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-210Б-23

Студент: Шведов А.И.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 03.10.24

Москва, 2024

Постановка задачи

Вариант 3:

Пользователь вводит команды вида: «число число число<endline>». 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); – создаёт пайп и помещает дескрипторы в fd[0], fd[1], для чтения и записи.
- int write(int fd, const void* buff, int count); – записывает по дескриптору fd count байт из buff.
- void exit(int number); – вызывает нормальное завершение программы с кодом number.
- int dup2(int fd1, int fd2); – делает эквиваентными дескрипторы fd1 и fd2.
- int exec(char* path, const char* argc); – заменяет текущий процесс на процесс path, с аргументами argc;
- int close(int fd); – закрывает дескриптор fd.
- pid_t wait(int status) — функция, которая приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится,

Я создал два файла client и server.

В файле server нужно сначала проверить количество аргументов и если их слишком мало нужно вернуть ошибку и закончить процесс. Далее нужно получить полный путь до файла, имя которого передалось первым аргументом командной строки. Потом нужно создать канал для переопределения потока ввода для дочернего процесса. Для основного процесса вывести его ID. Запустить дочерний процесс, передаём аргуметами путь до файла и ждём когда он закончится.

В файле client будет находиться реализация дочернего процесса. Открываем файл для записи, который мы получили аргументом командной строки. Считываем построчно из потока ввода в буффер символы. Из массива чаров делаем массив интов. Прожодимся по масииву и делим первое число на последующие, записывая результаты в открытый файл. При этом нужно проверять деление на 0 и возможную пустую строку, при появлении которых программа должна завершиться.

Код программы

Posix-ipc-server.c

```c
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>
```

```c
static char CLIENT_PROGRAM_NAME[] = "posix_ipc-client";

int main(int argc, char **argv) {
    if (argc == 1) {
        char msg[1024];
        uint32_t len = snprintf(msg, sizeof(msg) - 1, "usage: %s filename\n", argv[0]);
        write(STDERR_FILENO, msg, len);
        exit(EXIT_SUCCESS);
    }

    char progpath[1024];
    {
        ssize_t len = readlink("/proc/self/exe", progpath, sizeof(progpath) - 1);
        if (len == -1) {
            const char msg[] = "error: failed to read full program path\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }
        while (progpath[len] != '/') --len;
        progpath[len] = '\0';
    }

    // Open pipe
    int channel[2];
    if (pipe(channel) == -1) {
        const char msg[] = "error: failed to create pipe\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
    const pid_t child = fork();

    switch (child) {
        case -1: {
            const char msg[] = "error: failed to spawn new process\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        } break;

        case 0: {
            pid_t pid = getpid();

            dup2(STDIN_FILENO, channel[STDIN_FILENO]);
            close(channel[STDOUT_FILENO]);

            {
                char msg[64];
                const int32_t length = snprintf(msg, sizeof(msg), "%d: I'm a child\n",
pid);
                write(STDOUT_FILENO, msg, length);
            }

            {
                char path[1024];
```

```
                snprintf(path, sizeof(path) - 1, "%s/%s", progpath, CLIENT_PROGRAM_NAME);
                char *const args[] = {CLIENT_PROGRAM_NAME, argv[1], NULL};

                int32_t status = execv(path, args);

                if (status == -1) {
                    const char msg[] = "error: failed to exec into new exectuable
image\n";
                    write(STDERR_FILENO, msg, sizeof(msg));
                    exit(EXIT_FAILURE);
                }
            }
        } break;

        default: {
            pid_t pid = getpid();
            {
                char msg[64];
                const int32_t length =
                    snprintf(msg, sizeof(msg), "%d: I'm a parent, my child has PID %d\n",
pid, child);
                write(STDOUT_FILENO, msg, length);
            }
            int child_status;
            wait(&child_status);

            if (child_status != EXIT_SUCCESS) {
                const char msg[] = "error: child exited with error\n";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(child_status);
            }
        } break;
    }
}
```

Posix-ipc-client.c

```
#include <fcntl.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <ctype.h>

int MAX_LENGTH = 1024;
int MAX_NUMBERS = 100;

void convertStringToIntArray(const char *str, int intArray[], int *size) {
    int i = 0, num = 0;
```

```
    *size = 0;
    while (str[i] != '\n') {
        while (str[i] != '\n' && isspace(str[i])) {
            i++;
        }
        if (str[i] == '\n') {
            break;
        }
        num = 0;
        while (str[i] != '\n' && isdigit(str[i])) {
            num = num * 10 + (str[i] - '0');
            i++;
        }
        if (*size < MAX_NUMBERS) {
            intArray[*size] = num;
            (*size)++;
        }
    }
}

int main(int argc, char **argv) {
    char buf[4096];
    ssize_t bytes;

    pid_t pid = getpid();

    int32_t file = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC | O_APPEND, 0600);
    if (file == -1) {
        const char msg[] = "error: failed to open requested file\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    {
        char msg[128];
        int32_t len =
            snprintf(msg, sizeof(msg) - 1,
                     "%d: Start typing lines of text. Press 'Ctrl-D' or 'Enter' with no
input to exit\n", pid);
        write(STDOUT_FILENO, msg, len);
    }

    while (bytes = read(STDIN_FILENO, buf, sizeof(buf))) {
        if (bytes < 0) {
            const char msg[] = "error: failed to read from stdin\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        } else if (buf[0] == '\n') {
            break;
        }

        {
            int array[MAX_NUMBERS];
            int size = 0;
```

```
            convertStringToIntArray(buf, array, &size);
            if (size < 2) {
                const char msg[] = "You have written few numbers\n";
                int32_t written = write(STDOUT_FILENO, msg, sizeof(msg));
                if (written != sizeof(msg)) {
                    const char msg[] = "error: failed to write to file\n";
                    write(STDERR_FILENO, msg, sizeof(msg));
                    exit(EXIT_FAILURE);
                }
            }
            int divisible = array[0];
            for (int i = 1; i != size; ++i) {
                if (array[i] == 0) {
                    const char ermsg[] = "Division by zero\n";
                    write(STDERR_FILENO, ermsg, sizeof(ermsg));
                    exit(EXIT_FAILURE);
                }
                char msg[32];
                int32_t len = snprintf(msg, sizeof(msg) - 1, "%d : %d = %lf\n", divisible,
array[i],
                                       ((float)divisible / array[i]));
                int32_t written = write(file, msg, len);
                if (written != len) {
                    const char msg[] = "error: failed to write to file\n";
                    write(STDERR_FILENO, msg, sizeof(msg));
                    exit(EXIT_FAILURE);
                }
            }
        }
    }

    const char term = '\0';
    write(file, &term, sizeof(term));

    close(file);
}
```

Протокол работы программы

Тестирование:

```
859: I'm a parent, my child has PID 860
860: I'm a child
860: Start typing lines of text. Press 'Ctrl-D' or 'Enter' with no input to exit
1000 23 45 5 1
228 2 2 8
1337 1 33 7
4488 18 47 46
```

```
1000 : 23 = 43.478260
1000 : 45 = 22.222221
1000 : 5 = 200.000000
1000 : 1 = 1000.000000
228 : 2 = 114.000000
228 : 2 = 114.000000
228 : 8 = 28.500000
1337 : 1 = 1337.000000
1337 : 33 = 40.515152
1337 : 7 = 191.000000
4488 : 18 = 249.333328
4488 : 47 = 95.489365
4488 : 46 = 97.565216
```

```
1328: I'm a parent, my child has PID 1329
1329: I'm a child
1329: Start typing lines of text. Press 'Ctrl-D' or 'Enter' with no input to exit
1000 34 234 123 213
43425 24524 1324 1234 1234
324 123 0
Division by zero
error: child exited with error
```

```
1000 : 34 = 29.411764
1000 : 234 = 4.273504
1000 : 123 = 8.130081
1000 : 213 = 4.694836
43425 : 24524 = 1.770714
43425 : 1324 = 32.798340
43425 : 1234 = 35.190437
43425 : 1234 = 35.190437
324 : 123 = 2.634146
```

Strace:

execve("./posix_ipc-server", ["./posix_ipc-server", "1.txt"], 0x7fff32a4d6a8 /* 27 vars */) = 0

brk(NULL) = 0x558d6acc4000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffef97ad590) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7feb01cd2000

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=37207, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 37207, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7feb01cc8000

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\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) = 0x7feb01a9f000

mprotect(0x7feb01ac7000, 2023424, PROT_NONE) = 0

mmap(0x7feb01ac7000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7feb01ac7000

mmap(0x7feb01c5c000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7feb01c5c000

mmap(0x7feb01cb5000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7feb01cb5000

mmap(0x7feb01cbb000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7feb01cbb000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7feb01a9c000

arch_prctl(ARCH_SET_FS, 0x7feb01a9c740) = 0

set_tid_address(0x7feb01a9ca10) = 2160

set_robust_list(0x7feb01a9ca20, 24) = 0

rseq(0x7feb01a9d0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7feb01cb5000, 16384, PROT_READ) = 0

mprotect(0x558d695eb000, 4096, PROT_READ) = 0

mprotect(0x7feb01d0c000, 8192, PROT_READ) = 0

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

munmap(0x7feb01cc8000, 37207) = 0

readlink("/proc/self/exe", "/mnt/c/Users/mrshv/OneDrive/Desk"..., 1023) = 55

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

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7feb01a9ca10) = 2161

2161: I'm a child

getpid() = 2160

write(1, "2160: I'm a parent, my child has"..., 422160: I'm a parent, my child has PID 2161

) = 42

wait4(-1, 2161: Start typing lines of text. Press 'Ctrl-D' or 'Enter' with no input to exit

123 6 6 23 122

122 4 6 8 9

1 45 67 9

2 9 0

Division by zero

[{WIFEXITED(s) && WEXITSTATUS(s) == 1}], 0, NULL) = 2161

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=2161, si_uid=1000, si_status=1, si_utime=0, si_stime=0} ---

write(2, "error: child exited with error\n\0", 32error: child exited with error) = 32

exit_group(256) = ?

+++ exited with 0 +++

Вывод

Было интересно решать лабораторную работу. Я научился использовать некоторые системные вызовы, а также обмениваться данными между процессами с помощью каналов. Было интересно узнать как можно писать программы используя их. Возникли трудности с обработкой всех ошибок системных вызовов в программе.