# Операционные системы Лекция (24.12.2021)

…

1. Distributed Wound-Die  
   …
2. Distributed Wait-Wait

…

Рассматривали транзакции (определение, свойства…) … надо рассматривать это комплексно.

…

1. Distributed Deadlock-Detection

…

3 варианта … (?Детекторов?)

1. Централизованный детектор тупика  
   То есть одна локация назначается центральным детектором взаимоблокировки.
2. Иерархический детектор тупика.,  
   Детекторы сформированы в иерархию. Дело разработчика – как будет построено их взаимодействие
3. Детектор распределённых тупиков.  
   Все сайты участвуют в обнаружении и устранении тупика.

Три показательные теоремы… (Из книги ?Шоу?)

1. Теорема 1 – Граф (речь идёт о ?биохроматическом? графе Холта) является полностью сокращаемым если существует такая последовательность сокращений, которая устраняет все дуги. Если граф нельзя сократить полностью, то анализируемое состояние является тупиковым (см. рисунок полностью сокращаемого графа (прошлая лекция))
2. Цикл в графе повторно используемых ресурсов является необходимым условием тупика.
3. Если состояние S не является состоянием тупика и из S система переходит в состояние T, то состояние T является состоянием тупика, если операция процесса Pi (переход от P к T), является запросом и в результате Pi находится в T (в тупике). (Коротко: Тупик может наступить в системе только в результате запроса на дополнительные ресурсы)

Алгоритм обнаружения тупиков в распределённых системах “Алгоритм Chandy-Misra-Haas”

Если при очередном запросе в системе необходимый ресурс занят, то процесс запросивший этот ресурс генерирует специальное сообщение и посылает его процессам, которые предположительно могли захватить этот ресурс. (Очевидно, что в этой системе любой процесс извещён о своих соратниках) А этом специальном сообщении указывается три числа: первое – свой номер в системе; второе число – также свой номер; третье поле – номер процесса которому посылается сообщение. Процесс, получивший сообщение, проверяет ждёт ли он сам ресурс, занятый другим процессом (не тот же самый). Если он сам ждёт освобождения какого-то ресурса, то прописывает свой номер во второе поле, полученного сообщения, в третье поле он записывает номер процесса, который захватил ресурс освобождение он ожидает и посылает сообщение дальше. В результате, такой многократной пересылки этого сообщения, если процесс получает сообщение и в первом поле этого сообщения обнаруживает свой номер и (главное) в третьем поле этого сообщение также обнаруживает этот номер => делает вывод что система попала в тупик (надо уничтожить этот процесс). Но если в тупик попало много процессов, то такой метод очень расточительный.

Блокирование транзакции происходит тогда, когда транзакция, запрашивающая ресурс моложе той, которая ресурс удерживает.

## Архитектура ядер операционных систем

<https://github.com/Sunshine-ki/BMSTU5_OS/wiki/02>

Типы архитектур:

1. Монолитное ядро
2. Микроядро

### Первый тип (исторически) монолитное ядро.

Монолитное ядро представляет из себя единую программу, состоящую из подпрограмм (subroutines). … в результате любое изменение, вносимое в такую программу, требует её перекомпиляции. Очевидно, что современные системы с монолитным ядром. (Windows, Unix/Linux)

### Второй тип – микроядро.

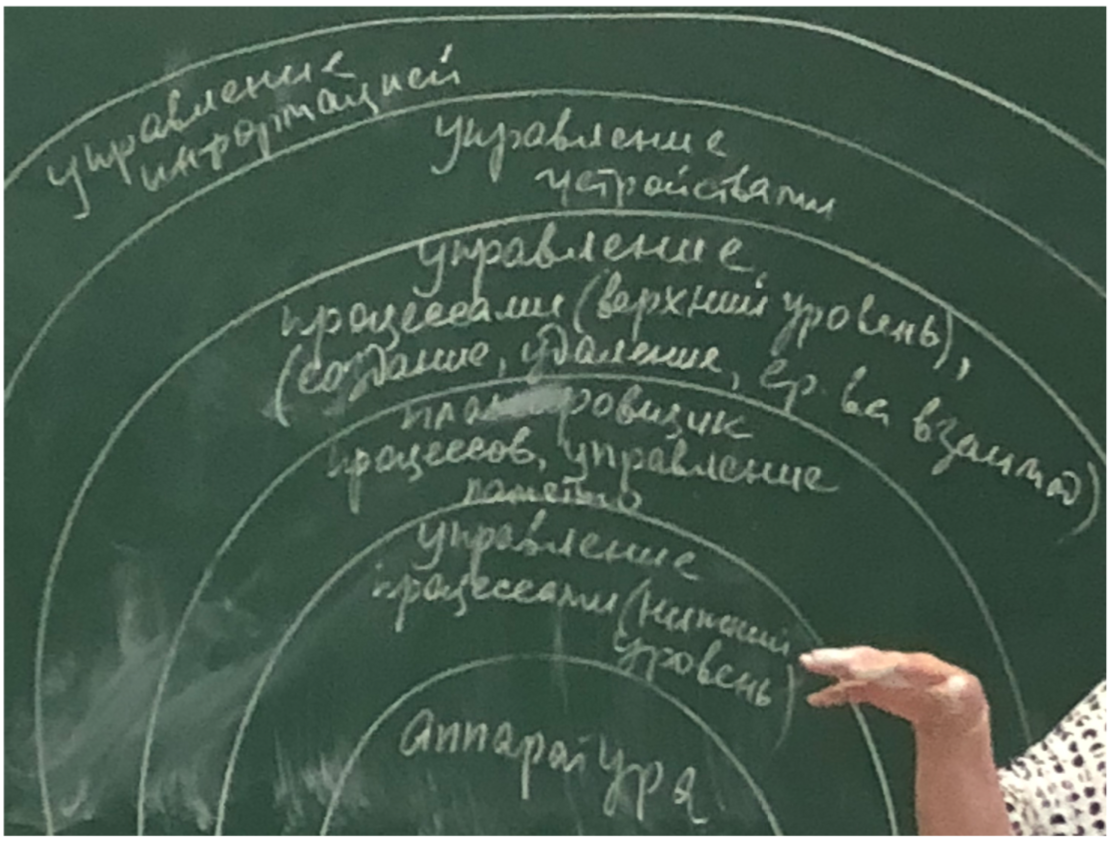
В отличии от монолитного, микро – это очень небольшое ядро, которое выполняет только самые низкоуровневые действия. При этом другие компоненты ОС являются самостоятельными программами, которые возможно выполняются в разных адресных пространствах, при чём в режиме пользователя.

Начнём с монолитного ядра. Ядра Unix/Linux являются минимизированными в отличии от Windows. Всё монолитное ядро построено на прерываниях. Существует система прерываний. В современных ОС выделяют 3 типа прерываний.

1. Системные вызовы (System call)
2. Исключения (Исключительные ситуации)
3. Аппаратные прерывания (Interrupts)

Системные вызовы, говорят, когда хотят подчеркнуть, что процесс обращается к системе за обслуживанием. Когда хотят, что ОС приложение предоставляет интерфейс говорят API, при этом API определена в POSIX. Но как мы видел функция signal не входи в POSIX, но она не перестаёт быть API.

Сама идея микроядерной архитектуры возникла не на пустом месте. Еще в середине 1960-х… Декларировали иерархическую машину Донавана. Она отображает расположение соответствующих функций ядра по отношению к аппаратной части. Чем ближе к аппаратной части, тем ниже уровень. (ЭВМ – это програмно управляемая машина) Часть времени управление исполняет ОС, а часть приложение.



Структура ядра системы: Программное обеспечение ставится на голое железо, на аппаратную часть.

* нижний уровень - аппаратная часть
* уровень 1 - управление процессорами (нижний уровень; диспетчеризация - выделение процессу процессорного времени);
* уровень 2 - (P, V) - семафорные операции, контр. доступа к разд. рес, планировщик процессов (операция более выс. уровня. постановка процессов в очередь к процессору или другим ресурсам), управление памятью; Процессор, оперативка, внешние устройства - аппаратные устройства
* уровень 3 - управление процессорами (верхний уровень -- создание и удаление процессов. Система принимает решение об этом, более высокоур. операция);
* уровень 4 - управление устройствами (подсистема ввода/вывода);
* уровень 5 - управление информацией (файловая система). Система предоставляет такие средства как наименование файлов, построения дерева катологов, удобных для пользователя.

Виртуальная машина - совокупность команд машины и команд ОС, которые могут использовать программы для получения сервиса ОС.