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-24

Студент: Ходырев Д.И.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 07.10.24

Москва, 2024

Постановка задачи

Вариант 2.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

2 вариант) Пользователь вводит команды вида: «число число число<endline>». 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:

- *pid_t fork(void)*; – создает дочерний процесс.
- *int pipe(int *fd)*; – создает однонаправленный канал для межпроцессорного взаимодействия;
- *int dup2(int oldfd, int newfd)*; – создает копию файлового дескриптора *oldfd* в указанном дескрипторе *newfd*.
- *int execv(const char *path, char *const argv[])*; — заменяет текущий образ
- процесса на новый исполняемый файл.
- *int open(const char* pathname, int flags, mode_t mode)*; – открывает файл по указанному пути с заданными флагами и правами доступа.
- *ssize_t read(int fd, void* buf, size_t count)*; – читает данные из файлового дескриптора в буфер.
- *ssize_t write(int fd, const void* buf, size_t count)*; – записывает данные из буфера в файловый дескриптор.
- *int close(int fd)*; – закрывает файловый дескриптор.
- *pid_t wait(int* status)*; – ожидает изменения состояния указанного дочернего процесса.
- *pid_t getpid(void)*; — возвращает PID текущего процесса. Используется для отладочного вывода.

В рамках лабораторной работы было создано два файла – parent и child. Первый принимает в качестве параметра имя файла, в который будет записываться результат работы программы. Второй файл нужен для самой обработки данных и записи в файл. Они связываются друг с другом посредством канала.

Первый файл принимает сроки с вещественными числами. Если введена пустая строка, то программа завершается. Полученные строки отправляются по каналу во вторую программу. Она считывает эти строки как стандартный поток ввода и находит сумму всех чисел в этой строке, после чего записывает ее в отдельную строку в файле вывода.

Код программы

parent.c

```
#include <stdint.h>
#include <stdbool.h>

#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>

static char SERVER_PROGRAM_NAME[] = "child";

int main(int argc, char* argv[]) {
    // Проверка правильности ввода команды
    if (argc != 2) {
        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);
    }

    // Находим путь к директории
    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';
    }

    // Открываем пайп между клиентом и сервером
    int client_to_server[2];
    if (pipe(client_to_server) == -1) {
        const char msg[] = "error: failed to create pipe\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    // Создаем новый процесс
    const pid_t child = fork();
```

```
switch (child) {
    case -1: // NOTE: Kernel fails to create another process
        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();

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

        close(client_to_server[1]);

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

        {
            char path[2048];
            snprintf(path, sizeof(path) - 1, "%s/%s", progpath, SERVER_PROGRAM_NAME);

            char* const args[] = {SERVER_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(); // NOTE: Get parent PID

            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);
        }
```

```
            close(client_to_server[0]);

            char buf[4096];
            ssize_t bytes;

            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') {
                  // NOTE: When Enter is pressed with no input, then exit client
                  break;
               }

               write(client_to_server[1], buf, bytes);

               // bytes = read(server_to_client[0], buf, sizeof(bytes));
               // write(STDOUT_FILENO, buf, bytes);
            }

            close(client_to_server[1]);

            wait(NULL);
         break;
      }
   }
```

child.c

```
#include <stdint.h>
#include <stdbool.h>
#include <ctype.h>

#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

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

   pid_t pid = getpid();

   // NOTE: `O_WRONLY` only enables file for writing
   // NOTE: `O_CREAT` creates the requested file if absent
   // NOTE: `O_TRUNC` empties the file prior to opening
   // NOTE: `O_APPEND` subsequent writes are being appended instead of overwritten
```

```
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);
}

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);
    }

    // NOTE: Transform data
    double sum = 0.0, tmp;
    char* ptr = buf;
    char* end;

    while (ptr < &(buf[bytes - 1])) {
        while (isspace(*ptr)) {
            ++ptr;
        }
        if (ptr == &(buf[bytes - 1])) {
            break;
        }

        tmp = strtod(ptr, &end);

        if (!isspace(*end) && *end != '\n') {
            const char msg[] = "error: uexpected symbol\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }

        sum += tmp;
        ptr = end;
    }

    char out[512];
    int32_t out_len= snprintf(out, sizeof(out), "%.4f\n", sum);
    if (out_len < 5) {
        const char msg[] = "error: failed to write to file\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
    out[out_len] = '\n';

    {
```

```
            // NOTE: Log to file
            int32_t written = write(file, out, out_len);
            if (written == 0) {
                const char msg[] = "error: failed to write to file\n";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            }
        }
    }

    // NOTE: Write terminator to the end file
    if (bytes == 0) {
        const char term = '\0';
        write(file, &term, sizeof(term));
    }

    close(file);
}
```

Протокол работы программы

```
d_khod@D-Khod-Laptop:/mnt/c/Users/g66xq/Уник/Лабы_по_ОС/mai_OS_labs/laba1/src$ ./parent result.txt
718: I'm a parent, my child has PID 719
719: I'm a child
1 2 3 4 5 6 7 8 9
0.5 0.5 0.6 6.7 5.2 7.2 6.9
0.4501 0.203 1.089
12345

d_khod@D-Khod-Laptop:/mnt/c/Users/g66xq/Уник/Лабы_по_ОС/mai_OS_labs/laba1/src$ cat result.txt
45.0000
27.6000
1.7421
12345.0000
d_khod@D-Khod-Laptop:/mnt/c/Users/g66xq/Уник/Лабы_по_ОС/mai_OS_labs/laba1/src$ strace -f ./parent
result.txt
execve("./parent", ["./parent", "result.txt"], 0x7ffeefdd5bc0 /* 27 vars */) = 0
brk(NULL)                               = 0x555cc759b000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fb982afa000
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=30919, ...}) = 0
mmap(NULL, 30919, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fb982af2000
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) = 0x7fb9828e0000
```

```
mmap(0x7fb982908000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fb982908000
mmap(0x7fb982a90000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7fb982a90000
mmap(0x7fb982adf000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7fb982adf000
mmap(0x7fb982ae5000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fb982ae5000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fb9828dd000
arch_prctl(ARCH_SET_FS, 0x7fb9828dd740) = 0
set_tid_address(0x7fb9828dda10)         = 920
set_robust_list(0x7fb9828dda20, 24)     = 0
rseq(0x7fb9828de060, 0x20, 0, 0x53053053) = 0
mprotect(0x7fb982adf000, 16384, PROT_READ) = 0
mprotect(0x555cc72b7000, 4096, PROT_READ) = 0
mprotect(0x7fb982b32000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7fb982af2000, 30919)           = 0
readlink("/proc/self/exe", "/mnt/c/Users/g66xq/\320\243\320\275\320\270\320\272/\320\233\320\260"..., 1023) =
75
pipe2([3, 4], 0)                        = 0
clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process 921 attached
, child_tidptr=0x7fb9828dda10) = 921
[pid   921] set_robust_list(0x7fb9828dda20, 24 <unfinished ...>
[pid   920] getpid( <unfinished ...>
[pid   921] <... set_robust_list resumed>) = 0
[pid   920] <... getpid resumed>)       = 920
[pid   921] getpid( <unfinished ...>
[pid   920] write(1, "920: I'm a parent, my child has "..., 40 <unfinished ...>
920: I'm a parent, my child has PID 921
[pid   921] <... getpid resumed>)       = 921
[pid   920] <... write resumed>)        = 40
[pid   920] close(3 <unfinished ...>
[pid   921] write(1, "921: I'm a child\n", 17 <unfinished ...>
921: I'm a child
[pid   920] <... close resumed>)        = 0
[pid   921] <... write resumed>)        = 17
[pid   920] read(0,  <unfinished ...>
[pid   921] close(4)                    = 0
[pid   921] dup2(3, 0)                  = 0
[pid   921] close(3)                    = 0
[pid   921]
execve("/mnt/c/Users/g66xq/\320\243\320\275\320\270\320\272/\320\233\320\260\320\261\321\213_\320\277\32
0\276_\320\236\320\241/mai_OS_labs/laba1/src/child", ["child", "result.txt"], 0x7fffed63dc90 /* 27 vars */) = 0
[pid   921] brk(NULL)                   = 0x558eafe7c000
[pid   921] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f87bf4d5000
[pid   921] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
[pid   921] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
[pid   921] fstat(3, {st_mode=S_IFREG|0644, st_size=30919, ...}) = 0
[pid   921] mmap(NULL, 30919, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f87bf4cd000
[pid   921] close(3)                    = 0
[pid   921] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
[pid   921] 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   921] 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   921] fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
[pid   921] 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   921] mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f87bf2bb000
[pid   921] mmap(0x7f87bf2e3000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f87bf2e3000
[pid   921] mmap(0x7f87bf46b000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1b0000) = 0x7f87bf46b000
[pid   921] mmap(0x7f87bf4ba000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f87bf4ba000
[pid   921] mmap(0x7f87bf4c0000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f87bf4c0000
[pid   921] close(3)                    = 0
[pid   921] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f87bf2b8000
[pid   921] arch_prctl(ARCH_SET_FS, 0x7f87bf2b8740) = 0
[pid   921] set_tid_address(0x7f87bf2b8a10) = 921
[pid   921] set_robust_list(0x7f87bf2b8a20, 24) = 0
[pid   921] rseq(0x7f87bf2b9060, 0x20, 0, 0x53053053) = 0
[pid   921] mprotect(0x7f87bf4ba000, 16384, PROT_READ) = 0
[pid   921] mprotect(0x558eafa27000, 4096, PROT_READ) = 0
[pid   921] mprotect(0x7f87bf50d000, 8192, PROT_READ) = 0
[pid   921] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
[pid   921] munmap(0x7f87bf4cd000, 30919) = 0
[pid   921] getpid()                    = 921
[pid   921] openat(AT_FDCWD, "result.txt", O_WRONLY|O_CREAT|O_TRUNC|O_APPEND, 0600) = 3
[pid   921] read(0, 1.2 5.2
 <unfinished ...>
[pid   920] <... read resumed>"1.2 5.2\n", 4096) = 8
[pid   920] write(4, "1.2 5.2\n", 8)    = 8
[pid   921] <... read resumed>"1.2 5.2\n", 4096) = 8
[pid   920] read(0,  <unfinished ...>
[pid   921] write(3, "6.4000\n", 7)     = 7
[pid   921] read(0,
 <unfinished ...>
[pid   920] <... read resumed>"\n", 4096) = 1
[pid   920] close(4)                    = 0
[pid   921] <... read resumed>"", 4096) = 0
[pid   920] wait4(-1,  <unfinished ...>
[pid   921] write(3, "\0", 1)           = 1
[pid   921] close(3)                    = 0
[pid   921] exit_group(0)               = ?
[pid   921] +++ exited with 0 +++
<... wait4 resumed>NULL, 0, NULL)       = 921
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=921, si_uid=1000, si_status=0, si_utime=0,
si_stime=1 /* 0.01 s */} ---
exit_group(0)                           = ?
+++ exited with 0 +++
d_khod@D-Khod-Laptop:/mnt/c/Users/g66xq/Уник/Лабы_по_ОС/mai_OS_labs/laba1/src$ cat result.txt
6.4000
d_khod@D-Khod-Laptop:/mnt/c/Users/g66xq/Уник/Лабы_по_ОС/mai_OS_labs/laba1/src$
```

Вывод

В ходе выполнения лабораторной работы я изучил основные системные вызовы в системе Linux для работы с межпроцессным взаимодействием. Была разработана программа для нахождения суммы вещественных чисел в строке и записи результата в файл. Она демонстрирует возможности создание одними процессами других, а также создание связей между процессами и передачу данных между ними.