# Основни Елементи на КС.

Основните елементи на компютърната система са:

* Процесор;
* Основна памет;
* Входно-изходни модули;
* Системна шина.

Процесорът се характеризира с контрол на операции, като извършва и обработка на данни. Притежава входно-изходен адресен регистър, както и входно-изходни буферни регистри, а освен това и два вътрешни регистъра:

* Memory address register (MAR). В този регистър се съхраняват адресите или на данните, които ще бъдат хванати от процесора, или тези, към които ще бъдат изпратени данните. С други думи в MAR се съхранява информация за местоположението на данните, които ще се достъпват.
* Memory buffer register (MBR). В този регистър се съхраняват данните, които се трансферират от и към паметта с директен достъп. Работи като буфер между процесора и RAM паметта, позволявайки им да работят независимо, без да се влияят от малки промени в операцията.

Основната памет се нарича още RAM (памет с произволен достъп). Характерно за нея е загуба на данните при изключване на системата. В нея се съдържат последователни номера на адреси, както и данни, и инструкции. Притежава висока скорост. Изисква регенерация през определен период.

Входно-изходните модули придвижват данните между компютъра и външната среда. Такива устройства са твърдия диск (HDD), както и различни комуникационни елементи като клавиатура, мишка, монитор и др. Определят се от входно-изходния адресен регистър.

Комуникацията между процесора, главната памет и входно-изходните модули се осъществява благодарение на системната шина.



# Регистри. Видове и функционалност .

Процесорни регистри. Те са по-бързи и по-малки от основната памет. Ползват се за контролиране работата на самия процесор.

Видимите регистри могат да се използват чрез машинни езици. Дават възможност на програмиста да сведе до минимум обръщането към паметта чрез използване на регистър. Могат да съхраняват данни, адреси и регистри с условен код.

Регистрите за данни са с общо предназначение, като се прилагат някои ограничения.

Адресните регистри биват индексни, сегментен указател и указател за стек.

Регистри за контрол и статус. Регистърът за инструкции (IR) съдържа само най-скоро прочетените такива. Информацията за статуса се съхранява в регистъра за статуса (PSW). Програмният брояч (PC) сигнализира дали компютъра е в нужната последователност от инструкции. В зависимост от конкретната машина, броячът съдържа или адреса на инструкцията, изпълняваща се в момента, или този на следващата за изпълнение.

Контролни регистри. Включват регистрите с условен код. Представляват набор от единични битове, показващи специфични условия в компютъра. Стойностите на условните кодове често са определени от изхода на по-важна операция, така че основната им работа е да избират най-подходяща алтернативна последователност. Характерни за тях са флаговете. Предназначени са само за четене (read-only).

# 3.Прекъсвания

Представляват прекъсвания на нормалната последователност на процесора. Това е направено с цел подобрение бързодействието му.

Тъй като повечето входно-изходни устройства са далеч по-бавни от процесора, той преминава към следващата дейност. Когато устройството е приключило работата си, сигнализира на процесора и той изпълнява необходимата дейност.

Работата на процесора може да бъде спряна и заради изпълнението на програма. Това се случва когато се опитвате да делите на нула, да стартирате неразрешена инструкция и други забранени действия.

Специален таймер също предизвиква прекъсване в работата на процесора. Целта на тези прекъсвания е да се позволи на операционната система да извършва определени функции през определен интервал от време.

Неблагоприятни прекъсвания могат да бъдат предизвикани при липса на захранване или грешка в паметта.



Прекъсванията спират нормалната последователност на изпълнение на програмата. Когато процесът на прекъсване е завършен, изпълнението продължава. Не е необходимо програмата да има допълнителен код, за да се пригоди към прекъсванията. Процесорът и операционната система отговарят за прекъсването на програмата и продължаването й след това от същото място.



За приспособяване на прекъсванията се добавя специален етап за тях към цикъла инструкции, както е показано на фигурата отгоре. Когато достигне този етап, процесорът проверява дали е настъпило прекъсване. За целта получава сигнал. Ако няма изчакващи прекъсвания, процесорът продължава към следващата фаза за прихващане и прихваща следващата инструкция. В противен случай процесорът спира изпълнението на текущата програма и изпълнява процедурата по управление на прекъсването. Тази процедура е част от ОС. Тя определя природата на прекъсването и извършва необходимите действия. След приключване на процедурата, процесорът може да продължи изпълнението на програмата от точката на прекъсване.

Съществуват късо и дълго прекъсване. Късото е почти идеален вариант, при който се пести неимоверно много време. Дългото е по-близко до реалната ситуация. При него не се печели много, но въпреки това има подобрение в бързодействието.

Множествените прекъсвания могат да представляват малък проблем. Представляват прекъсване по време на прекъсване. Има два подхода за справяне с тях. Първият начин е да се забранят прекъсвания по време на прекъсване. Вторият начин е да се използва приоритетна схема.

В заключение: процесорът изпълнява повече от една програма едновременно. Последователността на програмите се изпълнява в зависимост от приоритета им и дали те чакат за входно-изходно устройство. След прекъсване, манипулатор не може да се върне към програмата, която е изпълнена по време на прекъсването.

# 4. Йерархия на паметта

Сред основните ограничения в паметта са: размер, скорост и разход. Използват се множество от технологии, за да се намалят тези недостатъци, но се забелязват следните зависимости:

* По-бързо време за достъп, по-големи разходи на бит;
* По-голям капацитет, по-малко разход на бит;
* По-голям капацитет, по-бавна скорост на достъп



Горната фигура илюстрира йерархията в паметта. Отгоре надолу зависимостта е както следва:

* Намаляваща цена на бит;
* Увеличаване на капацитета;
* Увеличаване на времето за достъп;
* Намаляване честотата на достъп до паметта на процесора

По този начин по-малки, по-скъпи и по-бързи памети са добавени към по-големи, по-евтини и по-бавни памети. Ключът към успеха на подобна организция е намаляващата честота на достъп на по-ниските нива.

**Вътрешна (основна) памет**

Вътрешната памет е единствената, която може директно да бъде достъпена от процесора. Той непрекъснато чете инструкциите съхраняващи се там и ги изпълнява.

**Вторична памет**

Външната енергонезависима памет е позната като вторична или спомагателна памет. Използва се за съхранение на програми и информационни файлове.

# 5. Кеш памет

Кеш паметта е невидима за операционната система, но взаимодейства с други хардуерни управления на паметта.



Кеша съдържа копие на част от основната памет, затова процесорът първо се обръща към него. Така се ускоряват процесите. Процесорът генерира адреса на думата, която трябва да се прочете. Ако той вече е в кеш паметта, се доставя на процесора, ако не – блокът, съдържащ думата, се зарежда в кеша и след това се доставя на процесора.



Основната памет се състои от до 2nадресируеми думи като всяка има уникален n-битов адрес. За да се разчита по-лесно тази памет е разделена на сектори с определена дължина от К-думи всеки. Това прави M=2n/K сектора. Кеша се състои от С-слота от по К-думи всеки. Слотовете в кеша се много по-малко от секторите в основната памет. Някои подмножества на блоковете от основната памет се намират в слотовете на кеша. Секторите са повече от слотовете, определен слот не може да бъде постоянно свързан към даден сектор. Затова всеки слот съдържа етикет, който определя към точно кой сектор е свързан в момента. Етикета обикновено е поредица от битове от по-висок ред и се отнася до всички адреси, които започват с тази последователност.

Кеша се характеризира с размер, мапинг функции и разменящи алгоритми. Размера на кеша силно влияе върху проиводителността. Колкото по-голям е, толкова повече бързо достъпни части от паметта ще са възможни. Размерът на блока за обмен също е от важно значение. Твърде големия размер намалява шансовете за преизползване. Мапинг функциите определят коя част от кеша ще бъде използвана. По-голямата сложност на мапинг функциите води до увеличаване на разходите. Когато кеш паметта е пълна, разменящите алгоритми определят коя част от паметта трябва да се замени.

# 6. Еволюция на ОС

Компютърът е набор от средства за преместването, съхраняването и обработката на данни. Операционната система е отговорна за управлението на тези ресурси. Функциите на ОС наподобяват обикновения софтуер – програма или набор от програми,които се изпълняват от процесора.

В своето развитие ОС са подложени на поредица от различни промени, така се обособяват поколенията на ОС

-**нулево поколение**- 40-те години на миналия век (1942г.) се появява първия компютър. Нямало е ОС,потребителите са използвали машинен език.

-**първо поколение**- през 50-те години се появяват първите ОС. Първата ОС е създа[**ден**](http://www.referati.org/evoliuciq-i-klasifikaciq-na-os/44050/ref)а в лабораторията на General Morors за ABM. Основнатацел е да се ускори процеса от една задача към друга. При ОС от 1-поколение е характерно, че те са с-ми за пакетна обработка,т.е. отделните задачи се обединяват в групи, изпълнението на следващата задача започва след изпълнението на предходната. Извършва възтановяване на компютърните ресурси в първоначално състояние. Характери**стик**и:

 -Пакетна обработка

-Възможност за автоматично преминавене към нова програма

-Разработване на средства за автоматично почистване на компютъра при започване на нова задача или при аварийно завършване на някои от заданията

Появяват се концепции за именуване на системните файлове,като се цели да се постигне независимост на хардуера от софтуера.

**второ поколение**-появаява се през 60-те години на 20век. Характерно за тези ОС, е че те поддържат мултипрограмен [**режим**](http://www.referati.org/evoliuciq-i-klasifikaciq-na-os/44050/ref) и мултипрограмна обработка, което води до повишаване на бързодействието на компютърната обработка.



-Появява се независимост на програмите от външните у-ва, като започват да използват логиче**ски**и имена

-Разработват се с-ми с **време**деление ( time sharing) потребителите имат възможност да взаймодействат с програмите чрез терминали. Процесорното време е поделено между потребителите, които имат възможност за едновременен достъп до ресурсите. Увеличава се ефективността на програмите.

-За работа в реално време- компютъра се използва за управление на технологични и производсвени процеси-изисква се надежност, сигурност и време за отговор. Основните цели са ефективно използване на комп.ресурси, безотказна работа и високо бързодействие

**трето поколение**-появява се от средата на 60-те до средата на 70-те. Проявява се в разработването на големи компютри с общо предназначение. Големи машини с ниска ефективност.

Основната ОС е System 360 (ABM). ОС поддържат различни режими на работа. В края на 3-тото поколение се появява ОС Unix- в нея са реализирани някои от основните концепции на съвремените ОС

**четвърто поколение**- развитието на компютърните с-ми се характеризира с :
-широко разпространение на комп.мрежи;

-появата на micro процесори и macro компютри;

-усъвършенстване на диалога човек-компютър;

-появяват се базите от данни и с-ми за управление на БД;

-разширява се **кръг**а от задачи, решавани с компютъра

# 7. Състояния на процесите. Контрол на процесите

Ключовото понятие във всички ОС е понятието процес. Процесът представлява програма в момент на изпълнение. Той се състои от изпълнима програма, данните на програмата и програмния стек, програмния брояч, указателя на стека, другите регистри и останалата информация, необходима за изпълнението на програмата.

**Състояния на процесите:**

Процесът е блокиран, когато той логически не може да продължи, тъй катоочаква някакъв вход. Възможно е един процес да бъде готов за изпълнение, но еспрян, тъй като ОС е решила да отдели CPU за друг процес. Тези две състояния санапълно различни. Диаграмата на един процес може да има следните трисъстояния:

 Изпълняван – CPU е предоставен на процеса;

 Готов – възможно е да бъде изпълнен, временно е спрян, като се позволява да се изпълни друг процес;

 Блокиран – не е възможно процесът да се изпълни, докато не настъпинякакво външно събитие.

Логически първите две състояния са подобни. При третото състояние процесът неможе да се изпълни, дори и CPU да направи нещо..



1. Процесът е блокиран в очакване

 на определено събитие

2. Диспечерът избира друг процес за

 изпълнение

3. Диспечерът избира процеса за

 изпълнение

4. Настъпило е очакваното събитие

Моделът, състоящ се от три състояния може да бъде разширен свъвеждането на състояние на преустановяване. Процесът преустановява самсебе си, но може да бъде преустановен и от друг процес. Преустановеният процес може да продължи изпълнението си, само ако друг процес го възобнови



1. Преходи 1, 2, 3 и 4

 са както на горната

 фигура

5. Преустановяване

6. Възобновяване

Процесът може да бъде преустановен докато е бил в състояние изпълняван, готов или блокиран.

Някои процесиизпълняват програми, изпълняващи команди въведени от потребителя. Други

процеси са част от ОС и управляват задачите (изпълняват заявките за файловиуслуги или управляват на ниско ниво работата на хард диска). Когато се появипрекъсване към диска, ОС взема решение да спре текущия процес и изпълнявапроцеса на диска, който е бил блокиран.

**Контрол на процесите:**

Цялото програмно осигуряване, което се изпълнява в компютъра, е организирано като

последователни процеси. Процесът е изпълняваната програма, включително текущата стойност на програмния брояч (PC), регистрите и стойностите на променливите. Концептуално всеки процес има свой виртуален процесор. Вдействителност обаче реалният процесор се превключва от един процес към друг,осъществявайки псевдо-паралелизма.



Всеки от процесите има собствен алгоритъм (т.е. собствен логически програмен брояч, PC) и всеки се изпълнява независимо от останалите. Вдействителност има един физически PC, така че когато процесът се изпълнява, неговият логически PC се зарежда в реалния РС. След време физическият РС отново се съхранява в логическия РС на съответния процес. Процесите не се програмират с отчитане на предположенията за времето, за което те ще бъдат изпълнявани. Това се осъществява от ОС.

Съществуват четири основни събития, които предизвикват създаването напроцес:

 Инициализиране на системата

Изпълняване на системно извикване за създаване на процес

Заявка от потребителя за създаване на нов процес

Иницииране на пакетно задание

Причините за прекратяване на процес са следните:

Нормален изход (доброволно)

Погрешен изход (доброволно)

Фатална грешка (принудително)

Прекратен от друг процес (принудително)

# 8. Процеси и Нишки

* Процесите имат две характеристики:
	+ Собствени ресурси – всеки процес включва собствено адресно пространство за съхранение на данните му
	+ Планиране/изпълнение – процесът има път на изпълнение, който може да бъде съгласуван с други процеси.
* Тези две характеристики се третират независимо от операционната система.
* Единицата за изпълнение се отнася като нишка (thread) или лек процес.
* Единицата за собственост на ресурсите се отнася като процес или задача (*task)*

Процесите се използват, за да се групират ресурсите, а нишките са същности (обекти, единици), планирани за изпълнение от процесора.



На фиг. А) се виждат 3 традиционни процеса. Всеки процес има свое адресно пространство и една нишка на управление. На фиг. Б) има един един процес с три нишки на управление. Въпреки че в двата случая са налице три нишки, на фиг. А)всяка една от тях оперира в различно адресно пространство, докато на фиг Б) тритенишки поделят едно и също адресно пространство.

Чрез въвеждане на многонишково изпълнение се постига поделяне наресурсите, така че те да работят по-ефективно при изпълнение на някаквазадача.

Подобно на процеса, една нишка може да бъде в едно от следните състояния:

изпълнявана, блокирана, готова и прекратена. Преходите между отделнитесъстояния са същите, както преходите между състоянията на процеса.

Еднонишков подход:

* MS-DOS поддържа един потребителски процес и една нишка.
* UNIX поддържа множество потребителски процеси, но само една нишка в процес.

Многонишковост:Способността на ОС да поддържа множество конкурентни пътища на изпълнение в рамките на един процес.Java run-time environment (JRE) е един процес с много нишки. Множество процеси и нишки има в Windows, Solaris и много съвременни версии на UNIX.

Предимства на нишките:

- По-малко време е нужно за създаването на нова нишка, отколкото на нов процес

- По-малко време е необходимо за прекратяване на нишки отколкото на процес

- Превключването между две нишки отнема по-малко време отколко превключването между два процеса

- Нишките могат да комуникират една с друга без да извикват ядрото.

Нишки/Процеси



Основните причини за използване на нишки в модерните системи са:

Главната причина е, че в много приложения множество дейности сеизпълняват едновременно.

Втората причина е, че тъй като нишките нямат прикачени към тях ресурси,те по-лесно се създават и разрушават от процесите.

Третата причина е свързана с подобряване на производителността.

Четвъртата причина е, че нишките са полезни в многопроцесорни системи, където е възможен реален паралелизъм.

Пример – система за обработка на текст, Word.



# 9. SMP (SymmetricMultiprocessing)

Ядрото може да се изпълнява на всеки процесор, което позволява части от него да се изпълняват паралелно.

Обикновено всеки процесор си планира изпълнението на наличните процеси или нишки.

*СтандартнаSMP Организация*



Фактори при проектирането на мултипроцесорна ОС:

* Ключови фактори при проектиране:
	+ Еновременни и конкуриращи се процеси и нишки
	+ Планиране на изпълнението на процесите и нишките
	+ Синхронизация
	+ Управление на паметта
	+ Надеждност и устойчивост на откази

Поддръжка на SMP в Windows:

* Нишките могат да се изпълнят на всеки от процесорите
	+ Но приложението може да ограничи този ефект
* Soft Affinity
	+ Разпределителят (dispatcher)опитва да присвои готова нишка на съшият процесор от последното изпълнение.
	+ Това подпомага преизползването на данни, които са кеширани в паметта на процесора от последното изпълнение на нишката.
* Hard Affinity
	+ Приложението изисква изпълнението на нишките да бъде на определен процесор.

# 10. Микроядра.

Микроядрото е малко ядро на ОС, което осигурява основата за модулни разширения.

**Микроядрото** е основата на минимална по размер [компютърна](http://bg.wikipedia.org/w/index.php?title=Компютърна&action=edit&redlink=1) [операционна система](http://bg.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), която в най-чистата си форма не предлага никакви

операционни сервизи. В основата си, тя предлага само механизмите за [имплементиране](http://bg.wikipedia.org/w/index.php?title=Имплементиране&action=edit&redlink=1) на такива сервизи.
\*\* Архитектура на микроядро



\* Връзки между процесите - чрез съобщения, на принципа на порт-а

Комуникацията между процеси или нишки в ОС с микроядро се осяществява чрез съобщения (messages)

Съобщението включва:

* Заглавна част (header), който идентифицира изпращащият и получаващият процес
* Тяло (body), което съдържа данни, указател към част от данните, или контролна информация за процеса.
\* Ползи от микроядрата:
-- еднообразен интерфейс на заявките
-- продължаемост на ОС с нови елементи (услуги)
-- гъвкавост на ОС - лесно подмяна на елементи
-- преносимост - малък код за подмяна
-- надежност - малък код, надежно тестване, малък брой Application Programming Interface (API)
\* Организация на I/O и прекъсванията - микроядрото разпознава прекъсванията и ги дава за обработка на потребителски процес. В рамките на

 микроядрото е възможно да се обработят хардуерни прекъсвания като съобщения и да се добавят портовете за вход/изход в адресните пространства.

 На определен процес от потребителското ниво се възлага реагирането на прекъсването и ядрото поддържа обработката.

Предимства на ОС с микроядро:

* Унифицирани интерфейси относно запитвания, отправени от един процес.
* Разширяемост
* Гъвкавост
* Преносимост
* Надеждност
* Поддръжка на разпределени среди
* Обектно ориентирани ОС

Разликите при различните ОС относно поддръжката на процеси включват:

Именуване на процеси

Осигуряване на нишките

Представяне на процесите

Защитаване на ресурсите

Механизми за комуникация между процесите и синхронизация

* Връзки между процесите

11.1. Семафори и монитори

През 1965 година Дейкстра предлага решение на проблемите, свързани с конкурентните процеси. Той разглежда ОС като съвкупност от кооперативни последователни процеси и мотивира разработването на ефективни и сигурни механизми за поддържане на кооперативни процеси.

Основния принцип е следния: два или повече процеса могат да се кооперират чрез средствата на прости сигнали, чрез които процесът може да бъде спрян на определено място, докато приеме контролен сигнал.

За сигнализирането се използват специални променливи, наречени семафори. За да се предаде (изпрати) сигнал чрез семафора s, процесът изпълнява примитива signal(s) /V(s)/. За да се приеме сигнал чрез семафора s, процесът изпълнява примитива wait(s) / P(s) /. Ако съответния сигнал все още не е бил предаден, процесът се преустановява, докато не завърши предаването.

За да се постигне желания ефект, семафорът се разглежда като променлива, която има integer стойност, върху която се извършват три операции.

Примитивите за семафори могат да бъдат дефинирани по следния начин:

struct semaphore

{

 int count;

 queueType queue;

};

void wait(semaphore s)

{

 s.count--;

 if(s.count<0)

 {

 изпраща процеса в s.queue

 процесът се блокира

 }

}

void signal(semaphore s)

{

 s.count++;

 if(s.count<=0)

 {

 изтрива процеса от s.queue

 добавяне процеса към списъка на готовите процеси

 }

}

Приема се, че wait и signal са атомарни, т.е. те не могат да бъдат прекъсвани и всяка п.п. се третира като неделима стъпка.

Двоичните семафори са по-ограничена версия на семафорите. Те могат да заемат стойност 0 или 1. По-лесно се реализират двоичните семафори и имат същите възможности като общите семафори.

struct binSemaphore

{

 enum (zero, one) value;

 queueType queue;

};

void waitBin(binSemaphore s)

{

 if(s.value == 1)

 s.value = 0;

 else {

 изпраща процеса в s.queue

 процесът се блокира

 }

}

void signalBin(binSemaphore s)

{

 if(s.queue.isEmpty())

 s.value = 1;

 else {

 изтрива процеса от s.queue

 добавяне процеса към списъка на готовите процеси

 }

}

Семафорите са примитиви, осигуряващи взаимно изключване. Те координират процесите.

Мониторът е езикова конструкция, която осигурява еквивалентна функционалност като семафорите. Те по-лесно се управляват. Понятието е въведено за първи път от Хоар.

Мониторът е програмен модул, състоящ се от една или повече п.п., инициализационна последователност и локални данни. Основните характеристики на мониторите са следните:

- Локалните променливи са достъпни само от процедурите на монитора и не са достъп от никоя външна процедура;

- Процесът влиза в монитора чрез извикване на една неговите процедури;

- Само един процес може да се изпълни в монитора в даден момент от време; всеки друг процес, който извика монитора, се преустановява, изчаквайки монитора да стане достъпен.

За да бъде полезен за едновременната (конкурентна) обработка, мониторът трябва да включва средства за синхронизация. Мониторът поддържа синхронизация чрез използване на условни (логически) променливи, които се съдържат в самия монитор и са достъпни само в него. Използват се две функции за опериране с условни

променливи:

cwait(c) – преустановява изпълнението на извикващия процес при условие c; мониторът сега е достъпен за използване от друг процес.

csignal(c) – получава обратно изпълнението на някакъв, преустновен след cwait процес при същото условие. Ако има няколко такива процеса, избира се един от тях. Ако няма такива процеси не се прави нищо.

Трябва да се има предвид, че операциите на монитора wait/signal се различават от тези на семафора.

# 11. Архитектура на Windows ядро

Процесите и услугите (services), които са осигурени от ядрото на Windows, са сравнително прости и с общо предназначение

- Реализирани са като обекти

- Изпълним процес може да съдържа една или повече нишки

- Процесите и нишките са обекти и имат вградени възможности за синхронизация



1.Обща характеристика- преносима ОС, а тази преносимост се осъществява чрез няколко подхода:

-тя е в съответствие със слоестия модел, елементите зависими от хардуера са изолирани в модули: хардуерен абстр. слой, чдро на ОС.

-написани на език от високо ниво С, графична част- С++ и останалото на Assembler.

Други харектеристики: мултипроцесорна ОС. Има 2 варианта за реализация на мултипроцесорна ОС:

-асиметрична мултипроцесорност-един от процесорите е главен и към него се изпраща кода на главната система;

-симетрична- кода на ОС и потребителските приложения може да бъде разпределен за използване на всеки наличен своноден процесор;

В Windows NT е симетрична мултипроцесорност, тя е мащабируема ОС- може да управлява ресурси с различен мащаб, може да се прилага в различни области

2.Основни компоненти на архитектурата

Подсистеми на обкражението: OS/2, WIN32 и POSIX. Целта им е да се извърши преносимост на приложения предназначени за други с-ми.

1. POSIX- пренасяне на приложение за Linux,Unix i Windows. Тази подсистема се използва винаги. За да преодолее ограниченията на Posix, Windows използва системна библиотека NTDLL.DLL- специална системна библиотека съдържаща 2 групи функции:

-първата група ф-ии е за извикване на системни услуги от вътрешната част на Windows

-втора група-собствени ф-ии-използва се за системните процесори и потребителските приложения

Вътрешната част на Windows- горен слой + с-ма за управление на конфигурациите;

Допълнителни модули- мултипроцесор на обектите

1. стандартни библиотеки- обработка на низове, аритметични задачи и др;
2. Ядро- реализира най-важните операции в Windows, защото определя как се използва процесора или процесорите
3. ХАС-НАL- основен компонент за реализиране на преносимостта на ОС
4. Драйвери на устрйства:

-на хардуерни у-ва- Device Managment

-на файлова с-ма, които приемат заявки за В/И операции и за файлове;

-филтриращи драйвери-поддържат дублиране на дискове, пренасяне на дялове;

-мрежови и сърварни разпределители- драйвери, които приемат и обработват В/И заявки;

* Сист.процеси- част от тях се изпълнява в потребителски режим, част в системен режим:

-Idle- този се изпълнява в системен режим, съдържа по една нишка за всеки процесор и отчита времето за изчисляване на процеса

-System- използва се в сиситемен режим и съдържа системни нишки 32 SMSS.EXE;

-мениджър на системите -използва се в потребителски режим и реализира функции по стартиране на подсистемите на Windows, стартиране на процеса Log on и при прекратяване на някои задачи по спиране на ОС

-Log on –управлява започването и приключването на работата на дададен потребител . използва се в потребителски режим.

-контролер на услугите- управлява взаимодействието с услугите използвани в потребителски режим.

Поддръжка на SMP в Windows: Нишките могат да се изпълнят на всеки от процесорите

-Но приложението може да ограничи този ефект

---Soft Affinity

Разпределителят (dispatcher)опитва да присвои готова нишка на съшият процесор от последното изпълнение.

-Това подпомага преизползването на данни, които са кеширани в паметта на процесора от последното изпълнение на нишката.

----Hard Affinity

-Приложението изисква изпълнението на нишките да бъде на определен процесор.

12. Архитектура на Linux ядро

Всеки процес е представен чрез task\_struct структура. Вектора task е масив от указатели към всеки task\_struct в системата. Максималния брой процеси се ограничава от размера на вектора task, т.е 512. Когато се създаде нов процес се заделя нов task\_struct в паметта и се добавя към task вектора. Полетата на task\_struct са:

Положение – state ; той има следните стойности – работещ; чакащ; спрян; зомби; Зомби е спрян процес, който по някаква причина все още има task\_struct в task вектора.

Информация за разпределението – scheduling information – подредба на процесите според това кой заслужава да бъде пуснат най-напред.

Идентификатори – identifiers – това е число.

Комуникация между процесите – inter process communication

Други

http://www.linux-tutorial.info/modules.php?name=MContent&pageid=326

12. Deadlock

Взаимната блокировка може да бъде дефинирана като постоянно блокирано множество от процеси, които или се съревновават за системни ресурси, или комуникират помежду си. За разлика от другите проблеми в управлението на конкурентни процеси, в общия случай тука няма ефектно решение. Взаимната блокировка е конфликт между два или повече процеса за необходими ресурси. Най-общ пример е взаимната блокировка на нерегулирано кръстовище. Това е ситуация, при която четири автомобила пристигат от 4-те посоки едновременно на кръстовището.

**Условия за възникване на deadlock.**

За да възникне deadlock, трябва да са налице следните три условия:

-Взаимно изключване – в даден момент от време само един процес може да използва ресурса;

-Задържане и изчакване – процес може да задържи заделения ресурс, докато изчаква присвояването на други ресурси;

-Няма преразпределяне на ресурсите – не може ресурс насила да бъде отнет процес, който го държи.

Взаимната блокировка може да възникне при горните три условия, но те не винаги са достатъчни. Изисква се и 4-то условие:

-Кръгово (циклично) очакване - съществува затворена верига от процеси, така че всеки процес задържа поне един ресурс на следващия процес във веригата.

**Предпазване от мъртва хватка.**

Стратегията за предпазване от взаимна блокировка изключва вероятността за възникване на взаимна блокировка, макар и с цената на загуби на допълнителни ресурси. Методите за предпазване от взаимна блокировка могат да се разделят на два класа:

-Косвен метод – предпазва от появата на първите три условия;

-Пряк метод – да се предпази от появата на кръгово очакване.

**Избягване от мъртва хватка.**

Стратегията за избягване на взаимна блокировка гарантира, че макар и принципно да настъпи мъртва хватка, тя не възниква за конкретна група процеси и заявки, изпълнени в даден момент.

Ще се предложат два подхода за избягване на взаимна блокировка:

-Да не се стартира процес, ако неговите нужди водят до мъртва хватка;

-Да не се отпуска следваща заявка за ресурс към процес, ако това разпределение може да доведе до мъртва хватка.

**Откриване на взаимна блокировка.**

Стратегиите за предпазване са много консервативни. Те решават проблема чрез ограничаване на достъпа до ресурсите и чрез налагане на ограничения върху процесите. Стратегиите за откриване не ограничават достъпа до ресурсите и не ограничават действията на процесите.

Стратегията за откриване на мъртва хватка и след това на възстановяване на системата се основават на извода, че взаимните блокировки възникват рядко и че е по-скъпо предпазването и избягването от мъртва хватка, отколкото откриването и възстановяването.

**Възстановяване от взаимна блокировка.**

След като мъртвата хватка веднъж е била открита, необходима е някаква стратегия за възстановяване. Възможни са следните подходи:

-Спират се всички взаимно блокирани процеси.

-Връщане назад на всеки процес, изпаднал в мъртвата хватка, до някаква предишна точка за проверка и всички процеси се рестартират. Рискът при този подход е, че първоначалната хватка може да се повтори отново.

-Последователно спиране на блокираните процеси, докато вече няма мъртва хватка. След всяко спиране алгоритъмът за откриване на взаимна блокировка трябва да бъде извикан отново, за да се разбере дали все още съществуват взаимно блокирани процеси.

-Последователно присвояване на ресурсите, докато повече не съществува взаимна блокировка.

13.1. Алгоритъм на „Вечерящи философи”

Пет философа стоят на кръгла маса. Всеки от тях прекарва живота си в последователност от мислене и ядене. В центъра на масата има голяма купа със спагети. Всеки философ се нуждае от две вилици, за да изяде порция спагети.

По една вилица е сложена между всеки двойка философи. Те са се съгласи да използват само вилици, намиращи се непосредствено вляво и вдясно.

Всяка вилица е споделен ресурс с възможни действия взимам и оставям. Когато е гладен един философ той първо трябва да вземе лявата и дясната, преди да започне да яде.

**Възникване на проблеми:**

* Никои двама философи не могат да използват една и съща вилица по едно и също време;
* Нито един философ не трябва да умира от глад (избягване на мъртва хватка и недостиг на ресурси).

**Жизнен цикъл на философа:**

* мислещ философ – няма нужда от вилици;
* мислещия философ може да огладнее;
* гладния философ се опита да вземе вилицата от лява и става гладен философ с вилица в лявата ръка;
* една вилица не е достатъчна, за да почне да яде философа;
* вилицата вдясно може да се вземе, само ако е свободна;
* след ядене може да се направи само едно: да остави двете вилици и да се върне към мисленето.

**Лаком философ**

Лакомия философ не оставя вилицата, докато не се наяде и започне да мисли. Това може да доведе до безизходица.

**Благородния философ**

Благородния философ не настоява да живее успешен живот. След като вземе лявата вилица, но открие, че дясната е заета, той оставя лявата вилица и се връща към мисленето. Така че ако всички философи са благородни, то няма да се стигне до безизходица, но има опасност някой философ да умре от глад.

**Решения**

* Да се позволи най-много 4 философа да седят на една маса;
* Да се позволи да се вземат двете вилици, само ако и двете са свободни;
* Четен философ вдига първо лявата, след това дясната, а нечетен – обратното.

13.2 Алгоритъм „Читатели/Писатели”

Ресурс може да се чете от много процеси. Не е възможно писане и четене на общ ресурс в един и същи момент от време.

Решение на проблема четене-писане:

* Процес контролер
* Изпращат се заявки към него
* Съобщения за освобождаване на ресурс
* Контролерът позволява първо писане
* Чака съобщение за край и предоставя за четене
* Ако има няколко „четящи процеса” се изчакват до завършването им и тогава се предоставя ресурса за писане

13.3. Алгоритъм на Банкера

Банкера разполага с капитал. Клиентите заявяват заеми. Клиентите получават заеми, по-малки от заявените. Банкера следи за наличния ресурс.

Недостатъци:

* Валиден е за малък брой ресурси и процеси;
* Клиентите трябва да върнат ресурсите в крайно време – при ОС това не винаги е възможно.

***Въпрос 14. Основни функции на основната памет(оперативна)***

Основната (оператибната) памет е важен ресурс на компютърната система – за съществиването на който и да е процес, наред с процесора, е необходима и памет, в която да бъдат записани отговарящите му програма и данни.С основна памет процесорът работи при изпълнението на програмите.В мултипрограмните пакети ОС и в ОС с времеделене едновременно съществуват много процеси – системни и потребителски и на всеки трябва да се разпредели област от паметта.Заедно с това трябва да бъде осугурена защита на паметта.Средствата за защита са необходими, както за управление на достъпа до собствените му комади и данни на процеса, така и за предпазване от достъп на процеса до информация свързана с други процеси.

Оперативната памет е последователност от номерирани клетки с еднаква големина, съдържащи информация. Тази информация може да е инструкция, която процесорът трябва да изпълни, или данни, които трябва да се обработят. Всяка клетка има адрес, чрез който програмата се обръща към необходимото място в паметта. Оперативната памет също така бива наричана памет от 1-во ниво, според отдалечеността от процесора

Има два типа ОП: **постоянна (ROM)** и **памет с произволен достъп (RAM)**. Паметта ROM съхранява информация, дори когато компютърът е изключен. Най-важните програми на микрокомпютъра са записани в паметта ROM, включително и програмата, която при включване на компютъра проверява дали всичките му компоненти работят правилно. За разлика от паметта ROM, RAM съхранява информация, само когато компютърът е включен. Нейният обем се измерва в килобайтове и мегабайтове

***Въпрос 15. Разпределяне. Подялба на ресурси***

В една мултипрограмна система, наличната ОП основно се поделя между процесите. За програмиста не е възможно предварително да знае, кои други програми ще бъдат резидентни в ОП по същото време когато се изпълнява неговата програма. Необходимо би било да има възможност процесите да се натоварят (swap-in) или да се разтоварят (swap-out) в/от ОП, за да се увеличи използваемостта на ПЦ, чрез осигуряване на голям брой готови за изпълнение процеси. След като програмата веднъж е била разтоварена върху диск, голямо ограничение ще бъде, ако се изиска след повторното й натоварване в ОП, тя да заеме същото място, където е била преди това. Това налага преразпределяне на процеса в друга област от ОП

Не е възможно предварително да се знае къде ще бъде разположена програмата. Трябва да се даде възможност програмата да бъде премествана в ОП,в резултат на swap-a.Това налага някои особености свързани с адресирането.

Тъй като ОС управлява ОП и тя отговорна за доставянето на процеса в ОП, тези адреси лесно могат да бъдат получени. Процесорът трябва да се справи с обръщенията вътре в програмата.Инструкциите за преход съдържат адреса на следващата за изпълнение инструкция.Инструкциите за обръщение към данните съдържат адреса на байта или думата,към която се извършва обръщение. По някакъв начин процесорът и ОС трябва да преобразуват обръщенията към паметта от кода на програмата в действителни физически адреси, отразявайки текущото разположение на програмата в ОП.

Всеки процес трябва да бъде защитен срещу непозволена аварийна или умишлена намеса от други процеси. Така че програмите от други процеси не би трябвало да могат да се позовават за четене или запис към разпределението на паметта на процеса, без позволение. изпълнение, за да се гарантира, че те сочат към област от ОП, разпределена за този процес. Механизмите, които осигуряват преразпределяне, също поддържат и изискването за защита.

 Програмата на един процес не може да направи преход към инструкция в друг процес. Без специални позволения, програма от един процес не може да има достъп до областта на данните на друг процес. Процесорът трябва да отхвърли такива инструкции в процеса на изпълнение.

 Изискването за защита на ОП трябва да бъде постигнато по-скоро от процесора (на апаратно ниво), отколкото от ОС (на програмно ниво). Това е така защото ОС не може да предвиди всички позовавания към ОП, които програмата ще направи.

Всеки защитен механизъм трябва да има гъвкавостта да позволи на няколко процеса да имат достъп до една и съща област от ОП. Процеси, които са кооперирани върху някаква задача, могат да имат нужда да поделят достъпа до една и съща структура данни. Механизмите, които поддържат преразпределяне, поддържат и възможностите за поделяне.



а) Опашка от по един процес за дял б) Една опашка за всички процеси

Фигура 7.3. Разпределение на паметта при фиксирано разделяне на дялове

***Въпрос 16. Логическа организация на паметта***

ОП в компютърните системи е организирана като линейно, едномерно адресно пространство, състоящо се от последователни байтове или думи.

Вторичната памет, на физическо ниво, е организирана по същия начин. Тази организация отразява апаратното осигуряване. Тя не съответства на начина, по който са организирани програмите. Повечето от програмите са организирани в модули, някои от които не могат да бъдат променяни, а други съдържат данни, които могат да бъдат модифицирани. Ако ОС и апаратното осигуряване могат ефективно да се справят с потребителските програми и данните под формата на модули от някакъв тип, то могат да бъдат осъществени следните предимства:

Модулите могат да бъдат написани и компилирани независимо, с всички разрешени обръщения от един модул към друг по време на изпълнение; На различните модули могат да бъдат дадени различни степени на защита, с незначителни допълнителни загуби; Възможно е да се въведат механизми, чрез които модулите могат да бъдат поделяни между процесите. Предимство на осигуряването на поделяне на ниво модули, е че това съответства на начина, по който потребителят разглежда проблема.

Тези изисквания се постигат чрез сегментиране, което е една от техниките зауправление на паметта.

***Въпрос 17. Физическа организация на паметта***

Паметта на компютъра е организирана на две нива - оперативна памет (основна) и вторична памет. ОП осигурява бърз достъп на сравнително висока цена. Тя не е постоянна памет, не осигурява дълготрайно съхраняване на данните. Вторичната памет е по-бавна и по-евтина от ОП. Това е постоянна памет, с голям капацитет. Тя може да осигури продължително съхраняване на програми и данни, докато ОП съхранява временно текущо използваните програми и данни. В тази структура на две нива, организацията на потока от информация между ОП и вторичната памет е от съществен интерес. Отговорността за този поток може да бъде предоставена на отделния програмист, но това не е желателно порадиследните две причини:

Наличната ОП, предоставена на програмата и данните, може да се окаже недостатъчна. В този случай може да се използва овърлейна (overlay,препокриване) структура, в която програмата и данните са организирани по такъв начин, че различни модули могат да бъдат разположени в една и съща област от паметта, като отговорността за превключване на модулите се носи от главната програма. Овърлейното програмиране води до значителни загуби на програмистко време. В мултипрограмните среди, по време на кодиране на програмата програмистът не знае колко ще бъде наличното пространство в паметта и къде ще бъде разположено то.



Данните, съхранявани в памет от по-високо ниво, обикновено се съхраняват и в паметите от по долните нива. Ако процесърът не намери нужната информация на  „п-м” ниво, то той я търси на следващите по-долни нива.

Когато нужните данни бъдът намерени, те се пренасят на по-горните и по-бързи нива.

***Въпрос 18. Paging, Segmentation***

За да се преодолее вътрешната и външната фрагментация се използва странична организация на паметта. ОП се разделя на блокове с еднаква, относително малка дължина, наречени странични кадри (физически страници). Адресното (логическото) пространство на процеса също се разделя на блокове със същия размер, наречени страници. Страниците на процеса се присвояват на кадрите от ОП. В тази секция ще бъде показано, че загубите на пространство в ОП за всеки процес се дължат на вътрешно фрагментиране, състоящо се в една частица от последната страница на процеса. В даден момент от време някои от кадрите в паметта се използват, а други са свободни. ОС поддържа списък на свободните кадри. Процес А, който е съхранен върху диска се състои от 4 страници. Когато дойде време за неговото зареждане ОС намира 4 свободни кадъра и зарежда в тях четирите страници на процеса А (В). Последователно се зареждат процес В, състоящ се от 3 страници, и процес С, който се състои от 4 страници. След това процес В се преустановява и се разтоварва (swap-out) от ОП. По-късно всички процеси в ОП се блокират и ОС трябва да зареди нов процес D, който се състои от 5 страници. Предполага се, че няма достатъчно непрекъснати, неизползвани кадри за зареждане на процеса. За да бъде разделянето по страници по-удобно, размерът на страницата и оттук размерът на кадъра трябва да бъдат кратни на степените на 2. Лесно може да се покаже, че относителният адрес, който се дефинира като отместване спрямо началото на програмата, и логическият адрес, изчисляван като номер на страница и отместване, са еквивалентни.

***Paging***



Програмата и свързаните с нея данни се разделят на определен брой сегменти. Не се изисква всички сегменти във всички програми да бъдат с еднаква дължина, въпреки че има ограничение за максималната дължина на сегмента. Логическият адрес се състои от две части - <номер на сегмент> и <отместване>. Докато разделянето на страници е невидимо за програмиста, сегментирането обикновено е видимо за програмиста и е свързано с организиране кода на програмите и данните.Програмистът или компилаторът ще присвои кода и данните на различни сегменти. Друго следствие от различния размер на сегментите е, че няма проста връзка между логическите адреси и физическите адреси. Аналогично на разделянето на страници, при сегментната организация се поддържа таблица на сегментите за всеки процес и списък на свободните блокове от ОП. Всеки елемент в таблицата установява съответствието между сегмента и неговия начален адрес в ОП. Поддържа се информация за дължината на сегмента. По този начин се избягва обръщение към невалидни адреси. Когато процесът изпадне в състояние изпълняван, адресът на таблицата на сегментите му се зарежда в специален регистър, използван от апаратното осигуряване за управление на паметта.

***Segmentation***



# 19. Виртуална памет

**Виртуалната памет**е техника която позволява изпълнение на програми които не са поместени изцяло в основната памет. Главното и предимство е, че потребителските програми могат да бъдат по големи от основната памет. Виртуалната памет се реализира чрез средствата на странична или сегментна или странично/сегментна организация, и чрез механизма за размяна.

**Същност на виртуалната памет**

Виртуалната памет е отделяне на потребителската логическа памет от основната памет. Виртуалната памет съществува паралелно със основната памет като м/у тях се осъществява динамично съответствие ч/з механизмите на страничната организация.Виртуалната памет е адресно пространство разделено на страници разположени в/у външно запомнящо устройство. Работата и наподобява системите със странична организация и размяна. На всяко избрано за изпълнение задание се заделя област от виртуалната памет, в която то се разполага като някои от неговите страници се прехвърлят в основната памет като по време на изпълнение страници постоянно се въвеждат и извеждат. Системен механизъм наречен управление на вторичната памет, поддържа таблица с информация за съответствието м/у виртуалните адреси и физ. разположение на данните в/у диска.Съответствието м/у виртуалната и основната памет се управлява от страничен супервайзор, който динамично организира записа на необходимите страници в основната памет. Таблицата на страниците е допълнена с бит за присъствие показващ дали съответната стр. се намира в основната памет.

**Virtual Memory Addressing**



Защита на информацията и сигурност

* Проблемът включва контролиране на достъпа до компютърните системи и информацията, съхранявана в тях .
* Основни посоки:
	+ НАличност
	+ Секретност
	+ Цялостност на данните
	+ Автентичност

Key Elements of an Operating System



Различни архитектурни подходи

* Включват:
	+ Архитектура с микроядра
	+ Multithreading
	+ Разпределени ОС

Обектно ориентиран дизайн

# 20. Видове Scheduling

1**.Планиране на Високо (горно) ниво - HLP**

Oпределя кои програми да бъдат допуснати за обработка до системата. Управлява се степента на мулти-програмиране. свързано предимно с пакетния режим на работа. То се изпълнява от системен механизъм, който избира задание от множество постъпили в системата задания, създава процес за неговото изпълнение и го записва в опашката за обслужване от процесора.

 **2.Планиране на Ниско(делигирано) ниво - LLP**

Планиране на ниско ниво: Средствата на това ниво определят кой от готовите процеси да бъде избран за изпълнение и за какъв интервал от време процесът да разполага с процесора, след което предоставят процесора на избрания процес. За разлика от планирането на високо ниво, което се изпълнява еднократно от гледна точка на всяко задание, планирането на ниско ниво се изпълнява постоянно- един процес може да получи процесора моногократно. Новото разпределение на процесора е принципно възможно, когато: изпълнявания процес доброволно се отказва от продължаването на своята работа или се блокира от системен механизъм. За да не се допусне монополизиране на процесора от потребителски програми, в някои ОС има механизъм за отнемане на процесора. ОС установява часовник или таймер, за да се генерира прекъсване в определен момент, след което се избира нов процес. За управлението на зависещите от времето задачи е удобно да се организира списък на процесите, блокирани пред ресурса време и подредени по времето за събуждане.

**3.Планиране на междинно ниво**

Планирането на средно ниво е част от функцията за поддържане на ***swap***. Решението се основава на необходимостта да се управлява степента на мулти- програмиране. Това е решение за добавяне на процес към тези, които поне частично са в основната памет и следователно са налични за изпълнение.

**Връзката м/у функциите на планиране и диграмата на състояние на процеса е следната.**



# 21. Алгоритми за Scheduling



**C-SCAN**

* Ограничава сканирането в 1 посока
* След като е посетена последната писта от заявката, главата се връща в началото**.**



SCAN

* Главата се движи само в една посока докато не удовлетвори всички заявки и после се връща обратно



**First-in, first-out (FIFO) -** Това е най-простата дисциплина на обслуйване.Когато един процес стане готов той се включва в опашката на готовите процеси. Тя е по-благоприятна за процеси, които повече са обвързани с процесора от колкото тези , които са обвързани с В/И.FIFO е дисциплина без превключване. Тя не се използва самостоятелно в едно-процесорните системи. По-ефективна е при комбинирането й с преоритетна схема на обслужване.

* Последователни процеси
* Random scheduling



**Last-in, first-out**

* Добре е за системи за обработка на транзакции
* Устройството се дава на най-новия потребител,

**Shortest Service Time First**

* Избира се заявка, която изисква най-малкото движение на главата на диск от сегашната си позиция
* Избира се мин. Време за търсене



**Comparison of Disk Scheduling Algorithms**

# 22. Мултипроцесорен Scheduling

**Класификации на мултипроцесорните системи**

* Слабо интегрирани (Loosely coupled) процесори,
	+ Всеки има собствена памет и входно-изходни канали
* Функционално специализирани процесори
	+ Контролирани от главен (master) процесор
	+ Например входно-изходен процесор
* Tясно интегрирани (Tightly coupled) мултипроцесори
	+ Процесорите си поделят основната памет
	+ Контролират се от операционната система

**Гранулация (Granularity)**

* Или честота на синхронизация между процесите в системата.
* Пет категории, с различна гранулация:
	+ Независим паралелизъм (Independent Parallelism)
	+ Груб паралелизъм (Coarse Parallelism)
	+ Много грубо-гранулиран паралелизъм (Very Coarse-Grained Parallelism)
	+ Средно гранулиран паралелизъм (Medium-Grained Parallelism)
	+ Фин паралелизъм (Fine-Grained Parallelism)

**Независим паралелизъм**

* Между процесите няма изрична синхронизация
* Отделно приложение или задача
* Пример е системата с времеделене

**Груб и много грубо-гранулиран паралелизъм**

* Синхронизацията между процесите е много слаба
* Добър вариант за едновременно изпълняващи се процеси при мултипрограмиран еднопроцесор
	+ С малки промени може да се поддържа и в многопроцесорен вариант

**Средно гранулиран паралелизъм**

* Едно-единствено приложение е колекция от нишки
* Обикновено нишките често си взаимодействат, като това влияе върху производителността на цялото приложение

**Фин паралелизъм**

* Силно паралелни приложения
* Специализирана и фрагментирана област

**Синхронизиране на гранулирането и процесите **

**Планиране на дизайна**

* При мултипроцесорите планирането включва три взаимносвързани теми:
	+ Разпределяне на процесите по процесори
	+ Използване на мултипрограмиране за отделните процесори
	+ Действително разпределяне на процес
* Възприетият подход зависи от степента на гранулиране на приложенията и броя на наличните процеси

**Разпределяне на процесите по процесори**

* Приемайки, че всички процесори са еднакви, най-лесно е да ги третираме като пул от ресурси и да разпределяме процесите по процесори както е необходимо.
	+ Но дали е по-добре разпределението да е статично или динамично?
* Динамично разпределение
	+ Нишките се преместват от опашка за единия процесор на опашка за друг процесор;

**Статично разпределение**

* Даден процес се разпределя за постоянно на даден процесор
	+ Отделна краткосрочна опашка за всеки процесор
	+ По-малко допълнително натоварване
	+ Позволява ‘групово’ планиране (виж понатам)
* Но може да има бездействащ процесор, докато за другите има дълга опашка
	+ Решение: използвайте обща опашка

**Разпределяне на процесите по процесори**

* И динамичните, и статичните методи изискват някакъв начин за разпределяне на процесите по процесори
* Два метода:
	+ Господар/Подчинен (Master/Slave)
	+ Равнопоставени (Peer)
* Разбира се, има множество комбинации от двата метода.

**Планиране на процеси**

* Обикновено процесите не се разпределят по процесори
* Една опашка се използва за всички процеси
* Или пък няколко опашки се използват за приоритети
	+ Всички опашки захранват общ пул от процесори

**Планиране на нишки**

* Нишките се изпълняват отделно от останалата част от процеса
* Едно приложение може да бъде набор от нишки, които се кооперират и изпълняват едновременно в едно и също адресно пространство
* При многопроцесорни системи е възможно драстично повишаване на производителността
	+ В сравнение с работата при еднопроцесорните системи

**Подходи при планирането на нишки**

* Има много предложения, но четири от тях се открояват:
	+ Поделяне на натоварването (Load Sharing)
	+ Групово планиране (Gang Scheduling)
	+ Разпределяне на самостоятелен процесор (Dedicated processor assignment)
	+ Динамично планиране (Dynamic scheduling)

**Групово планиране**

* Множество от свързани нишки е планирано едновременно да се изпълни на множество от процесори

Паралелното изпълнение на тясно свързани процеси може да намали допълнителното натоварване, например превключването на процеси и блокиране на синхронизацията.

# 23. Real-TimeScheduling

**Планиране в реално време**

* Правилността на системата зависи не само от логическия резултат от изчисленията, но също и от времето, за което тези резултати се получават
* Задачите или процесите се опитват да контролират или реагират на събития, случващи се във външния свят
* Тези събития се случват в “реално време” и задачите трябва да могат да ги следват.

**Твърда срещу мека**

* “Твърда “ задача в реално време:
	+ Такава, която има краен срок
* “Мека задача” в реално време
	+ Има краен срок, но той е пожелателен, а не задължителен

**Периодични срещу непериодични**

* Периодични задачи
	+ Приключват се редовно, веднъж за период или през *T* единици от време
	+ Времева диаграма на периодична задача 
* Непериодични задачи
	+ Имат времеви ограничения или за край, или за начало

**Системи в реално време**

* Контрол на лабораторни експерименти
* Контрол на процесите в индустриалните заводи
* Роботика
* Контрол на въздушното движение
* Телекомуникации
* Военно командване и контролни системи

**Характеристики на системите в реално време**

* Системите в реално време имат изисквания в 5 области: :
	+ **Детерминизъм** (Determinism)
	+ **Отзивчивост** (Responsiveness)
	+ **Контрол от потребителя** (User control)
	+ **Надеждност** (Reliability)
	+ **Защитена от сривове работа** (Fail-soft operation)

**Детерминизъм**

* Операциите се извършват във фиксирани, предварително определени моменти или в предварително определени интервали от време
* Има значение с какво закъснение операционната система потвърждава прекъсване и дали има достатъчно капацитет за обработка на всички заявки в рамките на заявеното време

**Отзивчивост**

* Колко време след потвърждение е нужно на операционната система да обслужи прекъсването
* Отзивчивостта включва:
	+ Времето за започване изпълнението на прекъсването
	+ Времето за извършване на прекъсването
	+ Ефекта от влагането на прекъсването

**Контрол от потребителя**

* Важно е да се позволи на потребителя да има прецизен контрол върху приоритетите на задачите.
* Може да се позволи на потребителя да указва неща като странициране или замяна на процеси
* Използваните алгоритми за дисков трансфер
* Правата на процесите

**Характеристики**

* **Надеждност**
	+ Влошаване на производителността може да има катастрофални последствия
* **Защитена от сривове работа**
	+ Способността на системата да се срине по такъв начин, че да съхрани колкото се може повече възможности и данни
	+ Стабилността е важна – ако не е възможно да се спазят всичките крайни срокове, поне критичните такива трябва да се спазят.

**Свойства на операционни системи в реално време**

* Бързо превключване на процеси или нишки
* Малък размер
* Възможност бързо да отговори на външни прекъсвания
* Многозадачност с инструменти за вътрешно процесна комуникация, като семафори, сигнали и събития
* Употреба на специални последователни файлове, които могат бързо да акумулират данни
* Планиране с изместване на база приоритет
* Минимизиране на интервалите, през които прекъсванията са деактивирани
* Отлагане на задачи за фиксирано време
* Специални аларми и таймаути.

**Класове алгоритми за планиране в реално време**

* **Статично таблично (Static table-driven)**
	+ Изпълнението на задачата се определя при пускането й
* **Статично приоритетно с изместване (Static priority-driven preemptive)**
	+ Използва се традиционно планиране с приоритети
* **Динамично на базата на план (Dynamic planning-based)**
	+ При пускане се определя кое е най-доброто
* **Динамично на базата на best effort**
	+ Не се прави анализ кое е най-доброто

**Планиране на крайните срокове**

* При приложенията в реално време не е важна скоростта, а задачите да са завършени
* “Приоритетите” са груб инструмент и може да не успеят да уловят критичността във времето на задачите
* **Използвана информация**
	+ Кога да е готово
	+ Краен срок за начало
	+ Краен срок за завършване
	+ Време за обработка
	+ Ресурсни изисквания
	+ Приоритет
	+ Планиране на подзадачите

# 24. Файлови системи. Видове. Организация.

Важно е да се отбележат основните понятия, свързани с файлове.

Това са:

* Поле – основен елемент на данните;
* Запис – съвкупност от взаимосвързани полета, разглеждани като едно цяло в рамките на приложението;
* Файл – съвкупност от взаимосвързани записи;
* База от данни – съвкупност от свързани данни.

Системата за управление на файлове е системно програмно осигуряване, което предоставя услуги на потребителите и приложенията при използване на файлове. Единственият начин за осъществяване на достъп на потребителите или приложенията до файловете е чрез система за управление на файловете. Това освобождава потребителя или програмиста от необходимостта от разработване на специализиран софтуер за всяко приложение и осигурява на системата средства за управление на този тип ресурси.

Основните цели на системата за управление на файлове са следните:

* Задоволява нуждите и изискванията на потребителя при осъществяване управлението на данните, което включва съхраняване на данните и способност за изпълнение на горепосочените операции;
* Гарантира, до толкова, доколкото е възможно, валидността на данните във файла;
* Оптимизира производителността, от гледна точка на системата (количеството работа) и от гледна точка на потребителя (времето за отговор);
* Осигурява поддържане на вход и изход за различни типове устройства;
* Елиминира възможностите за загубване и разрушаване на данните;
* Поддържа вход и изход за голям брой потребители в една многопотребителска система.

Файловата система в най-общ вид може да бъде представена по следния начин:



Има много видове файлови системи. Първата е тази на UNIX. След това е разработена системата на MS-DOS. В Windows има възможност за избор между две файлови системи. Това са FAT32 и NTFS.

* FAT(File Allocation Table) се използва предимно в дискети, флаш памети, цифрови камери и други преносими устройства. Това е така, заради относителната простота на тази файлова система. Фактът, че е поддържана от може би всички операционни системи е още една причина да се използва за пренасяне на данни.
* NTFS (New Technology File System) е стандартната файлова система за всички операционни системи на Windows от NT насам.За разлика от FAT, NTFS поддържа метаданни и използва напреднали структури данни за подобряване производителността, надеждността, запълването на диска и други подобрения.

# 25. Управление на Вход/Изход и Schedulingна диска.

Управлението на входа и изхода е една от основните функции на ОС.

Устройствата се заемат и освобождават за индивидуално или съвместно използване. Основните операции над устройствата са: четене, запис и управление. За изпълнението на тези три операции, ядрото трябва да извърши следните действия:

* Иницииране на входно-изходни операции;
* Обработка на входно-изходни прекъсвания, постъпили от устройството при завършване на операцията или при възникване на грешка.

Основния входно-изходен контролер е отговорен за всички входно-изходни инициализации и прекратявания на файловете.

Контролните структури определят входно-изходните устройства, графика (scheduling, както е модерно да се казва в момента) и статута на файловете.

Контролерът е зает с графика за достъп на диска с цел повишаване на производителността. За целта са назначени входно-изходни буфери и е заделена вторична памет.

# 26. Вход/Изход – буфериране.

От презентацията:

Процесите трябва да изчакват входно-изходните операции. За да се избегне мъртвата хватка, някои страници трябва да останат в основната памет по време на входа/изхода.

Блоково буфериране. Блоково-насочено устройство съхранява информацията в блокове, които обикновено са с постоянен (фиксиран, както е модерно да се казва в наши дни) размер. Трансферите се осъществяват по един блок при всяко прехвърляне. В общи линии е възможно да се обърнете към данни, според техния блоков номер. Дисковете и USB ключовете са примери за блоково-насочени устройства.

Буфериране на потоци. Потоко-насочено устройство изпраща и приема данни като поток от байтове без блокова структура. Примери за такива устройства са: терминали, принтери, комуникационни портове, мишки.

Когато няма буфер, ОС директно достъпва устройството, от което се нуждае.



Единичен буфер. ОС възлага на буфер в основната памет за входно-изходна операция.



Блоков единичен буфер. При блоково-насочените устройства, входящите прехвърляния (трансфери) са направени към системния буфер. Когато прехвърлянето (трансферът) приключи, процесът мести блока в потребителското пространство и веднага изисква нов блок. Това е направено според най-често разумното предположение, че данните са с последователен достъп. Само в края на последователността един блок ще бъде прочетен без това да е необходимо.

Двоен буфер. Използват се два системни буфера. Процесът прехвърля данни от (към) един буфер, докато ОС изпразва (запълва) другия.



Кръгов буфер. Двойният буфер може да е неадекватен (неефективен) ако процесът извършва внезапни прекъсвания на входа/изхода. Проблемът често може да се реши чрез употребата на повече от два буфера. В този случай наборът от буфери се нарича кръгов буфер, като всеки буфер е отделна единица в кръговия буфер.



*Допълнително:*

*Буферирането се използва за припокриване работата на процесора и входа/изхода.*

*Повечето компютри имат машинни команди за пряко управление на входа и изхода. Процесорът изпраща сигнали и данни, които се интерпретират от контролер, управляващ устройството. За да се синхронизира работата на процесора с по-бавните устросйтва се използва буфериране. За да се изпълни входно-изходна операция, процесорът записва команда на съответния контролер. Когато командата се приеме, процесорът може да продължи работата си паралелно с входно-изходната операция. След като бъде изпълнена командата, контролерът изпраща прекъсване към процесора. Другият подход за съгласуване работата на процесора е чрез периодична проверка флаговете на контролера(вместо прекъсвания). Предимството е, че обработка се извършва само, когато процесорът е готов, но също така се губи и време за проверка, когато няма заявки. Много контролери поддържат директен достъп до паметта (DMA). Използва се за освобождаване на процесора от работа по прехвърляне на данни между контролера и паметта. Процесорът трябва да изпрати допълнителна информация до адрес от паметта, където да се разположи блокът от данни. При прехвърлянето на данни, контролерът заема системните шини, като изпраща сигнал към процесора и поема управлението им. Контролерът може да заема шините до предаване на целия блок или след всеки предаден байт да връща управлението на шините на процесора и да ги изисква пак за следващия байт. В големите системи се използват канали за данни (входно-изходни процесори), които се включват между процесора и контролерите. Каналът е програмируем процесор, изпълняващ канална програма. Някои канали могат да изпълняват еднокомандни програми, а други да изпълняват последователност от команди. Тъй като процесор и канал се конкурират за достъп до паметта, по-голям приоритет има каналът. Когато той може да обслужва само едно устройство, каналът се нарича селекторен и към него се включват бързите устройства. Бавните се свързват към мултиплексорен канал, който може едновременно да обслужва повече устройства. Блок-мултиплексорният канал съчетава принципите на горните два – изпълнява едновременно няколко канални програми. За да се повиши гъвкавостта на изпозлване на каналите и блоковете за управление, както и да се повиши сигурността, могат да се въведат алтернативни пътища към устройствата, т.е. един контролер да се свърже с няколко канала и едно устройство към няколко контролера.*

*Според йерархичния (слоев) модел на организация на ОС, системата има 6 нива. Ниво 3 управлява входно-изходните устройства и буферира информационните потоци от и към тях. Над това ниво всеки процес използва абстрактни входно-изходни устройства. Информацията в този буфер се съхранява временно, докато е необходима. Тя се съхранява в блокове с постоянен размер.*

*Входно-изходните буфери могат да бъдат заключени във времето. Това се прави с цел страницата, която текущо е била запазена да не може да бъде заменена.*

# 27. Дисков кеш

Дисковият кеш е буфер в основната памет, предназначен за дискови сектори. Кешът съдържа копие на някои от секторите на диска. Когато е отправена входно-изходна заявка за определен сектор, се прави проверка за определяне дали секторът вече се намира в кеша. Ако е там, заявката се удовлетворява от кеша, а ако не е – изисканият сектор се прочита в кеша от диска.

Поради феномена на локалната връзка, когато блок данни е прихванат от кеша, за да удовлетвори единична входно-изходна заявка, вероятно е вбъдеще да има обръщения към същия блок.

Модерните дискове притежават между 8 и 64 MB кеш. Колкото повече – толкова по-добре. Наличието на кеш увеличава многократо бързодействието при най-често срещаните операции.

# 28. Поделяне на файлове (FileSharing). Блокиране на записи

В системите, в които множество потребители имат достъп почти винаги има условие не всички файлове да са достъпни от определени потребители. Необходимо е да се установят правата за достъп и управлението на спонтанния достъп. Правата на достъп може да бъдат ограничени по отношение на изпълнението, четенето, добавянето на информация или изтриването на такавана даден файл.

Потребителите се разделят на следните класове – притежател, специфичен потребител, групи потребители, всички. Притежателят на даден файл обикновено е човека, който го е създал. Той притежава всички права за достъп и може да ги спдоеля с останалите. Специфичните потребители притежават определено ID. Групите потребители не са определени индивидуално, но системата има начин, за да определя даден потребител към групата. Всички са цялата съвкупност от потребители, имащи достъп до системата.

Съществуват няколко вида разпределение:

* Гранично разпреление



* Външна фрагментация – от време на време проверява за свободно място на диска и размества записите. Големината на файла трябва да се знае при създаването му, за да се отдели толкова място



* Верижно рапределение – всеки блок съдържа информация къде се намира следващия



* Подредено верижно рапределение



* Индексно рапределение – файлът е рапределен по различни блокове и всеки пази информация за местоположението на следващия. Може разпределението да е по един или няколко блока(фиксирана дължина) или да е на блокове с различна дължина.



Управлението на свободното пространство става с така наречените бит-таблици. Така по лесно се става рапределението. При този метод се използва вектор, съдържащ по един бит за всеки блок от диска. Ако бита съдържа 0, това съоветсва на свободен блок, всяка 1 - на зает блок.

Д*опълнително:*

Поделянето на файлове е разпространяването или осигураването на достъп до съхранена цифрова информация като компютърни програми, аудио и видео записи или документи. То може да се осъществи по различни начини. Съхранението, преноса и разпределението са често срещани методи за споделяне на информация, включващи ръчното споделяне чрез използване на преносими носители на информация, интернет базирани документи с хиперлинкове и др.

**P2P(peer-to-peer) поделяне на файлове**

При подобно поделяне на файловете потребителите използват софтуер, който да ги свърже към peer-to-peer мрежата, за да могат да споделят с други потребители, намиращи се в нея. Тогава файловете могат директно да бъдат сваляни от другите потребители. Ако има по-големи файлове, те се раздробяват на по-малки части, които могат да бъдат достъпени от много хора и в последствие съединени от програмата за сваляне.

**File hosting services**

Те са алтернатива на peer-to-peer софтуера. Това са сайтове, които хостват файлове, за да могат потребителите да ги свалят. Понякога се използват заедно с инструменти за интернет сътрудничество, като електронна поща, форуми, блогове и др., в чиито се поставят преки връзки за изтегляне на файлове.

# 29.UNIXI/O,LINUXI/O,WindowsI/O;

**UNIX**

В UNIX всяко входно-изходно утройство се свързва с определен файл. Той се управлява от файловата система и се записва и чете като стандартен файл с информация на потребителите. Това създава ясен, единен интерфейс за потребители и процеси. За да има достъп до устройството(за четене и писане), е необходима употребата на специален файл, свързан с устройството. Има два вида за вход и изход – буфериран и небуфериран. Небуферираният обикновено включва DMA(direct-memory access) устройство като трансфера заема място точно между I/O модула и мястото за обработка на данните от вход-изхода. Това е най-бързия метод. Устройството не можа да бъде достъпено за изпълнение на други процеси, а самия процес е заключен в основната памет. Буферирания минава през буферите на системата. Използват се два вида бифери – системен буферен кеш(system buffer caches) и опашки от символи(character queues).



Информацията, която се пренася между буферирания кеш и място за обработка от потребителя винаги предполага използването на DMA. За да се обработва буферирания кеш, се използват три списъка:

* Free list – списък със всички слотове в кеша (слота е свързан както буфера в UNIX; всеки слот съдържа един дисков сектор), които са достъпни за разпределяне.
* Device list – списък с всички буфери, които в момента са свързани с всеки диска.
* Driver I/O queue – лист пт буфери, които в момента се използват или изчакват за входно-изходен поток от определено устройство.

**LINUX**

LINUX I/O е много подобен на UNIX I/O. Ядрото на LINUX се свързва с определен файл за всяко I/O устройство. Всяка постъпваща операция се поставя в сортирана опашка. Същата операция се слага в опашката за четене (тип FIFO) при необходимост от четене и за писане(тип FIFO) при необходимост от писане. Всяка операция има определено време, закоето трябва да бъде изпълнена (0,5 сек. – за четене и 5сек. За писане).



След LINUX 2.4 има единна страница от кеша за целия трафик между диска и основната памет. Преимуществата са, че когато трябва да се пишат обратно страници на диска, могат да се подредят и да се запишат правилно. Другото преимущество е, че страниците в кеша може да се достъпят пак.

**Windows I/O**

При Windows управляващия вход-изхода е отговорен за всички входно-изходни процеси на ОС. Той предоставя единен интерфейс, който всички устройства могат да извикват.



I/O мениджъра работи пряко с кеш мениджъра, устройствата на файловите системи, мрежовите устройтсва и хардуерните устройства. Windows предоставя две възможности за опериране синхронно и асинхронно. Асинхроният начин се използва, когато е възможно да се оптимизира представянето на приложението.

# 30. Защита на файловите системи

Защитата има четири компонента: обекти на защитата,разрешени типове за достъп до файлове, потребители ,желаещи достъп до файловете, и процедури за защита на файловете.Всеки файл има собственик – потребителят ,който го е създал.Ако той реши , че файлът може да бъде използван от други потребители, трябва да укаже кой и как може да се обръща към файла.Процедурите за защита, реализиращи тези възможности, могат да използват глобални или локални методи ( свързани с всички файлове или с всеки файл). Процедурата трябва да има възможност да идентифицира потребителя.

При някои ОС се разчита на невъзможността потребител да достигне до файл, който не може да именува и следователно не може да работи с него.Достъпът до всеки файл може да бъде управляван подобно на достъпа до изчислителните системи чрез пароли. Идентификацията на потребителя при достъп до файл става, като му се даде парола.Система с пароли може да се реализира различно- с използване на глобални или локални методи.Възможно е да се определи една парола за достъп до всички файлове или по една парола за всеки за всеки тип на достъп. Всеки потребител, използващ някаква привилегия (право), знае съответната парола и назовавайки файл, въвежда и паролата за вида на достъп.Недостатък на този подход е, че колкото повече са потребителите, толкова е по-голяма възможността да се узнае паролата от потребителя без права.Стойността на подобна реализация е относително ниска.Друга възможност е да се свърже парола с всеки файл и да се свърже парола със всеки файл(и трябва да се реализира по- детайлна защита, трябват повече пароли).Вариант е с всеки файл да се свърже списък на достъпа (води се от собственика), съдържащ идентификатор на потребителя и парола(обикновено е достатъчна една парола за всички типове достъп).Всеки потребител има различна парола. Така ако се разкрие парола, засяга се само един потребител.Използват се и други схеми.Например някои системи позволяват да се свързва парола с подчправочник, а не с файл.

# 31. Видове вируси. Методи на атакуване.

***Видове вируси:***

* *Червеи* - Червеят е програма, която се размножава, но не инфектира други програми. Той се копира от и на флопи дискети, CD, DVD, Blu-Ray и "флашки", както и на различни дялове на хард диска. Ако заразеният компютър е част от мрежа, той може да инфектира и другите компютри в нея. Червеите често крадат и унищожават данни и се разпространяват основно чрез Интернет.

 **Модел на разпространение на червея:**

* *Троянски коне* – В [компютърния](http://bg.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8E%D1%82%D1%8A%D1%80) свят **троянският кон**, наричан още **троянец**, е злонамерена програма, за която се твърди, че притежава някаква полезна цел, а всъщност би причинила нещо съвсем различно при изпълнението си, например: превземането на канали в [***IRC***](http://bg.wikipedia.org/wiki/IRC), изтриване на съдържание от [твърдиядиск](http://bg.wikipedia.org/wiki/%D0%A2%D0%B2%D1%8A%D1%80%D0%B4_%D0%B4%D0%B8%D1%81%D0%BA), кражба на поверителни данни (пароли, информация за банкови сметки и кредитни карти) и др. Тези програми са получили името си от мита за [големия, кухдървенкон](http://bg.wikipedia.org/wiki/%D0%A2%D1%80%D0%BE%D1%8F%D0%BD%D1%81%D0%BA%D0%B8_%D0%BA%D0%BE%D0%BD), чрез който гърците печелят [Троянскатавойна](http://bg.wikipedia.org/wiki/%D0%A2%D1%80%D0%BE%D1%8F%D0%BD%D1%81%D0%BA%D0%B0_%D0%B2%D0%BE%D0%B9%D0%BD%D0%B0), като се скриват в него и се промъкват в укрепения град [Троя](http://bg.wikipedia.org/wiki/%D0%A2%D1%80%D0%BE%D1%8F).

Троянските коне обикновено се разпространяват под формата на изпълними файлове за [Microsoft Windows](http://bg.wikipedia.org/wiki/Microsoft_Windows): .**[exe](http://bg.wikipedia.org/w/index.php?title=Exe&action=edit&redlink=1)**, .[**scr**](http://bg.wikipedia.org/w/index.php?title=Scr&action=edit&redlink=1), .[**bat**](http://bg.wikipedia.org/w/index.php?title=Bat&action=edit&redlink=1) или .[**pif**](http://bg.wikipedia.org/w/index.php?title=Pif&action=edit&redlink=1)

* *Droppers* (контейнери) - "Контейнерът" е такава програма, която е създадена да преодолее антивирусната защита на компютъра - обикновено чрез криптиране, което пречи на антивирусната програма да го засече. Тя се спотайва в компютъра до настъпването на определено събитие и тогава го инфектира с вируса, който съдържа.
* *Бомби* - Бомбата е вреден скрипт или програма, която се задейства при изпълнението на определени условия. Някои бомби се активират на определени дати като използват системния часовник.
* *Boot секторен вирус* - Този тип вируси инфектира boot записа на хард диска. Отначало той премества или презаписва оригиналния boot запис като го заменя с инфектиран. Преместеният оригинален запис се записва в сектор, който вирусът маркира като повреден, за да не се използва повече.
* *Файлов вирус* - Това е един от най-разпространените видове вируси. Тези вируси търсят файлове с определено разширение (обикновено изпълними файлове като \*.com и \*.exe) и ги инфектират.
* *Compiler и Link вируси* - тези нападат source кода. Затова и те са специализирани за точно определен програмен език и версия. Обикновено се гнездят в някоя от файловите библиотеки и при свързването се сглобява с останалите модули. Отстраняват се лесно, поради това, че се намира в отключен source код.
* *Source вируси*  - в същността си са идентични със compiler и link вирусите, но за разлика от тях се мултиплицират единствено и само директно в source кода на програмата. Поради същността си са приложими само за интерпретативните езици като Lisp, Prolog, XL…
* *XP и AI вируси* - Този вид вируси не могат да повредят файловата система или да повлияят на хардуера на компютъра, но могат да изменят резултата, който експертната система връща, като по този начин да повлияе съдбоносно в определени ситуации.
* *Макро вируси* - вируси специално написани на "Word macro language" и на "Visual Basic Macro language", като създават отделни макроси - вируси. Тези вируси често предизвикват правописни и стилистични грешки по текстовете. Тези вируси променят своя код с всяка инфекция, което ги прави трудни за засичане.
* *Стелт вируси* - Те са най - коварните защото причиняват големи щети и остават резидентни в оперативната памет. Stealth  вирусите още прикриват симптомите на вирусната инфекция и взаймодействат с различни антивирусни програми, като принуждават програмата да съобщи че няма вируси. Тези вируси не показват промени по размерите на файла, който са инфектирали.
* *Презаписващи вируси* - Това е най-простата форма на вирус. При тях първоначалната програма или част от нея се презаписват от вируса. Вирусите се откриват лесно и се изолират
* *Непрезаписващи вируси* - В противоположност на презаписващите вируси, при непрезаписващите функцията на оригиналната програма остава запазена. Най-често кодът се прилепя към края на програмата.
* *Системни вируси* - Системните вируси са особено коварни, тъй като се настаняват в паметта още при зареждането на операционната система. По този начин те имат възможност да прилагат въздействието си върху всички действия на операционната система.
* *Live-and-Die и Seek-and-Hide вируси* - Тук става дума за вирусни програми, които се задържат определено време в дадена програма или система и от време на време напускат нападнатия софтуер.
* *Вируси, нападащи командния интерпретатор* - Тези вируси нападат програмата, която се грижи за потребителския интерфейс към операционната система.
* *Драйверни вируси*- този тип вируси използват кода на драйверните програми за периферните устройства, за да се разпространяват оттам и върху други файлове.

***Видове атаки:***Преглед на информация; Допълнителни команди; Разпадане на връзка; Маскиране; Атаки чрез спец. клавиши и комбинации; Непредвидени параметри; Измама на оператора; Прехващане; Логическа бомба; Троянски кон.

# 32. Защитни стени



ОС може да осигури различни степени на защита за различните обекти,

потребители или приложения. ОС трябва да създаде баланс между поделянето и

защитата на ресурсите между отделните потребители.

Три ключови цели стоят в същността на компютърната защита:

* Поверителност(Confidentiality)
* Цялост(Integrity)
* Наличност(Availability)
* Защитна стена(*firewall*)

Най-често защитните стени работят на нивото на мрежовия и транспортния слоеве (network layer, transport layer), където изследват пакетите данни на TCP/IP протоколите и обикновено взимат решенията си в зависимост от IP адреса на изпращача или дестинацията, порта, от който пакетът е получен или на който ще се изпрати, или всяка комбинация от тези параметри. Гледат се също така и опциите в заглавната част на пакета. Защитните стени, които работят на приложния слой от OSI модела, филтрират трафика между вътрешната и външната мрежи по отношение на пренасяната в пакетите информация, чрез зададени ключови думи и като следят за [спам](http://bg.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B0%D0%BC), [компютърни вируси](http://bg.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8E%D1%82%D1%8A%D1%80%D0%B5%D0%BD_%D0%B2%D0%B8%D1%80%D1%83%D1%81) и [троянски коне](http://bg.wikipedia.org/wiki/%D0%A2%D1%80%D0%BE%D1%8F%D0%BD%D1%81%D0%BA%D0%B8_%D0%BA%D0%BE%D0%BD_%28%D0%BA%D0%BE%D0%BC%D0%BF%D1%8E%D1%82%D1%80%D0%B8%29).



**Функции на защитната стена**

Фигуративно казано, защитната стена представлява "граничен контролен пункт" за желаещите да преминат пакети. Целият [трафик](http://bg.wikipedia.org/w/index.php?title=%D0%A2%D1%80%D0%B0%D1%84%D0%B8%D0%BA&action=edit&redlink=1) се осъществява през този пункт, който има за задача да пропуска само което е безопасно. При настройка на защитата има два генерални подхода:

1. пропускат се всички данни и услуги с изключение на изрично забранените,
2. забраняват се всички данни и услуги с изключение на специално разрешените.

**Основни функционалности**

Три са основните функции на защитната стена:

* да блокира данните, за които има вероятност да прикриват хакерски атаки,
* да скрива информация за мрежата, като за изходящия трафик [маскира](http://bg.wikipedia.org/w/index.php?title=%D0%9C%D1%80%D0%B5%D0%B6%D0%BE%D0%B2%D0%B0_%D0%BC%D0%B0%D1%81%D0%BA%D0%B0&action=edit&redlink=1) IP адреса на мрежата с IP адреса на защитната стена,
* да води дневници (logs) за информационния поток със записи на определени събития.

### Допълнителни функционалности

Освен основните си функционалности, защитната стена между мрежи има и допълнителни възможности:

* филтриране на съдържанието (content filtering),
* преобразуване на мрежови адреси и номера на портове (network address translation, port address translation),
* балансиране на натоварването (bandwidth shaping, QoS),
* откриване на пробиви в системата (intrusion detection).

## Видове защитни стени

Съществуват 3 основни вида защитни стени - филтриращи маршрутизатори (filtering routers), поддържащи връзката пакетни филтри(stateful packet filters), и приложни щлюзове (application gateways). Повечето защитни стени прилагат съчетания от гореспоменатите видове.

* Филтриращите маршрутизатори (англ. filtering routers)разглеждата всеки пакет отделно - т.е. не обръщат внимание на пакета като част от установена вече връзка.
* поддържащи връзката пакетни филтри(stateful packet filters ), както подстказва името им разглеждат всеки един пакет като част от вече установената връзка
* приложни щлюзове или проксита(англ. application gateways или application proxies) - програми намиращи се между крайният потребител и публичната мрежа - т.е. шлюзовете изпълняват методите вместо крайните потребители, защитавайки ги така от външни опасности. Тези приложения имат силни защитни свойства понеже крайните потребители никога не комуникират директно с хостове в Интернет

# 33. Антивирусни програми. Принципи на действие – Видове

***Anti-Virus Engine***

Антивирусен двигател (Antivirus engine) или още антивирусно ядро

Една от основните части на антивирусната програма е така наречения антивирусен двигател – модул, отговарящ за проверката на обектите и обезоръжаването на вредните програми и компоненти. От това как е разработен и какви методи за откриване и евристика използва той, зависи качеството на вирусите и като следствие, нивото на защитата, предоставено от антивирусната програма.

*Основните критерии, по които може да се определи качеството на антивирусния модул са:*
• Нивото на откриване с евристични анализатори.
• Нивото на лъжливите сработвания. Ако за 100% не заразени файлове, програмата рапортува за възможно заразен файл, то това е лъжливо сработване. Фалшива тревога.
• Поддръжката на голямо количество опаковчици и архиватори. Това е много важен фактор, защото създателите на вредни програми, написали и получили няколко различни вируса ги опаковат в няколко изпълними модули, след което ги разпространяват. За антивируси поддържащи почти всички познати архиватори, няма да е трудно да идентифицират всички тези модификации на вируса.
• Честота и размер на обновяванията на антивирусната база. Честото обновяване гарантира, че потребителя винаги ще бъде защитен и от току що появили се вируси.
• Възможност за обновяване на самия антивирусен двигател, без обновяване на цялатапрограма. В някои случаи, за изолирането на вирус трябва да се обнови не самоантивирусната база, но и самия антивирусен двигател. Ако софтуерът не поддържаподобна опция, то потребителя може да остане беззащитен пред новите вируси, а и тазивъзможност позволява да се подобри двигателя на антивирусната програма или да секоригират възникнали или открити пробиви и грешки.
Антивирусният двигател (Anti-Virus Engine) е основен компонент във всяка антивирусна програма.

За откриването на вируси са реализирани няколко технологии при повечето антивирусни двигатели:
*• Търсене по “сигнатура”* (байтове с уникална последователност)- ”. Сигнатура - това е уникална “линия” байтове, която еднозначно характеризира тази или онази вредоносна програма. Сигнатурното търсене се използва за изолиране на вируси и други вредни програми, като се прилага от времето на най-първите антивирусни програми да сега. Неоспоримо качество на сигнатурното търсене е - скоростта на работа и възможността да открива няколко вируса с една сигнатура.
• *Търсене по контролни суми или CRC* (контролни суми с уникална последователност на байтовете) - се явява модификация на търсенето по сигнатури. Същността на метода се състои в това, че за да се открие вредния код се взима не само “опорния” ред - сигнатура, а по-точно казано контролната сума на този ред и местоположението на сигнатурата в тялото на вредоносната програма. Местоположението се използва, за да не се събират контролни суми за целия файл.
*• Използване на редуцирани маски*- Използването на маски за разкриването на вредния код, много често е усложнено от наличието на шифрован код (при така наречените полиморфни вируси), защото в тези случаи, или е невъзможно да се избере маска, или маската с максимален размер не удовлетворява условието за еднозначна идентификация на вируса, без лъжливи сработвания
• *Криптоанализ*- Криптоанализът се изразява в следното: по известен базов код на вируса и по известен зашифрован код (или по подозрителен код, приличащ на зашифровано тяло на вирус) се възстановяват ключовете и алгоритъма на програмите на разшифровчика. След това, този алгоритъм се прилага към зашифрования участък и в резултат от това се явява разшифрованото тяло на вируса.
*• Статистически анализ*- Статистическият анализ също се използва за откриване на полиморфни вируси. По време на своята работа, скенера анализира честотата на използваните команди, прави таблица на срещащите се команди и на основата на тази информация прави извод за заразяването на файла с вирус. Този метод е ефективен за търсене на някои полиморфни вируси.
*• Евристичен анализ* - Представлява набор от програми, които анализират кода на изпълнимите файлове, макроси, скриптове, памети или застрашени сектори за изолиране в тях на различни типове вредоносни компютърни програми.
*• Емулация* - Идеята на този метод се състои в това, че емулира изпълнението на програмата в специален буфер. След емулацията в буфера се намира разшифрованото тяло на вируса, което е готово вече за идентифициране със стандартните методи

Съществуват два принципа на работа на анализатора.

 - Статичен метод. Търсене на общи кратки сигнатури, които присъстват в повечето вируси (така наречените “подозрителни команди”).
 - Динамичен метод. Същността на метода се състои в емулация на изпълнение на програмите и протоколиране на всички техни “подозрителни” действия.. За разлика от статичния, динамичния метод използва повече ресурси, но и нивото на откриване е по-високо.

# 34. Ауторизация и Автентикация

*Автентикация* е един или няколко ИТ процеса установяващи валидността между крайния потребител и неговата претендирана самоличност. Автентикацията използва един или няколко фактора, посредством които ИТ процесите извършват проверка на достоверността на крайния потребител. Автентикационният фактор е информация или устойство използвано от крайния потребител – например Парола, Цифров сертификат, Смарт карта, Биометрични данни и т.н. Прието е също така вторият фактор за автентикация да е с класификацията „да имаш нещо”, например смарт карта, USB интелегентно устройство, мобилен телефон по който да получиш допърнителна парола, и т.н. Често в световната практика се използват биометричните данни като трети фактор за автентикация, например отпечатък на пръст, форма на лицето, изображение на ириса, и т.н.

*Идентификация* е един или няколко ИТ метода които използва крайният потребител (човек или компютър) за да се обособи и различи от другите крайни подтебители в ИТ системата. Идентификационния ИТ метод определя как крайния потребител да се обособи и различи.

*Ауторизация* е един или няколно ИТ процеса осигуряващи достъп на крайния потребител до определени ИТ ресурси, според проведената негова идентификацията и автентикация.

От гледна точка на Идентификация и Автентикация, ИТ системите може да се разгледат в две групи – използващи Корпоративна мрежа, и такива използващи Интернет за комуникации.

**Методи за автентикация**

Предлагат се следните 5 методи за автентикация, които се базират на съответните фактори за автентикация:

А) Отворен фактор – не се изисква никаква форма на автентикация (това е метод, когато всички възможни потребители имат достъп до всяка информация от определена група данни);

Б) Фактор „Нещо което знае потребителя” – Потребителят знае своето потребитеско име и парола за достъп до уеб страница;

В) Фактор „Нещо което има потребителя” – използва се смарт карта вградена в USB носител с инсталирана двойка частен и публичен ключ, заедно с цифров сертификат, като потребителят има този USB носител;

Г) Фактор „Нещо което знае потребителя и нещо което има потребителя” – използва се смарт карт четец с ПИН клавиатура и смарт карта със защитни ключове, като потребителят знае ПИН кода за досът до смарт картата и притежава смарт карта;

*Хеширани пароли*

Широко използвана техника за съхраняване на пароли.

Защита срещу различни криптоаналитични атаки.

UNIX Password Scheme



Д) Фактор „Нещо което представя потребителя и нещо което има потребителя” – използва се смарт картов четец със скенер за биометрични пръстови отпечатъци и смарт карта със защитни ключове ,като потребителят дава пръстовия си отпечатък за сканиране ипритежава смарт карта;

*Статично биометрична автентикация*

Включва:
 - Черти на лицето,
 - Пръстовите отпечатъци
 - Геометрия на ръката
 - Модел на ретината
На базата на разпознаване на образи:
 - технически сложни и скъпи

*Динамично биометрична автентикация*

Моделите могат да се променят
Включва : Ирис, Подпис, Глас, Типичен ритъм

*Разходите в сравнение с точност*



# 35. CloudComputing – типове.

***Дефиниция***-Това е начин на организация и работа на компютърните системи, при който компютърни ресурси като процесорно време, памет, дисково място и т.н., могат да бъдат динамично променяни (увеличавани) и предлагани на потребителя като услуга чрез Интернет. Тези ресурси обикновенно са резултат от действието на виртуализационни технологии

***История:***

* публични услуги 1960
* telecom cloud – 1990
* Salesforce.com -1999 – бизнес приложения.
* Възниква идея : “on demand” – при поискване - SaaS – Software As A Service
* MS Web Service – 2000
* Amazon Web Service – 2005
* Google 2007 – Start Google Apps
* HP,Yahoo, Intel – 2008 – Cloud Computing Test Bed

***Видове Cloud Computing:***

* Публични “облаци”
* Частни “облаци”
* Смесени “облаци”

***Структура на CC***:Client – Application – Platform – Infrastructure - Servers

***Client – Application – Platform – Infrastructure – Servers***:

* Софтуер като услуга - SaaS
* Платформа като услуга – PaaS
* Инфраструктура като услуга – IaaS
* Cloud Application Programming Interface (Cloud API)

***“Физика на Данните”:***

* Връзка м/у данни и процеси
* Защита и физика на данните
* Практики за защити
* Стратегии за програмиране



***Видове Услуги в Azure:***

* Изчислителна услуга (Compute Service)
* Услуга за Съхранение (Storage Service)
* Структура (Fabric)
* Създаване на мащабируемо web приложение
* Създаване на приложение с паралелна обработка

***Предимства на модела CC:***

* Намалява времето за изпълнение и времето за отговор
* Намалява риска при работа с инфраструктура
* Намалява стойността на приложенията
* Увеличава скоростта на въвеждане на новостите

***Amazon Elastic Computing Cloud(Amazon EC2)***



***Web Service:***



# 36. Виртуални машини – оснвни принципи на работа

Представлява програмен процес, който симулира хадуерен компютър. От гледна точка на работната операционна система няма разлика между виртуална и истинска машина. Върху един хардуерен компютър, могат да работят едновременно множество виртуални машини. Единственото ограничение е мощността на процесора и наличната памет.

 От гледна точка на реалната система, всяко подобно "симулирано PC" представлява достатъчно голям файл, в който е записана конфигурацията, състоянието и данните на инсталираната операционна система и нейните софтуерни приложения. Технологията на виртуалните машини позволява по-ефикасно използване на хардуерните ресурси, намаляване на разходите за хардуер, електрозахранване, охлаждане, мрежова инфраструктура. Използването на тези технологии предоставя ред преимущества пред инсталиране на две (или повече) операционни системи на един твърд диск или зареждане на операционна система от т.нар. Live CD. Докато в изброените случаи в даден момент имаме само една активна операционна система, с виртуализиране можем да имаме активни две, три и повече ОС-и работещи на един и същ компютър (или сървър). Бихме могли да изучаваме нова ОС докато работим в друга. Виртуалната машина заедно с операционната система, която носи тя със себе си, както и всеки друг софтуер, инсталиран под тази операционна система, могат да бъдат записани като файл на твърдия диск, за да бъдат архивирани и използвани и в бъдеще.