

**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. А. Алгиничев
Группа: М8О-208БВ-24
Преподаватель: Е. С. Миронов
Дата сдачи: 05.10.2025

Москва, 2025

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 пересыпает их в pipe1 или в pipe2 в зависимости от правила фильтрации. 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: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы инвертируют строки.

Вариант

21

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 нечетные строки первому дочернему процессу, четные - второму. Каждый дочерний процесс получает строки из своего канала, реверсирует их и записывает в указанный файл.

Ключевые компоненты:

ParentProcess - управляет каналами и дочерними процессами

Pipe - кроссплатформенная реализация каналов

ChildProcess - запускает дочерние процессы

ChildProcessor - обрабатывает данные в дочерних процессах

Системные вызовы:

Windows: CreatePipe, CreateProcess, ReadFile, WriteFile

Linux: pipe, fork, exec, read, write

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

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: нечетные строки отправляются первому процессу, четные - второму.

Каждый дочерний процесс переворачивает полученные строки задом наперед и записывает результат в указанный файл.

Архитектура программы включает несколько модулей.

В main.cpp находится точка входа, создающая ParentProcess.

Класс ParentProcess (process.cpp) управляет всей работой: создает каналы, запрашивает имена файлов, запускает дочерние процессы и распределяет данные.

Класс Pipe (pipe.cpp) инкапсулирует работу с каналами, используя CreatePipe на Windows и pipe на Linux. Класс ChildProcess (childProcess.cpp/hpp) отвечает за запуск дочерних процессов через CreateProcess (Windows) или fork/exec (Linux).

ChildProcessor (childProcessor.cpp/hpp) обрабатывает данные в дочерних процессах: читает из канала, переворачивает строки и записывает в файл.

Результаты

Разработанная программа успешно реализует многопроцессную архитектуру для параллельной обработки текстовых данных.

В ходе решения были достигнуты следующие ключевые результаты:

Корректная работа системы межпроцессного взаимодействия

Реализованы два независимых канала передачи данных между родительским и дочерними процессами

Обеспечено четкое распределение строк по принципу четности/нечетности

Достигнута синхронизация процессов через блокирующие операции чтения/записи

Кросс-платформенная функциональность

Программа корректно работает как в Windows, так и в Linux/Unix системах

Реализована унифицированная абстракция для работы с каналами через класс Pipe

Обеспечен единый интерфейс для создания процессов на разных платформах

Выводы

В ходе лабораторной работы успешно разработана многопроцессная система обработки текстовых данных с использованием межпроцессного взаимодействия через каналы. Программа демонстрирует корректную работу в кроссплатформенном режиме на Windows и Unix системах.

Исходная программа

```
1 #include "process.hpp"
2
3 int main() {
4     ParentProcess parent;
5     parent.start();
6     return 0;
7 }
```

Листинг 1: main.cpp - точка входа, создает ParentProcess

```
1 #include <iostream>
2 #include <string>
3 #include "process.hpp"
4
5 #ifdef _WIN32
6     #include <windows.h>
7     #include <io.h>
8     #define close _close
9     #define read _read
10    #define write _write
11#else
12    #include <unistd.h>
13    #include <sys/wait.h>
14    #include <fcntl.h>
15#endif
16
17 ParentProcess::ParentProcess() {
18     std::string file1, file2;
19     std::cout << "    child1: ";
20     std::getline(std::cin, file1);
21     std::cout << "    child2: ";
22     std::getline(std::cin, file2);
23
24     pipe1 = new Pipe();
25     pipe2 = new Pipe();
26     child1 = new ChildProcess(pipe1, file1, true);
27     child2 = new ChildProcess(pipe2, file2, false);
28 }
29
30 void ParentProcess::start() {
31     child1->execute();
32     child2->execute();
33
34     close(pipe1->getReadFd());
35     close(pipe2->getReadFd());
36
37     std::string line;
38     int line_number = 0;
39
40     while (std::getline(std::cin, line))
41     {
42         line_number++;
43         if (line_number % 2 == 1) {
44             write(pipe1->getWriteFd(), line.c_str(), line.length());
45             write(pipe1->getWriteFd(), "\n", 1);
46     }
```

```

46     } else {
47         write(pipe2->getWriteFd(), line.c_str(), line.length());
48         write(pipe2->getWriteFd(), "\n", 1);
49     }
50 }
51
52     close(pipe1->getWriteFd());
53     close(pipe2->getWriteFd());
54
55 #ifdef _WIN32
56     Sleep(1000);
57 #else
58     /*
59     int status1, status2;
60     waitpid(child1->getPid(), &status1, 0);
61     waitpid(child2->getPid(), &status2, 0);
62     */
63 #endif
64 }
65
66 ParentProcess::~ParentProcess() {
67     delete child1;
68     delete child2;
69     delete pipe1;
70     delete pipe2;
71 }
```

Листинг 2: process.cpp - класс ParentProcess, управляет дочерними процессами

```

1 #include <string>
2 #include <iostream>
3 #include <cstdlib>
4 #include "childProcess.hpp"
5
6 #ifdef _WIN32
7     #include <windows.h>
8 #else
9     #include <unistd.h>
10    #include <sys/wait.h>
11 #endif
12
13 ChildProcess::ChildProcess(Pipe* p, const std::string& f, bool is_c1)
14     : pipe(p), file_name(f), is_child1(is_c1), pid(-1) {}
15
16 void ChildProcess::execute() {
17 #ifdef _WIN32
18     STARTUPINFOA si;
19     PROCESS_INFORMATION pi;
20     ZeroMemory(&si, sizeof(si));
21     si.cb = sizeof(si);
22     ZeroMemory(&pi, sizeof(pi));
23
24     std::string read_fd_str = std::to_string((int)pipe->getReadFd());
25     std::string prefix = is_child1 ? "child1" : "child2";
26
27     std::string command = "child.exe " + prefix + " " + file_name + " " + read_fd_str;
28 }
```

```

29     if (!CreateProcess(NULL, (LPSTR)command.c_str(), NULL, NULL,
30                         TRUE, 0, NULL, NULL, &si, &pi)) {
31         std::cerr << "    " << std::endl;
32         exit(1);
33     }
34
35     pid = pi.dwProcessId;
36     CloseHandle(pi.hThread);
37 #else
38     pid = fork();
39     if (pid == -1) {
40         perror("    ");
41         exit(1);
42     }
43
44     if (pid == 0) {
45         close(pipe->getWriteFd());
46
47         std::string read_fd_str = std::to_string(pipe->getReadFd());
48         std::string prefix = is_child1 ? "child1" : "child2";
49
50         execl("./child", "./child", prefix.c_str(), file_name.c_str(),
51               read_fd_str.c_str(), nullptr);
52
53         perror("    ");
54         exit(1);
55     }
56     else {
57         close(pipe->getReadFd());
58     }
59 #endif
60 }
61
62 #ifdef _WIN32
63 DWORD ChildProcess::getPid() const {
64     return pid;
65 }
66 #else
67 pid_t ChildProcess::getPid() const {
68     return pid;
69 }
70#endif

```

Листинг 3: childProcess.cpp - класс ChildProcess, запуск дочерних процессов

```

1 #include <iostream>
2 #include <cstdlib>
3 #include <string>
4 #include "childProcessor.hpp"
5
6 #ifdef _WIN32
7     #include <windows.h>
8     #include <io.h>
9 #endif
10
11 int main(int argc, char* argv[]) {
12     if (argc != 4) {

```

```

13     std::cerr << ":" << argv[0] << " <prefix> <output_file> <read_fd>" << std::endl;
14     return 1;
15 }
16
17 std::string prefix = argv[1];
18 std::string output_file = argv[2];
19 int read_fd = std::atoi(argv[3]);
20
21 std::cout << prefix << ":" << output_file << std::endl;
22
23 ChildProcessor processor(prefix);
24 return processor.process(read_fd, output_file);
25 }
```

Листинг 4: child main.cpp - точка входа дочернего процесса

```

1 #include <iostream>
2 #include <string>
3 #include <cstdlib>
4 #include "childProcessor.hpp"
5
6 #ifdef _WIN32
7     #include <windows.h>
8     #include <io.h>
9     #include <fcntl.h>
10    #define close _close
11    #define read _read
12    #define write _write
13    #define open _open
14    #define O_WRONLY _O_WRONLY
15    #define O_CREAT _O_CREAT
16    #define O_TRUNC _O_TRUNC
17 #else
18     #include <unistd.h>
19     #include <fcntl.h>
20 #endif
21
22 ChildProcessor::ChildProcessor(const std::string& pref) : prefix(pref) {}
23
24 std::string ChildProcessor::reverseString(const std::string& s) {
25     size_t len = s.length();
26     std::string result;
27     for (size_t i = len; i > 0; --i) {
28         result += s[i - 1];
29     }
30     return result;
31 }
32
33 int ChildProcessor::process(int read_fd, const std::string& output_file) {
34 #ifdef _WIN32
35     int fd = open(output_file.c_str(), O_WRONLY | O_CREAT | O_TRUNC, _S_IREAD |
36                 _S_IWRITE);
37 #else
38     int fd = open(output_file.c_str(), O_WRONLY | O_CREAT | O_TRUNC, 0644);
39 #endif

```

```

40     if (fd == -1) {
41         perror("    ");
42         return 1;
43     }
44
45     ssize_t count;
46     while ((count = read(read_fd, buffer, sizeof(buffer) - 1)) > 0)
47     {
48         buffer[count] = '\0';
49         std::string line(buffer);
50         std::string reversed = reverseString(line);
51         write(fd, reversed.c_str(), reversed.length());
52     }
53
54     if (count == -1) {
55         perror("    ");
56     }
57
58     close(read_fd);
59     close(fd);
60     return 0;
61 }
```

Листинг 5: childProcessor.cpp - класс ChildProcessor, обработка строк

```

1 #include <iostream>
2 #include <cstdlib>
3 #include "pipe.hpp"
4
5 #ifdef _WIN32
6 #include <io.h>
7 #define close _close
8 #define read _read
9 #define write _write
10#endif
11
12 Pipe::Pipe() {
13 #ifdef _WIN32
14     SECURITY_ATTRIBUTES sa;
15     sa.nLength = sizeof(SECURITY_ATTRIBUTES);
16     sa.bInheritHandle = TRUE;
17     sa.lpSecurityDescriptor = NULL;
18
19     if (!CreatePipe(&read_fd, &write_fd, &sa, 0)) {
20         std::cerr << "    " << std::endl;
21         exit(1);
22     }
23 #else
24     if (pipe(fd) == -1) {
25         perror("    ");
26         exit(1);
27     }
28#endif
29}
30
31 int Pipe::getReadFd() const {
32 #ifdef _WIN32
```

```
33     return (int)read_fd;
34 #else
35     return fd[0];
36 #endif
37 }
38
39 int Pipe::getWriteFd() const {
40 #ifdef _WIN32
41     return (int)write_fd;
42 #else
43     return fd[1];
44 #endif
45 }
46
47 Pipe::~Pipe() {
48 #ifdef _WIN32
49     CloseHandle(read_fd);
50     CloseHandle(write_fd);
51 #else
52     close(fd[0]);
53     close(fd[1]);
54 #endif
55 }
```

Листинг 6: pipe.cpp - класс Pipe, работа с каналами

Системные вызовы


```
= 0x79c2f68f1000
274130 arch_prctl(ARCH_SET_FS,0x79c2f68f1740) = 0
274130 set_tid_address(0x79c2f68f1a10) = 274130
274130 set_robust_list(0x79c2f68f1a20,24) = 0
274130 rseq(0x79c2f68f2060,0x20,0,0x53053053) = 0
274130 mprotect(0x79c2f63ff000,16384,PROT_READ) = 0
274130 mprotect(0x79c2f65fe000,4096,PROT_READ) = 0
274130 mprotect(0x79c2f6922000,4096,PROT_READ) = 0
274130 mprotect(0x79c2f686c000,45056,PROT_READ) = 0
274130 mprotect(0x63bcc532c000,4096,PROT_READ) = 0
274130 mprotect(0x79c2f6965000,8192,PROT_READ) = 0
274130 prlimit64(0,RLIMIT_STACK,NULL,{rlim_cur=8192*1024,rlim_max=RLIM64_INFINITY}) = 0
274130 munmap(0x79c2f6924000,33163) = 0
274130 futex(0x79c2f687a7bc,FUTEX_WAKE_PRIVATE,2147483647) = 0
274130 getrandom("\x2c\x58\x54\xd0\x53\xf6\x8a\x94",8,GRND_NONBLOCK) = 8
274130 brk(NULL) = 0x63bd01b85000
274130 brk(0x63bd01ba6000) = 0x63bd01ba6000
274130 fstat(1,{st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0x5),...}) = 0
274130 write(1,"\\320\\222\\320\\262\\320\\265\\320\\264\\320\\270\\321\\202\\320\\265 \\320\\270\\320\\321\\204\\320\\260\\320\\271\\320\\273\\320\\260"...,48) = 48
274130 fstat(0,{st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0x5),...}) = 0
274130 read(0,"file1\n",1024) = 6
274130 write(1,"\\320\\222\\320\\262\\320\\265\\320\\264\\320\\270\\321\\202\\320\\265 \\320\\270\\320\\321\\204\\320\\260\\320\\271\\320\\273\\320\\260"...,48) = 48
274130 read(0,"file2\n",1024) = 6
274130 pipe2([3,4],0) = 0
274130 pipe2([5,6],0) = 0
274130 clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,c = 274150
274150 set_robust_list(0x79c2f68f1a20,24 <unfinished ...>
274130 close(3 <unfinished ...>
274150 <... set_robust_list resumed>) = 0
274130 <... close resumed>) = 0
274150 close(4 <unfinished ...>
274130 clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD <unfinished ...>
274150 <... close resumed>) = 0
274150 execve("./child",["./child","child1","file1","3"],0x7ffd73be59b8 /* 36 vars */ <unfinished ...>
274130 <... clone resumed>,child_tidptr=0x79c2f68f1a10) = 274151
274130 close(5 <unfinished ...>
274151 set_robust_list(0x79c2f68f1a20,24 <unfinished ...>
274130 <... close resumed>) = 0
274151 <... set_robust_list resumed>) = 0
274130 close(3 <unfinished ...>
274150 <... execve resumed>) = 0
274130 <... close resumed>) = -1 EBADF (Bad file descriptor)
```



```
274151 <... close resumed>          = 0
274150 <... mmap resumed>          = 0x709cf5cc2000
274151 mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0 <unfinished ...
...>
274150 arch_prctl(ARCH_SET_FS,0x709cf5cc2740 <unfinished ...>
274151 <... mmap resumed>          = 0x77760c0a3000
274150 <... arch_prctl resumed>      = 0
274151 mmap(NULL,12288,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0
<unfinished ...>
274150 set_tid_address(0x709cf5cc2a10 <unfinished ...>
274151 <... mmap resumed>          = 0x77760c0a0000
274150 <... set_tid_address resumed> = 274150
274151 arch_prctl(ARCH_SET_FS,0x77760c0a0740 <unfinished ...>
274150 set_robust_list(0x709cf5cc2a20,24 <unfinished ...>
274151 <... arch_prctl resumed>      = 0
274150 <... set_robust_list resumed> = 0
274151 set_tid_address(0x77760c0a0a10 <unfinished ...>
274150 rseq(0x709cf5cc3060,0x20,0,0x53053053 <unfinished ...>
274151 <... set_tid_address resumed> = 274151
274150 <... rseq resumed>          = 0
274151 set_robust_list(0x77760c0a0a20,24) = 0
274150 mprotect(0x709cf5ff000,16384,PROT_READ <unfinished ...>
274151 rseq(0x77760c0a1060,0x20,0,0x53053053 <unfinished ...>
274150 <... mprotect resumed>        = 0
274151 <... rseq resumed>          = 0
274150 mprotect(0x709cf5dae000,4096,PROT_READ <unfinished ...>
274151 mprotect(0x77760bbff000,16384,PROT_READ <unfinished ...>
274150 <... mprotect resumed>        = 0
274151 <... mprotect resumed>        = 0
274150 mprotect(0x709cf5ddc000,4096,PROT_READ <unfinished ...>
274151 mprotect(0x77760bdd0000,4096,PROT_READ <unfinished ...>
274150 <... mprotect resumed>        = 0
274151 <... mprotect resumed>        = 0
274151 mprotect(0x77760bdfe000,4096,PROT_READ) = 0
274150 mprotect(0x709cf5c6c000,45056,PROT_READ) = 0
274151 mprotect(0x77760c06c000,45056,PROT_READ <unfinished ...>
274150 mprotect(0x575e22ada000,4096,PROT_READ <unfinished ...>
274151 <... mprotect resumed>        = 0
274150 <... mprotect resumed>        = 0
274151 mprotect(0x5cc39ee78000,4096,PROT_READ <unfinished ...>
274150 mprotect(0x709cf5e1f000,8192,PROT_READ <unfinished ...>
274151 <... mprotect resumed>        = 0
274150 <... mprotect resumed>        = 0
274151 mprotect(0x77760c0e6000,8192,PROT_READ <unfinished ...>
274150 prlimit64(0,RLIMIT_STACK,NULL, <unfinished ...>
274151 <... mprotect resumed>        = 0
274150 <... prlimit64 resumed>{rlim_cur=8192*1024,rlim_max=RLIM64_INFINITY})
= 0
```

```
274151 prlimit64(0,RLIMIT_STACK,NULL,{rlim_cur=8192*1024,rlim_max=RLIM64_INFINITY})  
= 0  
274150 munmap(0x709cf5dde000,33163 <unfinished ...>  
274151 munmap(0x77760c0a5000,33163 <unfinished ...>  
274150 <... munmap resumed> = 0  
274151 <... munmap resumed> = 0  
274150 futex(0x709cf5c7a7bc,FUTEX_WAKE_PRIVATE,2147483647) = 0  
274151 futex(0x77760c07a7bc,FUTEX_WAKE_PRIVATE,2147483647) = 0  
274150 getrandom("\xcd\xA7\x97\x3C\x1D\x9A\xD8\x81",8,GRND_NONBLOCK) = 8  
274151 getrandom( <unfinished ...>  
274150 brk(NULL <unfinished ...>  
274151 <... getrandom resumed>"\xf5\x9d\xac\xba\x11\xff\xdd\xc4",8,GRND_NONBLOCK)  
= 8  
274150 <... brk resumed> = 0x575e53eab000  
274151 brk(NULL <unfinished ...>  
274150 brk(0x575e53ecc000 <unfinished ...>  
274151 <... brk resumed> = 0x5cc3d829f000  
274150 <... brk resumed> = 0x575e53ecc000  
274151 brk(0x5cc3d82c0000) = 0x5cc3d82c0000  
274150 fstat(1, <unfinished ...>  
274151 fstat(1, <unfinished ...>  
274150 <... fstat resumed>{st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0x5),...}  
= 0  
274151 <... fstat resumed>{st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0x5),...}  
= 0  
274150 write(1,"child1: \320\267\320\260\320\277\321\203\321\211\320\265\320\275  
\321\201 \321\204\320\260\320\271"...,45 <unfinished ...>  
274151 write(1,"child2: \320\267\320\260\320\277\321\203\321\211\320\265\320\275  
\321\201 \321\204\320\260\320\271"...,45 <unfinished ...>  
274150 <... write resumed> = 45  
274151 <... write resumed> = 45  
274150 openat(AT_FDCWD,"file1",O_WRONLY|O_CREAT|O_TRUNC,0644 <unfinished ...>  
274151 openat(AT_FDCWD,"file2",O_WRONLY|O_CREAT|O_TRUNC,0644) = 3  
274150 <... openat resumed> = 4  
274151 read(5, <unfinished ...>  
274150 read(3, <unfinished ...>  
274130 <... read resumed>"hello\n",1024) = 6  
274130 write(4,"hello",5) = 5  
274150 <... read resumed>"hello",255) = 5  
274130 write(4,"\n",1 <unfinished ...>  
274150 write(4,"olleh",5 <unfinished ...>  
274130 <... write resumed> = 1  
274150 <... write resumed> = 5  
274130 read(0, <unfinished ...>  
274150 read(3,"\\n",255) = 1  
274150 write(4,"\\n",1) = 1  
274150 read(3, <unfinished ...>  
274130 <... read resumed>"rabbit\\n",1024) = 7
```

```
274130 write(6,"rabbit",6) = 6
274151 <... read resumed>"rabbit",255) = 6
274130 write(6,"\n",1) = 1
274151 write(3,"tibbar",6 <unfinished ...>
274130 read(0, <unfinished ...>
274151 <... write resumed>) = 6
274151 read(5,"\n",255) = 1
274151 write(3,"\n",1) = 1
274151 read(5, <unfinished ...>
274130 <... read resumed>"dig\n",1024) = 4
274130 write(4,"dig",3) = 3
274150 <... read resumed>"dig",255) = 3
274130 write(4,"\n",1 <unfinished ...>
274150 write(4,"gid",3 <unfinished ...>
274130 <... write resumed>) = 1
274150 <... write resumed>) = 3
274130 read(0, <unfinished ...>
274150 read(3,"\n",255) = 1
274150 write(4,"\n",1) = 1
274150 read(3, <unfinished ...>
274130 <... read resumed>"big\n",1024) = 4
274130 write(6,"big",3) = 3
274151 <... read resumed>"big",255) = 3
274130 write(6,"\n",1 <unfinished ...>
274151 write(3,"gib",3 <unfinished ...>
274130 <... write resumed>) = 1
274151 <... write resumed>) = 3
274130 read(0, <unfinished ...>
274151 read(5,"\n",255) = 1
274151 write(3,"\n",1) = 1
274151 read(5, <unfinished ...>
274130 <... read resumed>0x63bd01b976c0,1024) = ? ERESTARTSYS (To be restarted
if SA_RESTART is set)
274151 <... read resumed>0x7fffcc9869d0,255) = ? ERESTARTSYS (To be restarted
if SA_RESTART is set)
274150 <... read resumed>0x7ffc28419ee0,255) = ? ERESTARTSYS (To be restarted
if SA_RESTART is set)
274130 ---SIGINT {si_signo=SIGINT,si_code=SI_KERNEL} ---
274151 ---SIGINT {si_signo=SIGINT,si_code=SI_KERNEL} ---
274150 ---SIGINT {si_signo=SIGINT,si_code=SI_KERNEL} ---
274130 +++ killed by SIGINT +++
274151 +++ killed by SIGINT +++
274150 +++ killed by SIGINT +++
```