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: Кочкожаров Иван Вячеславович

Группа: М8О-208Б-22

Преподаватель: Миронов Евгений Сергеевич

Вариант: 20

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2023

Содержание

1 Репозиторий

https://github.com/kochkozharov/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() - 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.c

```c
#include "lab3.h"

#include <errno.h>
#include <fcntl.h>
#include <semaphore.h>
#include <signal.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>

#include "error_handling.h"
#include "utils.h"

int ParentRoutine(const char* pathToChild, FILE* stream) {
    int fds[2] = {-1, -1};
    char* line = NULL;
    size_t len = 0;
    char* extraLine = NULL;
    size_t extraLen = 0;

    SharedMemory shm1;
    SharedMemory shm2;
    GOTO_IF(CreateSharedMemory(&shm1, SHARED_MEMORY_NAME_1,
    MAP_SIZE),
            "CreateSharedMemory", err);
    GOTO_IF(CreateSharedMemory(&shm2, SHARED_MEMORY_NAME_2,
    MAP_SIZE),
            "CreateSharedMemory", err);

    SemaphorePair pair1;
    SemaphorePair pair2;

    GOTO_IF(CreateSemaphorePair(&pair1, W_SEMAPHORE_NAME_1,
                                REV_SEMAPHORE_NAME_1) == -1,
            "CreateSemaphorePair", err);
    GOTO_IF(CreateSemaphorePair(&pair2, W_SEMAPHORE_NAME_2,
                                REV_SEMAPHORE_NAME_2) == -1,
            "CreateSemaphorePair", err);

    errno = 0;
    ssize_t nread = getline(&line, &len, stream);
    GOTO_IF(errno == ENOMEM, "getline", err);
    if (nread == -1) {
        fprintf(stderr, "Input 2 filenames\n");
        goto err;
    }
    line[nread - 1] = '\0';

    errno = 0;
    nread = getline(&extraLine, &extraLen, stream);
    GOTO_IF(errno == ENOMEM, "getline", err);
    if (nread == -1) {
        fprintf(stderr, "Input 2 filenames\n");
        goto err;
```

```
55      }
56      extraLine[nread - 1] = '\0';
57
58      fds[0] = open(line, O_CREAT | O_WRONLY | O_TRUNC, MODE);
59      GOTO_IF(fds[0] == -1, "open", err);
60      fds[1] = open(extraLine, O_CREAT | O_WRONLY | O_TRUNC, MODE);
61      GOTO_IF(fds[1] == -1, "open", err);
62      free(extraLine);
63      extraLine = NULL;
64
65      pid_t pids[2];
66      pids[0] = fork();
67      GOTO_IF(pids[0] == -1, "fork", err);
68      if (pids[0]) {
69          pids[1] = fork();
70          GOTO_IF(pids[1] == -1, "fork", err);
71      }
72
73      if (pids[0] == 0) {  // child1
74          close(fds[1]);
75          fds[1] = -1;
76
77          GOTO_IF(dup2(shm1.fd, STDIN_FILENO) == -1, "dup2", err);
78          GOTO_IF(dup2(fds[0], STDOUT_FILENO) == -1, "dup2", err);
79          close(fds[0]);
80          fds[0] = -1;
81
82          GOTO_IF(execl(pathToChild, "child_lab3", pair1.wsemname,
83                        pair1.revsemname, NULL),
84                  "execl", err);
85      } else if (pids[1] == 0) {  // child2
86          close(fds[0]);
87          fds[0] = -1;
88
89          GOTO_IF(dup2(shm2.fd, STDIN_FILENO) == -1, "dup2", err);
90          GOTO_IF(dup2(fds[1], STDOUT_FILENO) == -1, "dup2", err);
91          close(fds[1]);
92          fds[1] = -1;
93
94          GOTO_IF(execl(pathToChild, "child_lab3", pair2.wsemname,
95                        pair2.revsemname, NULL),
96                  "execl", err);
97      } else {  // parent
98          close(fds[0]);
99          fds[0] = -1;
100         close(fds[1]);
101         fds[1] = -1;
102         while ((nread = getline(&line, &len, stream)) != -1) {
103             if (nread <= FILTER_LEN) {
104                 GOTO_IF(SemTimedWait(pair1.wsemptr) == -1, "
    semTimeout", err);
105                 strncpy(shm1.ptr + 1, line, nread + 1);
106                 shm1.ptr[0] = 0;
107                 sem_post(pair1.revsemptr);
108             } else {
109                 GOTO_IF(SemTimedWait(pair2.wsemptr) == -1, "
    semTimeout", err);
110                 strncpy(shm2.ptr + 1, line, nread + 1);
111                 shm2.ptr[0] = 0;
```

```
                sem_post(pair2.revsemptr);
            }
        }

        GOTO_IF(SemTimedWait(pair1.wsemptr) == -1, "semTimeout",
    err);
        GOTO_IF(SemTimedWait(pair2.wsemptr) == -1, "semTimeout",
    err);
        shm1.ptr[0] = -1;
        shm2.ptr[0] = -1;
        sem_post(pair2.revsemptr);
        sem_post(pair1.revsemptr);

        GOTO_IF(errno == ENOMEM, "getline", err);

        for (int i = 0; i < 2; ++i) {
            int status;
            int waitPid = wait(&status);
            GOTO_IF(status || !(waitPid == pids[0] || waitPid ==
    pids[1]),
                    "wait", err);
        }

        free(line);
        DestroySemaphorePair(&pair1);
        DestroySemaphorePair(&pair2);
        DestroySharedMemory(&shm1);
        DestroySharedMemory(&shm2);
    }

    return 0;

err:
    free(line);
    free(extraLine);
    errno = 0;
    close(fds[0]);
    close(fds[1]);
    DestroySemaphorePair(&pair1);
    DestroySemaphorePair(&pair2);
    DestroySharedMemory(&shm1);
    DestroySharedMemory(&shm2);
    if (errno == EIO) {
        abort();
    }
    return -1;
}
```

child.c

```
#include <errno.h>
#include <fcntl.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>
```

```
11
12 #include "error_handling.h"
13 #include "utils.h"
14
15 int main(int argc, char* argv[]) {
16     (void)argc;
17     char* ptr = (char*)mmap(NULL, MAP_SIZE, PROT_READ | PROT_WRITE
       , MAP_SHARED,
18                             STDIN_FILENO, 0);
19     SemaphorePair pair;
20
21     GOTO_IF(CreateSemaphorePair(&pair, argv[1], argv[2]) == -1,
22             "CreateSemaphorePair", err);
23     while (1) {
24         SemTimedWait(pair.revsemptr);
25         if (ptr[0] == 0) {
26             size_t len = strlen(ptr + 1);
27             ReverseString(ptr + 1, len - 1);
28             printf("%s", ptr + 1);
29             sem_post(pair.wsemptr);
30         } else {
31             break;
32         }
33     }
34     ABORT_IF(munmap(ptr, MAP_SIZE), "munmap");
35     ABORT_IF(sem_close(pair.wsemptr), "sem_close");
36     ABORT_IF(sem_close(pair.revsemptr), "sem_close");
37     return 0;
38 err:
39     ABORT_IF(munmap(ptr, MAP_SIZE), "munmap");
40     ABORT_IF(sem_close(pair.wsemptr), "sem_close");
41     ABORT_IF(sem_close(pair.revsemptr), "sem_close");
42     return -1;
43 }
```

utils.c

```
1 #include "utils.h"
2
3 #include <fcntl.h>
4 #include <semaphore.h>
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <sys/mman.h>
8 #include <time.h>
9 #include <unistd.h>
10
11 #include "error_handling.h"
12
13 void ReverseString(char* string, size_t length) {
14     for (size_t i = 0; i < length >> 1; ++i) {
15         char temp = string[i];
16         string[i] = string[length - i - 1];
17         string[length - i - 1] = temp;
18     }
19 }
20
21 int CreateSharedMemory(SharedMemory* shm, const char* name, size_t
       size) {
22     shm->fd = shm_open(name, O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
```

```
    if (shm->fd == -1) {
        return -1;
    }
    int status = ftruncate(shm->fd, (off_t)size);
    if (status == -1) {
        return -1;
    }
    shm->ptr =
        (char*)mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED
    , shm->fd, 0);
    if (!shm->ptr) {
        return -1;
    }
    shm->name = name;
    shm->size = size;
    return 0;
}

void DestroySharedMemory(SharedMemory* shm) {
    ABORT_IF(shm_unlink(shm->name) == -1, "shm_unlink");
    ABORT_IF(munmap(shm->ptr, shm->size) == -1, "munmap");
}

int SemTimedWait(sem_t* sem) {
    struct timespec absoluteTime;
    if (clock_gettime(CLOCK_REALTIME, &absoluteTime) == -1) {
        return -1;
    }
    absoluteTime.tv_sec += 10;
    return sem_timedwait(sem, &absoluteTime);
}

int CreateSemaphorePair(SemaphorePair* pair, const char* wname,
                        const char* revname) {
    pair->wsemptr = sem_open(wname, O_CREAT, S_IRUSR | S_IWUSR, 1)
    ;
    if (!pair->wsemptr) {
        return -1;
    }
    pair->wsemname = wname;
    pair->revsemptr = sem_open(revname, O_CREAT, S_IRUSR | S_IWUSR
    , 0);
    if (!pair->revsemptr) {
        return -1;
    }
    pair->revsemname = revname;
    return 0;
}

void DestroySemaphorePair(SemaphorePair* pair) {
    ABORT_IF(sem_unlink(pair->wsemname) == -1, "sem_unlink");
    ABORT_IF(sem_close(pair->wsemptr) == -1, "sem_close");
    ABORT_IF(sem_unlink(pair->revsemname) == -1, "sem_unlink");
    ABORT_IF(sem_close(pair->revsemptr) == -1, "sem_close");
}
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
```

```
3
4 #include "lab3.h"
5
6 int main(void) {
7     const char *childPath = getenv("PATH_TO_CHILD");
8     if (!childPath) {
9         fprintf(stderr, "Set environment variable PATH_TO_CHILD\n"
    );
10         exit(EXIT_FAILURE);
11     }
12     return ParentRoutine(childPath , stdin);
13 }
```

6 Тесты

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

extern "C" {
#include "lab3.h"
}

#include <array>
#include <filesystem>
#include <fstream>
#include <memory>

namespace fs = std::filesystem;

TEST(FirstLabTests, SimpleTest) {
    const char *childPath = getenv("PATH_TO_CHILD");
    ASSERT_TRUE(childPath);

    const std::string fileWithInput = "input.txt";
    const std::string fileWithOutput1 = "output1.txt";
    const std::string fileWithOutput2 = "output2.txt";

    constexpr int outputSize1 = 5;
    constexpr int outputSize2 = 3;
    constexpr int inputSize = outputSize1 + outputSize2;

    const std::array<std::string, inputSize> input = {
        "rev",
        "revvvvv",
        "162te16782r18223r2",
        "",
        "r",
        "pqwrpqlwfpqwoglqwpglqpgwpqw.,g;q.wg;q.wg;w.qg;.w",
        "12345678900",
        "1234567890",
    };

    const std::array<std::string, outputSize1> expectedOutput1 = {
        "ver",
        "vvvvver",
        "",
        "r",
        "0987654321",
    };

    const std::array<std::string, outputSize2> expectedOutput2 = {
        "2r32281r28761et261",
        "w.;gq.w;gw.q;gw.q;g,.wqpwgpqlgpwqlgowqpfwlqprwqp",
        "00987654321",
    };

    {
        auto inFile = std::ofstream(fileWithInput);
        inFile << fileWithOutput1 << '\n'
               << fileWithOutput2 << '\n';
        for (const auto &line : input) {
            inFile << line << '\n';
        }
```

```
    }

    auto deleter = [](FILE *file) {
        fclose(file);
    };

    std::unique_ptr<FILE, decltype(deleter)> inFile(fopen(
    fileWithInput.c_str(), "r"), deleter);

    ASSERT_EQ(ParentRoutine(childPath, inFile.get()), 0);

    auto outFile1 = std::ifstream(fileWithOutput1);
    auto outFile2 = std::ifstream(fileWithOutput2);
    ASSERT_TRUE(outFile1.good() && outFile2.good());

    std::string result;

    for (const std::string &line : expectedOutput1) {
        std::getline(outFile1, result);
        EXPECT_EQ(result, line);
    }

    for (const std::string &line : expectedOutput2) {
        std::getline(outFile2, result);
        EXPECT_EQ(result, line);
    }

    auto removeIfExists = [](const std::string &path) {
        if (fs::exists(path)) {
            fs::remove(path);
        }
    };

    removeIfExists(fileWithInput);
    removeIfExists(fileWithOutput1);
    removeIfExists(fileWithOutput2);
}

TEST(FirstLabTests, ZeroOutputFileTest) {
    const char *childPath = getenv("PATH_TO_CHILD");
    ASSERT_TRUE(childPath);
    const std::string fileWithInput = "input.txt";

    {
        auto inFile = std::ofstream(fileWithInput);
    }
    auto deleter = [](FILE *file) {
        fclose(file);
    };

    std::unique_ptr<FILE, decltype(deleter)> inFile(fopen(
    fileWithInput.c_str(), "r"), deleter);

    ASSERT_EQ(ParentRoutine(childPath, inFile.get()), -1);
}

TEST(FirstLabTests, OneOutputFileTest) {
    const char *childPath = getenv("PATH_TO_CHILD");
    ASSERT_TRUE(childPath);
```

```
115     const std::string fileWithInput = "input.txt";
116     const std::string fileWithOutput = "output.txt";
117 
118     {
119         auto inFile = std::ofstream(fileWithInput);
120         inFile << fileWithOutput << '\n';
121     }
122     auto deleter = [](FILE *file) {
123         fclose(file);
124     };
125 
126     std::unique_ptr<FILE, decltype(deleter)> inFile(fopen(
    fileWithInput.c_str(), "r"), deleter);
127 
128     ASSERT_EQ(ParentRoutine(childPath, inFile.get()), -1);
129     ASSERT_FALSE(fs::exists(fileWithOutput));
130 }
131 
132 TEST(FirstLabTests, EmptyInputTest) {
133     const char *childPath = getenv("PATH_TO_CHILD");
134     ASSERT_TRUE(childPath);
135 
136     const std::string fileWithInput = "input.txt";
137     const std::string fileWithOutput1 = "output1.txt";
138     const std::string fileWithOutput2 = "output2.txt";
139 
140     {
141         auto inFile = std::ofstream(fileWithInput);
142         inFile << fileWithOutput1 << '\n'
143                << fileWithOutput2 << '\n';
144     }
145 
146     auto deleter = [](FILE *file) {
147         fclose(file);
148     };
149 
150     std::unique_ptr<FILE, decltype(deleter)> inFile(fopen(
    fileWithInput.c_str(), "r"), deleter);
151 
152     ASSERT_EQ(ParentRoutine(childPath, inFile.get()), 0);
153 
154     auto outFile1 = std::ifstream(fileWithOutput1);
155     auto outFile2 = std::ifstream(fileWithOutput2);
156     ASSERT_TRUE(outFile1.good() && outFile2.good());
157     ASSERT_TRUE(fs::is_empty(fileWithOutput1) && fs::is_empty(
    fileWithOutput2));
158 
159     auto removeIfExists = [](const std::string &path) {
160         if (fs::exists(path)) {
161             fs::remove(path);
162         }
163     };
164 
165     removeIfExists(fileWithInput);
166     removeIfExists(fileWithOutput1);
167     removeIfExists(fileWithOutput2);
168 }
```

7 Консоль

```
ivan@asus-vivobook ~/c/o/b/lab3 (reports)> PATH_TO_CHILD=child_lab3 lab3
1
2
sds
dfsdfs
sdf
s
ccvvf
df

fdfffffffffffffg2f52g2
ivan@asus-vivobook ~/c/o/b/lab3 (reports)> ls
1  2  CMakeFiles/  CTestTestfile.cmake  Makefile  child_lab3*  cmake_install.cmake
ivan@asus-vivobook ~/c/o/b/lab3 (reports)> cat 1
sds
sfdsfd
fds
s
fvvcc
fd

ivan@asus-vivobook ~/c/o/b/lab3 (reports)> cat 2
2g25f2gfffffffffffffdf
```

8 Запуск тестов

```
ivan@asus-vivobook ~/c/o/b/tests (reports) [SIGINT]> PATH_TO_CHILD=../lab3/child_la
Running main() from /var/tmp/portage/dev-cpp/gtest-1.13.0/work/googletest-1.13.0/go
[==========] Running 4 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 4 tests from FirstLabTests
[ RUN      ] FirstLabTests.SimpleTest
[       OK ] FirstLabTests.SimpleTest (9 ms)
[ RUN      ] FirstLabTests.ZeroOutputFileTest
Input 2 filenames
[       OK ] FirstLabTests.ZeroOutputFileTest (0 ms)
[ RUN      ] FirstLabTests.OneOutputFileTest
Input 2 filenames
[       OK ] FirstLabTests.OneOutputFileTest (0 ms)
[ RUN      ] FirstLabTests.EmptyInputTest
[       OK ] FirstLabTests.EmptyInputTest (4 ms)
[----------] 4 tests from FirstLabTests (15 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 1 test suite ran. (16 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».