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-214Б-23

Студент: Ефременко К.А.

Преподаватель: Бахарев В.Д. (ФИИТ)

Оценка: _____

Дата: 01.11.24

Москва, 2024

Постановка задачи

Вариант 4.

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

Кратко опишите системные вызовы, которые вы использовали в лабораторной работе.

Использованные системные вызовы:

1. pipe(pipe_fd):

Создает односторонний канал (pipe), который позволяет двум процессам обмениваться данными. Возвращает массив из двух дескрипторов файлов: pipe_fd[0] для чтения из канала и pipe_fd[1] для записи в канал.

2. fork():

Создает новый процесс, являющийся копией родительского процесса. Возвращает идентификатор дочернего процесса, если вызов успешен, 0 в дочернем процессе и -1 в случае ошибки.

3. close(fd):

Закрывает файл с дескриптором fd.

4. read(fd, buf, count):

Считывает count байт данных из файла с дескриптором fd и записывает их в буфер buf. Возвращает количество прочитанных байт, 0, если достигнут конец файла, или -1 в случае ошибки.

5. write(fd, buf, count):

Записывает count байт данных из буфера buf в файл с дескриптором fd. Возвращает количество записанных байт или -1 в случае ошибки.

6. signal(sig, handler):

Устанавливает обработчик сигнала handler для сигнала sig.

7. kill(pid, sig):

Отправляет сигнал sig процессу с идентификатором pid.

8. getppid():

Возвращает идентификатор родительского процесса.

9. wait(NULL):

Ожидает завершения любого дочернего процесса.

10. open(path, flags, mode):

Открывает файл с именем path в заданном режиме. Возвращает дескриптор открытого файла или -1 в случае ошибки.

11. exit(status):

Завершает работу текущего процесса с кодом завершения status.

12. atof(str):

Преобразует строку str в число с плавающей точкой.

13. strncmp(str1, str2, n):

Сравнивает первые n символов строк str1 и str2.

14. strtok(str, delim):

Разбивает строку str на токены по разделителям delim.**1. Заголовки: Включить необходимые заголовки: stdlib.h, unistd.h, signal.h, string.h, fcntl.h, sys/wait.h.**

2. Константы: Определить константу BUFFER_SIZE для размера буфера.

3. Функция write_error(): Написать функцию для вывода ошибки на стандартный поток ошибок.

4. Функция float_to_string(): Написать функцию для преобразования числа с плавающей точкой в строку.

5. Функция write_number_to_file(): Написать функцию для записи результата деления в файл.

6. Функция handle_division(): Написать функцию, выполняющую деление чисел и запись результата в файл.

7. Обработчик сигнала sigusr1_handler(): Написать обработчик сигнала SIGUSR1, завершающий работу процессов при делении на 0.

8. Основная функция main():

- Установить обработчик сигнала SIGUSR1.

- Создать канал pipe().

- Создать дочерний процесс fork().

- В дочернем процессе:

- Закрыть конец записи канала.

- Считывать числа из канала.

- Выполнить деление чисел.

- Записать результат в файл.

- Закрыть конец чтения канала.

- Завершить работу процесса.

- В родительском процессе:

- Закрыть конец чтения канала.

- Считывать числа с клавиатуры

- Записать числа в канал.

- Ожидать завершения дочернего процесса.

- Закрыть конец записи канала.

- Завершить работу процесса.

- Считывать числа с клавиату

Код программы

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

#define BUFFER_SIZE 1024

void write_error(const char* msg) {
    while (*msg) {
        write(STDERR_FILENO, msg++, 1);
    }
}

void float_to_string(float number, char* buffer) {
    int int_part = (int)number;
    float frac_part = number - int_part;

    int length = 0;
    if (int_part < 0) {
        buffer[length++] = '-';
        int_part = -int_part;
    }

    char temp[20];
    int i = 0;
    do {
        temp[i++] = (int_part % 10) + '0';
        int_part /= 10;
    } while (int_part > 0);

    while (i > 0) {
        buffer[length++] = temp[--i];
    }

    if (frac_part > 0) {
        buffer[length++] = '.';
        for (int j = 0; j < 6; j++) {
            frac_part *= 10.0;
            int frac_digit = (int)frac_part;
            buffer[length++] = frac_digit + '0';
            frac_part -= frac_digit;
            if (frac_part <= 0) break;
        }
    }
```

```
    buffer[length] = '\0';  // Завершение строки
}

void write_number_to_file(float number) {
    int file = open("result.txt", O_WRONLY | O_CREAT | O_APPEND, 0644);
    if (file < 0) {
        write_error("Error opening file\n");
        exit(EXIT_FAILURE);
    }

    char buffer[100];
    float_to_string(number, buffer);
    write(file, "Result of division: ", 20);
    write(file, buffer, strlen(buffer));
    write(file, "\n", 1);
    close(file);
}

void handle_division(float* numbers, int count) {
    float result = numbers[0];
    for (int i = 1; i < count; i++) {
        if (numbers[i] == 0) {
            kill(getppid(), SIGUSR1);
            return;
        }
        result /= numbers[i];
    }

    write_number_to_file(result);
}

void sigusr1_handler(int signum) {
    write_error("Error: Division by zero. Terminating processes.\n");
    exit(EXIT_FAILURE);
}

int main() {
    signal(SIGUSR1, sigusr1_handler);

    int pipe_fd[2];
    if (pipe(pipe_fd) == -1) {
        write_error("Error creating pipe\n");
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();
    if (pid < 0) {
        write_error("Error creating process\n");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) { // Дочерний процесс
        close(pipe_fd[1]); // Закрыть конец записи канала

        float numbers[BUFFER_SIZE];
        int count = 0;

        // Чтение чисел из канала
        while (read(pipe_fd[0], &numbers[count], sizeof(float)) > 0) {
            if (numbers[count] == -1.0) break; // Проверка на маркер конца
            count++;
        }

        handle_division(numbers, count);
```

```
        close(pipe_fd[0]);
        exit(EXIT_SUCCESS);
    }
    else { // Родительский процесс
        close(pipe_fd[0]); // Закрыть конец чтения канала

        char input[BUFFER_SIZE];
        float numbers[BUFFER_SIZE];
        int count = 0;

        // Ввод чисел
        while (1) {
            write(STDOUT_FILENO, "Enter numbers (or 'exit' to quit): ", 36);
            read(STDIN_FILENO, input, BUFFER_SIZE);

            if (strncmp(input, "exit", 4) == 0) {
                break;
            }

            char* token = strtok(input, " ");
            while (token != NULL && count < BUFFER_SIZE) {
                numbers[count++] = atof(token);
                token = strtok(NULL, " ");
            }

            numbers[count] = -1.0; // Маркер конца
            write(pipe_fd[1], numbers, sizeof(float) * (count + 1)); // Отправка
чисел включая маркер
            count = 0; // Сброс счетчика для следующего ввода
            break;
        }

        close(pipe_fd[1]);
        wait(NULL); // Ожидание завершения дочернего процесса
    }

    return 0;}
```

Протокол работы программы

```
execve("./parent", ["./parent"], 0x7ffe995d45e8 /* 29 vars */) = 0
brk(NULL)                               = 0x55d2b12ac000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7ffa39d03000
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=19711, ...}) = 0
mmap(NULL, 19711, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ffa39cfe000
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) = 0x7ffa39aec000
mmap(0x7ffa39b14000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7ffa39b14000
mmap(0x7ffa39c9c000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7ffa39c9c000
mmap(0x7ffa39ceb000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7ffa39ceb000
mmap(0x7ffa39cf1000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7ffa39cf1000
close(3)                                = 0
```

```
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7ffa39ae9000
arch_prctl(ARCH_SET_FS, 0x7ffa39ae9740) = 0
set_tid_address(0x7ffa39ae9a10)         = 685
set_robust_list(0x7ffa39ae9a20, 24)     = 0
rseq(0x7ffa39aea060, 0x20, 0, 0x53053053) = 0
mprotect(0x7ffa39ceb000, 16384, PROT_READ) = 0
mprotect(0x55d2b0f1b000, 4096, PROT_READ) = 0
mprotect(0x7ffa39d3b000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7ffa39cfe000, 19711)           = 0
pipe2([3, 4], 0)                        = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace:
Process 686 attached
, child_tidptr=0x7ffa39ae9a10) = 686
[pid   686] set_robust_list(0x7ffa39ae9a20, 24 <unfinished ...>
[pid   685] close(3 <unfinished ...>
[pid   686] <... set_robust_list resumed>) = 0
[pid   685] <... close resumed>)        = 0
[pid   685] write(1, "Input numbers: \0", 16 <unfinished ...>
Input numbers: [pid   686] close(4 <unfinished ...>
[pid   685] <... write resumed>)        = 16
[pid   686] <... close resumed>)        = 0
[pid   685] read(0,  <unfinished ...>
[pid   686] dup2(3, 0)                  = 0
[pid   686] close(3)                    = 0
[pid   686] execve("./child", ["child"], 0x7ffdf4fca5b8 /* 29 vars */) = 0
[pid   686] brk(NULL)                   = 0x55b15c583000
[pid   686] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f3918667000
[pid   686] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or
directory)
[pid   686] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
[pid   686] fstat(3, {st_mode=S_IFREG|0644, st_size=19711, ...}) = 0
[pid   686] mmap(NULL, 19711, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f3918662000
[pid   686] close(3)                    = 0
[pid   686] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC)
= 3
[pid   686] 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
[pid   686] 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
[pid   686] fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
[pid   686] 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
[pid   686] mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f3918450000
[pid   686] mmap(0x7f3918478000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f3918478000
[pid   686] mmap(0x7f3918600000, 323584, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7f3918600000
[pid   686] mmap(0x7f391864f000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f391864f000
[pid   686] mmap(0x7f3918655000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f3918655000
[pid   686] close(3)                    = 0
[pid   686] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7f391844d000
[pid   686] arch_prctl(ARCH_SET_FS, 0x7f391844d740) = 0
[pid   686] set_tid_address(0x7f391844da10) = 686
[pid   686] set_robust_list(0x7f391844da20, 24) = 0
[pid   686] rseq(0x7f391844e060, 0x20, 0, 0x53053053) = 0
[pid   686] mprotect(0x7f391864f000, 16384, PROT_READ) = 0
[pid   686] mprotect(0x55b15bef7000, 4096, PROT_READ) = 0
[pid   686] mprotect(0x7f391869f000, 8192, PROT_READ) = 0
```

```
[pid   686] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
[pid   686] munmap(0x7f3918662000, 19711) = 0
[pid   686] read(0,  <unfinished ...>
[pid   685] <... read resumed>0x7ffdf4fca370, 255) = ? ERESTARTSYS (To be restarted
if SA_RESTART is set)
[pid   686] <... read resumed>0x7ffdededac4c, 4) = ? ERESTARTSYS (To be restarted if
SA_RESTART is set)
[pid   685] --- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
[pid   686] --- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
[pid   685] read(0,  <unfinished ...>
[pid   686] read(0,
 <unfinished ...>
[pid   685] <... read resumed>"\n", 255) = 1
[pid   685] write(4, "\0\0\0\0", 4)     = 4
[pid   686] <... read resumed>"\0\0\0\0", 4) = 4
[pid   686] read(0,  <unfinished ...>
[pid   685] write(4, "", 0 <unfinished ...>
[pid   686] <... read resumed>"", 0)    = 0
[pid   686] openat(AT_FDCWD, "output.txt", O_WRONLY|O_CREAT|O_TRUNC, 0644
<unfinished ...>
[pid   685] <... write resumed>)        = 0
[pid   685] close(4)                    = 0
[pid   685] wait4(-1,  <unfinished ...>
[pid   686] <... openat resumed>)       = 3
[pid   686] write(3, "Result: 0.000000\n", 17) = 17
[pid   686] close(3)                    = 0
[pid   686] exit_group(0)               = ?
[pid   686] +++ exited with 0 +++
<... wait4 resumed>NULL, 0, NULL)       = 686
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=686, si_uid=1000,
si_status=0, si_utime=0, si_stime=2 /* 0.02 s */} ---
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

Во время выполнения данной лабораторной работы я узнал много полезного про системные вызовы и про создание отдельных процессов. Лучше узнал ОС Ubuntu. Основная сложность была в изучении необходимых материалов коих было не очень много. В будущем я бы хотел попробовать написать свой простой драйвер.