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-211Б-23

Студент: Амелина А.Е.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 02.01.25

Москва, 2024

Постановка задачи

Вариант 7.

Выполнить с использованием shared memory и memory mapping. 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». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип float. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int open(const char *__file, int __oflag, …); – используется для открытия файла для чтения, записи или и того, и другого.
- ssize_t write(int __fd, const void *__buf, size_t __n); – записывает N байт из буфер(BUF) в файл (FD). Возвращает количество записанных байт или -1.
- void exit(int __status); – выполняет немедленное завершение программы. Все используемые программой потоки закрываются, и временные файлы удаляются, управление возвращается ОС или другой программе.
- int close(int __fd); – сообщает операционной системе об окончании работы с файловым дескриптором, и закрывает файл(FD).
- ssize_t read(int __fd, void *__buf, size_t __nbytes); – считывает указанное количество байт из файла(FD) в буфер(BUF).
- pid_t waitpid(pid_t pid, int *status, int options) - ожидание завершения дочернего процесса.
- int execl(const char *path, const char *arg, ..., nullptr); - заменяет текущий образ процесса на новый образ.
- int shm_open(const char *name, int oflag, mode_t mode); – создает и открывает новый (или открывает уже существующий) объект разделяемой памяти POSIX.
- int shm_unlink(const char *name); – удаляется имя объекта разделяемой памяти и, как только все процессы завершили работу с объектом и отменили его распределение, очищают пространство и уничтожают связанную с ним область памяти.
- int ftruncate(int fd, off_t length); – устанавливают длину файла с файловым дескриптором fd в length байт.
- void * mmap(void *start, size_t length, int prot , int flags, int fd, off_t offset); – отражает length байтов, начиная со смещения offset файла (или другого объекта), определенного файловым дескриптором fd, в память, начиная с адреса start.
- int munmap(void *start, size_t length); – удаляет все отражения из заданной области памяти, после чего все ссылки на данную область будут вызывать ошибку "неправильное обращение к памяти".

- sem_t *sem_open(const char *name, int oflag); ИЛИ sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value); – создаёт новый семафор или открывает уже существующий.
- int sem_wait(sem_t *sem); – уменьшает значение семафора на 1. Если семафор в данный момент имеет нулевое значение, то вызов блокируется до тех пор, пока либо не станет возможным выполнить уменьшение.
- int sem_post(sem_t *sem); – увеличивает значение семафора на 1.
- int sem_unlink(const char *name); – удаляет имя семафора из системы. После вызова этой функции другие процессы больше не смогут открыть этот семафор по имени. int sem_close(sem_t *sem); – закрывает указанный семафор, освобождая ресурсы, связанные с ним.

Для выполнения данной лабораторной работы я изучила указанные выше системные вызовы.

Программа parent.cpp запрашивает у пользователя имя файла с помощью write и считывает с помощью read. Введенное имя сохраняется в массиве file. Создаётся область разделяемой памяти с помощью shm_open и устанавливается ее размер с помощью ftruncate. Затем программа отображает эту область в адресное пространство процесса с помощью mmap. Также создается два семафора: data_sem и processing_sem, которые используются для синхронизации между родительским и дочерним процессами.

Родительский процесс открывает файл для чтения с помощью open. В цикле он читает данные из файла и записывает их в разделяемую память. После записи данных родительский процесс сигнализирует дочернему процессу с помощью sem_post(data_sem) и ожидает ответа с помощью sem_wait(processing_sem).

Создается дочерний процесс с помощью fork(). Программа открывает область разделяемой памяти, созданную родительским процессом, с помощью shm_open и отображает ее в адресное пространство процесса с помощью mmap. В цикле программа ожидает сигнала от родительского процесса с помощью sem_wait(data_sem). После получения данных она разбивает их на строки с помощью strtok. Для каждой строки создается объект std::istringstream, который позволяет извлекать числа и вычислять их сумму. Результат записывается в буфер output с помощью snprintf и выводится на экран с помощью write.

Код программы

parent.cpp

```
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <cstdlib>
#include <cstring>
#include <cstdio>
#include <semaphore.h>
#include <sys/mman.h>

const int SHM_SIZE = 4096;
const char* SHM_NAME = "/shared_memory";
const char* DATA_SEM_NAME = "/data_semaphore";
const char* PROCESSING_SEM_NAME = "/processing_semaphore";

int main() {
    char file[256];
    ssize_t read_bytes;
    int shm_fd;
    char* shm_ptr;

    write(STDOUT_FILENO, "Enter the file name: ", 21);
    read_bytes = read(STDIN_FILENO, file, sizeof(file) - 1);

    if (read_bytes > 0) {
        file[read_bytes - 1] = '\0';
    } else {
        perror("Problems with the name of file");
        exit(EXIT_FAILURE);
    }
    shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);

    if (shm_fd == -1) {
        perror("Problems with shm_open");
        exit(EXIT_FAILURE);
    }
    if (ftruncate(shm_fd, SHM_SIZE) == -1) {
        perror("Problems with ftruncate");
        exit(EXIT_FAILURE);
    }

    shm_ptr = (char*)mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,
0);

    if (shm_ptr == MAP_FAILED) {
        perror("Problems with mmap");
        exit(EXIT_FAILURE);
    }

    sem_t* data_sem = sem_open(DATA_SEM_NAME, O_CREAT, 0666, 0);
    if (data_sem == SEM_FAILED) {
        perror("Problems with sem_open");
        exit(EXIT_FAILURE);
    }

    sem_t* processing_sem = sem_open(PROCESSING_SEM_NAME, O_CREAT, 0666, 0);
    if (processing_sem == SEM_FAILED) {
        perror("Problems with sem_open");
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();
    if (pid == -1) {
        perror("Error with fork\n");
```

```
        exit(EXIT_FAILURE);

    }
    else if (pid == 0) {
        execl("./child", "./child", nullptr);
        perror("Error with execl\n");
        exit(EXIT_FAILURE);
    }
    else {
        int file_fd = open(file, O_RDONLY);
        if (file_fd == -1) {
            perror("Problems with opening file");
            exit(EXIT_FAILURE);
        }

        while ((read_bytes = read(file_fd, shm_ptr, SHM_SIZE - 1)) > 0) {
            shm_ptr[read_bytes] = '\0';
            sem_post(data_sem);
            sem_wait(processing_sem);
        }
        shm_ptr[0] = '\0';
        sem_post(data_sem);
        close(file_fd);
        waitpid(pid, nullptr, 0);
        munmap(shm_ptr, SHM_SIZE);
        shm_unlink(SHM_NAME);
        sem_close(data_sem);
        sem_unlink(DATA_SEM_NAME);
        sem_close(processing_sem);
        sem_unlink(PROCESSING_SEM_NAME);
    }

    return 0;
}
```

child.cpp

```
#include <unistd.h>
#include <sstream>
#include <string>
#include <cstring>
#include <iostream>
#include <semaphore.h>
#include <sys/mman.h>
#include <fcntl.h>

const int SHM_SIZE = 4096;
const char* SHM_NAME = "/shared_memory";
const char* DATA_SEM_NAME = "/data_semaphore";
const char* PROCESSING_SEM_NAME = "/processing_semaphore";

int main() {
    int shm_fd;
    char* shm_ptr;
    ssize_t read_bytes;

    shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("Problems with shm_open");
        exit(EXIT_FAILURE);
    }

    shm_ptr = (char*)mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,
0);
    if (shm_ptr == MAP_FAILED) {
        perror("Problems with mmap");
```

```
        exit(EXIT_FAILURE);
    }

    sem_t* data_sem = sem_open(DATA_SEM_NAME, 0);
    if (data_sem == SEM_FAILED) {
        perror("Problems with sem_open");
        exit(EXIT_FAILURE);
    }

    sem_t* processing_sem = sem_open(PROCESSING_SEM_NAME, 0);
    if (processing_sem == SEM_FAILED) {
        perror("Problems with sem_open");
        exit(EXIT_FAILURE);
    }

    while (true) {
        sem_wait(data_sem);
        if (shm_ptr[0] == '\0') {
            break;
        }
        char* line = strtok(shm_ptr, "\n");

        while (line) {
            std::istringstream stream(line);
            float number, sum = 0;
            while (stream >> number) {
                sum += number;
            }
            char output[256];
            int output_len = snprintf(output, sizeof(output), "Sum of elements:
%.2f\n", sum);
            write(STDOUT_FILENO, output, output_len);

            line = strtok(nullptr, "\n");
        }
        sem_post(processing_sem);
    }
    munmap(shm_ptr, SHM_SIZE);
    sem_close(data_sem);
    sem_close(processing_sem);
    return 0;
}
```

Протокол работы программы

anegamelina@LAPTOP-0ED9K3JN:/mnt/c/Users/Anega/CLionProjects/osi_labs/lab3$ g++ -o parent parent.cpp

anegamelina@LAPTOP-0ED9K3JN:/mnt/c/Users/Anega/CLionProjects/osi_labs/lab3$ g++ -o child child.cpp

anegamelina@LAPTOP-0ED9K3JN:/mnt/c/Users/Anega/CLionProjects/osi_labs/lab3$ cat file.txt

3.14 8.52 -7.1

0.01 32.1 3.3 1.2

1.11 2.22 3.33

83.32 -23.2

48.3 99.2 1.1 22.39

1.2 3.4 5.6 7.8 9.1 -1.0

anegamelina@LAPTOP-0ED9K3JN:/mnt/c/Users/Anega/CLionProjects/osi_labs/lab3$ ./parent

Enter the file name: file.txt

Sum of elements: 4.56

Sum of elements: 36.61

Sum of elements: 6.66

Sum of elements: 60.12

Sum of elements: 170.99

Sum of elements: 26.10

anegamelina@LAPTOP-0ED9K3JN:/mnt/c/Users/Anega/CLionProjects/osi_labs/lab3$ strace ./parent

execve("./parent", ["./parent"], 0x7ffca8052180 /* 26 vars */) = 0

brk(NULL) = 0x55a9bdd43000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd288ed020) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f2c4bac4000

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=19779, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 19779, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f2c4babf000

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\0P\237\2\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"..., 48, 848) = 48

pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\302\211\332Pq\2439\235\350\223\322\257\201\326\243\f"..., 68, 896)
= 68

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, 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, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f2c4b896000

mprotect(0x7f2c4b8be000, 2023424, PROT_NONE) = 0

mmap(0x7f2c4b8be000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f2c4b8be000

mmap(0x7f2c4ba53000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1bd000) = 0x7f2c4ba53000

mmap(0x7f2c4baac000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f2c4baac000

mmap(0x7f2c4bab2000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f2c4bab2000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f2c4b893000

arch_prctl(ARCH_SET_FS, 0x7f2c4b893740) = 0

set_tid_address(0x7f2c4b893a10) = 123963

set_robust_list(0x7f2c4b893a20, 24) = 0

rseq(0x7f2c4b8940e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f2c4baac000, 16384, PROT_READ) = 0

mprotect(0x55a99746e000, 4096, PROT_READ) = 0

mprotect(0x7f2c4bafe000, 8192, PROT_READ) = 0

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

munmap(0x7f2c4babf000, 19779) = 0

write(1, "Enter the file name: ", 21Enter the file name:) = 21

read(0, file.txt

"file.txt\n", 255) = 9

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

ftruncate(3, 4096) = 0

mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f2c4bafd000

openat(AT_FDCWD, "/dev/shm/sem.data_semaphore", O_RDWR|O_NOFOLLOW) = -1
ENOENT (No such file or directory)

getrandom("\x28\xc7\x9a\x80\xed\xac\x05\xc2", 8, GRND_NONBLOCK) = 8

newfstatat(AT_FDCWD, "/dev/shm/sem.sfuajs", 0x7ffd288ecc30, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)

openat(AT_FDCWD, "/dev/shm/sem.sfuajs", O_RDWR|O_CREAT|O_EXCL, 0666) = 4

write(4, "\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\0", 32) = 32

mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f2c4bac3000

link("/dev/shm/sem.sfuajs", "/dev/shm/sem.data_semaphore") = 0

newfstatat(4, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0

getrandom("\x32\x75\x36\x54\xd7\x62\xee\xaf", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x55a9bdd43000

brk(0x55a9bdd64000) = 0x55a9bdd64000

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

close(4) = 0

openat(AT_FDCWD, "/dev/shm/sem.processing_semaphore", O_RDWR|O_NOFOLLOW) = -1
ENOENT (No such file or directory)

getrandom("\x9e\xb0\x0d\x37\x45\x1c\xb6\xbe", 8, GRND_NONBLOCK) = 8

newfstatat(AT_FDCWD, "/dev/shm/sem.yhFPuJ", 0x7ffd288ecc30, AT_SYMLINK_NOFOLLOW) = -
1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/dev/shm/sem.yhFPuJ", O_RDWR|O_CREAT|O_EXCL, 0666) = 4

write(4, "\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\0", 32) = 32

mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f2c4bac2000

link("/dev/shm/sem.yhFPuJ", "/dev/shm/sem.processing_semaphore") = 0

newfstatat(4, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0

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

close(4) = 0

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f2c4b893a10) = 123979

openat(AT_FDCWD, "file.txt", O_RDONLY) = 4

read(4, "3.14 8.52 -7.1\r\n0.01 32.1 3.3 1."..., 4095) = 109

futex(0x7f2c4bac2000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANYSum of elements: 4.56

Sum of elements: 36.61

Sum of elements: 6.66

Sum of elements: 60.12

Sum of elements: 170.99

Sum of elements: 26.10

) = 0

read(4, "", 4095) = 0

futex(0x7f2c4bac3000, FUTEX_WAKE, 1) = 1

close(4) = 0

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=123979, si_uid=1000, si_status=0, si_utime=0, si_stime=1} ---

wait4(123979, NULL, 0, NULL) = 123979

munmap(0x7f2c4bafd000, 4096) = 0

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

munmap(0x7f2c4bac3000, 32) = 0

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

munmap(0x7f2c4bac2000, 32) = 0

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

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В ходе написания данной лабораторной работы я научилась работать с новыми системными вызовами в СИ, которые используются для работы с семафорами и shared memory. Я также научилась передавать данные посредством shared memory и контролировать доступ через семафоры.