Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский Авиационный Институт»

Национальный Исследовательский Университет

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №4

по курсу «Операционные системы»

Студент:	Хренникова А. С.
Группа:	М8О-208-19
Преподаватель:	Миронов Е. С.
Подпись:	
Оценка:	
Дата:	

Москва, 2020

Содержание

Цель работы

Приобретение практических навыков в:

- Освоении принципов работы с файловыми системами;
- Обеспечении обмена данными между процессами посредством «File mapping».

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.

Вариант 8: Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.
В файле записаны команды вида: «число число число<endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип int.

Общие сведения о программе

Программа состоит из двух файлов: main1.c и main2.c. В данных файлах используются заголовочные файлы: stdio.h, stdlib.h, string.h, unistd.h, sys/mman.h, fcntl.h, sys/wait.h, stddef.h.

1. fork – создает дочерний процесс, идентичный родительскому;
2. execv – заменяет образ памяти процесса;
3. close – закрывает открытый файл;
4. freopen – функции для открытия потоков;
5. ftruncate – укорачивает файл до указанной длины;
6. shm_open, shm_unlink – создает/открывает или снимает объекты разделяемой памяти POSIX;
7. mmap, munmap – отражает файлы или устройства в памяти или снимает их отражение;
8. waitpid – ожидает завершения процесса;
9. signal – определяет способ обработки сигналов.

Общий метод и алгоритм решения

- Читать имя файла для ввода как аргумент, создать дочерний процесс(с помощью fork. Дочерний процесс будет открывать файл в разделяемой памяти, расширять его до указанного значения и отображать его в свое адресное пространство. Результат работы дочернего процесса будет помещаться в разделяемую память. После окончания считывания, дочерний процесс должен удалить отображение разделяемой памяти и закрыть файловые дескрипторы.
- Для родительского процесса: открыть файл, запустить программу с дочерним процессом. Позже вывести данные, которые передаст дочерний процесс после их обработки.
- Для дочернего процесса: считывать числа по символам, производить деление(при делении на 0 завершить работу дочернего и родительского

процессов), передавать результат родительскому процессу, завершить работу.

Код программ

main1.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <stddef.h>

#define MEMORY_SIZE 4096
#define DATA_SIZE 256
#define MEMORY_NAME "lab4"

#if DATA_SIZE > MEMORY_SIZE
#warning Segfault may occur
#endif

typedef struct res {
	size_t size;
	int data[DATA_SIZE];
} res_t;

int main() {
	FILE *fp = NULL;
	printf("Enter file name to read data: ");
	char *name=(char *)malloc(256);
	scanf("%s",name);
	int pr = fork();
	if (pr == -1) {
		printf("Can\'t fork child!\n");
		exit(0);
	} else if (pr == 0) {
		fp = freopen(name, "r", stdin);
		if (fp == NULL) {
			printf("Can\'t open file!\n");
			exit(0);
		}
		char * const * argv = NULL;
		if (execv("main2", argv) == -1) {
			printf("Can\'t execute child process!\n");
			exit(0);
		}
	} else {
		int status;
		if (waitpid(pr, &status, 0) == -1) {
			printf("Waitpid error!\n");
		}
		if (WIFSIGNALED(status)) {
			fprintf(stderr, "Child process terminated by signal: %d\n", WTERMSIG(status));
			shm_unlink(MEMORY_NAME);
			exit(1);
		}
		if (WEXITSTATUS(status) != 0) {
```

```
            exit(1);
        }
        int fd = shm_open(MEMORY_NAME, O_RDONLY, S_IRUSR | S_IWUSR);
        if (fd == -1) {
            printf("Can\'t open shared memory file\n");
        }
        res_t *addr = mmap(NULL, MEMORY_SIZE, PROT_READ, MAP_SHARED, fd, 0);
        if (addr == (void *) -1) {
            printf("Mmap error!\n");
        }
        for (int i = 0; i < addr->size; i++) {
            printf("%d\n", addr->data[i]);
        }
        munmap(addr, MEMORY_SIZE);
        shm_unlink(MEMORY_NAME);
        close(fd);
    }
    fclose(stdin);
    return 0;
}
```

main2.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <stddef.h>

#define DATA_SIZE 256
#define MEMORY_SIZE 4096
#define MEMORY_NAME "lab4"

#if DATA_SIZE > MEMORY_SIZE
#warning Segfault may occur
#endif

typedef struct res {
    size_t size;
    int data[DATA_SIZE];
} res_t;

void termination_handler() {
   fprintf(stderr, "Segmentation fault occur, try decrease DATA_SIZE macro\n");
   shm_unlink("lab4");
   exit(1);
}

int main() {
    signal(SIGSEGV, termination_handler);
    int fd = shm_open(MEMORY_NAME, O_EXCL | O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
    if (fd == -1) {
        printf("Can't open shared memory file\n");
    }
```

```
    if (ftruncate(fd, MEMORY_SIZE) == -1) {
        printf("Can't extent shared memory file\n");
    }
    res_t *addr = mmap(NULL, MEMORY_SIZE, PROT_WRITE, MAP_SHARED, fd, 0);
    if (addr == (void *) -1) {
        printf("Mmap error!\n");
    }
    addr->size = 0;
    char c;
    int num = 0, res = 0, minus = 0, top = 1, pr = 0;
    while (scanf("%c", &c) > 0) {
        if (c == ' ' || c == '\t') {
            if (num == 0) {
                break;
            } else {
                if (top == 1)
                    top = 0;
                else if (minus == 1)
                    res = (res / num) * (-1);
                else res = res / num;
            }
            num = 0;
            top = 0;
            minus = 0;
            pr = 1;
        } else if (c == '-') {
            minus = 1;
            pr = 0;
        } else if (c == '\n') {
            if (num == 0) {
                if (pr == 1) res = res / 1;
                else break;
            } else {
                if (top == 1)
                    top = 0;
                else if (minus == 1)
                    res = (res / num) * (-1);
                else res = res / num;
            }
            addr->data[addr->size++] = res;
            num = 0;
            top = 1;
            minus = 0;
            res = 0;
            pr = 0;
        } else if (c >= '0' && c <= '9') {
            num = num *10 + c - '0';
                if (top == 1)
                    res = num;
            pr = 0;
        }
    }
    munmap(addr, MEMORY_SIZE);
    close(fd);
    return 0;
}
```

Демонстрация работы программы

```
lina_tucha@LAPTOP-44CRFC1U:~/labs/os/lab2$ cat test.txt
24 2 3
45 1 1 5
50 2 5
144 1 1 1 -1 1 1 2 1 3
108 1 1 1 1 5 1 1 1 3 1
144 1 1 1 1 1 12 1 1 1 1 1 2 1 1 1
44 4 1 1 1 1 0 1 1 1 11

lina_tucha@LAPTOP-44CRFC1U:~/labs/os/lab2$ ./main1
Enter file name to read data: test.txt
4
9
5
-24
7
6
lina_tucha@LAPTOP-44CRFC1U:~/labs/os/lab2$ cat test1.txt
78 2 1
77 11 1
210 2 1 1
108 5 1 1 1 1 1 1 2
210 2
450 1 5 1 0 1 1 1 2 1 1 3 1
102 1 1 1 2 1 1

lina_tucha@LAPTOP-44CRFC1U:~/labs/os/lab2$ ./main1
Enter file name to read data: test1.txt
39
7
105
10
105
```

Вывод strace

```
lina_tucha@LAPTOP-44CRFC1U:~/labs/os/lab4$ strace ./ga1 < test.txt
execve("./ga1", ["./ga1"], 0x7fffca99d280 /* 19 vars */) = 0
brk(NULL)                               = 0x7fffc7f4f000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
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
fstat(3, {st_mode=S_IFREG|0644, st_size=47603, ...}) = 0
mmap(NULL, 47603, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f6661e51000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/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\0\"\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=31680, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f6661e40000
mmap(NULL, 2128864, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f66619f0000
mprotect(0x7f66619f7000, 2093056, PROT_NONE) = 0
mmap(0x7f6661bf6000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x6000) = 0x7f6661bf6000
```

```
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
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\0\260\34\2\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2030544, ...}) = 0
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f66615f0000
mprotect(0x7f66617d7000, 2097152, PROT_NONE) = 0
mmap(0x7f66619d7000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f66619d7000
mmap(0x7f66619dd000, 15072, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f66619dd000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0000b\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=144976, ...}) = 0
mmap(NULL, 2221184, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f66613d0000
mprotect(0x7f66613ea000, 2093056, PROT_NONE) = 0
mmap(0x7f66615e9000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19000) = 0x7f66615e9000
mmap(0x7f66615eb000, 13440, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f66615eb000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f6661e30000
arch_prctl(ARCH_SET_FS, 0x7f6661e30740) = 0
mprotect(0x7f66619d7000, 16384, PROT_READ) = 0
mprotect(0x7f66615e9000, 4096, PROT_READ) = 0
mprotect(0x7f6661bf6000, 4096, PROT_READ) = 0
mprotect(0x7f6662201000, 4096, PROT_READ) = 0
mprotect(0x7f6661e27000, 4096, PROT_READ) = 0
munmap(0x7f6661e51000, 47603)           = 0
set_tid_address(0x7f6661e30a10)         = 33
set_robust_list(0x7f6661e30a20, 24)     = 0
rt_sigaction(SIGRTMIN, {sa_handler=0x7f66613d5cb0, sa_mask=[], sa_flags=SA_RESTORER|SA_SIGINFO,
sa_restorer=0x7f66613e28a0}, NULL, 8) = 0
rt_sigaction(SIGRT_1, {sa_handler=0x7f66613d5d50, sa_mask=[],
sa_flags=SA_RESTORER|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f66613e28a0}, NULL, 8) = 0
rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=8192*1024}) = 0
brk(NULL)                               = 0x7fffc7f4f000
brk(0x7fffc7f70000)                     = 0x7fffc7f70000
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f6661e30a10) = 34
wait4(34, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 34
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=34, si_uid=1000, si_status=0, si_utime=0,
si_stime=0} ---
statfs("/dev/shm/", {f_type=TMPFS_MAGIC, f_bsize=4096, f_blocks=62187775, f_bfree=26835227,
f_bavail=26835227, f_files=999, f_ffree=1000000, f_fsid={val=[1, 0]}, f_namelen=255, f_frsize=4096,
f_flags=ST_VALID|ST_NOSUID|ST_NODEV|ST_NOATIME}) = 0
futex(0x7f66615ee370, FUTEX_WAKE_PRIVATE, 2147483647) = 0
openat(AT_FDCWD, "/dev/shm/lab4", O_RDONLY|O_NOFOLLOW|O_CLOEXEC) = 3
mmap(NULL, 4096, PROT_READ, MAP_SHARED, 3, 0) = 0x7f6661e5c000
fstat(1, {st_mode=S_IFCHR|0660, st_rdev=makedev(4, 1), ...}) = 0
ioctl(1, TCGETS, {B38400 opost isig icanon echo ...}) = 0
write(1, "50\n", 350
)                 = 3
write(1, "10\n", 310
```

```
)                       = 3
write(1, "7\n", 27
)                       = 2
write(1, "50\n", 350
)                       = 3
munmap(0x7f6661e5c000, 4096)            = 0
unlink("/dev/shm/lab4")                 = 0
close(3)                                = 0
close(0)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

Отображение файлов дает удобство при работе с файлами, так как позволяет работать с областью файла как с обычным участком памяти. Другими словами, мы имеем доступ к каждому байту области памяти, которую мы отобразили, также количество системных вызовов по чтению и записи сводится к нулю, так как мы работаем с оперативной памятью. Но также отображение файлов дает нам возможность в межпроцессорном взаимодействии. При отображении файла на участок памяти, этой память могут разделять несколько процессов, но в отличии от pipe, теперь синхронизация остается на разработчике.

Данная лабораторная работа помогла мне разобраться с «File mapping» в теории и на практике. Я поняла, как создавать отображения физических файлов и как можно делать анонимные отображения. Также изучила работу с разделяемой памятью(shm_open/shm_unlink).