Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**Процессы в операционных системах**

| Студент | Гришин Алексей Юрьевич |
| --- | --- |
| Группа | М8О-208Б-21 |
| Вариант | 17 |
| Преподаватель | Соколов Андрей Алексеевич |
| Оценка | 5 |
| Дата | 03.10.2022 |
| Подпись |  |

Москва, 2022.

**Постановка задачи**

## Цель работы

## Приобретение практических навыков в:

## Управлении процессами в ОС

## Обеспечении обмена данных между процессами посредством каналов

## Задание

Составить и отладить программу на языке С, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решения задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы / события / или каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

## Общие сведения о программе

Программа состоит из двух частей: основной и дочерней.

Основная программа компилируется из файла manager.c. Задача основной программы заключается в общении с пользователем, считывании входных значений и делегировании дальнейшей работы одной из двух дочерних программ. Также, основная программа использует заголовочные файлы string.h, stdlib.h, unistd.h, fcntl.h, sys/wait.h, sys/types.h, sys/stat.h и вспомогательные модули io\_line и pipe.

Дочерняя программа компилируется из файла worker.c. Она запускается основной программой через системный вызов execv. Общение основной программы с дочерними происходят через каналы. Также, дочерняя программа использует заголовочные файлы: string.h, stdlib.h, fcntl.h, unistd.h. Для своей работы дочерняя программа использует вспомогательные модули io\_line и remove\_vowels.

Модуль io\_line создан с целью упрощения считывания и печатания строк. В него входят такие методы, как write\_line и read\_line, которые реализуют вывод и считывание строки соответственно.

Модуль pipe создан с целью абстрагирования от конкретной реализации каналов операционной системой, чтобы в будущем было легче переписать программы под операционную систему Windows. Модуль включает в себя методы init\_pipe, set\_mode, get\_input\_descriptor, get\_output\_descriptor. Метод init\_pipe предназначен для инициализации канала, set\_mode – для установки режима, в котором будет использоваться канал (получение или отправка сообщений), get\_input\_descriptor – для получения дескриптора входного файла, get\_output\_descriptor – для получения дескриптора выходного файла.

Метод remove\_vowels решает само задание – удаление из входной строки всех гласных букв. Имеет один метод remove\_vowels, который и выполняет поставленную задачу.

В программе используются следующие системные вызовы:

1. execv – выполняет программу, относительный путь к которой указан в filename. Аргументы командной строки передаются вторым аргументом в качестве массива C-строк (указатель char\*), последний элемент которого обязательно должен быть NULL. При успешном выполнении возвращает 0, в противном случае - -1.
2. dup2 – создает копию дескриптора, переданного в качестве первого аргумента. Дескриптор, который представляет собой копию, передается в качестве второго аргумента.
3. exit – немедленно останавливает текущую программу. В качестве аргумента передается статус, с которым завершается программа.
4. close – закрывает файл, используя дескриптор, переданный в качестве аргумента
5. open – открывает файл. Путь к файлу передается в качестве первого аргумента в виде строки. При успешном выполнении возвращает число, являющееся файловым дескриптором открытого файла, в противном случае возвращает -1. Существует 2 версии: с двумя аргументами и с трема. Первая версия ожидает в качестве второго аргумента флаги, с которыми нужно открыть файл (если необходимо использовать множество флагов, то используется побитовое «или». Вторая версия позволяет указать атрибуты доступа к файлу.
6. read – предназначен для считывания данных с открытого файла, используя дескриптор, переданный в качестве первого аргумента. Вторым и третьим аргументом должны передаться указатель на область памяти, куда необходимо занести считанную информацию, и размер этой области (в байтах). Возвращаемое значение – количество считанных байт.
7. write – предназначен для записи информации в открытый файл, дескриптор которого указан в качестве первого аргумента. Также, как и в системном вызове read, вторым и третьим аргументами служат указатель на области памяти, которую необходимо записать в файл и ее размер в байтах соответственно. Возвращаемое значение – количество записанных байт.
8. fork – клонирует текущий процесс. При этом создается иерархия в виде дочернего и родительского процесса. Возвращает число, которое имеет разную семантику в зависимости от его значения. Если возвращаемое число равно 0, то текущий процесс – дочерний, если больше нуля – родительский и обозначает идентификатор дочернего процесса. Если же возвращаемое число отрицательное, то при клонировании процесса произошла ошибка.
9. waitpid – ожидает дочерний процесс или группу процессов по его идентификатору, переданному в качестве первого аргумента. В качестве второго аргумента передается указатель на область в памяти, куда занесется статус, с которым завершился процесс. Если было передано значение NULL, то запись не производится. В качестве третьего аргумента можно передать дополнительные опции для выполнения ожидания. При успешном выполнении возвращает идентификатор процесса, состояние которого изменилось, в противном случае возвращает -1.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы open, read, write, close.
2. Изучить принципы работы fork, waitpid
3. Написать библиотеку io\_line для работы со считыванием и записью строк
4. Написать библиотеку pipe для абстрагирования работы с каналами
5. Написать библиотеку remove\_vowels для удаления гласных букв в строке

**Основные файлы программы**

**io\_line/io\_line.h**

| #ifndef IO\_LINE\_H  #define IO\_LINE\_H  #include <stdlib.h>  #define IO\_LINE\_OK 0  #define IO\_LINE\_INVALID\_DESCRIPTOR 11  #define IO\_LINE\_ACCESS\_DENIED 12  #define IO\_LINE\_NOT\_ENOUGH\_BUFFER\_SIZE 13  typedef int Descriptor;  typedef char\* String;  typedef size\_t StringSize;  int write\_line(Descriptor d, String line);  int read\_line(Descriptor d, String buffer, StringSize buffer\_size);  #endif |
| --- |

**io\_line/io\_line.c**

| #include "io\_line.h"  #include <stdio.h>  #include <unistd.h>  #include <string.h>  #include <errno.h>  #include <stdbool.h>  bool \_\_is\_endline\_symbol(char symbol) {  return (symbol == EOF || symbol == '\n');  }  bool \_\_try\_to\_write(Descriptor d, void \* data, size\_t data\_size) {  return write(d, data, data\_size) == data\_size;  }  bool \_\_try\_to\_read(Descriptor d, void \* data, size\_t data\_size) {  return read(d, data, data\_size) == data\_size;  }  int \_\_handle\_file\_error(int error) {  if (error == EACCES)  return IO\_LINE\_ACCESS\_DENIED;  if (error == EBADF)  return IO\_LINE\_INVALID\_DESCRIPTOR;  }  int \_\_get\_char(Descriptor d, char \* symbol) {  if (!\_\_try\_to\_read(d, symbol, 1 \* sizeof(char)))  return \_\_handle\_file\_error(errno);  return IO\_LINE\_OK;  }  int write\_line(Descriptor d, String line) {  StringSize line\_length = strlen(line);  if (!\_\_try\_to\_write(d, line, line\_length \* sizeof(char)))  return \_\_handle\_file\_error(errno);  if (!\_\_try\_to\_write(d, "\n", 1 \* sizeof(char)))  return \_\_handle\_file\_error(errno);  return IO\_LINE\_OK;  }  int read\_line(Descriptor d, String buffer, StringSize buffer\_size) {  char symbol;  size\_t index = 0;  int result;  while (1) {  result = \_\_get\_char(d, & symbol);  if (result != IO\_LINE\_OK)  return result;  if (\_\_is\_endline\_symbol(symbol))  break;  if (symbol == '\r')  continue;  if (index >= buffer\_size - 1)  return IO\_LINE\_NOT\_ENOUGH\_BUFFER\_SIZE;  buffer[index++] = symbol;  }  buffer[index] = '\0';  return IO\_LINE\_OK;  } |
| --- |

**pipe/pipe.h**

| #ifndef \_\_PIPE\_H\_\_  #define \_\_PIPE\_H\_\_  #include <stdbool.h>  #define PIPE\_OK 0  #define PIPE\_ERROR\_INIT\_PIPE 21  #define PIPE\_ERROR\_MODE\_ALREADY\_DEFINED 22  #define PIPE\_ERROR\_ACCESS\_DENIED 23  typedef enum {  WRITE,  READ  } PipeMode;  struct \_\_Pipe {  int input\_dp;  int output\_dp;  PipeMode mode;  bool mode\_defined;  };  typedef struct \_\_Pipe Pipe;  int init\_pipe(Pipe \* p);  int set\_mode(Pipe \* p, PipeMode mode);  int get\_input\_descriptor(Pipe \* p, int \* out\_value);  int get\_output\_descriptor(Pipe \* p, int \* out\_value);  void close\_pipe(Pipe \* p);  #endif |
| --- |

**pipe/pipe.c**

| #include "pipe.h"  #include <unistd.h>  int init\_pipe(Pipe \* p) {  int dp[2];  if (pipe(dp) != 0)  return PIPE\_ERROR\_INIT\_PIPE;  p -> input\_dp = dp[0];  p -> output\_dp = dp[1];  p -> mode\_defined = false;  return PIPE\_OK;  }  int set\_mode(Pipe \* p, PipeMode mode) {  if (p -> mode\_defined)  return PIPE\_ERROR\_MODE\_ALREADY\_DEFINED;  if (mode == WRITE)  close(p -> input\_dp);  if (mode == READ)  close(p -> output\_dp);  p -> mode = mode;  p -> mode\_defined = true;  return PIPE\_OK;  }  int get\_input\_descriptor(Pipe \* p, int \* out\_value) {  if (!p -> mode\_defined || p -> mode != READ)  return PIPE\_ERROR\_ACCESS\_DENIED;  \* out\_value = p -> input\_dp;  return PIPE\_OK;  }  int get\_output\_descriptor(Pipe \* p, int \* out\_value) {  if (!p -> mode\_defined || p -> mode != WRITE)  return PIPE\_ERROR\_ACCESS\_DENIED;  \* out\_value = p -> output\_dp;  return PIPE\_OK;  }  void close\_pipe(Pipe \* p) {  if (!p -> mode\_defined || p -> mode == WRITE)  close(p -> output\_dp);  if (!p -> mode\_defined || p -> mode == READ)  close(p -> input\_dp);  p -> mode\_defined = false;  } |
| --- |

**remove\_vowels/remove\_vowels.h**

| #ifndef \_\_REMOVE\_VOWELS\_H\_\_  #define \_\_REMOVE\_VOWELS\_H\_\_  #include <stdlib.h> // size\_t  #define REMOVE\_VOWELS\_OK 0  #define REMOVE\_VOWELS\_ERROR\_NOT\_ENOUGH\_BUFFER\_LEN 31  int remove\_vowels(char \* s, char \* buff, size\_t buff\_size);  #endif |
| --- |

**remove\_vowels/remove\_vowels.c**

| #include "remove\_vowels.h"  #include <string.h>  #include <stdbool.h>  char \* \_\_vowels = "eyuioaEYUIOA";  bool \_\_is\_vowel(char symbol) {  for (char \* c = \_\_vowels;\* c != '\0'; c++)  if ( \* c == symbol)  return true;  return false;  }  int remove\_vowels(char \* s, char \* buff, size\_t buff\_size) {  int index = 0;  size\_t s\_length = strlen(s);  int ret\_value = REMOVE\_VOWELS\_OK;  for (int j = 0;  (j < s\_length) && (index < buff\_size - 1); j++) {  if (\_\_is\_vowel(s[j]))  continue;  buff[index++] = s[j];  }  buff[index] = '\0';  if (index == buff\_size - 1)  return REMOVE\_VOWELS\_ERROR\_NOT\_ENOUGH\_BUFFER\_LEN;  return REMOVE\_VOWELS\_OK;  } |
| --- |

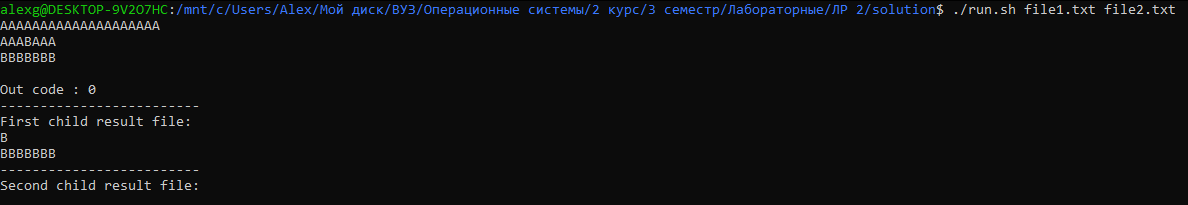
**manager.c**

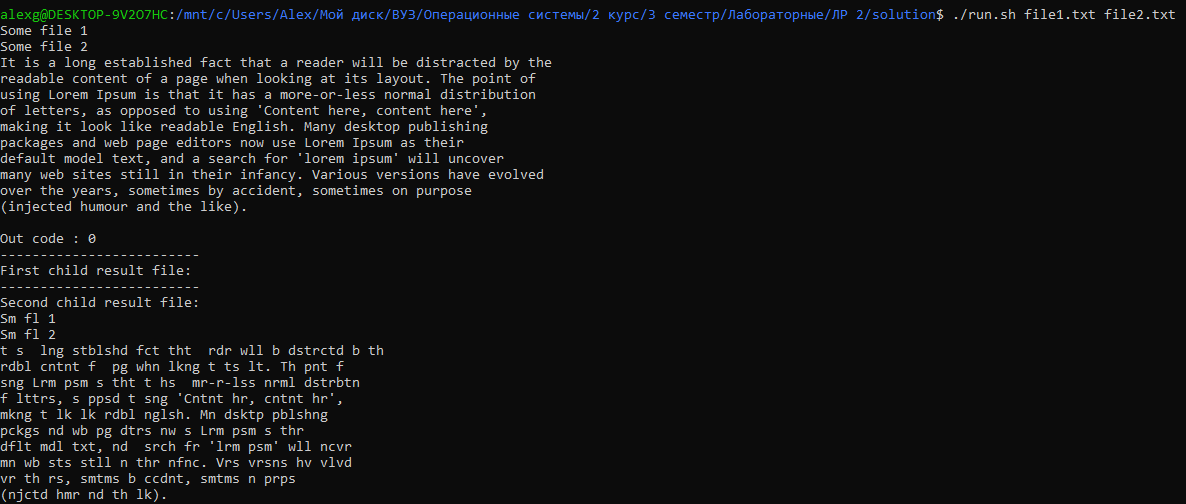
| #include "io\_line/io\_line.h"  #include "pipe/pipe.h"  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #include <fcntl.h>  #include <sys/wait.h>  #include <sys/types.h>  #include <sys/stat.h>  #define MAX\_FILEPATH\_LEN 256  #define MAX\_INPUT\_STRING\_LEN 256  #define CREATE\_PROCESS\_ERROR 41  #define CREATE\_PIPE\_ERROR 42  #define WORKER\_ERROR\_EXECUTE\_FILE 43  #define MANAGER\_ERROR\_CREATE\_RESULT\_FILE 44  #define MANAGER\_ERROR\_RESULT\_FILE\_NOT\_SPECIFIED 45  #define MANAGER\_ERROR\_OPEN\_RESULT\_FILE 46  #define WORKER\_PATH "worker"  void protected\_create\_file(char \* filename) {  int descriptor;  descriptor = open(filename, O\_CREAT, S\_IWRITE | S\_IREAD);  if (descriptor == -1)  exit(MANAGER\_ERROR\_CREATE\_RESULT\_FILE);  close(descriptor);  }  char \* protected\_get\_result\_file(int argc, char \* argv[], int index) {  int descriptor;  if (index >= argc)  exit(MANAGER\_ERROR\_RESULT\_FILE\_NOT\_SPECIFIED);  if (access(argv[index], F\_OK) != 0)  protected\_create\_file(argv[index]);  descriptor = open(argv[index], O\_WRONLY);  if (descriptor == -1)  exit(MANAGER\_ERROR\_OPEN\_RESULT\_FILE);  close(descriptor);  return argv[index];  }  void init\_worker(int input\_file\_descriptor, char \* result\_filepath) {  char \* args[] = {  WORKER\_PATH,  result\_filepath,  NULL  };  dup2(input\_file\_descriptor, STDIN\_FILENO);  execv(WORKER\_PATH, args);  exit(WORKER\_ERROR\_EXECUTE\_FILE);  }  void protected\_init\_pipe(Pipe \* p) {  if (init\_pipe(p) != PIPE\_OK)  exit(res);  }  int protected\_get\_input\_descriptor(Pipe \* p) {  int pipe\_dp;  int result;  result = get\_input\_descriptor(p, & pipe\_dp);  if (result != PIPE\_OK)  exit(result);  return pipe\_dp;  }  int protected\_get\_output\_descriptor(Pipe \* p) {  int pipe\_dp;  int result;  result = get\_output\_descriptor(p, & pipe\_dp);  if (result != PIPE\_OK)  exit(result);  return pipe\_dp;  }  void send\_data(int pipe\_descriptor, char \* data) {  size\_t string\_len;  string\_len = strlen(data);  data[string\_len] = '\n';  write(pipe\_descriptor, data, string\_len + 1);  }  pid\_t create\_worker(Pipe \* p, char \* result\_filepath) {  pid\_t pid = fork();  if (pid < 0)  exit(CREATE\_PROCESS\_ERROR);  if (pid == 0) {  set\_mode(p, READ);  init\_worker(protected\_get\_input\_descriptor(p), result\_filepath);  }  set\_mode(p, WRITE);  return pid;  }  void close\_worker(pid\_t worker\_id, int pipe\_descriptor) {  write(pipe\_descriptor, "\n", 1);  waitpid(worker\_id, NULL, 0);  }  int main(int argc, char \* argv[]) {  char \* first\_filepath;  char \* second\_filepath;  char input\_string[MAX\_INPUT\_STRING\_LEN];  pid\_t first\_worker\_id, second\_worker\_id;  Pipe pipe1, pipe2;  first\_filepath = protected\_get\_result\_file(argc, argv, 1);  second\_filepath = protected\_get\_result\_file(argc, argv, 2);  protected\_init\_pipe( & pipe1);  first\_worker\_id = create\_worker( & pipe1, first\_filepath);  protected\_init\_pipe( & pipe2);  second\_worker\_id = create\_worker( & pipe2, second\_filepath);  while (1) {  int res = read\_line(  STDIN\_FILENO,  input\_string,  MAX\_INPUT\_STRING\_LEN);  if (res != IO\_LINE\_OK)  exit(res);  if (strlen(input\_string) == 0)  break;  if (strlen(input\_string) > 10)  send\_data(  protected\_get\_output\_descriptor( & pipe2),  input\_string);  else  send\_data(  protected\_get\_output\_descriptor( & pipe1),  input\_string);  }  close\_worker(  first\_worker\_id,  protected\_get\_output\_descriptor( & pipe1));  close\_worker(  second\_worker\_id,  protected\_get\_output\_descriptor( & pipe2));  exit(0);  } |
| --- |

**worker.c**

| #include "io\_line/io\_line.h"  #include "remove\_vowels/remove\_vowels.h"  #include <string.h>  #include <stdlib.h>  #include <fcntl.h>  #include <unistd.h>  #define RESULT\_FILE\_NOT\_SPECIFIED 51  #define FAILED\_TO\_OPEN\_RESULT\_FILE 52  #define MAX\_INPUT\_LINE\_LEN 256  int protected\_open\_result\_file\_descriptor(int argc, char \* argv[]) {  int descriptor;  if (argc < 2)  exit(RESULT\_FILE\_NOT\_SPECIFIED);  descriptor = open(argv[1], O\_WRONLY | O\_TRUNC)  if (descriptor == -1)  exit(FAILED\_TO\_OPEN\_RESULT\_FILE);  return descriptor;  }  void protected\_get\_data(int descriptor, char \* buff, size\_t buff\_size) {  int result = read\_line(descriptor, buff, buff\_size);  if (result != IO\_LINE\_OK)  exit(result);  }  void protected\_write\_line(int descriptor, char \* line) {  int result = write\_line(descriptor, line);  if (result != IO\_LINE\_OK)  exit(result);  }  void protected\_format\_line(char \* input\_line, char \* buff, size\_t buff\_size) {  int result = remove\_vowels(input\_line, buff, buff\_size);  if (result != REMOVE\_VOWELS\_OK)  exit(result);  }  int main(int argc, char \* argv[]) {  int result\_file\_descriptor;  char input\_line[MAX\_INPUT\_LINE\_LEN];  char formatted\_line[MAX\_INPUT\_LINE\_LEN];  result\_file\_descriptor = protected\_open\_result\_file\_descriptor(  argc, argv);  protected\_get\_data(STDIN\_FILENO, input\_line, MAX\_INPUT\_LINE\_LEN);  while (strlen(input\_line) != 0) {  protected\_format\_line(  input\_line, formatted\_line, MAX\_INPUT\_LINE\_LEN);  protected\_write\_line(result\_file\_descriptor, formatted\_line);  protected\_get\_data(STDIN\_FILENO, input\_line, MAX\_INPUT\_LINE\_LEN);  }  close(result\_file\_descriptor);  return 0;  } |
| --- |

**Пример работы**

****

****

**Вывод**

В ходе выполнения данной лабораторной работы я получил навыки в разработке много процессных программ, изучил инструменты для межпроцессного взаимодействия. Ознакомился с основными системными вызовами Unix для работы с файлами, каналами, заменой образа программы.

Также, в ходе выполнения данной лабораторной работы я познакомился с особенностями организации памяти в программах, написанных на языке C, столкнулся с рядом проблем при передаче данных с использованием статических объектов.