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: Баталин Дмитрий Андреевич

Группа: М8О–206Б–20

Вариант: 14

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

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

Формулировка задания по варианту:

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.

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

**Общие сведения о программе**

Программа компилируется из файла main.c. В программе используются следующие системные вызовы:

1. **wait** – приостанавливает выполнение вызвавшего процесса до тех пор, пока не прекратит выполнение один из его потомков.
2. **pipe** – создаёт однонаправленный канал данных, который можно использовать для взаимодействия между процессами.
3. **close** – закрывает файловый дескриптор, который после этого не ссылается ни на один и файл и может быть использован повторно.
4. **write -**  записывает до count байтов из буфера buf в файл, на который ссылается файловый описатель fd.
5. **read -** пытается записать count байтов файлового описателя fd в буфер, адрес которого начинается с buf.
6. **fork –** создаёт новый процесс (потомок), который является практически полной копией процесса-родителя, выполняющего этот вызов.
7. **exit -** "немедленно" завершает работу программы.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы простейших системных вызовов (включая fork)
2. Реализовать алгоритм преобразования строк в нижний регистр
3. Реализовать алгоритм, который убирает все задвоенные пробелы в строке
4. Создать «архитектуру» программы, то есть создать два дочерних проекта, проложить между ними пайпы
5. Использовать написанные ранее алгоритмы в программе.

**Основные файлы программы**

**main.c:**

#include "stdio.h"

#include "stdlib.h"

#include "unistd.h"

#include "sys/wait.h"

#define STRING\_SIZE 100

**int** **create\_process**();

**void** **create\_pipe**(**int**\* pipe\_fd);

**void** **to\_lower**(**char**\* str);

**void** **remove\_spaces**(**char**\* str);

**void** **read\_input**(**char** \*buffer, **size\_t** bufferSize);

**int** **main**() {

**int** pipe\_fd[**2**]; create\_pipe(pipe\_fd);

**int** pipe\_fd\_children[**2**]; create\_pipe(pipe\_fd\_children);

**int** pipe\_fd\_final[**2**]; create\_pipe(pipe\_fd\_final);

**pid\_t** pid = create\_process();

**if** (pid == **0**) // child1

{

**char** parent\_string[STRING\_SIZE];

close(pipe\_fd[**1**]);

close(pipe\_fd\_children[**0**]);

read(pipe\_fd[**0**], &parent\_string, STRING\_SIZE \* **sizeof**(**char**));

to\_lower(parent\_string);

write(pipe\_fd\_children[**1**], &parent\_string, STRING\_SIZE \* **sizeof**(**char**));

close(pipe\_fd[**0**]);

close(pipe\_fd\_children[**1**]);

}

**else**

{

**pid\_t** pid = create\_process();

**if** (pid == **0**) // child2

{

**char** child\_string[STRING\_SIZE];

close(pipe\_fd[**0**]);

close(pipe\_fd[**1**]);

close(pipe\_fd\_children[**1**]);

close(pipe\_fd\_final[**0**]);

read(pipe\_fd\_children[**0**], &child\_string, STRING\_SIZE \* **sizeof**(**char**));

remove\_spaces(child\_string);

write(pipe\_fd\_final[**1**], &child\_string, STRING\_SIZE \* **sizeof**(**char**));

close(pipe\_fd\_children[**0**]);

}

**else** // parent

{

**char** string[STRING\_SIZE];

close(pipe\_fd[**0**]);

close(pipe\_fd\_final[**1**]);

read\_input(string, STRING\_SIZE);

write(pipe\_fd[**1**], &string, STRING\_SIZE \* **sizeof**(**char**));

wait(NULL);

read(pipe\_fd\_final[**0**], &string, STRING\_SIZE \* **sizeof**(**char**));

write(STDOUT\_FILENO, &string, STRING\_SIZE \* **sizeof**(**char**));

close(pipe\_fd[**1**]);

close(pipe\_fd\_final[**0**]);

}

}

**return** **0**;

}

**int** **create\_process**() {

**pid\_t** pid = fork();

**if** (-**1** == pid)

{

perror("Error while fork");

exit(-**2**);

}

**return** pid;

}

**void** **create\_pipe**(**int**\* pipe\_fd) {

**if** (pipe(pipe\_fd) == -**1**)

{

perror("Failed to create pipe");

exit(-**1**);

}

}

**void** **to\_lower**(**char** \*str) {

**while** (\*str) {

**if** ((\*str >= **65**) && (\*str <= **90**))

\*str = \*str + **32**;

str += **1**;

}

}

**void** **remove\_spaces**(**char**\* str) {

**int** len = STRING\_SIZE;

**if** (len <= **1**) {

**return**;

}

**char**\* dest = str;

\*dest++ = str[**0**];

**int** i = **1**;

**for** (; str[i] != **0** && i < len ; i++) {

**if** (!(str[i-**1**] == ' ' && str[i] == ' '))

{

\*dest = str[i];

dest++;

}

}

**for** (; i < len; i++) {

\*dest = **0**;

dest++;

}

}

**void** **read\_input**(**char** \*buffer, **size\_t** bufferSize) {

**ssize\_t** bytesRead;

bytesRead = read(STDIN\_FILENO, buffer, bufferSize - **1**);

**if** (bytesRead == -**1**) {

perror("Read error");

exit(-**1**);

}

**for** (**int** i = bytesRead; i < bufferSize; i++) {

buffer[i] = **0**;

}

}

**Пример работы**

moses@moses:~/cs/os/lab1/src$ ./build/lab1 < ../test/test01.txt

big string with spacesmoses@mo./build/lab1 < ../test/test02.txt

moses@moses:~/cs/os/lab1/src$ ./build/lab1 < ../test/test03.txt

moses@moses:~/cs/os/lab1/src$ ./build/lab1 < ../test/test04.txt

some text some text

some text in two stringsmoses@moses:~/cs/os/lab1/src$

**Вывод**

Подводя итоги выполненной работы, можно сказать, что, используя системные вызовы, можно вводить и выводить данные из программы, взаимодействовать между процессами, с помощью однонаправленных каналов связи и завершать работу программы.