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: ГенниховА.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата:15.10.25

Москва, 2025

Постановка задачи

Вариант 7.

В файле записаны команды вида: «число число число<endline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. 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 (pipe) для передачи данных, заполняя pipefd дескрипторами для чтения ([0]) и записи ([1]).
- int dup2(int oldfd, int newfd); – дублирует файловый дескриптор oldfd в дескриптор newfd, закрывая newfd, если он был открыт. Используется для перенаправления ввода/вывода.
- int execv(const char *pathname, char *const argv[]); – заменяет текущий образ процесса новым, загружая программу из pathname с аргументами argv.
- pid_t wait(int *wstatus); – приостанавливает выполнение родительского процесса до завершения дочернего процесса и получает его статус завершения.
- ssize_t read(int fd, void *buf, size_t count); – читает до count байт из файлового дескриптора fd в буфер buf.
- ssize_t write(int fd, const void *buf, size_t count); – записывает до count байт из буфера buf в файловый дескриптор fd.
- int open(const char *pathname, int flags); – открывает файл, указанный в pathname, в заданном режиме (flags, например, O_RDONLY).
- int close(int fd); – закрывает файловый дескриптор fd, освобождая системные ресурсы.
- void exit(int status); – немедленно завершает текущий процесс с кодом возврата status.

Программа parent.c запрашивает у пользователя имя файла, открывает его, после чего вызывает системный вызов pipe() для создания канала. Сразу после этого fork() порождает дочерний процесс. В родительской ветви процесса закрывается ненужный дескриптор для чтения из канала, и затем родитель поблочно считывает содержимое исходного файла (read) и записывает его в канал(write). После полной передачи файла родитель закрывает дескриптор записи, сигнализируя дочернему процессу об окончании ввода, и ожидает (wait) завершения работы потомка, чтобы получить его код выхода.

Программа child.c выполняет роль вычислителя. Сразу после fork() в дочерней ветви процесса она использует dup2() для перенаправления своего стандартного ввода (STDIN_FILENO) на конец канала для чтения, который был создан родителем. Затем с помощью execv() она заменяет свой код на исполняемый файл ./lab_01_child. Далее дочерний процесс построчно считывает данные из канала. Каждая строка обрабатывается функцией process_line, которая извлекает целые числа, проверяет, что их не менее двух, вычисляет их сумму (используя int64_t для предотвращения переполнения) и выводит результат на свой стандартный вывод. Дочерний процесс завершается с ошибкой, если строка некорректна.

Код программы

parent.c

```
#include <stdint.h>

#include <stdbool.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/wait.h>

#include <string.h>



char
CHILD_PROGRAM
_NAME[] = "./child";



int main()

{

   char
filename[256];

   {

      const char
prompt[] = "Enter
filename: ";

      write(STDOUT_
FILENO, prompt,
```

```
sizeof(prompt) - 1);


    ssize_t bytes =
read(STDIN_FILEN
O, filename,
sizeof(filename) - 1);
    if (bytes <= 0)
    {
        const char
msg[] = "error: failed
to read filename\n";
        write(STDER
R_FILENO, msg,
sizeof(msg) - 1);
        exit(EXIT_F
AILURE);
    }


    if
(filename[bytes - 1]
== '\n')
    {
        filename[byte
s - 1] = '\0';
    }
```

```
        else

        {

            filename[byte
s] = '\0';

        }

    }



    int32_t file =
open(filename,
O_RDONLY);

    if (file == -1)

    {

        const char msg[]
= "error: failed to
open file\n";

        write(STDERR_
FILENO, msg,
sizeof(msg) - 1);

        exit(EXIT_FAIL
URE);

    }



    int
parent_to_child[2];
```

```
    if
(pipe(parent_to_child
) == -1)
    {
        const char msg[]
= "error: failed to
create pipe\n";
        write(STDERR_
FILENO, msg,
sizeof(msg) - 1);
        close(file);
        exit(EXIT_FAIL
URE);
    }


    const pid_t
child_pid = fork();


    switch (child_pid)
    {
    case -1:
        {
            const char
msg[] = "error: failed
```

```
to spawn new
process\n";
        write(STDER
R_FILENO, msg,
sizeof(msg) - 1);
        close(file);
        close(parent_t
o_child[0]);
        close(parent_t
o_child[1]);
        exit(EXIT_F
AILURE);
    }
    break;

  case 0:
    {
        close(parent_t
o_child[1]);
        close(file);
        if
(dup2(parent_to_chil
d[0],
STDIN_FILENO) ==
-1)
```

```
        {
            const char
msg[] = "error: failed
to redirect stdin\n";
            write(STD
ERR_FILENO, msg,
sizeof(msg) - 1);
            exit(EXIT_
FAILURE);
        }
        close(parent_t
o_child[0]);

        char* const
args[] = {"child",
NULL};
        execv(CHILD
_PROGRAM_NAM
E, args);

        const char
msg[] = "error: failed
to exec into child
program\n";
        write(STDER
R_FILENO, msg,
```

```
sizeof(msg) - 1);
            exit(EXIT_F
AILURE);
        }
        break;

    default:
        {
            close(parent_t
o_child[0]);

            char
buf[4096];
            ssize_t bytes;

            while ((bytes
= read(file, buf,
sizeof(buf))) > 0)
            {
                int32_t
written =
write(parent_to_child
[1], buf, bytes);
```

```
        if (written
!= bytes)
        {
            const
char msg[] = "error:
failed to write to
pipe\n";
            write(ST
DERR_FILENO,
msg, sizeof(msg) - 1);
            close(file
);
            close(par
ent_to_child[1]);
            wait(NU
LL);
            exit(EXI
T_FAILURE);
        }
    }

    if (bytes < 0)
    {
        const char
msg[] = "error: failed
```

```
to read from file\n";
            write(STD
ERR_FILENO, msg,
sizeof(msg) - 1);
        }

        close(file);
        close(parent_t
o_child[1]);

        int status;
        wait(&status);

        if
(WIFEXITED(status)
)
        {
            exit(WEXI
TSTATUS(status));
        }
        else
        {
            const char
```

```
msg[] = "error: child
terminated
abnormally\n";

            write(STD
ERR_FILENO, msg,
sizeof(msg) - 1);

            exit(EXIT_
FAILURE);

          }

       }

       break;

   }



   return 0;

}
```

child.c

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

```
#include <float.h>

#define MAX_NUMBERS 100
#define MAX_NUM_LENGTH 64
#define BUFFER_SIZE 4096

bool parse_float(const char* str, float* result)
{
    char* endptr;
    float value = strtof(str, &endptr);



    if (endptr == str)
    {
        return false;
    }

    while (*endptr != '\0')
    {
        if (!isspace(*endptr))
        {
            return false;
        }
        endptr++;
    }

    if (value == HUGE_VALF || value == -HUGE_VALF || value == 0.0f)
    {
        return true;
    }
```

```
    *result = value;
    return true;
}

int32_t float_to_string(float num, char* str, int precision)
{
    if (isnan(num))
    {
        return snprintf(str, MAX_NUM_LENGTH, "nan");
    }

    if (isinf(num))
    {
        if (num > 0)
            return snprintf(str, MAX_NUM_LENGTH, "inf");
        else
            return snprintf(str, MAX_NUM_LENGTH, "-inf");
    }


    return snprintf(str, MAX_NUM_LENGTH, "%.*f", precision, num);
}

void process_line(const char* line, int32_t length)
{
    float numbers[MAX_NUMBERS];
    int32_t count = 0;

    char num_buffer[MAX_NUM_LENGTH];
    int32_t num_index = 0;
```

```
for (int32_t i = 0; i < length; ++i)
{
    char c = line[i];

    if ((c >= '0' && c <= '9') ||
        c == '-' || c == '+' ||
        c == '.' ||
        c == 'e' || c == 'E')
    {
        if (num_index < MAX_NUM_LENGTH - 1)
        {
            num_buffer[num_index++] = c;
        }
    }
    else if (isspace(c))
    {
        if (num_index > 0)
        {
            num_buffer[num_index] = '\0';
            float num;
            if (parse_float(num_buffer, &num) && count < MAX_NUMBERS)
            {
                numbers[count++] = num;
            }
            num_index = 0;
        }
    }
}

if (num_index > 0)
{
```

```
            num_buffer[num_index] = '\0';
            float num;
            if (parse_float(num_buffer, &num) && count < MAX_NUMBERS)
            {
                numbers[count++] = num;
            }
        }

        float sum = 0.0f;
        for (int32_t i = 0; i < count; ++i)
        {
            sum += numbers[i];
        }

        char result[64];
        int32_t len = float_to_string(sum, result, 6);

        char output[128];
        int32_t output_len = snprintf(output, sizeof(output),
                                      "Sum of %d numbers: %s\n", count, result);

        write(STDOUT_FILENO, output, output_len);
    }

    int main(void)
    {
        char buffer[BUFFER_SIZE];
        char line[BUFFER_SIZE];
        int32_t line_length = 0;
        ssize_t bytes;
```

```
int32_t line_number = 0;


while ((bytes = read(STDIN_FILENO, buffer, sizeof(buffer))) > 0)
{
   for (int32_t i = 0; i < bytes; ++i)
   {
      if (buffer[i] == '\n')
      {
         if (line_length > 0)
         {
            line_number++;
            char line_header[32];
            int32_t header_len = snprintf(line_header, sizeof(line_header),
                                          "Line %d: ", line_number);
            write(STDOUT_FILENO, line_header, header_len);


            process_line(line, line_length);
            line_length = 0;
         }
      }
      else
      {
         if (line_length < BUFFER_SIZE - 1)
         {
            line[line_length++] = buffer[i];
         }
      }
   }
}


if (bytes < 0)
```

```
        {
            const char msg[] = "error: failed to read from stdin\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);
            exit(EXIT_FAILURE);
        }



        return 0;
    }
```

Протокол работы программы

Тестирование:

```
10.5 20.3 30.7
-5.2 15.8 25.1 -35.6
100.0 200.5 300.75
1.1 2.2 3.3 4.4 5.5
3.14e-2 2.718 1.414


Line 1: Sum of 3 numbers: 61.500000
Line 2: Sum of 4 numbers: 0.100002
Line 3: Sum of 3 numbers: 601.250000
Line 4: Sum of 5 numbers: 16.500000
Line 5: Sum of 3 numbers: 4.163400
```

Strace:

```
Enter filename: 1356/0x35c2:  fork()          = 0 0
 1356/0x35c2:  open(".\0", 0x100000, 0x0)             = 3 0
 1356/0x35c2:  close(0x3)             = 0 0
 1356/0x35c2:  open("/\0", 0x20100000, 0x0)           = 3 0
 1356/0x35c2:  close(0x3)             = 0 0
 1356/0x35c2:  close(0x5)             = 0 0
 1356/0x35c2:  close(0x4)             = 0 0
 1356/0x35c2:  close(0x6)             = 0 0
 1356/0x35c2:  close(0x8)             = 0 0
 1356/0x35c2:  close(0x7)             = 0 0
 1356/0x35c2: open("/Users/kirill_bel/Desktop/programming/CLion/CLionProjects/OS_MAI_2025/cmake-build-debug/lab_01/lab_01_parent\0", 0x0, 0x0)       = 3 0
 1356/0x35c2:  close(0x3)             = 0 0
 1356/0x35c2: open("/Users/kirill_bel/Desktop/programming/CLion/CLionProjects/OS_MAI_2025/cmake-build-debug/lab_01/lab_01_parent\0", 0x0, 0x0)       = 3 0
 1356/0x35c2:  close(0x3)             = 0 0
 1356/0x35c2:  open("/dev/dtracehelper\0", 0x2, 0x0)          = 3 0
 1356/0x35c2:  close(0x3)             = 0 0
 1356/0x35c2:  shm_open(0x1849D0F29, 0x0, 0x10)         = 3 0
 1356/0x35c2:  close(0x3)             = 0 0
 1356/0x35c2: open("/Users/kirill_bel/Desktop/programming/CLion/CLionProjects/OS_MAI_2025/cmake-build-debug/lab_01\0", 0x0, 0x0)             = 3 0
 1356/0x35c2:  close(0x3)             = 0 0
 1356/0x35c2: open("/Users/kirill_bel/Desktop/programming/CLion/CLionProjects/OS_MAI_2025/cmake-build-debug/lab_01/Info.plist\0", 0x0, 0x0)         = -1 Err#2
 1356/0x35c2:  write(0x1, "Enter filename: \0", 0x10)          = 16 0
 1356/0x35c2:  read(0x0, "test.txt\n\0", 0xFF)          = 9 0
 1356/0x35c2:  open("test.txt\0", 0x0, 0x0)           = 3 0
 1356/0x35c2:  pipe(0x0, 0x0, 0x0)            = 4 0
 1356/0x35c2:  fork()          = 1381 0
 1381/0x38fc:  fork()          = 0 0
 1356/0x35c2:  close(0x4)             = 0 0
 1356/0x35c2:  read(0x3, "10 20 30\n5 15\n1 2 3 4 5\n\0", 0x1000)             = 24 0
 1356/0x35c2:  write(0x5, "10 20 30\n5 15\n1 2 3 4 5\n\0", 0x18)              = 24 0
 1356/0x35c2:  read(0x3, "\0", 0x1000)          = 0 0
 1356/0x35c2:  close(0x3)             = 0 0
 1356/0x35c2:  close(0x5)             = 0 0
 1381/0x38fc:  close(0x5)             = 0 0
 1381/0x38fc:  close(0x3)             = 0 0
 1381/0x38fc:  dup2(0x4, 0x0, 0x0)            = 0 0
 1381/0x38fc:  close(0x4)             = 0 0
 1381/0x38fd:  fork()          = 0 0
 1381/0x38fd:  open(".\0", 0x100000, 0x0)             = 3 0
 1381/0x38fd:  close(0x3)             = 0 0
 1381/0x38fd:  open("/\0", 0x20100000, 0x0)           = 3 0
 1381/0x38fd:  close(0x3)             = 0 0
 1381/0x38fd:  close(0x5)             = 0 0
 1381/0x38fd:  close(0x4)             = 0 0
 1381/0x38fd:  close(0x6)             = 0 0
 1381/0x38fd:  close(0x8)             = 0 0
 1381/0x38fd:  close(0x7)             = 0 0
 1381/0x38fd: open("/Users/kirill_bel/Desktop/programming/CLion/CLionProjects/OS_MAI_2025/cmake-build-debug/lab_01/lab_01_child\0", 0x0, 0x0)       = 3 0
```

```
 1381/0x38fd:  close(0x3)                = 0 0
 1381/0x38fd: open("/Users/kirill_bel/Desktop/programming/CLion/CLionProjects/OS_MAI_2025/cmake-build-debug/lab_01/lab_01_child\0", 0x0, 0x0)         = 3 0
 1381/0x38fd:  close(0x3)                = 0 0
 1381/0x38fd:  open("/dev/dtracehelper\0", 0x2, 0x0)             = 3 0
 1381/0x38fd:  close(0x3)                = 0 0
 1381/0x38fd:  shm_open(0x1849D0F29, 0x0, 0x6DBFF6C0)            = 3 0
 1381/0x38fd:  close(0x3)                = 0 0
 1381/0x38fd: open("/Users/kirill_bel/Desktop/programming/CLion/CLionProjects/OS_MAI_2025/cmake-build-debug/lab_01\0", 0x0, 0x0)                = 3 0
 1381/0x38fd:  close(0x3)                = 0 0
 1381/0x38fd: open("/Users/kirill_bel/Desktop/programming/CLion/CLionProjects/OS_MAI_2025/cmake-build-debug/lab_01/Info.plist\0", 0x0, 0x0)          = -1 Err#2
 1381/0x38fd:  read(0x0, "10 20 30\n5 15\n1 2 3 4 5\n\0", 0x1000)                = 24 0
 1381/0x38fd:  write(0x1, "60\n\0", 0x3)                = 3 0
 1381/0x38fd:  write(0x1, "20\n\0", 0x3)                = 3 0
 1381/0x38fd:  write(0x1, "15\n\0", 0x3)                = 3 0
 1381/0x38fd:  read(0x0, "\0", 0x1000)          = 0 0
 1356/0x35c2:  wait4(0xFFFFFFFFFFFFFFFF, 0x16D4ADF28, 0x0)               = 1381 0
```

Вывод

В ходе выполнения лабораторной работы были успешно изучены и применены системные вызовы для организации межпроцессного взаимодействия. Была реализована программа, демонстрирующая создание дочернего процесса, организацию канала связи для передачи данных и перенаправление стандартного потока ввода (dup2) дочернего процесса.

Основная сложность возникла на этапе отладки и анализа системных вызовов. Поскольку работа выполнялась на macOS, стандартная утилита strace была недоступна. Для использования ее аналога (dtruss) потребовалось выполнить отключение защиты целостности системы (SIP), что позволило успешно провести трассировку и проанализировать вызовы программы.