

**Московский авиационный институт
(национальный исследовательский университет)**

**Факультет информационных технологий и прикладной
математики
Кафедра вычислительной математики и программирования**

Лабораторная работа №2 по курсу «Операционные системы»

Межпроцессорное взаимодействие

Студент: Тумаков Данила Владимирович
Преподаватель: Соколов Андрей Алексеевич
Группа: М8О-206Б-19
Дата:
Оценка:
Подпись:

Москва, 2021

1 Постановка задачи

Цель работы:

Приобретение практических навыков в:

- Управление процессами в ОС
- Обеспечение обмена данных между процессами посредством каналов

Задание (вариант 18):

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.

Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы удаляют все гласные из строк.

2 Общие сведения о программе

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

1. **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 так же вернется 0. В случае неуспеха вернется -1.
2. **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.
3. **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.
4. **close** — принимает файловый дескриптор в качестве аргумента, удаляет файловый дескриптор из таблицы дескрипторов, в случае успеха вернет 0, в случае неуспеха вернет -1.
5. **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.
6. **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.

3 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
8. Написать тесты

4 Исходный код

main.c

```
1
2 #include <fcntl.h>
3 #include <stdlib.h>
4 #include <stdio.h>
5 #include <unistd.h>
6
7 void child_work(int from, int to) {
8     char buf[1];
9     while (read(from, buf, 1) > 0) {
10         char c = buf[0];
11         if (c != 'a' && c != 'e' && c != 'i' && c != 'o' && c != 'u' && c != 'y' &&
12             c != 'A' && c != 'E' && c != 'I' && c != 'O' && c != 'U' && c != 'Y') {
13             write(to, buf, 1);
14         }
15     }
16     close(to);
17     close(from);
18 }
19
20 void parent_work(int child1, int child2) {
21     char buf[1];
22     int is_even = 0;
23     while (read(STDIN_FILENO, buf, 1) > 0) {
24         if (!is_even) {
25             write(child1, buf, 1);
26         } else {
27             write(child2, buf, 1);
28         }
29         if (buf[0] == '\n') {
30             is_even = !is_even;
31         }
32     }
33
34     close(child1);
35     close(child2);
36 }
37
38 int open_file() {
39     const size_t NAME_SIZE = 64;
40     char f_name[NAME_SIZE];
41     char buf[1];
42     int idx = 0;
43     while (idx < NAME_SIZE && read(STDIN_FILENO, buf, 1) > 0) {
44         if (buf[0] == '\n') {
45             break;
46         }
47     }
```

```

47     f_name[idx++] = buf[0];
48 }
49 f_name[idx] = '\0';
50 return open(f_name, O_WRONLY | O_TRUNC);
51 }
52
53 int main(int argc, char* argv[]) {
54     int f1 = open_file();
55     if (f1 == -1) {
56         perror("File not found");
57         exit(1);
58     }
59     int f2 = open_file();
60     if (f2 == -1) {
61         perror("File not found");
62         exit(2);
63     }
64
65     int pipefd1[2];
66     if (pipe(pipefd1) == -1) {
67         perror("Cannot create pipe");
68         exit(3);
69     }
70
71     int child1 = fork();
72     if (child1 == -1) {
73         perror("Can not create process");
74         exit(4);
75     }
76     if (child1 == 0) {
77         close(pipefd1[1]);
78         child_work(pipefd1[0], f1);
79         return 0;
80     }
81     close(pipefd1[0]);
82
83     int pipefd2[2];
84
85     if (pipe(pipefd2) == -1) {
86         perror("Cannot create pipe");
87         exit(5);
88     }
89
90     int child2 = fork();
91     if (child2 == -1) {
92         perror("Can not create process");
93         exit(6);
94     }
95     if (child2 == 0) {

```

```
96     close(pipefd1[1]);
97     close(pipefd2[1]);
98     child_work(pipefd2[0], f2);
99     return 0;
100 }
101 close(pipefd2[0]);
102
103 parent_work(pipefd1[1], pipefd2[1]);
104
105 return 0;
106 }
```

5 Пример работы

Первый тест - проверка на обработку случая отсутствия первого файла

```
danila@LAPTOP-5N1LT0S0:/mnt/c/VUZ/OS/lab_2/src$ cat test1
f
danila@LAPTOP-5N1LT0S0:/mnt/c/VUZ/OS/lab_2/src$ ./a.out <test1
File not found
```

Второй тест - проверка на обработку случая отсутствия второго файла

```
danila@LAPTOP-5N1LT0S0:/mnt/c/VUZ/OS/lab_2/src$ cat test2
f1
f
danila@LAPTOP-5N1LT0S0:/mnt/c/VUZ/OS/lab_2/src$ ./a.out <test2
File not found
```

Третий и четвёртый тесты - проверка работоспособности программы на корректных данных

```
danila@LAPTOP-5N1LT0S0:/mnt/c/VUZ/OS/lab_2/src$ cat test3
f1
f2
a b c d e f g h i j k l
m n o p q r s t u v w x y z
danila@LAPTOP-5N1LT0S0:/mnt/c/VUZ/OS/lab_2/src$ ./a.out <test3
danila@LAPTOP-5N1LT0S0:/mnt/c/VUZ/OS/lab_2/src$ cat f1
b c d f g h j k l
danila@LAPTOP-5N1LT0S0:/mnt/c/VUZ/OS/lab_2/src$ cat f2
m n p q r s t v w x z
danila@LAPTOP-5N1LT0S0:/mnt/c/VUZ/OS/lab_2/src$ cat test4
f1
f2
m n o p q r s t u v w x y z
a b c d e f g h i j k l
m n o p q r s t u v w x y z
```



```

a b c d e f g h i j k l
m n o p q r s t u v w x y z
a b c d e f g h i j k l
m n o p q r s t u v w x y z
a b c d e f g h i j k l
danila@LAPTOP-5N1LT0S0:/mnt/c/VUZ/OS/lab_2/src$ ./a.out <test4
danila@LAPTOP-5N1LT0S0:/mnt/c/VUZ/OS/lab_2/src$ cat f1
m n p q r s t v w x z
m n p q r s t v w x z
m n p q r s t v w x z
m n p q r s t v w x z
danila@LAPTOP-5N1LT0S0:/mnt/c/VUZ/OS/lab_2/src$ cat f2
b c d f g h j k l
b c d f g h j k l
b c d f g h j k l
b c d f g h j k l

```

Пятый тест - проверка работоспособности программы на пустых данных

```

danila@LAPTOP-5N1LT0S0:/mnt/c/VUZ/OS/lab_2/src$ cat test5
f1
f2
danila@LAPTOP-5N1LT0S0:/mnt/c/VUZ/OS/lab_2/src$ ./a.out <test5
danila@LAPTOP-5N1LT0S0:/mnt/c/VUZ/OS/lab_2/src$ cat f1
danila@LAPTOP-5N1LT0S0:/mnt/c/VUZ/OS/lab_2/src$ cat f2

```

6 Вывод

Выполнив данную лабораторную работу 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 (pipe) как один из способов обмена данными между процессами также играют немаловажную роль.