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-211БВ-24

Студент: Николич С.Р.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 09.10.25

Москва, 2025

Постановка задачи

Вариант 20.

Группа вариантов 5: 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.

Вариант 20) Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы инвертируют строки.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void) – создает дочерний процесс. После вызова оба процесса выполняют один и тот же код, но в родительском процессе возвращается PID ребенка, а в дочернем процессе возвращается 0.
- int pipe(int fd[2]) – создает неименованный канал для межпроцессного взаимодействия. fd[0] - для чтения, fd[1] - для записи. Данные, записанные в fd[1], можно прочитать из fd[0].
- int dup2(int oldfd, int newfd) – копирует файловый дескриптор oldfd в newfd. Если newfd был открыт, он сначала закрывается. Используется для перенаправления ввода/вывода.
- ssize_t read(int fd, void *buf, size_t count) – читает до count байтов из файлового дескриптора fd в буфер buf. Возвращает количество прочитанных байтов.
- ssize_t write(int fd, const void *buf, size_t count) – записывает до count байтов из буфера buf в файловый дескриптор fd.
- int open(const char *pathname, int flags, mode_t mode) – открывает файл. flags определяет режим доступа (O_RDONLY, O_WRONLY, O_RDWR), а также дополнительные флаги (O_CREAT, O_TRUNC). mode задает права доступа к файлу при создании.
- int close(int fd) – закрывает файловый дескриптор, освобождая ресурсы системы.
- pid_t waitpid(pid_t pid, int *status, int options) – ожидает завершения указанного дочернего процесса и получает информацию о его завершении.
- int execl(const char *path, const char *arg, ...) – заменяет текущий образ процесса новым образом из указанного исполняемого файла.

В рамках лабораторной работы программа реализует многопроцессную обработку строк с использованием межпроцессного взаимодействия через каналы pipe. Родительский процесс создает два дочерних процесса и два канала pipe. Пользователь вводит имена файлов для каждого дочернего процесса. Родительский процесс читает строки из стандартного ввода, анализирует их длину и отправляет в соответствующий канал: короткие строки (≤10 символов) в pipe1 для child1, длинные строки (>10 символов) в pipe2 для child2. Дочерние процессы перенаправляют свой стандартный ввод на соответствующий канал чтения с помощью dup2, затем читают строки из pipe, инвертируют их и записывают результаты в указанные файлы. Когда пользователь завершает ввод (Ctrl+D),

родительский процесс закрывает каналы записи, что сигнализирует дочерним процессам о завершении данных. Дочерние процессы завершают работу после чтения всех данных, а родительский процесс ожидает их завершения с помощью waitpid перед своим завершением.

Код программы

parent.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <fcntl.h>

#define MAX_LENGTH 256

int main() {
    int pipe1[2];
    int pipe2[2];

    if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {
        perror("pipe failed");
        exit(EXIT_FAILURE);
    }

    char filename1[MAX_LENGTH], filename2[MAX_LENGTH];

    printf("Введите имя файла для child1: ");
    if (fgets(filename1, MAX_LENGTH, stdin) == NULL) {
        perror("fgets failed");
        exit(EXIT_FAILURE);
    }
    filename1[strcspn(filename1, "\n")] = 0;

    printf("Введите имя файла для child2: ");
    if (fgets(filename2, MAX_LENGTH, stdin) == NULL) {
        perror("fgets failed");
        exit(EXIT_FAILURE);
    }
    filename2[strcspn(filename2, "\n")] = 0;

    pid_t child1 = fork();

    if (child1 == -1) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    }

    if (child1 == 0) {
        close(pipe1[1]);
        close(pipe2[0]);
        close(pipe2[1]);
```

```
    dup2(pipe1[0], STDIN_FILENO);
    close(pipe1[0]);

    execl("./child1", "child1", filename1, NULL);
    perror("execl failed");
    exit(EXIT_FAILURE);
}

pid_t child2 = fork();

if (child2 == -1) {
    perror("fork failed");
    exit(EXIT_FAILURE);
}

if (child2 == 0) {
    close(pipe2[1]);
    close(pipe1[0]);
    close(pipe1[1]);

    dup2(pipe2[0], STDIN_FILENO);
    close(pipe2[0]);

    execl("./child2", "child2", filename2, NULL);
    perror("execl failed");
    exit(EXIT_FAILURE);
}

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

char buffer[MAX_LENGTH];

printf("Вводите строки (Ctrl+D для завершения):\n");

while (fgets(buffer, MAX_LENGTH, stdin) != NULL) {
    buffer[strcspn(buffer, "\n")] = 0;

    if (strlen(buffer) > 10) {
        write(pipe2[1], buffer, strlen(buffer));
        write(pipe2[1], "\n", 1);
    } else {
        write(pipe1[1], buffer, strlen(buffer));
        write(pipe1[1], "\n", 1);
    }
}

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

waitpid(child1, NULL, 0);
waitpid(child2, NULL, 0);

printf("Родительский процесс завершен.\n");
```

```
    return 0;
}
```

child1.c

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

#define MAX_LENGTH 256

void invert_string(char *str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        char temp = str[i];
        str[i] = str[len - i - 1];
        str[len - i - 1] = temp;
    }
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s filename\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    int file = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0600);
    if (file == -1) {
        perror("open failed");
        exit(EXIT_FAILURE);
    }

    char buffer[MAX_LENGTH];
    ssize_t bytes_read;

    printf("Child1: начало обработки коротких строк в файл %s\n", argv[1]);

    while ((bytes_read = read(STDIN_FILENO, buffer, MAX_LENGTH - 1)) > 0) {
        buffer[bytes_read] = '\0';

        char *line = buffer;
        char *end;

        while ((end = strchr(line, '\n')) != NULL) {
            *end = '\0';

            if (strlen(line) > 0) {
                char inverted[MAX_LENGTH];
                strcpy(inverted, line);
                invert_string(inverted);

                write(file, inverted, strlen(inverted));
```

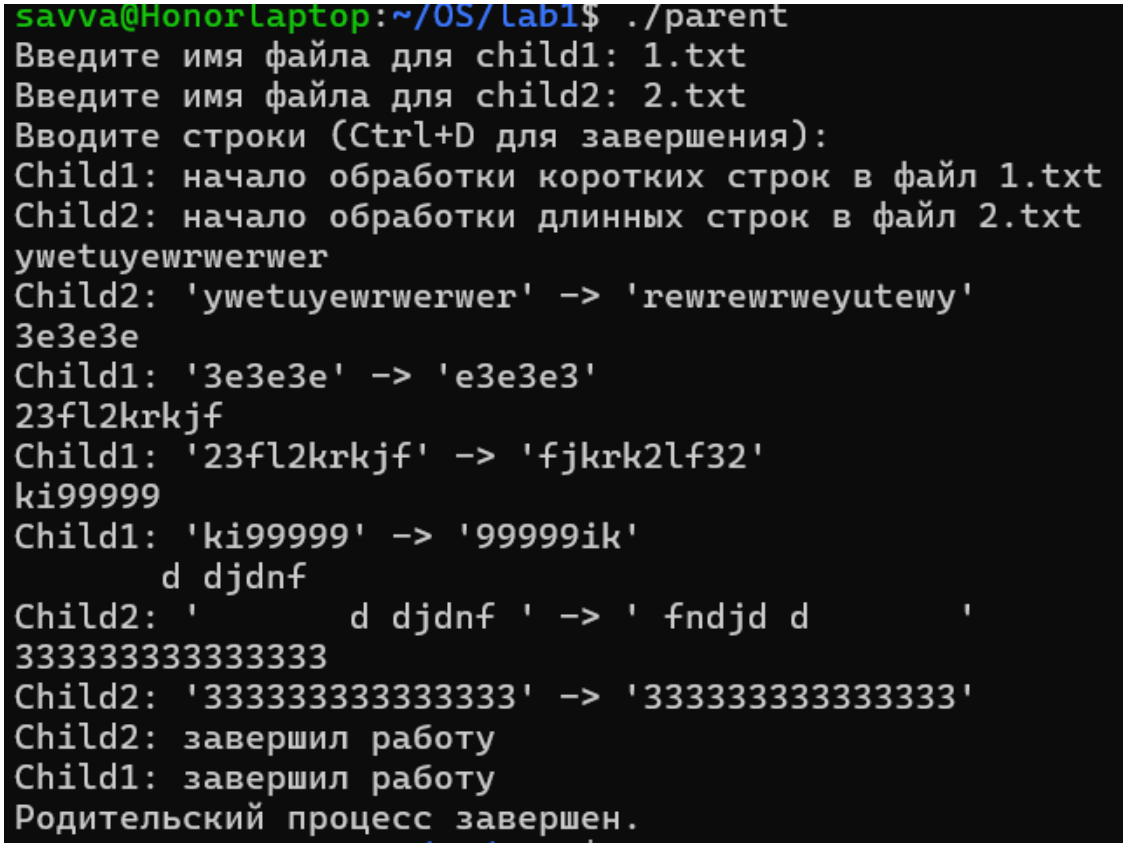
```
                write(file, "\n", 1);

                printf("Child1: '%s' -> '%s'\n", line, inverted);
            }

            line = end + 1;
        }
    }

    close(file);
    printf("Child1: завершил работу\n");
    return 0;
}
```

child2.c

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

#define MAX_LENGTH 256

void invert_string(char *str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        char temp = str[i];
        str[i] = str[len - i - 1];
        str[len - i - 1] = temp;
    }
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s filename\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    int file = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0600);
    if (file == -1) {
        perror("open failed");
        exit(EXIT_FAILURE);
    }

    char buffer[MAX_LENGTH];
    ssize_t bytes_read;

    printf("Child2: начало обработки длинных строк в файл %s\n", argv[1]);

    while ((bytes_read = read(STDIN_FILENO, buffer, MAX_LENGTH - 1)) > 0) {
        buffer[bytes_read] = '\0';
```

```
        char *line = buffer;
        char *end;

        while ((end = strchr(line, '\n')) != NULL) {
            *end = '\0';

            if (strlen(line) > 0) {
                char inverted[MAX_LENGTH];
                strcpy(inverted, line);
                invert_string(inverted);

                write(file, inverted, strlen(inverted));
                write(file, "\n", 1);

                printf("Child2: '%s' -> '%s'\n", line, inverted);
            }

            line = end + 1;
        }
    }

    close(file);
    printf("Child2: завершил работу\n");
    return 0;
}
```

Протокол работы программы



Вывод

В данной лабораторной работе я освоил управление процессами в операционной системе и научился организовывать между ними взаимодействие. Была реализована многопроцессная архитектура, где родительский процесс распределяет данные между дочерними процессами через механизм каналов (pipe). Каждый дочерний процесс выполнял свою задачу по обработке строк независимо, что позволило организовать параллельную обработку данных. В ходе работы я также научился правильно синхронизировать процессы и обрабатывать системные ошибки, возникающие при межпроцессном взаимодействии.