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: Ползикова Алина Владимировна

Группа: М8О-208Б-21

Вариант: 5

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/polzzzik/lab\_os

**Постановка задачи**

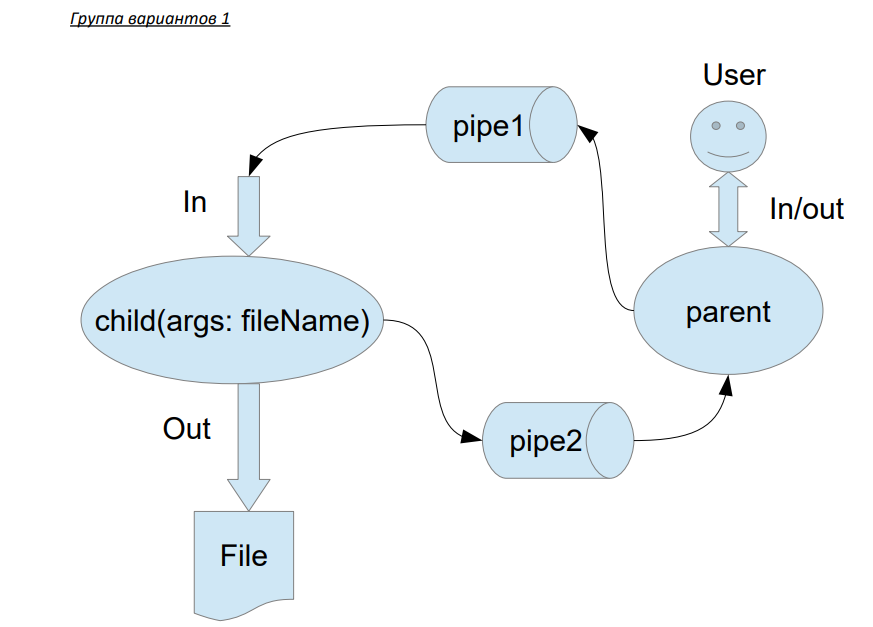
**Цель работы**

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.



Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль

родительского процесса пишет имя файла, которое будет передано при создании дочернего

процесса. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс передает команды пользователя через pipe1, который связан с

стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости

передает данные в родительский процесс через pipe2. Результаты своей работы дочерний

процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не

перенаправляя стандартный поток вывода.

**Общие сведения о программе**

Программа родительского процесса компилируется из parent.c, использует заголовочные файлы stdio.h, unistd.h, sys/stat.h, fcntl.h. Программа дочернего процесса компилируется из child.c, использует заголовочные файлы stdio.h, unistd.h, sys/stat.h, fcntl.h. В программах используются следующие системные вызовы:

1. mkfifo() – создание именованного канала
2. unlink() – удаление имени из файловой системы
3. fork() – создание дочернего процесса
4. open() – открытие файла
5. close() – закрытие файла
6. write() – запись последовательности байт
7. read() – чтение последовательности байт
8. execl() – замена образа памяти процесса
9. dup2() – переназначение файлового дескриптора

**Общий метод и алгоритм решения**

Родительский процесс получает имя файла, после чего создаётся дочерний процесс, при вызове execl() полученное имя файла передаётся в дочерний процесс в качестве аргументов командной строки. После того как оба процесса открыли каналы, они входят в циклы, условие выхода из которых – конец ввода. Родительский процесс передаёт введённое число в дочерний, после чего ждёт ответа от дочернего, первый байт в последовательности ответа – результат проверки (0 – число не подходит, 1 – число подходит)

**Исходный код**

|  |
| --- |
| **parent.c** |
| #include "unistd.h"  #include "stdio.h"  #include "sys/stat.h"  #include "sys/wait.h"  #include "fcntl.h"  #include "parent.h"  int ParentRoutine(FILE\* stream){  unlink("pipe1");  unlink("pipe2");  if (mkfifo("pipe1", S\_IREAD | S\_IWRITE) == -1 || mkfifo("pipe2", S\_IREAD | S\_IWRITE) == -1) {  perror("Parent: pipe create error");  return -1;  }  char\* fout;  size\_t k = 0;  int fout\_n = getline(&fout, &k, stream);  if (fout\_n <= 0) {  perror("Parent: file name error");  return -1;  }  int id = fork();  if (id == -1) {  perror("Parent: fork error");  return -1;  }  if (id != 0) {  int p1 = open("pipe1", O\_WRONLY);  int p2 = open("pipe2", O\_RDONLY);  if (p1 == -1 || p2 == -1) {  perror("Parent: pipe open error");  return -1;  }  char\* str;  size\_t n = 0;  int s = getline(&str, &n, stream);  while (s > 0) {  if (write(p1, str, s) == -1) {  perror("Parent: write error");  return -1;  }  char chek;  if (read(p2, &chek, 1) <= 0) {  perror("Parent: read error");  return -1;  }  if (chek == '0') {  printf("Parent: number is prime or negative\n");  return 0;  }    s = getline(&str, &n, stream);    }  wait(NULL);  close(p1);  close(p2);  unlink("pipe1");  unlink("pipe2");    } else {  fout[fout\_n - 1] = '\0';  char\* argv[3];  argv[0] = "child.c";  argv[1] = fout;  argv[2] = NULL;  if (execv("child.out", argv) == -1) {  perror("Child: exec error");  return -1;  }  }  return 0;  } |

|  |
| --- |
| **child.c** |
| #include "unistd.h"  #include "stdio.h"  #include "stdlib.h"  #include "sys/stat.h"  #include "fcntl.h"  int IsPrime(long long n) {  if (n <= 1) {  return 1;  }  for (long long i = 2; i\*i <= n; i++) {  if (n%i == 0) {  return 0;  }  }  return 1;  }  int main(const int argc, char\* argv[]) {  int p1 = open("pipe1", O\_RDONLY);  int p2 = open("pipe2", O\_WRONLY);  if (argc != 2) {  printf("Necessary arguments were not provided\n");  exit(EXIT\_FAILURE);  }  if (p1 == -1 || p2 == -1) {  perror("Child: pipe open error");  exit(EXIT\_FAILURE);  }  unlink(argv[1]);  int fout = open(argv[1], O\_CREAT | O\_WRONLY, S\_IREAD | S\_IWRITE);  if (fout == -1) {  perror("Child: file error");  exit(EXIT\_FAILURE);  }    if (dup2(p1, 0) == -1 || dup2(fout, 1) == -1) {  perror("Child: dup error");  exit(EXIT\_FAILURE);  }  char\* str;  size\_t n = 0;  int s = getline(&str, &n, stdin);  long long num = atol(str);  char chek[2] = "01";  while (s > 0) {  if (IsPrime(num) == 1) {  if (write(p2, &chek[0], 1) == -1) {  perror("Child: write error");  }  close(fout);  close(p1);  close(p2);  exit(EXIT\_FAILURE);  } else {  printf("%s", str);  fflush(stdout);  if (write(p2, &chek[1], 1) == -1) {  perror("Child: write error");  exit(EXIT\_FAILURE);  }  }  s = getline(&str, &n, stdin);  num = atol(str);  }  close(p1);  close(p2);  close(fout);  } |

**Демонстрация работы программы**

OS-labs/build/lab2$ ./parent.out

file

80000

4

406

99999999999

-1337

Parent: number is prime or negative

OS-labs/build/lab2$ cat file

80000

4

406

99999999999

**Выводы**

Составлена и отлажена программа на языке Си, осуществляющая работу с процессами. Тем самым, приобретены навыки в управлении процессами в ОС и обеспечении обмена данных между процессами посредством каналов.