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: Шестаков К. Р.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 15.11.24

Москва, 2024

**Постановка задачи**

**Вариант** 2**.**

Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип float. Количество чисел может быть произвольным.

Переделать с использованием shared memory и memory mapping

**Общий метод и алгоритм решения**

Задача решена с использованием разделяемой памяти и семафоров для синхронизации межпроцессного взаимодействия. Родительский процесс считывает данные от пользователя и записывает их в разделяемую память. Дочерний процесс считывает данные из разделяемой памяти, обрабатывает их и записывает результат в файл. Синхронизация доступа к разделяемой памяти осуществляется с помощью двух семафоров: один для записи (родительский процесс), другой для чтения (дочерний процесс). Для завершения работы дочернего процесса родительский записывает нулевой размер данных в разделяемую память.

* **CreateFileMapping**: Создает именованный объект разделяемой памяти.
* **MapViewOfFile**: Отображает разделяемую память в адресное пространство процесса.
* **UnmapViewOfFile**: Отменяет отображение разделяемой памяти.
* **OpenFileMapping**: Открывает существующий объект разделяемой памяти.
* **CreateSemaphore**: Создает именованный семафор.
* **OpenSemaphore**: Открывает существующий именованный семафор.
* **WaitForSingleObject**: Ожидает освобождения семафора.
* **ReleaseSemaphore**: Освобождает семафор.
* **CreateProcess**: Создает новый процесс.
* **GetExitCodeProcess**: Получает код выхода завершенного процесса.
* **CloseHandle**: Закрывает дескриптор объекта.
* **ReadFile**: Читает данные из файла или консоли.
* **WriteFile**: Записывает данные в файл или консоль.
* **GetFullPathName**: Получает полный путь к файлу.
* **GetModuleFileName**: Получает полный путь к исполняемому файлу текущего процесса.

Алгоритм работы программы

1. **Родительский процесс:**
   * Создает разделяемую память и два семафора (чтения и записи).
   * Создает дочерний процесс, передавая ему имя файла для вывода.
   * Циклически считывает данные от пользователя.
   * Захватывает семафор записи.
   * Копирует данные в разделяемую память, включая размер данных.
   * Освобождает семафор чтения.
   * После завершения ввода записывает нулевой размер данных, сигнализируя дочернему процессу о завершении.
   * Ожидает завершения дочернего процесса.
   * Освобождает ресурсы (разделяемая память, семафоры).
2. **Дочерний процесс:**
   * Открывает разделяемую память и семафоры.
   * Открывает файл для записи.
   * Циклически:
     + Захватывает семафор чтения.
     + Копирует данные из разделяемой памяти.
     + Освобождает семафор записи.
     + Если размер данных равен нулю, завершает цикл.
     + Обрабатывает данные (суммирует числа).
     + Записывает результат в файл.

**Код программы**

**parent.c**

#include <windows.h>

#define BUFFER\_SIZE 1024

#define SHARED\_MEM\_SIZE (BUFFER\_SIZE \* sizeof(char))

#define SHARED\_MEM\_NAME "Local\\SharedMemBuffer"

#define READ\_SEMAPHORE\_NAME "Local\\ReadSemaphore"

#define WRITE\_SEMAPHORE\_NAME "Local\\WriteSemaphore"

int main()

{

    char error\_msg[256];

    HANDLE hMapFile;

    LPVOID pBuf;

    HANDLE hReadSemaphore, hWriteSemaphore;

    STARTUPINFO si;

    PROCESS\_INFORMATION pi;

    char buffer[BUFFER\_SIZE];

    char filename[MAX\_PATH];

    DWORD bytes\_read, bytes\_written;

    if (!ReadFile(GetStdHandle(STD\_INPUT\_HANDLE), filename, sizeof(filename), &bytes\_read, NULL))

        return 1;

    if (bytes\_read >= 2 && filename[bytes\_read - 2] == '\r' && filename[bytes\_read - 1] == '\n')

        filename[bytes\_read - 2] = '\0';

    else if (bytes\_read >= 1 && (filename[bytes\_read - 1] == '\n' || filename[bytes\_read - 1] == '\r'))

        filename[bytes\_read - 1] = '\0';

    else

        filename[bytes\_read] = '\0';

    char full\_path[MAX\_PATH];

    DWORD result = GetFullPathName(filename, MAX\_PATH, full\_path, NULL);

    if (result == 0 || result > MAX\_PATH)

    {

        WriteFile(GetStdHandle(STD\_ERROR\_HANDLE), error\_msg, 22, &bytes\_written, NULL);

        return 1;

    }

    hMapFile = CreateFileMapping(

        INVALID\_HANDLE\_VALUE,

        NULL,

        PAGE\_READWRITE,

        0,

        SHARED\_MEM\_SIZE,

        SHARED\_MEM\_NAME);

    if (hMapFile == NULL)

    {

        WriteFile(GetStdHandle(STD\_ERROR\_HANDLE), error\_msg, 29, &bytes\_written, NULL);

        return 1;

    }

    pBuf = MapViewOfFile(

        hMapFile,

        FILE\_MAP\_ALL\_ACCESS,

        0,

        0,

        SHARED\_MEM\_SIZE);

    if (pBuf == NULL)

    {

        WriteFile(GetStdHandle(STD\_ERROR\_HANDLE), error\_msg, 26, &bytes\_written, NULL);

        CloseHandle(hMapFile);

        return 1;

    }

    hReadSemaphore = CreateSemaphore(NULL, 0, 1, READ\_SEMAPHORE\_NAME);

    hWriteSemaphore = CreateSemaphore(NULL, 1, 1, WRITE\_SEMAPHORE\_NAME);

    if (hReadSemaphore == NULL || hWriteSemaphore == NULL)

    {

        WriteFile(GetStdHandle(STD\_ERROR\_HANDLE), error\_msg, 27, &bytes\_written, NULL);

        UnmapViewOfFile(pBuf);

        CloseHandle(hMapFile);

        if (hReadSemaphore)

            CloseHandle(hReadSemaphore);

        if (hWriteSemaphore)

            CloseHandle(hWriteSemaphore);

        return 1;

    }

    ZeroMemory(&si, sizeof(STARTUPINFO));

    si.cb = sizeof(STARTUPINFO);

    si.hStdOutput = GetStdHandle(STD\_OUTPUT\_HANDLE);

    si.hStdError = GetStdHandle(STD\_ERROR\_HANDLE);

    si.dwFlags |= STARTF\_USESTDHANDLES;

    char cmd\_line[MAX\_PATH \* 2];

    wsprintf(cmd\_line, "child.exe \"%s\"", full\_path);

    char module\_dir[MAX\_PATH];

    GetModuleFileName(NULL, module\_dir, MAX\_PATH);

    char \*last\_slash = strrchr(module\_dir, '\\');

    if (last\_slash)

        \*last\_slash = '\0';

    if (!CreateProcess(NULL, cmd\_line, NULL, NULL, TRUE, 0, NULL, module\_dir, &si, &pi))

    {

        DWORD error = GetLastError();

        WriteFile(GetStdHandle(STD\_ERROR\_HANDLE), error\_msg, lstrlen(error\_msg), &bytes\_written, NULL);

        UnmapViewOfFile(pBuf);

        CloseHandle(hMapFile);

        CloseHandle(hReadSemaphore);

        CloseHandle(hWriteSemaphore);

        return 1;

    }

    CloseHandle(pi.hThread);

    while (ReadFile(GetStdHandle(STD\_INPUT\_HANDLE), buffer, BUFFER\_SIZE - sizeof(DWORD), &bytes\_read, NULL) && bytes\_read > 0)

    {

        WaitForSingleObject(hWriteSemaphore, INFINITE);

        memcpy(pBuf, buffer, bytes\_read);

        \*((DWORD \*)((char \*)pBuf + BUFFER\_SIZE - sizeof(DWORD))) = bytes\_read;

        ReleaseSemaphore(hReadSemaphore, 1, NULL);

    }

    WaitForSingleObject(hWriteSemaphore, INFINITE);

    \*((DWORD \*)((char \*)pBuf + BUFFER\_SIZE - sizeof(DWORD))) = 0;

    ReleaseSemaphore(hReadSemaphore, 1, NULL);

    WaitForSingleObject(pi.hProcess, INFINITE);

    DWORD exit\_code;

    GetExitCodeProcess(pi.hProcess, &exit\_code);

    CloseHandle(pi.hProcess);

    UnmapViewOfFile(pBuf);

    CloseHandle(hMapFile);

    CloseHandle(hReadSemaphore);

    CloseHandle(hWriteSemaphore);

    return exit\_code;

}

**child.c**

#include <windows.h>

#define BUFFER\_SIZE 1024

#define SHARED\_MEM\_SIZE (BUFFER\_SIZE \* sizeof(char))

#define SHARED\_MEM\_NAME "Local\\SharedMemBuffer"

#define READ\_SEMAPHORE\_NAME "Local\\ReadSemaphore"

#define WRITE\_SEMAPHORE\_NAME "Local\\WriteSemaphore"

float string\_to\_float(const char \*str)

{

    float result = 0;

    float fraction = 0;

    float div = 1;

    int negative = 0;

    while (\*str == ' ')

        str++;

    if (\*str == '-')

    {

        negative = 1;

        str++;

    }

    while (\*str >= '0' && \*str <= '9')

    {

        result = result \* 10 + (\*str - '0');

        str++;

    }

    if (\*str == '.')

    {

        str++;

        while (\*str >= '0' && \*str <= '9')

        {

            div \*= 10;

            fraction = fraction \* 10 + (\*str - '0');

            str++;

        }

    }

    result += fraction / div;

    return negative ? -result : result;

}

void float\_to\_string(float num, char \*str)

{

    int integer\_part = (int)num;

    float decimal\_part = num - integer\_part;

    int idx = 0;

    int temp;

    if (num < 0)

    {

        str[idx++] = '-';

        integer\_part = -integer\_part;

        decimal\_part = -decimal\_part;

    }

    temp = integer\_part;

    int start\_idx = idx;

    do

    {

        str[idx++] = '0' + (temp % 10);

        temp /= 10;

    } while (temp > 0);

    int end\_idx = idx - 1;

    while (start\_idx < end\_idx)

    {

        char t = str[start\_idx];

        str[start\_idx] = str[end\_idx];

        str[end\_idx] = t;

        start\_idx++;

        end\_idx--;

    }

    str[idx++] = '.';

    decimal\_part \*= 100;

    temp = (int)decimal\_part;

    str[idx++] = '0' + (temp / 10);

    str[idx++] = '0' + (temp % 10);

    str[idx++] = '\n';

    str[idx] = '\0';

}

BOOL validate\_and\_fix\_path(char \*path)

{

    int len = lstrlen(path);

    if (len > 0)

    {

        if (path[0] == '"')

        {

            memmove(path, path + 1, len);

            len--;

        }

        if (len > 0 && path[len - 1] == '"')

        {

            path[len - 1] = '\0';

            len--;

        }

    }

    if (len == 0 || len >= MAX\_PATH)

        return FALSE;

    for (int i = 0; i < len; i++)

    {

        if (path[i] < 32 || strchr("<>|?\*\"", path[i]))

            return FALSE;

    }

    return TRUE;

}

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

{

    char error\_msg[256];

    DWORD bytes\_written;

    HANDLE hMapFile;

    LPVOID pBuf;

    HANDLE hReadSemaphore, hWriteSemaphore;

    if (argc != 2)

    {

        WriteFile(GetStdHandle(STD\_ERROR\_HANDLE), error\_msg, 17, &bytes\_written, NULL);

        return 1;

    }

    char file\_path[MAX\_PATH];

    lstrcpy(file\_path, argv[1]);

    if (!validate\_and\_fix\_path(file\_path))

    {

        WriteFile(GetStdHandle(STD\_ERROR\_HANDLE), error\_msg, 17, &bytes\_written, NULL);

        return 1;

    }

    hMapFile = OpenFileMapping(

        FILE\_MAP\_ALL\_ACCESS,

        FALSE,

        SHARED\_MEM\_NAME);

    if (hMapFile == NULL)

    {

        WriteFile(GetStdHandle(STD\_ERROR\_HANDLE), error\_msg, 27, &bytes\_written, NULL);

        return 1;

    }

    pBuf = MapViewOfFile(

        hMapFile,

        FILE\_MAP\_ALL\_ACCESS,

        0,

        0,

        SHARED\_MEM\_SIZE);

    if (pBuf == NULL)

    {

        WriteFile(GetStdHandle(STD\_ERROR\_HANDLE), error\_msg, 26, &bytes\_written, NULL);

        CloseHandle(hMapFile);

        return 1;

    }

    hReadSemaphore = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, READ\_SEMAPHORE\_NAME);

    hWriteSemaphore = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, WRITE\_SEMAPHORE\_NAME);

    if (hReadSemaphore == NULL || hWriteSemaphore == NULL)

    {

        WriteFile(GetStdHandle(STD\_ERROR\_HANDLE), error\_msg, 25, &bytes\_written, NULL);

        UnmapViewOfFile(pBuf);

        CloseHandle(hMapFile);

        if (hReadSemaphore)

            CloseHandle(hReadSemaphore);

        if (hWriteSemaphore)

            CloseHandle(hWriteSemaphore);

        return 1;

    }

    HANDLE output\_file = CreateFile(

        file\_path,

        GENERIC\_WRITE,

        0,

        NULL,

        CREATE\_ALWAYS,

        FILE\_ATTRIBUTE\_NORMAL,

        NULL);

    if (output\_file == INVALID\_HANDLE\_VALUE)

    {

        DWORD error = GetLastError();

        WriteFile(GetStdHandle(STD\_ERROR\_HANDLE), error\_msg, lstrlen(error\_msg), &bytes\_written, NULL);

        UnmapViewOfFile(pBuf);

        CloseHandle(hMapFile);

        CloseHandle(hReadSemaphore);

        CloseHandle(hWriteSemaphore);

        return 1;

    }

    char num\_str[32];

    float sum;

    int num\_start = 0;

    BOOL running = TRUE;

    char debug\_msg[256];

    while (running)

    {

        WaitForSingleObject(hReadSemaphore, INFINITE);

        DWORD bytes\_read = \*((DWORD \*)((char \*)pBuf + BUFFER\_SIZE - sizeof(DWORD)));

        WriteFile(GetStdHandle(STD\_ERROR\_HANDLE), debug\_msg, lstrlen(debug\_msg), &bytes\_written, NULL);

        if (bytes\_read == 0)

        {

            WriteFile(GetStdHandle(STD\_ERROR\_HANDLE), error\_msg, 19, &bytes\_written, NULL);

            running = FALSE;

        }

        else

        {

            sum = 0;

            num\_start = 0;

            char \*buffer = (char \*)pBuf;

            for (int i = 0; i < bytes\_read; i++)

            {

                if (buffer[i] == ' ' || buffer[i] == '\n')

                {

                    if (i > num\_start)

                    {

                        int len = i - num\_start;

                        if (len < sizeof(num\_str))

                        {

                            memcpy(num\_str, buffer + num\_start, len);

                            num\_str[len] = '\0';

                            float val = string\_to\_float(num\_str);

                            sum += val;

                            WriteFile(GetStdHandle(STD\_ERROR\_HANDLE), debug\_msg, lstrlen(debug\_msg), &bytes\_written, NULL);

                        }

                    }

                    num\_start = i + 1;

                    if (buffer[i] == '\n')

                    {

                        char result[64];

                        float\_to\_string(sum, result);

                        WriteFile(GetStdHandle(STD\_ERROR\_HANDLE), debug\_msg, lstrlen(debug\_msg), &bytes\_written, NULL);

                        if (!WriteFile(output\_file, result, lstrlen(result), &bytes\_written, NULL))

                        {

                            WriteFile(GetStdHandle(STD\_ERROR\_HANDLE), "", 22, &bytes\_written, NULL);

                            running = FALSE;

                            break;

                        }

                        sum = 0;

                    }

                }

            }

        }

        ReleaseSemaphore(hWriteSemaphore, 1, NULL);

    }

    CloseHandle(output\_file);

    UnmapViewOfFile(pBuf);

    CloseHandle(hMapFile);

    CloseHandle(hReadSemaphore);

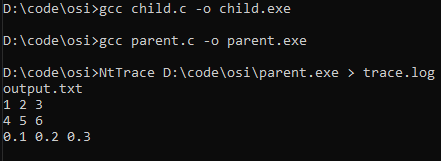
    CloseHandle(hWriteSemaphore);

    return 0;

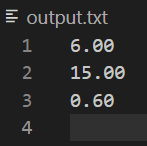
}

**Протокол работы программы**

**Тестирование:**



(Скриншот консоли родительского процесса)



(Содержимое файла output.txt)

**nttrace:**

**NtReleaseSemaphore**

**NtOpenSemaphore**

**WaitForSingleObject**

NtCreateMutant(MutantHandle=0x73bc9fe4f8 [0x6c], DesiredAccess=DELETE|READ\_CONTROL|WRITE\_DAC|WRITE\_OWNER|SYNCHRONIZE|0x1, ObjectAttributes=0x70:"Local\SM0:17600:304:WilStaging\_02", InitialOwner=false) => 0

NtWaitForSingleObject(Handle=0x6c, Alertable=false, Timeout=null) => 0

NtOpenSemaphore(SemaphoreHandle=0x73bc9fe2e8, DesiredAccess=DELETE|READ\_CONTROL|WRITE\_DAC|WRITE\_OWNER|SYNCHRONIZE|0x3, ObjectAttributes=0x70:"Local\SM0:17600:304:WilStaging\_02\_p0") => 0xc0000034 [2 'Не удается найти указанный файл.']

NtCreateEvent(EventHandle=0x7ffddaa1c478 [8], DesiredAccess=DELETE|READ\_CONTROL|WRITE\_DAC|WRITE\_OWNER|SYNCHRONIZE|0x3, ObjectAttributes=null, EventType=0 [NotificationEvent], InitialState=false) => 0

NtWaitForSingleObject(Handle=0xcc, Alertable=false, Timeout=null) => 0

NtQuerySystemInformation(SystemInformationClass=0 [SystemBasicInformation], SystemInformation=0xec5e9ff2c0, Length=0x40, ReturnLength=null) => 0

NtCreateMutant(MutantHandle=0xec5e9fe5b8 [0x60], DesiredAccess=DELETE|READ\_CONTROL|WRITE\_DAC|WRITE\_OWNER|SYNCHRONIZE|0x1, ObjectAttributes=0x64:"Local\SM0:13808:304:WilStaging\_02", InitialOwner=false) => 0

NtWaitForSingleObject(Handle=0x60, Alertable=false, Timeout=null) => 0

NtOpenSemaphore(SemaphoreHandle=0xec5e9fe3a8, DesiredAccess=DELETE|READ\_CONTROL|WRITE\_DAC|WRITE\_OWNER|SYNCHRONIZE|0x3, ObjectAttributes=0x64:"Local\SM0:13808:304:WilStaging\_02\_p0") => 0xc0000034 [2 'Не удается найти указанный файл.']

NtReleaseSemaphore(SemaphoreHandle=0xac, Count=1, PreviousCount=null) => 0

NtWaitForSingleObject(Handle=0xcc, Alertable=false, Timeout=null) => 0

NtReleaseSemaphore(SemaphoreHandle=0xc8, Count=1, PreviousCount=null) => 0

NtWaitForSingleObject(Handle=0xa8, Alertable=false, Timeout=null) => 0

NtWriteFile(FileHandle=0x58, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0xec5e9ff3c0 [0/1], Buffer=0xec5e9ff460, Length=1, ByteOffset=null, Key=null) => 0

NtWriteFile(FileHandle=0x58, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0xec5e9ff3c0 [0/1], Buffer=0xec5e9ff460, Length=1, ByteOffset=null, Key=null) => 0

NtWriteFile(FileHandle=0x58, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0xec5e9ff3c0 [0/1], Buffer=0xec5e9ff460, Length=1, ByteOffset=null, Key=null) => 0

NtWriteFile(FileHandle=0x58, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0xec5e9ff3c0 [0/1], Buffer=0xec5e9ff460, Length=1, ByteOffset=null, Key=null) => 0

NtWriteFile(FileHandle=0x58, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0xec5e9ff3c0 [0/1], Buffer=0xec5e9ff460, Length=1, ByteOffset=null, Key=null) => 0

NtWriteFile(FileHandle=0xb0, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0xec5e9ff3c0 [0/6], Buffer=0xec5e9ff420, Length=6, ByteOffset=null, Key=null) => 0

NtReleaseSemaphore(SemaphoreHandle=0xac, Count=1, PreviousCount=null) => 0

NtReadFile(FileHandle=0x50, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x73bc9feb90 [0/0xd], Buffer=0x73bc9ff160, Length=0x3fc, ByteOffset=null, Key=null) => 0

NtWaitForSingleObject(Handle=0xcc, Alertable=false, Timeout=null) => 0

NtReleaseSemaphore(SemaphoreHandle=0xc8, Count=1, PreviousCount=null) => 0

NtWaitForSingleObject(Handle=0xa8, Alertable=false, Timeout=null) => 0

NtWriteFile(FileHandle=0x58, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0xec5e9ff3c0 [0/1], Buffer=0xec5e9ff460, Length=1, ByteOffset=null, Key=null) => 0

NtWriteFile(FileHandle=0x58, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0xec5e9ff3c0 [0/1], Buffer=0xec5e9ff460, Length=1, ByteOffset=null, Key=null) => 0

NtWriteFile(FileHandle=0x58, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0xec5e9ff3c0 [0/1], Buffer=0xec5e9ff460, Length=1, ByteOffset=null, Key=null) => 0

NtWriteFile(FileHandle=0x58, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0xec5e9ff3c0 [0/1], Buffer=0xec5e9ff460, Length=1, ByteOffset=null, Key=null) => 0

NtWriteFile(FileHandle=0x58, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0xec5e9ff3c0 [0/1], Buffer=0xec5e9ff460, Length=1, ByteOffset=null, Key=null) => 0

NtWriteFile(FileHandle=0xb0, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0xec5e9ff3c0 [0/5], Buffer=0xec5e9ff420, Length=5, ByteOffset=null, Key=null) => 0

NtReleaseSemaphore(SemaphoreHandle=0xac, Count=1, PreviousCount=null) => 0

NtReadFile(FileHandle=0x50, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x73bc9feb90 [0x101/0], Buffer=0x73bc9ff160, Length=0x3fc, ByteOffset=null, Key=null) => 0x101 [739 'ERROR\_ALERTED']

Created thread: 15060 at 00007FFDD8383AA0

Created thread: 15044 at 00007FFDD8383AA0

NtDeviceIoControlFile(FileHandle=0x48, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0xec5ebfefe0, IoControlCode=0x00500016, InputBuffer=0xec5ebfeff0, InputBufferLength=0x30, OutputBuffer=null, OutputBufferLength=0) => 0xc00700bb [187 'Не найдено указанное имя системного семафора.']

NtSetEvent(EventHandle=0x3c, PrevState=null) => 0

NtWaitForSingleObject(Handle=0xcc, Alertable=false, Timeout=null) => 0

NtTestAlert() => 0

NtReleaseSemaphore(SemaphoreHandle=0xc8, Count=1, PreviousCount=null) => 0

NtDeviceIoControlFile(FileHandle=0x64, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x73bcbff1a0, IoControlCode=0x00500016, InputBuffer=0x73bcbff1b0, InputBufferLength=0x30, OutputBuffer=null, OutputBufferLength=0) => 0xc00700bb [187 'Не найдено указанное имя системного семафора.']

Exception: 40010005 at 00007FFDD83D48E7 (first chance)

NtSetEvent(EventHandle=0x40, PrevState=null) => 0

NtTerminateProcess(ProcessHandle=0, ExitStatus=0xc000013a) => 0

Thread 9928 exit code: 3221225786

NtTestAlert() => 0

NtQueryWnfStateData(StateName=0xec5ebff550 [0xa3bc1c75], TypeId=null, ExplicitScope=null, ChangeStamp=0xec5ebfe494 [0x00018bbf], Buffer=0xec5ebfe4f0, BufferSize=0xec5ebfe490 [0x450]) => 0

NtSetInformationThread(ThreadHandle=-2, ThreadInformationClass=3 [ThreadBasePriority], ThreadInformation=0x73bcbffbd8, Length=4) => 0

NtQueryWnfStateData(StateName=0xec5ebff3d8 [0xa3bc7c75], TypeId=null, ExplicitScope=null, ChangeStamp=0xec5ebff468 [0], Buffer=0xec5ebff4c0, BufferSize=0xec5ebff3c0 [0]) => 0

Process 13808 exit code: 3221225786

Exception: 40010005 at 00007FFDD83D48E7 (first chance)

NtTerminateProcess(ProcessHandle=0, ExitStatus=0xc000013a) => 0

Thread 15012 exit code: 3221225786

NtClose(Handle=0xf4) => 0

NtClose(Handle=0xf8) => 0

NtClose(Handle=0xd8) => 0

NtClose(Handle=0xdc) => 0

NtClose(Handle=0xa4) => 0

NtClose(Handle=0xac) => 0

**Вывод**

В ходе выполнения работы была разработана программа, реализующая взаимодействие между родительским и дочерним процессами через pipe в Windows. Родительский процесс передает данные дочернему, который обрабатывает их и записывает результат в файл. Были использованы системные вызовы CreateProcess, CreatePipe, ReadFile, WriteFile, CloseHandle, WaitForSingleObject.

Возникли сложности с корректной обработкой имени файла, переданного дочернему процессу. Потребовалось уделить внимание обработке переносов строк (CRLF) в Windows, чтобы избежать ошибок при чтении имени файла.