2 лекция, конспект

1. Мы говорили говорили говорили , итог все дела ,

```
Все программы исполняются OS API/OS Framework'ом ( обращаясь к ядру OC )

очевидные вещи

утилиты - системные проги с узким useg'em ( копи , паст и всё такое )

Нет другого способа работать с OC , кроме Frameworka OS

Никакой мистики
```

2. Системный вызов - спец механизм вызова прикладной программы ОС (которая не в ядре , но вызывают его функции) . СВ вызывается с помощью механизма ПРОГРАММНОГО ПРЕРЫВАНИЯ . Для этого процессор вызывает СВ своими инструкциями . для х86 - инструкция int , для х86_64 - SYSENTER/SYSEXIT , AMD - SYSCALL/ SYSRET . В составе ядра есть проги , которые являются обработчиками прерываний (handlers) . Прерывание - механизм , мы можем преостановить выполнение работы проца чтобы он исполнял другую прогу . После выполнения , он возращается к моменту , где его прервали.

ПП - программное прерывание.

Мы как программисты не можем вызвать инструкцию int напрямую . Поэтому вызываем с помощью спец функцию ПП

процессы , потоки , диспетчеризация , отсчёт времени , управления память , i/o , синхронизация - топики

OS много и каждая имеет специализацию (мобилки , embedded и т.д.)

Ядро гибкое короч, можно изменять, изгибать

переходим к след.части

3. Во всех ОС работает одинаково . В рамках ядра есть массив - ВЕКТОР ПРЕРЫВАНИЙ . Он живёт по 0-вому адресу . Представляет в х86 массив 4-байтных адресов . Если х64 - то 8 байт адреса . Когда СВ , то СВ задаёт 2 вещи - номер прерывания и параметры для передачи . Для этого он задаёт номер прерывания в команде , а в регистры запоминает параметры. Когда всё происходит , то проц АППАРАТНО умножает номер прерывания на 4/8 и вычисляет адрес в векторе , берёт адрес в векторе и исполняет по адресу . ВКУСНОТИЩАААААААААААААА

Надо сказать, что сам handler переключает процессор в другой режим работы. У проца есть РИНГИ (круги) - каждый круг включает в себя набор программ, которое может выполняться в этом круге (ШИРЕ == БОЛЬШЕ КОМАНД). Есть привелегированные команды, с которомы круг можем изменить.

Круги ща не юзаются, только 2 вроде используются. 2 режима - привелигированный и непривел.

В привелегированном - ВСЕ КОМАНДЫ, в непривел. - НЕ ВСЕ КОМАНДЫ. Кайф

В привел. режиме исполняется этот handler . У привел. режима ОТКЛЮЧАЕТСЯ ЗАЩИТА ПАМЯТИ И МОЖЕТ ЛЕЗТЬ КУДА УГОДНО .

дед какой-то эминем

СВ короч юзается повсеместно (в лекции пункт 4)

Процесс ОС - единица работы OS, объект (программный объект как в ООП) ядра

. Такой объект создаётся для каждого процесса . Процесс - это объект ядра + адресное пространство . (дальше инфа из 7 пункта) . При создании проца ему выделяется адресное пр-во (диапазон адресов работы программы) .

Всё что ОС юзает - это РЕСУРСЫ . Каждый проц создаётся другим процессом (родительским) . Страницы памяти - потом объяснит .

Процесс обычно соответствует программе . Программу запускает ЗАГРУЗЧИК . Этот загрузчик выполняет CB , по которому СОЗДАЁТСЯ ПРОЦЕСС .

В ядре ос хранится список ВСЕХ ПРОЦЕССОВ и его можно получить.

При остановке процесса инфа сохраняется в КОНТЕКСТЕ ПРОЦЕССА . Там инфа чтобы при возращении управления к процессу восстановить его работу и продолжить с того места где он остановился .

Все процессы изолированы друг от друга - ЭТО ЕГО ГЛАВНОЕ СВОЙСТВО . ИНКАПСУЛЯЦИЯ ВО ВСЕЙ КРАСЕЕЕЕЕЕЕЕЕЕЕЕ

В привел. режиме исполняется этот handler . У привел. режима ОТКЛЮЧАЕТСЯ ЗАЩИТА ПАМЯТИ И МОЖЕТ ЛЕЗТЬ КУДА УГОДНО .

дед какой-то эминем

СВ короч юзается повсеместно (в лекции пункт 4)

Процесс ОС - единица работы OS, объект (программный объект как в ООП) ядра . Такой объект создаётся для каждого процесса . Процесс - это объект ядра + адресное пространство . (дальше инфа из 7 пункта) . При создании проца ему выделяется адресное пр-во (диапазон адресов работы программы) .

Всё что ОС юзает - это РЕСУРСЫ . Каждый проц создаётся другим процессом (родительским). Страницы памяти - потом объяснит.

Процесс обычно соответствует программе . Программу запускает ЗАГРУЗЧИК . Этот загрузчик выполняет CB , по которому СОЗДАЁТСЯ ПРОЦЕСС .

В ядре ос хранится список ВСЕХ ПРОЦЕССОВ и его можно получить.

При остановке процесса инфа сохраняется в КОНТЕКСТЕ ПРОЦЕССА . Там инфа чтобы при возращении управления к процессу восстановить его работу и продолжить с того места где он остановился .

Все процессы изолированы друг от друга - ЭТО ЕГО ГЛАВНОЕ СВОЙСТВО . ИНКАПСУЛЯЦИЯ ВО ВСЕЙ КРАСЕЕЕЕЕЕЕЕЕЕЕЕ

Между процессами взаимодейсвие с помощью IPC (механизмы межпроцессного взаимодейсвия). IPC - БЕЗАЛЬТЕРНАТИВЕН

Процессу выделяется адресное пр-во . Ещё раз . Ещё раз . Ещё раз . Потом поговорим обо всём.

Смелову кстати душно.

Секция - 8 пункт лекции скрин

Контекст проца - это необходимая для сохранения инфа , для прерывания и восстановления процесса . Надо сохранить (список в пункте 9)

Если несколько процессов одновременно , то они могут переключатся между собой . В момент переключение - сохранение контекста .

Чё-то про потоки - пропустил я .

Есть системные процессы - находятся в рамках ядра и есть процессы , которые стартуют с ОС (в винде такие называются ВИНДОВС - СЕРВЕРЫ , в линуксе ДЕМОНЫ) ДЕМООООООООООН

Есть команда systemctl - ДЛЯ УПРАВЛЕНИЕМ ДЕМОНАМИ

дальше скрины всякие обсуждает для проверки процессов , всё по лекции Смелов рисует вилку онлайн без регистрации и комментариев