Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу

«Операционные системы»

Создание процессов

Студент: [ Нелюбин В.С ]

Группа: М8О–206Б–19

Вариант: 10

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).

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

В файле записаны команды вида: «число<endline>». Дочерний процесс производит проверку этого числа на простоту. Если число составное, то дочерний процесс пишет это число в стандартный поток вывода. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются.

**Общие сведения о программе**

Программа компилируется из файла whatt.c и whato.c (компилируются как c.out и b.out соответственно). Также используется заголовочные файлы: stdio.h stdlib.h unistd.h sys/types.h errno.h. Основные команды программы:

1. **fork** – Создаёт копию текущего процесса. Возвращает 0 в новом процессе, отрицательное значение при ошибке, и положительное значение в изначальном процессе. Часто используется с exec() или схожими функциями.
2. **execvp** – заменяет образ текущего процесса. Получает на вход имя исполнимого файла, и список аргументов для сказанного файла. Является частью семьи функций execl\*.
3. **pipe** – позволяет использовать каналы для обмена данными между процессами.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, execl.
2. Понять как пользоваться потоками pipe.
3. Написать программы и организовать их взаимодействие.

**Основные файлы программы**

**whatt.c:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <errno.h>

#define PIPE\_READ 0

#define PIPE\_WRITE 1

int main(){

int num=255;

char\*inpt2=malloc(sizeof(char)\*num);

char\*inpt;

fgets (inpt2, num, stdin);

for(int i=0;i<num;i++){

if(inpt2[i]=='\n'){

num=i;

inpt=malloc(sizeof(char)\*num);

for(int ee=0;ee<num;ee++){

inpt[ee]=inpt2[ee];

}

free(inpt2);

break;

}

}

pid\_t pid;

int status;

int aPipe[2];

if (pipe(aPipe) < 0) {

close(aPipe[PIPE\_READ]);

close(aPipe[PIPE\_WRITE]);

perror("allocating pipe for child output redirect");

return -1;

}

pid=fork();

if(pid==0){

close(aPipe[PIPE\_READ]);

if (dup2(aPipe[PIPE\_WRITE], STDOUT\_FILENO) == -1) {

exit(errno);

}

char\*args[]={"./b.out",inpt,NULL};

execvp(args[0],args);

\_exit (EXIT\_FAILURE);

}

else if(pid<0){

printf("something went wrong with fork()\n");

}

else{

close(aPipe[PIPE\_WRITE]);

num=1;

inpt=realloc(inpt,sizeof(char)\*num);

int n;

while((n=read(aPipe[PIPE\_READ],inpt,num))!=0){

if(inpt[0]=='-'){break;}

printf("%c",inpt[0]);

}

close(aPipe[PIPE\_READ]);

}

return 0;

}

**whato.c**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

int w(int sb){

for(int i=2;i\*i<sb;i++){

if((sb%i)==0){

printf("%d\n",sb);

return 0;

}

}

printf("%d\n",-1);

return 1;

}

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

for(int i=0;i<argc;i++){

}

if( argc != 2 ) {

printf("bad command line arguments :(\n");

exit(-1);

}

freopen(argv[1], "r",stdin);

int subj;

while (scanf("%d", &subj) == 1){

if(w(subj)){

break;

}

}

return 0;

}

**Пример работы**

aaaaa@TRASHBox:~/Desktop/OC/laba2$ ./cc.out

bruh

Error: file not found

aaaaa@TRASHBox:~/Desktop/OC/laba2$ ./cc.out

input.txt

12

15

321

aaaaa@TRASHBox:~/Desktop/OC/laba2$ cat input.txt

12

15

321

7

8

aaaaa@TRASHBox:~/Desktop/OC/laba2$

**Вывод**

Работа с несколькими процессами довольно проста. Это хорошо