

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-216Б-23

Студент: Громова В.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 10.10.24

Москва, 2024

Постановка задачи

Вариант 7.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в `pipe1`. Родительский процесс читает из `pipe1` и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами. В файле записаны команды вида: «число число число<newline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип `float`. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- `pid_t fork(void)`; – создает дочерний процесс.
- `int pipe(int *fd)`; – создает односторонний канал между процессами, `fd[0]` для чтения, `fd[1]` для записи.
- `int open(const char *pathname, int flags)`; – открывает файл и возвращает файловый дескриптор.
- `ssize_t read(int fd, void *buf, size_t count)`; – читает данные из файла или канала в буфер.
- `ssize_t write(int fd, const void *buf, size_t count)`; – записывает данные в файл, канал или стандартный вывод.
- `int dup2(int oldfd, int newfd)`; – перенаправляет один файловый дескриптор на другой.
- `int execl(const char *path, const char *arg0, ..., NULL)`; – заменяет текущий процесс другим исполняемым файлом.
- `int wait(int *status)`; – родительский процесс ждет завершения дочернего процесса.
- `void _exit(int status)`; – завершает процесс без вызова функций очистки стандартной библиотеки.

Краткое описание работы программы.

Родительский процесс запрашивает у пользователя имя файла, открывает его для чтения и создаёт `pipe` для передачи данных дочернему процессу.

С помощью `fork()` создаётся дочерний процесс, который перенаправляет стандартный ввод на файл, а стандартный вывод – в `pipe`, и запускает программу `child`.

Программа `child` читает числа из файла, выполняет необходимые вычисления (суммирует числа или проверяет их на простоту) и записывает результаты в стандартный вывод, который направлен в `pipe`.

Родительский процесс читает данные из pipe и выводит их на экран. После завершения работы дочернего процесса родительский процесс корректно завершает выполнение программы.

Код программы

parent.c

```
#include <unistd.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/wait.h>


int main() {

    char filename[256];

    const char prompt[] = "введите имя файла: ";

    write(STDOUT_FILENO, prompt, sizeof(prompt) - 1);

    ssize_t nameLength = read(STDIN_FILENO, filename, sizeof(filename) - 1);

    if (nameLength <= 0) {

        const char err[] = "ошибка ввода имени файла\n";

        write(STDERR_FILENO, err, sizeof(err) - 1);

        _exit(EXIT_FAILURE);

    }

    if (filename[nameLength - 1] == '\n')

        filename[nameLength - 1] = '\0';

    else

        filename[nameLength] = '\0';

    int fileDescriptor = open(filename, O_RDONLY);

    if (fileDescriptor == -1) {

        const char err[] = "ошибка открытия файла\n";
```

```

        write(STDERR_FILENO, err, sizeof(err) - 1);

        _exit(EXIT_FAILURE);
    }

    int pipefd[2];

    if (pipe(pipefd) == -1) {

        const char err[] = "ошибка создания pipe\n";

        write(STDERR_FILENO, err, sizeof(err) - 1);

        _exit(EXIT_FAILURE);
    }

    pid_t childPid = fork();

    if (childPid == -1) {

        const char err[] = "ошибка fork()\n";

        write(STDERR_FILENO, err, sizeof(err) - 1);

        _exit(EXIT_FAILURE);
    }

    if (childPid == 0) {

        close(pipefd[0]);

        dup2(fileDescriptor, STDIN_FILENO);

        dup2(pipefd[1], STDOUT_FILENO);

        close(fileDescriptor);

        close(pipefd[1]);

        execl("./child", "child", NULL);

        const char err[] = "ошибка execl()\n";

        write(STDERR_FILENO, err, sizeof(err) - 1);

        _exit(EXIT_FAILURE);
    } else {

        close(fileDescriptor);

        close(pipefd[1]);
    }

```

```

    char buffer[256];

    ssize_t bytesRead;

    while ((bytesRead = read(pipefd[0], buffer, sizeof(buffer))) > 0) {

        write(STDOUT_FILENO, buffer, bytesRead);

    }

    close(pipefd[0]);

    wait(NULL);

}

return 0;
}

```

child.c

```

#include <unistd.h>

#include <stdlib.h>

int main() {

    char buff[256];

    ssize_t bytesRead;

    float sm;

    char *curr;

    while ((bytesRead = read(STDIN_FILENO, buff, sizeof(buff) - 1)) > 0) {

        buff[bytesRead] = '\0';

        curr = buff;

        sm = 0.0f;

        while (*curr) {

            while (*curr == ' ' || *curr == '\t' || *curr == '\n')

                curr++;

```

```
int sign = 1;

if (*curr == '-') {

    sign = -1;

    curr++;

}

float number = 0.0f;

float fractional = 0.1f;

int has_dig = 0;

while (*curr >= '0' && *curr <= '9') {

    number = number * 10 + (*curr - '0');

    curr++;

    has_dig = 1;

}

if (*curr == '.') {

    curr++;

    while (*curr >= '0' && *curr <= '9') {

        number += (*curr - '0') * fractional;

        fractional *= 0.1f;

        curr++;

        has_dig = 1;

    }

}

if (has_dig)

    sm += number * sign;

while (*curr && *curr != ' ' && *curr != '\t' && *curr != '\n')
```

```

        curr++;

    }

    char output[64];

    int out_i = 0;

    int integerPart = (int)sm;

    float fractionPart = sm - integerPart;

    if (fractionPart < 0) fractionPart = -fractionPart;

    char tmpDigits[32];

    int tmp_i = 0;

    int abs_num;

    if (integerPart < 0)

        abs_num = -integerPart;

    else

        abs_num = integerPart;

    if (abs_num == 0) {

        tmpDigits[tmp_i++] = '0';

    } else {

        while (abs_num > 0) {

            tmpDigits[tmp_i++] = '0' + (abs_num % 10);

            abs_num = abs_num / 10;

        }

    }

    if (integerPart < 0)

        output[out_i++] = '-';

    while (tmp_i > 0) {

        output[out_i++] = tmpDigits[tmp_i - 1];
    }

```

```

        tmp_i--;

    }

    output[out_i++] = '.';

    for (int i = 0; i < 2; i++) {

        fractionPart = fractionPart * 10;

        int digit = (int)fractionPart;

        output[out_i++] = '0' + digit;

        fractionPart = fractionPart - digit;

    }

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

    write(STDOUT_FILENO, output, out_i);

}

return 0;
}

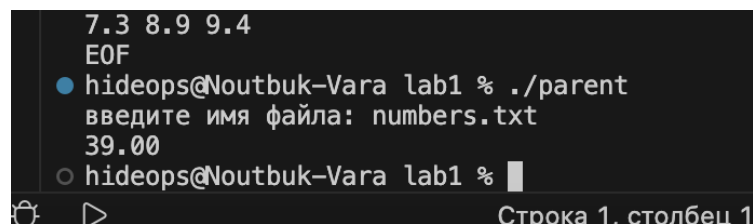
```

Протокол работы программы

Содержимое файла numbers.txt:

1.5 2.3 3.7
4.2 -5.1 6.8
7.3 8.9 9.4

Вывод программы:



```

7.3 8.9 9.4
EOF
hideops@Noutbuk-Vara lab1 % ./parent
введите имя файла: numbers.txt
39.00
hideops@Noutbuk-Vara lab1 %

```

Строка 1, столбец 1

Тестирование:

./parent

введите имя файла: numbers.txt

Strace:

execve("./parent", ["/parent"], 0x7ffd6d002e80 /* 27 vars */) = 0

brk(NULL) = 0x57dfba21a000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffea764a420) = -1 EINVAL (Invalid argument)


```

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f18d500a000
access("/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
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=59280, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 59280, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f18d4ffb000
close(3) = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0@\0\0\0\0\0\0@\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0"..., 48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\00{\f225\\=\201\327\312\301P\32$\230\266\235"...,
68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0@\0\0\0\0\0\0@\0\0\0\0\0\0"..., 784, 64) = 784
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f18d4c00000
mprotect(0x7f18d4c28000, 2023424, PROT_NONE) = 0
mmap(0x7f18d4c28000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f18d4c28000
mmap(0x7f18d4dbd000, 360448, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7f18d4dbd000
mmap(0x7f18d4e16000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f18d4e16000
mmap(0x7f18d4e1c000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f18d4e1c000
close(3) = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7f18d4ff8000
arch_prctl(ARCH_SET_FS, 0x7f18d4ff8740) = 0
set_tid_address(0x7f18d4ff8a10) = 8666
set_robust_list(0x7f18d4ff8a20, 24) = 0
rseq(0x7f18d4ff90e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f18d4e16000, 16384, PROT_READ) = 0
mprotect(0x57dfab56b000, 4096, PROT_READ) = 0
mprotect(0x7f18d5044000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) =
munmap(0x7f18d4ffb000, 59280) = 0
write(1, "\320\262\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"..., 34) = 34
read(0, "numbers.txt\n", 255) = 12
openat(AT_FDCWD, "numbers.txt", O_RDONLY) = 3
pipe2([4, 5], 0) = 0
clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f18d4ff8a10) = 8667
close(3) = 0
close(5) = 0
read(4, "39.00\n", 256) = 6
write(1, "39.00\n", 6) = 6
read(4, "", 256) = 0

```

```

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=8667, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
close(4) = 0
wait4(-1, NULL, 0, NULL) = 8667
exit_group(0) = ?
+++ exited with 0 +++

```

Вывод

Лабораторная работа показала, как использовать родительский и дочерний процессы с передачей данных через pipe. Программа корректно обрабатывает числа из файла и выводит результаты. Основная сложность заключалась в правильном перенаправлении потоков ввода/вывода через dup2. В будущем можно улучшить обработку ошибок и удобство ввода данных.