**Московский авиационный институт**

**(национальный исследовательский университет)**

Институт №8 «Информационные технологии и прикладная математика» Кафедра 806 «Вычислительная математика и программирование» Дисциплина «Операционные системы»

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**Управление процессами в ОС**

Студент: Кондратьев Егор Алексеевич

Группа: М80-206Б-19

Преподаватель: Миронов Е.С.

Дата:

Оценка:

Москва, 2021

**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.

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

**Алгоритм решения задачи**

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.

**Листинг программы**

#include <fcntl.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

void child\_work(int from, int to)

{

char buf[1];

while (read(from, buf, 1) > 0)

{

char c = buf[0];

if (c != 'a' && c != 'e' && c != 'i' && c != 'o' && c != 'u' && c != 'y' &&

c != 'A' && c != 'E' && c != 'I' && c != 'O' && c != 'U' && c != 'Y')

{

write(to, buf, 1);

}

}

close(to);

close(from);

}

void parrent\_work(int child1, int child2)

{

char buf[1];

int is\_even = 0;

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

{

if (!is\_even)

{

write(child1, buf, 1);

}

else

{

write(child2, buf, 1);

}

if (buf[0] == '\n')

{

is\_even = !is\_even;

}

}

close(child1);

close(child2);

}

int open\_file()

{

const size\_t NAME\_SIZE = 64;

char f\_name[NAME\_SIZE];

char buf[1];

int idx = 0;

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

{

if (buf[0] == '\n')

{

break;

}

f\_name[idx++] = buf[0];

}

f\_name[idx] = '\0';

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

}

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

{

int f1 = open\_file();

if (f1 == -1)

{

perror("File not open");

exit(1);

}

int f2 = open\_file();

if (f2 == -1)

{

perror("File not open");

exit(2);

}

int pipefd1[2];

if (pipe(pipefd1) == -1)

{

perror("Cannot create pipe");

exit(3);

}

int child1 = fork();

if (child1 == -1)

{

perror("Can not create process");

exit(4);

}

if (child1 == 0)

{

close(pipefd1[1]);

child\_work(pipefd1[0], f1);

return 0;

}

close(pipefd1[0]);

int pipefd2[2];

if (pipe(pipefd2) == -1)

{

perror("Cannot create pipe");

exit(5);

}

int child2 = fork();

if (child2 == -1)

{

perror("Can not create process");

exit(6);

}

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;

}

**Тесты и протокол исполнения**

***Первый тест - проверка на обработку случая отсутствия первого файла***

root@Du:/mnt/c/Users/egork/Desktop/os/lab\_2/src$ cat test1

f

root@Du:/mnt/c/Users/egork/Desktop/os/lab\_2/src$ ./a.out <test1

File not found

***Второй тест - проверка на обработку случая отсутствия второго файла***

root@Du:/mnt/c/Users/egork/Desktop/os/lab\_2/src$ cat test2

f1

f

root@Du:/mnt/c/Users/egork/Desktop/os/lab\_2/src$ ./a.out <test2

File not found

***Третий и четвертый тесты - проверка работоспособности программы на корректных данных***

root@Du:/mnt/c/Users/egork/Desktop/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

root@Du:/mnt/c/Users/egork/Desktop/os/lab\_2/src$ ./a.out <test3

root@Du:/mnt/c/Users/egork/Desktop/os/lab\_2/src$ cat f1

b c d f g h j k l

root@Du:/mnt/c/Users/egork/Desktop/os/lab\_2/src$ cat f2

m n p q r s t v w x z

root@Du:/mnt/c/Users/egork/Desktop/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

root@Du:/mnt/c/Users/egork/Desktop/os/lab\_2/src$ ./a.out <test4

root@Du:/mnt/c/Users/egork/Desktop/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

root@Du:/mnt/c/Users/egork/Desktop/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***

root@Du:/mnt/c/Users/egork/Desktop/os/lab\_2/src$ cat test5

f1

f2

root@Du:/mnt/c/Users/egork/Desktop/os/lab\_2/src$ ./a.out <test5

root@Du:/mnt/c/Users/egork/Desktop/os/lab\_2/src$ cat f1

root@Du:/mnt/c/Users/egork/Desktop/os/lab\_2/src$ cat f2

**Вывод**

Работая над данной лабораторной работой, я узнал о том, как работают процессы в операционной системе Linux. Среди прочего я узнал о командах: pipe() - создающей канал, fork() - создающей процесс, execl() - заменяющей один образ процесса другим и dup2() - создающий дубликат файлового дескриптора.

Я на конкретном примере 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 тоже играют немаловажную роль.

**Список литературы**

1. [Изучаем процессы в Linux / Хабр — Habr](https://habr.com/ru/post/423049/)
2. Таненбаум Э., Бос Х. *Современные операционные системы. — 4-е изд.*