Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу
«Операционные системы»

Взаимодействие между процессами

Студент: Мирошников Дмитрий Евгеньевич
Группа: М8О-210Б-22
Вариант: 4
Преподаватель: Соколов Андрей Алексеевич
Оценка:_____
Дата:_____
Подпись:_____

Москва, 2023

Постановка задачи

Цель работы

Целью является приобретение практических навыков в:

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

Задание

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.

Общие сведения о программе

Программа компилируется при помощи утилиты CMake и запускается путем запуска ./parent. Также используется заголовочные файлы: iostream, string, stdio.h, unistd.h, sys/wait.h, fcntl.h, vector. В программе используются следующие системные вызовы:

1. **read** – функция read() считывает count байт из файла, описываемого аргументом fd, в буфер, на который указывает аргумент buf Указателю положения в файле дается приращение на количество считанных байт. Если файл открыт в текстовом режиме, то может иметь место транслирование символов.
2. **write** – функция переписывает count байт из буфера, на который указывает bufy в файл, соответствующий дескриптору файла handle. Указателю положения в файле дается приращение на количество записанных байт. Если файл открыт в текстовом режиме, то символы перевода строки автоматически дополняются символами возврата каретки.
3. **pipe** – создаёт механизм ввода вывода, который называется конвейером. Возвращаемый файловый дескриптор можно использовать для операций

чтения и записи. Когда в конвейер что-то записывается, то буферизуется до 504 байтов данных, после чего процесс записи приостанавливается.

4. **fork** - вызов создаёт новый процесс посредством копирования вызывающего процесса. Новый процесс считается дочерним процессом. Вызывающий процесс считается родительским процессом.
5. **close** - закрывает файловый дескриптор, который после этого не ссылается ни на один и файл и может быть использован повторно. Все блокировки, находящиеся на соответствующем файле, снимаются (независимо от того, был ли использован для установки блокировки именно этот файловый дескриптор).
6. **dup2** - системная функция используется для создания копии существующего файлового дескриптора.

Общий метод и алгоритм решения

1) Изучить принципы работы fork, pipe, read, write, close, exec*, dup2.
2) Написать две программмы для родительского и дочернего процесса.
3) Использовать в parent.cpp fork, чтобы запустить дочерний процесс.
4) При помощи конструкции if/else организовать работу с дочерним и родительским процессом.
5) В дочернем процессе скопировать файловые дескрипторы пайпов в stdinи stdout и запустить child.c при помощи execl.
6) Скомпилировать обе программы при помощи CMake и запустить ./parent

Основные файлы программы

parent.cpp

```cpp
#include <iostream>
#include <unistd.h>
#include <string.h>
#include <string>
#include <sys/wait.h>
#include <fcntl.h>
#include <vector>

int main() {
    std::string name;
    char c = 1;
    write(STDOUT_FILENO, "Enter the name of file: ", 24);
    while (c != '\n') {
        read(STDIN_FILENO, &c, sizeof(char));
        if (c != '\n') {
            name += c;
        }
```

```
    }
```

```
    int fd1[2], fd2[2];
    int temp = pipe(fd1);
    if (temp == -1) {
        write(STDERR_FILENO, "An error occured with creating a pipe1", 39);
        return 1;
    }
    temp = pipe(fd2);
    if (temp == -1) {
        write(STDERR_FILENO, "An error occured with creating a pipe2", 39);
        return 1;
    }
```

```
    int write1 = fd1[1], read1 = fd1[0];
    int write2 = fd2[1], read2 = fd2[0];
```

```
    pid_t pid = fork();
    if (pid == -1) {
        write(STDERR_FILENO, "An error occured with creating a child process", 47);
        return 1;
    }
    if (pid == 0) {
        //child process
        close(write1);
        close(read2);
        temp = dup2(read1, STDIN_FILENO);
        if (temp == -1) {
            write(STDERR_FILENO, "An error occured with redirecting input", 40);
            return 1;
        }
        temp = dup2(write2, STDOUT_FILENO);
        if (temp == -1) {
            write(STDERR_FILENO, "An error occured with redirecting output", 41);
            return 1;
        }
        temp = execl("child", "child", name.c_str(), NULL);
        if (temp == -1) {
            write(STDERR_FILENO, "An error occured with runing program from a child process", 58);
            return 1;
        }
        exit(EXIT_FAILURE);
    }
    if (pid > 0) {
        //parent process
        close(read1);
        close(write2);
        char c;
        char prev = '?';
        std::vector<char> vec;
        int countvalues = 0;
        while(read(STDIN_FILENO, &c, 1)) {
            vec.push_back(c);
            if (c == '\n') {
                if ((prev >= '0') && (prev <= '9')) {
                    ++countvalues;
                }
                for (int i = 0; i < vec.size(); ++i) {
                    write(write1, &vec[i], 1);
                }
                vec.clear();
            }
            prev = c;
        }
        int temp = 0;
        while (read(read2, &c, 1)) {
            write(STDOUT_FILENO, &c, 1);
```

```
            if (c == '\n') {
                ++temp;
            }
            if (temp == countvalues) {
                break;
            }
            //write(STDOUT_FILENO, &c, 1);
        }
        close(write1);
        close(read2);
        wait(nullptr);
    }
    return 0;
}
```

child.cpp

```
#include <iostream>
#include <unistd.h>
#include <vector>
#include <fcntl.h>
#include <string>

int main(int argc, char* argv[]) {
    int file = open(argv[1], O_CREAT | O_WRONLY, S_IRWXU);
    if (file == -1) {
        write(STDERR_FILENO, "An error occured with opening a file", 37);
        return 1;
    }
    int temp = ftruncate(file, 0);
    if (temp == -1) {
        write(STDERR_FILENO, "An error occured with clearing a file", 38);
        return 1;
    }
    std::string value, answer;
    bool flag = false;
    bool one_number = true;
    bool nothing = true;
    char c;
    float result;
    while (read(STDIN_FILENO, &c, 1)) {
        if (c != '\n' ) {
            if (c !=  ' ') {
                nothing = false;
                value += c;
            } else {
                one_number = false;
                if (!flag) {
                    flag = true;
                    result = std::stof(value);
                } else {
                    if (std::stof(value) == 0) {
                        write(STDERR_FILENO, "An error occured with division by zero\n", 40);
                        exit(EXIT_FAILURE);
                    }
                    result /= std::stof(value);
                }
                value = "";
            }
        } else {
            if (nothing) {
                continue;
            }
            if (one_number) {
                result = std::stof(value);
            } else {
                if (std::stof(value) == 0) {
                    write(STDERR_FILENO, "An error occured with division by zero\n", 40);
```

```
                    exit(EXIT_FAILURE);
                }
                result /= std::stof(value);
            }
            flag = false;
            one_number = true;
            nothing = true;
            value = "";
            answer = std::to_string(result);
            for (int i = 0; i < answer.size(); ++i) {
                write(file, &answer[i], 1);
                write(STDOUT_FILENO, &answer[i], 1);
            }
            write(file, "\n", 1);
            write(STDOUT_FILENO, "\n", 1);
        }
    }

    return 0;
}
```

Пример работы

dmitrijmrsh@LAPTOP-7SMT8REA:~/Labs-git/os_lab_1/src/build$ ./parent
Enter the name of file: out
1 1 1 1
4 2 2 2
3 3 3 3

5 2 1 1
0 1 1 1
1.000000
0.500000
0.111111
2.500000
0.000000

dmitrijmrsh@LAPTOP-7SMT8REA:~/Labs-git/os_lab_1/src/build$ ./parent
Enter the name of file: out
1 1 1 1
1 1 0 1
An error occured with division by zero
1.000000

Вывод

В первой лабораторной работе я научился работать с процессами программ. Изучив работу каждого системного вызова, путем изучения их мануалов и информации из интернета и разобрав работу стандартных потоков, я понял, что умение и понимание этого позволит в будущем понимать более глубоко устройство программ и их процессов в работе. Любая современная функция работы с вводом/выводом в наше время, работает на основе read и write. А такие низкоуровневые функции, как exec* используются по сей день в

улучшенных оболочках. Управление процессами путем dup2, close и wait помогут в будущем более умело пользоваться многопроцессорными программами.