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

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. Создаёт канал (`pipe`).
3. Создаёт 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. Ждёт 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

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

int main() {
    char filename[256];
    // fd for parent -> children
    int pipefd[2];
    pid_t pid;

    if (pipe(pipefd) == -1) {
        fprintf(stderr, "Failed to create pipe\n");
        return -1;
    }

    printf("Filename:\n");
    if (scanf("%255s", filename) != 1) {
        fprintf(stderr, "Incorrect file name\n");
        return -1;
    }

    int fd_file = open(filename, O_RDONLY);
    if (fd_file == -1) {
        fprintf(stderr, "Failed to open file\n");
        return -1;
    }

    pid = fork();
    if (pid < 0) {
        fprintf(stderr, "Fork failed\n");
        close(fd_file);
        close(pipefd[0]);
        close(pipefd[1]);
        return -1;
    }

    if (pid == 0) {
        // Close pip for reading
        close(pipefd[0]);

        // input -> file
        if (dup2(fd_file, STDIN_FILENO) == -1) {
            fprintf(stderr, "dup2(fd_file->stdin) failed:\n");
            return -1;
        }

        // child proc -> pipe
        if (dup2(pipefd[1], STDOUT_FILENO) == -1) {
            fprintf(stderr, "dup2(pipe_write->stdout) failed:\n");
            return -1;
```

```
        }

        close(fd_file);
        close(pipefd[1]);

        // proc -> child
        char *argv[] = {"./child", NULL};
        execv(argv[0], argv);

        fprintf(stderr, "execv failed:\n");
        return -1;
    } else {
        // close fd for writing
        close(pipefd[1]);
        close(fd_file);

        // Any answer in terminal
        ssize_t n;
        char buffer[4096];
        while ((n = read(pipefd[0], buffer, sizeof(buffer))) > 0) {
            ssize_t out = 0;
            while (out < n) {
                ssize_t w = write(STDOUT_FILENO, buffer + out, n - out);
                if (w == -1) {
                    perror("write");
                    break;
                }
                out += w;
            }
        }

    if (n == -1) {
        fprintf(stderr, "Failed read from pipe");
    }

    close(pipefd[0]);
    if (wait(NULL) == -1) {
        perror("wait");
    }

    }

    return 0;
}
```

Листинг 1: Код программы родительского процесса

Файл child.c

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

int main() {
    float x, sum = 0.0f;
    int read_count = 0; // count numbers in str
    int c;
```

```c

    while (1) {
        int r = scanf("%f", &x);

        if (r == 1) {
            sum += x;
            read_count++;
        } else if (r == EOF) {
            // end of file
            if (read_count > 0) {
                printf("Sum: %.2f\n", sum);
            }
            break;
        } else {
            /* scanf cant read number */
            printf("Not number in file!\n");
            return -1;
        }

        // If next tab, space or n
        c = getchar();
        if (c == '\n' || c == EOF) {
            if (read_count > 0) {
                printf("Sum: %.2f\n", sum);
                sum = 0.0f;
                read_count = 0;
            }
            if (c == EOF) break;
        } else if (c == ' ' || c == '\t' || c == '\r') {
            // Correct symbols
        } else {
            // Incorrect Symbol
            printf("Error: Incorrect Symbol '%c'\n", c);
            return -1;
        }
    }

    return 0;
}
```

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
```