Московский авиационный институт
(Национальный исследовательский университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу “Операционные системы”

Студент: Немкова Анастасия Романовна

Группа: М8О-208Б-22

Преподаватель: Миронов Евгений Сергеевич

Вариант: 15

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2023

Содержание

Репозиторий

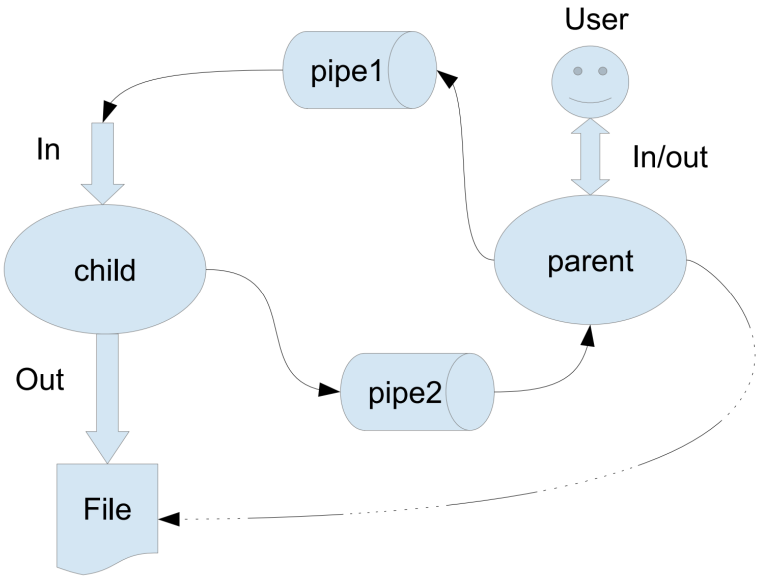
`https://github.com/anastasia-nemkova/OS_labs`

Постановка задачи

Цель работы:

Изучить управление процессами в ОС и обеспечение обмена данных между процессами посредством каналов

Задание:



Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода

Вариант 15) Правило проверки: строка должна начинаться с заглавной буквы

Общие сведения о программе

Программа компилируется из файлов parent.cpp с основным процессом, child.cpp с дочерним процессом и utils.cpp с вспомагательными функциями. Также имеются заголовочные файлы parent.hpp и utils.hpp и файл с тестами lab1_test.cpp. В программе работы были использованы следующие системные вызовы:

- fork() - создание нового процесса.
- execlp() - замена текущего образа процесса новым.
- pipe() - создание однонапрвленного канала для передачи строк родительским процессом дочернему
- read() - чтение из pipe
- write() - запись в pipe
- close() - закрытие файлового дискриптора
- dup2() - перенаправление одиного файлового дескриптора на другой

Общий метод и алгоритм решения

В родительском процессе до создания дочернего считываем имя файла, в который в последствии будем записывать строки, начиначинающиеся с большой буквы, и переопределяем поток ввода и вывода. Создаем два канала: для передачи строк из родительского процесса в дочерний и для передачи ошибок из дочернего процесса в родительский и используем execlp для запуска файла дочернего процесса. В родительском процессе считываем строки и записываем их в первый канал, в дочернем процессе обрабатываем эти строки, если они удовлетворяют правилу - записываем их в файл, иначе записываем ошибку и саму строку во второй канал. В родительском процессе с помощью функции ReadFromPipe2 читаем переданные дочерним процессом ошибки и выводим их на экран.

Исходный код

parent.hpp

```
#pragma once

#include "utils.hpp"

void ParentRoutine(const char* pathToChild);
```

parent.cpp

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

// fd[1] – запись
// fd[0] – чтение

void ParentRoutine(const char* pathToChild){
    std::string fileName;
    getline(std::cin, fileName);

    int fd_1[2];
    int fd_2[2];
    CreatePipe(fd_1); //pipe между родителем и дочерним
    CreatePipe(fd_2); //pipe между дочерним и родителем


    int pid = CreateChildProcess();// создаем дочерний процесс

    if (pid == 0) { // дочерний процесс
        close(fd_1[1]); //закрываем запись в pipe1
        close(fd_2[0]); //закрываем чтение из pipe2

        dup2(fd_1[0], STDIN_FILENO); // перенаправляем считанные с pipe1 данные на стандартный вывод
        dup2(fd_2[1], STDOUT_FILENO);

        if (execlp(pathToChild, pathToChild, fileName.c_str(), nullptr) == -1) { // запускаем child.cpp
            perror("Child start error");
            exit(EXIT_FAILURE);
        }

    } else { // родительский процесс


        close(fd_1[0]); //закрываем чтение из pipe1
        close(fd_2[1]); //закрываем запись в pipe2

        std::string str;
        while (std::getline(std::cin, str)) { //считываем строку с консоли и записываем в pipe1
            write(fd_1[1], str.c_str(), str.length());
            write(fd_1[1],"\n", 1);
        }
        close(fd_1[1]);

        ReadFromPipe2(fd_2[0]);
        close(fd_2[0]);

        wait(NULL);
    }
}
```

utils.hpp

```
#pragma once

#include <iostream>
```

```cpp
#include <string>
#include <fstream>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

void CreatePipe(int fd[2]);
pid_t CreateChildProcess();
bool StartsWithCapital(const std::string& str);
void ReadFromPipe2(int fd);
```

utils.cpp

```cpp
#include "utils.hpp"

void CreatePipe(int fd[2]) {
    if (pipe(fd) == -1) {
        perror("Error creating pipe");
        exit(EXIT_FAILURE);
    }
}

pid_t CreateChildProcess() {
    pid_t pid = fork();
    if (pid == -1) {
        perror("Error creating process");
        exit(EXIT_FAILURE);
    }
    return pid;
}

bool StartsWithCapital(const std::string& str) {
    return !(str.empty()) && isupper(str[0]);
}

void ReadFromPipe2(int fd) {
    constexpr int BUFFER_SIZE = 1024;
    char buffer[BUFFER_SIZE] = "";
    ssize_t bytesRead;

    while ((bytesRead = read(fd, &buffer, BUFFER_SIZE)) > 0) {
        std::cout.write(buffer, bytesRead);
    }

    std::cout << std::endl;

    if (bytesRead == -1) {
        perror("Error reading from pipe2");
        exit(EXIT_FAILURE);
    }
}
```

child.cpp

```cpp
#include "utils.hpp"

int main(const int argc, const char* argv[]) {
    if (argc != 2) {
        perror("Necessary arguments were not provided");
        exit(EXIT_FAILURE);
    }
    const char* fileName = argv[1];
    std::ofstream out(fileName);
    if(!out.is_open()) {
        perror("Error open");
        exit(EXIT_FAILURE);
    }
    std::string str;
    while (std::getline(std::cin, str)) {
```

```
16          if (StartsWithCapital(str)) {
17              out << str << std::endl;
18          } else {
19              std::string error  = "String: " + str + " - doesn't start with capital
        letter";
20              std::cout << error << std::endl;
21          }
22      }
23      exit(EXIT_FAILURE);
24  }
```

main.cpp

```
1  #include "parent.hpp"
2
3  int main() {
4
5      ParentRoutine(getenv("PATH_TO_CHILD"));
6
7      exit(EXIT_SUCCESS);
8  }
9  // "../lab1/child"
```

lab1_test.cpp

```
1  #include <gtest/gtest.h>
2
3
4  #include "parent.hpp"
5
6  void testingProgram(const std::vector<std::string>& input, const std::vector<std::
       string>& expectedOutput, const std::vector<std::string>& expectedFile) {
7      const char* fileName = "test.txt";
8
9      // Записываем входные данные в файл
10     std::ofstream outFile(fileName);
11     if (!outFile.is_open()) {
12         std::cerr << "Error opening file: " << fileName << std::endl;
13         return;
14     }
15
16     outFile << fileName << std::endl;
17     for (const std::string &line : input) {
18         outFile << line << std::endl;
19     }
20     outFile.close();
21
22     // Сохраняем старый буфер для восстановления
23     std::streambuf* oldCoutBuf = std::cout.rdbuf();
24     std::streambuf* oldCinBuf = std::cin.rdbuf();
25
26     // Подключаем файл ввода
27     std::ifstream inFile(fileName);
28     std::cin.rdbuf(inFile.rdbuf());
29
30     // Захватываем вывод программы
31     std::stringstream capturedOutput;
32     std::cout.rdbuf(capturedOutput.rdbuf());
33
34     ParentRoutine(getenv("PATH_TO_CHILD"));
35
36     // Восстанавливаем стандартные буферы
37     std::cin.rdbuf(oldCinBuf);
38     std::cout.rdbuf(oldCoutBuf);
39
40     // Проверяем вывод программы
41     std::istringstream outputStream(capturedOutput.str());
42     for (const std::string &expectation : expectedOutput) {
43         std::string result;
44         std::getline(outputStream, result);
```

```
45          EXPECT_EQ(result, expectation);
46      }
47
48      // Проверяем содержимое файла
49      std::ifstream fileStream(fileName);
50      for (const std::string &expectation : expectedFile) {
51          std::string result;
52          std::getline(fileStream, result);
53          EXPECT_EQ(result, expectation);
54      }
55
56      fileStream.close();
57      std::remove(fileName);
58  }
59
60  TEST(firstLabTests, simpleTest) {
61      std::vector<std::string> input = {"AAA"};
62      std::vector<std::string> expectedOutput = {};
63      std::vector<std::string> expectedFile = {"AAA"};
64      testingProgram(input, expectedOutput, expectedFile);
65  }
66
67  TEST(firstLabTests, emptystrTest) {
68      std::vector<std::string> input = {};
69      std::vector<std::string> expectedOutput = {};
70      std::vector<std::string> expectedFile = {};
71      testingProgram(input, expectedOutput, expectedFile);
72  }
73
74  TEST(firstLabTests, sonneTest) {
75      std::vector<std::string> input = {
76          "Alle warten auf das Licht",
77          "Furchtet euch, furchtet euch nicht.",
78          "die Sonne scheint mir aus den Augen,",
79          "sie wird heute Nacht nicht untergehen",
80          "Und die Welt zahlt laut bis zehn"
81      };
82
83      std::vector<std::string> expectedOutput = {
84          "String: die Sonne scheint mir aus den Augen, — doesn't start with capital
        letter",
85          "String: sie wird heute Nacht nicht untergehen — doesn't start with
      capital letter"
86      };
87
88      std::vector<std::string> expectedFile = {
89          "Alle warten auf das Licht",
90          "Furchtet euch, furchtet euch nicht.",
91          "Und die Welt zahlt laut bis zehn"
92      };
93      testingProgram(input, expectedOutput, expectedFile);
94  }
95
96  TEST(firstLabTests, deathTest) {
97      std::vector<std::string> input = {
98          "Can death be sleep, when life is but a dream,",
99          "And scenes of bliss pass as a phantom by?",
100         "the transient pleasures as a vision seem,",
101         "And yet we think the greatest pain's to die."
102     };
103
104     std::vector<std::string> expectedOutput = {
105         "String: the transient pleasures as a vision seem, — doesn't start with
      capital letter"
106     };
107
108     std::vector<std::string> expectedFile = {
109         "Can death be sleep, when life is but a dream,",
110         "And scenes of bliss pass as a phantom by?",
```

```
111         "And yet we think the greatest pain's to die."
112     };
113     testingProgram(input, expectedOutput, expectedFile);
114 }
115
116
117 int main(int argc, char *argv[]) {
118     testing::InitGoogleTest(&argc, argv);
119     return RUN_ALL_TESTS();
120 }
```

Демонстрация работы программы

```
arnemkova@LAPTOP-TA2RV74U:~/OS_labs/build$ ./tests/lab1_test
[==========] Running 4 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 4 tests from firstLabTests
[ RUN      ] firstLabTests.simpleTest
[       OK ] firstLabTests.simpleTest (2 ms)
[ RUN      ] firstLabTests.emptystrTest
[       OK ] firstLabTests.emptystrTest (1 ms)
[ RUN      ] firstLabTests.sonneTest
[       OK ] firstLabTests.sonneTest (1 ms)
[ RUN      ] firstLabTests.deathTest
[       OK ] firstLabTests.deathTest (1 ms)
[----------] 4 tests from firstLabTests (6 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 1 test suite ran. (6 ms total)
[  PASSED  ] 4 tests.
```

Вывод

В ходе выполнения данной лабораторной работы я изучила работу процессов, реализацию обмена информацией между дочерним и родительским процессами с использованием каналов в ОС Linux.

В процессе работы я познакомилась с системными вызовами, которые представляют собой интерфейс для взаимодействия прикладных программ с операционной системой.

Системные вызовы отличаются от обычных функций тем, что они выполняются на уровне ядра операционной системы и предоставляют доступ к ресурсам, защищенным от прямого доступа программ. Эти вызовы позволяют программам выполнять такие операции, как создание процессов (fork), создание каналов для межпроцессного взаимодействия (pipe), замещение текущего процесса новой программой (execlp) и дублирование файловых дескрипторов (dup2), что широко используется для реализации различных аспектов многозадачности и взаимодействия процессов в операционных системах.

Также я изучила устройство памяти процесса, которое делится на несколько секций, каждая из которых отвечает за определенные части работы программы.