Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Лабораторная работа №1 по курсу**

**«Операционные системы»**

Студент: Носов А.К.

Группа: М8О-206Б-22

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Постановка задачи**

**Группа вариантов 5.**

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 20.**

Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы инвертируют строки.

**Метод решения**

Использованные системные вызовы:

* pid\_t fork(void); – создает дочерний процесс.
* int pipe(int \*fd); – Функция pipe создает канал (неименованный) и помещает дескрипторы файла для чтения и записи (соответственно) в fd[0] и fd[1].
* int execl(char \*name, char \*arg0, ... /\*NULL\*/) – загружает и запускает указанную программу. Таким образом, новая программа полностью замещает текущий процесс.
* int dup2(int oldfd, int newfd) – делает newfd копией oldfd, закрывая newfd, если требуется.
* int close(int fd) - закрывает файловый дескриптор.
* ssize\_t read (int fd, void \* buffer, size\_t count) – считывает count байт из файлового дескриптора fd в buffer. Возвращает количество прочитанных байт или -1, если ошибка.
* ssize\_t write (int fd, const void \* buffer, size\_t count) – записывает count байтов из буфера buffer в файл с дескриптором fd, возвращая количество записанных байтов или -1 в случае ошибки.

После запуска программы пользователю нужно ввести в командную строку имя первого файла, затем на следующей строке имя второго файла. После этого функция open открывает файл с данным названием и очищает его. Если данного файла не было, то создаст его. Если все было введено корректно и два файла доступны для работы, то создаются два безымянных канала pipe. Далее будет создан первый дочерний процесс. В нем мы заменяем стандартный поток ввода на “чтение” из pipe (fd[READ]), а стандартный поток вывода на запись в файл с помощью функции dup2. Затем первый дочерний процесс запускает программу child.c, и программа main для этого процесса завершается. Аналогичные действия мы проделываем с вторым дочерним процессом. Далее в родительском процессе мы считываем все символы, которые вводит пользователь. Сначала мы их направляем в дескриптор первого pipe, откуда первый дочерний процесс считывает введенные данные, обрабатывает их и записывает в стандартный вывод, который подменен на вывод в файл в данном процессе. Когда пользователь введет \n, мы начинаем перенаправлять все символы в дескриптор второго pipe, откуда второй дочерний процесс так же читает данные, а потом выводит во второй файл. Программа завершает работу, когда встретит символ EOF.

**Код программы**

**lab1.c**

#include "create\_processe.h"

int main() {

const int max\_buffer\_size = 50;

printf("Введите названия файлов.\n");

char file\_name\_first[max\_buffer\_size];

scanf("%s", file\_name\_first);

int first\_file\_descriptior = open(file\_name\_first, O\_CREAT | O\_WRONLY, S\_IRWXU);

if(first\_file\_descriptior == -1){

perror("Opening file number 1");

return -1;

}

int pipe\_fisrt[2];

int first\_error = pipe(pipe\_fisrt);

if(first\_error == -1){

perror("Creature a pipe 1");

return -1;

}

pid\_t proccess\_id = create\_processe();

if(proccess\_id == 0){

close(pipe\_fisrt[1]);

dup2(pipe\_fisrt[0], STDIN\_FILENO);

dup2(first\_file\_descriptior, STDOUT\_FILENO);

execl("child", " ", NULL);

perror("Execl in child 1");

return -1;

}

char file\_name2[max\_buffer\_size];

scanf("%s", file\_name2);

int second\_file\_descriptior = open(file\_name2, O\_CREAT | O\_WRONLY, S\_IRWXU);

if(second\_file\_descriptior == -1){

perror("Opening file number 2");

return -1;

}

close(pipe\_fisrt[0]);

int pipe\_second[2];

int second\_error = pipe(pipe\_second);

if(second\_error == -1){

perror("Creature a pipe 2");

return -1;

}

proccess\_id = create\_processe();

if(proccess\_id == 0){

close(pipe\_second[1]);

dup2(pipe\_second[0], STDIN\_FILENO);

dup2(second\_file\_descriptior, STDOUT\_FILENO);

execl("child", " ", NULL);

perror("Execl in child2");

return -1;

}

close(pipe\_second[0]);

char string [max\_buffer\_size];

int count;

while((count = read(0, string, max\_buffer\_size))>0){

if(count > 11){

write(pipe\_second[1], string, count);

}

else{

write(pipe\_fisrt[1], string, count);

}

}

close(first\_file\_descriptior);

close(second\_file\_descriptior);

return 0;

}

**child.c**

#include "create\_processe.h"

void reverse\_string(char\* string, int size\_string){

for(int i = 0; i < (size\_string/2); ++i){

char symbol\_to\_replace = string[i];

string[i] = string[size\_string-1 - i];

string[size\_string - 1 - i] = symbol\_to\_replace;

}

}

int main(){

const int max\_buffer\_size = 50;

char string[50];

int count;

while((count = read(STDIN\_FILENO, string, 50))>0){

reverse\_string(string, count-1);

write(STDOUT\_FILENO, string, count);

}

close(STDOUT\_FILENO);

close(STDIN\_FILENO);

return 0;

}

**create\_process.h**

#include "create\_processe.h"

void reverse\_string(char\* string, int size\_string){

for(int i = 0; i < (size\_string/2); ++i){

char symbol\_to\_replace = string[i];

string[i] = string[size\_string-1 - i];

string[size\_string - 1 - i] = symbol\_to\_replace;

}

}

int main(){

const int max\_buffer\_size = 50;

char string[50];

int count;

while((count = read(STDIN\_FILENO, string, 50))>0){

reverse\_string(string, count-1);

write(STDOUT\_FILENO, string, count);

}

close(STDOUT\_FILENO);

close(STDIN\_FILENO);

return 0;

}

**Протокол работы программы**

**Тестирование:**

./parent

aaaa

bbbb

aaaaab

bbbbba

qwerty

qwertyu

ry

qwasdf

$ cat < aaaa

b

qwrt

r

qwsdf

$ cat < bbbb

bbbbb

qwrt

**Strace:**

execve("./lab1", ["./lab1"], 0x7ffe6c0a7de0 /\* 87 vars \*/) = 0

brk(NULL) = 0x55707f1be000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffe19005760) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f146b7d6000

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=70887, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 70887, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f146b7c4000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\302\211\332Pq\2439\235\350\223\322\257\201\326\243\f"..., 68, 896) = 68

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2220400, ...}, AT\_EMPTY\_PATH) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

mmap(NULL, 2264656, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f146b400000

mprotect(0x7f146b428000, 2023424, PROT\_NONE) = 0

mmap(0x7f146b428000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f146b428000

mmap(0x7f146b5bd000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7f146b5bd000

mmap(0x7f146b616000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x7f146b616000

mmap(0x7f146b61c000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f146b61c000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f146b7c1000

arch\_prctl(ARCH\_SET\_FS, 0x7f146b7c1740) = 0

set\_tid\_address(0x7f146b7c1a10) = 31857

set\_robust\_list(0x7f146b7c1a20, 24) = 0

rseq(0x7f146b7c20e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f146b616000, 16384, PROT\_READ) = 0

mprotect(0x55707efbb000, 4096, PROT\_READ) = 0

mprotect(0x7f146b810000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x7f146b7c4000, 70887) = 0

newfstatat(1, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}, AT\_EMPTY\_PATH) = 0

getrandom("\x87\x78\xdb\xf9\x95\x45\xc5\x14", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x55707f1be000

brk(0x55707f1df000) = 0x55707f1df000

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\275\320\260\320\267\320\262\320\260\320\275\320\270\321\217 "..., 46Введите названия файлов.

) = 46

newfstatat(0, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}, AT\_EMPTY\_PATH) = 0

read(0, a

"a\n", 1024) = 2

openat(AT\_FDCWD, "a", O\_WRONLY|O\_CREAT, 0700) = 3

pipe2([4, 5], 0) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f146b7c1a10) = 31880

read(0, b

"b\n", 1024) = 2

openat(AT\_FDCWD, "b", O\_WRONLY|O\_CREAT, 0700) = 6

close(4) = 0

pipe2([4, 7], 0) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f146b7c1a10) = 31881

close(4) = 0

read(0, aeferg

"aeferg\n", 50) = 7

write(5, "aeferg\n", 7) = 7

read(0, sergbvserbtkrnb sntrbs

"sergbvserbtkrnb sntrbs\n", 50) = 23

write(7, "sergbvserbtkrnb sntrbs\n", 23) = 23

read(0, rtbstrbtrb

"rtbstrbtrb\n", 50) = 11

write(5, "rtbstrbtrb\n", 11) = 11

read(0, strxbstrdbtrdbrstb

"strxbstrdbtrdbrstb\n", 50) = 19

write(7, "strxbstrdbtrdbrstb\n", 19) = 19

read(0, srtbsrt

"srtbsrt\n", 50) = 8

write(5, "srtbsrt\n", 8) = 8

read(0, bsdbtb

"bsdbtb\n", 50) = 7

write(5, "bsdbtb\n", 7) = 7

read(0, "", 50) = 0

close(3) = 0

close(6) = 0

lseek(0, -1, SEEK\_CUR) = -1 ESPIPE (Illegal seek)

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В ходе выполнения данной лабораторной работы я столкнулся с перенаправлением ввода и вывода, а также узнал о том, что одну программу можно “параллелить” на две и больше почти идентичные программы. Изначально были сложности с тем, чтобы держать все дочерние и родительский процессы в голове, и понимать, куда нужно перенаправлять их вводы и выводы. В итоге, я неплохо разобрался и осознал данную тему, что поможет мне в будущем справляться с более сложными задачами.