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-209Б-24

Студент: Галич А.П.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 15.10.25

Москва, 2025

Постановка задачи

Вариант 3.

Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. 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:

pipe() — создаёт каналы для межпроцессного взаимодействия; создано два канала pipe1 и pipe2 для обмена данными между процессами

fork() — создаёт дочерний процесс; используется для разделения программы на родительский и дочерний процессы

dup2() — перенаправляет стандартные потоки; дочерний процесс связывает свой стандартный ввод с pipe1 и стандартный вывод с pipe2

execl() — заменяет образ процесса; дочерний процесс запускает программу child для обработки чисел и записи результатов в файл

waitpid() — отслеживает состояние дочернего процесса;

close() — закрывает файловые дескрипторы; освобождает ресурсы каналов и предотвращает утечки

exit() — завершает выполнение процесса; используется при ошибках и нормальном завершении работы

write() — записывает данные в канал; родительский процесс отправляет строки с числами через pipe1 дочернему процессу

fgets() — читает строки из стандартного ввода; родительский процесс получает от пользователя имя файла и числа для обработки

Алгоритм:

Сначала создаются два канала - pipe1 для передачи данных от родителя к дочернему процессу и pipe2 для обратной связи. Пользователь вводит имя файла, в который будут записываться результаты вычислений, и программа проверяет возможность его создания. Затем с помощью системного вызова fork() создается дочерний процесс. В дочернем процессе происходит перенаправление стандартных потоков ввода-вывода: стандартный ввод связывается с pipe1 для чтения данных от родителя, а стандартный вывод - с pipe2 для возможной обратной связи. После этого выполняется загрузка программы child, которая будет обрабатывать поступающие данные. Родительский процесс принимает от пользователя строки, содержащие числа, разделенные пробелами, и передает их через pipe1 дочернему процессу. Одновременно родитель отслеживает состояние дочернего процесса с помощью waitpid(). Дочерний процесс читает поступающие строки, извлекает из них числа и выполняет последовательное деление первого числа на все последующие. При обнаружении деления на ноль дочерний процесс аварийно завершает работу. Результаты вычислений записываются в указанный файл. При нормальном завершении ввода или при ошибке деления на ноль оба процесса корректно закрывают файловые дескрипторы и завершают работу

Код программы

parent.c

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

int main() {
    int pipe1[2], pipe2[2];
    pid_t pid;
    char filename[100];
    char buffer[1024];

    printf("Введите имя файла: ");
    if (fgets(filename, sizeof(filename), stdin) == NULL) {
        printf("Ошибка ввода имени файла\n");
        exit(1);
    }

    filename[strcspn(filename, "\n")] = '\0';
```

```
    FILE* test_file = fopen(filename, "w");
    if (test_file == NULL) {
        printf("Ошибка: невозможно создать файл '%s'\n", filename);
        exit(1);
    }
    fclose(test_file);

    if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {
        printf("Ошибка создания каналов\n");
        exit(1);
    }

    pid = fork();
    if (pid == -1) {
        printf("Ошибка создания процесса\n");
        exit(1);
    }

    if (pid == 0) {
        close(pipe1[1]);
        close(pipe2[0]);

        dup2(pipe1[0], STDIN_FILENO);
        close(pipe1[0]);

        dup2(pipe2[1], STDOUT_FILENO);
        close(pipe2[1]);

        execl("./child", "child", filename, NULL);
        printf("Ошибка запуска дочерней программы\n");
        exit(1);
    } else {
        close(pipe1[0]);
        close(pipe2[1]);

        printf("Введите числа через пробел:\n");

        while (1) {
            if (fgets(buffer, sizeof(buffer), stdin) == NULL) {
                break;
            }

            write(pipe1[1], buffer, strlen(buffer));

            int status;
            pid_t result = waitpid(pid, &status, WNOHANG);
            if (result != 0) {
                printf("Дочерний процесс завершил работу\n");
                break;
            }
        }

        close(pipe1[1]);
```

```
        close(pipe2[0]);
        wait(NULL);
    }

    return 0;
}
```

child.c

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

int main(int argc, char* argv[]) {
    if (argc != 2) return 1;

    FILE* out = fopen(argv[1], "w");

    char* line = NULL;
    size_t len = 0;

    while (getline(&line, &len, stdin) != -1) {
        int first_num, num;
        int result;
        int count = 0;
        char* ptr = line;

        if (sscanf(ptr, "%d", &first_num) != 1) continue;

        result = first_num;
        count++;
        fprintf(out, "%d", first_num);

        while (*ptr && *ptr != ' ' && *ptr != '\n' && *ptr != '\t') ptr++;

        while (sscanf(ptr, "%d", &num) == 1) {
            fprintf(out, " / %d", num);

            if (num == 0) {
                fprintf(out, " - Деление на 0! Завершение работы.\n");
                fclose(out);
                free(line);
                exit(1);
            }

            result /= num;
            count++;

            while (*ptr && *ptr != ' ' && *ptr != '\n' && *ptr != '\t') ptr++;
            while (*ptr == ' ' || *ptr == '\t') ptr++;
        }
```

```
        if (count > 1) {
            fprintf(out, " = %d\n", result);
        } else {
            fprintf(out, "\n");
        }
        fflush(out);
    }

    free(line);
    fclose(out);
    return 0;
}
```

Протокол работы программы

kishaki@416:~/lab_OS1$ ./parent

Введите имя файла: result

Введите числа через пробел:

100 2 5 5

100 2 5

100 2

100 0

100 2

kishaki@416:~/lab_OS1$ cat result

100 / 2 / 2 / 5 / 5 = 1

100 / 2 / 2 / 5 = 5

100 / 2 / 2 = 25

100 / 0 - Деление на 0! Завершение работы.

Strace:

execve("./parent", ["./parent"], 0x7ffd238f4448 /* 29 vars */) = 0

brk(NULL) = 0x616ff6485000

mmap(NULL, 8192, PROT_READ|PROT_WRITE,

MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x73e1371b9000

```
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=19871, ...}) = 0
mmap(NULL, 19871, PROT_READ, MAP_PRIVATE, 3, 0) = 0x73e1371b4000
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) =
0x73e136e00000
mmap(0x73e136e28000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x73e136e28000
mmap(0x73e136fb0000, 323584, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x73e136fb0000
mmap(0x73e136fff000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x73e136fff000
mmap(0x73e137005000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x73e137005000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x73e1371b1000
arch_prctl(ARCH_SET_FS, 0x73e1371b1740) = 0
set_tid_address(0x73e1371b1a10)         = 94847
set_robust_list(0x73e1371b1a20, 24)     = 0
rseq(0x73e1371b2060, 0x20, 0, 0x53053053) = 0
mprotect(0x73e136fff000, 16384, PROT_READ) = 0
mprotect(0x616fd583e000, 4096, PROT_READ) = 0
mprotect(0x73e1371f1000, 8192, PROT_READ) = 0
```

```
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
munmap(0x73e1371b4000, 19871)           = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x4), ...}) = 0
getrandom("\x72\x0e\x1d\xa4\xbb\xe8\x81\x30", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x616ff6485000
brk(0x616ff64a6000)                     = 0x616ff64a6000
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x4), ...}) = 0
write(1, "\320\222\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"..., 34Введите имя
файла: ) = 34
read(0, result
"result\n", 1024)               = 7
openat(AT_FDCWD, "result", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
close(3)                                = 0
pipe2([3, 4], 0)                        = 0
pipe2([5, 6], 0)                        = 0
clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process
94860 attached
, child_tidptr=0x73e1371b1a10) = 94860
[pid 94860] set_robust_list(0x73e1371b1a20, 24 <unfinished ...>
[pid 94847] close(3 <unfinished ...>
[pid 94860] <... set_robust_list resumed>) = 0
[pid 94847] <... close resumed>)        = 0
[pid 94847] close(6 <unfinished ...>
[pid 94860] close(4 <unfinished ...>
[pid 94847] <... close resumed>)        = 0
[pid 94860] <... close resumed>)        = 0
[pid 94860] close(5 <unfinished ...>
```

[pid 94847] write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\207\320\270\321\201\320\273\320\260 \321\207\320\265\321\200"..., 51 <unfinished ...>

Введите числа через пробел:

[pid 94860] <... close resumed>)   = 0

[pid 94847] <... write resumed>)   = 51

[pid 94860] dup2(3, 0 <unfinished ...>

[pid 94847] read(0, <unfinished ...>

[pid 94860] <... dup2 resumed>)   = 0

[pid 94860] close(3)   = 0

[pid 94860] dup2(6, 1)   = 1

[pid 94860] close(6)   = 0

[pid 94860] execve("./child", ["child", "result"], 0x7ffd377692b8 /* 29 vars */) = 0

[pid 94860] brk(NULL)   = 0x5792563e1000

[pid 94860] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7bac81e75000

[pid 94860] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

[pid 94860] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

[pid 94860] fstat(3, {st_mode=S_IFREG|0644, st_size=19871, ...}) = 0

[pid 94860] mmap(NULL, 19871, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7bac81e70000

[pid 94860] close(3)   = 0

[pid 94860] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

[pid 94860] 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

[pid 94860] 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 94860] fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

[pid 94860] 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 94860] mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7bac81c00000

```
[pid 94860] mmap(0x7bac81c28000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7bac81c28000
[pid 94860] mmap(0x7bac81db0000, 323584, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7bac81db0000
[pid 94860] mmap(0x7bac81dff000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7bac81dff000
[pid 94860] mmap(0x7bac81e05000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7bac81e05000
[pid 94860] close(3)                    = 0
[pid 94860] mmap(NULL, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7bac81e6d000
[pid 94860] arch_prctl(ARCH_SET_FS, 0x7bac81e6d740) = 0
[pid 94860] set_tid_address(0x7bac81e6da10) = 94860
[pid 94860] set_robust_list(0x7bac81e6da20, 24) = 0
[pid 94860] rseq(0x7bac81e6e060, 0x20, 0, 0x53053053) = 0
[pid 94860] mprotect(0x7bac81dff000, 16384, PROT_READ) = 0
[pid 94860] mprotect(0x579227295000, 4096, PROT_READ) = 0
[pid 94860] mprotect(0x7bac81ead000, 8192, PROT_READ) = 0
[pid 94860] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
[pid 94860] munmap(0x7bac81e70000, 19871) = 0
[pid 94860] getrandom("\x0f\xe0\x16\x15\x6a\x60\xfa\xe8", 8, GRND_NONBLOCK) = 8
[pid 94860] brk(NULL)                   = 0x5792563e1000
[pid 94860] brk(0x579256402000)         = 0x579256402000
[pid 94860] openat(AT_FDCWD, "result", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
[pid 94860] fstat(0, {st_mode=S_IFIFO|0600, st_size=0, ...}) = 0
[pid 94860] read(0, 100 2 5 5
 <unfinished ...>
[pid 94847] <... read resumed>"100 2 5 5\n", 1024) = 10
[pid 94847] write(4, "100 2 5 5\n", 10) = 10
[pid 94847] wait4(94860,  <unfinished ...>
[pid 94860] <... read resumed>"100 2 5 5\n", 4096) = 10
```

```
[pid 94847] <... wait4 resumed>0x7ffd37768cdc, WNOHANG, NULL) = 0
[pid 94847] read(0,  <unfinished ...>
[pid 94860] fstat(3, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
[pid 94860] write(3, "100 / 2 / 2 / 5 / 5 = 1\n", 24) = 24
[pid 94860] read(0,  <unfinished ...>
[pid 94847] <... read resumed>"", 1024) = 0
[pid 94847] close(4)                    = 0
[pid 94860] <... read resumed>"", 4096) = 0
[pid 94847] close(5 <unfinished ...>
[pid 94860] close(3 <unfinished ...>
[pid 94847] <... close resumed>)        = 0
[pid 94860] <... close resumed>)        = 0
[pid 94847] wait4(-1,  <unfinished ...>
[pid 94860] exit_group(0)               = ?
[pid 94860] +++ exited with 0 +++
<... wait4 resumed>NULL, 0, NULL)       = 94860
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=94860, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

В результате выполнения лабораторной работы была разработана программа, организующая взаимодействие между процессами через механизм pipe-каналов. Родительский процесс отвечает за ввод данных от пользователя и их передачу дочернему процессу, который выполняет арифметические операции деления и записывает результаты в файл. При выполнении лабораторной работы наибольшую сложность вызвала организация корректного взаимодействия между процессами через pipe. Практическое применение системных вызовов fork(), pipe() и dup2() позволило лучше понять механизмы работы операционной системы с процессами и потоками данных.