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: Чекменев В.А.

Группа: М80-207Б-20

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата:

Содержание

1. Постановка задачи.
2. Общие сведения о программе.
3. Общий метод и алгоритм решения.
4. Код программы.
5. Демонстрация работы программы.
6. Вывод.

## **Постановка задачи**

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 через отображаемые файлы (memory-mapped files).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

**Группа вариантов № 1:**

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс принеобходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

**Вариант 2:**

Пользователь вводит команды вида: «число число число. Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип float. Количество чисел может быть произвольным.

## 

## **Общие сведения о программе**

Программа состоит из одного файла lab4.c.

Программа использует следующие системные вызовы:

1. **sem\_open –** для создания нового именнованного семафора.
2. **sem\_unlink** – для удаления именованного семафора.
3. **open -** для создания файла и его открытия.
4. **close** – для закрытия файлового дескриптора.
5. **mmap –** для отображения файла в память.
6. **fork** – для создания дочернего процесса.
7. **sem\_wait –** для блокировки семафора.
8. **sem\_post** – для разблокировки семафора.
9. **dup2** – для перенаправления потока вывода.
10. **getpid** - для получения id процесса.
11. **ftruncate** — обрезает/расширяет файл до заданного размера.
12. **remove** — для удаления файла.

**Общий метод и алгоритм решения**

* создать именованный семофор, проверить, создать файл shared\_fds1.txt**,** отобразить его в память c помощью **mmap** как shared\_fds**\_**1, создать дочерний процесс(с помощью **fork**) и обработать возможные ошибки.
* Из родительского процесса: блокируем семафор записываем в разделенную память данные (список чисел), разблокируем семафор.
* В дочернем процесс блокируем семафор, читаем из разделенной памяти список чисел и выводим их сумму в файл
* В конце родительского процесса удаляем созданные файлы, удаляем семафор

## **Код программы**

// to run: gcc lab4.c -o lab4 -lpthread -lrt

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <sys/stat.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <semaphore.h>

#define SEM "/semaphore"

#define SIZE 200

int count(char \*str)

{

int counter = 1;

for (int i = 0; i < strlen(str); ++i)

if (str[i] == ' ')

counter++;

return counter;

}

float sum(FILE \*f, int count)

{

float res = 0.0, a;

rewind(f);

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

fscanf(f, "%f", &a);

res += a;

}

return res;

}

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

{

// remove semaphore if it was not deleted

sem\_unlink(SEM);

// create semaphore with:

// name SEM, flag O\_CREAT|O\_EXCL means

// that an error is returned if a semaphore with the given name already exists

// create file with READ and WRITE permissions only for me

// initial value of the semaphore will be 1 (busy)

sem\_t\* semaphore = sem\_open(SEM, O\_CREAT|O\_EXCL, 0600, 1);

if (semaphore == SEM\_FAILED) {

perror("semaphore");

exit(1);

}

char file\_name[SIZE];

char read\_file\_name[SIZE];

char sequence\_of\_numbers[SIZE];

char read\_sequence\_of\_numbers[SIZE];

// create file with READ and WRITE permissions only for me

int shared\_fds1 = open("shared\_fds1.txt", O\_RDWR | O\_CREAT, 0600);

if (shared\_fds1 == -1) {

perror("create file");

exit(1);

}

// 0 - we dont care were our memory will be stored

// page size, read and write permissions

// file descriptor is our shared\_fds1

// shift is 0

char\* shared\_fds\_1 = (char\*)mmap(0, sysconf(\_SC\_PAGESIZE), PROT\_READ | PROT\_WRITE, MAP\_SHARED, shared\_fds1, 0);

if (shared\_fds\_1 == MAP\_FAILED){

perror("map");

exit(1);

}

printf("Print input name of file: ");

fgets(file\_name, SIZE, stdin);

if (file\_name[strlen(file\_name) - 1] == '\n')

file\_name[strlen(file\_name) - 1] = '\0';

// only writing

int f = open(file\_name, O\_WRONLY | O\_CREAT, 0600);

if (f == -1) {

perror("create file");

exit(1);

}

pid\_t child\_pid = fork();

if (child\_pid == -1) {

perror("fork");

exit(1);

}

// Child Process

else if (child\_pid == 0) {

// decrement semaphore to 0

sem\_wait(semaphore);

printf("[Child Process, id=%d]: computing...\n", getpid());

// f (FD) assign to 1 (stdout)

if (dup2(f, 1) == -1) {

perror("dup2");

exit(1);

}

for (int i = 0; i < strlen(shared\_fds\_1); i++) {

read\_sequence\_of\_numbers[i] = shared\_fds\_1[i];

}

int cnt = count(read\_sequence\_of\_numbers);

//

FILE \*write\_seq;

if ((write\_seq = fopen(file\_name, "w")) == NULL) {

perror("open file");

exit(1);

} // added error handling

fputs(read\_sequence\_of\_numbers, write\_seq);

fclose(write\_seq);

FILE \*read\_seq;

if ((read\_seq = fopen(file\_name, "r")) == NULL) {

perror("open file");

exit(1);

} // added error handling

float res = sum(read\_seq, cnt);

fclose(read\_seq);

printf("count is %d. Sum is %.10f\n", cnt, res);

// increment semaphore to 1

sem\_post(semaphore);

}

// Parent process

else {

// decrement semaphore to 0

sem\_wait(semaphore);

printf("[%d] It's parent. Child id: %d\n", getpid(), child\_pid);

// write to memory

printf("[Parent Process, id=%d]: Enter the sequence of real type numbers: ", getpid());

fgets(sequence\_of\_numbers, SIZE, stdin);

int length = strlen(sequence\_of\_numbers) \* sizeof(char);

// truncate shared mamory to the sequence\_of\_numbers size (in bytes)

if (ftruncate(shared\_fds1, length)) {

perror("ftruncate");

exit(1);

}

for(int i = 0; i < strlen(sequence\_of\_numbers); i++){

shared\_fds\_1[i] = sequence\_of\_numbers[i];

}

printf("[Parent Process, id=%d]: Writing to the shared\_fds. Message is sequence of numbers: %s\n\n", getpid(), sequence\_of\_numbers);

// increment semaphore to 1

sem\_post(semaphore);

close(shared\_fds1);

close(f);

sem\_close(semaphore);

remove("shared\_fds1.txt");

return 0;

}

}

## **Использование утилиты strace**

[suraba04@asusx512fl lab4]$ strace ./my\_try2

execve("./my\_try2", ["./my\_try2"], 0x7ffd7503c0c0 /\* 72 vars \*/) = 0

brk(NULL) = 0x55ccdc76e000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffe2081f4b0) = -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=212244, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 212244, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f844bac3000

close(3) = 0

openat(AT\_FDCWD, "/usr/lib/libpthread.so.0", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\300\200\0\0\0\0\0\0"..., 832) = 832

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, 792) = 80

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\7\310\371[O2Q\320\205P!z\330\241\363\20"..., 68, 872) = 68

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=154040, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f844bac1000

mmap(NULL, 131472, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f844baa0000

mprotect(0x7f844baa7000, 81920, PROT\_NONE) = 0

mmap(0x7f844baa7000, 61440, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x7000) = 0x7f844baa7000

mmap(0x7f844bab6000, 16384, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x16000) = 0x7f844bab6000

mmap(0x7f844babb000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1a000) = 0x7f844babb000

mmap(0x7f844babd000, 12688, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f844babd000

close(3) = 0

openat(AT\_FDCWD, "/usr/lib/librt.so.1", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\2207\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=39408, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 43520, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f844ba95000

mmap(0x7f844ba98000, 16384, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x3000) = 0x7f844ba98000

mmap(0x7f844ba9c000, 8192, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x7000) = 0x7f844ba9c000

mmap(0x7f844ba9e000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x8000) = 0x7f844ba9e000

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`|\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\0K@g7\5w\10\300\344\306B4Zp<G"..., 68, 928) = 68

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2150424, ...}, 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, 1880536, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f844b8c9000

mmap(0x7f844b8ef000, 1355776, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x26000) = 0x7f844b8ef000

mmap(0x7f844ba3a000, 311296, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x171000) = 0x7f844ba3a000

mmap(0x7f844ba86000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bc000) = 0x7f844ba86000

mmap(0x7f844ba8c000, 33240, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f844ba8c000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f844b8c6000

arch\_prctl(ARCH\_SET\_FS, 0x7f844b8c6740) = 0

mprotect(0x7f844ba86000, 12288, PROT\_READ) = 0

mprotect(0x7f844babb000, 4096, PROT\_READ) = 0

mprotect(0x7f844ba9e000, 4096, PROT\_READ) = 0

mprotect(0x55ccdb7f7000, 4096, PROT\_READ) = 0

mprotect(0x7f844bb25000, 8192, PROT\_READ) = 0

munmap(0x7f844bac3000, 212244) = 0

set\_tid\_address(0x7f844b8c6a10) = 16308

set\_robust\_list(0x7f844b8c6a20, 24) = 0

rt\_sigaction(SIGRTMIN, {sa\_handler=0x7f844baa7b70, sa\_mask=[], sa\_flags=SA\_RESTORER|SA\_SIGINFO, sa\_restorer=0x7f844bab3870}, NULL, 8) = 0

rt\_sigaction(SIGRT\_1, {sa\_handler=0x7f844baa7c10, sa\_mask=[], sa\_flags=SA\_RESTORER|SA\_RESTART|SA\_SIGINFO, sa\_restorer=0x7f844bab3870}, NULL, 8) = 0

rt\_sigprocmask(SIG\_UNBLOCK, [RTMIN RT\_1], NULL, 8) = 0

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

statfs("/dev/shm/", {f\_type=TMPFS\_MAGIC, f\_bsize=4096, f\_blocks=999075, f\_bfree=957080, f\_bavail=957080, f\_files=999075, f\_ffree=997786, f\_fsid={val=[0, 0]}, f\_namelen=255, f\_frsize=4096, f\_flags=ST\_VALID|ST\_NOSUID|ST\_NODEV}) = 0

futex(0x7f844bac0130, FUTEX\_WAKE\_PRIVATE, 2147483647) = 0

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

getrandom("\x46\x5e\x03\x30\xe2\xa9\xe9\xb4", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/A2sehg", 0x7ffe2081f070, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/dev/shm/A2sehg", O\_RDWR|O\_CREAT|O\_EXCL, 0600) = 3

write(3, "\1\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, 3, 0) = 0x7f844baf6000

link("/dev/shm/A2sehg", "/dev/shm/sem.semaphore") = 0

newfstatat(3, "", {st\_mode=S\_IFREG|0600, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

brk(NULL) = 0x55ccdc76e000

brk(0x55ccdc78f000) = 0x55ccdc78f000

unlink("/dev/shm/A2sehg") = 0

close(3) = 0

openat(AT\_FDCWD, "shared\_fds1.txt", O\_RDWR|O\_CREAT, 0600) = 3

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7f844baf5000

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, "Print input name of file: ", 26Print input name of file: ) = 26

read(0, 0x55ccdc76e710, 1024) = ? ERESTARTSYS (To be restarted if SA\_RESTART is set)

--- SIGWINCH {si\_signo=SIGWINCH, si\_code=SI\_KERNEL} ---

read(0, file11

"file11\n", 1024) = 7

openat(AT\_FDCWD, "file11", O\_WRONLY|O\_CREAT, 0600) = 4

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f844b8c6a10) = 16369

getpid() = 16308

write(1, "[16308] It's parent. Child id: 1"..., 37[16308] It's parent. Child id: 16369

) = 37

getpid() = 16308

write(1, "[Parent Process, id=16308]: Ente"..., 69[Parent Process, id=16308]: Enter the sequence of real type numbers: ) = 69

read(0, 08327460.3256 23572357.2 572372457.245725 235723.5723570x55ccdc76e710, 1024) = ? ERESTARTSYS (To be restarted if SA\_RESTART is set)

--- SIGWINCH {si\_signo=SIGWINCH, si\_code=SI\_KERNEL} ---

read(0, 364235.7235723 252.582 2357

"08327460.3256 23572357.2 5723724"..., 1024) = 83

ftruncate(3, 83) = 0

getpid() = 16308

write(1, "[Parent Process, id=16308]: Writ"..., 170[Parent Process, id=16308]: Writing to the shared\_fds. Message is sequence of numbers: 08327460.3256 23572357.2 572372457.245725 235723.572357364235.7235723 252.582 2357

) = 170

write(1, "\n\n", 2

) = 2

futex(0x7f844baf6000, FUTEX\_WAKE, 1) = 1

close(3) = 0

[Child Process, id=16369]: computing...

close(4) = 0

munmap(0x7f844baf6000, 32) = 0

unlink("shared\_fds1.txt") = 0

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=16369, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

exit\_group(0) = ?

+++ exited with 0 +++

## 

## **Распечатка протокола работы программы**

[suraba04@asusx512fl lab4]$ ./fin\_version

Print input name of file: f15

[22318] It's parent. Child id: 22319

[Parent Process, id=22318]: Enter the sequence of real type numbers: 346.246 247.27 248.2 4

[Parent Process, id=22318]: Writing to the shared\_fds. Message is sequence of numbers: 346.246 247.27 248.2 4

[Child Process, id=22319]: computing...

[suraba04@asusx512fl lab4]$ cat f15

845.716003[suraba04@asusx512fl lab4]$ ./fin\_version

Print input name of file: f16

[22376] It's parent. Child id: 22389

[Parent Process, id=22376]: Enter the sequence of real type numbers: 48967.2357 45 0

[Parent Process, id=22376]: Writing to the shared\_fds. Message is sequence of numbers: 48967.2357 45 0

[Child Process, id=22389]: computing...

[suraba04@asusx512fl lab4]$ cat f16

49012.234375[suraba04@asusx512fl lab4]$ ./fin\_version

Print input name of file: f17

[22426] It's parent. Child id: 22427

[Parent Process, id=22426]: Enter the sequence of real type numbers: 0

[Parent Process, id=22426]: Writing to the shared\_fds. Message is sequence of numbers: 0

[Child Process, id=22427]: computing...

[suraba04@asusx512fl lab4]$ cat f17

0.000000[suraba04@asusx512fl lab4]$

## **Вывод**

При выполнении данной работы я вспомнил как работать с процессами в си. Научился синхронизировать работу процессов с помощью семафора. Так как в ЛР2 обмен данными осуществлялся через пайп, там не обязательно было использовать мьютексы, семафоры и другие приспособления для синхронизации. Однако в ЛР4 обмен данными происходит через файл, который отображен в память, работа с ним идет быстрее нежели с обычным файлом, но, чтобы процессы не мешали друг другу, мы вынуждены использовать, например, семафор, который бы блокировал один процесс, пока другой делал что-то с общей памятью. В целом работа оказалась несложной, так как я не делал второй файл (для обратного обмена данными) и семафор для него, однако я все равно узнал много нового о программировании на си под POSIX совместимые системы.