Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

Группа: М80-207Б-18

Студент: Петрин Сергей Александрович

Преподаватель: Миронов Евгений Сергеевич

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

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

Москва, 2019.

***Постановка задачи***Составить программу на языке C, осуществляющую работу с процессами и взаимодействие между ними:

Вариант 6: На вход программе подается команда интерпретатора команд. Программа должна произвести вывод команды заменяя знаки табуляции и spaceна знаки «\_\_\_\_» и «\_\_» соответственно.

***Алгоритм***

Для выполнения данной задачи необходимо создать канал передачи данных, дочерний процесс, который с помощью канала будет принимать команду интерпретатора команд, введенного из родительского процесса, заменять ответ и передавать обратно через канал родителю. Родитель должен ждать окончания выполнения родительского процесса, считывать из канала ответ и выводить его. Также должна происходить обработка возможных ошибок.

***Описание программы***

Программа состоит из одного файла laba.c. В этом файле подключаются необходимые библиотеки для работы с системными вызовами; в начале нам нужно получить команду интерпретатора команд - если её нет, то выведем ошибку.  
Далее объявляются файловые дескрипторы int my\_pipe[2] и происходит создание канала pipe(my\_pipe) и одновременная проверка на ошибку при выполнении вызова;  
после этого, используя fork(), родительский процесс создает дочерний процесс, и при помощи идентификатора процесса производится работа  
 -с неудачным вызовом fork() (при child\_id == -1): происходит вывод ошибки;  
 -с родительским процессом (при pid > 0): принимает посимвольно вывод команды интерпритатора команд и заменяет нужные нам символы;

-c дочерним процессом (при pid == 0): перенаправляем stdout и пишем вывод команды интерпритатора команд туда.

***Листинг***

*laba2.c*

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

#include <string.h>

#include <sys/wait.h>

int main(int argc, char\* argv[])

{

if( argc <= 1 ) {

printf("use %s <bash command>\n", argv[0]);

return 0;

}

char\* arguments[] = {argv[1], NULL};

int wstatus;

int my\_pipe[2];

if(pipe(my\_pipe) == -1)

{

fprintf(stderr, "Error creating pipe\n");

}

pid\_t child\_id;

child\_id = fork();

// <https://habr.com/ru/post/125369/>

if(child\_id == -1)

{

fprintf(stderr, "Fork error\n");

}

if(child\_id == 0) // child process

{

close(my\_pipe[0]); // child doesn't read

(my\_pipe[1], 1); // redirect stdout

execvp(argv[1], arguments); // <https://habr.com/ru/post/423049/>

fprintf(stderr, "Exec failed\n");

}

else

{

wait( &wstatus ); // Функция wait() ждет завершения первого из всех возможных потомков родительского процесса.

close(my\_pipe[1]); // parent doesn't write

char reading\_buf[1];

while(read(my\_pipe[0], reading\_buf, 1) > 0)

{

if ( reading\_buf[0] == '\t' ) {

\*reading\_buf = '\_';

write(1, reading\_buf, 1);

write(1, reading\_buf, 1);

}

else if ( reading\_buf[0] == ' ' ) {

\*reading\_buf = '\_';

write(1, reading\_buf, 1);

}

else {

write(1, reading\_buf, 1);

}

}

close(my\_pipe[0]);

}

}

***Демонстрация работы программы***

~/Course2/OS/laba2(master\*) # gcc main.c root@DESKTOP-DL6JCR0

------------------------------------------------------------------------------------------------

~/Course2/OS/laba2(master\*) # ./a.out cal root@DESKTOP-DL6JCR0

\_\_\_December\_2019\_\_\_\_\_\_

Su\_Mo\_Tu\_We\_Th\_Fr\_Sa\_\_

\_1\_\_2\_\_3\_\_4\_\_5\_\_6\_\_7\_\_

\_8\_\_9\_10\_11\_12\_13\_14\_\_

15\_16\_17\_18\_19\_20\_21\_\_

22\_23\_24\_25\_26\_27\_28\_\_

29\_30\_31\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

------------------------------------------------------------------------------------------------

~/Course2/OS/laba2(master\*) # ./a.out date root@DESKTOP-DL6JCR0

Fri\_Dec\_13\_05:38:29\_MSK\_2019

------------------------------------------------------------------------------------------------

~/Course2/OS/laba2(master\*) # ./a.out dataaa root@DESKTOP-DL6JCR0

Exec failed

------------------------------------------------------------------------------------------------

~/Course2/OS/laba2(master\*) # ./a.out root@DESKTOP-DL6JCR0

use ./a.out <bash command>

***Вывод***

В результате выполнения данной лабораторной работы я получил первый опыт написания программ для ОС, использующих системные вызовы, такие как fork(), pipe(), write(), read(), wait(), close(), execvp(). Принципы работы данных системных вызовов не вызывают больших трудностей при понимании и написании программы:

* при вызове fork() из родительского процесса происходит создание дочернего процесса, при этом происходит копия родительского процесса в дочерний (копируется текст программы, данные, стек, таблица дескрипторов, текущая позиция указателя записи-чтения; при этом используется механизм copy-on-write - при чтении области данных используется общая копия, в случае изменения данных — создается новая копия); процесс-потомок и процесс-родитель получают разные коды возврата после вызова fork(), и благодаря этому можно написать отдельный код для каждого процесса.
* при помощи pipe() осуществляется взаимодействие между процессами, при этом создается канал передачи данных, использующий два файловых дескриптора для записи и чтения данных при помощи write() и read(); pipe также применяется для синхронизации (предотвращения “гонок” между процессами), при этом используется механизм закрытия дескрипторов чтения/записи при помощи close().
* с помощью wait() для одного доч. процесса - родительский процесс может приостанавливать свою работу до завершения работы дочерних процессов, что также используется для синхронизации.

Эти знания понадобятся мне при дальнейшей работе с ОС.