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-211Б-23

Студент: Тремель Д.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 30.10.24

Москва, 2024

# Постановка задачи

Вариант 6.

Цель работы

Приобретение практических навыков в:

- Управление процессами в ОС
- Обеспечение обмена данных между процессами посредством каналов

Задание

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы**.**

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

Использованные системные вызовы:

- pid_t fork(void); - создает дочерний процесс.
- int pipe(int *fd); - создает однонаправленный канал для обмена данными между процессами.
- int dup2(int oldfd, int newfd); - переназначение файлового дескриптора.
- ssize_t write(int fd, const void *buf, size_t count); - записывает count байт из буфера в файл.
- ssize_t read(int fd, void *buf, size_t count); - читает до count байт из файла
- int open(const char *pathname, int flags, mode_t mode); - открытие или создание файла.
- int close(int fd); — закрытие файла, связанного с файловым дескриптором fd.
- void exit(int status); — завершение выполнения процесса с возвратом кода status.
- int execl(const char *filename, char *const argv[]); — замена образа памяти текущего процесса программой, указанной в filename.

Главная программа открывает файл для чтения, создает канал (pipe) для передачи данных между процессами и запускает дочерний процесс с помощью fork(). Родительский процесс закрывает конец канала для чтения и начинает читать строки из файла, передавая их в канал через его левый конец (pipe[1]). В то же время дочерний процесс закрывает конец канала для записи, перенаправляет стандартный ввод (STDIN_FILENO) на чтение из канала с помощью dup2(pipe[0], STDIN_FILENO) и читает строки, поступающие через STDIN. После получения строки дочерний процесс разбивает её на отдельные числа, проверяет корректность каждого числа, суммирует их и выводит результат через стандартный вывод (STDOUT_FILENO).

# Код программы

**parent.c**

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

void error(const char* msg)
{
    write(STDERR_FILENO, msg, strlen(msg));
}

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

    FILE *file = fopen(argv[1], "r");
    if (file == NULL) {
        error("Error opening file");
        exit(EXIT_FAILURE);
    }

    int chanel[2];
    if (pipe(chanel) == -1) {
        error("Pipe failed");
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();
    if (pid < 0) {
        error("Fork failed");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {

        close(chanel[1]);

        dup2(chanel[0], STDIN_FILENO);
        close(chanel[0]);

        execv("./b","./b", NULL);
```

```
        error("execlp failed");
        exit(EXIT_FAILURE);
    } else {
        close(chanel[0]);

        char line[256];
        while (fgets(line, sizeof(line), file) != NULL) {
            write(chanel[1], line, strlen(line));
        }
        close(chanel[1]);
        fclose(file);

        wait(NULL);
    }

    return 0;
}
```

**child.c**

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

enum status{
    SUCCESS,
    INPUT_ERROR,
    OVERFLOW
};

int is_valid_int(const char* str) {
    if (*str == '\0') return 0;

    for (const char* p = str; *p; p++) {
        if (!isdigit(*p) && *p != '-' && *p != '+') {
            return 0;
        }
    }
    return 1;
}

enum status str_to_int(const char *str, int * result)
{
    if (!(is_valid_int(str))){
        return INPUT_ERROR;
    }
    size_t len = strlen(str);
    if (len > 11 || (len == 11 && str[0] == '-' && str[1] > '2') || (len == 10 &&
str[0] != '-')) {
```

```
        return OVERFLOW;
    }
    char *endptr;
    int temp;
    temp = strtol(str, &endptr, 10);
    if (*endptr != '\0') {
        return INPUT_ERROR;
    }
    *result = (int)temp;
    return SUCCESS;
}

void write_sum(int sum) {
    char msg[256];
    int len = snprintf(msg, sizeof(msg), "Sum: %d\n", sum);
    write(STDOUT_FILENO, msg, len);
}

int main() {
    char line[256];

    while (fgets(line, sizeof(line), stdin) != NULL) {
        int sum = 0;
        line[strcspn(line, "\n")] = '\0';
        char *token = strtok(line, " ");
        while (token != NULL) {
            int res = 0;
            enum status f = str_to_int(token, &res);

            if (f == OVERFLOW) {
                char msg[] = "Overflow\n";
                write(STDOUT_FILENO, msg, sizeof(msg));
                return -1;
            } else if (f == INPUT_ERROR) {
                char msg[] = "Data is incorrect\n";
                write(STDOUT_FILENO, msg, sizeof(msg));
                return -2;
            } else {
                sum += res;
            }

            token = strtok(NULL, " ");
        }

        write_sum(sum);
    }

    return 0;
}
```

# Протокол работы программы

```
u@DESKTOP-3U3OER0:/mnt/c/Users/u/CLionProjects/OS/lab_1$ cat
commands.txt

1 -2 3

10 20 30 40 50 60

4 5 6

3

0

1 -1

-100 11 890 11

u@DESKTOP-3U3OER0:/mnt/c/Users/u/CLionProjects/OS/lab_1$  ./a
commands.txt

Sum: 2

Sum: 210

Sum: 15

Sum: 3

Sum: 0

Sum: 0

Sum: 812

u@DESKTOP-3U3OER0:/mnt/c/Users/u/CLionProjects/OS/lab_1$ cat
commands.txt

1 1 1 1 1 1 1 1 1 1 1 1

10

88890 -88890

1 0

u@DESKTOP-3U3OER0:/mnt/c/Users/u/CLionProjects/OS/lab_1$ ./a
commands.txt

Sum: 13

Sum: 10

Sum: 0

Sum: 1
```

**Strace**

```
u@DESKTOP-3U3OER0:/mnt/c/Users/u/CLionProjects/OS/lab_1$ strace ./a
commands.txt

execve("./a", ["./a", "commands.txt"], 0x7ffd49b02b48 /* 25 vars */) = 0

brk(NULL)                               = 0x55c63e8da000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffea6bc1350) = -1 EINVAL (Invalid
argument)

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=57777, ...}) = 0

mmap(NULL, 57777, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe26713e000

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\300A\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) = 78

4

pread64(3,
"\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32,
848) = 32

pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\7\2C\n\357_\243\335\2449\206V>\237\374\3
04"..., 6

8, 880) = 68

fstat(3, {st_mode=S_IFREG|0755, st_size=2029592, ...}) = 0

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7fe26713c000

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) = 78

4

pread64(3,
"\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32,
848) = 32
```

```
pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\7\2C\n\357_\243\335\2449\206V>\237\374\3
04"..., 6

8, 880) = 68

mmap(NULL, 2037344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7fe266f4a000

mmap(0x7fe266f6c000, 1540096, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0

x22000) = 0x7fe266f6c000

mmap(0x7fe2670e4000, 319488, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19a000) =

0x7fe2670e4000

mmap(0x7fe267132000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x

1e7000) = 0x7fe267132000

mmap(0x7fe267138000, 13920, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0

) = 0x7fe267138000

close(3)                                = 0

arch_prctl(ARCH_SET_FS, 0x7fe26713d540) = 0

mprotect(0x7fe267132000, 16384, PROT_READ) = 0

mprotect(0x55c63d45f000, 4096, PROT_READ) = 0

mprotect(0x7fe26717a000, 4096, PROT_READ) = 0

munmap(0x7fe26713e000, 57777)           = 0

brk(NULL)                               = 0x55c63e8da000

brk(0x55c63e8fb000)                     = 0x55c63e8fb000

openat(AT_FDCWD, "commands.txt", O_RDONLY) = 3

pipe([4, 5])                            = 0

clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=

0x7fe26713d810) = 74

close(4)                                = 0

fstat(3, {st_mode=S_IFREG|0777, st_size=46, ...}) = 0

read(3, "1 1 1 1 1 1 1 1 1 1 1 1 1\n10\n888"..., 512) = 46

write(5, "1 1 1 1 1 1 1 1 1 1 1 1 1\n", 26) = 26
```

```
write(5, "10\n", 3)                       = 3

write(5, "88890 -88890\n", 13)           = 13

write(5, "1 0\n", 4)                      = 4

read(3, "", 512)                          = 0

close(5)                                  = 0

close(3)                                  = 0

wait4(-1, Sum: 13

Sum: 10

Sum: 0

Sum: 1

NULL, 0, NULL)                 = 74

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=74,
si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

exit_group(0)                             = ?

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

## Вывод

В ходе выполнения лабораторной работы я освоил навыки управления процессами в операционной системе и научился организовывать обмен данными между процессами с помощью каналов. В процессе выполнения работы я разработал и отладил программу на языке Си, обеспечивающую взаимодействие между процессами через передачу данных через каналы. Программа успешно выполняет передачу данных от родительского процесса к дочернему, корректно обрабатывает данные и выводит результаты. В ходе выполнения работы проблем не возникло.