

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<newline>». 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

```
9  #define SHM_NAME "/shared_memory"
10 #define SEM_NAME "/sync_semaphore"
11 #define BUFFER_SIZE 1024
12 #define NUM_LINES 100
13
14 void error_handler(const char *msg) {
15     write(fd: STDERR_FILENO, buf: msg, nbytes: strlen(s: msg));
16     write(fd: STDERR_FILENO, buf: "\n", nbytes: 1);
17     exit(EXIT_FAILURE);
18 }
19
20 int main() {
21     int shm_fd;
22     char *shared_mem;
23     sem_t *semaphore;
24     ssize_t bytesRead;
25
26     shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
27     if (shm_fd == -1) {
28         error_handler(msg: "Ошибка создания разделяемой памяти");
29     }
30
31     if (ftruncate(shm_fd, BUFFER_SIZE * NUM_LINES) == -1) {
32         error_handler(msg: "Ошибка изменения размера разделяемой памяти");
33     }
34
35     shared_mem = mmap(NULL, BUFFER_SIZE * NUM_LINES, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
36     if (shared_mem == MAP_FAILED) {
37         error_handler(msg: "Ошибка отображения разделяемой памяти");
38     }
39
40     semaphore = sem_open(SEM_NAME, O_CREAT, 0666, 0);
41     if (semaphore == SEM_FAILED) {
42         error_handler(msg: "Ошибка создания семафора");
43     }
44
45     char filename[BUFFER_SIZE];
46     const char *prompt = "Введите имя файла: ";
47     write(fd: STDOUT_FILENO, buf: prompt, nbytes: strlen(s: prompt));
48     bytesRead = read(STDIN_FILENO, filename, sizeof(filename));
49     if (bytesRead <= 0) {
50         error_handler(msg: "Ошибка чтения имени файла");
51     }
52
53     size_t len = strlen(s: filename);
54     if (len > 0 && filename[len - 1] == '\n') {
55         filename[len - 1] = '\0';
56     }
57
58     strncpy(shared_mem, filename, BUFFER_SIZE);
59
60     pid_t child_pid = fork();
61     if (child_pid == -1) {
62         error_handler(msg: "Ошибка создания дочернего процесса");
63     }
64
65     if (child_pid == 0) {
66         execl(path: "./child", arg0: "./child", NULL);
67         error_handler(msg: "Ошибка выполнения дочернего процесса");
68     } else {
69         sem_post(semaphore);
70
71         wait(NULL);
72
73         for (int i = 0; i < NUM_LINES; i++) {
74             if (strlen(s: shared_mem + i * BUFFER_SIZE) > 0) {
75                 write(
76                     fd: STDOUT_FILENO, buf: shared_mem + i * BUFFER_SIZE, nbytes: strlen(
77                         s: shared_mem + i * BUFFER_SIZE
78                     )
79                 );
80             }
81         }
82
83         munmap(shared_mem, BUFFER_SIZE * NUM_LINES);
84         shm_unlink(SHM_NAME);
85         sem_close(semaphore);
86         sem_unlink(SEM_NAME);
87
88         exit(EXIT_SUCCESS);
89     }
90 }
91
```

child.c

```
10 #define SHM_NAME "/shared_memory"
11 #define SEM_NAME "/sync_semaphore"
12 #define BUFFER_SIZE 1024
13 #define NUM_LINES 100
14
15 void HandleError(const char *message) {
16     write( fd: STDERR_FILENO, buf: message, nbytes: strlen( s: message));
17     exit(EXIT_FAILURE);
18 }
19
20 int safe_strtol(const char *str, char **endptr) {
21     long value = strtol(str, endptr, base: 10);
22     if (value > INT_MAX || value < INT_MIN) {
23         HandleError( message: "Ошибка: значение выходит за пределы диапазона int.\n");
24     }
25     return (int)value;
26 }
27
28 int main() {
29     int shm_fd;
30     char *shared_mem;
31     sem_t *semaphore;
32
33     shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
34     if (shm_fd == -1) {
35         HandleError( message: "Ошибка подключения к разделяемой памяти.\n");
36     }
37
38     shared_mem = mmap(NULL, BUFFER_SIZE * NUM_LINES, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
39     if (shared_mem == MAP_FAILED) {
40         HandleError( message: "Ошибка отображения разделяемой памяти.\n");
41     }
42
43     semaphore = sem_open(SEM_NAME, 0);
44     if (semaphore == SEM_FAILED) {
45         HandleError( message: "Ошибка подключения к семафору.\n");
46     }
47
48     sem_wait(semaphore);
49
50     char filename[BUFFER_SIZE];
51     strncpy(filename, shared_mem, BUFFER_SIZE);
52
53     #define SHM_NAME "/shared_memory"
54     #define SEM_NAME "/sync_semaphore"
55     #define BUFFER_SIZE 1024
56     #define NUM_LINES 100
57
58     void HandleError(const char *message) {
59         write( fd: STDERR_FILENO, buf: message, nbytes: strlen( s: message));
60         exit(EXIT_FAILURE);
61     }
62
63     int safe_strtol(const char *str, char **endptr) {
64         long value = strtol(str, endptr, base: 10);
65         if (value > INT_MAX || value < INT_MIN) {
66             HandleError( message: "Ошибка: значение выходит за пределы диапазона int.\n");
67         }
68         return (int)value;
69     }
70
71     int main() {
72         int shm_fd;
73         char *shared_mem;
74         sem_t *semaphore;
75
76         shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
77         if (shm_fd == -1) {
78             HandleError( message: "Ошибка подключения к разделяемой памяти.\n");
79         }
80
81         shared_mem = mmap(NULL, BUFFER_SIZE * NUM_LINES, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
82         if (shared_mem == MAP_FAILED) {
83             HandleError( message: "Ошибка отображения разделяемой памяти.\n");
84         }
85
86         semaphore = sem_open(SEM_NAME, 0);
87         if (semaphore == SEM_FAILED) {
88             HandleError( message: "Ошибка подключения к семафору.\n");
89         }
90
91         sem_wait(semaphore);
92
93         char filename[BUFFER_SIZE];
94         strncpy(filename, shared_mem, BUFFER_SIZE);
95     }
96 }
```

```

53     int file = open(filename, O_RDONLY);
54     if (file == -1) {
55         strncpy(shared_mem, "Ошибка: Не удалось открыть файл.\n", BUFFER_SIZE);
56         sem_post(semaphore);
57         exit(EXIT_FAILURE);
58     }
59
60     char buffer[BUFFER_SIZE];
61     ssize_t bytesRead;
62     int first_number, next_number;
63     char *current;
64     int line_number = 0;
65
66     while ((bytesRead = read(file, buffer, BUFFER_SIZE)) > 0) {
67         current = buffer;
68
69         while (current < buffer + bytesRead) {
70             while (*current == ' ' || *current == '\t') current++;
71             if (*current == '\n') {
72                 current++;
73                 continue;
74             }
75
76             char *endptr;
77             first_number = safe_strtol(str: current, &endptr);
78             current = endptr;
79
80             char result[BUFFER_SIZE];
81             int result_len = snprintf(result, BUFFER_SIZE, "Результат: %d", first_number);
82
83             while (current < buffer + bytesRead && *current != '\n') {
84                 while (*current == ' ' || *current == '\t') current++;
85                 if (*current == '\n') break;
86
87                 next_number = safe_strtol(str: current, &endptr);
88                 if (next_number == 0) {
89                     strncpy(shared_mem + line_number * BUFFER_SIZE, "Ошибка: Деление на ноль.\n", BUFFER_SIZE);
90                     sem_post(semaphore);
91                     exit(EXIT_FAILURE);
92                 }
93
94                 result_len += snprintf(result + result_len, BUFFER_SIZE - result_len, " %d / %d = %d",
95                                         first_number, next_number, first_number / next_number);
96                 current = endptr;
97             }
98
99             result[result_len++] = '\n';
100             strncpy(shared_mem + line_number * BUFFER_SIZE, result, result_len);
101             line_number++;
102
103             if (line_number >= NUM_LINES) break;
104         }
105     }
106
107     if (bytesRead == -1) {
108         strncpy(shared_mem, "Ошибка чтения файла.\n", BUFFER_SIZE);
109         sem_post(semaphore);
110         exit(EXIT_FAILURE);
111     }
112
113     close(file);
114     sem_post(semaphore);
115     exit(EXIT_SUCCESS);
116 }
117

```

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:

```
SYSCALL(args) = return
Введите имя файла: munmap(0x104654000, 0x84000) = 0 0
munmap(0x1046D8000, 0x8000) = 0 0
munmap(0x1046E0000, 0x4000) = 0 0
munmap(0x1046E4000, 0x4000) = 0 0
munmap(0x1046E8000, 0x48000) = 0 0
munmap(0x104730000, 0x4C000) = 0 0
open("./0", 0x100000, 0x0) = 3 0
fcntl(0x3, 0x32, 0x16B8DF0F8) = 0 0
close(0x3) = 0 0
fsgetpath(0x16B8DF108, 0x400, 0x16B8DF0E8) = 60 0
fsgetpath(0x16B8DF118, 0x400, 0x16B8DF0F8) = 14 0
csrctl(0x0, 0x16B8DF51C, 0x4) = -1 Err#1
__mac_syscall(0x18585BD62, 0x2, 0x16B8DF460) = 0 0
csrctl(0x0, 0x16B8DF50C, 0x4) = -1 Err#1
__mac_syscall(0x185858B95, 0x5A, 0x16B8DF4A0) = 0 0
sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16B8DEA08, 0x16B8DEA00, 0x18585A888, 0xD) = 0 0
sysctl([CTL_KERN, 157, 0, 0, 0, 0] (2), 0x16B8DEAB8, 0x16B8DEAB0, 0x0, 0x0) = 0 0
open("/0", 0x20100000, 0x0) = 3 0
openat(0x3, "System/Cryptexes/OS0", 0x100000, 0x0) = 4 0
dup(0x4, 0x0, 0x0) = 5 0
fstatat64(0x4, 0x16B8DE591, 0x16B8DE500) = 0 0
openat(0x4, "System/Library/dyld/0", 0x100000, 0x0) = 6 0
fcntl(0x6, 0x32, 0x16B8DE590) = 0 0
dup(0x6, 0x0, 0x0) = 7 0
dup(0x5, 0x0, 0x0) = 8 0
close(0x3) = 0 0
close(0x5) = 0 0
close(0x4) = 0 0
close(0x6) = 0 0
__mac_syscall(0x18585BD62, 0x2, 0x16B8DEF80) = 0 0
shared_region_check_np(0x16B8DEBA0, 0x0, 0x0) = 0 0
fsgetpath(0x16B8DF120, 0x400, 0x16B8DF048) = 82 0
fcntl(0x8, 0x32, 0x16B8DF120) = 0 0
close(0x8) = 0 0
```

```

close(0x7) = 0 0
getfsstat64(0x0, 0x0, 0x2) = 10 0
getfsstat64(0x10451C050, 0x54B0, 0x2) = 10 0
getattrlist("/^0", 0x16B8DF060, 0x16B8DEFD0) = 0 0
stat64("/Users/matveyd/CLionProjects/OS-labs/lab3/src/parent\0", 0x16B8DE870, 0x0) = 0 0
open("/Users/matveyd/CLionProjects/OS-labs/lab3/src/parent\0", 0x0, 0x0) = 3 0
mmap(0x0, 0x8628, 0x1, 0x40002, 0x3, 0x0) = 0x10451C000 0
fcntl(0x3, 0x32, 0x16B8DE988) = 0 0
close(0x3) = 0 0
munmap(0x10451C000, 0x8628) = 0 0
open("/Users/matveyd/CLionProjects/OS-labs/lab3/src/parent\0", 0x0, 0x0) = 3 0
__mac_syscall(0x18585BD62, 0x2, 0x16B8DC200) = 0 0
map_with_linking_np(0x16B8DC090, 0x1, 0x16B8DC0C0) = 0 0
close(0x3) = 0 0
mprotect(0x104514000, 0x4000, 0x1) = 0 0
open("/dev/dtracehelper\0", 0x2, 0x0) = 3 0
ioctl(0x3, 0x80086804, 0x16B8DB588) = 0 0
close(0x3) = 0 0
shared_region_check_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0) = 0 0
bsdthread_register(0x185B5E0F4, 0x185B5E0E8, 0x4000) = 1073746399 0
getpid(0x0, 0x0, 0x0) = 1849 0
shm_open(0x1859F5F41, 0x0, 0xFFFFFFFFF85B9C000) = 3 0
fstat64(0x3, 0x16B8DBC00, 0x0) = 0 0
mmap(0x0, 0x8000, 0x1, 0x40001, 0x3, 0x0) = 0x104524000 0
close(0x3) = 0 0
csops(0x739, 0x0, 0x16B8DBD3C) = 0 0
ioctl(0x2, 0x4004667A, 0x16B8DBCAC) = 0 0
mprotect(0x104534000, 0x4000, 0x0) = 0 0
mprotect(0x104540000, 0x4000, 0x0) = 0 0
mprotect(0x104544000, 0x4000, 0x0) = 0 0
mprotect(0x104550000, 0x4000, 0x0) = 0 0
mprotect(0x104554000, 0x4000, 0x0) = 0 0
mprotect(0x104560000, 0x4000, 0x0) = 0 0
mprotect(0x10452C000, 0xC8, 0x1) = 0 0
mprotect(0x10452C000, 0xC8, 0x3) = 0 0
mprotect(0x10452C000, 0xC8, 0x1) = 0 0
mprotect(0x104564000, 0x4000, 0x1) = 0 0
mprotect(0x104568000, 0xC8, 0x1) = 0 0
mprotect(0x104568000, 0xC8, 0x3) = 0 0
mprotect(0x104568000, 0xC8, 0x1) = 0 0
mprotect(0x10452C000, 0xC8, 0x3) = 0 0
mprotect(0x10452C000, 0xC8, 0x1) = 0 0
mprotect(0x104564000, 0x4000, 0x3) = 0 0
mprotect(0x104564000, 0x4000, 0x1) = 0 0
issetugid(0x0, 0x0, 0x0) = 0 0
getentropy(0x16B8DB318, 0x20, 0x0) = 0 0
getattrlist("/Users/matveyd/CLionProjects/OS-labs/lab3/src/parent\0", 0x16B8DBBA0, 0x16B8DBBBC) = 0 0
access("/Users/matveyd/CLionProjects/OS-labs/lab3/src\0", 0x4, 0x0) = 0 0
open("/Users/matveyd/CLionProjects/OS-labs/lab3/src\0", 0x0, 0x0) = 3 0
fstat64(0x3, 0x13A604470, 0x0) = 0 0
csrctl(0x0, 0x16B8DBD8C, 0x4) = 0 0
fcntl(0x3, 0x32, 0x16B8DBA88) = 0 0
close(0x3) = 0 0
proc_info(0x2, 0x739, 0xD) = 64 0
csops_audittoken(0x739, 0x10, 0x16B8DBE10) = 0 0
sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16B8DC168, 0x16B8DC160, 0x18926FD3A, 0x15) = 0 0
sysctl([CTL_KERN, 155, 0, 0, 0, 0] (2), 0x16B8DC1F8, 0x16B8DC1F0, 0x0, 0x0) = 0 0
shm_open(0x104513D92, 0x202, 0x1B6) = 3 0
ftruncate(0x3, 0x19000, 0x0) = 0 0
mmap(0x0, 0x19000, 0x3, 0x40001, 0x3, 0x0) = 0x104570000 0
sem_open(0x104513E7E, 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 / 47 = 1
Ошибка: Деление на ноль.
read(0x0, "input.txt\n\0", 0x400)          = 10 0
fork()          = 1850 0
sem_post(0x4, 0x0, 0x0)          = 0 0
wait4(0xFFFFFFFFFFFFFFFF, 0x0, 0x0)          = 1850 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 / 47 = 1\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(0x104570000, 0x19000)          = 0 0
shm_unlink(0x104513D92, 0x0, 0x0)          = 0 0
sem_close(0x4, 0x0, 0x0)          = 0 0
sem_unlink(0x104513E7E, 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.

