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-214БВ-24

Студент: Чернявская Алиса Алексеевна

Преподаватель: Бахарев В.Д. (ФИИТ)

Оценка: _____

Дата: 18.09.25

Москва, 2025

# Постановка задачи

**Вариант 5.**

Цель работы

- Приобретение практических навыков в:
- Управление процессами в ОС
- Обеспечение обмена данных между процессами посредством каналов

Задание

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe).

Родительский процесс создает дочерний процесс. 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.

Пользователь вводит команды вида: «число». 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:

- CreatePipe(&to_child_read, &to_child_write, &sa, 0) – создание pipe с режимом наследования для дочернего процесса
- CreateProcess("server.exe", NULL, NULL, NULL, TRUE, 0, NULL, NULL, &start_info, &proc_info) – создание нового процесса (запуск сервера) с разрешением наследования handles.
- CloseHandle() – закрытие handle, чтобы освободить системные ресурсы и чтобы не возникало утечек памяти
- ReadFile(GetStdHandle(STD_INPUT_HANDLE), buffer, BUF_SIZE, &bytes_read, NULL); - чтение данных через handles
- WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), response, response_len, &bytes_written, NULL); - запись данных через handles
- GetStdHandle() – получение handles стандартного ввода/вывода
- CreateFile() – создание/открытие файла по умолчанию

В моей программе находятся 2 файла, обозначающие работу сервера и клиента. Файл-клиент является родительским процессом, он запускает файл-сервер, как дочерний процесс. В

программе передача информации между файлами осуществляется через pipes – двусторонние каналы. Первый pipe – клиент пишет, сервер читает, второй pipe – сервер пишет, клиент читает. Сервер имеет доступ к pipes благодаря наследованию handles родительского процесса. Данные идут от клиента к серверу и обратно через буфер ядра, при этом считывание невозможно, пока не будут введены данные, а запись невозможна, пока буфер не будет очищен.

Пример работы программы: в клиенте пользователь вводит число, которое в клиенте же проверяется, точно ли оно является именно числом. Затем, если проверка успешна, оно попадает в буфер ядра и через pipe идет в сервер. Сервер считывает это число и тоже делает проверки. Если число = простое, 0, 1 или отрицательное, то сервер отправляет клиенту ответ «shutdown», тот принимает его и заканчивает работу. Если же число является составным, то сервер записывает его в файл, отправляет это же число обратно клиенту, клиент получает это число, выводит его и продолжает работу.

## Код программы

**client.c:**

```c
#include <windows.h>
#include <string.h>

#define BUF_SIZE 1024

// Функция проверки, что строка – целое число
int is_number(const char* str)
{
    if (*str == '-' || *str == '+')
        str++;
    if (*str == '\0')
        return 0;

    while (*str)
    {
        if (*str < '0' || *str > '9')
            return 0;
        str++;
    }
    return 1;
}

int main()
{

    SECURITY_ATTRIBUTES sa; //переменная типа ... (кто может наследовать и получить доступ к pipes)
    sa.nLength = sizeof(sa);
    sa.bInheritHandle = TRUE;  // разрешить наследование pipes
    sa.lpSecurityDescriptor = NULL;//использовать настройки безопасности по умолчанию

    // Создаем каналы pipes
    HANDLE to_child_read, to_child_write;   // родитель – ребенок
    HANDLE from_child_read, from_child_write; // ребенок – родитель

    // первый pipe (родитель пишет, ребенок читает)
    if (!CreatePipe(&to_child_read, &to_child_write, &sa, 0))
    {
        const char error[] = "Error: cannot create pipe to child\n";
        //получаем дескриптор об ошибке
        WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
        return 1;
    }
    // Создаем второй канал (ребенок пишет, родитель читает)
    if (!CreatePipe(&from_child_read, &from_child_write, &sa, 0))
```

```
    {
        const char error[] = "Error: cannot create pipe from child\n";
        WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
        return 1;
    }

    // Настраиваем информацию для запуска
    STARTUPINFO start_info;//как запустить процесс
    PROCESS_INFORMATION proc_info;//информация о созданном процессе

    ZeroMemory(&start_info, sizeof(start_info));//обнуление структуры создания процесса
    start_info.cb = sizeof(start_info);//размер структуры создания процесса
    start_info.dwFlags = STARTF_USESTDHANDLES;//использование handles для потоков ввода/вывода
    //то, что написано в строке сверху:
    start_info.hStdInput = to_child_read;// ребенок читает отсюда
    start_info.hStdOutput = from_child_write;// ребенок пишет сюда
    start_info.hStdError = GetStdHandle(STD_ERROR_HANDLE);

    ZeroMemory(&proc_info, sizeof(proc_info));//обнуление структуры информации о процессе
    // запуск сервера
    if (!CreateProcess("server.exe", NULL, NULL, NULL, TRUE, 0, NULL, NULL, &start_info,
&proc_info))
    {
        const char error[] = "Error: cannot start server.exe\n";
        WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
        return 1;
    }

    // Закрываем ненужные концы pipes в родителе
    CloseHandle(to_child_read);
    CloseHandle(from_child_write);

    const char str[] = "Enter numbers (empty line to exit):\n";
    WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), str, sizeof(str) - 1, NULL, NULL);
    char buffer[BUF_SIZE];
    DWORD bytes_read, bytes_written;//хранят количество прочитанных и записанных байтов
    int should_exit = 0;
    while (!should_exit)
    {
        //считывание ввода пользователя
        if (!ReadFile(GetStdHandle(STD_INPUT_HANDLE), buffer, BUF_SIZE, &bytes_read, NULL))
        {
            const char error[] = "Error reading input\n";
            WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
            break;
        }
        // проверяем пустую строку для выхода
        if (bytes_read == 2 && buffer[0] == '\r' && buffer[1] == '\n')
        {
            const char exiting[] = "Exiting...\n";
            WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), exiting, sizeof(exiting) - 1, NULL,
NULL);
            break;
        }
        // проверка что введено число
        buffer[bytes_read] = '\0';
        if (buffer[bytes_read - 2] == '\r')
            buffer[bytes_read - 2] = '\0';
        else if (buffer[bytes_read - 1] == '\n') //убираем символы \r\n из конца строки
            buffer[bytes_read - 1] = '\0';
        if (!is_number(buffer))
        {
            const char notnumber[] = "Error: please enter a number\n";
            WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), notnumber, sizeof(notnumber) - 1, NULL,
NULL);
            continue;//надеемся,что хоть со 2го раза введут число и продолжаем
        }
        // отправка числа серверу(запись числа в pipe к ребенку)
        if (!WriteFile(to_child_write, buffer, bytes_read, &bytes_written, NULL))
```

```
        {
            const char error[] = "Error sending to server\n";
            WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
            break;
        }
        // ожидание ответа от сервера(чтение ответа из pipe от ребенка)
        if (!ReadFile(from_child_read, buffer, BUF_SIZE, &bytes_read, NULL))
        {
            const char error[] = "Error reading from server\n";
            WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
            break;
        }
        // проверка не нужно ли завершиться
        if (bytes_read >= 8 && strncmp(buffer, "SHUTDOWN", 8) == 0)
        {
            const char shutdown[] = "Server requested shutdown (prime or negative number)\n";
            WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), shutdown, sizeof(shutdown) - 1, NULL,
NULL);
            should_exit = 1;//если ребенок прислал shutdown - устанавливаем флаг выхода
        }
        else
        {
            // Выводим результат(он хранится в буфере)
            WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), buffer, bytes_read, &bytes_written,
NULL);
            const char newline[] = "\n";
            WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), newline, sizeof(newline) - 1, NULL,
NULL);
        }
    }
    // Закрываем handles и ждем завершения ребенка
    CloseHandle(to_child_write);
    CloseHandle(from_child_read);
    CloseHandle(proc_info.hProcess);//последнее слово- имя для управления процессом
    CloseHandle(proc_info.hThread);//последнее слово - имя для управления потоками внутри
процесса

    return 0;

  }
```

**server.c**

```
#include <windows.h>
#include <string.h>

#define BUF_SIZE 1024

// Функция проверки на простоту
int is_prime(int n)
{
    if (n <= 1)
        return 0;
    if (n <= 3)
        return 1;
    if (n % 2 == 0 || n % 3 == 0)
        return 0;
    for (int i = 5; i * i <= n; i += 2)
    {
        if (n % i == 0)
            return 0;
    }
    return 1;
}

// Функция преобразования строки в число
int string_to_int(const char* str)
{
    int result = 0;
```

```
    int sign = 1;
    int i = 0;
    if (str[0] == '-')
    {
        sign = -1;
        i = 1;
    }
    while (str[i] >= '0' && str[i] <= '9')
    {
        result = result * 10 + (str[i] - '0');
        i++;
    }
    return result * sign;
}

// Функция преобразования числа в строку
void int_to_string(int num, char* buffer)
{
    if (num == 0)
    {
        buffer[0] = '0';
        buffer[1] = '\0';
        return;
    }
    int i = 0;
    int is_negative = 0;
    if (num < 0)
    {
        is_negative = 1;
        num = -num;
    }
    while (num > 0) //цифры получаются в обратном порядке
    {
        buffer[i++] = '0' + (num % 10);
        num /= 10;
    }
    if (is_negative)
    {
        buffer[i++] = '-';
    }
    buffer[i] = '\0';
    // получаем цифры в нормальном порядке
    for (int j = i - 1, k = 0; j > k; j--, k++)
    {
        char temp = buffer[k];
        buffer[k] = buffer[j];
        buffer[j] = temp;
    }
}


int main()
{
    // Открываем файл для записи составных чисел
    //только запись, другие могут читать файл, создать или перезаписать, обычный файл
    HANDLE file = CreateFile("composite_numbers.txt", GENERIC_WRITE, FILE_SHARE_READ,
        NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    if (file == INVALID_HANDLE_VALUE)
    {
        const char error[] = "Error: cannot create file composite_numbers.txt\n";
        WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
        return 1;
    }
    char buffer[BUF_SIZE];
    DWORD bytes_read, bytes_written;
    int continue_working = 1;
    while (continue_working)
    {
        // читаем число от клиента(из pipe)
```

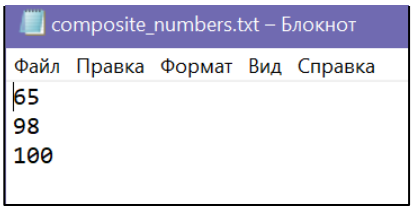
```
        if (!ReadFile(GetStdHandle(STD_INPUT_HANDLE), buffer, BUF_SIZE, &bytes_read, NULL))
        {
            const char error[] = "Error reading from client\n";
            WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
            break;
        }
        if (bytes_read == 0)
            break;
        // преобразуем в строку (убираем \r\n)
        buffer[bytes_read] = '\0';
        if (buffer[bytes_read - 2] == '\r')
            buffer[bytes_read - 2] = '\0';
        else if (buffer[bytes_read - 1] == '\n')
            buffer[bytes_read - 1] = '\0';
        // преобразуем строку в число
        int number = string_to_int(buffer);
        // проверяем условия
        if (number < 0)
        {
            // отрицательное число - завершаем работу
            const char response[] = "SHUTDOWN";//это отправляется родителю через pipe
            WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), response, sizeof(response) - 1,
&bytes_written, NULL);
            continue_working = 0;
            const char log[] = "Negative number, shutting down\n";//причина выхода
            WriteFile(GetStdHandle(STD_ERROR_HANDLE), log, sizeof(log) - 1, NULL, NULL);
        }
        else if (number == 0 || number == 1)
        {
            // Особые случаи
            const char response[] = "SHUTDOWN";  // Или другое сообщение
            WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), response, sizeof(response) - 1,
&bytes_written, NULL);
            continue_working = 0;
        }
        else if (is_prime(number))
        {
            // простое число - завершаем работу
            const char response[] = "SHUTDOWN";
            WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), response, sizeof(response) - 1,
&bytes_written, NULL);
            continue_working = 0;
            const char log[] = "prime number, shutting down\n";//причина выхода
            WriteFile(GetStdHandle(STD_ERROR_HANDLE), log, sizeof(log) - 1, NULL, NULL);
        }
        else
        {
            // составное число - пишем в файл и продолжаем
            char number_str[32];
            int_to_string(number, number_str);
            // записываем в файл
            int len = 0;
            while (number_str[len])
                len++;
            WriteFile(file, number_str, len, &bytes_written, NULL);
            WriteFile(file, "\r\n", 2, &bytes_written, NULL);
            // отправляем ответ клиенту
            char response[BUF_SIZE];
            int_to_string(number, response);
            int response_len = 0;
            while (response[response_len])
                response_len++;
            WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), response, response_len, &bytes_written,
NULL);
        }
    }
    CloseHandle(file);//системный вызов - закрытие файла
    return 0;
}
```

# Протокол работы программы

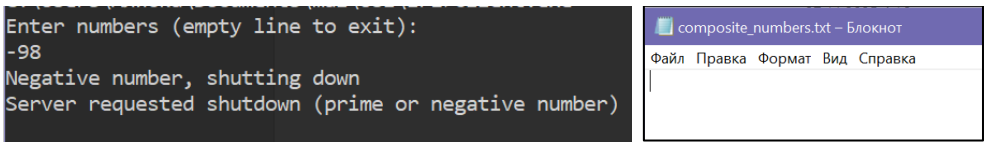
Ввод составных чисел и потом ввод 0:

```
Администратор: Командная строка
Microsoft Windows [Version 10.0.19045.3803]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Алиска>cd C:\Users\Алиска\Documents\mai\osi\lr1

C:\Users\Алиска\Documents\mai\osi\lr1>gcc client.c -o client.exe -lkernel32 -luser32

C:\Users\Алиска\Documents\mai\osi\lr1>gcc server.c -o server.exe -lkernel32 -luser32

C:\Users\Алиска\Documents\mai\osi\lr1>client.exe
Enter numbers (empty line to exit):
65
65
98
98
100
100
0
Server requested shutdown (prime or negative number)

C:\Users\Алиска\Documents\mai\osi\lr1>
```

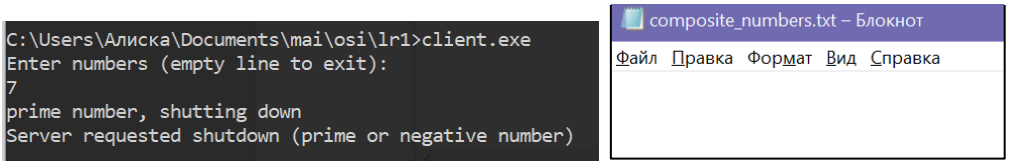
Что получилось в файле вывода:



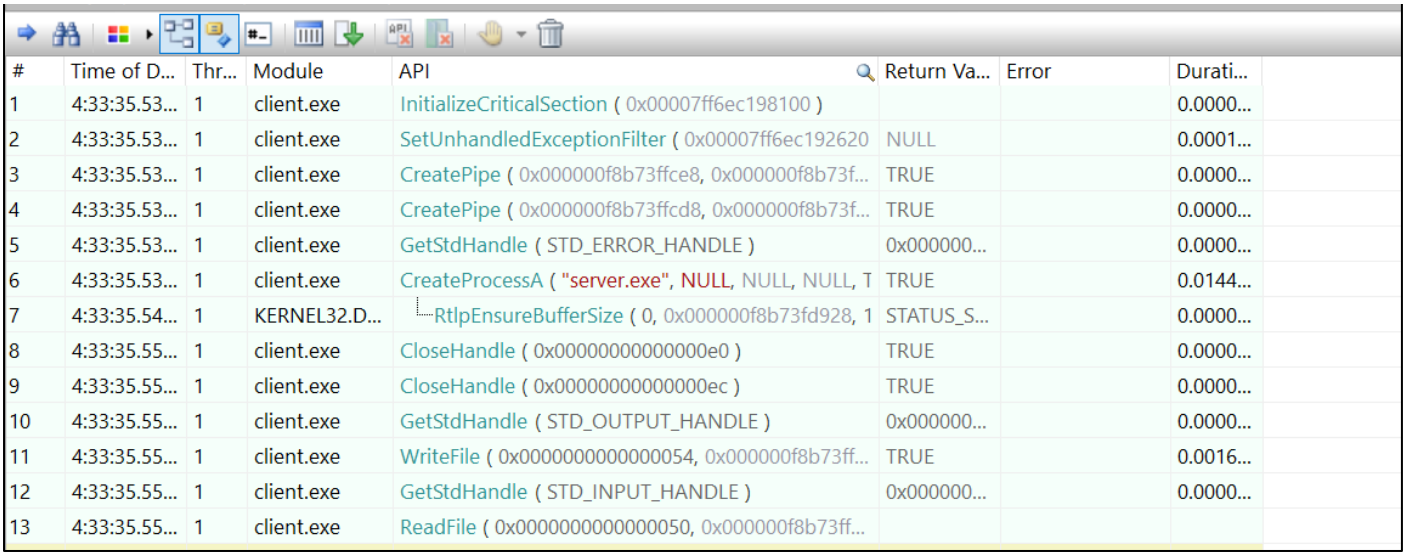

Ввод отрицательного числа:




Ввод простого числа:




Вывод системных вызовов



| # | Time of D... | Thr... | Module | API | Return Va... | Error | Durati... |
|---|---|---|---|---|---|---|---|
| 1 | 4:33:35.53... | 1 | client.exe | InitializeCriticalSection ( 0x00007ff6ec198100 ) | | | 0.0000... |
| 2 | 4:33:35.53... | 1 | client.exe | SetUnhandledExceptionFilter ( 0x00007ff6ec192620 | NULL | | 0.0001... |
| 3 | 4:33:35.53... | 1 | client.exe | CreatePipe ( 0x000000f8b73ffce8, 0x000000f8b73f... | TRUE | | 0.0000... |
| 4 | 4:33:35.53... | 1 | client.exe | CreatePipe ( 0x000000f8b73ffcd8, 0x000000f8b73f... | TRUE | | 0.0000... |
| 5 | 4:33:35.53... | 1 | client.exe | GetStdHandle ( STD_ERROR_HANDLE ) | 0x000000... | | 0.0000... |
| 6 | 4:33:35.53... | 1 | client.exe | CreateProcessA ( "server.exe", NULL, NULL, NULL, T | TRUE | | 0.0144... |
| 7 | 4:33:35.54... | 1 | KERNEL32.D... | RtlpEnsureBufferSize ( 0, 0x000000f8b73fd928, 1 | STATUS_S... | | 0.0000... |
| 8 | 4:33:35.55... | 1 | client.exe | CloseHandle ( 0x00000000000000e0 ) | TRUE | | 0.0000... |
| 9 | 4:33:35.55... | 1 | client.exe | CloseHandle ( 0x00000000000000ec ) | TRUE | | 0.0000... |
| 10 | 4:33:35.55... | 1 | client.exe | GetStdHandle ( STD_OUTPUT_HANDLE ) | 0x000000... | | 0.0000... |
| 11 | 4:33:35.55... | 1 | client.exe | WriteFile ( 0x0000000000000054, 0x000000f8b73ff... | TRUE | | 0.0016... |
| 12 | 4:33:35.55... | 1 | client.exe | GetStdHandle ( STD_INPUT_HANDLE ) | 0x000000... | | 0.0000... |
| 13 | 4:33:35.55... | 1 | client.exe | ReadFile ( 0x0000000000000050, 0x000000f8b73ff... | | | |

# Вывод

В ходе лабораторной работы была успешно реализована клиент-серверная система на языке C с использованием механизмов межпроцессорного взаимодействия Windows. Программа продемонстрировала создание анонимных каналов (pipes), создание дочернего процесса и организацию двустороннего обмена данными между процессами. Были использованы системные вызовы Windows API, включая CreatePipe, CreateProcess, ReadFile и WriteFile. Многие из них были реализованы с помощью дескрипторов handles. В конце программы все handles закрываются, чтобы не было утечек динамической памяти.