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-210БВ-24

Студент: Телепнева А.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 01.10.25

Москва, 2025

Постановка задачи

Вариант 19.

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int pipe(int fd[2]); – создает неименованный канал для межпроцессного взаимодействия.
- ssize_t read(int fd, void *buf, size_t count); - читает данные из файлового дескриптора.
- ssize_t write(int fd, const void *buf, size_t count); - записывает данные в файловый дескриптор.
- int dup2(int oldfd, int newfd); - создает копию файлового дескриптора для перенаправления стандартного ввода/вывода.
- int execl(const char *path, const char *arg0, …, NULL); - загружает новую программу в текущий процесс (заменяет образ процесса).
- pid_t wait(int *status); - ожидает завершения дочернего процесса.
- int open(const char *pathname, int flags, mode_t mode); – открывает файл для записи результатов.
- int close(int fd); – закрывает файловый дескриптор.

Описание работы программы:
Алгоритм работы родительского процесса (server.c):

1. Инициализация:
    - Создаются два неименованных канала (pipe1 и pipe2) для передачи данных дочерним процессам.
    - Инициализируется генератор псевдослучайных чисел (srand(time(NULL)))
2. Создание дочерних процессов:
    - С помощью системного вызова fork() создаются два дочерних процесса.
    - Для каждого дочернего процесса выполняется перенаправление стандартного ввода (STDIN) на соответствующий канал с использованием dup2().
3. Обработка пользовательского ввода
    - Родительский процесс считывает строки из стандартного ввода.
    - Для каждой строки генерируется случайное число от 0 до 99.
    - С вероятностью 80% строка передаётся в первый канал (pipe1), с вероятностью 20% — во второй канал (pipe2).
4. Завершение работы:
    - После получения EOF (конца ввода) закрываются концы каналов для записи.
    - Родительский процесс ожидает завершения обоих дочерних процессов с помощью wait(NULL).

Алгоритм работы дочернего процесса (client.c):

1. Инициализация:
    - Дочерний процесс получает имя выходного файла через аргументы командной строки.
2. Открытие файла:
    - С помощью open() с флагами O_WRONLY | O_CREAT | O_TRUNC создается или очищается файл для записи результатов.
3. Обработка данных:
    - Процесс читает данные из стандартного ввода, перенаправленного из канала.
    - Из каждой полученной строки удаляются все гласные буквы (как строчные, так и заглавные).
4. Вывод результатов:
    - Обработанные строки записываются в выходной файл с помощью write().
5. Завершение работы:
    - При получении признака конца файла (EOF) файл закрывается, и процесс завершает работу.

Код программы

server.c

```
#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>


#define BUF_SIZE 4096


int main(void) {

    int pipe1[2], pipe2[2];


    if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {

        write(STDERR_FILENO, "pipe failed\n", 12);

        exit(EXIT_FAILURE);

    }
```

```
srand(time(NULL));

pid_t child1 = fork();
if (child1 == -1) {
    write(STDERR_FILENO, "fork failed\n", 12);
    exit(EXIT_FAILURE);
}

if (child1 == 0) {
    close(pipe1[1]);
    dup2(pipe1[0], STDIN_FILENO);
    close(pipe1[0]);

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

    execl("./client", "child", "out1.txt", NULL);
    write(STDERR_FILENO, "exec failed\n", 12);
    exit(EXIT_FAILURE);
}

pid_t child2 = fork();
if (child2 == -1) {
    write(STDERR_FILENO, "fork failed\n", 12);
    exit(EXIT_FAILURE);
}

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

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

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

    execl("./client", "child", "out2.txt", NULL);
    write(STDERR_FILENO, "exec failed\n", 12);
    exit(EXIT_FAILURE);
}

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

char buf[BUF_SIZE];
ssize_t bytes;

while ((bytes = read(STDIN_FILENO, buf, sizeof(buf))) > 0) {
    int r = rand() % 100;

    if (r < 80) {
        write(pipe1[1], buf, bytes);
    } else {
        write(pipe2[1], buf, bytes);
    }
}

if (bytes < 0) {
    write(STDERR_FILENO, "read error\n", 11);
}
```

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

    wait(NULL);
    wait(NULL);

    return 0;
}
```

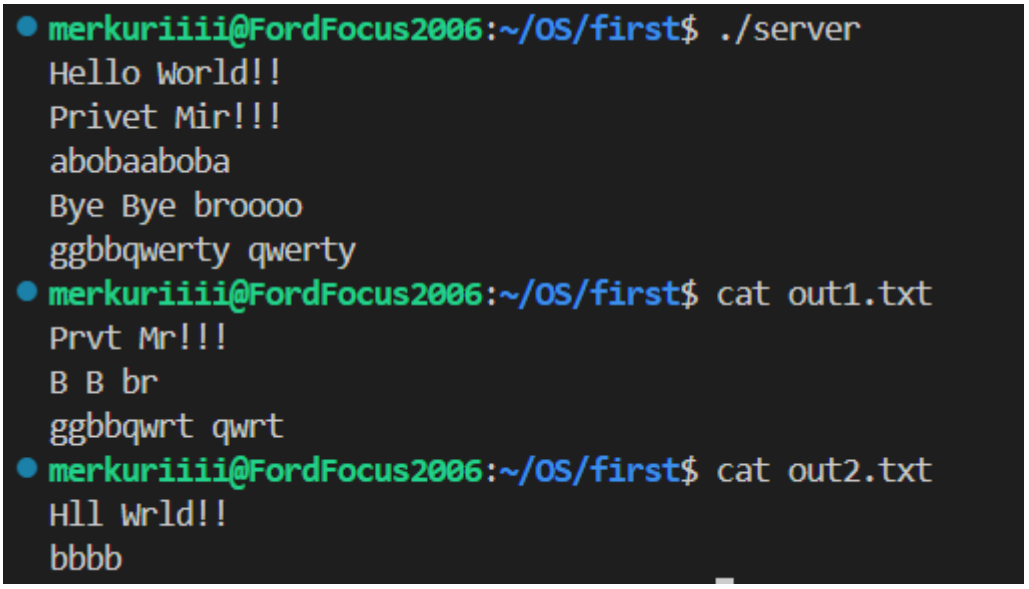
client.c

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

#define BUF_SIZE 4096

int is_vowel(char c) {
    const char *vowels = "aeiouyAEIOUY";
    return strchr(vowels, c) != NULL;
}

int main(int argc, char **argv) {
    if (argc != 2) {
        write(STDERR_FILENO, "Usage: child <output_file>\n", 27);
        exit(EXIT_FAILURE);
    }

    int file = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0644);
```

```
    if (file == -1) {
        write(STDERR_FILENO, "open failed\n", 12);
        exit(EXIT_FAILURE);
    }

    char buf[BUF_SIZE];
    char out[BUF_SIZE];
    ssize_t bytes;

    while ((bytes = read(STDIN_FILENO, buf, sizeof(buf))) > 0) {
        int j = 0;
        for (int i = 0; i < bytes; i++) {
            if (!is_vowel(buf[i])) {
                out[j++] = buf[i];
            }
        }
        write(file, out, j);
    }

    if (bytes < 0) {
        write(STDERR_FILENO, "read error\n", 11);
    }

    close(file);
    return 0;
}
```

Протокол работы программы

Тестирование:



Вывод

Программа успешно реализует межпроцессное взаимодействие через неименованные каналы, демонстрируя параллельную обработку данных двумя клиентами с вероятностным распределением нагрузки. Основные сложности возникли с синхронизацией закрытия каналов и обработкой завершения дочерних процессов.