

**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: В. М. Баянов

Группа: М8О-208БВ-24

Преподаватель: Е. С. Миронов

Москва, 2025

Содержание

1. Условие

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Цель работы

Изучение механизмов создания процессов, организации межпроцессного взаимодействия через pipes и обработки данных в многопроцессной архитектуре.

Задание

Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы удаляют все гласные из строк.

Вариант

17

2. Метод решения

Данная программа реализует многопроцессную обработку текстовых данных с использованием каналов (pipes) для межпроцессного взаимодействия. Основной алгоритм: родительский процесс читает строки из стандартного ввода и направляет строки длиной более 10 символов во второй дочерний процесс, остальные - в первый. Каждый дочерний процесс получает строки из своего канала, удаляет гласные буквы и выводит результат в стандартный вывод.

Ключевые компоненты:

ParentProcess - управляет каналами и дочерними процессами

PipeManager - кросс-платформенная реализация каналов

StringProcessor - обработка строк (удаление гласных и фильтрация по длине)

Platform - кроссплатформенные системные функции

Системные вызовы:

Windows: CreatePipe, CreateProcess, ReadFile, WriteFile

Linux: pipe, fork, execl, read, write

Программа использует объектно-ориентированный подход с инкапсуляцией платформно-зависимых особенностей, что обеспечивает кроссплатформенность и четкое разделение ответственности между модулями.

3. Описание программы

Программа реализует многопроцессную обработку текстовых данных через каналы (pipes).

Родительский процесс читает строки из стандартного ввода и распределяет их между двумя дочерними процессами согласно варианту 17: строки длиной более 10 символов отправляются во второй процесс (pipe2), остальные - в первый процесс (pipe1).

Каждый дочерний процесс удаляет все гласные буквы из полученных строк и выводит результат в стандартный вывод.

Архитектура программы включает несколько модулей.

В main.cpp находится точка входа, создающая ParentProcess и запрашивающая имена файлов для дочерних процессов.

Класс ParentProcess (parentProcess.cpp) управляет всей работой: создает каналы, запускает дочерние процессы и распределяет данные по длине строк.

Класс PipeManager (pipeManager.cpp) инкапсулирует работу с каналами, используя CreatePipe на Windows и pipe на Linux.

Класс StringProcessor (stringProcessor.cpp) реализует логику обработки строк: удаление гласных и проверку длины для фильтрации.
Platform (platform.cpp) предоставляет кроссплатформенные функции для работы с процессами и временными задержками.

Дочерние процессы (childProcess.cpp) являются отдельными исполняемыми файлами, которые получают строки через стандартный ввод, обрабатывают их и выводят результаты в стандартный вывод, соответствуя требованию условия.

4. Исходный код

4.1. Основные модули

main.cpp - точка входа программы:

```
1 #include <iostream>
2 #include <string>
3 #include "parentProcess.cpp"
4
5 int main() {
6     std::string filename1, filename2;
7
8
9     std::cout << "Enter filename for child1: ";
10    std::getline(std::cin, filename1);
11
12
13    std::cout << "Enter filename for child2: ";
14    std::getline(std::cin, filename2);
15
16    if (filename1.empty() || filename2.empty()) {
17        std::cerr << "Filenames cannot be empty!" << std::endl;
18        return 1;
19    }
20
21    ParentProcess parent(filename1, filename2);
22
23    if (!parent.initializePipes()) {
24        std::cerr << "Failed to initialize pipes!" << std::endl;
25        return 1;
26    }
27
28    if (!parent.createChildProcesses()) {
29        std::cerr << "Failed to create child processes!" << std
30        ::endl;
31        return 1;
32    }
```

```

31     }
32
33     parent.run();
34
35     return 0;
36 }

```

Listing 1: main.cpp

parentProcess.cpp - родительский процесс:

```

1
2 #pragma once
3
4 #include <iostream>
5 #include <string>
6 #include <vector>
7 #include "pipeManager.hpp"
8 #include "stringProcessor.hpp"
9 #include "platform.hpp"
10
11 #ifdef WINDOWS_PLATFORM
12     #include <windows.h>
13     #include <io.h>
14 #else
15     #include <unistd.h>
16     #include <sys/wait.h>
17 #endif
18
19 class ParentProcess {
20 private:
21     PipeManager pipe1;
22     PipeManager pipe2;
23     std::string filename1, filename2;
24
25 #ifdef WINDOWS_PLATFORM
26     std::vector<PROCESS_INFORMATION> childProcesses;
27 #else
28     std::vector<pid_t> childPids;
29 #endif
30
31     bool createChildProcess(const std::string& executable, const
32                             std::string& filename,
33                             const std::string& processNumber,
34                             PipeManager& inputPipe);
35 public:
36     ParentProcess(const std::string& file1, const std::string&
37                   file2)
38         : filename1(file1), filename2(file2) {}

```

```

37 ~ParentProcess();
38
39 bool initializePipes();
40 bool createChildProcesses();
41 void run();
42 void cleanup();
43 };

```

Listing 2: parentProcess.cpp

parentProcessImplementation.cpp - реализация родительского процесса:

```

1 #include "parentProcess.cpp"
2 #include <iostream>
3 #include <cstring>
4
5 #ifdef WINDOWS_PLATFORM
6
7 bool ParentProcess::createChildProcess(const std::string&
8     executable, const std::string& filename,
9     const std::string&
10     processNumber, PipeManager& inputPipe) {
11     std::string commandLine = executable + " " + filename + " "
12     + processNumber;
13
14     STARTUPINFOA si;
15     PROCESS_INFORMATION pi;
16
17     ZeroMemory(&si, sizeof(si));
18     si.cb = sizeof(si);
19     ZeroMemory(&pi, sizeof(pi));
20
21     si.hStdInput = (HANDLE)_get_osfhandle(inputPipe.getReadFD());
22     ;
23     si.hStdOutput = GetStdHandle(STD_OUTPUT_HANDLE);
24     si.hStdError = GetStdHandle(STD_ERROR_HANDLE);
25     si.dwFlags |= STARTF_USESTDHANDLES;
26
27     char* cmdLine = new char[commandLine.length() + 1];
28     strcpy(cmdLine, commandLine.c_str());
29
30     BOOL success = CreateProcessA(
31         NULL, cmdLine, NULL, NULL, TRUE, 0, NULL, NULL, &si, &pi
32     );
33
34     delete[] cmdLine;
35
36     if (success) {
37         childProcesses.push_back(pi);
38     }
39 }
40
41 #endif

```

```

35         return true;
36     }
37
38     std::cerr << "CreateProcess failed (" << GetLastError() << "
39 )" << std::endl;
40     return false;
41 }
42 #else
43
44 bool ParentProcess::createChildProcess(const std::string&
45     executable, const std::string& filename,
46     const std::string&
47     processNumber, PipeManager& inputPipe) {
48     pid_t pid = fork();
49
50     if (pid == 0) {
51         dup2(inputPipe.getReadFD(), STDIN_FILENO);
52
53         pipe1.closeReadEnd();
54         pipe1.closeWriteEnd();
55         pipe2.closeReadEnd();
56         pipe2.closeWriteEnd();
57
58         execl(executable.c_str(), executable.c_str(), filename.
59 c_str(), processNumber.c_str(), nullptr);
60         perror("execl failed");
61         exit(1);
62     } else if (pid > 0) {
63         childPids.push_back(pid);
64         return true;
65     }
66
67     perror("fork failed");
68     return false;
69 }
70 #endif
71
72 bool ParentProcess::initializePipes() {
73     return pipe1.createPipe() && pipe2.createPipe();
74 }
75
76 bool ParentProcess::createChildProcesses() {
77     bool success1 = createChildProcess("./childProcess",
78     filename1, "1", pipe1);
79     Platform::sleep(100);

```



```

79
80
81     bool success2 = createChildProcess("./childProcess",
82     filename2, "2", pipe2);
83     Platform::sleep(100);
84
85     pipe1.closeReadEnd();
86     pipe2.closeReadEnd();
87
88     return success1 && success2;
89 }
90
91 void ParentProcess::run() {
92     std::cout << "Parent process started. Enter strings (empty
93     line to exit):" << std::endl;
94
95     std::string input;
96     while (true) {
97         std::cout << "> ";
98         std::getline(std::cin, input);
99
100         if (input.empty()) {
101             break;
102         }
103
104         if (StringProcessor::shouldGoToPipe2(input)) {
105             std::cout << "Routing to PIPE2 (long string >10): "
106             << input << std::endl;
107             if (!pipe2.writeToPipe(input)) {
108                 std::cerr << "Failed to write to pipe2" << std::
109             endl;
110             }
111         } else {
112             std::cout << "Routing to PIPE1 (short string 10): "
113             << input << std::endl;
114             if (!pipe1.writeToPipe(input)) {
115                 std::cerr << "Failed to write to pipe1" << std::
116             endl;
117             }
118         }
119
120         pipe1.closeWriteEnd();
121         pipe2.closeWriteEnd();
122
123         std::cout << "Waiting for child processes to finish..." <<
124         std::endl;

```

```

121     cleanup();
122     std::cout << "Parent process finished." << std::endl;
123 }
124
125 void ParentProcess::cleanup() {
126 #ifdef WINDOWS_PLATFORM
127     for (auto& pi : childProcesses) {
128         WaitForSingleObject(pi.hProcess, INFINITE);
129         CloseHandle(pi.hProcess);
130         CloseHandle(pi.hThread);
131     }
132     childProcesses.clear();
133 #else
134     for (auto pid : childPids) {
135         waitpid(pid, nullptr, 0);
136     }
137     childPids.clear();
138 #endif
139 }
140
141 ParentProcess::~ParentProcess() {
142     cleanup();
143 }

```

Listing 3: parentProcessImplementation.cpp

childProcess.cpp - дочерний процесс:

```

1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  #include "stringProcessor.hpp"
5
6  int main(int argc, char* argv[]) {
7      if (argc != 3) {
8          std::cerr << "Usage: " << argv[0] << " <filename> <
9              process_number>" << std::endl;
10         return 1;
11     }
12
13     std::string filename = argv[1];
14     int processNumber = std::stoi(argv[2]);
15
16     std::ofstream file(filename);
17     if (!file.is_open()) {
18         std::cerr << "Error opening file: " << filename << std::
19         endl;
20         return 1;
21     }
22     file.close();

```

```

21
22     std::string input;
23     while (true) {
24         if (!std::getline(std::cin, input)) {
25             break;
26         }
27
28         if (input.empty()) {
29             continue;
30         }
31
32         std::string processed = StringProcessor::removeVowels(
33             input);
34         std::cout << "Child" << processNumber << " processed: '"
35             << input
36             << "' -> '" << processed << "'" << std::endl;
37         std::cout.flush();
38     }
39     return 0;

```

Listing 4: childProcess.cpp

4.2. Вспомогательные модули

pipeManager.cpp - управление каналами:

```

1  #include "pipeManager.hpp"
2  #include <iostream>
3  #include <cstring>
4
5  #ifdef WINDOWS_PLATFORM
6
7  PipeManager::PipeManager() : readHandle(INVALID_HANDLE_VALUE),
8                               writeHandle(INVALID_HANDLE_VALUE),
9                               isReadEndOpen(false), isWriteEndOpen(
10                                   false) {}
11
12 PipeManager::~PipeManager() {
13     closeBoth();
14 }
15
16 bool PipeManager::createPipe() {
17     SECURITY_ATTRIBUTES saAttr;
18     saAttr.nLength = sizeof(SECURITY_ATTRIBUTES);
19     saAttr.bInheritHandle = TRUE;
20     saAttr.lpSecurityDescriptor = NULL;

```

```

19
20     if (!CreatePipe(&readHandle, &writeHandle, &saAttr, 0)) {
21         std::cerr << "CreatePipe failed" << std::endl;
22         return false;
23     }
24
25     isReadEndOpen = true;
26     isWriteEndOpen = true;
27     return true;
28 }
29
30 bool PipeManager::writeToPipe(const std::string& data) {
31     if (!isWriteEndOpen) return false;
32
33     DWORD bytesWritten;
34     std::string dataWithNewline = data + "\n";
35     BOOL success = WriteFile(writeHandle, dataWithNewline.c_str
36     (), dataWithNewline.length(), &bytesWritten, NULL);
37     return success && (bytesWritten == dataWithNewline.length())
38     ;
39 }
40
41 bool PipeManager::readFromPipe(std::string& data) {
42     if (!isReadEndOpen) return false;
43
44     char buffer[1024];
45     DWORD bytesRead;
46     BOOL success = ReadFile(readHandle, buffer, sizeof(buffer) -
47     1, &bytesRead, NULL);
48
49     if (success && bytesRead > 0) {
50         buffer[bytesRead] = '\0';
51         data = std::string(buffer);
52         //
53         if (!data.empty() && data.back() == '\n') {
54             data.pop_back();
55         }
56         return true;
57     }
58     return false;
59 }
60
61 void PipeManager::closeReadEnd() {
62     if (isReadEndOpen) {
63         CloseHandle(readHandle);
64         isReadEndOpen = false;
65     }
66 }

```

```

65 void PipeManager::closeWriteEnd() {
66     if (isWriteEndOpen) {
67         CloseHandle(writeHandle);
68         isWriteEndOpen = false;
69     }
70 }
71
72 void PipeManager::closeBoth() {
73     closeReadEnd();
74     closeWriteEnd();
75 }
76
77 int PipeManager::getReadFD() const {
78     return _open_osfhandle((intptr_t)readHandle, _O_RDONLY);
79 }
80
81 int PipeManager::getWriteFD() const {
82     return _open_osfhandle((intptr_t)writeHandle, _O_WRONLY);
83 }
84
85 #else
86
87 PipeManager::PipeManager() : isReadEndOpen(false),
88     isWriteEndOpen(false) {
89     pipefd[0] = -1;
90     pipefd[1] = -1;
91 }
92
93 PipeManager::~PipeManager() {
94     closeBoth();
95 }
96
97 bool PipeManager::createPipe() {
98     if (pipe(pipefd) == -1) {
99         perror("pipe");
100         return false;
101     }
102     isReadEndOpen = true;
103     isWriteEndOpen = true;
104     return true;
105 }
106
107 bool PipeManager::writeToPipe(const std::string& data) {
108     if (!isWriteEndOpen) return false;
109     std::string dataWithNewline = data + "\n";
110     ssize_t bytesWritten = write(pipefd[1], dataWithNewline.
c_str(), dataWithNewline.length());
    return bytesWritten == static_cast<ssize_t>(dataWithNewline.
length());

```

```

111 }
112
113 bool PipeManager::readFromPipe(std::string& data) {
114     if (!isReadEndOpen) return false;
115
116     char buffer[1024];
117     ssize_t bytesRead = read(pipefd[0], buffer, sizeof(buffer) -
118                             1);
119
120     if (bytesRead > 0) {
121         buffer[bytesRead] = '\0';
122         data = std::string(buffer);
123         //
124         if (!data.empty() && data.back() == '\n') {
125             data.pop_back();
126         }
127         return true;
128     }
129     return false;
130 }
131
132 void PipeManager::closeReadEnd() {
133     if (isReadEndOpen) {
134         close(pipefd[0]);
135         isReadEndOpen = false;
136     }
137 }
138
139 void PipeManager::closeWriteEnd() {
140     if (isWriteEndOpen) {
141         close(pipefd[1]);
142         isWriteEndOpen = false;
143     }
144 }
145
146 void PipeManager::closeBoth() {
147     closeReadEnd();
148     closeWriteEnd();
149 }
150
151 int PipeManager::getReadFD() const { return pipefd[0]; }
152 int PipeManager::getWriteFD() const { return pipefd[1]; }
153 #endif

```

Listing 5: pipeManager.cpp

stringProcessor.cpp - обработка строк:

```

1 #include "stringProcessor.hpp"

```

```

2
3 std::string StringProcessor::removeVowels(const std::string&
  input) {
4     const std::string VOWELS = "aeiouAEIOU";
5     std::string result = input;
6     result.erase(std::remove_if(result.begin(), result.end(),
7         [&VOWELS](char c) {
8             return VOWELS.find(c) != std::string::npos;
9         }), result.end());
10    return result;
11 }
12
13 bool StringProcessor::shouldGoToPipe2(const std::string& input)
  {
14     return input.length() > 10;
15 }

```

Listing 6: stringProcessor.cpp

platform.cpp - кроссплатформенные функции:

```

1 #include "platform.hpp"
2 #include <thread>
3
4 void Platform::sleep(int milliseconds) {
5     #ifdef WINDOWS_PLATFORM
6         Sleep(milliseconds);
7     #else
8         usleep(milliseconds * 1000);
9     #endif
10 }

```

Listing 7: platform.cpp

5. Логи выполнения программы

```

274130 execve("./parent",["./parent"],0x7ffcd3e49d68 /* 36 vars */) = 0
274130 brk(NULL)      = 0x63bd01b85000
274130 mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7
274130 access("/etc/ld.so.preload",R_OK) = -1 ENOENT (No such file or directory)
274130 openat(AT_FDCWD,"/etc/ld.so.cache",O_RDONLY|O_CLOEXEC) = 3
274130 fstat(3,{st_mode=S_IFREG|0644,st_size=33163,...}) = 0
274130 mmap(NULL,33163,PROT_READ,MAP_PRIVATE,3,0) = 0x79c2f6924000
274130 close(3)        = 0
274130 openat(AT_FDCWD,"/lib/x86_64-linux-gnu/libstdc++.so.6",O_RDONLY|O_CLOEXEC
274130 read(3,"\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0"

```

```

274130 fstat(3,{st_mode=S_IFREG|0644,st_size=2592224,...}) = 0
274130 mmap(NULL,2609472,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x79c2f66000
274130 mmap(0x79c2f669d000,1343488,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|MAP_
274130 mmap(0x79c2f67e5000,552960,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
274130 mmap(0x79c2f686c000,57344,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_
274130 mmap(0x79c2f687a000,12608,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_
274130 close(3) = 0
274130 pipe2([3,4],0) = 0
274130 pipe2([5,6],0) = 0
274130 clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGC
274130 clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGC
274130 close(3) = 0
274130 close(5) = 0
274150 execve("./child",["./child","output1.txt"],0x7ffd73be59b8 /* 36 vars */)
274151 execve("./child",["./child","output2.txt"],0x7ffd73be59b8 /* 36 vars */)
274130 read(0,"hello world\n",1024) = 12
274130 write(4,"hello world",11) = 11
274130 write(4,"\n",1) = 1
274150 read(3,"hello world",255) = 11
274130 read(0,"short\n",1024) = 6
274130 write(4,"short",5) = 5
274130 write(4,"\n",1) = 1
274150 read(3,"short",255) = 5
274130 read(0,"this is very long string\n",1024) = 24
274130 write(6,"this is very long string",23) = 23
274130 write(6,"\n",1) = 1
274151 read(5,"this is very long string",255) = 23
274130 read(0,"test\n",1024) = 5
274130 write(4,"test",4) = 4
274130 write(4,"\n",1) = 1
274150 read(3,"test",255) = 4
274130 read(0,"\n",1024) = 1
274130 close(4) = 0
274130 close(6) = 0
274130 wait4(274150,NULL,0,NULL) = 274150
274130 wait4(274151,NULL,0,NULL) = 274151
274130 exit_group(0) = ?
274150 exit_group(0) = ?
274151 exit_group(0) = ?

```

Программа демонстрирует корректную работу механизма межпроцессного

взаимодействия через именованные каналы и правильное распределение строк между процессами согласно варианту 17.