Бурдонов И. Б., Копытов Г. В., Косачев А. С., Кузнецов С. Д., Смирнов Ю. П., Юдин В. Н.
КЛОС: операционная система и технология программирования.
Сб."Вопросы кибернетики. Программное обеспечение высокопроизводительной системы". Под ред. В.П. Иванникова. М., НСК АН СССР,1986 г.,стр.34-57.
24 стр.

Бурдонов И. Б., Копытов Г. В., Косачев А. С., Кузнецов С. Д., Смирнов Ю. П., Юдин В. Н.

КЛОС: ОПЕРАЦИОННАЯ СИСТЕМА И ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

В статье излагаются основные принципы организации операционной системы (КЛОС). Описываются объекты КЛОС и способы их взаимодействия. Коротко описаны разрабатываемые в КЛОС информационные системы – файловая система и система управления базами данных.

Создание больших программных комплексов (компиляторов, систем управления базами данных, транспортных служб в сетях ЭВМ, графических систем и т. п.) требует единого технологического подхода на всех направлениях и этапах работы. Интегральность разработки, обеспечивающая концептуальное единство проекта, основывается на единой технологии программирования. Для сложных систем особенно сильно влияет на эту технологию операционная система; это влияние ощущается каждый раз, когда становится существенным асинхронное выполнение различных компонентов с соответствующей синхронизацией и централизованным или распределенным управлением. В этом случае разработчикам приходится покидать «защитную оболочку» системы программирования и напрямую взаимодействовать с операционной системой. Такая ситуация имеет место, например, для систем управления файлами и базами данных.

Предлагаемая кластерная операционная система (КЛОС) может рассматриваться «узко» и «широко». В «широком смысле» КЛОС выполняет все те традиционные функции, которые требуются от системного математического обеспечения: управление вводом/выводом, внешней памятью, линиями связи, управление информацией и вычислительными процессами и т. п.. В этом смысле в КЛОС входят составной частью и файловая подсистема и подсистема управления базами данных. КЛОС «в узком смысле» – ядро КЛОС – рассматривается как самый нижний слой программного обеспечения, реализующий базовые объекты и средства «сборки» из таких объектов сложных составных подсистем.

Ядро КЛОС поставляет базу для единой технологии программирования, на которой выстраивается разнообразный «сервис»: системы проектирования и программирования. В частности, для использования всех преимуществ работы на уровне базовых объектов ядра требуется расширение используемых языков программирования. Все остальные подсистемы, включая: такую «внутреннюю» подсистему как подсистема управления вводом/выводом, строятся по единой технологии, обеспеченной ядром КЛОС.

Нужно особо отметить, что КЛОС строится как переносимая операционная система. Для ядра это означает прежде всего переносимый интерфейс между ядром и непривилегированными программами, что позволяет производить «легко переносимые» подсистемы.

В настоящей статье мы остановимся на некоторых особенностях ядра КЛОС. При этом упор будет делаться на наиболее нетрадиционные черты КЛОС, а также на те черты, которые наиболее сильно влияют на выбор структуры и технологии разработки

программных комплексов. В качестве примера таких комплексов будет кратко рассказано о системах управления файлами и базами данных в КЛОС.

1. Кластер – базовый объект КЛОС

Понятие кластера (или терминологически отличные, но близкие по смыслу понятия) традиционно используется в языках программирования с абстрактными типами данных. Кластер является неразрывной совокупностью объекта обработки и набора операций над этим объектом. Конкретная структура данных, представляющая объект, оказывается скрытой («инкапсулированной») за набором операций, определенных над объектом. Различают кластер-программу как «пучок» процедур, определяющих абстрактный тип данных, и кластер-объект как конкретное значение этого типа.

Основное расширение этого традиционного подхода, принятое в КЛОС, состоит в том, что кластер получает свойство асинхронного выполнения [I]. Это означает, что операция в кластере может выполняться независимо от обратившегося кластера, который после обращения продолжает свое выполнение, а для получения обратных параметров требуется обратное обращение и соответствующая взаимная синхронизация. Ядро КЛОС выполняет, следовательно, необходимую буферизацию обращений (параметров обращений) к кластерам. Таким образом, понятие кластера объединяет в себе не только понятия процедуры и данных, но и понятие процесса.

Второе существенное расширение семантики кластера касается проблемы синхронизации. Утверждается, что всякая синхронизация есть синхронизация операций над объектом, поэтому она является частью управления этим объектом и, следовательно, должна выполняться самим кластером-объектом. «Физически» это означает, что кластер–объект сам выбирает или не выбирает для выполнения обращения к различным своим входам (соответствующим операциям) от других кластеров в зависимости от своего состояния.

Асинхронные кластеры являются базовыми объектами КЛОС и атомарными составляющими подсистем. Остановимся более подробно на технике организации и функционирования кластера. Нам потребуются следующие дополнительные «под-объекты»: вход, группа входов, подключение, сегмент, входной: буфер, выходной буфер. Ядро КЛОС, реализующее кластеры, предоставляет шесть примитивов: *ПУСК, СТОП, ПОДКЛЮЧЕНИЕ, ОТКЛЮЧЕНИЕ, ЗАГРУЗКА, РАЗГРУЗКА.*

В каждом кластере описаны один или более входов. Каждый вход соответствует семантически отдельной операции в кластере. Описание входа не означает, что сразу после порождения кластера к нему можно будет обращаться по этому входу; но верно, что к кластеру можно будет обращаться только на описанные входы, когда это будет разрешено.

Все входы кластера распределяются в одну или более групп входов; разные группы могут включать одинаковые входы. Группа входов есть то, к чему другие кластеры могут иметь или не иметь доступа: если кластер имеет доступ к группе входов другого кластера, то он имеет право обращаться на все входы этой группы. Например, кластер, открывший файл

на чтение, имеет доступ к группе входов *(CLOSE, READ)* кластера-файла, а кластер, открывший файл на запись, имеет доступ, к группе входов *(CLOSE, READ, WRITE).*

Такой «доступ» описывается объектом, называемым подключение. Подключение содержит ссылку на группу входов кластера-объекта и ссылку на кластера-клиента, владеющего этим. подключением (доступом к этой группе входов). Например, если файл открыт на чтение для Трех кластеров-клиентов, то будут существовать три подключения к группе входов *(CLOSE, READ)* кластера-файла. Под кластером файла в этих примерах понимается кластер-объект, то есть кластер конкретного файла.

Некоторые группы входов кластера описываются как «статические». Если группа входов описана как статическая, это означает, что после порождения кластера будет разрешен доступ к этой группе входов. «Кому разрешен?» – это уже другой вопрос, обсуждаемый ниже при описании процесса компоновки подсистем. В процессе выполнения кластер имеет возможность динамически разрешать и отменять доступ к своим группам входов. Для разрешения доступа используется примитив *ПОДКЛЮЧЕНИЕ,* создающий подключение к указанной группе входов текущего кластера-объекта; владельцем такого подключения сначала становится сам кластер-объект. В дальнейшем он может передать подключение как параметр обращения другому кластеру-клиенту, становящемуся новым владельцем подключения.

Каждое подключение имеет уникальный идентификатор подключения (идентификатор доступа), сообщаемый в ответ на примитив ПОДКЛЮЧЕНИЕ. Используя в дальнейшем этот идентификатор, кластер-объект может различать обращения на свои входы, выполняемые через разные подключения, то есть различать разных клиентов. Для этого ядро КЛОС сообщает кластеру-объекту в качестве обязательного параметра идентификатор подключения, через которое выполнялось выбираемое объектом обращение от клиента (об одном исключении речь пойдет ниже). Об идентификаторах подключений к статическим группам входов, создаваемым при порождении кластера-объекта, а не в результате примитива ПОДКЛЮЧЕНИЕ, кластеру-объекту, конечно, ничего заранее не известно. Статические группы входов как бы «открыты» для доступа с точки зрения самого кластера-объекта. В качестве примера можно рассмотреть статическую группу входов *(OPEN–READ, OPEN– WRITE).* При обращении на вход, например, *OPEN–READ* кластер-файл выполняет ПОДКЛЮЧЕНИЕ к группе входов *(CLOSE, READ)* и в качестве обратного параметра сообщает обратившемуся клиенту идентификатор подключения к этой группе входов.

Примитив ОТКЛЮЧЕНИЕ может выполняться кластером-объектом с указанием идентификатора созданного ранее подключения. Ядро запрещает в дальнейшем все обращения через указанное подключение, хотя уже «проскочившие», но еще не выбранные объектом, обращения через данное подключение остаются.

Кластер-клиент может лишиться доступа к некоторой группе входов не только по инициативе кластера-объекта, выполнившего примитив ОТКЛЮЧЕНИЕ, но и по собственной инициативе – как автоматический результат выполнения обращений по некоторым входам из группы. Например, в результате обращения на вход *CLOSE* кластер-клиент теряет возможность работы с файлом. Такие входы кластера называются входами

отстройки. Технически обращение на вход отстройки через некоторое подключение приводит к тому, что это подключение передается кластеру-объекту как неявный параметр. Получив этот параметр, кластер-объект оказывается его владельцем и может передать его другому кластеру-клиенту или уничтожить (через примитив ОТКЛЮЧЕНИЕ). Таким способом осуществляется в КЛОС защита входов кластеров и контроль доступа к ним.

Если после обращения кластера-клиента к кластеру-объекту требуется передача «в обратном направлении» обратных параметров, то объект для выполнения такого ответного обращения должен иметь подключение к некоторой группе входов клиента. Ядро в этом месте не налагает никаких ограничений и вообще не различает прямые и обратные обращения. Один из возможных интерфейсов может предусматривать передачу объекту в качестве одного из прямых параметров подключения к группе входов клиента, состоящей из одного входа отстройки. В этом случае гарантируется, что на одно обращение к объекту клиент получит ровно одно ответное обращение. Симметричная схема реализует естественное обобщение сопрограммного взаимодействия («обобщение» заключается в том, что между передачей прямых параметров и получением обратных параметров кластер может выполнять любые действия асинхронно со своим партнером).

Теперь рассмотрим структуру памяти кластера и способы работы с ней. Пока будем считать, что кластеры не имеют общих данных, и что параметры от кластера к кластеру передаются только по значению (определенные послабления в этом вопросе мы рассмотрим ниже).

Каждый кластер состоит по крайней мере из двух сегментов памяти: сегмента программного кода и сегмента локального поля. При выполнении кластера оба эти сегмента должны находиться в виртуальной памяти кластера. (Поскольку **КЛОС** – переносимая система, мы трактуем понятие виртуальной памяти достаточно вольно – это просто адресное пространство кластера). Если архитектура ЭВМ позволяет, разные экземпляры одного кластера (разные кластеры-объекты, соответствующие одному кластеру-программе) используют общий программный сегмент. Если архитектура ЭВМ требует наличия стека, то он выделяется в локальном поле кластера. Сама КЛОС не использует стековую организацию памяти (конечно, для организации процедур **внутри** кластера может быть полезен стек, но это дело **компиляторов).**

Кластер может включать произвольное количество дополнительных динамических сегментов памяти. Эти сегменты помещаются в виртуальную память не автоматически, а по явному примитиву ЗАГРУЗКА: в ответ ядро сообщает виртуальный адрес начала загруженного в виртуальную память сегмента. Обратный примитив РАЗГРУЗКА не обязательно приводит к реальному изъятию сегмента из виртуальной памяти (к тому же не для всякой архитектуры ЭВМ возможно динамическое изменение адресного пространства). Этот примитив служит для информации ядра о том, что то место в виртуальной памяти, которое занимал сегмент, может быть освобождено и повторно использовано для загрузки других сегментов. Вообще эти примитивы работы с сегментами нужны прежде всего для независимости от малого размера виртуальной памяти кластеров на некоторых ЭВМ. Для машин с большой виртуальной памятью все

сегменты кластера можно постоянно хранить в ней и примитивы ЗАГРУЗКА и РАЗГРУЗКА будут вырожденными.

Каждый динамический сегмент имеет уникальный идентификатор сегмента, который и указывается как параметр примитивов ЗАГРУЗКА и РАЗГРУЗКА. Сегмент может быть передан другому кластеру как параметр обращения; в этом случае сегмент автоматически разгружается из виртуальной памяти обратившегося кластера, если он был в нее загружен, и меняет своего владельца.

Распределением памяти в системе занимается привилегированный кластер, реализующий запрос и возврат сегментов. Стоит отметить, что эти операции являются не примитивами ядра, а операциями (входами) **кластера** (хотя и привилегированного).

Теперь подробно рассмотрим, как же кластер-клиент выполняет обращения к кластеру-объекту, и как последний выбирает это обращение для выполнения. Для этого используются два примитива: ПУСК и СТОП.

Примитив ПУСК выполняется в кластере-клиенте и означает: «передать указанные параметры кластеру-объекту через указанное подключение **на** указанный вход». Все параметры ПУСКа должны располагаться в специальном выходном буфере параметров. Прежде всего там указываются идентификатор подключения и вход (в виде относительного номера входа в группе входов, на которую ссылается подключение), на который выполняется обращение. Далее перечисляются передаваемые параметры. Все параметры передаются только по значению! С точки зрения ядра КЛОС существуют параметры трех типов: подключения, сегменты и значения данных. Передача подключений и сегментов контролируется ядром: проверяется, что кластер-клиент является владельцем передаваемых подключений и сегментов, и происходит смена владельца – им становится кластер-получатель. Значения данных передаются ядром прозрачно. Выполняя примитив ПУСК, ядро осуществляет буферизацию параметров вне памяти кластера-клиента. При возврате в кластер выходной буфер вновь готов к использованию, но его содержимое неопределено.

Примитив СТОП выполняется в кластере-объекте и означает: «принять обращение по одному из указанных входов». Параметром СТОПа является упорядоченный список номеров входов кластера (необязательно всех). Если на один из указанных входов имеется буферизованное ядром обращение, то параметры этого обращения записываются ядром во входной буфер кластера; происходит возврат в кластер. Входных буферов в кластере может быть несколько, но каждому входу соответствует только один входной буфер, хотя нескольким входам может соответствовать общий входной буфер. Параметры сохраняются во входном буфере до следующего примитива СТОП, который выберет обращение на вход, ассоциированный с тем же входным буфером.

Кластер имеет возможность в качестве дополнительного параметра СТОПа указать идентификатор подключения к своей группе входов; в этом случае ядро выбирает только те обращения, которые выполнялись через указанное подключение. Это может использоваться, например, для «чистки» обращений, успевших «проскочить» до примитива ОТКЛЮЧЕНИЕ и еще не выбранных объектом.

Если нет обращений, параметры которых можно было бы передать кластеру в ответ на примитив СТОП, то обычно ядро переводит кластер в состояние «остановлен» до появления первого такого обращения. Существует модификация СТОПа, при которой кластер не останавливается, а получает возврат из ядра с кодом ответа «обращений нет».

Выходной и входные буфера являются частью локального поля кластера.

Таким образом для реального обращения из кластера-клиента к кластеру-объекту нужно выполнить примитив ПУСК в клиенте и примитив СТОП в объекте (для передачи обратных параметров нужно обратное обращение и, соответственно, ПУСК в объекте и СТОП в клиенте). При этом ядро должно выполнить перепись параметров из выходного буфера клиента, во входной буфер объекта, возможно, с промежуточной буферизацией на памяти самого ядра. Максимально будет выполнено две переписи параметров. Однако, для некоторых архитектур ЭВМ возможна реализация вообще без переписи. При страничной организации виртуальной памяти, например, можно сделать выходные и входные буфера кратными странице и вместо переписи содержимого страниц осуществлять подмену самих страниц – эта операция тоже будет выполняться дважды.

Для некоторых целей, в частности, для эффективной обработки прерываний от устройств ввода/вывода и внешней памяти, было бы полезно еще больше сократить накладные расходы на обращение к кластеру за счет сведения к минимуму передаваемых параметров. Некоторые входы в кластер объявляются входами «прерывания» (эти входы не могут быть одновременно входами отстройки) – при обращении на такой вход кластер-объект не получает никаких параметров (в том числе и обязательного в других случаях идентификатора подключения, через который выполнялся ПУСК), кроме самого факта обращения на данный вход. Для таких входов ядру не требуется сложной буферизации обращений; достаточно иметь при входе счетчик обращений, который увеличивался бы на «I» при ПУСКе на этот вход и уменьшался бы на «I» при выборке обращения по этому входу в результате СТОПа.

В подсистеме управления вводом/выводом каждому внешнему устройству соответствует свой кластер устройства. Прерывание от устройства, поступающее в ядро, универсально интерпретируется им как пуск на вход прерывания кластера устройства.

2. Подсистемы

При помощи базовых объектов – кластеров – реализуются все остальные функции как самой операционной системы, так и надстраиваемых над ней программных комплексов. В КЛОС поддерживается универсальный способ структурирования, для чего вводится формализованное понятие подсистемы.

Подсистема – это статически определенный набор компонентов со статически установленными связями между ними. Компонент подсистемы – это подсистема; элементарная подсистема содержит один кластер. Процесс построения подсистемы из компонентов происходит до ее использования и называется компоновкой подсистемы. Существует специальный язык компоновки подсистем, на котором можно описывать

подсистемы. Описание распадается на две части: описание внешнего интерфейса подсистемы и описание состава и внутренней структуры подсистемы.

Внешний интерфейс подсистемы (в частности, кластера) содержит описание групп входов подсистемы и описание ее внешних подключений. Группы входов подсистемы – это те «функции», для реализации которых она и создана; если кластер реализует атомарный неразложимый далее объект управления, то подсистема – это объект сложный, составной. Группы входов подсистемы – это группы операций над этим объектом. Группа входов подсистемы отождествляется с некоторой группой входов компонента подсистемы и, в конечном счете, реализуется группой входов некоторого кластера.

Для того, чтобы взаимодействовать с подсистемой, выполняя ПУСКи на ее входы, достаточно знать описание групп входов подсистемы. Однако, для того, чтобы подсистему можно было использовать как «строительный блок» – компонент более сложных подсистем, необходимо определять не только ее «входы», но и ее «выходы»– подключения к внешним для подсистемы объектам. В различных случаях такого использования подсистемы она попадает в различный контекст и ее требования к окружающей операционной среде удовлетворяется, вообще говоря, по-разному. Эти требования как раз и сформулированы в описании внешних подключений: это те подключения, владельцами которых являются кластеры подсистемы, а объектами – кластеры вне подсистемы, причем имеются в виду статические подключения. В кластерах–владельцах идентификаторы этих подключений оказываются неопределенными до момента определения внешних подключений подсистемы при её использовании как компонента более сложной подсистемы.

Теперь понятно, что описание внутренней структуры подсистемы как раз и предназначено для определения внешних подключении ее компонентов – эти подключения либо «замыкаются» на статические группы входов других компонентов, либо остаются неопределенными и выводятся «наружу» как внешние подключения компонуемой подсистемы.

На выходе компоновщика подсистем производится загрузочный модуль, содержащий скомпонованную подсистему. Этот модуль можно непосредственно загружать или использовать для компоновки более сложных подсистем.

Для того, чтобы можно было непосредственно загружать подсистемы с неопределенными внешними подключениями, нужно уметь «замыкать» эти внешние подключения в момент загрузки. В КЛОС используется понятие параметризованного загрузчика–это кластер (вообще говоря, подсистема), который имеет подключения ко всем тем статическим группам входов, на которые должны быть «замкнуты» внешние подключения загружаемых этим загрузчиком подсистем. Более подробно, параметризованный загрузчик рассчитан на загрузку определенного класса подсистем в определенном контексте. Этот класс подсистем определяется набором статических групп входов и набором внешних подключений, то есть внешним интерфейсом. Каждая подсистема из данного загрузочного класса должна иметь все те статические группы входов, которые определены для класса, хотя она может иметь и другие статические группы входов (не говоря уже о динамических группах входов). С другой стороны, набор внешних

подключений подсистемы должен быть подмножеством набора внешних подключений загрузочного класса. Загрузчик как раз и параметризуется указанием внешнего интерфейса загрузочного класса: указывается список имен статических групп входов и список имен внешних подключений. Контекст определяется тем, что каждое внешнее подключение класса объявляется внешним подключением загрузчика и при компоновке объемлющей для загрузчика подсистемы, в которую он входит как компонент, происходит «замыкание» этих внешних подключений.

Загрузчик имеет вход СОЗДАТЬ, при обращении на который указывается та подсистема (ее имя в библиотеке), которую **нужно загрузить. Загрузчик** осуществляет контроль того, что эта подсистема из его загрузочного класса, после чего выполняет загрузку. При этом нужные подключения загрузчика просто копируются им и копии становятся подключениями загружаемой подсистемы (понятно, что загрузчик – привилегированный кластер). Для каждой статической группы входов загрузочного класса создается подключение к соответствующей группе входов загружаемой подсистемы (одного из ее кластеров), и идентификаторы этих подключений передаются в качестве обратных параметров на операцию СОЗДАТЬ.

Выбор в качестве единицы загрузки скомпонованной подсистемы с уже установленными внутренними статическими связями, а также загрузка параметризованным загрузчиком с заранее установленными внешними статическими связями позволяют избегать лишней динамики, уменьшая тем самым накладные расходы времени выполнения.

В КЛОС понятие подсистемы используется очень широко, в частности, при управлении процессором. Это управление основано на представлении работающей системы как дерева вложенных подсистем. Корнем дерева является система в целом, а листьями – кластеры. При компоновке подсистемы ее компоненты получают локальные приоритеты, то есть приоритеты в пределах данной подсистемы. Каждый кластер, входящий в подсистему, пусть даже не непосредственно, а через промежуточные по уровню вложенности подсистемы, получает в данной подсистеме вектор приоритетов, соответствующий пути по дереву от этой подсистемы до листа–кластера.

Загрузчик при загрузке подсистемы из своего загрузочного класса присваивает ей тот локальный приоритет, который имеет он сам. Таким образом, каждый загруженный кластер окажется снабженным глобальным в системе приоритетом – вектором приоритетов, соответствующим пути от корня дерева подсистем до листа–кластера, и поэтому, как и в традиционных операционных системах, задачей управления процессором в КЛОС является выбор наиболее приоритетного активного кластера и запуск его выполнения на процессоре.

При многоуровневой приоритетной системе на каждом уровне может возникнуть несколько равноприоритетных компонентов одной подсистемы. Они образуют кольцо равноприоритетных подсистем, между которыми КЛОС квантует процессорное время. Тем самым КЛОС допускает и выполняет многоуровневое квантование. Имеется возможность варьировать частоту квантования: для каждой подсистемы можно указать. число квантов времени, по истечении которых следует перейти к обслуживанию следующей подсистемы в кольце равноприоритетных активных подсистем. В частности,

для некоторых колец равноприоритетных подсистем может быть запрещено квантование указанием «бесконечного» числа квантов, переключение процессора в кольце будет производиться лишь по остановке текущей подсистемы (всех ее кластеров).

Аналогичный подход, основанный на представлении системы как иерархического дерева подсистем вплоть до кластеров, а также на идее «старения» подсистем, используется при управлении подкачкой кластеров и подсистемы в КЛОС.

3. Предопределенные группы входов. Коммунальные подсистемы

Теперь рассмотрим два средства оптимизации, предусмотренные в КЛОС для уменьшения накладных расходов по организации кластерного взаимодействия.

Предопределенные группы входов предназначены для оптимизации выполнения часто встречающихся системных действий. Примером может служить операция ликвидации подсистемы, обратная загрузке подсистемы. Поскольку соответствующий примитив отсутствует, эта операция должна реализоваться как некоторый вход специального привилегированного кластера– ликвидатора. С другой стороны, ликвидация подсистемы, возможно, потребует от нее некоторых специфических действий, которые она может выполнить только сама. Здесь следует различать аварийную ликвидацию подсистемы, которую нужно уметь выполнять при любом ее состоянии, и нормальное завершение ее работы, хотя и выполняемое «по приказу» извне. В последнем случае речь идет фактически о самоликвидации-подсистемы. Если еще учесть, что отсутствует способ явно сослаться на подсистему как целое, то становится понятной интерпретация операции ликвидации подсистемы, принятая в КЛОС.

Каждая подсистема, которую нужно уметь ликвидировать, должна иметь вход ЛИКВИДАЦИЯ. При обращении на этот вход подсистема самоликвидируется. Если это должна быть аварийная ликвидация, то следует считать, что в подсистему автоматически при компоновке добавляется привилегированный кластер–ликвидатор, вход ЛИКВИДАЦИЯ которого становится входом всей подсистемы. Отметим, что ликвидатору в этом случае не надо сообщать, какую подсистему следует ликвидировать, – это всегда та подсистема, в которую ликвидатор входит непосредственно.

Если же нормальная ликвидация подсистемы должна включать некоторые ее индивидуальные действия, то вход ЛИКВИДАЦИЯ отождествляется с соответствующим входом обычного кластера подсистемы. Конечно, этот кластер для завершения самоликвидации подсистемы будет обращаться на вход ЛИКВИДАЦИЯ кластера–ликвидатора, который по-прежнему должен входить в подсистему.

Отметим, что тот внешний кластер, который принимает решение ликвидировать подсистему, в обоих случаях ведет себя одинаково: он выполняет ПУСК на вход ЛИКВИДАЦИЯ подсистемы и он не зависит от способа реализации этой операции. Вообще принцип независимости клиента от способа реализации операции – в объекте – один из самых важных методологических принципов кластерной технологии. Не менее важно и последовательное проведение принципа объединения объекта и программы управления: ликвидация объекта рассматривается как операция, определенная над

объектом, и поэтому реализуемая как вход соответствующей объекту подсистемы (кластера).

Вместе с тем очевидно, что слишком дорого размещать в каждой подсистеме еще один кластер–ликвидатор, даже с учетом того, что все такие кластеры разделяют один сегмент программного кода. Поэтому считается, что такой вход ЛИКВИДАЦИЯ **предопределен** в подсистеме (вообще говоря, не в каждой) – при компоновке подсистемы достаточно указать, что у нее есть такой вход и не указывать способа его реализации. В конечном счете это выльется в наличие специального признака в подключениях к такому входу (предопределенной группе входов). Ядру должны быть известны все предопределенные группы входов и каждой такой группе должна быть поставлена в соответствие группа входов реального системного кластера, который будет выполнять соответствующую операцию (например, ЛИКВИДАЦИЯ) как бы вместо подсистемы. В подключении к предопределенной группе входов в качестве объекта указывается подсистема (в ядре имеется скрытый от пользователя дескриптор для каждой загруженной подсистемы). ПУСК через такое подключение перехватывается ядром и переправляется на соответствующий вход системного кластера; последний, будучи привилегированным, по идентификатору подключения может определить подсистему, к которой относится запрашиваемая операция, и выполнить ее.

Ядро КЛОС может быть построено так, что таблица предопределенных групп входов является параметром генерации системы. Разумеется, такие входы как ЛИКВИДАЦИЯ, всегда являются предопределенными, когда речь идет об аварийной ликвидации.

Если группы предопределенных входов являются средством оптимизации, как правило, некоторых системных действий и используется это средство администратором системы, то коммунальные подсистемы – общедоступное средство оптимизации.

Выше отмечалось, что передача параметров типа подключений и сегментов из кластера в кластер происходит под контролем ядра КЛОС и со сменой владельца. Это довольно значительные накладные расходы и на них можно идти, если происходит отладка подсистемы или нет уверенности в правильности работы всех ее компонентов. Однако, если подсистема уже отлажена, то хотелось бы уметь отменить контроль межкластерных взаимодействий **внутри** подсистемы, хотя и оставить его при внешних взаимодействиях как защиту от других, возможно, еще неотлаженных подсистем.

Для этой цели КЛОС поддерживает наряду с обычными так называемые коммунальные подсистемы. Коммунальная подсистема (К-подсистема)–это подсистема со снятой межкластерной защитой. Это означает, что, во-первых, владельцам всех подключений и сегментов (кроме непередаваемых сегментов кластеров – сегментов программного кода и локальных полей) являются не кластеры по отдельности, а К-подсистема в целом, или, если угодно, каждый кластер этой К-подсистемы. Во-вторых, при ПУСКах между кластерами внутри одной К-подсистемы ядро не выполняет контроля передаваемых параметров – идентификаторы подключений и сегментов передаются так же прозрачно, как и значения данных. Это может привести, конечно, к передаче идентификатора «чужого» (не принадлежащего К-подсистеме) сегмента или подключения, если в подсистеме все же остались такие ошибки. Тогда ошибка будет поймана позже – при

попытке кластера–получателя использовать полученный параметр: выполнить ПУСК по подключению или загрузить сегмент в виртуальную память.

Коммунальная подсистема – это подсистема «общей ответственности» входящих в нее кластеров. Появление этого понятия может быть объяснено некоторыми «ножницами» между требованиями защиты и асинхронного выполнения: внутри единицы защиты (К-подсистемы) могут существовать параллельно выполняемые процессы (кластеры). Аварийная ликвидация кластера автоматически приводит к аварийной ликвидации объемлющей коммунальной подсистемы. Вложенности коммунальных подсистем быть не может.

Заметим попутно, что аварийная ликвидация подсистемы сопровождается ПУСКом без дополнительных параметров по входу отстройки для каждого принадлежащего подсистеме подключения к внешним объектам. Таким способом разрешаются возможные тупики синхронизации.

Есть два способа скомпоновать коммунальную подсистему. Первый наиболее универсальный, способ заключается в следующем. Подсистема компонуется и отлаживается как некоммунальная, то есть с полным контролем межкластерного взаимодействия. После отладки подсистема подвергается «коммунализации», то есть попросту объявляется коммунальной.

Другой подход предполагает конструирование К-подсистемы именно как коммунальной, явно рассчитывая на общность ресурсов (сегментов и подключений). В частности, появляется возможность одновременного доступа к общей памяти: загрузка сегмента в виртуальную память сразу нескольких кластеров внутри К-подсистемы. Мы допускаем такой способ работы, хотя им следует пользоваться крайне осторожно и лишь в тех редких случаях, когда это дает большой выигрыш в эффективности. Следует отметить, что ядро принципиально не предоставляет каких-либо специальных средств синхронизации по доступу к общим сегментам в коммунальной подсистеме. Разумеется, обычная кластерная синхронизация (ПУСКи друг друга) может использоваться для этой цели, но стоит подчеркнуть, что внутри К-подсистемы синхронизация отделена **от** защиты, **то** есть не является, как обычно, защищенной.

4. Основные черты кластерной технологии программирования

Приведенный выше краткий обзор особенностей КЛОС показывает, что при разработке подсистем на базе КЛОС – как подсистем самой операционной системы, так и подсистем «пользователя» – целесообразно использовать естественную технологию КЛОС. Это кластерная технология построения иерархических подсистем.

Базовый объект технологии – кластер – обладает следующими крайне важными особенностями: ориентацией на объектное программирование, асинхронностью выполнения, адекватной синхронизацией и защищенностью.

Особенно существенна связь объекта с синхронизацией операций над объектом, поскольку обычно наиболее трудно обнаруживаемые ошибки в асинхронных подсистемах

связаны именно с неверной синхронизацией. При централизации всех синхронизационных действий в самом объекте синхронизации ошибки будут находиться проще и их труднее будет сделать. Более того, защищенность касается не только памяти кластеров (отсутствие общих данных и передача параметров по значению), но и межкластерных ссылок (подключений) и взаимной идентификации кластеров. Механизм подключений, защищаемых ядром от кластеров, в сочетании с возможностью гибкой перенастройки связей обеспечивает удобную защиту такого рода.

Вместе с тем подход к защите в КЛОС не страдает догматизмом – в отлаженных подсистемах межкластерная защита может быть снята без переделки программ.

Указанные свойства базового объекта сохраняются и для составных объектов – подсистем. По своему внешнему интерфейсу подсистема вообще неотличима от кластера и это является отражением важного принципа независимости клиента от способа реализации объекта. Можно ли реализовать объект в виде одного кластера или нужна внутренняя асинхронность в объекте и потребуется создать целую подсистему – это никак не влияет ни на те кластеры–клиенты, которые будут обращаться на входы объекта, ни на программиста, использующего объект как компонент своей подсистемы.

Разработка подсистемы может вестись как сверху вниз, так и снизу вверх. Иерархическая структура подсистемы, которая, в частности, может отражать этапность разработки, остается и в конечном продукте и используется КЛОС.

В настоящей статье мы не имеем возможности останавливаться на языковой поддержке кластерной технологии программирования. Следует лишь упомянуть, что должна существовать система проектирования, позволяющая, в частности, создавать временно «пустые» подсистемы для использования их как компоненты более сложных подсистем. В дальнейшем, конечно, такие «пустые» подсистемы должны заполняться конкретными программами. Такой подход особенно важен при программировании «сверху вниз».

Система проектирования содержит и язык компоновки подсистем. Для эффективного использования возможностей КЛОС, конечно, нужны соответствующие расширения языков программирования – тех, в которых не предусмотрено асинхронное выполнение, понятия близкие кластеру и т. п. В частности, такое расширение создается для языка «С», на котором в основном пишется КЛОС.

5. Файловая подсистема

Файлы имеют большой диапазон применения и используются, например, самой операционной системой для хранения загрузочных модулей, компиляторами – как рабочие пространства на внешней памяти, пользователями – для хранения текстов программ или документации. Над системами файлов строятся другие информационные системы, более высокого уровня, такие, как системы управления базами данных (СУБД).

Основным объектом, поддерживаемые файловой подсистемой (ФП), является базовый файл, обмен с которым происходит блоками фиксированного размера. Все

структурированные организации файлов реализуются на основе базового файла, соответствующие кластеры не входят в ФП и являются кластерами–посредниками.

Рассмотрим теперь более подробно подсистему управления файлами в КЛОС. Понятие тома, используемое здесь, реализуется подсистемой ввода/вывода, скрывающей физические особенности накопителей и способ доступа к ним. Размер тома ограничен размерами дискового пакета; том занимает непрерывную область на пакете. Подсистема ввода/вывода дает возможность обмениваться с группой смежных блоков тома. Пользователи системы ввода/вывода, одним из которых является ФП, оповещаются о постановке и снятии каждого тома. Если файловая подсистема опознает предложенный том как свой, она запрашивает этот том у подсистемы ввода/вывода, после чего выполняет собственную процедуру подключения этого тома к себе.

Сначала рассмотрим организацию файлов на томе. Наличие внешних запоминающих устройств прямого доступа большой емкости и необходимость создания на них больших файлов для таких применений, как СУБД, обязывает ФП к эффективной работе с большими файлами. Это возможно, во-первых, только в том случае, когда для открытого файла вся таблица отображения логических номеров блоков файла в физические адреса блоков на томе всегда целиком находится в оперативной памяти (ОЗУ). Поэтому в ФП память на томе распределяется непрерывными физическими областями, превышающими по размеру один физический блок, – экстентами. Файл занимает одну или несколько таких областей.

Во-вторых, подсистема допускает за один обмен чтение/запись нескольких смежных блоков файла. Это позволяет, с одной стороны, сделать блок файла минимально возможным, равным физическому, что приводит к более рациональному использованию внешней памяти и ОЗУ, с другой – используя заложенную в аппаратуре возможность обмениваться за один раз с группой смежных блоков на диске, увеличить пропускную способность подсистемы.

По характеру выделения памяти файлы могут быть статическими, когда их пространство образуется при создании файла и остается неизменным, и динамическими, когда разрешается запрашивать и освобождать отдельные области файлов в рамках его максимального объявленного размера.

Организация файла, когда его заголовок, содержащий отображение файла на том, частично дублируется, причем его первый блок, оставаясь в единственном экземпляре, является как бы переключателем с одного экземпляра на другой, – такая организация позволяет поддерживать пространство файла **стойким от сбоев** процессора, приводящих к потере содержимого оперативной памяти (такие сбои будем называть «мягкими сбоями»).

Что касается поддержки логической целостности содержимого файлов, то в них по желанию пользователя может быть создан теневой механизм, являющийся развитием метода рабочей копии в применении к отображению файла экстентами. Этот метод, требуя, вообще говоря, большее количество внешней памяти (в пределе–вдвое большее), позволяет, во-первых, при мягких сбоях сохранить согласованное теневое состояние

файла на момент его последней фиксации. Во-вторых, возможно восстановление этого состояния в любой момент вместо текущего.

Неприменимый в приложении к СУБД, теневой механизм полезен для программ редактирования. Его можно создавать в любом файле, а при прекращении необходимости в нем – ликвидировать, уменьшив занимаемое файлом пространство тома.

Файловая подсистема поддерживает так называемые временные файлы. Они не имеют имени в архиве, создаются в рамках свободной памяти тома, а вся их служебная информация находится в ОЗУ. В случае мягкого сбоя эти файлы пропадают, не требуя никаких дополнительных мер по очистке от них архива при последующей раскрутке системы.

Теперь перейдем к структуре архива файловой подсистемы. Для ФП архив – это ориентированный граф, узлами которого являются справочники и файлы. Из справочника может выходить любое количество именованных связей к другим справочникам и файлам. Ограничений на связи нет (разрешаются даже циклы). Если от справочника к объекту ведет цепочка связей, то обобщенное имя объекта относительно этого справочника состоит из физического идентификатора справочника и логического имени – цепочки имен связей на этом пути. Для пользователя структура физического идентификатора прозрачна.

На томе один из справочников выделен и назван корневым. Имя тома интерпретируется как имя этого корневого справочника, точнее (поскольку имена имеют не справочники и файлы, а лишь связи) как имя связи, ведущей на корневой справочник тома из некоего глобального фиктивного «справочника томов». Физический идентификатор последнего – нулевой код.

Операция настройки на справочник по его собственному имени определяет и выдает физический идентификатор справочника, который в дальнейшем может быть использован для короткого именования объектов через этот справочник.

Существование объектов архива (справочников, файлов) не оговаривается существованием указывающих на них связей, и наоборот. Доступ к объекту всегда возможен по физическому идентификатору, который уникален во времени. Это позволяет определить некорректность ссылки, указывающей на исключенный объект.

Подключение справочников новых томов к централизованному архиву производится с помощью явного проведения связей к ним из справочников архива. Следует отметить, что это не обязательно для работы с объектами тома, поскольку одна связь – из фиктивного «справочника томов» уже существует. Связи на другие тома проводятся косвенно, через таблицу внешних связей тома. Это обеспечивает достаточную гибкость. при переименовании томов и перенесении их с одной ЭВМ на другую.

В заключении рассмотрения файловой подсистемы очень коротко остановимся на ее архитектуре. ФП содержит кластеры, соответствующие основным объектам архива –

томам (по числу «подключенных» томов) и файлам (по числу открытых файлов). Кластер–администратор осуществляет коммутацию обращений между кластерами томов.

Рассмотрим как происходит обработка операции «открыть. файл», в которой указано обобщенное имя файла. Операцию начинает выполнять администратор, который определяет нужный том и передает параметры открытия соответствующему кластеру тома. Если том еще не подключен, возможно взаимодействие с оператором системы и после постановки дискового пакета создание кластера тома через параметризованный загрузчик.

Кластер-том осуществляет поиск по логической цепочке имен в пределах своего тома. Если обнаруживается связь на другой: том, то кластеру другого тома – через посредничество администратора – передается физический идентификатор объекта, на который указывает связь, и остаток логического имени. Кластер-том, в котором окажется искомый файл, создаст для него кластер файла (через параметризованный загрузчик) или использует уже существующий (при открытии файла, который уже открыт другими клиентами).

Кластер-файл выполняет *ПОДКЛЮЧЕНИЕ* к своей группе входов, соответствующей режиму открытия, и это подключение передается, в конечном счете, клиенту как ответный параметр открытия файла. Через это подключение клиент будет работать с файлом до тех пор, пока не выполнит операцию закрытия файла.

Статически файловая подсистема состоит из кластера–администратора и загрузчика, параметризованного загрузочным классом «подсистема тома». Подсистема тома, в свою очередь, статически содержит кластер тома и загрузчик, параметризованный загрузочным классом кластера файла. Динамически порождаются лишь подключения к кластерам файлов от кластеров–пользователей файловой подсистемы.

Стоит отметить, что кластерная технология позволяет в данном случае обеспечить необходимую степень асинхронности и в то же время сводит к минимуму синхронизационные расходы. В самом деле, все кластеры файлов работают асинхронно друг с другом, а единственная синхронизация, которая требуется в их совместной работе с общим диском, осуществляется средствами подсистемы ввода/вывода (конкретно – кластером диска). При открытии файлов кластер–том, производя поиск в пределах своего тома, не осуществляет синхрозахватов объектов: он просто не выбирает следующего обращения, пока не найден файл, либо не встретилась связь на другой том. В то же время друг относительно друга, так же как относительно кластеров файлов, кластеры томов работают асинхронно; тогда же, когда требуется централизованная синхронизация, она осуществляется администратором.

6. Система управления базами данных

Базовый файл является не очень удобным объектом для прямой работы пользователя. Над файловой подсистемой надстраивается подсистема структурированных файлов, в которой реализуются различные типы структурированных файлов на основе базового файла. В настоящей статье мы не имеем возможности касаться этой подсистемы. С другой стороны,

базовый файл, ввиду своей простоты, оказывается весьма удобным для системы управления базами данных (СУБД). В КЛОС реализуется реляционная модель базы данных, о которой в настоящей статье мы можем сказать очень мало, ввиду ограниченности места.

Структуры данных. Хранимая информация, с которой работает СУБД (пользовательские данные, пути доступа и каталоги), располагается в сегментах. Сегмент–самая крупная единица хранения – реализуется в виде базового файла, если отвлечься от того, что небольшая часть файла используется для ведения учета занятого пространства сегмента.

Сегменты разбиваются на страницы фиксированного размера, который может совпадать с размером блока файла, либо быть кратным ему. Окончательный выбор размера страницы отложен на более поздний этап – опытную эксплуатацию СУБД.

Основным понятием реляционной модели данных является отношение – совокупность кортежей, каждый из которых состоит из одного или более полей постоянного или переменного размера.

В качестве путей доступа к данным используются индексы – инверсии отношений по одному или более полям или совокупности полей. Индексы организуются на отдельных страницах, соединяемых в виде B-дерева с соответствующей поддержкой расщепления, слияния и переливания информации между ними. Такая организация позволяет получить как прямой доступ к совокупности кортежей с заданным значением поля, так и последовательно сканировать кортежи в порядке увеличения или уменьшения значения данного поля.

Страница сегмента может содержать кортежи только одного отношения, причем размер кортежа не должен превышать размер страницы. Отношение вместе со своими индексами должно находиться в одном сегменте, в то время как сегмент может содержать несколько отношений. Каждый кортеж имеет уникальный в системе идентификатор, который он получает при помещении в страницу, и который используется для доступа к кортежу.

Описание полей каждого отношения оформляется в виде кортежа специального отношения – каталога. Существует аналогичное отношение описателей индексов. Идентификаторы кортежей этих отношений используются для именования соответствующих макрообъектов внутри системы. Кроме технологических удобств, которые дает представление служебной информации в виде отношений, оно удобно и для пользователей.

Основной способ повышения эффективности систем – кластеризация отношений. Отношение называется кластеризованным по некоторому полю, если при последовательном сканировании его по возрастанию/убыванию значений этого поля потребуется столько же чтений страниц, сколько занимает само отношение. Кластеризации добиваются, занося кортежи в нужном порядке при генерации отношения. Для того, чтобы сохранить ее при последующей работе, необходимо оставить в страницах резервное пространство, но полной гарантии это не дает. Если кластеризация все же будет

нарушена, воссоздать ее можно только лишь с помощью утилиты путем перезагрузки отношения.

Для того, чтобы воспользоваться кластеризацией в полной мере, необходимо иметь индекс на поле (полях), по которому отношение кластеризовано.

Подсистема управления памятью и синхронизацией (УПС). Эта подсистема представляет собой нижний уровень СУБД, призванный обеспечить возможность параллельного выполнения нескольких транзакций и восстановления баз данных (БД) после различного типа сбоев. С этой целью используется механизм синхронизации по доступу к объектам и ведется журнализация всех изменений в БД.

УПС обеспечивает интерфейс, достаточно низкого уровня, включающий команды добавления, исключения, обновления кортежей отношений, создания и уничтожения отношений и индексов, добавления полей к существующим отношениям, установления контрольных точек и отката до указанной контрольной точки, а также сканирования отношений с использованием индексов или без них.

Подсистема может работать в нескольких режимах, в зависимости от режима меняется ее конфигурация. Мы рассмотрим следующие режимы работы УПС: прямой, восстановление после мягких сбоев, восстановление после жестких сбоев, системную контрольную точку.

Постоянно присутствующий компонент УПС – кластер–администратор – управляет режимами работ, создает и ликвидирует кластеры, обслуживающие эти режимы, по обращениям извне (например, от оператора), а также в зависимости от внутренней операционной обстановки (обнаружение сбоя).

В режиме прямой работы в УПС входят кластеры Администратор, Журнал, Синхронизатор, а также кластеры Транзакции по числу одновременно работающих транзакций.

Возникновение каждой новой транзакции сопровождается обращением к Администратору, по которому он создает новый экземпляр кластера Транзакция (через загрузчик, параметризованный соответствующим загрузочным классом). Кластер Транзакция реализует практически весь интерфейс подсистемы, осуществляя в ходе выполнения операций обращения к различным кластерам.

Для осуществления логической синхронизации Транзакция обращается к Синхронизатору, который имеет информацию о захватах каждого объекта. Эти обращения являются синхронными (между ПУСКом и СТОПом для получения ответного обращения Транзакция ничего не делает), так что блокировка транзакции при необходимости выражается просто как отсутствие ответа от Синхронизатора.

Операция кластера Транзакция может привести к изменениям в одной или нескольких страницах БД. Эти изменения журнализуются как постранично (отдельно по каждой странице), так и на уровне внешнего интерфейса УПС. Записи последнего типа кластер Транзакция посылает, обращаясь к кластеру Журнал непосредственно. Записи первого

типа (постраничные) поступают в журнал другим путем, связанным с передачей страниц в подсистеме. Опишем его подробнее.

Кластер Буфер осуществляет буферизацию используемых в подсистеме страниц, храня каждую из них в сегменте памяти КЛОС такого же размера. (Это одна из причин отсутствия буферизации обменов с внешней памятью на более низком уровне – в файловой подсистеме. Отметим, что в подсистеме структурированных файлов имеется своя буферизация). В ходе выполнения кластером Транзакция операций может происходить передача страниц от Буфера к Транзакции (запрос) и обратно (возврат). По запросу страницы Буфер либо находит ее в свое пуле, либо производит замещение одной из страниц пула, задавая один или два обмена с внешней памятью в зависимости от того, модифицировалась ли замещаемая страница (алгоритм замещения есть разновидность алгоритма *LRU).* Страницы в зависимости от назначения запрашиваются в монопольном или совместном режиме, чтобы предотвратить возможное рассогласование содержащейся в них информации при параллельной работе нескольких Транзакций.

Для того, чтобы совместить передачу страницы с указанными примитивами физической синхронизации, она передается от Буфера к Транзакции и обратно через посредничество Синхронизатора.

Вместе с возвращаемой после изменения страницей Транзакция передает информацию об изменении в ней, оформленную в виде записи в журнал. Эта запись, попав к Буферу, направляется им в кластер Журнал. Последний накапливает попадающие к нему записи в порядке поступления и периодически большим массивом сбрасывает их на внешнюю память в отдельный файл, который служит журналом.

В результате одновременной работы нескольких транзакций возможно попадание двух или более из них в тупик. УПС производит распознавание тупиков только по внешнему обращению (например, от оператора или по времени). При таком обращении Синхронизатор создает кластер Детектор, который на основе информации Синхронизатора выявляет наличие или отсутствие тупика, и в первом случае сообщает список транзакций и их контрольных точек, откат до которых нужно выполнить в целях разрушения тупика. После этого Детектор ликвидируется.

Транзакции, которые должны быть откачены, будучи до этого заблокированными, вместо ответа на обращение по синхрозахвату получат от Синхронизатора номер контрольной точки и сами произведут откат, обращаясь к Журналу за командами, которые они ранее сами же записывали в журнал, и производя действия, обратные им.

Такие же действия будет выполнять Транзакция, получив извне индивидуальную команду откатиться.

УПС периодически (точнее, по концу каждой транзакции) фиксирует состояние БД по журналу так, чтобы оно соответствовало в точности тем изменениям, которые уже отражены в сегментах на внешней памяти. Эта процедура, называемая фиксацией, сводится сначала к сбросу накопленных в кластере Журнал записей, затем к сбросу

страниц из Буфера и, наконец, отметки в журнале о фиксации. Эти точки используются затем при восстановлении БД после сбоев.

Сбои, требующие восстановления, подразделяют на две категории, исходя из характера производимой при восстановлении работы: поломка на диске (жесткий сбой) и сбой процессора, приводящий к потере содержимого ОЗУ (мягкий сбой). Восстановление после жесткого сбоя называют «холодным рестартом», после мягкого – «горячим рестартом».

Ситуации, в которых начинаются холодный и горячий рестарты, различны. Холодному рестарту предшествует восстановление с магнитных лент содержимого БД, которое соответствует состоянию БД в момент начала журнала (системная контрольная точка).

При холодном рестарте, продвигаясь вперед по журналу, УПС производит (постранично) изменения, указанные в записях журнала, пока не дойдет до точки последней фиксации.

В начале горячего рестарта содержимое БД представляет собой рассогласованное на момент сбоя состояние. В этом режиме необходимо, отыскав точку сбоя в журнале, произвести все изменения (опять же постранично) в обратном порядке до точки последней фиксации, где состояние БД согласовано.

Последняя стадия обоих рестартов – это ликвидация изменений от незавершенных на момент сбоя транзакций путем прохода назад по журналу (по записям уровня внешнего интерфейса УПС) и выбора изменений только от них.

Конфигурация подсистемы во всех трех режимах восстановления довольно проста и однотипна. Кроме постоянно присутствующих кластеров Администратора, Буфера и Журнала, здесь присутствуют соответственно кластеры Холодный Рестарт, Горячий Рестарт или Ликвидатор, создаваемые Администратором в начале соответствующего этапа работы. По концу восстановления Администратор переводит УПС в прямой режим.

Системная контрольная точка – это момент копирования всего содержимого БД на ленты. После перевода УПС приказом оператора в этот режим Администратор приостанавливает создание новых кластеров Транзакций. После окончания текущих транзакций и выталкивания буферов создается кластер Копия, который и осуществляет копирование. После окончания копирования УПС переводится в прямой режим работы, а журнал ведется от начала.

Подсистема языкового уровня СУБД. Эта подсистема является второй подсистемой в рамках СУБД, взаимодействующей с УПС через ее внешний интерфейс. В качестве базового языка общения с СУБД выбран некоторый диалект языка SQL. *SQL* – мощный язык для работы с реляционными БД. С помощью его конструкций, например, легко выражаются все основные операции реляционной алгебры, но выражения языка *SQL* гораздо более наглядны и выразительны, чем соответствующие выражения в алгебраической форме.

Подсистема языкового уровня (ЯУП) выполняет роль прекомпилятора для текстов программ, написанных на традиционных языках программирования с включениями предложений языка SQL.

В целом ЯУП выполняет следующие функции: выбирает из объемлющей программы предложения SQL, производит их компиляцию, создавая в конечном счете кластер доступа, включающий программы, выполняющие соответствующие предложения, и заменяет в исходном тексте предложения SQL на обращения к кластеру доступа в интерфейсе объемлющего языка. Вкратце схема работы подсистемы следующая.

Выбранный текст предложения SQL передается кластеру грамматического разбора, который осуществляет синтаксический анализ полученного предложения и производит его внутреннее представление, включающее древовидную структуру, соответствующую синтаксису предложения, и таблицу имен объектов БД, употребленных в предложении. В случае нарушения синтаксиса кластер выдает развернутую диагностику и осуществляет подавление некоторых ошибок.

Внутреннее представление предложения SQL передается кластеру замены имен, который, взаимодействуя с УПС, на основании информации в каталогах-отношениях вырабатывает вместо имен отношений идентификаторы описывающих их кортежей, а вместо имен полей – их номера. При этом кластер производит контроль прав доступа пользователя, от имени которого происходит компиляция.

После этого кластер выбора пути создает внутреннее процедурное представление предложения SQL в расчете на интерфейс УПС. Выбор пути доступа производится на основе вида SQL предложения, наличия в БД подходящих путей доступа (индексов), кластеризованности отношений и прямых указаний пользователей. Указания содержат информацию о предпочтительности использования тех или иных индексов и о предпочтительности порядка сканирования отношений при выполнении некоторых операций.

Кластер генерации кодов получает внутреннее процедурное представление запроса и генерирует коды процедур выполнения этого запроса, а также компонует из этих процедур кластер доступа. Последний создается динамически при выполнении операции SQL «начало транзакции» и взаимодействует (по мере поступления к нему обращений) с УПС синхронно.

При выполнении запросов часто требуется производить сортировки. Для этих целей используется системный кластер внешних сортировок со слиянием, работающий на временных файлах файловой подсистемы. Результаты сортировок остаются *во* временных файлах.

ЛИТЕРАТУРА

Иванников В. П. Использование кластеров в операционной системе.–ДАН СССР, 1977, т. 237, № 2, 280–283.