Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу**

**«Операционные системы»**

**ВЗАИМОДЕЙСТВИЕ МЕЖДУ ПРОЦЕССАМИ**

Студент: Юрков Евгений Юрьевич

Группа: М8О–212Б–22

Вариант: 14

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* Управление процессами в ОС
* Обеспечение обмена данных между процессами посредством каналов

## Задание

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.

**Вариант 14:** Child1 переводит строки в нижний регистр. Child2 убирает все задвоенные пробелы.

**Общие сведения о программе**

Программа компилируется из файла main.c, также есть файлы child1.c и child2.c, необходимые для работы основной прогаммы. Также используется заголовочные файлы: stdio.h, unistd.h, stdlib.h, string.h, ctype.h, sys/wait.h . В программе используются следующие системные вызовы:

1. **fork** – создание дочернего процесса.
2. **execvp** – замена образа памяти процесса.
3. **pipe** – создание неименованного канала для передачи данных между процессами.
4. **waitpid** – ожидание завершения дочернего процесса.
5. **exit** –завершение выполнения процесса и возвращение статуса.
6. **dup2** – переназначение файлового дескриптора.
7. **close** – закрыть файл.
8. **read** – чтение данных из файла.
9. **write** – запись данных в файл.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, exec, pipe, dup2.
2. Написать файлы child1.c, child2.c, для обработки строки.
3. Написать файл main.c, в котором будут осуществляться запуски программ child1 и child2 и обмен данными между процессами.

**Основные файлы программы**

**child1.c:**

#include <stdio.h>  
#include <ctype.h>   
#include <unistd.h>   
   
int main() {   
    char s[100];   
   
    int i = 0;   
    do {   
        char c;   
        read(STDIN\_FILENO, &c, sizeof(char));   
        s[i++] = tolower(c);   
    } while (s[i - 1] != 0 && s[i - 1] != '\n');   
    s[i] = 0;   
   
    i = 0;   
    do {   
        write(STDOUT\_FILENO, &s[i++], sizeof(char));   
    } while (s[i - 1] != 0 && s[i - 1] != '\n');   
   
    return 0;   
}

**child2.c:**

#include <stdio.h>  
#include <unistd.h>   
   
int main() {   
    char s[100];   
   
    read(STDIN\_FILENO, s, sizeof(char));   
    int i = 1;   
    do {   
        char c;   
        read(STDIN\_FILENO, &c, sizeof(char));   
        if (!(c == ' ' && s[i - 1] == ' ')) {   
            s[i++] = c;   
        }   
    } while (s[i - 1] != 0 && s[i - 1] != '\n');   
    s[i] = 0;   
   
    write(STDOUT\_FILENO, s, sizeof(char) \* i);   
   
    return 0;   
}

**main.c**

#include <stdio.h>  
#include <stdlib.h>   
#include <string.h>   
#include <sys/wait.h>   
#include <unistd.h>   
   
void create\_two\_process(int\* a, int\* b) {   
    \*a = fork();   
    if (\*a > 0) {   
        \*b = fork();   
    }   
    if (\*a == -1 || \*b == -1) {   
        write(STDERR\_FILENO, "fork error\n", 12);   
        exit(-1);   
    }   
    return;   
}   
   
int main(int argc, char \*argv[]) {   
    int parent\_pid = getpid(), pid1, pid2;   
       
    int fd1[2]; // parent -> child1   
    if (pipe(fd1) == -1) {   
        write(STDERR\_FILENO, "pipe wasn't created", 20 \* sizeof(char));   
    }   
    int fd2[2]; // child1 -> child2   
    if (pipe(fd2) == -1) {   
        write(STDERR\_FILENO, "pipe wasn't created", 20 \* sizeof(char));   
    }   
    int fd3[2]; // child2 -> parent   
    if (pipe(fd3) == -1) {   
        write(STDERR\_FILENO, "pipe wasn't created", 20 \* sizeof(char));   
    }   
   
    char s[100];   
    // Reading of the string s   
    int i = 0;   
    do {   
        char c;   
        read(STDIN\_FILENO, &c, sizeof(char));   
        s[i] = c;   
        ++i;   
    } while (s[i - 1] != 0 && s[i - 1] != '\n');   
   
    create\_two\_process(&pid1, &pid2); // create child1 & child2   
   
    if (pid1 > 0 && pid2 > 0) { // Parent process   
        int s\_size = strlen(s);   
   
        close(fd1[0]);   
        write(fd1[1], s, s\_size \* sizeof(char)); // passes the string to the Child1 process   
        close(fd1[1]);   
   
        waitpid(pid1, NULL, WUNTRACED);   
        waitpid(pid2, NULL, WUNTRACED);   
   
        char res[100] = "";   
        close(fd3[1]);   
        read(fd3[0], res, s\_size \* sizeof(char)); // gets the result from the Child2 process   
   
        write(STDOUT\_FILENO, res, strlen(res) \* sizeof(char));   
        write(STDOUT\_FILENO, '\n', sizeof(char));   
        close(fd3[0]);   
    }   
    else if (pid1 == 0) { // Child1 process   
        wait(NULL);   
        close(fd1[1]);   
        dup2(fd1[0], STDIN\_FILENO);   
   
        close(fd2[0]);   
        dup2(fd2[1], STDOUT\_FILENO);   
   
        execvp("./child1", argv);   
    }   
    else { // Child2 process   
        wait(NULL);   
        close(fd2[1]);   
        dup2(fd2[0], STDIN\_FILENO);   
   
        close(fd3[0]);   
        dup2(fd3[1], STDOUT\_FILENO);   
   
        execvp("./child2", argv);   
    }   
   
    return 0;   
}

**Пример работы**

**Тесты:**

**[kruyneg@matebook14 build]$** cat ../lab1/tests/test1.txt  
ABCDef   ghI   
**[kruyneg@matebook14 build]$** cat ../lab1/tests/test2.txt  
ThE QuiCk   BrowN FoX JuMpS  OVER   the  LaZy dOg   
**[kruyneg@matebook14 build]$** cat ../lab1/tests/test3.txt  
I  LOVE TO EAT PIZZA    AND  play video games.

**Результат**:  
**[kruyneg@matebook14 build]$** ./main < ../lab1/tests/test1.txt  
abcdef ghi   
**[kruyneg@matebook14 build]$** ./main < ../lab1/tests/test2.txt  
the quick brown fox jumps over the lazy dog   
**[kruyneg@matebook14 build]$** ./main < ../lab1/tests/test3.txt  
i love to eat pizza and play video games.

**Вывод**

Проделав работу, я разработал программу, которая использовала несколько процессов для обработки строки. Я узнал, как создавать новые процессы в ОС UNIX и управлять их выполнением. Также я научился использовать pipe для передачи данных между процессами. Одной из проблем, с которыми я столкнулся при написании программы, было осуществление работы процессов в нужном порядке.

Программы с дочерними процессами имеют ряд преимуществ в разных областях. Одним из них является возможность параллельного выполнения задач, что позволяет ускорить обработку данных или выполнение вычислений. Также программы с дочерними процессами позволяют реализовать механизм отказоустойчивости, когда при остановке одного процесса, другие продолжают работу.