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»

Группа: М8О-212БВ-24

Студент: Федосов Д.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 04.10.25

Москва, 2025

Постановка задачи

Вариант 18.

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.

Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы удаляют все гласные из строк.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int pipe(int *fd); – создает канал для однонаправленной передачи данных между процессами.
- close(int *fd); - используется для закрытия файлового дескриптора.
- dup2(int *fd1, int *fd2); - создает второй дескриптор файла для открытого файла.
- exit(int status); - завершение выполнения процесса и возвращение статуса.
- int execve(const char *filename, char *const argv[], char *const envp[]) - замена образа памяти процесса

Алгоритм решения:

A. Parent

1. Получение пути к директории исполняемого файла
2. Чтение имен файлов для дочерних процессов из стандартного ввода
3. Создание двух каналов (pipe):
4. Fork() первого процесса child1:
5. Закрытие неиспользуемых концов каналов
6. Перенаправление stdin на pipe1 через dup2()
7. Запуск программы child1 с передачей имени файла
8. Fork() второго процесса child2:
9. Закрытие неиспользуемых концов каналов
10. Перенаправление stdin на pipe2 через dup2()
11. Запуск программы child2 с передачей имени файла
12. Чтение строк произвольной длины из стандартного ввода
13. Применение правила фильтрации:
14. Нечетные строки → отправка в pipe1 (child1)
15. Четные строки → отправка в pipe2 (child2)
16. Завершение при вводе пустой строки
17. Родитель закрывает каналы записи
18. Дочерние процессы получают EOF и завершаются
19. Родитель ожидает завершения детей через wait()

B. Child

1. Получение имени файла через аргументы командной строки
2. Открытие файла для записи
3. Чтение данных из перенаправленного stdin (из pipe)
4. Удаление всех гласных букв
5. Запись результата в соответствующий файл

Код программы

parent.c

```c
#include <stdint.h>
#include <stdbool.h>

#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>

static char CHILD1_PROGRAM_NAME[] = "child1";
static char CHILD2_PROGRAM_NAME[] = "child2";

int main(int argc, char* argv[]) {
   // Получение директории файла

   char progpath[1024];
   {
       ssize_t len = readlink("/proc/self/exe", progpath,
                              sizeof(progpath) - 1);
       if (len == -1) {
           const char msg[] = "error: failed to read full program path\n";
           write(STDERR_FILENO, msg, sizeof(msg));
           exit(EXIT_FAILURE);
       }

       while (progpath[len] != '/')
```

```
        --len;

    progpath[len] = '\0';
}

// Считывание имён для файлов

char filePath1[512];
char filePath2[512];

fgets(filePath1, 512, stdin);
filePath1[strcspn(filePath1, "\n")] = '\0';

fgets(filePath2, 512, stdin);
filePath2[strcspn(filePath2, "\n")] = '\0';

// Создание pipe'ов

int pipe1[2]; // К child1
if (pipe(pipe1) == -1) {
    const char msg[] = "error: failed to create pipe\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
}

int pipe2[2]; // К child2
if (pipe(pipe2) == -1) {
    const char msg[] = "error: failed to create pipe\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
}

// Процесс child1

const pid_t child1 = fork();
```

```
    switch (child1) {
    case -1: {
        const char msg[] = "error: failed to spawn new process\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    } break;

    case 0: {
        {
            close(pipe1[1]);
            dup2(pipe1[0], STDIN_FILENO);
            close(pipe1[0]);

            close(pipe2[0]);
            close(pipe2[1]);

            char path[1024];
            snprintf(path, sizeof(path) - 1, "%s/%s", progpath,
CHILD1_PROGRAM_NAME);

            char *const args[] = {CHILD1_PROGRAM_NAME, filePath1, NULL};

            int32_t status = execv(path, args);

            if (status == -1) {
                const char msg[] = "error: failed to exec into new exectuable
image\n";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            }

        }

    } break;
```

```
    }


    // Процесс child2


    const pid_t child2 = fork();


    switch (child2) {
    case -1: {
        const char msg[] = "error: failed to spawn new process\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    } break;


    case 0: {
        {
            close(pipe2[1]);
            dup2(pipe2[0], STDIN_FILENO);
            close(pipe2[0]);


            close(pipe1[0]);
            close(pipe1[1]);


            char path[1024];
            snprintf(path, sizeof(path) - 1, "%s/%s", progpath,
CHILD2_PROGRAM_NAME);


            char *const args[] = {CHILD2_PROGRAM_NAME, filePath2, NULL};


            int32_t status = execv(path, args);


            if (status == -1) {
                const char msg[] = "error: failed to exec into new exectuable
image\n";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            }
```

```
    }

} break;
}

close(pipe1[0]);
close(pipe2[0]);

// Обработка ввода пользователя

char buf[4096];
int counter = 0;
ssize_t bytes;
while (bytes = read(STDIN_FILENO, buf, sizeof(buf))) {
    if (bytes < 0) {
        const char msg[] = "error: failed to read from stdin\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    } else if (buf[0] == '\n') {
        break;
    }

    counter++;

    if (counter % 2 == 1) {
        write(pipe1[1], buf, bytes);
    } else {
        write(pipe2[1], buf, bytes);
    }
}

close(pipe1[1]);
close(pipe2[1]);
```

```
    // Ожидание дочерних процессов

    wait(NULL);
}
```

child1.c

```
#include <stdint.h>
#include <stdbool.h>
#include <ctype.h>

#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char* argv[]) {

    // Получение директории файла

    char filePath[512];
    strcpy(filePath, argv[1]);

    // Открытие файла для записи

    int32_t file = open(filePath, O_WRONLY | O_CREAT | O_TRUNC | O_APPEND,
0600);
    if (file == -1) {
        const char msg[] = "error: failed to open requested file\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    // Обработка строк
```

```
    char buf[4096];
    char outBuf[4096];
    ssize_t bytes;
    const char banwords[] = {'a', 'e', 'i', 'o', 'u', 'y', 'A', 'E', 'I',
'O', 'U', 'Y'};

    while (bytes = read(STDIN_FILENO, buf, sizeof(buf))) {
        if (bytes < 0) {
            const char msg[] = "error: failed to read from stdin\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }

        int anotherCounter = 0;
        for (uint32_t i = 0; i < bytes; ++i) {
            if (memchr(banwords, buf[i], 12) == NULL) {
                outBuf[anotherCounter++] = buf[i];
            }
        }

        if (anotherCounter > 0) {
            write(file, outBuf, anotherCounter);
        }
    }

    close(file);
}
```

child2.c

```
#include <stdint.h>
#include <stdbool.h>
#include <ctype.h>

#include <stdlib.h>
#include <unistd.h>
```

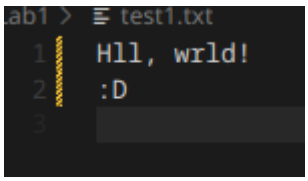
```c
#include <fcntl.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char* argv[]) {

    // Получение директории файла

    char filePath[512];
    strcpy(filePath, argv[1]);

    // Открытие файла для записи

    int32_t file = open(filePath, O_WRONLY | O_CREAT | O_TRUNC | O_APPEND,
0600);
    if (file == -1) {
        const char msg[] = "error: failed to open requested file\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    // Обработка строк

    char buf[4096];
    char outBuf[4096];
    ssize_t bytes;
    const char banwords[] = {'a', 'e', 'i', 'o', 'u', 'y', 'A', 'E', 'I',
'O', 'U', 'Y'};

    while (bytes = read(STDIN_FILENO, buf, sizeof(buf))) {
        if (bytes < 0) {
            const char msg[] = "error: failed to read from stdin\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }
```

```
        int anotherCounter = 0;
        for (uint32_t i = 0; i < bytes; ++i) {
            if (memchr(banwords, buf[i], 12) == NULL) {
                outBuf[anotherCounter++] = buf[i];
            }
        }

        if (anotherCounter > 0) {
            write(file, outBuf, anotherCounter);
        }
    }

    close(file);
}
```

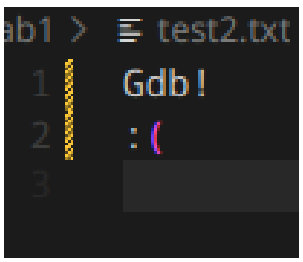
Протокол работы программы

```
snadon@fedora:~/OS-Labs/Lab1$ ./parent
test1.txt
test2.txt
Hello, world!
Goodbye!
:D
:(

snadon@fedora:~/OS-Labs/Lab1$
```

test1.txt:



test2.txt:



Вывод

В процессе выполнения лабораторной работы я составил программу, осуществляющую работу с процессами и взаимодействие между ними. Я приобрел базовые практические навыки в управлении процессами в ОС и обеспечении обмена между процессами посредством каналов. Одной из основных сложностей была в понимании принципа работы процессов и системных вызовов.