















[illegible]



**Вопрос 97**  
**Принципы синхронизации Windows. EventObject, Mutex, Mutual.**  
**ЕвентОбъект**  
• Может обращаться вручную, автоматически и гибридо  
• Используются для оповещения о наступлении события для типов: Синхронизации одним из нитей, нитей события для типов: Когда наступит состояние Signalred, EO автоматически сбрасывается в Not-Signaled  
• Оповещение происходит как вручную, требует ручного сброса состояния  
• Операции  
KeinWaitableObject - Initiale an event object  
KeinWaitEvent - Set/Reset event object data atomically  
KeinWaitableEvent - Read state of event object  
KeinWaitEvent - Set event object to Not-Signaled state  
KeinWaitEvent - Set event object to Signaled state  
По сути он содержит немого время нити **CRITICAL\_SECTION, HEADER.**  
Спрос вручную дождаться события, необходимо вызвать процедуру которую сбрасывает состояние оивидания.  
Pulse() - "Мы уже вводим и оно сразу сбрасывается, тоем мымисл"

В основном используется для оивидания в общем смысле наступления какого-либо системного события: один поток это событие регистрирует, а другие несколько потоков будут это состояние оивидать.

**MutualEx Muto**  
• Mutually exclusive, deadlock free доступ к разделяемым ресурсам  
• В нормальном режиме создается в «granted» состоянии  
• Возможно рекурсивный захват (сколько захватов, столько освобождений)  
• Завлаченный mutex блокирует выход из Kernel Mode  
• Сигнал Lock mutext: любой поток (abandoned state):  
mutex - выданы  
• В mutex запрещены APC и Sleep - нет.  
• Возможно использовать mutex в Userland  
• Испытание - добавляется к переносу потока если wait satisfied  
• Wait за KeinWaitable сразу создат функции ожидания (освобожд и звать в рамках катодной операции)

Решение в ядре бытны мислосы в мутуэ, но со временем мислос стал реализован через основные функции поддержки мутуэ. Эм-примитив имеет общую структуру - XMMUNIT.  
Состоит из:  
**Header** - заголовка  
**OwnerThread** - поток, который захватил блокировку  
**FastWaitable** (использованный выск процессор) - разрешение или запрещение  
**Abandoned** - является ли блокировка похищенной.

Если использовать мутуэ в Userland, то они как бы синхронизованы между собой и не создают проблем.

Когда оивидание закончилось, то необходимо потоком прирест процесс, чтобы событие было быстрее обработано.

Wait - необходимо, чтобы диспетчер выполнил операцию как одно целое, чтобы не произошло дублицирования и процесс не ушел на другой процессор.  
Почем работем  
• Соединяет мислосы в состоянии signaled.  
• Первый wait попадет в критическую область и при этом можно будет эту блокировку захватить (signaled = 0).  
• Следующий wait будет оивидан (signaled = 0).

Освобождение блокировки происходит через KeinWaitableMutex.

**Вопрос 98**  
**Принципы синхронизации Windows. Fast mutex, Guarded mutex.**  
**Fast mutex и Guarded mutex**  
• Быстрые мислосы на «нормальных» мислосах  
• Point Count: 0-6 bit - lock, 1-4 single write woken  
• Невозь захватывать рекурсивно  
• Операции:  
KeinWaitableObjectMutex  
EAsQuickWaitableObject  
EAsQuickWaitableObject  
EAsQuickWaitableObject  
EAsQuickWaitableObject  
• До Windows 8 разные реализации, после критичный рекурсивный мислос - это особый вариант взаимного исключения (мислоса), которое может быть заблокировано несколько раз одним и тем же процессором, но выходя взаимноблокированно. Главная чтобы блокировка и разблокировка был одинакового количества.

Оивидание мислоса происходит локально по отношению к процессору.  
Point - количество оивиданий. 0 bit - lock, 1 - signal write woken.  
**Owner** - кто, кто захватил мислос.  
**Contention** - есть ли оивидание за захват  
**idling** - уровень прерываний  
Guarded mutex - always fast mutex.

**Вопрос 99**  
**Принципы синхронизации Windows. Semaphore, spinlock**  
**Semaphore**  
• Управление доступом разделяемого ресурса  
Limit - максимальное количество титов ресурса  
• Count - неимальное состояние, помещается в Header.SignalRedate  
• Семафор оивид, когда SignalRedate = 0  
• Если SignalRedate=Adjustment + Semaphore Limit  
вызывается KeinWaitableObject(STATUS\_SEMAPHORE\_LIMIT\_EXCEEDED)  
• Испытание - добавляется к переносу потока если wait satisfied  
• Wait - на KeinWaitableObject сразу следует функции оивидания (освобожд и звать в рамках катодной операции)  
Точно также как и мислосы создается в signaled state.  
Оивидание дублируется с тем, что вышло  
SignalRedate + Adjustment + Semaphore Limit (если значение свободной ресурса превышает максимальное допустимое, то выисывается исключение KeinWaitableObject).

**Spinlock (Guarded Spinlock)**  
• Используются атомарная операция test-and-modify  
Синхронизация на мислососных системах при помощи локальной переменнй.  
• Один владелец и один момент времени  
• Реализация ядра являю мислос  
trylock (LOCK, RPT, KERN, LOCK)  
• До Windows XP рекомендуется использовать AcquireWaitLock

Также как и на явлюе оивидание предизначение - быстрая обработка при ивенте оивидания, и в стропе мислос.  
Есть оивиды и разные локальные переменные, чтобы считать busy contention, когда атомарная операция блокирует шину при оивиде блокировки.  
Младшие биты также используются, чтобы помешать что есть оивидание на другой блокировке.

Интересный факт стоит не.  
Снижение довольно часто с драйверов.  
Во время оивидания повышается уровень нити, чтобы другой процесс не мог быть запланирован на шину.

**Вопрос 100**  
**Механизмы таблицы страниц SPARC64.**  
В простейшем случае (не спид) она выглядит так. Как это работает:

На выходе есть 64-битный адрес. Он попадает в 16-бит-функцию, и в результате получается 16-разрядный адрес, который используется как мислос в таблице. В таблице хранится запись о виртуальном адресе, переводящий который существует. Далее происходит сравнение **pageing on**, который существует с заданным. Матрица в форме "бидент" связана **chainon**. Затем по chainu вы linked list мы проходимся и находим искомую как запись физического адреса. Если в этой ячейке ноль, то других связанных записей нет. После данной реализации: сокращение таблицы **pageing on**.

**Разделение SPARC64.**  
Страничный захват и неше определяет расположение одной виртуальной страницы и называется TTE (Translation Table Entry). TTE состоит из тета и данных. Размер того и другого - 64 бита.  
Почем осуществляется по тету, состоянию из коимее количества и номера виртуальной страницы. При совпадении используется информация из Data - номер физической страницы (VA), и т.д.  
В строке TTB промака, управление передается к ОС соответствующему обработчику промака. ОС может делать что угодно для поиска физ адреса, однако разумно предположить, что сначала она посмотрит в специальную структуру TSB (Translation Storage Buffer).  
Для каждого процесса существует свой набор таких структур. Кроме того у каждого процесса имеется таблица TSB, расположенная в основной памяти и используемая в качестве программно-управляемого кэша. Туда помещается ограниченный количество записей о расположении виртуальных страниц. Овее строка TSB занимает 128 биты по содержанию практически совпадает с TTE, размещающаяся в TTB.  
Набор структуры **tlbe\_3bits** оивидает одно виртуальное адресное пространство в организовано в виде **двухразрядной таблицы, с адресацией.**  
После TTB промака, ОС посылаете найти запись о запрошенном виртуальном адресе в TSB. Если запись найдена, то ОС загрузит ее в TTB, иначе попытается найти ее в **tlbe\_3bits**.

**Вопрос 101**  
**Виртуальная память Linux. 3.2 - разделение модели.**  
Любой процесс, который запущен, состоит из сегментов. Накало памяти (симу) старается давать неразмещенные, чтобы код, который работает в стропе операционных систем, чтобы им вызывал page fault и эта память была недоступна.  
Do-NOBODY4000 - память неразмещенная.  
Сегменты:  
Text (code) - чужой код  
Data - чужие данные  
BSS (zero set segment) Data - статические переменные, инициализированные нулями данные.  
Heap:  
Начинается с start, брт, растет до bkt (program break) вверх. Malloc как правило (если не реализован через mmap) двигает указатель bkt.  
Mmap:  
Решение как вверх, сайе-а-вниз. Туда загружаются библиотеки или когда мы вызываем **urcall mmap**.  
Если mmap и heap слишком большие, то чтобы данные не перемешались, то получим ENOMEM.

Решение: За стекном кэшем (кэше бу) находится защитная область памяти.  
Stack: Выск. За стекном кэшем (кэше бу) находится защитная область памяти.  
Enb in **split**: Там хранятся переменные оивидения.  
На kernel user память делится 1,3.

**Данные**  
Делится на 3 зоны: ZONE\_DMA, ZONE\_NORMAL, ZONE\_HIGHMEM.  
ZONE\_DMA: находится область памяти для старых устройств (16MB).  
ZONE\_NORMAL: содержит сегмент кода ядра (12MB).  
ZONE\_HIGHMEM: по юмца.

До BSEEM находится область памяти **lowmem**, которая один с другим не является в нулевой адрес. Поэтому юмца только одна запись в TTB.

**Highmem**: Для них можно создать запись в **highmem** и переключить страницы по необходимости. Так можно выделить динамическую память (или кэш), матрица для ввода вывода (inputmap), **inputmap**: подключение и отключение страниц по необходимости, **inputmap**: временные kernel матрицы, чтобы не было подгрузки страниц и т.д.

Адро может менять порядок процесса через регистр **cr2**, который указывает на таблицу страниц.

**Вопрос 102**  
**Виртуальная память Linux. 6.4 - разделение модели.**  
Следует обратить внимание, что слова **четырёхуровневый** (4T бит), а строка **пятиуровневая** таблица страниц (48 бит).  
**Page table** - обусловлена необходимостью развить эти адреса, так как каждый памяти юмца. Поэтому пришло решение расширить старый знак. В связи с этим данный кусок памяти не может быть **small-разделен**. Четырёхуровневый - 16 миллионов **тербайт**, в пяти - 16 тысяч **тербайт**. Данные начинаются каро.  
**Page-table isolation**: для защиты ядра. Из пользовательского пространства выисков нельзя получить и памяти доступ по всей памяти ядра.  
**Pagefault**: ХТТ устранил эти утечки, полностью разделив таблицы страниц пространства пользователя и пространства ядра. Один **table** страниц, выискает адреса пространства ядра и пространства пользователя, как и раньше, но не используется только тогда, когда система работает в режиме ядра. Второй набор таблиц страниц для использования в пользовательском режиме содержит юмца пользовательского пространства и минимальный набор отбражений пространства ядра, который предоставляет информацию, необходимую для ввода или вывода из системных выисков, **preemption** и **исключения**.  
Так как адресное пространство юмца больше, то вся физическая память последовательно матит в пространство ядра начиная с **page\_offset\_base**. То есть для того, чтобы получить доступ, юмца **номер** бизнесного файла, прибавить к **page\_offset\_base** и получить нужный виртуальный адрес.

**vmalloc/kegmem** - область динамической аллокации.  
**input\_map** - там находится структуры, описывающие страницы. То есть если ядро юмца создаст страницу, то оно все системные структуры поинтит туда.

данные находятся куда юмца: KASAN, EFL, доп. отладочная информация.  
**Kernel text** - код ядра  
**Modules** - модули ядра  
**Page** - временная таблица для временных матриц, которые не выисываются блокировкой.

**Вопрос 103**  
**Виртуальная память Linux. Структуры памяти.**  
В первую очередь это **tbl\_struct** (в юмца юмца (улу сетки)) юмца, где расположено информация, относящаяся к процессу и находящаяся в памяти ядра. Из **tbl\_struct**, где находится описание памяти для каждого процесса.  
Здесь выделено несколько переменных (парам из них указывают на mmap, Mmap - эта такая штука, которая предоставляет связанный список всех сегментов текущего процесса (предоставляющий **исключения** справа)).  
Первый **tbl\_struct**, который сегмент оивидается структурный **tbl\_page\_struct**. И в этой структуре находится описательные элементы этого сегмента (например, есть адрес, где сегмент находится и где заканчивается **tbl\_start**, **tbl\_end**, есть **tbl\_page**, указывающая на то, что разделение дается в этой самой странице: **thead**, **exec**, и есть указатели на **следующий** и **предыдущий** сегмент, чтобы в случае необходимости не юмца было подгрузить **исключения**).  
В **tbl\_page\_struct** в юмца **вид** под каждым сегментом подписано, как блокируется данные. Т.е. условия говоря, что юмца под этим самым сегментом. Имеется в виду, что сегменты **нужны** себе юмца либо свои **область** и зарезервированное место в сегменте, либо они хранят юмца себе **файл**.  
И, соответственно, есть структура (**tbl\_struct**, **tbl\_page\_struct**), которая соответствует каждому сегменту, т.е. в явном виде есть ссылка на файл прямо из структуры **tbl\_page\_struct**, где оивидается, что это файл, и там есть **tbl\_page**, **tbl\_page**, которое указывает на юмца файла.  
Кроме того, есть анимированный **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца юмца адресу, существует юмца **tbl\_page\_struct**, юмца **tbl\_page\_struct**. Переменная в **tbl\_struct** (называемая **tbl\_page\_struct**) юмца **tbl\_page\_struct** по сегменту, необходимому для того, чтобы юмца необходимый сегмент, который оивидает тот юмца **tbl\_page\_struct**.  
Таким образом, все сегменты, которые имеют юмца **tbl\_page\_struct**, имеют ссылку на **tbl\_page\_struct**. Все сегменты, которые анимированы имеют структуру, соответствующую анимированной юмца.  
Страница на **tbl\_page\_struct** **tbl\_page\_struct** **tbl\_page\_struct** (показаны в юмца **tbl\_page\_struct** улу улу улу). И юмца **tbl\_page\_struct** структура, которая называется **tbl\_page\_struct**, которая оивидает анимированную область памяти.  
Следует отметить, что сегменты в текущем процессе могут быть юмца юмца. Поэтому, для того чтобы юмца сегмент по юмца



1. Архитектура компьютерных систем. Архитектура Фон-Неймана и Гарвардская архитектура. Принципы архитектуры Фон-Неймана.
2. Архитектуры NUMA и UMA.
3. Общая организация процессора, памяти, организации вычислений.
4. Организация прерываний, типы прерываний, контроллер прерываний.
5. Тетичные функции операционной системы. Интерфейсы ОС.
6. Работа ОС как замена оператора ЭВМ.
7. Платные обработка. Системный монитор.
8. Анализ общественной эффективности, как предусловие многозадачности. Многозадачность, как способ повышения системной эффективности. Системы разделения времени.
9. Процессы, проблемы современных процессоров. Планирование выполнения процессов и управление ресурсами.
10. Управление памятью, виртуальная память. Защита информации и безопасность ОС.
11. Структура ядра операционной системы. Архитектуры монолитного ядра, ядра динамически загружаемыми модулями и микроядра.
12. Поток исполнения, многопоточность, модели многопоточности.
13. Симметричная и асимметричная многопроцессорная обработка.
14. Виртуализация. Типы виртуализации.
15. Сбой и отказоустойчивость ОС. Причины появления отказов в ОС и способы борьбы с ними.
16. Надянность. Среднее время восстановления. Коэффициент доступности и время простоя.
17. Реагирование и отказоустойчивость.
18. История и развитие ОС GNU/Linux. Single UNIX Specification и POSIX.
19. Понятие дистрибутива, дистрибутивы Linux.
20. Архитектура и основные подсистемы Linux. Linux Kernel Map.
21. История и развитие Windows.
22. Общая архитектура Windows. Windows API.
23. Сервисы, функции и важные компоненты Windows.
24. Процесс, характеристика процесса в момент выполнения. Состояние процесса. Разделение ресурсов.
25. Модель процесса с типичными состояниями, назначение состояний.
26. Paging и Swapping. Модель процесса с семью состояниями.
27. Управление таблицей процесса. Обработка процесса.
28. Управляющий блок процесса (PCB), состав PCB.
29. Функции ОС, связанные с процессами. Создание процесса, переключение процессов.
30. Процессы в ОС UNIX SVR4. Диаграмма состояний, основные структуры.
31. Понятие потока выполнения, связь потока и процесса. Преимущества потоков.
32. Состояния потока, User Level Thread vs Kernel Level Thread.
33. Многопроцессорность и многопоточность. Закон Аmdala.
34. Механизм параллельных вычислений, функции ОС.
35. Проблемы параллельного выполнения: взаимное исключение, взаимоблокировка, голодание. Требования к взаимным исключениям. Уровни взаимного исключения в потоках.
36. Принципы синхронизации ОС. Предназначение примитивов синхронизации.
37. Принципы синхронизации ОС. Семафоры и мониторы. Бинарный семафор.
38. Принципы синхронизации ОС. Условные переменные, условия.
39. Принципы синхронизации ОС. Мониторы, флаги событий, передача сообщений.
40. Принципы синхронизации ОС. Неблокирующие примитивы синхронизации и неблокирующие структуры данных.
41. Управление памятью, основные определения и требования к организации.
42. Физическое и динамическое размещение программ в памяти.
43. Модели аппаратного перемещения программ.
44. Простой страничный поиск и простая сегментная организация.
45. Виртуальная память, основные определения и принципы организации аппаратуры и управления программ.
46. Виртуальный страничный обмен. Двухуровневая организация MMU и TLB BOSTHE.
47. Инвертированная таблица страниц.
48. Сегментно-страничная виртуальная память.
49. Влияние размера страницы виртуальной памяти на ОС.
50. Стратегия ОС по работе с виртуальной памятью.
51. Статистика замещения страниц ОС. Числовой Алгоритм.
52. Управление резидентной частью процесса.
53. Виды планирования процессов. Критерии краткосрочного планирования. Приоритеты.
54. Использование приоритетов.
55. Стратегии планирования FCFS, RR, SPN, SRT, HRRN, Feedback.
56. Feedback планирование и классы планирования ОС UNIX SVR4.
57. Сравнительное планирование.
58. Планирование в многопроцессорных системах. Типы многопроцессорных систем с точки зрения организации планирования. Граничность и проектирование планирования процессов и потоков для многопроцессорных систем.
59. ОС реального времени и планирование. Дебайне-планирование.
60. Проблема инверсии приоритетов, типы инверсии и способы решения в планировании.
61. Ввод-вывод. Сравнение устройств и скорости обмена, развитие способов ввода-вывода, логическая структура ввода-вывода.
62. Буферизация ввода-вывода. Ввод-вывод в UNIX SVR4.
63. Диски и дисковое планирование.
64. Концепция RAID.
65. RAID-0, 1, 10, 0+1.
66. RAID 4, 5, 6. Аппаратные дисковые массивы.
67. Файловый ввод-вывод, основные определения. Задачи ОС по управлению файлами. Совместное использование файлов.
68. Управление файлами в UNIX SVR4.
69. Каталог файлов. Элементы каталога, операции ОС.
70. Размещение записей и файлов в блоках данных. Сложность и типы организации размещения.
71. Непрерывное размещение файлов (на примере ОС RT-11).
72. Цепочечное размещение файлов (на примере ОС FAT).
73. Индексированное размещение (на примере файловой системы UNIX UFS).
74. Linux: стандартные средства для наблюдения сетчатого ядра.
75. Linux: файловая система /proc.
76. Linux: транскрипция системных вызовов и библиотек.
77. Linux: Профилирование perf и FlameGraph.
78. Linux: SystemTap.
79. Linux: Отладка ядра.
80. Windows: стандартные отладочные средства.
81. Windows: утилита Sysinternals.
82. Windows: отладчик WinDBG и ID.
83. Аппаратная поддержка взаимных исключений.
84. Элементы поиска и блокировки (Столлинг, гл. 5.1).
85. Принципы взаимного блокирования (Столлинг, гл. 6.1).
86. Поддержка взаимоблокировок, устранение взаимоблокировок, обнаружение блокировок. (Столлинг, гл. 6.2, 6.3, 6.4).
87. Ядро и обобщающие философия (Столлинг, гл. 6.6).
88. Процессы в Linux: структура task\_struct, поля структуры, связь с другими структурами ядра.
89. Диаграмма состояния процесса Linux.
90. Создание процесса Linux на уровне пользовательского процесса.
91. Создание и завершение процесса Linux на уровне ядра. Вызываемые функции.
92. Особенности реализации потоков в Linux. KThread. Tasklet.
93. Принципы синхронизации Linux. Spinlock и spinlock.
94. Принципы синхронизации Linux. Semaphore и Mutex.
95. Принципы синхронизации Linux. rw\_semaphore, seqlock.
96. Типы процессов и потоков Windows.
97. Структура процесса и потока в Windows. Поля структуры.
98. Диаграмма состояний процесса и потока Windows.
99. Создание и завершение процесса Windows.
100. Принципы синхронизации Windows. Понятие Dispatcher Object. Ожидание наступления события, вызовы Wait.
101. Принципы синхронизации Windows. EventObject, Mutex, Mutex.
102. Принципы синхронизации Windows. Fast mutex, Guarded mutex.
103. Принципы синхронизации Windows. Semaphore, spinlock.
104. Таблицы страниц, таблицы страниц, PAGING.
105. Виртуальная память Linux. 32-х разрядная модель.
106. Виртуальная память Linux. 64-х разрядная модель.
107. Виртуальная память Linux. Структуры памяти.
108. Способы выделения памяти для пользовательских процессов Linux.
109. Способы выделения памяти в пространстве ядра Linux.
110. Служба аллокации SLAB/SLUB/SLKM.
111. Copy on write и writeback в Linux.
112. Замещение страниц в Linux. Kswapd.