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-214Б-24

Студент: Василянская А.Н.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 08.10.25

Москва, 2024

Постановка задачи

Вариант 6.

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». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип int. Количество чисел может быть произвольным

Общий метод и алгоритм решения

Родительский процесс запрашивает имя файла и открывает на чтение. В дочернем процессе перенаправляет STDIN на входной файл, а STDOUT - в pipe, затем запускает программу child. Родительский процесс читает результаты из pipe и выводит их в консоль.Дочерний процесс читает данные из STDIN построчно, обрабатывает каждую строку: проверяет наличие только числовых символов, пробелов и знаков, при невалидных данных выводит ошибку, при валидных - вычисляет сумму чисел и выводит результат в STDOUT.

Использованные системные вызовы:

- pid_t fork(void) – создание дочернего процесса
- int pipe(int *fd) – создание неименованного канала
- ssize_t write(int fd, const void *buf, size_t count) – запись данных
- ssize_t read(int fd, void *buf, size_t count) – чтение данных
- int open(const char *pathname, int flags) – открытие файла
- int close(int fd) – закрытие файлового дескриптора
- int dup2(int oldfd, int newfd) – перенаправление потоков
- int execl(const char *path, const char *arg, ...) – запуск программы
- pid_t wait(int *status) – ожидание завершения процесса
- void exit(int status) – завершение процесса

Код программы

parent.c

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

int main(int argc, char *argv[]) {
    char progpath[1024];
    {
        const char msg[] = "input file: ";
        write(STDOUT_FILENO, msg, sizeof(msg) - 1);

        ssize_t n = read(STDIN_FILENO, progpath, sizeof(progpath) - 1);
        if (n <= 0) {
            const char msg[] = "error: can't read the file\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);
            exit(EXIT_FAILURE);
        }
        progpath[n - 1] = '\0';
    }

    int child_to_parent[2];
    if (pipe(child_to_parent) == -1) {
        const char msg[] = "error: can't make the pipe\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }
```

```
pid_t pid = fork();

switch(pid) {
    case -1: {
        const char msg[] = "error: can't make a new process\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    } break;

    case 0: {
        close(child_to_parent[0]);

        int file = open(progpath, O_RDONLY);
        if (file == -1) {
            const char msg[] = "error: can't open file\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);
            exit(EXIT_FAILURE);
        }

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

        dup2(child_to_parent[1], STDOUT_FILENO);
        close(child_to_parent[1]);

        execl("./child", "child", NULL);
        const char msg[] = "error: can't exec child\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
```

```
        } break;

        default: {
            close(child_to_parent[1]);

            char buf[4096];
            ssize_t n;
            while ((n = read(child_to_parent[0], buf, sizeof(buf))) > 0) {
                write(STDOUT_FILENO, buf, n);
            }

            close(child_to_parent[0]);
            wait(NULL);
        } break;
    }

    return 0;
}
```

child.c

```
#include <stdlib.h>
#include <unistd.h>
#include <ctype.h>
#include <string.h>
#include <stdio.h>

int main() {
    char buf[4096];
    ssize_t n;
```

```
size_t pos = 0;

while ((n = read(STDIN_FILENO, buf + pos, sizeof(buf) - pos - 1)) > 0) {
    pos += n;
    buf[pos] = '\0';

    char *cur_line = buf;
    char *tmp_line;

    while ((tmp_line = strchr(cur_line, '\n'))) {
        *tmp_line = '\0';

        int only_spaces = 1;
        for (char *p = cur_line; *p; p++) {
            if (!isspace((unsigned char)*p)) {
                only_spaces = 0;
                break;
            }
        }
        if (only_spaces) {
            cur_line = tmp_line + 1;
            continue;
        }

        int sum = 0;
        int invalid_input = 0;
        char *p = cur_line;

        while (*p) {
            while (*p && isspace((unsigned char)*p)) p++;
```

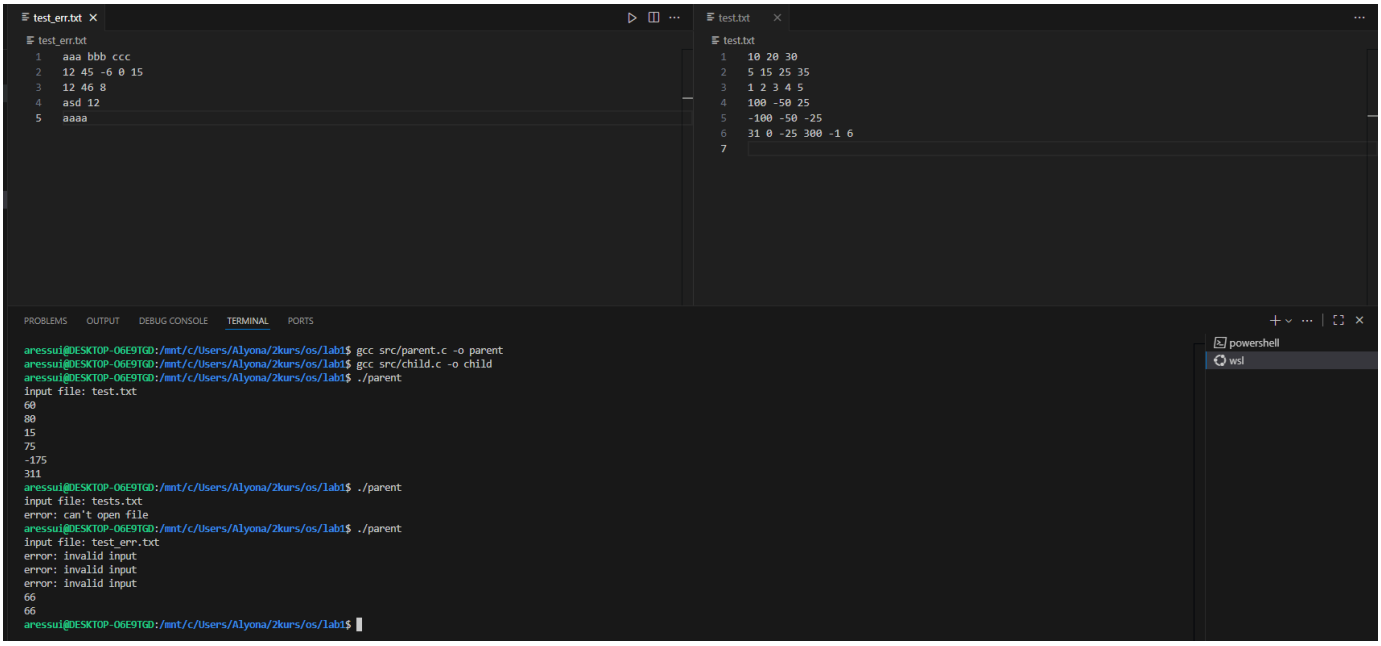
```
    if (!*p) break;

    int is_neg = 1;
    if (*p == '-') { is_neg = -1; p++; }

    if (!isdigit((unsigned char)*p)) {
        invalid_input = 1;
        break;
    }

    int num = 0;
    while (*p && isdigit((unsigned char)*p)) {
        num = num * 10 + (*p - '0');
        p++;
    }
    if (*p && !isspace((unsigned char)*p)) {
        invalid_input = 1;
        break;
    }
    sum += num * is_neg;

    while (*p && !isspace((unsigned char)*p)) p++;
}

if (invalid_input) {
    const char msg[] = "error: invalid input\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
} else {
    char out[64];
    int len = snprintf(out, sizeof(out), "%d\n", sum);
```

```
        write(STDOUT_FILENO, out, len);
    }

    cur_line = tmp_line + 1;
}

if (*cur_line) {
    int only_spaces = 1;
    for (char *p = cur_line; *p; p++) {
        if (!isspace((unsigned char)*p)) {
            only_spaces = 0;
            break;
        }
    }
    if (!only_spaces) {
        int sum = 0;
        int invalid_input = 0;
        char *p = cur_line;

        while (*p) {
            while (*p && isspace((unsigned char)*p)) p++;
            if (!*p) break;

            int is_neg = 1;
            if (*p == '-') { is_neg = -1; p++; }

            if (!isdigit((unsigned char)*p)) {
                invalid_input = 1;
                break;
            }
```

```
                int num = 0;
                while (*p && isdigit((unsigned char)*p)) {
                    num = num * 10 + (*p - '0');
                    p++;
                }

                sum += num * is_neg;

                while (*p && !isspace((unsigned char)*p)) p++;
            }

            if (invalid_input) {
                const char msg[] = "error: invalid input\n";
                write(STDERR_FILENO, msg, sizeof(msg) - 1);
            } else {
                char out[64];
                int len = snprintf(out, sizeof(out), "%d\n", sum);
                write(STDOUT_FILENO, out, len);
            }
        }
    }

    pos = 0;
  }

  return 0;
}
```

Протокол работы программы

Тестирование:



Strace:

aressui@DESKTOP-O6E9TGD:/mnt/c/Users/Alyona/2kurs/os/lab1$ strace -f ./parent

execve("./parent", ["./parent"], 0x7ffdd9769078 /* 26 vars */) = 0

brk(NULL) = 0x5d3f9bf03000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7d9a5feec000

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

fstat(3, {st_mode=S_IFREG|0644, st_size=20243, ...}) = 0

mmap(NULL, 20243, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7d9a5fee7000

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\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) = 0x7d9a5fc00000

mmap(0x7d9a5fc28000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7d9a5fc28000

mmap(0x7d9a5fdb0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7d9a5fdb0000

mmap(0x7d9a5fdff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1fe000) = 0x7d9a5fdff000

mmap(0x7d9a5fe05000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x7d9a5fe05000

close(3)   = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7d9a5fee4000

arch_prctl(ARCH_SET_FS, 0x7d9a5fee4740) = 0

set_tid_address(0x7d9a5fee4a10) = 111825

set_robust_list(0x7d9a5fee4a20, 24) = 0

rseq(0x7d9a5fee5060, 0x20, 0, 0x53053053) = 0

mprotect(0x7d9a5fdff000, 16384, PROT_READ) = 0

mprotect(0x5d3f83f46000, 4096, PROT_READ) = 0

mprotect(0x7d9a5ff24000, 8192, PROT_READ) = 0

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

munmap(0x7d9a5fee7000, 20243) = 0

write(1, "input file: ", 12input file:) = 12

read(0, test.txt

"test.txt\n", 1023) = 9

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

**clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|
SIGCHLDstrace: Process 111891 attached**

, child_tidptr=0x7d9a5fee4a10) = 111891

[pid 111825] close(4 <unfinished ...>

[pid 111891] set_robust_list(0x7d9a5fee4a20, 24 <unfinished ...>

[pid 111825] <... close resumed>) = 0

[pid 111891] <... set_robust_list resumed>) = 0

[pid 111825] read(3, <unfinished ...>

[pid 111891] close(3) = 0

[pid 111891] openat(AT_FDCWD, "test.txt", O_RDONLY) = 3

[pid 111891] dup2(3, 0) = 0

[pid 111891] close(3) = 0

[pid 111891] dup2(4, 1) = 1

[pid 111891] close(4) = 0

[pid 111891] execve("./child", ["child"], 0x7fffd098ee78 /* 26 vars */) = 0

[pid 111891] brk(NULL) = 0x63557771d000

[pid 111891] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x71960ad58000

[pid 111891] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

[pid 111891] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

[pid 111891] fstat(3, {st_mode=S_IFREG|0644, st_size=20243, ...}) = 0

[pid 111891] mmap(NULL, 20243, PROT_READ, MAP_PRIVATE, 3, 0) = 0x71960ad53000

[pid 111891] close(3) = 0

[pid 111891] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

[pid 111891] 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"..., 832) = 832

[pid 111891] 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

[pid 111891] fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

[pid 111891] 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

[pid 111891] mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x71960aa00000

[pid 111891] mmap(0x71960aa28000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x71960aa28000

[pid 111891] mmap(0x71960abb0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1b0000) = 0x71960abb0000

[pid 111891] mmap(0x71960abff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x71960abff000

[pid 111891] mmap(0x71960ac05000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x71960ac05000

[pid 111891] close(3) = 0

[pid 111891] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x71960ad50000

[pid 111891] arch_prctl(ARCH_SET_FS, 0x71960ad50740) = 0

[pid 111891] set_tid_address(0x71960ad50a10) = 111891

[pid 111891] set_robust_list(0x71960ad50a20, 24) = 0

[pid 111891] rseq(0x71960ad51060, 0x20, 0, 0x53053053) = 0

[pid 111891] mprotect(0x71960abff000, 16384, PROT_READ) = 0

[pid 111891] mprotect(0x635560ea1000, 4096, PROT_READ) = 0

[pid 111891] mprotect(0x71960ad90000, 8192, PROT_READ) = 0

[pid 111891] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

[pid 111891] munmap(0x71960ad53000, 20243) = 0

[pid 111891] read(0, "10 20 30\n5 15 25 35\n1 2 3 4 5\n10"..., 4095) = 72

[pid 111891] write(1, "60\n", 3)   = 3

[pid 111825] <... read resumed>"60\n", 4096) = 3

[pid 111891] write(1, "80\n", 3 <unfinished ...>

[pid 111825] write(1, "60\n", 3 <unfinished ...>

[pid 111891] <... write resumed>)   = 3

60

[pid 111825] <... write resumed>)   = 3

[pid 111891] write(1, "15\n", 3 <unfinished ...>

[pid 111825] read(3, <unfinished ...>

[pid 111891] <... write resumed>)   = 3

[pid 111825] <... read resumed>"80\n15\n", 4096) = 6

[pid 111891] write(1, "75\n", 3 <unfinished ...>

[pid 111825] write(1, "80\n15\n", 6 <unfinished ...>

[pid 111891] <... write resumed>)   = 3

80

15

[pid 111825] <... write resumed>)   = 6

[pid 111891] write(1, "-175\n", 5 <unfinished ...>

[pid 111825] read(3, <unfinished ...>

[pid 111891] <... write resumed>)   = 5

[pid 111825] <... read resumed>"75\n-175\n", 4096) = 8

[pid 111891] write(1, "311\n", 4 <unfinished ...>

[pid 111825] write(1, "75\n-175\n", 8 <unfinished ...>

[pid 111891] <... write resumed>)   = 4

75

-175

[pid 111825] <... write resumed>)   = 8

[pid 111825] read(3, <unfinished ...>

[pid 111891] read(0, <unfinished ...>

[pid 111825] <... read resumed>"311\n", 4096) = 4

[pid 111825] write(1, "311\n", 4311

) = 4

[pid 111891] <... read resumed>"", 4095) = 0

[pid 111825] read(3, <unfinished ...>

[pid 111891] exit_group(0) = ?

[pid 111825] <... read resumed>"", 4096) = 0

[pid 111825] close(3) = 0

[pid 111825] wait4(-1, <unfinished ...>

[pid 111891] +++ exited with 0 +++

<... wait4 resumed>NULL, 0, NULL) = 111891

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=111891, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В лабораторной работе использовались системные вызовы Linux для создания процессов и организации межпроцессного взаимодействия. Была реализована программа с разделением на родительский и дочерний процессы, общающиеся через pipe. Родительский процесс управляет вводом-выводом и запуском, а дочерний выполняет обработку данных - проверку строк на корректность числового формата и вычисление суммы чисел.