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:

```
D:\code\osi>NtTrace D:\code\osi\parent.exe > trace.log
output.txt
1 2 3
4 5 6
0.1 0.2 0.3
```

(Скриншот консоли родительского процесса)

output.txt

```
1    6.00
2    15.00
3    0.60
4
```

(Содержимое файла output.txt)

nttrace:

NtReleaseSemaphore

NtOpenSemaphore

WaitForSingleObject

```
NtReleaseSemaphore(SemaphoreHandle=0xc8, Count=1, PreviousCount=null) => 0
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
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.']
NtCreateSemaphore(SemaphoreHandle=0xec5e9fe298 [0x68],
DesiredAccess=DELETE|READ_CONTROL|WRITE_DAC|WRITE_OWNER|SYNCHRONIZE|0x3,
ObjectAttributes=0x64:"Local\SM0:13808:304:WilStaging_02_p0", InitialCount=0x68ee1350,
MaxCount=0x68ee1350) => 0
NtCreateSemaphore(SemaphoreHandle=0xec5e9fe298 [0x6c],
DesiredAccess=DELETE|READ_CONTROL|WRITE_DAC|WRITE_OWNER|SYNCHRONIZE|0x3,
ObjectAttributes=0x64:"Local\SM0:13808:304:WilStaging_02_p0h", InitialCount=0x144,
MaxCount=0x144) => 0
NtOpenSemaphore(SemaphoreHandle=0xec5e9ff3b8 [0xa8],
DesiredAccess=DELETE|READ_CONTROL|WRITE_DAC|WRITE_OWNER|SYNCHRONIZE|0x3,
ObjectAttributes=0x64:"Local\ReadSemaphore") => 0
NtOpenSemaphore(SemaphoreHandle=0xec5e9ff3b8 [0xac],
DesiredAccess=DELETE|READ_CONTROL|WRITE_DAC|WRITE_OWNER|SYNCHRONIZE|0x3,
ObjectAttributes=0x64:"Local\WriteSemaphore") => 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.']
NtCreateSemaphore(SemaphoreHandle=0x73bc9fe1d8 [0x74],
DesiredAccess=DELETE|READ_CONTROL|WRITE_DAC|WRITE_OWNER|SYNCHRONIZE|0x3,
ObjectAttributes=0x70:"Local\SM0:17600:304:WilStaging_02_p0", InitialCount=0x09c35348,
MaxCount=0x09c35348) => 0
NtCreateSemaphore(SemaphoreHandle=0x73bc9fe1d8 [0x78],
DesiredAccess=DELETE|READ_CONTROL|WRITE_DAC|WRITE_OWNER|SYNCHRONIZE|0x3,
ObjectAttributes=0x70:"Local\SM0:17600:304:WilStaging_02_p0h", InitialCount=0x102,
MaxCount=0x102) => 0
```

Вывод

Лабораторная работа продемонстрировала эффективное использование разделяемой памяти и семафоров для межпроцессного взаимодействия в Windows. Правильная синхронизация с помощью двух семафоров и явная передача размера данных обеспечили надежный обмен информацией между родительским и дочерним процессами.

Сложности возникли с корректной реализацией условия завершения работы дочернего процесса и обеспечением безопасного доступа к разделяемой памяти.