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-210Б-23

Студент: Дворников М.Д.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 18.12.24

Москва, 2024

Постановка задачи

Вариант 8.

В файле записаны команды вида:«число число число<endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. 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.

Общий метод и алгоритм решения

Использованные системные вызовы:

- **pid_t fork(void)** — создаёт новый процесс (дочерний).
- **int pipe(int *fd)** — создаёт канал связи (pipe) и помещает дескрипторы для чтения и записи в fd[0] и fd[1].
- **int write(int fd, const void* buffer, size_t count)** — записывает данные из buffer в ресурс, связанный с файловым дескриптором fd.
- **int read(int fd, void* buffer, size_t count)** — читает данные из ресурса, связанного с файловым дескриптором fd.
- **int dup2(int oldfd, int newfd)** — дублирует файловый дескриптор oldfd в newfd, перенаправляя потоки ввода/вывода.
- **int execl(const char *path, const char *arg, ...)** — заменяет текущий процесс новой программой, указанной в path.
- **int open(const char *pathname, int flags, mode_t mode)** — открывает файл, возвращая его дескриптор.
- **int close(int fd)** — закрывает файловый дескриптор fd.
- **pid_t wait(int *status)** — заставляет родительский процесс ожидать завершения дочернего.
- **void exit(int status)** — завершает текущий процесс с кодом status.
- **int shm_open(const char *name, int oflag, mode_t mode)** — создаёт или открывает разделяемую память, возвращая файловый дескриптор, с помощью которого осуществляется доступ к памяти.
- **int ftruncate(int fd, off_t length)** — изменяет размер разделяемой памяти. Используется для выделения достаточного объёма памяти для хранения данных.
- **void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset)** — отображает разделяемую память в адресное пространство процесса, возвращая указатель на начало области памяти.
- **int munmap(void *addr, size_t length)** — удаляет отображение разделяемой памяти из адресного пространства процесса, освобождая ресурсы.
- **int shm_unlink(const char *name)** — удаляет разделяемую память после завершения всех операций.

Программа parent.c запрашивает у пользователя имя файла, содержащего команды, создаёт разделяемую память (shared memory) для взаимодействия с дочерним процессом и использует системный вызов fork() для его создания. Родительский процесс записывает имя файла в shared memory и уведомляет дочерний процесс о готовности данных с помощью семафора. Дочерний процесс, выполняющий программу child.c, считывает имя файла из разделяемой памяти, открывает его и обрабатывает строки чисел. В каждой строке первое число используется как делимое, а

последующие — как делители. Результаты деления формируются дочерним процессом и записываются обратно в shared memory, откуда их впоследствии считывает родительский процесс. В случае ошибок, таких как деление на 0 или некорректный ввод данных, дочерний процесс завершает свою работу и уведомляет об этом родительский процесс через семафор. Родительский процесс с помощью wait() ожидает завершения дочернего процесса, а затем выводит результаты вычислений на экран, читая их из shared memory.

Код программы

parent.c

```c
#define SHM_NAME "/shared_memory"
#define SEM_NAME "/sync_semaphore"
#define BUFFER_SIZE 1024
#define NUM_LINES 100

void error_handler(const char *msg) {
    write( fd: STDERR_FILENO,  buf: msg,  nbyte: strlen( s: msg));
    write( fd: STDERR_FILENO,  buf: "\n",  nbyte: 1);
    exit(EXIT_FAILURE);
}

int main() {
    int shm_fd;
    char *shared_mem;
    sem_t *semaphore;
    ssize_t bytesRead;

    shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        error_handler( msg: "Ошибка создания разделяемой памяти");
    }

    if (ftruncate(shm_fd, BUFFER_SIZE * NUM_LINES) == -1) {
        error_handler( msg: "Ошибка изменения размера разделяемой памяти");
    }

    shared_mem = mmap(NULL, BUFFER_SIZE * NUM_LINES, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (shared_mem == MAP_FAILED) {
        error_handler( msg: "Ошибка отображения разделяемой памяти");
    }

    semaphore = sem_open(SEM_NAME, O_CREAT, 0666, 0);
    if (semaphore == SEM_FAILED) {
        error_handler( msg: "Ошибка создания семафора");
    }

    char filename[BUFFER_SIZE];
    const char *prompt = "Введите имя файла: ";
    write( fd: STDOUT_FILENO,  buf: prompt,  nbyte: strlen( s: prompt));
    bytesRead = read(STDIN_FILENO, filename, sizeof(filename));
    if (bytesRead <= 0) {
        error_handler( msg: "Ошибка чтения имени файла");
    }

    size_t len = strlen( s: filename);
    if (len > 0 && filename[len - 1] == '\n') {
        filename[len - 1] = '\0';
    }

    strncpy(shared_mem, filename, BUFFER_SIZE);

    pid_t child_pid = fork();
    if (child_pid == -1) {
        error_handler( msg: "Ошибка создания дочернего процесса");
    }

    if (child_pid == 0) {
        execl( path: "./child",  arg0: "./child", NULL);
        error_handler( msg: "Ошибка выполнения дочернего процесса");
    } else {
        sem_post(semaphore);

        wait(NULL);

        for (int i = 0; i < NUM_LINES; i++) {
            if (strlen( s: shared_mem + i * BUFFER_SIZE) > 0) {
                write(
                        fd: STDOUT_FILENO,  buf: shared_mem + i * BUFFER_SIZE,  nbyte: strlen(
                                s: shared_mem + i * BUFFER_SIZE
                            )
                    );
            }
        }

        munmap(shared_mem, BUFFER_SIZE * NUM_LINES);
        shm_unlink(SHM_NAME);
        sem_close(semaphore);
        sem_unlink(SEM_NAME);

        exit(EXIT_SUCCESS);
    }
}
```

child.c

```c
#define SHM_NAME "/shared_memory"
#define SEM_NAME "/sync_semaphore"
#define BUFFER_SIZE 1024
#define NUM_LINES 100

void HandleError(const char *message) {
    write( fd: STDERR_FILENO,  buf: message,  nbyte: strlen( s: message));
    exit(EXIT_FAILURE);
}

int safe_strtol(const char *str, char **endptr) {
    long value = strtol(str, endptr,  base: 10);
    if (value > INT_MAX || value < INT_MIN) {
        HandleError( message: "Ошибка: значение выходит за пределы диапазона int.\n");
    }
    return (int)value;
}

int main() {
    int shm_fd;
    char *shared_mem;
    sem_t *semaphore;

    shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shm_fd == -1) {
        HandleError( message: "Ошибка подключения к разделяемой памяти.\n");
    }

    shared_mem = mmap(NULL, BUFFER_SIZE * NUM_LINES, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (shared_mem == MAP_FAILED) {
        HandleError( message: "Ошибка отображения разделяемой памяти.\n");
    }

    semaphore = sem_open(SEM_NAME, 0);
    if (semaphore == SEM_FAILED) {
        HandleError( message: "Ошибка подключения к семафору.\n");
    }

    sem_wait(semaphore);

    char filename[BUFFER_SIZE];
    strncpy(filename, shared_mem, BUFFER_SIZE);

```


```c
#define SHM_NAME "/shared_memory"
#define SEM_NAME "/sync_semaphore"
#define BUFFER_SIZE 1024
#define NUM_LINES 100

void HandleError(const char *message) {
    write( fd: STDERR_FILENO,  buf: message,  nbyte: strlen( s: message));
    exit(EXIT_FAILURE);
}

int safe_strtol(const char *str, char **endptr) {
    long value = strtol(str, endptr,  base: 10);
    if (value > INT_MAX || value < INT_MIN) {
        HandleError( message: "Ошибка: значение выходит за пределы диапазона int.\n");
    }
    return (int)value;
}

int main() {
    int shm_fd;
    char *shared_mem;
    sem_t *semaphore;

    shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shm_fd == -1) {
        HandleError( message: "Ошибка подключения к разделяемой памяти.\n");
    }

    shared_mem = mmap(NULL, BUFFER_SIZE * NUM_LINES, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (shared_mem == MAP_FAILED) {
        HandleError( message: "Ошибка отображения разделяемой памяти.\n");
    }

    semaphore = sem_open(SEM_NAME, 0);
    if (semaphore == SEM_FAILED) {
        HandleError( message: "Ошибка подключения к семафору.\n");
    }

    sem_wait(semaphore);

    char filename[BUFFER_SIZE];
    strncpy(filename, shared_mem, BUFFER_SIZE);

```

```
    int file = open(filename, O_RDONLY);
    if (file == -1) {
        strncpy(shared_mem, "Ошибка: Не удалось открыть файл.\n", BUFFER_SIZE);
        sem_post(semaphore);
        exit(EXIT_FAILURE);
    }

    char buffer[BUFFER_SIZE];
    ssize_t bytesRead;
    int first_number, next_number;
    char *current;
    int line_number = 0;

    while ((bytesRead = read(file, buffer, BUFFER_SIZE)) > 0) {
        current = buffer;

        while (current < buffer + bytesRead) {
            while (*current == ' ' || *current == '\t') current++;
            if (*current == '\n') {
                current++;
                continue;
            }

            char *endptr;
            first_number = safe_strtol( str: current, &endptr);
            current = endptr;

            char result[BUFFER_SIZE];
            int result_len = snprintf(result, BUFFER_SIZE, "Результат: %d", first_number);

            while (current < buffer + bytesRead && *current != '\n') {
                while (*current == ' ' || *current == '\t') current++;
                if (*current == '\n') break;

                next_number = safe_strtol( str: current, &endptr);
                if (next_number == 0) {
                    strncpy(shared_mem + line_number * BUFFER_SIZE, "Ошибка: Деление на ноль.\n", BUFFER_SIZE);
                    sem_post(semaphore);
                    exit(EXIT_FAILURE);
                }

                result_len += snprintf(result + result_len, BUFFER_SIZE - result_len, ", %d / %d = %d",
                                       first_number, next_number, first_number / next_number);
                current = endptr;
            }

            result[result_len++] = '\n';
            strncpy(shared_mem + line_number * BUFFER_SIZE, result, result_len);
            line_number++;

            if (line_number >= NUM_LINES) break;
        }
    }

    if (bytesRead == -1) {
        strncpy(shared_mem, "Ошибка чтения файла.\n", BUFFER_SIZE);
        sem_post(semaphore);
        exit(EXIT_FAILURE);
    }

    close(file);
    sem_post(semaphore);
    exit(EXIT_SUCCESS);
}
```

Протокол работы программы

Тестирование:

```
❯ ./parent
Введите имя файла: input.txt
Результат: 12, 12 / 3 = 4, 12 / 4 = 3, 12 / 6 = 2, 12 / 12 = 1
Результат: 20, 20 / 2 = 10, 20 / 2 = 10, 20 / 5 = 4, 20 / 10 = 2, 20 / 4 = 5
Результат: 10, 10 / 5 = 2, 10 / 2 = 5, 10 / 1 = 10, 10 / 10 = 1
Результат: 52, 52 / 2 = 26, 52 / 4 = 13
Ошибка: Деление на ноль.
```

child.c | input.txt | lab3/.../parent.c

```
1  12 3  4 6    12
2  20 2   2 5            10 4
3  10 5        2 1 10
4  52 2  4
5  50      0
6
```

dtrace:

```
__mac_syscall(0x18DA23D62, 0x2, 0x16B008200)            = 0 0
map_with_linking_np(0x16B008090, 0x1, 0x16B0080C0)              = 0 0
close(0x3)              = 0 0
mprotect(0x104DE8000, 0x4000, 0x1)              = 0 0
open("/dev/dtracehelper\0", 0x2, 0x0)           = 3 0
ioctl(0x3, 0x80086804, 0x16B007588)             = 0 0
close(0x3)              = 0 0
shared_region_check_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0)            = 0 0
access("/AppleInternal/XBS/.isChrooted\0", 0x0, 0x0)            = -1 Err#2
bsdthread_register(0x18DD260F4, 0x18DD260E8, 0x4000)            = 1073746399 0
getpid(0x0, 0x0, 0x0)           = 65245 0
shm_open(0x18DBBDF41, 0x0, 0xFFFFFFFF8DD64000)          = 3 0
fstat64(0x3, 0x16B007C00, 0x0)          = 0 0
mmap(0x0, 0x8000, 0x1, 0x40001, 0x3, 0x0)               = 0x104DF8000 0
close(0x3)              = 0 0
csops(0xFEDD, 0x0, 0x16B007D3C)         = 0 0
ioctl(0x2, 0x4004667A, 0x16B007CAC)             = 0 0
mprotect(0x104E08000, 0x4000, 0x0)              = 0 0
mprotect(0x104E14000, 0x4000, 0x0)              = 0 0
mprotect(0x104E18000, 0x4000, 0x0)              = 0 0
mprotect(0x104E24000, 0x4000, 0x0)              = 0 0
mprotect(0x104E28000, 0x4000, 0x0)              = 0 0
mprotect(0x104E34000, 0x4000, 0x0)              = 0 0
mprotect(0x104E00000, 0xC8, 0x1)                = 0 0
mprotect(0x104E00000, 0xC8, 0x3)                = 0 0
mprotect(0x104E00000, 0xC8, 0x1)                = 0 0
mprotect(0x104E38000, 0x4000, 0x1)              = 0 0
mprotect(0x104E3C000, 0xC8, 0x1)                = 0 0
mprotect(0x104E3C000, 0xC8, 0x3)                = 0 0
mprotect(0x104E3C000, 0xC8, 0x1)                = 0 0
mprotect(0x104E00000, 0xC8, 0x3)                = 0 0
mprotect(0x104E00000, 0xC8, 0x1)                = 0 0
mprotect(0x104E38000, 0x4000, 0x3)              = 0 0
mprotect(0x104E38000, 0x4000, 0x1)              = 0 0
issetugid(0x0, 0x0, 0x0)                = 0 0
getentropy(0x16B007318, 0x20, 0x0)              = 0 0
getattrlist("/Users/matveyd/CLionProjects/OS-labs-git/lab3/src/parent\0", 0x16B007BA0, 0x16B007BBC)             = 0 0
access("/Users/matveyd/CLionProjects/OS-labs-git/lab3/src\0", 0x4, 0x0)         = 0 0
open("/Users/matveyd/CLionProjects/OS-labs-git/lab3/src\0", 0x0, 0x0)           = 3 0
fstat64(0x3, 0x14C604470, 0x0)          = 0 0
csrctl(0x0, 0x16B007D8C, 0x4)           = 0 0
fcntl(0x3, 0x32, 0x16B007A88)           = 0 0
```

```
close(0x3)              = 0 0
open("/Users/matveyd/CLionProjects/OS-labs-git/lab3/src/Info.plist\0", 0x0, 0x0)               = -1 Err#2
proc_info(0x2, 0xFEDD, 0xD)             = 64 0
csops_audittoken(0xFEDD, 0x10, 0x16B007E10)             = 0 0
sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16B008168, 0x16B008160, 0x191437D3A, 0x15)               = 0 0
sysctl([CTL_KERN, 155, 0, 0, 0, 0] (2), 0x16B0081F8, 0x16B0081F0, 0x0, 0x0)             = 0 0
shm_open(0x104DE7D92, 0x202, 0x1B6)             = 3 0
ftruncate(0x3, 0x19000, 0x0)            = 0 0
mmap(0x0, 0x19000, 0x3, 0x40001, 0x3, 0x0)              = 0x104E44000 0
sem_open(0x104DE7E7E, 0x200, 0x1B6)             = 4 0
write(0x1, "\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: \0", 0x22)             = 34 0
input.txt
Результат: 12, 12 / 3 = 4, 12 / 4 = 3, 12 / 6 = 2, 12 / 12 = 1
Результат: 20, 20 / 2 = 10, 20 / 2 = 10, 20 / 5 = 4, 20 / 10 = 2, 20 / 4 = 5
Результат: 10, 10 / 5 = 2, 10 / 2 = 5, 10 / 1 = 10, 10 / 10 = 1
Результат: 52, 52 / 2 = 26, 52 / 4 = 13
Ошибка: Деление на ноль.
read(0x0, "input.txt\n\0", 0x400)               = 10 0
fork()          = 65248 0
sem_post(0x4, 0x0, 0x0)         = 0 0
wait4(0xFFFFFFFFFFFFFFFF, 0x0, 0x0)             = 65248 0
write(0x1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202: 12, 12 / 3 = 4, 12 / 4 = 3, 12 / 6 = 2, 12 / 12 = 1\n\0", 0x48)           = 72 0
write(0x1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202: 20, 20 / 2 = 10, 20 / 2 = 10, 20 / 5 = 4, 20 / 10 = 2, 20 / 4 = 5\n\0", 0x56)            = 86 0
write(0x1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202: 10, 10 / 5 = 2, 10 / 2 = 5, 10 / 1 = 10, 10 / 10 = 1\n\0", 0x49)          = 73 0
write(0x1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202: 52, 52 / 2 = 26, 52 / 4 = 13\n\0", 0x31)          = 49 0
write(0x1, "\320\236\321\210\320\270\320\261\320\272\320\260: \320\224\320\265\320\273\320\265\320\275\320\270\320\265 \320\275\320\260 \320\275\320\276\320\273\321\214.\n\0", 0x2C)          = 44 0
munmap(0x104E44000, 0x19000)            = 0 0
shm_unlink(0x104DE7D92, 0x0, 0x0)               = 0 0
sem_close(0x4, 0x0, 0x0)                = 0 0
sem_unlink(0x104DE7E7E, 0x0, 0x0)               = 0 0
```

Вывод:

В процессе выполнения данной лабораторной работы я изучил новые системные вызовы на языке Си, которые позволяют эффективно работать с разделяемой памятью и семафорами. Освоил передачу данных между процессами через shared memory и управление доступом с использованием семафоров. Все задачи успешно выполнены, явных сложностей не наблюдалось.