

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-210БВ-24

Студент: Кудряшова Т.И.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 02.10.25

Москва, 2025

Постановка задачи

Вариант 10.

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 файл из shared memory, выходные данные возвращаются через него же.

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

Использованные системные вызовы:

- `shm_open(SHM_NAME, O_RDWR | O_CREAT | O_TRUNC, 0600)` – создание/перезапись shared memory с названием SHM_NAME на чтение и запись для владельца
- `ftruncate(shm, SHM_SIZE)` – установка размера сегмента разделяемой памяти на 4096 байт в физ памяти
- `mmap(NULL, SHM_SIZE, PROT_WRITE, MAP_SHARED, shm, 0)` – отображает shared memory в виртуальное пространство
-
- `pid_t fork(void);` – создает дочерний процесс.
- `readlink("/proc/self/exe", proppath, sizeof(proppath) - 1)` – получение пути к текущей директории.
- `write(STD..._FILENO, msg, sizeof(msg))` – запись в поток STD..._FILENO.
- `bytes = read(STD..._FILENO, buf, sizeof(buf) - 1);` - чтение из потока STD..._FILENO.
- `STDERR_FILENO` – поток для ошибок.
- `STDIN_FILENO` – поток IN.
- `STDOUT_FILENO` - поток OUT.

В рамках решения задачи алгоритм программы выглядит так:

- 1) Получение пути к текущей директории.
- 2) Открытие каналов между родителем и ребенком, а также ребенком и родителем.
- 3) Ответвление ребенка.
- 4) Чтение названия файла из stdin.
- 5) Открытие файла и заброс его содержимого на поток ввода процесса child.
Далее действия в child.
- 6) Расшифровка полученных данных.
- 7) Вывод составных чисел на поток вывода.
- 8) При получении простого или отрицательного - окончание программы.

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>
#include <string.h>
#include <fcntl.h>

static char SERVER_PROGRAM_NAME[] = "child";

int main(int argc, char **argv) {

    // NOTE: Get full path to the directory, where program resides
    char progpath[1024];
    {
        // NOTE: Read full program path, including its name
        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);
        }

        // NOTE: Trim the path to first slash from the end
        while (progpath[len] != '/')
            --len;

        progpath[len] = '\0';
    }

    // NOTE: Open pipes
    int client_to_server[2]; // AB
    if (pipe(client_to_server) == -1) {
        const char msg[] = "error: failed to create pipe\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    int server_to_client[2]; // BA
    if (pipe(server_to_client) == -1) {
        const char msg[] = "error: failed to create pipe\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    // NOTE: Spawn a new process
    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: { // NOTE: We're a child, child doesn't know its pid after fork

        close(client_to_server[1]);
        close(server_to_client[0]);

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

```

```

dup2(server_to_client[1], STDOUT_FILENO);
close(server_to_client[1]);

{
    char path[1024];
    snprintf(path, sizeof(path) - 1, "%s/%s", proppath, SERVER_PROGRAM_NAME);

    // NOTE: Start server without extra args; will read filename from stdin
    char *const args[] = {SERVER_PROGRAM_NAME, 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: { // NOTE: We're a parent, parent knows PID of child after fork

    close(client_to_server[0]);
    close(server_to_client[1]);

    char buf[4096];
    ssize_t bytes;

    // 1) Read filename from our stdin
    bytes = read(STDIN_FILENO, buf, sizeof(buf) - 1);
    if (bytes <= 0) {
        const char msg[] = "error: failed to read filename from stdin\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }
    buf[bytes] = '\0';
    char *nl = strchr(buf, '\n');
    if (nl) *nl = '\0';

    // 2) Open file and stream its contents into server stdin
    int fd = open(buf, O_RDONLY);
    if (fd == -1) {
        const char msg[] = "error: failed to open input file\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    while ((bytes = read(fd, buf, sizeof(buf))) > 0) {
        if (write(client_to_server[1], buf, bytes) != bytes) {
            const char msg[] = "error: failed to write to server stdin\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);
            close(fd);
            exit(EXIT_FAILURE);
        }
    }
    close(fd);
    close(client_to_server[1]); // signal EOF to server

    // 3) Read server response until EOF and print to stdout
    while ((bytes = read(server_to_client[0], buf, sizeof(buf))) > 0) {
        write(STDOUT_FILENO, buf, bytes);
    }

    close(server_to_client[0]);

    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>  
#include <string.h>  
  
int is_composit(long long x) {  
    if (x < 2) return 0;  
    for (long long i = 2; i * i <= x; ++i) {  
        if (x % i == 0) return 1;  
    }  
    return 0;  
}  
  
int main(int argc, char **argv) {  
    char buf[4096];  
    ssize_t bytes;  
  
    // Read numbers line by line from stdin  
    char line[256];  
    uint32_t linelen = 0;  
    while ((bytes = read(STDIN_FILENO, buf, sizeof(buf))) > 0) {  
        for (uint32_t i = 0; i < (uint32_t)bytes; ++i) {  
            char c = buf[i];  
            if (c == '\n') {  
                line[linelen] = '\0';  
                // Parse integer  
                char *endptr = NULL;  
                long long value = strtoll(line, &endptr, 10);  
                if (endptr != line) {  
                    if (!is_composit(value)) {  
                        exit(EXIT_SUCCESS);  
                    }  
                    char out[64];  
                    int32_t outlen = snprintf(out, sizeof(out), "%lld\n", value);  
                    if (write(STDOUT_FILENO, out, outlen) != outlen) {  
                        const char msg[] = "error: failed to echo\n";  
                        write(STDERR_FILENO, msg, sizeof(msg) - 1);  
                        exit(EXIT_FAILURE);  
                    }  
                }  
                linelen = 0;  
            } else {  
                if (linelen + 1 < sizeof(line)) {  
                    line[linelen++] = c;  
                } else {  
                    // Line too long, reset to avoid overflow  
                    linelen = 0;  
                }  
            }  
        }  
    }  
  
    // If the last line doesn't end with newline, process it as well  
    if (linelen > 0) {  
        line[linelen] = '\0';
```

```
char *endptr = NULL;
long long value = strtoll(line, &endptr, 10);
if (endptr != line) {
    if (!is_composit(value)) {
        exit(EXIT_SUCCESS);
    }
    char out[64];
    int32_t outlen = snprintf(out, sizeof(out), "%lld\n", value);
    write(STDOUT_FILENO, out, outlen);
}

if (bytes < 0) {
    const char msg[] = "error: failed to read from stdin\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}
```

Протокол работы программы

Тест 1:

Содержание 1.txt:

000010

8

000005

10

В консоли:

./parent

1.txt

Вывод:

10

8

Тест 2:

Содержание 2.txt:

4

25

-5

В консоли:

./parent

2.txt

Вывод:

4

25

Вывод

Интересно, что фактически одно и то же можно сделать несколькими способами и у каждого есть свои плюсы и минусы. Хотя минимальные изменения после первой лабы были приятны 😊.