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»

Выполнил: Д. Н. Дружинин
Группа: М8О-207БВ-24
Преподаватель: Е. С. Миронов

Москва, 2025

Условие

Цель работы: Изучение механизмов создания и взаимодействия процессов в операционных системах. Освоение принципов работы с системными вызовами и разработка кроссплатформенного приложения с использованием обёрток над функциями операционной системы.

Задание: Разработать программу, состоящую из двух взаимодействующих процессов — родительского и дочернего. Родительский процесс считывает данные из файла, выполняет арифметические операции (деление чисел) с проверкой деления на ноль и передаёт результаты дочернему процессу для вывода. Запрещено прямое использование системных вызовов — необходимо реализовать функции-обёртки для открытия, чтения и закрытия файлов, создания процессов и вывода информации. Программа должна корректно компилироваться и работать как в среде Windows, так и в Linux.

Вариант: 8

Метод решения

Программа состоит из двух взаимодействующих процессов: родительского (*parent*) и дочернего (*child*). Для обмена данными между ними используется канал (pipe), организованный с помощью функции-обёртки над системными вызовами ОС.

Родительский процесс открывает файл `input.txt`, содержащий команды вида «число число число». Каждая строка файла представляет последовательность чисел, над которыми нужно выполнить деление первого числа на все последующие. Родительский процесс считывает строку, передаёт её дочернему процессу через канал и ожидает результат.

Дочерний процесс получает данные, выполняет арифметические операции деления и возвращает результат обратно в стандартный поток вывода. Если в процессе вычислений встречается деление на ноль, дочерний процесс сообщает об этом и завершает работу, что влечёт завершение и родительского процесса.

Для обеспечения кроссплатформенности системные вызовы (`fopen`, `popen`, `fgets`, `exit` и др.) были скрыты за единым интерфейсом — функциями-обёртками, реализованными в модуле `os_wrapper`. В зависимости от операционной системы, используется реализация для POSIX (`os_wrapper_posix.cpp`) или для Windows (`os_wrapper_win.cpp`).

Программа была разработана на языке C++ с ограничениями языка C, чтобы обеспечить совместимость с системными вызовами и сохранить переносимость.

Описание программы

Проект состоит из следующих файлов:

- **parent.cpp** — реализует родительский процесс. Открывает файл с входными данными, создаёт дочерний процесс и передаёт ему строки для обработки. Использует функции `os_open_file`, `os_read_line`, `os_open_process`, `os_close_file`, `os_exit`.
- **child.cpp** — реализует дочерний процесс. Получает строку от родительского процесса, разбивает её на числа, выполняет деление первого числа на последующие. Проверяет деление на ноль и выводит результат в стандартный поток.
- **os_wrapper.h / os_wrapper_posix.cpp / os_wrapper_win.cpp** — модуль-обёртка, скрывающий различия между системными вызовами Windows и POSIX. Опреде-

ляет универсальные функции работы с файлами и процессами (`os_open_file`, `os_read_line`, `os_open_process`, `os_close_process` и др.).

- **common.h** — заголовочный файл с константами (`MAX_LINE`, `MAX_NUMS`) и типами данных, используемыми обеими частями программы.
- **CMakeLists.txt** — файл сборки, описывающий компиляцию под разные операционные системы и создание двух исполняемых файлов (`parent`, `child`).

В ходе работы программы используются системные вызовы: `fopen`, `fclose`, `fgets`, `popen`, `pclose`, `printf`, `exit`. Эти вызовы инкапсулированы в соответствующих функциях-обёртках для обеспечения переносимости и удобства сопровождения.

Результаты

В результате выполнения лабораторной работы была разработана кроссплатформенная программа, состоящая из двух взаимодействующих процессов — родительского и дочернего. Между процессами реализован обмен данными через канал (*pipe*), что позволило обеспечить асинхронную передачу строковых команд без использования внешних файлов.

Родительский процесс читает входные данные из файла `input.txt` и запускает дочерний процесс с помощью функции-обёртки `os_open_process`. Каждая строка файла содержит набор чисел, над которыми выполняется операция последовательного деления. Результаты вычислений выводятся в стандартный поток:

```
Result: 5.0000
Result: 1.5000
Division by zero detected. Terminating.
Parent terminated.
```

Программа корректно обрабатывает деление на ноль — при обнаружении такого случая дочерний процесс выводит сообщение об ошибке и завершает работу, после чего завершается и родительский процесс.

Особенностью реализации является использование модуля `os_wrapper`, который инкапсулирует различия между системными вызовами POSIX и Windows. Это позволило обеспечить переносимость кода: программа компилируется и функционирует как в среде Linux (через `popen/pclose`), так и в Windows (через `_popen/_pclose`).

Выводы

В ходе выполнения лабораторной работы были изучены механизмы взаимодействия процессов в операционной системе и способы организации обмена данными через канал (pipe). Разработана кроссплатформенная программа, состоящая из родительского и дочернего процессов, осуществляющих взаимодействие с помощью стандартных потоков ввода-вывода.

В процессе работы были реализованы функции-обёртки над системными вызовами, позволившие скрыть различия между ОС семейства POSIX и Windows. Это обеспечило переносимость программы и её корректную работу в разных средах.

Были получены практические навыки использования функций для работы с файлами, создания процессов и организации межпроцессного взаимодействия. Кроме того, был

закреплён навык структурирования программы на отдельные модули и использования системы сборки CMake.

Реализованная программа успешно решает поставленную задачу, корректно обрабатывает ошибочные ситуации (например, деление на ноль) и демонстрирует принципы работы процессов и каналов связи в операционных системах.

Исходная программа

В данном разделе приведён полный исходный код программы, реализующей взаимодействие родительского и дочернего процессов с использованием обёрток над системными вызовами. Код написан на языке C++ с ограничениями стандарта C для обеспечения кроссплатформенности.

Файл common.h

```
#pragma once
#define MAX_LINE 256
#define MAX_NUMS 64
```

Листинг 1: Файл common.h — общие константы

Файл os_wrapper.h

```
#pragma once
#include <cstdio>
#include <cstddef>

FILE* os_open_file(const char* name, const char* mode);
void os_close_file(FILE* f);
bool os_read_line(FILE* f, char* buf, size_t size);

FILE* os_open_process(const char* cmd, const char* mode);
void os_close_process(FILE* p);

void os_print(const char* msg);
void os_exit(int code);
```

Листинг 2: Файл os_wrapper.h — интерфейс обёрток системных функций

Файл os_wrapper_posix.cpp

```
#include "os_wrapper.h"
#include <cstdlib>

FILE* os_open_file(const char* name, const char* mode) {
  return fopen(name, mode);
}

void os_close_file(FILE* f) {
  if (f) fclose(f);
}

bool os_read_line(FILE* f, char* buf, size_t size) {
  return fgets(buf, size, f) != nullptr;
}

FILE* os_open_process(const char* cmd, const char* mode) {
```

```
  return popen(cmd, mode);
}

void os_close_process(FILE* p) {
  if (p) pclose(p);
}

void os_print(const char* msg) {
  printf("%s", msg);
}

void os_exit(int code) {
  exit(code);
}
```

Листинг 3: Файл os_wrapper_posix.cpp — реализация для POSIX систем

Файл os_wrapper_win.cpp

```
#include "os_wrapper.h"
#include <cstdlib>

FILE* os_open_file(const char* name, const char* mode) {
  return fopen(name, mode);
}

void os_close_file(FILE* f) {
  if (f) fclose(f);
}

bool os_read_line(FILE* f, char* buf, size_t size) {
  return fgets(buf, size, f) != nullptr;
}

FILE* os_open_process(const char* cmd, const char* mode) {
  return _popen(cmd, mode);
}

void os_close_process(FILE* p) {
  if (p) _pclose(p);
}

void os_print(const char* msg) {
  printf("%s", msg);
}

void os_exit(int code) {
  exit(code);
}
```

Листинг 4: Файл os_wrapper_win.cpp — реализация для Windows систем

Файл parent.cpp

```
#include "os_wrapper.h"
#include "common.h"
#include <cstdlib>
#include <cstring>

int main() {
  FILE* f = os_open_file("input.txt", "r");
  if (!f) {
    os_print("Error: cannot open input.txt\n");
    os_exit(1);
  }

  char line[MAX_LINE];
  while (os_read_line(f, line, sizeof(line))) {
    double nums[MAX_NUMS];
    int count = 0;

    char* token = strtok(line, " \t\n");
    while (token && count < MAX_NUMS) {
      nums[count++] = atof(token);
      token = strtok(NULL, " \t\n");
    }
    if (count == 0) continue;

    double result = nums[0];
    bool div_by_zero = false;

    for (int i = 1; i < count; ++i) {
      if (nums[i] == 0.0) {
        div_by_zero = true;
        break;
      }
      result /= nums[i];
    }

    if (div_by_zero) {
      os_print("Division by zero detected. Terminating.\n");
      os_close_file(f);
      os_exit(0);
    }

    char out[128];
    sprintf(out, "Result: %.4f\n", result);
    os_print(out);
  }

  os_close_file(f);
  os_exit(0);
}
```

Листинг 5: Файл parent.cpp — родительский процесс

Файл child.cpp

```
#include "os_wrapper.h"
#include "common.h"
#include <cstdio>
#include <cstdlib>

int main() {
  char buf[MAX_LINE];
  while (fgets(buf, sizeof(buf), stdin)) {
    double a, b;
    if (sscanf(buf, "%lf %lf", &a, &b) == 2) {
      if (b == 0.0) {
        printf("Division by zero\n");
        break;
      }
      double result = a / b;
      printf("Result: %.4f\n", result);
    }
  }
  return 0;
}
```

Листинг 6: Файл child.cpp — дочерний процесс

Файл CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)
project(lab1_var8)

set(CMAKE_CXX_STANDARD 17)

add_library(os_wrapper
src/os_wrapper_posix.cpp
src/os_wrapper_win.cpp
)

add_executable(parent src/parent.cpp)
add_executable(child src/child.cpp)

target_link_libraries(parent os_wrapper)
target_link_libraries(child os_wrapper)
```

Листинг 7: CMakeLists.txt — сборочная конфигурация проекта

Фрагмент вывода утилиты `strace`

```
execve("./parent", ["./parent"], 0x7ffde264e100 /* 37 vars */) = 0
pipe2([3, 4], O_CLOEXEC) = 0
clone3({flags=CLONE_VM|CLONE_VFORK|CLONE_CLEAR_SIGHAND, ...}, 88) = 13067
read(3, "Result: 2\nResult: 5\n...", 4096) = 122
write(1, "Result: 2\n", 10) = 10
```

```
write(1, "Result: 5\n", 10) = 10
write(1, "Result: 3\n", 10) = 10
write(1, "Division by zero detected. Terminating.\n", 40) = 40
close(3) = 0
wait4(13067, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 13067
exit_group(0) = ?
+++ exited with 0 +++
```

Листинг 8: Вывод системных вызовов для программы parent