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-24

Студент: Ельцова Д.А.

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 команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

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:

- CreateProcessA(...) – создает дочерний процесс с возможностью перенаправления стандартных потоков (stdin, stdout, stderr) и наследования дескрипторов.
- CreatePipe(HANDLE *readPipe, HANDLE *writePipe, LPSECURITY_ATTRIBUTES sa, DWORD size) – создает канал (pipe) между процессами; возвращает два дескриптора: один для чтения, другой для записи.
- WriteFile(HANDLE hFile, LPCVOID buffer, DWORD nNumberOfBytesToWrite, LPDWORD lpNumberOfBytesWritten, LPOVERLAPPED lpOverlapped) – записывает данные в дескриптор (pipe, файл или stdout).
- ReadFile(HANDLE hFile, LPVOID buffer, DWORD nNumberOfBytesToRead, LPDWORD lpNumberOfBytesRead, LPOVERLAPPED lpOverlapped) – читает данные из дескриптора (pipe или файл).
- GetStdHandle(DWORD nStdHandle) – возвращает дескриптор стандартного потока (stdin, stdout, stderr).
- CloseHandle(HANDLE hObject) – закрывает дескриптор, освобождая ресурсы операционной системы.
- WaitForSingleObject(HANDLE hHandle, DWORD dwMilliseconds) – ожидает завершения указанного процесса или события, блокируя выполнение до сигнала или таймаута.
- fopen(const char *filename, const char *mode) – открывает файл и возвращает указатель на поток FILE.
- fgets(char *str, int n, FILE *stream) – считывает строку из файла или потока.
- fclose(FILE *stream) – закрывает файл и освобождает ресурсы.
- sscanf(const char *str, const char *format, ...) – преобразует строку в данные, например строку в число ("%d").
- sprintf(char *str, const char *format, ...) – форматирует данные в строку (используется для подготовки строки для WriteFile).

В рамках лабораторной работы мы реализовали программу, состоящую из двух отдельных процессов: родительского (client.exe) и дочернего (server.exe). Основная цель программы —

организация межпроцессного взаимодействия с помощью каналов (pipe) и проверка чисел на простоту в дочернем процессе.

Ход работы программы:

1. Запуск родительского процесса (client.exe): пользователь в командной строке вводит имя файла с числами (например, .\client.exe 1.txt) и родительский процесс открывает указанный файл для чтения.
2. Создание pipe для передачи данных дочернему процессу: родитель создаёт pipe (pipe_stdin) для передачи содержимого файла в дочерний процесс; конец pipe для чтения (pipe_stdin[0]) перенаправляется на stdin дочернего процесса через STARTUPINFOA.si.hStdInput; конец pipe для записи (pipe_stdin[1]) используется родителем для записи данных из файла.
3. Создание pipe для получения данных от дочернего процесса: создаётся pipe (pipe1) для передачи stdout дочернего процесса родителю; конец pipe для записи (pipe1[1]) назначается stdout дочернего процесса; конец pipe для чтения (pipe1[0]) используется родителем для получения составных чисел и вывода их в консоль.
4. Создание дочернего процесса (server.exe): родитель запускает дочерний процесс через CreateProcessA с флагом bInheritHandles = TRUE, чтобы child унаследовал дескрипторы pipe; в STARTUPINFOA задаются перенаправленные дескрипторы stdin и stdout для child.
5. Передача данных из файла в дочерний процесс: родитель читает файл построчно через fgets и пишет строки в pipe_stdin с помощью WriteFile; после передачи всех данных родитель закрывает конец pipe для записи (pipe_stdin[1]) — child получает сигнал EOF и начинает обработку данных.
6. Обработка чисел в дочернем процессе: child читает строки из stdin (pipe_stdin) построчно; преобразует строки в числа через sscanf; для каждого числа выполняется проверка на простоту (если число составное, оно записывается в stdout через WriteFile (pipe1); если число простое или отрицательное, дочерний процесс завершает работу).
7. Чтение данных родителем и вывод в консоль: родитель читает данные из pipe1 (ReadFile) и выводит их на стандартный поток вывода (WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), ...)); после завершения дочернего процесса родитель закрывает все дескрипторы и завершает выполнение.

Код программы

client.c

```
#include <windows.h>

#include <stdio.h>

#include <string.h>


static char SERVER_PROGRAM_NAME[] = "server.exe";


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

    if (argc < 2) {

        const char msg[] = "Usage: client.exe <filename>\n";

        WriteFile(GetStdHandle(STD_ERROR_HANDLE), msg, sizeof(msg)-1, NULL, NULL);
```

```
    return 1;
}

char filename[256];
strcpy(filename, argv[1]);

// pipe: child stdout → parent
HANDLE pipe[2];
SECURITY_ATTRIBUTES sa = { sizeof(sa), NULL, TRUE };
if (!CreatePipe(&pipe[0], &pipe[1], &sa, 0)) {
    const char msg[] = "ERROR: cannot create pipe\n";
    WriteFile(GetStdHandle(STD_ERROR_HANDLE), msg, sizeof(msg)-1, NULL, NULL);
    return 1;
}

// путь к server.exe
char serverPath[MAX_PATH];
GetModuleFileNameA(NULL, serverPath, sizeof(serverPath));
char *lastSlash = strrchr(serverPath, '\\');
if (lastSlash) *lastSlash = '\0';
strcat(serverPath, "\\");
strcat(serverPath, SERVER_PROGRAM_NAME);

STARTUPINFOA si;
PROCESS_INFORMATION pi;
ZeroMemory(&si, sizeof(si));
si.cb = sizeof(si);

// pipe для stdin сервера
HANDLE pipe_stdin[2];
if (!CreatePipe(&pipe_stdin[0], &pipe_stdin[1], &sa, 0)) {
    const char msg[] = "ERROR: cannot create stdin pipe\n";
    WriteFile(GetStdHandle(STD_ERROR_HANDLE), msg, sizeof(msg)-1, NULL, NULL);
    return 1;
}

si.hStdInput  = pipe_stdin[0];
```

```
si.hStdOutput = pipe[1];
si.hStdError  = GetStdHandle(STD_ERROR_HANDLE);
si.dwFlags = STARTF_USESTDHANDLES;

if (!CreateProcessA(
    serverPath,
    NULL,
    NULL,
    NULL,
    TRUE, // наследовать дескрипторы
    0,
    NULL,
    NULL,
    &si,
    &pi
)) {
    const char msg[] = "ERROR: cannot start server.exe\n";
    WriteFile(GetStdHandle(STD_ERROR_HANDLE), msg, sizeof(msg)-1, NULL, NULL);
    return 1;
}

CloseHandle(pipe[1]);       // закрываем запись stdout сервера в родителе
CloseHandle(pipe_stdin[0]); // закрываем чтение stdin сервера в родителе

// читаем файл построчно и пишем в stdin сервера
FILE *file = fopen(filename, "r");
if (!file) {
    const char msg[] = "ERROR: cannot open file\n";
    WriteFile(GetStdHandle(STD_ERROR_HANDLE), msg, sizeof(msg)-1, NULL, NULL);
    return 1;
}

char line[128];
DWORD written;
while (fgets(line, sizeof(line), file)) {
    WriteFile(pipe_stdin[1], line, (DWORD)strlen(line), &written, NULL);
}
```

```
    fclose(file);
    CloseHandle(pipe_stdin[1]);

    // читаем вывод сервера и пишем в консоль
    char buffer[256];
    DWORD bytesRead;
    while (ReadFile(pipe[0], buffer, sizeof(buffer), &bytesRead, NULL) && bytesRead > 0) {
        WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), buffer, bytesRead, NULL, NULL);
    }

    CloseHandle(pipe[0]);
    WaitForSingleObject(pi.hProcess, INFINITE);
    CloseHandle(pi.hThread);
    CloseHandle(pi.hProcess);

    return 0;
}
```

server.c

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

int is_prime(int n) {
    if (n <= 1) return 0;
    for (int i = 2; i * i <= n; i++)
        if (n % i == 0) return 0;
    return 1;
}

int main() {
    HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
    char line[128];

    while (fgets(line, sizeof(line), stdin)) {
        int num;
        if (sscanf(line, "%d", &num) == 1) {
```

```
            if (num < 0 || is_prime(num)) break; // завершение
            char output[32];
            int len = sprintf(output, "%d\n", num);
            DWORD written;
            WriteFile(hOut, output, len, &written, NULL);
        }
    }
    return 0;
}
```

Протокол работы программы

Тестирование:

```
PS C:\Users\daria\Desktop\oslr1> gcc -o client.exe client.c
PS C:\Users\daria\Desktop\oslr1> gcc -o server.exe server.c
PS C:\Users\daria\Desktop\oslr1> .\client.exe 1.txt
4
6
8
9
10
12
14
15
Stop number found: 7
Program finished.
PS C:\Users\daria\Desktop\oslr1> .\client.exe 2.txt
4
6
Stop number found: 3
Program finished.
PS C:\Users\daria\Desktop\oslr1> .\client.exe 3.txt
9
10
Stop number found: -1
Program finished.
```

```
PS C:\Users\daria\Desktop\oslr1> .\client.exe 4.txt
Stop number found: 3
Program finished.
PS C:\Users\daria\Desktop\oslr1> .\client.exe 5.txt
100
Stop number found: 101
Program finished.
PS C:\Users\daria\Desktop\oslr1>
```

Procmon:

Time of Day Process Name Operation Path Result Detail

20:00.3 server.exe Process Start C:\Users\daria\Desktop\oslr1 SUCCESS Parent PID: 22348, Command line: "C:\Users\daria\Desktop\oslr1\server.exe", Current directory: C:\Users\daria\Desktop\oslr1\

20:00.3 server.exe Thread Create - SUCCESS Thread ID: 16104

20:00.3 server.exe Load Image C:\Users\daria\Desktop\oslr1\server.exe SUCCESS Image Base: 0x7ff73e460000, Image Size: 0x13000

20:00.3 server.exe CreateFile C:\Users\daria\Desktop\oslr1 SUCCESS Desired Access: Execute/Traverse, Synchronize, Disposition: Open

20:00.3 server.exe Thread Create - SUCCESS Thread ID: 23992

20:00.3 server.exe QueryNameInformationFile C:\Users\daria\Desktop\oslr1\server.exe SUCCESS Name: \Users\daria\Desktop\oslr1\server.exe

20:00.3 server.exe Thread Exit - SUCCESS Thread ID: 23992, User Time: 0.0000000, Kernel Time: 0.0000000

20:00.3 server.exe Thread Exit - SUCCESS Thread ID: 16104, User Time: 0.0000000, Kernel Time: 0.0156250

20:00.3 server.exe Process Exit - SUCCESS Exit Status: 0, User Time: 0.0000000 seconds, Kernel Time: 0.0156250 seconds, Private Bytes: 643,072, Peak Private Bytes: 643,072, Working Set: 4,296,704, Peak Working Set: 4,296,704

20:00.3 server.exe CloseFile C:\Users\daria\Desktop\oslr1 SUCCESS

Вывод

В ходе лабораторной работы я изучила, как программа взаимодействует с операционной системой через системные вызовы. Основной задачей было проанализировать выполнение программы и отследить важные события с помощью Procmon. Работа помогла мне лучше понять процессы и механизмы работы программ в Windows.