Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №1 по курсу

«Операционные системы»

Группа: М8О-214Б-23

Студент: Ефременко К.А.

Преподаватель: Бахарев В.Д. (ФИИТ)

Оценка: _____

Дата: 01.11.24

Москва, 2024

Постановка задачи

Вариант 4.

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

Кратко опишите системные вызовы, которые вы использовали в лабораторной работе.

Использованные системные вызовы:

- **`shm_open()`**

Создает или открывает объект общей памяти, предоставляя возможность совместного использования данных между процессами.

- **`ftruncate()`**

Устанавливает размер объекта общей памяти. Используется после `shm_open()` для выделения необходимого объема памяти.

- **`mmap()`**

Отображает объект общей памяти в адресное пространство процесса, предоставляя доступ к общей памяти как к обычному массиву данных.

- **`sem_open()`**

Создает или открывает именованный семафор для управления доступом к общей памяти.

- **`sem_wait()`**

Блокирует процесс до тех пор, пока значение семафора не станет положительным, после чего уменьшает его на единицу (используется для синхронизации).

- **`sem_post()`**

Увеличивает значение семафора, сигнализируя другим процессам, что ресурс доступен.

- **`write()`**

Используется для записи данных в файл или стандартный вывод. Работает на уровне системных вызовов.

- **`open()`**

Открывает файл для записи (или чтения, если требуется), возвращая файловый дескриптор.

- **`close()`**

Закрывает открытый файловый дескриптор, освобождая ресурс.

- **`fork()`**

Создает новый процесс (дочерний), который является копией родительского.

- **`wait()`**

Приостанавливает выполнение родительского процесса до завершения дочернего.

- **`munmap()`**

Удаляет отображение общей памяти из адресного пространства процесса.

- **`shm_unlink()`**

Удаляет объект общей памяти.

- **`sem_unlink()`**

Удаляет именованный семафор

- **Инициализация общей памяти и семафоров**:
  - Создается объект общей памяти (`shm_open()`), устанавливается его размер (`ftruncate()`), и он отображается в адресное пространство процесса (`mmap()`).
  - Создаются два семафора: для синхронизации записи и чтения.
- **Разделение процессов**:
  - Родительский процесс считывает данные (три числа) от пользователя через стандартный ввод.
  - Дочерний процесс создается с помощью `fork()`.
- **Обработка в родительском процессе**:
  - Родитель записывает введенные числа в общую память.
  - Уведомляет дочерний процесс через семафор, что данные готовы (`sem_post()`).
- **Обработка в дочернем процессе**:
  - Ожидает готовности данных от родителя (`sem_wait()`).
  - Считывает числа из общей памяти, выполняет расчеты (деление первого числа на второе и результат на третье).
  - Записывает результат в файл через системные вызовы `open()` и `write()`.
- **Очистка ресурсов**:
  - После завершения работы процессы закрывают и освобождают общую память (`munmap()` и `shm_unlink()`) и семафоры (`sem_close()` и `sem_unlink()`).
  - Родитель ждет завершения дочернего процесса (`wait()`).

Код программы

Lab3.c

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

#define SHARED_MEMORY_NAME "/shared_mem"
#define SEM_WRITE_NAME "/sem_write"
#define SEM_READ_NAME "/sem_read"
#define BUFFER_SIZE 128

void handle_error(const char *msg)
{
    write(STDERR_FILENO, msg, strlen(msg));
    exit(EXIT_FAILURE);
}

typedef struct
{
    float numbers[3];
    char result[BUFFER_SIZE];
} SharedData;

int main()
{
    int fd;
    SharedData *sharedData;
    sem_t *semWrite, *semRead;

    fd = shm_open(SHARED_MEMORY_NAME, O_CREAT | O_RDWR, 0666);
    if (fd == -1)
    {
        handle_error("Failed to create shared memory\n");
    }
    if (ftruncate(fd, sizeof(SharedData)) == -1)
    {
        handle_error("Failed to set size for shared memory\n");
    }

    sharedData = mmap(NULL, sizeof(SharedData), PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);
    if (sharedData == MAP_FAILED)
    {
        handle_error("Failed to map shared memory\n");
    }
```

```
    semWrite = sem_open(SEM_WRITE_NAME, O_CREAT, 0666, 0);
    if (semWrite == SEM_FAILED)
    {
        handle_error("Failed to create semWrite\n");
    }

    semRead = sem_open(SEM_READ_NAME, O_CREAT, 0666, 1);
    if (semRead == SEM_FAILED)
    {
        handle_error("Failed to create semRead\n");
    }

    pid_t pid = fork();
    if (pid < 0)
    {
        handle_error("Fork error\n");
    }

    if (pid > 0)
    {

        float a, b, c;
        char input[BUFFER_SIZE];
        write(STDOUT_FILENO, "Enter three numbers: ", 35);
        ssize_t bytes_read = read(STDIN_FILENO, input, sizeof(input) - 1);
        if (bytes_read <= 0)
        {
            handle_error("Failed to read input\n");
        }
        input[bytes_read - 1] = '\0';

        if (sscanf(input, "%f %f %f", &a, &b, &c) != 3)
        {
            handle_error("Invalid input, expected three numbers\n");
        }

        sharedData->numbers[0] = a;
        sharedData->numbers[1] = b;
        sharedData->numbers[2] = c;

        sem_post(semWrite);
        sem_wait(semRead);

        int fd_output = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
        if (fd_output < 0)
        {
            handle_error("Failed to open output file\n");
        }

        write(fd_output, sharedData->result, strlen(sharedData->result));
        close(fd_output);
```

```
        wait(NULL);
    }
    else
    {
        sem_wait(semWrite);

        float a = sharedData->numbers[0];
        float b = sharedData->numbers[1];
        float c = sharedData->numbers[2];

        if (b == 0.0f || c == 0.0f)
        {

            snprintf(sharedData->result, BUFFER_SIZE, "Error: division by
zero\n");
        }
        else
        {

            float result = a / b / c;
            snprintf(sharedData->result, BUFFER_SIZE, "Result: %.2f / %.2f / %.2f
= %.2f\n", a, b, c, result);
        }

        sem_post(semRead);

        exit(EXIT_SUCCESS);
    }

    sem_close(semWrite);
    sem_close(semRead);
    sem_unlink(SEM_WRITE_NAME);
    sem_unlink(SEM_READ_NAME);
    munmap(sharedData, sizeof(SharedData));
    shm_unlink(SHARED_MEMORY_NAME);

    return 0;
}
```

lab3-client.c

```
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <stdio.h>
```

```c
#define SHARED_MEMORY_NAME "/shared_mem"
#define SEM_WRITE_NAME "/sem_write"
#define SEM_READ_NAME "/sem_read"
#define BUFFER_SIZE 128

void handle_error(const char *msg)
{
    write(STDERR_FILENO, msg, strlen(msg));
    exit(EXIT_FAILURE);
}

typedef struct
{
    float numbers[3];
    char result[BUFFER_SIZE];
} SharedData;

int main()
{
    int fd;
    SharedData *sharedData;
    sem_t *semWrite, *semRead;

    fd = shm_open(SHARED_MEMORY_NAME, O_RDWR, 0666);
    if (fd == -1)
    {
        handle_error("Failed to open shared memory\n");
    }

    sharedData = mmap(NULL, sizeof(SharedData), PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);
    if (sharedData == MAP_FAILED)
    {
        handle_error("Failed to map shared memory\n");
    }

    semWrite = sem_open(SEM_WRITE_NAME, 0);
    if (semWrite == SEM_FAILED)
    {
        handle_error("Failed to open semWrite\n");
    }

    semRead = sem_open(SEM_READ_NAME, 0);
    if (semRead == SEM_FAILED)
    {
        handle_error("Failed to open semRead\n");
    }

    sem_wait(semWrite);

    float a = sharedData->numbers[0];
```

```
    float b = sharedData->numbers[1];
    float c = sharedData->numbers[2];

    if (b == 0.0f || c == 0.0f)
    {
        snprintf(sharedData->result, BUFFER_SIZE, "Error: division by zero\n");
    }
    else
    {
        float result = a / b / c;
        snprintf(sharedData->result, BUFFER_SIZE, "Result: %.2f / %.2f / %.2f =
%.2f\n", a, b, c, result);
    }
    int fd_output = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd_output < 0)
    {
        handle_error("Failed to open output file\n");
    }

    write(fd_output, sharedData->result, strlen(sharedData->result));
    close(fd_output);

    sem_post(semRead);

    exit(EXIT_SUCCESS);
}
```

Протокол работы программы

```
execve("./lab3", ["./lab3", "10", "4", "2"], 0x7ffe51c3b4c8 /* 31 vars */) = 0
brk(NULL)                                       = 0x56484919e000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f829defe000
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=23179, ...}) = 0
mmap(NULL, 23179, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f829def8000
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\220\243\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
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 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, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f829dce6000
mmap(0x7f829dd0e000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f829dd0e000
mmap(0x7f829de96000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7f829de96000
mmap(0x7f829dee5000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f829dee5000
mmap(0x7f829deeb000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f829deeb000
close(3)                                        = 0
```

```
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f829dce3000
arch_prctl(ARCH_SET_FS, 0x7f829dce3740) = 0
set_tid_address(0x7f829dce3a10)         = 7255
set_robust_list(0x7f829dce3a20, 24)     = 0
rseq(0x7f829dce4060, 0x20, 0, 0x53053053) = 0
mprotect(0x7f829dee5000, 16384, PROT_READ) = 0
mprotect(0x564847790000, 4096, PROT_READ) = 0
mprotect(0x7f829df36000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f829def8000, 23179)           = 0
openat(AT_FDCWD, "/dev/shm/shared_mem", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) =
3
ftruncate(3, 140)                       = 0
mmap(NULL, 140, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f829defd000
openat(AT_FDCWD, "/dev/shm/sem.sem_write", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT
(No such file or directory)
getrandom("\xba\x43\xcd\xa3\xab\x9b\x81\xb0", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.Qy6j28", 0x7ffe47dc00b0, AT_SYMLINK_NOFOLLOW) = -
1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.Qy6j28", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC,
0666) = 4
write(4, "\0\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, 4, 0) = 0x7f829defc000
link("/dev/shm/sem.Qy6j28", "/dev/shm/sem.sem_write") = 0
fstat(4, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0
getrandom("\x0a\x65\x34\x21\x85\x12\x0c\x2b", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x56484919e000
brk(0x5648491bf000)                     = 0x5648491bf000
unlink("/dev/shm/sem.Qy6j28")           = 0
close(4)                                = 0
openat(AT_FDCWD, "/dev/shm/sem.sem_read", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT
(No such file or directory)
getrandom("\xd0\x6b\x00\xea\xca\xe6\x7c\x18", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.ekaeEf", 0x7ffe47dc00b0, AT_SYMLINK_NOFOLLOW) = -
1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.ekaeEf", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC,
0666) = 4
write(4, "\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, 4, 0) = 0x7f829defb000
link("/dev/shm/sem.ekaeEf", "/dev/shm/sem.sem_read") = 0
fstat(4, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0
unlink("/dev/shm/sem.ekaeEf")           = 0
close(4)                                = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f829dce3a10) = 7256
futex(0x7f829defc000, FUTEX_WAKE, 1)    = 1
openat(AT_FDCWD, "output.txt", O_WRONLY|O_CREAT|O_TRUNC, 0644) = 4
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=7256, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
write(4, "Result: 10.00 / 4.00 / 2.00 = 1."..., 35) = 35
close(4)                                = 0
wait4(-1, NULL, 0, NULL)                = 7256
munmap(0x7f829defc000, 32)              = 0
munmap(0x7f829defb000, 32)              = 0
unlink("/dev/shm/sem.sem_write")        = 0
unlink("/dev/shm/sem.sem_read")         = 0
munmap(0x7f829defd000, 140)             = 0
unlink("/dev/shm/shared_mem")           = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

Во время выполнения данной лабораторной работы я узнал много полезного про системные вызовы и про создание семафоров с общей памятью