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-209БВ-24

Студент: Фомичев Н.С.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 15.10.25

Москва, 2024

**Постановка задачи**

**Вариант 3.**

Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип int. 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 канал и возвращает два дескриптора
* int close(int fd) - закрывает файловый дескриптор
* int execl(const char \*path, const char \*arg, ...) - загружает и запускает новую программу, полностью заменяя текущий процесс

В рамках лабораторной работы была реализована программа, демонстрирующая межпроцессное взаимодействие в операционных системах с использованием системных вызовов. Родительский процесс создает дочерний процесс с помощью fork(), организует между ними связь через каналы (pipe) и передает данные для обработки. Родительский процесс читает команды пользователя (числа для деления) и передает их через pipe1 дочернему процессу, перенаправив его стандартный ввод на чтение из канала с помощью dup2. Дочерний процесс, запускаемый через execl(), выполняет арифметические операции (деление первого числа на последующие) и записывает результаты в файл, проверяя возможность деления на ноль, что приводит к аварийному завершению обоих процессов. Программа наглядно иллюстрирует работу с процессами, перенаправлением потоков и обработкой ошибок.

**Код программы**

**parent.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <string.h>

int main()

{

    int pipe1[2];

    int pipe2[2];

    pid\_t pid;

    char filename[50];

    if (pipe(pipe1) == -1 || pipe(pipe2) == -1){

      perror("Не удалось создать pipe1 или pipe2\n");

      exit(1);

    }

    printf("%s", "Введите имя файла: ");

    if (fgets(filename, sizeof(filename), stdin) == NULL){

      perror("Не получилось прочитать название!\n");

      exit(1);

    }

    filename[strcspn(filename, "\n")] = 0;

    pid = fork();

    if(pid<0){

      perror("Сегодня без fork\n");

      exit(1);

    }

    else if(pid == 0){

      close(pipe1[1]);

      close(pipe2[0]);

      dup2(pipe1[0], STDIN\_FILENO);

      close(pipe1[0]);

      execl("./child", "child", filename, NULL);

      perror("Ничего ты не понимаешь в execl\n");

      exit(1);

    }

    else{

      close(pipe1[0]);

      close(pipe2[1]);

      char input[1024];

      printf("%s", "Введите числа, разделенные пробелом, а затем нажмите enter\n");

      if (fgets(input, sizeof(input), stdin) == NULL) {

          perror("У меня что то при чтении случилось\n");

        }

      write(pipe1[1], input, strlen(input));

      close(pipe1[1]);

      wait(NULL);

      printf("%s", "Родительский процесс успешно завершился\n");

    }

    return 0;

}

**child.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

int main(int argc, char \*argv[]){

    char \*filename = argv[1];

    FILE \*output\_file = fopen(filename, "w");

    if (output\_file == NULL) {

        perror("Не получилось открыть файл с таким названием\n");

        exit(1);

    }

    char buffer[1024];

    while (1){

        if (fgets(buffer, sizeof(buffer), stdin) == NULL) {

            break;

        }

        buffer[strcspn(buffer, "\n")] = 0;

        int numbers[100];

        int count = 0;

        char \*token = strtok(buffer, " ");

        while (token != NULL && count < 100) {

            numbers[count++] = atoi(token);

            token = strtok(NULL, " ");

        }

        if (count < 2) {

            fprintf(output\_file, "Ошибка: Нужно хотя бы два числа\n");

            fflush(output\_file);

            continue;

        }

        int has\_zero = 0;

        for (int i = 1; i < count; i++) {

            if (numbers[i] == 0) {

                fprintf(output\_file, "Ошибка: Деление на нуль\n");

                fflush(output\_file);

                has\_zero = 1;

                break;

            }

        }

        if (has\_zero) {

            fclose(output\_file);

            exit(1);

        }

        fprintf(output\_file, "%d", numbers[0]);

        int result = numbers[0];

        for (int i = 1; i < count; i++) {

            result /= numbers[i];

            fprintf(output\_file, " / %d", numbers[i]);

        }

        fprintf(output\_file, " = %d\n", result);

        fflush(output\_file);

    }

    fclose(output\_file);

    return 0;

}

**Протокол работы программы**

strace -o trace.log ./parent

Введите имя файла: Tantumbek.txt

Введите числа, разделенные пробелом, а затем нажмите enter

1024 2 2 2 2 2 2

Родительский процесс успешно завершился

cat < Tantumbek.txt

1024 / 2 / 2 / 2 / 2 / 2 / 2 = 16

**Strace:**

execve("./parent", ["./parent"], 0x7ffc927ce590 /\* 26 vars \*/) = 0

brk(NULL)                               = 0x58fbd6f1d000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7bcb95d45000

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

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=19991, ...}) = 0

mmap(NULL, 19991, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7bcb95d40000

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\0\3\0>\0\1\0\0\0\220\243\2\0\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

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2125328, ...}) = 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, 2170256, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7bcb95a00000

mmap(0x7bcb95a28000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7bcb95a28000

mmap(0x7bcb95bb0000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x7bcb95bb0000

mmap(0x7bcb95bff000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x7bcb95bff000

mmap(0x7bcb95c05000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7bcb95c05000

close(3)                                = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7bcb95d3d000

arch\_prctl(ARCH\_SET\_FS, 0x7bcb95d3d740) = 0

set\_tid\_address(0x7bcb95d3da10)         = 538

set\_robust\_list(0x7bcb95d3da20, 24)     = 0

rseq(0x7bcb95d3e060, 0x20, 0, 0x53053053) = 0

mprotect(0x7bcb95bff000, 16384, PROT\_READ) = 0

mprotect(0x58fbb9b4e000, 4096, PROT\_READ) = 0

mprotect(0x7bcb95d7d000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x7bcb95d40000, 19991)           = 0

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

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

fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

getrandom("\xe7\xe3\x99\xa2\x4a\x12\x23\x68", 8, GRND\_NONBLOCK) = 8

brk(NULL)                               = 0x58fbd6f1d000

brk(0x58fbd6f3e000)                     = 0x58fbd6f3e000

fstat(0, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 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"..., 34) = 34

read(0, "Tantumbek.txt\n", 1024)        = 14

**clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7bcb95d3da10) = 539**

close(3)                                = 0

close(6)                                = 0

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\207\320\270\321\201\320\273\320\260, \321\200\320\260\320"..., 103) = 103

read(0, "1024 2 2 2 2 2 2\n", 1024)     = 17

write(4, "1024 2 2 2 2 2 2\n", 17)      = 17

close(4)                                = 0

wait4(-1, NULL, 0, NULL)                = 539

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=539, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

write(1, "\320\240\320\276\320\264\320\270\321\202\320\265\320\273\321\214\321\201\320\272\320\270\320\271 \320\277\321\200\320\276\321"..., 76) = 76

exit\_group(0)                           = ?

+++ exited with 0 +++

**Вывод**

В ходе лабораторной работы было успешно реализовано межпроцессное взаимодействие с использованием системных вызовов fork, pipe и dup2. Программа продемонстрировала эффективный механизм обмена данными между родительским и дочерним процессами, включая обработку арифметических операций и контроль ошибок.  Основные сложности возникли с корректным закрытием файловых дескрипторов pipe после перенаправления потоков, что приводило к блокировкам процессов. Также требовалось тщательно отслеживать состояние дочернего процесса для немедленного реагирования на деление на ноль