Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа № 1 по курсу «Операционные системы»

Управление процессами в ОС.
Обеспечение обмена данных между процессами посредством каналов

Студент: Арусланов К. А.
Преподаватель: Миронов Е. С.
Группа: М8О-203Б-23
Дата:
Оценка:
Подпись:

Москва, 2024

Условие

Составить и отладить программу на языке C/C++, 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 каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Задание

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через pipe1, который связан со стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип float. Количество чисел может быть произвольным.

Метод решения

Были созданы родительский и дочерний процессы, которые обмениваются данными посредством пайпов. Родитель записывал данные (числа, строку и т. п.), а дочерний процесс читал их, выполнял требуемую операцию (например, суммирование), и записывал результат в файл, а также передавал его родителю через пайп.

Код программы

lab1.h

```
#ifndef LAB1_H
#define LAB1_H

#include <iostream>
#include <iomanip> // setprecision()
#include <fstream> // Для работы с файлами через std::ofstream и std::ifstream
#include <unistd.h> // Для системных вызовов POSIX (fork(), pipe(), dup2(), read(),
write())
#include <sys/wait.h> // wait()
#include <cstdlib> // exit()

void RunParentProcess(std::istream&);
```

```
#endif
```

child.cpp

```
#include "lab1.h"

int main() {
    // Чтение имени файла из pipe1 (STDIN)
    char fileName[50];
    read(0, fileName, 50);

    // Чтение чисел из pipe1 и подсчет суммы
    float num, sum = 0;
    while (read(0, &num, sizeof(num)) > 0) {
        sum += num;
    }

    {
        // Открытие файла для записи
        std::ofstream file(fileName);
        if (!file.is_open()) {
            std::cerr << "Ошибка при открытии файла" << std::endl;
            exit(1);
        }

        // Запись суммы в файл
        file << std::fixed << std::setprecision(6);
        file << "Сумма: " << sum << '\n';
    }

    // Передача результата родительскому процессу через pipe2 (STDOUT)
    std::cout << std::fixed << std::setprecision(6);
    std::cout << "Сумма: " << sum << '\n';

    return 0;
}
```

parent.cpp

```
#include "lab1.h"

void RunParentProcess(std::istream& stream) {
    int pipe1[2], pipe2[2];

    if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {
        std::cerr << "Ошибка создания pipe" << std::endl;
        exit(1);
    }
```

```
pid_t pid = fork();

if (pid < 0) {
    std::cerr << "Ошибка fork" << std::endl;
    exit(1);
}

if (pid > 0) {
    // Родительский процесс

    // Закрывваем неиспользуемое родительским процессом
    close(pipe1[0]);
    close(pipe2[1]);

    char fileName[50];
    std::cout << "Введите имя файла:\n";
    stream >> fileName;

    // Отправляем имя файла дочернему процессу
    if (write(pipe1[1], &fileName, 50) == -1) {
        std::cerr << "Ошибка write" << std::endl;
        close(pipe1[1]);
        exit(1);
    }

    // Отправляем числа дочернему процессу
    float num;
    std::cout << "Введите числа (EOF для завершения):\n";
    while (stream >> num) {
        if (write(pipe1[1], &num, sizeof(num)) == -1) {
            std::cerr << "Ошибка write" << std::endl;
            close(pipe1[1]);
            exit(1);
        };
    }
    close(pipe1[1]);

    // Ждем завершения дочернего процесса
    wait(nullptr);

    // Получаем результат из pipe2
    char buffer[100];
    int bytesRead = read(pipe2[0], buffer, sizeof(buffer) - 1);
    if (bytesRead > 0) {
        buffer[bytesRead] = '\0';  // Завершаем строку
    }
    close(pipe2[0]);

    std::cout << "\nРезультат от дочернего процесса: " << buffer;
} else {
    // Дочерний процесс

    // Закрываем неиспользуемое
```

```
        close(pipe1[1]);
        close(pipe2[0]);

        // Готовим почву для последующего принятия fileName через pipe и передачи
результата родителю
        if (dup2(pipe1[0], 0) == -1 or dup2(pipe2[1], 1) == -1) {
            std::cerr << "Ошибка dup2" << std::endl;
            exit(1);
        }
        close(pipe1[0]);  // Закрываем конец чтения pipe1, он больше не нужен, раз мы
переопределили вход
        close(pipe2[1]);  // Закрываем конец записи pipe2

        // Дочерний процесс через exec
        const char* pathToChild = std::getenv("PATH_TO_CHILD");
        execlp(pathToChild, pathToChild, nullptr);

        std::cerr << "Ошибка exec" << std::endl;
        exit(1);
    }
}
```

main.cpp

```
#include "lab1.h"

int main() {
    std::cout << "Запуск родительского процесса...\n";
    RunParentProcess(std::cin);
    return 0;
}
```

CMakeLists.txt

```
add_executable(lab1 main.cpp src/parent.cpp include/lab1.h)
add_executable(child src/child.cpp include/lab1.h)

target_include_directories(lab1 PRIVATE include)
target_include_directories(child PRIVATE include)
```

lab1_test.cpp

```
#include <gtest/gtest.h>
#include <lab1.h>

TEST(ParentProcessTest, CheckSumCalculation) {
    const char* testFileName = "output.txt";
```

```
    {
        // Ввод для родительского процесса
        std::ofstream testInput("test_input.txt");
        testInput << testFileName << "\n7.6 5.5";
    }
    {
        std::ifstream testFile("test_input.txt");
        ASSERT_TRUE(testFile.is_open()) << "Не удалось открыть файл";

        // Запуск родительского процесса
        RunParentProcess(testFile);  // Передаем поток для тестирования

        // Проверяем, что файл был создан программой
        std::ifstream resultFile(testFileName);
        ASSERT_TRUE(resultFile.good()) << "Файл не был создан";

        // Читаем результат
        std::string line;
        std::getline(resultFile, line);
        EXPECT_EQ(line, "Сумма: 13.100000") << "Неверный результат суммы в файле";
    }

    // Смываем
    std::remove(testFileName);
    std::remove("test_input.txt");
}

TEST(ParentProcessTest, EmptyInput) {
    const char* testFileName = "empty_test_file.txt";

    {
        // Ввод для родительского процесса
        std::ofstream testInput("empty_input.txt");
        testInput << testFileName;
    }
    {
        std::ifstream testFile("empty_input.txt");
        ASSERT_TRUE(testFile.is_open()) << "Не удалось открыть файл";

        // Запуск родительского процесса
        RunParentProcess(testFile);  // Передаем поток для тестирования

        // Проверяем, что файл был создан программой
        std::ifstream resultFile(testFileName);
        ASSERT_TRUE(resultFile.good()) << "Файл не был создан";

        // Читаем результат
        std::string line;
        std::getline(resultFile, line);
        EXPECT_EQ(line, "Сумма: 0.000000") << "Неверный результат для пустого ввода";
    }
```

```
    // Удаляем тестовые файлы
    std::remove(testFileName);
    std::remove("empty_input.txt");
}

TEST(ParentProcessTest, LargeNumbersInput) {
    const char* testFileName = "large_test_file.txt";

    {
        // Ввод для родительского процесса
        std::ofstream testInput("large_input.txt");
        testInput << testFileName << "\n6.1234 5.230011 1.2 3 .1";
    }
    {
        std::ifstream testFile("large_input.txt");
        ASSERT_TRUE(testFile.is_open()) << "Не удалось открыть файл";

        // Запускаем родительский процесс
        RunParentProcess(testFile);  // Передаем поток для тестирования

        // Проверяем, что файл был создан программой
        std::ifstream resultFile(testFileName);
        ASSERT_TRUE(resultFile.good()) << "Файл не был создан";

        // Читаем результат
        std::string line;
        std::getline(resultFile, line);
        EXPECT_EQ(line, "Сумма: 15.653411") << "Неверный результат для больших чисел";
    }

    // Удаляем тестовый файл
    std::remove(testFileName);
    std::remove("large_input.txt");
}
int main(int argc, char **argv) {
    testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

Выводы

Были хорошо изучены базовые системные вызовы такие как write, read, exec, dup и dup2… Получены навыки написания программ с использованием нескольких процессов и с осуществлением обмена между ними посредством каналов.