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-206Б-23

Студент: Абдыкалыков Н. А.

Преподаватель: Миронов Е. С.

Оценка: _____

Дата: 28.02.25

Москва, 2025

Постановка задачи

Вариант 6.

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.

В файле записаны команды вида: «число число число». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип int. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

В данной задаче используется межпроцессное взаимодействие через каналы, чтобы передать числа от родительского процесса в дочерний и получить результат.

1. **pipe(int pipefd[2])** – для создания неименованного канала для межпроцессного взаимодействия; **pipefd[2]** – это массив из двух элементов, где:

   pipe[0] – это дескриптор для чтения из канала.

   pipe[1] – это дескриптор для записи в канал.

2. **pid_t pid = fork()** — для создания дочернего процесса, который является копией родительского. Значение функции зависит от места вызова:

   В родительском возвращает PID дочернего процесса.

   В дочернем возвращает ноль.

Это позволяет программе различать, какой процесс выполняет код: родительский или дочерний. **pid_t –** специальный тип данных для хранения идентификаторов процессов (pid).

3. **execl(“./child”, “child”, NULL)** – это системный вызов, который загружает и запускает программу, указанную в аргументе.
4. **dup2(fd, STDIN_FILENO) -** перенаправляет стандартный ввод на fd.
5. **close(fd) -** закрывает дескриптор fd.
6. **dup2(pipe1[1], STDOUT_FILENO) -** перенаправляет стандартный вывод в канал pipe1[1].
7. **close(pipe1[1])** - закрывает запись в канал. Это используется для перенаправления ввода и вывода между процессами.
8. **wait(NULL)** - эта функция заставляет родительский процесс ожидать завершения дочернего, возвращая его статус завершения. NULL в данном случае означает, что родительский процесс не интересуется кодом завершения дочернего процесса. То есть, он просто ожидает завершения дочернего процесса, но не использует его код выхода.
9. **exit(EXIT_FAILURE)** - эта команда завершает программу с кодом завершения, равным **EXIT_FAILURE**. Это указывает на ошибку при выполнении программы.

Реализация: идея и подход:

Программа создаёт канал и запускает дочерний процесс. Родительский процесс передает имя файла, дочерний процесс читает его, считает сумму чисел и отправляет результат обратно через канал. Родитель выводит результат.

Используются:

- pipe() для канала.
- fork() для создания дочернего процесса.
- dup2() для перенаправления ввода и вывода.
- execl() для выполнения дочерней программы.

Код программы

Parent.c

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


int main() {
    int pipe1[2];
    pid_t pid;
    char filename[32];
    if (pipe(pipe1) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }
    printf("Введите имя файла: ");
    scanf("%s", filename);
    pid = fork();
    if (pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }
    if (pid == 0) {
        close(pipe1[0]);
        int fd = open(filename, O_RDONLY);
        if (fd == -1) {
            perror("open");
            exit(EXIT_FAILURE);
        }
```

```
        dup2(fd, STDIN_FILENO);
        close(fd);
        dup2(pipe1[1], STDOUT_FILENO);
        close(pipe1[1]);
        execl("./child", "child", NULL);
        perror("execl");
        exit(EXIT_FAILURE);
    } else {
        close(pipe1[1]);
        char buffer[256];
        ssize_t count = read(pipe1[0], buffer, sizeof(buffer) - 1);
        if (count == -1) {
            perror("read");
            exit(EXIT_FAILURE);
        }
        buffer[count] = '\0';
        printf("Сумма чисел: %s\n", buffer);
        wait(NULL);
    }
    return 0;
}
```

Child.c

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int sum = 0;
    int number;
    while (scanf("%d", &number) == 1) {
        sum += number;
```

```
    }
    printf("%d\n", sum);
    return 0;
}
```

Протокол работы программы

Работа программы и её вывод:

```
nursultan@DESKTOP-JMKISMP:/mnt/c/Users/abdyk/Desktop/MAI_OS_LABS/lr1$ gcc -o parent parent.c
nursultan@DESKTOP-JMKISMP:/mnt/c/Users/abdyk/Desktop/MAI_OS_LABS/lr1$ gcc -o child child.c
nursultan@DESKTOP-JMKISMP:/mnt/c/Users/abdyk/Desktop/MAI_OS_LABS/lr1$ ./parent
Введите имя файла: number.txt
Сумма чисел: 150
```

Strace:

execve("./parent", ["./parent"], 0x7ffd957f4b60 /* 23 vars */) = 0

brk(NULL) = 0x55eba5d28000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6138fc7000

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

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=18354, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 18354, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f6138fc2000

close(3) = 0

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

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20t\2\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"..., 784, 64) = 784

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1922136, ...}, AT_EMPTY_PATH) = 0

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

mmap(NULL, 1970000, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f6138de1000

mmap(0x7f6138e07000, 1396736, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26000) = 0x7f6138e07000

mmap(0x7f6138f5c000, 339968, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17b000) = 0x7f6138f5c000

mmap(0x7f6138faf000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1ce000) = 0x7f6138faf000

mmap(0x7f6138fb5000, 53072, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f6138fb5000

close(3)   = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f6138dde000

arch_prctl(ARCH_SET_FS, 0x7f6138dde740) = 0

set_tid_address(0x7f6138ddea10)   = 976

set_robust_list(0x7f6138ddea20, 24)   = 0

rseq(0x7f6138ddf060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f6138faf000, 16384, PROT_READ) = 0

mprotect(0x55eb91b47000, 4096, PROT_READ) = 0

mprotect(0x7f6138ff9000, 8192, PROT_READ) = 0

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

munmap(0x7f6138fc2000, 18354)   = 0

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

newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0

getrandom("\xbe\x13\xfd\x75\x55\x4f\x8a\x0a", 8, GRND_NONBLOCK) = 8

brk(NULL)   = 0x55eba5d28000

brk(0x55eba5d49000)   = 0x55eba5d49000

newfstatat(0, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\270\320\274\321\217
\321\204\320\260\320\271\320\273\320\260"..., 34Введите имя файла:) = 34

read(0, number.txt

"number.txt\n", 1024)   = 11

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f6138ddea10) = 977

close(4)   = 0

read(3, "150\n", 255)   = 4

write(1, "\320\241\321\203\320\274\320\274\320\260 \321\207\320\270\321\201\320\265\320\273:
150\n", 27Сумма чисел: 150

) = 27

write(1, "\n", 1

) = 1

wait4(-1, NULL, 0, NULL) = 977

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=977, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

lseek(0, -1, SEEK_CUR) = -1 ESPIPE (Illegal seek)

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В ходе выполнения работы я понял, как работать с межпроцессным взаимодействием с помощью канала (pipe) и как перенаправлять ввод/вывод с помощью dup2(). Также понял, как использовать fork() для создания дочерних процессов и execl() для выполнения внешней программы.