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: М80-214Б-23
Студент: Камышников И.С.
Преподаватель: Бахарев В.Д.
Оценка:_____
Дата: 13.02.25

Москва, 2025

Постановка задачи

Вариант 6.
Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода.

Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

В файле записаны команды вида: «число число число». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип int. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Общий метод и алгоритм решения
Использованные системные вызовы:
pid_t fork(void); – создает дочерний процесс.
int pipe(int *fd); – создает канал.
int read(int fd, void* buf, size_t count); – считывает count байт из fd в buf.
int write(int fd, void* buf, size_t count); – записывает count байт из buf в fd.
int open(const char *pathname, int flags, ...); – открывает файловый дескриптор.
int close(int fd); – закрывает файловый дескриптор.
pid_t waitpid(pid_t pid, int *_Nullable wstatus, int options); – ожидает завершения процесса.

Описание работы программы

Программа main
Инициализирует переменную prog_name значением из argw[1], если оно есть, или строкой "./child.out".
Создает канал (pipe).
Считывает имя входного файла из stdin.
Открывает файловый дескриптор и проверяет успешность открытия.
Вызывает fork().

В родительском процессе:
Создается буфер, выполняется последовательное чтение из pipe1 и запись в стандартный вывод.
Родитель ждет завершения дочернего процесса (waitpid).

В дочернем процессе:
Стандартный вход перенаправляется на открытый файл (dup2).
Стандартный вывод направляется в канал (pipe1).
Выполняется execv() для запуска prog_name.

Программа child
Читает строки из стандартного ввода (read_line).
Каждая строка разбивается на числа, которые суммируются.
Результат записывается в буфер и выводится.

Код программы

main.c
```
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>

#include "func.h"

int main(int argc, char *argw[]) {
   char *prog_name;
   if (argc >= 2)
      prog_name = argw[1];
   else
      prog_name = "./child.out";

   int pipefd[2];
   if (pipe(pipefd) == -1) {
      print(STDERR_FILENO, "ERROR: failed to create pipe\n");
      exit(-1);
   }
```

```
int buf_size = 64;
char *fname = malloc(buf_size);
*fname = 0;
if (read_line(STDIN_FILENO, &fname, &buf_size) <= 0) {
    print(STDERR_FILENO,
          "ERROR: failed read filename from standart input\n");
    exit(-1);
}
int fd = open(fname, O_RDONLY);
if (fd == -1) {
    print(STDERR_FILENO, "ERROR: failed to open file: \"");
    print(STDERR_FILENO, fname);
    print(STDERR_FILENO, "\"\n");
    print(STDERR_FILENO, strerror(errno));
    print(STDERR_FILENO, "\n");
    exit(-1);
}
free(fname);

pid_t pid = fork();
if (pid == 0) {
    dup2(fd, STDIN_FILENO);
    close(fd); // close fd, it was duplicated by dup2

    close(pipefd[0]); // close read. we only need to write to pipe
    dup2(pipefd[1], STDOUT_FILENO); // redirect output into pipe
    dup2(pipefd[1], STDERR_FILENO);
    close(pipefd[1]); // close pipe, it was duplicated

    char *argv[] = {prog_name, "", NULL};
    if (execv(prog_name, argv) == -1) {
        print(STDERR_FILENO, "ERROR: failed to launch process \"");
        print(STDERR_FILENO, prog_name);
        print(STDERR_FILENO, "\"\n");
        print(STDERR_FILENO, strerror(errno));
        print(STDERR_FILENO, "\n");
        exit(-1);
    }
} else if (pid == -1) {
```

```
        print(STDERR_FILENO, "ERROR: failed to fork process\n");
        print(STDERR_FILENO, strerror(errno));
        print(STDERR_FILENO, "\n");
        exit(-1);
    } else {
        char buffer[128];
        close(pipefd[1]);
        while (1) {
            int n = read(pipefd[0], buffer, sizeof(buffer));
            if (n == 0)
                break;
            write(STDOUT_FILENO, buffer, n);
        }
        waitpid(pid, 0, 0);
    }
    return 0;
}
```

child.c

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#include "func.h"

int main(void) {
    int buf_size = 2;
    char *input_buffer = malloc(buf_size);
    memset(input_buffer, 0, buf_size);
    char output_buffer[64];

    if (!input_buffer) {
        return -1;
    }
    while (1) {
        int count = read_line(STDIN_FILENO, &input_buffer, &buf_size);
        if (count <= 0)
            break;
```

```
        int res = 0;
        char *ptr = input_buffer;
        while (*ptr) {
            int f = atoi(ptr);
            res += f;
            ptr = strchr(ptr, ' ');
            if (!ptr)
                break;
            ptr++;
        }
        int n = itoa(res, output_buffer, 1);
        output_buffer[n++] = '\n';
        write(STDOUT_FILENO, output_buffer, n);
    }
    free(input_buffer);
    return 0;
}
```

func.h

```
#ifndef __FUNC_H__
#define __FUNC_H__

#include <stddef.h>

int itoa(long n, char *res, int d);
char *strnchr(const char *buf, char c, size_t len);
int read_line(int fd, char **buf, int *buf_size);
int print(int fd, const char *s);

#endif
```

func.c

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

```
int itoa(long n, char *res, int d) {
    int neg = 0;
    if (n < 0) {
        neg++;
        n *= -1;
        res[0] = '-';
        res++;
    }
    int i = 0;
    while (n > 0) {
        res[i++] = '0' + (n % 10);
        n /= 10;
    }
    while (i < d) {
        res[i++] = '0';
    }
    for (int j = 0; j < i / 2; j++) {
        char tmp = res[j];
        res[j] = res[i - j - 1];
        res[i - j - 1] = tmp;
    }
    res[i] = 0;
    return i + neg;
}

char *strnchr(const char *buf, char c, size_t len) {
    for (size_t i = 0; i < len; i++) {
        if (buf[i] == c)
            return (char *)(buf + i);
        if (buf[i] == 0)
            return 0;
    }
    return 0;
}

int read_line(int fd, char **buf, int *buf_size) {
    int count = strnchr(*buf, 0, *buf_size) - *buf + 1;
    for (int i = 0; i < *buf_size - count; i++) {
        (*buf)[i] = (*buf)[i + count];
    }
```

```
    count = strnchr(*buf, 0, *buf_size) - *buf;
    int read_cur;
    do {
        if (count + 1 >= *buf_size) {
            int new_size = *buf_size * 2;
            char *tmp = realloc(*buf, new_size);
            if (!tmp)
                return count;
            *buf = tmp;
            *buf_size = new_size;
        }
        read_cur = read(fd, *buf + count, *buf_size - count - 1);
        count += read_cur;
        (*buf)[count] = 0;
    } while (read_cur > 0 && !strnchr(*buf, '\n', *buf_size));
    int line_len = strnchr(*buf, '\n', *buf_size) - *buf;
    if (line_len < 0)
        return count;
    (*buf)[line_len] = 0;
    return line_len + 1;
}

int print(int fd, const char *s) {
    int n = strlen(s);
    return write(fd, s, n);
}
```

Тестирование

a.txt:
1 1 1
2 3 4
-1 -2 -3

ivankamblshnikov@MacBook-Air-Rec0il-y lab1 % ./main.out
a.txt
3
9
-6

sudo dtruss -f ./main.out a.txt

Password:
PID/THRD SYSCALL(args) = return
1162/0x2612: fork() = 0 0
1162/0x2612: munmap(0x1025AC000, 0x98000) = 0 0
1162/0x2612: munmap(0x102644000, 0x8000) = 0 0
1162/0x2612: munmap(0x10264C000, 0x4000) = 0 0
1162/0x2612: munmap(0x102650000, 0x4000) = 0 0
1162/0x2612: munmap(0x102654000, 0x5C000) = 0 0
1162/0x2612: crossarch_trap(0x0, 0x0, 0x0) = -1 Err#45
1162/0x2612: open(".\0", 0x100000, 0x0) = 3 0
1162/0x2612: fcntl(0x3, 0x32, 0x16DB87298) = 0 0
1162/0x2612: close(0x3) = 0 0
1162/0x2612: fsgetpath(0x16DB872A8, 0x400, 0x16DB87288) = 51 0
1162/0x2612: fsgetpath(0x16DB872B8, 0x400, 0x16DB87298) = 14 0
1162/0x2612: csrctl(0x0, 0x16DB876BC, 0x4) = -1 Err#1
1162/0x2612: __mac_syscall(0x18555E908, 0x2, 0x16DB87600) = 0 0
1162/0x2612: csrctl(0x0, 0x16DB876DC, 0x4) = -1 Err#1
1162/0x2612: __mac_syscall(0x18555B75E, 0x5A, 0x16DB87670) = 0 0
1162/0x2612: sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16DB86BD8, 0x16DB86BD0, 0x18555D3AF, 0xD) = 0 0
1162/0x2612: sysctl([CTL_KERN, 140, 0, 0, 0, 0] (2), 0x16DB86C88, 0x16DB86C80, 0x0, 0x0) = 0 0
1162/0x2612: open("/\0", 0x20100000, 0x0) = 3 0
1162/0x2612: openat(0x3, "System/Cryptexes/OS\0", 0x100000, 0x0) = 4 0
1162/0x2612: dup(0x4, 0x0, 0x0) = 5 0
1162/0x2612: fstatat64(0x4, 0x16DB86761, 0x16DB866D0) = 0 0
1162/0x2612: openat(0x4, "System/Library/dyld/\0", 0x100000, 0x0) = 6 0

```
1162/0x2612: fcntl(0x6, 0x32, 0x16DB86760) = 0 0
1162/0x2612: dup(0x6, 0x0, 0x0) = 7 0
1162/0x2612: dup(0x5, 0x0, 0x0) = 8 0
1162/0x2612: close(0x3) = 0 0
1162/0x2612: close(0x5) = 0 0
1162/0x2612: close(0x4) = 0 0
1162/0x2612: close(0x6) = 0 0
1162/0x2612: shared_region_check_np(0x16DB86D70, 0x0, 0x0) = 0 0
1162/0x2612: fsgetpath(0x16DB872C0, 0x400, 0x16DB87218) = 82 0
1162/0x2612: fcntl(0x8, 0x32, 0x16DB872C0) = 0 0
1162/0x2612: close(0x8) = 0 0
1162/0x2612: close(0x7) = 0 0
1162/0x2612: getfsstat64(0x0, 0x0, 0x2) = 10 0
1162/0x2612: getfsstat64(0x10268AAA0, 0x54B0, 0x2) = 10 0
1162/0x2612: getattrlist("/\0", 0x16DB871F0, 0x16DB87160) = 0 0
1162/0x2612:
stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld/dyld_shared
_cache_arm64e\0", 0x16DB87550, 0x0) = 0 0
dtrace: error on enabled probe ID 1690 (ID 845: syscall::stat64:return): invalid
address (0x0) in action #12 at DIF offset 12
1162/0x2612: stat64("/Users/ivankamblshnikov/Documents/os/lab1/main.out\0",
0x16DB86A00, 0x0) = 0 0
1162/0x2612: open("/Users/ivankamblshnikov/Documents/os/lab1/main.out\0",
0x0, 0x0) = 3 0
1162/0x2612: mmap(0x0, 0x88C8, 0x1, 0x40002, 0x3, 0x0) = 0x1026CC000 0
1162/0x2612: fcntl(0x3, 0x32, 0x16DB86B18) = 0 0
1162/0x2612: close(0x3) = 0 0
1162/0x2612: munmap(0x1026CC000, 0x88C8) = 0 0
1162/0x2612: stat64("/Users/ivankamblshnikov/Documents/os/lab1/main.out\0",
0x16DB86F70, 0x0) = 0 0
1162/0x2612: stat64("/usr/lib/libSystem.B.dylib\0", 0x16DB85F00, 0x0) = -1 Err#2
1162/0x2612:
stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/libSystem.B.dylib\0",
0x16DB85EB0, 0x0) = -1 Err#2
1162/0x2612: stat64("/usr/lib/system/libdispatch.dylib\0", 0x16DB83B10, 0x0) = -1
Err#2
1162/0x2612:
stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/system/libdispatch.dylib\0
", 0x16DB83AC0, 0x0) = -1 Err#2
1162/0x2612: stat64("/usr/lib/system/libdispatch.dylib\0", 0x16DB83B10, 0x0) = -1
Err#2
1162/0x2612: open("/dev/dtracehelper\0", 0x2, 0x0) = 3 0
```

1162/0x2612: ioctl(0x3, 0x80086804, 0x16DB85B58) = 0 0
1162/0x2612: close(0x3) = 0 0
1162/0x2612: open("/Users/ivankamblshnikov/Documents/os/lab1/main.out\0", 0x0, 0x0) = 3 0
1162/0x2612: __mac_syscall(0x18555E908, 0x2, 0x16DB851D0) = 0 0
1162/0x2612: map_with_linking_np(0x16DB85060, 0x1, 0x16DB85090) = 0 0
1162/0x2612: close(0x3) = 0 0
1162/0x2612: mprotect(0x10227C000, 0x4000, 0x1) = 0 0
1162/0x2612: shared_region_check_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0) = 0 0
1162/0x2612: mprotect(0x102688000, 0x40000, 0x1) = 0 0
1314/0x30f7: mprotect(0x100F10000, 0x40000, 0x1) = 0 0
1314/0x30f7: mprotect(0x100F10000, 0x40000, 0x3) = 0 0
1314/0x30f7: mprotect(0x100F10000, 0x40000, 0x1) = 0 0
1314/0x30f7: getattrlist("/Users/ivankamblshnikov/Documents/os/lab1/main.out\0", 0x16F2FDF70, 0x16F2FDF88) = 0 0
1314/0x30f7: access("/Users/ivankamblshnikov/Documents/os/lab1\0", 0x4, 0x0) = 0 0
1314/0x30f7: open("/Users/ivankamblshnikov/Documents/os/lab1\0", 0x0, 0x0) = 3 0
1314/0x30f7: fstat64(0x3, 0x13E6044C0, 0x0) = 0 0
1314/0x30f7: csrctl(0x0, 0x16F2FE19C, 0x4) = 0 0
1314/0x30f7: fcntl(0x3, 0x32, 0x16F2FDE58) = 0 0
1314/0x30f7: close(0x3) = 0 0
1314/0x30f7: open("/Users/ivankamblshnikov/Documents/os/lab1/Info.plist\0", 0x0, 0x0) = -1 Err#2
1314/0x30f7: proc_info(0x2, 0x522, 0xD) = 64 0
1314/0x30f7: csops_audittoken(0x522, 0x10, 0x16F2FE1E0) = 0 0
1314/0x30f7: sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16F2FE538, 0x16F2FE530, 0x188BFED3D, 0x15) = 0 0
1314/0x30f7: sysctl([CTL_KERN, 138, 0, 0, 0, 0] (2), 0x16F2FE5C8, 0x16F2FE5C0, 0x0, 0x0) = 0 0
1314/0x30f7: csops(0x522, 0x0, 0x16F2FE66C) = 0 0
1314/0x30f7: mprotect(0x100F10000, 0x40000, 0x3) = 0 0
1314/0x30f7: pipe(0x0, 0x0, 0x0) = 3 0

ERROR: failed to open file: ""
No such file or directory
1314/0x30f7: read(0x0, "\n\0", 0x3F) = 1 0
1314/0x30f7: open("\0", 0x0, 0x0) = -1 Err#2
1314/0x30f7: write(0x2, "ERROR: failed to open file: \"\0", 0x1D) = 29 0
1314/0x30f7: write(0x2, "\0", 0x0) = 0 0

```
1314/0x30f7: write(0x2, "\"\n\0", 0x2) = 2 0
1314/0x30f7: write(0x2, "No such file or directory\0", 0x19) = 25 0
1314/0x30f7: write(0x2, "\n\0", 0x1) = 1 0

ivankamblshnikov@MacBook-Air-Rec0il-y lab1 % clear

ivankamblshnikov@MacBook-Air-Rec0il-y lab1 % sudo dtruss -f ./main.out a.txt
PID/THRD SYSCALL(args) = return
1328/0x3247: fork() = 0 0
1328/0x3247: munmap(0x10111C000, 0x98000) = 0 0
1328/0x3247: munmap(0x1011B4000, 0x8000) = 0 0
1328/0x3247: munmap(0x1011BC000, 0x4000) = 0 0
1328/0x3247: munmap(0x1011C0000, 0x4000) = 0 0
1328/0x3247: munmap(0x1011C4000, 0x5C000) = 0 0
1328/0x3247: crossarch_trap(0x0, 0x0, 0x0) = -1 Err#45
1328/0x3247: open(".\0", 0x100000, 0x0) = 3 0
1328/0x3247: fcntl(0x3, 0x32, 0x16EF2F298) = 0 0
1328/0x3247: close(0x3) = 0 0
1328/0x3247: fsgetpath(0x16EF2F2A8, 0x400, 0x16EF2F288) = 51 0
1328/0x3247: fsgetpath(0x16EF2F2B8, 0x400, 0x16EF2F298) = 14 0
1328/0x3247: csrctl(0x0, 0x16EF2F6BC, 0x4) = -1 Err#1
1328/0x3247: __mac_syscall(0x18555E908, 0x2, 0x16EF2F600) = 0 0
1328/0x3247: csrctl(0x0, 0x16EF2F6DC, 0x4) = -1 Err#1
1328/0x3247: __mac_syscall(0x18555B75E, 0x5A, 0x16EF2F670) = 0 0
1328/0x3247: sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16EF2EBD8, 0x16EF2EBD0,
0x18555D3AF, 0xD) = 0 0
1328/0x3247: sysctl([CTL_KERN, 140, 0, 0, 0, 0] (2), 0x16EF2EC88, 0x16EF2EC80,
0x0, 0x0) = 0 0
1328/0x3247: open("/\0", 0x20100000, 0x0) = 3 0
1328/0x3247: openat(0x3, "System/Cryptexes/OS\0", 0x100000, 0x0) = 4 0
1328/0x3247: dup(0x4, 0x0, 0x0) = 5 0
1328/0x3247: fstatat64(0x4, 0x16EF2E761, 0x16EF2E6D0) = 0 0
1328/0x3247: openat(0x4, "System/Library/dyld/\0", 0x100000, 0x0) = 6 0
1328/0x3247: fcntl(0x6, 0x32, 0x16EF2E760) = 0 0
1328/0x3247: dup(0x6, 0x0, 0x0) = 7 0
1328/0x3247: dup(0x5, 0x0, 0x0) = 8 0
1328/0x3247: close(0x3) = 0 0
1328/0x3247: close(0x5) = 0 0
1328/0x3247: close(0x4) = 0 0
1328/0x3247: close(0x6) = 0 0
1328/0x3247: shared_region_check_np(0x16EF2ED70, 0x0, 0x0) = 0 0
1328/0x3247: fsgetpath(0x16EF2F2C0, 0x400, 0x16EF2F218) = 82 0
```

```
1328/0x3247: fcntl(0x8, 0x32, 0x16EF2F2C0) = 0 0
1328/0x3247: close(0x8) = 0 0
1328/0x3247: close(0x7) = 0 0
1328/0x3247: getfsstat64(0x0, 0x0, 0x2) = 10 0
1328/0x3247: getfsstat64(0x1012E2AA0, 0x54B0, 0x2) = 10 0
1328/0x3247: getattrlist("/\0", 0x16EF2F1F0, 0x16EF2F160) = 0 0
1328/0x3247:
stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld/dyld_shared
_cache_arm64e\0", 0x16EF2F550, 0x0) = 0 0
dtrace: error on
```

Вывод

В ходе выполнения лабораторной работы была составлена и отлажена программа на языке C, осуществляющая вызов дочернего процесса, переопределение стандартных потоков ввода-вывода и взаимодействие между процессами в ОС macOS.

Также была написана программа, осуществляющая ввод, обработку и вывод информации без использования stdio.h.