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: Мутагиров Т.Р.

Преподаватель: Бахарев В.Д. (ФИИТ)

Оценка: _____

Дата: 24.09.25

Москва, 2025

Постановка задачи

Вариант 14.

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

Child1 переводит строки в нижний регистр. Child2 убирает все задвоенные пробелы.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int pipe(int *fd); – создает канал между двумя файловыми дескрипторами. Возвращает -1, если возникла ошибка при создании. Заполняет массив fd.
  - fd[0] – файловый дескриптор для чтения
  - fd[1] – файловый дескриптор для записи
- ssize_t readlink(char *path, char* buf, ssize_t bufsiz); - считывает содержание символической ссылки и записывает в buf.
- int write(int fd, const void *buf, size_t count); – записывает данные из буфера по файловому дескриптору.
- int read(int fd, void* buf, size_t count); - читает данные из файла по файловому дескриптору и записывает в buf.
- int execv(const char *path, char *const argv[]); - заменяет текущий процесс новым процессом, загружая и выполняя указанную программу
- pid_t wait(int *wstatus); - ожидает завершения любого из дочерних процессов, возвращает информацию о его завершении
- pid_t waitpid(pid_t pid, int *wstatus, int options); - ожидает завершения конкретного дочернего процесса.

Я реализовал межпроцессорное взаимодействие с помощью системных вызовов. Есть родительский процесс, который порождает два дочерних процесса. Первый преобразует все символы в нижний регистр, а второй удаляет все сдвоенные пробелы. Общаются между собой процессы с помощью канала, созданным функцией pipe. Пользователь общается только с родительским процессом.

Код программ

client.c

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

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

static char CHILD_PROGRAM_NAME[] = "child";

int main() {
    char progpath[1024];
    {
        // NOTE: Read full program path, including its name
        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);
        }
```

```
        // NOTE: Trim the path to first slash from the end
        while (progpath[len] != '/')
            --len;

        progpath[len] = '\0';
    }
    // parent -> child1
    int pipe1[2];
    if(pipe(pipe1) == -1) {
        const char msg[] = "error: failed to create pipe\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
    // child2 -> parent
    int pipe2[2];
    if(pipe(pipe2) == -1) {
        const char msg[] = "error: failed to create pipe\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
    // child1 -> child2
    int pipe3[2];
    if(pipe(pipe3) == -1) {
        const char msg[] = "error: failed to create pipe\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
    const pid_t child1 = fork();
    if(child1 == -1) {
        const char msg[] = "error: failed to spawn new process\n";
        write(STDERR_FILENO, msg, sizeof(msg));
```

```
            exit(EXIT_FAILURE);
        } else if(child1 == 0) {
            {
                pid_t pid = getpid(); // NOTE: Get child PID

                char msg[64];
                const int32_t length = snprintf(msg, sizeof(msg),
                    "%d: I'm a child\n", pid);
                write(STDOUT_FILENO, msg, length);
            }
            close(pipe2[0]);
            close(pipe2[1]);
            dup2(pipe1[0], STDIN_FILENO);
            close(pipe1[0]);
            close(pipe1[1]);
            dup2(pipe3[1], STDOUT_FILENO);
            close(pipe3[1]);
            close(pipe3[0]);
            {
                char path[1024];
                snprintf(path, sizeof(path) - 1, "%s/%s", progpath, CHILD_PROGRAM_NAME);
                const char *argw[] = {CHILD_PROGRAM_NAME, "1", NULL};
                int32_t status = execv(path, argw);
                if(status == -1) {
                    const char msg[] = "error: failed to exec into new exectuable image\n";
                    write(STDERR_FILENO, msg, sizeof(msg));
                    exit(EXIT_FAILURE);
                }
            }
            exit(EXIT_SUCCESS);
        }
```

```
        const pid_t child2 = fork();
        if(child2 == -1) {
            const char msg[] = "error: failed to spawn new process\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        } else if(child2 == 0) {
            {
                pid_t pid = getpid(); // NOTE: Get child PID


                char msg[64];
                const int32_t length = snprintf(msg, sizeof(msg),
                    "%d: I'm a child\n", pid);
                write(STDOUT_FILENO, msg, length);
            }
            close(pipe1[0]);
            close(pipe1[1]);
            dup2(pipe3[0], STDIN_FILENO);
            close(pipe3[0]);
            close(pipe3[1]);
            dup2(pipe2[1], STDOUT_FILENO);
            close(pipe2[0]);
            close(pipe2[1]);
            {
                char path[1024];
                snprintf(path, sizeof(path) - 1, "%s/%s", progpath, CHILD_PROGRAM_NAME);
                const char *argw[] = {CHILD_PROGRAM_NAME, "0", NULL};
                int32_t status = execv(path, argw);
                if(status == -1) {
                    const char msg[] = "error: failed to exec into new exectuable image\n";
                    write(STDERR_FILENO, msg, sizeof(msg));
                    exit(EXIT_FAILURE);
```

```
            }
        }
        exit(EXIT_SUCCESS);
    }
    printf("I am client\n");
    // client
    close(pipe3[0]);
    close(pipe3[1]);
    close(pipe2[1]);
    close(pipe1[0]);
    char buf[4096];
    ssize_t sz;
    while(sz = read(STDIN_FILENO, buf, sizeof(buf))) {
        if(sz < 0) {
            const char msg[] = "error: failed to read from stdin\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }
        write(pipe1[1], buf, sz);
        sz = read(pipe2[0], buf, sizeof(buf)); // ?
        write(STDOUT_FILENO, buf, sz);
    }
    close(pipe1[1]);
    close(pipe2[0]);
    wait(NULL);
    return 0;
}
```

child.c

```c
#include <stdint.h>

#include <stdbool.h>

#include <ctype.h>



#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <stdio.h>



int main(int argc, const char **argv) {

    bool first = argv[1][0] - '0';

    char buf[4096];

    ssize_t sz;

    while(sz = read(STDIN_FILENO, buf, sizeof(buf))) {

        if(sz < 0) {

            const char msg[] = "error: failed to read from stdin\n";

            write(STDERR_FILENO, msg, sizeof(msg));

            exit(EXIT_FAILURE);

        }

        // преобразовать buf

        if(first) { // child 1

            for(size_t i = 0; i < sz; ++i) buf[i] = tolower(buf[i]);

            write(STDOUT_FILENO, buf, sz);

        } else { // child 2

            char nbuf[4096];

            size_t nsz = 0;

            for(size_t i = 0; i < sz; ++i) {

                if(buf[i] != ' ' || i == sz - 1 || buf[i + 1] != ' ') {

                    nbuf[nsz] = buf[i];

                    nsz++;
```

```
                }

            }

            write(STDOUT_FILENO, nbuf, nsz);

        }

    }

    return 0;

}
```

Протокол работы программы

./client
70726: I'm a Parent
70728: I'm a child
70727: I'm a child
Hello World! K
hello world! k
Too Manyy Spaces ,,
too manyy spaces ,,

Вывод

В ходе данной лабораторной работы я научился управлять процессами в ОС. Также обеспечил обмен данных между процессами посредством каналов.