Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

Студент: Пивницкий Д.С.

Группа: М8о–206Б–19

Вариант: 11

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

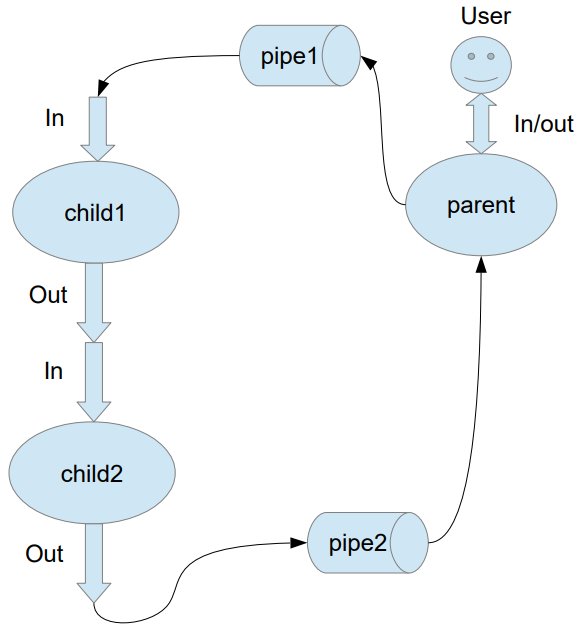
**Цель работы**

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:Child1 переводит строки в верхний регистр. Child2 превращает все пробельные символы в символ «\_».

**Общие сведения о программе**

Программа компилируется из файла lab2.c. Также используется заголовочные файлы: unistd.h, stdio.h , stdlib.h, ctype.h. В программе используются следующие системные вызовы:

1. **fork** - создает копию текущего процесса, который является дочерним процессом для текущего процесса
2. **pipe** - создаёт однонаправленный канал данных, который можно использовать для взаимодействия между процессами.
3. **fflush** - если поток связан с файлом, открытым для записи, то вызов приводит к физической записи содержимого буфера в файл. Если же поток указывает на вводимый файл, то очищается входной буфер.
4. **close** - закрывает файл.
5. **read** - читает количество байт(третий аргумент) из файла с файловым дескриптором(первый аргумент) в область памяти(второй агрумент).
6. **write -**  записывает в файл с файловым дескриптором(первый аргумент) из области памяти(второй аргумент) количество байт(третий аргумент).
7. **perror –** вывод сообщения об ошибке.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, pipe, fflush, close, read, write.
2. Написать программу, которая будет работать с 3-мя процессами: один родительский и два дочерних, процессы связываются между собой при помощи pipe-ов.
3. Организовать работу с выделением памяти под строку неопределенной длины и запись длины в массив строки в качестве первого элемента для передачи между процессами через pipe.

**Основные файлы программы**

**lab2.c:**

#include "unistd.h"

#include "stdio.h"

#include "stdlib.h"

#include "ctype.h"

int main()

{

int fd\_0[2];

int fd\_1[2];

int fd\_2[2];

pipe(fd\_0);

pipe(fd\_1);

pipe(fd\_2);

int pid\_0 = 0;

int pid\_1 = 0;

if((pid\_0 = fork()) > 0)

{ //родительский процесс Parent

if((pid\_1 = fork()) > 0)

{ //Parent

fflush(stdout);

fflush(stdin);

char\* in = (char\*)malloc(2\*sizeof(char));

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]+1] = '\0';

write(fd\_0[1], in, (in[0]+2)\*sizeof(char));

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

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

{

read(fd\_2[0], &out[i], sizeof(char));

}

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

{

printf("%c", out[i]);

}

printf("\n");

fflush(stdout);

close(fd\_2[0]);

close(fd\_0[1]);

free(in);

free(out);

}

else if(pid\_1 == 0)

{ //Child2

fflush(stdout);

fflush(stdin);

char\* in = (char\*)malloc(sizeof(char));

read(fd\_1[0], &in[0], sizeof(char));

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

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

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

{

read(fd\_1[0], &in[i], sizeof(char));

}

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

{

out[i] = in[i];

if(in[i] == ' ')

out[i] = '\_';

}

out[0] = in[0];

write(fd\_2[1], out, (out[0]+2)\*sizeof(char));

fflush(stdout);

close(fd\_2[1]);

close(fd\_1[0]);

free(in);

free(out);

}

else

{ //Parent

perror("fork error\n");

exit(-1);

}

}

else if (pid\_0 == 0)

{ //дочерний процесс Child1

fflush(stdout);

fflush(stdin);

char\* in = (char\*)malloc(sizeof(char));

read(fd\_0[0], &in[0], sizeof(char));

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

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

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

{

read(fd\_0[0], &in[i], sizeof(char));

}

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

{

out[i] = toupper(in[i]);

}

out[0] = in[0];

write(fd\_1[1], out, (out[0]+2)\*sizeof(char));

fflush(stdout);

close(fd\_0[0]);

close(fd\_1[1]);

free(in);

free(out);

}

else

{ // Parent

perror("fork error\n");

exit(-1);

}

return 0;

}

**Пример работы**

daniel@daniel-Ideapad-Z570: ~ cd os/oslab2  
daniel@daniel-Ideapad-Z570: ~ oslab2 cat test.txt

heLlo woRld

gooDbye tyna NonaMe

reaD my Prooggma

Typak NonE gooD

EEeee enD thnks!

The Core Differences Between C and C++%   
daniel@daniel-Ideapad-Z570: ~ oslab2 gcc lab2.c -o out -pthread

daniel@daniel-Ideapad-Z570: ~ oslab2 ./out <test.txt

HELLO\_WORLD

GOODBYE\_TYNA\_NONAME

READ\_\_MY\_PROOGGMA

\_\_TYPAK\_NONE\_\_\_GOOD

EEEEE\_END\_THNKS!

THE\_CORE\_DIFFERENCES\_BETWEEN\_C\_AND\_C++

**Вывод**

Существуют специальные системные вызовы(fork) для создания процессов, также существуют специальные каналы pipe, которые позволяют связать процессы и обмениваться данными при помощи этих pipe-ов. При использовании fork важно помнить, что фактически создается копию вашего текущего процесса и неправильная работа может привести к неожиданным результатам и последствиям, однако создание процессов очень удобно, когда вам нужно выполнять несколько действий параллельно. Также у каждого процесса есть свой id, по которому его можно определить. Также важно работать с чтением и записью из канала, помня что read, write возвращает количество успешно считанных/записанных байт и оно не обязательно равно тому значению, которое вы указали. Также важно не забывать закрывать pipe после завершения работы.