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-216Б-24

Студент: Седов М. А

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 19.11.25

Москва, 2025

Постановка задачи

Вариант 2.

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.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип float. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

● pid_t fork(void) – создает дочерний процесс. Возвращает PID дочернего процесса в родительском процессе, 0 в дочернем процессе, или -1 в случае ошибки.

● int pipe(int pipefd[2]) – создает неименованный канал (pipe) для передачи данных между процессами. Массив `pipefd` содержит два файловых дескриптора: `pipefd[0]` для чтения и `pipefd[1]` для записи. Возвращает 0 при успехе, -1 при ошибке.

● int dup2(int oldfd, int newfd) – переназначает файловый дескриптор. Копирует `oldfd` в `newfd`, закрывая `newfd` если он был открыт. Используется для перенаправления стандартных потоков ввода/вывода. Возвращает новый дескриптор при успехе, -1 при ошибке.

● int execv(const char *pathname, char *const argv[]) – заменяет образ памяти текущего процесса новым исполняемым файлом. `pathname` – путь к исполняемому файлу, `argv` – массив аргументов командной строки (должен заканчиваться NULL). При успехе не возвращает управление, при ошибке возвращает -1.

● ssize_t read(int fd, void *buf, size_t count) – читает данные из файлового дескриптора. Возвращает количество прочитанных байт, 0 при достижении конца файла, -1 при ошибке.

● ssize_t write(int fd, const void *buf, size_t count) – записывает данные в файловый дескриптор. Возвращает количество записанных байт, -1 при ошибке.

● int open(const char *pathname, int flags, mode_t mode) – открывает или создает файл. flags определяет режим доступа (O_WRONLY, O_CREAT, O_TRUNC и т.д.), mode – права доступа при создании файла. Возвращает файловый дескриптор при успехе, -1 при ошибке.

● int close(int fd) – закрывает файловый дескриптор. Освобождает ресурсы, связанные с дескриптором. Возвращает 0 при успехе, -1 при ошибке.

● pid_t waitpid(pid_t pid, int *status, int options) – ожидает завершения дочернего процесса. `pid` – PID дочернего процесса, `status` – указатель для сохранения статуса завершения, `options` – опции ожидания. Возвращает PID завершенного процесса при успехе, -1 при ошибке.

● ssize_t readlink(const char *pathname, char *buf, size_t bufsiz) – читает содержимое символической ссылки. Используется для получения пути к исполняемому файлу через `/proc/self/exe`. Возвращает количество прочитанных байт при успехе, -1 при ошибке.

● void _exit(int status) – завершает выполнение процесса немедленно, не вызывая функции очистки. status – код возврата процесса.

Алгоритм работы программы:

1. Родительский процесс (parent.c):

- Читает первую строку из стандартного ввода – имя файла для записи результатов

- Создает два pipe: `parent_to_child` для передачи команд и `child_to_parent` для получения ответов

- Вызывает `fork()` для создания дочернего процесса

- В дочернем процессе: закрывает ненужные концы pipe, перенаправляет stdin/stdout через `dup2()`, вызывает `execv()` для запуска программы child

- В родительском процессе: закрывает ненужные концы pipe, в цикле читает строки с числами из stdin, передает их дочернему процессу через pipe, получает ответы и выводит их в stdout

- После завершения ввода закрывает pipe и ожидает завершения дочернего процесса через `waitpid()`

2. Дочерний процесс (child.c):

- Получает имя файла как аргумент командной строки

- Открывает файл для записи через `open()` с флагами O_WRONLY | O_CREAT | O_TRUNC

- В цикле читает строки из stdin (который перенаправлен на pipe от родителя)

- Парсит строку, извлекая числа с плавающей точкой и вычисляя их сумму

- Форматирует результат и записывает его в файл и в stdout (который перенаправлен на pipe к родителю)

- При ошибке парсинга отправляет сообщение об ошибке родителю

- После завершения ввода закрывает файл и завершает работу

Код программы

parent.c

```c
#define _POSIX_C_SOURCE 200809L
#include <errno.h>
#include <fcntl.h>
#include <limits.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
	#include <unistd.h>

	#define CHILD_PROGRAM_NAME "lab_01_child"
	#define MAX_LINE_LENGTH 4096

	static size_t string_length(const char *text) {
		size_t length = 0;
		while (text[length] != '\0') ++length;
		return length;
	}
	// write all bytes to fd если не все записалось
	static void write_all(int fd, const char *buffer, size_t length) {
		while (length > 0) {
			ssize_t written = write(fd, buffer, length);
			if (written < 0) _exit(EXIT_FAILURE);
			buffer += (size_t)written;
			length -= (size_t)written;
		}
	}

	static void fail(const char *message) {
		write_all(STDERR_FILENO, message, string_length(message));
		_exit(EXIT_FAILURE);
	}
	//
	static ssize_t read_line(int fd, char *buffer, size_t capacity) {
		if (capacity == 0) return -1;

		size_t offset = 0;
		while (offset + 1 < capacity) {
			char ch;
			ssize_t bytes = read(fd, &ch, 1);
			if (bytes < 0) {
				return -1;
			}
			if (bytes == 0) break;

			buffer[offset++] = ch;
			if (ch == '\n') break;
		}
		buffer[offset] = '\0';
		return (ssize_t)offset;
	}
	// input
	static void trim_trailing_newline(char *line) {
		size_t length = string_length(line);
		if (length == 0) {
			return;
		}
		if (line[length - 1] == '\n') line[length - 1] = '\0';
	}

	static void build_child_path(char *result, size_t capacity) {
		char executable_path[PATH_MAX];
		ssize_t len = readlink("/proc/self/exe", executable_path,
  sizeof(executable_path) - 1);
		if (len == -1){
			fail("error: failed to read /proc/self/exe\n");
		}

```

```
69.           executable_path[len] = '\0';
70.
71.           while (len > 0 && executable_path[len] != '/') --len;
72.           if (len == 0) {
73.                fail("error: executable path is invalid\n");
74.           }
75.
76.           executable_path[len] = '\0';
77.
78.           size_t dir_length = string_length(executable_path);
79.           size_t name_length = string_length(CHILD_PROGRAM_NAME);
80.           if (dir_length + 1 + name_length + 1 > capacity) {
81.                fail("error: child path buffer is too small\n");
82.           }
83.
84.           size_t index = 0;
85.           for (size_t i = 0; i < dir_length; ++i) result[index++] =
   executable_path[i];
86.           result[index++] = '/';
87.           for (size_t i = 0; i < name_length; ++i) result[index++] =
   CHILD_PROGRAM_NAME[i];
88.
89.           result[index] = '\0';
90.      }
91.      // write line to output_fd if line does not end with \n
92.      static void forward_line(int output_fd, const char *line) {
93.           size_t length = string_length(line);
94.           if (length == 0 || line[length - 1] != '\n') {
95.                write_all(output_fd, line, length);
96.                write_all(output_fd, "\n", 1);
97.                return;
98.           }
99.           write_all(output_fd, line, length);
100.     }
101.
102.     int main(void) {
103.          char filename[MAX_LINE_LENGTH];
104.          ssize_t filename_len = read_line(STDIN_FILENO, filename,
   sizeof(filename));
105.          if (filename_len <= 0) {
106.               fail("error: failed to read filename\n");
107.          }
108.
109.          trim_trailing_newline(filename);
110.          if (string_length(filename) == 0) {
111.               fail("error: filename must not be empty\n");
112.          }
113.          // parent to child pipe
114.          int parent_to_child[2];
115.          if (pipe(parent_to_child) == -1) {
116.               fail("error: failed to create pipe\n");
117.          }
118.          // child to parent pipe
119.          int child_to_parent[2];
120.          if (pipe(child_to_parent) == -1) {
121.               fail("error: failed to create pipe\n");
122.          }
123.          // fork child process
124.          pid_t child = fork();
125.          if (child == -1) {
126.               fail("error: failed to fork\n");
127.          }
128.
129.          if (child == 0) {
130.               if (close(parent_to_child[1]) == -1 ||
   close(child_to_parent[0]) == -1) {
131.                    fail("error: failed to close descriptors in
   child\n");
132.               }
```

```
133.
134.                 if (dup2(parent_to_child[0], STDIN_FILENO) == -1) {
135.                     fail("error: failed to redirect stdin\n");
136.                 }
137.
138.                 if (dup2(child_to_parent[1], STDOUT_FILENO) == -1) {
139.                     fail("error: failed to redirect stdout\n");
140.                 }
141.
142.                 if (close(parent_to_child[0]) == -1 ||
    close(child_to_parent[1]) == -1) {
143.                     fail("error: failed to close redundant
    descriptors\n");
144.                 }
145.
146.                 char child_path[PATH_MAX];
147.                 build_child_path(child_path, sizeof(child_path));
148.                 // pass filename to child process
149.                 char *const args[] = {CHILD_PROGRAM_NAME, filename,
    NULL};
150.                 // замена текущего процесса parent на child ожидвет
    завершения
151.                 execv(child_path, args);
152.                 // закрываем длчерний процесс
153.                 fail("error: exec failed\n");
154.             }
155.             // is pipe closed
156.             if (close(parent_to_child[0]) == -1 ||
    close(child_to_parent[1]) == -1) {
157.                 fail("error: failed to close descriptors in parent\n");
158.             }
159.
160.             char line_buffer[MAX_LINE_LENGTH];
161.             while(true) {
162.                 ssize_t line_length = read_line(STDIN_FILENO,
    line_buffer, sizeof(line_buffer));
163.                 if (line_length == -1) {
164.                     fail("error: failed to read input line\n");
165.                 }
166.
167.                 if (line_length == 0) break;
168.
169.                 if (line_buffer[0] == '\n') break;
170.
171.                 // write line to parent to child pipe
172.                 write_all(parent_to_child[1], line_buffer,
    (size_t)line_length);
173.
174.                 char response[MAX_LINE_LENGTH];
175.                 ssize_t response_length = read_line(child_to_parent[0],
    response, sizeof(response));
176.                 if (response_length <= 0) {
177.                     fail("error: child response failed\n");
178.                 }
179.                 forward_line(STDOUT_FILENO, response);
180.             }
181.
182.             if (close(parent_to_child[1]) == -1 ||
    close(child_to_parent[0]) == -1) {
183.                 fail("error: failed to close pipes\n");
184.             }
185.             // wait for child process to finish
186.             int status = 0;
187.             if (waitpid(child, &status, 0) == -1) {
188.                 fail("error: waitpid failed\n");
189.             }
190.             if (WIFEXITED(status)) {
191.                 return WEXITSTATUS(status);
192.             } else {
```

```
193.                 return EXIT_FAILURE;
194.           }
195.     }
```

child.c

```
1. #include <errno.h>
2. #include <fcntl.h>
3. #include <stdbool.h>
4. #include <stdint.h>
5. #include <stdlib.h>
6. #include <string.h>
7. #include <unistd.h>
8.
9. #define BUFFER_SIZE 4096
10.
11.     static size_t string_length(const char *text) {
12.           size_t length = 0;
13.           while (text[length] != '\0') ++length;
14.           return length;
15.     }
16.
17.     static void write_all(int fd, const char *buffer, size_t length) {
18.           while (length > 0) {
19.                 ssize_t written = write(fd, buffer, length);
20.                 if (written < 0) _exit(EXIT_FAILURE);
21.
22.                 buffer += (size_t)written;
23.                 length -= (size_t)written;
24.           }
25.     }
26.
27.     static void fail(const char *message) {
28.           write_all(STDERR_FILENO, message, string_length(message));
29.           _exit(EXIT_FAILURE);
30.     }
31.
32.     static ssize_t read_line(char *buffer, size_t capacity) {
33.           if (capacity == 0) return -1;
34.
35.           size_t offset = 0;
36.           while (offset + 1 < capacity) {
37.                 char ch;
38.                 ssize_t bytes = read(STDIN_FILENO, &ch, 1);
39.                 if (bytes < 0) return -1;
40.
41.                 if (bytes == 0) break;
42.
43.                 buffer[offset++] = ch;
44.                 if (ch == '\n') break;
45.           }
46.           buffer[offset] = '\0';
47.           return (ssize_t)offset;
48.     }
49.
50.     static bool parse_and_sum(char *line, double *result) {
51.           char *cursor = line;
52.           double total = 0.0;
53.           bool has_value = false;
54.           while (*cursor != '\0') {
55.                 while (*cursor == ' ' || *cursor == '\t') ++cursor;
56.
57.                 if (*cursor == '\0') break;
58.                 errno = 0;
59.                 char *next = NULL;
60.                 // string -> double
61.                 double value = strtod(cursor, &next);
62.                 if (cursor == next || errno == ERANGE) return false;
63.
```

```
64.                 total += value;
65.                 has_value = true;
66.                 cursor = next;
67.           }
68.           if (!has_value) return false;
69.           *result = total;
70.           return true;
71.      }
72.      // double -> string
73.      static size_t format_double(double value, char *buffer, size_t
   capacity) {
74.           if (capacity == 0) return 0;
75.           size_t index = 0;
76.           if (value < 0.0) {
77.                buffer[index++] = '-';
78.                value = -value;
79.                if (index >= capacity) return 0;
80.           }
81.
82.           long long integer_part = (long long)value;
83.           double fractional_part = value - (double)integer_part;
84.
85.           char digits[32];
86.           size_t digit_count = 0;
87.           do {
88.                digits[digit_count++] = (char)('0' + (integer_part %
   10));
89.                integer_part /= 10;
90.           } while (integer_part > 0 && digit_count < sizeof(digits));
91.
92.           while (digit_count > 0 && index < capacity) {
93.                buffer[index++] = digits[--digit_count];
94.           }
95.           if (index >= capacity) {
96.                return 0;
97.           }
98.
99.           buffer[index++] = '.';
100.          for (int i = 0; i < 6 && index < capacity; ++i) {
101.               fractional_part *= 10.0;
102.               int digit = (int)fractional_part;
103.               buffer[index++] = (char)('0' + digit);
104.               fractional_part -= digit;
105.          }
106.
107.          size_t end = index;
108.          while (end > 0 && buffer[end - 1] == '0') {
109.               --end;
110.          }
111.          if (end > 0 && buffer[end - 1] == '.') ++end;
112.
113.          if (end >= capacity) return 0;
114.          buffer[end] = '\0';
115.          return end;
116.     }
117.
118.     int main(int argc, char **argv) {
119.          // check how many arg
120.          if (argc < 2) {
121.               fail("error: file name argument is missing\n");
122.          }
123.          // O_WRONLY - write only, O_CREAT - create if not exists,
   O_TRUNC - truncate if exists, 0600 - R & W
124.          int file = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0600);
125.          if (file == -1) {
126.               fail("error: failed to open file\n");
127.          }
128.
129.          char line[BUFFER_SIZE];
```

```
130.          while(true) {
131.               ssize_t line_length = read_line(line, sizeof(line));
132.               if (line_length == -1){
133.                    fail("error: failed to read input\n");
134.               }
135.               if (line_length == 0) break;
136.               if (line[line_length - 1] == '\n')line[line_length - 1] =
     '\0';
137.               double sum = 0.0;
138.               if (!parse_and_sum(line, &sum)) {
139.                    const char warning[] = "error: invalid input\n";
140.                    write_all(STDOUT_FILENO, warning, sizeof(warning) -
     1);
141.                    continue;
142.               }
143.
144.               char value_buffer[128];
145.               size_t value_length = format_double(sum, value_buffer,
     sizeof(value_buffer));
146.               if (value_length == 0) {
147.                    fail("error: failed to format result\n");
148.               }
149.
150.               const char prefix[] = "sum: ";
151.               const char newline = '\n';
152.               // write to file
153.               write_all(file, prefix, sizeof(prefix) - 1);
154.               write_all(file, value_buffer, value_length);
155.               write_all(file, &newline, 1);
156.               // write to ter
157.               write_all(STDOUT_FILENO, prefix, sizeof(prefix) - 1);
158.               write_all(STDOUT_FILENO, value_buffer, value_length);
159.               write_all(STDOUT_FILENO, &newline, 1);
160.          }
161.
162.          if (close(file) == -1) {
163.               fail("error: failed to close file\n");
164.          }
165.          return EXIT_SUCCESS;
166.     }
167.  }
```

Протокол работы программы

Тестирование:

Программа была протестирована на различных входных данных. Примеры тестов:

Тест 1: Простые числа

```
1. $ ./build/lab_01_parent
2. results.txt
3. 1.5 2.5 3.0
4. sum: 7.0
5. 10.25 20.75
6. sum: 31.0
7. -5.5 10.5
8. sum: 5.0
9.
10. $ cat results.txt
11. sum: 7.0
12. sum: 31.0
13. sum: 5.0
14.
```

Тест 2: Множество чисел в одной строке

1. $ ./build/lab_01_parent
2. output.txt
3. 1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 9.9 10.0
4. sum: 59.5
5. 0.1 0.2 0.3 0.4 0.5
6. sum: 1.5
7.
8. $ cat output.txt
9. sum: 59.5
10. sum: 1.5

Тест 3: Обработка ошибок

1. $ ./build/lab_01_parent
2. test.txt
3. 1.5 2.5 abc
4. error: invalid input
5. 3.0 4.0
6. sum: 7.0
7. invalid
8. error: invalid input
9.
10. $ cat test.txt
11. sum: 7.0

Тест 4: Отрицательные числа и нули

1. $ ./build/lab_01_parent
2. negative.txt
3. -10.5 5.5
4. sum: -5.0
5. 0.0 0.0 0.0
6. sum: 0.0
7. -1.1 -2.2 -3.3
8. sum: -6.6
9.
10. $ cat negative.txt
11. sum: -5.0
12. sum: 0.0
13. sum: -6.6

Strace:

Для анализа системных вызовов использовалась утилита `strace` с флагом `-f` для отслеживания дочерних процессов. Программа была запущена через WSL:

$ strace -f -o strace_output.txt ./build/lab_01_parent

Ниже представлен фрагмент вывода `strace` с выделенными системными вызовами, используемыми в нашей программе. Написанные ниже строки соответствуют системным вызовам, вызванным непосредственно из нашего кода:

```
1232  execve("./lab_01_parent", ["./lab_01_parent"], 0x7ffce6175568 /* 26 vars */) = 0

1232  brk(NULL)                         = 0x6090f23ec000

1232  arch_prctl(0x3001 /* ARCH_??? */, 0x7ffcb04aa0b0) = -1 EINVAL (Invalid argument)

1232  mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7a9dcc968000

1232  access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

1232  openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

1232  newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=20892, ...}, AT_EMPTY_PATH) = 0

1232  mmap(NULL, 20892, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7a9dcc962000

1232  close(3)                          = 0

1232  openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

1232  read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) =
832

1232  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

1232  pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48,
848) = 48

1232  pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O{\f\225\\=\201\327\312\301P\32$\230\266\235"..., 68, 896) =
68

1232  newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0

1232  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

1232  mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a9dcc600000

1232  mprotect(0x7a9dcc628000, 2023424, PROT_NONE) = 0

1232  mmap(0x7a9dcc628000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7a9dcc628000

1232  mmap(0x7a9dcc7bd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x7a9dcc7bd000

1232  mmap(0x7a9dcc816000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x215000) = 0x7a9dcc816000

1232  mmap(0x7a9dcc81c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7a9dcc81c000

1232  close(3)                          = 0
```

```
1232  mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7a9dcc95f000

1232  arch_prctl(ARCH_SET_FS, 0x7a9dcc95f740) = 0

1232  set_tid_address(0x7a9dcc95fa10)   = 1232

1232  set_robust_list(0x7a9dcc95fa20, 24) = 0

1232  rseq(0x7a9dcc9600e0, 0x20, 0, 0x53053053) = 0

1232  mprotect(0x7a9dcc816000, 16384, PROT_READ) = 0

1232  mprotect(0x6090cd85f000, 4096, PROT_READ) = 0

1232  mprotect(0x7a9dcc9a2000, 8192, PROT_READ) = 0

1232  prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

1232  munmap(0x7a9dcc962000, 20892)     = 0

1232  read(0, "t", 1)                   = 1

1232  read(0, "e", 1)                   = 1

1232  read(0, "s", 1)                   = 1

1232  read(0, "t", 1)                   = 1

1232  read(0, "_", 1)                   = 1

1232  read(0, "o", 1)                   = 1

1232  read(0, "u", 1)                   = 1

1232  read(0, "t", 1)                   = 1

1232  read(0, "p", 1)                   = 1

1232  read(0, "u", 1)                   = 1

1232  read(0, "t", 1)                   = 1

1232  read(0, ".", 1)                   = 1

1232  read(0, "t", 1)                   = 1

1232  read(0, "x", 1)                   = 1

1232  read(0, "t", 1)                   = 1

1232  read(0, "\n", 1)                  = 1

1232  pipe2([3, 4], 0)                  = 0

1232  pipe2([5, 6], 0)                  = 0

1232  clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7a9dcc95fa10) = 1233

1233  set_robust_list(0x7a9dcc95fa20, 24 <unfinished ...>

1232  close(3 <unfinished ...>

1233  <... set_robust_list resumed>)    = 0

1232  <... close resumed>)              = 0

1233  close(4 <unfinished ...>
```

```
1232  close(6 <unfinished ...>
1233  <... close resumed>)              = 0
1232  <... close resumed>)              = 0
1233  close(5 <unfinished ...>
1232  read(0,  <unfinished ...>
1233  <... close resumed>)              = 0
1232  <... read resumed>"2", 1)         = 1
1233  dup2(3, 0 <unfinished ...>
1232  read(0,  <unfinished ...>
1233  <... dup2 resumed>)               = 0
1232  <... read resumed>".", 1)         = 1
1233  dup2(6, 1 <unfinished ...>
1232  read(0,  <unfinished ...>
1233  <... dup2 resumed>)               = 1
1232  <... read resumed>"0", 1)         = 1
1233  close(3 <unfinished ...>
1232  read(0,  <unfinished ...>
1233  <... close resumed>)              = 0
1232  <... read resumed>" ", 1)         = 1
1233  close(6 <unfinished ...>
1232  read(0,  <unfinished ...>
1233  <... close resumed>)              = 0
1232  <... read resumed>"2", 1)         = 1
1233  readlink("/proc/self/exe",  <unfinished ...>
1232  read(0,  <unfinished ...>
1233  <... readlink resumed>"/mnt/c/MAI/3_sem/OS/OS_LAB_1/lab"..., 4095) = 42
1232  <... read resumed>".", 1)         = 1
1233  execve("/mnt/c/MAI/3_sem/OS/OS_LAB_1/lab_01_child", ["lab_01_child",
"test_output.txt"], 0x7ffcb04aa288 /* 26 vars */ <unfinished ...>
1232  read(0, "5", 1)                   = 1
1232  read(0, " ", 1)                   = 1
1232  read(0, "3", 1)                   = 1
1232  read(0,  <unfinished ...>
1233  <... execve resumed>)             = 0
1232  <... read resumed>".", 1)         = 1
1233  brk(NULL <unfinished ...>
```

```
1232  read(0,  <unfinished ...>
1233  <... brk resumed>)                = 0x58f004a77000
1232  <... read resumed>"5", 1)         = 1
1233  arch_prctl(0x3001 /* ARCH_??? */, 0x7ffe6b206b70 <unfinished ...>
1232  read(0,  <unfinished ...>
1233  <... arch_prctl resumed>)         = -1 EINVAL (Invalid argument)
1232  <... read resumed>"\n", 1)        = 1
1233  mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished
...>
1232  write(4, "2.0 2.5 3.5\n", 12 <unfinished ...>
1233  <... mmap resumed>)               = 0x7c1ed51f2000
1232  <... write resumed>)              = 12
1233  access("/etc/ld.so.preload", R_OK <unfinished ...>
1232  read(5,  <unfinished ...>
1233  <... access resumed>)             = -1 ENOENT (No such file or directory)
1233  openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
1233  newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=20892, ...}, AT_EMPTY_PATH) = 0
1233  mmap(NULL, 20892, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7c1ed51ec000
1233  close(3)                          = 0
1233  openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
1233  read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) =
832
1233  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
1233  pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48,
848) = 48
1233  pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O{\f\225\\=\201\327\312\301P\32$\230\266\235"..., 68, 896) =
68
1233  newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
1233  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
1233  mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7c1ed4e00000
1233  mprotect(0x7c1ed4e28000, 2023424, PROT_NONE) = 0
1233  mmap(0x7c1ed4e28000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7c1ed4e28000
1233  mmap(0x7c1ed4fbd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x7c1ed4fbd000
1233  mmap(0x7c1ed5016000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x215000) = 0x7c1ed5016000
```

```
1233  mmap(0x7c1ed501c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7c1ed501c000

1233  close(3)                          = 0

1233  mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7c1ed51e9000

1233  arch_prctl(ARCH_SET_FS, 0x7c1ed51e9740) = 0

1233  set_tid_address(0x7c1ed51e9a10)   = 1233

1233  set_robust_list(0x7c1ed51e9a20, 24) = 0

1233  rseq(0x7c1ed51ea0e0, 0x20, 0, 0x53053053) = 0

1233  mprotect(0x7c1ed5016000, 16384, PROT_READ) = 0

1233  mprotect(0x58efcd084000, 4096, PROT_READ) = 0

1233  mprotect(0x7c1ed522c000, 8192, PROT_READ) = 0

1233  prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

1233  munmap(0x7c1ed51ec000, 20892)     = 0

1233  openat(AT_FDCWD, "test_output.txt", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 3

1233  read(0, "2", 1)                   = 1

1233  read(0, ".", 1)                   = 1

1233  read(0, "0", 1)                   = 1

1233  read(0, " ", 1)                   = 1

1233  read(0, "2", 1)                   = 1

1233  read(0, ".", 1)                   = 1

1233  read(0, "5", 1)                   = 1

1233  read(0, " ", 1)                   = 1

1233  read(0, "3", 1)                   = 1

1233  read(0, ".", 1)                   = 1

1233  read(0, "5", 1)                   = 1

1233  read(0, "\n", 1)                  = 1

1233  write(3, "sum: ", 5)              = 5

1233  write(3, "8.0", 3)                = 3

1233  write(3, "\n", 1)                 = 1

1233  write(1, "sum: ", 5 <unfinished ...>

1232  <... read resumed>"s", 1)         = 1

1233  <... write resumed>)              = 5

1232  read(5,  <unfinished ...>

1233  write(1, "8.0", 3 <unfinished ...>

1232  <... read resumed>"u", 1)         = 1
```

```
1233  <... write resumed>)              = 3
1232  read(5,  <unfinished ...>
1233  write(1, "\n", 1 <unfinished ...>
1232  <... read resumed>"m", 1)         = 1
1233  <... write resumed>)              = 1
1232  read(5,  <unfinished ...>
1233  read(0,  <unfinished ...>
1232  <... read resumed>":", 1)         = 1
1232  read(5, " ", 1)                   = 1
1232  read(5, "8", 1)                   = 1
1232  read(5, ".", 1)                   = 1
1232  read(5, "0", 1)                   = 1
1232  read(5, "\n", 1)                  = 1
1232  write(1, "sum: 8.0\n", 9)         = 9
1232  read(0, "1", 1)                   = 1
1232  read(0, "0", 1)                   = 1
1232  read(0, " ", 1)                   = 1
1232  read(0, "2", 1)                   = 1
1232  read(0, "0", 1)                   = 1
1232  read(0, " ", 1)                   = 1
1232  read(0, "3", 1)                   = 1
1232  read(0, "0", 1)                   = 1
1232  read(0, "\n", 1)                  = 1
1232  write(4, "10 20 30\n", 9)         = 9
1233  <... read resumed>"1", 1)         = 1
1232  read(5,  <unfinished ...>
1233  read(0, "0", 1)                   = 1
1233  read(0, " ", 1)                   = 1
1233  read(0, "2", 1)                   = 1
1233  read(0, "0", 1)                   = 1
1233  read(0, " ", 1)                   = 1
1233  read(0, "3", 1)                   = 1
1233  read(0, "0", 1)                   = 1
1233  read(0, "\n", 1)                  = 1
1233  write(3, "sum: ", 5)              = 5
```

```
1233  write(3, "60.0", 4)               = 4
1233  write(3, "\n", 1)                = 1
1233  write(1, "sum: ", 5 <unfinished ...>
1232  <... read resumed>"s", 1)         = 1
1233  <... write resumed>)              = 5
1232  read(5,  <unfinished ...>
1233  write(1, "60.0", 4 <unfinished ...>
1232  <... read resumed>"u", 1)         = 1
1233  <... write resumed>)              = 4
1232  read(5,  <unfinished ...>
1233  write(1, "\n", 1 <unfinished ...>
1232  <... read resumed>"m", 1)         = 1
1233  <... write resumed>)              = 1
1232  read(5,  <unfinished ...>
1233  read(0,  <unfinished ...>
1232  <... read resumed>":", 1)         = 1
1232  read(5, " ", 1)                   = 1
1232  read(5, "6", 1)                   = 1
1232  read(5, "0", 1)                   = 1
1232  read(5, ".", 1)                   = 1
1232  read(5, "0", 1)                   = 1
1232  read(5, "\n", 1)                  = 1
1232  write(1, "sum: 60.0\n", 10)       = 10
1232  read(0, "\n", 1)                  = 1
1232  close(4)                          = 0
1233  <... read resumed>"", 1)          = 0
1232  close(5 <unfinished ...>
1233  close(3 <unfinished ...>
1232  <... close resumed>)              = 0
1232  wait4(1233,  <unfinished ...>
1233  <... close resumed>)              = 0
1233  exit_group(0)                     = ?
1233  +++ exited with 0 +++
1232  <... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 1233
1232  --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=1233, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
```

```
1232  exit_group(0)                     = ?

1232  +++ exited with 0 +++
```

Ключевые системные вызовы из нашего кода:

1. **readlink("/proc/self/exe", ...)** (строка 82) – получение пути к исполняемому файлу родительского процесса для построения пути к дочернему процессу

2. **pipe2([3, 4], 0)** и **pipe2([5, 6], 0)** (строки 50-51) – создание двух каналов для двусторонней связи между процессами (pipe)

3. **clone(...)** (строка 52) – создание дочернего процесса (fork)

4. **close()** (строки 53, 57, 58, 62, 74, 78, 103, 117, 215, 217, 218) – закрытие ненужных файловых дескрипторов

5. **dup2()** (строки 66, 70) – перенаправление стандартных потоков ввода/вывода на pipe

6. **execve()** (строка 86) – замена образа процесса на дочернюю программу (execv)

7. **openat()** (строка 128) – открытие файла для записи результатов (open)

8. **read()** (строки 34-49, 61, 65, 69, 73, 77, 81, 85, 88-92, 94, 129-140, 147, 151, 155, 156, 158-162, 164-175, 178, 179-189, 196, 200, 204, 205, 207-212, 214) – чтение данных из stdin/pipe

9. **write()** (строки 93, 141-143, 144, 148, 152, 163, 176, 190-192, 193, 197, 201, 213) – запись данных в файл и в stdout/pipe

10. **wait4()** (строка 220) – ожидание завершения дочернего процесса (waitpid)

11. **exit_group()** (строки 222, 226) – завершение процесса (_exit)

Все системные вызовы проверяются на ошибки: возвращаемые значения сравниваются с -1 (или 0 для некоторых вызовов), и при ошибке программа выводит сообщение и завершается.

Вывод

В ходе выполнения лабораторной работы была реализована программа для межпроцессного взаимодействия между родительским и дочерним процессами с использованием неименованных каналов (pipe). Программа успешно выполняет поставленную задачу: родительский процесс передает строки с числами дочернему процессу, который вычисляет их сумму и записывает результаты в файл.

Основные достижения: реализовано корректное использование системных вызовов fork(), pipe(), dup2(), execv(), read(), write(), open(), close(), waitpid() и других для организации межпроцессного взаимодействия. Все системные вызовы проверяются на ошибки, что обеспечивает надежность работы программы. Программа корректно обрабатывает различные входные данные, включая отрицательные числа, нули и некорректный ввод.

При выполнении работы были изучены механизмы создания процессов, перенаправления потоков ввода/вывода и организации двусторонней связи между процессами через неименованные каналы. Программа успешно прошла тестирование и готова к демонстрации.