Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**
**«Операционные системы»**

**Тема работы**
**“Изучение взаимодействий между процессами”**

Студент: Ханнанов Руслан Маратович
Группа: М8О-208Б-20
Вариант: 14
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

**Содержание**

**Репозиторий**

https://github.com/Naksen/OS

**Постановка задачи**

Задача: Родительский процесс создает два дочерних процесса. Через pipe fd_1 идет передача данных из родительского процесса в первый дочерний. Первый дочерний процесс переводит строки в нижний регистр, оправляет уже изменные данные через pipe fd в во второй дочерний процесс. Второй дочерний процесс убирает все задвоенные пробелы и отправляет данные назад в родительский процесс через pipe fd_2. Родительский процесс производит вывод данных в консоль.

**Общие сведения о программе**

Реализация программы была бы невозможна без специальной библиотеки “unistd.h” для операционной системы Linux, которая позволяет работать с процессами и системными вызовами. По мере реализации задания используются такие строки(команды), как:

int fd[2] - создание массива из 2 дескрипторов, 0 - чтение (read), 1 - передача (write):

pipe(fd) - конвейер, с помощью которого выход одной команды подается на вход другой (оно же “труба”);

int pid_1 = fork () - создание дочернего процесса, в переменной pid_1 будет лежать “специальный код” процесса (-1 - ошибка fork, 0 - дочерний процесс, >0 - родительский);

read(...) - команда, предназначенная для чтения данных, посланных из

другого процесса, принимающая на вход три параметра: элемент массива дескрипторов с индексом 0, значение получаемого объекта (переменной, массива и т.д.), размер получаемого объекта (например, в случае переменной int - sizeof(int), в случае массива из 10 переменных типа int - sizeof(int) * 10);

write(...) - команда, принимающая на вход три параметра: элемент массива дескрипторов с индексом 1, значение посылаемого объекта (переменной, массива и т.д.), размер посылаемого объекта (например, в случае переменной int - sizeof(int), в случае массива из 10 переменных типа int - sizeof(int) * 10);

close(...) - команда, использующаяся, когда нам больше не нужно передавать, либо считывать что-либо из другого процесса.

**Общий метод и алгоритм решения**

Программа получает на вход символьные данные. После чего данные считываются в “буфферный” массив. В первом дочернем процессе изменяется стандартный поток ввода stdin на дескриптор чтения fd_1[0]. А стандартный поток вывода stdout на дескриптор записи fd[1]. Во втором дочернем процессе тоже происходит замена стандартных поток ввода и вывода. Каждый символ строки переходит в нижний регистр использованием функции tolower. А поиск на задвоенные пробелы проходит с помощью поиска двух символов пробела стоящих в строке подряд. Как следствие, первый и второй дочерние процессы производят обработку данных и возвращают их в родительский процесс. Программа завершена.
Лабораторная работа была выполнена в среде Visual Studio code, название файла – main.cpp.
Собирается программа при помощи команды Makefile, находящегося в папке src.

**Исходный код**

**main.c:**

```cpp
#include <iostream>
#include <unistd.h>
#include <cctype>
#include <algorithm>
#include <sys/wait.h>

using namespace std;

int main() {
    int fd[2];
    int fd_1[2];
    int fd_2[2];
    pipe(fd); // pipe: Child_1 -> Child_2
    pipe(fd_1); // pipe: Parent -> Child_1
    pipe(fd_2); // pipe: Child_2 -> Parent
    int pid_1 = 0;
    int pid_2 = 0;
    //reading

    auto *in = new char[2];
    in[0] = 0;
    char c;
    while ((c = getchar()) != EOF) {

        in[0] += 1;
        in[in[0]] = c;
        in = (char *) realloc(in, (in[0] + 2) * sizeof(char));

    }
    in[in[0]] = '\0';

    if ((pid_1 = fork()) > 0) { // Parent 1

        //cout << "||Parent 1||" << "\n";
        write(fd_1[1], in, (in[0] + 2) * sizeof(char));
        close(fd_1[1]);
        close(fd_1[0]);
        close(fd[1]);

        if ((pid_2 = fork()) > 0) { // Parent 2

            //cout << "||Parent 2||" << "\n";

            close(fd_2[1]);
            close(fd[1]);
            close(fd[0]);

            char ch;
            while(read(fd_2[0], &ch, sizeof(char)) > 0){
                putchar(ch);
            }

        } else if (pid_2 == 0) { //Child_2
```

```
            if (dup2(fd_2[1], fileno(stdout)) == -1 ) {

                perror("dup2");
                exit(1);

            }
            close(fd_2[1]);

            if (dup2(fd[0], fileno(stdin)) == -1) {

                perror("dup2");
                exit(1);

            }
            close(fd[0]);

            //cout << "child2 is opening...\n";
            execl("child2", "child2", NULL);
            perror("execl");


        } else {

            cout << "fork error" << endl;
            exit(-1);

        }

    } else if (pid_1 == 0) { //Child_1

        //cout << "||Child 1 ||" << "\n";
        if (dup2(fd[1], fileno(stdout)) == -1 ) {

            perror("dup2");
            exit(1);

        }
        close(fd[1]);
        if (dup2(fd_1[0], fileno(stdin)) == -1) {

            perror("dup2");
            exit(1);

        }
        close(fd_1[0]);
        close(fd_1[1]);
        //cout << "child1 is opening...\n";
        execl("child1", "child1", NULL);
        perror("execl");

    } else {

        cout << "Fork error" << endl;
        exit(-1);

    }
    return 0;
}
```

## child1.cpp:

```cpp
#include <unistd.h>
#include <iostream>

using namespace std;

int main(){

    auto *in = new char[2];
    in[0] = 0;
    char c;
    c = getchar();
    while ((c = getchar()) != EOF) {

        in[0] += 1;
        in[in[0]] = c;
        in = (char *) realloc(in, (in[0] + 2) * sizeof(char));

    }
    in[in[0]] = '\0';

    for (int i = 1; i < in[0]; i++) {

        in[i] = tolower(in[i]);

    }

    for (int i = 1; i < in[0] + 1; i++) {

        cout << in[i];

    }

    //printf("\n");
    //delete(in);
    fflush(stdout);
    close(STDOUT_FILENO);
    return 0;
}
```

## child2.cpp:

```cpp
#include <unistd.h>
#include <iostream>

using namespace std;

int main(){

    auto *in = new char[2];
    in[0] = 0;
    char c;
    while ((c = getchar()) != EOF) {

        in[0] += 1;
        in[in[0]] = c;
        in = (char *) realloc(in, (in[0] + 2) * sizeof(char));

    }
    in[in[0]] = '\0';

    char *out = (char *) malloc(2 * sizeof(char));
    out[0] = 0;

    for (int i = 1; i < in[0] + 1; i++) {

        if (in[i] == ' ' && in[i + 1] == ' ') {

            i++;
            continue;

        }

        out[0]++;
        out[out[0]] = in[i];
        out = (char *) realloc(out, (out[0] + 2) * sizeof(out));

    }

    out[0]++;
    out[out[0]] = '\0';

    for (int i = 1; i < out[0]; i++) {

        cout << out[i];

    }

    printf("\n");
    fflush(stdout);
    //delete(in);
    //delete(out);
    close(STDOUT_FILENO);
    return 0;
}
```

## Демонстрация работы программы

```
naksan@LAPTOP-TL9L61MA:~/cprog/lab2/2_lab/src$ ./prog
jJSDHABJBajdbjasdJDJSADBBDBBDBBBB   dsabdbDBBDBDBDBD
BBBB NNNNN   NNNNNSAD
jjsdhabjbajdbjasdjdjsadbbdbbdbbbb dsabdbdbbdbdbdbd
bbbb nnnnn nnnnnsad
```

## Выводы

После выполнения данной лабораторной работы я с уверенностью могу сказать, что хорошо ознакомился с темой создания процессов в Linux. Я на

примере собственного задания осознал принципы работы вышеперечисленных команд pipe, fork, write, read, close, dup2, научился ими пользоваться и даже познал некоторые тонкости (например, все, что было создано в родительском процессе, есть и в дочернем). Уверен, что полученные навыки помогут мне дальше осваивать курс по операционным системам.