Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №3 по курсу

«Операционные системы»

Группа: М8О-214БВ-24

Студент: Чернявская Алиса Алексеевна

Преподаватель: Бахарев В.Д. (ФИИТ)

Оценка: _____

Дата: 20.10.25

Москва, 2025

Постановка задачи

Вариант 5.

Задание

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.

Программу выполнить с помощью использования семафоров для синхронизации и shared memory.

Общий метод и алгоритм решения

Использованные системные вызовы:

- CreateFileMapping() – создает объект разделяемой памяти
- OpenFileMapping() – открывает существующий объект разделяемой памяти
- MapViewOfFile() – отображает разделяемую память в адресное пространство процесса
- UnmapViewOfFile() – отключает отображение разделяемой памяти
- CreateSemaphore() – создает объект семафора
- OpenSemaphore() – открывает существующий семафор
- WaitForSingleObject() – захватывает семафор (уменьшает счетчик)
- ReleaseSemaphore() – освобождает семафор (увеличивает счетчик)
- CreateFile() – создает/открывает файл для записи составных чисел
- WriteFile() – записывает данные в файл или стандартный вывод
- ReadFile() – читает данные из стандартного ввода
- WriteFile() – выводит данные в стандартный вывод/ошибок
- GetStdHandle() – получает хэндлы стандартного ввода/вывода

В моей программе находятся 2 файла, обозначающие работу сервера и клиента. Файл-сервер создает разделяемую память shared memory и семафор, к которым может обращаться файл-клиент. Семафор нужен для ограничения доступа файлов к разделяемой памяти, поэтому оба процесса по очереди захватывают и отпускают его, что позволяет полностью владеть доступом к памяти и не потерять данные.

Пример работы программы: в клиенте пользователь вводит число, которое там же проверяется, точно ли оно является именно числом. Затем, если проверка успешна, клиент захватывает семафор (аналог wait в unix), и копирует число в разделяемую память. Далее клиент освобождает семафор и к разделяемой памяти может получить доступ сервер. Он уже захватывает семафор, читает число из shared memory и тоже делает проверки. Если число простое, 0, 1 или отрицательное, то сервер отправляет клиенту ответ «END» (также через разделяемую память), отпускает семафор, клиент принимает его и заканчивает работу. Если же число является

составным, то сервер записывает его в файл, отправляет это же число обратно клиенту и слово “continue”, клиент получает это число и ответ через разделяемую память (тоже используя семафор), выводит его и продолжает работу.

Код программы

client.c:

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

#define BUF_SIZE 1024
#define SHARED_MEM_SIZE 4096 //размер shared memory

//функция для проверки является ли введенная строчка числа
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(int argc, char* argv[]) //имя разделенной памяти и имя семафоры
{
    if (argc != 3) //количество аргументов
    {
        const char error[] = "wrong count of arguments\n";
        WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
        return 1;
    }

    const char* shared_mem_name = argv[1];
    const char* semaphore_name = argv[2];

    // открытие shared memory с полным доступом и без наследия дочернями процессами
    HANDLE hMapFile = OpenFileMapping(FILE_MAP_ALL_ACCESS, FALSE, shared_mem_name);
    if (hMapFile == NULL)
    {
        const char error[] = "cannot open shared memory. start server first!\n";
        WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
        return 1;
    }

    //отображение разделенной памяти в пространство процесса
    char* shared_mem = (char*)MapViewOfFile(hMapFile, FILE_MAP_ALL_ACCESS, 0, 0,
SHARED_MEM_SIZE);
    if (shared_mem == NULL)
    {
        const char error[] = "cannot map shared memory\n";
        WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
        CloseHandle(hMapFile);
        return 1;
    }

    // открытие семафоры
    HANDLE hSemaphore = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE, semaphore_name);
    if (hSemaphore == NULL)
    {
        const char error[] = "cannot open semaphore\n";
```

```
        WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
        UnmapViewOfFile(shared_mem);
        CloseHandle(hMapFile);
        return 1;
    }

    const char ready[] = "client is ready. enter numbers\n";
    WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), ready, sizeof(ready) - 1, NULL, NULL);

    char buffer[BUF_SIZE];
    DWORD bytes_read;
    int running = 1;

    while (running)
    {
        WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), "> ", 2, NULL, NULL);//вывод

        if (!ReadFile(GetStdHandle(STD_INPUT_HANDLE), buffer, BUF_SIZE, &bytes_read, NULL))
        {
            const char error[] = "error of input\n";
            WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
            break;
        }

        // убираем \r\n
        if (bytes_read >= 2 && buffer[bytes_read - 2] == '\r' && buffer[bytes_read - 1] ==
'\n')
        {
            buffer[bytes_read - 2] = '\0';
        }
        else if (bytes_read >= 1 && buffer[bytes_read - 1] == '\n')
        {
            buffer[bytes_read - 1] = '\0';
        }

        // если введена пустая строчка
        if (buffer[0] == '\0')
        {
            WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), "exit\n", 5, NULL, NULL);
            WaitForSingleObject(hSemaphore, INFINITE);//захват семафоы - то же что wait
            //теперь клиент может что-то делать, а сервер - нет
            strcpy(shared_mem, "EX");//записываем в память команду выхода и она отправится
серверу
            ReleaseSemaphore(hSemaphore, 1, NULL);//отпускаем семафору, теперь и сервер может
писать(счетчик+1)
            break;
        }


        if (!is_number(buffer))
        {
            const char error[] = "not valid number\n";
            WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), error, sizeof(error) - 1, NULL, NULL);
            continue;//продолжаем ввод
        }

        // отправка числа серверу
        WaitForSingleObject(hSemaphore, INFINITE);//захват семафоры,чтобы клиент мог писать в
shared memory
        strcpy(shared_mem, buffer);//запись в shared memory введенного числа
        ReleaseSemaphore(hSemaphore, 1, NULL);//отпускаем симафору, счетчик +=1 и сервер
моожет использовать

        //ожидание ответа от сервера
        int flag = 0;//флаг получен ли отет
        while (!flag && running)//пока ответ не получен
        {
            Sleep(100);//даем время получить ответ
            WaitForSingleObject(hSemaphore, INFINITE);//предполагается, что ответ получили
```

```
            //теперь захватываем симафору, чтоб обработать ответ
            if (strncmp(shared_mem, "STOP:", 5) == 0)
            {
                // найдено простое число или отрицательное значит остановка сервера
                const char msg[] = "server stopped";
                WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), msg, sizeof(msg) - 1, NULL, NULL);
                WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), shared_mem + 5, strlen(shared_mem +
5), NULL, NULL);
                WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), "\n", 1, NULL, NULL);
                running = 0;//останавливаем цикл
                flag = 1;//ответ получен
            }
            else if (strcmp(shared_mem, "CONTINUE") == 0)
            {
                // если получено составное число то продолжаем
                const char msg[] = "composite number - continue\n";
                WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), msg, sizeof(msg) - 1, NULL, NULL);
                flag = 1;//ответ получен
            }
            else if (strcmp(shared_mem, "END") == 0)
            {
                // если сервер завершил работу
                const char msg[] = "that's all\n";
                WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), msg, sizeof(msg) - 1, NULL, NULL);
                running = 0;//прерываем цикл обработки сообщения
                flag = 1;
            }

            if (flag)
            {
                shared_mem[0] = '\0';//добавляем в конец строчки символ конца строчки
            }
            ReleaseSemaphore(hSemaphore, 1, NULL);//отпускаем сервер, счетчик+=1
        }
    }
    CloseHandle(hSemaphore);
    UnmapViewOfFile(shared_mem);
    CloseHandle(hMapFile);
    return 0;

}
```

server.c

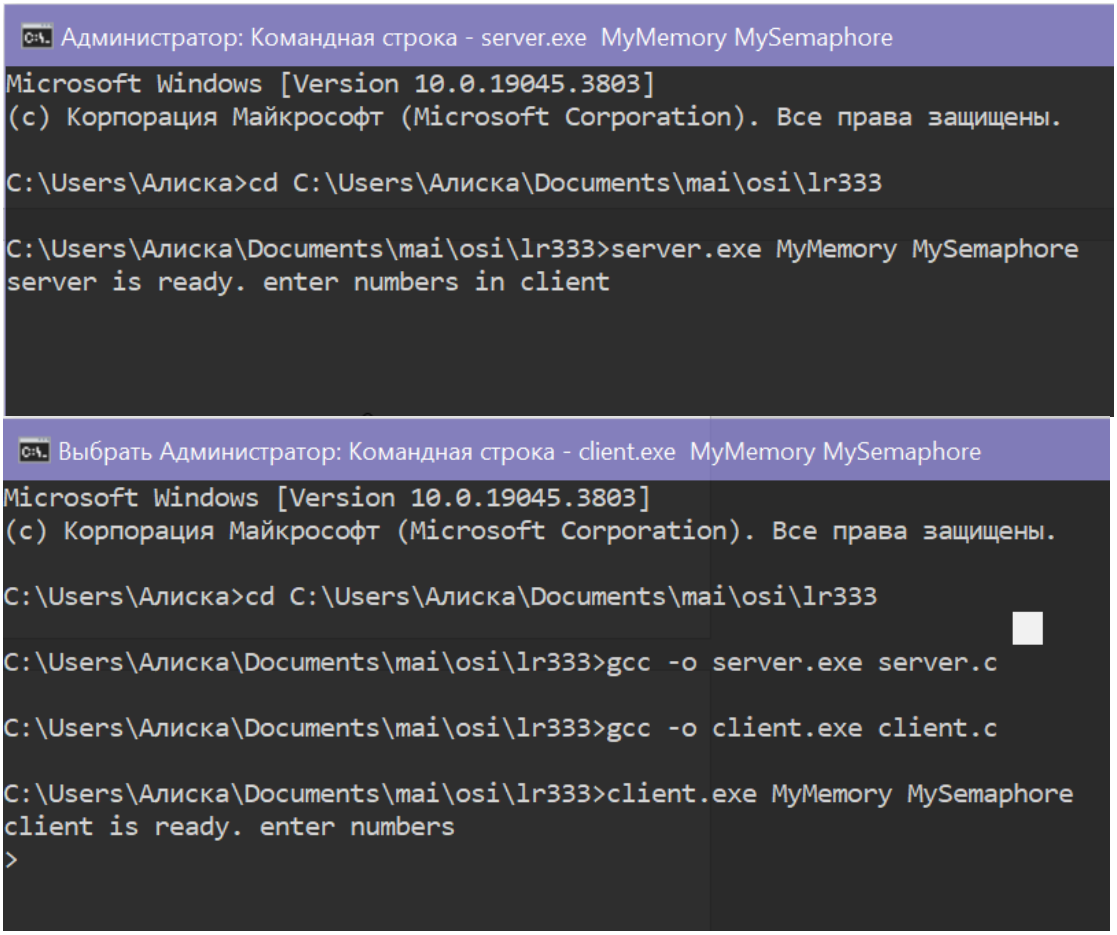
```
#include <windows.h>
#include <string.h>

#define BUF_SIZE 1024
#define SHARED_MEM_SIZE 4096

//функция проверки является ли число простым
int is_prime(int n)
{
    if (n <= 1) return 0;//0 и 1 ни простые, ни составные
    if (n <= 3) return 1;// 2 и 3 т.е 1 простое число
    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;
}

int main(int argc, char* argv[])
{
    if (argc != 3) //количество аргументов(имя программы и имена shared memory и семафоры
    {
        const char error[] = "wrong count of arguments\n";
        WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
```

```
        return 1;
    }

    const char* shared_mem_name = argv[1];
    const char* semaphore_name = argv[2];

    // создание shared memory с использованием системного файла подкачки
    HANDLE hMapFile = CreateFileMapping(INVALID_HANDLE_VALUE, NULL, PAGE_READWRITE, 0,
SHARED_MEM_SIZE, shared_mem_name);
    if (hMapFile == NULL)
    {
        const char error[] = "Error: cannot create shared memory\n";
        WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
        return 1;
    }

    //отображение shared memory в пространство процессора
    char* shared_mem = (char*)MapViewOfFile(hMapFile, FILE_MAP_ALL_ACCESS, 0, 0,
SHARED_MEM_SIZE);
    if (shared_mem == NULL)
    {
        const char error[] = "cannot create shared memory\n";
        WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
        CloseHandle(hMapFile);
        return 1;
    }

    // создание семафора с начальными значениями 1 по умолчанию и 1 как максимальное значение
    HANDLE hSemaphore = CreateSemaphore(NULL, 1, 1, semaphore_name);
    if (hSemaphore == NULL)
    {
        const char error[] = "cannot create semaphore\n";
        WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
        UnmapViewOfFile(shared_mem);
        CloseHandle(hMapFile);
        return 1;
    }

    // создание файла с результатами
    HANDLE file = CreateFile//системный вызов
    ("composite numbers.txt",
        GENERIC_WRITE,
        0,
        NULL,
        CREATE_ALWAYS,
        FILE_ATTRIBUTE_NORMAL,
        NULL);

    if (file == INVALID_HANDLE_VALUE) //если не получилось создать файл
    {
        const char error[] = "cannot create file\n";
        WriteFile(GetStdHandle(STD_ERROR_HANDLE), error, sizeof(error) - 1, NULL, NULL);
    }

    // инициализация shared memory
    shared_mem[0] = '\0';

    const char ready[] = "server is ready. enter numbers in client\n";
    WriteFile(GetStdHandle(STD_ERROR_HANDLE), ready, sizeof(ready) - 1, NULL, NULL);

    int running = 1;

    while (running)
    {
        WaitForSingleObject(hSemaphore, INFINITE);//захват семафоры чтобы использовать shared
memory
        // смотрим есть ли данные от клиента
        if (shared_mem[0] != '\0') //если получили данные от клиента
        {
```

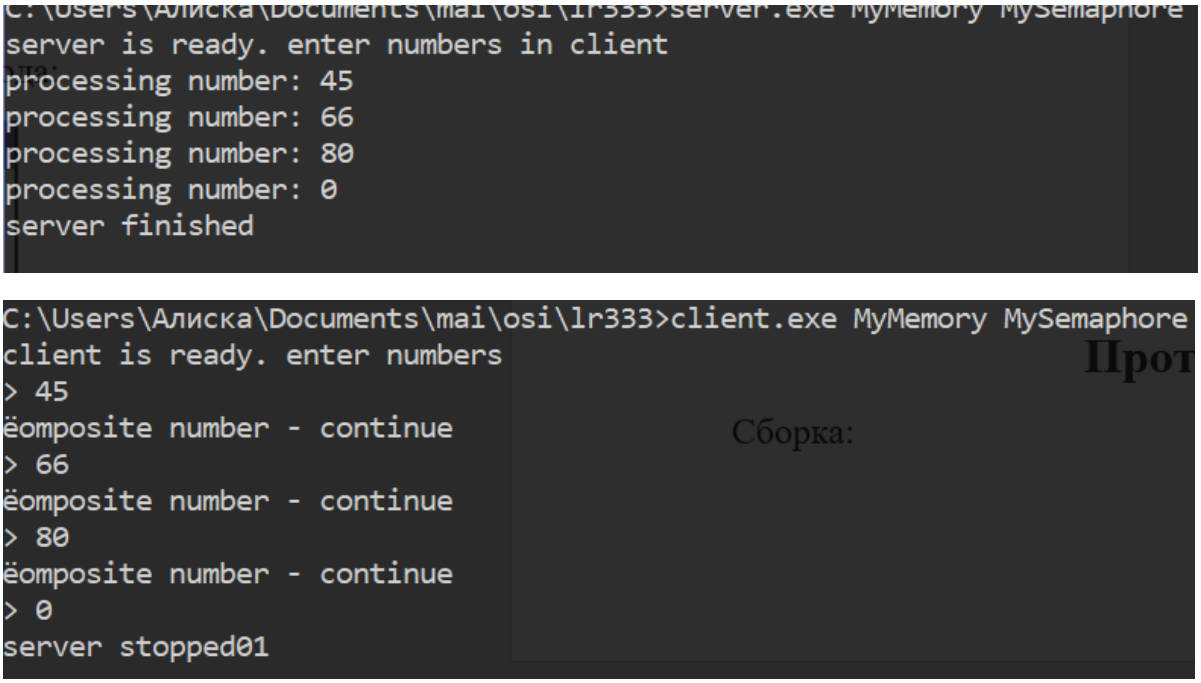
```
            if (shared_mem[0] == 'E' && shared_mem[1] == 'X') //если получили команду выхода
            {
                const char msg[] = "client stopped\n";
                WriteFile(GetStdHandle(STD_ERROR_HANDLE), msg, sizeof(msg) - 1, NULL, NULL);
                running = 0;
                strcpy(shared_mem, "END");//пишем в shared память команду завершиться
            }
            else
            {
                // обработка числа
                int num = atoi(shared_mem);//перевод числа из строки

                char str[100];
                strcpy(str, "processing number: ");
                strcat(str, shared_mem);
                WriteFile(GetStdHandle(STD_ERROR_HANDLE), str, strlen(str), NULL, NULL);
                WriteFile(GetStdHandle(STD_ERROR_HANDLE), "\n", 1, NULL, NULL);

                if (num < 0)
                {
                    strcpy(shared_mem, "STOP:negative");
                    running = 0;
                }
                else if (num == 0 || num == 1)
                {
                    strcpy(shared_mem, "STOP:01");
                    running = 0;
                }
                else if (is_prime(num))
                {
                    strcpy(shared_mem, "STOP:prime");
                    running = 0;
                }
                else
                {
                    // если составное число то записываем в файл и продолжаем
                    if (file != INVALID_HANDLE_VALUE)
                    {
                        DWORD bytes_written;
                        WriteFile(file, shared_mem, strlen(shared_mem), &bytes_written, NULL);
                        WriteFile(file, "\r\n", 2, &bytes_written, NULL);
                    }
                    strcpy(shared_mem, "CONTINUE");//отправляем команду продолжать
                }
            }
        }
        ReleaseSemaphore(hSemaphore, 1, NULL);//отпускаем семафору ура
        Sleep(100);//ждем чтоб полегче стало
    }
    const char finish[] = "server finished\n";
    WriteFile(GetStdHandle(STD_ERROR_HANDLE), finish, sizeof(finish) - 1, NULL, NULL);

    if (file != INVALID_HANDLE_VALUE)
    {
        CloseHandle(file);
    }
    CloseHandle(hSemaphore);
    UnmapViewOfFile(shared_mem);
    CloseHandle(hMapFile);
    return 0;

}
```

Протокол работы программы

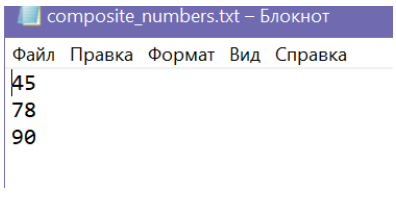
Сборка:



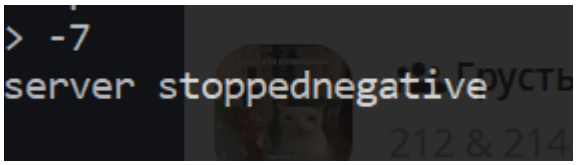
Ввод составных чисел и потом ввод 0:

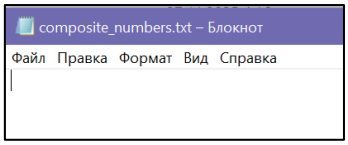


Что получилось в файле вывода:

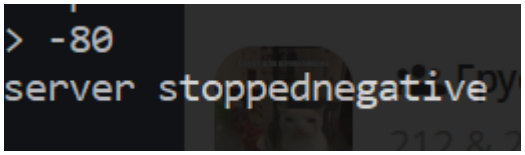


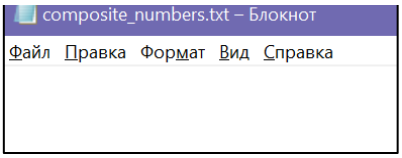
Ввод отрицательного числа:



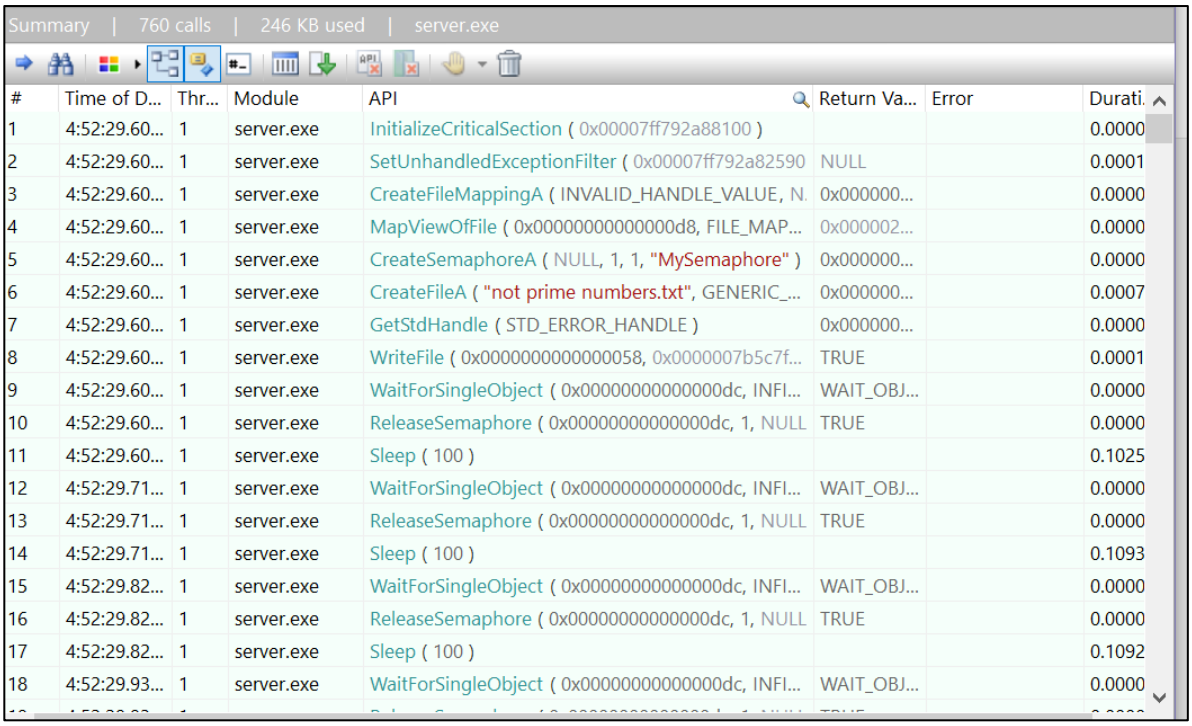


Ввод простого числа:





Вывод системных вызовов через API monitor:



#	Time of D...	Thr...	Module	API	Return Va...	Error	Durati.
1	4:52:29.60...	1	server.exe	InitializeCriticalSection (0x00007ff792a88100)			0.0000
2	4:52:29.60...	1	server.exe	SetUnhandledExceptionFilter (0x00007ff792a82590	NULL		0.0001
3	4:52:29.60...	1	server.exe	CreateFileMappingA (INVALID_HANDLE_VALUE, N.	0x000000...		0.0000
4	4:52:29.60...	1	server.exe	MapViewOfFile (0x00000000000000d8, FILE_MAP...	0x000002...		0.0000
5	4:52:29.60...	1	server.exe	CreateSemaphoreA (NULL, 1, 1, "MySemaphore")	0x000000...		0.0000
6	4:52:29.60...	1	server.exe	CreateFileA ("not prime numbers.txt", GENERIC_...	0x000000...		0.0007
7	4:52:29.60...	1	server.exe	GetStdHandle (STD_ERROR_HANDLE)	0x000000...		0.0000
8	4:52:29.60...	1	server.exe	WriteFile (0x0000000000000058, 0x0000007b5c7f...	TRUE		0.0001
9	4:52:29.60...	1	server.exe	WaitForSingleObject (0x00000000000000dc, INFI...	WAIT_OBJ...		0.0000
10	4:52:29.60...	1	server.exe	ReleaseSemaphore (0x00000000000000dc, 1, NULL	TRUE		0.0000
11	4:52:29.60...	1	server.exe	Sleep (100)			0.1025
12	4:52:29.71...	1	server.exe	WaitForSingleObject (0x00000000000000dc, INFI...	WAIT_OBJ...		0.0000
13	4:52:29.71...	1	server.exe	ReleaseSemaphore (0x00000000000000dc, 1, NULL	TRUE		0.0000
14	4:52:29.71...	1	server.exe	Sleep (100)			0.1093
15	4:52:29.82...	1	server.exe	WaitForSingleObject (0x00000000000000dc, INFI...	WAIT_OBJ...		0.0000
16	4:52:29.82...	1	server.exe	ReleaseSemaphore (0x00000000000000dc, 1, NULL	TRUE		0.0000
17	4:52:29.82...	1	server.exe	Sleep (100)			0.1092
18	4:52:29.93...	1	server.exe	WaitForSingleObject (0x00000000000000dc, INFI...	WAIT_OBJ...		0.0000

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.