ПРИНЦИПЫ РЕАЛИЗАЦИИ СИСТЕМЫ УПРАВЛЕНИЯ ФАЙЛАМИ В ПАРАЛЛЕЛЬНОЙ СУБД ОМЕГА ДЛЯ МВС-100*

Л.Б. Соколинский, М.Л. Цымблер
Челябинский государственный университет

Аннотация. В статье описываются принципы разработки и программная структура системы управления файлами (СУФ) ядра параллельной СУБД Омега для отечественного суперкомпьютера МВС-100. Формулируются требования к СУФ. Даются описание общей структуры СУФ и описание ее компонент. Предлагается стратегия вытеснения страниц, основанная на динамических и статических рейтингах страниц. Описываются алгоритмы простой выборки страниц и выборки с упреждением. Предлагается эффективный протокол для взаимодействия с дисковой подсистемой. Описывается архитектура эмулятора дисковой подсистемы. Описанная СУФ реализована на МВС-100 и допускает перенос на МВС-1000 при минимальных доработках.

Ключевые слова: система баз данных, управление файлами, алгоритм вытеснения страниц, параллельная СУБД, мультипроцессорные системы с массовым параллелизмом.

1. Введение

Высокопроизводительные параллельные системы баз данных в настоящее время являются одним из основных приложений многопроцессорных вычислительных систем с массовым параллелизмом [1]. Хотя в данной области достигнут существенный прогресс и уже имеются промышленно эксплуатируемые системы (см., например, [2, 3, 4]), в области параллельных баз данных остается много открытых вопросов и нерешенных проблем [5]. Одной из таких проблем является выбор аппаратной архитектуры для параллельной СУБД.

В соответствие с классификацией Стоунбрейкера [6] аппаратная архитектура параллельных СУБД традиционно делилась на три класса, - архитектура с разделяемой памятью, архитектура с разделяемыми дисками и архитектура без совместного использования ресурсов. Сравнительный анализ данных архитектур показал, что каждая из них имеет свои преимущества и недостатки [6, 7]. Так, например, системы с разделяемой памятью имеют наилучшие показатели по производительности. Однако такие системы плохо масштабируются и отличаются относительно высокой стоимостью. Системы без совместного использования ресурсов, наоборот, хорошо масштабируются, но для таких систем трудно достичь хорошего баланса загрузки [8].

В соответствие с этим, в настоящее время сложилась тенденция к использованию систем с гибридной или иерархической архитектурой [9, 10]. Гибридные архитектуры обладают большим многообразием, чем традиционные, и, в настоящее время, являются еще недостаточно исследованными.

В работе [11] была предложена двухуровневая иерархическая архитектура без совместного использования ресурсов. Данная архитектура была использована при разработке параллельной СУБД Омега на платформе суперЭВМ МВС-100.

Отечественная суперЭВМ МВС-100 [12, 13] разработки НИИ "Квант", ИПМ им. М.В. Келдыша РАН и ИММ УрО РАН представляет собой многопроцессорную систему с

* Работа выполнена при поддержке Российского фонда фундаментальных исследований (грант No. 97-07-90148).

модульной архитектурой. В соответствие с таксономией Флинна [14] МВС-100 относится к классу MIMD архитектур.

Основу МВС-100 составляют процессорные модули стандартной структуры [13]. Структура процессорного модуля схематично изображена на Рис. 1. Каждый модуль представляет собой двухпроцессорную ЭВМ, состоящую из вычислительного и коммуникационного процессоров. Коммуникационный и вычислительный процессоры разделяют общую статическую память **SRAM** объемом 16-32 Мбайт. Помимо этого, коммуникационный процессор имеет свою собственную более быструю динамическую память **DRAM** объемом 4 Мбайта. Синхронизация коммуникационного и вычислительного процессоров реализована аппаратно. Единственными внешними устройствами процессорного модуля являются скоростные двунаправленные каналы – линки, имеющиеся у коммуникационного процессора. Коммуникационный процессор имеет четыре линка. С помощью этих линков процессорные модули соединяются друг с другом и с дисковой подсистемой. Система может включать в себя сотни вычислительных модулей, образующих сеть процессорных элементов. Полученная таким образом сеть связана одним своим линком с управляющей ЭВМ (Host-машиной), представляющей собой IBM PC совместимую рабочую станцию.

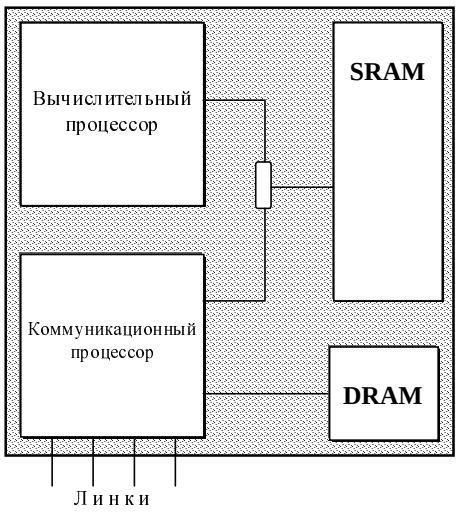

Рис. 1. Структура процессорного модуля МВС-100

В качестве операционной системы на МВС-100 используется ОС Router разработки ИПМ им. М.В. Келдыша РАН. В качестве операционной системы Host-машины обычно используется ОС UNIX/Linux. ОС Router, поставляемая с МВС-100, не предоставляет фактически никаких средств для работы с файлами дисковой подсистемы. Отсюда возникает необходимость разработки специальной системы управления файлами для СУБД Омега.

Остаток статьи организован следующим образом. Раздел 2 содержит обзор аппаратной и программной архитектуры системы Омега. В разделе 3 описывается общая структура СУФ, и приводятся требования к СУФ. Раздел 4 содержит изложение принципов проектирования и реализации менеджера дисков, описание эмулятора дисковой подсистемы и описание протокола обмена с дисковой подсистемой. В разделе 5 описываются принципы проектирования и реализации менеджера страниц, а также приводятся алгоритмы простой выборки страниц и выборки с упреждением. Раздел 6 посвящен принципам разработки менеджера файлов. Раздел 7 содержит заключение.

2. Архитектура системы Омега

2.1. Аппаратная архитектура системы Омега

Аппаратная архитектура системы Омега представляет собой двухуровневую иерархическую архитектуру без совместного использования ресурсов.

На первом уровне иерархии процессорные модули объединяются в Омега-кластеры стандартной структуры. Возможная структура Омега-кластера для МВС-100 изображена на Рис. 2. Омега кластер состоит из четырех процессорных модулей, объединенных в кольцо. Каждый процессорный модуль соединен одним линком с дисковой подсистемой. Дисковая подсистема представляет собой особый модуль, включающий в себя коммуникационный процессор, который посредством SCSI шины соединен с четырьмя дисковыми накопителями (по одному на каждый процессорный модуль). Омега кластер имеет четыре свободных линка для связи с другими Омега-кластерами.

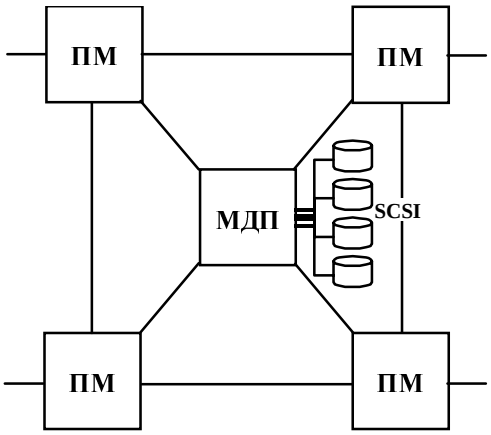

Рис. 2. Структура Омега-кластера
(ПМ - процессорный модуль; МДП - модуль дисковой подсистемы)

На втором уровне иерархии Омега-кластеры связываются в единую вычислительную систему. Топология межкластерных соединений не фиксируется и может быть различной в различных Омега-системах. На Рис. 3 приведен пример возможной топологии соединений, называемой "простой линейкой". Наличие двух Host-машин в системе мотивируется соображениями отказоустойчивости системы в целом. Описанная архитектура является отказоустойчивой по любому аппаратному компоненту (процессор, линк, дисковый накопитель, Host-машина и др.).

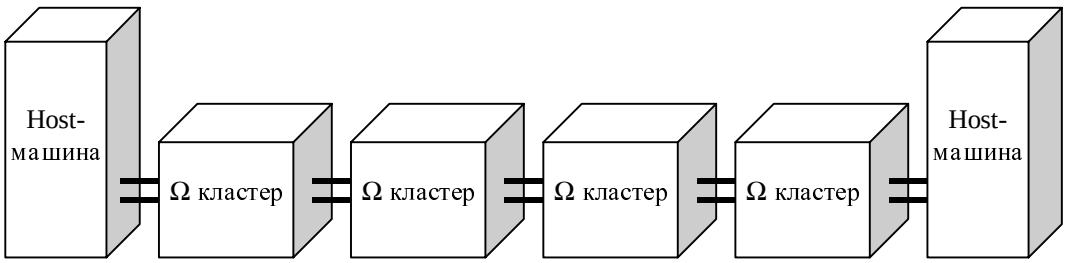

Рис. 3. Возможная конфигурация системы Омега ("простая линейка")

2.2. Программная архитектура системы Омега

Программная структура СУБД Омега изображена на Рис. 4. Программная структура системы Омега имеет два уровня абстракции. Первый уровень абстракции составляет так называемый *слой операционного окружения* ядра СУБД. Во второй уровень входят собственно *ядро СУБД* и *утилиты*.

Операционное окружение СУБД Омега включает в себя следующие подсистемы: модуль топологии, менеджер нитей (легковесных процессов), Омега-кондуктор, Омега-маршрутизатор и менеджер дисков.

В качестве промежуточного слоя между аппаратной платформой и операционным окружением используется ОС Router. ОС Router представляет собой распределенную операционную систему класса toolset. ОС Router обеспечивает следующие основные функции: - загрузку программ пользователя с Host-машины на процессорные модули; - обмен данными между программой и Host-машиной в виде некоторого подмножества системных функций ввода/вывода UNIX; - обмен сообщениями по линкам между программами, запущенными на различных вычислительных процессорах.

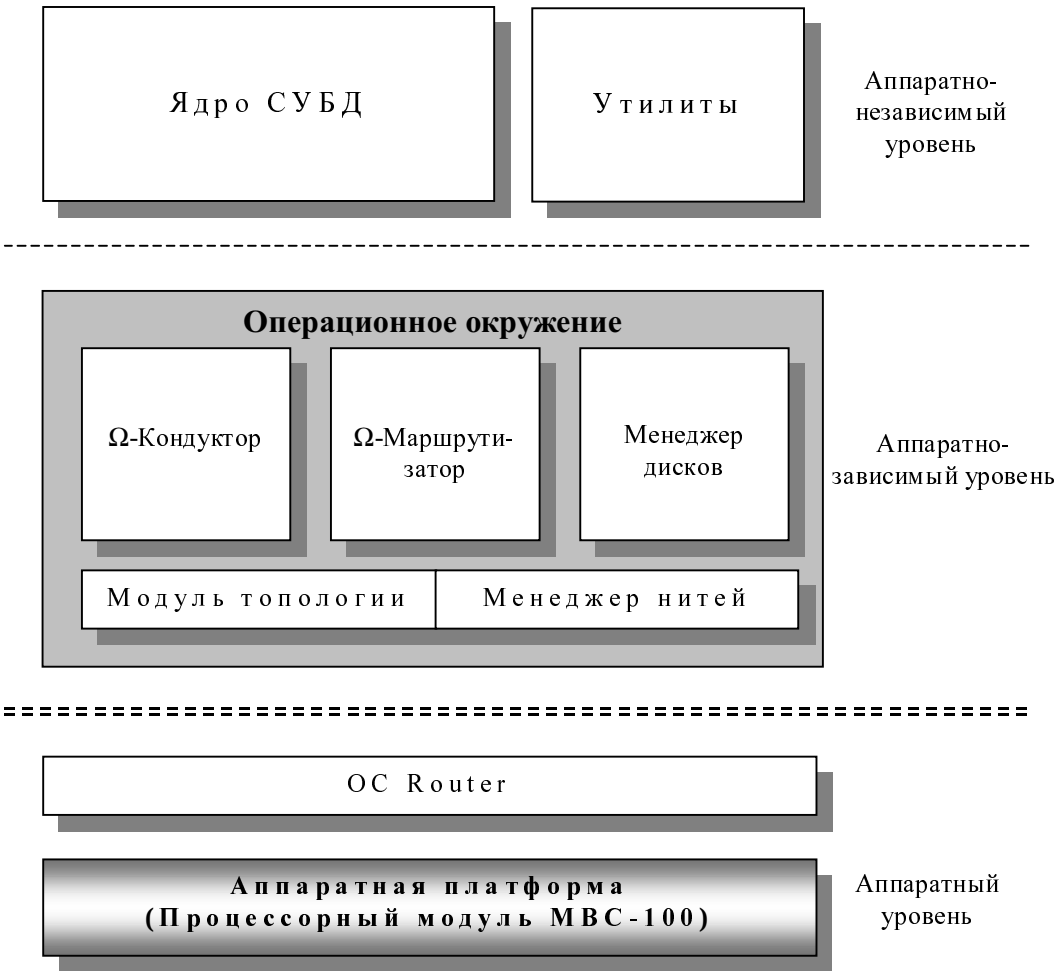

Рис. 4. Программная структура СУБД Омега

Модуль топологии инкапсулирует аппаратные особенности топологии МВС-100 и позволяет рассматривать его как совокупность Омега-кластеров. Адресом процессорного узла, таким образом, является пара: адрес кластера и номер узла в кластере. Реализация модуля топологии является аппаратно зависимой.

Менеджер нитей обеспечивает поддержку легковесных процессов (нитей или потоков управления). Поддерживается два типа нитей: системные нити и пользовательские нити. Системные нити порождаются в операционном окружении. Пользовательские нити порождаются в ядре СУБД и в утилитах. Управления нитями и их синхронизация осуществляются на основе модели производитель-потребитель, описанной в работе [15].

Омега-кондуктор обеспечивает эффективную реализацию внутрикластерных обменов сообщениями на базе механизма виртуальных каналов. Протокол обмена базируется на фиксированной сильносвязанной топологии внутрикластерных соединений [16]. Данный протокол оптимизирован для выполнения большого количества параллельных обменов короткими сообщениями.

Омега-маршрутизатор обеспечивает эффективную реализацию межкластерных обменов сообщениями на базе механизма виртуальных каналов. Протокол обмена оптимизирован для случая сравнительно небольшого количества параллельных обменов длинными сообщениями [16].

Менеджер дисков обеспечивает представление дисковой подсистемы Омега-кластера, как набора виртуальных дисков со страничной организацией, позволяя абстрагироваться от особенностей аппаратной реализации модуля дисковой подсистемы. Менеджер дисков входит в состав СУФ в качестве подсистемы нижнего уровня.

3. Общая структура СУФ

СУФ является связующим звеном между ядром СУБД Омега и дисковой подсистемой, и функционирует на каждом рабочем узле кластера.

Основным назначением СУФ является поддержка понятия файла, как набора записей фиксированной длины. Данные файлы используются на более высоких уровнях системной иерархии для представления отношений (таблиц), индексов и других объектов хранимой базы данных.

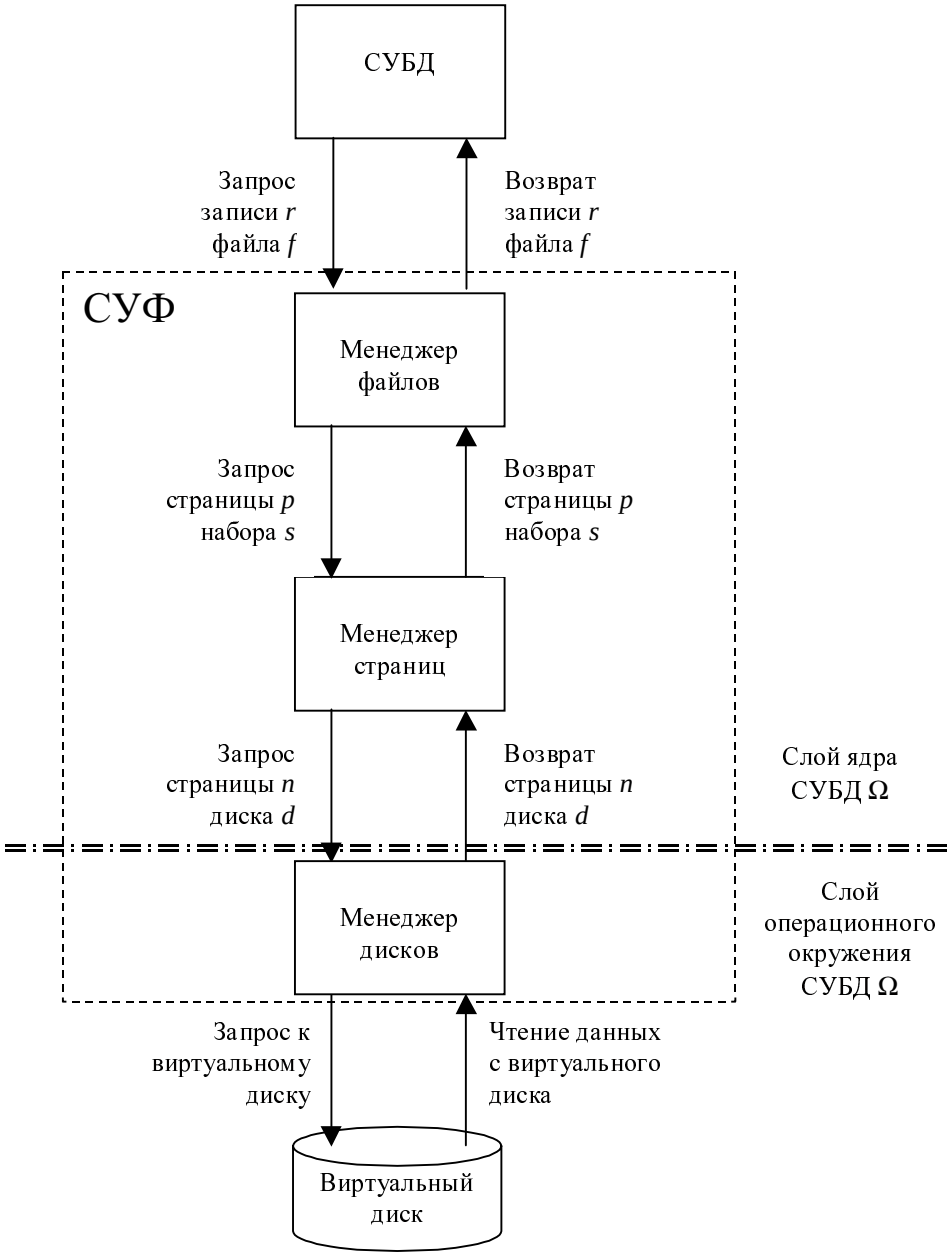

Рис. 5. Общая структура СУФ

Основными требованиями к СУФ являются следующие.

- СУФ должна поддерживать файлы, состоящие из записей фиксированной длины. Каждый файл должен иметь идентификатор, уникальный в пределах диска. Каждая запись файла должна иметь идентификатор, уникальный в пределах диска.
- СУФ должна предусматривать возможность введения внутрифайловой кластеризации на более высоких уровнях системной иерархии. Межфайловая кластеризация в системе Омега не поддерживается.
- СУФ должна поддерживать буферизацию страниц на основе единого внутреннего буферного пула. Доступ к содержимому файла возможен только через буфер.
- СУФ должна поддерживать виртуальную архитектуру без совместного использования ресурсов. Это означает, что за каждым процессорным узлом Омега-кластера

закрепляется отдельный виртуальный диск (дисковый пул). Процессорный узел не вправе обращаться к чужим виртуальным дискам (дисковым пулам).

- СУФ не должна предоставлять средств для работы с дисками, не принадлежащими данному Омега-кластеру.
- СУФ не должна поддерживать фрагментацию и репликацию файлов. Репликация и фрагментация реализуются на более высоких уровнях системной иерархии. Это означает, что каждый фрагмент или реплика отношения (таблицы), с точки зрения СУФ, реализуется в виде отдельного независимого файла.

Общая структура СУФ изображена на Рис. 5. СУФ включает в себя менеджер дисков, менеджер страниц и менеджер файлов.

Менеджер дисков обеспечивает представление виртуального диска в виде набора пронумерованных страниц по 4КБайта. В будущем предполагается добавить возможность, указывать для файла желаемый размер страницы, исходя из имеющихся вариантов (например, 4, 24, или 32 КБайт).

Менеджер страниц обеспечивает представление базы данных (точнее ее части), хранящейся на данном виртуальном диске, в виде совокупности наборов страниц, представляющих собой связные списки. Менеджер страниц позволяет создавать и удалять наборы страниц, добавлять страницы в набор, удалять страницы из набора, осуществлять последовательный просмотр набора страниц, а также прямую выборку страницы с указанным идентификатором.

Менеджер файлов обеспечивает представление базы данных в виде файлов, представляющих собой наборы неструктурированных записей одинаковой длины (запись имеет только одно информационное поле *info*). Менеджер файлов позволяет создавать и удалять файлы, добавлять в файл новые записи, удалять записи из файла, изменять содержимое записи (значение поля *info)*, осуществлять прямой доступ к записи по ее идентификатору, а также создавать итераторы, выдающие без повторений указатели на все записи файла.

4. Менеджер дисков

Менеджер дисков реализует интерфейс ввода-вывода для организации обменов с дисковой подсистемой. Менеджер дисков позволяет абстрагироваться от аппаратных особенностей дисковой подсистемы кластера и рассматривать ее как виртуальное устройство, состоящее из некоторого числа логических дисков со страничной организацией.

Количество дисков берется равным количеству процессорных модулей кластера. За каждым процессорным модулем закрепляется свой диск. Процесс не может обращаться к "чужим" дискам. Данный подход моделирует архитектуру без совместного использования ресурсов.

Размер страницы диска является параметром генерации системы Омега и берется равным 4 КБайтам.

4.1. Эмулятор дисковой подсистемы

Аппаратная платформа, на которой выполнялась реализация прототипа системы Омега, представляет собой мультипроцессор МВС-100 в конфигурации из четырех процессорных

модулей без дисковой подсистемы. Поэтому при разработке прототипа СУБД Омега возникла необходимость создания *эмулятора дисковой подсистемы*.

Для создания виртуальных дисков могут использоваться следующие два подхода:

- использование оперативной памяти одного из процессорных модулей для создания электронных дисков;
- использование дисков Host-машины.

Эмулятор дисковой подсистемы (ЭДП) на базе электронного диска будет значительно превосходить ЭДП на базе дисков Host-машины по эффективности доступа. Это может ощутимым образом сказаться на эффективности процесса отладки и профилирования системы в целом. Отрицательной стороной использования электронных дисков является их энергозависимая природа. При сбоях и рестартах системы информация на электронных дисках будет теряться, что, в свою очередь, осложнит процесс отладки.

Исходя из этого, был выбран промежуточный вариант, предполагающий совместное использование и электронных дисков процессорного модуля и магнитных дисков Host-машины. При инициализации ЭДП происходит автоматическое считывание базы данных с диска Host-машины и загрузка ее в электронные диски некоторого выделенного процессорного модуля. При нормальном завершении работы системы происходит обратный сброс базы данных с электронного диска на диск Host-машины. В процессе работы ЭДП могут выполняться промежуточные сбросы на диск Host-машины.

Для моделирования дисковой подсистемы на имеющейся конфигурации МВС-100 был выделен процессорный модуль, являющийся корневым. Данный модуль непосредственно связан по линку с Host-машиной. Процессорный модуль дисковой подсистемы не используется как вычислительный модуль. Таким образом, фактически моделировался Омега-кластер с тремя процессорными модулями и одной дисковой подсистемой.

Эмулятор дисковой подсистемы инкапсулирует в себе аппаратные особенности дисковой подсистемы. Данный подход позволяет при переходе на аппаратную конфигурацию с реальной дисковой подсистемой оставить практически неизменным интерфейс менеджера дисков, что, в свою очередь, позволяет ограничиться небольшими переделками при переносе системы в целом.

4.2. Принципы проектирования менеджера дисков

Менеджер дисков реализует архитектуру виртуальных каналов. Это означает, что по линку, соединяющему процессорный модуль с дисковой подсистемой, параллельно может производиться любое количество операций чтения-записи.

Основные операции, входящие в интерфейс менеджера дисков, изображены на Рис. 6.

```
/* Записать страницу на диск в асинхронном режиме */
    int /* No. канала*/ ds_write(int /*номер страницы */, void * /* указатель на буфер */);
/* Считать страницу с диска в асинхронном режиме */
    int /* No. канала*/ ds_read(int /*номер страницы */, void * /* указатель на буфер */);
/* Проверить завершение write/read операции */
    int /* 1-да, 0-нет */ ds_done(int /* No. канала*/);
/* Сброс содержимого виртуального диска в файл на host-машине */
    void ds_dump(char * /* имя файла */);
/* Загрузка содержимого виртуального диска из файла на host-машине */
    void ds_reset(char * /* имя файла */);
```

Рис. 6. Основные функции интерфейса менеджера дисков

При проектировании менеджера дисков была использована технология клиент-сервер. Клиентская часть менеджера дисков, запускается на каждом рабочем узле. Серверная часть запускается на модуле дисковой подсистемы.

Сервер обслуживает запросы клиентов на выполнение операций с дисковой подсистемой. Клиенты и сервер обмениваются сообщениями, которые состоят из двух частей, - заголовка и информационной части *info*. Заголовок сообщения имеет следующие поля: идентификатор операции, тип операции, номер страницы, номер узла клиента. Информационная часть не имеет структуры; это байтовый массив, длина которого совпадает с размером страницы. Передача сообщения выполняется в два этапа, - сначала передается заголовок, а затем - часть *info*.

4.3. Принципы реализации менеджера дисков

При выполнении клиентом операций **ds_read** и **ds_write** дескрипторы этих операций добавляются в таблицу операций. Таблица операций организована как очередь и имеет следующие поля: идентификатор операции, номер страницы, номер узла клиента, указатель на буфер и состояние операции. Таблица операций обрабатывается системной нитью клиента. Фактор-функция системной нити [15] пишется таким образом, чтобы нить активизировалась только при завершении асинхронных операций чтения/записи по линкам, все остальное время нить находится в неактивном состоянии.

Сервер оформляется как самостоятельный процесс, который обрабатывает аналогичную таблицу операций на стороне дисковой подсистемы. Этот процесс циклически выполняет следующую последовательность действий.

1) Ожидание заголовка сообщения от клиентов;
2) Обработка заголовка (создание нового элемента таблицы операций);
3) Просмотр таблицы операций и выполнение необходимых действий (чтение страницы с диска, сброс диска и т.п.);
4) Инициация операции чтение заголовка на освободившихся линках.

Клиент и сервер для обмена данными используют следующий *специальный протокол обмена сообщениями*.

Сервер находится в постоянном ожидании поступления заголовка сообщения от клиента (все сообщения имеют заголовки одинаковой структуры). При получении заголовка сервер действует в зависимости от типа операции, указанной в заголовке сообщения.

В случае операции "запись страницы", немедленно запускается чтение информационной части сообщения от клиента, и в таблицу операций сервера помещается запись о том, что операция выполняется.

Если указана операция "чтение страницы", то в таблицу операций сервера просто помещается запись о том, что соответствующая операция выполняется.

Если необходимо выполнить операцию восстановления (сброса) диска, то запись в таблицу операций сервера помещается только в том случае, если на данный момент в ней отсутствуют записи об операциях чтения страницы (записи страницы) от данного клиента.

После этого сервер обрабатывает текущий элемент очереди из таблицы операций, в зависимости от типа операции, а затем удаляет эту запись из очереди.

Операции "сброс диска" и "восстановление диска" выполняются немедленно, после чего клиенту высылается заголовок с кодом "операция завершена".

В случае операции "запись страницы", заголовок с кодом "операция завершена" посылается клиенту только после завершения выполнения соответствующей операции.

В случае операции "чтение страницы", сервер прочитывает страницу с диска, затем посылает заголовок с кодом "приготовиться к чтению информационной части", после чего происходит посылка самой информационной части.

Клиент находится в постоянном ожидании поступления ответа от сервера. При получении ответа клиент действует в зависимости от типа инициированной операции.

В случае операции "запись страницы" линк блокируется по записи; немедленно запускается передача информационной части на сервер; соответствующая запись добавляется в таблицу операций.

В случае операции "чтение страницы" линк блокируется по чтению; запускается чтение информационной части с сервера; соответствующая запись добавляется в таблицу операций.

Для операций "сброс диска" и "восстановление диска" добавление соответствующих записей в таблицу операций происходит только в том случае, если на данный момент в ней отсутствуют записи об операциях чтения и записи страницы от данного клиента (в противном случае происходит аварийный выход из функций **ds_dump** или **ds_reset**).

5. Менеджер страниц

Менеджер страниц обеспечивает представление хранимой базы данных в виде совокупности наборов страниц. Набор страниц представляет собой связный список страниц. Страница состоит из заголовка и одного информационного поля *info*.

Менеджер страниц должен удовлетворять следующим основным требованиям.

- Должна поддерживаться таблица размещения наборов страниц - SAT (page Set Allocation Table). SAT размещается в непрерывной области, начиная с нулевой страницы диска.
- Должен поддерживаться особый набор страниц - ССП (Список Свободного Пространства).
- Должна поддерживаться возможность добавления в набор непрерывного блока страниц (например, по 64 страницы).
- Должна поддерживаться буферизация страниц на основе единого буферного пространства. Должна поддерживаться таблица подкачки (индекс буферного пула), содержащая номера всех страниц, находящихся в данный момент в буфере. Алгоритмы вытеснения и подкачки страниц должны эффективно работать для полностью заполненного буфера (это штатная ситуация в случае систем баз данных, - страница вытесняется из буфера только тогда, когда это действительно необходимо для загрузки некоторой другой страницы).
- Доступ к содержимому страницы может быть осуществлен только через ее образ в оперативной памяти.
- Должен поддерживаться УИС (Уникальный Идентификатор Страницы). УИС должен быть уникальным в пределах диска. УИС должен обеспечивать прямой доступ к странице. В простейшем случае в качестве УИС может использоваться номер страницы на диске.

- Должна поддерживаться выборка страниц с упреждением.

5.1. Принципы проектирования менеджера страниц

Доступ к содержимому страницы обеспечивается специальным оператором *выборки страницы* **pg_fetch**, который выдает указатель на образ страницы в буфере.

Если в процессе работы с содержимым страницы прямо или косвенно вызывалась операция диспетчеризации **th_schedule** менеджера нитей [15], то перед продолжением работы с образом страницы рекомендуется вторично выполнить операцию **pg_fetch**. Это связано с тем, что во время передачи управления другой нити могло произойти откребление и вытеснение из буфера образа данной страницы в результате выполнения "параллельного" **pg_fetch**.

Оператор **pg_fetch** является синхронным в том смысле, что он возвращает управление только после того, как страница загружена в буфер (с возможным предшествующим вытеснение какой-либо другой страницы). Если после загрузки страницы в буфер заранее известно, какую страницу придется читать следующей (а в случае баз данных это, как правило, известно), можно воспользоваться оператором *выборки с упреждением* **pg_prefetch.** Данный оператор осуществляет загрузку страницы в буфер в асинхронном режиме. Перед доступом к странице, после оператора **pg_prefetch**, должен быть выполнен оператор **pg_fetch** для того, чтобы гарантировать завершение загрузки страницы в буфер.

Открытие набора страниц осуществляется с помощью операции **pg_open.** Для каждого открытого набора вводится атрибут *текущая страница.* При открытии набора данный атрибут устанавливается в значение **NIL**, что соответствует пустой ссылке.

Оператор выборки страницы **pg_fetch** устанавливает указатель текущей страницы на указанную страницу. Данная страница набора *закрепляется* в буфере, то есть она не может быть вытеснена из буфера ни при каких условиях. При выполнении следующего оператора **pg_fetch** над данным набором указатель текущей страницы переустанавливается на новую страницу. При этом предыдущая текущая страница набора *откребляется* от буфера, то есть она при необходимости может быть вытеснена из буфера.

С каждой страницей, находящейся в буфере, связывается некоторое значение, называемое *рейтингом страницы*. Различаются статический и динамический рейтинги. *Статический рейтинг* задается пользователем при загрузке страницы в буфер. *Динамический рейтинг* является функцией от статического рейтинга и вычисляется менеджером страниц. Если возникает необходимость освободить в буфере место для новой страницы, то из буфера вытесняется страница с наименьшим динамическим рейтингом. Значение динамического рейтинга страницы может изменяться со временем.

5.2. Интерфейс менеджера страниц

Основу интерфейса менеджера страниц составляют функции, изображенные на Рис. 7.

```
/* Создать пустой набор страниц */
    int /* >=0 - OK, иначе - ошибка */ pg_createSet(int /* идентификатор набора */);
/* Удалить набор страниц */
    int /* >=0 - OK, иначе - ошибка */ pg_dropSet(int /* идентификатор набора */);
/* Открыть набор страниц */
    int /* >=0 - OK, иначе - ошибка */ pg_open(int /* идентификатор набора */);
/* Закрыть набор страниц */
    int /* >=0 - OK, иначе - ошибка */ pg_close(int /* идентификатор набора */);
/* Добавить n страниц в конец набора */
    int /* >=0 - идентификатор первой добавленной страницы, иначе - ошибка */
    pg_append(int /* идентификатор набора */, int /* количество страниц */);
/* Удалить страницу из набора */
    int /* >=0 - OK, иначе - ошибка */
    pg_delete(int /* идентификатор набора */, int /* идентификатор страницы */);
/* Выборка страницы */
    void * /* указатель на образ страницы в буфере */
    pg_fetch(int /* идентификатор набора */, int /* идентификатор страницы, или NIL */, int /* статический рейтинг */);
/* Выборка страницы с упреждением */
    int /* >=0 - OK, иначе - ошибка */
    pg_prefetch(int /* идентификатор набора */, int /* идентификатор страницы */, int /* статический рейтинг */);
/* Установка атрибута образа страницы modified */
    int /* >=0 - OK, иначе - ошибка */
    pg_setModified(int /* идентификатор страницы */, char /* новое значение атрибута */);
```

Рис. 7. Основные функции интерфейса менеджера страниц

Функция **pg_fetch** обеспечивает *прямой* доступ к содержимому указанной страницы. Данная функция проверяет наличие указанной страницы в системном буфере и выдает указатель на образ страницы в буфере. Если указанная страница в буфере отсутствует, то производится ее подкачка. Если при этом в буфере нет места, то происходит вытеснение из буфера на диск образа неиспользуемой страницы, имеющей наименьший динамический рейтинг. Если буфер полностью заполнен и все образы станиц являются в данный момент используемыми, то операция переходит в состояние ожидания.

После выполнения операции **pg_fetch** указанная страница становится *текущей страницей* указанного набора и помечается как *используемая*. Предыдущая текущая страница при этом помечается как *неиспользуемая (открепляется)*.

Выполнение операции **pg_fetch** со значением параметра "идентификатор страницы", равным **NIL**, приводит к откреплению предыдущей *текущей страницы*. В качестве результата в этом случае выдается NULL.

Функция **pg_prefetch** обеспечивает опережающую загрузку страницы в буфер в асинхронном режиме. Если в буфере отсутствует свободное место, происходит вытеснение на диск некоторой неиспользуемой страницы с наименьшим динамическим рейтингом.

5.3. Принципы реализации менеджера страниц

Иерархия модулей и объектов, представляющих реализацию менеджера страниц, изображена на Рис. 8.

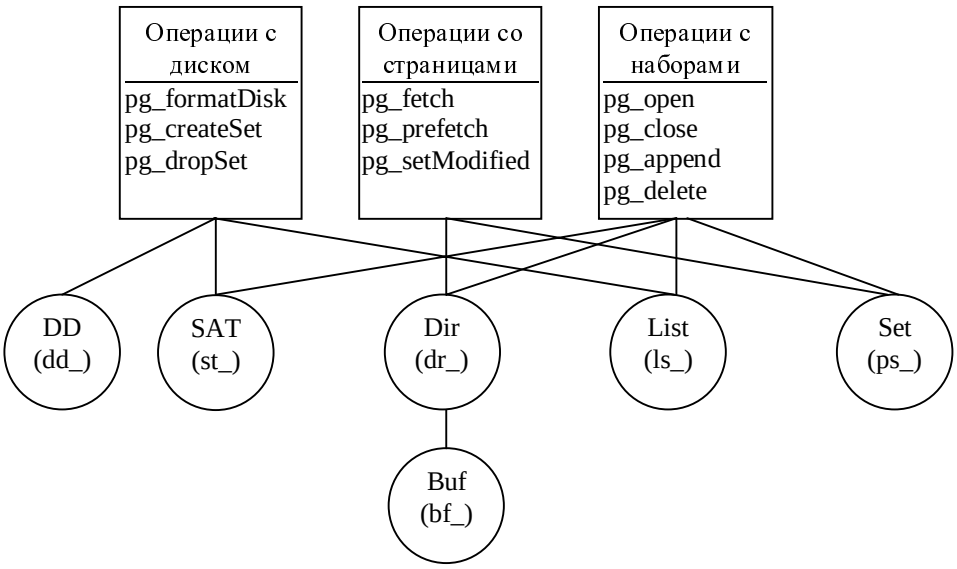

Рис. 8. Иерархия модулей и объектов менеджера страниц

Реализация основных операций менеджера страниц строится на базе следующих объектов.

- **Buf** (Buffer) - буфер для буферизации страниц;
- **Dir** (Buffer Directory)- избыточный индекс буферизованных страниц (индекс для **Buf**);
- **List** (Space List) - список свободного пространства (ССП);
- **DD** (Disk Directory) - заголовок диска;
- **SAT** (Set Allocation Table) - таблица размещения наборов;
- **Set** (Open Page Set Descriptor Table) - таблица дескрипторов открытых наборов.

5.3.1. Объект Buf (менеджер буферного пула)

Менеджер буферного пула обеспечивает резервирование и освобождение непрерывных участков памяти в буферном пуле на основе списка свободного пространства (ССП). Минимальной единицей выделения памяти является блок размером 4КБайта.

Интерфейс менеджера буферного пула включает в себя две основные операции: **bf_alloc** - выделить блок из ССП, и **bf_free** - вернуть блок в ССП.

Конкретным представлением буферного пула является байтовый массив, место под который выделяется при инициализации подсистемы путем однократного динамического выделения памяти. Размер данного массива является параметром системы и хранится в файле настроек. Никаких дополнительных динамических выделений памяти под буферное пространство в дальнейшем не происходит.

Менеджер буферного пула поддерживает внутренний список свободных блоков буферного пула, хранящийся в памяти. Данный список представляет собой байтовый массив: **char _xbf[N]**, где **N** - общее количество блоков в буферном пуле. Элемент **_xbf[i]** принимает значение 1 тогда и только тогда, когда блок с номером i является свободным. Поиск свободного блока - линейный. Для ускорения поиска используется следующая простая оптимизация. Поддерживается указатель **top** на вершину списка (<первый с конца занятый элемент>+1). Выделение свободного блока всегда осуществляется по

указателю **top**. Если свободного места в хвосте буферного пула нет, то производится сквозной поиск в массиве **_xbf[ ]** первого элемента, равного 1. Указатель **top** пересчитывается при каждом выполнении операции освобождения блока.

5.3.2. Объект Dir

Объект **Dir** реализует избыточный индекс буферного пула. Избыточность понимается в том смысле, что индекс **Dir** содержит количество позиций, превышающее количество страниц, которое может быть размещено в буферном пуле. Данная избыточность по существу используется в реализации метода **pg_prefetch** (см. ниже). Кроме того, избыточность индекса **Dir** может быть использована при определении стратегии вытеснения, путем анализа "истории" загрузки страниц в буфер (в частности, при достаточных размерах **Dir** могут быть обнаружены и оптимизированы *циклы подкачки*).

Индекс **Dir** используется также для организации асинхронных операций загрузки, сохранения и замещения образа страницы в буферном пуле. Это достигается тем, что с каждым элементом **Dir** может быть связан код одной из перечисленных операций. Вводится специальная системная нить **_dr_sys** с приоритетом *(**TH_MINNICE** + 1)*. Данная системная нить периодически просматривает свою внутреннюю таблицу **_dr_sysT**. Таблица **_dr_sysT** содержит отображение операций из **Dir** в соответствующие асинхронные операции менеджера диска. Техника реализации системной нити **_dr_sys** близка к технике реализации системной нити кондуктора [16].

Основные методы объекта **Dir** изображены на Рис. 9.

```
/* Вычисление индекса элемента */
    int /* >=0 - индекс элемента, иначе - ошибка */ dr_iDir(int /* идентификатор набора */, int /* идентификатор
    страницы */);
/* Вставить новый элемент */
    int /* индекс нового элемента */ dr_insDir(int /* идентификатор набора */, int /* идентификатор страницы */);
/* Найти "жертву" */
    int /* >=0 - индекс жертвы, иначе - ошибка */ dr_findVictim(void);
```

Рис. 9. Основные методы объекта **Dir**

Метод **dr_insDir** добавляет в **Dir** новый элемент, с указанными значениями атрибутов. Если в **Dir** нет свободных позиций, то производится поиск неиспользуемого элемента с минимальным динамическим рейтингом. Новый элемент вставляется на место найденной "жертвы".

Метод **dr_findVictim** производит поиск в **Dir** элемента с наименьшим динамическим рейтингом среди элементов, содержащих непустую ссылку в поле **pBuf**, имеющих статус **DR_NOTUSING** или **DR_PREFETCHING**, и такт операции, равный **NIL**. Если такие элементы в **Dir** отсутствуют, то выдается код ошибки **PG_ENOMEM**.

Конкретное представление объекта **Dir** включает в себя две таблицы: собственно таблицу **Dir**, и хеш-таблицу **_dr_HT**.

Таблица **Dir** реализуется в виде статического массива, элементы которого имеют тип **dr_elem_t**. Размер массива **Dir** должен быть равен kM, $k>1$ (по умолчанию $k=8$). Здесь M - длина буферного пула в страницах. Таким образом, **Dir** содержит избыточное количество элементов, в том смысле, что не все элементы связаны с образами страниц в буфере, - часть элементов не используется.

Структура элемента таблицы **Dir** показана на Рис. 10.

```
typedef struct{
    int pageSet;      /* идентификатор набора */
    int page;         /* идентификатор страницы */
    int reit;         /* рейтинг элемента */
    char updated;/* признак модификации образа */
    int status;       /* статус (состояние) элемента:
       DR_NOTUSING - не используется; образ страницы может отсутствовать в буфере.
       DR_USING - используется; образ страницы обязательно присутствует в буфере.
       DR_PREFETCHING - выборка с упреждением.
       DR_FETCHING - выборка. */
    pg_page_t * pBuf;     /* указатель на образ страницы в буфере */
    int opcode;       /* КОП (код операции):
                         OP_LOAD - загрузить.
                              Такт 1: считывает страницу page в буфер pBuf.
                         OP_SAVE - сохранить.
                              Такт 1: записывает содержимое буфера pBuf на страницу page.
                         OP_REPLACE - заместить.
                              Такт 1: записывает содержимое буфера pBuf на страницу arg;
                              Такт 2: считывает страницу page в буфер pBuf. */
    int arg;          /* аргумент операции */
    int time;         /* такт операции:
       == 0 - операция не выполнялась;
       > 0 - выполняется указанный такт;
       == NIL (-1) - выполнение операции завершено. */
} dr_elem_t;
```

Рис. 10. Структура элемента таблицы **Dir**

Прямой доступ к элементам **Dir** обеспечивается использованием хеш-таблицы. Для разрешения коллизий используется метод цепочек. Память для элементов цепочки выделяется динамически. При удалении соответствующего элемента из **Dir** элемент цепочки лучше не удаляется, вместо этого в поле **iDir** помещается значение **NIL**, означающее, что данный элемент цепочки свободен. При этом освободившийся элемент перемещается в направлении конца цепочки так, чтобы он стал первым свободным элементом в цепочке.

5.3.3. Объект List (менеджер ССП)

Менеджер списка свободного пространства (ССП) обеспечивает выделение непрерывных блоков страниц и утилизацию освобожденных страниц на виртуальном диске. Менеджер ССП реализуется в виде объекта **List.** Основные методы объекта **List** изображены на Рис. 11.

```
/* Выделить непрерывный блок */
    int /* >=0 - номер первой страницы блока, иначе - нет места на диске */
    ls_alloc(int n /* длина блока в страницах */ );
/* Вернуть страницу в ССП*/
    void ls_free(int /* номер страницы */);
/* Вернуть все страницы диска в ССП */
    void ls_clear(void);
```

Рис. 11. Основные методы объекта **List**

Конкретным представлением объекта **List** является внутренний индекс ССП, хранящийся в заголовке диска. Индекс ССП представляет собой байтовый массив **char _xsl[N]**, где **N** - общее количество страниц на диске. Значение элемента **_xsl[i]** равно **1** тогда и только тогда, когда страница с номером **i** принадлежит ССП.

Поиск непрерывного блока страниц в ССП - линейный. Для ускорения поиска используется следующая простая оптимизация. Поддерживается указатель **top** (<первый, занятый с конца>+1). Выделение блока страниц всегда осуществляется по указателю **top**. Если свободного места в хвосте ССП нет, то в ССП производится сквозной поиск блока нужной длины. При возвращении страницы в ССП всегда пересчитывается указатель **top**.

5.3.4. Объект DD

Объект **DD** предоставляет методы для работы с заголовком диска. Заголовок диска занимает нулевую страницу диска. В заголовке диска хранится список свободного пространства, таблица размещения наборов и другая служебная информация.

Заголовок диска загружается в оперативную память с помощью метода **dd_loadDirectory**. Сохранение изменений образа заголовка диска осуществляется с помощью метода **dd_saveDirectory.**

```
int pg_prefetch(int pageSet, int page, int reit) {
int iDir, victim;
pg_page_t *pBuf; /* указатель на образ страницы в буфере */

if (page < 0) return PG_EWRONGPARAM;
if ((iDir=dr_iDir(pageSet, page)) == NIL)
        /* дескриптор страницы отсутствует в Dir */
        iDir = dr_insDir(pageSet, page);
Dir[iDir].reit = reit;
if (Dir[iDir].status != DR_ NOTUSING)
        return iDir;
Dir[iDir].status = DR_PREFETCHING;
if (Dir[iDir].pBuf != NULL) /* …образ страницы присутствует в буфере. */
        return iDir;
/* Образ страницы отсутствует в буфере… */
Dir[iDir].updated = FALSE;
pBuf = (pg_page_t) bf_alloc();

if (pBuf != NULL){ /* …удалось найти свободный блок в Buf. */
        Dir[iDir].pBuf = pBuf;
        Dir[iDir].opcode = OP_LOAD; /* КОП: загрузить */
        Dir[iDir].time = 0; /* такт операции */
        _dr_insSysT(iDir, DS_READ, Dir[iDir].page, Dir[iDir].pBuf);
                                        /* - вставить запись в _dr_sysT */
} else { /* Не удалось найти свободный блок в Buf… */
        victim = dr_findVictim();
        if (victim < 0) return PG_ENOMEM;
        Dir[iDir].pBuf = pBuf = Dir[victim].pBuf;
        Dir[victim].pBuf = NULL;
        Dir[victim].status = DR_NOTUSING;
        if (Dir[victim].updated == TRUE){ /* Жертва обновлялась, нужно сохранить… */
                Dir[iDir].opcode = OP_REPLACE; /* КОП: заменить */
                Dir[iDir].arg = Dir[victim].page; /* что заменить */
                Dir[iDir].time = 0; /* такт операции */
                _dr_insSysT(iDir, DS_WRITE, Dir[iDir].arg, Dir[iDir].pBuf);
                                /* - вставить запись в _dr_sysT */
        } else { /* Жертва не обновлялась, можно не сохранять… */
                Dir[iDir].opcode = OP_LOAD; /* КОП: загрузить */
                Dir[iDir].time = 0; /* такт операции */
                _dr_insSysT(iDir, DS_READ, Dir[iDir].page, Dir[iDir].pBuf);
        }
};
return iDir;};
```

Рис. 12. Схема реализации алгоритма выборки с упреждением

5.3.5. Объект SAT

Объект **SAT** реализует таблицу размещения наборов. Данная таблица хранится в заголовке диска и считывается в оперативную память каждый раз при инициализации менеджера страниц. Элементы таблицы **SAT** содержат следующую информацию: идентификатор набора, идентификатор первой страницы набора, идентификатор последней страницы набора и другую служебную информацию.

Конкретным представлением объекта **SAT** является статический массив.

5.3.6. Объект Set

Объект **Set** реализует таблицу дескрипторов открытых наборов. Таблица **Set** представляется в виде статического массива записей, организованного в виде хеш-таблицы. Для разрешения коллизий используется метод открытой адресации.

5.3.7. Алгоритмы простой выборки и выборки с упреждением

Схема реализации алгоритма выборки с упреждением (операция **pg_prefetch)** изображена на Рис. 12. Данная реализация обеспечивает асинхронный характер выполнения операции выборки с упреждением.

Схема реализации алгоритма простой выборки (операция **pg_fetch)** изображена на Рис. 13.

```
ivoid * pg_fetch(int pageSet, int page, int reit) {
int iDir,          /* номер дескриптора образа в Dir */
        prevPage,/* номер страницы предыдущего fetch для данного набора */
        iSet;    /* номер дескриптора набора в Set */
iSet = ps_iSet(pageSet);
prevPage = Set[iSet].curPage;
iDir = dr_iDir(pageSet, prevPage);
if (prevPage != NIL){
        while (Dir[iDir].time != NIL)
                th_schedule();
        if (prevPage == page)
                return Dir[iDir].pBuf;
        else
                Dir[iDir].status = DR_NOTUSING;
};
Set[iSet].curPage = page;
if (page == NIL) return NULL;
iDir = pg_prefetch(pageSet, page, reit);
if (iDir == NIL) return NULL;
Dir[iDir].status = DR_FETCHING;
while (Dir[iDir].time != NIL)
        th_schedule();
Dir[iDir].status = DR_USING;
return Dir[iDir].pBuf;
};
```

Рис. 13. Схема реализации алгоритма простой выборки

Операция выборки страницы выполняется в синхронном режиме. Если указанная страница уже находится в буфере, например, в результате выполнения операции выборки с упреждением, то операция выборки страницы немедленно завершается. Если образ указанной страницы в буфере отсутствует, то операция выборки завершится только после считывания данной страницы с диска.

6. Менеджер файлов

Менеджер файлов обеспечивает представление БД в виде совокупности файлов. Файл - это последовательный набор записей одинаковой длины. Запись состоит из заголовка и одного информационного поля *info*. Заголовок записи размещается в конце записи, то есть после поля *info*. В этом случае указатель на запись может быть преобразован в указатель на *info*, что позволяет скрыть от пользователя структуру заголовка.

Реализация хранимых записей осуществляется с использованием техники, описанной в [17]. Каждый файл размещается в отдельном наборе страниц. Соответствие между файлами и наборами страниц хранится в - FAT (File Allocation Table). FAT размещается в непрерывной области диска, начиная с нулевой страницы.

Поддерживается УИД (Уникальный ИДентификатор записи). УИД является уникальным в пределах диска. Значение УИД остается неизменным на протяжении всего времени жизни записи. УИД обеспечивает прямой доступ к записи (обращение к любой записи по ее УИД требует не более одной операции чтения/записи страницы). В качестве УИД используется пара (<УИС>, <номер байта от конца страницы, содержащий адрес первого байта записи на странице>).

Основные функции менеджера файлов изображены на Рис. 14.

```
/* Создать пустой файл */
    int /* >=0 - OK, иначе - ошибка */ fl_createFile(int /* идентификатор файла */, int /* длина info */);
/* Удалить файл */
    int /* >=0 - OK, иначе - ошибка */ fl_dropFile(int /* идентификатор файла */);
/* Открыть файл */
    int /* >=0 - OK, иначе - ошибка */
    fl_open(int /* идентификатор файла */,
    char /* режим 0 (FL_READ) - только чтение, 1 (FL_WRITE) - запись и чтение */);
/* Закрыть файл */
    int /* >=0 - OK, иначе - ошибка */ fl_close(int /* идентификатор файла */);
/* Добавить запись в конец файла */
    int /* >=0 - УИД первой добавленной записи, иначе - ошибка */ fl_append(int /* идентификатор файла */);
/* Пометить запись на удаление */
    int /* >=0 - OK, иначе - ошибка */ fl_delete(int /* идентификатор файла */, int /* УИД записи */);
/* Удалить все записи с пометкой на удаление */
    int /* >=0 - OK, иначе - ошибка */ fl_pack(int /* идентификатор файла */, int /* УИД записи */);
/* Выборка записи */
    void* /* <>NULL - указатель на info записи, иначе - ошибка */
    fl_fetch(int /* идентификатор файла */, int /* УИД записи */);
```

Рис. 14. Основные функции менеджера файлов

7. Заключение

В данной работе были изложены принципы разработки и программная структура системы управления файлами (СУФ) ядра параллельной СУБД Омега для отечественного суперкомпьютера МВС-100. Были сформулированы требования к СУФ и дано описание ее общей структуры. Приведены описания всех основных программных компонент СУФ.

Предложен механизм вытеснения страниц, основанный на динамических и статических рейтингах страниц. Данный механизм позволяет реализовывать практически любы стратегии вытеснения страниц.

Даны описания алгоритмов простой выборки страниц и выборки с упреждением. Предложен эффективный протокол для взаимодействия с дисковой подсистемой. Описана архитектура эмулятора дисковой подсистемы.

Описанная СУФ реализована на МВС-100 в системе программирования Си. Предложенная архитектура системы управления файлами допускает перенос полученной реализации на платформу МВС-1000 при минимальных доработках исходных текстов.

Описанная реализация СУФ, прежде всего, ориентирована на использование в системах баз данных, однако она может быть использована и в других задачах, требующих интенсивных обменов с дисками.

Литература

1. Девитт Д., Грэй Д. Параллельные системы баз данных: будущее высоко эффективных систем баз данных // СУБД. 1995. №2. С. 8-31.
2. Baru C. K., et al. DB2 Parallel Edition // IBM System Journal. 1995. Vol. 34. No. 2. P. 292-322.
3. Page J. A Study of a Parallel Database Machine and its Performance the NCR/Teradata DBC/1012 // Advanced Database Systems, 10th British National Conference on Databases, BNCOD 10, Aberdeen, Scotland, July 6-8, 1992, Proceedings. Lecture Notes in Computer Science, Vol. 618, Springer. 1992. P. 115-137.
4. Tandem Database Group NonStop SQL: A Distributed, High-Performance, High-Availability Implementation of SQL // High Performance Transaction Systems, 2nd Int. Workshop, Pacific Grove, California, USA, September 28-30, 1987, Proceedings. Lecture Notes in Computer Science, Vol. 359, Springer. 1989. P. 60-104.
5. Valduries P. Parallel Database Systems: Open Problems and New Issues // Distributed and Parallel Databases. April 1993. Vol. 1. No. 2. P. 137-165.
6. Stonebraker M. The case for shared nothing // Database Engineering Bulletin. March 1986. Vol. 9. No. 1. P. 4-9.
7. Valduries P. Parallel Database Systems: the case for shared-something // Proc. of 9th International Conference on Data Engineering. Vienna, Austria. April 19-20, 1993. P. 460-465.
8. Bhide A., Stonebraker M. A Performance Comparison of Two Architectures for Fast Transaction Processing // Proceedings of the 4th International Conference on Data Engineering, Los Angeles, CA. February 1988. P. 536-545.
9. Bouganim L., Florescu D., Valduriez P. Dynamic Load Balancing in Hierarchical Parallel Database Systems // VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, Mumbai (Bombay), India. September 1996. P. 436-447.
10. Xu Y., Dandamudi S.P. Performance Evaluation of a Two-Level Hierarchical Parallel Database System // Proceedings of the Int. Conf. Computers and Their Applications, Tempe, Arizona. March 1997. P. 242-247.
11. Sokolinsky L., Axenov O., Gutova S. Omega: The Highly Parallel Database System Project // Proceedings of the First East-European Symposium on Advances in Database and Information Systems (ADBIS'97), St.-Petersburg. September 2-5, 1997. Vol. 2. P. 88-90.
12. Zabrodin A.V., Levin V.K., Korneev V.V. The Massively Parallel Computer System MBC-100 // Proceedings of PaCT-95. Lecture Notes in Computer Science. Volume 964. 1995. P. 342-356.
13. Гольштейн М.Л. Мультипроцессорная вычислительная система на базе транспьютерной идеологии // Алгоритмы и программные средства параллельных вычислений: [Сб. науч. Тр.]. Екатеринбург: УрО РАН. 1995. С. 61-68.
14. Flynn M.J., Rudd K.W. Parallel architectures // ACM Computing Surveys. March 1996. Vol. 28. No. 1. P. 67-70.
15. Соколинский Л.Б. Эффективная организация легковесных процессов в параллельной СУБД Омега для МВС-100 // Фундаментальные и прикладные аспекты разработки больших распределенных программных комплексов: Тез. докл. Всероссийск. науч. конф. (21-26 сентября 1998 г., г. Новороссийск). -М.: Изд-во МГУ. 1998. С. 132-138.

16. Sokolinsky L.B. Interprocessor Communication Support in the Omega Parallel Database System // CSIT'99, Proceedings of the 1st International Workshop on Computer Science and Information Technologies, January 18-22, 1999, Moscow, Russia. MEPhI Publishing. 1999. Vol. 2. (http://msu.jurinfor.ru/CSIT99/CSIT99.html)
17. Date C.J. An Introduction to Database Systems (6th edition). Reading, Mass.: Addison-Wesley, 1995. 839 p.

Abstract. The paper describes a structure and implementation principles of a File Management System (FMS) for the parallel Omega DBMS. This DBMS is designed for the MVS-100 massively parallel computer system. The paper specifies requirements for the FMS and describes its structure and components. The paper proposes a page replacement strategy based on the static and dynamic page rating. The paper gives a specification of fetch and prefetch algorithms. The paper suggests some effective protocol for interaction with the Disk Subsystem. The paper describes Disk Subsystem Emulator's architecture. FMS described in this paper is implemented under the MVS-100 and can be ported to the MVS-1000 after minimal rework.