6 Управление вводом-выводом и файловая система

6.1 Модели ввода-вывода

Подсистема ввода-вывода обеспечивает связь вычислительной системы с внешними (по отношению к ЦП и памяти) узлами и устройствами. В современных системах многие из таких устройств бывают представлены на логическом уровне в виде файлов (псевдофайлов), поэтому понятия «управление вводом-выводом» и «управление файлами» в значительной мере синонимичны.

Подходы к организации ввода-вывода:

– «***Наглый***» (или «***наивный***») ввод-вывод – выполнение операций по инициативе ЦП (программы) без предварительной проверки, не заботясь о готовности устройств к обмену

– С ***предварительной*** «ручной» (заложенной в алгоритм программы) ***проверкой*** состояния устройства. Подход реализуется достаточно легко и универсально, но постоянные проверки требуют значительных затрат времени (критично для многозадачной системы)

– По ***прерываниям*** (по инициативе устройства) – транзакции инициируются ориентируясь на события самого устройства. Подход предполагает наличие обработчиков событий, активизирующихся на фоне других алгоритмов, что усложняет проектирование программы.

– С использованием ***ПДП*** (***DMA***) – прямого доступа в память. Это предполагает, что данные передаются между устройством и памятью напрямую, минуя процессор, под управлением отдельного специального контроллера. Процессор в это время может выполнять другие шаги алгоритма, не связанные с обращением к памяти. Максимальная производительность, но необходима аппаратная поддержка и права для выполнения привилегированных инструкций – хорошо подходит для ***драйверов***.

Модели ввода-вывода:

– блокирующий ввод-вывод (простейший, традиционный)

– неблокирующий ввод-вывод

– мультиплексированный ввод-вывод

– ввод-вывод по прерываниям

– многопоточная организация

– асинхронный ввод-вывод

– отображение файлов в память

Блокирующий ввод-вывод (может называться блокируемым, а также синхронным). Инициировавший транзакцию поток приостанавливается и ждет ее окончания.

Неблокирующий ввод-вывод (может не вполне корректно называться ***асинхронным***) – предварительная проверка состояния устройства и в случае его неготовности немедленное завершение транзакции с выдачей признака ошибки. Проверка выполняется системой внутри вызванной функции, ответственной за данную транзакцию.

Мультиплексированный ввод-вывод – программа анализирует состояния нескольких (многих) устройств (файлов, каналов, сокетов) и инициирует транзакции только для тех из них, которые уже готовы к обмену. Так обеспечивается более или менее параллельный обмен с несколькими устройствами (файлами, узлами сети и т.д.).

– Управляемый ***сигналами*** ввод-вывод – развитие метода обмена по прерываниям. Транзакции инициируются в рамках обработки события устройства – обычно достижение готовности к обмену, завершение предыдущей транзакции. Как и мультиплексирование, позволяет вести обмен с несколькими устройствами в однопоточной программе.

– ***Многопоточная*** реализация ввода-вывода – блокирующие операции перестают быть блокирующими, если они выполняются в отдельных потоках. Модель предполагает поддержку многозадачности системой и соответствующее «многозадачное» проектирование программы.

– ***Асинхронный*** ввод-вывод (также называемый «перекрывающимся» – ***overlapped***). Используются внутренние системные механизмы, берущие на себя организацию параллельных транзакций и прикладных потоков. Чаще всего это системные потоки (потоки системных сервисов).

Отображение (***проецирование***) файлов в память и получение к его содержимому прямого доступа (подобно массиву данных). Доступ к ячейкам такого массива синхронный, но ввиду малой длительности обращения эта операция не рассматривается как блокирующая. Своевременную передачу данных между файлом и буфером в памяти обеспечивают внутренние системные механизмы (системные потоки).

Наиболее естественная и «прозрачная» модель – блокирующий ввод-вывод, но он не раскрывает в полной мере возможности и преимущества многозадачной системы. Прочие более сложны, но позволяют организовать более эффективное совмещение операций и более эффективное использование ресурсов системы.

6.2 Файлы и файловая система

Файл – упорядоченный набор данных (обычно подразумеваются данные на внешнем носителе), пригодный для использования прикладными программами в вычислительной системе. Удобно представлять файл как совокупность ***данных*** (используются прикладными программами) и ***метаданных*** (данные о размещении данных, используются системными программами).

Файловая система (в зависимости от контекста):

– способ организации данных (файлов), в первую очередь на внешних носителях;

– модули ОС, отвечающие за работу с этими данными;

– сами данные, включая служебные, содержащиеся в логическом запоминающем устройстве или в его ***разделе***, организованные соответствующим образом (т.е. конкретный экземпляр логического раздела вместе с его содержимым).

Поддерживаемые MS Windows файловые системы:

– семейство FAT (FAT 12, FAT 16, FAT 32, exFAT);

– NTFS – «родная» для Win NT;

– HPFS – файловая система OS/2 (номинальная поддержка, вероятно уже прекращена);

– файловые системы иных устройств, например CDFS.

Иерархическое (древовидное) построение файловой системы (как правило): ***корневой директорий*** (***каталог***, ***папка***, список файлов), содержащий файлы и другие ***директории***, которые, в свою очередь, также содержат файлы и директории. В файловых системах Microsoft традиционно (DOS, Win 16) каждый ***логический диск*** имел собственное дерево директориев, корнем которого было имя («буква») диска. В Win NT был введен по аналогии с Unix-системами общий корневой узел, объединяющий логические диски как свои директории, но не все программы показывают его явным образом.

Идентификатор файла – его ***имя*** в файловой системе.
Полное имя файла – имя самого файла и ***путь*** к нему (перечисление вышестоящих директориев).
Путь может быть ***абсолютным*** (начиная от корневого директория) или ***относительным*** (записывается начиная от текущего директория).

Имя должно быть уникально в пределах текущего директория, полное имя – в пределах всей файловой системы.

Глубина иерархии обычно не лимитируется, но ограничена общая длина полных имен.

Имена (в файловых системах Microsoft):

– «короткие» («DOS», «8.3») – ограничены по длине 8 символами имени и 3 символами «расширения», кодировка – ASCII (FAT12/16) или расширенные таблицы 8-битных символов (VFAT)

– «длинные» – до 255 символов, кодировка Unicode; поддержка в FAT 32, NTFS, exFAT

Типы файлов:

Обычный (***регулярный***) файл – файл без специальных характеристик: программы или данные.

Директории – списки других файлов и директориев.

Файлы-***ссылки*** – хранимые в виде файлов символические имена других файлов. В Windows как самостоятельный тип не выделяются, роль ссылок выполняют регулярные файлы со специальными соглашениями об именах (**`*.lnk`**, **`*.pif`**).

Файлы – ***логические устройства***:
– ***символьные*** – псевдофайлы, связанные с соответствующими реальными или виртуальными устройствами, доступны в общем случае для чтения и записи всеми программами;
– ***блочные*** – служат для размещения на них файловых систем.

Файлы – ***коммуникационные ресурсы***: ***каналы*** (***pipe***), «***почтовые ящики***» (***mailslot***), ***сокеты*** (***socket***) и т.п.

Основные ***атрибуты*** файлов:
– тип в виде кодового значения или отдельных флагов («директорий», «системный», «скрытый», «только для чтения» и т.п.)
– размер
– дата и время (обычно отдельно создания, последней модификации, последнего доступа)
– владелец (обычно также владелец-группа)
– права доступа (в Windows управление доступом возложено на отдельную подсистему безопасности)

Имя файла может быть неотъемлемой частью информации о нем (файловые системы FAT, NTFS) или быть отделено от остальных атрибутов и храниться только в директориях (файловые системы ufs, ext* и др., характерные для Unix-систем).

6.3 API для работы с файлами в Windows

Объект-файл – идентификация файловым дескриптором **`HANDLE`**. Особенность: невалидному значению дескриптора соответствует не **`0`** (**`NULL`**), а константа **`INVALID_HANDLE_VALUE`** (численно равна **`-1`**). Удобнее понимать этот объект как «открытый файл».

Группы функций:
– работа с файлами «на диске» (по именам)
– работа с файлами как объектами (открытыми файлами): чтение, запись и т.д.
– работа с директориями и другими структурами файловой системы
– получение информации о файловой системы и др. служебные функции.

Универсальная функция для открытия/создания файла, коммуникационного ресурса, открытия логического устройства:

```
HANDLE CreateFile(
  LPCTSTR lpFileName,
  DWORD dwDesiredAccess, DWORD dwShareMode,
  LPSECURITY_ATTRIBUTES pSecurityAttr,
  DWORD dwCreationDisposition,
  DWORD dwFlagsAndAttribs,
  HANDLE hTemplateFile
)
```

Большое количество дополнительных параметров и флагов делает применение функции очень разнообразным.

Закрытие файла (прекращение действия его Handle) – общая функция

```
CloseHandle( hFile)
```

Наиболее типичные операции с открытым файлом – чтение и запись:

```
BOOL ReadFile(
  HANDLE hFile,
  void* pBuffer,
  DWORD nBytesCntToRead, DWORD* pBytesCntRead,
  LPOVERLAPPED pOverlapped
);
BOOL WriteFile(
  HANDLE hFile,
  void* lpBuffer,
  DWORD nBytesCntToWrite, DWORD* pBytesCntWritten,
  LPOVERLAPPED pOverlapped
);
```

В расширенных (-Ex) версиях функций есть возможность использования callback-вызовов по завершении операции (актуально для ***асинхронного*** ввода-вывода).

Обычный блокирующий ввод-вывод и ввод-вывод с предварительной проверкой состояния большого интереса не представляют в силу своей простоты и прозрачности. (При этом они на практике покрывают большинство реально возникающих задач.)

Многопоточная реализация ввода-вывода близка к блокирующей модели, но потенциально длительные операции обмена с файлом (устройством) выносятся в отдельные потоки. Архитектура приложения обычно предусматривает наличие выделенного потока-«монитора» и некоторого количества потоков-«исполнителей», которые «монитор» создает и контролирует. Блокировка затрагивает только отдельные «исполнительные» потоки, другие же сохраняют работоспособность.

Преимущества:
– простая логика внутри потока-«исполнителя», подобная блокирующей реализации;
– эффективное использование ресурсов многопроцессорных (многоядерных) систем.

Вместе с тем, появляются проблемы корректного взаимодействия между потоками, необходимость обеспечить ***потокобезопасность*** программы.

Отображение файлов в память основано на использовании средств управления виртуальной памятью, поэтому фактически относится как к вводу-выводу (обмен с файлом), так и к подсистеме памяти. Характерны универсальность и разнообразие применений для решения различных задач, а не только обеспечения доступа к данным. Подробно рассматривается в разделе управления памятью.

6.4 Мультиплексированный ввод-вывод

Основан на использовании вызова **`select()`**, описываемого в POSIX и поддерживаемого как в Unix/Linux, так и в Windows, либо ему подобных. Фактически это расширение модели ввода-вывода с предварительной проверкой, отличающееся тем, что выполнение проверок вместе с реализацией состояния ожидания возлагается на систему, и появляется возможность эффективно управлять множеством источников и получателей данных – ***мультиплексировать*** их. Алгоритм не дожидается готовности каждого конкретного файла (устройства), а выполняет те действия (из нескольких своих ветвей), которые могут быть выполнены в данный момент. Это приводит к существенному усложнению алгоритма, который должен реализовывать логику своего рода ***конечного автомата***. Важно, что при этом не требуется поддержка многопоточности (и даже многозадачности) для прикладных программ.

Вызов **`select()`** выполняет проверку нескольких списков дескрипторов (Handle открытых файлов) и оставляет в списках только те из них, которые в данный момент соответствуют условиям (готовность к операциям ввода-вывода). Транзакции инициируются только для прошедших проверку дескрипторов. При этом может быть задано время, в течение которого функция ждет изменения состояния связанных с дескрипторами объектов (функция является блокирующей). Детали поведения функции зависят от реализации в различных ОС: ожидание в любом случае, ожидание только при отсутствии хотя бы одного изначально готового к обмену дескриптора, отражение в соответствующей переменной неиспользованного времени ожидания, и т.д. Функция работает с дескрипторами файлов, каналов, сокетов, логических устройств.

```
int select(
  int nfds,
  fd_set* readfds,
  fd_set* writefds,
  fd_set* exceptfds,
  const timeval* timeout
);
```

Списки передаются в виде «множеств» (set), представленных типом **fd_set**: проверяемые на готовность к чтению, готовность к записи и на наличие ошибок. Любой из списков может отсутствовать (указатель **NULL**), тогда соответствующая проверка пропускается. Первый параметр – количество элементов списка – в большинстве реализаций игнорируется (в т.ч. в Windows).

Внутренняя реализация может быть различной: например, массив в Windows или битовая маска во многих Unix-реализациях. Для работы с «множествами» предоставляется процедурный интерфейс (типичная реализация – макросы):

FD_SET(int *fd*, fd_set* *fdset*) – *добавление дескриптора в множество*
FD_CLR(int *fd*, fd_set* *fdset*) – *удаление дескриптора из множества*
FD_ISSET(int *fd*, fd_set* *fdset*) – *проверка наличия дескриптора в множестве*
FD_ZERO(fd_set* *fdset*) – *полная очистка множества*

Общая схема организации мультиплексированного ввода-вывода с помощью select()

Использование **`select()`** требует аккуратности и учета особенностей его работы. Например:

– наличие в списке невалидного дескриптора может прерывать проверку без анализа остальных дескрипторов, которые останутся в списке независимо от их реального состояния;

– нахождение хотя бы одного проходящего проверку дескриптора в любом из списков может немедленно прерывать дальнейшую проверку; содержимое списков будет корректно, но пауза выдерживаться не будет.

Другие системные вызовы, обслуживающие мультиплексированный ввод-вывод: функции **`WSAPoll()`** и др. для WSASocket-ов, методы Poll и др. .NET-класса **`Socket`**, использование функций **`WaitFor**()`**.

6.5 Асинхронный (перекрывающийся) ввод-вывод

По достигаемому эффекту подобен многопоточной реализации ввода-вывода, но используются внутренние системные потоки и специализированный API, что избавляет от самостоятельной организации взаимодействия потоков.

Если при открытии файла (создании объекта «открытый файл») вызовом **`CreateFile()`** среди флагов будет передан **`FILE_FLAG_OVERLAPPED`**, все последующие операции с дескриптором (Handle) этого файла, инициированные вызовами **`ReadFile()`**, **`WriteFile()`** будут выполняться как асинхронные (перекрывающиеся).

Структура **OVERLAPPED** – описание одной операции ввода-вывода (транзакции). Содержит, в частности, позицию начала обмена (в случае перекрывающихся транзакций единый «курсор» в файле невозможен), количество обрабатываемых байт, код результативности. Структура создается целиком в пользовательском адресном пространстве и идентифицируется указателем; этот же указатель передается в качестве аргумента в функции чтения/записи и служит идентификатором конкретной операции.

Организация асинхронных операций ввода-вывода

При использовании асинхронного ввода-вывода, особенно в случае множества параллельно выполняемых операций, необходимо синхронизироваться с ходом их выполнения и обеспечивать целостность общего конечного результата.

Несколько способов контроля состояния операции ввода-вывода:

– «Ручная» проверка: анализ полей структуры **`OVERLAPPED`**, макросы и функции: **`GetOverlappedResult()`**, **`HasOverlappedIoCompleted()`** и т.п.

– Функции ожидания (**`WaitFor***()`**), примененные к дескриптору файла. Файл как объект ожидания будет считаться «сработавшим», когда он свободен от транзакций. Нельзя различить транзакции, инициированные параллельно.

– Использование объектов Event, ассоциированных с каждой транзакцией (посредством структуры **`OVERLAPPED`**). Функции ожидания будут применены именно к объектам Event, что позволит контролировать каждую из транзакций. Приходится создавать множество объектов.

– Использование callback-обработчиков завершения транзакций. Потребуются «расширенные» версии вызовов: **`Read/WriteFileEx()`**, **`WaitFor***Ex()`**, **`SleepEx()`**.

6.6 Порты завершения ввода-вывода (I/O Completion Ports)

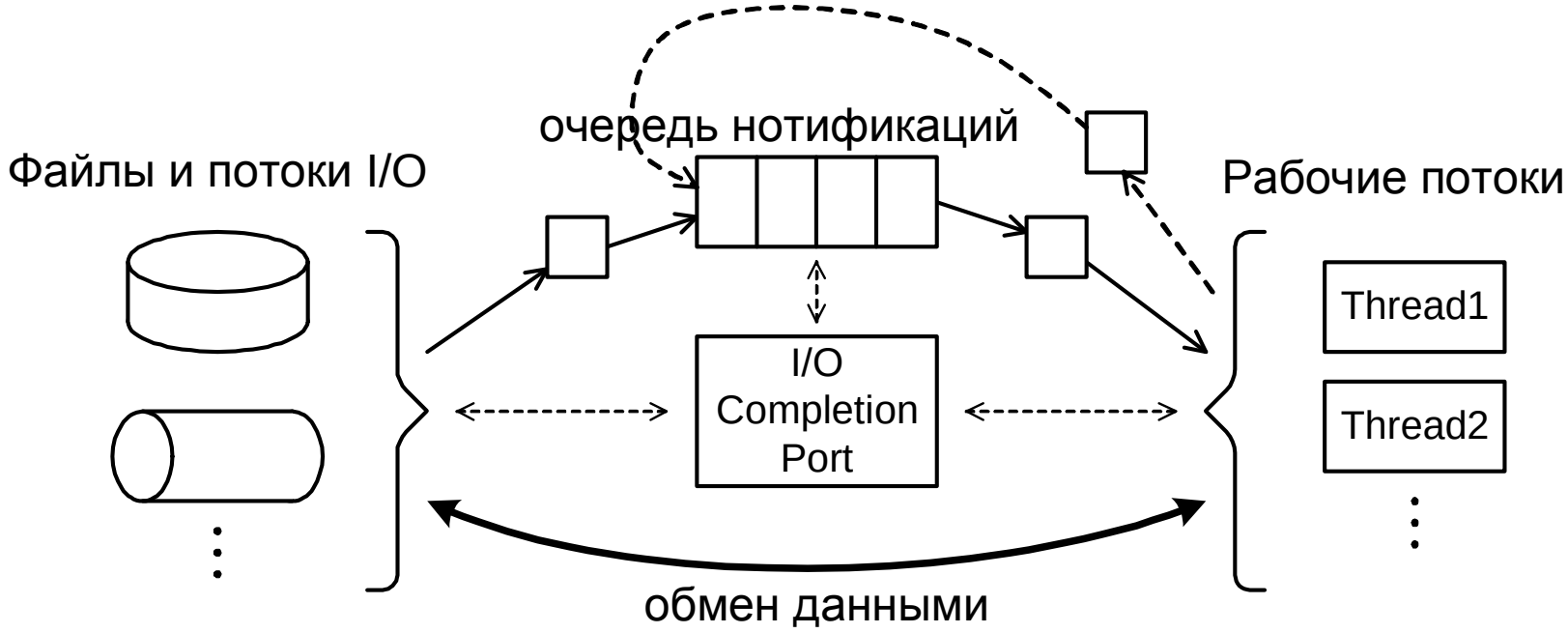
Предлагаемый Windows механизм для унификации управления многопоточной реализацией обработки ввода-вывода. Используется объект ***I/O Completion Port*** и извещения ***Completion Notification Packet***, передаваемые через очередь.

Порты завершения являются потенциально более производительной и потенциально более удобной с точки зрения архитектуры альтернативой самостоятельной реализации управления многопоточным вводом-выводом

Порт завершения ассоциируется с открытым файлом (каналом, устройством), который обязательно должен быть открыт для асинхронного ввода-вывода (флаг **OVERLAPPED**). Порт может быть ассоциирован с несколькими файлами (?). Каждому из таких файлов сопоставляется уникальное значение-«ключ», позволяющее однозначно определить источник событий этого файла (для мультиплексирования потока событий и данных).

Далее, с портом связываются программные потоки (Thread), которые будут обрабатывать события файла, с которым ассоциирован порт. При наступлении событий генерируется извещение – Completion Notification Packet – которое помещается в очередь порта. Потоки запрашивают извещения из очереди и ожидают их появления. Поток не может быть связан более чем с одним портом.

В любом случае, через порт проходят только события – извещения или ***нотификации*** (***notification packets***) , но не данные. Сами операции ввода-вывода инициируются связанным с портом потоками, они же обрабатывают их результаты.



Использование портов завершения ввода-вывода

Механизм портов завершения действует для операций чтения/записи (**`ReadFile()`**, **`WriteFile()`**, но не их **`-Ex`**-версии!), а также некоторых других: **`DeviceIoControl()`**, **`ConnectNamedPipe()`**, **`TransactNamedPipe()`**, **`WaitCommEvent()`** и т.п.

Создание порта завершения или ассоциация существующего порта с файлом:

```
HANDLE CreateIoCompletionPort(
 HANDLE hFile; HANDLE hExistingComplPort;
 ULONG_PTR CompletionKey;
 DWORD nConcurrentThreads
)
```

Запрос извещения из очереди порта (блокирующая функция):

```
GetQueuedCompletionStatus();
```

Также есть возможность искусственно сгенерировать извещение и поместить его в очередь, откуда оно будет извлечено для обработки наравне с остальными (полезно для унификации внутренней логики программы):

```
PostQueuedCompletionStatus();
```

6.9 Использование файлов устройств

Основная характеристика – необходимость учитывать особенности и соблюдать требования физических устройств: тайминги, количественные ограничения, низкоуровневые протоколы обмена. Также во многих случаях необходимо использование привилегированных инструкций ЦП, например чтение/запись аппаратных портов. Как следствие, требуется достаточно сложное низкоуровневое программирование, возможно – написание специализированного драйвера. Унификация доступа к устройствам посредством файлов (псевдофайлов) позволяет существенно упростить эту задачу для прикладных программ.

Пример: традиционные потоки ввода-вывода **`stdin`**, **`stdout`**, **`stderr`**; файлы **`CON`**, **`PRN`**, **`LPT*`**, **`AUX`**, **`COM*`** и т.п.

Дополнительно система может предлагать специфический API для конкретных устройств.

Пример: поддержка асинхронного последовательного интерфейса (UART, RS232, COM) в Win API (функции **-Comm-**):

```
BOOL SetupComm( HANDLE hFile,
 DWORD dwInQueue, DWORD dwOutQueue);

BOOL GetCommState( HANDLE hFile, DCB* pDCB);

BOOL SetCommState( HANDLE hFile, DCB* pDCB);

BOOL GetCommTimeouts( HANDLE hFile,
 COMMTIMEOUTS* pCommTimeouts);

BOOL SetCommTimeouts( HANDLE hFile,
 COMMTIMEOUTS* pCommTimeouts);
```

и другие.

Структуры **DCB** и **COMMTIMEOUTS** – параметры устройства.