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: Жаворонков Н. Д.

Преподаватель: Бахарев В. Д.

Оценка: _____

Дата: 16.10.24

Москва, 2024

Постановка задачи

Вариант 10.

В файле записаны команды вида: «число». Дочерний процесс производит проверку этого числа на простоту. Если число составное, то дочерний процесс пишет это число в стандартный поток вывода. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

1. **Чтение файла:** Родительский процесс считывает числа из файла построчно.
2. **Создание дочернего процесса:** Для каждого прочитанного числа родительский процесс создает дочерний процесс с помощью системного вызова fork().
3. **Проверка на простоту:** Дочерний процесс проверяет число на простоту. Алгоритм проверки:
   - Если число меньше 2, то оно не является простым.
   - Если число равно 2, то оно является простым.
   - Если число больше 2, то необходимо проверить, делится ли оно на числа от 2 до квадратного корня из числа. Если делится, то число составное.
4. **Вывод результата:**
   - Если число составное, то дочерний процесс выводит его в стандартный поток вывода.
   - Если число отрицательное или простое, то дочерний процесс завершается.
5. **Завершение родительского процесса:** Родительский процесс ожидает завершения всех дочерних процессов с помощью системного вызова wait(). После завершения всех дочерних процессов родительский процесс также завершается.

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int pipe(int *fd); – создает именованный канал для межпроцессного взаимодействия.
- wait(int *status); – ожидает завершения дочернего процесса.
- read(int fd, void *buf, size_t count); – считывает данные из файла или канала.
- write(int fd, const void *buf, size_t count); – записывает данные в файл или канал.

Программа состоит из двух частей: родительской и дочерней. Родительский процесс открывает файл, читает из него числа и пересылает их через специальный канал на вход дочернему процессу. По завершении пересылки канал закрывается. Дочерний процесс считывает данные из потока ввода и определяет, составное ли поданное на вход число. Если число не составное – программа завершается. Сверка чисел происходит пока не будут прочитаны все числа на входе.

Код программы

server.c

```
// -=-=-=-=-=-=-=-=-=-= Supertos Industries ( 2012 - 2024 ) =-=-=-=-=-=-=-=-=-=-
// Author; Supertos
//
// Exercise 1
// -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
#define _GNU_SOURCE

#include <stdbool.h>

#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
```

```
#include <sys/stat.h>
#include <fcntl.h>

#include "commandline.h"
#include "utils.h"

/* Custom defines start here */
#define CLIENT_NAME "/client.out"

struct ProgramState { // Empty structures unallowed, see C99 spec.
	int dummy;
};

/* Returns this executable directory */
char* getexecpath() {
	char* path = malloc( MAX_PATH_LEN * 2 );
	size_t len = readlink( "/proc/self/exe", path, MAX_PATH_LEN );

	if( !len ) {
		free( path );
		return NULL;
	}

	while( path[len] != '/' ) len--;
	path[len] = '\0';

	return path;
}

/* Reads path from stdin */
char* readpath( int argc, char** argv ) {
	CMD* command = initCMD( argc, argv );
	if( !command ) return NULL;

	char* path = expectCMD( command, /* CMDType */ String );
	freeCMD( command );

	return path;
}

/* Setups pipe; overrides [override] with pipe [end] and closes the other [end] */
void setpipe( int* pipe, size_t end, size_t override ) {
	dup2( pipe[end], override );
	close( pipe[!end] ); // Since pipe is always bi-directional
}

/* Reads single character from [file] to [trg] */
static inline int readchar( int file, char* trg ) {
	return read( file, trg, 1 );
}

/* Returns true if [buf] is empty */
static inline bool strempty( char* buf ) {
	return buf[0] == '\0';
}

/* Reads unsigned int from [file] to [buf] of size [n] */
int readuint( const int file, char* buf, size_t n ) {
	char c;
	bool foundint = false;
	while( readchar( file, &c ) ) {
		if( IS_BLANK(c) )
			if( foundint ) break; else continue;

		if( !IS_DIGIT(c) ) return ERR_INVALID_INPUT;
		foundint = true;
```

```
                *(buf++) = c;
                n--;
                if( !n ) return ERR_INVALID_INPUT;
        }
        *(buf++) = '\0';

        return NO_ERR;
}

/* Entry point */
int main( int argc, char** argv ) {
        char* path;
        if( !(path=getexecpath()) ) return error( ERR_PATH_READ );

        char* target;
        if( !(target=readpath( argc, argv )) ) return error( ERR_INVALID_INPUT );

        int childPipe[2];
        if( pipe2( childPipe, O_DIRECT ) ) return error( ERR_CANT_INIT_PIPE );

        switch( fork() ) {
                case -1: // An error occured
                        return error( ERR_CANT_INIT_CHILD );
                case 0: // We're child
                        setpipe( childPipe, 0, stdin );

                        char *const args[] = { CLIENT_NAME, NULL };
                        strcpy( &path[strlen(path)], CLIENT_NAME );

                        return execv( path, args );
                default: // We're parent
                        close( childPipe[0] ); // It is imperative we close the other end AND preserve out
stdin/stdout

                        int file;
                        if( (file=open( target, 0, "r" )) == -1 ) return error( ERR_NO_SUCH_FILE );

                        char buf[MAX_NUM_LEN];
                        while( !readuint( file, buf, MAX_NUM_LEN ) && !strempty(buf) )
                                write( childPipe[1], buf, strlen(buf) );

                        close( childPipe[1] );
                        close( file );

                        int childStatus;
                        wait(&childStatus);
                        if( !WIFEXITED( childStatus ) ) return error( childStatus );

                        break;
        }
} // return 0 implicit, see C99 spec.

/* SISP */
```

client.c

```
// -=-=-=-=-=-=-=-=-=-= Supertos Industries ( 2012 - 2024 ) =-=-=-=-=-=-=-=-=-=-
// Author: Supertos
//
// Exercise 1
// -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
#include <math.h>
#include <stdbool.h>

#include <unistd.h>
#include <sys/wait.h>
#include <fcntl.h>
```

```c
#include "commandline.h"
#include "utils.h"
#include "number.h"

/* Custom defines start here */
#define stdin 0
#define stdout 1

struct ProgramState { // Empty structures unallowed, see C99 spec.
	int dummy;
};

/* Helper function: checks if a is *a* prime */
int isPrime( double a ) {
	for( size_t i = 2; i <= sqrt(a); i++ )
		if( (size_t)a % i == 0 ) return 0;

	return 1;
}

/* Entry point */
int main( int argc, char** argv ) {
	char buffer[MAX_NUM_LEN + 1];
	while( read(stdin, buffer, MAX_NUM_LEN) ) {
		if( cstrtod( buffer ) < 0 || isPrime(cstrtod( buffer )) ) return NO_ERR;

		write( stdout, buffer, strlen(buffer) );
		write( stdout, "\n", 1 );
	}

} // return 0 implicit, see C99 spec.

/* SISP */
```

utils.c

```c
// -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
// Supertos Industries ( 2012 - 2024 )
// Author: Supertos
// Command line utilities
// -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
#include "utils.h"

static char* ITOA_ALPHABET = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";

char* revstr( char* s, size_t n ) {
	char temp;
	for( size_t i = 0; i < n / 2; i++ ) {
		temp = s[i];
		s[i] = s[n-i-1];
		s[n-i-1] = temp;
	}
	return s;
}

size_t strlen( const char* s ) {
	size_t out = 0;
	while( *(s++) != '\0' ) out++;

	return out;
}

void itoa( int n, char s[], uint8_t base ) {
	uint8_t i = 0;
	int sign = n < 0 ? -1 : 1;

    if( sign < 0 ) n = -n;
```

```
        do {
            s[i++] = ITOA_ALPHABET[n % base];
        }while( (n /= base) > 0 );

        if( sign < 0 ) s[i++] = '-';

        s[i] = '\0';
        revstr(s, i);
    }

    int print( const char* text ) {
        return (int)write( stdout, text, strlen(text) );
    }

    int printn( const char* text, const size_t n ) {
        return (int)write( stdout, text, n );
    }

    int readn( char* text, const size_t n ) {
        return (int)read( stdin, text, n );
    }

    int error( const int code ) {
        char* buf = malloc( ITOA_BUF_SIZE );
        itoa( code, buf, 10 );

        print( "An error has occured: " );
        print( buf );
        print( "\n" );

        free( buf );
        return code;
    }
```

number.c

```
// -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
// Supertos Industries ( 2012 - 2024 )
// Author: Supertos
// Number utilities
// -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
#include <math.h>
#include <string.h>

double cstrtod( const char* data ) {
    double whole = 0;
    double frac = 0;

    double sign = 1;
    if( data[0] == '-' ) {
        sign = -1;
        data++;
    }

    int exponent = 0; // 0 - before dot, >0 - after

    for( register char n = *data; n; n = *(++data) ) {
        if( n == '\0' ) break;

        if( n == '.' ) {
            if( exponent ) return 0;
            exponent = 1;
        }else if( n < '0' || n > '9' ) {
            return 0;
        }else if( exponent ) {
            frac += (n - '0') / pow(10, exponent++);
        }else{
```

```
                        whole = whole*10 + (n - '0');
                }
        }
        return (whole + frac) * sign;
}

long double fact_simple( long double a ) {
        if( a == 0 ) return 1;

        for( long double n = a - 1; n > 0; n-- )
                a *= n;
        return a;
}

long double fact_simple_eveness( long double a ) {
        if( a == 0 ) return 1;

        long double init = a;
        for( long double n = a - 1; n > 0; n-- )
                if( (long int)init % 2 == (long int)a % 2 ) a *= n;
        return a;
}

long double fact_simple_trimmed( long double a, long double limit ) {
        if( a == 0 ) return 1;

        for( long double n = a - 1; n > limit; n-- )
                a *= n;
        return a;
}
```

commandline.c

```
// -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
// Supertos Industries ( 2012 - 2024 )
// Author: Supertos
// Command line utilities
// -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

#include "commandline.h"
#include <stdio.h>

void freeCMD( CMD* command ) {
        free( command );
}

CMD* initCMD( int argc, char** argv ) {
        CMD* command = malloc( sizeof(CMD) );
        *command = (CMD){ .argc = argc, .argv = argv, .argp = 1 };

        return command;
}

int isCMDEmpty( CMD* command ) {
        return command->argc <= command->argp;
}

char* readCMD( CMD* command ) {
        if( isCMDEmpty( command ) ) return NULL;

        return command->argv[command->argp++];
}


char* expectCMD( CMD* command, CMDType expect ) {
        char* token = readCMD( command );
        if( !token ) return NULL;
```

```
	switch( expect ) {
		case Double:
		case Int:
			int flagDot = 0;
			for( size_t i = strlen( token ) - 1; i > 0; i-- ) { // CMP 0 is faster
				if( ( token[i] < '0' || token[i] > '9' ) && token[i] != '-' && ( expect != Double ||
flagDot || token[i] != '.' ) ) return NULL;
				if( token[i] == '.' ) flagDot = 1;
			}
			break;
		case Flag:
			if( token[0] != '-' || strlen(token) < 2 ) return NULL;
			break;
		case String:
			break;
		default:
			return NULL;
	}
	return token;
}

int executeFlagHandler( CMD* command, const char* flag, FlagHandle handles[], size_t handlec, ProgramState* state ) {
	for( size_t i = 0; i < handlec; i++ ) {
		if( !strcmp( flag, handles[i].flag ) ) {
			return handles[i].handle(command, state);
		}
	}
	return 32; // No such flag err
}


int processCMD( CMD* command, FlagHandle handles[], size_t handlec, ProgramState* state ) {
	while( !isCMDEmpty( command ) ) {
		char* flag = expectCMD( command, Flag );
		if( !flag ) return 1;
		int code = executeFlagHandler( command, flag, handles, handlec, state );
		if( code ) return code;
	}
	return 0;
}
```

Протокол работы программы

Тестирование:

```
supertos@DESKTOP-77RBNCB:~/oslab$ ./server.out tests/a.txt
18
36
96
48
supertos@DESKTOP-77RBNCB:~/oslab$
```

Strace

```
execve("./server.out", ["./server.out", "tests/a.txt"], 0x7fffa30a9898 /* 23 vars */) = 0
brk(NULL)                               = 0x56118ac5c000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc64abe940) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f5454f98000
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=66755, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 66755, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f5454f87000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.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\0\0\0\0\0\0\0"..., 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=940560, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 942344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f5454ea0000
mmap(0x7f5454eae000, 507904, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0xe000) = 0x7f5454eae000
mmap(0x7f5454f2a000, 372736, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x8a000) =
0x7f5454f2a000
mmap(0x7f5454f85000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0xe4000) = 0x7f5454f85000
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\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\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\0\0"..., 784, 64) = 784
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f5454c77000
mprotect(0x7f5454c9f000, 2023424, PROT_NONE) = 0
mmap(0x7f5454c9f000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f5454c9f000
mmap(0x7f5454e34000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000)
= 0x7f5454e34000
mmap(0x7f5454e8d000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x215000) = 0x7f5454e8d000
mmap(0x7f5454e93000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f5454e93000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f5454c74000
arch_prctl(ARCH_SET_FS, 0x7f5454c74740) = 0
set_tid_address(0x7f5454c74a10)         = 752
```

set_robust_list(0x7f5454c74a20, 24) = 0
rseq(0x7f5454c750e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f5454e8d000, 16384, PROT_READ) = 0
mprotect(0x7f5454f85000, 4096, PROT_READ) = 0
mprotect(0x56118a8b1000, 4096, PROT_READ) = 0
mprotect(0x7f5454fd2000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f5454f87000, 66755) = 0
getrandom("\x55\xe6\x05\x60\xbe\xb3\xd3\xd0", 8, GRND_NONBLOCK) = 8
brk(NULL) = 0x56118ac5c000
brk(0x56118ac7d000) = 0x56118ac7d000
readlink("/proc/self/exe", "/home/supertos/oslab/server.out", 512) = 31 // server.c 29:18
pipe2([3, 4], O_DIRECT) = 0 // server.c 98:9
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f5454c74a10) = 753
close(3) = 0 // server.c 111:13
openat(AT_FDCWD, "tests/a.txt", O_RDONLY) = 3 // server.c 114:23
read(3, "1", 1) = 1 // server.c 61:12
read(3, "8", 1) = 1 // server.c 61:12
read(3, "\r", 1) = 1 // server.c 61:12
write(4, "18", 218) = 2 // server.c 118:17
read(3, "\n", 1) = 1 // server.c 61:12
read(3, "3", 1) = 1 // server.c 61:12
read(3, "6", 1) = 1 // server.c 61:12
read(3, "\r", 1) = 1 // server.c 61:12
write(4, "36", 236) = 2 // server.c 118:17
read(3, "\n", 1) = 1 // server.c 61:12
read(3, "9", 1) = 1 // server.c 61:12
read(3, "6", 1) = 1 // server.c 61:12
read(3, "\r", 1) = 1 // server.c 61:12
write(4, "96", 296) = 2 // server.c 118:17
read(3, "\n", 1) = 1 // server.c 61:12
read(3, "4", 1) = 1 // server.c 61:12
read(3, "8", 1) = 1 // server.c 61:12
read(3, "\r", 1) = 1 // server.c 61:12
write(4, "48", 248) = 2 // server.c 118:17
read(3, "\n", 1) = 1 // server.c 61:12
read(3, "3", 1) = 1 // server.c 61:12
read(3, "0", 1) = 1 // server.c 61:12
read(3, "\r", 1) = 1 // server.c 61:12
write(4, "30", 230) = 2 // server.c 118:17
read(3, "\n", 1) = 1 // server.c 61:12
read(3, "", 1) = 0 // server.c 61:12
close(4) = 0 // server.c 120:13
close(3) = 0 // server.c 121:13
wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 753 // server.c 124:13
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=753, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
exit_group(0)

Вывод

В ходе выполнения лабораторной работы была реализована программа, которая считывает числа из файла, создает дочерние процессы для проверки каждого числа на простоту и выводит составные числа в стандартный поток вывода. Программа успешно выполняет поставленную задачу и демонстрирует применение межпроцессного взаимодействия с использованием системного вызова fork().

В процессе реализации возникла трудность с корректным завершением родительского процесса после завершения всех дочерних процессов. Необходимо было использовать системный вызов wait(), чтобы родительский процесс ожидал завершения каждого дочернего процесса, прежде чем завершиться сам.

В целом, лабораторная работа была интересной и позволила закрепить знания о работе с системными вызовами и межпроцессном взаимодействии. Желательно было бы добавить возможность работы с большим количеством чисел и более эффективных алгоритмов проверки на простоту.