Лабораторная работа №4 по курсу "Операционные системы"

Студент группы: М80-207Б-21, Крючков Артемий Владимирович
Контакты: artemkr2003@mail.ru
Работа выполнена: 17.09.2022
Преподаватель: Миронов Евгений Сергеевич

Задание

Лабораторные работы №4

Цель работы

Приобретение практических навыков в:

- Освоение принципов работы с файловыми системами
- Обеспечение обмена данных между процессами посредством технологии «File mapping»

Задание

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.

Варианты задания

См. лабораторная работа №2.

Варианты выбираются такие же как и в лабораторной работе №2.

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

Методы и алгоритмы решения

```
#include "../includes/header.h"

int main(void)
```

```
{
    char *filename;
    size_t len = 0;

    cout << "Name of file: ";

    if (getline(&filename, &len, stdin) == -1) {
        _exit(EXIT_FAILURE);
    }

    size_t map_size = 0;
    char *in = (char *)malloc(sizeof(char));
    char c;

    while ((c = getchar()) != EOF) {
        in[map_size] = c;
        in = (char *)realloc(in, (++map_size + 1) * sizeof(char));
    }

    in[map_size++] = '\0';

    /*
    shm_open создает и открывает новый (или открывает уже существующий)
    объект разделяемой памяти POSIX. Объект разделяемой памяти
    POSIX - это обработчик, используемый несвязанными процессами для исполнения
mmap(2)
    на одну область разделяемой памяти. Функция shm_unlink выполняет обратную
операцию,
    удаляя объект, предварительно созданный с помощью shm_open.
    Операция shm_open аналогична open(2). name определяет собственно
создаваемый объект
    разделяемой памяти для создания или открытия. Для использования в
портируемых программах
    name должно иметь в начале косую черту (/) и больше не содержать их внутри
имени.
    */
    int fd = shm_open(CommonFile, O_RDWR | O_CREAT, mode);
    if (fd == -1) {
        _exit(EXIT_FAILURE);
    }

    sem_t* semptr = sem_open(SemaphoreName, O_CREAT, mode, 1); // 1 - начальное
значение для семафора
    if (semptr == SEM_FAILED) {
        _exit(EXIT_FAILURE);
    }

    ftruncate(fd, (off_t)map_size); // Функции truncate и ftruncate
устанавливают длину обычного файла с именем path или файловым дескриптором fd в
length байт.
    // Это тип данных, определенный в разделе sys/types.h заголовочный файл
(основного типа unsigned long) и используется для измерения смещения файла в
```

```
байтах от начала файла.
    char* memptr = (char*) mmap(NULL, map_size, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0); // создает новое сопоставление в виртуальном адресном
пространстве вызывающего процесса.
    if (memptr == MAP_FAILED) {
        _exit(EXIT_FAILURE);
    }

    sprintf(memptr, "%s", in);
    free(in);

    int val;
    if (sem_getvalue(semptr, &val) != 0) { // sem_getvalue(sem_t *sem, int
*sval); омещает текущее значение семафора, заданного в sem, в виде целого, на
которое указывает sval.
        _exit(EXIT_FAILURE);
    }

    /*
    Функция sem_post() увеличивает (разблокирует) семафор, на который указывает
sem.
    Если значение семафора после этого становится больше нуля, то другой
процесс или нить
    заблокированная в вызове sem_wait(3), проснётся и заблокирует семафор.
    */
    while (val++ < 1) {
        sem_post(semptr);
    }

    switch (auto pid = fork())
    {
    case -1:
        return 0;
    case 0:
        munmap(memptr, map_size);
        close(fd);
        sem_close(semptr);
        execl("child", "child", filename, NULL);
    }

    while (true) {
        if (sem_getvalue(semptr, &val) != 0) {
            _exit(EXIT_FAILURE);
        }

        if (val == 0) {
            /*
            Функция sem_wait() уменьшает (блокирует) семафор, на который
указывает sem.
            Если значение семафор больше нуля, то выполняется уменьшение и
функция сразу завершается.
            */
```

```
            if (sem_wait(semptr) == -1) {
                _exit(EXIT_FAILURE);
            }
            return 0;
        }
    }
}
```

```
#include "../includes/header.h"

int main(int argc, char** argv)
{
    int map_fd = shm_open(CommonFile, O_RDWR, mode);
    if (map_fd < 0) {
        _exit(EXIT_FAILURE);
    }

    struct stat statbuf;
    fstat(map_fd, &statbuf); // Эти функции возвращают информацию о файле в буфере. Вызов fstat() идентичен stat(), но опрашиваемый файл задаётся в виде файлового дескриптора fd.
    const size_t map_size = statbuf.st_size;

    char* memptr = (char*)mmap(NULL, map_size, PROT_READ | PROT_WRITE, MAP_SHARED, map_fd, 0);
    if (memptr == MAP_FAILED) {
        _exit(EXIT_FAILURE);
    }

    sem_t* semptr = sem_open(SemaphoreName, O_CREAT, mode, 1);
    if (semptr == SEM_FAILED) {
        _exit(EXIT_FAILURE);
    }

    /*
    Функция sem_wait() уменьшает (блокирует) семафор, на который указывает sem.
    Если значение семафор больше нуля, то выполняется уменьшение и функция сразу завершается.
    */

    if (sem_wait(semptr) != 0) {
        _exit(EXIT_FAILURE);
    }

    char* out = (char*)malloc(sizeof(char));
    size_t m_size = 0;
    string first;
    string second;

    FILE* filename = fopen(argv[1], "w");
    if (filename == NULL) {
        _exit(EXIT_FAILURE);
```

```
    }

    int flag = 0;

    for (int i = 0; i + 1 < map_size; ++i) {
        if (flag == 0) {
            first.push_back(memptr[i]);
        } else if (flag == 1) {
            second.push_back(memptr[i]);
        }

        if (memptr[i] == ' ' && flag == 0) {
            flag = 1;
        } else if ((memptr[i] == ' ' || memptr[i] == '\n') && flag == 1) {
            if (atof(second.c_str()) == 0) {
                cout << "ERROR!!!" << endl;
                break;
            }

            first = to_string(atof(first.c_str()) / atof(second.c_str()));
            second = "";

            if (memptr[i] == '\n') {
                fprintf(filename, "%s\n", first.c_str());
                flag = 0;
                first = "";
                second = "";
            }
        }
    }

    fclose(filename);
    out[m_size++] = '\0';
    ftruncate(map_fd, (off_t)m_size);
    memset(memptr, '\0', m_size);
    sprintf(memptr, "%s", out);
    free(out);
    close(map_fd);
    sem_post(semptr);
    sem_close(semptr);
    return 0;
}
```

Strace

```
[crewch@pc build]$ strace ./main
execve("./main", ["./main"], 0x7ffde831c3b0 /* 83 vars */) = 0
brk(NULL)                               = 0x56440b118000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffcc1784b50) = -1 EINVAL (Недопустимый
аргумент)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или
```

```
каталога)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=158163, ...}, AT_EMPTY_PATH) =
0
mmap(NULL, 158163, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fb0fe621000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/libstdc++.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"...,
832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=19198496, ...}, AT_EMPTY_PATH)
= 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fb0fe61f000
mmap(NULL, 2320384, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7fb0fe3e8000
mmap(0x7fb0fe481000, 1138688, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x99000) = 0x7fb0fe481000
mmap(0x7fb0fe597000, 487424, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1af000) = 0x7fb0fe597000
mmap(0x7fb0fe60e000, 57344, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x225000) = 0x7fb0fe60e000
mmap(0x7fb0fe61c000, 10240, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fb0fe61c000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/libm.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"...,
832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=944600, ...}, AT_EMPTY_PATH) =
0
mmap(NULL, 946368, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fb0fe300000
mmap(0x7fb0fe30e000, 499712, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe000) = 0x7fb0fe30e000
mmap(0x7fb0fe388000, 385024, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x88000) = 0x7fb0fe388000
mmap(0x7fb0fe3e6000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe5000) = 0x7fb0fe3e6000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/libgcc_s.so.1", 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\0\0\0\0\0\0\0\0\0"...,
832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=571848, ...}, AT_EMPTY_PATH) =
0
mmap(NULL, 127304, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fb0fe2e0000
mmap(0x7fb0fe2e3000, 94208, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x3000) = 0x7fb0fe2e3000
mmap(0x7fb0fe2fa000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1a000) = 0x7fb0fe2fa000
mmap(0x7fb0fe2fe000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1d000) = 0x7fb0fe2fe000
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\3\0>\0\1\0\0\0P4\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=1953472, ...}, 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, 1994384, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7fb0fe0f9000
mmap(0x7fb0fe11b000, 1421312, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7fb0fe11b000
mmap(0x7fb0fe276000, 356352, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x17d000) = 0x7fb0fe276000
mmap(0x7fb0fe2cd000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1d4000) = 0x7fb0fe2cd000
mmap(0x7fb0fe2d3000, 52880, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fb0fe2d3000
close(3)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fb0fe0f7000
arch_prctl(ARCH_SET_FS, 0x7fb0fe0f8200) = 0
set_tid_address(0x7fb0fe0f84d0)         = 8118
set_robust_list(0x7fb0fe0f84e0, 24)     = 0
rseq(0x7fb0fe0f8b20, 0x20, 0, 0x53053053) = 0
mprotect(0x7fb0fe2cd000, 16384, PROT_READ) = 0
mprotect(0x7fb0fe2fe000, 4096, PROT_READ) = 0
mprotect(0x7fb0fe3e6000, 4096, PROT_READ) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fb0fe0f5000
mprotect(0x7fb0fe60e000, 53248, PROT_READ) = 0
mprotect(0x564409122000, 4096, PROT_READ) = 0
mprotect(0x7fb0fe679000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7fb0fe621000, 158163)          = 0
getrandom("\xb5\x51\x42\x47\xbd\x4e\x7f\x57", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x56440b118000
brk(0x56440b139000)                     = 0x56440b139000
futex(0x7fb0fe61c6bc, FUTEX_WAKE_PRIVATE, 2147483647) = 0
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x2), ...},
AT_EMPTY_PATH) = 0
newfstatat(0, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x2), ...},
AT_EMPTY_PATH) = 0
write(1, "Name of file: ", 14Name of file: )          = 14
read(0, 1
"1\n", 1024)                    = 2
read(0, 1 2 3 4 5 6
"1 2 3 4 5 6\n", 1024)          = 12
read(0, 32 23 12312
"32 23 12312\n", 1024)          = 12
read(0, "", 1024)                       = 0
```

```
openat(AT_FDCWD, "/dev/shm/pipe", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0644) =
3
openat(AT_FDCWD, "/dev/shm/sem.semaphore", O_RDWR|O_NOFOLLOW) = 4
newfstatat(4, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7fb0fe647000
close(4)                                = 0
ftruncate(3, 25)                        = 0
mmap(NULL, 25, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7fb0fe646000
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fb0fe0f84d0) = 8187
futex(0x7fb0fe647000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY) = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```