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»**

**Взаимодействие между процессами**

Группа: М80-210Б-22

Студент: Бонокин Д.С.

Вариант:11

Преподаватель: Соколов А.А.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2023

**Постановка задачи**

**Цель работы:**

Приобретение практических навыков в:

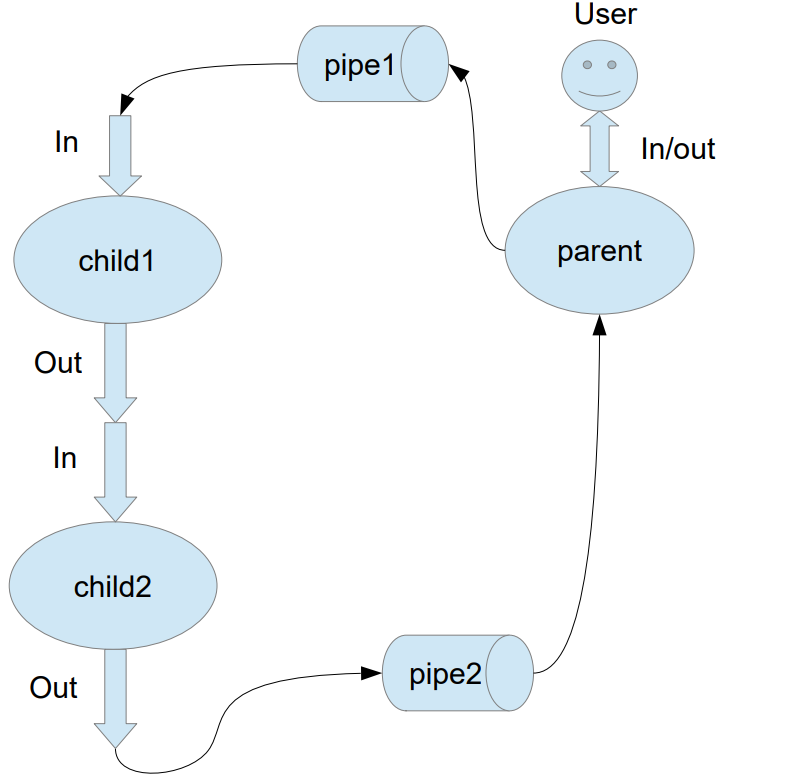
• Управление процессами в ОС

• 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.

**Вариант 11.**



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 превращает все пробельные символы в символ «\_».

**Общие сведения о программе**

Программа компилируется при помощи утилиты Make и запускается путем

запуска ./parent. Также используется заголовочные файлы: iostream,

unistd.h, cctype. В программе используются следующие системные вызовы:

* int pipe(int \*fd); – создаёт канал (пайп).
* pid\_t fork(void); – создаёт дочерний процесс.
* int dup2(int oldfd, int newfd); – делает newfd копией дескриптора oldfd, закрывая newfd, если требуется.
* int execl(const char \*path, const char \*arg, ...); – заменяет текущий образ процесса новым образом процесса.
* int close(int fd); – закрывает файловый дескриптор.
* size\_t write(int fd, const void \*buf, size\_t count); – записывает до count байтов из буфера buf в файл, на который ссылается файловый описатель fd.
* size\_t read(int fd, void \*buf, size\_t count); – пытается записать count байтов файлового описателя fd в буфер, адрес которого начинается с buf.

**Общий метод и алгоритм решения**

Создал три канала для связи дочерних процессов и родительского с дочерними с помощью pipe(). Далее создал два дочерних процесса с помощью fork() и вызвал скомпилированные child1.cpp и child2.cpp с помощью execl(). В родительском процессе читал символы, которые пишет пользователь и сначала посылал в child1. Первый дочерний процесс с помощью toupper() переводил символы в верхний регистр и посылал их child2 через канал. Второй дочерний процесс заменял пробел на «\_» и посылал обратно родительскому процессу, который уже выводил их на стандартный вывод.

**Код программы**

**main.cpp**

#include <unistd.h>

#include <iostream>

#include <cctype>

pid\_t create\_process() {

pid\_t pid = fork();

if (pid == -1) {

perror("fork error!\n");

exit(-1);

}

return pid;

}

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

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

perror("pipe error!\n");

exit(-1);

}

}

void dup\_fd(int oldfd, int newfd) {

if (dup2(oldfd, newfd) == -1) {

perror("dup2 error!\n");

exit(-1);

}

}

void new\_program(const char \*name){

if(execl(name, name, NULL) == -1){

perror("execl error!\n");

exit(-1);

}

}

int main() {

int pipe1\_fd[2], pipe2\_fd[2];

create\_pipe(pipe1\_fd);

create\_pipe(pipe2\_fd);

pid\_t child1 = create\_process();

if (child1 == 0) {

close(pipe1\_fd[1]);

close(pipe2\_fd[0]);

int pipe3\_fd[2];

create\_pipe(pipe3\_fd);

pid\_t child2 = create\_process();

if (child2 == 0) {

close(pipe3\_fd[0]);

close(pipe2\_fd[1]);

dup\_fd(pipe1\_fd[0], STDIN\_FILENO);

dup\_fd(pipe3\_fd[1], STDOUT\_FILENO);

new\_program("child2");

close(pipe3\_fd[1]);

close(pipe1\_fd[0]);

} else {

close(pipe1\_fd[0]);

close(pipe3\_fd[1]);

dup\_fd(pipe3\_fd[0], STDIN\_FILENO);

dup\_fd(pipe2\_fd[1], STDOUT\_FILENO);

new\_program("child1");

close(pipe1\_fd[0]);

close(pipe2\_fd[1]);

}

} else {

close(pipe1\_fd[0]);

close(pipe2\_fd[1]);

char c = getchar();

char new\_c;

while (c != EOF) {

write(pipe1\_fd[1], &c, sizeof(c));

read(pipe2\_fd[0], &new\_c, sizeof(c));

putchar(new\_c);

c = getchar();

}

close(pipe1\_fd[1]);

close(pipe2\_fd[0]);

}

return 0;

}

**child1.cpp**

#include <iostream>

#include <unistd.h>

int main() {

char c;

while (read(STDIN\_FILENO, &c, sizeof(c)) != -1) {

c = toupper(c);

write(STDOUT\_FILENO, &c, sizeof(c));

}

close(STDIN\_FILENO);

close(STDOUT\_FILENO);

}

**child2.cpp**

#include <iostream>

#include <unistd.h>

int main() {

char c;

while (read(STDIN\_FILENO, &c, sizeof(c)) != -1) {

if (c == ' ') {

c = '\_';

}

write(STDOUT\_FILENO, &c, sizeof(c));

}

close(STDIN\_FILENO);

close(STDOUT\_FILENO);

}

**Протокол работы программы**

**Тестирование:**

danil@danil-1-2:~/lab/lab1/build$ ./main

cvbs ddsq

CVBS\_\_\_DDSQ

derftv DDc fvt

DERFTV\_DDC\_FVT

**Вывод**

В ходе лабораторной работы я написал программу, которая делает системные вызовы. Я научился работать с каналами и процессами.