Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Лабораторная работа №1 по курсу**

**«Операционные системы»**

Группа: М8О-210Б-23

Студент: Гула Дмитрий Александрович

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 01.11.24

Москва, 2024

**Постановка задачи**

**Вариант 7.**

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

В файле записаны команды вида: «число число число<endline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип float. Количество чисел может быть произвольным.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* **fork()** — создаёт новый процесс, который является копией текущего (родительского) процесса. Возвращает PID дочернего процесса в родительском процессе и 0 в дочернем.
* **pipe()** — создаёт однонаправленный канал для обмена данными между процессами, возвращая два дескриптора: один для чтения, другой для записи.
* **dup2()** — дублирует дескриптор файла, перенаправляя один дескриптор на другой. Используется для перенаправления стандартного ввода/вывода, например, из файла или в канал.
* **execl()** — заменяет текущий процесс на новый, исполняя указанную программу. Если успешно выполняется, текущий код процесса завершается и выполняется новый код.
* **open()** — открывает файл и возвращает файловый дескриптор. Используется для открытия файла в определённом режиме (например, только для чтения).
* **close()** — закрывает дескриптор файла, освобождая ресурсы, связанные с файлом или каналом.
* **waitpid()** — приостанавливает выполнение родительского процесса до завершения дочернего, позволяя родителю дождаться завершения конкретного дочернего процесса и получить его статус.

В рамках лабораторной работы мы разработали две программы: **родительскую** и **дочернюю**, которые взаимодействуют между собой с использованием каналов и перенаправлением потоков. Целью программы было создать механизм, при котором дочерний процесс считывает команды из файла, производит вычисления и передает результаты родительскому процессу для отображения.

#### 

#### 

#### Родительский процесс

1. **Запрос имени файла**: Родительский процесс запрашивает у пользователя имя файла, в котором находятся команды для обработки.
2. **Создание канала**: Системный вызов pipe() используется для создания однонаправленного канала связи между процессами. Этот канал позволяет передавать данные от дочернего процесса к родительскому.
3. **Создание дочернего процесса**: Системный вызов fork() создаёт новый (дочерний) процесс. Родительский и дочерний процессы продолжают выполнять разные задачи.
4. **Перенаправление потоков для дочернего процесса**:
   * В дочернем процессе стандартный ввод перенаправляется из файла, выбранного пользователем, с помощью dup2().
   * Стандартный вывод перенаправляется в канал, так что результаты вычислений дочернего процесса можно будет прочитать родителем.
5. **Запуск дочернего процесса**: Дочерний процесс заменяет себя с помощью execl(), чтобы начать выполнение другой программы, child.
6. **Чтение и вывод данных**: Родительский процесс считывает результаты из канала и выводит их на экран.

#### Дочерний процесс

1. **Чтение данных из файла**: После перенаправления стандартного ввода дочерний процесс считывает команды из файла.
2. **Обработка команд**: Дочерний процесс считывает числа, суммирует их и записывает результат в стандартный вывод (перенаправленный на канал).
3. **Передача результатов родительскому процессу**: Итоговая сумма отправляется через канал, после чего дочерний процесс завершает выполнение.

### Принципы работы программы

1. **Перенаправление потоков**: Важно было перенаправить стандартный ввод и вывод дочернего процесса, чтобы он читал данные из файла, а результаты записывал в канал.
2. **Межпроцессное взаимодействие через каналы**: Использование канала pipe() позволило дочернему процессу передавать данные родительскому процессу. Это организовало одностороннюю связь, обеспечив безопасную передачу результатов вычислений.
3. **Использование fork() и exec() для создания и замены процессов**: fork() позволил создать параллельно выполняющийся дочерний процесс, а execl() заменил текущий код дочернего процесса новым, чтобы тот выполнял свою задачу.

### 

### Результат работы программы

1. Пользователь вводит имя файла с командами.
2. Родительский процесс создает канал и дочерний процесс.
3. Дочерний процесс читает числа из указанного файла, выполняет вычисления и передает результаты родителю.
4. Родительский процесс выводит результаты на экран.

Таким образом, программа продемонстрировала работу с системными вызовами для управления процессами, взаимодействие между ними и передачу данных через каналы. Это позволяет реализовать базовые функции IPC (межпроцессного взаимодействия) в UNIX-подобных системах.

**Код программы**

**parent.cpp**

#include <iostream>

#include <unistd.h>

#include <fcntl.h>

#include <sys/wait.h>

#include <string>

#include <cstdlib>

void create\_pipe(int\* pipe\_fd) {

if (pipe(pipe\_fd) == -1) {

perror("Pipe error!\n");

exit(EXIT\_FAILURE);

}

}

int main() {

std::string filename;

std::cout << "Введите имя файла: ";

std::getline(std::cin, filename);

int pipe1\_fd[2];

create\_pipe(pipe1\_fd);

pid\_t pid = fork();

if (pid == -1) {

perror("Fork error!\n");

exit(EXIT\_FAILURE);

} else if (pid == 0) {

close(pipe1\_fd[0]);

int file\_fd = open(filename.c\_str(), O\_RDONLY);

if (file\_fd == -1) {

perror("File open error!\n");

exit(EXIT\_FAILURE);

}

dup2(file\_fd, STDIN\_FILENO);

dup2(pipe1\_fd[1], STDOUT\_FILENO);

close(file\_fd);

close(pipe1\_fd[1]);

execl("./child", "./child", nullptr);

perror("execl error!\n");

exit(EXIT\_FAILURE);

} else {

close(pipe1\_fd[1]);

char buffer[256];

ssize\_t bytesRead;

while ((bytesRead = read(pipe1\_fd[0], buffer, sizeof(buffer) - 1)) > 0) {

buffer[bytesRead] = '\0';

std::cout << buffer;

}

close(pipe1\_fd[0]);

int status;

waitpid(pid, &status, 0);

}

return 0;

}

**child.cpp**

#include <iostream>

#include <sstream>

#include <string>

int main() {

std::string line;

while (std::getline(std::cin, line)) {

std::istringstream iss(line);

float number, sum = 0;

while (iss >> number) {

sum += number;

}

std::cout << "Сумма: " << sum << std::endl;

}

return 0;

}

**Протокол работы программы**

**Тестирование:**

➜ src git:(main) ./parent

Введите имя файла: data.txt

Сумма: 60.9

Сумма: 13

Сумма: 11

Сумма: 151

Сумма: 36.5

Сумма: 9.5

➜ src git:(main) cat < data.txt

10.5 20.3 30.1

5.5 3.2 4.3

1.1 2.2 3.3 4.4

100.25 50.75

7.7 8.8 9.9 10.10

7.7 8.8 9.9 10.10 -1.5 -25.5

**Dtruss:**

​​➜ src git:(main) ✗ sudo dtruss ./parent

dtrace: system integrity protection is on, some features will not be available

SYSCALL(args) = return

Введите имя файла: data.txt

Сумма: 60.9

Сумма: 13

Сумма: 11

Сумма: 151

Сумма: 36.5

Сумма: 9.5

**Вывод**

В результате лабораторной работы была реализована программа, где родительский и дочерний процессы взаимодействуют через канал: родитель запрашивает у пользователя файл с командами, дочерний процесс читает из него числа, вычисляет их сумму и передает результат родителю для вывода на экран. Программа продемонстрировала успешное применение системных вызовов для организации межпроцессного взаимодействия и перенаправления потоков ввода-вывода.

Проблемы возникли при корректном перенаправлении стандартного ввода и вывода дочернего процесса на файл и канал, а также при обработке ошибок, связанных с неверными входными данными. Было бы полезно глубже изучить механизм каналов и особенности работы с dup2 для более сложных сценариев перенаправления потоков.