ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Егоров В.Ю.

Системное программирование в ОС Windows на основе Win32 API

Методические указания
к выполнению лабораторных работ
по курсам «Архитектура операционных систем»,
«Основы операционных систем»

Пенза

Даны указания по выполнению лабораторных работ и задания к ним. Лабораторные работы предназначены для изучения прикладного программного интерфейса Win32 API операционной системы Windows и для постановки лабораторного практикума по дисциплинам «Архитектура операционных систем» и «Основы операционных систем».

Общие методические указания

Работы выполняются в среде семейства операционных систем Windows. Для их выполнения необходимо использовать MSDN Subscriptions Library, конспект курса лекций, а также литературу, приведенную в конце пособия.

Написание программ следует выполнять в среде программирования Microsoft Visual C. Отладку необходимо производить встроенным отладчиком среды программирования.

При разработке программ основное внимание следует уделять не усложнению функций пользовательского интерфейса, а продуманной структурной организации внутренней программной логики.

Требования к оформлению работ

По каждой лабораторной работе необходимо составить отчет, который должен содержать:

- титульный лист;
- название и цель работы;
- лабораторное задание;
- описание данных и при необходимости описание структуры программы;
- текст программы;
- результаты выполнения программы;
- выводы по результатам выполнения работы.

Отчет должен быть представлен в виде файла в формате MS Word.

Лабораторная работа №1
Порождение и завершение процессов и нитей

Цель работы

Изучение методов и средств порождения процессов и нитей. Изучение способов синхронизации процессов и нитей через ожидание окончания их выполнения.

Список используемых системных вызовов

`CloseHandle, CreateProcess, CreateThread, ExitProcess, ExitThread, GetCommandLine, GetCurrentProcess, GetCurrentProcessId, GetCurrentThread, GetCurrentThreadId, GetExitCodeProcess, OpenProcess, OpenThread, ResumeThread, SuspendThread, Sleep, ThreadProc, TerminateProcess, TerminateThread, WaitForSingleObject, WaitForMultipleObjects.`

Методические указания

В лабораторной работе все процессы консольные. В том случае, если в задании указано создание нескольких процессов, то это должны быть процессы одной и той же программы.

Эмуляцию работы процесса следует выполнять с помощью функции `Sleep`.

Для использования функций Win32 API достаточно подключить заголовочный файл windows.h

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

Для порождения нового процесса используется функция `CreateProcess`. При этом порождается процесс и одна начальная нить. После создания процесса параметр *`lpProcessInformation`* (адрес структуры `PROCESS_INFORMATION`) заполняется значениями идентификаторов и дескрипторов процесса и начальной нити. Дескрипторы следует закрыть сразу после того, как они становятся не нужны, с помощью функции `CloseHandle`.

Процесс может создавать дополнительные нити с помощью функции `CreateThread`. Основная функция порожденной нити вызовется средствами операционной системы. Такие функции называются callback-функциями.

Нить процесса можно породить в приостановленном состоянии. Для этого в параметре *`dwCreationFlags`* следует установить бит CREATE_SUSPENDED. Останов и возобновление работы нити можно

также производить с помощью функций SuspendThread, ResumeThread.

Процесс завершается в следующих случаях:

- Вызовом функции ExitProcess, либо выходом из функции WinMain или main;
- Вызовом функции ExitThread для каждой нити процесса, либо выходом из основной функции нити;
- Уничтожением всех нитей процесса с помощью функции TerminateThread (не рекомендуется);
- Уничтожением процесса с помощью функции TerminateProcess.

Нити и процессы можно синхронизировать, используя группу WaitFor-функций (WaitForSingleObject, WaitForMultipleObjects и др.). По завершению процесса или нити эти функции возвращают значение WAIT_OBJECT_0.

Задания

1. Породить цепочку из 10 процессов. Предыдущие процессы в цепочке должны завершаться по завершении любого из следующих.
2. Породить несколько процессов из одного. Все процессы должны завершаться по завершению первого.
3. Породить два процесса. По завершению одного (любого) из них второй должен сразу же порождать еще один процесс. Предусмотреть путь корректного завершения работы обоих процессов.
4. В одном процессе необходимо выводить двумя нитями текст на экран пословно.
5. В одном процессе необходимо выводить двумя нитями текст на экран посимвольно.
6. Написать менеджер задач, порождающий и уничтожающий процессы по команде пользователя.
7. Разработать индикатор, показывающий, что выбранные процессы продолжают выполняться или завершены.
8. В процессе одновременно порождается 10 нитей. Однако работать одновременно могут только 3 из них. Остальные должны быть остановлены, и должны поочерёдно приступать к работе только после завершения одной из активных нитей.
9. В процессе одновременно порождается 10 нитей. Однако работать одновременно могут только 2 из них. Остальные должны быть

остановлены, и должны поочерёдно приступать к работе только после завершения обеих активных нитей.

10. Разработать программу, делегирующую выполнение каждого действия (вывод на экран) вновь создаваемой нити.
11. Разработать менеджер процессов, уничтожающий процессы, выполняющиеся дольше заданного срока.
12. Разработать программу, останавливающую и возобновляющую работу подчиненных нитей по очереди через равные интервалы времени.

Контрольные вопросы

1. Что такое идентификатор процесса, описатель (дескриптор) процесса?
2. Как соотносятся процесс и нить?
3. Перечислите все способы завершения процесса.
4. Как работают функции WaitForSingleObject и WaitForMultipleObjects?

Лабораторная работа №2
Синхронизация нитей с использованием событий и Interlocked-функций

Цель работы

Изучение способа организации критических секций с использованием группы Interlocked-функций и структур CRITICAL_SECTION. Изучение способа синхронизации процессов и нитей с использованием событий (Events).

Список используемых системных вызовов

`CreateEvent, SetEvent, ResetEvent, PulseEvent, CloseHandle, InterlockedExchange, InterlockedIncrement, InterlockedDecrement, InterlockedCompareExchange, InterlockedExchangeAdd, InitializeCriticalSection, EnterCriticalSection, TryEnterCriticalSection, LeaveCriticalSection, DeleteCriticalSection, WaitForSingleObject, WaitForMultipleObjects.`

Методические указания

Для данной работы критическим ресурсом будет выступать окно консольного приложения.

Группа Interlocked-функций позволяет производить атомарные операции над двойным словом данных. С помощью функции InterlockedExchange организуются критические секции, как показано ниже:

```
// Инициализация
static LONG volatile Status = 0;
...
// Вход
while (1 == InterlockedExchange(&Status, 1))
{
    Sleep(20);
}
// Критическая секция
Status = 0;
// Конец критической секции
```

Критическую секцию можно организовать и с использованием структуры `CRITICAL_SECTION`. Вариант критической секции показан ниже:

```
// Инициализация
CRITICAL_SECTION cs;
InitializeCriticalSection(&cs);
...
// Вход
EnterCriticalSection(&cs);
// Критическая секция
LeaveCriticalSection(&cs);
// Конец критической секции
...
// Деинициализация
DeleteCriticalSection(&cs);
```

Синхронизацию нитей можно осуществлять с помощью событий (Events). Для создания объекта события используется функция `CreateEvent`. События бывают с ручным и автоматическим сбросом. События с автоматическим сбросом переходят в несигнальное состояние сразу после возврата из ожидающей WaitFor-функции. События с ручным сбросом необходимо сбрасывать с помощью функции `ResetEvent`. Перевод события в сигнальное состояние осуществляется с помощью функции `SetEvent`. После завершения работы событием необходимо закрыть дескриптор события с помощью функции `CloseHandle`.

Задания

1. Имеется 9 нитей. Каждая нить в цикле выводит на экран свой уникальный символ. Вывод символа на экран производится из критической секции на основе Interlocked-функций.
2. То же, что 1, но для структур `CRITICAL_SECTION`.

3. То же, что 1, но для событий с автоматическим сбросом.
4. Организовать циклическую цепочку срабатывания нитей с использованием Interlocked-функций.
5. То же, что 4, но для структур `CRITICAL_SECTION`.
6. То же, что 4, но для событий.
7. Имеется две нити, выводящих на экран некий текст пословно. Вывод каждого слова должен производиться после занятия критической секции. Организацию критических секций производить с использованием Interlocked-функций.
8. То же, что и 7, но для `CRITICAL_SECTION`.
9. То же, что и 7, но для событий.
10. Имеется 3 нити. Две из них производят инкремент переменной на 1. Одна – декремент на 2. Работа с переменной должна осуществляться с помощью критической секции на основе Interlocked-функций. Вывести на экран текущее значение переменной и крайние значения.
11. То же, что и 10, но для `CRITICAL_SECTION`.
12. То же, что и 10, но для событий с автоматическим сбросом.

Контрольные вопросы

1. Дайте определения понятиям ресурс, критический ресурс, критическая секция.
2. Можно ли синхронизировать процессы с помощью группы Interlocked-функций, если переменная находится в куче процесса или в стеке?
3. Что означает ключевое слово volatile?
4. Какие типы событий (event) вы знаете, и чем они отличаются?

Лабораторная работа №3
Работа с разделяемой памятью

Цель работы

Изучение способов работы с разделяемой памятью на основе файлов, проецируемых в память. Изучение способов синхронизации процессов и нитей с использованием мьютексов (mutex) и семафоров.

Список используемых системных вызовов

`CreateFileMapping, OpenFileMapping, MapViewOfFile, UnmapViewOfFile, CloseHandle, CreateMutex, OpenMutex, CreateSemaphore, OpenSemaphore, ReleaseMutex,`

`ReleaseSemaphore, WaitForSingleObject, WaitForMultipleObjects.`

Методические указания

Для получения разделяемой памяти необходимо создать файл, проецируемый в память не на диске, а в свопинге системы, с помощью функции `CreateFileMapping`. Это достигается заданием в первом параметре *`hFile`* значения `INVALID_HANDLE_VALUE`. Проекция файла на адресное пространство процесса задается с помощью функций `MapViewOfFile` или `MapViewOfFileEx`. По окончании работы следует вызвать функции `UnmapViewOfFile` и `CloseHandle` для дескриптора спроецированного в память файла.

Мьютексы и семафоры позволяют синхронизировать нити в разных процессах (как и события), поскольку являются объектами операционной системы. Мьютекс, по сути, является упрощенной формой семафора. Принципиальным отличием мьютекса от семафора является возможность повторного занятия мьютекса одной и той же нитью.

Мьютексы и семафоры порождаются с помощью функций `CreateMutex` и `CreateSemaphore`. Для занятия мьютекса или семафора используется любая из группы WaitFor-функций. Для освобождения объектов используются соответственно функции `ReleaseMutex` и `ReleaseSemaphore`.

Задания

1. Имеется 3 процесса. Два из них производят инкремент переменной в разделяемой памяти на 1. Одна – декремент на 2. Работа с переменной должна осуществляться с помощью критической секции на основе мьютексов. Вывести на экран текущее значение переменной и крайние значения.
2. То же, что и 1, но для семафоров.
3. Разработать программу, показывающую текущее количество запущенных экземпляров этой программы.
4. Разработать программу, передающую текстовые сообщения между двумя запущенными экземплярами программы в любом направлении с помощью буфера, расположенного в разделяемой памяти. Синхронизация с использованием семафоров.
5. То же, что и 3, но для мьютексов.
6. Разработать программу, показывающую количество запусков этой программы, пока хотя бы один экземпляр программы находится в памяти.

7. Имеется 2 клиента и 1 сервер в виде отдельных процессов. Необходимо осуществлять передачу данных от клиентов к серверу через один и тот же буфер в разделяемой памяти. Синхронизация с использованием семафоров.
8. То же, что и 7, но для мьютексов.
9. Разработать программу, синхронизирующую информацию на консольных экранах для двух своих запусков с помощью буфера в разделяемой памяти. Синхронизация с использованием семафоров.
10. То же, что и 9, но для мьютексов.
11. Разработать систему из двух программ. Первая программа предназначена для ввода системных команд с командной строки. Вторая программа демонстрирует результаты выполнения команд. Синхронизация с использованием семафоров.
12. То же, что и 11, но для мьютексов.

Контрольные вопросы

1. Что такое разделяемая память?
2. Чем мьютекс отличается от семафора?
3. Как с помощью семафоров обеспечить множественный доступ нитей к ресурсу?
4. Почему мьютексы облегчают написание рекурсивных функций с доступом к критическому ресурсу по сравнению с семафорами?

Лабораторная работа №4
Передача данных с использованием почтовых ящиков и таймеров ожидания

Цель работы

Изучение построения клиент-серверного взаимодействия процессов с использованием почтовых ящиков. Изучение особенностей работы нитей при использовании таймеров ожидания.

Список используемых системных вызовов

`CreateMailslot, CreateFile, GetMailslotInfo, SetMailslotInfo, ReadFile, ReadFileEx, WriteFile, WriteFileEx, CreateWaitableTimer, OpenWaitableTimer, SetWaitableTimer, TimerAPCProc, CancelWaitableTimer, SleepEx, WaitForSingleObjectEx, WaitForMultipleObjectsEx.`

Методические указания

Почтовые ящики позволяют процессам обмениваться данными в пределах локальной сети Microsoft. Для создания почтового ящика на стороне сервера используется функция `CreateMailslot`. На стороне клиента используется функция работы с файлами `CreateFile`. Для записи и чтения из почтового ящика используются функции `ReadFile` и `WriteFile` соответственно. (Для работы почтового ящика совместно с ждущим таймером следует использовать функции `ReadFileEx` и `WriteFileEx`). Для получения информации о наличии данных в почтовом ящике на стороне сервера используется функция `GetMailslotInfo`. Следует обратить внимание, что в программе необходимо проверять ящик на наличие сообщения следующим способом:

```
if (dwNextSize != MAILSLOT_NO_MESSAGE)
{
    // Данные есть
}
```

Таймеры ожидания используются для выполнения определенных действий в заданное время или через заданный интервал времени. Таймер создается с помощью функции `CreateWaitableTimer`. Ожидание срабатывания таймера можно производить с помощью WaitFor-функций. Другой способ – задать специальную callback-функцию, которая выполнится сразу после перехода таймера в сигнальный режим. При этом нить должна находится в особом состоянии, называемом «тревожным» (alertable). Нить находится в этом состоянии при выполнении функций ожидания с суффиксом Ex: `ReadFileEx`, `WriteFileEx`, `SleepEx`, `WaitForSingleObjectEx`, `WaitForMultipleObjectsEx`, и других.

Таймеры ожидания, как и мьютексы, бывают с ручным сбросом и синхронизирующие. Синхронизирующие таймеры используются для временной синхронизации процессов и нитей или для однократного вызова callback-функции. Таймеры ожидания также используются для периодического вызова callback-функции.

С помощью таймеров ожидания можно легко программировать максимальное время ожидания поступления данных в почтовый ящик.

Задания

1. Разработать программу периодической проверки свободного места на жестком диске (функция `GetDiskFreeSpace`) и сбора данной информации на едином сервере с использованием почтового ящика.
2. Разработать программу получения статистики по использованию памяти процессом (функция `GetProcessMemoryInfo`) и сбора

данной информации на едином сервере с использованием почтового ящика.

3. Разработать программу передачи сообщений между двумя пользователями с использованием почтовых ящиков.
4. Разработать программу автоматического уведомления пользователя о завершении текущего занятия. Причем действие установления факта завершения занятия и уведомление пользователя об этом должны находиться в разных процессах.
5. Разработать программу-будильник (функция `MessageBeep`). Установка будильника должна осуществляться с другого компьютера с использованием почтового ящика.
6. Разработать программу измерения загрузки центрального процессора компьютера в процентном отношении для параллельно выполняющихся нитей с помощью ждущего таймера и функции `Sleep`.
7. Разработать программу, позволяющую исследовать структуру каталога “c:\” на удаленном компьютере с помощью функций `FindFirstFile`, `FindNextFile`.
8. Разработать программу вычисления разницы в значениях текущего времени на двух компьютерах с помощью функции `GetLocalTime`.
9. Разработать программу передачи сообщения по запросу с одного компьютера на другой.
10. Разработать программу, передающую на сервер данные о текущем пользователе системы (функция `GetUserName`).
11. Разработать программу, передающую на сервер данные о версии операционной системы (функция `GetVersionEx`).
12. Разработать программу, передающую на сервер данные о каталоге операционной системы и каталоге временных файлов пользователя (функции `GetWindowsDirectory`, `GetTempPath`).

Контрольные вопросы

1. Чем таймер ожидания с ручным сбросом отличается от синхронизирующего таймера ожидания?
2. Как таймер ожидания можно использовать для пробуждения операционной системы?
3. Что означает «тревожное» состояние нити? Какие функции переводят нить в «тревожное» состояние?
4. Какова область действия почтовых ящиков?

5. Могут ли существовать несколько почтовых ящиков с одинаковыми именами?

Лабораторная работа №5
Динамически загружаемые библиотеки

Цель работы

Изучить связывание процесса с динамически загружаемыми библиотеками на этапе загрузки и на этапе выполнения.

Список используемых системных вызовов

`LoadLibrary`, `FreeLibrary`, `FreeLibraryAndExitThread`, `GetModuleHandle`, `GetProcAddress`, `DllMain`.

Методические указания

При выполнении данной лабораторной работы решение (solution) в среде MS Visual Studio должно одновременно содержать в себе проекты программ и динамически загружаемых библиотек.

Динамически загружаемые библиотеки (DLL) связываются с программами и между собой с помощью специальных таблиц экспорта и импорта, находящихся внутри файлов библиотек и программ. Просмотреть содержимое этих таблиц можно с помощью утилиты dumpbin.exe. Ниже приведены примеры использования этой утилиты для библиотеки kernel32.dll:

```
dumpbin.exe kernel32.dll \exports > kernel32.exports
dumpbin.exe kernel32.dll \imports > kernel32.imports
```

Таблица импорта содержит запрашиваемые ресурсы, таблица экспорта – предоставляемые ресурсы библиотеки.

Ещё один вариант просмотра таблиц экспорта и импорта – использование утилиты depends.exe.

Для создания динамически загружаемой библиотеки необходимо указать при создании тип проекта – DLL. У DLL отсутствует функция `WinMain`. Вместо нее используется функция `DllMain`, вызываемая в четырех случаях:

1. при загрузке библиотеки процессом (отображении в виртуальное адресное пространство процесса);
2. при создании новой нити;
3. при завершении созданной нити;
4. при завершении процесса или при выгрузке библиотеки.

Любую функцию, переменную или класс библиотеки можно сделать экспортируемыми, т.е. подключаемыми извне с помощью таблицы экспорта. Чтобы адрес ресурса был помещен в таблицу экспорта, необходимо указать непосредственно перед определением ресурса ключевые слова `__declspec(dllexport)`. Примеры экспортирования ресурсов:

```
__declspec(dllexport) int i;
__declspec(dllexport) void func();
class __declspec(dllexport) Class;
```

DLL могут загружаться процессом при старте программы (динамическое связывание) или явно с помощью функции `LoadLibrary`. После компиляции библиотеки компоновщик создает два файла для каждой динамически загружаемой библиотеки – с расширениями .lib и .dll. Файл с расширением .lib необходимо подключить при компоновке программы, использующей динамическое связывание. Это делается в опциях проекта для компоновщика «additional dependencies». Программа может импортировать ресурсы из DLL с помощью ключевых слов `__declspec(dllimport)`. Примеры импортирования ресурсов:

```
__declspec( dllimport ) int i;
__declspec(dllimport) void func();
class __declspec(dllimport) Class;
```

При запуске программы файл библиотеки с расширением .dll должен находиться в одном каталоге с файлом программы или быть доступен по путям поиска.

Второй способ загрузки динамической библиотеки основывается на вызове функции `LoadLibrary`. В параметре *`lpFileName`* явно указывается путь до файла библиотеки. Результатом вызова этой функции является загрузка библиотеки в виртуальное адресное пространство процесса. Процессу становится доступен дескриптор (`HMODULE`) библиотеки. Для получения адреса ресурса, находящегося в библиотеке, необходимо вызвать функцию `GetProcAddress`. Параметр *`lpProcName`* должен содержать указатель на верное имя ресурса. Следует отметить, что имя ресурса в таблице экспорта отличается от имени ресурса, определенного в библиотеке из-за механизма декорирования имён. Имя ресурса следует получать с помощью утилиты dumpbin.exe.

Для выгрузки DLL из виртуального адресного пространства процесса используется функция `FreeLibrary`. В параметре *`hModule`* следует указать дескриптор библиотеки, полученный с помощью функции `LoadLibrary`.

Задания

Необходимо разработать программу, состоящую из головной программы и двух динамически загружаемых библиотек. Одна библиотека должна загружаться с использованием динамического связывания, а другая – с использованием функции LoadLibrary. Библиотеки должны выполнять действия по вариантам заданий из лабораторной работы №4.

Контрольные вопросы

1. Перечислите, в каких случаях вызывается функция `DllMain`.
2. Как с использованием DLL создать собственный API?
3. Для чего предназначены таблицы экспорта и импорта функций?
4. Как с использованием DLL организовать работу подключаемых модулей (plugins)?

Лабораторная работа №6
Перехват и фильтрация информации с использованием системных перехватчиков

Цель работы

Изучить методику установки и снятия системных перехватчиков для эффективной реакции программы на действия пользователя.

Список используемых системных вызовов

`SetWindowsHookEx, UnhookWindowsHookEx, CallNextHookEx.`

Методические указания

Для перехвата оконных сообщений, системных сообщений, нажатий на клавиатуру, событий изменения положения и размера окна и т. п. используются системные перехватчики. Системный перехватчик можно установить на выбранную нить или на все нити текущего рабочего стола. Необходимо отметить, что при использовании операционных систем Windows 8 и Windows 10 для успешной работы перехватчика пользователь должен входить в группу администраторов.

Для установки системного перехватчика используется функция `SetWindowsHookEx`. Параметр *`idHook`* используется для выбора типа перехватчика. Параметр *`dwThreadId`* задает идентификатор нити, на которую устанавливается перехватчик. Если он равен нулю, то перехватчик устанавливается на все нити. Функция перехватчика (параметр *`lpfn`*) в этом случае должна находиться в DLL, а параметр *`hMod`* должен содержать дескриптор библиотеки. Функция возвращает дескриптор на объект перехватчика.

Внутри перехватчика должна вызываться функция `CallNextHookEx` для обеспечения цепочки вызова перехватчиков. Вызов этой функции не обязателен, но настоятельно рекомендован. Если не вызывать эту функцию, то все перехватчики, установленные ранее, не получат управление.

Для отключения перехватчика используется функция `UnhookWindowsHookEx`. Параметр *`hhk`* должен содержать закрываемый дескриптор системного перехватчика.

Задания

1. Разработать программу, выводящую на экран текстовое сообщение при нажатии пользователем комбинации клавиш Alt+S.
2. Разработать программу, выводящую на экран текстовое сообщение при приведении курсора мыши в левый верхний угол экрана.
3. Разработать программу, выводящую на экран текстовое сообщение при минимизации любого окна на рабочем столе.
4. Разработать программу, выводящую на экран текстовое сообщение при активизации любого окна, содержащего в имени слово «СПО».
5. Разработать программу, выводящую на экран текстовое сообщение при закрытии любого окна верхнего уровня.
6. Разработать программу, выводящую на экран текстовое сообщение при вводе пользователем с клавиатуры слова «СПО».
7. Разработать программу, выводящую на экран текстовое сообщение при изменении размеров или перемещении любого окна.
8. Разработать программу, выводящую на экран текстовое сообщение при нажатии пользователем левой кнопки мыши в правом нижнем углу окна.
9. Разработать программу, выводящую на экран текстовое сообщение при нажатии пользователем левой кнопки мыши в меню окна.
10. Разработать программу, выводящую на экран текстовое сообщение при нажатии пользователем левой кнопки мыши в заголовке окна.
11. Разработать программу, выводящую на экран текстовое сообщение при отсутствии движений мышью в течение 1 минуты.
12. Разработать программу, выводящую на экран текстовое сообщение при запуске нового оконного приложения.

Контрольные вопросы

1. Как с использованием системных перехватчиков запрограммировать реакцию программы на нажатие «горячего» сочетания клавиш?

2. Как с использованием системных перехватчиков запрограммировать реакцию программы на нахождение курсора мыши в заранее заданной области?
3. Как с использованием системных перехватчиков запрограммировать всплывающие подсказки пользователю?

Лабораторная работа №7
Работа с файлами в асинхронном режиме

Цель работы

Изучение способов работы с файлами в Win32 API. Изучение асинхронного режима работы с файлами.

Список используемых системных вызовов

`CreateFile`, `CloseHandle`, `ReadFile`, `WriteFile`, `CancelIo`, `WaitForSingleObject`, `WaitForMultipleObjects`, `GetFilePointer`, `SetFilePointer`, `FindFirstFile`, `FindNextFile`, `FindClose`, `SearchFile`.

Методические указания

При выполнении данной лабораторной работы запрещается пользоваться группами функций Win32 API CopyFile… и MoveFile…, а также функциями библиотеки shell32.dll.

Для создания/открытия файла в Win32 API используется функция CreateFile (параметр *`dwCreationDisposition`* используется для задания режима открытия или создания файла). Чтение из файла и запись в файл реализуются с помощью функций `ReadFile` и `WriteFile`. Дескриптор файла закрывается с помощью стандартной функции `CloseHandle`.

Поиск файла в заданном каталоге возможен с помощью функций `FindFirstFile`, `FindNextFile`, `FindClose`. Поиск файла по путям поиска производится с помощью функции `SearchFile`.

В Win32 API реализована возможность задания асинхронного режима работы с файлами. При использовании этого режима чтение и запись в файл реализуются параллельно работе нити, задавшей файловую операцию. Для обеспечения возможности асинхронного режима работы с файлом необходимо в параметре *`dwFlagsAndAttributes`* функции `CreateFile` задать битовый флаг `FILE_FLAG_OVERLAPPED`.

После этого параметр *lpOverlapped* у функций чтения и записи в файл должен являться указателем на экземпляр структуры OVERLAPPED, представленной ниже:

```
typedef struct _OVERLAPPED {
  ULONG_PTR Internal;
  ULONG_PTR InternalHigh;
  DWORD Offset;
  DWORD OffsetHigh;
  HANDLE hEvent;
} OVERLAPPED;
```

Параметры структуры Offset и OffsetHigh задают начальное смещение от начала файла для выполнения файловой операции. Дескриптор hEvent должен быть верным дескриптором события, порожденного с помощью функции CreateEvent. Параметры Internal и InternalHigh используются самой ОС.

После задания асинхронной операции с файлом функция ReadFile или WriteFile немедленно возвращает управление. После окончания файловой операции событие hEvent переходит в сигнальное состояние и может быть проверено с помощью стандартных WaitFor-функций.

Функция CancelIo немедленно прекращает все ждущие асинхронные файловые операции по заданному дескриптору файла для текущей нити.

Задания

1. Разработать программу, копирующую файл из одного каталога в другой. Операции копирования файла должны выполняться одной нитью в асинхронном режиме.
2. Разработать программу, осуществляющую подсчет количества символов латинского алфавита в файле. Чтение файла должно осуществляться параллельно подсчету. Процесс программы должен состоять из одной нити.
3. Разработать программу копирования больших файлов блоками. Операции чтения и записи должны производиться одной нитью в асинхронном режиме одновременно. Необходимо подобрать размер блока, который приводит к максимальной скорости копирования.
4. Разработать менеджер копирования файлов, позволяющий прервать текущую операцию копирования по запросу пользователя.

5. Разработать менеджер копирования файлов, выполняющий операции копирования в фоновом режиме с помощью одной нити и визуализацию процесса копирования в другой нити.
6. Разработать программу, позволяющую выполнять одновременно до трех файловых операций в асинхронном режиме, используя три предварительно созданных события. Каждая следующая файловая операция должна работать с одним из свободных событий. Если все события заняты, то необходимо дождаться освобождения любого из них.
7. Разработать менеджер копирования файлов, состоящий из двух нитей. Одна нить должна инициировать операцию копирования, а вторая нить уведомлять о завершении операции.
8. Разработать программу, осуществляющую подсчет числа строк в большом текстовом файле. Чтение файла должно осуществляться параллельно подсчету. Процесс программы должен состоять из одной нити.
9. Разработать программу поиска всех файлов в каталоге «c:\». По каждому найденному файлу должна быть сделана запись в файле лога с помощью асинхронного режима записи.
10. Разработать программу, осуществляющую подсчет количества символов русского алфавита в файле. Чтение файла должно осуществляться параллельно подсчету. Процесс программы должен состоять из одной нити.
11. Разработать программу, осуществляющую подсчет числа слов в текстовом файле. Чтение файла должно осуществляться параллельно подсчету. Процесс программы должен состоять из одной нити.
12. Разработать программу, осуществляющую транслитерацию символов русского алфавита в текстовом файле в символы латинского алфавита. Чтение из файла, запись в файл и транслитерация должны осуществляться параллельно подсчету. Процесс программы должен состоять из одной нити.

Контрольные вопросы

1. Чем отличаются синхронные и асинхронные операции?
2. Может ли одна нить работать с несколькими файлами одновременно?
3. Можно ли изначально асинхронную операцию сделать синхронной с помощью программных средств?
4. Можно ли изначально синхронную операцию сделать асинхронной с помощью программных средств?

Список литературы

1. Гордеев А.В., Молчанов А.Ю. Системное программное обеспечение. – СПб.: Питер, 2001. – 736 с.
2. Дж. Рихтер. Windows для профессионалов. Создание эффективных WIN32-приложений с учетом специфики 64-разрядной версии Windows. Изд-ва: Питер, Русская Редакция, 2001 г., 752 стр. ISBN 5-272-00384-5, 1-57231-996-8
3. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования.: Пер. с англ.— М.: Издательский дом «Вильямс», 2003. — 512 с.
4. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. — СПб.: Питер, 2015. — 1120 с.
5. Робачевский А.М. Операционная система UNIX®. – СПб.: БХВ–Санкт-Петербург, 1999. – 528 с.
6. Соломон Д., Руссинович М. Внутреннее устройство Microsoft Windows 2000. Мастер-класс. / Пер. с англ.– СПб.: Питер; Издательско-торговый дом «Русская редакция», 2001. – 752 с.
7. The Unicode® Standard. Version 11.0 – Core Specification. [Электронный ресурс] 2018 URL: http://www.unicode.org/versions/Unicode11.0.0/UnicodeStandard-11.0.pdf (Дата обращения 20.01.2019)
8. Уорд Б. Внутреннее устройство Linux. — СПб.: Питер, 2016. — 384 с.
9. Лав Р. Linux. Системное программирование. 2-е изд. — СПб.: Питер, 2014. — 448 с.
10. Advanced Configuration and Power Interface (ACPI) Specification. Version 6.2 [Электронный ресурс] 2017. URL: http://www.uefi.org/sites/default/files/resources/ACPI%206_2_A_Sept29.pdf (дата обращения: 11.01.2019)
11. Unified Extensible Firmware Interface (UEFI) Specification Version 2.7 Errata A [Электронный ресурс] 2017 URL: http://www.uefi.org/sites/default/files/resources/UEFI%20Spec%202_7_A%20Sept%206.pdf (дата обращения: 11.01.2019)
12. Дж. Рихтер, Дж. Кларк. Программирование серверных приложений для Microsoft Windows 2000. Изд-ва: Питер, Русская Редакция, 2001 г., 592 стр.

Содержание