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:

Москва, 2021

Содержание

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 с 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"..., 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\200\0\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.