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: Демина В.А.

Группа: М8О–306Б–19

Вариант: 22

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

**Цель работы**

Целью является приобретение практических навыков в:

* Управление потоками в ОС
* Обеспечение обмена данных между процессами посредством каналов

**Задание**

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 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.

Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы инвертируют строки.

**Общие сведения о программе**

Программа компилируется из файла main.c. Подключены заголовочные файлы: unistd.h, fcntl.h, stdlib.h, time.h, string.h. В программе используются следующие системные вызовы:

**pipe** – принимает массив из двух целых чисел, в случае успеха массив будет содержать два файловых дескриптора, которые будут использоваться для конвейера, первое число в массиве предназначено для чтения, второе для записи, а также вернется 0. В случае неуспеха вернется -1.

**fork** – создает новый процесс, который является копией родительского процесса, за исключением разных process ID и parent process ID. В случае успеха fork() возвращает 0 для ребенка, число больше 0 для родителя – child ID, в случае ошибки возвращает -1.

**open** – создает или открывает файл, если он был создан. В качестве аргументов принимает путь до файла, режим доступа (запись, чтение и т.п.), модификатор доступа ( при создании можно указать права для файла ). Возвращает в случае успеха файловый дескриптор – положительное число, иначе возвращает -1.

**close** – принимает файловый дескриптор в качестве аргумента, удаляет файловый дескриптор из таблицы дескрипторов, в случае успеха вернет 0, в случае неуспеха вернет -1.

**read** – предназначена для чтения какого-то числа байт из файла, принимает в качестве аргументов файловый дескриптор, буфер, в который будут записаны данные и число байт. В случае успеха вернет число прочитанных байт, иначе -1.

**write** – предназначена для записи какого-то числа байт в файл, принимает в качестве аргументов файловый дескриптор, буфер, из которого будут считаны данные для записи и число байт. В случае успеха вернет число записанных байт, иначе -1.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы pipe и fork.
2. Написать функцию считывания имён выходных файлов
3. Создать каналы связи для каждого из дочерних процессов
4. Создать функцию обработки ввода
5. Создать функцию фильтрации в родительском процессе
6. Создать функцию фильтрации в дочерних процессах
7. Написать обработку ошибок

**Основные файлы программы**

**main.c:**

#include <fcntl.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <time.h>

void reverse(char name[]) // разворачивает строки

{

    char tmp;

    int len = strlen(name);

    --len;

    for (int i = 0; i < len; i++) {

        tmp = name[len - 1];

        name[len - 1] = name[i];

        name[i] = tmp;

        len--;

    }

}

void child\_work(int from, int to) {

    char buf;

    char string[100];

    int i = 0;

    while (read(from, &buf, 1) > 0) {

        string[i++] = buf;

        string[i] = '\0';

        if ((buf == '\n')||(buf == '\r')) {

            reverse(string);

            write(to, string, i);

            i = 0;

        }

    }

    close(to);

    close(from);

}

void parrent\_work(int child1, int child2) {

  char buf[1];

  srand(time(NULL));

  int n = (rand() % 10) + 1;

  while (read(STDIN\_FILENO, buf, 1) > 0) {

    if (n > 2) {

      write(child1, buf, 1);

    } else {

      write(child2, buf, 1);

    }

    if ((buf[0] == '\n')||(buf[0] == '\r')) {

      n = (rand() % 10) + 1;

    }

  }

  close(child1);

  close(child2);

}

int read\_name\_and\_open\_file() {

  const size\_t FILE\_NAME\_SIZE = 64;

  char f\_name[FILE\_NAME\_SIZE];

  char buf[1];

  int idx = 0;

  while (idx < FILE\_NAME\_SIZE && read(STDIN\_FILENO, buf, 1) > 0) {

      // reed(файловый дескриптор, буфер для записи, число байт)

      // STDIN\_FILENO - поток ввода(терминал)

    if (buf[0] == '\n') {

      break;

    }

    f\_name[idx++] = buf[0];

  }

  f\_name[idx++] = '\0';

  return open(f\_name, O\_WRONLY | O\_TRUNC);

  // open(имя файла(путь), режим доступа)

  // O\_WRONLY - только запись

  // O\_TRUNC - очистить при необходимости

}

void error(char\* buf, size\_t size) { // посимвольно выводит сообщение об ошибке

    write(STDERR\_FILENO, buf, size);

    // write(файловый дескриптор, буфер из которого читается, число байт)

    // STDERR\_FILENO - поток диагностики(вывода ошибок)

}

void check\_file\_id(int id) {

  if (id == -1) {

    error("File not found\n", 15);

    exit(-1);

  }

}

void check\_pipe\_creation(int\* pipefd) {

    if (pipe(pipefd) == -1) {

        error("Can not create pipe\n", 20);

        exit(-2);

    }

}

int check\_fork() {

    int fd = fork();

    if (fd == -1) {

        error("Can not create process\n", 23);

        exit(-3);

    }

    return fd;

}

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

    int f1 = read\_name\_and\_open\_file();

    check\_file\_id(f1);

    int f2 = read\_name\_and\_open\_file();

    check\_file\_id(f2);

    int pipefd1[2];

    check\_pipe\_creation(pipefd1);

    int child1 = check\_fork();

    if (child1 == 0) {

        close(pipefd1[1]);

        child\_work(pipefd1[0], f1);

        return 0;

    }

    close(pipefd1[0]);

    int pipefd2[2];

    check\_pipe\_creation(pipefd2);

    int child2 = check\_fork();

    if (child2 == 0) {

        close(pipefd1[1]);

        close(pipefd2[1]);

        child\_work(pipefd2[0], f2);

        return 0;

    }

    close(pipefd2[0]);

    parrent\_work(pipefd1[1], pipefd2[1]);

    return 0;

}

**Вывод**

В процессе работы над лабораторной я научился основам работы с конвейерами и процессами в Си. Процессы занимают важную роль в разработке ПО, так как программы зачастую состоят из нескольких, относительно обособленных, подпрограмм, то есть процессов. Конвейеры как один из способов обмена данными между процессами таже играют немаловажную роль.