Аппаратные средства защиты информации в микропроцессоре

Если в памяти одновременно могут находиться несколько независимых программ, необходимы специальные меры по предотвращению или ограничению обращений одной программы к областям памяти, используемым другими программами. Программы могут содержать такие ошибки, которые, если этому не воспрепятствовать, приводят к искажению информации, принадлежащей другим программам. Последствия таких ошибок особенно опасны, если разрушению подвергнутся программы операционной системы. Другими словами, надо исключить несанкционированное воздействие программы пользователя на работу программ других пользователей и программ операционной системы.

Чтобы воспрепятствовать разрушению одних программ другими, достаточно защитить область памяти данной программы от попыток записи в нее со стороны других программ, а в некоторых случаях и своей программы (*защита от записи*), при этом допускается обращение других программ к этой области памяти для считывания данных.

В других случаях, например, при ограничениях на *доступ* к информации, хранящейся в системе, необходимо иметь возможность запрещать другим программам производить как *запись*, так и считывание в данной области памяти. Такая *защита от записи* и считывания помогает отладке программы, при этом осуществляется *контроль* каждого случая выхода за область памяти своей программы.

Для облегчения отладки программ желательно выявлять и такие характерные ошибки в программах, как попытки использования данных вместо команд или команд вместо данных в собственной программе, хотя эти ошибки могут и не разрушать информацию.

Средства *защиты памяти* должны предотвращать:

- неразрешенное взаимодействие пользователей друг с другом;
- несанкционированный доступ пользователей к данным;
- повреждение программ и данных из-за ошибок в программах;
- намеренные попытки разрушить *целостность системы*;
- случайные искажения данных.

Средства защиты микропроцессора делятся на 2 группы:

- **защиту при управлении памятью** и
- **защиту по привилегиям**.

Средства управления памятью обнаруживают большинство программных ошибок.

До загрузки селектора в *сегментный регистр* и кэширования дескриптора осуществляется несколько контро-льных проверок:*процессор* проверяет, что *поле* Index **селектора** находится в пределах таблицы, определяемой его битом TI ;

- при загрузке **селектора** в сегментный регистр данных (*DS*, ES, FS, GS) **тип дескриптора** должен разрешать считывание из сегмента.

  Только выполняемые сегменты для этих регистров не допускаются, но сегменты с разрешенными операциями выполнения/считывания допустимы;

- в случае сегментного регистра стека (SS) в сегменте должны быть разрешены операции считывания и записи;
- при загрузке регистра CS сегмент должен быть обязательно исполняемым;
- в регистр LDTR можно загружать только селектор, указывающий на дескриптор сегмента типа *LDT* ;
- в регистр TR можно загружать только селектор, указывающий на дескриптор сегмента состояния задачи.

Если хотя бы одна проверка дала отрицательный результат, то формируется особый случай и *загрузка* **селектора** не производится.

После загрузки селектора при фактическом обращении к памяти *процессор* контролирует, чтобы запрашиваемая операция (чтение/*запись*)для этого сегмента была разрешена. На этом этапе обнаруживаются и отвергаются попытки записи в сегмент кода или в только считываемые *сегменты* данных, а также считывание из сегмента кода, для которого разрешено только выполнение. Здесь же проводится проверка превышения сформированного **смещения в сегменте** длины сегмента, указанной в **поле предела дескриптора**. Такие ситуации невозможно выявить при загрузке селектора.

Защита по привилегиям фиксирует более тонкие ошибки и намеренные попытки нарушить *целостность системы*.

Под **привилегиями** понимается свойство, определяющее, какие *операции* и обращения к памяти разрешается производить процессору при выполнении текущей задачи.

На аппаратном уровне в процессоре поддерживаются 4 **уровня привилегий**. Распознаваемым процессором объектам назначается *значение* от 0 до 3, причем 0 соответствует высшему, а 3 - низшему уровню привилегий. С помощью указания уровня привилегий и правил защиты обеспечивается *управляемый доступ* к процедурам и данным операционной системы и других задач.

Привилегии устанавливаются значениями соответствующих полей в следующих основных *системных объектах* микропроцессора:

- *DPL* - **уровень привилегий сегмента** (находится в **байте доступа** дескриптора сегмента);
- *RPL* - биты <0,1> **селектора**, хранящегося в *сегментном регистре*;

текущий *уровень привилегий* программы *CPL* задается полем *RPL* селектора, хранящегося в *сегментном регистре* CS ;

- IOPL - поле **регистра флагов**, которое указывает, на каком уровне привилегий разрешено выполнять операции ввода/вывода, а также в некоторых других объектах, используемых, например, при переключении задач и обработке прерываний.

Так как число программ, которые могут выполняться на более высоком **уровне привилегий**, уменьшается к уровню 0 и так как программы уровня 0 действуют как *ядро* системы, **уровни привилегий** обычно изображаются в виде четырех **колец защиты** (*Protection Rings*) (рис. 5.1).

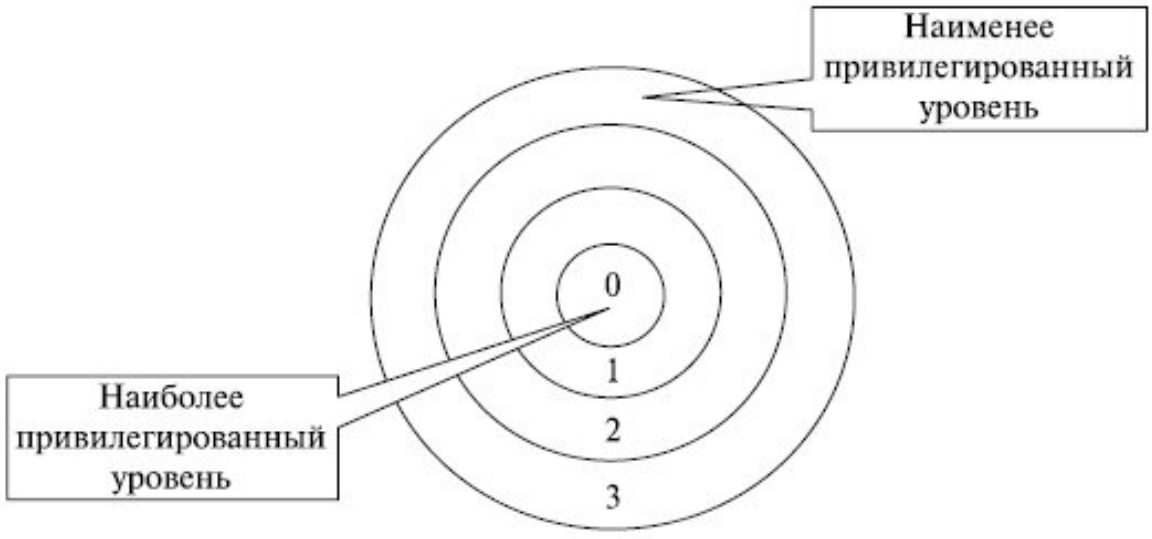

Рис. 5.1. "Кольца защиты"

Типовое распределение программ по кольцам защиты выглядит следующим образом:

- **уровень 0**: ядро ОС, обеспечивающее инициализацию работы, управление доступом к памяти, защиту и ряд других жизненно важных функций, нарушение которых полностью выводит из строя процессор;
- **уровень 1**: основная часть программ ОС (утилиты);
- **уровень 2**: служебные программы ОС (драйверы, СУБД, специализированные подсистемы программирования и т. д.);
- **уровень 3**: прикладные программы пользователя.

Аппаратные средства процессора, работающего в **защищенном режиме**, постоянно контролируют, что текущая *программа* достаточно привилегированна для того, чтобы:

- выполнять некоторые команды, называемые привилегированными;
- выполнять операции ввода/вывода на том или ином внешнем устройстве;
- обращаться к данным других программ;
- передавать управление внешнему (по отношению к самой программе) коду командами межсегментной передачи управления.

Привилегированные команды - это те команды, которые влияют на *механизмы* управления памятью, защиты и некоторые другие жизненно важные функции. Это, например, команды загрузки таблиц дескрипторов *GDT*, *IDT*, *LDT*, команды обмена с *регистрами управления* CRi. Они могут выполняться только программами, имеющими наивысший (нулевой)

уровень привилегий. Это приводит к тому, что простую незащищенную систему можно целиком реализовать только в кольце 0, так как в других **кольцах защиты** не будут доступны все команды.

Операции ввода/вывода разрешено выполнять программам, *уровень привилегий* которых не ниже значения, установленного в *поле* IOPL **регистра флагов**. То есть должно выполняться соотношение: *CPL* IOPL.

Обращение к данным других программ разрешается только на своем и менее привилегированном уровнях защиты (рис. 5.2).

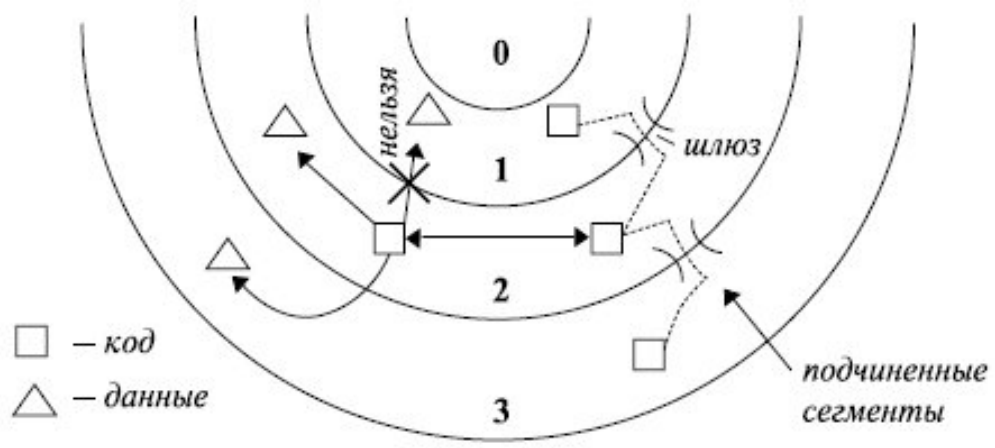

Рис. 5.2. Порядок взаимодействия программ и данных на разных уровнях привилегий

Передачи управления между программами ограничиваются только текущим **кольцом защиты**. В то же время в процессе выполнения любой программы необходимо обращаться к программам, находящимся на более высоком уровне привилегий, например, к драйверам или *СУБД*. Для этих целей используются специально установленные точки входа в эти программы (шлюзы). Передача управления на более низкий *уровень привилегий* осуществляется с помощью механизма подчиненных сегментов.

При передаче управления подчиненному сегменту действует правило: *DPL*>=max (*CPL*, *RPL*). Однако при этом подчиненный код будет выполняться на том же уровне привилегий, что и вызвавший его код (*CPL* не изменится). Ограничивая передачу управления в пределах одного **кольца защиты**, *процессор* предотвращает *произвольное* изменение **уровней привилегий**. Ели бы *значение CPL* можно было легко изменить, все остальные средства защиты по привилегиям потеряли бы смысл.

Использование шлюзов вызова

В процессе работы программам постоянно приходится обращаться к программам, находящимся на более высоком уровне привилегий, например, к драйверам внешних устройств, системам программирования. *Прямой* бесконтрольный вызов таких программ запрещается средствами защиты. Однако если к какой-либо системной программе предусматривается обращение со стороны менее привилегированных программ, то для нее создается специальный *объект* - **шлюз вызова**.

Здесь наиболее интересный момент связан с реализацией этого шлюза. С одной стороны, пользователю необходимо предоставить возможность выполнить необходимую ему более привилегированную программу. Но, с другой стороны, бесконтрольный вызов таких программ, например, *запуск* драйвера с его середины вследствие ошибок программирования или злого умысла, может привести к непредсказуемым последствиям.

Таким образом, необходимо дать возможность обращаться к системным программам, но обращаться только начиная с определенной фиксированной точки кода. **Шлюзы вызова** - это некоторые системные объекты, которые обеспечивают вход в строго определенную точку программы, находящейся на более высоком уровне привилегий.

Шлюз должен быть предварительно помещен в таблицу дескрипторов. Дескрипторы шлюзов не определяют никакого адресного пространства, поэтому в них нет полей базы и предела, то есть они фактически не являются дескрипторами. Обращение к более привилегированным программам производится командами, аналогичными командам обращения к подпрограммам в другом кодовом сегменте (команды типа FAR CALL).

То есть нельзя перейти в более привилегированный сегмент командой с полной передачей управления: "пришел на уровень с более высокими привилегиями и там остался", а переход возможен только с помощью команд с возвратом. Эти команды должны адресовать **шлюз вызова**, а не сегмент кода назначения. **Шлюз вызова** определяет сегмент кода, которому передается управление, и точное смещение в этом сегменте, где начинается выполнение процедуры.

Формат **шлюза вызова** представлен на рис. 5.3.

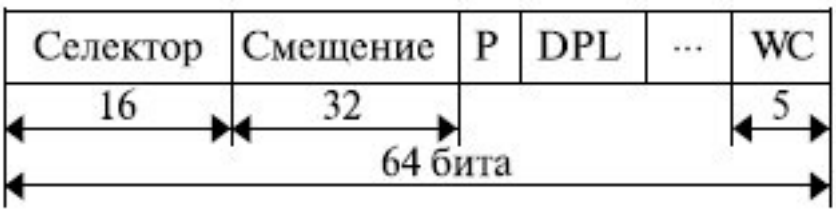

Рис. 5.3. Формат шлюза вызова

Шлюз вызова содержит селектор того сегмента, куда передается управление, и смещение в этом сегменте. Это, с одной стороны, позволяет найти данную программу, но, с другой стороны, строго определяет точку входа в программу, чтобы можно было запустить ее только со строго определенного места.

Другими важными параметрами, определяемыми шлюзом вызова, являются: P - **бит присутствия** ; WC - количество параметров, передаваемых из стека текущей программы в *стек* вызываемой программы; *DPL* - **уровень привилегий**.

При использовании шлюза вызова проводится следующий *анализ* **уровней привилегий**:

- значение *DPL* шлюза вызова должно быть больше или равно значению текущего уровня привилегий *CPL* и значению *RPL*селектора, вызывающего шлюз;
- значение *DPL* шлюза вызова должно быть больше или равно значению *DPL* целевого сегмента кода;
- значение *DPL* целевого сегмента кода должно быть меньше или равно значению текущего уровня привилегий *CPL*.

Порядок использования **шлюза вызова** представлен на рис. 5.4.

1. Как любая команда межсегментного перехода, команда FAR CALL содержит селектор сегмента и смещение в этом сегменте. Смещение, которое указано в команде, микропроцессор игнорирует: положение вызываемого кода в более привилегированном сегменте определяется не им, а шлюзом вызова. По селектору, определенному в команде, идет обращение к **таблице дескрипторов**. По **типу дескриптора** определяется, что это системный объект типа "шлюз вызова".
2. **Селектор** из шлюза вызова заносится в регистр CS микропроцессора, а смещение - в **регистр - указателя команд** EIP.
3. По полученному **селектору** обращаемся к **дескриптору** сегмента более привилегированной программы.

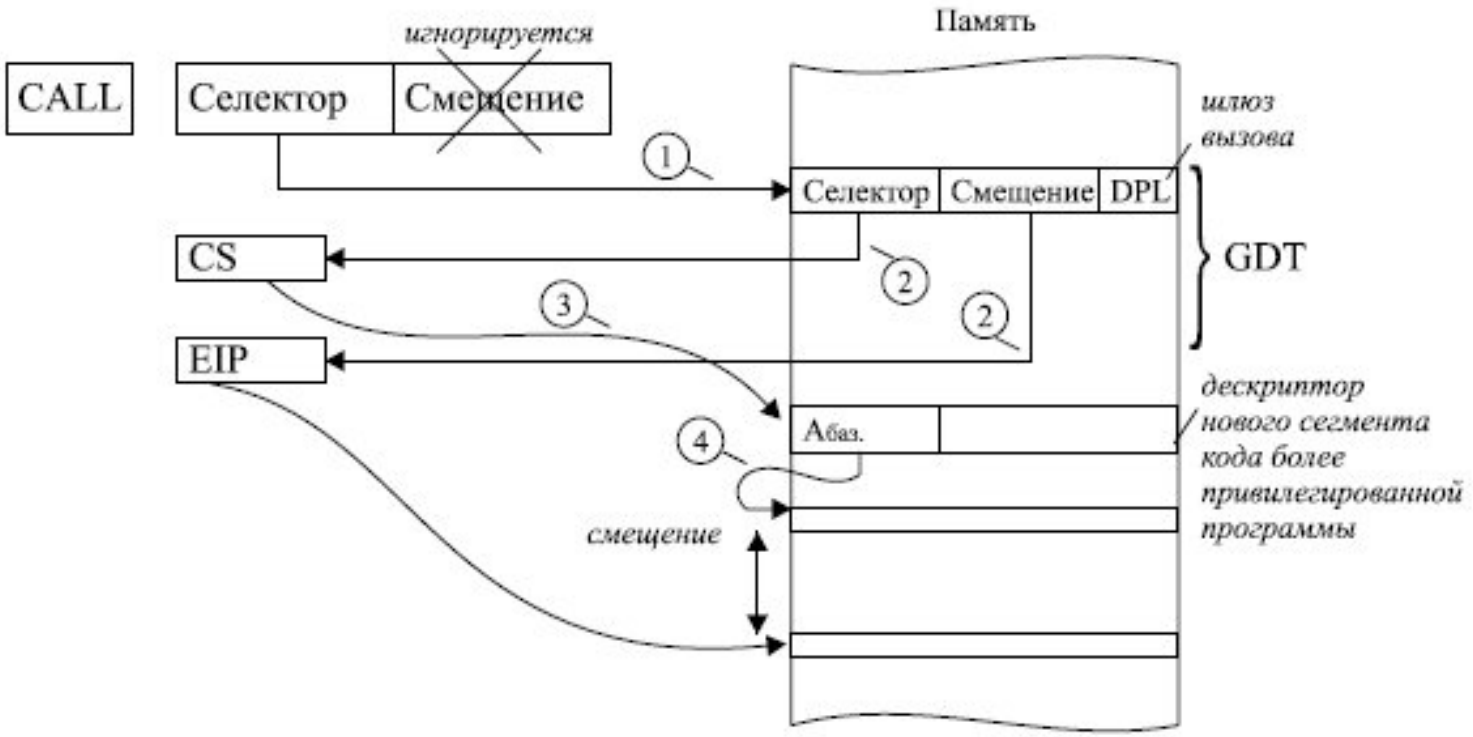

Рис. 5.4. Использование шлюза вызова для обращения к программам на более высоком уровне привилегий

4. Из дескриптора извлекается базовый адрес нового сегмента. Его суммирование со значением смещения из шлюза вызова, занесенного в EIP, определяет физический адрес начала новой программы.

Иначе говоря, с помощью такой многоступенчатой обработки команды перехода мы получаем *доступ* к более привилегированной программе. Но для этого *поле DPL* шлюза должно быть установлено таким, чтобы к нему могла обратиться менее привилегированная *программа*. То есть если мы хотим, чтобы пользовательские программы могли вызывать некоторую программу, находящуюся, например, на **уровне привилегий** 1, то *DPL* шлюза этой программы должен быть равен 3.

Предположим теперь, что *пользователь* хочет воспользоваться некоторыми системными утилитами. Пусть пользовательская *программа* имеет *уровень привилегий* 3, *ядро* ОС - уровень 0, утилиты ОС - уровень 1 (рис. 5.5).

В этом случае *шлюз* утилиты должен иметь *DPL* = 3. Это позволит пользовательской программе вызвать утилиты ОС, но не ее *ядро*. При необходимости к ядру операционной системы могут обратиться утилиты.

Механизм вызова:

- шлюзу утилит присваивается **уровень привилегий** 3, обеспечивая его доступность пользовательским программам;
- шлюзу ядра присваивается **уровень привилегий** 1, что делает его доступным для программ-утилит, но обращение пользовательских программ к шлюзу ядра невозможно.

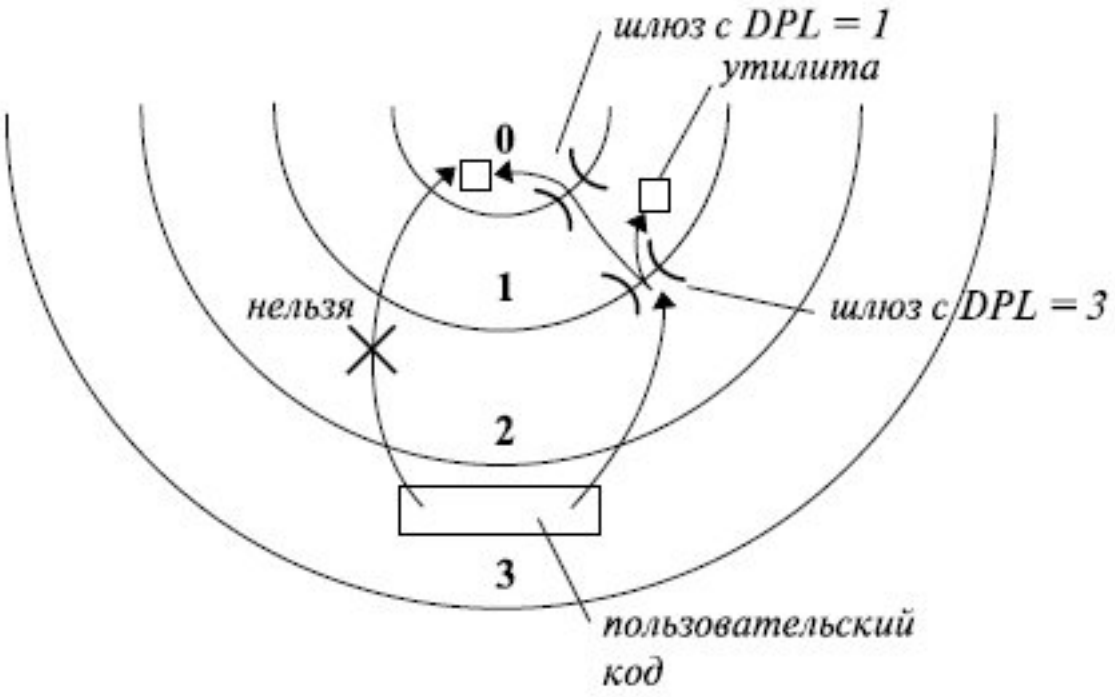

Рис. 5.5. Последовательное обращение к более привилегированным программам

Таким образом, *программа* может последовательно обратиться к ряду более привилегированных программ.

Конкретная ОС необязательно должна поддерживать все 4 **уровня привилегий**. Так, ОС *UNIX* работает с двумя **кольцами защиты**: *супервизор* (уровень 0) и *пользователь* (уровни 1, 2, 3). *Операционная система OS/2* поддерживает 3 уровня: код Ос работает в кольце 0, специальные процедуры для обращения к устройствам ввода/вывода действуют в кольце 1, а прикладные программы выполняются в кольце 3. В *Windows* NT используются только два **уровня привилегий**: нулевое и третье кольцо. В нулевом кольце работает *ядро* системы и системные драйверы, а в третьем - все запущенные приложения. Привилегированные команды и ввод-ввод для третьего кольца запрещены. Для взаимодействия с аппаратной частью компьютера пользовательские программы вызывают системные сервисы ядра ОС, обращение к которым оформлено как **шлюзы**. При вызове такого **шлюза**процесс переходит в нулевое кольцо, и там *ядро* ОС и драйверы обрабатывают *запрос* и возвращают результаты приложению. После перехода в нулевое кольцо *приложение* не может как-либо контролировать свое *исполнение*, пока управление не будет возвращено коду третьего кольца. Это является необходимым условием защиты, обеспечивающим *безопасность* всей системы.

Защита по привилегиям начинает работать уже на этапе загрузки **селектора** в сегментные регистры. При загрузке селектора в сегментные регистры данных должно выполняться соотношение: *DPL*< max (*CPL*, *RPL*), а при загрузке **селектора** в *сегментный регистр* стека SS должно быть выполнено соотношение: *DPL* = *CPL*.

При страничном преобразовании адреса применяется простой двухуровневый механизм **защиты по привилегиям**: *пользователь* (соответствует уровню 3 привилегий сегмента) и *супервизор* (уровни 0, 1, 2), указываемый в бите U/S ЭТС.

При сегментно-страничной организации памяти производится *объединение* защиты сегментов и страниц: сначала реализуется защита сегментов, а затем защита страниц. Например, допускается определить большой сегмент данных, в котором некоторые части будут только считываемые, а другие допускают

считывание и *запись*. В такой ситуации элементы каталога таблиц страниц и/или элементы таблиц страниц должны иметь соответствующие значения атрибута R/W.