHILD
8     exit(EXIT_SUCCESS);
9 }

```

6 Тесты

```
1 #include <gtest/gtest.h>
2
3 #include <filesystem>
4 #include <memory>
5 #include <vector>
6
7 #include "parent.hpp"
8
9 namespace fs = std::filesystem;
10
11 void testingProgram(const std::vector<std::string> &input, const
    std::vector<std::string> &expectedOutput, const std::vector<std
    ::string> &expectedFile) {
12     const char *fileName = "file.txt";
13
14     std::stringstream inFile;
15     inFile << fileName << std::endl;
16     for (std::string line : input) {
17         inFile << line << std::endl;
18     }
19
20     std::streambuf* oldInBuf = std::cin.rdbuf(inFile.rdbuf());
21
22     ASSERT_TRUE(fs::exists(getenv("PATH_TO_CHILD")));
23
24     testing::internal::CaptureStdout();
25
26     ParentProcess(getenv("PATH_TO_CHILD"));
27     std::cin.rdbuf(oldInBuf);
28
29     std::stringstream errorOut(testing::internal::
    GetCapturedStdout());
30     for(const std::string &expectation : expectedOutput) {
31         std::string result;
32         getline(errorOut, result);
33         EXPECT_EQ(result, expectation);
34     }
35
36     std::ifstream fin(fileName);
37     if (!fin.is_open()) {
38         perror("Couldn't open the file");
39         exit(EXIT_FAILURE);
40     }
41     for (const std::string &expectation : expectedFile) {
42         std::string result;
43         getline(fin, result);
44         EXPECT_EQ(result, expectation);
45     }
46     fin.close();
47     std::remove(fileName);
48 }
49
50 TEST(thirdLabTests, emptyTest) {
51     std::vector<std::string> input = {};
52
53     std::vector<std::string> expectedOutput = {};
54 }
```



```

55     std::vector<std::string> expectedFile = {};
56
57     testingProgram(input, expectedOutput, expectedFile);
58 }
59
60 TEST(thirdLabTests, simpleTest) {
61     std::vector<std::string> input = {
62         "No,",
63         "you'll never be alone.",
64         "When darkness comes;",
65         "I'll light the night with stars",
66         "Hear my whispers in the dark!"
67     };
68
69     std::vector<std::string> expectedOutput = {
70         "ERROR with string: No,",
71         "ERROR with string: I'll light the night with stars",
72         "ERROR with string: Hear my whispers in the dark!"
73     };
74
75     std::vector<std::string> expectedFile = {
76         "you'll never be alone.",
77         "When darkness comes;"
78     };
79
80     testingProgram(input, expectedOutput, expectedFile);
81 }
82
83 TEST(thirdLabTests, aQuedaTest) {
84     std::vector<std::string> input = {
85         "A QUEDA:",
86         "E venha ver os deslizes que eu vou cometer;",
87         "E venha ver os amigos que eu vou perder;",
88         "N o t cobrando entrada, vem ver o show na faixa.",
89         "Hoje tem open bar pra ver minha desgra a."
90     };
91
92     std::vector<std::string> expectedOutput = {
93         "ERROR with string: A QUEDA:"
94     };
95
96     std::vector<std::string> expectedFile = {
97         "E venha ver os deslizes que eu vou cometer;",
98         "E venha ver os amigos que eu vou perder;",
99         "N o t cobrando entrada, vem ver o show na faixa.",
100        "Hoje tem open bar pra ver minha desgra a."
101    };
102
103    testingProgram(input, expectedOutput, expectedFile);
104 }
105
106 TEST(thirdLabTests, anotherTest) {
107     std::vector<std::string> input = {
108         "But I set fire to the rain.",
109         "Watched it pour as- I touched your- face-",
110         "Well, it burned while I cried!!!!!!!!!!",
111         "Cause I heard it screamin' out your name;",
112         "Your name."
113     };

```

```

114
115     std::vector<std::string> expectedOutput = {
116         "ERROR with string: Watched it pour as- I touched your-
face-",
117         "ERROR with string: Well, it burned while I cried!!!!!!!!!!"
118     };
119
120     std::vector<std::string> expectedFile = {
121         "But I set fire to the rain.",
122         "Cause I heard it screamin' out your name;",
123         "Your name."
124     };
125
126     testingProgram(input, expectedOutput, expectedFile);
127 }
128
129 int main(int argc, char *argv[]) {
130     std::cout << getenv("PATH_TO_CHILD") << std::endl;
131
132     testing::InitGoogleTest(&argc, argv);
133     return RUN_ALL_TESTS();
134 }

```

7 Консоль

```
kristinab@LAPTOP-SFU9B1F4:~/ubuntu_main/OS_labs/build/lab1$ ./lab3 file.txt
No,
you'll never be alone.
When darkness comes;
I'll light the night with stars
Hear my whispers in the dark!
ERROR with string: I'll light the night with stars
ERROR with string: No,
ERROR with string: Hear my whispers in the dark!
kristinab@LAPTOP-SFU9B1F4:~/ubuntu_main/OS_labs/build/lab1$ ./lab3 file.txt
A QUEDA:
E venha ver os deslizes que eu vou cometer;
E venha ver os amigos que eu vou perder;
Não tô cobrando entrada, vem ver o show na faixa.
Hoje tem open bar pra ver minha desgraça.
ERROR with string: A QUEDA:
kristinab@LAPTOP-SFU9B1F4:~/ubuntu_main/OS_labs/build/lab1$ ./lab3 file.txt
But I set fire to the rain.
Watched it pour as- I touched your- face-
Well, it burned while I cried!!!!!!!
Cause I heard it screamin' out your name;
Your name.
ERROR with string: Watched it pour as- I touched your- face-
ERROR with string: Well, it burned while I cried!!!!!!!
```

8 Запуск тестов

```
kristinab@LAPTOP-SFU9B1F4:~/ubuntu_main/OS_labs/build/tests$ ./lab3_test
/home/kristinab/ubuntu_main/OS_labs/build/lab3/child3
[=====] Running 4 tests from 1 test suite.
[-----] Global test environment set-up.
[-----] 4 tests from thirdLabTests
[ RUN      ] thirdLabTests.emptyTest
[      OK  ] thirdLabTests.emptyTest (0 ms)
[ RUN      ] thirdLabTests.simpleTest
[      OK  ] thirdLabTests.simpleTest (7 ms)
[ RUN      ] thirdLabTests.aQuedaTest
[      OK  ] thirdLabTests.aQuedaTest (1 ms)
[ RUN      ] thirdLabTests.anotherTest
[      OK  ] thirdLabTests.anotherTest (1 ms)
[-----] 4 tests from thirdLabTests (10 ms total)

[-----] Global test environment tear-down
[=====] 4 tests from 1 test suite ran. (10 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. Я приобрела практические навыки в освоении принципов работы с файловыми системами и обеспечении обмена данных между процессами посредством технологии «File mapping».