

**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»**

Выполнил: А. А. Устинов
Группа: М8О-207БВ-24
Преподаватель: Е. С. Миронов
Дата: _____

Москва, 2025

Условие

Цель работы: Приобретение практических навыков в управлении процессами в ОС и обеспечении обмена данными 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

Вариант: 7

Метод решения

Задача заключается в организации взаимодействия между двумя процессами в операционной системе Linux с помощью системных вызовов и канала (pipe).

Архитектура

Родительский процесс (parent.c):

1. Запрашивает у пользователя имя входного файла.
2. Создает канал (pipe).
3. Создает дочерний процесс (fork).
4. В дочернем процессе перенаправляет стандартный ввод на файл, а стандартный вывод на запись в канал (dup2).
5. Запускает дочернюю программу child.c через execv.
6. Считывает из канала результаты, которые формирует дочерний процесс, и выводит их в терминал.
7. Ждет завершения дочернего процесса (waitpid).

Дочерний процесс (child.c):

1. Считывает данные из стандартного ввода (перенаправленного на файл).
2. Обрабатывает построчно входные данные.
3. Для каждой строки суммирует числа.
4. Выводит результат в стандартный вывод, который перенаправлен в канал.

Внешние источники

1. man-страницы Linux (man 2 fork, man 2 pipe, man 2 dup2, man 3 scanf).
2. Документация GNU C Library.

Описание программы

Файл `parent.c`

Основные шаги:

- `scanf` — получение имени файла от пользователя.
- `pipe` — создание неименованного канала.
- `fork` — создание дочернего процесса.

В дочернем процессе:

- `open` — открытие файла.
- `dup2` — переназначение стандартного ввода на файл, стандартного вывода на канал.
- `execv` — замена текущего образа процесса на `child`.

В родительском процессе:

- `read` — чтение данных из канала.
- `write` — вывод в стандартный вывод (экран).
- `waitpid` — ожидание завершения дочернего процесса.

Системные вызовы:

- `pipe(int pipefd[2])` — создаёт канал для обмена данными между процессами.
- `fork(void)` — создаёт копию текущего процесса.
- `open(const char *pathname, int flags, mode_t mode)` — открывает файл.
- `dup2(int oldfd, int newfd)` — перенаправляет один файловый дескриптор в другой.
- `execv(const char *path, char *const argv[])` — запускает новую программу в текущем процессе.
- `close(int fd)` — закрывает файловый дескриптор.
- `read(int fd, void *buf, size_t count)` — читает данные из файла или канала.
- `write(int fd, const void *buf, size_t count)` — записывает данные в файл или канал.

Файл `child.c`

Функции и логика:

- Чтение строк из `stdin` (перенаправленного на файл).
- Подсчёт суммы чисел строки.
- Проверка ошибок.
- Вывод результата в `stdout` (перенаправлен в `pipe`).

Системные вызовы:

- `getchar` — чтение символов/чисел.
- `printf` — вывод результата.

Результаты

Разработанное решение состоит из двух программ на языке C: `parent.c` и `child.c`, которые взаимодействуют между собой с использованием механизмов межпроцессного взаимодействия (IPC) в операционной системе Linux.

Ключевые особенности

1. Родительский процесс отвечает за управление (создание канала, запуск дочернего процесса, получение и вывод результата), а дочерний процесс выполняет исходную задачу.
2. Стандартный поток вывода дочернего процесса перенаправляется в канал, из которого читает родитель.
3. С помощью системного вызова `dup2` стандартный ввод дочернего процесса заменяется файлом, выбранным пользователем, а стандартный вывод перенаправляется в канал.
4. Родительский и дочерний процесс реализованы как разные исполняемые файлы.

Выводы

В ходе выполнения данной лабораторной работы была разработана программа на языке C, демонстрирующая взаимодействие процессов в операционной системе Linux.

Основные результаты работы:

- Освоено создание дочерних процессов с помощью системного вызова `fork`.
- Реализовано взаимодействие процессов через канал (`pipe`), включая перенаправление стандартного ввода и вывода с помощью `dup2`.
- Изучено использование системных вызовов `open`, `read`, `write`, `close` для работы с файлами и каналами.
- Создано разделение обязанностей между процессами.

Исходная программа

Файл parent.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5  #include <fcntl.h>
6  #include <string.h>
7  #include <errno.h>
8
9  int main() {
10     char filename[256];
11     // fd for parent -> children
12     int pipefd[2];
13     pid_t pid;
14
15     if (pipe(pipefd) == -1) {
16         fprintf(stderr, "Failed to create pipe\n");
17         return -1;
18     }
19
20     printf("Filename:\n");
21     if (scanf("%255s", filename) != 1) {
22         fprintf(stderr, "Incorrect file name\n");
23         return -1;
24     }
25
26     int fd_file = open(filename, O_RDONLY);
27     if (fd_file == -1) {
28         fprintf(stderr, "Failed to open file\n");
29         return -1;
30     }
31
32     pid = fork();
33     if (pid < 0) {
34         fprintf(stderr, "Fork failed\n");
35         close(fd_file);
36         close(pipefd[0]);
37         close(pipefd[1]);
38         return -1;
39     }
40
41     if (pid == 0) {
42         // Close pip for reading
43         close(pipefd[0]);
44
45         // input -> file
46         if (dup2(fd_file, STDIN_FILENO) == -1) {
47             fprintf(stderr, "dup2(fd_file->stdin) failed:\n");
48             return -1;
49         }
50
51         // child proc -> pipe
52         if (dup2(pipefd[1], STDOUT_FILENO) == -1) {
53             fprintf(stderr, "dup2(pipe_write->stdout) failed:\n");
54             return -1;
```

```

55     }
56
57     close(fd_file);
58     close(pipefd[1]);
59
60     // proc -> child
61     char *argv[] = { "./child", NULL };
62     execv(argv[0], argv);
63
64     fprintf(stderr, "execv failed:\n");
65     return -1;
66 } else {
67     // close fd for writing
68     close(pipefd[1]);
69     close(fd_file);
70
71     // Any answer in terminal
72     ssize_t n;
73     char buffer[4096];
74     while ((n = read(pipefd[0], buffer, sizeof(buffer))) > 0) {
75         ssize_t out = 0;
76         while (out < n) {
77             ssize_t w = write(STDOUT_FILENO, buffer + out, n - out);
78             if (w == -1) {
79                 perror("write");
80                 break;
81             }
82             out += w;
83         }
84     }
85
86     if (n == -1) {
87         fprintf(stderr, "Failed read from pipe");
88     }
89
90     close(pipefd[0]);
91     if (wait(NULL) == -1) {
92         perror("wait");
93     }
94
95 }
96
97 return 0;
98 }

```

Листинг 1: Код программы родительского процесса

Файл child.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     float x, sum = 0.0f;
6     int read_count = 0; // count numbers in str
7     int c;

```

```

8
9     while (1) {
10         int r = scanf("%f", &x);
11
12         if (r == 1) {
13             sum += x;
14             read_count++;
15         } else if (r == EOF) {
16             // end of file
17             if (read_count > 0) {
18                 printf("Sum: %.2f\n", sum);
19             }
20             break;
21         } else {
22             /* scanf cant read number */
23             printf("Not number in file!\n");
24             return -1;
25         }
26
27         // If next tab, space or n
28         c = getchar();
29         if (c == '\n' || c == EOF) {
30             if (read_count > 0) {
31                 printf("Sum: %.2f\n", sum);
32                 sum = 0.0f;
33                 read_count = 0;
34             }
35             if (c == EOF) break;
36         } else if (c == ' ' || c == '\t' || c == '\r') {
37             // Correct symbols
38         } else {
39             // Incorrect Symbol
40             printf("Error: Incorrect Symbol '%c'\n", c);
41             return -1;
42         }
43     }
44
45     return 0;
46 }

```

Листинг 2: Код программы дочернего процесса

Вывод strace

```

execve("./parent", ["/parent"], 0x7fff77b42d10) = 0
pipe2([3,4],0) = 0
write(1,"Filename:\n",10) = 10
read(0,"input.txt\n",1024) = 10
openat(AT_FDCWD,"input.txt",O_RDONLY) = 5
clone(...) = 6474
read(3,"Sum: 3354.30\n",4096) = 13
write(1,"Sum: 3354.30\n",13) = 13
---SIGCHLD ---
wait4(-1,NULL,0,NULL) = 6474

```