

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: М8О-209БВ-24

Студент: Хамзин Т.Н.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 15.12.24

Москва, 2024

Постановка задачи

Вариант 4.

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. Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. 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 float. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- `int ftruncate(int fd, off_t length)` - Создание/открытие именованного shared memory
- `int shm_open(const char *name, int oflag, mode_t mode)` - Установка размера shared memory
- `void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset)` - Отображение shared memory в адресное пространство процесса
- `int sem_init(sem_t *sem, int pshared, unsigned int value)` - Инициализация семафора
- `int sem_post(sem_t *sem)` - Увеличение значения семафора (освобождение)
- `int sem_wait(sem_t *sem)` - Уменьшение значения семафора (захват)
- `int sem_destroy(sem_t *sem)` - Уничтожение семафора

Я считал данные в родительском процессе и прописал поведение программы для разных pid, выделил память для доступа процессов к общим ресурсам, затем в дочернем процессе произвел необходимые математические действия (согласно заданию) ответ записал в файл, обработал возможный случай деления числа на 0 (написал условие для завершения обоих процессов)

Код программы

Main.c

```
#include <stdio.h>

#include <semaphore.h>

#include "stdlib.h"

#include "sys/wait.h"

#include <sys/mman.h>
```

```

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <pthread.h>

#include <string.h> //подключил для strcpy!


int main(){
    typedef struct{
        char file_name[100];

        float numbers[100];

        sem_t sem;

        int count;
    } shared_data;


    int fd=shm_open("/my_shared_memory", O_CREAT | O_RDWR, 0666);

    if (fd==-1){
        fprintf(stderr,"shm_open error\n");
        exit(1);
    }

    ftruncate(fd, sizeof(shared_data));


    shared_data *data = mmap(NULL, sizeof(shared_data), PROT_READ | PROT_WRITE, MAP_SHARED,
fd, 0);

    if (data==MAP_FAILED){
        fprintf(stderr,"mmap error\n");
        exit(1);
    }

    if (sem_init(&data->sem, 1, 0)==-1){
        fprintf(stderr,"Sem error\n");
        exit(1);
    }

    // обработка родителем

    fgets(data->file_name,sizeof(data->file_name),stdin);

```

```

    data->file_name[strcspn(data->file_name, "\n")] = '\0';

    int i=0;

    while (scanf("%f",&data->numbers[i])==1){

        ++i;

    }

    data->count=i;


    pid_t pid = fork();

    if (pid==-1){

        fprintf(stderr,"fork error\n");

        exit(1);

    }

    if (pid==0){

        FILE *file = fopen(data->file_name,"w");

        if (file==NULL){

            fprintf(stderr,"File error");

            exit(1);

        }

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

            if (data->numbers[i] == 0){

                sem_post(&data->sem);

                fprintf(stderr,"division by zero, stopping both processes\n");

                exit(EXIT_FAILURE);

            }

            float result = data->numbers[0] / data->numbers[i];

            fprintf(file, "%f\n", result);

            fflush(file);

        }

        fclose(file);

        sem_post(&data->sem);

        exit(EXIT_SUCCESS);

```

```

    } else{

sem_wait(&data->sem);

int status;

wait(&status);

if (WIFEXITED(status))

{

    int exit_code = WEXITSTATUS(status);

    if (exit_code == EXIT_FAILURE)

    {

        fprintf(stderr,"division by zero detected. Parent exiting.\n");

        munmap(data, sizeof(shared_data));


        close(fd);

        shm_unlink("/my_shared_memory");

        exit(EXIT_FAILURE);

    }

}

munmap(data, sizeof(shared_data));

close(fd);

shm_unlink("/my_shared_memory");

}

}

```

Протокол работы программы

Тестирование:

```
$ ./main.c
```

```
test.txt
```

```
1.2 4.3268 83.25 0.36
```

```
cat test.txt
```

```
0.277341
```

```
0.014414
```

```
3.333333
```

strace -f ./main

```
execve("./main", [ "./main" ], 0x7ffebd0711d8 /* 31 vars */) = 0

brk(NULL)                                     = 0x3b7d0000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ddfa96db000

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=33091, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 33091, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ddfa96d2000

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\0\20t\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

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1926232, ...}, 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, 1974096, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ddfa94f0000

mmap(0x7ddfa9516000, 1400832, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x26000) = 0x7ddfa9516000

mmap(0x7ddfa966c000, 339968, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17c000) =
0x7ddfa966c000

mmap(0x7ddfa96bf000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1cf000) = 0x7ddfa96bf000

mmap(0x7ddfa96c5000, 53072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1,
0) = 0x7ddfa96c5000

close(3)                                     = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ddfa94ed000

arch_prctl(ARCH_SET_FS, 0x7ddfa94ed740) = 0

set_tid_address(0x7ddfa94eda10)             = 29426

set_robust_list(0x7ddfa94eda20, 24)         = 0

rseq(0x7ddfa94ee060, 0x20, 0, 0x53053053) = 0

mprotect(0x7ddfa96bf000, 16384, PROT_READ) = 0

mprotect(0x403000, 4096, PROT_READ)         = 0

mprotect(0x7ddfa970e000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
```

```

munmap(0x7ddfa96d2000, 33091) = 0

openat(AT_FDCWD, "/dev/shm/my_shared_memory", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3

ftruncate(3, 544) = 0

mmap(NULL, 544, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7ddfa96da000

newfstatat(0, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0

getrandom("\x2f\xec\xa1\x7f\xf9\x78\x2c\xb8", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x3b7d0000

brk(0x3b7f1000) = 0x3b7f1000

read(0, file.txt
"file.txt\n", 1024) = 9

read(0, 1 2 3 3 4 5 6 7 8 9 9 10
"1 2 3 3 4 5 6 7 8 9 9 10\n", 1024) = 25

read(0, "", 1024) = 0

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process
29626 attached
, child_tidptr=0x7ddfa94eda10) = 29626

[pid 29626] set_robust_list(0x7ddfa94eda20, 24 <unfinished ...>

[pid 29426] futex(0x7ddfa96da1f8, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 29626] <... set_robust_list resumed>) = 0

[pid 29626] openat(AT_FDCWD, "file.txt", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 4

[pid 29626] newfstatat(4, "", {st_mode=S_IFREG|0644, st_size=0, ...}, AT_EMPTY_PATH) = 0

[pid 29626] write(4, "1.000000\n", 9) = 9

[pid 29626] write(4, "0.500000\n", 9) = 9

[pid 29626] write(4, "0.333333\n", 9) = 9

[pid 29626] write(4, "0.333333\n", 9) = 9

[pid 29626] write(4, "0.250000\n", 9) = 9

[pid 29626] write(4, "0.200000\n", 9) = 9

[pid 29626] write(4, "0.166667\n", 9) = 9

[pid 29626] write(4, "0.142857\n", 9) = 9

[pid 29626] write(4, "0.125000\n", 9) = 9

[pid 29626] write(4, "0.111111\n", 9) = 9

[pid 29626] write(4, "0.111111\n", 9) = 9

[pid 29626] write(4, "0.100000\n", 9) = 9

```

```

[pid 29626] close(4)                                = 0
[pid 29626] futex(0x7ddfa96da1f8, FUTEX_WAKE, 1 <unfinished ...>
[pid 29426] <... futex resumed>)                    = 0
[pid 29626] <... futex resumed>)                    = 1
[pid 29426] wait4(-1, <unfinished ...>
[pid 29626] exit_group(0)                           = ?
[pid 29626] +++ exited with 0 +++

<... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 29626

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=29626, si_uid=0, si_status=0,
si_etime=0, si_stime=0} ---

munmap(0x7ddfa96da000, 544)                          = 0
close(3)                                              = 0
unlink("/dev/shm/my_shared_memory")                 = 0
exit_group(0)                                         = ?
+++ exited with 0 +++

```

Вывод

Данная лабораторная работа демонстрирует механизм межпроцессного взаимодействия через разделяемую память (shared memory). Родительский процесс создает общую область памяти, считывает входные данные и запускает дочерний процесс, который выполняет математические вычисления и записывает результаты в файл. Реализация включает обработку ошибок (деление на ноль) и корректное освобождение ресурсов, подтверждая эффективность shared memory для обмена данными между процессами без излишнего копирования.