

**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»**

**Выполнила: В. А. Гузова
Группа: М8О-207БВ-24
Преподаватель: Е. С. Миронов**

Москва, 2025

Условие

Цель работы: Приобретение практических навыков в:

Управлении процессами в ОС.

Обеспечении обмена данных между процессами посредством каналов.

Создании кроссплатформенных консольных приложений.

Задание:

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.

Вариант: 7

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

Метод решения

Parent запрашивает у пользователя имя файла, создает `pipe`, передает имя полученного файла `child` (ребенку). `Child` в `stdin` читает числа из файла, складывает их и `stdout` направляет в `pipe` возвращает результат. Parent выводит ответ в `stdout`.

Кроссплатформенность достигается разной реализацией функций, описанных в файле `os.h`, в файлах `os_unix.cpp`, `os_win.cpp`

Описание программы

Разделение по файлам, описание основных типов данных и функций. Обязательно написать используемые системные вызовы.

`parent.cpp` — точка входа в программу. Создает дочерний процесс, передает ему файл с цифрами и через канал выводит итоговый результат сложения чисел.

`child.cpp` — исполняемый файл дочернего процесса. Читает из файла цифры, складывает их и передает результат в канал.

`os.h` — объявление функций-обёрток над системными вызовами ОС для управления процессами и каналами.

`os_unix.cpp` — реализация для Unix-подобных ОС.

`os_win.cpp` — реализация для Windows.

Основные функции:

- `int CreateChildProcess(const StartProcess& args);` — создаёт дочерний процесс. Заменяет `fork()` + `dup2()` + `close()` + `execl()`
- `bool CreatePipe(PipeHandle& readpipe, PipeHandle& writepipe);` — создаёт канал для передачи данных. Возвращает дескрипторы для записи и чтения из канала. Заменяет `pipe()`.

- `int ReadPipe(PipeHandle pipe, void* buf, int count)` — читает из канала определенное количество байт в буфер. Заменяет `read()`
- `void ClosePipe(PipeHandle pipe);` — закрывает конец канала. Заменяет `close()`
- `Exit(int code);` — завершает текущий процесс. Обёртка над `_exit()`.

Архитектура взаимодействия:

Родительский процесс (`parent.cpp`) считывает строку от пользователя с названием файла, передает его содержимое в `stdin` дочернего процесса (`child1`), который выполняет сложение чисел и через канал `pipe` возвращает родительскому процессу результат сложения. Родитель читает результат из `pipe` и выводит его на экран.

Результаты

Программа получает на вход название файла, создает дочерий процесс, который из файла считывает числа в поток ввода и складывает их. Результатом является число в выходном потоке данных. Если не удалось создать канал для передачи данных между процессами или дочерний процесс завершился, то программа безопасно прекращает свою работу.

Выводы

В ходе выполнения лабораторной работы были приобретены навыки в управлении процессами в ОС, обеспечение обмена данных между процессами посредством каналов.

Была составлена и отлажена программа на языке C++, осуществляющая работу с процессами и взаимодействие между ними в операционной системе с ядром Unix или Windows. В результате работы программа (основной процесс) создает дочерний процесс.

Взаимодействие между процессами осуществляется через каналы.

Обработаны системные ошибки, которые могут возникнуть в результате работы.

Есть возможность поддержки кроссплатформенности за счет изменений системных вызовов в файле

Исходная программа

```
1  #pragma once
2
3  #include <string>
4  #include <stdint>
5
6  namespace os {
7  using PipeHandle = intptr_t;
8  struct StartProcess {
9      std::string path;
10     std::string filename;
11     PipeHandle pipe = -1;
12 };
13
14 int CreateChildProcess(const StartProcess& args);
15 bool CreatePipe(PipeHandle& readpipe, PipeHandle& writepipe);
16 int ReadPipe(PipeHandle pipe, void* buf, int count);
17 void ClosePipe(PipeHandle pipe);
18 void Exit(int code);
19 }
```

Листинг 1: include/os.h

```
1  #include "os.h"
2  #include <unistd.h>
3  #include <sys/types.h>
4  #include <stdio>
5
6  namespace os {
7
8  int CreateChildProcess(const StartProcess& info) {
9      pid_t pid = fork();
10     if (pid == -1) {
11         perror("Error fork");
12         return -1;
13     }
14
15     if (pid == 0) {
16         if (info.pipe != -1) {
17             if (dup2(info.pipe, STDOUT_FILENO) == -1) {
18                 perror("Error dup2");
19                 os::Exit(1);
20             }
21             close(info.pipe);
22         }
23         execl(info.path.c_str(), info.path.c_str(), info.filename.c_str(), nullptr);
24         perror("Error execl");
25         os::Exit(1);
26     } else {
27         if (info.pipe != -1) {
28             close(info.pipe);
29         }
30         return pid;
31     }
32     return -1;
33 }
```

```

34
35 bool CreatePipe(PipeHandle& readpipe, PipeHandle& writepipe) {
36     int pipefd[2];
37     if (pipe(pipefd) != 0) {
38         perror("Error pipe");
39         return false;
40     }
41     readpipe = pipefd[0];
42     writepipe = pipefd[1];
43     return true;
44 }
45
46 int ReadPipe(PipeHandle pipe, void* buf, int count) {
47     if (pipe == -1 || buf == nullptr || count <= 0) {
48         return -1;
49     }
50     int bytes = read(pipe, buf, count);
51     if (bytes == -1) {
52         perror("Error read");
53     }
54     return bytes;
55 }
56
57 void ClosePipe(PipeHandle pipe) {
58     if (pipe != -1) {
59         close(pipe);
60     }
61 }
62
63 void Exit(int code) {
64     _exit(code);
65 }
66
67 }

```

Листинг 2: src/os_unix.cpp

```

1  #include "os.h"
2  #include <windows.h>
3  #include <string>
4
5  namespace os {
6
7  int CreateChildProcess(const StartProcess& args) {
8      std::string cmd = args.path + " " + args.filename;
9
10     STARTUPINFOA si{};
11     PROCESS_INFORMATION pi{};
12     si.cb = sizeof(si);
13     si.dwFlags = STARTF_USESTDHANDLES;
14     si.hStdInput = GetStdHandle(STD_INPUT_HANDLE);
15
16     if (args.pipe != -1) {
17         si.hStdOutput = static_cast<HANDLE>(args.pipe);
18         si.hStdError = si.hStdOutput;
19     } else {
20         si.hStdOutput = GetStdHandle(STD_OUTPUT_HANDLE);

```

```

21     si.hStdError = GetStdHandle(STD_ERROR_HANDLE);
22 }
23 BOOL success = CreateProcessA(nullptr, const_cast<char*>(cmd.c_str()), nullptr,
24     nullptr, TRUE, 0, nullptr, nullptr, &si, &pi
25 );
26 if (!success) {
27     return -1;
28 }
29 int pid = static_cast<int>(pi.dwProcessId);
30
31 CloseHandle(pi.hThread);
32 CloseHandle(pi.hProcess);
33
34 return pid;
35 }
36
37 bool CreatePipe(PipeHandle& readpipe, PipeHandle& writepipe) {
38     SECURITY_ATTRIBUTES sa{};
39     sa.nLength = sizeof(sa);
40     sa.bInheritHandle = TRUE;
41     HANDLE hRead, hWrite;
42
43     if (!CreatePipe(&hRead, &hWrite, &sa, 0)) {
44         return false;
45     }
46     SetHandleInformation(hRead, HANDLE_FLAG_INHERIT, 0);
47
48     readpipe = static_cast<PipeHandle>(static_cast<intptr_t>(hRead));
49     writepipe = static_cast<PipeHandle>(static_cast<intptr_t>(hWrite));
50     return true;
51 }
52
53 int ReadPipe(PipeHandle pipe, void* buf, int count) {
54     if (pipe == -1) {
55         return -1;
56     }
57     HANDLE h = static_cast<HANDLE>(pipe);
58     DWORD bytesRead = 0;
59
60     if (!ReadFile(h, buf, static_cast<DWORD>(count), &bytesRead, nullptr)) {
61         return -1;
62     }
63     return static_cast<int>(bytesRead);
64 }
65
66 void ClosePipe(PipeHandle pipe) {
67     if (pipe != -1) {
68         HANDLE h = static_cast<HANDLE>(pipe);
69         CloseHandle(h);
70     }
71 }
72
73 void Exit(int code) {
74     ExitProcess(static_cast<UINT>(code));
75 }

```

Листинг 3: src/os_win.cpp

```

1  #include <iostream>
2  #include <string>
3  #include "os.h"
4
5  int main() {
6      std::cout << "Enter filename: ";
7      std::string filename;
8      std::cin >> filename;
9
10     os::PipeHandle readpipe;
11     os::PipeHandle writepipe;
12
13     if (!os::CreatePipe(readpipe, writepipe)) {
14         std::cerr << "Error create pipe" << std::endl;
15         return 1;
16     }
17
18     os::StartProcess psi;
19     psi.path = "./child";
20     psi.filename = filename;
21     psi.pipe = writepipe;
22
23     int pid = os::CreateChildProcess(psi);
24     if (pid == -1) {
25         std::cerr << "Error start child process" << std::endl;
26         return 1;
27     }
28
29     os::ClosePipe(writepipe);
30     writepipe = -1;
31
32     char buffer[256];
33     int bytes = os::ReadPipe(readpipe, buffer, sizeof(buffer) - 1);
34     if (bytes > 0) {
35         buffer[bytes] = '\0';
36         std::cout << "Answer: " << buffer << std::endl;
37     } else {
38         std::cerr << "Error";
39     }
40
41     os::ClosePipe(readpipe);
42     return 0;
43 }

```

Листинг 4: src/parent.cpp

```

1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  #include <sstream>
5  #include "os.h"

```

```

6
7 float SumFile(const std::string& filename) {
8     std::ifstream file(filename);
9     if (!file) {
10         return 0.0f;
11     }
12     float sum = 0.0f;
13     std::string line;
14     while (std::getline(file, line)) {
15         std::istringstream iss(line);
16         float num;
17         while (iss >> num) {
18             sum += num;
19         }
20     }
21     return sum;
22 }
23
24 int main(int argc, char* argv[]) {
25     if (argc < 2) {
26         os::Exit(1);
27     }
28     float answer = SumFile(argv[1]);
29     std::cout << answer << std::endl;
30     return 0;
31 }

```

Листинг 5: src/child.cpp

Strace

```

execve("./parent", [".parent"], 0x7ffe5a59fbb0 /* 27 vars */) = 0
brk(NULL)                               = 0x5c99716f7000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd5d856b00) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
↳ 0x76f66a93e000
access("/etc/ld.so.preload", R_OK)       = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=28532, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 28532, PROT_READ, MAP_PRIVATE, 3, 0) = 0x76f66a937000
close(3)                                 = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libstdc++.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0",
↳ 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=2260296, ...}, AT_EMPTY_PATH) =
↳ 0
mmap(NULL, 2275520, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x76f66a600000
mprotect(0x76f66a69a000, 1576960, PROT_NONE) = 0
mmap(0x76f66a69a000, 1118208, PROT_READ|PROT_EXEC,
↳ MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x9a000) = 0x76f66a69a000
mmap(0x76f66a7ab000, 454656, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
↳ 0x1ab000) = 0x76f66a7ab000
mmap(0x76f66a81b000, 57344, PROT_READ|PROT_WRITE,
↳ MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x21a000) = 0x76f66a81b000

```

```

mmap(0x76f66a829000, 10432, PROT_READ|PROT_WRITE,
↳ MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x76f66a829000
close(3) = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libgcc_s.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"...,
↳ 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=125488, ...}, AT_EMPTY_PATH) =
↳ 0
mmap(NULL, 127720, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x76f66a917000
mmap(0x76f66a91a000, 94208, PROT_READ|PROT_EXEC,
↳ MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x3000) = 0x76f66a91a000
mmap(0x76f66a931000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
↳ 0x1a000) = 0x76f66a931000
mmap(0x76f66a935000, 8192, PROT_READ|PROT_WRITE,
↳ MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1d000) = 0x76f66a935000
close(3) = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0"...,
↳ 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@ \0\0\0\0\0\0\0\0@ \0\0\0\0\0\0\0\0@ \0\0\0\0\0\0\0"...,
↳ 784, 64) = 784
pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0"...,
↳ 48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\325\31p\226\367\t\200\30)\261\30\25
↳ 7\33\1366c"..., 68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) =
↳ 0
pread64(3, "\6\0\0\0\4\0\0\0@ \0\0\0\0\0\0\0\0@ \0\0\0\0\0\0\0\0@ \0\0\0\0\0\0\0"...,
↳ 784, 64) = 784
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x76f66a200000
mprotect(0x76f66a228000, 2023424, PROT_NONE) = 0
mmap(0x76f66a228000, 1658880, PROT_READ|PROT_EXEC,
↳ MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x76f66a228000
mmap(0x76f66a3bd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
↳ 0x1bd000) = 0x76f66a3bd000
mmap(0x76f66a416000, 24576, PROT_READ|PROT_WRITE,
↳ MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x76f66a416000
mmap(0x76f66a41c000, 52816, PROT_READ|PROT_WRITE,
↳ MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x76f66a41c000
close(3) = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"...,
↳ 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=940560, ...}, AT_EMPTY_PATH) =
↳ 0
mmap(NULL, 942344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x76f66a830000
mmap(0x76f66a83e000, 507904, PROT_READ|PROT_EXEC,
↳ MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe000) = 0x76f66a83e000
mmap(0x76f66a8ba000, 372736, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
↳ 0x8a000) = 0x76f66a8ba000
mmap(0x76f66a915000, 8192, PROT_READ|PROT_WRITE,
↳ MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe4000) = 0x76f66a915000

```

```

close(3) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
↳ 0x76f66a82e000
arch_prctl(ARCH_SET_FS, 0x76f66a82f3c0) = 0
set_tid_address(0x76f66a82f690) = 1532
set_robust_list(0x76f66a82f6a0, 24) = 0
rseq(0x76f66a82fd60, 0x20, 0, 0x53053053) = 0
mprotect(0x76f66a416000, 16384, PROT_READ) = 0
mprotect(0x76f66a915000, 4096, PROT_READ) = 0
mprotect(0x76f66a935000, 4096, PROT_READ) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
↳ 0x76f66a82c000
mprotect(0x76f66a81b000, 45056, PROT_READ) = 0
mprotect(0x5c996ef69000, 4096, PROT_READ) = 0
mprotect(0x76f66a978000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
↳ rlim_max=RLIM64_INFINITY}) = 0
munmap(0x76f66a937000, 28532) = 0
getrandom("\xbd\xe0\xd2\xe9\x4c\xaf\x25\x06", 8, GRND_NONBLOCK) = 8
brk(NULL) = 0x5c99716f7000
brk(0x5c9971718000) = 0x5c9971718000
futex(0x76f66a82977c, FUTEX_WAKE_PRIVATE, 2147483647) = 0
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...},
↳ AT_EMPTY_PATH) = 0
write(1, "Enter filename: ", 16) = 16
newfstatat(0, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...},
↳ AT_EMPTY_PATH) = 0
read(0, "test.txt\n", 1024) = 9
pipe2([3, 4], 0) = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
↳ child_tidptr=0x76f66a82f690) = 1533
close(4) = 0
close(4) = -1 EBADF (Bad file descriptor)
read(3, "30.48\n", 255) = 6
write(1, "Answer: 30.48\n", 14) = 14
write(1, "\n", 1) = 1
close(3) = 0
lseek(0, -1, SEEK_CUR) = -1 ESPIPE (Illegal seek)
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=1533, si_uid=1000,
↳ si_status=0, si_utime=0, si_stime=0} ---
exit_group(0) = ?
+++ exited with 0 +++

```