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: Лабутин М. А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 02.10.25

Москва, 2025

Постановка задачи

Вариант 12.

Группа вариантов 3. 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 верхний 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
- int write(int fd, const void *buf, size_t count); – записывает данные из буфера по файловому дескриптору.
- int dup2(int oldfd, int newfd); – перенаправляет файловый дескриптор, позволяя процессу использовать канал (pipe) вместо стандартного ввода/вывода.
- int read(int fd, void* buf, size_t count); - читает данные из файла по файловому дескриптору и записывает в buf.
- int execl(const char *path, const char *arg, ...); - заменяет текущий процесс новым процессом, загружая и выполняя указанную программу;
- pid_t waitpid(pid_t pid, int *wstatus, int options); - ожидает завершения конкретного дочернего процесса.
- int close(int fd); – закрывает файловый дескриптор.

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

Код программы

parent.c

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

```c
#define BUFFER_SIZE 1024
#define EXIT_SUCCESS 0
#define EXIT_FAILURE 1
int main(int argc, char *argv[]) {
    if (argc != 1) {
        printf("Usage: %s\n", argv[0]);
        printf("No arguments needed. Program reads from stdin and writes to stdout.\n");
        return EXIT_FAILURE;
    }
    int pipe_parent_to_child1[2];
    int pipe_child1_to_child2[2];
    int pipe_child2_to_parent[2];
    // Создаем каналы
    if (pipe(pipe_parent_to_child1) == -1 ||
        pipe(pipe_child1_to_child2) == -1 ||
        pipe(pipe_child2_to_parent) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }
    // Создаем Child1
    pid_t child1 = fork();
    if (child1 == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }
    if (child1 == 0) {
        close(pipe_parent_to_child1[1]);
        close(pipe_child1_to_child2[0]);
        close(pipe_child2_to_parent[0]);
        close(pipe_child2_to_parent[1]);
        dup2(pipe_parent_to_child1[0], STDIN_FILENO);
        dup2(pipe_child1_to_child2[1], STDOUT_FILENO);
        close(pipe_parent_to_child1[0]);
        close(pipe_child1_to_child2[1]);
        execl("./child1", "child1", NULL);
```

```
            perror("execl child1");
            exit(EXIT_FAILURE);
        }
        // Создаем Child2
        pid_t child2 = fork();
        if (child2 == -1) {
            perror("fork");
            exit(EXIT_FAILURE);
        }
        if (child2 == 0) {
            close(pipe_parent_to_child1[0]);
            close(pipe_parent_to_child1[1]);
            close(pipe_child1_to_child2[1]);
            close(pipe_child2_to_parent[0]);
            dup2(pipe_child1_to_child2[0], STDIN_FILENO);
            dup2(pipe_child2_to_parent[1], STDOUT_FILENO);
            close(pipe_child1_to_child2[0]);
            close(pipe_child2_to_parent[1]);
            execl("./child2", "child2", NULL);
            perror("execl child2");
            exit(EXIT_FAILURE);
        }
        close(pipe_parent_to_child1[0]);
        close(pipe_child1_to_child2[0]);
        close(pipe_child1_to_child2[1]);
        close(pipe_child2_to_parent[1]);
        char buffer[BUFFER_SIZE];
        ssize_t bytes_read;
        printf("The parent process is running. Enter the lines (Ctrl+D to end):\n");
        while (fgets(buffer, BUFFER_SIZE, stdin) != NULL) {
            write(pipe_parent_to_child1[1], buffer, strlen(buffer));
            bytes_read = read(pipe_child2_to_parent[0], buffer, BUFFER_SIZE - 1);
            if (bytes_read > 0) {
                buffer[bytes_read] = '\0';
                printf("Result: %s\n", buffer);
            }
```

```
    }

    close(pipe_parent_to_child1[1]);

    close(pipe_child2_to_parent[0]);

    waitpid(child1, NULL, 0);

    waitpid(child2, NULL, 0);

    printf("The parent process is completed.\n");

    return EXIT_SUCCESS;

}
```

child1.c

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <ctype.h>

#include <string.h>


#define BUFFER_SIZE 1024

#define EXIT_SUCCESS 0

#define EXIT_FAILURE 1

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

    if (argc != 1) {

        fprintf(stderr, "Usage: %s\n", argv[0]);

        fprintf(stderr, "This program reads from stdin and converts text to uppercase.\n");

        return EXIT_FAILURE;

    }

    char buffer[BUFFER_SIZE];

    ssize_t bytes_read;

    printf("Child1 started. PID: %d\n", getpid());

    while ((bytes_read = read(STDIN_FILENO, buffer, BUFFER_SIZE - 1)) > 0) {

        buffer[bytes_read] = '\0';


        for (int i = 0; i < bytes_read; i++) {

            buffer[i] = toupper(buffer[i]);

        }


        write(STDOUT_FILENO, buffer, bytes_read);

    }
```

```
    return EXIT_SUCCESS;
}
```

child2.c

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

#define BUFFER_SIZE 1024
#define EXIT_SUCCESS 0
#define EXIT_FAILURE 1
int main(int argc, char *argv[]) {
    if (argc != 1) {
        fprintf(stderr, "Usage: %s\n", argv[0]);
        fprintf(stderr, "This program reads from stdin and processes spaces.\n");
        return EXIT_FAILURE;
    }
    char buffer[BUFFER_SIZE];
    char result[BUFFER_SIZE];
    ssize_t bytes_read;
    printf("Child2 started. PID: %d\n", getpid());
    while ((bytes_read = read(STDIN_FILENO, buffer, BUFFER_SIZE - 1)) > 0) {
        buffer[bytes_read] = '\0';

        char *src = buffer;
        char *dst = result;
        int space_count = 0;
        int in_space_sequence = 0;

        while (*src) {
            if (isspace((unsigned char)*src)) {
                if (!in_space_sequence) {
                    in_space_sequence = 1;
                    space_count = 1;
```

```
                } else {
                    space_count++;
                }
            } else {
                if (in_space_sequence) {
                    if (space_count % 2 != 0) {
                        *dst++ = ' ';
                    }
                    in_space_sequence = 0;
                    space_count = 0;
                }
                *dst++ = *src;
            }
            src++;
        }

        if (in_space_sequence && space_count % 2 != 0) {
            *dst++ = ' ';
        }

        *dst = '\0';

        // Отправляем результат родителю
        write(STDOUT_FILENO, result, strlen(result));
    }
    return EXIT_SUCCESS;
}
```

Протокол работы программы

```
mxtv1x@WIN-EMVSNHTNF8D:/mnt/c/Users/User/OS-25/lab1$ ./parent
The parent process is running. Enter the lines (Ctrl+D to end):
PrIvEt MiR
Result: PRIVET MIR
hahahaha   hahahhahahah
Result: HAHAHAHA HAHAHHAHAHAH
hochu       sdat   labu
Result: HOCHU SDAT LABU
```

Вывод

В ходе выполнения данной лабораторной работы было изучено взаимодействие процессов через каналы (pipe) и механизмы их работы при создании дочерних процессов с помощью fork(). Я научился управлять процессами в ОС. Также обеспечил обмен данных между процессами посредством каналов.