Министерство образования Республики Беларусь

Учреждение образования

Белорусский государственный университет информатики и радиоэлектроники

Факультет непрерывного и дистанционного обучения

Кафедра информатики

Электронный учебно-методический комплекс

по дисциплине

Системное программирование

Для студентов специальности

1-31 03 04 Информатика

Минск 2011

# Общие сведения

## Сведения об ЭУМК

Электронный учебно-методический комплекс по дисциплине «Системное программирование» предназначен для студентов специальности «Информатика» вузов, а также может быть использован магистрантами, аспирантами и специалистами соответствующего профиля.

Электронный учебно-методический комплекс составлен на основе рабочей учебной программы по курсу «Системное программирование», утверждённой деканом факультета непрерывного и дистанционного обучения <дата утверждения>, регистрационный № УД 11‑XX‑YY/Р и рабочего учебного плана специальности 1-31 03 04 «Информатика».

**Составители:**

**С.И. Сиротко,** доцент кафедры информатики Учреждения образования «Белорусский государственный университет информатики и радиоэлектроники», кандидат физико-математических наук.

Рассмотрен и рекомендован к изданию на заседании кафедры информатики, протокол № 3 от 26.09.2011.

Одобрен и рекомендован к изданию методической комиссией факультета компьютерных систем и сетей, протокол № \_\_ от \_\_.\_\_.2011.

## Методические рекомендации по изучению дисциплины

В соответствии с учебным планом студенты дистанционной формы обучения специальности «Информатика» изучают курс «Системное программирование».

Учебным планом по данному курсу предусмотрено изучение теоретических вопросов, выполнение 2-х индивидуальных практических и 2-х контрольных работ. Изучение курса заканчивается сдачей экзамена. К сдаче экзамена студенты допускаются только при условии выполненных и защищенных индивидуальных практических и контрольных работ.

Рекомендуется изучать курс «Системное программирование» в соответствии с рабочей программой. Сначала необходимо ознакомиться с содержанием курса, затем изучить рекомендуемую литературу, обращая внимание на вопросы, выделенные в рабочей программе, после чего изучить теоретическое изложение курса по приведенным разделам, темам и вопросам, выполнить задачи для решения (выполнения контрольных работ) в соответствии с заданием.

## Рабочая учебная программа

**Учреждение образования**

**«Белорусский государственный университет**

**информатики и радиоэлектроники»**

УТВЕРЖДАЮ

Декан факультета непрерывного и дистанционного обучения

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. М. Бондарик

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2011 г.

Регистрационный № УД-11-23-\_\_\_/р.

**Системное программирование**

Рабочая учебная программа

для специальности 1‑31 03 04

**Информатика**

Факультет **непрерывного и дистанционного обучения**

Кафедра **информатики**

Курс **4**

Контрольные работы 2 **работы**

Индивидуальные практические работы **2 работы**

Всего часов **138 часов**

Экзамен 7 **семестр**

Форма получения

высшего образования **дистанционная**

Минск 2011

Рабочая учебная программа составлена на основе базового учебного плана специальности 1-31 03 04 «Информатика», утвержденного Министерством Образования Республики Беларусь 18.04.2006 г., рег. № G.31-025/баз.

**Составитель:**

**С. И. Сиротко,** доцент кафедры информатики Учреждения образования «Белорусский государственный университет информатики и радиоэлектроники», кандидат физико-математических наук.

Рассмотрена и рекомендована к утверждению на заседании кафедры информатики, протокол № 3 от 26.09.2011.

Зав. кафедрой информатики \_\_\_\_\_\_\_\_\_\_\_ Л. И. Минченко

Одобрена и рекомендована к утверждению Советом факультета компьютерных систем и сетей Учреждения образования «Белорусский государственный университет информатики и радиоэлектроники», протокол № \_\_ от \_\_.\_\_.2011.

Председатель \_\_\_\_\_\_\_\_\_\_\_ В. А. Прытков

СОГЛАСОВАНО

Начальник ОМОУП \_\_\_\_\_\_\_\_\_\_\_ Ц. С. Шикова

ПРОТОКОЛ СОГЛАСОВАНИЯ УЧЕБНОЙ ПРОГРАММЫ  
ПО ИЗУЧАЕМОЙ УЧЕБНОЙ ДИСЦИПЛИНЕ  
С ДРУГИМИ ДИСЦИПЛИНАМИ СПЕЦИАЛЬНОСТИ

|  |  |  |  |
| --- | --- | --- | --- |
| Название дисцип­лины, с которой требуется согласо­вание | Кафедра, обеспе­чивающая изуче­ние дисциплины | Предложения об изменениях в со­держании учеб­ной программы по изу­чаемой дисциплине | Решение, принятое кафедрой, разработав­шей учебную программу (с указанием даты и номера протокола) |
| Алгоритмы и структуры данных | Информатики | нет | согласовано |
| Компьютерные сети | Информатики | нет | согласовано |
| Архитектура компьютеров | Информатики | нет | согласовано |
| Операционные системы | Информатики | нет | согласовано |
| Операционные системы и среды | Информатики | нет | согласовано |

СОГЛАСОВАНО:

Зав. кафедрой информатики Л.И. Минченко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

**Цель преподавания дисциплины.** Целью изучения дисциплины «Системное программирование» является рассмотрение и изучение студентами современного системного программного обеспечения (ПО) в плане как теоретических основ и принципов построения и функционирования, так и практической разработки и эксплуатации, а также создать базу для последующего успешного освоения других курсов.

**Задачи изучения дисциплины.** В результате изучения курса студенты должны изучить теоретически и практически различные аспекты функционирования и разработки системного ПО в среде одно- и многозадачных операционных систем (ОС).

**Знать:**

– структуру, назначение, виды системного ПО;

– основные подсистемы ОС и их функции;

– программные интерфейсы ОС и других программных средств системного назначения;

– типичные задачи, решаемые на системном уровне, и предъявляемые при этом требования;

– особенности и принципы разработки системного ПО;

**Уметь:**

– разрабатывать и отлаживать системное ПО различного назначения для одно- и многозадачных сред выполнения;

– эффективно пользоваться программными интерфейсами ОС производства Microsoft и их компонентов;

– решать типовые задачи, связанные с эксплуатацией и сопровождением системного ПО;

**Иметь представление о:**

– архитектурах ОС, их особенностях, общих принципах организации и функционирования;

– методах и приемах программирования для многозадачных систем и организации взаимодействия процессов.

**Перечень дисциплин, усвоение которых необходимо для изучения данной дисциплины**

|  |  |  |
| --- | --- | --- |
| №  п./п. | Название дисциплины | Раздел, тема |
| 1. | Программирование | Все разделы дисциплины |
| 2. | Алгоритмы и структуры данных | Все разделы дисциплины |

Кроме того, она тесно связана с дисциплинами "Операционные системы", "Операционные системы и среды", "Архитектура компьютеров", "Компьютерные сети".

СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № темы | Название и содержание тем | Контрольная работа | Индивидуальная практическая работа | Оснащение индивидуальных практических работ | Литература (номера) | Рекомендуемый объём для изучения (в часах) | Форма контроля знаний |

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Четвертый курс** | | | | | | | |
| 1 | **Тема 1. Вводная: Системное программное обеспечение.** Цели и задачи курса. Системное программное обеспечение, его виды, назначение, особенности. Классификация системного ПО. |  |  |  | 1, 3, 7, 8, 10, 11 | 4 |  |
| 2 | **Тема 2. Операционные системы.** Общая структура и подсистемы ОС. Архитектуры ОС. Архитектура Win 32. |  |  |  | 1, 3, 7, 8, 10, 11 | 4 |  |
| 3 | **Тема 3. Вычислительные процессы. Управление процессами.** Понятие вычислительного процесса. Структура вычислительного процесса. Атрибуты и состояния, жизненный цикл процесса. Управление процессами, подсистема управления процессами в ОС. |  |  |  | 1-3, 7, 8, 10, 11 | 6 |  |
| 4 | **Тема 4. Приложения в ОС Microsoft.** Структура приложений MS-DOS и Win 32. Виды приложений Win 32. Событийное управление (Window messages) в Win 32. Перехват сообщений. |  |  |  | 2, 4, 5, 7, 8, 10, 11 | 4 |  |
| 5 | **Тема 5. Оконный интерфейс Windows.** Структура оконного интерфейса Windows. Объекты и API оконного интерфейса. Функционирование оконного интерфейса. |  |  |  | 2, 5, 7, 8, 10-12 | 4 |  |
| 6 | **Тема 6. Графическая подсистема Windоws.** Назначение, структура, концепции графической подсистемы Windows (GDI). Графический контекст. Объекты, инструменты, свойства, API GDI. |  | 1 | 1, 3‑5 | 2, 5, 7, 8, 10-12 | 4 | Зачёт по индивидуальной практической работе |
| 7 | **Тема 7. Управление памятью.** Задачи управления памятью. Подсистема управления памятью, ее функции, требования. Методы распределения памяти. API подсистемы управления памятью Win 32. | 1 |  |  | 1-3, 7, 8, 10, 11 | 8 | Зачёт по контрольной работе |
| 8 | **Тема 8. Многозадачность и многопоточность.** Концепции и виды многозадачности. Подходы к реализации многозадачности. Понятие потока и многопоточности. Планировщик задач, дисциплины планирования. Приоритеты. |  |  |  | 1-3, 7, 8 | 6 |  |
| 9 | **Тема 9. Взаимодействие процессов.** Особенности выполнение процессов в многозадачной среде. Задачи и проблемы взаимодействия. Средства межпроцессного и межпоточного взаимодействия (IPC). | 2 |  |  | 1-3, 7, 8, 10, 11 | 8 | Зачёт по контрольной работе |
| 10 | **Тема 10. Подсистема внешней памяти, файловая система.** Внешняя память. Организация дисковой подсистемы, доступ к дискам. Концепция файлов, файловой системы. Подходы к построению файловых систем. Функции файловых систем, доступ к ресурсам посредством файловой системы. |  | 2 | 1‑6 | 4, 5, 19, 21, 22 | 6 | Зачёт по индивидуальной практической работе |
| 11 | **Тема 11. Расширенные возможности работы с файлами в Win 32.** Отображение файлов в память. Асинхронный ввод-вывод. Сервисные функции работы с файлами. |  |  |  | 4, 5, 19, 21, 22 | 4 |  |
| 12 | **Тема 12. Управление внешними устройствами.** Организация доступа к внешним устройствам. Понятие логического устройства, драйвера. Виды логических устройств. Доступ к внешним устройствам в ОС DOS и Windows (примеры). |  |  |  | 2, 5, 9 | 4 |  |
| 13 | **Тема 13. Технологии сложных и разнородных приложений.** Динамически связываемые библиотеки (DLL) и их использование. Службы (services). Совместное использование приложений Native Win 32 и .NET. |  |  |  | 1, 2, 3, 4, 8, 36 | 6 |  |
| 14 | **Тема 14. Системный реестр и журналирование Win 32.** Системный реестр: назначение, структура, структуры данных, API доступа, правила использования. Назначение журналов, подсистема журналирования Win 32, структура журналов, объекты и API подсистемы журналирования. |  | 2 |  | 2, 7 | 4 |  |
| 15 | **Тема 15. Подсистема безопасности.** Понятие о безопасности в вычислительных системах. Угрозы, защищаемые объекты, участники подсистемы безопасности. Задачи подсистемы безопасности. Реализация подсистемы безопасности в ОС семейства Win NT. |  |  |  | 2, 7, 9 | 4 |  |
|  | | | | | | | Экзамен (30 ч) |

1. Наименование тем, их содержание

Тема 1. Вводная: Системное программное обеспечение.

Цели и задачи курса. Системное программное обеспечение, его виды, назначение, особенности. Классификация системного ПО.

**Рекомендуемая литература:** 1, 3, 7, 8, 10, 11.

Тема 2. Операционные системы.

Понятие операционной системы (ОС). Назначение, функции, общая структура и подсистемы ОС. Архитектуры ОС. Архитектура Win 32.

**Рекомендуемая литература:** 1, 3, 7, 8, 10, 11.

Тема 3. Вычислительные процессы. Управление процессами.

Понятие вычислительного процесса. Структура вычислительного процесса. Атрибуты и состояния, жизненный цикл процесса. Управление процессами, подсистема управления процессами в ОС.

**Рекомендуемая литература:** 1-3, 7, 8, 10, 11.

Тема 4. Приложения в ОС Microsoft.

Структура приложений MS-DOS и Win 32. Виды приложений Win 32. Событийное управление (Window messages) в Win 32. Перехват сообщений.

**Рекомендуемая литература:** 2, 4, 5, 7, 8, 10, 11.

Тема 5. Оконный интерфейс Windows.

Структура оконного интерфейса Windows. Объекты и API оконного интерфейса. Функционирование оконного интерфейса.

**Рекомендуемая литература:** 2, 5, 7, 8, 10-12.

Тема 6. Графическая подсистема Windоws.

Назначение, структура, концепции графической подсистемы Windows (GDI). Графический контекст. Объекты, инструменты, свойства, API GDI.

**Рекомендуемая литература:** 2, 5, 7, 8, 10-12.

Тема 7. Управление памятью.

Задачи управления памятью. Подсистема управления памятью, ее функции, требования. Методы распределения памяти. API подсистемы управления памятью Win 32.

**Рекомендуемая литература:** 1-3, 7, 8, 10, 11.

Тема 8. Многозадачность и многопоточность.

Концепции и виды многозадачности. Подходы к реализации многозадачности. Понятие потока и многопоточности. Планировщик задач, дисциплины планирования. Приоритеты и управление приоритетами.

**Рекомендуемая литература:** 1-3, 7, 8.

Тема 9. Взаимодействие процессов.

Особенности выполнение процессов в многозадачной среде. Задачи и проблемы взаимодействия. Средства межпроцессного и межпоточного взаимодействия (IPC).

**Рекомендуемая литература:** 1-3, 7, 8, 10, 11.

Тема 10. Подсистема внешней памяти, файловая система.

Внешняя память. Организация дисковой подсистемы, доступ к дискам. Концепция файлов, файловой системы. Подходы к построению файловых систем. Функции файловых систем, доступ к ресурсам посредством файловой системы.

**Рекомендуемая литература:** 4, 5, 19, 21, 22.

Тема 11. Расширенные возможности работы с файлами в Win 32.

Отображение файлов в память. Асинхронный ввод-вывод. Сервисные функции работы с файлами.

**Рекомендуемая литература:** 4, 5, 19, 21, 22.

Тема 12. Управление внешними устройствами.

Организация доступа к внешним устройствам. Понятие логического устройства, драйвера. Виды логических устройств. Доступ к внешним устройствам в ОС DOS и Windows (примеры).

**Рекомендуемая литература:** 2, 5, 9.

Тема 13. Технологии сложных и разнородных приложений.

Динамически связываемые библиотеки (DLL) и их использование. Службы (services). Совместное использование приложений Native Win 32 и .NET.

**Рекомендуемая литература:** 1, 2, 3, 4, 8, 36.

Тема 14. Системный реестр и журналирование Win 32.

Системный реестр: назначение, структура, поддерживаемые типы и структуры данных, API доступа, правила использования. Назначение журналов, подсистема журналирования Win 32, структура журналов, объекты и API подсистемы журналирования.

**Рекомендуемая литература:** 2, 7.

Тема 15. Подсистема безопасности.

Понятие о безопасности в вычислительных системах. Угрозы, защищаемые объекты, участники подсистемы безопасности. Задачи подсистемы безопасности. Реализация подсистемы безопасности в ОС семейства Win NT.

**Рекомендуемая литература:** 2, 7, 9

2. Индивидуальные практические работы, их характеристика

|  |  |  |  |
| --- | --- | --- | --- |
| № п./п. | Тема | Содержание | Объём в часах |

| 1 | 2 | 3 | 4 |
| --- | --- | --- | --- |
| 1. | Оконное приложение Win 32 | Типовой каркас приложения Windows, использование сообщений, оконных объектов, взаимодействие с графической подсистемой (реализация на уровне системных вызовов Win 32 API) | 8 |
| 2. | Файловая система | Реализация низкоуровневого доступа к файловой система семейства FAT | 8 |
| **Итого** | | | **16** |

3. Контрольные работы, их характеристика

|  |  |  |  |
| --- | --- | --- | --- |
| №  п./п. | Тема | Характеристика | Объём в часах |

| 1 | 2 | 3 | 4 |
| --- | --- | --- | --- |
|  | Принципы управления памятью, управление памятью в различных системах. | Цель работы – более детальное ознакомление с методами и алгоритмами управления памятью, закрепление полученных знаний, проверка качества усвоения материала. Задание предполагает описание и примеры различных механизмов управления памятью | 8 |
|  | Управление процессами и потоками и их взаимодействие | Цель работы – подробное изучение средств управления процессами и потоками как объектами Windows, а также базовых средств межпроцессного взаимодействия в Windows. Задание предполагает описание и примеры соответствующих системных объектов и функций Win 32 API. | 8 |
| **Итого** | | | **16** |

4. ЛИТЕРАТУРА

4.1. ОСНОВНАЯ

1. Бек, Л. Введение в системное программирование / Л. Бек : Пер. с англ. – М.: Мир, 1988. – 448 с., ил.

2. Вильямс, А. Системное программирование в Windows 2000 для профессионалов / А. Вильямс : Пер. с англ. – СПб.: Питер, 2001. – 624 с., ил.

3. Гордеев, А.В., Молчанов, А.Ю. Системное программное обеспечение / А.В. Гордеев, А.Ю. Молчанов – СПб.: Питер, 2001. – 736 с.: ил.

4. Джордейн, Р. Справочник программиста персональных компьютеров типа IBM PC, XT и AT / Р. Джордейн : Пер. с англ. – М.: Финансы и статистика, 1992. – 544 с.

5. Зубков, С.В. Assembler для DOS, Windows и UNIX / С.В. Зубков – 3-е изд. – М.: ДМК пресс, СПб.: Питер, 2006. – 608 с.

6. Рихтер, Дж. Программирование на платформе Microsoft .NET Frame­work / Дж. Рихтер – 3‑е изд. – СПб.: Питер, Русская Редакция, 2005. – 486 с.

7. Рихтер, Дж. Windows для профессионалов / Дж. Рихтер – СПб.: Питер, 2000. – 752 с.

8. Руссинович, М. Внутреннее устройство Microsoft Windows. / М. Руссинович, Д. Соломон – 4-е изд. – СПб.: Питер, Русская Редакция, 2005. – 992 с.

9. Сорокина, С.И. и др. Программирование драйверов и систем безопасности: Учеб. пособие. / Сорокина С.ТИ., Тихонов А.Ю., Щербаков А.Ю. – СПб.: БХВ-Петербург, М.: издатель Молчанов С.В. – 2002. – 256 с.: ил.

10. Таненбаум, Э. Современные операционные системы. / Э. Таненбаум – 2-е изд. – СПб.: Питер, 2002. – 1040 с.

11. Таненбаум, Э., Вудхалл, А. Операционные системы. Разработка и реализация. / Э. Таненбаум, А. Вудхалл – 3-е изд. – СПб.: Питер, 2007. – 704 с.

4.2. ДОПОЛНИТЕЛЬНАЯ

12. Григорьев, В.Л. Видеосистемы ПК фирмы IBM / В.Л. Григорьев – М.: Радио и связь, 1993. – 192 с.

13. Григорьев, В.Л. Микропроцессор i486. Архитектура и программирование / В.Л. Григорьев (в 4-х книгах) – М.: ГРАНАЛ, 1993

14. Гук, М. Аппаратные средства IBM PC / М. Гук – СПб, Питер, 1996. – 224 с.: ил.

15. Гук, М. Аппаратные интерфейсы ПК. Энциклопедия / М. Гук – СПб.: Питер, 2002. – 528 с.: ил.

16. Гук, М. Процессоры Pentium 4, Athlon и Duron / М. Гук, В. Юров – СПб.: Питер, 2001. – 512 с.: ил.

17. Кулаков, В. Программирование дисковых подсистем / В. Кулаков – СПб.: Питер, 2002

18. Кулаков, В. Программирование на аппаратном уровне. Спец. справочник / В. Кулаков – 2-е изд. – СПб.: Питер, 2003. – 848 с.

19. Майко, Г.В. Ассемблер для IBM PC / Г.В. Майко – Бизнес-Информ, 1997. – 212 с.

20. Мешков, А.В., Тихомиров, Ю.В. Visual C++ и MFC / А.В. Мешков, Ю.В. Тихомиров : Пер. с англ. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2002. – 1040 с., ил.

21. Юров, В.И. Assembler. Учебник для вузов / В.И. Юров – 2-е изд. – СПб.: Питер, 2007. – 640 с.

22. Assembler: практикум / В.И. Юров. – СПб.: Питер, 2002. – 400 с.: ил.

5. ПЕРЕЧЕНЬ КОМПЬЮТЕРНЫХ ПРОГРАММ, НАГЛЯДНЫХ И ДРУГИХ   
ПОСОБИЙ, МЕТОДИЧЕСКИХ УКАЗАНИЙ И МАТЕРИАЛОВ И ТЕХНИЧЕСКИХ СРЕДСТВ ОБУЧЕНИЯ

1. Системное программирование : электронный учеб.-метод. комплекс [Электронный ресурс]. – Минск : БГУИР, 2011. – Режим доступа : www.bsuir.by.

2. Borland IDE 3.1: C/C++ compiler, assembler – среда разработки, DOS

3. Microsoft Developer Network (MSDN)

4. Microsoft Software Developer Kit (SDK)

5. Microsoft Visual Studio – среда разработки, Win 32

6. TeachHelp – DOS

# Теоретический раздел

## Лекции

### Введение в дисциплину

Дисциплина «Системное программирование», преподаваемая для специальности «Информатика», имеет ряд особенностей, отличающих ее от других специальностей схожего направления. Во-первых, для ряда предметов, традиционно относимых к системному ПО (например, теория трансляторов) в учебном плане специальности предусмотрены отдельные самостоятельные дисциплины. Во-вторых, в плане присутствуют еще две близких по направленности дисциплины – «Операционные системы» и «Операционные системы и среды», изучаемых в следующих семестрах, что делает логичной своего рода специализацию дисциплин.

В результате, курс «Системное программирование» относительно невелик по объему и сконцентрирован на наиболее базовых вопросах системного ПО, построения и функционирования ПО вычислительных систем, отдельных компонентов (подсистем), причем практическое приложение изучаемого материала ориентировано на платформу Microsoft. Последующие курсы рассматривают ОС семейства Unix и, далее, более детально – различные аспекты многозадачного программирования, взаимодействия параллельных процессов, низкоуровневого программирования, специализированных ОС и т.д.

### Тема 1. Системное программное обеспечение

#### 1.1. Структура системного программного обеспечения

В сложившейся терминологии, системным принято называть ту часть программного обеспечения (ПО) вычислительной системы (ВС), которые обеспечивают работу самой ВС и исполнение ею прикладных программ. Так как функции системного ПО и его набор обычно мало зависят от характера прикладных задач, его относят к базовому (универсальному) ПО, в отличие от прикладного (специализированного). В то же время, значительная часть системного ПО зависит от особенностей самой ВС, т.е. является аппаратно-зависимым.

Примечание. Под системным программированием может также пониматься решение высокоуровневых задач проектирования сложных программных систем, разработка системы в целом. Здесь системным называется программирование на нижнем уровне, написание системных программ, взаимодействие с ними и с аппаратными средствами.

В свою очередь, системное программное обеспечение делится на следующие группы: операционные системы (ОС), драйверы и системные утилиты (см. рис. 1.1).



Рис. 1.1 — Разновидности программного обеспечения

Операционная система – центральная часть системного ПО, набор программ для обеспечения функционирования аппаратного и программного обеспечения вычислительной системы (устройства) как единого комплекса для решения конкретных задач.

Драйвер – программа, управляющая работой некоторого устройства – физического, логического или виртуального. Как правило, когда идет речь об обращении к логическому устройству, имеется в виду взаимодействие с его драйвером. Толкование понятия драйвера в различных системах может существенно различаться, равно как и технологические особенности самих драйверов.

К т.н. утилитам относят прочие программы общесистемного назначения: средства программирования (трансляторы – компиляторы и интерпретаторы, компоновщики и др.), отладчики, дисковые утилиты, планировщики, файловые менеджеры и т.д. Критерием классификации их как системных может служить универсальность выполняемых функций по отношению к прикладным задачам, ориентация на собственно вычислительную систему и приближенность к техническим средствам. Граница между именно системными и сугубо прикладными утилитами часто бывает размыта. Например, командный интерпретатор shell в Unix или explorer (Проводник) в Windows можно считать как частью системы, так и программами прикладного уровня, обеспечивающими интерфейс пользователя при решении повседневных задач.

### Тема 2. Операционные системы

#### 2.1. Основные функции (задачи) ОС

ОС является центральным элементом системного ПО и призвана обеспечивать:

– согласованную работу аппаратных и программных средств;

– возможность общего управления системой.

К основным функциям ОС относятся:

– управление процессами (задачами, потоками, программами и т.д.), то есть распределение процессорного времени объектам, которые его потребляют;

– управление ресурсами: памятью, внешними устройствами, файловой системой;

– управление вводом-выводом;

– обеспечение интерфейса с пользователем;

– общее управление и синхронизация.

Состав функций конкретных ОС, их содержание, полнота реализации могут очень сильно различаться в зависимости от назначения (специализации) системы, ее сложности, особенностей и ограничений технической платформы и т.д.

#### 2.2. Общая структура ОС

Саму ОС можно представить как ядро, набор функциональных подсистем (не включенных в ядро) и программный интерфейс (см. рис. 1.2).



Рис. 1.2 – Структура операционной системы

Ядро ОС осуществляет планирование и управление всеми ресурсами и процессами системы, т.е. наиболее важные функции из указанных выше. За исполнение этих функций обычно отвечают соответствующие модули ядра, а те или иные ресурсы и иные сущности, с которыми оперирует ядро, абстрагируются программными моделями – объектами ядра.

Программный интерфейс ядра (Application Program Interface — API) обеспечивает доступ к функциям (и объектам) ядра со стороны прикладных программ.

Функциональные подсистемы – программные модули, реализованные вне ядра как решающие второстепенные с точки зрения системы задачи и не требующие системных привилегий. Одновременно эти модули оказываются обычно более переносимыми и легче заменяемыми, чем модули ядра.

Аппаратное обеспечение системы (физические устройства) для большинства программ доступны не напрямую, а через специальную прослойку – слой аппаратных абстракций или абстрагирования оборудования (Hardware Abstraction Layer – HAL). HAL фактически является частью ядра, изолирующей остальные его компоненты от особенностей аппаратуры; он не всегда выделяется явным образом, но это желательно с точки зрения переносимости и многоплатформенности системы.

Для абстрагирования аппаратуры в виде логических устройств служат также драйверы. Они обычно рассматриваются как самостоятельные программные модули вне ядра, однако имеют привилегии ядра для доступа к аппаратным ресурсам.

Подробности внутреннего устройства ОС и взаимодействия ее компонентов, объединяемые под понятием архитектуры, также могут быть очень различными. Более подробное рассмотрение архитектур ОС отнесено к дисциплине «Операционные системы и среды».

### Тема 3. Вычислительные процессы. Управление процессами

Понятие, концепция вычислительного процесса.

Понятие потока (thread).

Структура вычислительного процесса. Образ процесса, сегменты памяти процесса.

Адресное пространство и пространство дескрипторов процесса.

Атрибуты и состояния, жизненный цикл процесса.

Уровни выполнения процесса: user mode, kernel mode.

Управление процессами, подсистема управления процессами в ОС.

### Тема 4. Приложения в ОС Microsoft

Структура приложений MS-DOS и Win 32.

Форматы исполняемых файлов MS-DOS и Win 32.

Виды приложений Win 32: оконные, консольные, службы, драйверы, библиотеки (DLL).

Основные общие особенности: изоляция адресных пространств, распределение адресного пространства, сегменты.

Событийное управление (Window messages) в Win 32.

Перехват сообщений.

### Тема 5. Оконный интерфейс Windows

#### 5.1. Краткие теоретические сведения

Окно (window) – один из наиболее важных объектов в ОС Windows. Окно представляет собой прямоугольную область, в которой приложение отображает выводимую информацию и из которой получает вводимую. Кроме того, окно участвует и в программных интерфейсах – оно является получа­телем со­общений. Существует много разновидностей окон: диалоговые, окна много­документных интерфейсов (MDI), окна сообщений, окна элементов управления. Присутствующие в системе окна образуют иерархию, в которой они могут находиться в отношениях владелец – подчиненный и предок – потомок. Окна имеют заголовки (Title, Caption), но в большинстве случаев обращение к ним происходит по описателям – Handle (системный тип HWND)

Класс окна (Window Class) определяет общие свойства и особенности поведения группы окон. Классы делятся на системные глобальные (к ним относятся, например, стандартные элементы управления), прикладные глобаль­ные (реализуются в DLL, требуют регистрации, после чего могут использо­ваться любыми приложениями) и прикладные локальные (регистрируются как доступные единственному приложению). Именно класс связывает окно с его оконной процедурой. Классы идентифицируются, как правило, по их именам.

Для создания и регистрации классов служат функции RegisterClass() и RegisterClassEx(), для создания окон любых видов – функции CreateWindow() и CreateWindowEx().

Оконная процедура (Window Procedure или WndProc) является получа­телем всех сообщений, адресованных окну. Обычно представляет собой кон­струкцию вида switch – case, распознающую поступившие сообщения и выпол­няющую их обработку:

LRESULT CALLBACK

MainWndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case M\_CLOSE: //команда на закрытие окна

DestroyWindow( hWnd);

break;

case WM\_DESTROY: //подтверждение о готовности деинициализации

PostQuitMessage( 0);

break;

case …

break;

default: //для всех остальных сообщений

return DefWindowProc( hWnd, uMsg, wParam, lParam);

}

return 0;

}

Примечание. В данном примере правильнее было бы проверять адресата сообщений и завершать приложение, только если это его главное окно.

Сообщения (Window Message) лежат в основе механизмов событийного управления приложениями. Они представляют собой структуры, передаваемые между приложениями и несущими информацию о событиях, команды, данные и так далее. Для представления сообщений служит структура MSG, поля которой содержат: тип сообщения, пару параметров wParam, lParam, адресата сообщения, время и экранные координаты его возникновения (имеет смысл для некоторых сообщений).

Сообщения передаются в приложение либо напрямую, либо через очередь. Прямая передача сообщения оконной процедуре является фактически косвенным вызовом этой процедуры, в том числе и из другого приложения. Для этого служит системная функция SendMessage(). Важно, что выполнение отправителя сообщения блокируется до завершения оконной процедуры.

Многие сообщения передаются не напрямую, а через очередь, что избавляет программы от взаимной зависимости и блокировок. Для этого служит функция PostMessage(). Кроме того, есть некоторые другие функции, сочетающие свойства синхронной и асинхронной отсылки, например SendNotifyMessage(), которая передает сообщение минуя очередь, но и не ожидает окончания его обработки, поэтому блокирующей не является.

Цикл обработки сообщений выбирает сообщения из очереди и перенаправляет их в соответствующие оконные процедуры. В упрощенном представлении, цикл образуют три основные функции:

MSG msg;

while (GetMessage(&msg,NULL,0,0)) //выборка сообщений из очереди

{

TranslateMessage(&msg); //доп. обработка (трансляция) сообщения

DispatchMessage(&msg); //передача сообщения в оконную процедуру

}

Здесь выбираются все сообщения для всех окон приложения, однако параметры GetMessage() позволяют накладывать на них фильтры.

Функция GetMessage() является блокирующей – при отсутствии сообще­ний в очереди она переводит приложение в состояние ожидания. При наличии подходящего по фильтрам сообщения оно удаляется из очереди, и его содержимое переносится в структуру MSG. Функция возвращает нулевое значение при получении единственного сообщения WM\_QUIT, для всех остальных – ненулевое. Как следствие цикл прекращается после приема этого сообщения.

Типичное приложение Windows строится по следующей общей схеме:

1) WinMain() — головной модуль программы;

2) инициализация внутренних переменных и прочие подготовительные операции;

3) RegisterClass() – регистрация локального класса;

4) CreateWindow() – создание главного окна приложения (если требу­ется, могут быть и другие классы и окна;

5) ShowWindow(), UpdateWindow() – отображение главного окна;

6) цикл обработки сообщений: GetMessage() – TranslateMessage() – DispatchMessage().

7) освобождение выделенных ресурсов, деинициализация внутренних переменных;

8) оконная процедура, прямых вызовов нет, так как предполагается «обратный вызов» (callback) со стороны системы.

Маршрут прохождения сообщения, стадии исполнения, взаимодействие с системой.

Вызовы генерации сообщений: SendMessage(), PostMessage(), SendNotifyMessage(), PostThreadMessage().

Некоторые группы сообщений: ввод-вывод, элементы управления, таймер, user, copydata.

### Тема 6. Графическая подсистема Windоws

GDI (Graphics Device Interface) представляет собой единый унифициро­ванный интерфейс устройств (средств) отображения графической информации в Windows.

Работа GDI базируется на понятии контекста устройства (device context – DC), который абстрагирует свойства реальных устройств: экран (окно на экране), принтер, битовый образ в памяти и так далее. Контекст иденти­фицируется его описателем (тип handle DC – HDC).

Получив контекст, программа может обращаться к нему с единым набором функций, причем поведение контекста (изображения контекста) долж­но быть одинаковым независимо от того, с каким устройством он связан.

Для получения контекста служат функции: GetDC(), GetWindowDC(), GetDCEx(). Они применимы для оконных (экранных) контекстов. Функции CreateDC() и CreateCompatibleDC() создают контексты, связанные с иными (не окна) объектами либо «контексты в памяти», не связанные с реальным устрой­ством. Освобождение выполняется функциями ReleaseDC() для оконных кон­текстов и DeleteDC() – для остальных.

Для формирования изображения в контексте служат функции графи­ческих примитивов (например, Ellipse(), DrawText() и так далее) и графические инструменты (объекты). Основными инструментами являются: «перо» (Pen), «кисть» (Brush) и «шрифт» (Font). Логика GDI такова, что система при отображении примитива сама выбирает соответствующий инструмент: напри­мер, при изображении закрашенного полигона текущее «перо» будет использо­вано для прорисовки его контура, а текущая «кисть» – для заполнения внутрен­ней области.

Объекты типа Bitmap или Metafile сами способны хранить изображение, поэтому могут служить «рабочей поверхностью» контекста. Объекты «па­литра», «область отсечения» и некоторые другие дополнительно влияют на формирование изображения.

Для создания инструментов служат соответствующие функции GDI API, например CreatePen(), CreateBrush() и так далее. Для сложных объектов может быть определено несколько функций, различающихся параметрами и получа­емым эффектом.

Контекст может иметь только по одному активному объекту каждого вида. Для их переключения служит функция SelectObject(), которая принимает экземпляр объекта (инструмента) и делает его текущим в заданном контексте; тип объекта определяется автоматически, и предыдущий объект этого типа выталкивается из контекста.

Общая схема отрисовки следующая:

– получение контекста;

– установка набора инструментов;

– формирование изображения из примитивов, в том числе со сменой используемых инструментов;

– освобождение контекста.

Для оконных контекстов, которые после окончания их использования не удаляются, а лишь освобождаются, перед освобождением следует восстановить те же инструменты, которые были активными при получении контекста.

В типичном случае перерисовка содержимого окна инициируется сооб­щением WM\_PAINT – оно сигнализирует, что текущее видимое содержи­мое в окне было утрачено, и система уже выполнила автоматическую пере­рисовку самого окна и его фона.

Настройка отображения в контексте: система координат, управление цветом, отсечение.

Помимо описанных выше графических примитивов GDI работает также и с растровыми изображениями, то есть целыми массивами (матрицами) пик­селей. В простейшем случае изображение может формироваться попиксельно с помощью функции SetPixel(), но обычно этот путь слишком медленный и неудобный. (Имеется также функция GetPixel() – доступ к отдельным пикселям изображения.)

GDI предоставляет ряд функций для манипулирования целыми фраг­мен­тами изображений:

BitBlt() – копирование прямоугольной области из одного контекста в другой без каких-либо преобразований;

StretchBlt() – копирование области с масштабированием по обеим осям (обычно с уменьшением, так как при растяжении изображение заметно портится);

MaskBlt() – перенос с маскированием части изображения;

PlgBlt() – перенос прямоугольного фрагмента в непрямоугольную область с соответствующим геометрическим искажением.

В ряде случаев формирование всего изображения заново может быть слишком длительным – например, различного рода анимация или изобра­жения из очень большого числа элементов. В этом случае эффективно использование «теневых» контекстов или «контекстов в памяти». Такие контексты не имеют связи с реальным устройством, вместо этого рабочую поверхность для них образует битовая карта (bitmap) или метафайл. (Под метафайлом здесь понима­ется объект Metafile, фактически хранящий последо­вательность приме­няв­шихся к нему графических примитивов и способный их воспроизвести.)

«Теневой» контекст создается совместимым с текущим экранным режимом, и с ним ассоциируется «битовая карта»:

hShadowDC = CreateCompatibleDC( NULL);

hBitmap = CreateCompatibleBitmap( hPrimDC, width, height);

SelectObject( hShadowDC, hBitmap);

Здесь hPrimeDC – «реальный» контекст, в котором необходимо формировать изображение.

После создания «теневого» контекста в нем формируется необходимое изображение. Далее в процессе работы программы это изображение или его фрагменты переносятся на «основной» (видимый) контекст.

### Тема 7. Управление памятью

Задачи управления памятью.

Подсистема управления памятью, ее функции, требования.

Методы распределения памяти.

API подсистемы управления памятью Win 32.

Подсистема памяти MS-DOS (как пример простейшей реализации)

### Тема 8. Многозадачность и многопоточность

#### 8.1. Концепции и виды многозадачности

#### 8.2. Основные подходы к многозадачности

#### 8.2. Процессы

#### 8.3. Потоки

Поток (Thread) – системный объект, соответствующий последователь­ности («по­току») команд, выполняемой независимо и асинхронно по отно­шению к другим подоб­ным последовательностям. Поток – базовый объект, для которого планировщик задач ОС распределяет время центрального процессора. Каждый процесс имеет как минимум один главный (первичный – primary) поток. Главный поток может порождать подчиненные (вторичные) потоки, которые будут выполняться одновременно с ним, равно как и с потоками прочих процессов.

Многопоточность, свойственная Win 32 и ряду других современных ОС, удачно дополняет их многозадачность и существенно повышает гибкость системы. Процесс в Win 32 играет роль владельца ресурсов и «контейнера потоков» планирования ресурсов. Ресурсы процесса доступны всем его пото­кам, и все потоки одного процесса совместно используют его виртуальное адресное пространство (но каждый поток имеет собственный стек). В то же время процессорное время распределяется именно между потоками, а не между процессами.

Поток может находиться в нескольких состояниях, начиная от его иници­ализации до завершения, но наибольший интерес представляют следу­ющие:

– выполнение (running) или активность (active) – поток обладает всеми необходимыми ресурсами, включая процессорное время, и выполняется;

– готовность (ready) – поток обладает ресурсами для выполнения, но время ему не выделено, и он ожидает активизации в очереди планировщика;

– ожидание (wait), или заблокированное (blocked), или «спящее» (sleep) – поток не обладает требуемыми ресурсами и не может выполняться, он ожидает выполнения необходимых условий или наступления каких-либо событий.

Например, активный поток переходит в состояние ожидания после обращения к блокирующему системному вызову до его завершения, затем перемещается в очередь планировщика, откуда выбирается для следующей активизации. В простейшем случае переход в состояние ожидания произойдет при выполнении вызова Sleep().

Планировщик задач ОС в рамках принятой в Win 32 модели много­за­дач­ности выделяет процессорное время отдельным потокам квантами. Пере­клю­чение потоков и изменение их состояний происходит по мере израсхо­дования выделенных квантов и в соответствии с приоритетами потоков.

Потоки идентифицируются их описателями (handle), уникальными в рамках одного процесса, и идентификаторами (уникальны в системе). Однако большинство функций работают именно с описателями. Поток всегда сам является обладателем специального локального описателя, поэтому закрытие «внешнего» его описателя не приводит к удалению объекта. Одновременно поток как объект системы не может быть удален даже после прекращения его выполнения, если «внешний» описатель не был закрыт. Неверная работа с описателями может приводить при интенсивном создании и разрушении потоков к заметному накоплению «мусора» и дальнейшим ошибкам.

Новые потоки порождаются из функций потока с помощью вызова CreateThread(). Функция потока – одна из функций головной программы, соответствующая требованиям по типу, параметрам и формату вызова:

DWORD WINAPI ThreadProc( LPVOID pParam)

{

… //действия, выполняемые в потоке

return result;

}

Интерпретации как параметра, так и возвращаемого значения произ­вольны и зависят от конкретной программы. Например, параметр может быть адресом структуры или просто целым числом (используя приведение типа).

Все операции, выполняемые этой функцией, включая вызовы других подпрограмм, будут выполняться внутри созданного потока. Поток заверша­ется после завершения его функции. Возвращаемое ею значение будет доступ­но после завершения потока до его окончательного разрушения.

Последним параметром CreateThread() передается указатель на ячейку типа DWORD, куда записывается возвращаемый идентификатор созданного потока. В Window 9x этот указатель не должен быть нулевым. В Windows NT такого ограничения нет.

Уже существующий поток может быть открыт вызовом OpenThread(), для этого необходимо знать его идентификатор.

Для корректного завершения потока служит функция ExitThread(), вызы­ваемая «изнутри» потока, её аргументом является код возврата потока. Факти­чески результат ничем не отличается от стандартного завершения функции потока. Принудительное завершение потока, в том числе и «извне», возможно с помощью функции TerminateThread(), но при этом глобальные данные и другие ресурсы, с которыми он в этот момент работал, могут остаться в некорректном состоянии. Поэтому предпочтительнее организовать взаимо­дей­ствие так, чтобы завершение потоков согласовывалось с их текущим состоянием, оптимально – обеспечивалось самим же потоком.

Кроме завершения, поток может быть приостановлен и вновь запущен функциями SuspendThread() и ResumeThread() соответственно. В действи­тельности для потока ведется счетчик, при нулевом значении которого он может выполняться, при ненулевом (положительном) – останавливается. SuspendThread() увеличивает «счетчик остановок», ResumeThread() уменьшает его, но не ниже нулевого значения. Это позволяет делать «вложенные» при­оста­новки потока без риска ошибочно разблокировать его, но требует следить за сбалансированностью приостановок.

Остановленный поток не участвует в планировании выполнения, и его разблокирование зависит только от описанного счетчика.

Помимо потоков, в Windows (начиная с Win 2000) поддерживается еще одна, более мелкая единица планирования выполнения – нить (fiber). Нити создаются только в рамках потока. Переключение между нитями одного потока выполняется в любой момент времени, но только явным образом, по инициативе активной нити, путем обращения к соответствующей системной функции.

#### 8.4. Дисциплина планирования и приоритеты процессов и потоков

В системах с псевдопараллельным исполнением программ процессорное время выделяется им поочередно и в общем случае неравномерно – в соответствии с дисциплиной планирования. Как отмечалось выше, в универсальных системах наиболее часто реализуется вытесняющая многозадачность, предусматривающая выделение времени квантами. Для таких систем дисциплина планирования будет включать в себя способ выбора одной из нескольких конкурирующих программ для предоставления ей очередного кванта, количество предоставляемых квантов, а также ситуации досрочной передачи управления.

Существуют различные дисциплины планирования, однако в универсальных системах наиболее приемлемым оказалось использование приоритетов – численных значений, показывающих уровень «первенства» в очереди за квантом. Обычно программе с более низким приоритетом время не выделяется, если на него претендует более приоритетная. Более того, менее низкоприоритетные программы могут прерываться до истечения кванта времени, если в очереди появляется более приоритетная. Но в сбалансированной системе высокоприоритетный код, способный блокировать выполнение большинства других программ, выполняет, как правило, короткие операции, связанные с реагированием на события. Кроме того, само событийное управление процессами предполагает частое переклю­чение в состояние ожидания. Это дает шанс на исполнения программам с низким уровнем приоритета.

Примечание. В различных системах значения («номера») приоритетов могут назначаться как по возрастанию, так и по убыванию: более высокому уровню приоритета могут соответствовать как бóльшие, так и мéньшие его численные значения.

Для многопоточных систем, в том числе Windows, правильнее говорить о приоритетах потоков, так как в планировании участвуют именно потоки. Однако многоступенчатая схема назначения и учета приоритетов в Windows учитывает также и принадлежность потоков объектам более высокого уровня иерархии – процессам.

Уровни приоритетов варьируются в диапазоне от 0 (низший) до 31 (высший).

Действующий приоритет каждого потока образуют три составляющие: класс приоритета процесса, уровень приоритета потока внутри класса при­оритета процесса, динамический уровень приоритета.

Класс приоритета процесса и уровень приоритета потока внутри класса определяют базовый уровень приоритета потока. Привилегированными счита­ются приоритеты от 16 до 31, они резервируются за системными программами реального времени. Прикладная программа получить приоритет выше 15 не может.

Определены следующие классы приоритетов, которым соответствуют константы:

– Idle (простаивающий) – IDLE\_PRIORITY\_CLASS (4), процесс активи­зируется только при простое других процессов;

– Normal (нормальный) – NORMAL\_PRIORITY\_CLASS (7), большинство процессов в системе, в частности, все процессы пользователя; приоритет владеющего активным окном процесса повышается на 2 и составляет 9;

– High (высокий) – HIGH\_PRIORITY\_CLASS (13), системные процессы, реагиру­ющие на соответствующие события;

– Real time (реального времени) – REALTIME\_PRIORITY\_CLASS (24), некоторые системные процессы в "особых случаях".

Внутри классов приоритетов процессов определены уровни приоритетов потоков:

– низший (THREAD\_PRIORITY\_LOWEST) – –2 от уровня класса;

– пониженный (THREAD\_PRIORITY\_BELOW\_NORMAL) – –1 от уровня класса;

– нормальный (THREAD\_PRIORITY\_NORMAL) – равен уровню класса;

– повышенный (THREAD\_PRIORITY\_ABOVE\_NORMAL) – +1 к уровню класса;

– высший (THREAD\_PRIORITY\_HIGHEST) – +2 к уровню класса;

– простаивающий (THREAD\_PRIORITY\_IDLE) – равен 16 для REALTIME\_­PRIORITY\_­CLASS и 1 для остальных классов;

– «критический» (THREAD\_PRIORITY\_TIME\_CRITICAL) – равен 31 для REALTIME\_PRIORITY\_CLASS и 15 для остальных классов.

Для класса REALTIME\_PRIORITY\_CLASS может использоваться также расширенный диапазон значений – от –7 до +6. Начиная с Windows 2003 были добавлены два специальных значения: THREAD\_MODE\_BACKGROUND\_BEGIN и THREAD\_MODE\_BACKGROUND\_END, они связаны с дополнительными возмож­ностями планировщика.

Динамический уровень приоритета образуется повышением базового уровня потока на две единицы при поступлении сообщений в его очередь; по истечении некоторого времени восстанавливается исходное значение. Вре­менное повышение приоритета делается также и для долго не получавшего управление потока. Эти правила действуют только для потоков с уровнем приоритета не выше 15.

Для управления приоритетами выполнения процессов и потоков служат следующие функции.

GetPriorityClass() – получение текущего класса приоритета для процесса;

SetPriorityClass() – установка класса приоритета для процесса;

GetThreadPriority() – получение текущего приоритета выполнения потока;

SetThreadPriority() – установка приоритета выполнения потока.

### Тема 9. Взаимодействие процессов

В многозадачной (многопоточной) среде возникает проблема одновре­мен­ного доступа к одним и тем же ресурсам (данным) со стороны нескольких потоков. В случае если конкретный ресурс не допускает такого использования, возникает конфликт, требующий разрешения. Это называют задачей синхро­ни­зации (взаимного исключения).

Критический ресурс – ресурс (объект, данные), который в силу своей физической природы либо логики использования не может быть доступен одно­временно нескольким пользователям. В зависимости от контекста речь может идти об определенном сочетании обращений или любых обращениях вообще (например, одновременное чтение допустимо, но одновременная запись или чтение и запись – нет).

Примерами критических ресурсов могут служить устройство вывода (одновременно выводимые данные нескольких источников будут перемешаны), записываемый файл (чтение до окончания записи дает неверные результаты), счетчик цикла (постороннее изменение нарушает работу цикла) и так далее.

Критическая секция – участок кода, выполняющий обращение или логи­чески связную последовательность обращений к критическому ресурсу. В зависимости от контекста может быть удобнее представлять, что критическая секция одна и определяется связью ее с критическим ресурсом, либо однотип­ные критические секции принадлежат различным пользователям, но связаны посредством единого критического ресурса.

В этих терминах задача исключения сводится к обеспечению единст­венности пользователя, находящегося внутри критической секции, связанной с данным критическим ресурсом.

Проблема усугубляется еще и тем, что проверка условий возможности доступа должна быть неотделима от самого доступа или хотя бы блокировки его для других пользователей, то есть атомарность. Так как прикладная программа обычно не может обеспечить это самостоятельно, много­задачные системы обязательно предоставляют механизмы синхронизации.

Простейшее средство исключения в Windows – объект CriticalSection. Он представляет собой обычную структуру и не имеет глобальной идентификации, поэтому может использоваться только потоками одного процесса. Исполь­зование CriticalSection выглядит в общем случае следующим образом:

CRITICAL\_SECTION cs;

InitializeCriticalSection( &cs);

…

EnterCriticalSection( &cs);

… //защищенная критическая секция

LeaveCriticalSection( &cs);

…

DeleteCriticalSection( &cs);

Вызов EnterCriticalSection – попытка входа в критическую секцию. Если секция уже занята другим потоком, то поток блокируется до момента ее осво­бождения. Повторное вхождение в критическую секцию допускается толь­ко для одного и того же потока, что помогает избегать самоблокировок. Блоки­ровка управляется счетчиком, каждое вхождение в секцию увеличивает счет­чик, освобождение – уменьшает, поэтому их количество должно быть сбаланси­ровано.

Для синхронизации потоков разных процессов предусмотрены специ­альные объекты: Event, Mutex, Semaphore, WaitableTimer. Все они являются именованными и допускают глобальную идентификацию по именам.

Сам перевод потока в состояние ожидания осуществляется функциями WaitForSingleObject(), WaitForMultipleObjects() и их разновидностями. В зави­симости от параметров эти функции блокируют выполнение потока до обнаружения одного или нескольких переданных им объектов в состоянии «готовности» (signaled).

Объект «событие» (Event) – наиболее простая разновидность. Объекты создаются функцией CreateEvent() и бывают двух типов: «ручные» и «авто­матические». Открытие существующего объекта «событие» по его имени – OpenEvent(). Установка события (перевод в состояние signaled) происходит всегда явно функцией SetEvent(), сброс – для «ручных» событий явно, функ­цией ResetEvent(), для «автоматических» – автоматически при успешном вы­полне­нии Wait-функции. Также имеется функция PulseEvent() – временная установ­ка события, активизация всех ожидавших его потоков и автоматический сброс. Кроме того, объекты Event ассоциируются с файлами при организации асинхронного ввода-вывода.

Объект «мьютекс» (Mutex) – простейший двузначный семафор для организации критических секций. Принято считать, что он «захватывается» потоком, и если мьютекс в этот момент уже занят, очередной захватывающий его поток блокируется. Подобно CriticalSection, Mutex допускает повторный захват, но только одним и тем же потоком. Создание нового мьютекса – вызов CreateMutex(), открытие существующего по имени – OpenMutex(), осво­бож­де­ние – ReleaseMutex(), захват с возможной блокировкой – Wait-функции.

Объект «семафор» (Semaphore) – отличается от мьютекса тем, что является счетчиком и может принимать множество значений от нуля и выше. «Занятым» считается семафор с нулевым значением, с ненулевым – свобод­ным. При попытке опустить значение ниже нуля происходит блокировка. Работа семафора не зависит от того, разные потоки обращаются к нему или один и тот же. Создание нового семафора – вызов CreateSemaphore(), открытие сущест­вующего по имени – OpenSemaphore(), «подъём» счетчика (разблоки­рование) – ReleaseSemaphore(), проверка и «опускание», в том числе блоки­рование – Wait-функции.

Кроме специализированных объектов синхронизации Wait-функции мо­гут работать также и с другими объектами, например:

– процессы и потоки – ожидание завершения;

– файлы – ожидание окончания текущей операции, и так далее.

### Тема 10. Подсистема внешней памяти, файловая система

Данные на внешних носителях принято организовывать в виде файлов.

Файл — упорядоченный набор данных, пригодный для использования прикладными программами. Удобно представлять файл как совокупность данных (используются прикладными программами) и метаданных (данные о размещении данных, используются системными программами).

Для организации хранения данных, предоставления доступа к ним и обеспечения другого сервиса служит файловая система (ФС). В это понятие входят как структуры для собственно хранения данных и метаданных, так и включенные в ПО средства работы с ними.

Типичной для файловых систем является иерархическое (древовидное) построение: файлы могут объединяются в каталоги (директорий – directory, папка – folder), причем сами каталоги также могут быть вложенными. Таким образом, каталог также является файлом, но специального вида – предназна­ченным для хранения информации о включенных в него файлах и других каталогах.

Помимо каталогов, предусматриваются и другие виды специальных файлов: файлы логических устройств, файлы-ссылки, файлы – метки тома и т.д. В отличие от специальных типов объектов файловой системы «обычные» файлы программ или данных называют регулярными.

Файлы принято идентифицировать по их именам. Учитывая наличие каталогов и подкаталогов, для однозначной локализации файла требуется также и путь к нему (path) – перечисление всей цепочки каталогов.

Для ФС Microsoft характерна отдельная иерархия каталогов для каждого логического диска. Логический диск может соответствовать физическому нако­пителю или одному из его разделов, реже – виртуальному устройству. С точки зрения системы каждый логический диск рассматривается как самостоятельное устройство и идентифицируется буквой: A, B, C и т.д.

Таким образом, полный (абсолютный) путь к файлу будет начинаться от «буквы» логического диска, например: A:\directory1\directory2\...\file.ext. Относительный путь начинается от текущего каталога в файловой системе.

Традиционными для MS-DOS и ранних версий Windows были файловые системы FAT (более подробно см. ИПР №2). Первоначально имена ограничивались восемью символами собственно «имени» и тремя символами «расширения» (extention), причем набор символов ограничивался латинскими буквами, цифрами и некоторыми дополнительными символами: +, –, \_ и т.д. Позже в именах были допущены символы национальных алфавитов, а затем введены и «длинные» имена, причем уже с использованием символов Unicode. Пределом длины «длинного» имени считается 255 символов, а имени вместе с путём – 260 символов.

Традиционные имена «8.3» содержат буквы только в верхнем регистре, и приведение их выполняется автоматически, поэтому для пользователя имена не чувствительны к регистру букв. «Длинные» имена хранятся с учётом регистра букв, однако их сравнения всё равно регистронечувствительны. Поэтому имена будут соответствовать своим файлам независимо от регистра букв в них, а в одном каталоге не могут быть два файла, имена которых различаются только регистром букв.

Гораздо большие возможности предоставляют AC семейства NTFS, поддержка которых была введена начиная с Win NT.

Файловая система (ФС) FAT отличается простотой, что позволяет относи­тельно несложно запрограммировать низкоуровневый доступ к ней. По этой причине она до сих пор применяется для сменных носителей (флэш-накопи­телей, флэш-карт) и специализированных устройств с ограниченными ресур­сами, а ее поддержка присутствует практически во всех современных ОС.

Файлы в ФС FAT хранятся в виде связных списков блоков – кластеров. Кластер пред­ставляет собой объединение от 1 до 64 целых секторов и, следовательно, имеют размер от 512 байт до 32 Кбайт. Таким образом, каждый файл, в зависимости от размера, представляется одним или несколькими кластерами, образующими цепочку (chain).

Ссылки, обеспечивающие связность списка, хранятся отдельно от дан­ных, в специальной структуре – таблице размещения файлов (File Alloca­tion Table – FAT), размещение которой на диске фикси­ровано и заранее известно. Каждая ячейка таблицы позиционно соответствует одному кластеру и содержит номер следующего кластера в цепочке. Кроме того, следующие значения ячеек зарезервированы для специальных случаев: 0 – пустой кластер, FFFh – послед­ний кластер в цепочке, FF7h – кластер помечен как дефектный и не исполь­зуется (здесь приведены значения для FAT12, для прочих они аналогичны). Первые две ячейки таблицы объединены и содержат код типа носителя.

Наибольшее количество кластеров, представимых таблицей, зависит от разрядности ее ячеек. На практике встречаются 12‑, 16‑ и 32‑разрядные версии FAT – соответственно FAT12, FAT16 и FAT32 (в FAT32 реально используются лишь 28-разрядные значения, но хранятся они в 4‑байтовых ячейках). Малое число кластеров на разделах большого объема приводит к слишком большому размеру каждого кластера и, как следствие, непроизводительному расходу дис­кового пространства. Для сравнения: у фай­ловых систем, свободных от ограни­чений FAT, характерный размер распреде­ляемых блоков составляет несколько килобайт.

Для обеспечения надежности на диске присутствует обычно две копии FAT. Функции ОС, работающие с файловой системой, модифицируют обе ко­пии согласованно, а также распознают повреждение таблиц, но автомати­ческое восстановление не предусмотрено. В своих программах при непо­средственных обращениях к FAT также необходимо поддерживать коге­рент­ность её копий.

Сканирование цепочки кластеров требует знания ее начала – номера первого кластера. Он содержится в записи директория, описывающего файл. Записи директориев – 32-байтовые, содержат имя файла, его атрибуты и некоторую служебную информацию, в том числе и номер первого кластера файла (часть записей может быть занято «длинными» именами файлов – они хранятся разбитыми на фрагменты в нескольких соседних записях). Поэтому первоначально нужно считать содержимое директория, найти в нем нужное имя файла и взять из этой записи номер первого кластера.

Сведения о размещении таблицы FAT, ее размере, размере кластера и другая служебная информация о разделе могут быть получены из блока параметров BIOS (BIOS Parameters Block – BPB). Он содержится вместе с кодом загрузчика раздела (Boot Record) в первом его логическом секторе. Если носитель не предполагает наличия разделов (например дискета), то этот же сектор является и первым физическим, и других загрузчиков носитель не имеет.

При наличии разделов (логических дисков) потребуется также обраба­тывать таблицы разделов – Partition Tables, описывающие их размещение, тип и размеры. Первая таблица разделов содержится в «главной» загрузочной записи (MBR – Master Boot Record), в первом физическом секторе диска. Если среди разделов есть расширенные, то они сами включают вложенные разделы и поэтому в первом своем секторе также содержат таблицу разделов.

Работа с FAT требует организовать низкоуровневый доступ к диску, который осуществляется на уровне секторов. Сектор имеет фиксированный в пределах одного диска размер, стандартно 512 байт, и является минимальной одновременно передаваемой порцией данных при обмене с блочным устрой­ством. Логическими или относительными называют секторы в пределах одного логического диска (раздела), они адресуются линейно, т.е. фактически поряд­ковыми номерами или индексами. Нумерация начинается с 1 (в случае MS-DOS). Физические или абсолютные секторы отсчитываются в рамках всего диска, и для них действуют два основных подхода к адресации. Традиционная адресация CHS (Cylinder-Head-Sector) предполагает три компонента идентифи­кации сектора: цилиндр (номер дорожки), головка (поверхность) и номер сек­тора в дорожке. Для современных дисков компоненты CHS-адреса уже не соответствуют их реальной физической конфигурации и автоматически транслируются контроллером, поэтому их правильнее воспринимать как «логические» цилин­дры и поверхности. Поддержка больших объемов дисков для CHS-адресации ограничена разрядностью компонентов адреса: 1024 «цилиндра», 256 «поверхностей», 63 «сектора». Линейная адресация (LBA – Linear Block Addressing) гибче, проще в использовании и более масштабируема в сторону увеличения числа секторов.

Логические секторы в MS-DOS читаются и пишутся прерываниями int 25h и 26h соответственно, при этом передаются параметры: AL – номер устройства; DX – номер первого обрабатываемого сектора; CX – количество секторов; ES:BX – адрес буфера в памяти. Признаком ошибки служит нену­левое значение флага CF. Кроме того, после возврата из прерываний стек всегда остается невыровненным: на его верхушке находится слово с кодом результата, которое необходимо извлечь.

Так как доступ по абсолютным адресам потребуется, скорее всего, только для секторов в начале диска, то достаточно будет традиционных функций BIOS. Сервис BIOS доступа к дискам представлен прерыванием int 13h, имеющим ряд функций, в том числе чтение и запись группы секторов. Код функции всегда передается в AH (02h – чтение, 03h – запись); DL – номер устройства; CH – младшие 8 бит номера цилиндра; DH – номер поверхности; CL – номер начального сектора (биты 5..0) и два старших бита номера цилиндра (биты 7..6); AL – счетчик секторов (не более чем на одной дорожке); ES:BX – адрес буфера в памяти. Признаком ошибки выполнения служит единичное значение флага CF, в этом случае код ошибки возвращается в AH.

Использование прерываний DOS, как и многих других низкоуровневых операций, в среде Win 32 представляет собой серьезную проблему. Вместе с тем, готовых решений более высокого уровня эти системы также не предоставляют, так как это нарушает концепции безопасности исполнения приложений. Однако в Win NT имеется возможность обращения к псевдофайлам в специальном виртуальном каталоге //./, на которые, среди прочего, отображаются структуры разделов дисковых устройств.

Операции над внутренними структурами файловой системы потенци­ально опасны, поэтому рекомендуется ограничиваться, хотя бы на этапе отладки, съемными носителями, не содержащими уникальных данных. При выполнении большинства заданий для данной работы потре­буется дискета, отформатированная под FAT. Все задания, если не огово­рено иное, выполняются для корневого каталога этой дискеты. Вместо реаль­ной (физической) дискеты можно использовать виртуальный диск A:, создан­ный, например, программой vfd. Также может использоваться и флэш-накопителях, отформатированный в FAT.

Для доступа к файлу он должен быть предварительно открыт, все последующие действия будут выполняться над системным объектом «откры­тый файл». Первоначально в MS-DOS применялись FCB (File Control Block) – структуры, содержащие информацию о файле и выполняемой над ним операции; в ту же структуру могли заноситься и результаты операции. Позже был введен более удобный подход, основанный на использовании файловых дескрипторов (file descriptor). Дескриптор представляет собой целое число, получаемое после открытия или создания файла и однозначно идентифициру­ющее его. Полученный дескриптор действует до закрытия файла. Вся слу­жебная информация об объекте остается скрытой от прикладной программы, что упрощает работу. Сейчас FCB-ориентированные функции считаются устаревшими и поддерживаются лишь для совместимости с ранее написанным ПО, а основным методом доступа в современных ОС является дескрипторный.

### Тема 11. Расширенные возможности работы с файлами в Win 32

Отображение файлов в память.

Асинхронный ввод-вывод.

Сервисные функции работы с файлами.

### Тема 12. Управление внешними устройствами

Организация доступа к внешним устройствам.

Понятие логического устройства, драйвера.

Виды логических устройств.

Доступ к внешним устройствам в ОС DOS и Windows

Пример – доступ к COM-порту.

Пример – доступ к LPT порту

Пример – прямой доступ к портам ввода-вывода (разрешение доступа)

### Тема 13. Технологии сложных и разнородных приложений

#### 13.1. Многомодульная компоновка программ

Многомодульная компоновка, взаимодействие модулей на различных языках (отсылка к курсу «Программирование»)

#### 13.2. Технологии оверлеев и динамических библиотек (DLL)

Динамически связываемые (динамически подключаемые) библиотеки (Dynamic Link Library, DLL) являются наряду с .EXE-файлами разновидностью исполняемых модулей в Win32. Они имеют в целом аналогичную структуру, но не могут выполняться самостоятельно, а присоединяются к другим исполняемым файлам. Первоначально, подобно оверлеям (overlay) DOS, рассматривались как средство экономии объема программного обеспечения путем множественного использования посредством унифицированных интерфейсов одних и тех же модулей, реализующих типовые функции. Впоследствии стали также активно использоваться как средство структурирования крупных программных комплексов. Сама ОС Windows представляет собой в значительной мере набор динамических библиотек, считающихся стандартными.

В любом случае, библиотека DLL существует отдельно от использующего ее файла, и ее код присоединяется к основному лишь в процессе выполнения последнего. Характерно, что в Win32 каждый модуль DLL включается в адресное пространство основного процесса, причем каждое новое подключение модуля дает фактически новый его экземпляр, не зависящий от остальных экземпляров того же модуля, т.е. новый набор переменных и т.д. Для сравнения: в Win16 все экземпляры (подключения) модуля разделяли одно и то же адресное пространство, что усложняло программирование и снижало безопасность системы. Загрузка библиотеки опирается на те же механизмы, что и создание нового процесса (см. выше). Ввиду того, что сегмент кода библиотеки может разделяться всеми его экземплярами, а вся библиотека разделяется многими процессами и может загружаться ими многократно, для *каждой* библиотеки, загружаемой *каждым* процессом, создается счетчик привязок (загрузок), увеличивающийся при загрузке ее *данным процессом* и уменьшающийся при выгрузке. По достижении нулевого значения счетчика библиотека удаляется из адресного пространства *процесса*. Экземпляры библиотеки, используемые различными процессами, полностью независимы.

Библиотека содержит как сугубо внутренние, так и видимые извне *символы* – объекты, идентифицируемые их именами. Символами являются в первую очередь методы (функции), а также переменные и классы. Предоставление символа внешним по отношению к библиотеке программам принято называть *экспортом*. Традиционно экспортируются в основном методы. Экспорт переменных считается нежелательным, хотя в ряде случаев необходим; экспорт классов в рамках стандартной методики может оказаться излишне трудоемким. Информация об экспортируемых символах помещается в соответствующую таблицу модуля, откуда она считывается при подключении библиотеки в процессе *импорта* – получении точек доступа к символам. Таким образом, помимо его имени, экспортируемый символ может идентифицироваться также индексом – порядковым номером в таблице. Нумерация символов не должна выходить за пределы двухбайтового слова.

Имена в таблице экспортируемых символов не обязательно совпадают с именами, под которыми соответствующие объекты известны внутри модуля, т.к. компиляторы предоставляют возможность явно задавать "внешние" имена (см. ниже). Индексы символов не обязательно идут последовательно, т.к. также могут указываться явно.

Кроме символов, в библиотеке могут содержаться также ресурсы.

Примечание. Аналогичным образом загружаться и использоваться для получения символов или ресурсов могут также и другие исполняемые файлы, например .EXE.

Можно выделить 2 существенно различных способа подключения (импорта) DLL-модуля к программе: статический и динамический.

Динамическое подключение и импорт предусматривают использование специальных функций API (интерпретация Visual C++). При этом используются функции:

HINSTANCE LoadLibrary(LPCTSTR *lpModuleName*);

HMODULE GetModuleHandle(LPCTSTR *lpModuleName*);

Функции загружают указанный библиотечный модуль и включяют его в адресное пространство процесса либо получают описатель уже загруженного модуля. Возвращаемое значение – описатель загруженного модуля (экземпляра). Параметры:

lpModuleName – имя библиотечного модуля (имя файла).

Для освобождения (выгрузки) модуля служит функция:

BOOL FreeLibrary(HMODULE hModule);

Она в действительности уменьшает счетчик загрузок библиотеки на единицу, и по достижении нуля освобождает экземпляр библиотеки и исключение его из адресного пространства процесса. Возвращаемое значение – только признак успешности выполнения.

Параметры:

hModule – описатель загруженного модуля.

Наконец, после загрузки модуля можно получить доступ к ней, а точнее ее адрес в модуле с помощью функции:

FARPROC GetProcAddress(HMODULE *hModule*, LPCTSTR *lpProc*);

Функция возвращает адреса функции или любого другого символа, импортируемого из библиотечного модуля. Формат возвращаемого значения – указатель на функцию вида (\_\_stdcall \*)(void). Это универсальный абстрактный указатель, поэтому при экспорте символов другого типа требуется соответствующее его приведение.

Параметры вызова:

hModule – описатель загруженного модуля.

lpProc – указатель на строку, содержащую имя импортируемой функции или другого символа, если числовое значение этого указателя больше 0ffffh (импорт по имени), либо порядковый номер этого символа в таблице модуля, если значение не более 0ffffh (импорт по индексу).

Динамическое подключение и, соответственно, отключение модулей могут быть выполнены прикладной программой в любой (естественно, предусмотренный программистом) момент, что обеспечивает ему (программисту) наиболее широкие возможности в управлении программой, например, позволяет изменять состав загруженных модулей, производить поиск модулей и символов в них, и так далее. Временная загрузка и удаление библиотек могут снижать ресурсоемкость приложения. Одновременно при этом увеличивается трудоемкость программирования в типичных случаях, то есть однократная загрузка заранее известных библиотек.

Статические подключение и импорт включают в себя принципиально те же стадии, но выполняются они не прикладным процессом во время его исполнения, а загрузчиком при создании процесса. Компоновщик (linker) системы программирования помещает в заголовок .EXE-файла необходимую для этого информацию. После успешного завершения процедуры загрузки символы из библиотеки становятся доступными для головного модуля. В случае ошибки (обычно отсутствие необходимого модуля) процесс не создается, как правило, с выдачей сообщения о системной ошибке.

Статическое подключение менее трудоемко (для программиста), чем динамическое, но менее гибко. Кроме того, реализация данного подхода в различных средах программирования может сильно различаться на уровне ключевых слов, соглашений, структур описаний и т.д., что ухудшает переносимость проектов (см. ниже).

Независимо от способа подключения, библиотека имеет точку входа, описываемую функцией DllMain (Visual C++):

BOOL APIENTRY DllMain(HANDLE *hModule*, DWORD *fCall*, LPVOID *lpReserved*);

Тип возвращаемого значения BOOL совместим со стандартным int, ненулевое значение (TRUE) соответствует успешному выполнению, нулевое (FALSE) – неуспешному, в этом случае действие, инициировавшее обращение к точке входа, завершится с ошибкой, если результат для него критичен.

Модификатор APIENTRY соответствует стандартному \_\_stdcall, что означает принятый для Win32 API формат вызова: прямой (PASCAL) порядок аргументов, очистка стека вызванной подпрограммой, различие прописных и строчных букв и именах.

Параметр hModule передает описатель библиотечного модуля, возможно, нужный ему для дальнейшего использования.

Параметр fCall указывает на тип (причину) обращения к точке входа:

– DLL\_PROCESS\_ATTACH – первичное включение модуля в адресное пространство процесса. Модуль может создать необходимые локальные структуры данных. В Win16 реакция на данное обращение была практически необходима.

– DLL\_THREAD\_ATTACH – создание головным процессом нового потока.

– DLL\_THREAD\_DETACH – завершение одного из потоков.

– DLL\_PROCESS\_DETACH – выгрузка модуля из адресного пространства процесса.

Среда C++ Builder именует эту функцию DllEntryPoint, формат и действие не отличаются. Среда Delphi использует несколько иной подход с предопределенными именами и секциями initialization и finalization.

Как было отмечено, описание статически подключаемых библиотек зависит от конкретной среды программирования. Так, Visual C++ 6.0 предусматривает следующую технологию. В проекте библиотеки экспортируемые символы объявляются с модификатором \_\_declspec(dllexport). В проекте головной программы импортируемые символы объявляются с модификатором \_\_declspec(dllimport), кроме того, в этот проект должен быть включен .LIB-файл, сгенерированный вместе с библиотечным модулем. Для удобства (среда делает это самостоятельно при создании проекта) модификаторы оформляются в макроподстановки внутри директив условной компиляции в заголовочном файле с декларациями, и этот файл включается в оба проекта (библиотека и головная программа). Символами могут быть переменные, символы и классы. Обращаться к ним можно в любой точке головной программы без иных подготовительных действий. Отметим, что .H-файл содержит информацию для компилятора, а .LIB-файл – для компоновщика (подобно обычному .OBJ), поэтому их ошибки будут проявляться на соответствующих стадиях.

Недостатком технологии является несоответствие *реальных* имен экспортируемых символов объявленным в исходных текстах программ: компилятор снабжает их «довесками», несущими информацию о типе объекта, вследствие чего воспользоваться данной библиотекой из программы, созданной другим компилятором, практически невозможно. Аналогично не удается и подключить “чужеродную” библиотеку. Возможно, избавиться от “дополнения” имен можно настройками опций и предопределяемых средой директив, но наиболее радикальным приемом является явное стандартное описание символов модуля. В средствах разработки на языке С для этого традиционно служит .DEF-файл (текстовый формат с ключевыми словами), в котором создается секция EXPORTS, где перечисляются все экспортируемые имена. Библиотечный модуль пригоден как для динамического, так и для статического импорта, если только применяемый компилятор “целевого” проекта будет придерживаться стандартных соглашений об именах. Данные неспецифических типов проблем обычно не вызывают; функции объявляются с модификатором \_\_stdcall; экспорт и импорт классов сильно затруднен, так как требует согласования их описаний и/или трудоемкого явного импорта всех методов.

#### 13.3. Mixed Assembly – взаимодействие .NET и native Win 32 кода

Взаимодействие .NET и Native Win 32 – mixed-сборки

### Тема 14. Системный реестр и журналирование Win 32

#### 14.1. Системный реестр Windows

Реестр (Registry) – специальная системная база данных, в которой приложения и операционная система могут сохранять информацию о конфигурации.

Системный реестр служит для осуществления ряда функций:

– хранение конфигураций оборудования и сведений об устройствах «Plug-and-Play»;

– хранение списка драйверов и их параметров;

– описания программных интерфейсов (например, интерфейсов COM-серверов);

– таблица ассоциаций файлов данных;

– хранение конфигурации и значений параметров программ;

– обслуживание различных административных программ, например, панели управления (Control Panel).

Реестр имеет иерархическую древовидную структуру. Узлы дерева называются ключами (key). Каждый ключ может содержать любое количество подключей (sub-key) и значений (values), причем и те, и другие организованы в виде неупорядоченных списков, элементам которых присвоены индексы, начиная с нуля. Различие между ключами и подключами в действительности условно, фактически все ключи являются подключами различного уровня нескольких предопределенных ключей. Значения ключей могут быть строко­выми, двоич­ными и числовыми; также ключ может быть ссылкой на другой ключ. Каждый ключ идентифицируется его именем, уни­кальным относительно вышестоящего ключа; открытым ключам присва­иваются описатели (handle) HKEY. Значения идентифици­руются именами и индексами в списке.

Данные в системном реестре хранятся в двоичном виде. Для работы с ними приложения должны использовать специальные системные функции. Различают hive (букв. «улей») ключей, то есть их двоичный образ в структурах в памяти, называемых собственно реестром, и файлы данных реестра.

Для работы с ключом приложение должно открыть его. При открытии ключа необходимо указать открытый ранее ключ в качестве вышестоящего. Система всегда предоставляет четыре предопределенных (predefined) ключа верхнего уровня, которые считаются открытыми всегда и могут использоваться как точки входа в реестр: HKEY\_LOCAL\_MACHINE – описание известных на этот момент конфигураций компьютера; HKEY\_CLASSES\_ROOT – описание текущей конфигурации машины, ссылка на одну из конфигураций HKEY\_LOCAL\_MACHINE); HKEY\_USERS – описание всех имеющихся пользо­вателей; HKEY\_CURRENT\_USER – описание текущего пользователя, ссылка на одного из пользователей HKEY\_USERS; HKEY\_CURRENT\_CONFIG – текущий подключ Config (ссылка) ключа HKEY\_LOCAL\_MACHINE;

Система предоставляет ряд функций для доступа к реестру. Перечислим некоторые из них:

– RegCreateKey(), RegCreateKeyEx() – создание нового или открытие су­щест­ву­ющего ключа в реестре; создаваемый или открываемый ключ обяза­тельно должен быть подключом уже открытого ключа;

– RegOpenKey(), RegOpenKeyEx() – только открытие существующего клю­ча, в остальном аналогичны предыдущим;

– RegCloseKey() – закрытие открытого ключа;

– RegDeleteKey() – удаление ключа;

– RegFlushKey() – выгрузка содержимого ключа в соответствующий файл реестра;

– RegSaveKey(), RegLoadKey() – выгрузка содержимого ключа в отдель­ный файл и загрузка из него;

– RegEnumKey(), RegEnumKeyEx() – получение списка подключей;

– RegEnumValue() – получение списка значений;

– RegQueryValue(), RegQueryValueEx() – чтение значения по его имени;

– RegQueryMultipleValues() – чтение нескольких значений.

Ввиду относительной сложности и громоздкости обращений к реестру можно реализовать собственные «оберточные» функции для доступа по именам и «путям» реестре, аналогично работе с файловой системой.

На пользовательском уровне для работы с реестром служит утилита regedit, поддерживающая как интерактивный, так и командный (выполнение пакетных файлов) режимы.

#### 14.2. Журналирование, журналы Windows NT

Цели и задачи журналирования (протоколирования) событий.

Реализация журналирования.

Объекты, структуры данных, типы, сипользуемые в журналах.

### Тема 15. Подсистема безопасности

Понятие о безопасности в вычислительных системах.

Общие сведения о подсистеме безопасности.

Угрозы, защищаемые объекты, участники подсистемы безопасности.

Задачи подсистемы безопасности.

Реализация подсистемы безопасности в ОС семейства Win NT.

# Практический раздел

## Контрольные работы

### Контрольная работа №1

Принципы управления памятью, управление памятью в различных системах

#### Указания по выбору варианта

Цель работы – более детальное ознакомление с методами и алгоритмами управления памятью, закрепление полученных знаний, проверка качества усвоения материала. Задание предполагает описание и примеры различных механизмов управления памятью.

Контрольная работа должна быть оформлена в соответствии с общеустановленными нормами и правилами, предъявляемыми к выполнению контрольных работ. При выполнении задания необходимо пользоваться несколькими источниками.

Контрольная работа не предполагает строгого разделения на теоретическую и практическую части. Материал, соответствующий заданию, должен содержать как описания методов, принципов, механизмов, объектов, функций и т.д., так и примеры, необходимые для иллюстрации излагаемого.

Выбор вариантов контрольной работы осуществляется студентом самостоятельно на основании номера зачетной книжки (последние цифры, после номера группы). Если порядковый номер превышает количество вариантов, для выбора номера варианта используется остаток от деления порядкового номера на количество вариантов. В отдельных случаях задание может быть сформулировано преподавателем индивидуально.

Таблица 1 – Варианты заданий контрольной работы № 1

|  |  |
| --- | --- |
| № п/п | Задание |
| 1. | Память как ресурс вычислительной системы. Подсистема памяти. |
| 2. | Адресное пространство. Виды адресов. |
| 3. | Задачи управления памятью. Особенности их решения в различных ОС. |
| 4. | Методы распределения памяти без создания виртуального адресного пространства. |
| 5. | Виртуальное адресное пространство. Средства поддержки виртуальной памяти. |
| 6. | Методы распределения памяти на основе виртуального адресного пространства |
| 7. | Средства управления памятью однозадачных ОС (на примере MS-DOS) |
| 8. | Подсистема управления памятью многозадачной ОС (на примере MS Windows) |
| 9. | Группы функций Win 32 API для управления памятью. |
| 10. | Критерии и факторы эффективности подсистемы памяти. |

### Контрольная работа №2

Управление процессами и потоками и их взаимодействие

#### Указания по выбору варианта

Цель работы – подробное изучение средств управления процессами и потоками как объектами Windows, а также базовых средств межпроцессного взаимодействия в Windows. Задание предполагает описание и примеры соответствующих системных объектов и функций Win 32 API.

Контрольная работа должна быть оформлена в соответствии с общеустановленными нормами и правилами, предъявляемыми к выполнению контрольных работ. При выполнении задания необходимо пользоваться несколькими источниками.

Контрольная работа не предполагает строгого разделения на теоретическую и практическую части. Материал, соответствующий заданию, должен содержать как описания методов, принципов, механизмов, объектов, функций и т.д., так и примеры, необходимые для иллюстрации излагаемого.

Выбор вариантов контрольной работы осуществляется студентом самостоятельно на основании номера зачетной книжки (последние цифры, после номера группы). Если порядковый номер превышает количество вариантов, для выбора номера варианта используется остаток от деления порядкового номера на количество вариантов. В отдельных случаях задание может быть сформулировано преподавателем индивидуально.

Таблица 2 – Варианты заданий контрольной работы № 2

|  |  |
| --- | --- |
| № п/п | Задание |
| 1. | Многозадачность и многопоточность, их роль в программировании, поддержка ОС |
| 2. | Вычислительный процесс как системный объект – характеристика, атрибуты, адресное пространство. |
| 3. | Поток как системный объект – характеристика, атрибуты, использование. |
| 4. | Состояния и жизненный цикл процессов и потоков. |
| 5. | Организация вычислительного процесса в однозадачных ОС (на примере MS-DOS) |
| 6. | Взаимодействие процессов и потоков – основные задачи, проблемы, решения. |
| 7. | Управление процессами (базовый уровень) – порождение, завершение, получение доступа |
| 8. | Управление потоками (базовый уровень) – порождение, завершение, получение доступа |
| 9. | Взаимодействие процессов (базовый уровень) – использование файлов, каналов, разделяемой памяти. |
| 10. | Синхронизация и обеспечение взаимодействия потоков (базовый уровень) |

## Индивидуальные практические работы

### Индивидуальная практическая работа №1

Оконное приложение Win 32

#### Указания по выбору варианта

Цель данной работы – практическое изучение построения и функционирования типичных оконных приложений, элементов управления, сообщений, средств графического вывода, используемых при этом основных функций Win32 API, а также ознакомление со средой программирования. В рамках заданий предполагается создание каркаса приложения, использование сообщений, оконных объектов, взаимодействие с графической подсистемой, реализованные на уровне системных вызовов Win 32 API.

Индивидуальная практическая работа предполагает защиту выполненного задания, а также ответы на теоретические вопросы, демонстрирующие владение тематикой.

Индивидуальная практическая работа оформляется в соответствии с общеустановленными нормами и правилами.

Выбор вариантов индивидуальной практической работы осуществляется студентом самостоятельно на основании номера зачетной книжки (последние цифры, после номера группы). Если порядковый номер превышает количество вариантов, для выбора номера варианта используется остаток от деления порядкового номера на количество вариантов. В отдельных случаях задание может быть сформулировано преподавателем индивидуально.

#### Теоретическая часть (вопросы)

1. Стандартная структура оконного приложения, его основные функции.

2. Окно как системный объект. Функция создания нового окна и ее параметры.

3. Оконный класс, виды оконных классов, создание оконного класса.

4. Пользовательские (настраиваеые) атрибуты окон.

5. Функции UpdateWindow() и ShowWindow().

6. Сообщение Windows (Win Message), его основные параметры. Очередь сообще­ний. Способы отправки сообщений.

7. Цикл приема и обработки сообщений, функции GetMessage(), TranslateMessage(), DispatchMessage(), завершение цикла.

8. Оконная процедура – назначение, структура, параметры, способы вызова.

9. Перерисовка окна, последовательность формирования изображения, сообщение WM\_PAINT, функции GetWindowRect(), InvalidateRect().

10. Создание и использование элементов управления, меню. Сообщение WM\_COMMAND.

11. Таймеры. Создание, удаление, использование таймера (сообщение WM\_TIMER).

12. Элементов управления – виды, создание, использование. Элементы управления Edit, Button, ListBox, ComboBox и т.д.

13. Обработка событий элемента управления (на примере элемента Button).

14. Группировка элементов управления, доступ к группам, обработка сообщений (на примере RadioButton, CheckBox).

15. Получение доступа к данным элемента управления (на примере текста элемента Edit).

16. Элементы управления со сложными структурами данных (на примере элементов ListBox, ComboBox и т.п.).

17. Контексты графических устройств.

18. Настройка графического контекста – система координат, единицы измерения, управление цветом и т.д.

19. Объекты (инструменты) графической подсистемы. Порядок применения инструментов для формирования изображения.

20. Активные инструменты Pen, Brush, Font, их создание и параметры.

21. Векторная графика, графические примитивы. Функции графических примитивов, их параметры.

22. Средства растровой графики GDI.

23. Самоотрисовывающиеся (ownerdraw) элементы управления (на примере кнопки) – формирование пользовательского изображения, обработка событий.

24. Использование теневых графических контекстов. Анимация изображений.

25. Сообщения как средство интерфейса между окнами. Использование сообщений WM\_USER.

26. Регистрация и использование пользовательских сообщений.

#### Практическая часть (варианты заданий)

##### Вариант задания №1.

Написать программу, которая создает окно с собственными курсором и иконкой, а также с дополнительным запросом перед завершением программы. При отказе от завершения приложение продолжает выполняться. В окне отображаются текстовые сообщения о попытках завершения.

##### Вариант задания №2.

Реализация бегущей строки с управлением из меню: в главное меню программы добавить подменю с двумя пунктами Start и Stop, при нажатии на пункт меню Start в окне должна появиться движущаяся надпись, после этого пункта меню Stop должен приостанавли­вать движение надписи, а пункта Start – возобновлять его (используется таймер и обработчик сообщения WM\_PAINT).

##### Вариант задания №3.

Реализация бегущей строки с управлением элементами управления Button: создаются кнопки Start и Stop, при нажатии на кнопку Start в окне должна появиться движущаяся надпись, после этого кнопка Stop должна приостанавли­вать движение надписи, а кнопка Start – возобновлять его (используется таймер и обработчик сообщения WM\_PAINT).

##### Вариант задания №4.

Программа с взаимодействующими элементами управления: окно программы содержит два элемента ListBox, один Edit и четыре Button («Add», «Clear», «ToRight» и «Delete»). При нажатии на кнопку «Add» текст из Edit должен добавляться в первый ListBox, если такого текста там еще нет (необходимо выполнить проверку). Нажатие кнопки «Clear» очищает оба ListBox-а. Нажатие кнопки «ToRight» копирует выделенную строку из первого ListBox во второй (если там еще нет такой строки). Нажатие кнопки «Delete» удаляет выделенные строки в каждом из ListBox-ов.

##### Вариант задания №5.

Программа со сложным фоном главного окна: окно содержит изображение, сформированное из примитивов (не менее 10-15 экземпляров). Необходимо предусмотреть корректную перерисовку изображения, а также масштабирование или отсечение его при изменении размеров окна.

##### Вариант задания №6.

Программа со сложным фоном главного окна: окно содержит изображение (сформированное из примитивов или в виде загружаемого растра). Необходимо предусмотреть корректную перерисовку изображения, а также его отсечение при изменении размеров окна.

##### Вариант задания №7.

Использование фигурного текста: окно программы содержит элемент управления Edit, с помощь которого задается текстовая строка. Эта строка прорисовывается в окне с дополнительным эффектами: вывод вдоль кривой линии, переменный размер символа, изменяющийся угол наклона и т.п. Необходимо предусмотреть корректную перерисовку изображения, а также масштабирование или отсечение его при изменении размеров окна.

##### Вариант задания №8.

Использование самоотрисовывающихся кнопок (OWNERDRAW): окно приложения снабжается не менее чем двумя кнопками, для каждой из которых задается собственный рисунок. При нажатии кнопка выводит в окно свой текущий рисунок (укрупненный).

##### Вариант задания №9.

Анимированные изображения: реализовать поочередное отображение нескольких заранее заготовленных изображений небольшого размера, соответствующих стадиям движения объекта. При нажатии кнопки «мыши» начинается движение объекта из текущей позиции в позицию указателя «мыши» с поочередной сменой изображений для имитации фаз дви­жения.

##### Вариант задания №10.

Передача сообщений между приложениями: пара программ, каждая из которых создает окно, содержащее группу RadioButton. Пусть состояние соответствует выбору цвета фона окна. Обеспечить синхронное изменение состояние группы и цвета фона обоих окон при переключении RadioButton в любом из них

### Индивидуальная практическая работа №2

Файловая система

#### Указания по выбору варианта

Цель данной работы – углубленное теоретическое и практическое изучение файловых систем (на примере семейства FAT) и низкоуровневого доступа к их структурам. Прямой доступ низкоуровневым структурам дисковой памяти рассчитан в первую очередь на выполнение в среде однозадачной ОС, однако возможна реализация и в среде Windows, однако это потребует использования соответствующей поддержки такого доступа (отображение на псевдофайлы).

Индивидуальная практическая работа предполагает защиту выполненного задания, а также ответы на теоретические вопросы, демонстрирующие владение тематикой.

Индивидуальная практическая работа оформляется в соответствии с общеустановленными нормами и правилами.

Выбор вариантов индивидуальной практической работы осуществляется студентом самостоятельно на основании номера зачетной книжки (последние цифры, после номера группы). Если порядковый номер превышает количество вариантов, для выбора номера варианта используется остаток от деления порядкового номера на количество вариантов. В отдельных случаях задание может быть сформулировано преподавателем индивидуально.

#### Теоретическая часть (вопросы)

1. Понятие файловой системы.

2. Структура файловой системы FAT.

3. Разновидности FAT.

4. Загрузочная запись.

5. Организация каталога.

6. Организация FAT-таблицы.

7. Организация покластерного доступа к файлу.

#### Практическая часть (варианты заданий)

##### Вариант задания №1.

С клавиатуры вводится имя файла. Необходимо найти запись для данного файла в корневом каталоге; составить карту его размещения (список кластеров, в которых записано содержимое этого файла); покластерно прочитать его. Содержимое файла вы­вести на экран или (по выбору) сохранить в файл.

##### Вариант задания №2.

При помощи анализа цепочек выяснить, какой из файлов корневого каталога занимает наибольшее число кластеров. Имя этого файла показать на экране. (Аналогичные задания могут быть сформулированы для других критериев выбора.)

##### Вариант задания №3.

Формирование списков файлов: фрагментированных, нефрагментированных, занимающих строго один кластер. (Файл считается нефрагментированным, если кластера, которые для него выделены, следуют подряд по порядку их номеров.)

##### Вариант задания №4.

Информация о фрагментации заданного файла: общее количество выделенных кластеров, количество цепочек кластеров, средняя длина цепочки, самая длинная цепочка кластеров; процент нефрагментированного (размещенного в наибольшей непрерывной цепочке) содержимого файла.

##### Вариант задания №5.

С клавиатуры вводится номер кластера. Необходимо выяснить, какому из файлов выделен данный кластер, и вывести имя этого файла.

##### Вариант задания №6.

Оценка свободного пространства на диске и анализ его фрагментации: общее количество свободных кластеров, количество непрерывных последовательностей свободных кластеров, наибольшая непрерывная последовательность, и так далее.

##### Вариант задания №7.

Оценка потерь дискового пространства вследствие кластерного выделения: размер кластера, количество файлов, суммарный объем файлов, суммарный объем выделенных кластеров, «потерянное» пространство как абсолютное значение и как процент от использованного.

Потери дискового пространства возникают, когда размер файла не кратен размеру кластера, вследствие чего последний выделенный ему кластер остается частично неиспользованным. В среднем можно считать, что на каждом файле теряется около половины кластера. При большом количестве файлов малого размера такие потери могут достигать существенной величины.

##### Вариант задания №8.

Информация о фрагментации заданного раздела (повышенной сложности): сводная информация по файлам всех каталогов этого раздела. Требуется обход дерева каталогов и анализ всех файлов в них.

##### Вариант задания №9.

Дефрагментация заданного файла или группы файлов (повышенной сложности).

Для дефрагментации выделенные файлу кластеры переразмещается таким образом, чтобы он занимал одну непрерывную цепочку из подряд выделенных кластеров. При выполнении задания можно предполагать, что суммарный объем дефрагментируемых файлов достаточно мал по сравнению с доступным свободным пространством на диске.

##### Вариант задания №10.

Восстановление удаленного файла (повышенной сложности).

Файл при удалении его средствами MS-DOS не уничтожается полностью. Соответствующая ему запись в каталоге помечается как «удаленная» (символ E5h в начале имени файла), и все принадлежавшие файлу кластеры помечаются в таблице FAT как свободные. Ссылка на первый кластер и содержимое этих кластеров не затрагивается, поэтому сохраняется теоретическая возможность восстановления файла. Однако успешность восстановления не гарантируется, так как информация о порядке следования кластеров теряется, а сами они могут быть присоединены к другим файлам.