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-213БВ-24

Студент: Кретов А.В.

Преподаватель: Бахарев В.Д. (ФИИТ)

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

Дата: 29.09.25

Москва, 2025

**Постановка задачи**

**Вариант 16.**

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.

Правило проверки: строка должна оканчиваться на «.» или «;»

**Общий метод и алгоритм решения**

* fork() – создание дочернего процесса. Родительский процесс вызывает fork(), создавая точную копию себя. Дальнейшее выполнение различается по возвращаемому значению: 0 – в дочернем процессе, PID ребёнка – в родительском процессе.
* pipe() – создание неименованного канала (inter-process communication). Используется два канала: pipe1: родитель → ребёнок (передача строк); pipe2: ребёнок → родитель (передача ошибок).
* dup2() – перенаправление стандартных потоков ввода/вывода. В дочернем процессе: dup2(pipe1[0], STDIN\_FILENO) – чтение из pipe1 вместо стандартного ввода; dup2(pipe2[1], STDERR\_FILENO) – запись ошибок в pipe2 вместо stderr; dup2(fileFd, STDOUT\_FILENO) – перенаправление вывода в файл.
* execl() – замена образа процесса на программу child. Передаёт имя файла как аргумент. Если вызов fails, ребёнок завершается с ошибкой.
* open() – открытие файла в дочернем процессе. Флаги: O\_WRONLY | O\_CREAT | O\_TRUNC (запись, создание, очистка файла). Права доступа: 0644 (владелец: чтение/запись, остальные: чтение).
* read() / write() – безопасное чтение/запись с обработкой прерываний (EINTR). Реализованы в функциях SafeRead() и SafeWrite().
* close() – закрытие неиспользуемых файловых дескрипторов. Например, после dup2() исходные дескрипторы каналов закрываются.
* waitpid() – ожидание завершения дочернего процесса. Родитель блокируется до завершения ребёнка.

В рамках лабораторной работы была реализована система взаимодействия процессов через неименованные каналы с использованием системных вызовов UNIX. Основой взаимодействия стала функция fork(), создающая дочерний процесс, и два канала pipe(), обеспечивающие двустороннюю связь между процессами.

Родительский процесс начинает работу с функции SafeWrite(), выводящей приглашение к вводу имени файла, затем считывает данные через SafeRead() с обработкой прерываний. После создания каналов с помощью pipe() и порождения дочернего процесса через fork(), программа использует dup2() для перенаправления потоков - дочерний процесс перенаправляет свой стандартный ввод на чтение из первого канала, стандартный вывод на запись в файл (открытый через open() с флагами O\_WRONLY | O\_CREAT | O\_TRUNC), а стандартный вывод ошибок - на запись во второй канал.

Обработка данных построена на алгоритме буферизации: родительский процесс с помощью SafeRead() получает строки произвольной длины, разбивает их по символу новой строки и передает через первый канал с помощью SafeWrite(). Дочерний процесс анализирует каждую строку, проверяя окончание на '.' или ';', и в зависимости от результата либо записывает строку в файл через перенаправленный стандартный вывод, либо отправляет сообщение об ошибке через SafeWrite() в перенаправленный stderr. Завершение работы синхронизируется с помощью waitpid(), гарантируя корректное завершение дочернего процесса.

**Код программы**

**parent.cpp**

#include <string>

#include <vector>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <errno.h>

#include <string.h>

#include <stdlib.h>

const char PROMPT[] = "Введите имя файла: ";

const char E\_READ\_FILENAME[] = "Ошибка чтения имени файла\n";

const char E\_EMPTY\_FILENAME[] = "Пустое имя файла\n";

const char E\_PIPE1[] = "pipe1 error\n";

const char E\_PIPE2[] = "pipe2 error\n";

const char E\_FORK[] = "fork error\n";

const char E\_DUP2\_STDIN[] = "child(before exec): dup2 stdin failed\n";

const char E\_DUP2\_STDERR[] = "child(before exec): dup2 stderr failed\n";

const char E\_EXECL[] = "execl failed\n";

static ssize\_t SafeWrite(int fd, const char \*buf, size\_t count) {

size\_t written = 0;

while (written < count) {

ssize\_t w = write(fd, buf + written, count - written);

if (w < 0) {

if (errno == EINTR) continue;

return -1;

}

written += (size\_t)w;

}

return (ssize\_t)written;

}

static ssize\_t SafeRead(int fd, char \*buf, size\_t count) {

while (1) {

ssize\_t r = read(fd, buf, count);

if (r < 0) {

if (errno == EINTR) continue;

}

return r;

}

}

int main() {

// Считываем имя файла

if (SafeWrite(STDOUT\_FILENO, PROMPT, sizeof(PROMPT) - 1) < 0) {

return 1;

}

char tmpbuf[512];

ssize\_t rn = SafeRead(STDIN\_FILENO, tmpbuf, sizeof(tmpbuf));

if (rn <= 0) {

SafeWrite(STDOUT\_FILENO, E\_READ\_FILENAME, sizeof(E\_READ\_FILENAME) - 1);

return 1;

}

size\_t fnLen = (size\_t)rn;

while (fnLen > 0 && (tmpbuf[fnLen - 1] == '\n' || tmpbuf[fnLen - 1] == '\r')) fnLen--;

std::string filename(tmpbuf, tmpbuf + fnLen);

if (filename.empty()) {

SafeWrite(STDOUT\_FILENO, E\_EMPTY\_FILENAME, sizeof(E\_EMPTY\_FILENAME) - 1);

return 1;

}

int pipe1[2]; // parent -> child

int pipe2[2]; // child -> parent (errors)

if (pipe(pipe1) == -1) {

SafeWrite(STDOUT\_FILENO, E\_PIPE1, sizeof(E\_PIPE1) - 1);

return 1;

}

if (pipe(pipe2) == -1) {

SafeWrite(STDOUT\_FILENO, E\_PIPE2, sizeof(E\_PIPE2) - 1);

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

return 1;

}

pid\_t pid = fork();

if (pid == -1) {

SafeWrite(STDOUT\_FILENO, E\_FORK, sizeof(E\_FORK) - 1);

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

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

return 1;

}

if (pid == 0) {

// === дочерний (до exec) ===

close(pipe1[1]);

close(pipe2[0]);

if (dup2(pipe1[0], STDIN\_FILENO) == -1) {

SafeWrite(STDERR\_FILENO, E\_DUP2\_STDIN, sizeof(E\_DUP2\_STDIN) - 1);

\_exit(1);

}

if (dup2(pipe2[1], STDERR\_FILENO) == -1) {

SafeWrite(STDERR\_FILENO, E\_DUP2\_STDERR, sizeof(E\_DUP2\_STDERR) - 1);

\_exit(1);

}

close(pipe1[0]);

close(pipe2[1]);

execl("./child", "child", filename.c\_str(), (char\*)nullptr);

SafeWrite(STDERR\_FILENO, E\_EXECL, sizeof(E\_EXECL) - 1);

\_exit(1);

} else {

// === родитель ===

close(pipe1[0]);

close(pipe2[1]);

std::vector<char> buf(4096);

std::string acc;

ssize\_t r;

while ((r = SafeRead(STDIN\_FILENO, buf.data(), buf.size())) > 0) {

acc.append(buf.data(), buf.data() + r);

size\_t pos;

while ((pos = acc.find('\n')) != std::string::npos) {

std::string line = acc.substr(0, pos + 1);

acc.erase(0, pos + 1);

SafeWrite(pipe1[1], line.c\_str(), line.size());

}

}

if (!acc.empty()) {

acc.push\_back('\n');

SafeWrite(pipe1[1], acc.c\_str(), acc.size());

}

close(pipe1[1]);

while ((r = SafeRead(pipe2[0], buf.data(), buf.size())) > 0) {

SafeWrite(STDOUT\_FILENO, buf.data(), (size\_t)r);

}

close(pipe2[0]);

int status = 0;

waitpid(pid, &status, 0);

}

return 0;

}

**child.cpp**

#include <string>

#include <vector>

#include <unistd.h>

#include <fcntl.h>

#include <errno.h>

#include <string.h>

#include <stdlib.h>

const int FILE\_PERMISSIONS = 0644;

const char E\_MISSING\_FILENAME[] = "child: missing filename\n";

const char E\_OPEN\_FAILED[] = "child: open failed: ";

const char E\_DUP2\_FAILED[] = "child: dup2 file->stdout failed: ";

const char E\_RULE\_ERROR[] = "Error: line must end with '.' or ';'\n";

static ssize\_t SafeWrite(int fd, const char \*buf, size\_t count) {

size\_t written = 0;

while (written < count) {

ssize\_t w = write(fd, buf + written, count - written);

if (w < 0) {

if (errno == EINTR) continue;

return -1;

}

written += (size\_t)w;

}

return (ssize\_t)written;

}

static ssize\_t SafeRead(int fd, char \*buf, size\_t count) {

while (1) {

ssize\_t r = read(fd, buf, count);

if (r < 0) {

if (errno == EINTR) continue;

}

return r;

}

}

int main(int argc, char \*argv[]) {

if (argc < 2) {

SafeWrite(STDERR\_FILENO, E\_MISSING\_FILENAME, sizeof(E\_MISSING\_FILENAME) - 1);

return 1;

}

const char \*filename = argv[1];

int fileFd = open(filename, O\_WRONLY | O\_CREAT | O\_TRUNC, FILE\_PERMISSIONS);

if (fileFd < 0) {

std::string err = E\_OPEN\_FAILED;

err += strerror(errno);

err.push\_back('\n');

SafeWrite(STDERR\_FILENO, err.c\_str(), err.size());

return 1;

}

if (dup2(fileFd, STDOUT\_FILENO) == -1) {

std::string err = E\_DUP2\_FAILED;

err += strerror(errno);

err.push\_back('\n');

SafeWrite(STDERR\_FILENO, err.c\_str(), err.size());

close(fileFd);

return 1;

}

close(fileFd);

std::vector<char> buf(4096);

std::string acc;

ssize\_t r;

while ((r = SafeRead(STDIN\_FILENO, buf.data(), buf.size())) > 0) {

acc.append(buf.data(), buf.data() + r);

size\_t pos;

while ((pos = acc.find('\n')) != std::string::npos) {

std::string line = acc.substr(0, pos);

acc.erase(0, pos + 1);

if (!line.empty() && (line.back() == '.' || line.back() == ';')) {

line.push\_back('\n');

SafeWrite(STDOUT\_FILENO, line.c\_str(), line.size());

} else {

SafeWrite(STDERR\_FILENO, E\_RULE\_ERROR, sizeof(E\_RULE\_ERROR) - 1);

}

}

}

if (!acc.empty()) {

if (!acc.empty() && (acc.back() == '.' || acc.back() == ';')) {

acc.push\_back('\n');

SafeWrite(STDOUT\_FILENO, acc.c\_str(), acc.size());

} else {

SafeWrite(STDERR\_FILENO, E\_RULE\_ERROR, sizeof(E\_RULE\_ERROR) - 1);

}

}

return 0;

}

**Протокол работы программы**

user@WIN-LVQ040U9NHA:/mnt/d/!labs/OS/lab1$ g++ -o child child.cpp

user@WIN-LVQ040U9NHA:/mnt/d/!labs/OS/lab1$ g++ -o parent parent.cpp

user@WIN-LVQ040U9NHA:/mnt/d/!labs/OS/lab1$ ./parent

Введите имя файла: a1

asd;

caas;.;

asz.

adf;j.

All heil the string!

..

Error: line must end with '.' or ';'

user@WIN-LVQ040U9NHA:/mnt/d/!labs/OS/lab1$ ./parent

Введите имя файла: a2

All heil the dot!.

.

dots...

user@WIN-LVQ040U9NHA:/mnt/d/!labs/OS/lab1$ ./parent

Введите имя файла: a3

Let's all love ;

Praise the ;

;;

;;;;;;;.....;;;;

All my students use;

;\_;

user@WIN-LVQ040U9NHA:/mnt/d/!labs/OS/lab1$ ./parent

Введите имя файла: a4

She opened up her eyes and thought;

Oh what a morning.

It`s not a day for dot;

It`s a day for catching errors

Just laying in the log and having fun

Error: line must end with '.' or ';'

Error: line must end with '.' or ';'

user@WIN-LVQ040U9NHA:/mnt/d/!labs/OS/lab1$ ./parent

Введите имя файла: a5

.;

user@WIN-LVQ040U9NHA:/mnt/d/!labs/OS/lab1$ ./parent

Введите имя файла: a6

user@WIN-LVQ040U9NHA:/mnt/d/!labs/OS/lab1$ cat a1

asd;

caas;.;

asz.

adf;j.

..

user@WIN-LVQ040U9NHA:/mnt/d/!labs/OS/lab1$ cat a2

All heil the dot!.

.

dots...

user@WIN-LVQ040U9NHA:/mnt/d/!labs/OS/lab1$ cat a3

Let's all love ;

Praise the ;

;;

;;;;;;;.....;;;;

All my students use;

;\_;

user@WIN-LVQ040U9NHA:/mnt/d/!labs/OS/lab1$ cat a4

She opened up her eyes and thought;

Oh what a morning.

It`s not a day for dot;

user@WIN-LVQ040U9NHA:/mnt/d/!labs/OS/lab1$ cat a5

.;

user@WIN-LVQ040U9NHA:/mnt/d/!labs/OS/lab1$ cat a6

**Вывод**

В ходе выполнения работы удалось реализовать взаимодействие между родительским и дочерним процессами с использованием только системных вызовов. Родитель передаёт строки через pipe, дочерний проверяет их по правилу и выводит либо в файл (валидные строки), либо возвращает ошибки родителю.

Основная сложность заключалась в правильной настройке перенаправления потоков и учёте ограничений: использование только std::string и std::vector из C++ STL, а также отказ от stdio.h и iostream.

В дальнейшем хотелось бы иметь больше наглядных примеров по организации пайпов и exec в условиях строгих ограничений, так как именно эта часть заняла больше всего времени при отладке.