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. Д. Бадамшин
Группа: М8О-208Б-24
Преподаватель: Е. С. Миронов

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 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: Приобретение практических навыков в управлении процессами в ОС и обеспечении обмена данных между процессами посредством каналов.

Задание: Правило фильтрации: с вероятностью 80 процентов строки отправляются в pipe1, иначе в pipe2. Дочерние процессы удаляют все гласные из строк.

Вариант: 19

Метод решения

Для выполнения задания используется многопроцессная обработка входных данных с последующей записей в разные файлы. Разделение позволяет логически изолировать разные части функциональности: родительский процесс отвечает за получение строк, а дочерний — за обработку и запись в файл.

Ключевые элементы:
parent - родительский процесс, принимает имена файлов, в которые записывать строки, затем сами строки. Создаёт дочерние процессы. Полученные строки с 80 процентной вероятностью перенаправляет в первый дочерний процесс и с 20 процентной вероятностью - во второй;
child1 - первый дочерний процесс, принимает по каналу (pipe) строки, удаляет все латинские гласные, затем записывает полученную строку в файл, название которого ранее было получено по тому же каналу;
child2 - второй дочерний процесс, идентичен первому дочернему процессу;
ProcessManager/Win_ProcessManager - кросс-платформенный обработчик системных вызовов. Для Unix-систем используется ProcessManager, для Windows - Win_ProcessManager

Описание программы

Программа использует объектно-орентированный подход с инкапсуляцией платформозависимых функций, что обеспечивает кросс-платформенность и разделение между модулями:

- *parent.cpp* - программа родительского процесса. Отвечает за получение данных, создание дочерних процессов, передачу данных в них.
- *child1.cpp* - программа первого дочернего процесса. Отвечает за получение данных от родительского процесса, обработку и запись их в файлы.
- *child2.cpp* - программа второго дочернего процесса. Идентична программе первого дочернего процесса.

- *ProcessManager.h* - заголовочный файл с декларациями функций для работы с системными вызовами Unix-систем.
- *ProcessManager.cpp* - реализация функций для файла ProcessManager.h.
- *Win_ProcessManager.h* - заголовочный файл с декларациями функций для работы с системными вызовами Windows.
- *Win_ProcessManager.cpp* - реализация функций для файла Win_ProcessManager.h.

Результаты

В результате выполнения работы была создана программа, полностью соответствующая требованиям задания. Ключевые особенности решения:

- Родительский процесс управляет потоками данных, процессами и взаимодействует с пользователем, а дочерний процесс инкапсулирует логику обработки данных.
- Обеспечено распределение строк по файлам.
- Реализовано применение каналов и перенаправления файловых дескрипторов для корректного взаимодействия между процессами.
- Кросс-платформенная функциональность.
- Синхронизация процессов.

Выводы

Мои знания и практические навыки в области межпроцессного взаимодействия программ значительно улучшены. Полученные умения применены на практике, в следствие чего в ходе лабораторной работы успешно разработана многопроцессная система обработки входных данных.

Исходная программа

```
#ifdef _WIN32
#include "Win_ProcessManager.h"
#elif __linux__
#include "ProcessManager.h"
#endif


int main() {
    try {
        std::string filename1, filename2;

        std::cout << "Hello from ProgramLabOSVar19v1! \nFirst, enter the name of the first file! \nRemember, name can only "
                "consist of latin letters, numbers, and a symbol _!\n"
             << std::endl;
        std::getline(std::cin, filename1);
        if (!IsFileNameGood(filename1)) {
            throw std::invalid_argument("String must consist only 'a'-'Z' && '0'-'9' && '_'!");
        }
        std::cout << "Then enter the name of the second file! The rules are same!\n" << std::endl;
        std::getline(std::cin, filename2);
        if (!IsFileNameGood(filename2)) {
            throw std::invalid_argument("String must consist only 'a'-'Z' && '0'-'9' && '_'!");
        }

        Pipe pip_child1, pip_child2;
#ifdef _WIN32
        char *child1_argv[] = {const_cast<char *>("./child1.exe"), const_cast<char *>(filename1.c_str()), nullptr};
        char *child2_argv[] = {const_cast<char *>("./child2.exe"), const_cast<char *>(filename2.c_str()), nullptr};
#elif __linux__
        char *child1_argv[] = {const_cast<char *>("./child1"), const_cast<char *>(filename1.c_str()), nullptr};
        char *child2_argv[] = {const_cast<char *>("./child2"), const_cast<char *>(filename2.c_str()), nullptr};
#endif

        pid_t pid1, pid2;
#ifdef _WIN32
        HANDLE hPipe1Read = ProcessManager::Get_Handle(pip_child1.Read_fd());
        HANDLE hPipe2Read = ProcessManager::Get_Handle(pip_child2.Read_fd());
        pid1 = ProcessManager::ChangeProcess("./child1.exe", child1_argv, nullptr, hPipe1Read);
        pid2 = ProcessManager::ChangeProcess("./child2.exe", child2_argv, nullptr, hPipe2Read);

#else
        pid1 = ProcessManager::Create_process();
        if (pid1 == 0) {
            pip_child1.Close_write();
            pip_child2.Close_read();
            pip_child2.Close_write();
```

```
47
48              ProcessManager::Dup2(pip_child1.Read_fd(), 0);
49              pip_child1.Close_read();
50              ProcessManager::ChangeProcess("./child1", child1_argv, nullptr);
51          }
52
53          pid2 = ProcessManager::Create_process();
54          if (pid2 == 0) {
55              pip_child2.Close_write();
56              pip_child1.Close_read();
57              pip_child1.Close_write();
58
59              ProcessManager::Dup2(pip_child2.Read_fd(), 0);
60              pip_child2.Close_read();
61              ProcessManager::ChangeProcess("./child2", child2_argv, nullptr);
62          }
63  #endif
64
65          std::cout << "Parent started with id: " << ProcessManager::Get_current_pid() <<
                '\n' << std::endl;
66          pip_child1.Close_read();
67          pip_child2.Close_read();
68
69          std::cout << "Now you can enter lines that are 80 procent likely to end up in
                the first file and 20 procent likely "
70                       "to end up in the second file!\n"
71                    << std::endl;
72          std::cout << "Enter will ending when you enter STOP" << std::endl;
73
74          srand(time(NULL) + ProcessManager::Get_current_pid());
75
76          std::string current_string;
77          std::getline(std::cin, current_string);
78          while (current_string != "STOP") {
79              int res = rand() % 100;
80              if (res < 80) {
81                  size_t length_str = current_string.size();
82                  int bytes_written = write(pip_child1.Write_fd(), &length_str, sizeof(
                        length_str));
83                  if (bytes_written == -1) {
84                      throw std::system_error(errno, std::system_category(), "Error
                            writing length to pipe1");
85                  }
86                  bytes_written = write(pip_child1.Write_fd(), current_string.c_str(),
                        length_str);
87                  if (bytes_written == -1) {
88                      throw std::system_error(errno, std::system_category(), "Error
                            writing string to pipe1");
89                  }
90              } else {
91                  size_t length_str = current_string.size();
92                  int bytes_written = write(pip_child2.Write_fd(), &length_str, sizeof(
                        length_str));
93                  if (bytes_written == -1) {
94                      throw std::system_error(errno, std::system_category(), "Error
                            writing length to pipe2");
95                  }
```

```
                bytes_written = write(pip_child2.Write_fd(), current_string.c_str(),
                    length_str);
                if (bytes_written == -1) {
                    throw std::system_error(errno, std::system_category(), "Error
                        writing string to pipe2");
                }
            }
            std::getline(std::cin, current_string);
        }

        pip_child1.Close_write();
        pip_child2.Close_write();

        ProcessManager::Wait_for_child(pid1);
        ProcessManager::Wait_for_child(pid2);

    } catch (const std::exception &e) {
        std::cerr << "Error: " << e.what() << std::endl;
        return -1;
    }

    return 0;
}
```

Листинг 1: файл parent.cpp

```
#include <system_error>
#ifdef _WIN32
#include "Win_ProcessManager.h"
#elif __linux__
#include "ProcessManager.h"
#endif


int main(int argc, char *argv[]) {
    try {
        std::cout << "Child1 started with id: " << ProcessManager::Get_current_pid() <<
             '\n' << std::endl;
        if (argc != 2) {
            throw std::invalid_argument("Usage: " + std::string(argv[0]) + " <filename>
                ");
        }
        std::string output_filename = argv[1];

        while (true) {
            size_t length_str;
            int bytes_read = read(0, &length_str, sizeof(length_str));

            if (bytes_read == 0) {
                break;
            }
            if (bytes_read == -1) {
                throw std::invalid_argument("Error reading length from stdin");
            }

            std::string received_str;
            received_str.resize(length_str);
```

```
            bytes_read = read(0, &received_str[0], length_str);

            if (bytes_read == -1) {
                throw std::invalid_argument("Error reading length from stdin");
            }
            if (bytes_read != static_cast<int>(length_str)) {
                throw std::invalid_argument("Partial read occurred");
            }

            std::string result_str = RemoveVowels(received_str);
            if (!WriteFile(output_filename, result_str + "\n")) {
                throw std::system_error(errno, std::system_category(), "Error writing
                    to file: " + output_filename);
            }
        }

    } catch (const std::exception &e) {
        std::cerr << "Error: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}
```

Листинг 2: файл child1.cpp (аналогичен child2.cpp)

```
#pragma once

#ifdef __linux__
#include <array>
#include <cstdlib>
#include <ctime>
#include <exception>
#include <fcntl.h>
#include <iostream>
#include <sched.h>
#include <string>
#include <sys/stat.h>
#include <sys/wait.h>
#include <system_error>
#include <unistd.h>

class Pipe {
private:
    std::array<int, 2> fd_;

public:
    Pipe();
    ~Pipe();
    int Read_fd() const;
    int Write_fd() const;
    void Close_read();
    void Close_write();
};

class ProcessManager {
public:
```

```
32      static pid_t Create_process();
33      static void Wait_for_child(pid_t proc);
34      static void Dup2(pid_t fd, int f_num);
35      static void ChangeProcess(const char *filename, char *const argv[], char *const
            envp[] = nullptr);
36      static pid_t Get_current_pid();
37  };
38
39  bool IsFileNameGood(const std::string &st);
40  std::string RemoveVowels(const std::string &str);
41  bool WriteFile(const std::string &filename, const std::string &content);
42  #endif
```

Листинг 3: файл ProcessManager.h

```
1   #include "ProcessManager.h"
2
3   Pipe::Pipe() {
4       if (pipe(fd_.data()) == -1) {
5           throw std::system_error(errno, std::system_category(), "pipe creation failed");
6       }
7   }
8
9   Pipe::~Pipe() {
10      if (fd_[0] != -1) {
11          close(fd_[0]);
12      }
13      if (fd_[1] != -1) {
14          close(fd_[1]);
15      }
16  }
17
18  int Pipe::Read_fd() const { return fd_[0]; }
19
20  int Pipe::Write_fd() const { return fd_[1]; }
21
22  void Pipe::Close_read() {
23      if (fd_[0] != -1) {
24          close(fd_[0]);
25          fd_[0] = -1;
26      }
27  }
28
29  void Pipe::Close_write() {
30      if (fd_[1] != -1) {
31          close(fd_[1]);
32          fd_[1] = -1;
33      }
34  }
35
36  pid_t ProcessManager::Create_process() {
37      pid_t pid = fork();
38      if (pid == -1) {
39          throw std::system_error(errno, std::system_category(), "fork failed");
40      }
41      return pid;
42  }
```

```
43
44 void ProcessManager::Wait_for_child(pid_t proc) {
45     int status;
46     waitpid(proc, &status, 0);
47 }
48
49 void ProcessManager::Dup2(pid_t fd, int f_num) { dup2(fd, f_num); }
50
51 void ProcessManager::ChangeProcess(const char *filename, char *const argv[], char *
       const envp[]) {
52     execve(filename, argv, envp);
53 }
54
55 pid_t ProcessManager::Get_current_pid() { return getpid(); }
56
57 bool IsFileNameGood(const std::string &st) {
58     if (st.empty()) {
59         return false;
60     }
61     if (st.find_first_not_of("
           abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_") != std::
           string::npos) {
62         return false;
63     }
64     return true;
65 }
66
67 std::string RemoveVowels(const std::string &str) {
68     std::string new_str = "";
69     std::string pattern = "aeiouyAEIOUY";
70     for (char chr : str) {
71         if (pattern.find(chr) == std::string::npos) {
72             new_str += chr;
73         }
74     }
75     return new_str;
76 }
77
78 bool WriteFile(const std::string &filename, const std::string &content) {
79     int fd = open(filename.c_str(), O_CREAT | O_WRONLY | O_APPEND, 0644);
80     if (fd == -1) {
81         perror("open");
82         return false;
83     }
84     int bytes_written = write(fd, content.c_str(), content.size());
85     if (bytes_written == -1) {
86         perror("write");
87         close(fd);
88         return false;
89     }
90     if (bytes_written != static_cast<int>(content.size())) {
91         std::cerr << "Partial write" << std::endl;
92         close(fd);
93         return false;
94     }
95     if (close(fd) == -1) {
96         perror("close");
97         return false;
```

```
98      }
99      return true;
100 }
```

Листинг 4: файл ProcessManager.cpp