

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №1 по курсу

«Операционные системы»

Группа: М8О-210Б-23

Студент: Дворников М.Д.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 01.12.24

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`.

Программа `parent.c` запрашивает у пользователя имя файла, содержащего команды, создает канал (pipe) для связи с дочерним процессом и использует системный вызов `fork()` для его создания. Родительский процесс перенаправляет стандартный поток вывода дочернего процесса через pipe, чтобы прочесть результаты вычислений.

Дочерний процесс, выполняющий программу `child.c`, перенаправляет стандартный ввод на открытый файл, указанный родительским процессом, считывает строки чисел и производит деление первого числа строки на последующие. Результаты деления выводятся в стандартный поток вывода и записываются в файл `output.txt`.

В случае ошибок, таких как деление на 0 или некорректный ввод, дочерний процесс завершает свою работу, уведомляя об этом родительский процесс через системный вызов. Родительский процесс ожидает завершения дочернего процесса с помощью `wait()` и выводит результаты вычислений на экран.

Код программы

Parent.c

```
1  #include <stdlib.h>
2  #include <unistd.h>
3  #include <fcntl.h>
4  #include <sys/wait.h>
5  #include <string.h>
6  #include <stdio.h>
7
8  #define BUFFER_SIZE 1024
9
10 void error_handler(const char *msg) {
11     write( fd: STDERR_FILENO, buf: msg, nbyte: strlen( s: msg));
12     write( fd: STDERR_FILENO, buf: "\n", nbyte: 1);
13     exit(EXIT_FAILURE);
14 }
15
16 int main() {
17     int pipe1[2];
18     pid_t child_pid;
19     char filename[BUFFER_SIZE];
20     ssize_t bytesRead;
21
22     const char *prompt = "Введите имя файла: ";
23     write( fd: STDOUT_FILENO, buf: prompt, nbyte: strlen( s: prompt));
24
25     bytesRead = read(STDIN_FILENO, filename, sizeof(filename));
26     if (bytesRead <= 0) {
27         error_handler( msg: "Ошибка чтения имени файла");
28     }
29
30     if (filename[bytesRead - 1] == '\n') {
31         filename[bytesRead - 1] = '\0';
32     }
33
34     printf("[INFO] Создаем pipe...\n");
35     if (pipe(pipe1) == -1) {
36         error_handler( msg: "Ошибка создания pipe");
37     }
38
39     printf("[INFO] Создаем дочерний процесс...\n");
40     child_pid = fork();
41     if (child_pid == -1) {
42         error_handler( msg: "Ошибка создания процесса");
43     }
```

```

43     }
44
45     if (child_pid == 0) {
46         close(pipe1[0]);
47
48         printf("[INFO] Дочерний процесс: перенаправляем вывод в pipe...\n");
49         if (dup2(pipe1[1], STDOUT_FILENO) == -1) {
50             error_handler(msg: "Ошибка перенаправления вывода");
51         }
52         close(pipe1[1]);
53
54         printf("[INFO] Дочерний процесс: выполняем child...\n");
55         execl(path: "./child", arg0: "child", filename, NULL);
56         error_handler(msg: "Ошибка выполнения дочернего процесса");
57     } else {
58         close(pipe1[1]);
59
60         char buffer[BUFFER_SIZE];
61         ssize_t readBytes;
62         printf("[INFO] Родительский процесс: читаем из pipe...\n");
63         while ((readBytes = read(pipe1[0], buffer, BUFFER_SIZE)) > 0) {
64             if (write(fd: STDOUT_FILENO, buf: buffer, nbyte: readBytes) == -1) {
65                 error_handler(msg: "Ошибка записи в консоль");
66             }
67         }
68         if (readBytes == -1) {
69             error_handler(msg: "Ошибка чтения из pipe");
70         }
71
72         close(pipe1[0]);
73         wait(NULL);
74         printf("[INFO] Родительский процесс: завершено ожидание дочернего процесса.\n");
75         exit(EXIT_SUCCESS);
76     }
77
78     return 0;
79 }

```

Child.c

```
1  #include <stdlib.h>
2  #include <unistd.h>
3  #include <fcntl.h>
4  #include <string.h>
5  #include <errno.h>
6  #include <stdio.h>
7  #include <limits.h>
8
9  #define BUFFER_SIZE 1024
10
11 void HandleError(const char *message) {
12     write( fd: STDERR_FILENO, buf: message, nbyte: strlen( s: message));
13     exit(EXIT_FAILURE);
14 }
15
16 int safe_strtol(const char *str, char **endptr) {
17     long value = strtol(str, endptr, base: 10);
18
19     if (value > INT_MAX || value < INT_MIN) {
20         HandleError( message: "Ошибка: значение выходит за пределы допустимого диапазона int.\n");
21     }
22     return (int)value;
23 }
24
25 int main(int argc, char *argv[]) {
26     if (argc < 2) {
27         HandleError( message: "Ошибка: необходимо указать путь к файлу.\n");
28     }
29
30     int file = open(argv[1], O_RDONLY);
31     if (file == -1) {
32         char error_msg[BUFFER_SIZE];
33         snprintf(error_msg, sizeof(error_msg), "Ошибка открытия файла '%s': %s\n", argv[1], strerror( errnum: errno));
34         HandleError( message: error_msg);
35     }
36
37     int output_fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
38     if (output_fd == -1) {
39         HandleError( message: "Ошибка создания файла output.txt.\n");
40     }
41
42     if (dup2(file, STDIN_FILENO) == -1) {
43         HandleError( message: "Ошибка перенаправления ввода.\n");
44     }
```

```

45     close(file);
46
47     char buffer[BUFFER_SIZE];
48     ssize_t bytesRead;
49     char *current;
50     char output[BUFFER_SIZE];
51     int output_len;
52
53     while ((bytesRead = read(STDIN_FILENO, buffer, BUFFER_SIZE)) > 0) {
54         current = buffer;
55
56         while (current < buffer + bytesRead) {
57             while (*current == ' ' || *current == '\t') {
58                 current++;
59             }
60             if (*current == '\n') {
61                 current++;
62                 continue;
63             }
64
65             char *endptr;
66             int first_number = safe_strtol(str: current, &endptr);
67             int original_number = first_number;
68             current = endptr;
69
70             output_len = snprintf(output, BUFFER_SIZE, "Число: %d", original_number);
71
72             while (1) {
73                 while (*current == ' ' || *current == '\t') {
74                     current++;
75                 }
76
77                 if (*current == '\n' || current >= buffer + bytesRead) {
78                     break;
79                 }
80
81                 int next_number = safe_strtol(str: current, &endptr);
82                 if (next_number == 0) {
83                     output_len = snprintf(output, BUFFER_SIZE, "Ошибка: деление на недопустимый символ.\n");
84                     write(fd: STDOUT_FILENO, buf: output, nbyte: output_len);
85                     close(output_fd);
86                     exit(EXIT_FAILURE);
87

```

```

86                     exit(EXIT_FAILURE);
87                 }
88
89                 output_len += snprintf(output + output_len, BUFFER_SIZE - output_len, " %d / %d = %d", original_number, next_number, original_number / next_number);
90
91                 current = endptr;
92             }
93
94             output[output_len++] = '\n';
95
96             write(fd: STDOUT_FILENO, buf: output, nbyte: output_len);
97             write(fd: output_fd, buf: output, nbyte: output_len);
98
99             while (*current != '\n' && current < buffer + bytesRead) {
100                 current++;
101             }
102             if (*current == '\n') {
103                 current++;
104             }
105         }
106     }
107
108     if (bytesRead == -1) {
109         HandleError( message: "Ошибка чтения файла.\n");
110     }
111
112     close(output_fd);
113     exit(EXIT_SUCCESS);
114 }
115

```

Протокол работы программы

Тестирование:

<pre>cd lab1/src/) ./child Ошибка: необходимо указать путь к файлу.) ./child input.txt Число: 12, 12 / 3 = 4, 12 / 4 = 3, 12 / 6 = 2, 12 / 12 = 1 Число: 20, 20 / 2 = 10, 20 / 2 = 10, 20 / 5 = 4, 20 / 10 = 2, 20 / 4 = 5 Число: 10, 10 / 5 = 2, 10 / 2 = 5, 10 / 1 = 10, 10 / 10 = 1 Число: 52, 52 / 2 = 26, 52 / 4 = 13 Ошибка: деление на недопустимый символ.</pre>	<table><tr><td>1</td><td>12 3 4 6 12</td><td></td></tr><tr><td>2</td><td>20 2 2 5</td><td>10 4</td></tr><tr><td>3</td><td>10 5</td><td>2 1 10</td></tr><tr><td>4</td><td>52 2 4</td><td></td></tr><tr><td>5</td><td>50 25</td><td>> </td></tr></table>	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 25	>
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 25	>														

1	Число: 12, 12 / 3 = 4, 12 / 4 = 3, 12 / 6 = 2, 12 / 12 = 1
2	Число: 20, 20 / 2 = 10, 20 / 2 = 10, 20 / 5 = 4, 20 / 10 = 2, 20 / 4 = 5
3	Число: 10, 10 / 5 = 2, 10 / 2 = 5, 10 / 1 = 10, 10 / 10 = 1
4	Число: 52, 52 / 2 = 26, 52 / 4 = 13
5	

<pre>./child input.txt Число: 12, 12 / 3 = 4, 12 / 4 = 3, 12 / 6 = 2, 12 / 12 = 1 Число: 20, 20 / 2 = 10, 20 / 2 = 10, 20 / 5 = 4, 20 / 10 = 2, 20 / 4 = 5 Число: 10, 10 / 5 = 2, 10 / 2 = 5, 10 / 1 = 10, 10 / 10 = 1 Число: 52, 52 / 2 = 26, 52 / 4 = 13 Ошибка: деление на недопустимый символ.</pre>	<table><tr><td>1</td><td>12 3 4 6 12</td><td></td></tr><tr><td>2</td><td>20 2 2 5</td><td>10 4</td></tr><tr><td>3</td><td>10 5</td><td>2 1 10</td></tr><tr><td>4</td><td>52 2 4</td><td></td></tr><tr><td>5</td><td>50 2</td><td>5 0</td></tr></table>	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 2	5 0
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 2	5 0														

dtrace:

```
SYSCALL(args) = return
Введите имя файла: munmap(0x1032B8000, 0x84000) = 0 0
munmap(0x10333C000, 0x8000) = 0 0
munmap(0x103344000, 0x4000) = 0 0
munmap(0x103348000, 0x4000) = 0 0
munmap(0x10334C000, 0x48000) = 0 0
munmap(0x103394000, 0x4C000) = 0 0
open("\0", 0x100000, 0x0) = 3 0
fcntl(0x3, 0x32, 0x16CE5F0F8) = 0 0
close(0x3) = 0 0
fsgetpath(0x16CE5F108, 0x400, 0x16CE5F0E8) = 60 0
fsgetpath(0x16CE5F118, 0x400, 0x16CE5F0F8) = 14 0
csrctl(0x0, 0x16CE5F51C, 0x4) = -1 Err#1
__mac_syscall(0x18BF17D62, 0x2, 0x16CE5F460) = 0 0
csrctl(0x0, 0x16CE5F50C, 0x4) = -1 Err#1
__mac_syscall(0x18BF14B95, 0x5A, 0x16CE5F4A0) = 0 0
sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16CE5EA08, 0x16CE5EA00, 0x18BF16888, 0xD) = 0 0
sysctl([CTL_KERN, 157, 0, 0, 0, 0] (2), 0x16CE5EAB8, 0x16CE5EAB0, 0x0, 0x0) = 0 0
open("\0", 0x20100000, 0x0) = 3 0
openat(0x3, "System/Cryptexes/OS\0", 0x100000, 0x0) = 4 0
dup(0x4, 0x0, 0x0) = 5 0
fstatat64(0x4, 0x16CE5E591, 0x16CE5E500) = 0 0
openat(0x4, "System/Library/dyld\0", 0x100000, 0x0) = 6 0
fcntl(0x6, 0x32, 0x16CE5E590) = 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(0x18BF17D62, 0x2, 0x16CE5EF80) = 0 0
shared_region_check_np(0x16CE5EBA0, 0x0, 0x0) = 0 0
fsgetpath(0x16CE5F120, 0x400, 0x16CE5F048) = 82 0
fcntl(0x8, 0x32, 0x16CE5F120) = 0 0
close(0x8) = 0 0
close(0x7) = 0 0
getfsstat64(0x0, 0x0, 0x2) = 10 0
getfsstat64(0x102F9C050, 0x54B0, 0x2) = 10 0
getattrlist("\0", 0x16CE5F060, 0x16CE5EFD0) = 0 0
stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld/dyld_shared_cache_arm64e\0", 0x16CE5F3C0, 0x0) = 0
stat64("/Users/matveyd/CLionProjects/OS-labs-active/lab1/src/parent\0", 0x16CE5E870, 0x0) = 0 0
open("/Users/matveyd/CLionProjects/OS-labs-active/lab1/src/parent\0", 0x0, 0x0) = 3 0
```



```

mmap(0x0, 0x84E8, 0x1, 0x40002, 0x3, 0x0)          = 0x102F9C000 0
fcntl(0x3, 0x32, 0x16CE5E988)                    = 0 0
close(0x3)                                         = 0 0
munmap(0x102F9C000, 0x84E8)                       = 0 0
open("/Users/matveyd/CLionProjects/OS-labs-active/lab1/src/parent\0", 0x0, 0x0)           = 3 0
__mac_syscall(0x18BF17D62, 0x2, 0x16CE5C200)      = 0 0
map_with_linking_np(0x16CE5C0C0, 0x1, 0x16CE5C0F0) = 0 0
close(0x3)                                         = 0 0
mprotect(0x102F94000, 0x4000, 0x1)                = 0 0
open("/dev/dtracehelper\0", 0x2, 0x0)              = 3 0
ioctl(0x3, 0x80086804, 0x16CE5B588)               = 0 0
close(0x3)                                         = 0 0
shared_region_check_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0) = 0 0
bsdthread_register(0x18C21A0F4, 0x18C21A0E8, 0x4000) = 1073746399 0
getpid(0x0, 0x0, 0x0)                             = 1490 0
shm_open(0x18C0B1F41, 0x0, 0xFFFFFFFFF8C258000)   = 3 0
fstat64(0x3, 0x16CE5BC00, 0x0)                    = 0 0
mmap(0x0, 0x8000, 0x1, 0x40001, 0x3, 0x0)          = 0x102FA4000 0
close(0x3)                                         = 0 0
csops(0x5D2, 0x0, 0x16CE5BD3C)                    = 0 0
ioctl(0x2, 0x4004667A, 0x16CE5BCAC)               = 0 0
mprotect(0x102FB4000, 0x4000, 0x0)                 = 0 0
mprotect(0x102FC0000, 0x4000, 0x0)                 = 0 0
mprotect(0x102FC4000, 0x4000, 0x0)                 = 0 0
mprotect(0x102FD0000, 0x4000, 0x0)                 = 0 0
mprotect(0x102FD4000, 0x4000, 0x0)                 = 0 0
mprotect(0x102FE0000, 0x4000, 0x0)                 = 0 0
mprotect(0x102FAC000, 0xC8, 0x1)                   = 0 0
mprotect(0x102FAC000, 0xC8, 0x3)                   = 0 0
mprotect(0x102FAC000, 0xC8, 0x1)                   = 0 0
mprotect(0x102FE4000, 0x4000, 0x1)                 = 0 0
mprotect(0x102FE8000, 0xC8, 0x1)                   = 0 0
mprotect(0x102FE8000, 0xC8, 0x3)                   = 0 0
mprotect(0x102FE8000, 0xC8, 0x1)                   = 0 0
mprotect(0x102FAC000, 0xC8, 0x3)                   = 0 0
mprotect(0x102FAC000, 0xC8, 0x1)                   = 0 0
mprotect(0x102FE4000, 0x4000, 0x3)                 = 0 0
mprotect(0x102FE4000, 0x4000, 0x1)                 = 0 0
issetugid(0x0, 0x0, 0x0)                         = 0 0
getentropy(0x16CE5B318, 0x20, 0x0)                = 0 0
getattrlist("/Users/matveyd/CLionProjects/OS-labs-active/lab1/src/parent\0", 0x16CE5BBA0, 0x16CE5BBBC) = 0 0
access("/Users/matveyd/CLionProjects/OS-labs-active/lab1/src\0", 0x4, 0x0)               = 0 0
open("/Users/matveyd/CLionProjects/OS-labs-active/lab1/src\0", 0x0, 0x0)               = 3 0
fstat64(0x3, 0x13E604470, 0x0)                = 0 0
csrctl(0x0, 0x16CE5BD8C, 0x4)                  = 0 0
fcntl(0x3, 0x32, 0x16CE5BA88)                  = 0 0
close(0x3)                                       = 0 0
open("/Users/matveyd/CLionProjects/OS-labs-active/lab1/src/Info.plist\0", 0x0, 0x0)      = -1 Err#2
proc_info(0x2, 0x5D2, 0xD)                     = 64 0
csops_audittoken(0x5D2, 0x10, 0x16CE5BE10)       = 0 0
sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16CE5C168, 0x16CE5C160, 0x18F92BD3A, 0x15)       = 0 0
sysctl([CTL_KERN, 155, 0, 0, 0, 0] (2), 0x16CE5C1F8, 0x16CE5C1F0, 0x0, 0x0)            = 0 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
[INFO] Создаем pipe...
[INFO] Создаем дочерний процесс...
[INFO] Родительский процесс: читаем из pipe...
[INFO] Дочерний процесс: перенаправляем вывод в pipe...
[INFO] Дочерний процесс: выполняем child...
[INFO] Родительский процесс: завершено ожидание дочернего процесса.
read(0x0, "input.txt\n\0", 0x400)              = 10 0
fstat64(0x1, 0x16CE6EC80, 0x0)                  = 0 0
ioctl(0x1, 0x4004667A, 0x16CE6ECCC)              = 0 0
write_nocancel(0x1, "[INFO] \320\241\320\276\320\267\320\264\320\260\320\265\320\274 pipe...\n\0", 0x1E) = 30 0
pipe(0x0, 0x0, 0x0)                             = 3 0
write_nocancel(0x1, "[INFO] \320\241\320\276\320\267\320\264\320\260\320\265\320\274
\320\264\320\276\321\207\320\265\321\200\320\275\320\270\320\271
\320\277\321\200\320\276\321\206\320\265\321\201\321\201...\n\0", 0x39) = 57 0

```



```

fork()          = 1492 0
close(0x4)      = 0 0
write_nocancel(0x1, "[INFO]
\320\240\320\276\320\264\320\270\321\202\320\265\320\273\321\214\321\201\320\272\320\270\320\271
\320\277\321\200\320\276\321\206\320\265\321\201\321\201: \321\207\320\270\321\202\320\260\320\265\320\274
\320\270\320\267 pipe...\n\0", 0x4A)          = 74 0
read(0x3, "[INFO] \320\224\320\276\321\207\320\265\321\200\320\275\320\270\320\271
\320\277\321\200\320\276\321\206\320\265\321\201\321\201:
\320\262\321\213\320\277\320\276\320\273\320\275\321\217\320\265\320\274 child...\n\0", 0x400)      = 68 0
write(0x1, "[INFO] \320\224\320\276\321\207\320\265\321\200\320\275\320\270\320\271
\320\277\321\200\320\276\321\206\320\265\321\201\321\201:
\320\262\321\213\320\277\320\276\320\273\320\275\321\217\320\265\320\274 child...\n\0", 0x44)      = 68 0
read(0x3, "\0", 0x400)          = 0 0
close(0x3)          = 0 0
wait4(0xFFFFFFFFFFFFFFFF, 0x0, 0x0)          = 1492 0
write_nocancel(0x1, "[INFO]
\320\240\320\276\320\264\320\270\321\202\320\265\320\273\321\214\321\201\320\272\320\270\320\271
\320\277\321\200\320\276\321\206\320\265\321\201\321\201:
\320\267\320\260\320\262\320\265\321\200\321\210\320\265\320\275\320\276
\320\276\320\266\320\270\320\264\320\260\320\275\320\270\320\265
\320\264\320\276\321\207\320\265\321\200\320\275\320\265\320\263\320\276
\320\277\321\200\320\276\321\206\320\265\321\201\321\201\320\260.\n\0", 0x79)          = 121 0

```

Вывод

Эта лабораторная работа помогла мне разобраться в том, как процессы взаимодействуют между собой с помощью системных вызовов. Я научился работать с такими механизмами, как `fork()` для создания процессов и `pipe()` для передачи данных между ними. Кроме того, я разобрался, как использовать системные вызовы для перенаправления ввода/вывода через `dup2()`, а также как обрабатывать ошибки системных вызовов, таких как `read()` и `write()`. Особенно полезным был опыт работы с проверкой деления на ноль и реализацией завершения процессов в случае ошибок. Работа оказалась сложной, но очень интересной и практической. Я лучше понял, как устроены базовые механизмы взаимодействия в операционных системах, и получил ценные навыки, которые пригодятся в дальнейшем изучении ОС.