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-215Б-23

Студент: Моисеев К.В.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 27.11.24

Москва, 2024

Постановка задачи

Вариант 3.

Составить и отладить программу на языке Си, демонстрирующую работу с процессами и их взаимодействие через разделяемую память. Программа должна создавать один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files). Необходимо реализовать обработку системных ошибок, возникающих в процессе выполнения программы.

В файле записаны команды вида: «число число число». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип float. Количество чисел может быть произвольным.

- # Общий метод и алгоритм решения

Использованные системные вызовы:

- **HANDLE CreateFileMapping(HANDLE hFile, LPSECURITY_ATTRIBUTES lpAttributes, DWORD flProtect, DWORD dwMaximumSizeHigh, DWORD dwMaximumSizeLow, LPCWSTR lpName)** — создание объекта отображаемого файла.
- **LPVOID MapViewOfFile(HANDLE hFileMappingObject, DWORD dwDesiredAccess, DWORD dwFileOffsetHigh, DWORD dwFileOffsetLow, SIZE_T dwNumberOfBytesToMap)** — отображение файла в память процесса.
- **BOOL UnmapViewOfFile(LPCVOID lpBaseAddress)** — снятие отображения файла из памяти.
- **BOOL CreateProcess(LPCWSTR lpApplicationName, LPWSTR lpCommandLine, LPSECURITY_ATTRIBUTES lpProcessAttributes, LPSECURITY_ATTRIBUTES lpThreadAttributes, BOOL bInheritHandles, DWORD dwCreationFlags, LPVOID lpEnvironment, LPCWSTR lpCurrentDirectory, LPSTARTUPINFOW lpStartupInfo, LPPROCESS_INFORMATION lpProcessInformation)** — создание дочернего процесса.
- **DWORD WaitForSingleObject(HANDLE hHandle, DWORD dwMilliseconds)** — ожидание завершения дочернего процесса.
- **HANDLE CreateFile(LPCWSTR lpFileName, DWORD dwDesiredAccess, DWORD dwShareMode, LPSECURITY_ATTRIBUTES lpSecurityAttributes, DWORD dwCreationDisposition, DWORD dwFlagsAndAttributes, HANDLE hTemplateFile)** — открытие файла для чтения.
- **BOOL ReadFile(HANDLE hFile, LPVOID lpBuffer, DWORD nNumberOfBytesToRead, LPDWORD lpNumberOfBytesRead, LPOVERLAPPED lpOverlapped)** — чтение данных из файла.
- **BOOL WriteConsole(HANDLE hConsoleOutput, const VOID* lpBuffer, DWORD nNumberOfCharsToWrite, LPDWORD lpNumberOfCharsWritten, LPVOID lpReserved)** — вывод данных на консоль.

Алгоритм работы программы

Общая идея

Программа демонстрирует взаимодействие между родительским и дочерним процессами через механизм отображаемой памяти (memory-mapped files) в операционной системе Windows. Родительский процесс создаёт дочерний процесс, передаёт ему имя файла для обработки и получает от него результат вычислений. Обмен данными осуществляется через общую область памяти, доступную обоим процессам.

Алгоритм

Родительский процесс начинает выполнение, запрашивая у пользователя имя файла. После этого он создаёт объект отображаемого файла с помощью системного вызова `CreateFileMapping`, выделяя область памяти для обмена данными. Затем он отображает этот файл в виртуальное адресное пространство с использованием функции `MapViewOfFile`. В полученную память родительский процесс записывает имя файла, которое затем будет использовано дочерним процессом.

Далее родительский процесс создаёт дочерний процесс, используя системный вызов `CreateProcess`. Дочернему процессу передаётся доступ к отображаемому файлу, и он начинает выполнение. В это время родительский процесс приостанавливает выполнение и ожидает завершения дочернего процесса с помощью `WaitForSingleObject`.

Дочерний процесс, после запуска, открывает объект отображаемого файла через `OpenFileMapping` и получает указатель на общую память с помощью `MapViewOfFile`. Из этой памяти он считывает имя файла, открывает указанный файл для чтения и начинает обработку его содержимого. Дочерний процесс читает данные из файла, интерпретирует их как числа с плавающей запятой и суммирует их. После завершения вычислений результат записывается обратно в общую память.

После завершения дочернего процесса родительский процесс возобновляет выполнение. Он читает результат вычислений из памяти и выводит его на консоль с использованием `WriteConsole`. Завершающим этапом является освобождение всех ресурсов: снимается отображение памяти, закрываются дескрипторы объектов памяти и завершается выполнение программы.

Программа также включает обработку ошибок на всех этапах выполнения. В случае возникновения ошибки выводится соответствующее сообщение, и программа корректно освобождает ресурсы перед завершением.

Код программы

parent.c

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

#define MAPPING_NAME "Local\\SharedMemoryExample"
#define BUFFER_SIZE 256

// Функция для вывода текста на консоль с использованием WriteConsole
void printMessage(HANDLE hConsole, const char* message) {
    DWORD written;
    WriteConsole(hConsole, message, strlen(message), &written, NULL);
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        printMessage(GetStdHandle(STD_OUTPUT_HANDLE), "Usage: parent.exe <filename>\n");
        return 1;
    }

    const char* filename = argv[1];

    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE); // Получаем дескриптор консоли
    if (hConsole == INVALID_HANDLE_VALUE) {
        fprintf(stderr, "Error getting console handle (%d).\n", GetLastError());
        return 1;
    }

    // Создание отображаемого файла
    HANDLE hMapFile = CreateFileMapping(
        INVALID_HANDLE_VALUE,
        NULL,
        PAGE_READWRITE,
        0, BUFFER_SIZE,
        MAPPING_NAME
    );

    if (hMapFile == NULL) {
        DWORD error = GetLastError();
        char errorMessage[256];
        sprintf(errorMessage, "Could not create file mapping object. Error code: %d\n",
error);
        printMessage(hConsole, errorMessage);
        return 1;
    }

    // Получаем указатель на отображаемую память
    LPVOID pBuf = MapViewOfFile(hMapFile, FILE_MAP_ALL_ACCESS, 0, 0, BUFFER_SIZE);
    if (pBuf == NULL) {
```

```
        printMessage(hConsole, "Could not map view of file.\n");
        CloseHandle(hMapFile);
        return 1;
    }

    // Записываем имя файла в память
    sprintf((char*)pBuf, "%s", filename);

    // Создание дочернего процесса
    PROCESS_INFORMATION pi;
    STARTUPINFO si = { sizeof(si) };

    if (!CreateProcess(NULL, "child.exe", NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)) {
        printMessage(hConsole, "CreateProcess failed.\n");
        UnmapViewOfFile(pBuf);
        CloseHandle(hMapFile);
        return 1;
    }

    // Ожидание завершения дочернего процесса
    WaitForSingleObject(pi.hProcess, INFINITE);

    // Читаем результат из памяти и выводим его
    printMessage(hConsole, "Result from child process: ");
    printMessage(hConsole, (char*)pBuf);
    printMessage(hConsole, "\n");

    // Освобождение ресурсов
    UnmapViewOfFile(pBuf);
    CloseHandle(hMapFile);
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);

    return 0;
}
```

child.c

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

#define MAPPING_NAME "Local\\SharedMemoryExample"
#define BUFFER_SIZE 256

// Функция для вывода текста на консоль с использованием WriteConsole
void printMessage(HANDLE hConsole, const char* message) {
    DWORD written;
    WriteConsole(hConsole, message, strlen(message), &written, NULL);
}

int main() {
```

```
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    if (hConsole == INVALID_HANDLE_VALUE) {
        return 1;
    }

    // Открываем отображаемый файл
    HANDLE hMapFile = OpenFileMapping(FILE_MAP_ALL_ACCESS, FALSE, MAPPING_NAME);
    if (hMapFile == NULL) {
        DWORD error = GetLastError();
        char errorMessage[256];
        sprintf(errorMessage, "Could not open file mapping object. Error code: %d\n",
error);
        printMessage(hConsole, errorMessage);
        return 1;
    }

    // Получаем указатель на память
    LPVOID pBuf = MapViewOfFile(hMapFile, FILE_MAP_ALL_ACCESS, 0, 0, BUFFER_SIZE);
    if (pBuf == NULL) {
        printMessage(hConsole, "Could not map view of file.\n");
        CloseHandle(hMapFile);
        return 1;
    }

    // Чтение имени файла из памяти
    char filename[BUFFER_SIZE];
    strcpy(filename, (char*)pBuf);

    // Открытие файла через WinAPI
    HANDLE hFile = CreateFile(
        filename,               // Имя файла
        GENERIC_READ,           // Открыть для чтения
        FILE_SHARE_READ,        // Разрешить совместное чтение
        NULL,                   // Атрибуты безопасности
        OPEN_EXISTING,          // Открыть существующий файл
        FILE_ATTRIBUTE_NORMAL,  // Обычный файл
        NULL                    // Нет шаблона файла
    );

    if (hFile == INVALID_HANDLE_VALUE) {
        sprintf((char*)pBuf, "Failed to open file.\n");
        UnmapViewOfFile(pBuf);
        CloseHandle(hMapFile);
        return 1;
    }

    // Чтение содержимого файла и подсчёт суммы
    char buffer[BUFFER_SIZE];
    DWORD bytesRead;
    float sum = 0.0;

    while (ReadFile(hFile, buffer, sizeof(buffer) - 1, &bytesRead, NULL) && bytesRead > 0)
{
        buffer[bytesRead] = '\0';  // Завершаем строку
```

```
        char* token = strtok(buffer, " \n");
        while (token != NULL) {
            sum += atof(token);  // Преобразуем и суммируем числа
            token = strtok(NULL, " \n");
        }
    }

    // Закрываем файл
    CloseHandle(hFile);

    // Запись результата обратно в отображаемую память
    sprintf((char*)pBuf, "Sum: %.2f", sum);

    // Освобождение ресурсов
    UnmapViewOfFile(pBuf);
    CloseHandle(hMapFile);

    return 0;
}
```

Протокол работы программы

Тестирование:

PS C:\Users\klosh\Desktop\OS3\build> .\parent.exe ..\test\test.txt

Result from child process: Sum: 53.20

NtTrace:

C:\Users\klosh\Desktop\NtTrace-main\NtTrace-main>NtTrace -filter NtCreateSection,NtMapViewOfSection,NtUnmapViewOfSection,NtOpenFile C:\Users\klosh\Desktop\OS3\build\parent.exe C:\Users\klosh\Desktop\OS3\test\test.txt
Process 11108 starting at 00000000004014E0 with command line: "C:\Users\klosh\Desktop\OS3\build\parent.exe C:\Users\klosh\Desktop\OS3\test\test.txt"
C:\Users\klosh\Desktop\OS3\build\parent.exe
Loaded DLL at 00007FFD423F0000 ntdll.dll
NtOpenFile(FileHandle=0x61f398 [0x54], DesiredAccess=SYNCHRONIZE|0x20, ObjectAttributes="\??\C:\Users\klosh\Desktop\NtTrace-main\NtTrace-main\", IoStatusBlock=0x61f308 [0/1], ShareAccess=3, OpenOptions=0x21) => 0
Loaded DLL at 00007FFD40C40000 C:\WINDOWS\System32\KERNEL32.DLL
NtMapViewOfSection(SectionHandle=0x58, ProcessHandle=-1, BaseAddress=0x674b80 [0x00007ffd40c40000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x674ad8 [0x000c4000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0
Loaded DLL at 00007FFD3F6F0000 C:\WINDOWS\System32\KERNELBASE.dll

NtMapViewOfSection(SectionHandle=0x68, ProcessHandle=-1, BaseAddress=0x6752c0 [0x00007ffd3f6f0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x675218 [0x003b9000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0

NtCreateSection(SectionHandle=0x61e900 [0x6c], DesiredAccess=DELETE|READ_CONTROL|WRITE_DAC|WRITE_OWNER|0x1f, ObjectAttributes=null, SectionSize=0x61e8f0 [65536], Protect=4, Attributes=0x08000000, FileHandle=0) => 0

NtMapViewOfSection(SectionHandle=0x58, ProcessHandle=-1, BaseAddress=0x61e8f8 [0x00007ff4fdec0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x61e908 [0x00100000], InheritDisposition=2 [ViewUnmap], AllocationType=0x00500000, Protect=2) => 0

NtMapViewOfSection(SectionHandle=0x98, ProcessHandle=-1, BaseAddress=0x61e890 [0x001d0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x61e8a0 [0x3000], InheritDisposition=2 [ViewUnmap], AllocationType=0, Protect=2) => 0

NtMapViewOfSection(SectionHandle=0x9c, ProcessHandle=-1, BaseAddress=0x61e890 [0x001e0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x61e8a0 [0x3000], InheritDisposition=2 [ViewUnmap], AllocationType=0, Protect=2) => 0

NtMapViewOfSection(SectionHandle=0xa0, ProcessHandle=-1, BaseAddress=0x61e890 [0x001f0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x61e8a0 [0x1000], InheritDisposition=2 [ViewUnmap], AllocationType=0, Protect=2) => 0

Loaded DLL at 00007FFD409C0000 C:\WINDOWS\System32\msvcrt.dll

NtMapViewOfSection(SectionHandle=0xb8, ProcessHandle=-1, BaseAddress=0x677f30 [0x00007ffd409c0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x675078 [0x000a7000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0

Initial breakpoint

NtCreateSection(SectionHandle=0x61fb40 [0xbc], DesiredAccess=DELETE|READ_CONTROL|WRITE_DAC|WRITE_OWNER|0x7, ObjectAttributes=0x80:"Local\SharedMemoryExample", SectionSize=0x61fb30 [256], Protect=4, Attributes=0x08000000, FileHandle=0) => 0

NtCreateSection используется для создания секции, которая может быть сопоставлена с памятью процесса. Первый вызов, который создает секцию для общей памяти

NtMapViewOfSection(SectionHandle=0xbc, ProcessHandle=-1, BaseAddress=0x61fb98 [0x00620000], ZeroBits=0, CommitSize=0, SectionOffset=0x61fb90 [0], ViewSize=0x61fba0 [0x1000], InheritDisposition=1 [ViewShare], AllocationType=0, Protect=4) => 0

В Windows аналог mmap — это NtMapViewOfSection. Этот вызов используется для сопоставления представления секции с адресным пространством процесса.

Loaded DLL at 00007FFD41E90000 C:\WINDOWS\System32\sechost.dll

NtMapViewOfSection(SectionHandle=0xc0, ProcessHandle=-1, BaseAddress=0x679000 [0x00007ffd41e90000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x675078 [0x000a7000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0

Loaded DLL at 00007FFD3FAB0000 C:\WINDOWS\System32\bcrypt.dll
NtMapViewOfSection(SectionHandle=0xc4, ProcessHandle=-1, BaseAddress=0x679490 [0x00007ffd3fab0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x6793f8 [0x00028000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0
NtOpenFile(FileHandle=0x7ffd3fad27a0 [0xd4], DesiredAccess=SYNCHRONIZE|0x3, ObjectAttributes="\Device\KsecDD", IoStatusBlock=0x61cf10 [0/0], ShareAccess=7, OpenOptions=0x20) => 0
Process 18948 starting at 00000000004014E0 with command line: "child.exe" C:\Users\klosh\Desktop\OS3\build\child.exe
Loaded DLL at 00007FFD423F0000 ntdll.dll
NtOpenFile(FileHandle=0x61f398 [0x5c], DesiredAccess=SYNCHRONIZE|0x20, ObjectAttributes="\??\C:\Users\klosh\Desktop\NtTrace-main\NtTrace-main\", IoStatusBlock=0x61f308 [0/1], ShareAccess=3, OpenOptions=0x21) => 0
Loaded DLL at 00007FFD40C40000 C:\WINDOWS\System32\KERNEL32.DLL
NtMapViewOfSection(SectionHandle=0x60, ProcessHandle=-1, BaseAddress=0x744af0 [0x00007ffd40c40000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x744a48 [0x000c4000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0
Loaded DLL at 00007FFD3F6F0000 C:\WINDOWS\System32\KERNELBASE.dll
NtMapViewOfSection(SectionHandle=0x64, ProcessHandle=-1, BaseAddress=0x745230 [0x00007ffd3f6f0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x745188 [0x003b9000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0
NtCreateSection(SectionHandle=0x61e900 [0x64], DesiredAccess=DELETE|READ_CONTROL|WRITE_DAC|WRITE_OWNER|0x1f, ObjectAttributes=null, SectionSize=0x61e8f0 [65536], Protect=4, Attributes=0x08000000, FileHandle=0) => 0
NtMapViewOfSection(SectionHandle=0x60, ProcessHandle=-1, BaseAddress=0x61e8f8 [0x00007ff4fdec0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x61e908 [0x00100000], InheritDisposition=2 [ViewUnmap], AllocationType=0x00500000, Protect=2) => 0
NtMapViewOfSection(SectionHandle=0x94, ProcessHandle=-1, BaseAddress=0x61e890 [0x001d0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x61e8a0 [0x3000], InheritDisposition=2 [ViewUnmap], AllocationType=0, Protect=2) => 0
NtMapViewOfSection(SectionHandle=0x98, ProcessHandle=-1, BaseAddress=0x61e890 [0x001e0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x61e8a0 [0x3000], InheritDisposition=2 [ViewUnmap], AllocationType=0, Protect=2) => 0
NtMapViewOfSection(SectionHandle=0x9c, ProcessHandle=-1, BaseAddress=0x61e890 [0x001f0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x61e8a0 [0x1000], InheritDisposition=2 [ViewUnmap], AllocationType=0, Protect=2) => 0
Loaded DLL at 00007FFD409C0000 C:\WINDOWS\System32\msvcrt.dll

NtMapViewOfSection(SectionHandle=0xb4, ProcessHandle=-1, BaseAddress=0x747e50 [0x00007ffd409c0000], ZeroBits=0, CommitSize=0, SectionOffset=null, ViewSize=0x744fe8 [0x000a7000], InheritDisposition=1 [ViewShare], AllocationType=0x00800000, Protect=0x80) => 0

Initial breakpoint

NtMapViewOfSection(SectionHandle=0x60, ProcessHandle=-1, BaseAddress=0x61fa18 [0x00620000], ZeroBits=0, CommitSize=0, SectionOffset=0x61fa10 [0], ViewSize=0x61fa20 [0x1000], InheritDisposition=1 [ViewShare], AllocationType=0, Protect=4) => 0

Этот вызов отображает ту же секцию в адресное пространство дочернего процесса

NtUnmapViewOfSectionEx(ProcessHandle=-1, BaseAddress=0x620000, Flags=0) => 0

Process 18948 exit code: 0

дочерний процесс удаляет отображение секции из своего адресного пространства

Result from child process: Sum: 53.20

NtUnmapViewOfSectionEx(ProcessHandle=-1, BaseAddress=0x620000, Flags=0) => 0

Process 11108 exit code: 0

родительский процесс также удаляет отображение секции из своего адресного пространства

Вывод

Лабораторная работа оказалась очень интересной и познавательной. Было интересно сравнить разные подходы к передаче данных между процессами, а именно memory mapping и pipe. Оба способа эффективны по-своему и нужно с умом применять их.