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-213Б-23

Студент: Марьин Д. А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 17.10.24

Москва, 2024

Постановка задачи

Вариант 9.

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<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 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 однонаправленный канал.
- size_t read(int fd, void *buf, size_t count); - читает данные из файлового дескриптора в буфер.
- int dup2(int oldfd, int newfd); - дублирует файловые дескрипторы.
- int execlp(const char *file, const char *arg, ..., (char *) NULL); - выполнение другой программы в текущем процессе.
- size_t write(int fd, const void *buf, size_t count); - записывает данные в файловый дескриптор.
- int close(int fd); - закрывает файловый дескриптор.
- int open(const char *pathname, int flags); - открывает файлы.

Для начала создаем два файла parent.c и child.c для родительского и дочернего процессов соответственно. Далее в файле parent.c инициализируем pipe и делаем fork данного процесса. Делаем условие: pid = 0, тогда выполняем блок кода для дочернего процесса, иначе блок кода родительского процесса. В дочернем процессе закрываем дескриптор чтения из канала, перенаправляем stdout в канал, запускаем программу для чтения файла (child.c) с помощью команды execve. В программе child открываем файл, путь до которого ввели в родительском процессе, перенаправляем stdin в файл, читаем файл и обрабатываем вещественные числа, командой write записываем в stdout полученный результат. В родительском процессе закрываем дескриптор записи в канал, читаем результат из канала и выводим его на экран, ожидаем завершения дочернего процесса с помощью wait.

Код программы

parent.c

```
#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <fcntl.h>



#define BUFSIZE 1024



int main() {



    const char
start_msg[] = "Type name
of your file: ";

    write(STDOUT_FILENO,
start_msg,
sizeof(start_msg));



    char
filename[BUFSIZE];

    int n =
read(STDIN_FILENO,
filename,
sizeof(filename) - 1);



    if (n > 0 && n <
sizeof(filename)) {

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

    } else {


filename[sizeof(filename)
- 1] = '\0';

    }



    int fd[2];

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

        const char msg[]
= "ERROR: pipe does not
open\n";


write(STDOUT_FILENO, msg,
sizeof(msg));


exit(EXIT_FAILURE);

    }



    pid_t pid = fork();
```

```
    switch (pid)

    {

    case -1:

        const char msg[]
= "ERROR: new process has
not been created\n";


write(STDOUT_FILENO, msg,
sizeof(msg));


exit(EXIT_FAILURE);

        break;

    case 0:

        close(fd[0]);

        dup2(fd[1],
STDOUT_FILENO);

        close(fd[1]);



        execlp("./child",
"./child", filename,
(char*)NULL);



        const char
exec_msg[] = "ERROR:
process has not started\
n";


write(STDERR_FILENO,
exec_msg,
sizeof(exec_msg));


exit(EXIT_FAILURE);

        break;

    default:

        close(fd[1]);

        char
print_msg[BUFSIZE];

        int n =
read(fd[0], print_msg,
sizeof(print_msg));


write(STDOUT_FILENO,
print_msg, n);

        close(fd[0]);

        break;

    }

}
```

child.c

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

#define BUFSIZE 1024

int is_separator(const char c) {
    return (c == ' ') || (c == '\n') || (c == '\t') || (c == '\0');
}

float my_pow(float base, int exp) {
    float result = 1.0;
    int is_negative = 0;

    if (exp < 0) {
        exp = -exp;
        is_negative = 1;
    }

    while (exp > 0) {
        if (exp % 2 == 1) {
            result *= base;
        }
        base *= base;
        exp /= 2;
    }

    if (is_negative) {
        result = 1.0 / result;
    }

    return result;
}

int validate_float_number(const char* str) {
    int len_str = strlen(str);
    if (len_str == 0) {
        return 0;
    }
    int num = 0;
    int was_sep = 0;
    for (int i = 0; i < len_str; ++i) {
        if (str[i] == '-') {
            if (i == 0) {
                continue;
            } else {
                return 0;
            }
        }
        if (str[i] == '.' || str[i] == ','){
            if (was_sep) {
                return 0;
            }
            was_sep = 1;
        } else if ((('0' > str[i] || str[i] > '9') && str[i] != '.' && str[i] != ',' ) ||
is_separator(str[i])) {
            return 0;
        }
    }
    return 1;
}

int reverse(char* str, int length) {
```

```
    int start = 0;
    int end = length - 1;
    while (start < end) {
        char temp = str[start];
        str[start] = str[end];
        str[end] = temp;
        start++;
        end--;
    }
    return 0;
}

int int_to_str(int num, char* str, int d) {
    int i = 0;
    int is_negative = 1 ? num < 0: 0;
    if (is_negative) {
        num *= -1;
    }
    if (num == 0) {
        str[i++] = '0';
    } else {
        while (num != 0) {
            str[i++] = (num % 10) + '0';
            num /= 10;
        }
    }

    while (i < d) {
        str[i++] = '0';  // добавляем нули, если нужно
    }

    if (is_negative) {
        str[i++] = '-';
    }
    reverse(str, i);
    str[i] = '\0';
    return i;
}

int float_to_str(float n, char* res, int afterpoint) {
    int ipart = (int)n;

    float fpart = n - (float)ipart;
    int i = int_to_str(ipart, res, 0);

    if (afterpoint != 0) {
        res[i] = '.';
        i++;

        fpart = fpart * my_pow(10, afterpoint);

        int_to_str((int)fpart, res + i, afterpoint);
    }
    return 0;
}

int main(int argc, char* argv[]) {
    if (argc != 2) {
        const char msg[] = "Usage: ./child <filename>\n";
             write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    int file = open(argv[1], O_RDONLY);

    if (file == -1) {
```

```
    const char msg[] = "ERROR: wrong file\n";
    write(STDOUT_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
}

dup2(file, STDIN_FILENO);
close(file);


char buffer[BUFSIZE];
char msg[BUFSIZE];
char ch[BUFSIZE];
int index = 0;
float current_number = 0;
float first_number = 0;
int is_first_number = 1;
int n = 0;
while ((n = read(STDIN_FILENO, ch, sizeof(char))) > 0)
{
     if (is_separator(*ch)) {
        buffer[index] = '\0';
        if (strlen(buffer) == 0) {
            continue;
        }
        if (validate_float_number(buffer)) {
            current_number = atof(buffer);
        } else {
            const char msg[] = "ERROR: wrong number\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }
        if (is_first_number) {
            first_number = current_number;
            is_first_number = 0;
        } else {
            if (current_number != 0) {
                first_number /= current_number;
            } else {
                const char msg[] = "ERROR: division by zero\n";
                write(STDOUT_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            }
        }
        index = 0;
    } else {
        buffer[index++] = *ch;
    }
}
// Обработка последнего числа в файле, если не было разделителя
if (index > 0) {
    buffer[index] = '\0';
    if (validate_float_number(buffer)) {
        current_number = atof(buffer);
    } else {
        const char msg[] = "ERROR: wrong number\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
    if (is_first_number) {
        first_number = current_number;
    } else {
        if (current_number != 0) {
            first_number /= current_number;
        } else {
            const char msg[] = "ERROR: division by zero\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
```

```
                exit(EXIT_FAILURE);
            }
        }
    }
    char str[BUFSIZE];
    float_to_str(first_number, str, 5);
    size_t size_msg = strlen(str);
    str[size_msg] = '\n';
    write(STDOUT_FILENO, str, ++size_msg);
}
```

Протокол работы программы

Тестирование:

```
$ ./parent.c
execve("./a.out",
["./a.out"],
0x7ffe99b92af0 /* 77 vars
*/) = 0

brk(NULL)
= 0x5e776e8d3000

mmap(NULL, 8192,
PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS,
-1, 0) = 0x789ea9748000

access("/etc/
ld.so.preload", R_OK)
= -1 ENOENT (Нет такого
файла или каталога)

openat(AT_FDCWD,
"/usr/local/cuda-12.6/lib6
4/glibc-hwcaps/x86-64-v3/l
ibc.so.6", O_RDONLY|
O_CLOEXEC) = -1 ENOENT
(Нет такого файла или
каталога)

newfstatat(AT_FDCWD,
"/usr/local/cuda-12.6/lib6
4/glibc-hwcaps/x86-64-
v3/", 0x7ffc0d5ac6f0, 0) =
-1 ENOENT (Нет такого
файла или каталога)

openat(AT_FDCWD,
"/usr/local/cuda-12.6/lib6
4/glibc-hwcaps/x86-64-v2/l
ibc.so.6", O_RDONLY|
O_CLOEXEC) = -1 ENOENT
(Нет такого файла или
каталога)

newfstatat(AT_FDCWD,
"/usr/local/cuda-12.6/lib6
4/glibc-hwcaps/x86-64-
v2/", 0x7ffc0d5ac6f0, 0) =
-1 ENOENT (Нет такого
файла или каталога)

openat(AT_FDCWD,
```

```
"/usr/local/cuda-12.6/lib6
4/libc.so.6", O_RDONLY|
O_CLOEXEC) = -1 ENOENT
(Нет такого файла или
каталога)

newfstatat(AT_FDCWD,
"/usr/local/cuda-12.6/lib6
4/", {st_mode=S_IFDIR|
0755, st_size=4096, ...},
0) = 0

openat(AT_FDCWD,
"/etc/ld.so.cache",
O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|
0644, st_size=78375, ...})
= 0

mmap(NULL, 78375,
PROT_READ, MAP_PRIVATE, 3,
0) = 0x789ea9734000

close(3)
= 0

openat(AT_FDCWD,
"/lib/x86_64-linux-gnu/lib
c.so.6", O_RDONLY|
O_CLOEXEC) = 3

read(3, "\177ELF\
2\1\1\3\0\0\0\0\0\0\0\0\3\
0>\
0\1\0\0\0\220\243\2\0\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\0\0\0"..., 784,
64) = 784

fstat(3, {st_mode=S_IFREG|
0755,
st_size=2125328, ...}) = 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\0\0\0"..., 784,
64) = 784

mmap(NULL, 2170256,
PROT_READ, MAP_PRIVATE|
MAP_DENYWRITE, 3, 0) =
```

```
0x789ea9400000

mmap(0x789ea9428000,
1605632, PROT_READ|
PROT_EXEC, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) =
0x789ea9428000

mmap(0x789ea95b0000,
323584, PROT_READ,
MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3,
0x1b0000) = 0x789ea95b0000

mmap(0x789ea95ff000,
24576, PROT_READ|
PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE,
3, 0x1fe000) =
0x789ea95ff000

mmap(0x789ea9605000,
52624, PROT_READ|
PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS, -
1, 0) = 0x789ea9605000

close(3)
= 0

mmap(NULL, 12288,
PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS,
-1, 0) = 0x789ea9731000

arch_prctl(ARCH_SET_FS,
0x789ea9731740) = 0

set_tid_address(0x789ea973
1a10)         = 400941

set_robust_list(0x789ea973
1a20, 24)     = 0

rseq(0x789ea9732060, 0x20,
0, 0x53053053) = 0

mprotect(0x789ea95ff000,
16384, PROT_READ) = 0

mprotect(0x5e776d6b8000,
4096, PROT_READ) = 0

mprotect(0x789ea9780000,
8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK,
NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY})
= 0
```

```
munmap(0x789ea9734000,
78375)           = 0

write(1, "Type name of
your file: \0", 25Type
name of your file: ) = 25

read(0, input.txt

"input.txt\n", 1023)
= 10

pipe2([3, 4], 0)
= 0

clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID
|CLONE_CHILD_SETTID|
SIGCHLD,
child_tidptr=0x789ea9731a1
0) = 412918

close(4)
= 0

read(3, "0.78003\n", 1024)
= 8

--- SIGCHLD
{si_signo=SIGCHLD,
si_code=CLD_EXITED,
si_pid=412918,
si_uid=1000, si_status=0,
si_utime=0, si_stime=0}
---

write(1, "0.78003\n",
80.78003

)                = 8

close(3)
= 0

exit_group(0)
= ?

+++ exited with 0 +++
```

Вывод

Успешно была организована связь между родительским и дочерним процессами с использованием межпроцессных каналов и системных вызовов. Результаты обработки вещественных чисел, полученных из файла, корректно передавались от дочернего процесса к родительскому и отображались на экране. Получены практические навыки в обеспечении обмена данными между процессами с помощью каналов.