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-216БВ-24

Студент: Гуськов А.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 23.09.25

Москва, 2025

Постановка задачи

Вариант 8.

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 *pipe*. Родительский процесс читает из *pipe* 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 команды, 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 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 межпроцессного взаимодействия.
- *int execl(const char* path, const char* arg, ...)*; – заменяет образ текущей программы, на указанную, принимая аргументы в качестве списка.
- *int dup2(int oldfd, int newfd)*; – создает копию файлового дескриптора *oldfd* в указанном дескрипторе *newfd*.
- *int open(const char* pathname, int flags, mode_t mode)*; – открывает файл по указанному пути с заданными флагами и правами доступа.
- *ssize_t read(int fd, void* buf, size_t count)*; – читает данные из файлового дескриптора в буфер.
- *ssize_t write(int fd, const void* buf, size_t count)*; – записывает данные из буфера в файловый дескриптор.
- *int close(int fd)*; – закрывает файловый дескриптор.
- *pid_t wait(int* status)*; – ожидает изменения состояния указанного дочернего процесса.

В рамках лабораторной работы создавалась программа, которая создает дочерний процесс (с помощью системного вызова *fork()*), подменяет образ программы (с помощью системного вызова *execl()*). Предварительно создается канал межпроцессного взаимодействия (с помощью системного вызова *pipe()*).

Родительский процесс запрашивает у пользователя название файла, который содержит команды для обработки. Дочерний процесс перенаправляет свой стандартный ввод на содержимое указанного файла с помощью *dup2*, а стандартный вывод - в созданный канал. После этого дочерний процесс запускает программу *child.c*, которая читает команды из файла в формате "число число число<*endline*>", выполняет деление первого числа на последующие числа в строке и выводит результаты. Родительский процесс читает результаты из канала и выводит их в свой стандартный вывод. При обнаружении деления на ноль дочерний процесс завершается с ошибкой, что приводит к завершению родительского процесса.

Код программы

main.c

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


int main(int argc, char** argv)
{
  // We write the file name to filename
  char filename[1024];
  {
     const char message[] = "Input filename: ";
     write(STDOUT_FILENO, message, sizeof(message));

     int bytes = read(STDIN_FILENO, filename, sizeof(filename) - 1);
     if (bytes <= 0)
     {
        const char err_msg[] = "error: Impossible to read from user\n";
        write(STDERR_FILENO, err_msg, sizeof(err_msg));
        exit(EXIT_FAILURE);
     }
     filename[bytes - 1] = '\0';
  }

  // Creating a pipe for interprocess communication
  int file_descriptor[2];
  if (pipe(file_descriptor) == -1)
  {
     const char err_msg[] = "error: Impossible to create pipe\n";
     write(STDERR_FILENO, err_msg, sizeof(err_msg));
     exit(EXIT_FAILURE);
  }

  pid_t child = fork();
  switch (child)
  {
     case -1:
     { // We cannot spawn a child process
        const char err_msg[] = "error: impossible to spawn new process\n";
        write(STDERR_FILENO, err_msg, sizeof(err_msg));
        exit(EXIT_FAILURE);
     } break;

     case 0:
     { // We're in child process
        close(file_descriptor[0]);
        dup2(file_descriptor[1], STDOUT_FILENO);
        close(file_descriptor[1]);

        int file = open(filename, O_RDONLY);
        if (file == -1)
```

```
    {
        const char err_msg[] = "error: Error opening file\n";
        write(STDERR_FILENO, err_msg, sizeof(err_msg));
        exit(EXIT_FAILURE);
    }

    // Duplicate the file descriptor as an input stream for the child process
    dup2(file, STDIN_FILENO);
    close(file);

    int status = execl("./child", "child", NULL);

    if (status == -1) {
        const char err_msg[] = "error: failed to exec into new exectuable image\n";
        write(STDERR_FILENO, err_msg, sizeof(err_msg));
        exit(EXIT_FAILURE);
    }
}

default:
{ // We are in parent's process
    close(file_descriptor[1]);


    // Read output from child process
    char buff[1024];
    int status;
    int bytes;
    while ((bytes = read(file_descriptor[0], buff, sizeof(buff))) > 0)
    {
        write(STDOUT_FILENO, buff, bytes);
    }

    close(file_descriptor[0]);

    wait(&status);

    // Checking the exit status of a child process
    if (WIFEXITED(status))
    {
        if (WEXITSTATUS(status) != 0)
        {
            const char err_msg[] = "error: the child process terminated with an error\n";
            write(STDERR_FILENO, err_msg, sizeof(err_msg));
            exit(EXIT_FAILURE);
        }
        else
        {
            exit(EXIT_SUCCESS);
        }
    }
    else
    {
        const char err_msg[] = "error: The process has not completed its execution\n";
        write(STDERR_FILENO, err_msg, sizeof(err_msg));
```

```
            exit(EXIT_FAILURE);
        }
    }
  }
}
```

child.c

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


int string_to_int(const char *str)
{
  int result = 0;
  int sign = 1;
  int i = 0;

  if (str[0] == '-')
  {
    sign = -1;
    i = 1;
  }

  while (str[i] != '\0')
  {
    if (str[i] >= '0' && str[i] <= '9')
    {
      result = result * 10 + (int)(str[i] - '0');
    }
    i++;
  }

  return result * sign;
}


void int_to_string(int num, char *str)
{
  int i = 0;
  int is_negative = 0;

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

  do
  {
    str[i++] = num % 10 + '0';
    num /= 10;
  } while (num > 0);

  if (is_negative)
  {
    str[i++] = '-';
```

```
    }

    str[i] = '\0';

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


int main(int argc, char** argv)
{
    char buff[1024];
    ssize_t bytes;

    while ((bytes = read(STDIN_FILENO, buff, sizeof(buff) - 1)) > 0)
    {
        int numbers[99];    // Numbers storage
        int count = 0;      // Counter of the quantity of numbers
        int ptr = 0;        // Ptr to the first character of the number

        for (int i = 0; i < bytes; ++i)
        {
            if (buff[i] == ' ' || buff[i] == '\n' || buff[i] == '\t')
            {
                if (i > ptr)
                {
                    char number[20];
                    int number_length = i - ptr;
                    for (int j = 0; j < number_length; ++j)
                    {
                        number[j] = buff[ptr + j];
                    }
                    number[number_length] = '\0';

                    numbers[count++] = string_to_int(number);
                }
                ptr = i + 1;

                if (buff[i] == '\n' && count > 0)
                {
                    if (count < 2)
                    {
                        const char err_msg[] = "error: Not enough arguments passed\n";
                        write(STDERR_FILENO, err_msg, sizeof(err_msg));
                    }
                    else
                    {
```

```
                int result = numbers[0];

                for (int j = 1; j < count; j++)
                {
                    if (numbers[j] == 0)
                    {
                        const char err_msg[] = "error: Division by zero\n";
                        write(STDERR_FILENO, err_msg, sizeof(err_msg));
                        exit(EXIT_FAILURE);
                    }
                    result /= numbers[j];
                }
                char result_str[20];

                char output[50];
                int pos = 0;

                int_to_string(result, result_str);
                for (int k = 0; result_str[k] != '\0'; k++)
                {
                    output[pos++] = result_str[k];
                }
                output[pos++] = '\n';

                write(STDOUT_FILENO, output, pos);
            }

            count = 0;
        }
    }
  }

}
```

Протокол работы программы

Тестирование:

```
~/Projects/MAI-OS-LW/lab1/src main ?1 gcc child.c -o child
~/Projects/MAI-OS-LW/lab1/src main ?1 gcc main.c -o main
~/Projects/MAI-OS-LW/lab1/src main ?1 ./create_test.sh
Тестовые файлы созданы!
~/Projects/MAI-OS-LW/lab1/src main ?1 ls
child           main                    test_edge_cases.txt     test_large_numbers.txt   test_negative.txt    test_only_spaces.txt
child.c         main.c                  test_empty.txt          test_mixed_spaces.txt    test_no_newline.txt  test_single_number.txt
create_test.sh  test_division_by_zero.txt  test_invalid_chars.txt  test_multiple_lines.txt  test_normal.txt
~/Projects/MAI-OS-LW/lab1/src main ?1 ./main
Input filename: test_normal.txt
1
2
1
4
1
~/Projects/MAI-OS-LW/lab1/src main ?1 ./main
Input filename: test_division_by_zero.txt
error: Division by zero
1
error: the child process terminated with an error
~/Projects/MAI-OS-LW/lab1/src main ?1 ./main
Input filename: test_empty.txt
~/Projects/MAI-OS-LW/lab1/src main ?1 ./main
Input filename: test_only_spaces.txt
1
5
~/Projects/MAI-OS-LW/lab1/src main ?1 ./main
Input filename: test_large_numbers.txt
357913941
-107374182
100000
~/Projects/MAI-OS-LW/lab1/src main ?1 ./main
Input filename: test_single_number.txt
error: Not enough arguments passed
error: Not enough arguments passed
25
1
~/Projects/MAI-OS-LW/lab1/src main ?1
```

Strace:

```
$ strace -f ./main

execve("./main", ["./main"], 0x7ffdef5f41f8 /* 63 vars */) = 0

brk(NULL)                                = 0x5563b497a000

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=161167, ...}) = 0

mmap(NULL, 161167, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fbd40df6000

close(3)                                 = 0

openat(AT_FDCWD, "/usr/lib/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\0000x\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\0"..., 896, 64)
= 896

fstat(3, {st_mode=S_IFREG|0755, st_size=2149728, ...}) = 0

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fbd40df4000

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\0"..., 896, 64)
= 896

mmap(NULL, 2174000, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fbd40a00000
```

```
     mmap(0x7fbd40a24000, 1515520, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x24000) = 0x7fbd40a24000

     mmap(0x7fbd40b96000, 454656, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x196000) = 0x7fbd40b96000

     mmap(0x7fbd40c05000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x204000) = 0x7fbd40c05000

     mmap(0x7fbd40c0b000, 31792, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7fbd40c0b000

     close(3)                                = 0

     mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fbd40df1000

     arch_prctl(ARCH_SET_FS, 0x7fbd40df1740) = 0

     set_tid_address(0x7fbd40df1a10)         = 17692

     set_robust_list(0x7fbd40df1a20, 24)     = 0

     rseq(0x7fbd40df1680, 0x20, 0, 0x53053053) = 0

     mprotect(0x7fbd40c05000, 16384, PROT_READ) = 0

     mprotect(0x556390a77000, 4096, PROT_READ) = 0

     mprotect(0x7fbd40e5d000, 8192, PROT_READ) = 0

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

     getrandom("\xfe\x79\xf9\x84\xad\x00\xfe\xbc", 8, GRND_NONBLOCK) = 8

     munmap(0x7fbd40df6000, 161167)          = 0

     write(1, "Input filename: \0", 17Input filename: )      = 17

     read(0test_normal.txt

     , "test_normal.txt\n", 1023)      = 16

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

     rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

     clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace:
Process 17701 attached

     , child_tidptr=0x7fbd40df1a10) = 17701

     [pid 17692] rt_sigprocmask(SIG_SETMASK, [] <unfinished ...>

     [pid 17701] set_robust_list(0x7fbd40df1a20, 24) = 0

     [pid 17692] <... rt_sigprocmask resumed>, NULL, 8) = 0

     [pid 17692] close(4 <unfinished ...>

     [pid 17701] rt_sigprocmask(SIG_SETMASK, [] <unfinished ...>

     [pid 17692] <... close resumed>)        = 0

     [pid 17701] <... rt_sigprocmask resumed>, NULL, 8) = 0

     [pid 17692] read(3 <unfinished ...>

     [pid 17701] close(3)                    = 0
```

[pid 17701] dup2(4, 1)   = 1

[pid 17701] close(4)   = 0

[pid 17701] openat(AT_FDCWD, "test_normal.txt", O_RDONLY) = 3

[pid 17701] dup2(3, 0)   = 0

[pid 17701] close(3)   = 0

[pid 17701] execve("./child", ["child"], 0x7ffff953b448 /* 63 vars */) = 0

[pid 17701] brk(NULL)   = 0x5575800d0000

[pid 17701] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

[pid 17701] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

[pid 17701] fstat(3, {st_mode=S_IFREG|0644, st_size=161167, ...}) = 0

[pid 17701] mmap(NULL, 161167, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f98bcb54000

[pid 17701] close(3)   = 0

[pid 17701] openat(AT_FDCWD, "/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

[pid 17701] read(3,
"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0000x\2\0\0\0\0\0"..., 832) = 832

[pid 17701] 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"..., 896, 64) = 896

[pid 17701] fstat(3, {st_mode=S_IFREG|0755, st_size=2149728, ...}) = 0

[pid 17701] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f98bcb52000

[pid 17701] 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"..., 896, 64) = 896

[pid 17701] mmap(NULL, 2174000, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f98bc800000

[pid 17701] mmap(0x7f98bc824000, 1515520, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x24000) = 0x7f98bc824000

[pid 17701] mmap(0x7f98bc996000, 454656, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x196000) = 0x7f98bc996000

[pid 17701] mmap(0x7f98bca05000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x204000) = 0x7f98bca05000

[pid 17701] mmap(0x7f98bca0b000, 31792, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f98bca0b000

[pid 17701] close(3)   = 0

[pid 17701] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f98bcb4f000

[pid 17701] arch_prctl(ARCH_SET_FS, 0x7f98bcb4f740) = 0

[pid 17701] set_tid_address(0x7f98bcb4fa10) = 17701

[pid 17701] set_robust_list(0x7f98bcb4fa20, 24) = 0

[pid 17701] rseq(0x7f98bcb4f680, 0x20, 0, 0x53053053) = 0

[pid 17701] mprotect(0x7f98bca05000, 16384, PROT_READ) = 0

```
[pid 17701] mprotect(0x55754411f000, 4096, PROT_READ) = 0
[pid 17701] mprotect(0x7f98bcbbb000, 8192, PROT_READ) = 0
[pid 17701] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
[pid 17701] getrandom("\x3d\x8e\x1a\xfe\xc9\xdf\xcd\xbf", 8, GRND_NONBLOCK) = 8
[pid 17701] munmap(0x7f98bcb54000, 161167) = 0
[pid 17701] read(0, "12 3 4\n20 5 2\n100 10 2 5\n8 2\n15 "..., 1023) = 36
[pid 17701] write(1, "1\n", 2)          = 2
[pid 17692] <... read resumed>, "1\n", 1024) = 2
[pid 17701] write(1, "2\n", 2 <unfinished ...>
[pid 17692] write(1, "1\n", 2 <unfinished ...>
1
[pid 17701] <... write resumed>)        = 2
[pid 17692] <... write resumed>)        = 2
[pid 17701] write(1, "1\n", 2 <unfinished ...>
[pid 17692] read(3 <unfinished ...>
[pid 17701] <... write resumed>)        = 2
[pid 17692] <... read resumed>, "2\n1\n", 1024) = 4
[pid 17701] write(1, "4\n", 2 <unfinished ...>
[pid 17692] write(1, "2\n1\n", 4 <unfinished ...>
2
1
[pid 17701] <... write resumed>)        = 2
[pid 17692] <... write resumed>)        = 4
[pid 17701] write(1, "1\n", 2 <unfinished ...>
[pid 17692] read(3 <unfinished ...>
[pid 17701] <... write resumed>)        = 2
[pid 17692] <... read resumed>, "4\n1\n", 1024) = 4
[pid 17701] read(0 <unfinished ...>
[pid 17692] write(1, "4\n1\n", 4 <unfinished ...>
4
1
[pid 17701] <... read resumed>, "", 1023) = 0
[pid 17692] <... write resumed>)        = 4
[pid 17692] read(3 <unfinished ...>
[pid 17701] exit_group(0)               = ?
```

```
[pid 17692] <... read resumed>, "", 1024) = 0

[pid 17701] +++ exited with 0 +++

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=17701, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---

close(3)                                = 0

wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 17701

exit_group(0)                           = ?

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

Вывод

В ходе выполнения лабораторной работы были успешно изучены и применены основные системные вызовы для работы с процессами и межпроцессным взаимодействием в ОС Linux. Была реализована программа, демонстрирующая создание процессов, организацию каналов связи между ними и перенаправление стандартных потоков ввода-вывода.

Столкнулся с проблемами корректной обработки входных данных, особенно при работе с различными форматами пробелов и разделителей. Решил эти проблемы путем реализации собственных функций для разбора строк и преобразования чисел. Также возникли сложности с правильным закрытием файловых дескрипторов для избежания утечек ресурсов.