|  |
| --- |
| ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - ГАБРОВО |
| МРЕЖОВИ ОПЕРАЦИОННИ СИСТЕМИ |
| ОКС „Магистър“ |

|  |
| --- |
| 4/24/2017 |

**Модул I: Мрежови операционни системи**

**ТЕМА 1.1.**

**Мрежови операционни системи (МОС).**

**Основни характеристики. Класификация.**

**1. Операционна система.**

Операционната система е набор от програми, които са необходими за работа на компютъра. Тя е средството, чрез което се осъществява връзката между потребителя на компютъра и хардуера, тя е основна част от компютърния системен софтуер, която управлява и координира ресурсите на хардуера и софтуера и обслужва изпълняваните компютърни програми. Приложният софтуер обикновено има нужда от ОС, за да работи. ОС разпределя задачите по време, планира ефикасното използване на ресурсите на системата и може да включва специализиран софтуер за изчисление на ресурси: стойност на процесорното време, използвана памет, ресурси за печат и други.

**2. Видове операционни системи.**

Видовете ОС са свързани с развитието на понятия като пакетен режим, разделение по време и многозадачност, разделение по пълномощия, реален временен мащаб, работа с файлови системи.

Исторически необходимостта от оптимално използване на скъпите изчислителни ресурси води до появата на концепцията „пакетен режим“ на изпълнение на програмите, който предполага наличието на определен ред, при което ОС може да зарежда програмата в оперативната памет от външни носители на данни, без да изчаква завършването на изпълнението на предходната програма и така се избягва излишен престой на процесора. В развития вариант на пакетния режим времето на процесора се разпределя между изпълнението на няколко програми (понятието за „многозадачност“ или „мултипрограмиране“). Исторически понятието многозадачност възниква през 60-те години, когато входно-изходните устройства са отначало телетипи, а по-късно терминали. Тъй като скоростта на въвеждането на данни от оператор е много по-ниска от скоростта на обработка на същите данни от процесора, използването на компютъра от само един оператор води до прахосване на скъпи изчислителни ресурси. „Разделението по време“ (time sharing) позволява създаването на многопотребителски системи, в които централният процесор и блокът на оперативната памет обслужват много потребители. При това част от задачите (като въвеждане или редактиране на данни) могат да се изпълняват в диалогов режим чрез терминали, а други (като обемните изчисления) – в пакетен режим.

След появата на персоналните компютри и работните станции разделението по време на практика е заменено с многозадачност.

**2.1. Видове операционни системи според броя на потрбителите и задачите:**

* Еднопотребителска, еднозадачна - MS-DOS - на дадения компютър работи само един потребител и той може да стартира само една задача.
* Еднопотребителска, многозадачна – Windows, OS/2, MAC OS - на дадения компютър работи само един потребител, но той може да стартира много задачи.
* Многопотребителски, еднозадачни - много потребители могат да работят на дадения компютър, но се стартира само една задача.
* Многопотребителски, многозадачни - UNIX - на дадения компютър могат да работят много потребители и могат да се стартират едновременно по няколко програми (задачи) на един компютър.

**2.2. Многопроцесорни операционни системи.**

* Хомогенни - имат няколко еднакви централни процесора -2, 4, 8 (от 2 до 8). Работят с обща оперативна памет. Няколко програми могат да работят едновременно (паралелно);
* Нехомогенни - имат няколко процесора, но с различна функция: за обикновени действия - събиране, изваждане, умножение, деление; за аритметични сложни действия – SIN, COS и други.

**2.3. Мрежови ОС**

Операционни системи, които управляват компютри, свързани в мрежа. Мрежата представлява група от компютри, свързани чрез комуникационни линии, по които обменят информация. Мрежовият достъп дава на потребителите възможност да се обръщат към програми и данни, намиращи се не само в неговия компютър, но и в други компютри Създава се мрежова ОС по пътя на разширение на съществуващото програмно осигуряване със средства за връзка. ОС общува с други подобни или специални системи, обикновено разработени от същата фирма.

**2.4. Виртуални ОС**

ОС, които създават виртуални машини, т.е. на 1 компютър може да има няколко ОС. Такава ОС представлява една реална машина във вид на няколко виртуални машини и управлява едновременно работата на няколко различни ОС, като всяка от тях се изпълнява на собствена виртуална машина и предоставя на потребителите пълният си набор от функции. Всеки от потребителите може да избере своя ОС, като при работата с нея има чувството, че на негово разположение е цялата изчислителна система.

**2.5. Системи в реално време (Real-Time Systems)**

Те се използват се за следене и управление на процеси в производството. Тези ОС най.често се проектират за управление на технологични процеси, затова още се наричат ОС за управление на процеси. Биват:

* Системи, управляващи процес – например при пещ, която трябва да поддържа температурата в определени граници, то системата следи температурата и включва или изключва нагревателите и информира.
* Системи, следящи процес – приемат информация и сигнализират, ако нещо стане-светлинен, звуков сигнал подават, а операторът решава, какво да прави. Използват се при военните, за производство на ракети.

ОС за реално време свеждат до минимум участието на човека в процеса на управление. Обикновено се предвижда вмешателство на оператора при непредвидени ситуации или при възникване на грешки.

**3. Подходи при проектиране на ОС**

Монолитни системи: нямат строга структура, а са набор процедури, които могат да се викат 1 друга. Този подход за изграждане на ОС отговаря на представата, че тя е една голяма програма. ОС няма вътрешна структура, написана е като съвкупност от процедури. Всяка процедура има дефиниран интерфейс (входни и изходни параметри) и може да вика коя да е друга. Всяка процедура е видима за останалите. Отделните процедури се компилират и свързват в една програма, представляваща изпълнимата програма на ОС.

Слоев (йерархичен модел): Формулиран е от Дейкстра. Той се основава в разпределение на функциите на ОС по нива, разположени едно над друго. Най-ниското ниво е свързано с апаратурата, а най-високото-с потребителя.

* Ниво 0: управление на ЦП и на процесите
* Ниво 1: управление на паметта, виртуална памет
* Ниво 2: комуникация м/у процес и оператор
* Ниво 3: управление на входно-изходните устройства
* Ниво 4: потребителски програми
* Ниво 5: системен оператор

Ниските нива са свързани с хардуери. С представянето на ОС във вид на йерархична структура са свързани две идеи. Първата е,че на програмите, отнасящи се към по-високите нива, трябва да бъде предоставена възможност да ползват услугите на програмите, отнасящи се само към по-ниските нива. Другата е, че при прехода към по-ниските нива, съответстващите им функции трябва да се изпълняват все по-бързо.

**Виртуални машини** - използването на принципа на виртуалните машини прави възможна работата едновременно на различни ОС на една реална машина.

**IBM-VM**

Всички 4 ОС (последната програма не е под управление на ОС, а работи самостоятелно в супервайзорен режим) работят на една реална машина, благодарение на монитора на виртуалната машина. Всеки потребител има впечатлението, че разполага със собствена машина и се обслужва по традиционния начин.

Реализацията на виртуалната машина не е лека. Програмите на най-ниското ниво на ОС (ядрото) работят фактически с монитора на виртуалната машина, който предоставя същите ресурси,обслужване и функции, като реалната машина(без да е видим за ОС или за потребителя). Трудностите произтичат при точно дублиране на реалната машина. Всеки процесор има поне два режима: супервайзорен и потребителски. Мониторът може да работи в супервайзорен режим, защото тук той всъщност е ОС. Потребителите на отделните виртуални машини (в случая ОС1, ОС2 и т.н.) трябва да работят в потребителски режим. Следователно, трябва да се организира виртуален супервайзорен и виртуален потребителски режим, всеки от които работи при физически потребителски режим. Действията на потребителя (ОС), които изискват превключване в супервайзорен режим трябва да предизвикват превключване във виртуалните режими на машината.

Мониторът трябва да извърши всички съществени функции на ОС-той разпределя процесора между отделните виртуални машини и пази информация за състоянието им. Мониторът отделя за всяка виртуална машина собствена виртуална памет.

**Модел клиент-сървър-ядрото е универсално, а над него са програмите.**

Премества се част от системния код в по-високите нива, оставяйки минимално по размер ядро. Обикновено по-голяма част от функциите на ОС се реализират като потребителски процеси. Когато му е необходимо обслужване от ОС, потребителският процес (сега наричан процес-клиент) изпраща заявка към процес-сървър, който изпълнява заявката и връща отговор.

В този модел ядрото осъществява само комуникацията между процесите. ОС е разделена на части, всяка изпълняваща някаква нейна функция. Отделните части са малки, лесно се управляват и се изпълняват като процеси в потребителски режим. Това означава, че те нямат директен достъп до апаратурата и при повреда на отделен сървър не пропада цялата ОС.

**Мрежови операционни системи**

Към класа на слабо свързаните системи могат да се отнесат мрежовите операционни системи. Те осигуряват среда, посредством която потребител от своя локален PC може да получи достъп до ресурсите на всеки друг отдалечен PC на мрежата. За да се осигури такава среда, мрежовите ОС трябва да предоставят като минимум две възможности:

1. влизане като потребител в ОС на отдалечен компютър.
2. прехвърляне на данни м/у локалния и всеки друг отдалечен PC.

При необходимост, потребителят може дистанционно да влезе в ОС на всяка друга машина, използвайки команда от вида:login *<machine>*

Прехвърлянето на файлове от една машина на друга може да се извърши с команда от типа:copy machine1:file1 machine2:file2

С тази команда файлът file1 се копира от работната станция machine1 в работната станция machine2 под името file2.

Стандартът POSIX (Portable Operating System) дефинира множество от библиотечни процедури, които всяка отворена ОС трябва да притежава, за да комуникира с друга отворена ОС. Първоначално стандартът [POSIX](http://bg.wikipedia.org/wiki/POSIX) е разработен за съгласуване на различните версии на системите UNIX, но впоследствие се  разпространява и за други ОС.

Реализацията на услугите в мрежовите OC се извършват съгласно модела клиент-сървър. Комуникацията м/у клиентите и сървъра се реализира с помощта на съобщения със специфичен формат. Клиентите изпращат към сървъра съобщения заявки (request), а сървърът връща резултата от изпълнението на заявката на клиентите със съобщение отговор (reply).

Възможностите, включени в мрежовите OC, позволяват върху тях да се реализират различни разпределени приложни с-ми.

Мрежовите операционни системи служат като платформа за разпределени инструментални среди, известни като междинни (middleware). Middleware е допълнително ниво на абстракция над мрежовите OC, реализиращо услуги с общо предназначение (general purpose). Основната цел на **междинните системи** се състои в предоставяне на простота и удобство за програмиране на разпределени системи. Тази цел се постига чрез:

осигуряване на прозрачност (transparency) при програмиране. Прозрачността означава скриване на детайлите и особеностите на: реализацията на мрежата   и апаратурата; комуникацията между програмни компоненти;

Маскиране на различията на OC на компютрите в мрежата и [езиците за програмиране](http://www.technologybulgaria.com/resources/%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B8%D1%80%D0%B0%D0%BD%D0%B5).

Разработени са различни класове системи middleware, различаващи се според мощността на програмните абстракции, които предоставят и възможностите за използване в хетерогенни мрежи и компютърни архитектури:

Отдалечено извикване на процедури (Remote Procedure Call – RPC). Предоставя програмна абстракция за обръщение към процедури през мрежата;

Middleware системи, ориентирани на съобщения (Message-oriented Middleware – MOM). MOM осигуряват абстракция на опашка от съобщения, достъпна през мрежата. Тя  е добре известната като – пощенска кутия (mailbox);

Middleware системи, ориентирани на обекти (Distributed Object Middleware). Осигуряват програмна абстракция за обръщение към методите на отдалечен обект по същия начин, както към обект, разположен в адресното пространство на извикващата програма. Най-известните с-ми от този клас са: CORBA; DCOM и др.

2.Разпределени ОС

Към класа на силно свързаните системи се отнасят разпределените ОС. Разпределените ОС са предназначени за работа в РС мрежи без обща памет. Те позволяват на отделните потребители еднотипен достъп до локалните и отдалечените РС ресурси, като създават илюзията за работа в еднопроцесорна с-ма с времеделение. Основните характеристики на разпределените ОС са:

* единен глобален механизъм за междупроцесна комуникация;
* единен глобален механизъм на защита на програми и данни;
* един и същ механизъм за управление на процесите във всички машини с еднакъв интерфейс от системни извиквания за създаване, унищожаване, стартиране и преустановяване на процесите;
* глобална файлова с-ма с добре дефинирана семантика;

При разпределените с-ми в/у вс. машини се изпълняват идентични ядра на ОС, които координират по подходящ начин своите действия.

Предимства на разпространените операционни системи:

* Възможност за достъп до общи за всички процеси ;
* Съвместно използване на ресурсите;
* Възможност за комуникация в реално време между потребителите;
* Гъвкавост при експлоатация;
* Висока надеждност.
* Разпределените системи  и редица специфични проблеми за разрешаване, към които могат да се отнесат:
* Необходимост от разработване на разпределен [софтуер](http://www.statii.info/%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B8%D1%80%D0%B0%D0%BD%D0%B5-%D0%BD%D0%B0-c.html);
* Мрежови проблеми, свързани с необходимост от надеждна комуникация м/у отделните РС в мрежата;
* Необходимост от сигурност и защита на общите данни;

**Цели при проектиране на разпределени системи**

При проектиране и разработка на разпределени системи се преследва постигане на следните основни характеристики:

Прозрачност (transparency). Прозрачността се дефинира като скриване от потребителите и приложните програмисти на разпределената с-ма, така че тя да изглежда като еднопроцесорна с-ма.

Гъвкавост (flexibility). Гъвкавостта позволява лесно добавяне на нови функционални възможности в с-мата. Два подхода за изграждане, които оказват влияние върху гъвкавостта:

монолитно ядро на ОС (monolithic kernel). Представлява е ядро на централизирана операционна система, разширено с мрежови средства и интегрирани услуги за отдалечен достъп. Предимство- висока производителност. Недостатък – ниската гъвкавост на с-мата;

микроядро на ОС (microkernel). Предоставя минимален набор услуги – механизъм за междупроцесна комуникация, управление на паметта, управление на вход-изход на ниско ниво, ограничено управление и планиране на процесите на ниско ниво.

Надеждност (reliability). Неизправностите в разпределените системи като правило са частични, ако някои компоненти откажат, другите продължават да функционират.

Производителност (performance). Постигането на по-висока производителност по сравнение с централизираните с-ми е най-важната преследвана цел на разпределените с-ми. Според степента на разпаралелване се различават дребнозърнест (fine-grained) и едрозърнест (coarse-grained) паралелизъм. Дребнозърнестият паралелизъм предполага паралелно изпълнение на множество малки програмни фрагменти. Степента на паралелизъм е висока, и честотата на взаимодействие м/у фрагментите е висока. При едрозърнестия паралелизъм степента на паралелизъм е по-малка, тъй като паралелно се изпълняват големи програмни фрагменти. Тяхното взаимодействие е по-слабо, което води до намаляване на комуникационните разходи.

Разширяемост (scalability). Една система е разширяема, ако остава ефективна при значително увеличаване както на ресурсите, така и на потребителите в с-мата.. Подходящи са само разпределени алгоритми, които имат следните характеристики:

* няма единствена машина, която да събира информация за цялата на система;
* машините вземат решения, базирани само на локално достъпна информация;
* срив в една машина не предизвиква срив в работата на алгоритмите;

**Модел клиент-сървър**

1.Въведение – Той отразява взаимодействието между паралелно изпълняващи се програмни процеси. клиент-сървър, при които клиентите и сървърите се изпълняват върху различни машини, свързани в мрежа – локална (LAN) или глобална (WAN).

Комуникацията м/у клиентите и сървърите се базира на протокол без установяване на връзка, наречен протокол заявка-отговор . За комуникацията клиент-сървър в локална мрежа са необходими само три протоколни нива. Физическото и каналното ниво са необходими за доставянето на пакетите

Системното извикване send() изпраща съобщението от адрес &message на процес с идентификатор destination. Изпращащият процес да се блокира, докато съобщението не бъде изпратено. Системното извикване receive() блокира процеса, до получаване на съобщение за него.

Oсновните проблеми, които трябва да се решат са следните:

1. Начин на адресиране на процесите;
2. Блокиране или неблокиране на процеса при изпълнение на send() и receive();
3. Буфериране или небуфериране на изпращаните и получавани съобщения;
4. Осигуряване или не на надеждност при доставката на съобщенията;
5. Избор на архитектура на сървъра – последователна или паралелна;

Възможност за разширяемост.

2.Адресиране  
За да може да изпрати съобщение-заявка към нужния сървър, клиентът трябва да знае неговия адрес.

Адресът на сървъра може да бъде открит и използван по един от следните начини:

Твърдо зададен адрес в кода на програмата на клиента. Всяка промяна в адреса на сървъра, изисква преработка и прекомпилиране на програмата на клиента;

Откриване на сървъра чрез broadcast . За да се получи лесно преместваем сървър е необходимо присвояване на уникален адрес на процеса, който да не съдържа адрес на машина. Този адрес може да се получи по два различни начина: чрез обръщение към специализиран централен процес или самоназначаване на адрес. В първия случай е необходим централен процес, раздаващ последователни адреси на процесите. Във втория случай процесът сам избира собствен идентификатор от голямо разредно адресно пространство.

Преди да изпрати съобщение-заявка към сървъра клиентът трябва да открие машината, върху която той се изпълнява. За целта той изпраща по broadcast специален локализиращ пакет, съдържащ адреса на сървъра. Този пакет се приема от всички машини. В случай на положителна проверка, съответната ОС изпраща обратно съобщение към клиента, съдържащо мрежовия адрес на машината. Клиентът използва този адрес за изпращане на съобщение-заявка към сървъра.

За намаляване на трафика от broadcast съобщения клиентът може да кешира получаваните отговори. При необходимост от следващо обръщение към сървъра, клиентът първо търси адреса на машината в локалната си кеш.

Откриване на адреса чрез сървър на имена (name server) (фиг.3-c).  В този случай е необходима допълнителна машина, върху която се изпълнява сървър на имена, даващ съответствие между име на услуга и адрес на машина. Подходът дава възможност за обръщение към сървър по име на услугата, която предоставя. Името на услугата е постоянно и може да бъде заложено в програмата на клиента. При необходимост от получаване на услугата, клиентът първо се обръща към сървъра на имена, запитвайки за адреса на машината, върху която нужният  сървър текущо е зареден. След получаване на отговор с адреса на сървъра, клиентът се обръща директно към него със съобщение-заявка.

Локализация на сървър:

а) твърдо зададен адрес;

b) използване на broadcast;

c) използване на сървър на имена

3.Блокиращи и не-блокиращи комуникационни примитиви

Описаните системни извиквания send() и receive() реализират блокираща (blocking) комуникация между процесите. Този тип комуникация е известна още като синхронна (synchronous).

**Блокираща (синхронна) комуникация.**

Когато процес изпълни send(), той задава адреса на назначението и указва адреса на буфера, чието съдържание трябва да се изпрати по това назначение. Докато ядрото на ОС изпълни нужните действия по формиране и изпращане на съобщението, процесът се блокира. На него ще му се разреши да продължи изпълнението си със следващата след send() инструкция едва, след като съобщението бъде окончателно изпратено.

Блокиращите примитиви са лесни за реализация. Те не изискват буфериране на съобщенията в адресното пространство на ядрото, поради което съобщенията по-бързо се изпращат в мрежата.

**Не-блокираща (асинхронна) комуникация.**

Примитивите send() и receive() могат да бъдат реализирани и като не-блокиращи, т.е. асинхронни.

Ако send() е не-блокиращ примитив, управлението се връща на извикващия процес веднага, преди да бъде изпратено съобщението. По този начин се предоставя възможност процесът да може да продължи изпълнението си едновременно с изпращането на съобщението. Процесът в този случай обаче не може да разбере, кога съобщението е изпратено и е безопасно да използва повторно буфера за ново съобщение. Решаването на проблемa може да се постигне по два начина:

- копиране на съобщението в буфер на ядрото;

- генериране на прекъсване към процеса, щом съобщението бъде изпратено.

В първия случай съобщението се копира в буфер на ядрото, а буферът на процеса се освобождава. Недостатъкът се състои в това, че всяко изходящо съобщение трябва да се копира от адресното пространство на процеса в адресното пространство на ядрото. Това копиране при голяма изходяща комуникация, от една страна, може да доведе до снижаване на производителността, а от друга до загуба на съобщения, поради ограниченото пространство за буфери в системата.

При втория случай след извикване на send() на процеса се разрешава да продължи изпълнението си. Съобщението се изпраща от буфера на процеса и не се губи време за копиране. Този метод е високо ефективен и позволява висока степен на паралелизъм.

**4.Буфериране**

При изпълнение на примитива receive(address, &message) процесът съобщава на ОС своя адрес, по който очаква да получи съобщение (address) и буфера (&message), където да се запише пристигналото за този адрес съобщение. В случая се казва, че процесът слуша по указания адрес и е готов да получи едно съобщение, изпратено на този адрес. Когато съобщението пристигне, ядрото на ОС го записва в зададения буфер и събужда процеса . Тази реализация работи добре, ако сървърът изпълни receive() преди клиентът да извика send(). Ако send() се изпълни преди  receive(), получаващата ОС няма да знае кой от нейните процеси използва адреса, към който е адресирано новопристигналото съобщение и къде да го запише. Възможни решения на проблема са:

Отхвърляне на съобщението. Пристигналото съобщение не се приема;

Съхраняване на пристигналите съобщения в ядрото на ОС за кратко време;

Използване на буферирани примитиви. Менажирането на буферите се опростява чрез дефиниране на специална структура от данни, наречена пощенска кутия (mailbox);

Забрана на процес да изпраща съобщение, ако няма място за неговото приемане. Изпращащият процес се блокира, докато не се получи потвърждение, че изпратеното съобщение е получено.

**5.Архитектура на сървър**

Всеки сървър се изпълнява като спящ (sleeping) процес, който се активира при пристигане на заявка от клиент. Обикновено сървърният процес е безкраен процес. Според броя едновременно обслужвани клиенти, сървърите могат да бъдат реализирани като:

Последователни (sequential). Клиентите се обслужват последователно един по един. В този случай трябва да се организира опашка (mailbox), в която да се съхраняват постъпващите заявки от клиенти;

Паралелни (concurrent). Паралелните сървъри могат да обслужват едновременно множество клиенти. Те се състоят от един главен (master) процес и множество подчинени (slave) процеси . Главният процес е безкраен и се грижи за приемане на заявки от клиентите. Когато пристигне заявка, той създава нов процес за обслужването й. Пример за многопроцесна реализация е Web сървърът Apache за

**Unix и Linux.**

С паралелната реализация на сървърите

се постига висока ефективност на обслужване на клиентите. Недостатъкът се състои в значителната системна работа, необходима за създаване, унищожаване и превключване на процесите. Тя може да бъде съкратена, ако вместо подчинени процеси се използват нишки (threads). Работата на сървъра при организация с нишки е аналогична. Главната нишка приема потребителските заявки и за изпълнението на всяка заявка назначава подчинена нишка. Пример за многонишкови сървъри са Web сървърите IIS и Apache върху Windows платформа.

Сървърите могат да изпълняват клиентските заявки:

Самостоятелно. Обикновено сървърът извършва сам всички логически последователности от действия, за да изпълни клиентската заявка;

Взаимодействайки с други [сървъри](http://www.webdesign-bulgaria.com/webdesign/kakvo-e-server.html). В редица случаи сървърът не може самостоятелно да обслужи клиента, обикновено поради липса на достъп до нужната информация. В този случай той може да потърси съдействието на други

сървъри, изпращайки им заявки. По аналогичен начин работят DNS сървърите.

**Структура на директориите в Linux**

Файловата система на Linux е йерархична дървовидна структура от директории . Главната директория се нарича корен и се отбелязва със символа ‘/’. Тя съдържа списък на директориите от първо ниво, всяка от които съдържа списък на директории от второ ниво (поддиректории) и (или) файлове и т.н.

Предназначението на най-важните директории от първо ниво е следното:

/bin – съдържа най-важните команди на UNIX;

/sbin – разширява директорията /bin с команди, използвани предимно от най-привилегирования потребител;

/usr е корен на поддърво, директориите на което съдържат: изпълними файлове на команди (/usr/bin), стандартни библиотеки (/usr/lib), справочни файлове (/usr/man), команди за администриране и др.

/boot – съдържа файлове, използвани по време на начално зареждане на системата;

/etc – съдържа специфични за машината конфигурационни файлове;

/var. Поддървото на /var съдържа файлове, които се променят по време на работа на системата. Това са директории с опашки към различни устройства (например принтери) или програми (електронна поща, новини и др.), журнални файлове, временни файлове и т.н.;

/root – домашна директория на най-привилегирования потребител;

/dev – съдържа специални файлове за всички устройства на машината;

/home. Тази директория съдържа потребителските “домашни” (home) директории. Започвайки от своята home директория всеки потребител може да строи собствено дърво на директории, достъпно само за него;

/proc – поддървото на /proc се разполага в паметта, а не върху диска. Използва за съхраняване на информация за процесите и за системата, която се използва динамично по време на работа.

1.2 Спецификация на файл

За да може да се работи с даден файл, той трябва да бъде напълно определен. Пълната спецификация на файл включва:

<пътека>/име,където:

<пътека> – пътека до файла;

име – име на файла.

Пътеката задава местоположението на файла в дървото на директориите. Имената на директориите в пътеката се отделят едно от друго със символа ‘/’. Пътеката до файл може да бъде задавана по два различни начина:

Абсолютна пътека. Определя се от коренната директория. /home/nada/books

Относителна пътека. Относителната пътека се задава от текущата директория, като текущата директория се задава със знака ‘.’.:

./books

Друг начин за задаване на относителна пътека е по отношение на потребителската home директория, която в пътеката се заменя със знака ‘~’. ~/books

Името на файла се приема от интерпретатора като еднороден символен низ, без отделени по смисъл полета. В символния низ могат да се съдържат и редица специални символи, в това число и символът ‘.’ :

‘\*’ – заменя n произволни символа. В командите се използва за групиране на файлове;

‘?’ – заменя един произволен символ в името на файла;

[x-y] – задава област от символи, към която принадлежи символ от името на файла;

[a,b,c] – задава множество от символи, от което може да бъде символ в името на файла.

Например  
\*a?[m-x]\*.b[l,m,q]?

**Защита на файлове и директории**

В UNIX и Linux се използват три типа права на достъп до файл:

* Право за четене (read).  за преглеждане на файла;
* Право за запис (write). за промяна или изтриване на файла;
* Право за изпълнение (execution). Файлът да се стартира като програма;
* Когато даден файл се създаде, системата присвоява определени права за достъп до него по премълчаване.
* Правата за достъп до директориите имат различно значение по сравнение с файловете:
* Право за четене (read). Дава право за разпечатване на съдържанието;
* Право за запис (write). Дава право за добавяне и премахване на файлове;
* Право за изпълнение (execution). Това право позволява получаване на информация за файловете й. Директорията може да стане текуща работна директория.
* Според правото на собственост към определен файл потребителите се делят на три категории:
* Собственик (user). В правата за достъп се обозначава с u. Това е потребителят, създал файла.
* Група (group). Обозначава се с g. Групата се състои от колектив потребители.

**ТЕМА 1.2.**

**Архитектура на МОС**

1. **Ядро и спомагателни модули (двуслойна архитектура)**



**ядро** - модули, който използват основните функции

**модули** - който използват спомагателни функции

**1.1 Ядро – резидентно**

* управление на процесите;
* управление на паметта;
* управление на устойствата по В/И.
  1. **Вътрешно системни функции**
* превключване на контекста;
* обработка на прекъсванията;
* и други.
  1. **Функции API (Application Program Interface)**
* функции за поддържане на приложенията;
* четене на файл;
* търсене на файл;
* извеждане на информация на дисплея;
* и други.

1. **Спомагателните (транзитните) модули се разделят на следните групи**

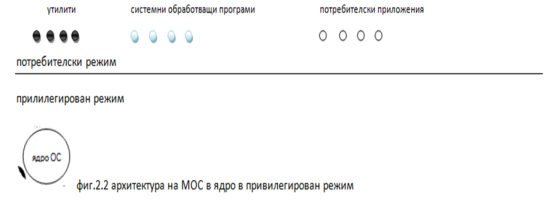
* Утилити – за архивиране на данните, инструменти за дефрагментация на диска;
* Системни обработващи програми – текстови и графични редактори, компилатори, транзистори, свързващи редактори и др.;
* Програми за допълнителни услуги на потребителите – калкулатори, игри и други;
* Библиотеки с процедури – математически функции и други.

Както всички приложения и спомагателните програми се обръщат към функциите на ядрото посредством системни извиквания.

Осигуряването на привилегии на ОС става възможно със специални средства за апаратна поддръжка. Апаратната част на КС е длъжна да поддържа минимум два режима на работа.

Потребителски режим (user mode)

Привилегирован режим (режим на ядрото kernel mode) или режим на супервайзора (supervisor mode)

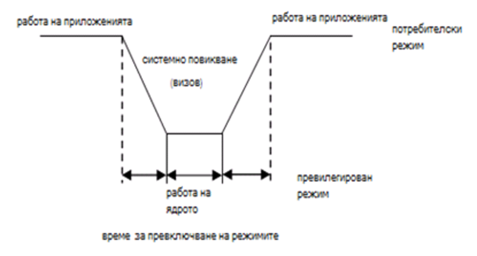


Функциите на ОС се контролират от ядрото и то определя условията и функциите, които са разрешени или безусловно забранени за потребителски режим.

Ако (апаратурата) хардуера на КС поддържат две или повече нива на привилегии, то ОС може да създаде по програмен път необходимата система за защита, да поддържа няколко нива на привилегии, като обработва йерархията. Това повишава устойчивостта на МОС.

На основата на двата режима на привилегии на процесора МОС може да изгради сложна система за защита на ресурсите (например за файлове и каталози). Такава система позволява да се определят (задават) за всеки потребител определени права за достъп до всички файлове и каталози.

Повишената устойчивост при обезпечаване на прехода на ядрото в привилегирован режим се постига за сметка на забавяне изпълнението на системните (извиквания) обръщения.



**Смяна на режима при изпълнение на системния (визов) към потребителски режим**

Архитектура на МОС, основата на привилегировано ядро може да се приеме за класическа. Такава е архитектурата на UNIX, VAX VMS, IBM OS/390, OS/2 и на Windows NT (определят модификация (различие).

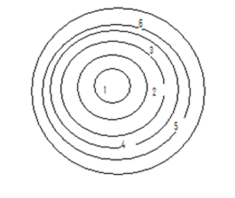
**3. Многослойна структура на МОС (МОС с многослойна структура)**



В съответствие с многослойния подход системните се състоят от периферни слоеве. Всеки слой, изпълнява набор функции. Те образуват междуслойния интерфейс. На основата на по-долния слой горния съдържа свои функции – по-сложни и по-мощни и т.н. строгите правила касаят взаимодействията между слоевете на системата. Между модулите е даден слой връзките могат да бъдат произволни. Даден модул може да изпълнява своите функции самостоятелно, или като се обръща към другите модули от своя слой, или към по-ниско стоящия слой чрез междуслойния интерфейс. Предимства на тази организация са:

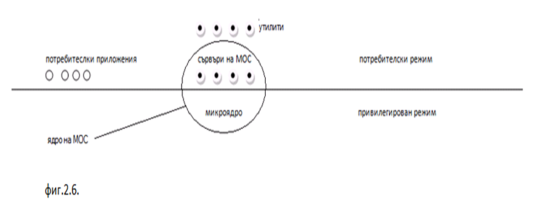
* лесно създаване;
* лесно разширение;
* лесна модификация.

1. **Многослойният подход се прилага и към структурата на ядрото МОС.**

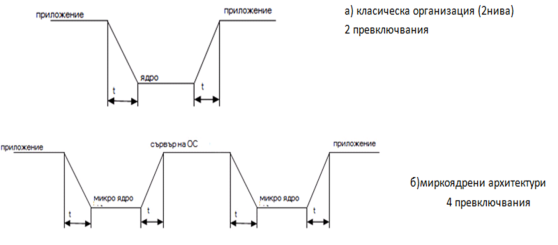


**Многослойна структура на ядрото на ОС**

1. хардуер.
2. средства за апаратно поддържане на ОС – тези, които участват в изчислителните операции, средствата за поддържане на привилегироване, средства за защита на паметта и други.
3. Машинно зависими елементи на ОС – образуван от програмните модули, които отразяват спецификата на апаратната платформа на компютъра.
4. Базови механизми на ядрото – изпълнява най-примитивните операции на ядрото, като диспециризация на прекъсванията, преместване на страници от паметта на диска и обратно и т.н. Модулите от този слой не участва в решения за разпределение на ресурсите.
5. Мениджър на ресурсите – състои се от модули, които реализират стратегически задачи за управление на основните ресурси на КС. Това са модули за мениджиране (диспечиране) на процесите, входа-изхода, файловата система и ОП.
6. Интерфейс на системните възовод – този слой изпълнява функциите API и представя достъп към системните ресурси в удобна и компактна форма, без значение на тяхното физическо разположение.
7. **Микроядренна архитектура**



Сигурността на микроядренната архитектура се състои в следното: в привилегирован режим остава да работи само малка част от ядрото на ОС наречени микроядро (фиг. 2.6.). То е защитено от останалите части на ОС и приложенията. В него влизат обикновено машинно-зависимите модули, а също и модулите изпълняващи базови функции (но не всички) на ядрото по управление на процесите, обработка на прекъсванията, управление на ВП, четене от ресурсите. Набора от функции на микроядрото обикновено съответства на функциите на слоя на базовите механизми на другите ядра т.е. функции, които не могат да се изпълняват в потребителски режим. Всички останали функции на ядрото от високите нива се оформят като приложения (сървъри) и се изпълняват в потребителски режим.



Най-често в потребителски режим се насочва файловата система, подсистемата за управление на ВП и процесите, мениджъра на безопасност и други. Мениджърите на ресурси, изнесени за изпълнение в потребителски режим се наричат сървъри на ОС.

Предимства и недостатъци на тази архитектура:

* Недостатъци – по-ниска производителност (тук има 4 привилегировани режима) фиг.2.7.;
* Предимства – голяма степен на преносимост; разширяемост; поддържане на механизъм за разпределени изчисления.

**ТЕМА 1.3.**

**Многослоен модел на системата ВХОД-ИЗХОД**

1. **Система за ВХОД-ИЗХОД**

В съвременните ОС функциите по обмен на данните между компютрите и периферните устройства се реализират от подсистемите за вход-изход. Потребители (клиенти) на тази подсистема са потребителите и приложенията, както и някои елементи на ОС като например подсистемата за управление на процесите. При смяна на състоянието на активния процес, следствие наложило се прекъсване на диска се записва състоянието (контекста) на прекъснатия процес и тази информация се чете (използва) при възстановяване (етикиране, пускане) на процес.

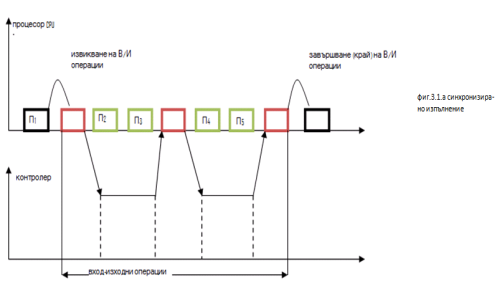
Основни елементи на подсистемите за вход-изход (В/И) са драйверите и файловите системи (ФС). Драйверите служат за управление на външните устройства. ФС може да се разглежда и като отделен елемент ОС в предвид на нейната сложност, специфичност и важност. Но тя може да се разглежда и съвместно с подсистемата за В/И тъй като в основата на повечето механизми за достъп до устройства.

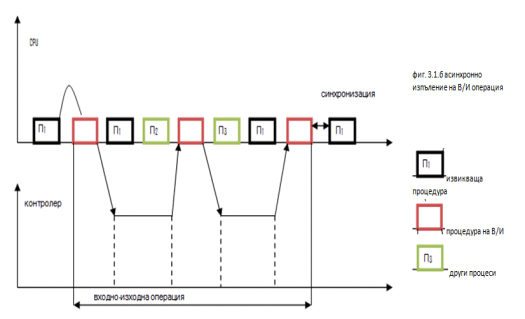
Към подсистемата за В/И може да се отнесе и диспечера на прекъсванията. Той не обслужва само тази подсистема, а и подсистемата за управление на процесите. Затова диспечера на прекъсванията може да се разгледа условно като част (елемент) от подсистемата за В/И.

Input – Output Subsystem – подсистеми за В/И

1. **Задачи на МОС свързани с управлението на файловете и устройствата:**

* Организация на паралелната работа на процесора;
* Съгласуване на скоростта на обмен на данните и кеширане на данните;
* Разпределение на устройствата и данните между процесите;
* Осигуряване на удобен логически интерфейс между устройствата и останалите елементи на системата;
* Поддържане на широк спектър от драйвери с възможност за включване и на нови в системата;
* Поддържане на няколко файлови системи – архитектурата на подсистемата вход-изход (ПВ/И) трябва да позволява лесно и просто включване в нейния състав на нови типове ФС без да е необходимо да се прекъсва кода на ОС. Обикновено в МОС имат специален слой програмно осигуряване което решава тази задача. Този слой най-често се нарича виртуална файлова система (Virtual File System VFS) (UNIX, System VRevuese 4).
* Поддържане на синхрони и асинхронни определени по вход/изход – операциите по В/И могат да се използват в синхронен и асинхронен режим на работа. При синхронния режим активния процес, програмният модел е (спрях, блокиран) през цялото време на изпълнение на В/И операция (фиг.3.1.а)





* При асинхронния режим неактивния процес (програмен модул) продължава да се използва в мултипрамен режим заедно с В/И операция (фиг.3.1.б).
* ПВ/И трябва да предоставя на своите клиенти (потребителските процеси и кода на ядрото) възможност за синхронизиране и асинхронизиране на В/И операции. Обикновено системите В/И операции, а вътрешните (например В кода на ядрото) като асинхронни.
* При МОС с микроядрена архитектура се използва асинхронно изпълнение на потребителските процеси.

**Многослоен модел на подсистемата за В/И. Обща схема.**

* Многослойният модел се базира на йерархичната структура на програмното осигуряване. Той позволява да се намери баланса между две възможни противоположни изисквания, а именно:
* Особеностите на всяко устройство е необходимо да бъдат отнети;
* Да се осигури единно логическо представяне на унифицирания интерфейс за устройствата от всички типове.
* Обобщената многослойна структура е представена на фиг.3.2. (рис.3.2., стр.262).
* Програмното осигуряване на ПВ/И се разделя на хоризонтални и вертикални слоеве. Вертикалните слоеве обединяват различни групи В/И устройства (например дискове, графични периферни устройства, мрежови устройства и други). Във всяка вертикална подсистема съществуат няколко хоризонтални слоя модули. В най-долния слой са драйверите, които управляватабстрактно съответните устройства. Те осъществяват обмен на байтове и/или блокове байтове. Не са свързани (не използват по-горните нива).
* Горните нива са свързани с логическата организация на данните (с файловете).

**Мениджър на В/И**

* В състава на модулите на ПВ/И има модули с универсално предназначение. Те организират работата на останалите елементи на системата, взаимодействат с другите подсистеми, както и с потребителските процеси. Тези функции, както и управлението на устройствата се реализират по всяко ниво и обратно обвивката наречена мениджър на В/И.
* Най-горния слой на ПВ/И мениджърът поддържат (потребителски (интерфейс на В/И))В/И потребителски интерфейс. той предлага на потребителите и програмите максимална употреба при работа с устройства и с разположените в тях данни.
* На най-долния слой мениджърът реализира взаимодействието с контролите на вътрешните устройства, екранират драйверите от особеностите на апаратната платформа на компютъра (шината на В/И, системните прекъсвания и други). Този слой прилича от драйверите въпросите(заявките) на обмен на данните от регистрите на контролера в опредената обобщена форма. След това ги преобразуват (въпросите, заявките) в зависеща от старата платформа формат. Самият диспечер на стартовата платформа формат. Самият диспечър на прекъсванията може да бъде част от мениджъра на В/И или да е отделен модул в ядрото на МОС.
* Важна функция на мениджърът по В/И е създаването на среда за взаимодействие между останалите модули, както и организация на взаимодействието на модулите по В/И с другите модули на МОС, като например подсистемата за управление на процеси, виртуалната памет и други.
* В/И мениджър в ОС UNIX се явява средата STREAMS, Windows NT пакет въпроси по В/И IRP( I/O Request Packet). Получените въпроси от процедурите за системна заявка се формулират като IRP и се предават на съответния драйвер. След изпълнение на операцията от устройството, драйвера връща отговор в други IRP пакет на мениджъра. Той от своя страна по необходимост може да го предаде на други драйвери и т.н. Мениджърът позволява да се задават взаимовръзките (bindings) между драйверите и на тази основа да се организират предаването на IRP пакетите.

1. **Драйвери на няколко нива (многонивови драйвери)**

Под драйвер, с тесния смисъл на думата, се разбират програмен модул, които:

* Влиза в състава на ядрото на МОС и работи в привилегирован режим;
* Непосредствено управлява външните устройства чрез взаимодействие с техните контролери;
* Обработва прекъсвания от контролера на устройството;
* Предоставя на приложните програми и потребителите удобен логически интерфейс за работа с устройството;
* Взаимодейства с други модули от ядрото на МОС по определени правила. Последните задават(описват) формата на предаваните данни, структурата на буферите, начините на включване на драйверите, наборът от процедури на подсистемата по В/И.

В някой МОС и в техните ПВ/И драйверите не са отделни в отделен слой. В развитието на ОС освен традиционните драйвери се появяват и драйверите на по-високите йерархични нива. Това съответства на многонивовата структура на подсистемата за В/И. Традиционните драйвери се наричат апаратни драйвери, драйвери на устройствата или ниско нивови драйвери.

Високонивовите драйвери са разположени над ранните и обикновено заемат от 2-ро до 5-то ниво. Те също изпълняват функции по управление от прекъсване както апаратните драйвери. Мениджърът по В/И управляват драйверите по един и същи начин независимо от нивото, на което са разположени.

Апаратните драйвери взаимодействат със системата за прекъсванията. Те се извикват по прекъсвания, а драйверите от горните нива се (извикват) по инициатива драйверите или на тези от по-ниско ниво. Драйверите имат определена структура, в която може да се отдели секция за обработка на прекъсванията (Interupt Service Routine, ISR). Тя се извиква при постъпване на въпрос от диспечъра на прекъсванията.

В ОС UNIX всички драйвери могат да се разделят на две основни групи:

* Блок ориентирани (block oriental) драйвери;
* Байт ориентирани (character oriental) драйвери.

Към вторият тип се отнасят драйверите на графичните устройства, както и тези на мрежовите устройства, а към първият тип се отнасят драйверите за управление на дисковете.

Блок ориентираните драйвери управляват устройствата с пряк достъп, на който се съхранява информация в блокове с фиксиран размер, всеки от който има собствен адрес. Адресуемостта на блоковете в устройствата с пряк достъп (каквито са дисковете) дават възможност за кеширане на данните в ОП.

Устройствата, които работят с байт-ориентирани драйвери не са адресуеми и не позволяват реализиране на операция по търсене на данни. Те генерират или използват байтови последователности. Такива устройства са терминалите, принтерите, мрежовите адаптери. Те също са устройства с последователен достъп.

* Блок или байт ориентацията е характеристика за всяко устройство, и за драйверите. Ако едно устройство не поддържа обмен на адресируеми блокови данни, а позволяват да се запишат или четат байтови последователности, то устройството и неговия драйвер са байт-ориентирани.
* Блок-ориентираните устройства могат да се управляват и от байт-ориентирани драйвери. Например дискйт може да се разглежда като набор блокове, така и като набор байтове. Физическият обмен с контролера на устройството преди всичко се осъществява от блокове, но байт-ориентираните драйвери на устройството преобразуват блока в последователност от байтове. Ето защо за устройствата с пряк достъп често се разработват двойка драйвери, които позволяват обръщението към него да се извършва и от байт-ориентираните, и по блок-ориентирани интерфейс в зависимост от необходимиостта.

**Специални файлове**

* В ОС UNIX всички устройства се разглеждат като виртуални (специални) файлове. Това позволява да се използва общ (един и същ) списък от операции по В/И за всяко устройство, независимо от неговата специфика.
* Специалните (виртуални) файлове не са свързани със статични данни, съхранени на диск, а от удобно унифицирано представяне на В/И устройства.
* Те представят устройствата като неструктурирана последователност от байтове. Към него са възможни операциите над останалите файлове (отваряне, затваряне, редактиране, четене, добавяне и т.н.). Може да се използват същите команди за работа с файлове като: create, open, read, write and close.
* Обикновено специалните файлове в UNIX са разположени в директорията (каталога) dev, но те могат да бъдат създадеи и записани и във всяка друга директория от ФС. При появата на ново устройство и съответно нов драйвер администраторът може да създаде нов запис с помощта на командата mkmod. Например, командата:
* mdmod (dev/dsk) sc4d3s 38 32 33 създава блок-ориентиран специален файл

**Входно изходна система на WINDOWS NT**

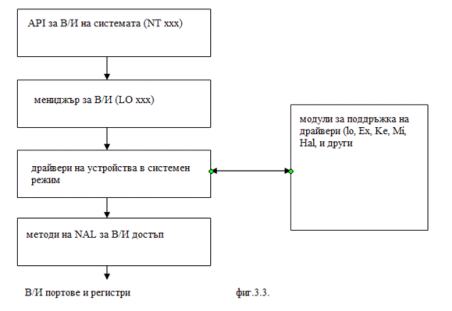
* Входно изходната система (В/ИС) приемат В/И заявки направени от системни и/или потребителски режим и ги предават в различна форма към В/И устройства. Те се намират във файлът NTOSKRNL.EXE и извършва функции, които са обичайни за, или изисквани от, повече от един драйвер.
* В/ИС дефинират модели на В/И обработка в WINDOWS NT.тя създава пакети от входно/изходни заявки и ги насочва към различни драйвери, връщат резултати на извиквания, когато вход.изходът завърши. Вътрешно В/ИС работи асинхронно, за да постигне висока производителност и предоставя на потребителите приложения синхронни и асинхронни В/И функции.
* Драйверите за устройствата не включват само апаратните драйвери, но и за файлова система, мрежови и слоести драйвери. Всички драйвери имат обща структура и комуникират един с друг и с мениджъра на В/И, чрез ползването на стандартни механизми.
* Драйверите могат да бъдат поставени е слоеве един над друг, за да се постигне модулност и да се намали дублирането между тях. Те могат да се пишат на езици от високо ниво, за да се намали времето за разработка и да се подобри преносимостта им. Трябва да бъдат проектирани така, че да работят правилно на мултипроцесорни (многопроцесорни) системи.

**Целите и задачите на В/И са следните:**

* Да се направи бърза В/И обработка за едно- и многопроцесорни системи;
* Да се защитават делими ресурси като използват стандартни механизми за защита на сигурността в WINDOWS;
* Да се удовлетворяват изискванията за В/И услугу, определени от MS WIN32, OS/2 и POSIX подсистемите;
* Да се позволи създаването (писането) на драйвери на езици от високо ниво;
* Да ползват динамично добавяне или махане от устройствата на системата;
* Да се поддържат няколко инсталируеми ФС – като FAT, CDFS, NTFS и други;
* Да се предостави възможност за съпоставяне на файловия вход/изход в паметта, за активиране на изпълними файлове, кеширане на файлове и използване от приложения

**Структура на В/И система**

* В WINDOWS NT програмите извършват В/И от виртуални файлове. Виртуален файл означава произволен източник или предназначение за В/И, който се приема (третира) като файл (сякаш е файл) (като файлове, папки, канали и mailslots). Всички данни, които се четат или записват се разглеждат като поток от байтове, насочени към тези виртуални файлове. Приложенията в потребителски режим викат документирани функции, за да четат от файл, да пишат във файл и извършват други операции. Мениджърът на В/И динамично насочва тези заявки за виртуални файлове към подходящия драйвер на устройствата.
* На фиг.3.3 е показана структурата на В/И система.



* Представлява набор от вътрешни системни услуги (като Nt ReadFile, NtWriteFile), с които се реализират документираните функции за В/И на подсистемата;
* Е отговорен за извършване на обработката на В/И заявки;
* Превръща В/И заявки специални управляващи заявки към хардуерните устройства;
* Се извикват от драйверите на устройствата за извършване на В/И им заявки;
* Методите на абстрактен хардуерен слой (HAL) за В/И достъп изолират драйверите на устройствата от разликите в хардуерната платформа. Позволяват изпълнимите им файлове да бъдат преносими в дадена архитектурна фамилия и програмните им файлове да бъдат преносими между хардуерните архитектури, поддържат се от Windows NT.

Мениджъра на В/И дефинира рамка или модел, в която В/И заявка се предава на драйверите към устройствата. В/И система се управлява от пакети (пакети от В/И заявки – IRP->I/O request packet), които пътуват от един В/И елемент към друг.

IRP е структура от данни, която контролира как В/И операция се обработва на всички етапи.

* WINDOWS NT поддържа няколко вида драйвери на устройствата:
* Драйвери на виртуални устройства (VDD – virtual device driver) – използва се, за да емулират 16 битови MS-DOS приложения. WINDOWS NT е напълно защитена ОС система тъй като приложенията работят в потребителски режим и нямат директен достъп до хардуера;
* Драйвери за екрана (монитора) и драйвери за принтера т.е. графични драйвери в системен режим;
* Драйвери за устройства в системен режим – само те директно контролират и осъществяват достъп до хардуерните устройства. Има няколко типа такива драйвери и това:
* Драйвери на хардуерните устройства;
* Типове драйвери – реализират В/И обработка за определен тип устройства;
* Драйвери за портове – реализират обработка на В/И заявки;
* Драйвери за мини порт – съпоставят същински В/И заявки към определен вид порт;
* Драйвери за файлови системи – приемат В/И заявки за файлове и си използват, като правят свои собствени подробни заявки към драйверите на физическите устройства;
* Филтриращи драйвери на ФС – прихващат В/И заявки, извършват допълнителна обработка и ги предават към драйверите от по- ниското ниво (например FIDISK.SYS дисков драйвер с корекция за грешките).
* стр. 426 структура на драйвер

**ТЕМА 1.4.**

**Мрежова файлова система. Функции и структура.**

1. **Цели и задачи на файловата система (ФС)**

ФС е част (модул, елемент) на МОС (OS), която включва:

* съвкупност от всички файлове на диска;
* набора структури от данни, които се използват за управление на файловете – каталог от файловете, дискриптор на файловете, таблици с разпределение на свободните и заетите пространства на диска;
* комплекти от системни програмни средства, с които се реализират различните операции над файловете.

ФС играе роля на посредник между потребителите и КС. Тя разпределя дисковата памет, поддържа именуването на файлове, отразява имената на файловете в съответните адреси във външната памет, осигурява достъп до данните, поддържа разделянето, защитата и възстановяването на фаловете.

Задачите на ФС зависят от вида на ОС. Най-малко от задачите решавани от ФС при еднопотребителските и едно задачни ОС. Това са:

* имената на файлове (предоставяне на привилегии за обработване на имената на файлове);
* програмен интерфейс за приложенията;
* логически модул на ФС и неговото съпоставяне (отразяване) на физическата организация на съхранение на данните;
* осигуряване на устойчивост на ФС апаратни и програмни грешки.

Като допълнителни задачи в мултипроцесорните ОС се явяват осигуряване на достъп до файловете на няколко процеси едновременно т.е. използване на файловете като общи, делими ресурси на ОС. Това налага необходимостта от синхронизация на действията, за да се гарантира обслужване на процесите за крайно време и използване на възможност за появата на ключ.

При многопотребителските ОС, каквито са всички МОС се налага решаването на още една задача като: защита на файловете на един потребител от нефунционирания достъп на други потребители.

1. **Мрежови файлове системи (МФС) (NFS)**

МФС е основен елемент, както на МОС така и на разпределените системи. В разпределените системи МФС е също разпределени. И в двата типа МОС основните функции на МФС са свързани със съхранение на програмите и данните и предоставяне на достъп до тях при необходимост. При РФС данните и програмите се съхраняват на един или няколко компютъра. Компютрите, които позволяват на потребителите да получават достъп до своите файлове се наричат файлови сървъри. Те обработват въпросите за четене или запис на файловете подадени на другите компютри в мрежата, които в случая се разглеждат като клиенти на ФС.

Файловите сървъри обикновено съдържат йерархически файлови системи, всички от който има основен каталог и каталози на останалите в по-ниски нива. В много МФС клиентския компютър може да присъедини и монтира тези ФС към своята локална ФС. При това монтираните ФС физически никога не се преместват. Те остават на сървъра.

От програмна гледна точка РФС е мрежова служба с типичната за тези служби структури (тема 1 и 3). Файловата служба включва програми – сървъри и програми – клиенти. Те си взаимодействат по мрежата с помощта на определени протоколи.

В КМ могат едновременно да работят няколко програмни файлове сървъри. Всеки от тях може да предлагат различни услуги.

Освен това на един компютър в мрежата могат да бъдат предоставени услуги от различни ФС на потребителите на мрежата.

Файловата служба в РФС, както в централизираната ФС има две функционални части:

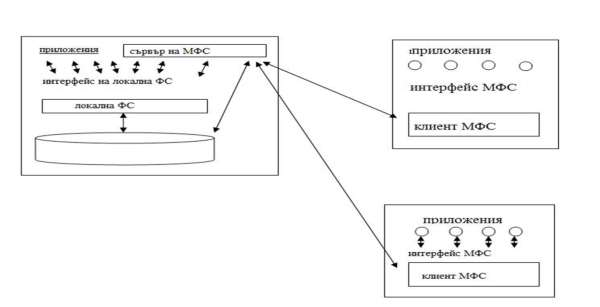
* собствена файлова служба;
* служба на каталозите на ФС (т.е. каталожната служба на ФС.

Първият е свързан с операциите нас файловете, а втората със създаването на каталозите и тяхното управление, с добавяне или отделяне на файлове от каталозите и т.н.

**Модул на МФС (NFS)**

МФС включва следните елементи: фиг.4.1.

Фиг. 4.1. модул на мрежова файлова система



* локална ФС;
* интерфейс на локалната ФС;
* сървър на ФС;
* клиент на МФС;
* интерфейс на МФС;
* протокол на клиент – сървър на МФС.

Клиентите на МФС са програми, които работят на компютрите включени в мрежата. Те обслужват заявките на приложенията за достъп до файловете, съхранени на отдалечени компютри. Клиента на МФС предава на мрежата въпросите към сървъра на МФС. Сървърът може да изпълни заявката самостоятелно или да е предаде (насочи) към локалната ФС за обработка. След получаване на отговор от локалната ФС сървърът го предава на клиента по мрежата, а той от своя страна го предава (препраща) на приложението.

Приложенията се обръщат към клиента на МФС чрез интерфейса на МФС.

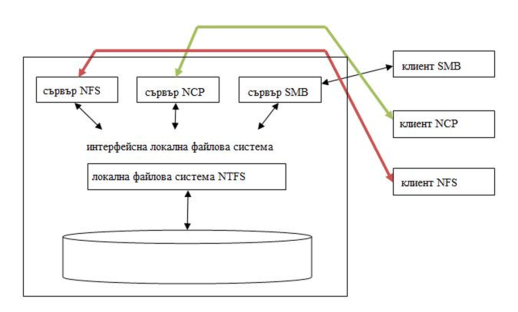
Клиентът и сървърът на МФС си взаимодействат по мрежата по определен протокол. Един такъв протокол е SMB (Server Message Block) – разработен от Microsoft, Inkl и IBM. SMB е клиент-сървър протокол. Неговата разширена версия за WINDOWS е CIFS (Common Internet File System). Тези протоколи се използват на приложеното ниво на OSI модела (OSI – Open System Interconnection, модел на взаимодействие на откритите системи).

SMB съобщенията могат да се предават с помощта на протоколите TSP/UDP и IPX.

Към една и съща локална ФС могат да се използват различни протоколи на МФС (фиг. 4.2.).

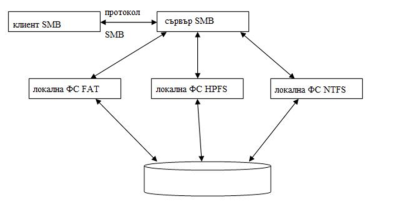
NCP (NetWare Control Protocol) – в OС NetWare на Novell

NFS (Network File System) – протокол на МФС на ОС UNIX, разработен от Sun Microsystems.



**фиг.4.2.**

Възможно е използването на един и същи протокол за достъп до различни локални файлови системи, например FAT, NTFS и HPFS. Тези файлови системи могат да са разположени на един или на няколко компютъра (фиг. 4.3.).



**Фиг. 4.3. достъп до различни файлови системи с един протокол клиент-сървър**

HPFS (High Performance File System)(High Performance) – високо производителна файлова система

NTFS е файлова система на W’NT (Windows NT File System)

В ОС UNIX най-разпространени са две МФС – FTP и NFS. FTP – File Transfer Protocol; NFS – Network File System.

Те използват собствени протоколи клиент-сървър и предоставят интерфейс на своите клиенти към ФС s5/ufs.

**Интерфейс на МФ служби**

Интерфейсът предоставя файловата служба на потребителите. Той не отразява нейното устройство. Съществуват няколко типа интерфейси, които се отличават по:

1) структурата на файловете - В МОС се поддържат различни типове логическа организация на файловете. Всеки тип може да има различни свойства. Файлът може да бъде организиран като последователност от записи, но може да бъде и като последователност от байтове т.е. да бъде не структурирана. Това налага МФ интефейс да позволява четене на всеки обмен от неструктурирания файл, както и на записи с определен формат за структурираните файлове.

2) модифицируемостта на файловете - Файловете в МОС могат да бъдат неизменяеми и изменяеми. Над неизменяемите файлове могат да се реализират само операциите create (създаване) и read (четене, прочитане). За тях много лесно се реализира кеширане, както и репликация.

3) семантиката на разделяне на файлове - Семантиката на разделяне на файловете се намаля поради използването им като общи, делими ресурси. Тя касае четенето и записването във файловете.

4) контролирането на достъпът до файловете - С всеки разделяем файл се нуждаете от контролни данните до него с цел осигуряване на защита от несанкциониран достъп. Това се задава в списък за контрол на достъпа ACL(Access Control List).

5) наличност на достъп до файлове (единица достъп).

**Реализация на МФС**

Начина на реализация на МФС обуславя нейната ефективност т.е. нейната производителност, отказоустойчивост и мащабируемост.

Въпросите свързани с реализацията на МФС се отнасят до:

* разместване (разпределение) на клиентите и сървърите по компютрите в мрежата и в ОС;
* кеширане;
* репликация.

В някой МОС като например Windows ‘NT Server (това е сървърна версия на ОС). Файловият сървър позволява обслужване едновременно на голям брой потребители, повече отколкото файловия сървър на клиентските компютрите т.е. в самата ОС за файловия сървър има отделни достъпни ресурси.

В други ОС файловия сървър е специален елемент на сървърната ОС, които не се записва на клиентските компютри. За повишаване на ефективността на работа на файловия сървър и клиент се явяват модули на ОС, които работят в привилигирован режим.

В съвременните МОС тези елементи са оформени като високонивови драйвери, работещи в състава на подсистемата (П В/И) за В/И. По-голямата ефективност при тази организация се постигат за сметка на прекия достъп, до всички вътрешни модули на ОС, в това число на дисковия кеш, без да е необходимо да се използват допълнителни операции и смяна на потребителския с привилегирован режим.

Файлов сървър и клиенти могат да се оформят и като модули, които работят в потребителски режим. Но обикновено работата на файловия сървър в потребителски режим понижава производителността на ОС.

**Файлови сървъри stateful и stateless**

Файловият сървър може да бъде реализиран по една от двете схеми:

* схеми stateful – със запомняне на данните за последователността на файловите операции зададени от клиента;
* схеми stateless – без запомняне на такива данни. Клиентът отправя запитване (заявка) към сървъра, сървърът я използва, връща отговор, а след това изтрива (отделя) от своите вътрешни таблици цялата информация свързана със запитването (заявката). Не се съхранява никаква текуща информация за клиента.

Ако такава информация се съхранява имаме схема – statefull.

Предимствата на тези сървъри са следните:

* Stateless
* отказоустойчивост;
* не е необходим извикване на open/close;
* по-малко памет е необходима за съхраняване на таблицата на отворените файлове;
* няма ограничение за броя на отворените файлове;
* отказва на клиента не създава проблеми на сървъра.

**Stateful**

* много кратки съобщения при въпроси;
* по-добра производителност;
* възможно е блокировка на файловете;
* възможно е четене.

**Кеширане**

Кеширането на данни в оперативната памет повишава скоростта на достъпа до файловете съхранени на диска, повишават мащабуруемостта и надеждността на ФС.

Кеширането в МФС налага решаването на 3 основни задачи, а именно:

* място за разполагане на кеша;
* начин, способи за разпространения на модификациите;
* проверка на достъпността на кеша.

1. Кеширането може да се отнася до цял файл или до части от него, обикновено тези към които имат обръщение.

В клиент- сървър системите имат потенциално три различни места за съхранение на кешираните файлове или на кешираните им части: ОП на сървъра, диска на клиента (ако има такъв) и ОП на клиента.

ОП на сървъра практически винаги се използва за намиране на файлове, към който има обръщения от потребителите и приложенията. Кеширането в паметта намалява времето за достъп до мрежата тъй като не се налага обмен с диска на сървъра.

Кеширането може да се реализира в клиентския компютър. Така се освобождава сървъра от работа по прехвърляне на файла от и към клиента. Използването (кеширането) на клиентския диск позволява кеширане на големи файлове. Дискът се явява по-надеждно устройство за съхранение на информация в сравнение с ОП. Този начин увеличава времето за достъп и е неприложим за компютри без дискове.

Кеширането в ОП на клиента (увеличават) ускорява достъпа до данните, но ограничават размера на кешираните данни съобразно различния обем в паметта.

1. Съществуването на няколко копия на един и същ файл на няколко места в КМ обуславя необходимостта от съгласуване на копията. Необходимо е модификацията на данните на едно от копията да бъде разпространено на всички останали копия. Това може да стане по няколко начина. От избрания начин зависи семантиката на разделяне на файловете.

Разпространението на модификацията решава проблема за съгласуване на основното копие на сървъра, с копията на клиентските компютри. Не дава информация за това кой трябва да си обнови данните в клиентския кеш. Ето това налага проверка дали данните в кеша са доставени. За целта е необходима проверка на кеша, която може да се направи по два подхода.

1)Иницииране на проверка на клиента;

2)Иницииране на проверка на сървъра.

При 1) клиентът се свързва със сървъра и проверява, съответстват ли данните в неговия кеш с данните, копието на файла на сървъра. При това той може да направи тази проверка:

* преди всяко обръщение към файла (всяко използване)
* периодическа проверка;
* проверка при отваряне на файла.

Проверка от тип 2) се използва при централизирано управление, когато даден файл се отваря (open), то клиента, който изпълнява това действие, изпраща съответстващо съобщение на файловия сървър, в което указва режима на отворения файл – четене или запис. Файловия сървър съхранява тази информация. Той позволява и на други клиенти да отварят даден файл, който е отворен вече като файл за четене и забранява – ако е отворен за запис. При затваряне на файла отново е необходимо да се информира файловия сървър. Така (той) последният обновява своите таблици. Този подход за проверка е много ефективен, но той налага файловия сървър да е от тип stateful. Освен това кодът на сървъра е нестандартен и достатъчно сложен. Необходимо е да се използва механизъм за иницииране на проверка за достоверността на клиента при отворени файлове.

**Репликация (тиражиране)**

Под репликация (replication) се разбира съществуването на няколко копия на един и същ файл, всеки от които се съхранява на отделен файлов сървър и съгласуването на данните в копията е осигурено автоматично. Репликацията се реализира от отделна служба на ОС. С нея се увеличава надеждността (има независими копия на всеки файл съхранен на различни файлови сървъри) и се разпределя натоварването между няколко сървъра (клиентите могат да се обръщат към репликирания файл на най-близкия до тях сървър, на който има копие).

Репликите на файла се съхраняват на файловите сървъри и са достъпни на всички клиенти. МОС (ОС) осигурява достоверност на данните в репликите както и защитата им.

Прозрачността на репликациите зависи от два фактора:

1) използваните схеми за именуване на репликите;

2) степента на участие на потребителите в управлението на репликацията на файловете.

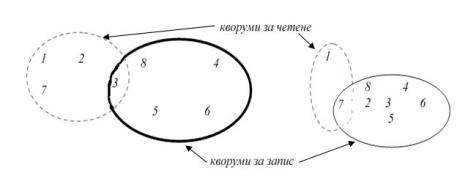
Съществуват и два режима на управление на репликациите.

1) Явна репликация (explicit replication) – при нея на потребителите (приложените програмисти) се дава възможност за управление на процеса на репликация.

2) Неявна репликация (implicit replication) – при нея изборът на броя и местата на репликите се определя автоматично, без участието на потребителите от самата ФС.

Необходимо е да има съгласуване на репликите за да се гарантира съхраняване на последните данни във всяка реплика. Съгласуването на репликите може да се реализира по няколко начина (способа):

* Четенето на произволни – записи на всеки (read any write all) – четенето се реализира от произволно копие. Недостатък: не винаги във всички сървъри могат да се реализират записи. Някой от нях може да е неработоспособен.
* Запис в достъпно копие (available copies) – четенето се реализира също от произволно копие.
* Първична реплика (primary copy) – запис се разрешава само в една реплика на файла, наречена първична (primary). Всички останали реплики се явяват вторични (secondary) и от тях могат само да се четат данни. Недостатък на метода е ниската надежност (при отказ на първия сървър модифицирането на файла става невъзможно).
* Кворум (anorum) – така в мрежата има n на брой реплики на някой файл. Алгоритъмът се базира на това, че при модификацията на файла измененията се записват в w реплики, а при четене на файл клиентът се обръща към r реплики. При това w + r > n. Ако това условие е изпълнено, то в една от всяка w реплики ще се намери последния запис (последното обновление).



фиг. 4.4. илюстрация на работата по метода на кворума

Ако n=8, r-4, w=5 (w+r>n; 5+4>8) – условието е удовлетворено. Ако е направен за реплика 3,4,5,6,8, а четенето за 1,2,3,7, то репликата 3 се явява обща за четене и писане. Така, че в репликата 3 се намират коректните данни.

Това се отнася и за случая, когато r=2, w=7, 2+7>8 и в реплика 7, която е обща се намират последните коректни данни.

Ако r=1, w=n, то този метод се преобразува в (Read Any Write All) четене на произволен- запис във всеки.

Файлова система NFS(Network File System)

МФС-NFS

ФС NFS е създадена от Sun Microsystems. Тя е стандартна МФС за ОС UNIX. Поддържат отдалечен достъп до файловете. Позволява поддържането на ФС на сървъра да се реализира като поддърво на локалната файлова система.

Основната идея на NFS е да позволи на произволна група потребители да раздели обща файлова система. NFS може да се реализира и в глобалната и в локалната КМ. Всеки NFS сървър не съхранява данни за отворените файлове т.е. тя е от типа stateless.

В работата си NFS използва основно два протокола. Първият NFS протокол управлява монтирането. А вторият протокол – за достъп до отдалечени файлове и каталози.

В NFS не се поддържа репликация и използва кеширане на клиента. Данните в кеша се пренасят поблоково.

**Файлова система NTFS (Windows NT File System)**

NTFS се характеризира с възобновяямост, сигурност, дублиране на данни, устойчивост при грешки, поддържане на големи дискове (носители) и файлове, отколкото HPFS (High Performance FS е разработен за ОС OS/2 и може да поддържа дискове до 2ТВ (терабайта) или приблизително 2 трилиона байта).

NTFS предоставя възобновяване на файловата система, базирана на модела на транзикционна обработка. Транзикционна обработка е техника е техника за управление на модификациите в БД така, че системните сривове да не влияят на верността или целостта на базата от данни. Някои операции върху БД, наречени транзакции, се извършват на принципа всичко или нищо.

Транзакцията се дефинира като В/И операция, която променя данните на ФС или модифицира структурата на томът. Ако срив на системата прекъсне транзакцията частта, която не е завършила, трябва да се откаже или върне обратно (rollback). Операцията по отказване връща БД в предишното устойчиво състояние, сякаш транзакцията никога не се е извършвала.

Сигурността в NTFS директно произлиза от обектния модел на Windows NT. Отворен файл се реализира като файлов обект с дескриптор на сигурността, съхранен на диска като част от файла. Преди процеса да може да отвори манипулатор към файлов обект системата за сигурност на ОС проверява дали процесът има съответните права за това.

Дублирането на данни в NTFS се използва да се осигури предварително ниво на защитата им. А това възстановяване от грешки се използва слоестия драйвер модел на Windows NT. NTFS комуникира с дисков драйвер, устойчив на грешки, които от своя страна, комуникира с драйвер на твър диск за запис на данните върху диска.

Има възможност драйверът да отрази, или дублира, данните от един диск на друг така, че излишното копие винаги да може да извлече. Това е така наречената поддръжка RAID ниво 1. Данните могат да се записват на части между три или повече диска, използвайки ги като един диск, поддържащ информация за възстановяване. Ако данните от един диск се загубят или станат недостъпни може да пресъздаде съдържанието на диска чрез exclusive – OR(изключващо или) операции. Тази поддръжка се нарича RAID ниво 5.

P.S. FAT използва 16-битова таблица за запис на разпределението на дисков том. Томът се разделя на равни по размер единици наречени клъстери. Всеки клъстер е с уникален 16 битов No, FAT може да поддържа максимум 216 или 65536 клъстери на том. Един FAT том е ограничен да съдържа 65 518 файла, независимо от размера на диска.

NTFS заделя клъстери и използва 64 бита, за да ги номерира. Възможни са 264 клъстера, всички до 64КВ. Всеки файл може да е 264 байта дълъг. Размерът на клъстера е променлив. Обикновено варира от 512 байта за малки дискове, до 64 КВ за големи дискове.

В NTFS всяка информация, асоциирана с файл е реализирана като файлов атрибут (обектен атрибут). Всеки атрибут се състои от 1 поток т.е. от проста последователност от байтове. Това прави лесно добавяне на нови и повече атрибути (т.е. на повече потоци) към файл.

NTFS файловете (и папки) могат да съдържат множество потоци от данни. Всеки поток има различен размер (колко дисково пространство е заделено за него), действителен размер (колко байтове е използвал извикващия) и и дължината на валидните данни (каква част от блока е била инициализирана). Всеки поток може да се заключва отделно т.е. може да заключи обмен от байтове и да се разреши едновременен достъп до тях. Поделянето в NTFS се прави за всеки файл, а не за всеки поток с цел да се намали излишната обработка.

NTFS е Unicode съвместимост т.е. тя позволява да се използват Unicode символи за съхраняване на имена на файлове, папки и томове. Unicode е 16-битова схема за кодиране на символи. Тя позволява всеки символ да има единствено предимство във всеки от основните човешки езици.

Всяка папка или име на файл може да има до 255 символа дължина, и може да съдържа Unicode символи, интервали и няколко многоточия.

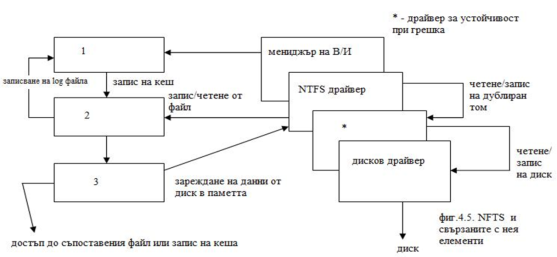
NTFS позволява индексиране и сортиране на имена на файлове, както и индексиране на файловите атрибути на дисковия том (ФС FAT индексира имена на файловете, но не ги сортират).

NTFS прави (позволява) динамично преадресиране на лоши клъстери. Заделя нов клъстер, като замества клъстера, в който се намира лошия сектор и кешира данните в новия клъстер. Маркира лошия клъстер и вече не го използва.

NTFS поддържа стандартна POSIX 1 т.е. поддържа case-sensitive (различаващи големи и малки букви) имена на файлове и папки, „време на промяна на файла”, (различно от „време на последна промяна”) и „твърди” връзки (множество елементи на папки, които сочат към един и същ файл).

Вътрешна структура на NTFS

В/И система, NTFS и другите ФС са зареждащи се драйвери на устройствата, които се използват в системен режим (фиг.9-1, стр.509 за NT). Те се извикват косвено от приложенията



NTFS драйвера взаимодейства с три други вътрешни NT елемента (фиг.4.5.). Това са:

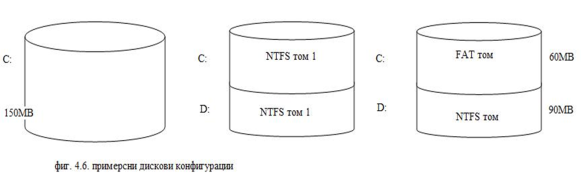
1) Услугата за log файл (LFS – Log File Service) – част от NTFS, която предоставя услуги за поддържане на списък с модификациите на диска. Log-файлът се използва за възстановяване на форматиран том в случай на системен срив;

2) Мениджър на кеша – предоставя системни услуги за кеширане на NTFS и драйвери за други ФС;

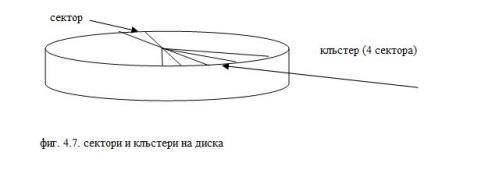
3) Мениджър на виртуална памет.

Дискова структура на NTFS

Тя описва томовете, клъстерите, файловете, папките, как се съхраняват файловите данни и атрибути на диска и как работи компресирането на данни в NTFS.



Структурата на NTFS започва с том. Том съответства на логическа част от диска и се създава, когато се формира диска или част от диск. Дискът може да има един или няколко тома (фиг. 4.6.).



Томът се състои от серия файлове и допълнително нечетно място на дисковия дял.

NTFS използва клъстера като основни единици при заделяне на области от диска. Размерът на клъстера на диска има клъстерния фактор, се установява, при формиране на тома. Той е различен при различен размер на тома, но винаги е няколко физически сектора, степен на 2 (1,2,4,8 и т.н. сектора). (фиг.4.7.). клъстерния фактор се изразява като брой байтове в клъстер, като 512 байта,1КВ, 2КВ.

Вътрешно NTFS се образува към клъстера и не зависи от размера на дисковия сектор. NTFS използва клъстера като единица при заделяне на памет, за да не зависи от размера на физическия сектор.

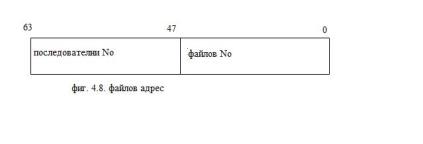
NTFS адресира физически позиции на диска чрез логически номер на клъстер (LCN – logical duster number). LCNs са номерата на всички клъстери от началото до края на тома.

NTFS адресира данните във файла чрез виртуални клъстерни номера (VCNs- virtual cluster numbers). VCNs номерират клъстерите, принадлежащи на отделен файл от 0 до m.

В NTFS всички данни, съхранени в тома, се съдържат във файлове. Основен елемент е структурата на NTFS тома е главната файлова таблица (MFT – Master File table). Тя е реализирана като масив от файлови запаси. Размерът на всеки файлов запис е фиксиран на 1КВ, независимо от размера на клъстера. Логически MFT съдържа един чрез за всеки файл в тома, включително и ред за самата главна таблица (стр.515 фиг.9.6.).

В NTFS файлът се идентифицира от 64-битова стойност т.е. от файловия адрес. Файловият адрес се систои от ф.No и пореден No.

Ф No се отнася за местоположения на записа на този файл в MFT-1. Последователният Nо позволява NTFS да извършва вътрешни проверки за устойчивост. Той се увеличава всеки път, когато файловия запис в MFS се използва отново.



NTFS съхранява файловете като група от двойки атрибут/стойност. Това са име на файла, дескриптор на сигурността, дата и час на създаване и др. Всеки файлов атрибут се съхранява като отделен поток от файлове във файл. NTFS разпознава атрибутите по имената им (пишат се с големи букви) и се предхождат от знакът $ (например $FILENAME, $DATA).

NTFS поддържат компресиране на основата на единични файлове, папки или томове. Компресиране се реализира за разпръснати данни (файлове) и за неразпръснати.

Под разпръснати файлове се разбират големи файлове, които съдържат малко количество, спрямо техния размер, ненулеви данни. Например разпръснати матраци, съхранена на диск e пример за разпръснат файл (стр.531).

**ТЕМА 1.5.**

**Управление на информацията в МОС**

1. **Транзакция** – входно-изходна операция, която променя данните на файловата система или модифицира структурата на тома.

Ако срив на системата прекъсне транзакцията, частта която не е завършена трябва да се откаже или да се върне обратно. Операцията по отказване връща предишното устойчиво състояние.

1. **Мрежова файлова система** **МФС**

МФС описва томовете, клъстерите, папките и файловете, това как се съхраняват файловите данни и как работи компресирането на данни.

Структурата започва със том (Volume) – логическа част от диска и се създава, когато се форматира диска или част от диска. Томът се състои от файлове и допълнително незаето място на дисковия дял.

Мрежовата файлова система NTFS използва **клъстери** като основни единици при заделяне на области от диска. Размера на клъстера се нарича клъстерен фактор и се установява при форматиране на диска. Той е различен при различните размери на тома, но винаги има няколко сектора, които по брой са степени на 2.

Фактора се изразява в брой байтове.

За NTFS те могат да биват 512В, 1024В, 2048В, **4096В (по подразбиране)**, 8192В, 16КВ, 32КВ, 64 КВ

Вътрешно ФС се обръща към клъстера и не зависи от размера на дисковия сектор. ФС използва клъстера като единица при заделяне на памет. За да не зависи от размера на физическия сектор ФС адресира физическите позиции на диска чрез логическия номер на клъстера – LCN (Logical Cluster Number).

ФС адресира данните във файл, чрез виртуални клъстерни номера (VCN). С тях се намират клъстерите принадлежащи на отделен файл от 0 до N.

Основен елемент при клъстера е файловата таблица. Реализирана като масив от файлови записи, размерът на такъв запис е фиксиран на 1 КВ независимо от размера на клъстера.

Файловата система поддържа стандарт на POSIX за имена на файловете и папките. Освен името на файла се съдържа и информация за неговия размер.

Поддържа се време на промяна на файла, различно от времето за последна промяна, времето когато файла е бил достъпен за последно и „твърди връзки“, които сочат към един и същи файл.

Входно/изходната подсистема в мрежовата ФС и други файлови системи се зарежда заедно с драйверите на устройствата и се използва в системен режим. Те се извикват косвено от приложенията.

Мрежовите ФС взаимодействат с други 3 елемента:

* Мениджър на кеша;
* Мениджър на виртуалната памет;
* Услугата за log файл.

Тя предоставя услуги за поддръжка на списък с модификатори на диска. Мениджъра на кеша предоставя системни услуги за кеширане на ФС и драйверите за другите ФС-и.

**ТЕМА 1.6.**

**Защита на информацията в МОС**

Има 2 групи проблеми свързани със защитата и безопасността на данните в компютърните мрежи и системи:

1. Безопасност на компютърната система – когато КС се разглежда като автономна такава.
2. Мрежова безопасност.

За по-голяма защита е удачно използването на безопасна информационна система, която има следните характеристики:

* Защита на данните от несанкциониран достъп;
* Съхранение на информацията и организация на достъпа до нея;
* Конфиденциалност;
* Достъпност;
* Цялостност (интегрираност).

Видове опасности:

* Неумишлени;
* Умишлени.

За осигуряване на безопасност е необходимо да се използва системния подход и да се използват всички средства за защита, както и основни базови принципи и политики за изграждане (осигуряване) на безопасността. Такива са:

* Минимално ниво на привилегии на потребителите;
* Използване на многонивова защита (multilevel);
* Единен контрол на ниво пропуск и вход на информацията;
* Баланс на натоварването и други.

Най-използваните принципи – забрана за всичко, което не е разрешено в явна форма и разрешаване на всичко, което не е забранено в явна форма.

За безопасността се използват базови техногогии:

* Автентикация;
* Оторизация;
* Одит;
* Криптиране.

**Модул II: Управление на процеси**

**ТЕМА 2.1.**

**Процеси и нишки в МОС.**

1. В режим на многозадачна работа процесора, паметта и другите ресурси се разпределят на типове единици за работа – като такива са приети процесът и нишката. В някои ОС-и понятието процес се заменя със задача (task), а вместо нишка – поток.

Обединяващо понятие е терминът задача.

Процесът е изпълнима програма или част от програма по време на изпълнение, за което е заделено процесорно време. Нишката единица от процес, която се разглежда като изпълнима единица и има следните елементи:

* Уникален идентификатор ID;
* Собствена област на паметта;
* 2 стека (един за изпълнение в системен режим и един за потребителски режим);
* Към нишката се отнася съдържанието на определени регистри представящи състоянието на процесора.

Процесът представлява множество от ресурси, резервирани за нишките, които изпълняват програмата. Процесът се състои от:

* Уникален идентификатор ID;
* Собствено виртуално адресно пространство;
* Изпълнима програма;
* Системни ресурси (семафори, комуникационни портове, файлове и други).

Програмите се разглеждат като статична последователност от инструкции. Нишката се идентифицира по стек и контекст. Контекстът на нишката са регистрите, стековете и собствената област от паметта. От своя страна нишките се състоят от подмножества наречени „леки“ нишки или влакна.

**Период (квантум)** – времето, за което нишката използва процесора еднократно.

**Порт** – механизъм за предаване на съобщения между процесите.

**Семафори** – използват се за броене на ресурси и за разпределяне на ресурси (забраняване или разрешаване).

**Mutex** – механизъм за сериализиране на достъпа до процесите.

**Профил на процесите** – механизъм за измерване на времето за изпълнение на процес.

Разделянето на процесите на влакна и нишки е обусловено от новите технологии, заложени в мрежовите ОС. То има за цел паралелното разпределяне на последователност от команди, с което да се увеличи производителността на ОС. ОС планира и диспечира изпълнението на нишките и процесите, планирането е обвързано с определена моментна нишка и избора на нова нишка за изпълнение. Диспечеризацията е свързана с реализирането на приетата стратегия за работа и тя предоставя процесора от една нишка на друга в зависимост от избраната стратегия.

Нишката може да се намира в 1 от следните състояния:

* Изпълнение – активно състояние;
* Готовност – пасивно състояние;
* Очакване – пасивно състояние.

В „изпълнение“ нишката използва всички активни ресурси както и процесора.

В „готовност“ – може да използва процесора, но той е зает.

В „очакване“ – пасивно състояние, при което нишката е блокирана от други нишки, освобождаване на ресурс, завършване на В/И операция и др.

1. **Multitasking (Многозадачност) –** свързана с увеличаване на общата производителност. Възможността за работа на няколко потребители, както и на работа на един потребител с няколко приложения по едно и също време.

ОС е реактивна, тя реагира на кратки интервали от време м/у периода от стартиране, до получаване на резултата.

Системите (МОС) в зависимост от изброените критерии за ефективност се разделят на:

* Системи за пакетна обработка;
* Системи за времеделение;
* Системи за реално време.

Всяка група има специфични вътрешни механизми, различни начини на обработка.

Мултипрограмиране в системите за пакетна обработка – използват се за изчислителни задачи, където не е определящо бързото получаване на резултат. Основният критерий при тях е решаване на по-голям брой задачи за единица време. Те функционират по следния начин: първоначално се формират пакетите от задания. Всеки един от тях съдържа краен брой временно изпълнявани задачи, както и изисквания към системните ресурси. Превключването на процесора е по инициатива на активната задача. Тези системи гарантират по-висока ефективност на работата на мрежите, но по-малка ефективност при работата с потребители

Мултипрограмиране в системите за времеделение – при тези ОС-и на потребителите се предоставя възможност за интерактивна работа с няколко приложения. Всяко приложение получава процесора за определено време. След изтичане на времето (кванта) процесите са длъжни да освободят процесора и той се предоставя на следващият от опашката с процеси. Тези системи имат по-малка пропусквателна способност, но при тях критерият е удобството на потребителите. Превключването на процесора става от ОС, а не от активната задача.

Мултипрограмиране в системите за реално време –тези ОС-и се използват, когато КС-и и/или мрежите се използват за управление на различни обекти (технологични процеси). За тях определящо е времето, в рамките на което трябва да се изпълни дадена нишка или процес. Критерий за ефективност на тези системи в времето за реакция при тях предварително е ясно, кое приложение, кой процес или нишка ще получат процесора. Изпълнението се реализира в режим на прекъсване.

**ТЕМА 2.2.**

**Алгоритми за планиране на процеси и нишки в МОС.**

Планирането е свързано с решаване на 2 основни задачи:

* Определяне на момента на смяна на текущия процес/нишка
* избор на процес/нишка от опашката на готовите;

Това зависи от избраната стратегия за управление.

Планирането може да се реализира статично и динамично. При динамичното решенията се взимат в хода на работа на системата на базата на анализ на текущото състояние. МОС работят в условия на неопределеност. Процесите и нишките се създават в необичайни/случайни моменти и това планиране е заложено в универсалните системи. В специализираните системи се използва статично планиране – последователността и времето за изпълнение на всяка задача е предварително известно.

Диспечеризацията на процесите е свързана с реализацията на приетото решение от планирането. Тя се свежда до следните основни действия

* съхранение на контекста на новия процес/нишка;
* избор на контекст на новия процес/нишка;
* пускане за изпълнение на новия процес/нишка.

Контекста на процес отразява състоянието на апаратната част на компютърната система в момента на прекъсване, брояча на команди, съдържанието на регистрите с общо предназначение, режима на работа на процесора, маските за прекъсване и др.

Повечето МОС в ядрото на ОС са вградени модулите за планиране и диспечиране на процесите. Алгоритмите за планиране се базират на две основни стратегии:

1. без превключване;
2. с превключване.

Без превключване – отнемането и предаването на процесора става по инициатива на процеса.

С превключване – по инициатива на ОС.

Принципна разлика се явява централизацията. От своя страна алгоритмите за диспечиране и планиране могат да бъдат разделени на групи в зависимост от това, какво е значимо за тях:

* алгоритми основани на квантуване;
* алгоритми основани на приоритета;
* алгоритми от смесен тип.

Алгоритми основани на квантиране – те са базирани на превключването. При тях смяната на активният процес/нишка се извършва ако процеса в завършил. Изчерпал е кванта процес време, има грешка при изпълнението, процесът е блокиран, нишката е в състояние на готовност и изчаква да му бъде предоставен друг квант процес време. Квантът може да бъде еднакъв или различен за отделните процеси.

Алгоритми основани на приоритета - алгоритмите се обслужват съобразно техните приоритети. Обикновено те са числа, които характеризират степен на привилигированост на процесите при използване на ресурсите на компютъра и мрежата в частност на процесорното време. Той се назначава от ОС при тяхното създаване, включва се като част от данните на блока за управление на процеса и е определящ за нишките. Зависи от това какъв е процеса, какъв е статуса на родителя на процеса, ако процеса е дъщерен. Приоритетите могат да бъдат статични и динамични. Статичните не се променят до края на изпълнението на процеса.

Алгоритми от смесен тип – те използват и двата механизма, размера на кванта се определя от приоритета на процеса.

В UNIX-базираните ОС-и планирането се организира само на нивопроцеси. Всеки процес се отнася към един от трите приоритетни класа:

1) процеси за реално време.

2) системни процеси – използват фиксирани приоритети.

3) процеси с режим на времеделение.

Системни процеси се определят от ядрото на ОС и не се променят във времето.

Процеси за реално време – също използват фиксирани приоритети, но потребителите могат да ги променят.

Процеси с режим на времеделение – използват динамични приоритети, които се определят от 2 съставящи: потребителска и системна. Потребителската част е приоритетна. Потребителите могат да променят приоритета на процеса. Нишките на даден процес се обслужват в цикличен ред.

Синхронизацията на процесите и нишките се реализира чрез семафори, критични секции, мутекси, събития, таймери. Тя е необходима, за да се изключи възможността за появата на клинч и постоянно отлагане при използването на процесите. В качеството на синхронизиращи обекти могат да се използват файловете, процесите и нишките.

Други обекти могат да бъдат събитията, мутексите, семафорите и други. Обикновено семафорите и мутексите се използват за управление на достъпа на данни, а събитията да покажат на други процеси, че определени действия са завършили, в резултат на което ОС деблокира и/или активира процесите.

**ТЕМА 2.3.**

**Взаимодействие на процеси.**

**Методи и алгоритми за реализация.**

Основните взаимодействия са:

* + взаимоизключване;
  + синхронизация.

**Взаимоизключване** се налага от факта, че ресурсите са винаги по-малко от процесите, а някои от ресурсите – паметта и процесора са използват от всички процеси.

**Синхронизацията** е свързана с възможността за нежелани състояния като безкрайно отлагане и мъртва хватка. Комуникацията между процесите е обусловена от комуникацията между потребителите, както и между взаимодействащите си процеси. Тя също така е обусловена и между коопериращите се процеси.

Проблемите на междупроцесните взаимодействия са свързани основно с предаване на информация от един процес на друг процес, съгласуване на дейността на процесите, както и управление на процесите. Управлението на тези взаимодействия се реализира най-често апаратно, програмно и/или със средствата на ОС. В зависимост от това, кое е определящо се говори за HW, SW подход и чрез средствата на ОС.

1. Взаимоизключване на процеси – физическите и логическите ресурси в КС и мрежи могат да бъдат разделяни – обикновено в даден момент само 1 процес има достъп до даден ресурс – например процесора. Такъв ресурс е прието да се нарича критичен. Ако няколко процеса имат заявка за използване на критичен ресурс се налага синхронизация на действията, при което е необходимо да се гарантира следното:

* Само 1 процес да може да използва ресурса в определен момент;
* Ако няколко процеса едновременно желаят да ползват ресурса, то той трябва да бъде предоставен на всеки един от тях за крайно време;
* Ако един процес получи ресурс, той трябва да го освободи в крайно време.

Един от начините за организиране на процеса на взаимодействие е използването на критични секции (сесии). Казваме,че един процес е в това състояние за времето, когато използва критичен ресурс. При взаимоизключване е необходимо само 1 процес да се намира в критична сесия в даден момент. Тя се характеризира с крайно време. Ако 1 процес иска да влезе в критична сесия, трябва да му бъде дадена тази възможност в крайно време. Това междупроцесно взаимодействие може да се делегира от явно, като се използват машиннозависими езици като асемблер (HW подход) или чрез машиннонезависими езици (SW подход).

Най-често това се прави с блокировка на паметта, например операциите четене и запис в дадена област са взаимоизключващи.

Известно е, че ако един процес иска запис на данни в една реплика, другите чакат и не им е разрешена операция четене докато не завърши успешно репликацията. Операциите четене и запис са критични за кратки интервали.

Взаимоизключването може да се разглежда и по отношение на процесора – например машинна команда не се прекъсва по време на изпълнение.

При използването на софтуерен подход, както и при използването на средствата на ОС междупроцесните взаимодействия се разглеждат между два процеса, както и взаимоизключване на N процеса. Процесите могат да се разглеждат като безкраен цикъл, а влизането в критична секция се управлява от оператора while/do.

От критичните секции се излиза, когато на променливата се присвои номер, равен на друг процес. Това решение не е приемливо за независими процеси, тъй като е нарушено третото условие. За неговото ненарушаване е необходимо процесите стриктно да се редуват при влизането в критична секция, но ако двата процеса са различна продължителност пак не е удачно решение. В този случай е удачно използването на една променлива и в случая се помни само, кой процес може да влезе в критична сесия, но не и състоянието на всеки от процесите. това решение се нарича – алгоритъм с твърда синхронизация.

Основният проблем при този алгоритъм, е че зависи критично от точното съгласуване във времето. Той налага процес, който не се намира в критична сесия да е в състояние на активно очакване до момента, в който да продължи изпълнението си. Това се отстранява като за всеки процес се назначава флаг и той се установява преди да влезе процеса в цикъл на очакване. Така обаче не се гарантира безкрайно отлагане, но има възможност за клинч. Възможно е двата процеса на вдигнат флаговете си преди да влязат в цикъл. За да се изключи тази възможност, периодично всеки процес нулира флага си и така се изключва възможността за клинч. При алгоритмите за взаимоизключване са работили по методите на Питерсън, Декер и др.

При взаимоизключването на N процеса обикновено се използва алгоритъм на Дейкстра, по късно той е усъвършенстван от Кнут. Тук приложение намират и алгоритми за разпределяне на ресурси, които гарантират надеждното състояния на системата – това е състояние на ОС, при което тя гарантира обслужването на всички процеси за крайно време. Такива алгоритми са алгоритъма на хлебаря, алгоритъма на банкера (в ролята на мениджър влиза ядрото на ОС, в ролята на парични единици са ресурсите, а в качеството на клиенти на банката се явяват процесите).

**ТЕМА 2.4.**

**Синхронизация на процеси.**

**Методи и алгоритми за реализация.**

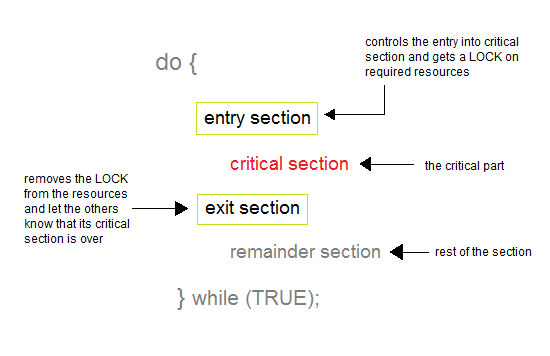
**Process Synchronization**

Process Synchronization means sharing system resources by processes in a such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.

Process Synchronization was introduced to handle problems that arose while multiple process executions. Some of the problems are discussed below.

Critical Section Problem

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.



**Solution to Critical Section Problem**

A solution to the critical section problem must satisfy the following three conditions :

1. **Mutual Exclusion**

Out of a group of cooperating processes, only one process can be in its critical section at a given point of time.

1. **Progress**

If no process is in its critical section, and if one or more threads want to execute their critical section then any one of these threads must be allowed to get into its critical section.

1. **Bounded Waiting**

After a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section, before this process's request is granted. So after the limit is reached, system must grant the process permission to get into its critical section.

**Synchronization Hardware**

Many systems provide hardware support for critical section code. The critical section problem could be solved easily in a single-processor environment if we could disallow interrupts to occur while a shared variable or resource is being modified.

In this manner, we could be sure that the current sequence of instructions would be allowed to execute in order without pre-emption. Unfortunately, this solution is not feasible in a multiprocessor environment.

Disabling interrupt on a multiprocessor environment can be time consuming as the message is passed to all the processors.

This message transmission lag, delays entry of threads into critical section and the system efficiency decreases.

**Mutex Locks**

As the synchronization hardware solution is not easy to implement from everyone, a strict software approach called Mutex Locks was introduced. In this approach, in the entry section of code, a LOCK is acquired over the critical resources modified and used inside critical section, and in the exit section that LOCK is released.

As the resource is locked while a process executes its critical section hence no other process can access it.

**Semaphores**

In 1965, Dijkstra proposed a new and very significant technique for managing concurrent processes by using the value of a simple integer variable to synchronize the progress of interacting processes. This integer variable is called **semaphore**. So it is basically a synchronizing tool and is accessed only through two low standard atomic operations, wait and signal designated by P() and V() respectively.

The classical definition of wait and signal are :

* Wait : decrement the value of its argument S as soon as it would become non-negative.
* Signal : increment the value of its argument, S as an individual operation.

**Properties of Semaphores**

1. Simple
2. Works with many processes
3. Can have many different critical sections with different semaphores
4. Each critical section has unique access semaphores
5. Can permit multiple processes into the critical section at once, if desirable.

**Types of Semaphores**

Semaphores are mainly of two types:

* **Binary Semaphores :** They can only be either 0 or 1. They are also known as mutex locks, as the locks can provide mutual exclusion. All the processes can share the same mutex semaphore that is initialized to 1. Then, a process has to wait until the lock becomes 0. Then, the process can make the mutex semaphore 1 and start its critical section. When it completes its critical section, it can reset the value of mutex semaphore to 0 and some other process can enter its critical section.
* **Counting Semaphores :** They can have any value and are not restricted over a certain domain. They can be used to control access a resource that has a limitation on the number of simultaneous accesses. The semaphore can be initialized to the number of instances of the resource. Whenever a process wants to use that resource, it checks if the number of remaining instances is more than zero, i.e., the process has an instance available. Then, the process can enter its critical section thereby decreasing the value of the counting semaphore by 1. After the process is over with the use of the instance of the resource, it can leave the critical section thereby adding 1 to the number of available instances of the resource.

**Limitations of Semaphores**

1. Priority Inversion is a big limitation os semaphores.
2. Their use is not enforced, but is by convention only.
3. With improper use, a process may block indefinitely. Such a situation is called Deadlock. We will be studying deadlocks in details in coming lessons.

**ТЕМА 2.5.**

**Комуникация между процеси в МОС.**

In computer science, inter-process communication or interprocess communication (IPC) refers specifically to the mechanisms an operating system provides to allow processes it manages to share data. Typically, applications can use IPC, categorized as clients and servers, where the client requests data and the server responds to client requests. Many applications are both clients and servers, as commonly seen in distributed computing. Methods for achieving IPC are divided into categories which vary based on software requirements, such as performance and modularity requirements, and system circumstances, such as network bandwidth and latency.

IPC is very important to the design process for microkernels and nanokernels. Microkernels reduce the number of functionalities provided by the kernel. Those functionalities are then obtained by communicating with servers via IPC, increasing drastically the number of IPC compared to a regular monolithic kernel.

Inter processor communication in a multiprocessor system─ used to generate information about certain sets of computations finishing on one processor and to let the other processors waiting for finishing the computations take note of the information

Similarly, there is inter process communication from a process (task or thread to another).

**Need of Inter Process Communication**

• Assume that there is need to send through the kernel an output data (a message of a known size with or without a header or a flag to notify an event) for processing or taking note of by another task

• Global variables problems ─ shared data and no encapsulation of the data or message accessibility by other tasks

• Inter Process Communication (IPC) functions in the OS provide the solutions

Inter Process Communication (IPC) means that a process (scheduler, task or ISR) generates some information by signal (for other process start) or value (for example of semaphore) or generates an output so that it lets another process take note or use it through the kernel functions for the IPCs

Inter Process Communication (IPC) in Multitasking System

Used to signal for other process to start or • post a token or flag or • generate message from the certain sets of computations finishing on one task and to let the other tasks take note of signal or get the message.

Operating System Provisioning for the Operating System Provisioning for the IPC functions

Signal for other process to start

• Semaphore (as token, mutex) or counting semaphores for the inter task communication between tasks sharing a common buffer

* Аспекти при комуникация между процеси

(Inter process communication или IPC)

* Предаване на данни между процесите
* Гарантиране на коректно взаимодействие на процесите
* Видове взаимодействия между процеси

Процеси, конкуриращи се за общ ресурс

* Проблемът се нарича **състезание** (**race condition**), а решението му - **взаимно изключване** (**mutual exclusion**).

**Процеси, извършващи обща работа**

* Производител-Потребител (Producer-Consumer)
* Читатели-Писатели (Readers and Writers)
* Задачата за обядващите философи (Dining Philosophers)
* За коректното взаимодействие на процесите е необходима **синхронизация** (**synchronization**).
* Механизми на ОС за решаване на IPC проблемите
* Обща памет (Shared memory)
* Семафори (Semaphores)
* Съобщения (Message passing)
* Взаимно изключване чрез обща памет
  + Алгоритъм на Декер
  + Алгоритъм на Питерсон
* Семафори
  + Взаимно изключване чрез семафори
  + Синхронизация чрез семафори
* Съобщения
  + Адресиране на съобщенията
  + Буфериране на съобщенията

**Взаимно изключване Mutual exclusion - Решение чрез обща памет**

Критичен участък (critical section)

* Тази част от програмата, в която се осъществява достъп до обща памет или се вършат неща, които могат да доведат до състезание Дейкстра нарича критичен участък.
* Процес е в критичния си участък, ако е започнал и не е завършил изпълнението му, независимо от състоянието си.

Условия за избягване на състезанието

1. Във един момент най-много един процес може да се намира в критичния си участък.

2. Никой процес да не остава в критичния си участък безкрайно дълго.

3. Никой процес, намиращ се вън от критичния си участък, да не пречи на друг процес да влезе в своя критичен участък.

4. Решението не бива да се основава на предположения за относителните скорости на процесите.

**Семафори Взаимно изключване и синхронизация чрез семафори**

* Семафорът е обект, който има
* брояч - цяло неотрицателно число
* списък на процеси в състояние блокиран по семафора
* Операции над семафор
* Инициализация

**semaphore s = *n*;**

* Операция провери - **P(s)** или **down(s)**

**P(s){**

**if ( s > 0 ) s = s - 1;**

**else блокира процеса, изпълняващ P по семафора s;**

**}**

* Операция увеличи - **V(s)** или **up(s**)

**V(s){**

**if (има блокирани по семафора s процеси)**

**събужда един процес;**

**else s = s + 1;**

**}**

**Методи за междупроцесни комуникации**

* Чрез обща памет и семафори
* Двата аспекта на комуникацията - предаването на данните и синхронизацията, се решават с различни механизми.
* Отговорността за комуникацията е на програмиста.
* Този метод не е безопасен.
* Чрез съобщения
* Двата аспекта на комуникацията се решават от механизма.
* Отговорността за комуникацията е на ОС.
* Този метод е по-безопасен.

**Съобщения Междупроцесни комуникации чрез съобщения**

* Между процесите се установява комуникационна връзка.
* Процесите обменят събщения чрез примитиви:

**send(*destination*, *message*)**

**receive(*source*, *message*)**

1. Как се установява комуникационна връзка между процесите?

2. Може ли една комуникационна връзка да свързва повече от два процеса?

3. Колко комуникационни връзки може да има между два процеса?

4. Еднопосочна или двупосочна е комуникационната връзка?

5. Какво е капацитет на комуникационната връзка?

6. Има ли някакви изисквания за размер и/или структура на съобщенията предавани по определена комуникационна връзка?

Адресиране на съобщенията

* Директна комуникация

Всеки процес именува явно другия процес в примитива.

Процес P Процес Q

send(**Q**, message); receive(**P**, message);

1. Комуникационната връзка се установява автоматично между двойката процеси, но всеки от тях трябва да знае идентификатора на другия.

2. Комуникационната връзка свързва точно два процеса.

3. Между всяка двойка процеси може да има само една комуникационна връзка.

4. Комуникационната връзка е двупосочна.

* Асиметричен вариант на директна комуникация

receive(**ANY**, message);

* Косвена комуникация

Съобщения се изпращат в и получават от специален обект:

* пощенска кутия (mailbox)
* опашка на съобщенията (message queue)

Процес P Процес Q

send(**A**, message); receive(**A**, message);

1. Комуникационната връзка се установява между процесите когато те използват обща пощенска кутия.

2. Комуникационната връзка може да свързва и повече от два процеса.

3. Между два процеса може да съществува повече от една комуникационна връзка.

4. Комуникационната връзка може да е еднопосочна или двупосочна.

Буфериране на съобщенията

* Съобщения без буфериране

Процесите изпращач и получател се синхронизират в момента на предаване на всяко съобщение.

Предаване на съобщения по метода на **рандеву**.

* Съобщения с автоматично буфериране

Определен брой (обем) изпратени и още неполучени съобщения временно се съхраняват в буфер на комуникационната връзка.

* След като изпълни send, изпращачът не може да знае дали съобщението му е получено.

**Модул III: Управление на разпределени ресурси**

**ТЕМА 3.1.**

**Организация и управление на оперативната памет.**

1. Паметта като ресурс на операционните системи – организация и управление , използване на различните видове памет в компютърните системи. Под организация се разбира начина за представяне и използване от потребителя.

Възможностите са:

* + паметта да е представена на един потребител или да много потребители
  + паметта да е разделена на блокове или да няма такова разделение. Може да бъде с еднакъв или различен размер.
  + потребителя да има различен достъп или еднакъв

Управление – набор от стратегии за използване на паметта , които гарантират оптимални характеристики на системата

Стратегиите биват:

* за въвеждане на информация
* за разполагане на информация
* за извеждане на информация

регистри 🡨🡪 кеш памет 🡨🡪 оперативна памет 🡨🡪 външна памет

2. Организация на оперативната памет

|  |
| --- |
| ОС |
| програма за управление |
|  |

|  |  |
| --- | --- |
| М1 |  |
|  | М2 |
| Мк |  |

3. Стратегии за управление – въвеждането става по заявка

|  |
| --- |
| ядро на ОС |
| 21к |
| \\\\\\\\\\\\\\\\\\\\\\ |
| ////////////////////// |
| 16k |
| \\\\\\\\\\\\\\\\\\\\\\ |
| ////////////////////// |
| 45k |

|  |  |  |
| --- | --- | --- |
| П1 | П2 | Пк |
| 15к |  |  |
|  |  |  |

16к , 21к , 45к или 45к , 21к , 16к

**ТЕМА 3.2.**

**Организация и управление на виртуалната памет.**

1. Организация на виртуалната памет

Виртуалната памет е комплекс от технически и програмни средства , които позволяват потребителите да адресират пространства по – големи по размер на оперативната памет.

* странична организация – VA = (p,d) , където р – номер на блока , d – отместване
* сегментна организация – блоково използване на паметта , с филтрирани размери – VA (s,d) , където s – номер на сегмента
* сегментно – странична VA = (s,p,d)
* странично – сегментна VA = (p,s,d)

Преобразуване на виртуални адреси в реални адреси се извършва от специализиран механизъм.

Механизъм за преобразуване на виртуални адреси в реални адреси.

Странична организация използва се таблица за съседство на страниците. Всеки ред от тази таблица носи информация за една страница , която може да бъде:

* флаг за наличност на страница
* флаг за разрешени операции (поле за защита)
* флаг за модификация
* реални адреси се получават от съответния запис от таблицата за състоянието на страниците и се добавят отместването получено от изместването на виртуалните адреси , при това последната страница може да бъде поставена в кой да е кадър на оперативната памет. Таблица за съседство на страниците и виртуалните адреси освен в оперативната памет могат да се съхраняват и в асоциативната памет и по този начин се увеличава бързодействието. Тогава говорим за механизъм на преобразуване на адресите с асоциативно преместване

Възможно е да има и комбинирано преобразуване (пряко – асоциативно или обратното)

при него не цялата таблица се намира в асоциативната памет.

При сегментната организация се използва таблица за състоянието на сегментите и механизъм за преобразуване на адресите използва информацията от нея да се получат реалните адреси. Може да има пряко или асоциативно представяне. Изпълнимите процеси се обръщат по VA = (s,p) , S се добавя към началния адрес намиращ се в регистъра “b” полученото число показва от кои ред от таблицата за състоянието на сегментите , ще се прочете информацията необходима за получаване на реалните адреси към , които от таблицата се добавя отместване d от виртуалния адрес и се получава реалния адрес.

Възможно е дължината на сегмента да е по – голяма от наличното свободно пространство в този случай се получава прекъсване.

Сравнение между странична и сегментна организация на паметта.

Общото между тях е: и двете имат блокова организация на виртуалната памет ; и при двата вида могат да се използват по големи виртуални памети

Различия: при страничната организация блока е с фиксирани размери , а при сегментната организация не е с фиксирани размери; програмистите не е нужно да знаят за вида на организацията при странична организация , докато при сегментна се налага;

при страничната организация има едно линейно адресно пространство , докато при сегментната организация са много; при сегментната организация е възможно кода и данните да се защитават по отделно; при сегментната организация могат лесно да се съхраняват променливи данни;

2. Управление на виртуалната памет

* за въвеждане на информация (по заявка)
* за разполагане на информация
* за извеждане на информация

Априорно въвеждане на информация е необходима локалност на сегментите (страниците).

Временна локалност – обръщение към една и съща страница

Проста локалност – обръщение към последната страница за определен период от време

Страници с най – голяма локалност образуват т.н. работно множество.

При сегментната организация се разполагат и трите стратегии , а при странична само първи срещнати.

Генератор на случайни числа – извеждане на случайна страница

Извеждане на първа постъпила страница (първи влязъл – първи излязъл)

Извеждане на страница , която най – дълго време не е модифицирана

Извеждане на страница , която най – дълго време не е използвана

Извеждане на страница , която най – дълго време не е модифицирана и използвана

**ТЕМА 3.3.**

**Организация и управление на кеш памет.**

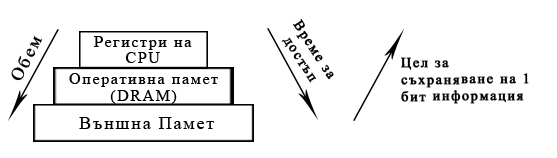
**Кеш-памет (cache) като ресурс на ОС.**

Кеш-паметта е начин за функиониране на 2 типа запомнящи устройства, които се отличават по времето за достъп до информацията и по цена. При нея информацията от “бавното” ЗУ (Запазващо устройство) динамично се копира в бързото ЗУ. Това се прави за най-често използваната информация.

Чрез използването на кеш-памет се намалява времето за достъп до данните и необходимостта от скъпа бързодействаща памет.

Кеш-паметта се разглежда по-често като работа на “бързо” ЗУ. Те имат < обем. По-скъпа и заема междинно мясту м/у основната (оперативната памет) и регистрите. Като “бавно” ЗУ се разглежда основната (оперативната) памет.

Фиг.1: Иерархия на запомнящите устройства (паметта на КС)



Свойство на кеш-паметта се явяват нейната прозрачност за програмистите и потребителите. Те не участват в организацията и управлението й. Прехврърлянето (преместването) на информацията от едното към другото ЗУ се реализират автоматично.

Кеширането е универсален метод, използван за намаляване на времето за достъп до оперативната памет, до твърдия диск или до други ЗУ. Обикновено (в качеството) като кеш памет се използва бързодействаща статическа памет за намаляване на средното време за достъп до опер. памет или буфери в опер. памет (ОП) – за намаляване времето за достъп до данни записани на твърдия диск. Виртуалната памет (ВП) може също да се счита като вариант за реализация на принципа за кеширане на данни. В този случей ОП играе роля на кеш по отношение на външната памет (например твърдия диск).

Регистрите са малки, свръхбързи RAM клетки, вградени в процесора.

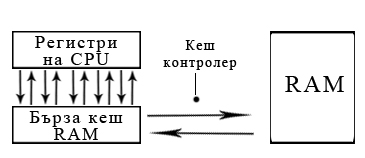
DRAM (динамична (RAM) оперативна памет) – клетки памет, които се използват за стандартни RAM модули. Изискват постоянен опреснителен сигнал, за да помнят информацията.

SRAM (статична RAM) – вид ОП, която (за разлика от DRAM) съхранява информацията си без опреснителен сигнал. SRAM е по-бърза и по-скъпа от DRAM. Използва се за кеш. Кеш RAM е много по-бърза, отколкото обикновенната ОП.

Кеш паметта (кешът) доставя информацията си на регистрите на CPU. Това са малки единици памет, които са разположени директно в ядрото на CPU и са най-бързата съществуваща RAM. Размерът и броят на регистрите се определя в зависимост от вида на процесора.

Кешът е основен мост м/у оперативната памет и регистрите. Информацията постоянно се прехвърля от регистрите на CPU към кеш Level 1 (първо ниво), от кеш L1 към регистрите, от един регистър към друг, от кеш L1 към кеш L2 и т.н.

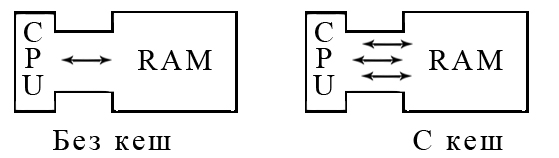
Фиг. 2



Кеш паметта се изпозлва за по-ефективен преход м/у бързия процесор и бавният RAM. Кеш RAM работи с по-висока тактова честота, отколкото обикновенната RAM. Поради това информацията в нея може да бъде прочетена по-бързо.

Кешът намалява специф. проблеми в КС наречени “стеснения” т.е. преходите м/у бързи и бавни системи, където бързите трябва да \*чакът\* преди да доставят или получат информация.

Фиг. 3



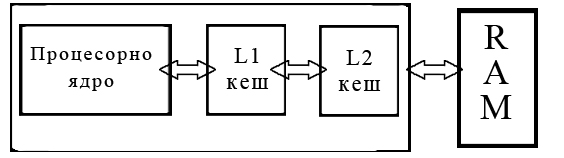
Замисалът на кешът е да функционира като “затворен запас” от бърза RAM. От него CPU може винаги да се снабдява.

Кеш контролерът непрекъснато управлява работния процес на CPU. Той чете данните, необходими на CPU. Когато кешът съдържа нужната информация това се нарича кеш попадение (cache hit), а когато не съдържа – кеш пропуск (cache miss).

В КС обикновенно има 2 или 3 “затворени запаса/нива” кеш памет. Наричат се L1 (Level 1) – ниво 1, L2 (второ ниво) и ако (е възможно) Level 3 кеш. Наприер в CPU Itanium на Intel има 3 нива на кеша, които се използват само за много специални сървър приложения.

Обикновено в КС имат L1 и L2 кеш.

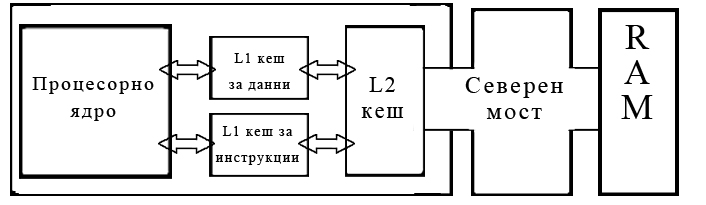
Фиг. 4



Кеш L1 е вградена в ядрото на CPU. Тя е частица от RAM, обикновенно 8, 16, 20, 32, 64 или 128 килобайта (KB), която работи със същата тактова честота, както и останалата част на CPU. Затова казваме, че кешът L1 е част от CPU.

L1 се разделя на 2 части ендата е за информация, а другата за инструкции (Фиг. 5). Ако е обща говорим за унифицирана кеш (както е горе в схемата).

Фиг. 5



Начинът, по които CPU използва кешът от първо и второ ниво, имат решаващо значение за оползотворяването на високата тактова честота.

Второто ниво на кеша, кеш L2 е обикновенно по-голямо (и унифицирано), например 512 KB, 1 MB, 2MB и т.н. Целта на кеша L2 е непрекъснато да сортира големи количества информация от оперативната памет, така че те да са достъпни за L1.

L2 не е толкова бърз колкото L1, но си остава по-бърз от обикновенната ОП.

Колкото повече кеш памет L1 и L2 има в ядна КС, толкова по-ефективен е процесорът.

Кешът е решаващ фактор за оползотворяване на високите тактови честоти на CPU. L1 е изключително ефективен. В 96÷98% от случаите CPU е способен да открие в кешът данните и инструкциите от които се нуждае.

В кеш паметта се съхраняват записи за данните в основната памет. Всеки запис съдържа:

|  |  |  |
| --- | --- | --- |
| (1) | (2) | (3) |
|  |  |  |

-адрес на данните в ОП (1)

-данни (2)

-управляваща информация (3)

Кеш-памета не е адресируема памет.

Търсенето на (нужните) необходимите данни се прави по съдържание.

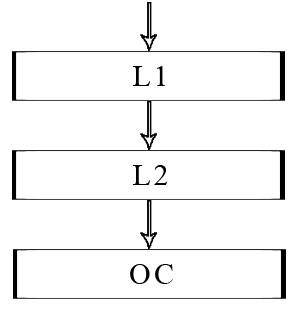
Ефективността на кеша (кеширането) зависи от вероятността за попадение в кеша. Процентът на попадение е обикновенно > 90% Това се обяснява с наличието на обективните свойства на данните, а именно – пространствена и временна локалност.

Проблема за съгласуване на данните:

Наличието на 2 копия на данни – в основната и в кеш паметта поражда проблема за съгласуване на данните. Ако се направи запис в ОП на някои адрес, а съдържанието на тази клетка е от паметта е в кеша, то съответстващия запис в кеша е недостоверен.

В повечето КС има 2 нива на кеш памет. Кеш памет от ниво 1 е по-малкият по обем и има много по-голямо бързодействие в сравнение с L2. L2 играе роля на основна памет по отношение на L1.

Фиг. 6



Управлението на кеша се реализира от мениджър на кешът (\*на същия\*). Той предоставя високоскоростен интелигентен механизъм за намаляване на действия вход/изход и увеличаване на цялосттния системен обем. Като кешира, на базата на виртуални блокове, кеш мениджъра може да извърши интелигентно предварително четене. Той може да предостави специален бърз входно/изходен механизъм за намаляване на процесорното време, необходимо при операции за четене и писане. Така се увеличава ефективността и се намалява дублирането на код. Последното се (извършва) реализира чрез глобалния (единствен) мениджър на паметта, които управлява физическата памет.

**ТЕМА 3.4.**

**Методи и стратегии за диспечиране.**

**ТЕМА 3.5.**

**Управление на устройства.**

**Модул IV: Active Directory**

**ТЕМА 4.1.**

**Създаване на Active Directory с помощта Windows 2008 Server.**

Мрежови ОС на Microsoft

- те създават домейни и Active Directory

- домейн – група от компютри, които са под общо административно управление

- може да се изгради така нареченото дърво на домейни

- Active Directory – директориини услуги – те осигуряват централизиран контрол на мрежовите ресурси (компютри, потребители, споделени папки и т.н.) в цялата мрежа.Възможност за организиране на информацията и опростено управление на информацията.

Организиране на информацията в директорията (директория - база от данни с иерархична структура), в нея се поддържа информация за мрежовите ресурси и обекти.

Всички споделени ресурси се публикуват в директорията.

Потребителите могат да откриват и осъществяват достъп до всеки един мрежов ресурс без да познава неговото местоположение.

За да се създаде една такава директория е необходимо да имаме един компютър наречен контролер в който съхраняваме базата данни, може и повече от един контролер, като базата автоматично се репликира между останалите.

Клиента използва протокол LDAP за да осъщесвти връзка с контролера и съответно да осъществи връзка с търсения обект.

В Windows контролера се нарича Domain Controller – домейн контролер.

Всеки обект в конторлера има ACL – Access Control List – в него се съдържат правата за достъп асоциирани с обекта. Правата за достъп могат да бъдат от “всичко е позволено” до “всичко е забранено”

Разликите м/у Windows 2000 Server и Windows 2003 Server трябва да имаме PDC – Primary Domain Controller и Backup Domain Controller. Промените се правят единствено в PDC и след това се репликират в BDC. От BDC може само да се чете и той поема управлението единствено ако отсъства PDC. При Windows 2008 Server няма нужда от PDC и BDC, нужно е да имаме само Domain Controller, като можем да имаме повече от 1 съответно като на всеки един се репликира информацията от основния.

ctive Directory (AD) е Windows услуга, която улеснява работата с взаимосвързани, сложни и различни мрежови ресурси в единен начин.

**Какво означава Active Directory**

Първоначално Active Directory е използвана в Windows 2000 Server и преработена с допълнителни функции в Windows Server 2008. Active Directory осигурява общ интерфейс за организиране и поддържане на информация, свързана с ресурси, свързани с различни мрежови директории. Указателите могат да бъдат системно-базирания (като Windows OS), специфични за приложението или мрежови ресурси, като принтери. Active Directory служи като единствен обект за бърз достъп до данни за всички потребители и контрол на достъпа на потребителите въз основа на политиката за сигурност.

**Техническо обяснение за активната директория**

Active Directory предоставя следните мрежови услуги:

* Lightweight Directory Access Protocol (LDAP) – отворен стандарт, използван за достъп до други услуги директория;
* Услуги за сигурност с помощта на принципите на Secure Sockets Layer (SSL) и Kerberos-базирано идентифициране;
* Йерархична и вътрешна памет на организационни данни в централизирано място за по-бърз достъп и по-добро управление на мрежата;
* Наличие на данни в множество сървъри с едновременни известия осигуряват по-добра мащабируемост.

Всеки възел в структурата на дървото се нарича обект и свързва мрежови ресурси като потребители или услуги. Като концепция на тема базата данни, схемата на Active Directory се използва за определяне на атрибут и вид за определен обект на Active Directory, което улеснява търсенето на свързаните ресурси на мрежата на базата на определени атрибути. Например, ако даден потребител трябва да използва принтер с възможност за цветен печат, атрибут на обекта може да се настрои с подходяща дума, така че да е по-лесно да намерите цялата мрежа и да се определят местоположението на обекта въз основа на тази ключова дума.

Домейна се състои от обекти, съхранявани в специална, сигурна и свързана дървовидна структура. Един домейн може да е разположен на  множество сървъри – всеки от които е с възможност за съхраняване на множество обекти. В този случай, организационни данни се съхраняват на различни места, така че един домейн може да има множество сайтове за един домейн. Всеки обект може да има няколко домейн контролери за архивиране. Множество домейни могат да бъдат свързани, за да се образува дървовидна структура, която споделя обща схема, конфигурация и глобален каталог (използван за търсене в различни домейни).

**Инсталиране на активна директория под Windows Server 2012**

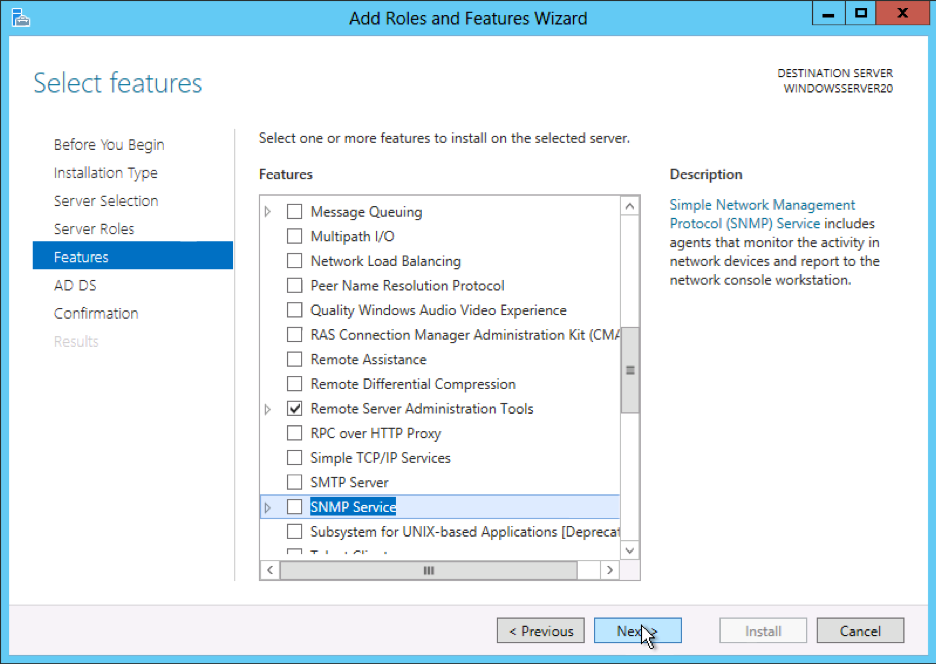
**Инсталиране на активна директория**

1. Отваряме **Server Manager** от таскбара.
2. От**Server Manager** Dashboard, избираме **Add roles and features**.

Това ще стартира помощника за инсталиране на роли.

1. Избираме **Role-based or features-based**и избираме **Next**.
2. Избираме **Next**.
3. От Server Roles избираме **Active Directory Domain Services.**

Ако се появи съобщение, че иска инсталиране на някой Feature кликваме **Next**.



1. Преглеждаме информацията от **AD DS tab** избираме **Next**.
2. От **Confirm installation selections**, преглеждаме инсталацията и избираме **Install**.

**Стартиране на сървъса**

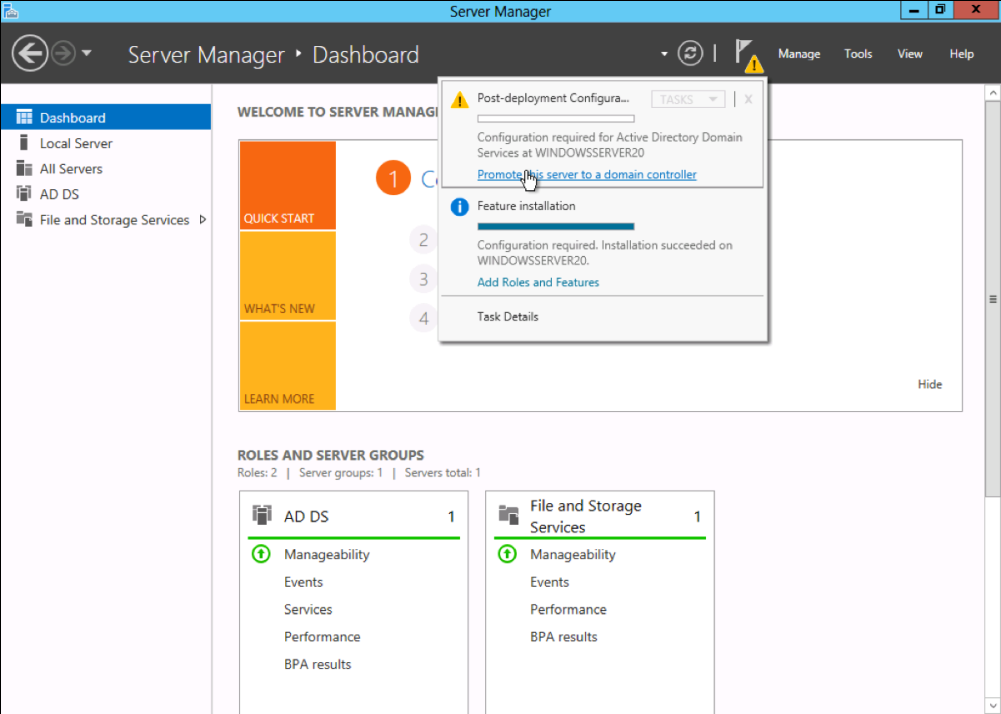
1. Кликваме на **Start > Control Panel**.
2. От **Services**, щракваме с десен бутон върху **Remote Registry** и отваряме **Properties**.
3. От падащото меню, избираме **Automatic**.
4. От **Service Status**, избираме **Start**.

Сървиса ще се стартира.

**Конфигуриране на активната директория**

След като се инсталира DS ролята на АД сървъра ще трябва да бъде конфигуриран вашия домейн.

1. Отваряме **Open the Server Manager** от таск бара.
2. Отваряме Notifications Pane и избираме **Notifications icon** от горната част на Server Manager. Избираме **Promote this server to a domain controller**.



1. От **Deployment Configuration** избираме **Add a new forest** от радио бутона. Избираме домейн и кликваме на **Next**.
2. Избираме **Domain and Forest functional level** и въвеждаме парола за **Directory Services Restore Mode** **(DSRM)** от полето за парола.

**Какво е Active Directory?**

Активната директория е услуга с която можем да управляваме мрежовите ресурси. Също така тя седи в основата на Microsoft Windows. Технологията на АД е базирана на стандартни интернет протоколи и е проектирана така, че да можем ясно да дефинираме структурата на нашата мрежа. АД използва DNS (Domain Name System), която организира в домейни група от компютри. Dns се използва за определяне на съответствия между имена на хостове и числови адреси. Също така DNS е основна част от АД. Преди да сме инсталирали АД трябва да имаме конфигуриран DNS в мрежата.

**Какво е домейн?**

Домейнът (domain name) е уникално име, което сочи към дадена интернет страница. Названието произхожда от английската дума domain - област, територия, сфера, обсег. В интернет не могат да съществуват два напълно еднакви домейна (имена на уеб сайтове). За да работи един уеб сайт неговото домейн име трябва да е свързано с конкретен IP адрес. IP адресът е уникален номер на компютъра, върху който физически е разположен сайта. IP адресите представляват поредица от цифри и точки - например 213.45.78.5, 85.128.60.178 и т.н.

**Какво е Домейн контролер?**

Домейн контролерът е сървър, който поддържа Active Directory  базата с цяла информация в нея и предоставя информация на крайните клиенти или услуги, които се нуждаят от нея. Отговаря за удостоверяването на всеки в мрежата. Има 2 типа Domain Controller.

1. Writable - Този тип могат да четат и да променята инф.в базата. Можем да имаме повече от един записващ домейн контролер.
2. Read Only - могат само да четат дадена информация.

**Какво е Forest?**

Forest е по-голямо от домейните. Винаги да имаме домейн, знаем, че имаме форест с един единствен домейн. На посочената картинка Forest root-a 1вият домейн създаден в средата e Beraxo.local, а др.2 домейна са продължение във Forest-a. Специфичното за Foresta e че имаме схема, която е обща за целия Forest, тоест за всички домейни.

Конфигурацията е дял, който съдържа инф. и обекти за самата конфигурация за фореста.

Глобалния каталог е дял, които репликира инфо за всеки един обект в целия форест.

**Примерни въпроси**

Повечето въпроса са свързани с active directory

1. Команда replicate

Репликация (тиражиране) Под репликация (replication) се разбира съществуването на няколко копия на един и същ файл, всеки от които се съхранява на отделен файлов сървър и съгласуването на данните в копията е осигурено автоматично. Репликацията се реализира от отделна служба на ОС. С нея се увеличава надеждността (има независими копия на всеки файл съхранен на различни файлови сървъри) и се разпределя натоварването между няколко сървъра (клиентите могат да се обръщат към репликирания файл на най-близкия до тях сървър, на който има копие).

1. repadmin

Repadmin отстранява проблеми с репликацията на Active Directory, но също така включва някои команди, които администраторите на Windows може да не разпознават.

4. redirect дали се изпълнява на сървър се изпълнява на сървър или клиент

5. режим ядро: защитен или реален

6. какво е необходимо за да има active directory ? - системата да е в ntfs ; да има хGB free space

* 133 MHz or higher Pentium-compatible CPU.
* 256 MB of RAM recommended minimum. (128 MB minimum supported; 4 GB maximum.)
* 2 GB hard disk with a minimum of 1 GB of free space. (Additional free hard disk space is required if you are installing over a network.)
* Windows 2000 Server supports up to four CPUs on one machine.

Minimum: 1.4 GHz 64-bit processor

An Intel Itanium 2 processor is required for Windows Server 2008 R2 for Itanium-Based Systems.

RAM Minimum: 512 MB

Disk space requirements

The following are the estimated minimum disk space requirements for the system partition.

Minimum: 32 GB

7. какво е необходимо за да има ADS - компютър на който е инсталиран MS Server

8. коя архитектура е по-производителна - многоядрена

9. какви са многослойни драйвери - йерархични, на няколко нива

*10. какво е stateful /stateless на MS Server*

Файлови сървъри stateful и stateless

Файловият сървър може да бъде реализиран по една от двете схеми:

* схеми stateful – със запомняне на данните за последователността на файловите операции зададени от клиента;
* схеми stateless – без запомняне на такива данни. Клиентът отправя запитване (заявка) към сървъра, сървърът я използва, връща отговор, а след това изтрива (отделя) от своите вътрешни таблици цялата информация свързана със запитването (заявката). Не се съхранява никаква текуща информация за клиента.

Ако такава информация се съхранява имаме схема – statefull.

отворени въпроси

* 1. Разлика между процес и нишка

Процеса е програма или част от програмата, която се изпълнява от процесора. Всяка програма става процес след нейното стартиране. Всеки един процес може да се намира в едно от следните състояния: изпълнение, готовност и блокиране. Процесът е множество от ресурси, които представлява програмата.

Нишката е изпълнима единица от процес. Има следните елементи: ID, собствена област от паметта,

2 стека (един за изпълнение в системен режим и друг за потребителски режим), съдържание на споделени регистри, представящи състоянията на процесора.

* 1. Видове МОС. Основни характеристика

Мрежови ОС: -локални и глобални -разпределени и неразпределени -централизирани и равноправни -слабо свързани и силно свързани

* 1. *NFS характеристики. особености. Предназначения*

NTFS се характеризира с възобновяемост, сигурност, дублиране на данни, устойчивост при грешки, поддържане на големи дискове (носители) и файлове, отколкото HPFS (High Performance FS е разработен за ОС OS/2 и може да поддържа дискове до 2ТВ (терабайта)

Това са според Николай въпросите и дадените възможности за отговор. Тук няма верен отговор даден :)

P.S. Имало 2 въпроса за Windows NT