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

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

**Лабораторная работа № 4 по курсу «Операционные системы»**

Студент: Абросимов Алексей Дмитриевич

Группа: М8О-207Б-20

Преподаватель: Е. С. Миронов

Вариант:

Дата:

Оценка:

Москва, 2021

## **1. Описание**

Данная лабораторная работа будет выполняться в ОС Unix.

**Задание:**  Имеется три процесса: родительский и два дочерних. Взаимодействие между процессами будет осуществляться с помощью двух shared memory. Процесс-родитель осуществляет чтение названий двух файлов из стандартного ввода, в который процесс-child будет осуществлять вывод инвертированных строк, пришедших с shared memory. Процесс-ребенок перенаправляет поток ввода на один из shared memory и поток вывода на файл и будет инвертировать строки,находящиеся в неименованном канале, и выводить их в нужный файл.

Для работы программы нужно научиться создавать shared memory и передавать информацию между процессами.

## **2. Исходный код**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/mman.h>

#include <semaphore.h>

#include <string.h>

#include "shmem.h"

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

{

char c = 'c';

int i = 0;

int ni = 0;

int val;

char \*string = (char \*)malloc(sizeof(char));

char \*nstring = (char \*)malloc(sizeof(char));

int a = atoi(argv[2]);

char \*empty = (char \*)malloc(map\_size \* sizeof(char));

const char \*filename = argv[1];

FILE \*file = fopen(filename, "w");

int map\_fd;

sem\_t \*semptr;

if (a == 1)

{

map\_fd = shm\_open(BackingFile1, O\_RDWR, AccessPerms);

semptr = sem\_open(SemaphoreName1, O\_CREAT, AccessPerms, 2);

}

else

{

map\_fd = shm\_open(BackingFile2, O\_RDWR, AccessPerms);

semptr = sem\_open(SemaphoreName2, O\_CREAT, AccessPerms, 2);

}

if (map\_fd < 0)

{

perror("shm\_open");

exit(EXIT\_FAILURE);

}

caddr\_t memptr = mmap(

NULL,

map\_size,

PROT\_READ | PROT\_WRITE,

MAP\_SHARED,

map\_fd,

0);

if (memptr == MAP\_FAILED)

{

perror("mmap");

exit(EXIT\_FAILURE);

}

if (semptr == SEM\_FAILED)

{

perror("semptr");

exit(EXIT\_FAILURE);

}

if (sem\_wait(semptr) != 0)

{

perror("sem\_wait");

exit(EXIT\_FAILURE);

}

while (memptr[0] != EOF)

{

if (sem\_getvalue(semptr, &val) != 0)

{

perror("sem\_getvalue");

exit(EXIT\_FAILURE);

}

if (val == 2)

{

continue;

}

if (sem\_wait(semptr) != 0)

{

perror("sem\_wait");

exit(EXIT\_FAILURE);

}

if (memptr[0] == EOF)

{

break;

}

if (memptr[0] == '\0')

{

if (sem\_post(semptr) != 0)

{

perror("sem\_post");

exit(EXIT\_FAILURE);

}

continue;

}

nstring = (char \*)realloc(nstring, strlen(memptr) \* sizeof(char));

string = (char \*)realloc(string, strlen(memptr) \* sizeof(char));

strcpy(string, memptr);

ni = 0;

i = strlen(string);

while (ni < i)

{

nstring[i - 1 - ni] = string[ni];

ni += 1;

}

i = 0;

ni = 0;

if (sem\_post(semptr) != 0)

{

perror("sem\_post");

exit(EXIT\_FAILURE);

}

if (strcmp(nstring, empty) != 0)

{

fprintf(file, "%s\n", nstring);

}

fflush(file);

memset(memptr, '\0', map\_size);

free(string);

free(nstring);

string = (char \*)calloc(1, sizeof(char));

nstring = (char \*)calloc(1, sizeof(char));

}

return 0;

}

————————————————————————-

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <sys/wait.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <semaphore.h>

#include <stdbool.h>

#include "shmem.h"

int main()

{

char \*empty\_string = malloc(sizeof(char) \* map\_size);

int fd1, fd2;

char temp = 'c';

char \*filename;

char \*filename2;

int i = 0;

char \*num = (char \*)malloc(sizeof(char));

fd1 = shm\_open(BackingFile1, O\_RDWR | O\_CREAT, AccessPerms);

fd2 = shm\_open(BackingFile2, O\_RDWR | O\_CREAT, AccessPerms);

filename = (char \*)malloc(sizeof(char));

printf("Enter file1 name: ");

while (temp != '\n')

{

scanf("%c", &temp);

i++;

filename = (char \*)realloc(filename, i \* sizeof(char));

filename[i - 1] = temp;

}

filename[i - 1] = '\0';

int file = open(filename, O\_WRONLY);

if (file == -1)

{

perror("open");

exit(EXIT\_FAILURE);

}

temp = 'c';

filename2 = (char \*)malloc(sizeof(char));

i = 0;

printf("Enter file2 name: ");

while (temp != '\n')

{

scanf("%c", &temp);

i++;

filename2 = (char \*)realloc(filename2, i \* sizeof(char));

filename2[i - 1] = temp;

}

filename2[i - 1] = '\0';

int file2 = open(filename2, O\_WRONLY);

if (file2 == -1)

{

perror("open");

exit(EXIT\_FAILURE);

}

sem\_t \*semptr1 = sem\_open(SemaphoreName1, O\_CREAT, AccessPerms, 2);

sem\_t \*semptr2 = sem\_open(SemaphoreName2, O\_CREAT, AccessPerms, 2);

if (semptr1 == SEM\_FAILED)

{

perror("sem1\_open");

exit(EXIT\_FAILURE);

}

if (semptr2 == SEM\_FAILED)

{

perror("sem2\_open");

exit(EXIT\_FAILURE);

}

int val1;

int val2;

ftruncate(fd1, map\_size);

ftruncate(fd2, map\_size);

caddr\_t memptr1 = mmap(

NULL,

map\_size,

PROT\_READ | PROT\_WRITE,

MAP\_SHARED,

fd1,

0);

caddr\_t memptr2 = mmap(

NULL,

map\_size,

PROT\_READ | PROT\_WRITE,

MAP\_SHARED,

fd2,

0);

if (memptr1 == MAP\_FAILED)

{

perror("mmap1");

exit(EXIT\_FAILURE);

}

if (sem\_getvalue(semptr1, &val1) != 0)

{

perror("sem1\_getvalue");

exit(EXIT\_FAILURE);

}

memset(memptr1, '\0', map\_size);

if (memptr2 == MAP\_FAILED)

{

perror("mmap2");

exit(EXIT\_FAILURE);

}

if (sem\_getvalue(semptr2, &val2) != 0)

{

perror("sem2\_getvalue");

exit(EXIT\_FAILURE);

}

memset(memptr2, '\0', map\_size);

while (val1++ < 2)

{

sem\_post(semptr1);

}

while (val2++ < 2)

{

sem\_post(semptr2);

}

pid\_t pid2;

pid\_t pid = fork();

if (pid == 0)

{

munmap(memptr1, map\_size);

close(fd1);

sem\_close(semptr1);

num[0] = '1';

execl("child.out", "child.out", filename, num, NULL);

perror("execl");

exit(EXIT\_FAILURE);

}

if (pid > 0)

{

char \*str = (char \*)malloc(sizeof(char));

int counter = 0;

i = 0;

pid2 = fork();

if (pid2 == 0)

{

munmap(memptr2, map\_size);

close(fd2);

sem\_close(semptr2);

num[0] = '2';

execl("child.out", "child.out", filename2, num, NULL);

perror("execl");

exit(EXIT\_FAILURE);

}

if (pid2 > 0)

{

while ((temp = getchar()) != EOF)

{

if (counter)

{

if (temp != '\n')

{

++i;

str = (char \*)realloc(str, i \* sizeof(char));

str[i - 1] = temp;

}

else

{

i=0;

while (true)

{

if (sem\_wait(semptr2) == 0)

{

if (strcmp(memptr2, empty\_string) != 0)

{

if (sem\_post(semptr2) != 0)

{

perror("sem\_post");

exit(EXIT\_FAILURE);

}

continue;

}

i = 0;

sprintf(memptr2, "%s", str);

free(str);

str=(char\*)calloc(1,sizeof(char));

if (sem\_post(semptr2) != 0)

{

perror("sem\_post");

exit(EXIT\_FAILURE);

}

break;

}

else

{

perror("sem\_wait");

exit(EXIT\_FAILURE);

}

}

counter = 0;

i = 0;

}

}

else

{

if (temp != '\n')

{

++i;

str = (char \*)realloc(str, i \* sizeof(char));

str[i - 1] = temp;

}

else

{

while (true)

{

if (sem\_wait(semptr1) == 0)

{

if (strcmp(memptr1, empty\_string) != 0)

{

if (sem\_post(semptr1) != 0)

{

perror("sem\_post");

exit(EXIT\_FAILURE);

}

continue;

}

sprintf(memptr1, "%s", str);

free(str);

str=(char\*)calloc(1,sizeof(char));

i = 0;

if (sem\_post(semptr1) != 0)

{

perror("sem\_post");

exit(EXIT\_FAILURE);

}

break;

}

else

{

perror("sem\_wait");

exit(EXIT\_FAILURE);

}

}

counter = 1;

i = 0;

}

}

}

}

}

memptr1[0]=EOF;

memptr2[0]=EOF;

return 0;

}

———————————————————————————-

#ifndef INC\_4\_LAB\_SHRMEM\_H

#define INC\_4\_LAB\_SHRMEM\_H

#include <fcntl.h>

const size\_t map\_size = 4096;

const char \* BackingFile1 = "os\_lab4.1.back";

const char \* SemaphoreName1 = "os\_lab4.1.semaphore";

const char \* BackingFile2 = "os\_lab4.2.back";

const char \* SemaphoreName2 = "os\_lab4.2.semaphore";

unsigned AccessPerms = S\_IWUSR | S\_IRUSR | S\_IRGRP | S\_IROTH;

#endif //INC\_4\_LAB\_SHRMEM\_H

## **3. Консоль**

**$ strace ./prog.out**

execve("./prog.out", ["./prog.out"], 0x7ffd035d9bf0 /\* 58 vars \*/) = 0

brk(NULL) = 0x55c0fa2e0000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fff5fbc0520) = -1 EINVAL (Invalid argument)

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=154731, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 154731, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f2b0cac2000

close(3) = 0

openat(AT\_FDCWD, "/usr/lib/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\0\320\324\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"..., 80, 848) = 80

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\205vn\235\204X\261n\234|\346\340|q,\2"..., 68, 928) = 68

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2463384, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f2b0cac0000

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, 2136752, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f2b0c8b6000

mprotect(0x7f2b0c8e2000, 1880064, PROT\_NONE) = 0

mmap(0x7f2b0c8e2000, 1531904, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2c000) = 0x7f2b0c8e2000

mmap(0x7f2b0ca58000, 344064, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1a2000) = 0x7f2b0ca58000

mmap(0x7f2b0caad000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1f6000) = 0x7f2b0caad000

mmap(0x7f2b0cab3000, 51888, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f2b0cab3000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f2b0c8b3000

arch\_prctl(ARCH\_SET\_FS, 0x7f2b0c8b3740) = 0

set\_tid\_address(0x7f2b0c8b3a10) = 3553

set\_robust\_list(0x7f2b0c8b3a20, 24) = 0

rseq(0x7f2b0c8b40e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f2b0caad000, 12288, PROT\_READ) = 0

mprotect(0x55c0f8b85000, 4096, PROT\_READ) = 0

mprotect(0x7f2b0cb1d000, 8192, PROT\_READ) = 0

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

munmap(0x7f2b0cac2000, 154731) = 0

getrandom("\x84\x1f\xa0\xe4\x94\x70\xd0\x7c", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x55c0fa2e0000

brk(0x55c0fa301000) = 0x55c0fa301000

openat(AT\_FDCWD, "/dev/shm/os\_lab4.1.back", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0644) = 3

openat(AT\_FDCWD, "/dev/shm/os\_lab4.2.back", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0644) = 4

newfstatat(1, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0x1), ...}, AT\_EMPTY\_PATH) = 0

newfstatat(0, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0x1), ...}, AT\_EMPTY\_PATH) = 0

write(1, "Enter file1 name: ", 18Enter file1 name: ) = 18

read(0, res

"res\n", 1024) = 4

openat(AT\_FDCWD, "res", O\_WRONLY) = 5

write(1, "Enter file2 name: ", 18Enter file2 name: ) = 18

read(0, res2

"res2\n", 1024) = 5

openat(AT\_FDCWD, "res2", O\_WRONLY) = 6

openat(AT\_FDCWD, "/dev/shm/sem.os\_lab4.1.semaphore", O\_RDWR|O\_NOFOLLOW) = -1 ENOENT (No such file or directory)

getrandom("\xf9\x97\x49\x3b\x12\x67\xd2\x49", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.1kW9gf", 0x7fff5fbc01e0, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/dev/shm/sem.1kW9gf", O\_RDWR|O\_CREAT|O\_EXCL, 0644) = 7

write(7, "\2\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32

mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 7, 0) = 0x7f2b0cb1c000

link("/dev/shm/sem.1kW9gf", "/dev/shm/sem.os\_lab4.1.semaphore") = 0

newfstatat(7, "", {st\_mode=S\_IFREG|0644, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

unlink("/dev/shm/sem.1kW9gf") = 0

close(7) = 0

openat(AT\_FDCWD, "/dev/shm/sem.os\_lab4.2.semaphore", O\_RDWR|O\_NOFOLLOW) = -1 ENOENT (No such file or directory)

getrandom("\x76\xa9\x57\x87\xef\xb7\xc4\x17", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.qRVQrE", 0x7fff5fbc01e0, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/dev/shm/sem.qRVQrE", O\_RDWR|O\_CREAT|O\_EXCL, 0644) = 7

write(7, "\2\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32

mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 7, 0) = 0x7f2b0cae7000

link("/dev/shm/sem.qRVQrE", "/dev/shm/sem.os\_lab4.2.semaphore") = 0

newfstatat(7, "", {st\_mode=S\_IFREG|0644, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

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

close(7) = 0

ftruncate(3, 4096) = 0

ftruncate(4, 4096) = 0

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7f2b0cae6000

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 4, 0) = 0x7f2b0cae5000

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f2b0c8b3a10) = 3556

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f2b0c8b3a10) = 3557

read(0, stringone

"stringone\n", 1024) = 10

futex(0x7f2b0cb1c000, FUTEX\_WAKE, 1) = 1

read(0, stringtwo

"stringtwo\n", 1024) = 10

futex(0x7f2b0cae7000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL, FUTEX\_BITSET\_MATCH\_ANY) = -1 EAGAIN (Resource temporarily unavailable)

futex(0x7f2b0cae7000, FUTEX\_WAKE, 1) = 1

read(0, "", 1024) = 0

exit\_group(0) = ?

+++ exited with 0 +++

**4. Вывод**

Данная лабораторная работа позволила мне ознакомиться с методами работы с общей памятью, что невероятно полезно при написании современных программ, тк позволяет крайне эффективно связать несколько процессов. Мне это показалось невероятно интересным и полезным