Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работ №2 по курсу**

**«Операционные системы»**

**Управление процессами в ОС**

Студент: Блистунова В.С.

Группа: М80 – 201Б-18

Вариант: 4

Преподаватель: Соколов А.А.

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

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

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Москва, 2019**

1. Постановка задачи

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 №4. Рекурсивная проверка простоты числа, где каждый отдельный уровень рекурсии вычисляется в отдельном процессе.

1. Ссылка на github

<https://github.com/val-lera/OS>

1. Описание программы

На вход программы поступает число. В рекурсивной функции Prime создается два процесса с помощью fork(). Дочерний процесс проверяет, делится ли исходное число на k (изначально равное двум). Если число делится без остатка на k, то дочерний процесс записывает в pipe ноль, иначе единицу. Родительский процесс, ожидающий завершения дочернего процесса с помощью wait(0), читает из pipe число. Если число равно нулю, то функция возвращает ноль, и программа выводит сообщение, что число не является простым; если число равно единице, то родительский процесс вызывает функцию Prime, только теперь k увеличивается на единицу. Программа выполняет проверку, пока исходное число больше k, в противном случае функция возвращает единицу

1. Набор testcases

|  |  |
| --- | --- |
| Входное число | Предполагаемый результат |
| 2 | Число 2 простое |
| 23 | Число 23 простое |
| 78 | Число 78 не является простым |
| 101 | Число 101 простое |
| 34 | Число 34 не является простым |

1. Результаты выполнения тестов.

Предполагаемые результаты совпали с фактическими

1. Листинг программы

#include<sys/wait.h>

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int Prime(int n,int k){

if(n==1) return 0;

if(n>k){

pid\_t child;

int fd[2];

if (pipe(fd) == -1){

write(STDERR\_FILENO, "error: pipe\n", sizeof(char));

exit(EXIT\_FAILURE);

}

int s;

child = fork();

if (child > 0){

close(fd[1]);

wait(0);

read(fd[0], &s, sizeof(s));

if(s==0)

return 0;

else return Prime(n,k+1);

}

else if (child == 0){

close(fd[0]);

if((n%k)==0)

s=0;

else

s=1;

write(fd[1], &s, sizeof(s));

if(write(fd[1], &s, sizeof(s)) == -1)

write(STDERR\_FILENO, "error: write\n", sizeof "error: write\n"-1);

exit(0);

}

else

write(STDERR\_FILENO, "error: fork\n", sizeof "error: fork\n"-1);

exit(EXIT\_FAILURE);;

}

else return 1;

}

int main()

{

char i[4];

int n;

write(STDERR\_FILENO, "Введите число: ", sizeof "Введите число: "-1);

//scanf("%d", &n);

read(STDIN\_FILENO, i, 4);

n=atoi(i);

if(Prime(n,2))

write(STDERR\_FILENO, "Число простое\n", sizeof "Число простое\n"-1);

else

write(STDERR\_FILENO, "Число не является простым\n", sizeof "Число не является простым\n"-1);

return 0;

}

7. Вывод.

В данной лабораторной работе я научилась работать с системными вызовами . Думаю, что такие системные вызовы как fork, wait могут быть полезными и в дальнейшем при работе с более сложными программами. Также в этой лабораторной работе для межпроцессорного взаимодействия я могла использовать как канал связи pipe, так сигналы. В этом случае я предпочла работать с pipe, так как этот способ показался мне удобнее.

Помимо управления процессами я получила опыт работы с командой man.