Билет № 11

1) Погружение процессов в вычислительной системе. Критерии оценки алгоритмов погружения.

Погружение процессов в вычислительной системе. Критерии оценки алгоритмов погружения.

Критерии планирования и требования к алгоритмам

Для каждого уровня планирования процессов можно предложить много различных алгоритмов. Выбор конкретного алгоритма определяется классом задач, решаемых вычислительной системой, и целями, которых мы хотим достичь, используя планирование. К числу таких целей можно отнести следующие:

- Справедливость – гарантировать каждому заданию или процессу определенную часть времени использования процессора в компьютерной системе, стараясь не допустить возникновения ситуации, когда процесс одного пользователя постоянно занимает процессор, в то время как процесс другого пользователя фактически не начинал выполняться.

- Эффективность – постараться занять процессор на все 100% рабочего времени, не позволяя ему простаивать в ожидании процессов, готовых к исполнению. В реальных вычислительных системах загрузка процессора колеблется от 40 до 90%.

- Сокращение полного времени выполнения (turnaround time) – обеспечить минимальное время между стартом процесса или постановкой задания в очередь для загрузки и его завершением.

- Сокращение времени ожидания (waiting time) – сократить время, которое проводят процессы в состоянии готовность и задания в очереди для загрузки.

- Сокращение времени отклика (response time) – минимизировать время, которое требуется процессу в интерактивных системах для ответа на запрос пользователя.

Независимо от поставленных целей планирования желательно также, чтобы алгоритмы обладали следующими свойствами.

- Были предсказуемыми. Одно и то же задание должно выполняться приблизительно за одно и то же время. Применение алгоритма планирования не должно приводить, к примеру, к извлечению корня квадратного из 4 за сотые доли секунды при одном запуске и за несколько суток – при втором запуске.

- Были связаны с минимальными накладными расходами. Если на каждые 100 миллисекунд, выделенные процессу для использования процессора, будет приходиться 200 миллисекунд на определение того, какой именно процесс получит процессор в свое распоряжение, и на переключение контекста, то такой алгоритм, очевидно, применять не стоит.

- Равномерно загружали ресурсы вычислительной системы, отдавая предпочтение тем процессам, которые будут занимать малоиспользуемые ресурсы.

- Обладали масштабируемостью, т. е. не сразу теряли работоспособность при увеличении нагрузки. Например, рост количества процессов в системе в два раза не должен приводить к увеличению полного времени выполнения процессов на порядок.

2) ОС в оперативной памяти.

3) Особенности UFS.

Файловая система UNIX V7

Даже в ранних версиях системы UNIX применялась довольно сложная многопользовательская файловая система, так как в основе этой системы лежала операционная система MULTICS. Ниже будет рассмотрена файловая система V7, разработанная для компьютера PDP-11, сделавшего систему UNIX знаменитой. Современные версии будут обсуждаться в главе 10.

Файловая система представляет собой дерево, начинающееся в корневом каталоге, с добавлением связей, формирующих направленный ациклический граф. Имена файлов могут содержать до 14 символов, включающих в себя любые символы ASCII, кроме косой черты (использовавшейся в качестве разделителя компонентов пути) и символа NUL (использовавшегося для дополнения имен короче 14 символов). Символ NUL обозначается байтом 0.

Каталог UNIX содержит по одной записи для каждого файла этого каталога. Каждая каталоговая запись максимально проста, так как в системе UNIX используется схема i-узлов (см. рис. 6.12). Каталоговая запись состоит всего из двух полей: имени файла (14 байт) и номера i-узла для этого файла (2 байт), как показано на рис. 6.33. Эти параметры ограничивают количество файлов в файловой системе числом 64 К.

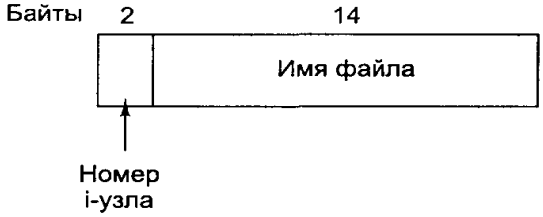

Рис. 6.33. Каталоговая запись файловой системы UNIX V7

Как и i-узлы на рис. 6.12, i-узлы системы UNIX содержат некоторые атрибуты. К этим атрибутам относятся такие параметры, как размер файла, три указателя времени (создания, последнего доступа и последнего изменения), идентификатор владельца, номер группы, информация о защите и счетчик каталоговых записей, указывающих на этот i-узел. Последнее поле необходимо для связей. При добавлении новой связи к i-узлу счетчик в i-узле увеличивается на единицу. При удалении связи счетчик в i-узле уменьшается на единицу. Когда значение счетчика достигает нуля, i-узел освобождается, а блоки диска, которые занимал файл, возвращаются в список свободных блоков.

Для учета дисковых блоков файла используется обобщение схемы, показанной на рис. 6.12, позволяющее работать с очень большими файлами. Первые 10 дисковых адресов хранятся в самом i-узле. Таким образом, для небольших файлов вся необходимая информация содержится прямо в i-узле, считываемом с диска при открытии файла. Для файлов большего размера один из адресов в i-узле представляет собой адрес блока диска, называемого **одинарным косвенным блоком**. Этот блок содержит дополнительные дисковые адреса. Если и этого недостаточно, используется другой адрес в i-узле, называемый **двойным косвенным блоком**, и содержащий адрес блока, в котором хранятся адреса однократных косвенных блоков. Если и этого мало, используется **тройной косвенный блок**. Полная схема показана на рис. 6.34.

При открытии файла файловая система по имени файла находит его блоки на диске. Рассмотрим на примере открытие файла */usr/ast/mbox*. В качестве примера будем использовать файловую систему UNIX, хотя основы алгоритма одинаковы для всех иерархических каталоговых систем. Сначала файловая система открывает корневой каталог. В системе UNIX его i-узел располагается в фиксированном месте диска. По этому i-узлу система определяет положение корневого каталога, который может находиться в любом месте диска, в данном примере — в блоке 1.

Затем файловая система считывает корневой каталог и ищет в нем первый компонент пути, *usr*, чтобы определить номер i-узла файла */usr*. Определить местоположение i-узла несложно, так как для каждого из них предусмотрено фиксированное место на диске. По этому i-узлу файловая система находит каталог */usr* и находит в нем следующий компонент, *ast*. Найдя описатель *ast*, файловая система получает i-узел для каталога */usr/ast*. По этому i-узлу она получает доступ к самому каталогу, в котором ищет файл *mbox*. При этом i-узел файла *mbox* считывается в память и остается там, пока файл не будет закрыт. Процесс поиска проиллюстрирован на рис. 6.35.

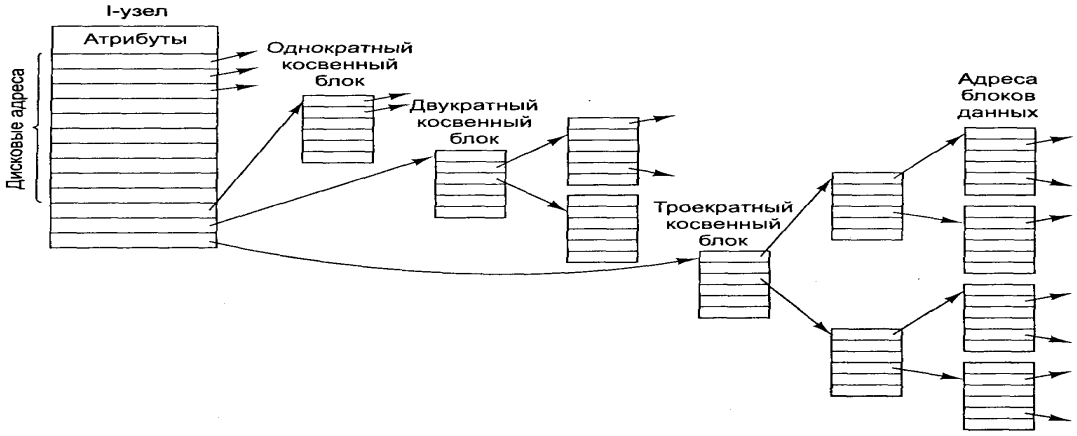

Рис. 6.34. I-узел файловой системы UNIX V7

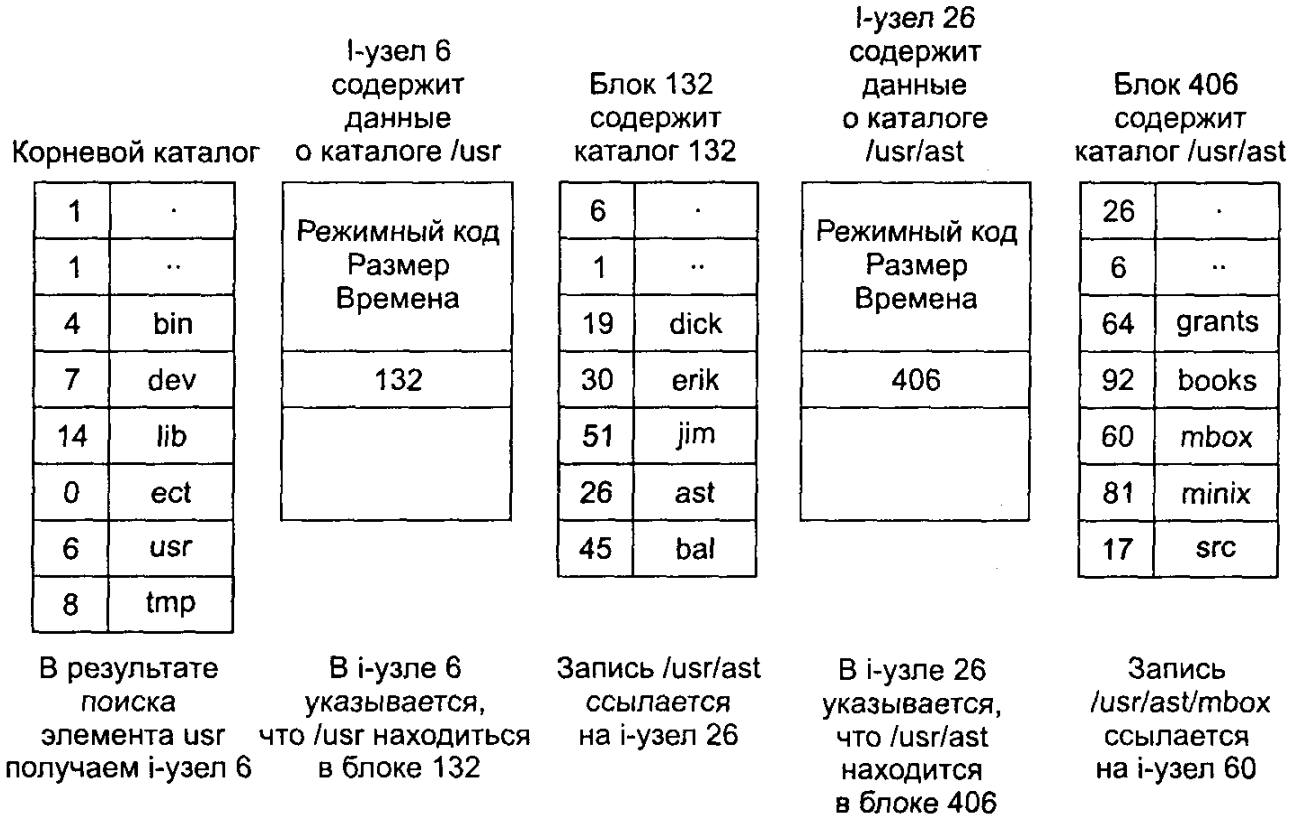

Рис. 6.35. Этапы поиска файла */usr/ast/mbox*

Относительные пути файлов обрабатываются так же, как и абсолютные, с той разницей, что алгоритм начинает работу не с корневого, а с рабочего каталога. В каждом каталоге есть элементы «.» и «..», помещаемые в каталог в момент его создания. Элемент «.» содержит номер i-узла текущего каталога, а элемент «..» — номер i-узла родительского каталога. Таким образом, процедура, ищущая файл *../dick/prog.c*, просто находит «..» в рабочем каталоге, разыскивает в нем номер i-узла родительского каталога, в котором ищет описатель каталога *dick*. Для обработки этих имен не требуется специального механизма. Имена рабочего и родительского каталогов представляют собой обычные ASCII-строки, не отличающиеся от любых других имен.

4) Особенности ОС с распределенной обработкой информации.

Распределенная система обработки данных (РСОД) или распределенная вычислительная система (РВС) — это среда, в которой компоненты системы или ресурсы: процессоры, память, принтеры, графические станции, программы, данные и т.д., связанны вместе посредством сети, которая позволяет пользователям представлять ВС как единую вычислительную среду и иметь доступ к ее ресурсам [52]. Распределенная ВС имеет некоторые особенности, характерные только для нее:

*0 *Ограниченность*. (failure). Она связана с тем, что количество узлов в сети ограничено и они являются независимыми компонентами сети.

*1 *Идентификация.* (naming). Каждый из ресурсов сети должен однозначно идентифицироваться.

*2 *Распределенное управление* (distributing control). Каждый из узлов должен иметь программно-аппаратные возможности выполнения функций управления сетью. Однако повышенные требования к надежности распределенной системы вызывают необходимость избегать применения алгоритмов управления, выделяющих привилегированные по сравнению с другими узлы в сети.

*3 *Гетерогенность* (heterogeneity). Узлы сети могут быть разнородны, с различной длиной слов и байтовой организацией. Этот аспект может касаться и программного обеспечения, как прикладного так и системного, применяемого для каждого из узлов.

При написании прикладных программ пользователям не требуется знать особенности аппаратного обеспечения сети. Пользователь распределенной вычислительной системы не обязательно должен иметь представление о деталях системы, с которой он работает. Все необходимые пользователю действия при взаимодействии с системными ресурсами сети обеспечиваются для него посредством операционной системы.

Рассмотрим более подробно тот раздел программного обеспечения, который, собственно, и решает задачи, связанные с обеспечением сокрытия вышеупомянутых особенностей системы и облегчением работы пользователя в распределенной вычислительной среде, а также обеспечением эффективного функционирования РСОД. В дальнейшем для определения этого системного программного обеспечения используется термин "распределенная операционная система" (РОС) (distributed operating system).

Ответ:

Доп.инфа:

---

5 Отличие канала от сообщения в межпроцессорном обмене.

СООБЩЕНИЯ

передача сообщений. Этот метод межпроцессного взаимодействия использует два примитива: send и receive, которые скорее являются системными вызовами, чем структурными компонентами языка (что отличает их от мониторов и делает похожим на семафоры). Поэтому их легко можно поместить в библиотечные процедуры, например

```
send(destination, &message);
receive(source, &message);
```

Первый запрос посылает сообщение заданному адресату, а второй получает сообщение от указанного источника (или от любого источника, если это не имеет значения). Если сообщения нет, второй запрос блокируется до поступления сообщения либо немедленно возвращает код ошибки.

ШПОРА

Каналы

Именованные каналы – любой процесс может послать информацию или забрать канал.

Неименованные каналы – могут быть созданы только между родственными процессами.

Канал имеет вход, выход и буфер. Недостаток канала: если в канале есть информация (сообщения), то мы можем считывать, пока не освободим канал; объем считываемой информации определяется пользователем.

Если описатель пишет в канал, а он заполнен, то блокируется (это не критическая блокировка). Также есть проблемы при освобождении (попытка считать пустой буфер) из-за заполнения буферов.

Очереди сообщений или сообщение считаются более информационно емкими в межпроцессорном взаимодействии и являются составляющей частью ОС и разделенным операционным ресурсом.

В системе существует или может существовать несколько очередей сообщений. Поэтому, каждая очередь именованная. Можно создавать мультиплексирование сообщений.

Система поддерживает специальные структуры для каждой очереди в виде связного списка и списко-указатель на само сообщение. Надо знать в очереди: адрес, размер и кому предназначено.

Каналы

Средства локального межпроцессного взаимодействия реализуют высокопроизводительную, детерминированную передачу данных между процессами в пределах одной системы.

К числу наиболее простых и в то же время самых употребительных средств межпроцессного взаимодействия принадлежат каналы, представляемые файлами соответствующего типа. Стандарт POSIX-2001 различает именованные и безымянные каналы. Напомним, что первые создаются функцией mkfifo() и одноименной служебной программой, а вторые - функцией pipe(). Именованным каналам соответствуют элементы файловой системы, ко вторым можно обращаться только посредством файловых дескрипторов. В остальном эти разновидности каналов эквивалентны.

Взаимодействие между процессами через канал может быть установлено следующим образом: один из процессов создает канал и передает другому соответствующий открытый файловый дескриптор. После этого процессы обмениваются данными через канал при помощи функций read() и write().

РАЗНИЦА В ТИПЕ ВЗАИМОДЕЙСТВИЯ ПРОЦЕССОВ

1 к 1 - КАНАЛЫ

1 к многим -- multicast сообщения

1 к нескольким или несколько к одному