

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-211Б-23

Студент: Бачурин Н.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 20.11.24

Москва, 2024

Постановка задачи

Вариант 8.

Цель работы

Приобретение практических навыков в:

- Управление процессами в ОС
- Обеспечение обмена данными между процессами посредством каналов

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<endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. 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 новый процесс, который является копией текущего процесса.
Возвращает идентификатор процесса (PID): В родительском процессе — PID дочернего процесса. В дочернем процессе — 0. При ошибке возвращает -1.
Используется для разделения программы на родительский и дочерний процесс.
- `int pipe(int *fd)`
Создает канал (pipe) для межпроцессной связи, который представляет собой буфер в памяти. `fd[0]`: Конец для чтения. `fd[1]`: Конец для записи. Возвращает 0 при успешном выполнении, -1 при ошибке.
- `int dup2(int oldfd, int newfd)`
Дублирует файловый дескриптор `oldfd` в указанный дескриптор `newfd`. Используется для перенаправления стандартного ввода или вывода. В данной работе используется для перенаправления конца канала pipe в стандартный ввод дочернего процесса.
- `int execl(const char *path, const char *arg, ...)`
запускает исполняемый файл в контексте уже существующего процесса, заменяя предыдущий исполняемый файл.
- `int wait(int *wstatus)`
Ожидает завершения дочернего процесса. Возвращает PID завершившегося дочернего процесса.
- `size_t write(int fd, const void *buf, size_t count)`
Записывает `count` байт данных из буфера `buf` в файловый дескриптор `fd`.
Используется для вывода сообщений об ошибках и передачи данных через канал pipe.

- `size_t read(int fd, void *buf, size_t count)`
читает до `count` байт из файла.
- `FILE * open(const char *pathname, const char *mode)`
Открывает файл в указанном режиме (`r` — только чтение). Используется для чтения входного файла.
- `void exit(int status)`
Завершение выполнения процесса с возвратом кода `status`.
- `int close(int fd)`
закрытие файла, связанного с файловым дескриптором `fd`.

В рамках выполнения данной лабораторной работы была написана программа для работы с процессами, использующая межпроцессорное взаимодействие через каналы.

Работа программы:

Родительский процесс:

- Проверяет аргументы командной строки и открывает входной файл.
- Создает канал для связи с дочерним процессом.
- Создает дочерний процесс с помощью `fork()`.
- В дочернем процессе перенаправляет ввод через канал и запускает программу `child` с помощью `execl`.
- Передает строки из файла в канал.
- Ожидает завершения дочернего процесса.

Дочерний процесс:

- Читает строки из стандартного ввода (перенаправленного из канала).
- Выполняет деление первого числа в строке на последующие.
- Проверяет ошибки, такие как деление на ноль и переполнение.
- Выводит результат или сообщение об ошибке в стандартный вывод.

Особенности реализации:

- Проверка ошибок реализована как в родительском, так и в дочернем процессе.
- Родительский процесс управляет передачей данных через канал.
- Дочерний процесс обрабатывает входные данные и выполняет вычисления.

Программа корректно выполняет межпроцессное взаимодействие.

Ошибки, такие как деление на ноль, неверный ввод или переполнение, обрабатываются и возвращаются с соответствующими сообщениями.

Код программы

parent.c

```

#include <stdint.h>

#include <stdbool.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

void error_print(const char * str)
{

    if (str == NULL)
    {
        write(STDERR_FILENO, "ERROR", 6);
    }
    write(STDERR_FILENO, str, strlen(str));
    exit(EXIT_FAILURE);
}

int main(int args, char *argv[])
{

    if (args != 2)
    {
        error_print("wrong input, try one file\n");
    }

    FILE *file = fopen(argv[1], "r");
    if (!file)
    {
        error_print("file didnt opened\n");
    }

```

```
int fd[2];

if (pipe(fd) == -1)
{
    error_print("pipe failed\n");
}

pid_t pid = fork();

if (pid == 0)
{

    close(fd[1]);

    dup2(fd[0], STDIN_FILENO);
    close(fd[0]);

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

    error_print("execl failed \n");

}
else if (pid < 0)
{
    error_print("fork failed\n");
}
else
{
    close(fd[0]);
```

```
char file_buffer[BUFSIZ];

while (fgets(file_buffer, sizeof(file_buffer), file) != NULL)
{
    if (write(fd[1], file_buffer, strlen(file_buffer)) == -1)
    {
        close(fd[1]);
        error_print("write error\n");

    }
}

close(fd[1]);

fclose(file);
wait(NULL);
}

return 0;
}
```

child.c

```
#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <limits.h>

#include <errno.h>

#include <stdio.h>

#define BUFFER_SIZE 512

typedef enum {

    SUCCESS = 0,

    INVALID_INPUT,

    DIVISION_BY_ZERO,

    INT_OVERFLOW,

} ERROR_CODES;

ERROR_CODES string_to_int(const char *str_number, int *int_result)

{ if (str_number == NULL || int_result == NULL)

    return INVALID_INPUT;

    char *endptr;

    errno = 0;

    long result = strtol(str_number, &endptr, 10);

    if ((result == LONG_MAX || result == LONG_MIN) && errno == ERANGE)

        return INT_OVERFLOW;

    else if (*endptr != '\0' || result > INT_MAX || result < INT_MIN)

        return INVALID_INPUT;
```

```
    *int_result = (int)result;

    return SUCCESS;
}
```

```
void error_print(const char *error_str) {
    if (error_str == NULL) {
        write(STDOUT_FILENO, "ERROR\n", 6);
    } else {
        write(STDOUT_FILENO, error_str, strlen(error_str));
    }
}
```

```
void print_division_result(int result) {
    char buffer[BUFFER_SIZE];

    int length = snprintf(buffer, sizeof(buffer), "Division result: %d\n",
        result); write(STDOUT_FILENO, buffer, length);
}
```

```
int main() {
    char buffer[BUFFER_SIZE];

    while (fgets(buffer, sizeof(buffer), stdin) != NULL)
        { int result = 0;

        int is_first_number = 1;

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

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

        while (token != NULL) {
            int current_value;
```



```

ERROR_CODES error = string_to_int(token, &current_value);

if (error == INT_OVERFLOW) {
    error_print("ERROR: Integer overflow\n");
    return INT_OVERFLOW;
} else if (error == INVALID_INPUT) {
    error_print("ERROR: Invalid input\n");
    return INVALID_INPUT;
} else if (error == SUCCESS) {

    if (current_value == 0) {
        error_print("ERROR: Division by zero\n");
        return DIVISION_BY_ZERO;
    }

    if (is_first_number) {
        result = current_value;
        is_first_number = 0;
    } else {

        if (result == INT_MIN && current_value == -1) {
            error_print("ERROR: Integer overflow on division\n");
            return INT_OVERFLOW;
        }

        result /= current_value;
    }
}

token = strtok(NULL, " ");

}

print_division_result(result);
}

```

```
    return SUCCESS;
}
```

Makefile

```
CC = gcc
```

```
CFLAGS = -Wall -Wextra
```

```
TARGETS = parent child
```

```
all: $(TARGETS)
```

```
parent: parent.c
```

```
    $(CC) $(CFLAGS) -o parent parent.c -lm
```

```
child: child.c
```

```
    $(CC) $(CFLAGS) -o child child.c -lm
```

```
run: all
```

```
    @echo "Running parent program..."
```

```
    strace ./parent file1
```

```
clean:
```

```
    rm -f $(TARGETS)
```

```
.PHONY: all run clean
```

Протокол работы программы

Тест 1:

```

os > lab1 > ≡ file1
1    2 1 1
2    5 2 1
3    4 2 1
4    49 7 1
5    52 52 1
6    52 3 4
7    -52 52 52
8    -8 2 2
9    52 52 0
10

```

Тестирование:

```
lausniko@DESKTOP-MATHSNO:~/os/lab1$ make
```

```
gcc -Wall -Wextra -o parent parent.c -lm
```

```
gcc -Wall -Wextra -o child child.c -lm
```

```
lausniko@DESKTOP-MATHSNO:~/os/lab1$ ./parent file1
```

Division result: 2

Division result: 2

Division result: 2

Division result: 7

Division result: 1

Division result: 4

Division result: 0

Division result: -2

ERROR: Division by zero

```
lausniko@DESKTOP-MATHSNO:~/os/lab1$
```

Strace:

```
lausniko@DESKTOP-MATHSNO:~/os/lab1$ make gcc
```

```
-Wall -Wextra -o parent parent.c -lm gcc -
```

```
Wall -Wextra -o child child.c -lm
```

```
lausniko@DESKTOP-MATHSNO:~/os/lab1$ make run
```

```
Running parent program... strace ./parent
```

```
file1
```

```
execve("./parent", [ "./parent", "file1"], 0x7ffcb1838e38 /* 40 vars */) = 0
```

```

brk(NULL)                                = 0x556317ea5000

arch_prctl(0x3001 /* ARCH_??? */, 0x7fff958387b0) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f578f003000

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=36939, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 36939, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f578eff9000

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\0P\237\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

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"... ,
48, 848) = 48

pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"... , 68, 896) =
68

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, 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, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f578edd0000
mprotect(0x7f578edf8000, 2023424, PROT_NONE) = 0 mmap(0x7f578edf8000, 1658880,
PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f578edf8000

mmap(0x7f578ef8d000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1bd000) = 0x7f578ef8d000

mmap(0x7f578efe6000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f578efe6000

mmap(0x7f578efec000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7f578efec000

close(3)                                  = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f578edcd000

arch_prctl(ARCH_SET_FS, 0x7f578edcd740) = 0

set_tid_address(0x7f578edcda10)           = 36024

set_robust_list(0x7f578edcda20, 24)       = 0

rseq(0x7f578edce0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f578efe6000, 16384, PROT_READ) = 0

mprotect(0x5562ec062000, 4096, PROT_READ) = 0

```

```

mprotect(0x7f578f03d000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f578eff9000, 36939) = 0
getrandom("\x09\xa7\x15\x4c\xb3\xb6\x21\xd6", 8, GRND_NONBLOCK) = 8
brk(NULL) = 0x556317ea5000
brk(0x556317ec6000) = 0x556317ec6000
openat(AT_FDCWD, "file1", O_RDONLY) = 3
pipe2([4, 5], 0) = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f578edcda10) = 36025
close(4) = 0
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=68, ...}, AT_EMPTY_PATH) = 0
read(3, "2 1 1\n5 2 1 \n4 2 1\n49 7 1\n52 52 "..., 4096) = 68 write(5, "2
1 1\n", 6Division result: 2
) = 6
write(5, "5 2 1 \n", 7) = 7
Division result: 2
write(5, "4 2 1\n", 6Division result: 2
) = 6
write(5, "49 7 1\n", 7Division result: 7
) = 7
write(5, "52 52 1\n", 8Division result: 1
) = 8
write(5, "52 3 4\n", 7Division result: 4
) = 7
write(5, "-52 52 52 \n", 11Division result: 0
) = 11
write(5, "-8 2 2\n", 7Division result: -2
) = 7
write(5, "52 52 0\n", 8ERROR: Division by zero
) = 8
write(5, "\n", 1) = -1 EPIPE (Broken pipe)
--- SIGPIPE {si signo=SIGPIPE, si code=SI_USER, si pid=36024, si uid=1000} ---
+++ killed by SIGPIPE +++
make: *** [Makefile:15: run] Broken pipe
lausniko@DESKTOP-MATHSNO:~/os/lab1$

```

Тест 2

```
os > lab1 > ≡ file1
1  2 1 1
2  5 2 1
3  4 2 1
4  49 7 1
5  52 52 1
6  52 3 4
7  -52 52 52
8  -8 2 2
9  74234567890987654345678 3247 234792837
10 -0 -0 -0
11 214 234 234234|
12
13
```

```
lausniko@DESKTOP-MATHSNO:~/os/lab1$ ./parent file1
```

Division result: 2

Division result: 2

Division result: 2

Division result: 7

Division result: 1

Division result: 4

Division result: 0

Division result: -2

ERROR: Division by zero

```
lausniko@DESKTOP-MATHSNO:~/os/lab1$ ./parent file1
```

Division result: 2

Division result: 2

Division result: 2

Division result: 7

Division result: 1

Division result: 4

Division result: 0

Division result: -2

ERROR: Integer overflow

```
lausniko@DESKTOP-MATHSNO:~/os/lab1$ strace ./parent file1
```

```
execve("./parent", [".parent", "file1"], 0x7ffc962281c8 /* 35 vars */) = 0
```

```
brk(NULL) = 0x5618e4203000
```

```
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffebd4c2360) = -1 EINVAL (Invalid argument)
```

```

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f2221f15000

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=36939, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 36939, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f2221f0b000

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\0P\237\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"...,
784, 64) = 784

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0"...,
48, 848) = 48

pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) =
68

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, 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"...,
784, 64) = 784

mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f2221ce2000
mprotect(0x7f2221d0a000, 2023424, PROT_NONE) = 0 mmap(0x7f2221d0a000, 1658880,
PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f2221d0a000

mmap(0x7f2221e9f000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1bd000) = 0x7f2221e9f000

mmap(0x7f2221ef8000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f2221ef8000

mmap(0x7f2221efe000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f2221cdf000

arch_prctl(ARCH_SET_FS, 0x7f2221cdf740) = 0

set_tid_address(0x7f2221cdfa10) = 51150

set_robust_list(0x7f2221cdfa20, 24) = 0

rseq(0x7f2221ce00e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f2221ef8000, 16384, PROT_READ) = 0

mprotect(0x5618aa8e4000, 4096, PROT_READ) = 0

mprotect(0x7f2221f4f000, 8192, PROT_READ) = 0

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

```

```

munmap(0x7f2221f0b000, 36939) = 0

getrandom("\x7e\xd3\x29\x94\x26\x62\x54\x91", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x5618e4203000

brk(0x5618e4224000) = 0x5618e4224000

openat(AT_FDCWD, "file1", O_RDONLY) = 3

pipe2([4, 5], 0) = 0

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f2221cdfa10) = 51151

close(4) = 0

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=123, ...}, AT_EMPTY_PATH) = 0

read(3, "2 1 1\n5 2 1 \n4 2 1\n49 7 1\n52 52 "..., 4096) = 123

write(5, "2 1 1\n", 6Division result: 2
) = 6

write(5, "5 2 1 \n", 7Division result: 2
) = 7

write(5, "4 2 1\n", 6Division result: 2
) = 6

write(5, "49 7 1\n", 7Division result: 7
) = 7

write(5, "52 52 1\n", 8Division result: 1
) = 8

write(5, "52 3 4\n", 7Division result: 4
) = 7

write(5, "-52 52 52 \n", 11Division result: 0
) = 11

write(5, "-8 2 2\n", 7Division result: -2
) = 7

write(5, "74234567890987654345678 3247 234"... , 39ERROR: Integer overflow
)=39

write(5, "-0 -0 -0\n", 9) = -1 EPIPE (Broken pipe)

--- SIGPIPE {si_signo=SIGPIPE, si_code=SI_USER, si_pid=51150, si_uid=1000} ---
+++ killed by SIGPIPE +++

```


Вывод

В процессе выполнения лабораторной работы я научился управлять процессами в операционной системе и реализовывать обмен данными между ними с использованием каналов. В рамках работы была создана и отлажена программа на языке Си, обеспечивающая эффективное взаимодействие процессов путем передачи данных через pipe.