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:Заворотный А.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 28.10.24

Москва, 2024

Постановка задачи

Вариант 5.

Пользователь вводит команды вида: «число<endline>». Далее это число передается от родительского процесса в дочерний. Дочерний процесс производит проверку на простоту. Если число составное, то в это число записывается в файл. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются.

Общий метод и алгоритм решения

Использованные системные вызовы:

- int channel[2]; pipe(channel); – создает канал связи.
- const pid_t child = fork(); – создает дочерний процесс.
- pid_t pid = getpid(); – получает id 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. Считываю строку
3. С помощью функций написанных выше связываю родительский процесс с дочерним.
4. В дочернем процессе получаю строку, переданную от родительского процесса проверяю её на валидность, затем проверяю число на простоту, если число простое, то записываю полученную строку в файл, иначе завершаю выполнение процессов.

Код программы

Server.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[] = "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';
    }
    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;
    }
}
```

Client.c

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

int isPrime(int N) {
    if (N == 1) {
        return 0;
    }
    if (N == 2) {
        return 1;
    }
    if (N % 2 == 0){
        return 0;
    }
    for (int i = 3; i * i <= N; i++) {
        if (N % i == 0) {
            return 0;
        }
        i++;
    }
    return 1;
}

int Validatenum(const char* argv) {
    int cnt = 0;
    while (*argv != '\0' && *argv != '\n') {
        if (!isdigit(*argv)) {
            return 0;
        }
        cnt++;
        argv++;
    }
    if (cnt > 8){
        return -1;
    }
    return 1;
}

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 not prime numbers. Type prime number or 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;
        }
```

```
        {
            buf[bytes - 1] = '\0';
            switch(Validatenum(buf)){
                case 0:
                    const char msg[] = "error: one of arguments is not a number\n";
                    write(STDERR_FILENO, msg, sizeof(msg));
                    exit(EXIT_FAILURE);
                case -1:
                    const char msg1[] = "error: too many symbols\n";
                    write(STDERR_FILENO, msg1, sizeof(msg1));
                    exit(EXIT_FAILURE);
                default: break;
            }
            int num = atoi(buf);
            if (isPrime(num) == 1) {
                const char msg[] = "error: one of numbers is prime\n";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            }
            buf[bytes - 1] = '\n';
            int32_t written = write(file, buf, bytes);
            if (written != bytes) {
                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

```
$ ./server file1.txt
8451: I'm a parent, my child has PID 8452
8452: I'm a child
8452: Start typing not prime numbers. Type prime number or press 'Ctrl-D'
or 'Enter' with no input to exit
32453
353
error: one of numbers is prime
error: child exited with error
$ cat file1.txt
32453

Strace:
$ strace -f ./server
execve("./server", ["./server"], 0x7fffc491d228 /* 26 vars */) = 0
brk(NULL)                               = 0x55e8f59b9000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f07a4eb3000
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=20071, ...}) = 0
mmap(NULL, 20071, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f07a4eae000
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\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\0\0"..., 784, 64)
= 784
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f07a4c9c000
mmap(0x7f07a4cc4000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7f07a4cc4000
mmap(0x7f07a4e4c000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7f07a4e4c000
mmap(0x7f07a4e9b000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1fe000) = 0x7f07a4e9b000
mmap(0x7f07a4ea1000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -
1, 0) = 0x7f07a4ea1000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f07a4c99000
arch_prctl(ARCH_SET_FS, 0x7f07a4c99740) = 0
set_tid_address(0x7f07a4c99a10)         = 9493
set_robust_list(0x7f07a4c99a20, 24)     = 0
rseq(0x7f07a4c9a060, 0x20, 0, 0x53053053) = 0
mprotect(0x7f07a4e9b000, 16384, PROT_READ) = 0
mprotect(0x55e8f3c98000, 4096, PROT_READ) = 0
mprotect(0x7f07a4eeb000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f07a4eae000, 20071)           = 0
write(2, "usage: ./server filename\n", 25usage: ./server 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 записи строк в разные файлы. Проблем при выполнении работы не возникло.