Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу

«Операционные системы»

Группа:М8О-215Б-23

Студент: Венгер Ирина Витальевна

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: _____

Москва, 2024.

Содержание

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.

3 вариант) 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 int. Количество чисел может быть произвольным.

Общие сведения о программе

Также, как и в 1 лабораторной работе, нужно реализовать 2 процесса, родительский и дочерний, логика такая же, как и в первой лабораторной, пользователь вводит имя файла, в который он хочет произвести запись результата, а затем вводит 2 числа: делимое и делитель. Программа выводит в консоль результат деления внутри дочернего процесса, а также записывает результат в файл. Главное отличие от лабораторной работы номер 1 – реализация, здесь используется для передачи данных в родительский процесс не пайп, а память(shared memory), такая технология называется File mapping, т.е. с помощью семафоров оба процесса могут обращаться к одному участку памяти. Внутри самой программы определена структура того участка памяти, с которым мы работаем: command и filename, command постоянно поддаётся изменению со стороны обоих процессов, так как здесь передаётся не только команда, но и результат работы дочернего процесса. А осуществить контроль над разделяемой памятью, а точнее над временем считывания из неё помогут семафоры. Всего используется 2 файла: main.c – для родительского процесса – пользователь запускает именно его, child.c – реализация родительского процесса.

Общий метод и алгоритм решения

Сначала мы создаём структуру нашей, разделяемой памяти, в ней будет всего 2 поля, имя файла, с которым работает программа и команда, которая по совместительству будет ещё и возвращать результат процессу-родителю. Далее идёт функция main, в которой и происходит действие программы. В main.c и child.c внутри функций main начало похоже, мы обьявляем структуру shared memory, только если в main.c мы создаём этот раздел, то внутри child.c мы уже подключаемся к нему и ведём работу с ним. Функция ftruncate устанавливает размер выделенной памяти, который равен размеру структуры, которую мы обьявили в самом начале. Далее используем функцию mmap, которая отображает этот участок памяти в адресное пространство процессора. Далее с помощью функции sem_open() создаём семафоры, а затем уже используем знакомую функцию fork(), создавая дочерний процесс. Если вернулся 0 – то мы находимся внутри дочернего процесса. Далее происходит та же логика, что и в первой лабораторной работе, запускается дочерний процесс и начинается его работа. Далее идёт уже работа пользователя с родительским процессом. Происходит считывание имени файла, а затем и команды, которую ввёл пользователь. Затем происходит запись этой команды в разделённую память. Со стороны child.c происходит считывание имени файла и команды, далее происходит обработка команды, и запись результата в файл. В случае деления на 0 выбрасывается Zero Division сообщение. Благодаря функции sem_wait() мы можем наладить логику взаимодействия между процессами, чтобы они не опережали друг друга и не было такой ситуации, что ребёнок обращается к shared_memory, а там ничего нет. Функция sem_post() увеличивает счётчик семафора на 1 выше, что сигнализирует ребёнку о том, что тот может начать работу. В конце работы все семафоры закрываются, а функция munmap() освобождает занятую память.

Код программы

Код программы смотрите в приложении 1.

Использование утилиты strace

Скриншоты strace представлены в приложении 2. По результатам strace видно, какие системные вызовы использует ОС во время работы нашей программы: mmap – выделение памяти под какую-то структуру или переменную. ftruncate – устанавливает размер выделенной памяти. Clone – создала процесс и вернула его PID. Futex – примитив синхронизации, который используется для реализации семафоров, мьютексов, в этой программе мы используем семафоры, поэтому его появление вполне логично.

Демонстрация работы программы

(base) boopie@MacBook-Air-Irina src % ./main

Введите имя файла: file.txt

Введите числа, разделенные пробелом или 'выход' для завершения: 3 4 5

Результат деления: 0

Введите числа, разделенные пробелом или 'выход' для завершения: 0

Результат деления: 0

Введите числа, разделенные пробелом или 'выход' для завершения: 0 0

Результат деления: Ошибка деления на 0

Введите числа, разделенные пробелом или 'выход' для завершения: выход

Выход из child...

Как мы видим, программа работает корректно, а в файле file.txt произошла запись результата, по команде ‘выход’ дочерний процесс завершает свою работу, а затем и родительский процесс также завершает свою работу.

Вывод

В данной лабораторной работе я познакомился с ещё одним способом передачи сообщений и информации между процессами: file mapping. У этого способа есть свои преимущества по сравнению с pipe, например, сразу несколько процессов могут обращаться к одному адресу в памяти и взаимодействовать с ним, в пайпах пришлось бы знать количество процессов, чтобы посчитать количество каналов, с точки зрения написания кода это было бы не очень удобно. Недостатком file mapping по сравнению с pipe является тяжеловесность shared_memory, всё таки пайпы легковеснее и на более простых задачах, где количество процессов небольшое, в плане памяти использовать пайпы было бы эффективнее. Оба способа имеют свои плюсы и минусы, я рад, что у меня получилось познакомиться с ними и понять, как взаимодействовать между процессами.

Приложения

Приложение 1 – код программы:

Main.c

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

#define MAX_COMMAND_LEN 256
#define SHM_NAME "/shared_memory"
#define SEM_PARENT "/sem_parent"
#define SEM_CHILD "/sem_child"

typedef struct {
    char filename[MAX_COMMAND_LEN];
    char command[MAX_COMMAND_LEN];
} shared_data_t;

int main() {
    //создаём разделяемую память с правами доступа 0666, где 0 - отсутствие спец флагов, а 6 = 4 + 2, т.е. чтение плюс запись для владельца, группы и остальных
    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("Ошибка shm_open");
        exit(1);
    }
    //устанавливаем размер разделяемой памяти равный размеру созданной структуры
    ftruncate(shm_fd, sizeof(shared_data_t));
    //mmap отображает эту область памяти в адресное пространство процесса, shared_mem указывает на область памяти, которую видят и родительский и дочерний процессы
    shared_data_t *shared_mem = (shared_data_t *)mmap(0, sizeof(shared_data_t), PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (shared_mem == MAP_FAILED) {
        perror("Ошибка mmap");
        exit(1);
    }
    //создаём 2 семафора для уведомления дочернего процесса, что команда введенаб а второй семафор, что результат готов
    sem_t *sem_parent = sem_open(SEM_PARENT, O_CREAT, 0666, 0);
    sem_t *sem_child = sem_open(SEM_CHILD, O_CREAT, 0666, 0);
    //делаем fork и создаём дочерний процесс
    pid_t pid = fork();
    if (pid < 0) {
```

```
        perror("Ошибка fork");
        exit(1);
    }
    //находимся в дочернем
    if (pid == 0) {
        execl("./child", "child", NULL);
        perror("Ошибка execl");
        exit(1);
    } else {
        //внутри родительского
        printf("Введите имя файла: ");
        fgets(shared_mem->filename, sizeof(shared_mem->filename), stdin);
        shared_mem->filename[strcspn(shared_mem->filename, "\n")] = 0;

        char command[MAX_COMMAND_LEN];
        while (1) {
            printf("Введите числа через пробел или ‘выход’ для завершения: ");
            //считываем команду из stdin
            fgets(command, sizeof(command), stdin);
            command[strcspn(command, "\n")] = 0;
            //копируем команду в shared memory
            strcpy(shared_mem->command, command);
            //увеличиваем счётчик семафора и дочерний процесс может быть
выполнен
            sem_post(sem_parent);
            //проверка на exit
            if (strcmp(command, "выход") == 0) {
                break;
            }
            //ждём выполнение дочернего процесса,
            sem_wait(sem_child);
            printf("Результат деления: %s\n", shared_mem->command);
        }

        wait(NULL);
        //закрываем все семафоры и освобождаем ресурсы
        sem_close(sem_parent);
        sem_close(sem_child);
        sem_unlink(SEM_PARENT);
        sem_unlink(SEM_CHILD);
        munmap(shared_mem, sizeof(shared_data_t));
        shm_unlink(SHM_NAME);
        close(shm_fd);
    }

    return 0;
}
```

Child.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

```c
#include <fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>

#define MAX_COMMAND_LEN 256
//область разделяемой памяти
#define SHM_NAME "/shared_memory"
//семафоры для синхронизации между процессами
#define SEM_PARENT "/sem_parent"
#define SEM_CHILD "/sem_child"

typedef struct {
    char filename[MAX_COMMAND_LEN];
    char command[MAX_COMMAND_LEN];
} shared_data_t;

int main() {
    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("Ошибка shm_open");
        exit(1);
    }
    //устанавливаем размер разделяемой памяти равный размеру созданной структуры
    ftruncate(shm_fd, sizeof(shared_data_t));
    //mmap отображает эту область памяти в адресное пространство процесса,
shared_mem указывает на область памяти, которую видят и родительский и дочерний
процессы
    shared_data_t *shared_mem = (shared_data_t *)mmap(0, sizeof(shared_data_t),
PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (shared_mem == MAP_FAILED) {
        perror("Ошибка mmap");
        exit(1);
    }
    //создаём 2 семафора для уведомления дочернего процесса, что команда
введенаб а второй семафор, что результат готов
    sem_t *sem_parent = sem_open(SEM_PARENT, 0);
    sem_t *sem_child = sem_open(SEM_CHILD, 0);

    while (1) {
        sem_wait(sem_parent);

        if (strcmp(shared_mem->command, "выход") == 0) {
            printf("Выход из child\n");
            break;
        }
        //открываем файл, имя которого было передано через shared mem
        FILE *file = fopen(shared_mem->filename, "a");
        if (file == NULL) {
            perror("Ошибка открытия файла");
            exit(1);
        }

        int num1, num2, result;
        char *token = strtok(shared_mem->command, " ");
```

```
        num1 = atoi(token);
        result = num1;

        int division_by_zero = 0;
        while ((token = strtok(NULL, " ")) != NULL) {
            num2 = atoi(token);
            if (num2 == 0) {
                strcpy(shared_mem->command, "Ошибка деления на 0");
                fprintf(file, "Ошибка деления на 0\n");
                division_by_zero = 1;
                break;
            }
            result /= num2;
        }

        if (!division_by_zero) {
            sprintf(shared_mem->command, "%d", result);
            fprintf(file, "Результат деления: %d\n", result);
        }

        fflush(file);
        fclose(file);
        //говорим родительскому процессу, что ребёнок завершил работу
        sem_post(sem_child);
    }

    munmap(shared_mem, sizeof(shared_data_t));
    close(shm_fd);
    sem_close(sem_parent);
    sem_close(sem_child);
    return 0;
}
```

Приложение 2 – strace:

```
root@77ea7f5394a0:/src# strace ./main
execve("./main", ["./main"], 0xffffe969e980 /* 9 vars */) = 0
brk(NULL)                               = 0xaaaaedfcc000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff9616a000
faccessat(AT_FDCWD, "/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=8467, ...}) = 0
mmap(NULL, 8467, PROT_READ, MAP_PRIVATE, 3, 0) = 0xffff96167000
close(3)                                = 0
openat(AT_FDCWD, "/lib/aarch64-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\267\0\1\0\0\0\360\206\2\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1722920, ...}) = 0
mmap(NULL, 1892240, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_DENYWRITE, -1, 0) = 0xffff95f64000
mmap(0xffff95f70000, 1826704, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0) = 0xffff95f70000
munmap(0xffff95f64000, 49152)           = 0
munmap(0xffff9612e000, 16272)           = 0
mprotect(0xffff9610a000, 77824, PROT_NONE) = 0
mmap(0xffff9611d000, 20480, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19d000) = 0xffff9611d000
mmap(0xffff96122000, 49040, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xffff96122000
close(3)                                = 0
set_tid_address(0xffff9616afb0)         = 1187
set_robust_list(0xffff9616afc0, 24)     = 0
rseq(0xffff9616b600, 0x20, 0, 0xd428bc00) = 0
mprotect(0xffff9611d000, 12288, PROT_READ) = 0
mprotect(0xaaaab2cef000, 4096, PROT_READ) = 0
mprotect(0xffff96170000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0xffff96167000, 8467)            = 0
openat(AT_FDCWD, "/dev/shm/shared_memory", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3
ftruncate(3, 512)                       = 0
mmap(NULL, 512, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0xffff96169000
openat(AT_FDCWD, "/dev/shm/sem.sem_parent", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file or directory)
getrandom("\x8d\xe6\xcc\xc8\x7f\x62\x44\xe7", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.jQC1bM", 0xffffc838da98, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.jQC1bM", 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", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0xffff96168000
linkat(AT_FDCWD, "/dev/shm/sem.jQC1bM", AT_FDCWD, "/dev/shm/sem.sem_parent", 0) = 0
fstat(4, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0
getrandom("\x8c\xec\x7d\xf3\xa6\x3d\xc4\xf3", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0xaaaaedfcc000
brk(0xaaaaedfed000)                     = 0xaaaaedfed000
unlinkat(AT_FDCWD, "/dev/shm/sem.jQC1bM", 0) = 0
close(4)                                = 0
openat(AT_FDCWD, "/dev/shm/sem.sem_child", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file or directory)
getrandom("\x40\x90\x44\xea\x9a\x8e\xa8\x76", 8, GRND_NONBLOCK) = 8
```

```
newfstatat(AT_FDCWD, "/dev/shm/sem.4o5EZi", 0xffffc838da98, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.4o5EZi", 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) = 0xffff96167000
linkat(AT_FDCWD, "/dev/shm/sem.4o5EZi", AT_FDCWD, "/dev/shm/sem.sem_child", 0) = 0
fstat(4, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0
unlinkat(AT_FDCWD, "/dev/shm/sem.4o5EZi", 0) = 0
close(4)                                = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0xffff9616afb0) = 1188
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\270\320\274\321\217 \321\204\320\260\320\271\320\273\320\260"..., 34Введите имя файла: ) = 34
read(0, file.txt
"file.txt\n", 1024)             = 9
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\207\320\270\321\201\320\273\320\260, \321\200\320\260\320"..., 116Введите числа, разделенные пробелом или 'выход' для завершения: ) = 116
read(0, 1 6 7
"1 6 7\n", 1024)                = 6
futex(0xffff96168000, FUTEX_WAKE, 1)    = 1
futex(0xffff96167000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY) = 0
write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202 \320\264\320\265\320\273\320\265\320\275\320\270\321"..., 37Результат деления: 0
) = 37
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\207\320\270\321\201\320\273\320\260, \321\200\320\260\320"..., 116Введите числа, разделенные пробелом или 'выход' для завершения: ) = 116
read(0, 9 8 0
"9 8 0\n", 1024)                = 6
futex(0xffff96168000, FUTEX_WAKE, 1)    = 1
futex(0xffff96167000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY) = 0
write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202 \320\264\320\265\320\273\320\265\320\275\320\270\321"..., 70Результат деления: Ошибка деления на 0
) = 70
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\207\320\270\321\201\320\273\320\260, \321\200\320\260\320"..., 116Введите числа, разделенные пробелом или 'выход' для завершения: ) = 116
read(0, выход
"\320\262\321\213\321\205\320\276\320\264\n", 1024) = 11
futex(0xffff96168000, FUTEX_WAKE, 1)    = 1
Выход из child...
wait4(-1, NULL, 0, NULL)                = 1188
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=1188, si_uid=0, si_status=0, si_utime=0, si_stime=0} ---
munmap(0xffff96168000, 32)              = 0
munmap(0xffff96167000, 32)              = 0
unlinkat(AT_FDCWD, "/dev/shm/sem.sem_parent", 0) = 0
unlinkat(AT_FDCWD, "/dev/shm/sem.sem_child", 0) = 0
munmap(0xffff96169000, 512)             = 0
```

```
unlinkat(AT_FDCWD, "/dev/shm/shared_memory", 0) = 0
close(3)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
root@77ea7f5394a0:/src#
```