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 прочитать 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

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

#define BUFFER_SIZE 1024

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 pipe1[2];
    pid_t child_pid;
    char filename[BUFFER_SIZE];
    ssize_t bytesRead;

    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");
    }

    if (filename[bytesRead - 1] == '\n') {
        filename[bytesRead - 1] = '\0';
    }

    printf("[INFO] Создаем pipe...\n");
    if (pipe(pipe1) == -1) {
        error_handler( msg: "Ошибка создания pipe");
    }

    printf("[INFO] Создаем дочерний процесс...\n");
    child_pid = fork();
    if (child_pid == -1) {
        error_handler( msg: "Ошибка создания процесса");
    }
```

```
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  }
80
```

Child.c

```c
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <errno.h>
#include <stdio.h>
#include <limits.h>

#define BUFFER_SIZE 1024

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 допустимого диапазона int.\n");
    }
    return (int)value;
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        HandleError( message: "Ошибка: необходимо указать путь к файлу.\n");
    }

    int file = open(argv[1], O_RDONLY);
    if (file == -1) {
        char error_msg[BUFFER_SIZE];
        snprintf(error_msg, sizeof(error_msg), "Ошибка открытия файла '%s': %s\n", argv[1], strerror( errnum: errno));
        HandleError( message: error_msg);
    }

    int output_fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (output_fd == -1) {
        HandleError( message: "Ошибка создания файла output.txt.\n");
    }

    if (dup2(file, STDIN_FILENO) == -1) {
        HandleError( message: "Ошибка перенаправления ввода.\n");
    }
```

```
    close(file);

    char buffer[BUFFER_SIZE];
    ssize_t bytesRead;
    char *current;
    char output[BUFFER_SIZE];
    int output_len;

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

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

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

            output_len = snprintf(output, BUFFER_SIZE, "Число: %d", original_number);

            while (1) {
                while (*current == ' ' || *current == '\t') {
                    current++;
                }

                if (*current == '\n' || current >= buffer + bytesRead) {
                    break;
                }

                int next_number = safe_strtol( str: current, &endptr);
                if (next_number == 0) {
                    output_len = snprintf(output, BUFFER_SIZE, "Ошибка: деление на недопустимый символ.\n");
                    write( fd: STDOUT_FILENO,  buf: output,  nbyte: output_len);
                    close(output_fd);
                    exit(EXIT_FAILURE);
                }

                output_len += snprintf(output + output_len, BUFFER_SIZE - output_len, ", %d / %d = %d", original_number, next_number, original_number / next_number);

                current = endptr;
            }

            output[output_len++] = '\n';

            write( fd: STDOUT_FILENO,  buf: output,  nbyte: output_len);
            write( fd: output_fd,  buf: output,  nbyte: output_len);

            while (*current != '\n' && current < buffer + bytesRead) {
                current++;
            }
            if (*current == '\n') {
                current++;
            }
        }
    }

    if (bytesRead == -1) {
        HandleError( message: "Ошибка чтения файла.\n");
    }

    close(output_fd);
    exit(EXIT_SUCCESS);
}
```

Протокол работы программы

Тестирование:

```
) 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.
```

```
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
```

```
) ./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.
```

```
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(0x1025EC000, 0x84000) = 0 0
munmap(0x102670000, 0x8000) = 0 0
munmap(0x102678000, 0x4000) = 0 0
munmap(0x10267C000, 0x4000) = 0 0
munmap(0x102680000, 0x48000) = 0 0
munmap(0x1026C8000, 0x4C000) = 0 0
crossarch_trap(0x0, 0x0, 0x0) = -1 Err#45
open(".\0", 0x100000, 0x0) = 3 0
fcntl(0x3, 0x32, 0x16DA6B0F8) = 0 0
close(0x3) = 0 0
fsgetpath(0x16DA6B108, 0x400, 0x16DA6B0E8) = 60 0
fsgetpath(0x16DA6B118, 0x400, 0x16DA6B0F8) = 14 0
csrctl(0x0, 0x16DA6B51C, 0x4) = -1 Err#1
__mac_syscall(0x181047D62, 0x2, 0x16DA6B460) = 0 0
csrctl(0x0, 0x16DA6B50C, 0x4) = -1 Err#1
__mac_syscall(0x181044B95, 0x5A, 0x16DA6B4A0) = 0 0
sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16DA6AA08, 0x16DA6AA00, 0x181046888, 0xD) = 0 0
sysctl([CTL_KERN, 157, 0, 0, 0, 0] (2), 0x16DA6AAB8, 0x16DA6AAB0, 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, 0x16DA6A591, 0x16DA6A500) = 0 0
openat(0x4, "System/Library/dyld/\0", 0x100000, 0x0) = 6 0
fcntl(0x6, 0x32, 0x16DA6A590) = 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(0x181047D62, 0x2, 0x16DA6AF80) = 0 0
shared_region_check_np(0x16DA6ABA0, 0x0, 0x0) = 0 0
fsgetpath(0x16DA6B120, 0x400, 0x16DA6B048) = 82 0
fcntl(0x8, 0x32, 0x16DA6B120) = 0 0
close(0x8) = 0 0
close(0x7) = 0 0
getfsstat64(0x0, 0x0, 0x2) = 10 0

getfsstat64(0x102390050, 0x54B0, 0x2) = 10 0
getattrlist("/\0", 0x16DA6B060, 0x16DA6AFD0) = 0 0
stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyl
d/dyld_shared_cache_arm64e\0", 0x16DA6B3C0, 0x0) = 0 0
syscall::stat64:return): invalid address (0x0) in action #11 at DIF
offset 12
stat64("/Users/matveyd/CLionProjects/OS-labs-
active/lab1/src/parent\0", 0x16DA6A870, 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) =
0x102390000 0
fcntl(0x3, 0x32, 0x16DA6A988) = 0 0
close(0x3) = 0 0
munmap(0x102390000, 0x84E8) = 0 0
**open("/Users/matveyd/CLionProjects/OS-labs-
active/lab1/src/parent\0", 0x0, 0x0) = 3 0**
__mac_syscall(0x181047D62, 0x2, 0x16DA68200) = 0 0
map_with_linking_np(0x16DA680C0, 0x1, 0x16DA680F0)
= 0 0
close(0x3) = 0 0
mprotect(0x102388000, 0x4000, 0x1) = 0 0
open("/dev/dtracehelper\0", 0x2, 0x0) = 3 0
ioctl(0x3, 0x80086804, 0x16DA67588) = 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(0x18134A0F4, 0x18134A0E8, 0x4000) =
1073746399 0
getpid(0x0, 0x0, 0x0) = 5357 0
shm_open(0x1811E1F41, 0x0, 0xFFFFFFFF81388000) = 3 0
fstat64(0x3, 0x16DA67C00, 0x0) = 0 0
mmap(0x0, 0x8000, 0x1, 0x40001, 0x3, 0x0) =
0x102398000 0
close(0x3) = 0 0
csops(0x14ED, 0x0, 0x16DA67D3C) = 0 0
ioctl(0x2, 0x4004667A, 0x16DA67CAC) = 0 0
mprotect(0x1023A8000, 0x4000, 0x0) = 0 0
mprotect(0x1023B4000, 0x4000, 0x0) = 0 0
mprotect(0x1023B8000, 0x4000, 0x0) = 0 0
mprotect(0x1023C4000, 0x4000, 0x0) = 0 0
mprotect(0x1023C8000, 0x4000, 0x0) = 0 0
mprotect(0x1023D4000, 0x4000, 0x0) = 0 0
mprotect(0x1023A0000, 0xC8, 0x1) = 0 0
mprotect(0x1023A0000, 0xC8, 0x3) = 0 0
mprotect(0x1023A0000, 0xC8, 0x1) = 0 0
mprotect(0x1023D8000, 0x4000, 0x1) = 0 0
mprotect(0x1023DC000, 0xC8, 0x1) = 0 0
mprotect(0x1023DC000, 0xC8, 0x3) = 0 0
mprotect(0x1023DC000, 0xC8, 0x1) = 0 0
mprotect(0x1023A0000, 0xC8, 0x3) = 0 0
mprotect(0x1023A0000, 0xC8, 0x1) = 0 0
mprotect(0x1023D8000, 0x4000, 0x3) = 0 0
mprotect(0x1023D8000, 0x4000, 0x1) = 0 0
issetugid(0x0, 0x0, 0x0) = 0 0
getentropy(0x16B0D7338, 0x20, 0x0) = 0 0
getattrlist("/Users/matveyd/CLionProjects/OS-labs-active/lab1/src/child", 0x16B0D7BC0, 0x16B0D7BDC) = 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, 0x12E604470, 0x0) = 0 0
csctl(0x0, 0x16B0D7DAC, 0x4) = 0 0
fcntl(0x3, 0x32, 0x16B0D7AA8) = 0 0
close(0x3) = 0 0
open("/Users/matveyd/CLionProjects/OS-labs-active/lab1/src/Info.plist", 0x0, 0x0) = -1 Err#2
proc_info(0x2, 0xAA7, 0xD) = 64 0
csops_audittoken(0xAA7, 0x10, 0x16B0D7E30) = -1 Err#22

sysctl([unknown, 3, 0, 0, 0], (2), 0x16B0D8180, 0x191437D3A, 0x15) = 0 0
input.txt
open("/Users/matveyd/CLionProjects/OS-labs-active/lab1/src/input.txt\0", 0x0, 0x0) = 3 0
open("output.txt\0", 0x601, 0x1A4) = 4 0
dup2(0x3, 0x0, 0x0) = 0 0
close(0x3) = 0 0
execve("./child", ["", "output.txt"], 0x7ffd258bde74 /* 34 vars */) = 0
read(0x0, "12 3 4 6   12\n20 2   2 5   10 4\n10 5   2 1 10\n52 2 4\n50   >\n\0", 0x400) = 80 0
write(0x1, "12 / 3 = 4, 12 / 6 = 2, 12 / 12 = 1\n\0", 0x40) = 64 0
write(0x4, "12 / 3 = 4, 12 / 6 = 2, 12 / 12 = 1\n\0", 0x40) = 64 0
write(0x1, "20 / 2 = 10, 20 / 5 = 4, 20 / 10 = 2, 20 / 20 = 1\n\0", 0x4E) = 78 0
write(0x4, "20 / 2 = 10, 20 / 5 = 4, 20 / 10 = 2, 20 / 20 = 1\n\0", 0x4E) = 78 0
write(0x1, "10 / 5 = 2, 10 / 2 = 5, 10 / 1 = 10, 10 / 10 = 1\n\0", 0x4E) = 65 0
write(0x4, "10 / 5 = 2, 10 / 2 = 5, 10 / 1 = 10, 10 / 10 = 1\n\0", 0x4E) = 65 0
write(0x1, "52 / 2 = 26, 52 / 4 = 13\n\0", 0x29) = 41 0
write(0x4, "52 / 2 = 26, 52 / 4 = 13\n\0", 0x29) = 41 0
write(0x4, "\320\236\321\210\320\200\320\270\320\275\320\276\320\272\320\260: \321\201\321\214\320\270\320\265
\320\275\320\260
\320\275\320\255\320\267\320\260\321\200\321\217\320\276\321\204\320\276\320\274\320\276\320\275\320\270\321\202\321\214\32
1\201\321\217\320\264\320\276\320\276\320\273\320\264\320\270\321\202\320\270\320\273\321\214\320\275\320\276\320\270\321\2
01\321\202\321\214\321\207\320\265\321\210\320\265\320\275\320\270\321\217\320\262\321\205\320\276\320\264\320\270\321\202
321\214\320\264\320\276\321\201\320\277\321\203\320\277\321\203\320\266\320\265\321\200\320\275\320\260\320\272\320\277\32
0\265\321\200\320\265\320\264\320\260\320\262\321\202\320\276\321\200\320\270\320\267\320\260
321\201\320\270\321\202\320\276\320\273.\n\0", 0x49) = 73 0
close(0x4) = 0 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.