БИЛЕТ № 4

1)Супервизор, его функции и состав.

Супервизор – это все управляющие программы, часть из которых размещается в ОП и называется ядром Супервизор

В супервизоре принимаются решения по использованию ресурсов, необходимых для выполнения его многочисленных функций. В него входят менеджер реальной памяти, менеджер виртуальной памяти, менеджер ресурсов и менеджер вспомогательной памяти, В нем также содержатся программы, управляющие мультипрограммной работой.

Супервизор — обеспечивает управление мультипрограммированием. Он создает диспетчеризуемую единицу работы, осуществляет переключения (диспетчеризация), обеспечивает последовательное использование ресурсов (например, предоставляет средства, обеспечивающие взаимоисключение).

Место, где находится резиденция системы, называется " резидентным" гомом, а сама система "резиденцией". Из всех программных модулей в резидентном томе часть используется по мере необходимости, а некоторые нз них обеспечивают базовое функционирование ВС и составляют *резидентные* программы, находящиеся в системной области оперативной памяти. Совокупность программ, обеспечивающих функционирование ВС и находящихся в системной области оперативной памяти, составляет ядро супервизора.

При загрузке ОС необходимо выполнить связывание, размещение всех частей, входящих у ядро супервизора, т.е. размещение резидентных программ на своих местах, формирование специальных системных структур данных в области данных операционной системы, формирование постоянной области, активизацию процессов для начала работы операционной системы. Этот процесс называется *инициализацией* операционной системы.

Доп. Инфа

Как обрабатывается прерывание по обращению к супервизору.

Прерывания по обращению к супервизору. Вызываются при выполнении процессором **команды обращения к супервизору** (вызов функции операционной системы). Обычно такая команда инициируется выполняемым процессом при необходимости получения дополнительных ресурсов либо при взаимодействии с устройствами ввода/вывода.

Структура ядра супервизора.

Совокупность программ, обеспечивающих функционирование ВС и находящихся в системной области оперативной памяти, составляет ядро супервизора.

Структура ядра:

-- резидентные программы

-- системные таблицы

2) Потоки и особенности их применений при организации вычислений.

Внутри процесса могут сосуществовать несколько параллельных потоков (нитей). Каждый из таких потоков является по сути "процессом в процессе", но имеет существенное отличие - поток не имеет собственных ресурсов. Поэтому потоки делят между собой адресное пространства процесса, в котором они были порождены. Потоки могут успешно применятся как для обеспечения кажущегося совмещения, так и для истинного параллелизма в мультипроцессорных системах. Поскольку потоки делят адресное пространство своего процесса, необходимо следить за выполнением синхронизации и взаимного исключения, если потоки зависят друг от друга и используют общие данные.

3) Теория рабочего множества и ее влияние на эффективность систем управления памятью. Особенности реализации в современных ОС.

Доп. Инфа

Понятие окна рабочего множества.

Процессы начинают работать, не имея в памяти необходимых страниц. В результате при выполнении первой же машинной инструкции возникает page fault, требующий подкачки порции кода. Следующий page fault происходит при локализации глобальных переменных и еще один - при выделении памяти для стека. После того как процесс собрал большую часть необходимых ему страниц, page faults возникают редко.

Таким образом, существует набор страниц (P1, P2, ... Pn), активно использующихся вместе, который позволяет процессу в момент времени t в течение некоторого периода T производительно работать, избегая большого количества page faults. Этот набор страниц называется рабочим множеством W(t,T) (working set) процесса. Число страниц в рабочем множестве определяется параметром T, является неубывающей функцией T и относительно невелико. Иногда T называют размером окна рабочего множества, через которое ведется наблюдение за процессом (см. рис. 10.3).

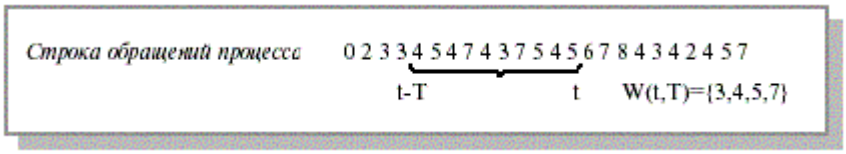

Рис. 10.3. Пример рабочего множества процесса

Легко написать тестовую программу, которая систематически работает с большим диапазоном адресов, но, к счастью, большинство реальных процессов не ведут себя подобным образом, а проявляют свойство локальности. В течение любой фазы вычислений процесс работает с небольшим количеством страниц.

Когда процесс выполняется, он двигается от одного рабочего множества к другому. Программа обычно состоит из нескольких рабочих множеств, которые могут перекрываться. Например, когда вызвана процедура, она определяет новое рабочее множество, состоящее из страниц, содержащих инструкции процедуры, ее локальные и глобальные переменные. После ее завершения процесс покидает это рабочее множество, но может вернуться к нему при новом вызове процедуры. Таким образом, рабочее множество определяется кодом и данными программы. Если процессу выделять меньше кадров, чем ему требуется для поддержки рабочего множества, он будет находиться в состоянии трешинга.

Принцип локальности ссылок препятствует частым изменениям рабочих наборов процессов. Формально это можно выразить следующим образом. Если в период времени (t-T, t) программа обращалась к страницам W(t,T), то при надлежащем выборе T с большой вероятностью эта программа будет обращаться к тем же страницам в период времени (t, t+T). Другими словами, принцип локальности утверждает, что если не слишком далеко заглядывать в будущее, то можно достаточно точно его прогнозировать исходя из прошлого. Понятно, что с течением времени рабочий набор процесса может изменяться (как по составу страниц, так и по их числу).

Наиболее важное свойство рабочего множества - его размер. ОС должна выделить каждому процессу достаточное число кадров, чтобы поместилось его рабочее множество. Если кадры еще остались, то может быть инициирован другой процесс. Если рабочие множества процессов не помещаются в память и начинается трешинг, то один из процессов можно выгрузить на диск.

4) Принципы повышения эффективности файловых систем.

Ответ:

Доп.инфа:

Производительность ФС:

-кєширование(использовать буфера в ОП) –для быстрого поиска в кєше используется хєш таблица

-опережающее чтение блока

-снижение времени перемещения блока головок

-файловая система с журнальной структурой LFS (очень мутная)

Кэширование

Для минимизации к-ва обращений к диску применяется блочный кэш или буферный кэш (набор блоков, логически принадлежащих диску, но хранящихся в ОП).

Перехватка всех запросов чтения к диску и проверке наличия требующихся блоков в кэше. Если блок присутствует в кэше, то запрос чтения блока может быть удовлетворен без обращения к диску. В противном случае блок сначала считывается с диска в кэш, а оттуда копируется по нужному адресу памяти. По следующие обращения к тому же блоку могут удовлетворяться из кэша.

Опережающее чтение блока

Получение блоков диска в кэш прежде, чем они потребуются. Многие файлы считываются последовательно. Когда файловая система получает запрос на чтение блока к файла, она выполняет его, но после этого сразу проверяет, есть ли в кэше блок к + 1. Если этого блока в кэше нет, файловая система читает его в надежде, что к тому моменту, когда он понадобится, этот блок уже будет считан в кэш. В крайнем случае, он уже будет на пути туда.

Если обращения к блокам файла производятся в случайном порядке, опережающее чтение не помогает.

Снижение времени перемещения блока головок

Другой важный метод состоит в уменьшении затрат времени на перемещение блока головок. Достигается это помещением блоков, к которым высока вероятность доступа в течение короткого интервала времени, близко друг к другу, желательно на одном цилиндре. Когда записывается выходной файл, файловая система должна зарезервировать место для чтения таких блоков за одну операцию. Если свободные блоки учитываются в битовом массиве, а весь битовый массив помещается в оперативной памяти, то довольно легко выбрать свободный блок как можно ближе к предыдущему блоку. В случае когда свободные блоки хранятся в списке, часть которого в оперативной памяти, а часть на диске, сделать это значительно труднее.

---

5) Фазы прерываний и их особенности.

В процессе обработки прерывания можно выделить следующие **фазы прерывания** (рис. 2.20):

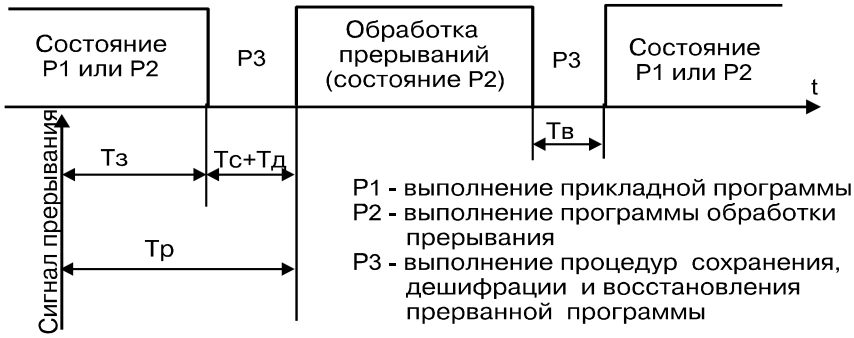

Рис. 2.20. Фазы прерывания

1. $T_з$ — **время задержки** между моментом возникновения сигнала прерывания и прерыванием активного процесса. Оно зависит от принятого в системе (процессоре) способа обработки сигнала прерывания (смотри выше).
2. $T_с$ — **время сохранения** необходимой информации. Зависит от количества сохраняемой информации при принятом способе обработки сигнала прерывания.
3. $T_д$ — **время дешифрации** сигнала прерывания. Зависит от аппаратуры, дешифрирующей сигнал прерывания.
4. $T_в$ — **время восстановления** прерванного процесса. Зависит от количества восстанавливаемой информации.

Время между возникновением сигнала прерывания и началом выполнения обработчика прерывания называется **временем реакции системы на сигнал прерывания** ($T_р$).

Для реализации механизма прерываний необходима аппаратная схема фиксации сигнала запроса на прерывание. Такая схема обычно содержит регистр, на котором фиксируется наличие сигналов во входных линиях **запросов на прерывания**. Объединенные схемой "ИЛИ", сигналы с разрядов регистра формируют общий сигнал о наличии запроса на прерывание. Затем все разряды регистра опрашиваются в порядке приоритетов входных линий. При этом используются 2 варианта реализации опроса:

1. Полноупорядоченная схема**.** Регистры опрашиваются по порядку приоритетов, от высшего к низшему. Это простая схема, однако при возникновении прерываний с низким приоритетом необходимо проверить *все* регистры запросов на прерывания с более высоким приоритетом.
2. Частично упорядоченная схема**.** Все прерывания делятся на **классы** и вводится 2-уровневая система приоритетов: первый уровень — среди различных классов, второй — внутри каждого класса. При этом, сначала по полноупорядоченной схеме определяется класс прерывания, на которое поступил запрос, а затем, также по полноупорядоченной схеме внутри установленного класса, определяется само прерывание. Это ускоряет поиск прерывания с низким приоритетом, однако усложняет процедуру поиска.