Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Группа: М80-206Б-20

Студент: Носов А.К.

Преподаватель: Миронов Е.С.

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 первый отображаемый файл (memory-mapped file) или во второй отображаемый файл 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.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* pid\_t fork(void); – создает дочерний процесс.
* int shm\_open(const char \*name, int oflag, mode\_t mode) - создает и открывает новый (или открывает уже существующий) объект разделяемой памяти POSIX.
* void \* mmap(void \*start, size\_t length, int prot , int flags, int fd, off\_t offset) - отражает файлы или устройства в памяти или снимает их отражение. При удачном выполнении mmap возвращает указатель на область с отраженными данными.
* int ftruncate(int fd, off\_t length) - приводит файл к заданному размеру.
* 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.
* void (\*signal (int signal, void (\*sigfunc) (int func)))(int) - дает указание выполнить функцию, на которую указывает sigfunc, в случае получения сигнала signal.
* int kill(pid\_t pid, int sig) - посылает сигнал процессу или выводит список допустимых сигналов.
* int unlink(const char \*pathname) - unlink удаляет имя из файловой системы
* void exit(int status) - приводит к обычному завершению программы

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 новых файла в разделяемой памяти (далее mmf файл), получим указатели на них. Далее будет создан первый дочерний процесс. В нем мы заменяем стандартный поток вывода на запись в файл с помощью функции dup2. Затем первый дочерний процесс запускает программу child.c, в которую передаем число, отвечающее за имя файла, который находится в разделяемой памяти, 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, записываем их в mmf файл. После символа ‘\n’ родительский процесс отправляет сигнал дочернему. Приняв сигнал, первый дочерний процесс запускает функцию, в которой считывает из mmf файла введенные данные, обрабатывает их и записывает в стандартный вывод, который подменен на вывод в файл в данном процессе. Далее все описанные действия выполняться для второго родительского процесса. Программа завершает работу, когда встретит символ EOF.

**Код программы**

**lab3.c**

#include "create\_processe.h"

int main() {

printf("Введите названия файлов.\n");

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

char\* list\_semafore\_name[] = {"/semafor1", "/semafor2", "/semafor3"};

sem\_t\* list\_semafore[count\_semafore];

for(int i = 0; i < count\_semafore; ++i){

list\_semafore[i] = sem\_open(list\_semafore\_name[i], O\_CREAT, S\_IRWXU, 0);

}

int list\_file\_description[count\_processes];

for(int i = 0; i < count\_processes; ++i){

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

if((list\_file\_description[i] = open(file\_name, O\_CREAT | O\_RDWR, S\_IRWXU)) == -1){

perror("Opening file");

return -1;

}

}

pid\_t list\_pid[count\_processes];

char mmap\_file\_name[] = "/mmap\_file";

int mmap\_file\_descriptior;

if ((mmap\_file\_descriptior = shm\_open(mmap\_file\_name, O\_CREAT | O\_RDWR, S\_IRWXU)) == -1) {

perror("open mmap\_file\_descriptior");

return -1;

}

for(int i = 0; i < count\_processes; ++i){

list\_pid[i] = create\_processe();

if(list\_pid[i] == 0){

dup2(mmap\_file\_descriptior, STDIN\_FILENO);

dup2(list\_file\_description[i], STDOUT\_FILENO);

execl("child2", " ", list\_semafore\_name[i], list\_semafore\_name[2], NULL);

perror("Execl in child");

return -1;

}

}

char string [max\_buffer\_size];

int count;

char\* mp;

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

if (ftruncate(mmap\_file\_descriptior, count) == -1) {

perror("Could not set size of mfile.txt");

return 1;

}

if ((mp = mmap(NULL, count, PROT\_WRITE, MAP\_SHARED, mmap\_file\_descriptior, 0)) == MAP\_FAILED) {

perror("mmap");

return -1;

}

for(int i = 0; i < count; ++i){

mp[i] = string[i];

}

if(count > 11){

sem\_post(list\_semafore[1]);

}

else{

sem\_post(list\_semafore[0]);

}

sem\_wait(list\_semafore[2]);

}

unlink(mmap\_file\_name);

for(int i = 0; i < count\_processes; ++i){

if (kill(list\_pid[i], SIGTERM) == -1) {

perror("Ошибка при отправке сигнала");

}

close(list\_file\_description[0]);

close(list\_file\_description[1]);

}

for(int i = 0; i < count\_semafore; ++i){

sem\_close(list\_semafore[i]);

sem\_unlink(list\_semafore\_name[i]);

}

return 0;

}

**child2.c**

#include "create\_processe.h"

sem\_t\* child\_list\_semafor[2];

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;

}

}

void handle\_sigterm()

{

for(int i = 0; i < count\_processes; ++i){

close(i);

sem\_close(child\_list\_semafor[i]);

}

exit(0);

}

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

for(int i = 0; i < count\_processes; ++i)

child\_list\_semafor[i] = sem\_open(argv[i+1], 0);

char string[max\_buffer\_size];

struct stat sd;

while(1){

if (signal(SIGTERM, handle\_sigterm) == SIG\_ERR) {

perror("Error while setting a signal handler");

return EXIT\_FAILURE;

}

sem\_wait(child\_list\_semafor[0]);

if(fstat(STDIN\_FILENO, &sd) == -1){

perror("could not get file size. \n");

}

char\* mp = mmap(NULL, sd.st\_size, PROT\_READ, MAP\_SHARED, STDIN\_FILENO, 0);

for(int i = 0; i < sd.st\_size; ++i){

string[i] = mp[i];

}

reverse\_string(string, sd.st\_size - 1);

if(write(STDOUT\_FILENO, string, sd.st\_size) == -1){

perror("write");

}

sem\_post(child\_list\_semafor[1]);

}

}

**create\_process.h**

#ifndef SUM\_H

#define SUM\_H

const int max\_buffer\_size = 128;

const int count\_semafore = 3;

const int count\_processes = 2;

#include <sys/stat.h>

#include "stdio.h"

#include "stdlib.h"

#include <fcntl.h>

#include "unistd.h"

#include "sys/wait.h"

#include <semaphore.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <signal.h>

int create\_processe(){

pid\_t pid = fork();

if(pid == -1){

perror("Сreating a process");

exit(-1);

}

return pid;

};

#endif

**Протокол работы программы**

**Тестирование:**

$ ./parent

aaaa

bbbb

qwertyyyy

dddderayt

adassssddeeee

uuuyturrrtt

iooottttt

$ cat < aaaa

qwrt

dssssdd

ttttt

$ cat < bbbb

ddddrt

trrrtt

**Strace:**

execve("./lab3", ["./lab3"], 0x7fff4b3a8c30 /\* 87 vars \*/) = 0

brk(NULL) = 0x55b6735d7000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffdab20ca00) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f063c296000

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) = 0x7f063c284000

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) = 0x7f063c000000

mprotect(0x7f063c028000, 2023424, PROT\_NONE) = 0

mmap(0x7f063c028000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f063c028000

mmap(0x7f063c1bd000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7f063c1bd000

mmap(0x7f063c216000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x7f063c216000

mmap(0x7f063c21c000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f063c21c000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f063c281000

arch\_prctl(ARCH\_SET\_FS, 0x7f063c281740) = 0

set\_tid\_address(0x7f063c281a10) = 22935

set\_robust\_list(0x7f063c281a20, 24) = 0

rseq(0x7f063c2820e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f063c216000, 16384, PROT\_READ) = 0

mprotect(0x55b672912000, 4096, PROT\_READ) = 0

mprotect(0x7f063c2d0000, 8192, PROT\_READ) = 0

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

munmap(0x7f063c284000, 70887) = 0

newfstatat(1, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0x2), ...}, AT\_EMPTY\_PATH) = 0

getrandom("\x51\x0c\x4a\xfc\x91\x1a\x1a\x9f", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x55b6735d7000

brk(0x55b6735f8000) = 0x55b6735f8000

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

openat(AT\_FDCWD, "/dev/shm/sem.semafor1", O\_RDWR|O\_NOFOLLOW) = 3

newfstatat(3, "", {st\_mode=S\_IFREG|0700, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7f063c2cf000

close(3) = 0

openat(AT\_FDCWD, "/dev/shm/sem.semafor2", O\_RDWR|O\_NOFOLLOW) = 3

newfstatat(3, "", {st\_mode=S\_IFREG|0700, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7f063c295000

close(3) = 0

openat(AT\_FDCWD, "/dev/shm/sem.semafor3", O\_RDWR|O\_NOFOLLOW) = 3

newfstatat(3, "", {st\_mode=S\_IFREG|0700, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7f063c294000

close(3) = 0

newfstatat(0, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0x2), ...}, AT\_EMPTY\_PATH) = 0

read(0, a

"a\n", 1024) = 2

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

read(0, b

"b\n", 1024) = 2

openat(AT\_FDCWD, "b", O\_RDWR|O\_CREAT, 0700) = 4

openat(AT\_FDCWD, "/dev/shm/mmap\_file", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0700) = 5

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f063c281a10) = 22954

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f063c281a10) = 22955

read(0, shghdzhgiuherdtgfz

"shghdzhgiuherdtgfz\n", 128) = 19

ftruncate(5, 19) = 0

mmap(NULL, 19, PROT\_WRITE, MAP\_SHARED, 5, 0) = 0x7f063c293000

futex(0x7f063c295000, FUTEX\_WAKE, 1) = 1

read(0, ebsuhtrbiuhsuerhdztbe

"ebsuhtrbiuhsuerhdztbe\n", 128) = 22

ftruncate(5, 22) = 0

mmap(NULL, 22, PROT\_WRITE, MAP\_SHARED, 5, 0) = 0x7f063c292000

futex(0x7f063c295000, FUTEX\_WAKE, 1) = 1

read(0, srtbrstbtrstrrstbrst

"srtbrstbtrstrrstbrst\n", 128) = 21

ftruncate(5, 21) = 0

mmap(NULL, 21, PROT\_WRITE, MAP\_SHARED, 5, 0) = 0x7f063c291000

futex(0x7f063c295000, FUTEX\_WAKE, 1) = 1

read(0, rbtstrbrstbrts

"rbtstrbrstbrts\n", 128) = 15

ftruncate(5, 15) = 0

mmap(NULL, 15, PROT\_WRITE, MAP\_SHARED, 5, 0) = 0x7f063c290000

futex(0x7f063c295000, FUTEX\_WAKE, 1) = 1

read(0, btrstbrbrsttrbsbrtnbdfhnhdgxfn

"btrstbrbrsttrbsbrtnbdfhnhdgxfn\n", 128) = 31

ftruncate(5, 31) = 0

mmap(NULL, 31, PROT\_WRITE, MAP\_SHARED, 5, 0) = 0x7f063c28f000

futex(0x7f063c295000, FUTEX\_WAKE, 1) = 1

read(0, dfxhnfg

"dfxhnfg\n", 128) = 8

ftruncate(5, 8) = 0

mmap(NULL, 8, PROT\_WRITE, MAP\_SHARED, 5, 0) = 0x7f063c28e000

futex(0x7f063c2cf000, FUTEX\_WAKE, 1) = 1

read(0, "", 128) = 0

unlink("/mmap\_file") = -1 ENOENT (No such file or directory)

kill(22954, SIGTERM) = 0

close(3) = 0

close(4) = 0

kill(22955, SIGTERM) = 0

close(3) = -1 EBADF (Bad file descriptor)

close(4) = -1 EBADF (Bad file descriptor)

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=22954, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

munmap(0x7f063c2cf000, 32) = 0

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=22955, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

unlink("/dev/shm/sem.semafor1") = 0

munmap(0x7f063c295000, 32) = 0

unlink("/dev/shm/sem.semafor2") = 0

munmap(0x7f063c294000, 32) = 0

unlink("/dev/shm/sem.semafor3") = 0

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

exit\_group(0) = ?

+++ exited with 0 +++

1

**Вывод**

В ходе выполнения данной лабораторной работы я узнал, что существуют memory-mapped файлы, которые являются отличной альтернативой pipe. Было очень интересно разбираться в том, как именно работают все системные вызовы и где хранятся созданные memory-mapped файлы. Уверен, эти знания помогут мне в понимании внутренних процессов компьютера во время выполнения более сложных задач.