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: К. С. Шульц
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 пересылает их в pipe1. Процессы child1 и child2 производят работу над строками, передавая данные последовательно через pipe2. Child2 пересылает результат своей работы родительскому процессу через pipe3.

Цель работы:

Приобретение практических навыков в: 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 однонаправленную 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 управляется 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 кросс-платформенная библиотека системных вызовов.

Исходная программа

systemCall.cpp

```
#include "systemCall.h"
#include <iostream>

#ifdef _WIN32
#include <tchar.h>
#endif

bool PipeCreate(pipeT* pipe) {
    if (!pipe) {
        return false;
    }

#ifdef _WIN32
    SECURITY_ATTRIBUTES sa;
    sa.nLength = sizeof(SECURITY_ATTRIBUTES);
    sa.bInheritHandle = TRUE;
    sa.lpSecurityDescriptor = NULL;

    if (!CreatePipe(&pipe->read_end, &pipe->write_end, &sa, 0)) {
        return false;
    }

    return true;
#else
    int fds[2];
    if (::pipe(fds) == -1) {
        return false;
    }
    pipe_ptr->read_end = fds[0];
    pipe_ptr->write_end = fds[1];
    return true;
#endif
}

void PipeClose(pipeT* pipe) {
    if (!pipe) {
        return;
    }

#ifdef _WIN32
    if (pipe->read_end != INVALID_PIPE_HANDLE) {
        CloseHandle(pipe->read_end);
        pipe->read_end = INVALID_PIPE_HANDLE;
    }
    if (pipe->write_end != INVALID_PIPE_HANDLE) {
        CloseHandle(pipe->write_end);
        pipe->write_end = INVALID_PIPE_HANDLE;
    }
#else
    if (pipe->read_end != INVALID_PIPE_HANDLE) {
        close(pipe->read_end);
        pipe->read_end = INVALID_PIPE_HANDLE;
    }
    if (pipe->write_end != INVALID_PIPE_HANDLE) {
        close(pipe->write_end);
```

```
56          pipe->write_end = INVALID_PIPE_HANDLE;
57      }
58  #endif
59  }
60
61  void PipeCloseWriteEnd(pipeT* pipe) {
62      if (!pipe) {
63          return;
64      }
65
66  #ifdef _WIN32
67      if (pipe->write_end != INVALID_PIPE_HANDLE) {
68          CloseHandle(pipe->write_end);
69          pipe->write_end = INVALID_PIPE_HANDLE;
70      }
71  #else
72      if (pipe->write_end != INVALID_PIPE_HANDLE) {
73          close(pipe->write_end);
74          pipe->write_end = INVALID_PIPE_HANDLE;
75      }
76  #endif
77  }
78
79  process ProcessCreate(const char* program, pipeT* stdin_pipe, pipeT* stdout_pipe) {
80      process process_info;
81      process_info.is_valid = false;
82
83  #ifdef _WIN32
84      STARTUPINFOA si;
85      ZeroMemory(&si, sizeof(si));
86      si.cb = sizeof(si);
87      si.dwFlags = STARTF_USESTDHANDLES;
88
89      si.hStdInput = stdin_pipe ? stdin_pipe->read_end : GetStdHandle(STD_INPUT_HANDLE);
90      si.hStdOutput = stdout_pipe ? stdout_pipe->write_end : GetStdHandle(
            STD_OUTPUT_HANDLE);
91      si.hStdError = GetStdHandle(STD_ERROR_HANDLE);
92
93      PROCESS_INFORMATION pi;
94      ZeroMemory(&pi, sizeof(pi));
95
96      if (CreateProcessA(NULL, (LPSTR)program, NULL, NULL, TRUE, 0, NULL, NULL, &si, &pi
            )) {
97          process_info.process_info = pi;
98          process_info.is_valid = true;
99          CloseHandle(pi.hThread);
100     }
101 #else
102     pid_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         }
121         if (stdout_pipe) {
122             close(stdout_pipe->write_end);
123         }
124 
125         process_info.pid = pid;
126         process_info.is_valid = true;
127     }
128 #endif
129 
130     return process_info;
131 }
132 
133 int ProcessTerminate(process* process_info) {
134     if (!process_info || !process_info->is_valid) {
135         return 0;
136     }
137 
138 #ifdef _WIN32
139     TerminateProcess(process_info->process_info.hProcess, 0);
140     CloseHandle(process_info->process_info.hProcess);
141     return 0;
142 #else
143     kill(process_info->pid, SIGTERM);
144         return 0;
145 #endif
146     process_info->is_valid = false;
147     return 1;
148 }
149 
150 bool ReadStringFromPipe(PIPE_HANDLE pipe, std::string& output) {
151     char buffer[1024];
152 
153 #ifdef _WIN32
154     DWORD bytes_read;
155     if (ReadFile(pipe, buffer, sizeof(buffer) - 1, &bytes_read, NULL) && bytes_read >
        0) {
156         buffer[bytes_read] = '\0';
157         output = buffer;
158         return 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 
```

```
    return false;
}

bool WriteStringToPipe(PIPE_HANDLE pipe, const std::string& input) {
    if (input.empty()) {
        return true;
    }

#ifdef _WIN32
    DWORD bytes_written;
    return WriteFile(pipe, input.c_str(), input.length(), &bytes_written, NULL) &&
           bytes_written == input.length();
#else
    ssize_t bytes_written = write(pipe, input.c_str(), input.length());
    return bytes_written == (ssize_t)input.length();
#endif
}
```

systemCall.h

```
#ifndef SYSTEMCALL_H
#define SYSTEMCALL_H

#include <string>

#ifdef _WIN32
    #include <windows.h>
    #define PIPE_HANDLE HANDLE
    #define INVALID_PIPE_HANDLE INVALID_HANDLE_VALUE
#else
    #include <unistd.h>
    #include <sys/wait.h>
    #define PIPE_HANDLE int
    #define INVALID_PIPE_HANDLE -1
#endif

typedef struct {
    PIPE_HANDLE read_end;
    PIPE_HANDLE write_end;
} pipeT;

typedef struct {
#ifdef _WIN32
    PROCESS_INFORMATION process_info;
#else
    pid_t pid;
#endif
    bool is_valid;
} process;

bool PipeCreate(pipeT* pipe);
void PipeClose(pipeT* pipe);
void PipeCloseWriteEnd(pipeT* pipe);

process ProcessCreate(const char* program, pipeT* stdin_pipe, pipeT* stdout_pipe);
int ProcessTerminate(process* process_info);

bool ReadStringFromPipe(PIPE_HANDLE pipe, std::string& output);
bool WriteStringToPipe(PIPE_HANDLE pipe, const std::string& input);
```

```

#endif
```

parent.cpp

```cpp
#include "systemCall.h"
#include <iostream>
#include <string>

int main() {
    pipeT pipe1, pipe2, pipe3;
    process child1, child2;

    pipe1.read_end = INVALID_PIPE_HANDLE;
    pipe1.write_end = INVALID_PIPE_HANDLE;
    pipe2.read_end = INVALID_PIPE_HANDLE;
    pipe2.write_end = INVALID_PIPE_HANDLE;
    pipe3.read_end = INVALID_PIPE_HANDLE;
    pipe3.write_end = INVALID_PIPE_HANDLE;

    std::cout << "Creating pipes and processes..." << std::endl;

    if (!PipeCreate(&pipe1) || !PipeCreate(&pipe2) || !PipeCreate(&pipe3)) {
        std::cerr << "Failed to create pipes" << std::endl;
        return 1;
    }

    #ifdef _WIN32
        child1 = ProcessCreate("child1.exe", &pipe1, &pipe2);
        child2 = ProcessCreate("child2.exe", &pipe2, &pipe3);
# else
        child1 = ProcessCreate("./child1", &pipe1, &pipe2);
        child2 = ProcessCreate("./child2", &pipe2, &pipe3);
    #endif

    if (!child1.is_valid || !child2.is_valid) {
        std::cerr << "Failed to create child processes" << std::endl;
        PipeClose(&pipe1);
        PipeClose(&pipe2);
        PipeClose(&pipe3);
        return 1;
    }

    std::cout << "Ready. Enter strings (empty line to exit):" << std::endl;

    std::string input;
    while (true) {
        std::cout << "> ";
        std::getline(std::cin, input);

        if (input.empty()) {
            break;
        }

        if (WriteStringToPipe(pipe1.write_end, input + "\n")) {
            //   pipe3
            std::string result;
            if (ReadStringFromPipe(pipe3.read_end, result)) {
                std::cout << "Result: " << result;
```

```
            } else {
                std::cerr << "Failed to read result" << std::endl;
                break;
            }
        } else {
            std::cerr << "Failed to send data" << std::endl;
            break;
        }
    }

    std::cout << "Program is ending..." << std::endl;

    PipeCloseWriteEnd(&pipe1);
    PipeCloseWriteEnd(&pipe2);
    PipeCloseWriteEnd(&pipe3);

    if (child1.is_valid) {
        std::cout << "exit_code for child1 -- " << ProcessTerminate(&child1) << std::
            endl;
    }
    if (child2.is_valid) {
        std::cout << "exit_code for child2 -- " << ProcessTerminate(&child2) << std::
            endl;
    }

    PipeClose(&pipe1);
    PipeClose(&pipe2);
    PipeClose(&pipe3);

    std::cout << "Program finished" << std::endl;
    return 0;
}
```

child1.cpp

```
#include <iostream>
#include <string>
#include <algorithm>
#include <cctype>

int main() {
    std::string line;
    while (std::getline(std::cin, line)) {
        for (char& c : line) {
            c = std::tolower(c);
        }
        std::cout << line << std::endl;
    }
    return 0;
}
```

child2.cpp

```
int main() {
    std::string line;
    while (std::getline(std::cin, line)) {
        std::string result;
        bool prev_space = false;

```

```
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

```
ksenomorf@ksenVM:~/lab1OS/build/bin$ strace ./parent
execve("./parent",["./parent"],0x7ffc2dfadb80 /* 82 vars */) = 0
brk(NULL)                                       = 0x5ff53dada000
arch_prctl(0x3001 /* ARCH_??? */,0x7fff6c1d1b40) = -1 EINVAL (Недопустимый
аргумент)
mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x75648c107000
access("/etc/ld.so.preload",R_OK)      = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD,"/etc/ld.so.cache",O_RDONLY|O_CLOEXEC) = 3
newfstatat(3,"",{st_mode=S_IFREG|0644,st_size=62143,...},AT_EMPTY_PATH) = 0
mmap(NULL,62143,PROT_READ,MAP_PRIVATE,3,0) = 0x75648c0f7000
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\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) = 0x75648be00000
mprotect(0x75648be9a000,1576960,PROT_NONE) = 0
mmap(0x75648be9a000,1118208,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3
= 0x75648be9a000
mmap(0x75648bfab000,454656,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x1ab000)
= 0x75648bfab000
mmap(0x75648c01b000,57344,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,
= 0x75648c01b000
mmap(0x75648c029000,10432,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,-1
= 0x75648c029000
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\3\0>\0\1\0\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) = 0x75648c0d7000
mmap(0x75648c0da000,94208,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0
= 0x75648c0da000
mmap(0x75648c0f1000,16384,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x1a000)
= 0x75648c0f1000
mmap(0x75648c0f5000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0
= 0x75648c0f5000
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\3\0>\0\1\0\0\0P\237\2\0\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"...,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\0"...,48,848)
= 48
pread64(3,"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O{\f\225\\=\201\327\312\301P\32$\230\266\235
= 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"...,784,64)
= 784
mmap(NULL,2264656,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x75648ba00000
mprotect(0x75648ba28000,2023424,PROT_NONE) = 0
mmap(0x75648ba28000,1658880,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3
= 0x75648ba28000
mmap(0x75648bbbd000,360448,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x1bd000)
= 0x75648bbbd000
mmap(0x75648bc16000,24576,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,
= 0x75648bc16000
mmap(0x75648bc1c000,52816,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,-1
= 0x75648bc1c000
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\3\0>\0\1\0\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) = 0x75648bd19000
mmap(0x75648bd27000,507904,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,
= 0x75648bd27000
mmap(0x75648bda3000,372736,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x8a000)
= 0x75648bda3000
mmap(0x75648bdfe000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0
= 0x75648bdfe000
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_ti
= 3890
close(3)                                = 0
close(6)                                = 0
clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,child_ti
= 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,"AAAA    NNN      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)                                = -1 EBADF (Неправильный дескриптор
файла)
close(8)                                = -1 EBADF (Неправильный дескриптор
файла)
kill(3890,SIGTERM)                      = 0
---SIGCHLD {si_signo=SIGCHLD,si_code=CLD_KILLED,si_pid=3890,si_uid=1000,si_status=SIG
---
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)                                = -1 EBADF (Неправильный дескриптор
файла)
close(5)                                = -1 EBADF (Неправильный дескриптор
файла)
close(7)                                = 0
write(1,"Program finished\n",17Program finished
)      = 17
exit_group(0)                           = ?
+++ exited with 0 +++
```