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: Гусев С.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 28.10.24

Москва, 2024

Постановка задачи

Вариант 2.

Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип float. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- int channel[2];
  pipe(channel); – создает два канала связи.
- const pid_t child = fork(); – создает дочерний процесс.
- pid_t pid = getpid(); – получает номер текущего процесса.
- dup2(STDIN_FILENO, channel[STDIN_FILENO]); – перенаправляет стандартный ввод на дескриптор канала связи.
- int32_t status = execv(path, args); – ● заменяет код новым программным кодом, указанным в path.
- wait(&child_status); – родительский процесс ждет завершения дочернего процесса.

Решение:

1. Обрабатываю путь переданный через аргументы командной строки.
2. С помощью функций написанных выше связываю родительский процесс с дочерним(передаю обработанный ввод).
3. В дочернем процессе считываю строку (используя read()) и проверяю находящиеся там символы на соответствие вводу указанному в задании («число число число»), параллельно заменяя знаки пробела на ‘\0’(дабы потом удобно было переводить в числа типа float).
4. Циклом прохожу по строке и складываю числа (переводя их в тип float с помощью atof).
5. Ответ вывожу в канал связи с родительским процессом (использую write()).

Код программы

parent.c

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


static char CLIENT_PROGRAM_NAME[] = "child";

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';
}

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 executable
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;
    }
}
```

child.c

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

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

```
    char ans[4096];

    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)))) {
        float sum = 0;
        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;
        }

        {
            char msg[32];
            int32_t len = snprintf(msg, sizeof(msg) - 1,
                                   "Sum of your numbers: ");

            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);
            }
        }

        {
            buf[bytes] = '\0';
            int point_cnt = 0;
            int numb_cnt = 1;
            for (int i = 0; i < bytes - 1; ++i) {
                if (isdigit(buf[i]) || (buf[i] == '.' && !point_cnt)) {
                    if (buf[i] == '.') point_cnt++;
```

```
                    continue;
                }
                if (buf[i] == ' ') {
                    point_cnt = 0;
                    buf[i] = '\0';
                    continue;
                }
                const char msg[] = "error: value is not a number\n";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            }
        }

        {
            char *ptr = buf;
            float numb = 0;
            sum += atof(ptr);
            for(int i = 0; i < bytes - 1; ++i) {
                if (buf[i] == '\0' && bytes > i + 1) {
                    numb = atof(ptr + i + 1);
                    sum += numb;
                }
            }

            size_t ansLen = snprintf(ans, sizeof(ans), "%.5f\n", sum);
            int32_t written = write(file, ans, ansLen);
            if (written != ansLen) {
                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:

```
$ ./parent filename.txt
633: I'm a parent, my child has PID 634
634: I'm a child
634: Start typing lines of text. Press 'Ctrl-D' or 'Enter' with no input to exit
1.2 1.3 1.5
1.2345 53.124 8911.132
0.00134 0.12209 0.00252 0.01919 1.12602
```

```
1234 3567 2378.541 5678 7890 6.5678 45.45678
181
0.678 567.672

$ cat filename.txt
Sum of your numbers: 4.00000
Sum of your numbers: 8965.49023
Sum of your numbers: 1.27116
Sum of your numbers: 20799.56641
Sum of your numbers: 181.00000
Sum of your numbers: 568.34998
```

Strace:

```
$ strace -f ./parent
execve("./parent", ["./parent"], 0x7ffc60e304b8 /* 26 vars */) = 0
brk(NULL)                               = 0x5569e780c000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f6c98685000
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=19787, ...}) = 0
mmap(NULL, 19787, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f6c98680000
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) = 0x7f6c9846e000
mmap(0x7f6c98496000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7f6c98496000
mmap(0x7f6c9861e000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7f6c9861e000
mmap(0x7f6c9866d000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1fe000) = 0x7f6c9866d000
mmap(0x7f6c98673000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7f6c98673000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f6c9846b000
arch_prctl(ARCH_SET_FS, 0x7f6c9846b740) = 0
set_tid_address(0x7f6c9846ba10)         = 3011
set_robust_list(0x7f6c9846ba20, 24)     = 0
rseq(0x7f6c9846c060, 0x20, 0, 0x53053053) = 0
mprotect(0x7f6c9866d000, 16384, PROT_READ) = 0
mprotect(0x5569e77a4000, 4096, PROT_READ) = 0
mprotect(0x7f6c986bd000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f6c98680000, 19787)           = 0
write(2, "usage: ./parent filename\n", 25usage: ./parent filename
```

```
) = 25
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

В результате выполнения лабораторной работы удалось познакомиться с системными вызовами (такими как pipe(), fork(), dup2(), execv(), wait()) и реализовать программу сложения нескольких чисел записанных в строку через пробел. Проблем при выполнении работы не возникло.