

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ В **WINDOWS**

- Синхронизация потоков
- Каналы передачи данных и почтовые ящики
- Виртуальная память и файлы
- Асинхронная обработка данных
- Безопасность доступа к объектам

**Наиболее
полное
руководство**



В ПОДЛИННИКЕ [®]

УДК 681.3.06
ББК 32.973.26-018.1
П41

Побегайло А. П.

П41 Системное программирование в Windows. — СПб.: БХВ-Петербург, 2006. — 1056 с.: ил.

ISBN 5-94157-792-3

Подробно рассматриваются вопросы системного программирования с использованием интерфейса Win32 API. Описываются управление потоками и процессами, включая их диспетчеризацию; синхронизация потоков; передача данных между процессами, с использованием анонимных и именованных каналов, а также почтовых ящиков; структурная обработка исключений; управление виртуальной памятью; управление файлами и каталогами; асинхронная обработка данных; создание динамически подключаемых библиотек; разработка сервисов. Отдельная часть книги посвящена управлению безопасностью объектов в Windows. Каждая тема снабжена практическими примерами использования функций Win32 API, которые представлены работающими листингами. Это позволяет использовать книгу в качестве пособия по системному программированию или справочника для системного программиста. Прилагаемый компакт-диск содержит листинги и проекты всех программ, рассмотренных в книге.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Игорь Шишигин</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Андрей Смышляев</i>
Компьютерная верстка	<i>Натальи Караваевой</i>
Корректор	<i>Наталья Першакова</i>
Оформление обложки	<i>Елены Беляевой</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 15.01.06.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 85,14.

Тираж 3000 экз. Заказ №

"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Отпечатано с готовых диапозитивов

в ОАО "Техническая книга"

190005, Санкт-Петербург, Измайловский пр., 29.

ISBN 5-94157-792-3

© Побегайло А. П., 2006

© Оформление, издательство "БХВ-Петербург", 2006

Оглавление

Предисловие	15
Глава 1. Операционные системы и их интерфейсы	19
1.1. Назначение операционной системы.....	19
1.2. Типы операционных систем.....	19
1.3. Интерфейс программирования приложений Win32 API.....	21
1.4. Типы данных в Win32 API.....	22
1.5. Объекты и их дескрипторы в Windows.....	24
ЧАСТЬ I. УПРАВЛЕНИЕ ПОТОКАМИ И ПРОЦЕССАМИ	27
Глава 2. Потоки и процессы	29
2.1. Определение потока.....	29
2.2. Контекст потока.....	31
2.3. Состояния потока.....	33
2.4. Диспетчеризация и планирование потоков.....	37
2.5. Определение процесса.....	40
Глава 3. Потоки в Windows	41
3.1. Определение потока.....	41
3.2. Создание потоков.....	42
3.3. Завершение потоков.....	47
3.4. Приостановка и возобновление потоков.....	49
3.5. Псевдодескрипторы потоков.....	52
3.6. Обработка ошибок в Windows.....	53
Глава 4. Процессы в Windows	58
4.1. Определение процесса.....	58
4.2. Создание процессов.....	58

4.3. Завершение процессов.....	64
4.4. Наследование дескрипторов	67
4.5. Дублирование дескрипторов.....	75
4.6. Псевдодескрипторы процессов	81
4.7. Обслуживание потоков.....	82
4.8. Динамическое изменение приоритетов потоков.....	88

ЧАСТЬ II. СИНХРОНИЗАЦИЯ ПОТОКОВ И ПРОЦЕССОВ.....93

Глава 5. Синхронизация.....95

5.1. Непрерывные действия и команды.....	95
5.2. Определение синхронизации.....	96
5.3. Программная реализация синхронизации	97
5.4. Аппаратная реализация синхронизации.....	101
5.5. Примитивы синхронизации.....	104

Глава 6. Синхронизация потоков в Windows..... 109

6.1. Критические секции	109
6.2. Объекты синхронизации и функции ожидания	115
6.3. Мьютексы.....	121
6.4. События.....	128
6.5. Семафоры.....	137

Глава 7. Взаимоисключающий доступ к переменным..... 143

7.1. Атомарные операции	143
7.2. Замена значения переменной.....	144
7.3. Условная замена значения переменной	146
7.4. Инкремент и декремент переменной	148
7.5. Изменение значения переменной.....	150

Глава 8. Тупики..... 153

8.1. Определение тупиков	153
8.2. Классификация системных ресурсов.....	154
8.3. Обнаружение тупиков.....	156
8.4. Восстановление заблокированного процесса	158
8.5. Предотвращение тупиков.....	160
8.6. Безопасное завершение потоков в Windows	161

ЧАСТЬ III. ПРОГРАММИРОВАНИЕ КОНСОЛЬНЫХ ПРИЛОЖЕНИЙ	165
Глава 9. Структура консольного приложения	167
9.1. Структура консоли	167
9.2. Входной буфер консоли	167
9.3. Буфер экрана	171
Глава 10. Работа с консолью	172
10.1. Создание консоли	172
10.2. Освобождение консоли.....	177
10.3. Стандартные дескрипторы ввода-вывода	178
Глава 11. Работа с окном консоли.....	180
11.1. Получение дескриптора окна консоли	180
11.2. Получение и изменение заголовка консоли	181
11.3. Определение максимального размера окна	183
11.4. Установка координат окна	184
Глава 12. Работа с буфером экрана.....	188
12.1. Создание и активация буфера экрана.....	188
12.2. Определение и установка параметров буфера экрана	191
12.3. Функции для работы с курсором	194
12.4. Чтение и установка атрибутов консоли.....	197
Глава 13. Ввод-вывод на консоль	203
13.1. Ввод-вывод высокого уровня.....	203
13.2. Ввод низкого уровня.....	207
13.3. Вывод низкого уровня	215
13.4. Режимы ввода-вывода консоли	225
13.5. Прокрутка буфера экрана.....	229
ЧАСТЬ IV. ОБМЕН ДАННЫМИ МЕЖДУ ПАРАЛЛЕЛЬНЫМИ ПРОЦЕССАМИ	235
Глава 14. Передача данных.....	237
14.1. Способы передачи данных между процессами.....	237
14.2. Связи между процессами	239
14.3. Передача сообщений.....	240

14.4. Синхронный и асинхронный обмен данными	241
14.5. Буферизация	242
Глава 15. Работа с анонимными каналами в Windows.....	243
15.1. Анонимные каналы.....	243
15.2. Создание анонимных каналов.....	244
15.3. Соединение клиентов с анонимным каналом.....	245
15.4. Обмен данными по анонимному каналу.....	246
15.5. Примеры работы с анонимными каналами	247
15.6. Перенаправление стандартного ввода-вывода.....	257
Глава 16. Работа с именованными каналами в Windows.....	265
16.1. Именованные каналы	265
16.2. Создание именованных каналов	266
16.3. Соединение сервера с клиентом	268
16.4. Соединение клиентов с именованным каналом	269
16.5. Обмен данными по именованному каналу	272
16.6. Копирование данных из именованного канала.....	285
16.7. Передача транзакций по именованному каналу.....	289
16.8. Определение и изменение состояния именованного канала.....	295
16.9. Получение информации об именованном канале.....	303
Глава 17. Работа с почтовыми ящиками в Windows	307
17.1. Концепция почтовых ящиков.....	307
17.2. Создание почтовых ящиков	308
17.3. Соединение клиентов с почтовым ящиком	309
17.4. Обмен данными через почтовый ящик	311
17.5. Получение информации о почтовом ящике	315
17.6. Изменение времени ожидания сообщения.....	321
ЧАСТЬ V. СТРУКТУРНАЯ ОБРАБОТКА ИСКЛЮЧЕНИЙ	325
Глава 18. Фреймовая обработка исключений	327
18.1. Исключения и их обработчики	327
18.2. Получение кода исключения	330
18.3. Функции фильтра.....	332
18.4. Получение информации об исключении	334
18.5. Генерация программных исключений	337
18.6. Необработанные исключения	340
18.7. Обработка исключений с плавающей точкой.....	342

18.8. Обработка вложенных исключений	344
18.9. Передача управления и выход из фрейма	346
18.10. Встраивание SEH в механизм исключений C++	348
Глава 19. Финальная обработка исключений	351
19.1. Финальные блоки фрейма	351
19.2. Проверка завершения фрейма	353
19.3. Обработка вложенных финальных блоков	354
ЧАСТЬ VI. РАБОТА С ВИРТУАЛЬНОЙ ПАМЯТЬЮ	357
Глава 20. Виртуальная память.....	359
20.1. Концепция виртуальной памяти	359
20.2. Организация виртуальной памяти.....	360
20.3. Алгоритмы замещения страниц.....	362
20.4. Рабочее множество процесса	363
20.5. Организация виртуальной памяти в Windows.....	363
Глава 21. Работа с виртуальной памятью в Windows	367
21.1. Состояния виртуальной памяти процесса.....	367
21.2. Резервирование, распределение и освобождение виртуальной памяти	368
21.3. Блокирование виртуальных страниц в реальной памяти	376
21.4. Изменение атрибутов доступа к виртуальной странице.....	378
21.5. Управление рабочим множеством страниц процесса	380
21.6. Инициализация и копирование блоков виртуальной памяти	383
21.7. Определение состояния памяти	385
21.8. Работа с виртуальной памятью в другом процессе	388
Глава 22. Работа с кучей в Windows	393
22.1. Создание и удаление кучи.....	393
22.2. Распределение и освобождение памяти из кучи	395
22.3. Перераспределение памяти из кучи.....	401
22.4. Блокирование и разблокирование кучи	403
22.5. Проверка состояния кучи.....	406
22.6. Уплотнение кучи	411

ЧАСТЬ VII. УПРАВЛЕНИЕ ФАЙЛАМИ 415**Глава 23. Общие концепции 417**

23.1. Накопители на жестких магнитных дисках	417
23.2. Секторы и кластеры.....	418
23.3. Форматирование дисков.....	419
23.4. Функции файловой системы	420
23.5. Каталоги	420
23.6. Буферизация ввода-вывода	421
23.7. Кэширование ввода-вывода	421

Глава 24. Работа с файлами в Windows 423

24.1. Именованние файлов в Windows	423
24.2. Создание и открытие файлов	424
24.3. Заккрытие и удаление файлов.....	427
24.4. Запись данных в файл	428
24.5. Освобождение буферов файла	430
24.6. Чтение данных из файла	433
24.7. Копирование файла	435
24.8. Перемещение файла	437
24.9. Замещение файла	438
24.10. Работа с указателем позиции файла	440
24.11. Определение и изменение атрибутов файла	446
24.12. Определение и изменение размеров файла.....	449
24.13. Блокирование файла	455
24.14. Получение информации о файле	459

Глава 25. Работа с каталогами (папками) в Windows 468

25.1. Создание каталога	468
25.2. Поиск файлов в каталоге.....	470
25.3. Удаление каталога	473
25.4. Перемещение каталога	476
25.5. Определение и установка текущего каталога.....	477
25.6. Наблюдение за изменениями в каталоге.....	479

ЧАСТЬ VIII. АСИНХРОННАЯ ОБРАБОТКА ДАННЫХ 483**Глава 26. Асинхронный вызов процедур 485**

26.1. Механизм асинхронного вызова процедур	485
26.2. Установка асинхронных процедур	486

26.3. Приостановка потока.....	487
26.4. Ожидание события.....	489
26.5. Оповещение и ожидание события	494

Глава 27. Асинхронный доступ к данным 499

27.1. Концепция асинхронного ввода-вывода	499
27.2. Асинхронная запись данных.....	500
27.3. Асинхронное чтение данных	506
27.4. Блокирование файлов.....	511
27.5. Определение состояния асинхронной операции ввода-вывода	518
27.6. Отмена асинхронной операции ввода-вывода.....	522
27.7. Процедуры завершения ввода-вывода	528
27.8. Асинхронная запись данных с процедурами завершения.....	529
27.9. Асинхронное чтение данных с процедурами завершения.....	532

Глава 28. Порты завершения..... 536

28.1. Концепция порта завершения	536
28.2. Создание порта завершения.....	537
28.3. Получение пакета из порта завершения.....	538
28.4. Посылка пакета в порт завершения.....	539

Глава 29. Работа с ожидающим таймером 544

29.1. Ожидающий таймер	544
29.2. Создание ожидающего таймера.....	545
29.3. Установка ожидающего таймера	546
29.4. Отмена ожидающего таймера	549
29.5. Открытие существующего ожидающего таймера	552
29.6. Процедуры завершения ожидания	555

ЧАСТЬ IX. ДИНАМИЧЕСКИ ПОДКЛЮЧАЕМЫЕ БИБЛИОТЕКИ..... 559

Глава 30. Отображение файлов в память 561

30.1. Концепция механизма отображения файлов в память	561
30.2. Создание и открытие объекта, отображающего файл.....	562
30.3. Отображение файла в память	564
30.4. Обмен данными между процессами через отображаемый в память файл.....	569
30.5. Сброс вида в файл.....	573

Глава 31. Динамически подключаемые библиотеки	578
31.1. Концепция динамически подключаемых библиотек	578
31.2. Создание DLL.....	579
31.3. Динамическая загрузка и отключение DLL.....	581
31.4. Использование DLL.....	584
31.5. Использование файла определений.....	588
31.6. Статическая загрузка DLL	592
Глава 32. Локальная память потока	594
32.1. Динамическая локальная память потока.....	594
32.2. Распределение и освобождение локальной памяти потока	595
32.3. Запись и чтение из локальной памяти потока	595
32.4. Статическая локальная память потока	602
ЧАСТЬ X. РАЗРАБОТКА СЕРВИСОВ В WINDOWS	605
Глава 33. Сервисы в Windows.....	607
33.1. Концепция сервиса	607
33.2. Структура сервиса	608
33.3. Организация функции <i>main</i>	609
33.4. Организация функции <i>ServiceMain</i>	611
33.5. Организация обработчика управляющих команд.....	617
Глава 34. Работа с сервисами в Windows	620
34.1. Открытие доступа к базе данных сервисов	620
34.2. Установка сервиса	621
34.3. Открытие доступа к сервису	627
34.4. Запуск сервиса	627
34.5. Определение и изменение состояния сервиса.....	630
34.6. Определение и изменение конфигурации сервиса	634
34.7. Определение имени сервиса	641
34.8. Управление сервисом.....	646
34.9. Удаление сервисов	649
34.10. Блокирование базы данных сервисов	653
ЧАСТЬ XI. УПРАВЛЕНИЕ БЕЗОПАСНОСТЬЮ В WINDOWS	659
Глава 35. Система информационной безопасности	661
35.1. Контроль доступа к ресурсам	661
35.2. Политика безопасности.....	662

35.3. Модель безопасности.....	663
35.4. Дискреционная политика безопасности.....	664
35.5. Дискреционная модель безопасности.....	665
35.6. Реализация дискреционной модели безопасности.....	668
Глава 36. Управление безопасностью в Windows	671
36.1. Модель безопасности в Windows.....	671
36.2. Учетные записи	672
36.3. Домены	674
36.4. Группы.....	676
36.5. Идентификаторы безопасности.....	678
36.6. Дескрипторы безопасности.....	682
36.7. Списки управления доступом ACL.....	683
36.8. Маркеры доступа.....	687
36.9. Создание новых объектов.....	693
36.10. Контроль доступа к охраняемому объекту	694
36.11. Аудит доступа к охраняемому объекту.....	696
36.12. Структура системы безопасности.....	696
Глава 37. Управление пользователями	699
37.1. Создание учетной записи пользователя	699
37.2. Получение информации о пользователе	704
37.3. Перечисление пользователей.....	706
37.4. Перечисление групп, которым принадлежит пользователь	710
37.5. Изменение учетной записи пользователя	715
37.6. Изменение пароля пользователя	719
37.7. Удаление учетной записи пользователя	721
Глава 38. Управление группами.....	724
38.1. Создание локальной группы.....	724
38.2. Получение информации о локальной группе	727
38.3. Перечисление локальных групп	729
38.4. Изменение информации о локальной группе	732
38.5. Добавление членов локальной группы	736
38.6. Установка членов локальной группы.....	742
38.7. Перечисление членов локальной группы.....	745
38.8. Удаление членов локальной группы	748
38.9. Удаление локальной группы.....	754
Глава 39. Работа с идентификаторами безопасности.....	756
39.1. Структура идентификатора безопасности	756
39.2. Создание идентификатора безопасности	757

39.3. Определение учетной записи по идентификатору безопасности.....	764
39.4. Определение идентификатора безопасности по имени учетной записи	769
39.5. Получение характеристик идентификатора безопасности.....	773
39.6. Копирование и сравнение идентификаторов безопасности	777
39.7. Строковое представление идентификатора безопасности	782

Глава 40. Работа с дескрипторами безопасности..... 788

40.1. Форматы дескрипторов безопасности	788
40.2. Создание нового дескриптора безопасности	791
40.3. Определение длины дескриптора безопасности.....	797
40.4. Получение дескриптора безопасности по имени объекта.....	802
40.5. Получение дескриптора безопасности по дескриптору объекта	806
40.6. Получение данных из дескриптора безопасности.....	810
40.7. Получение состояния управляющих флагов дескриптора безопасности	815
40.8. Изменение дескриптора безопасности по имени объекта	818
40.9. Изменение дескриптора безопасности по дескриптору объекта.....	823
40.10. Изменение состояния управляющих флагов дескриптора безопасности	827
40.11. Строковое представление дескрипторов безопасности	831

Глава 41. Работа со списками управления доступом на высоком уровне..... 840

41.1. Структура <i>TRUSTEE</i>	840
41.2. Инициализация структуры <i>TRUSTEE</i>	842
41.3. Структура <i>EXPLICIT_ACCESS</i>	846
41.4. Инициализация структуры <i>EXPLICIT_ACCESS</i>	849
41.5. Создание нового списка управления доступом	850
41.6. Модификация списка управления доступом	862
41.7. Получение элементов из списка управления доступом	870
41.8. Получение информации из структуры <i>TRUSTEE</i>	871
41.9. Получение прав доступа из списка управления доступом.....	874
41.10. Получение из списка управления доступом прав, которые подвергаются аудиту	878

Глава 42. Работа с привилегиями 885

42.1. Локальные идентификаторы привилегий.....	885
42.2. Инициализация локального идентификатора.....	887
42.3. Получение локального идентификатора привилегии	888
42.4. Получение имени привилегии.....	888
42.5. Получение имени привилегии для отображения	891

Глава 43. Работа с маркерами доступа	894
43.1. Открытие маркера доступа процесса	894
43.2. Открытие маркера доступа потока	896
43.3. Структуры, используемые для работы с маркером доступа	896
43.4. Получение информации из маркера доступа	900
43.5. Изменение информации в маркере доступа	908
43.6. Настройка привилегий	917
43.7. Настройка групп	918
43.8. Создание маркера ограниченного доступа	920
43.9. Дублирование маркеров доступа	927
43.10. Замещение маркеров доступа потока	929
43.11. Проверка идентификатора безопасности на принадлежность маркеру доступа	932
Глава 44. Работа со списками управления доступом на низком уровне	939
44.1. Структура списка управления доступом	939
44.2. Структура элемента списка управления доступом	940
44.3. Инициализация списка управления доступом	943
44.4. Проверка достоверности списка управления доступом	944
44.5. Добавление элементов в список управления доступом	945
44.6. Получение элементов из списка управления доступом	972
44.7. Удаление элементов из списка управления доступом	977
44.8. Получение информации о списке управления доступом	981
44.9. Установка версии списка управления доступом	985
44.10. Определение доступной памяти	986
Глава 45. Управление безопасностью объектов на низком уровне	987
45.1. Доступ к информации о владельце объекта	988
45.2. Доступ к информации о первичной группе владельца объекта	992
45.3. Доступ к списку DACL	997
45.4. Доступ к списку SACL	1004
45.5. Защита файлов и каталогов	1006
45.6. Защита объектов ядра	1016
45.7. Защита сервисов	1024
45.8. Защита ключей реестра	1031
45.9. Защита объектов пользователя	1037
Приложение. Описание компакт-диска	1045
Предметный указатель	1047

Глава 1



Операционные системы и их интерфейсы

1.1. Назначение операционной системы

Физическими или *аппаратными ресурсами компьютера* называются физические устройства, из которых состоит компьютер. К таким устройствам относятся центральный процессор, оперативная память, внешняя память, шины передачи данных и различные устройства ввода-вывода информации. *Логическими* или *информационными ресурсами компьютера* называются данные и программы, которые хранятся в памяти компьютера. Когда говорят обо всех ресурсах компьютера, включая как физические, так и логические ресурсы, то обычно используют термины *ресурсы компьютера* или *системные ресурсы*.

Для выполнения на компьютере какой-либо программы необходимо, чтобы она имела доступ к ресурсам компьютера. Этот доступ обеспечивает операционная система. Можно сказать, что *операционная система* — это комплекс программ, который обеспечивает доступ к ресурсам компьютера и управляет ими. Другими словами, операционная система — это администратор или менеджер ресурсов компьютера. Назначение операционной системы состоит в обеспечении пользователя программными средствами для использования ресурсов компьютера и эффективном разделении этих ресурсов между пользователями. Отсюда следует, что главными функциями операционной системы являются управление ресурсами компьютера и диспетчеризация или планирование этих ресурсов.

1.2. Типы операционных систем

Все программы, которые работают на компьютере под управлением операционной системы, называются *пользовательскими программами*. Совокупность пользовательских программ, которая предназначена для решения определенной задачи, называется *приложением*. Если операционная система одновременно

может исполнять только одну пользовательскую программу, то она называется *однопрограммной* или *однопользовательской*. Если же под управлением операционной системы могут одновременно выполняться несколько пользовательских программ, то такая операционная система называется *мультипрограммной* или *многопользовательской*.

В зависимости от назначения операционной системы и аппаратуры компьютера, на котором она работает, можно определить несколько типов операционных систем. Если операционная система может работать только на компьютере с одним процессором, то такая операционная система называется *однопроцессорной*. Если же операционная система может работать также и на компьютере, который содержит несколько процессоров, то такая операционная система называется *мультипроцессорной*.

Следует делать различие между операционными системами, которые предназначены для обработки информации под управлением пользователя, и операционными системами, которые предназначены для управления объектами при помощи компьютера в реальном времени без участия пользователя. Такими объектами могут быть, например, робот или самолет. Операционная система, предназначенная для работы в режиме реального времени, называется *операционной системой реального времени*. Главное отличие операционных систем реального времени заключается в их быстром реагировании на внешние события и надежности функционирования. Если пользователь, сидя у компьютера, будет только раздражен медленной или ненадежной работой операционной системы, то медленная или ненадежная работа операционной системы реального времени может вызвать поломку оборудования и аварию.

В дальнейшем будут рассматриваться только операционные системы фирмы Microsoft, а именно Windows 98 и Windows 2000, которые предназначены для использования на персональных компьютерах. Эти операционные системы отличаются своей внутренней организацией, но используют один и тот же интерфейс для программирования приложений — Win32 API. Мы не будем рассматривать операционную систему Windows CE, которая предназначена для использования в таких различных устройствах, как, например, устройства бытовой электроники, контроллеры для управления технологическими процессами и устройства управления коммуникационным оборудованием. Но, разобравшись в изложенном материале, вы получите опыт, который поможет вам как в изучении Windows CE, так и других операционных систем.

Относительно операционной системы Windows XP можно сказать следующее. Те приемы системного программирования, которые рассмотрены в этой книге для операционной системы Windows 2000, также работают и в операционной системе Windows XP.

1.3. Интерфейс программирования приложений Win32 API

Интерфейс программирования приложений Win32 API представляет собой набор функций и классов, которые используются для программирования приложений, работающих под управлением операционных систем фирмы Microsoft. Следует отметить, что в работе многих функций Win32 API существуют различия, которые зависят от типа операционной системы. Кроме того, некоторые функции работают только в операционной системе Windows 2000 и не поддерживаются операционной системой Windows 98. Все эти случаи будут отмечаться отдельно. Но все же в работе функций Win32 API в разных версиях операционных систем гораздо больше общего, чем различий. Поэтому чаще всего мы будем говорить, что функции Win32 API предназначены для разработки приложений на платформах операционных систем Windows, не делая различия между операционными системами Windows 98 и Windows 2000. Это соглашение значительно облегчит изложение материала, не загромождая его ненужными подробностями, которые отвлекают от сути рассматриваемых вопросов.

Функционально Win32 API подразделяется на следующие категории:

- Base Services (базовые сервисы);
- Common Control Library (библиотека общих элементов управления);
- Graphics Device Interface (интерфейс графических устройств);
- Network Services (сетевые сервисы);
- User Interface (интерфейс пользователя);
- Windows NT Access Control (управление доступом для Windows NT);
- Windows Shell (оболочка Windows);
- Windows System Information (информация о системе Windows).

Кратко опишем функции, которые выполняются в рамках этих категорий. Функции базовых сервисов обеспечивают приложениям доступ к ресурсам компьютера. Категория Common Control Library содержит классы окон, которые часто используются в приложениях. Интерфейс графических устройств обеспечивает функции для вывода графики на дисплей, принтер и другие графические устройства. Сетевые сервисы используются при работе компьютеров в компьютерных сетях. Интерфейс пользователя обеспечивает функции для взаимодействия пользователя с приложением, используя окна для ввода-вывода информации. Категория Windows NT Access Control содержит функции, которые используются для защиты информации путем контроля и ограничения доступа к защищаемым объектам. Категории Windows Shell и Windows System Information содержат соответственно функции для работы с оболочкой и конфигурацией операционной системы Windows.

В курсе системного программирования главным образом изучается назначение и использование функций из категорий Base Services и Windows NT Access Control. Функции из категорий Common Control Library, Graphics Device Interface и User Interface используются для разработки интерфейса приложений, а курс, который изучает назначение и использование этих функций, как правило, называется "Программирование пользовательских интерфейсов в Windows". Изучив два этих курса и добавив сюда свои знания по программированию на языке C++, вы получите довольно содержательное представление о разработке приложений на платформе Win32 API.

В связи с тем, что программирование графических пользовательских интерфейсов в Windows само по себе является довольно трудоемким занятием, мы будем изучать функции ядра Windows, работая только с консольными приложениями. Это упростит изложение предмета и избавит нас от большого количества кода, не относящегося к существу рассматриваемых вопросов.

1.4. Типы данных в Win32 API

Прежде всего заметим, что интерфейс программирования приложений Win32 API ориентирован на язык программирования C или, в более широком смысле, на процедурные языки программирования. Поэтому в этом интерфейсе, не используются такие возможности языка программирования C++, как классы, ссылки и механизм обработки исключений.

Чтобы сделать интерфейс Win32 API более независимым от конкретного языка программирования или, может быть, более соответствующим аппаратному обеспечению компьютера, разработчики этого интерфейса определили новые простые типы данных. Эти типы данных используются в прототипах функций интерфейса Win32 API.

Новые простые типы данных определены как синонимы простых типов данных языка программирования C. Чтобы отличать эти типы от других типов, их имена определены прописными буквами. Общее количество простых типов данных, определенных в интерфейсе Win32 API, довольно велико. Поэтому ниже приведены определения только тех простых типов данных из этого интерфейса, которые очевидным образом переименовывают простые типы данных языка программирования C.

```
typedef char CHAR;

typedef unsigned char UCHAR;
typedef UCHAR *PUCHAR;
typedef unsigned char BYTE;
typedef BYTE *PBYTE;
typedef BYTE *LPBYTE;
```

```
typedef short SHORT;

typedef unsigned short USHORT;
typedef USHORT *PUSHORT;
typedef unsigned short WORD;
typedef WORD *PWORD;
typedef WORD *LPWORD;

typedef int INT;
typedef int *PINT;
typedef int *LPINT;
typedef int BOOL;
typedef BOOL *PBOOL;
typedef BOOL *LPBOOL;

typedef unsigned int UINT;
typedef unsigned int *PUINT;

typedef long LONG;
typedef long *LPLONG;

typedef unsigned long ULONG;
typedef ULONG *PULONG;
typedef unsigned long DWORD;
typedef DWORD *PDWORD;
typedef DWORD *LPDWORD;

typedef float FLOAT;
typedef FLOAT *PFLOAT;

typedef void *LPVOID;
typedef CONST void *LPCVOID;
```

Остальные простые типы данных, определенные в интерфейсе Win32 API, имеют, как правило, специфическое назначение и поэтому они будут описаны при их использовании.

Кроме того, в интерфейсе Win32 API определены символические константы FALSE и TRUE для обозначения соответственно ложного и истинного логических значений. Определения этих констант приведены ниже.

```
#ifndef FALSE
#define FALSE 0
```

```
#endif

#ifdef TRUE
#define TRUE 1
#endif
```

В интерфейсе Win32 API также определено множество сложных типов данных, таких как структуры и перечисления. Как правило, эти типы данных имеют специфическое назначение и поэтому будут описаны при их непосредственном использовании.

1.5. Объекты и их дескрипторы в Windows

Объектом в Windows называется структура данных, которая представляет системный ресурс. Таким ресурсом может быть, например, файл, канал, графический рисунок. Операционные системы Windows предоставляют приложению объекты трех категорий:

- User (объекты интерфейса пользователя);
- Graphics Device Interface (объекты интерфейса графических устройств);
- Kernel (объекты ядра).

Категория User включает объекты, которые используются приложением для интерфейса с пользователем. К таким объектам относятся, например, окна и курсоры. Категория Graphics Device Interface включает объекты, которые используются для вывода информации на графические устройства. К таким объектам относятся, например, кисти и перья. Категория Kernel включает объекты ядра операционной системы Windows. К таким объектам относятся, например, файлы и каналы. При изучении системного программирования подробно рассматриваются только объекты категории Kernel. Объекты двух оставшихся категорий рассматриваются при изучении программирования графических интерфейсов.

Под доступом к объектам понимается возможность приложения выполнять над объектом некоторые функции. Приложение не имеет прямого доступа к объектам, а обращается к ним косвенно. Для этого в операционных системах Windows каждому объекту ставится в соответствие дескриптор (handle). В Win32 API дескриптор имеет тип HANDLE. *Дескриптор объекта* представляет собой запись в таблице, которая поддерживается системой и содержит адрес объекта и средства для идентификации типа объекта. Дескрипторы объектов создаются операционной системой и возвращаются функциями Win32 API, которые создают объекты. За редким исключением, эти функции имеют вид CreateObject, где слово Object заменяется именем конкретного объекта. Например, процесс создается при помощи вызова функции CreateProcess. Как правило, такие функции возвращают дескриптор соз-

данного объекта. Если это значение не равно `NULL` (или отрицательному значению), то объект создан успешно.

После завершения работы с объектом его дескриптор нужно закрыть, используя функцию `CloseHandle`, которая имеет следующий прототип:

```
BOOL CloseHandle(  
    HANDLE hObject    // дескриптор объекта  
);
```

При успешном завершении функция `CloseHandle` возвращает ненулевое значение, в противном случае — `FALSE`. Функция `CloseHandle` удаляет дескриптор объекта, но сам объект удаляется не всегда. Дело в том, что в Windows на один и тот же объект могут ссылаться несколько дескрипторов, которые создаются другими функциями для доступа к уже созданному ранее объекту. Функция `CloseHandle` уничтожает объект только в том случае, если на него больше не ссылается ни один дескриптор.



Часть I

Управление потоками и процессами

Глава 2. Потоки и процессы

Глава 3. Потоки в Windows

Глава 4. Процессы в Windows

Глава 2



Потоки и процессы

2.1. Определение потока

Определение потока тесно связано с последовательностью действий процессора во время исполнения программы. Исполняя программу, процессор последовательно выполняет инструкции программы, иногда осуществляя переходы в зависимости от некоторых условий. Такая последовательность выполнения инструкций программы называется *потоком управления* внутри программы. Отметим, что поток управления зависит от начального состояния переменных, которые используются в программе. В общем случае различные исходные данные порождают различные потоки управления. Поток управления можно представить как нить в программе, на которую нанизаны инструкции, выполняемые микропроцессором. Поэтому часто поток управления также называется *нитью* (thread). В русскоязычной литературе за потоком управления закрепилось название *поток*. Для пояснения понятия потока рассмотрим следующую программу, которая выводит минимальное число из двух целых чисел или сообщение о том, что числа равны.

```
#include <iostream.h>
int main()
{
    int a, b;

    cout << "Input two integers: ";
    cin >> a >> b;
    if (a == b)
    {
        cout << "There is no min." << endl;
        return 0;
    }
}
```

```
if (a < b)
    cout << "min = " << a << endl;
else
    cout << "min = " << b << endl;
return 0;
}
```

Предположим, что перегруженные операторы ввода-вывода не образуют новых потоков. Тогда в зависимости от входных данных эта программа образует один из трех возможных потоков управления. А именно, если выполняется условие ($a == b$), то образуется поток:

```
cout << "Input two integers: ";
cin >> a >> b;
if (a == b)
{
    cout << "There is no min." << endl;
    return 0;
}
```

Если выполняется условие ($a < b$), то образуется поток:

```
cout << "Input two integers: ";
cin >> a >> b;
if (a == b)
if (a < b)
    cout << "min = " << a << endl;
return 0;
```

Если же выполняется условие ($a > b$), то образуется поток

```
cout << "Input two integers: ";
cin >> a >> b;
if (a == b)
if (a < b)
    cout << "min = " << b << endl;
return 0;
```

Теперь перейдем к классификации программ в зависимости от количества определяемых ими параллельных потоков управления. Будем говорить, что программа является *многопоточной*, если в ней может одновременно существовать несколько потоков. Сами потоки в этом случае называются *параллельными*. Если в программе одновременно может существовать только один поток, то такая программа называется *однопоточной*. Например, сле-

дующая программа, которая просто вычисляет сумму двух чисел, является однопоточной:

```
#include <iostream.h>
int sum(int a, int b)
{
    return a + b;
}
int main()
{
    int a, b;
    int c = 0;
    cout << "Input two integers: ";
    cin >> a >> b;
    c = sum(a, b);
    cout << "Sum = " << c << endl;
    return 0;
}
```

Теперь предположим, что после вызова функции `sum` функция `main` не ждет возвращения значения из функции `sum`, а продолжает выполняться. В этом случае получим программу, состоящую из двух потоков, один из которых определяется функцией `main`, а второй — функцией `sum`. Причем эти потоки независимы, т. к. они не имеют доступа к общим или, другими словами, разделяемым переменным. Правда в этом случае не гарантируется, что поток `main` выведет сумму чисел `a` и `b`, т. к. инструкция вывода значения суммы может отработать раньше, чем поток `sum` вычислит эту сумму.

Из этих рассуждений видно, что для того чтобы отметить функцию, которая порождает новый поток в программе, должна использоваться специальная нотация. В операционных системах Windows для обозначения того, что функция образует поток, используются специальные спецификаторы функции. Такая функция обычно также называется потоком.

2.2. Контекст потока

В общем случае содержимое памяти, к которой поток имеет доступ во время своего исполнения, называется *контекстом потока*. Определим, каким ограничениям на доступ к памяти должны удовлетворять функции, чтобы их можно было безопасно вызывать в параллельных потоках. Для этого рассмотрим следующую функцию:

```
int f(int n)
{
```



```
if (n > 0)
    --n;
if (n < 0)
    ++n;
return n;
}
```

Сколько бы раз эта функция не вызывалась параллельно работающими потоками, она будет корректно изменять значение переменной n , т. к. эта переменная является локальной в функции f . То есть для каждого нового вызова функции f будет создан новый локальный экземпляр переменной n . Такая функция f называется *безопасной для потоков*. Теперь введем глобальную переменную n и изменим нашу функцию следующим образом:

```
int n;
void g()
{
    if (n > 0)
        --n;
    if (n < 0)
        ++n;
}
```

В этом случае параллельный вызов функции g несколькими потоками может дать некорректное изменение значения переменной n , т. к. значение этой переменной будет изменяться одновременно несколькими функциями g . В этом случае функция g не является безопасной для потоков.

Та же проблема встречается и в случае, когда функция использует статические переменные. Для разбора этого случая рассмотрим функцию

```
int count()
{
    static int n = 0;
    ++n;
    return n;
}
```

которая возвращает количество своих вызовов. Если эта функция будет вызвана несколькими параллельно исполняемыми потоками, то нельзя точно определить значение переменной n , которое вернет эта функция, т. к. это значение изменяется всеми потоками параллельно.

В общем случае функция называется *повторно входимой* или *реентерабельной* (reentrant или reenterable), если она удовлетворяет следующим требованиям:

- не использует глобальные переменные, значения которых изменяются параллельно исполняемыми потоками;

- не использует статические переменные, определенные внутри функции;
- не возвращает указатель на статические данные, определенные внутри функции.

В системном программировании часто также рассматриваются программы в кодах микропроцессора, выполнение которых может прерываться и возобновляться в любой момент времени. Причем одна и та же программа может запускаться прежде, чем завершилось исполнение предыдущего экземпляра этой программы. В этом случае также необходимо, чтобы программный код допускал корректное параллельное выполнение нескольких экземпляров программы. Это условие обеспечивается в том случае, если программа не изменяет свой код во время исполнения. Здесь под кодом подразумеваются как команды, так и данные, принадлежащие программе. Программа в кодах микропроцессора, которая не изменяет свой код, также называется *реентерабельной*.

В дополнение к реентерабельным функциям определяют также функции, безопасные для вызова параллельно исполняемыми потоками. Функция называется *безопасной для потоков*, если она обеспечивает блокировку доступа к ресурсам, которые она использует. Как обеспечить блокирование доступа к ресурсам, рассматривается в гл. 6, 7, посвященных синхронизации потоков. Сейчас же только скажем, что в этом случае решается задача взаимного исключения доступа к разделяемым ресурсам, используя примитивы синхронизации.

Очевидно, что если функция не является реентерабельной, то она также не является и безопасной для потоков, т. к. в этом случае несколько потоков разделяют общую память, не блокируя доступ к ней. А память, как уже говорилось, также является системным ресурсом.

2.3. Состояния потока

Как видно из определения, поток описывает динамическое поведение всей программы или какой-либо функции в программе. Для удобства обозначений предположим, что программа является однопоточной. Тогда поток можно рассматривать как пару:

поток = (процессор, программа).

Программа может исполняться процессором только в том случае, если она готова к исполнению. То есть все системные ресурсы, которые необходимы для исполнения этой программы, свободны для использования. Кроме того, для исполнения программы необходимо, чтобы и сам процессор был свободен и готов к исполнению этой программы. Для более формального описания этих ситуаций вводятся понятия "состояние процессора" и "состояние

программы". При этом предполагают, что процессор и программа могут находиться в следующих состояниях.

□ Состояния процессора:

- процессор не выделен для исполнения программы;
- процессор выделен для исполнения программы.

□ Состояния программы:

- программа не готова к исполнению процессором;
- программа готова к исполнению процессором.

Для краткости записи введем для этих состояний следующие названия:

□ Состояния процессора:

- "не выделен";
- "выделен".

□ Состояния программы:

- "не готова";
- "готова".

Тогда мы можем определить *состояние потока* как пару состояний:

состояние потока = (состояние процессора, состояние программы).

Перечислив различные комбинации состояний процессора и программы, можно описать все возможные состояния потока. Введем для состояний потока следующие названия:

□ поток заблокирован = ("не выделен", "не готова");

□ поток готов к выполнению = ("не выделен", "готова");

□ поток выполняется = ("выделен", "готова");

Будем считать, что состояние ("выделен", "не готова") является недостижимым для потока. То есть программе, не готовой к исполнению, процессор не выделяется. Более кратко эти состояния потока будем просто обозначать словами: "блокирован", "готов" и "выполняется". Для полноты картины нужно ввести для потоков еще два состояния: "новый" и "завершен", которые описывают соответственно поток, еще не начавший свою работу, и поток, завершивший свою работу. Тогда диаграмма возможных переходов потока из состояния в состояние может быть изображена, как это показано на рис. 2.1.

В результате мы получили простейшую диаграмму переходов потока из состояния в состояние. Сами переходы потока из состояния в состояние, которые на диаграмме обозначаются дугами, описывают некоторые операции

над потоком. Названия этих операций указаны рядом со стрелками. Кратко опишем эти операции.

- ❑ Операция `Create` выполняется потоком, который создает новый поток из функции. Эта операция переводит поток из состояния "новый" в состояние "готов".
- ❑ Операция `Exit` выполняется самим исполняемым потоком в случае его завершения. Эта операция переводит поток из состояния "выполняется" в состояние "завершен".

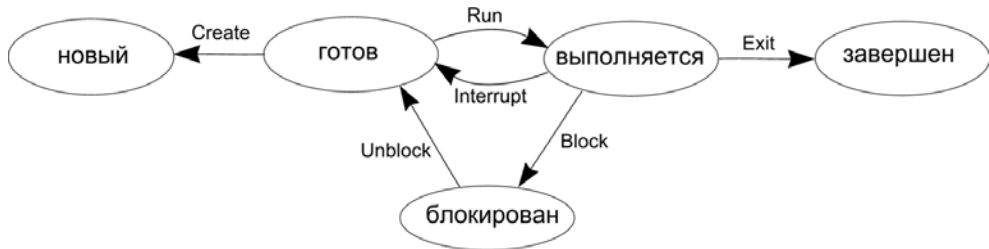


Рис. 2.1. Модель пяти состояний потока.

Оставшиеся четыре операции выполняются операционной системой.

- ❑ Операция `Run` запускает готовый поток на выполнение, т. е. выделяет ему процессорное время. Эта операция переводит поток из состояния "готов" в состояние "выполняется". Поток получает процессорное время в том случае, если подошла его очередь к процессору на обслуживание.
- ❑ Операция `Interrupt` задерживает исполнение потока и переводит его из состояния "выполняется" в состояние "готов". Эта операция выполняется над потоком в том случае, если истекло процессорное время, выделенное потоку на исполнение, или исполнение потока прервано по каким-либо другим причинам.
- ❑ Операция `Block` блокирует исполнение потока, т. е. переводит его из состояния "выполняется" в состояние "блокирован". Эта операция выполняется над потоком в том случае, если он ждет наступления некоторого события, например, завершения операции ввода-вывода или освобождения ресурса.
- ❑ Операция `Unblock` разблокирует поток, т. е. переводит его из состояния "блокирован" в состояние "готов". Эта операция выполняется над потоком в том случае, если событие, ожидаемое потоком, наступило.

Разрешим потокам также выполнять операции друг над другом. Для этого введем операции `Suspend` и `Resume`.

- ❑ Операция `Suspend` приостанавливает исполнение потока.
- ❑ Операция `Resume` возобновляет исполнение потока.

Используя эти операции, один поток может соответственно приостановить или возобновить исполнение другого потока независимо от того, в каком состоянии этот последний поток находится. Впрочем, заметим, что поток может приостановить и свое исполнение. Если над потоком выполнена операция *Suspend*, то будем говорить, что поток находится в *приостановленном* или *подвешенном состоянии*. Кратко будем говорить, что в этом случае поток "подвешен". Дополним диаграмму состояний потока, изображенную на рис. 2.1, этими новыми операциями и состояниями. Получим более полную диаграмму состояний потока, которая показана на рис. 2.2.

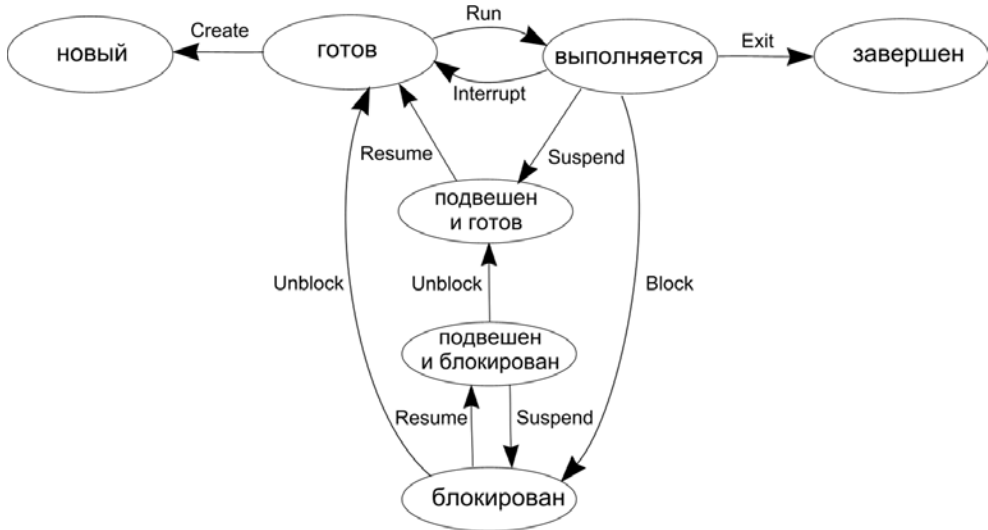


Рис. 2.2. Модель семи состояний потока

Теперь разрешим потоку выполнять операции над самим собой. Для этого введем операцию *Sleep*.

- Операция *Sleep* позволяет потоку приостановить свое исполнение на некоторый интервал времени или, другими словами, заснуть.

Разбудить поток должна операционная система по истечении заданного интервала времени, используя операцию *Wakeup*. Если поток выполнил операцию *Sleep*, то будем говорить, что он перешел в *сонное состояние* или "спит".

- Операция *Wakeup* позволяет операционной системе разбудить поток.

В результате можно построить полную диаграмму состояний потока, которая и приведена на рис. 2.3.

В заключение этого параграфа скажем, что в конкретных операционных системах для работы с потоками могут быть определены и другие состояния,

а также операции, которые переводят потоки в эти состояния. В гл. 3 будет рассмотрена модель состояний потока в операционной системе Windows 2000.

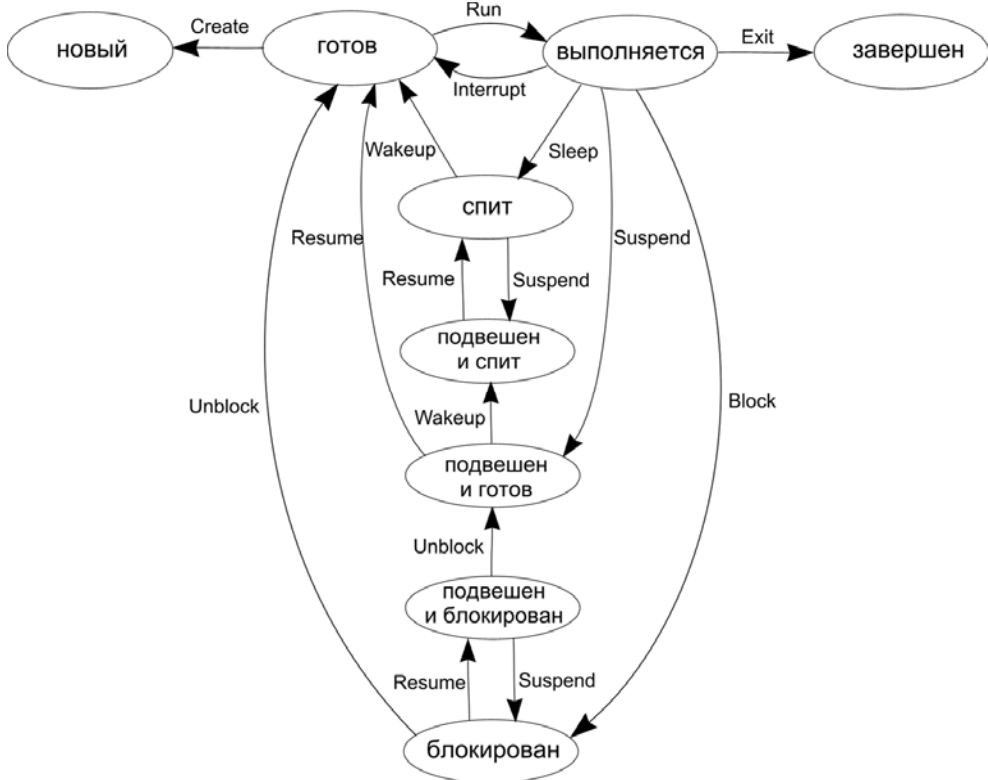


Рис. 2.3. Модель девяти состояний потока

2.4. Диспетчеризация и планирование потоков

В однопрограммной операционной системе одновременно может выполняться только один поток, которому доступны все ресурсы компьютера. Поэтому блокировка потока может происходить только в случаях ожидания этим потоком события, отмечающего завершение операций ввода-вывода. Недостатком однопрограммных операционных систем является их низкая производительность, т. к. процессор простаивает, если поток блокирован.

В мультипрограммных операционных системах одновременно могут существовать несколько потоков, что повышает производительность компьютера. Однако в этом случае требуется некоторая дисциплина обслуживания этих