

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. Губиев

Группа: М8О-208БВ-24

Преподаватель: Е.С. Миронов

Москва, 2025

**Условие:**

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода организовано через систему каналов (pipes). Родительский и дочерние процессы должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересыпает их в pipe1. Процессы child1 и child2 производят работу над строками, передавая данные последовательно через pipe2. Child2 пересыпает результат своей работы родительскому процессу через pipe3.

**Цель работы:**

Приобретение практических навыков: управление процессами в ОС; обеспечение обмена данных между процессами посредством каналов.

**Задание:**

Обработка строк: Child1 переводит строки в нижний регистр. Child2 убирает все задвоенные пробелы. **Вариант: 14**

**Метод решения**

Данная программа реализует многопроцессную обработку текстовых данных с использованием каналов (pipes) для межпроцессного взаимодействия. Родительский процесс читает строки из стандартного ввода и направляет их через цепочку дочерних процессов, каждый из которых выполняет преобразование данных.

**Основные компоненты:**

Parent - управляет созданием каналов, запуском дочерних процессов, принимает пользовательский ввод и выводит конечный результат;

Child1 - приводит текст к нижнему регистру;

Child2 - удаляет все задвоенные пробелы;

Разделение системных вызовов в отдельную библиотеку systemCall.

**Описание программы**

**Структура проекта:**

lab1/ report/

...

include/

systemCall.h // Заголовочный файл библиотеки

src/

    systemCall.cpp // Реализация системных функций

    parent.cpp // Родительский процесс

    child1.cpp // Дочерний процесс 1 (нижний регистр)

    child2.cpp // Дочерний процесс 2 (удаление пробелов)

    CMakeLists.txt

**Основные типы данных:**

1. Структура pipeT (канал)

Содержит два конца: один для чтения данных, другой для записи

На Windows использует дескрипторы HANDLE, на Linux - файловые дескрипторы

Позволяет организовать одностороннюю передачу данных между процессами

2.Структура process (информация о процессе)

Хранит идентификатор запущенного процесса

На Windows содержит подробную информацию о процессе, на Linux - просто номер процесса (PID)

Содержит флаг is-valid, который показывает, работает ли процесс корректно

3.Строки std::string

4.Логические флаги (bool)

#### **Принцип работы с типами данных:**

Программа создает несколько каналов pipe t, через которые передаются строки std::string. Каждый дочерний процесс управляет через свою структуру process info t, а логические флаги следят за тем, чтобы вся система работала без ошибок.

#### **Основные функции программы:**

PipeCreate() - создает новый канал для передачи данных

PipeClose() - полностью закрывает канал, освобождая ресурсы

ProcessCreate() - запускает дочерний процесс

ProcessTerminate() - принудительно завершает процесс

ReadStringFromPipe() - читает строку из канала

WriteStringToPipe() - записывает строку в канал

#### **Используемые системные вызовы:**

Для Windows:

CreatePipe() - создание канала;

CreateProcess() - создание процесса;

CloseHandle() - закрытие дескриптора;

ReadFile()/WriteFile() - работа с каналами;

TerminateProcess() - принудительное завершение;

Для Linux:

pipe() - создание канала; fork() - создание

процесса; exec() - загрузка новой программы;

close() - закрытие дескриптора; read()/write() -

работа с каналами; kill() - отправка сигнала

процессу; dup2() - перенаправление

стандартных потоков.

## **Результаты**

Разработана многопроцессная система для конвейерной обработки текстовых данных, состоящая из трех взаимосвязанных процессов, взаимодействующих через систему каналов.

Ключевые результаты: Реализованы три канала передачи данных, образующие последовательный конвейер обработки, обеспечена надежная передача текстовых данных от родительского процесса через два дочерних процесса; Реализована корректная обработка системных ошибок, дочерние процессы корректно завершаются при получении сигнала EOF от родительского процесса; Программа корректно работает как в Windows, так и в Linux/Unix системах

## **Выводы**

В ходе лабораторной работы была успешно реализована многопроцессная система конвейерной обработки текстовых данных. Программа демонстрирует эффективное использование механизма каналов для организации межпроцессного взаимодействия. Реализована кроссплатформенная библиотека системных вызовов.

## Исходная программа

|| systemCall.cpp

```
1     #include "systemCall.h"
2     #include <iostream>
3
4     #ifndef _WIN32
5 #include <tchar.h>
6 #endif
```



```
8bool PipeCreate(pipeT* pipe) {
9if (!pipe) {
10return false;
11}
12
13#ifndef _WIN32
14SECURITY_ATTRIBUTES sa;
15sa.nLength = sizeof(SECURITY_ATTRIBUTES);
16sa.bInheritHandle = TRUE;
17sa.lpSecurityDescriptor = NULL;
18
19if (!CreatePipe(&pipe->read_end, &pipe->write_end, &sa, 0)) {
20return false;
21}
22
23return true;
24#else
25int fds[2];
26if (::pipe(fds) == -1) {
27return false;
28}
29pipe_ptr->read_end = fds[0];
30pipe_ptr->write_end = fds[1];
31return true;
32#endif
33}
34
35void PipeClose(pipeT* pipe) {
36if (!pipe) {
37return;
38}
39
40#ifndef _WIN32
41if (pipe->read_end != INVALID_PIPE_HANDLE) {
42CloseHandle(pipe->read_end);
43pipe->read_end = INVALID_PIPE_HANDLE;
44}
45if (pipe->write_end != INVALID_PIPE_HANDLE) {
46CloseHandle(pipe->write_end);
47    pipe->write_end = INVALID_PIPE_HANDLE;
```

```
48      }
49      #else
50      if (pipe->read_end != INVALID_PIPE_HANDLE) {
51          close(pipe->read_end);
52          pipe->read_end = INVALID_PIPE_HANDLE;
53      }
54      if (pipe->write_end != INVALID_PIPE_HANDLE) {
55          close(pipe->write_end);
56          pipe->write_end = INVALID_PIPE_HANDLE;
57      }
58      #endiff
59  }
60
61  void PipeCloseWriteEnd(pipeT* pipe) {
62      if (!pipe) {
63          return;
64      }
```



```
5
66#define _WIN32
67if (pipe->write_end != INVALID_PIPE_HANDLE) {
68CloseHandle(pipe->write_end);
69pipe->write_end = INVALID_PIPE_HANDLE;
70}
71#else
72if (pipe->write_end != INVALID_PIPE_HANDLE) {
73close(pipe->write_end);
74pipe->write_end = INVALID_PIPE_HANDLE;
75}
76#endif
77}
78
79process ProcessCreate(const char* program, pipeT* stdin_pipe, pipeT* stdout_pipe) {
80process process_info;
81process_info.is_valid = false;
82
83#define _WIN32
84STARTUPINFOA si;
85ZeroMemory(&si, sizeof(si));
86si.cb = sizeof(si);
87si.dwFlags = STARTF_USESTDHANDLES;
88
89si.hStdInput = stdin_pipe ? stdin_pipe->read_end : GetStdHandle(STD_INPUT_HANDLE);
90si.hStdOutput = stdout_pipe ? stdout_pipe->write_end : GetStdHandle( STD_OUTPUT_HANDLE);
91si.hStdError = GetStdHandle(STD_ERROR_HANDLE); 92
93PROCESS_INFORMATION pi;
94ZeroMemory(&pi, sizeof(pi));
95
96if (CreateProcessA(NULL, (LPSTR)program, NULL, NULL, TRUE, 0, NULL, NULL, &si, &pi )) {
97process_info.process_info = pi;
98process_info.is_valid = true;
99CloseHandle(pi.hThread);
100}
101#else
102pid_t pid = fork();
103    if (pid == 0) {
104        if (stdin_pipe) {
105            dup2(stdin_pipe->read_end, STDIN_FILENO);
```

```
106     close(stdin_pipe->read_end);
107     close(stdin_pipe->write_end);
108 }
109 if (stdout_pipe) {
110     dup2(stdout_pipe->write_end, STDOUT_FILENO);
111     close(stdout_pipe->read_end);
112     close(stdout_pipe->write_end);
113 }
114
115     execl(program, program, NULL);
116     exit(1);
117 } else if (pid > 0) {
118     if (stdin_pipe) {
119         close(stdin_pipe->read_end);
120     }
```

```
121if (stdout_pipe) {  
122close(stdout_pipe->write_end);  
123}
```

```
124
125process_info.pid = pid;
126process_info.is_valid = true;
127}
128#endif
129
130return process_info;
131}
132
133int ProcessTerminate(process* process_info) {
134if (!process_info || !process_info->is_valid) {
135return 0;
136}
137
138#ifndef _WIN32
139TerminateProcess(process_info->process_info.hProcess, 0);
140CloseHandle(process_info->process_info.hProcess);
141return 0;
142#else
143kill(process_info->pid, SIGTERM);
144return 0;
145#endif
146process_info->is_valid = false;
147return 1;
148}
149
150bool ReadStringFromPipe(PIPE_HANDLE pipe, std::string& output) {
151char buffer[1024];
152
153#ifndef _WIN32
154DWORD bytes_read;
155if (ReadFile(pipe, buffer, sizeof(buffer) - 1, &bytes_read, NULL) && bytes_read > 0) {
156buffer[bytes_read] = '\0';
157output = buffer;
158return true;
159}
160#else
161    ssize_t bytes_read = read(pipe, buffer, sizeof(buffer) - 1);
162    if (bytes_read > 0) {
163        buffer[bytes_read] = '\0';

```

```
164     output = buffer;
165     return true;
166 }
167 #endif
168
```

```
169     return false;
170 }
171
172     bool WriteStringToPipe(PIPE_HANDLE pipe, const std::string& input) {
173         if (input.empty()) {
174             return true;
175         }
176
177 #ifdef _WIN32
```

```
178DWORD bytes_written;
179return WriteFile(pipe, input.c_str(), input.length(), &bytes_written, NULL) &&
180bytes_written == input.length();
181#else
182ssize_t bytes_written = write(pipe, input.c_str(), input.length());
183return bytes_written == (ssize_t)input.length();

184#endif 185}
```

## systemCall.h

```
1#ifndef SYSTEMCALL_H
```

```
2#define SYSTEMCALL_H
3
4#include <string>
5
6#ifndef _WIN32
7#include <windows.h>
8#define PIPE_HANDLE HANDLE
9#define INVALID_PIPE_HANDLE INVALID_HANDLE_VALUE
10#else
11#include <unistd.h>
12#include <sys/wait.h>
13#define PIPE_HANDLE int
14#define INVALID_PIPE_HANDLE -1
15#endif
16
17typedef struct {
18PIPE_HANDLE read_end;
19PIPE_HANDLE write_end;
20} pipeT;
21
22typedef struct {
23#ifndef _WIN32
24PROCESS_INFORMATION process_info;
25#else
26pid_t pid;
27#endif
28bool is_valid;
29} process;
30
31bool PipeCreate(pipeT* pipe);
32void PipeClose(pipeT* pipe);
33void PipeCloseWriteEnd(pipeT* pipe);
34
35process ProcessCreate(const char* program, pipeT* stdin_pipe, pipeT* stdout_pipe);
36int ProcessTerminate(process* process_info); 37
38bool ReadStringFromPipe(PIPE_HANDLE pipe, std::string& output);
39bool WriteStringToPipe(PIPE_HANDLE pipe, const std::string& input);
40
41 #endif parent.cpp
```

```
1#include "systemCall.h"
2#include <iostream>
3#include <string>
```

```
||

5int main() {
6pipeT pipe1, pipe2, pipe3;
7process child1, child2;
8
9pipe1.read_end = INVALID_PIPE_HANDLE;
10pipe1.write_end = INVALID_PIPE_HANDLE;
11pipe2.read_end = INVALID_PIPE_HANDLE;
12pipe2.write_end = INVALID_PIPE_HANDLE;
13pipe3.read_end = INVALID_PIPE_HANDLE;
14pipe3.write_end = INVALID_PIPE_HANDLE;
15
16std::cout << "Creating pipes and processes..." << std::endl;
17
18if (!PipeCreate(&pipe1) || !PipeCreate(&pipe2) || !PipeCreate(&pipe3)) {
19std::cerr << "Failed to create pipes" << std::endl;
20return 1;
21}
22
23#ifndef _WIN32
24child1 = ProcessCreate("child1.exe", &pipe1, &pipe2);
25child2 = ProcessCreate("child2.exe", &pipe2, &pipe3);
26#else
27child1 = ProcessCreate("./child1", &pipe1, &pipe2);
28child2 = ProcessCreate("./child2", &pipe2, &pipe3);
29#endif
30
31if (!child1.is_valid || !child2.is_valid) {
32std::cerr << "Failed to create child processes" << std::endl;
33PipeClose(&pipe1);
34PipeClose(&pipe2);
35PipeClose(&pipe3);
36return 1;
37}
38
39std::cout << "Ready. Enter strings (empty line to exit):" << std::endl;
40
41std::string input;
42while (true) {
43std::cout << "> ";
44std::getline(std::cin, input);
45
46if (input.empty()) {
47break;
48}
49
50if (WriteStringToPipe(pipe1.write_end, input + "\n")) {
51// pipe3
```

```
||

52std::string result;
53if (ReadStringFromPipe(pipe3.read_end, result)) {
54std::cout << "Result: " << result;
55        } else {
56            std::cerr << "Failed to read result" << std::endl;
```

57break;

```
||

58}
59} else {
60std::cerr << "Failed to send data" << std::endl;
61break;
62}
63}
64
65std::cout << "Program is ending..." << std::endl;
66
67PipeCloseWriteEnd(&pipe1);
68PipeCloseWriteEnd(&pipe2);
69PipeCloseWriteEnd(&pipe3);
70
71if (child1.is_valid) {
72std::cout << "exit_code for child1 -- " << ProcessTerminate(&child1) << std::endl;
73}
74if (child2.is_valid) {
75std::cout << "exit_code for child2 -- " << ProcessTerminate(&child2) << std::endl;
76}
77
78PipeClose(&pipe1);
79PipeClose(&pipe2);
80PipeClose(&pipe3);
81
82std::cout << "Program finished" << std::endl;
83return 0; 84}
```

### child1.cpp

```
1#include <iostream>
2#include <string>
3#include <algorithm>
4#include <cctype>
5
6int main() {
7std::string line;
8while (std::getline(std::cin, line)) {
9for (char& c : line) {
10c = std::tolower(c);
11}
12std::cout << line << std::endl;
13}
14return 0; 15}
```

### child2.cpp

```
1int main() {
2std::string line;
3while (std::getline(std::cin, line)) {
```

```
4std::string result;
5bool prev_space = false;
6
7        for (char c : line) {
8            if (std::isspace(c)) {
9                if (!prev_space) {
10                    result += c;
11                    prev_space = true;
12                }
13            } else {
14                result += c;
15                prev_space = false;
16            }
17        }
18        std::cout << result << std::endl;
19    }
20    return 0;
21}
```

## Запуск на Linux вывод strace





```
close(3) = 0
mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x75648c0d5000
arch_prctl(ARCH_SET_FS,0x75648c0d63c0) = 0
set_tid_address(0x75648c0d6690) = 3889 set_robust_list(0x75648c0d66a0,24) = 0
rseq(0x75648c0d6d60,0x20,0,0x53053053) = 0 mprotect(0x75648bc16000,16384,PROT_READ) = 0
mprotect(0x75648bdfe000,4096,PROT_READ) = 0 mprotect(0x75648c0f5000,4096,PROT_READ) = 0
mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x75648c0d3000
mprotect(0x75648c01b000,45056,PROT_READ) = 0 mprotect(0x5ff5071d2000,4096,PROT_READ) = 0
mprotect(0x75648c141000,8192,PROT_READ) = 0
prlimit64(0,RLIMIT_STACK,NULL,{rlim_cur=8192*1024,rlim_max=RLIM64_INFINITY}) = 0
munmap(0x75648c0f7000,62143) = 0
getrandom("\x4c\x29\xee\xf2\xde\xe2\xf6\x4e",8,GRND_NONBLOCK) = 8
brk(NULL) = 0x5ff53dada000 brk(0x5ff53dafb000) = 0x5ff53dafb000
futex(0x75648c02977c,FUTEX_WAKE_PRIVATE,2147483647) = 0
newfstatat(1,"",{st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0),...},AT_EMPTY_PATH) = 0
write(1,"Creating pipes and processes...\n",32Creating pipes and processes...
) = 32
pipe2([3,4],0) = 0 pipe2([5,6],0) = 0 pipe2([7,8],0) = 0
clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,child_tid = 3890
) = 3890
close(3) = 0 close(6) = 0
clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,child_tid = 3891
) = 3891
close(5) = 0 close(8) = 0
write(1,"Ready. Enter strings (empty line)...",43Ready. Enter strings (empty line to exit):
) = 43
write(1,>,2>) = 2
newfstatat(0,"",{st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0),...},AT_EMPTY_PATH) = 0
read(0,AAAA NNN 89887 "AAAA NNN 89887\n",1024) = 23
write(4,"AAAANNNN 89887\n",23) = 23 read(7,"aaaa nnn 89887\n",1023) = 15
write(1,"Result: aaaa nnn 89887\n",23Result: aaaa nnn 89887
) = 23
write(1,>,2>) = 2 read(0,
"\n",1024) = 1
write(1,"Program is ending...\n",21Program is ending...
) = 21
close(4) = 0
close(6) файла = -1 EBADF (Неправильный дескриптор
)
close(8) файла = -1 EBADF (Неправильный дескриптор
)
kill(3890,SIGTERM) = 0
---SIGCHLD {si_signo=SIGHLD,si_code=CLD_KILLED,si_pid=3890,si_uid=1000,si_status=SIGT
```

```
--write(1,"exit_code for child1 --0\n",26exit_code for child1 --0
) = 26
kill(3891,SIGTERM)                      = 0
write(1,"exit_code for child2 --0\n",26exit_code for child2 --0
) = 26
close(3) файла)                         = -1 EBADF (Неправильный дескриптор
close(5) файла)                         = -1 EBADF (Неправильный дескриптор

close(7)                                = 0
write(1,"Program finished\n",17Program finished
)      = 17 exit_group(0)    = ?
+++ exited with 0 +++
```