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

Лабораторная работа №2 по курсу “Операционные системы”

Студент: Сибирцев Роман Денисович

Группа: М8О-208Б-22

Преподаватель: Миронов Евгений Сергеевич

Вариант: 13

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2023

Содержание

1 Репозиторий

https://github.com/RomanSibirtsev/MAIOSlabs

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 использовал 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

utils.hpp

```
#pragma once

#include <iostream>
#include <fstream>
#include <string>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <semaphore.h>

sem_t* CreateSemaphore(const char *name, int value);
int CreateShm(const char* name);
char* MapSharedMemory(const int size, int fd);
int CreateFork();

constexpr const char *SEM_1 = "SEM_1";
constexpr const char *SEM_2 = "SEM_2";
constexpr const char *SEM_3 = "SEM_3";

const int FILE_SIZE = 1024;
```

utils.cpp

```
#include "utils.hpp"

sem_t* CreateSemaphore(const char *name, int value) {
    sem_t *semptr = sem_open(name, O_CREAT, S_IRUSR | S_IWUSR,
    value);
    if (semptr == SEM_FAILED){
        perror("Couldn't open the semaphore");
        exit(EXIT_FAILURE);
    }
    return semptr;
}

int CreateShm(const char* name) {
    int fd = shm_open(name, O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
    if (fd == -1) {
        std::cerr << "Failed shm_open\n";
        exit(-1);
    }
    ftruncate(fd, 1024);
    return 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;
}

int CreateFork() {
    int pid;
    pid = fork();
    if (pid == -1) {
        std::cerr << "Failed fork()\n";
        exit(-2);
    }
    return pid;
}
```

parent.hpp

```
#pragma once


#include "utils.hpp"

int ParentWork();
```

parent.cpp

```
#include "parent.hpp"
#include "utils.hpp"

int ParentWork() {
    pid_t pid;
    int status;;
    char *child1;
    char *child2;
    std::string line;

    char* name1 = "/shm1";
    sem_t *semptr1 = CreateSemaphore(SEM_1, 0);
    int shd_fd1 = CreateShm(name1);
    char *memptr1 = MapSharedMemory(FILE_SIZE, shd_fd1);

    char* name2 = "/shm2";
    sem_t *semptr2 = CreateSemaphore(SEM_2, 0);
    int shd_fd2 = CreateShm(name2);
    char *memptr2 = MapSharedMemory(FILE_SIZE, shd_fd2);

    char* name3 = "/shm3";
    sem_t *semptr3 = CreateSemaphore(SEM_3, 0);
    int shd_fd3 = CreateShm(name3);
    char *memptr3 = MapSharedMemory(FILE_SIZE, shd_fd3);

    child1 = getenv("PATH_CHILD1");
    child2 = getenv("PATH_CHILD2");

    pid = CreateFork();
    if (pid == 0) { // child 1
        if (execlp(child1, "lower.out", name1, name2, nullptr) ==
    -1) {
            std::cerr << "Failed execlp()\n";
            exit(-5);
        }
    }
```

```
36
37      pid = CreateFork();
38      if (pid == 0) { // child 2
39          if (execlp(child2, "underscore.out", name2, name3, nullptr
    ) == -1) {
40              std::cerr << "Failed execlp()\n";
41              exit(-5);
42          }
43      }
44
45      if (pid != 0) {
46          while (std::getline(std::cin, line)) {
47              line += '\n';
48              strcpy(memptr1, line.c_str());
49              sem_post(semptr1);
50
51              if (line == "\n") {
52                  strcpy(memptr1, "\0");
53                  sem_post(semptr1);
54                  break;
55              }
56              sem_wait(semptr3);
57              std::string_view st(memptr3);
58              std::string s = {st.begin(), st.end()};
59              if (s != "\n") {
60                  std::cout << s << std::endl;
61                  strcpy(memptr3, "\n");
62              }
63          }
64      }
65
66      waitpid(-1, &status, 0);
67      waitpid(-1, &status, 0);
68
69      sem_close(semptr1);
70      sem_unlink(SEM_1);
71      shm_unlink("/shm1");
72      munmap(memptr1, FILE_SIZE);
73      close(shd_fd1);
74
75      sem_close(semptr2);
76      sem_unlink(SEM_2);
77      shm_unlink("/shm2");
78      munmap(memptr2, FILE_SIZE);
79      close(shd_fd2);
80
81      sem_close(semptr3);
82      sem_unlink(SEM_3);
83      shm_unlink("/shm3");
84      munmap(memptr3, FILE_SIZE);
85      close(shd_fd3);
86      return 0;
87  }
```

underscore.cpp

```
1  #include "utils.hpp"
2
3  int main (int argc, char** argv) {
4      if (argc != 3) {
```

```
5           std::cerr << "Wrong argc in underscore.out\n";
6           exit(-6);
7       }
8
9       const char* fileName = argv[1];
10      const char* fileName2 = argv[2];
11
12      sem_t *semptr1 = CreateSemaphore(SEM_2, 0);
13      int shd_fd1 = CreateShm(fileName);
14      char *memptr1 = MapSharedMemory(FILE_SIZE, shd_fd1);
15
16      sem_t *semptr2 = CreateSemaphore(SEM_3, 0);
17      int shd_fd2 = CreateShm(fileName2);
18      char *memptr2 = MapSharedMemory(FILE_SIZE, shd_fd2);
19
20      while(true) {
21          sem_wait(semptr1);
22          std::string_view st(memptr1);
23          std::string s = {st.begin(), st.end()};
24          for (int i = 0; i < s.size(); ++i) {
25              if (s[i] == ' ') {
26                  s[i] = '_';
27              }
28          }
29
30          if (s != "\n")  {
31              strcpy(memptr2, s.c_str());
32              strcpy(memptr1, "\n");
33              sem_post(semptr2);
34          }
35
36          if (s == "\0") {
37              break;
38          }
39      }
40
41      sem_close(semptr1);
42      sem_unlink(SEM_2);
43      shm_unlink(fileName);
44      munmap(memptr1, FILE_SIZE);
45      close(shd_fd1);
46
47      sem_close(semptr2);
48      sem_unlink(SEM_3);
49      shm_unlink(fileName2);
50      munmap(memptr2, FILE_SIZE);
51      close(shd_fd2);
52
53  }
```

6 Тесты

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

#include "parent.hpp"

void TestParent(std::string &src, std::string &res) {
    std::istringstream srcStream(src);

    std::streambuf* buf = std::cin.rdbuf(srcStream.rdbuf());

    testing::internal::CaptureStdout();
    ParentWork();
    ASSERT_EQ(testing::internal::GetCapturedStdout(), res + '\n');

    std::cin.rdbuf(buf);
}


TEST(cin_test, ONE) {
    std::string src = "       \n\n";
    std::string res = "_______\n";
    TestParent(src, res);
}

TEST(cin_test, TWO) {
    std::string src = "AHAHAHAHAH\n\n";
    std::string res = "ahahahahah\n";
    TestParent(src, res);
}

TEST(cin_test, THREE) {
    std::string src = "   HELLO wOrLd 12 3\n\n";
    std::string res = "___hello_world_12_3\n";
    TestParent(src, res);
}
```

7 Запуск тестов

```
roman@DESKTOP-K3DH39N:~/MAI/MAI_OS_labs/build/tests$ ./lab3_test
Running main() from /home/roman/MAI/MAI_OS_labs/build/_deps/googletest-src/googlete
[==========] Running 3 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 3 tests from cin_test
[ RUN      ] cin_test.ONE
[       OK ] cin_test.ONE (14 ms)
[ RUN      ] cin_test.TWO
[       OK ] cin_test.TWO (1 ms)
[ RUN      ] cin_test.THREE
[       OK ] cin_test.THREE (1 ms)
[----------] 3 tests from cin_test (16 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test suite ran. (16 ms total)
[  PASSED  ] 3 tests.
```

8 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 приобрел практические навыки в освоении принципов работы с файловыми системами и обеспечении обмена данных между процессами посредством технологии «File mapping».