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: Знай Артемий Олегович

Группа: М8О-201Б-21

Вариант: 18

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/znako/OS\_LABS/

**Постановка задачи**

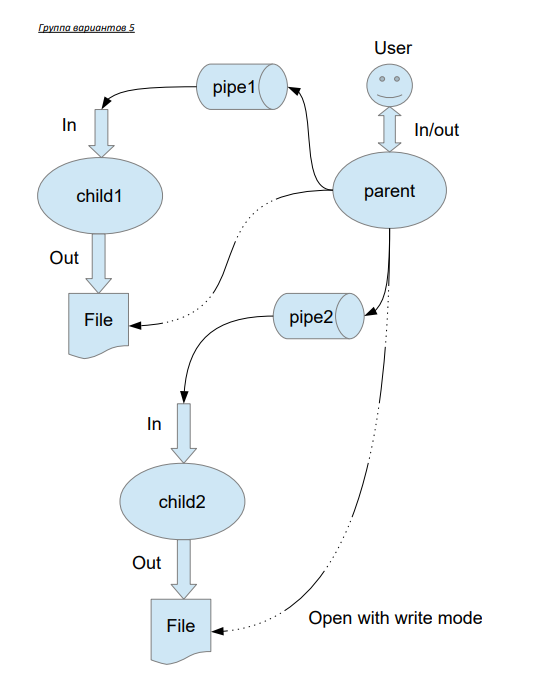
**Цель работы**

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 два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод. Вариант 18) Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы удаляют все гласные из строк.

**Общие сведения о программе**

Программа родительского процесса компилируется из parent.c, использует заголовочные файлы parent.h, utils.h. Программа дочернего процесса компилируется из child.c, использует заголовочные файлы utils.h. В программах используются следующие системные вызовы:

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

**Общий метод и алгоритм решения**

Родительский процесс получает имя двух файлов, после чего создаются два дочерних процесса, при вызове execv() полученное имя файла передаётся в дочерний процесс в качестве аргументов командной строки. После чего родительский процесс получает строки, определяет в какой дочерний процесс ее отправить и передает. Дочерний процесс отфильтровывает гласные буквы из строки и выводит в свой файл.

**Исходный код**

|  |
| --- |
| **parent.c** |
| #include "parent.h"  #include "utils.h"  int ChoosePipe(int len){  return (len - 1) % 2 != 0;  }  void ParentRoutine(char\* pathToChild, FILE\* fin)  {  char\* fileName1 = ReadString(fin, &FilterNone);  char\* fileName2 = ReadString(fin, &FilterNone);  fileName1[strlen(fileName1) - 1] = '\0';  fileName2[strlen(fileName2) - 1] = '\0';  unlink(fileName1);  unlink(fileName2);  int fd1[2], fd2[2];  if (pipe(fd1) == -1 || pipe(fd2) == -1)  {  perror("creating pipe error )");  exit(EXIT\_FAILURE);  }  int outputFile1, outputFile2;  if ((outputFile1 = open(fileName1, O\_WRONLY | O\_CREAT, S\_IRWXU)) < 0)  {  perror("opening output file 1 error )");  exit(EXIT\_FAILURE);  }  if ((outputFile2 = open(fileName2, O\_WRONLY | O\_CREAT, S\_IRWXU)) < 0)  {  perror("opening output file 2 error )");  exit(EXIT\_FAILURE);  }  free(fileName1);  free(fileName2);  char\* argv[2];  argv[0] = "child";  argv[1] = NULL;  pid\_t pid1 = fork();  pid\_t pid2 = 1;  if (pid1 > 0){  pid2 = fork();  }  if (pid1 < 0 || pid2 < 0)  {  perror("process error )");  exit(EXIT\_FAILURE);  }  if (pid1 == 0)  {  close(fd1[1]);  if (dup2(fd1[0], 0) < 0)  {  perror("duping pipe error )");  exit(EXIT\_FAILURE);  }  if (dup2(outputFile1, 1) < 0)  {  perror("duping output file error )");  exit(EXIT\_FAILURE);  }  execv(pathToChild, argv);  }  else if (pid2 == 0)  {  close(fd2[1]);  if (dup2(fd2[0], 0) < 0)  {  perror("duping pipe error )");  exit(EXIT\_FAILURE);  }  if (dup2(outputFile2, 1) < 0)  {  perror("duping output file error )");  exit(EXIT\_FAILURE);  }  execv(pathToChild, argv);  }  else  {  close(fd1[0]);  close(fd2[0]);  char\* strInput = NULL;  while ((strInput = ReadString(fin, &FilterNone)) != NULL)  {  int strSize = strlen(strInput);  if (strSize > 0)  {  if (ChoosePipe(strSize))  {  write(fd1[1], strInput, strSize);  }  else  {  write(fd2[1], strInput, strSize);  }  }  free(strInput);  }  close(fd1[1]);  close(fd2[1]);  }  } |

|  |
| --- |
| **child.c** |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #include <fcntl.h>  #include <ctype.h>  #include <sys/wait.h>  #include <sys/stat.h>  #include "utils.h"  int main(int argc, char\* argv[])  {  if (argc < 1)  {  perror("too few arguments )");  exit(EXIT\_FAILURE);  }    char\* strInput;  while ((strInput = ReadString(stdin, &FilterVowels)) != NULL)  {  write(1, strInput, strlen(strInput));  free(strInput);  }  return 0;  } |

|  |
| --- |
| **utils.c** |
| #include <utils.h>  char\* ReadString(FILE\* stream, TFilter filter)  {  if(feof(stream)) {  return NULL;  }  const int chunkSize = 256;  char\* buffer = (char\*)malloc(chunkSize);  int bufferSize = chunkSize;  if(buffer == NULL)  {  printf("Couldn't allocate buffer");  exit(EXIT\_FAILURE);  }  int readChar;  int idx = 0;  while ((readChar = getc(stream)) != EOF)  {  if(filter(readChar)){  continue;  }  buffer[idx++] = readChar;  if (idx == bufferSize)  {  buffer = realloc(buffer, bufferSize + chunkSize);  bufferSize += chunkSize;  }  if (readChar == '\n') {  break;  }  }  buffer[idx] = '\0';  return buffer;  }  int FilterNone(char c)  {  return c-c;  }  int FilterVowels(char c)  {  char\* vowels = {"AEIOUYaeiouy"};  for (int i = 0; i < (int)strlen(vowels); ++i)  {  if (c == vowels[i]){  return 1;  }  }  return 0;  } |

**Демонстрация работы программы**

znako@znako-VirtualBox:~/UtoW/OS\_LABS/build/lab2$ ./lab2

o1.txt

o2.txt

aksjgksag

kasjg

kajsgkjsgs

znako@znako-VirtualBox:~/UtoW/OS\_LABS/build/lab2$ cat o1.txt

ksjgksg

ksjg

znako@znako-VirtualBox:~/UtoW/OS\_LABS/build/lab2$ cat o2.txt

kjsgkjsgs

**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.