

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: Забродин Р.У.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 23.12.25

Москва, 2025

Постановка задачи

Вариант 4.

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программы (основной процесс) должен создать для решения задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe). 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 float. 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 pipe для межпроцессорного взаимодействия
- int dup2(int oldfd, int newfd); - перенаправление файловых дескрипторов
- int close(int fd); – закрытие файловых дескрипторов
- int execl(const char *path,const char *arg,...,NULL); - заменяет текущий процесс новым
- ssize_t read(int fd,void *buf,size_t count); - читает данные из файлового дескриптора
- pid_t wait(int *status); - ожидает завершения дочернего процесса

Я считал данные в родительском процессе parent и прописал поведение программы для разных pid, для дочернего процесса перенаправил ввод и вывод с помощью функции dup2, считал входные данные из pipe1 и произвел необходимые математические действия (согласно заданию) ответ записал в файл, обработал возможный случай деления числа на 0 (написал условие для завершения обоих процессов)

Код программы

Parent.c

```
#include "stdio.h"

#include "stdlib.h" // pid_t declaration is here

#include "unistd.h" // fork() and getpid() declarations are here

#include <sys/types.h>

#include "sys/wait.h"

int main(){

    char filename[1000];

    char Buffer[1000];
```

```
float numbers[100];

//комменты сделал для себя

int pipe_fd_1[2];//parent->child

int pipe_fd_2[2];//child->parent

if (pipe(pipe_fd_1)==-1 || pipe(pipe_fd_2)==-1){

    perror("pipeerror");

}

fgets(filename, sizeof(filename), stdin);

int i=0;

while (scanf("%f",&numbers[i])==1){

    ++i;

}

int count=i;//Сохраняю для передачи в child

pid_t pid = fork();

if (pid== -1){

    perror("fork error");

    return -1;

} else if (pid==0){

    close(pipe_fd_1[1]);

    close(pipe_fd_2[0]);



    dup2(pipe_fd_1[0],STDIN_FILENO);

    dup2(pipe_fd_2[1],STDOUT_FILENO);

    close(pipe_fd_1[0]);

    close(pipe_fd_2[1]);

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

} else{

    close(pipe_fd_2[1]);

    close(pipe_fd_1[0]);
```

```

write(pipe_fd_1[1], filename, sizeof(filename));
write(pipe_fd_1[1], &count,sizeof(count));
for (i=0;i<count;i++){
    write(pipe_fd_1[1], &numbers[i], sizeof(numbers[i]));
}
close(pipe_fd_1[1]);
int status;
wait(&status);
if (WIFEXITED(status)) {
    int exit_code = WEXITSTATUS(status);
    if (exit_code == EXIT_FAILURE) {
        perror("division by zero detected. Parent exiting.\n");
        exit(EXIT_FAILURE);
    }
}
close(pipe_fd_2[0]);
}
}

```

Child.c

```

#include "stdio.h"
#include "stdlib.h"
#include "unistd.h"
#include <string.h>
int main(){
char filename[1000];
int count;
float numbers[100];
read(STDIN_FILENO, filename, sizeof(filename));
read(STDIN_FILENO, &count, sizeof(count));
for (int i = 0; i < count; i++) {

```

```

read(STDIN_FILENO, &numbers[i], sizeof(numbers[i]));
}

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

FILE *file = fopen(filename, "w");

if (file == NULL) {
    perror("Failed to open file");
    exit(EXIT_FAILURE);
}

for (int i = 1; i < count; i++) {
    if (numbers[i] == 0) {
        perror("zero division");
        fclose(file);
        exit(EXIT_FAILURE);
    }

    float result = numbers[0] / numbers[i];
    fprintf(file, "%f\n", result);
    fflush(file);
}

fclose(file);
exit(EXIT_SUCCESS);
}

```

Протокол работы программы

Тестирование:

```

$ ./parent.c
test.txt
1.2 4.3268 83.25 0.36
cat test.txt
0.277341
0.014414

```

3.333333

Strace:

```
strace -f ./parent

execve("./parent", [ "./parent"], 0x7ffdcb9e2d28 /* 31 vars */) = 0
brk(NULL) = 0x104d5000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x76d445143000
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=33091, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 33091, PROT_READ, MAP_PRIVATE, 3, 0) = 0x76d44513a000
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\0\0\3\0>\0\1\0\0\0\20t\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
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1926232, ...}, 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, 1974096, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x76d444f58000
mmap(0x76d444f7e000, 1400832, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26000) = 0x76d444f7e000
mmap(0x76d4450d4000, 339968, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17c000) = 0x76d4450d4000
mmap(0x76d445127000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1cf000) = 0x76d445127000
mmap(0x76d44512d000, 53072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x76d44512d000
close(3) = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x76d444f55000
arch_prctl(ARCH_SET_FS, 0x76d444f55740) = 0
set_tid_address(0x76d444f55a10) = 5695
```

```
set_robust_list(0x76d444f55a20, 24)      = 0
rseq(0x76d444f56060, 0x20, 0, 0x53053053) = 0
mprotect(0x76d445127000, 16384, PROT_READ) = 0
mprotect(0x403000, 4096, PROT_READ)      = 0
mprotect(0x76d445176000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x76d44513a000, 33091)          = 0
pipe2([3, 4], 0)                      = 0
pipe2([5, 6], 0)                      = 0
newfstatat(0, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0
getrandom("\xca\xd5\xc0\x2c\x81\xe1\x93\x23", 8, GRND_NONBLOCK) = 8
brk(NULL)                            = 0x104d5000
brk(0x104f6000)                     = 0x104f6000
read(0, test.txt
"test.txt\n", 1024)                 = 9
read(0, 1.2 4.3268 83.25 0.36
"1.2 4.3268 83.25 0.36\n", 1024) = 22
read(0, "", 1024)                   = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process
6043 attached
, child_tidptr=0x76d444f55a10) = 6043
[pid 6043] set_robust_list(0x76d444f55a20, 24 <unfinished ...>
[pid 5695] close(6 <unfinished ...>
[pid 6043] <... set_robust_list resumed>) = 0
[pid 5695] <... close resumed>)      = 0
[pid 6043] close(4 <unfinished ...>
[pid 5695] close(3)                  = 0
[pid 6043] <... close resumed>)      = 0
```

```
[pid  5695] write(4,
"test.txt\n\0\24E\324v\0\0` \267\225\222\377\177\0\0\1\0\0\0\324v\0\0"..., 1000 <unfinished
...>

[pid  6043] close(5 <unfinished ...>

[pid  5695] <... write resumed>          = 1000

[pid  6043] <... close resumed>          = 0

[pid  5695] write(4, "\4\0\0\0", 4)       = 4

[pid  6043] dup2(3, 0 <unfinished ...>

[pid  5695] write(4, "\232\231\231?", 4 <unfinished ...>

[pid  6043] <... dup2 resumed>          = 0

[pid  5695] <... write resumed>          = 4

[pid  6043] dup2(6, 1 <unfinished ...>

[pid  5695] write(4, "%u\212@", 4 <unfinished ...>

[pid  6043] <... dup2 resumed>          = 1

[pid  5695] <... write resumed>          = 4

[pid  6043] close(3 <unfinished ...>

[pid  5695] write(4, "\0\200\246B", 4 <unfinished ...>

[pid  6043] <... close resumed>          = 0

[pid  5695] <... write resumed>          = 4

[pid  6043] close(6 <unfinished ...>

[pid  5695] write(4, "\354Q\270", 4)      = 4

[pid  6043] <... close resumed>          = 0

[pid  5695] close(4 <unfinished ...>

[pid  6043] execve("./child", ["child"], 0x7fff9295bb38 /* 31 vars */ <unfinished ...>

[pid  5695] <... close resumed>          = 0

[pid  5695] wait4(-1,  <unfinished ...>

[pid  6043] <... execve resumed>         = 0

[pid  6043] brk(NULL)                     = 0x2a368000
```



```
[pid 6043] mprotect(0x7600cd7d8000, 16384, PROT_READ) = 0
[pid 6043] mprotect(0x403000, 4096, PROT_READ) = 0
[pid 6043] mprotect(0x7600cd827000, 8192, PROT_READ) = 0
[pid 6043] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
[pid 6043] munmap(0x7600cd7eb000, 33091) = 0
[pid 6043] read(0, "test.txt\n\0\24E\324v\0\0` \267\225\222\377\177\0\0\1\0\0\0\324v\0\0"..., 1000) = 1000
[pid 6043] read(0, "\4\0\0\0", 4) = 4
[pid 6043] read(0, "\232\231\231?", 4) = 4
[pid 6043] read(0, "%u\212@", 4) = 4
[pid 6043] read(0, "\0\200\246B", 4) = 4
[pid 6043] read(0, "\354Q\270>", 4) = 4
[pid 6043] getrandom("\x97\x0d\xaf\xe0\xce\x08\x5a\xad", 8, GRND_NONBLOCK) = 8
[pid 6043] brk(NULL) = 0x2a368000
[pid 6043] brk(0x2a389000) = 0x2a389000
[pid 6043] openat(AT_FDCWD, "test.txt", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
[pid 6043] newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=0, ...}, AT_EMPTY_PATH) = 0
[pid 6043] write(3, "0.277341\n", 9) = 9
[pid 6043] write(3, "0.014414\n", 9) = 9
[pid 6043] write(3, "3.333333\n", 9) = 9
[pid 6043] close(3) = 0
[pid 6043] exit_group(0) = ?
[pid 6043] +++ exited with 0 +++
<... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 6043
--- SIGCHLD {si_signo=SIGHLD, si_code=CLD_EXITED, si_pid=6043, si_uid=0, si_status=0, si_utime=0, si_stime=1 /* 0.01 s */} ---
close(5) = 0
exit_group(0) = ?
```

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

Вывод

Данная лабораторная работа демонстрирует механизм межпроцессного взаимодействия через pipe. В ходе выполнения лабораторной работы была разработана и отлажена программа на языке Си, реализующая взаимодействие между процессами через механизм pipe (каналы).

Родительский процесс создает два канала, читает данные из stdin и передает их дочернему процессу через pipe. Дочерний процесс перенаправляет стандартные потоки ввода-вывода с помощью dup2, получает данные от родителя, выполняет вычисления и записывает результаты в файл, что подтверждается успешными вызовами write в strace.