КОЛЕЖ ПО КОМУНИКАЦИИ И ПОЩИ

доц. д-р Огнян Железов

ВВМУ, курс
Администриране на БД
лектор доц. О.Железов

Операционни системи

(записки)

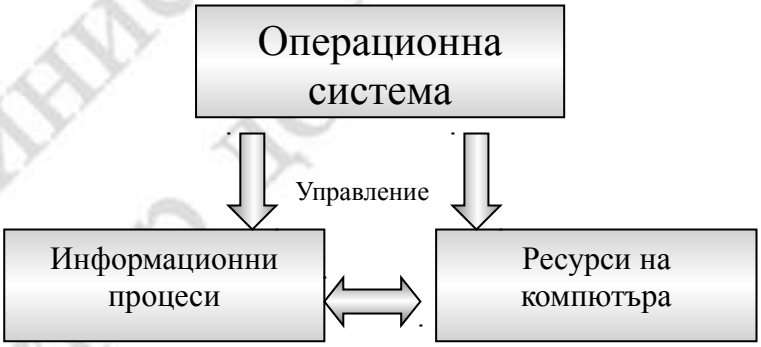
Съдържание

ВВМУ Курс Администриране на БД лектор доц. О.Железов

1. Дефиниция на понятието операционна система.

Операционната система (ОС) е основна част от компютърния системен софтуер, която управлява и координира работата на процесора и устройствата в компютърната система. Тя обслужва работата на приложния софтуер, като заделя необходимите за това хардуерни ресурси и контролира достъпа на различните приложения до тях.

ОС дават достъп на приложния софтуер и на потребителите до определен набор от функционалности, наричани системни ресурси. Приложният софтуерът ги използва чрез извикване на системни функции или приложно-програмен интерфейс (API), като предава данни и инструкции на ОС и получава от нея резултатите от тяхното изпълнение. Потребителите могат да взаимодействат с ОС чрез потребителски интерфейс - например интерфейс с команден ред (CLI) или графичен потребителски интерфейс (GUI). Като цяло при по-прости компютърни системи този интерфейс се приема за част от ОС, а при по-сложните е организиран като самостоятелен приложен софтуер, работещ независимо от самата ОС.



2. История на развитието на операционните системи

Програмите, предшествуващи разработката на ОС, са служебни програми (за зареждане и мониторинг), както и библиотеки на често използвани подпрограми, които започват да се разработват с появата на компютрите от първо поколение в края на 40-те години. Служебните програми намаляват обема на необходимите физически манипулации от

страна на оператора, а библиотеките позволяват да се избегне многократното програмиране на едни и същи действия (осъществяване на операции по вход/ изход, изчисление на математически функции и т.н.).

Първите операционни системи са били само с команден ред. Такива са ДОС, UNIX. През 50-те и 60-те години водеща роля в разработката на ОС играе IBM, която е лидер на пазара за т.н. “големи” (мейнфрейм) компютри. През този период се сформират и реализират основните идеи, определящи функционалностите на ОС: пакетен режим, разделение по време и многозадачност, разделение по пълномощия, реален временен мащаб, работа с файлови структури и файлови системи.

3.1. История на операционна система MS-DOS

Историята на операционната система MS-DOS започва през 1980 година, когато Тим Патерсън от Seattle Computer Products създава QDOS (Quick and Dirty Operating System). Като търговски продукт тя се е предлагала под името 86-DOS, тъй като е била създадена за процесора на Intel 8086. Операционната система QDOS е била създадена на базата на по-известната по това време ОС CP/M, на фирмата Digital Research, но е използвала друг тип файлова система. Microsoft закупува лиценз за 86-DOS за 50 хиляди долара и я предоставя на IBM през декември 1980 година. През юли 1981 година, малко преди появяването на пазара на новия персонален компютър IBM PC, фирмата IBM изцяло изкупува правата за 86-DOS, като доплаща още 80 хиляди долара.

Първата версия на MS-DOS е съдържала множество грешки, които в последствие са отстранени от програмистите на IBM. Коригираната версия се е използвала с името PC DOS на оригиналните компютри на фирма IBM а на многочислените клонинги се е изпозвала под името MS-DOS. Така на практика тези две ОС са се обединили.

3.2. История на операционна система Windows

Първата версия на OS Windows е създадена от Microsoft през 1983 г. като графичен интерфейс (GUI) за тяхната операционна система (MS-DOS). . От тогава от Microsoft са създали много версии на Windows и този продукт се е превърнал от графичен интерфейс в модерна операционна система.

Първата версия на Microsoft Windows включвала проста графична програма за рисуване, наречена Windows Paint, програма за писане Windows Write, календар с възможност за вписване на уговорки в него, личен

информационен мениджър - "cardfile", бележник Microsoft Notepad , часовник, контролен панел, компютърен терминал, клипборд - "Clipboard" и RAM драйвер, а също и MS-DOS Executive и игра "Reversi".

Windows 3.0

Windows 3.0, пуснат през 1990 г е първата версия на Windows, която позволявала на потребителите да извършват по няколко задачи едновременно (мултитаскинг) на по-стар софтуер, базиран на MS-DOS. Windows 3.0 можел да бъде пуснат в режимите Real, Standard или 386 Enhanced и е бил съвместим с всеки Intel процесор от 8086/8088 до 80286 и 80386. Това била първата версия на Windows, която можела да изпълнява програми в защитен режим.

Windows NT

Windows NT е първата версия на Windows, базирана изцяло на Windows платформа. Microsoft наели през Август 1988 г. Дейв Кътлър, един от ръководителите в проектирането на VMS в Digital Equipment Corporation (корпорация, купена по-късно от Compaq, а сега - част от Hewlett-Packard). за да направи заместник на OS/2, но вместо това Кътлър създал съвършено нова операционна система.

Първоначално било планирано да се разработи Windows NT с потребителски и програмен (API) интерфейси в стил на OS/2, обаче OS/2 нямала голям пазарен успех, докато Windows 3.0 се продавала добре на софтуерния пазар. Вземайки предвид тези пазарни ориентири и възможните проблеми, свързани с развитието и поддръжката на две несовместими системи, Microsoft решила да промени своя курс по посока на стратегията за разработване единна цялостна операционна система. Тази стратегия включва разработването на семейство базирани на Windows операционни системи, които да работят на различни типове компютри, от най-малките преносими ноутбук компютри до най-големите многопроцесорни работни станции. Windows NT, както е било наречено следващото поколение Windows-системи, заема най-престижната позиция в семейството на Windows. Той поддържа графичния потребителски интерфейс (GUI) на Windows, и също така е първата, базирана на Windows операционна система на Microsoft, поддържаща Win32 API, 32 битов програмен интерфейс за разработване на нови приложения. Приложният програмен интерфейс Win32 API прави достъпни за приложенията новите възможности на ОС, например многонишковите процеси, синхронизацията на процесите, I/O, управление на обекти.

Windows 95

Windows 95 (с първоначално кодово име Chicago) е хибридна 16-битова/ 32-битова графична операционна система, издадена на 24 август 1995 г. Windows 95 е директен наследник на дотогава отделните продукти на „Майкрософт“ MS-DOS и Windows, както и първата система от тази линия, която не поддържа стари, 16-битови x86-процесори, а задължително изисква Intel 80386 или съвместим с него по-бърз процесор, работещ в защитéн режим. Системата представя значителни подобрения в графичния потребителски интерфейс и други нови идеи и е първият Windows, обвързан с определена версия на DOS (DOS 7.0 на „Майкрософт“).

Значението на MS-DOS е сведено до средство за зареждане на ядрото на Windows в защитéн режим. Все пак той още може да се използва за стари драйвери, за обратна съвместимост, но „Майкрософт“ не препоръчват те да се използват, тъй като така се нарушават многозадачните функции и стабилността на системата. Ядрото на Windows все още използва MS-DOS-драйвери в така наречения безопасен режим на Windows, но той се използва само за решаване на проблеми с Windows-драйверите в защитéн режим.

32-битовият достъп до файлове прави възможни дългите имена на файлове, представени в Windows 95. Те са достъпни едновременно за програмите в Windows и за MS-DOS-програмите, ако се стартират от Windows (те трябва леко да се адаптират, защото дългите имена на файлове изискват по-големи буфери за път до файла и заради това различни системни повиквания).

Windows 95 за първи път представя Старт-бутона и лентата на задачите (таскбара), които стават задължителна част от всеки следващ Windows.

На пазара Windows 95 има несравним успех и го3дина-две след издаването си става най-успешната операционна система на всички времена.

Windows 98, Windows 2000 и Windows XP

Windows 95 е последван от Windows 98, Windows 98 SE, Windows ME, Windows 2000 и Windows XP. Базираното на Windows NT ядро, използвано в Windows 2000 и Windows XP, се доказва като по-мощно от своя предшественик в Windows 95, Windows 98, и Windows ME. Поради това тези версии на Windows биват постепенно изоставени. На 31 декември 2002 г. „Майкрософт“ приключва поддръжката за Windows 95.

Windows 7

Windows 7 първоначално се появява през 2003 г. в олекотена версия под името Windows Vista. След като за кратък период от време през 2003 г. три големи вируса злоупотребяват с недостатъци в Windows ОС, от Майкрософт променят приоритетите си, задържайки части от основната работа над Windows 7, с цел да разработят нови сервизни пакети за Windows XP и Windows Server 2003.

Официалният дебют на Windows 7 е на 22 октомври 2009 г. Първоначално е достъпен на 13 езика, а от 31 октомври е достъпен на още 21 допълнителни, сред които и български език.

Windows 7 включва набор от подобрения, като напредък в разпознаването на допир, реч и ръкопис, поддръжка на виртуални харддискове, подобрена производителност при мултиядрени процесори, подобрена прозиводителност при стартиране и подобрения в ядрото на операционната система.

Windows 7 добавя поддръжка за използване на няколко видеокарти от различен модел и дори прозиводител, нова версия на Windows Media Center, вече интегрирани с Windows Explorer притурки, притурка за Windows Media Center, способност визуално да се закачат и откачат елементи от старт менюто и лентата със задачи, подобрени медийни функции (като например множество вградени кодеци), интегриран XPS (формат подобен на PDF)-услуги пакет, вградена Windows PowerShell (програма за управление на системата чрез команден ред подобно на Command Prompt, но много по-напреднала) и преработен калкулатор с много възможности, включително и режими „Програмиране“ и „Статистика“, заедно с преобразуване на мерни единици.

При лентата със задачи се забелязват най-големи визуални промени. Лентата с инструменти „Бързо стартиране“ е слята с бутоните на изпълняващите се задачи, за да се получи подобрена лента със задачи или нещо, което от Майкрософт наричат "Suberbar" (Супербар). Подобрената лента със задачи също добавя функцията „Изскачащи списъци“, която позволява лесен достъп до най-честите задачи. Така наречения Супербар също позволява произволно разбъркване на всички бутони (включително и на тези от областта за уведомяване), като по този начин се улеснява организирането на работещите приложения от потребителя.

Windows 7 - системни изисквания

	Препоръчителни изисквания	Минимални изисквания

Честота на Процесора	1 GHz (32-bit или 64-bit)	800 MHz (32-bit)
Оперативна памет	2GB	1 GB
Видео карта	DirectX 9.0 capable	DirectX 9.0 capable
Памет на видео картата	256 MB (за Windows 7 Ultimate Professional)	128 MB
Свободно място на твърдия диск	25GB	20GB
Други устройства	DVD-ROM	DVD-ROM
Аудио	Звук	Звук

3.3. История на операционна система Unix

Unix (официалната търговска марка е UNIX) е компютърна операционна система, разработена през 60-те и 70-те години на 20-ти век от група специалисти от AT&T, работеща в Bell Labs. В състава на работната група влизат Кен Томпсън, Денис Ричи и Дъглас МакИлрой. В днешно време Unix има различни версии, разработени през годините от AT&T, както и от други комерсиални доставчици и некомерсиални организации.

Сегашният собственик на търговската марка UNIX е консорциумът за индустриални стандарти The Open Group. Правата върху изходния код на Unix обаче са оспорвани в дело от 2004 г., в което доставчикът на UNIX SCO Group обвинява Novell в оклеветяване на името (slander of title). Забележете, че собственикът на търговската марка използва названието UNIX, а не Unix. Терминът UNIX не е акроним, а следва историческата конвенция компютърните системи да се именуват с главни букви като ENIAC и MISTIC.

Unix е проектирана като преносима, многозадачна и многопотребителска, когато е във времеразделяща конфигурация. За Unix системите са характерни следните особености: използването на неформатиран текст за запазване на данни; йерархична файлова система; устройствата и някои видове процеси (inter-process communication) (IPC) се третират като файлове; използват се голям брой малки програми, които могат да бъдат навързани заедно чрез интерпретатор на командна линия и чрез канали (pipe), в контраст с използването на една монолитна програма със същата функционалност. Тези концепции са известни като философия на Unix.

Официалното признаване на ОС UNIX е през 1971г. , когато тя е внедрена в патентния отдел на AT&T. Разделянето на основателите на ОС

UNIX на две групи дава началото на двата основни клона на ОС – SVR и BSD

През 1991, група от BSD разработчици (Дон Сели Donn Seeley, Майк Карелс Mike Karels, Бил Жолиц Bill Jolitz, и Треант Хейн Trent Hein) напуска Калифорнийския университет за да основе Berkeley Software Design, Inc (BSDI). BSDI произвежда напълно функционална комерсиална версия на BSD Unix за Интел базирани компютри, което постави началото на използването на евтини компютри за сериозни изчисления. Скоро след основаването на компанията Бил Жолиц напуска BSDI за да създаде 386BSD, положил началото на FreeBSD, OpenBSD, и NetBSD.

Полезно е да се запомнят няколко характерни особености на ОС UNIX:

1. Unix популяризира йерархичната файлова система с произволно вложени субдиректории, първоначално въведена от Multics. Тя е възприета от всички съвременни файлови системи
2. Unix използва ASCII текст за почти всички файлови формати.
3. Друго нововъведение на Multics, разпространено от Unix, е създаването на команден програмен интерпретатор с командна линия, достъпна на потребителско ниво и с допълнителни команди, които са представени като отделни програми. Такъв команден интерпретатор е въведен в последствие и в MS-DOS.

3.4. История на операционна система Lunux

Линукс (Linux) води началото си от MINIX – един от многобройните клонинги на UNIX. MINIX е била безплатна операционна система и нейният създател Андрю Таненбаум предоставил нейния код за общо ползване. Точно на базата на този код през 1991 е било създадено и първото ядро на Линукс от студента Линус Торвалдс. Версии на това ядро все още може да се намерят за свободно сваляне на адрес http://www.kernel.org/pub/linux/kernel/Historic .

3.4.1. Принципи на софтуера с отворен код

Основната разлика между Линукс и Windows е принципа на отворения код, на базата на който се разработва Линукс ядрото. Софтуер, който се разпространява с отворен код спазва правилата дефинирани в Open Source лиценза.

- Първото изискване на OSD (Open Source Definition – Дефиниция на отворения код) е, че всеки пакет с отворен код трябва да се разпространява напълно свободно, като това не изключва негова продажба.

- Второто изискване е, че освен програмата, вие трябва да предоставите и изходния код на тази програма. На това се базира и модела на отворения код за подобрение и развитие на софтуера. Получавайки изходения код хиляди програмисти могат да работят над тази програма, да оправят грешки и да добавят нова функционалност.

- Третото изискване е променения изходен код да се разпространява по същия начин както и оригиналния.

- Четвъртото изискване е, че не може една програма компилирана от променен изходен код да се разпространява под същото име и номер на версията както и оригиналната.

3.4.2. История на GNU обществото

Съкращението GNU означава "GNU's Not UNIX" или преведено "ГНУ не е UNIX".

GNU е анонсирана от Ричард Сталман към края на Септември, 1983 година, с идеята да се създаде UNIX – подобна операционна система, която може да се разпространява свободно. Фактически GNU проекта официално стартира през Януари 1984г.

До тогава разпространението на свободен софтуер е ставало без никакви лицензи, което позволило на много компании да използват свободен софтуер за направата на комерсиални продукти. Именно това била причината да се замисли нов лиценз, който би могъл да предотврати комерсиализация и затваряне на продуктите.

Това е така наречения GNU General Public License (GPL) лиценз, който изисква всяка модификация на GPL софтуер също да бъде разпространявана под този лиценз. Това гарантирало, че свободния софтуер, каквото и развитие да търпи, ще си остане свободен.

GPL лиценза е един от няколкото лиценза прилагани при разпространението на свободен софтуер. Другите по често използвани са Artistic License и BSD лиценз. Общото между тях е следното:

Софтуера може да се инсталира на неограничен брой компютри.

Броят на хората използващи софтуера също е неограничен.

Софтуера може да се копира неограничено (свободно или отворено редистрибутиране).

Няма ограничения по модифицирането на софтуера с изключение на запазването на базовата функционалност, за който е бил създаден.

Няма ограничение по разпространението, дори продажбата на софтуера.

3.4.3. Линукс дистрибуции

Различните Линукс дистрибуции са насочени към различни целеви потребителски групи. Могат да се различават по графичния интерфейс, който използват, приложните програми, които идват при инсталацията, системните инструменти, които използват (напр. мениджъри на пакети), както и по много други показатели. Като най-популярни дистрибуции могат да се посочат Ubuntu, Mandriva (получена от сливане на Mandrake и Conectiva), PCLinuxOS, Fedora (некомерсиална версия на Red Hat) и др.

Литература

http://bg.wikipedia.org/wiki/Операционна_система
http://bg.wikipedia.org/wiki/История_на_Microsoft_Windows
http://bg.wikipedia.org/wiki/Windows_95
http://bg.wikipedia.org/wiki/Unix

Web сайт с учебни материали, поддържан от от авторите:
http://rusalka-bg.com/dt/

ВВМУ Курс Администриране на БД лектор доц. О. Желязов

4. Процеси

Терминът процес е основна абстракция, която се реализира в съвременните операционни системи.

Още през 60-те години компютърните системи са можели да извършват няколко дейности едновременно. С цел по-ефективно използване на ресурсите се въвежда т.н. multiprogramming (буквално мултипрограмиране). В паметта са заредени няколко програми и въпреки, че централният процесор във всеки момент изпълнява само една инструкция от една програма, той може много бързо да превключи към изпълнение на друга програма. Същността на тази идея е, че процесорът изпълнява няколко дейности едновременно. За да може да се организира това е необходимо операционната система да съхранява информация за тези дейности – когато процесорът превключи от една програма към друга, той трябва да запомни информацията за старата програма за да може впоследствие да се върне към нейното изпълнение.

4.1. Състояния на процесите

По време на съществуването си всеки процес може да бъде в едно от три основни състояния:

Готов:	Процесът е подготвен за изпълнение, но не му е дадено процесорно време (не е стартиран)
Изпълняван	Процесът се изпълнява
Блокиран	Процесът очаква сигнал, свободен ресурс или съобщение за да продължи изпълнението си.

Допълнителни състояния в мултипрограмните системи

В мултипрограмните системи процесите имат две допълнителни състояния:

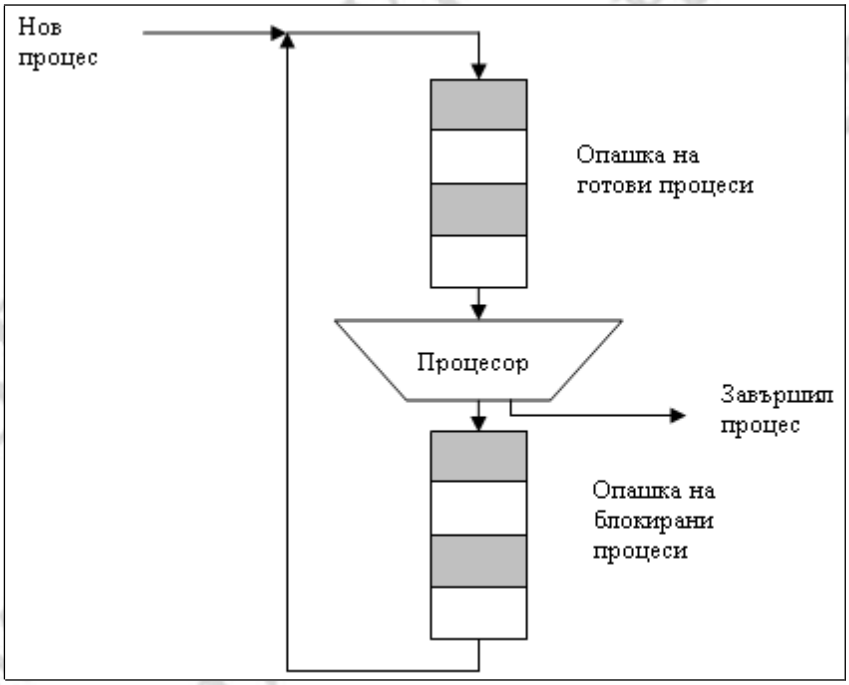
Нов:	Процесът е създаден, но още не е включен в опашката от съобщения
Завършил	Процесът е приключил, но не е предал данните си или още не е унищожен

ВВМУ, Курс на ДА
Администриране на ДА
лектор Юл. О.Желязков

Важно е да се запомни, че операционната система може да преустановява изпълнението на процесите, например за да се освободят ресурси за процес с по-висок приоритет, поради автодиагностика или някакви аварийни ситуации.

4.2. Циклична стратегия при многозадачно изпълнение на процесите

На дадената по-долу фигура е показана цикличната стратегия при многозадачно изпълнение на процеси с предоставяне на квант процесорно време.



Основните елементи на този модел на управление на процесите са опашките за готови и блокирани процеси. Готовите процеси изчакват в опашка да получат квант процесорно време, а блокираните процеси очакват в опашка сигнал за разблокиране.

4.3. Модел на процесите

В една операционна система, която поддържа процеси целият софтуер е организиран в процеси, т.е. във всеки момент централният процесор изпълнява процес и нищо друго. Операционната система трябва да притежава средства за създаване и за унищожаване на процеси. При това, тъй

като се интересуваме от многопроцесни операционни системи, трябва да се предостави начин за идентификация на процесите.

В UNIX/LINUX всеки процес има уникален идентификатор, който се нарича pid (process identifier) и представлява цяло неотрицателно число. Този идентификатор остава неизменен през целия живот на процеса.

4.4. Дървовидна структура на процесите родител-наследник по POSIX стандарта

В UNIX/LINUX и въобще по POSIX стандарта процес се създава със системния примитив fork. Така достигаме до още едно определение за процес, валидно в UNIX/LINUX системите – процес е обект, който се създава от fork. Когато един процес изпълни fork, ядрото създава нов процес, който е точно негово копие. Първият процес се нарича процес-баща, а вторият процес се нарича процес-син. След създаване на процеса-син, той работи асинхронно с баща си. В този смисъл бащата, както и синът могат отново да изпълнят fork и да създадат нови процеси. По този начин всички процеси в системата са свързани в дървовидна йерархия.

4.5. Проблеми при използване на общ ресурс от конкурентни процеси. Критичен участък

Ще разгледаме един пример. Нека два процеса П1 и П2 ползват обща променлива counter и изпълняват следните действия в указаната последователност:

1. П1 чете counter в своя локална променлива с1;
2. П2 чете counter в своя локална променлива с2;
3. П2 увеличава с2 с 1;
4. П2 записва стойността на с2 в counter;
5. П1 увеличава с1 с 1;
6. П1 записва стойноста на с1 в counter.

И двата процеса пробват да увеличат counter с 1, т.е. counter трябва да се увеличи с 2, но в крайна сметка след изпълнение на тези действия, counter се увеличава само с 1. Това е така, защото изменението, направено от П2 се губи. По този начин получаването на верния резултат зависи от реда, в който процесите се изпълняват. Този проблем се нарича състезание (race condition). Неговото решение се нарича взаимно изключване (mutual exclusion) – когато един от конкурентните процеси осъществява достъп до общия ресурс се

изключва възможността другите конкуретни процеси да използват този ресурс.

Могат да се посочат и други примери за съвместно използване на общ ресурс и по-общо, за комуникация между процеси, при които могат да възникнат конфликти. Общото в тези модели е, че работата на процесите трябва да се организира по такъв начин, че те да взаимодействат коректно. По-нататък ще разгледаме три такива средства – *обща памет*, *семафори* и *съобщения*.

4.6. Взаимно изключване чрез обща памет

Проблемът за избягване на състезанието е бил формулиран от Дейкстра с помощта на термина *критичен участък* (critical section). Това е част от програмния код на процес, в която той осъществява достъп до общ ресурс или извършва действия, които могат да предизвикат състезание. Процес се намира в критичен участък, ако той е започнал да изпълнява код от критичен участък, без значение в какво състояние се намира. Условията за коректност на взаимното изключване са следните:

1) Във всеки един момент най-много един процес да се намира в критичен участък;
2) никой процес да не се намира безкрайно дълго в критичен участък;
3) никой процес, който се намира вън от критичния си участък да не пречи на други процеси да влязат в своя критичен участък;
4) решението на проблема не бива да се базира на относителната скорост на процесите.

4.6.1. Алгоритъм за взаимно изключване чрез флагове

По-долу е даден програмния код на два процеса, които осъществяват достъп до общ ресурс, като съгласуването се извършва с флагове.

```
P0 : while (1)
{
    wants0 = 1;              // P0 вдига флага за влизане в критичния си участък
    while (wants1);          // Ако P1 е в своя критичен участък, P0 изчаква
    critical_section0();     // P0 изпълнява кода в критичния участък
    wants0 = 0;              // P0 сваля флага за влизане в критичния си участък
    non_critical_section(); //P0 изпълнява код извън критичния си участък
}
```

Аналогични са операциите, които се извършват при от втория процес

```
P1 : while (1)
{
     wants1 = 1;              // P1 вдига флага за влизане в критичния си участък
    while (wants0);           // Ако P0 е в своя критичен участък, P1 изчаква
    critical_section1();      // P1 изпълнява кода в критичния участък
    wants1 = 0;               // P1 сваля флага за влизане в критичния си участък
non_critical_section1(); //P1 изпълнява код извън критичния си участък
}
```

4.6.2. Алгоритъм на Декер за взаимно изключване чрез флагове

Алгоритъмът на Декер е модификация на алгоритъма с флаговете. Той въвежда допълнителна обща променлива turn, чрез която се задава ред за влизане в критичната секция: при стойност 0 реда е на P0, а при стойност 1 – на P1. С алгоритъмът на Декер се осигурява съгласуване и случая когато двата процеса изпълняват едновременно тялото на основните си цикли. В такъв случай, в зависимост от стойността на turn единият процес отстъпва на другия и изчаква докато той излезе от критичния си участък.

По-долу е даден кода на първия процес при използване на алгоритъма на Декер

```
shared int wants0 = 0; //този флаг (при 1) указва, че P0 трябва да влезе в
//своя критичен участък
shared int wants1 = 0; //този флаг (при 1) указва, че P1 трябва да влезе в
//своя критичен участък
shared int turn = 0; //задава номерът на процеса, който е на ред да
//изпълнява критичния си участък
P0 : while (1)
{
 wants0 = 1;
 while (wants1)   //Изчаква P1 да няма право да влезе в критичния участък
 if (turn == 1) {   //Ако P1 е наред да влезе в критичния участъл
   wants0 = 0;         // P0 сваля флага за влизане в критичния си участък
   while (turn == 1); //P0 изчаква реда си за влизане
   wants0 = 1;         // P1 вдига флага за влизане в критичния си участък
 }
 critical_section0(); //P0 изпълнява критичния си участък
 turn = 1;               //P0 отстъпва на P1 реда за влизане в крит. участък
```

```
  wants0 = 0;           //P0 сваля флага за влизане в критичния си участък
  non_critical_section0();//P0 изпълнява код извън критичния си участък
}
```

Кода за втория процес е аналогичен

```
P1 : while (1)
{ wants1 = 1;
while (wants0)
if (turn == 0) {
wants1 = 0;
while (turn == 0);
wants1 = 1;
}
critical_section1();
turn = 0;
wants1 = 0;
non_cricial_section1();
}
```

4.6.3. Алгоритъм на Питерсън

```
Алгоритъм на Питерсън
shared int turn = 0; //задава номерът на процеса, който е на ред да
//изпълнява критичния си участък
shared int interested0 = 0; //този флаг (при 1) указва, че P0 трябва да влезе в
//своя критичен участък
shared int interested1 = 0; //този флаг (при 1) указва, че P1 трябва да влезе в
//своя критичен участък
P0 : while (1)
{
  interested0 = 1; // P0 вдига флаг за влизане в крит. участък
  turn = 0;          //P0 записва във флага turn своя номер
  while (turn == 0 && interested1); //P0 изчаква докато turn=0 и interested1=1
critical_section0();//P0 изпълнява критичния си участък
interested0 = 0;  //P0 сваля флага за влизане в крит. участък
non_critical_section0();//P0 изпълнява код извън критичния си участък
}
```

Аналогичен е кода за процеса P1

ВВМУ, курс Администриране на БД Доц. О. Железов

```
P1 : while (1)
{
  interested1 = 1; // P1 вдига флаг за влизане в крит. участък
  turn = 1;
  while (turn == 1 && interested0);
  critical_section1();
  interested1 = 0;
  non_critical_section1();
}
```

Лесно се вижда, че алгоритъмът на Питерсън е коректен. Ако процесите едновременно изпълняват тялото на основните си цикли, то процесът който пръв изпълни присвояването на стойност на turn ще влезе в критичния си участък.

Основният недостатък на алгоритмите на Декер и Питерсън е, че за осъществяване на взаимното изключване се използват цикли на чакане. Това води до неефективност. Много по-ефективно е чакащият процес да бъде блокиран и след това събуден когато е възможно влизането в неговия критичен участък.

ВВМУ, курс Администриране на БД, лектор доц. О. Лелезов

4.7. Семафори. Взаимно изключване и синхронизация чрез семафори.

Семафорите са предложени от Дейкстра през 60-те години за решаване на проблеми при междупроцесни комуникации. В съвременните операционни системи семафорите са средства, които се предоставят от ядрото. Ние ще разгледаме общ вариант на семафорите, така както са били предложени от Дейкстра, без да навлизаме в детайли за конкретната им реализацията.

Семафорът е тип обект, с който се свързва брояч и списък от блокирани процеси. Броячът може да приема само неотрицателни стойности.

Не уточняваме организацията на списъка, но обикновено той е опашка. Ако един процес е в списъка на даден семафор, казваме че процесът е блокиран по този семафор.

Върху семафорите се определят три операции:

инициализация създаване на семафор и задаване на начална стойност на брояча. Списъкът от процесите, блокирани по семафора в началото е празен.

операция P намалява се брояча на семафора, ако е възможно; в противен случай, процесът който изпълнява операцията се блокира по този семафор. Например ако броячът на семафора има стойност 1, при изпълнение на операция P от даден процес броячът ще получи стойност 0, което ще блокира другите процеси, които искат да влязат в критичния участък.

В псевдокод на C операцията P има вида:

```
P (s) { //s е семафор
  if (s > 0) then s--;
  else блокиране_на_процеса_по_семафора_s;
}
```

операция V – ако има процеси, блокирани по семафора, то се събужда точно един от тях; в противен случай, брояча на семафора се увеличава.
В псевдокод на C операцията V има вида:

```
V (s) { //s е семафор
  if (има_процеси_блокирани_по_семафора_s) then
  събуждане_на_един_процес_блокиран_по_семафора_s;
```

```
    else s++;
  }
```

Естествено, ще считаме, че семафорите са общи за процесите и ще ги дефинираме примерно по следния начин: semaphore name = value, въпреки, че това не е коректно в синтаксиса на C (name е името на семафора, value е началната стойност на брояча).

Обикновено, операциите P и V са реализирани като системни примитиви в операционните системи.

Съществено за операциите с даден семафор е тяхната атомарност. С други думи, ако един процес изпълнява операция с даден семафор, то друг процес не може да изпълнява операция в същия момент със същия семафор. Най-простият начин за осигуряване на такова взаимно изключване е чрез машинните команди за разрешаване и забраняване на прекъсване. Тези команди са привилегировани и могат да се използват само в режим ядро.

Взаимно изключване на N процеса чрез двоичен семафор

Взаимното изключване на N процеса се осигурява точно с един семафор. Да разгледаме псевдокода, който се изпълнява от процесите, които се съгласуват чрез семафор:

semaphore mutex = 1; //При стойност 0 процесите се блокират в P операцията

```
P0 : while (1)   //Безкраен цикъл, който се изпълнява от процеса P0
{
   P(mutex);    //Изпълнение на операция P (блокират се останалите процеси)
   critical_section0(); //Изпълнение на критичната секция от P0
   V(mutex);    //Изпълнение на операция V събужда един блокиран процес
   non_critical_section0(); //Изпълнение на некритична секция от процес P0
}
```

Аналогичен е кода, който се изпълнява от останалите процеси. За процеса PN той има вида:

```
PN : while (1)   //Безкраен цикъл, който се изпълнява от процеса PN
{
   P(mutex);    //Изпълнение на операция P (блокират се останалите процеси)
   critical_sectionN();//Изпълнение на критичната секция от PN
```

```
   V (mutex);    //Изпълнение на операция V събужда един блокиран процес
   non_critical_sectionN();//Изпълнение на некритична секция от процес PN
}
```

Всеки процес загражда критичният си участък с операциите P и V. Съществено е, че броячът на семафора mutex се инициализира с 1. Ако един процес влезе в критичния си участък, останалите процеси се блокират в P-операцията преди своя критичен участък, тъй като броячът на семафора mutex е станал 0. Когато излезе от критичния си участък, процесът ще освободи някой блокиран процес, който ще влезе в своя критичен участък. Ако няма блокирани процеси, стойността на брояча на mutex ще се възстанови на 1. Вижда се, че броячът може да приема само стойности 0 и 1. Такъв семафор се нарича двоичен.

Примери за синхронизация чрез семафори

Пример 1: Синхронизиране на два процеса P0 и P1 така, че P0 да се изпълни първи.

Разглеждаме два процеса P0 и P1, които работят паралелно и асинхронно. Искаме даден оператор в първия процес да се изпълни преди даден оператор във втория процес. Предполагаме, че операторите не са част от тялото на цикъл.

Ще решим задачата с един двоичен семафор.

```
semaphore s = 0; //Блокира процесите в операция P(s)
P0 () {
 … оператор1; //P0 изпълнява оператор1
     V (s);         //P0 увеличава s с 1 и така разблокира P1
}
P1 () {
 … P (s);         //P1 изчаква разблокиране (s да стане >0)
    оператор2; //P1 изпълнява оператор2
}
```

Ако оператор1 не е изпълнен, то P1 ще се блокира в P-операцията преди оператор2. След изпълнението на оператор1 P0 ще събуди P1, ако е бил блокиран. В противен случай, P0 ще увеличи броячът на s, което означава, че след това P1 ще премине успешно през P-операцията.

Сега ще решим класическите проблеми производител-потребител и читатели-писатели с използването на семафори и обща памет.

Пример 2: Синхронизация производител-потребител с използване на семафори

Както вече споменахме, производителят изчислява някакви данни, а потребителят ги обработва. Задачата е да се организира коректно тяхната работа, при предположение, че процесите работят паралелно и асинхронно. Данните ще се предават чрез буфер, който е част от общата памет. Този буфер може да заема достатъчен брой елементи, изчислени от производителя. За простота елементите са с фиксиран размер – например цели числа. Организацията на буфера не е съществена.

Четенето и писането в буфера не са елементарни операции и те трябва да са атомарни (т.е. да се изпълняват само изцяло, а не по части). Всъщност, трябва да постигнем следните три ефекта:

- ако буферът е пълен, то производителят да не пише;
- ако буферът е празен, то потребителят да не чете;
- четенето и писането в буфера трябва да са критични участъци.

В решението ще използваме три семафора.

```
shared buffer buf;          //буферът се състои от N елемента
semaphore mutex = 1;    //осигурява взаимното изключване
semaphore empty = N;   //стойността на брояча на empty е броят на
                        //незаетите слотове (позиции) в буфера. При 0 блокира
писането
semaphore full = 0;       //стойността на брояча на full е броят на
                        //заетите слотове в буфера. При 0 блокира четенето
```

```
producer ()
{
   int item;
   while (1) //Безкраен цикъл за производител (писател)
   {
      produce_item (&item);
      P (empty); //Операция P със семафора empty. Блокира при пълен буфер
      P (mutex); // Операция P със семафора mutex. Блокира при mutex=0
      add_item (item); //Производителят записва във буфера
      V (mutex); // Операция V със семафора mutex. Разблокира потребителя
                  // ако е бил блокиран по mutex
      V (full);    // Операция V със семафора full. Разблокира потребителя
                  // ако е бил блокиран поради празен буфер
   }
```

```
}
```

Цикъл с операции за производител

```
consumer()
{
   int item;
   while (1)                  //Безкраен цикъл за потребител (читател)
   {
     P(full);        //Операция P със семафора full. Блокира при празен буфер
     P(mutex);   // Операция P със семафора mutex. Блокира при mutex=0
     get_item (&item); //Потребителят чете елемент от буфера
     V(mutex); ); //Операция V със семафора mutex. Разблокира производителя
                    //ако е бил блокиран по mutex
     V(empty);    //Операция V със семафора empty. Разблокира производителя
                   // ако е бил блокиран поради пълен буфер
    consume_item (item); //Операции с прочетените данни
   }
}
```

Цикъл с операции на потребител

Синхронизацията се осигурява чрез семафорите full и empty.

Ако буферът е пълен и производителят пробва да пише, той ще се блокира в P-операцията на empty. След като потребителят освободи слот в буфера, той ще събуди производителя с V-операцията на empty.

Ако буферът е празен и потребителят пробва да чете, той ще се блокира в P-операцията на full. След като производителят запише елемент в буфера, той ще събуди потребителя с V-операцията на full.

Съществено е, че P-операцията и V-операцията на mutex са вътрешни по отношение на P-операциите и V-операциите на empty и full. Лесно се вижда, че ако разменим местата им може да се достигне до едновременно блокиране на двата процеса.

Частен случай: синхронизация на производител-потребител при буфер, състоящ се само от един елемент (N=1)

В случай че N = 1, т.е. буферът се състои само от един елемент, семафорът mutex става излишен и може да се премахне. Това е така, тъй като след първия запис буферът ще бъде пълен, което ще блокира производителя в операция P(empty) и потребителят може безпрепятствено да чете. Аналогично след първото прочитане буферът ще бъде празен, което блокира

ВВМУ, курс Администриране на БД, лектор доц. О. Желязов

потребителя в операция P(full) и производителят може безпрепятствено да записва. Така при N=1 за синхронизацията се оказват достатъчни само семафорите full и empty.

ВВМУ , курс
Администриране на БД
лектор доц. О.Железов

Пример 3: Читатели-писатели

Разполагаме с някаква база данни. Процесите-читатели могат само да четат данни от базата, докато процесите-писатели могат само да пишат в нея. Естествено, читателите и писателите може да са повече от един.

Синхронизацията в случая означава, че в даден момент могат да работят един или повече читатели или най-много един писател.

В решението се използват два двоични семафора.

```
shared int rcount = 0; //брой на активните читатели
semaphore mutex = 1; //осигурява взаимно изключване
semaphore db = 1; //управлява достъпа до базата данни
```

```
reader()
{ //важи за всички процеси-читатели
   while (1)  //Безкраен цикъл за читател
   {
      //*********** Влизане в първи критичен участък за rcount **********
      P(mutex);  // Операция P със семафора mutex. Блокира другите читатели
      rcount++;  // Отброява текущия читател като активен
      if(rcount == 1) P(db); //Ако читателя е единствен, блокира писателите
      V(mutex); // Операция V със семафора mutex. Разблокира читател

      read_data(); //Прочитане на данни
      //********** Влизане във втори критичен участък за rcount *********
      P(mutex);   //Операция P със семафора mutex. Блокира другите читатели
      rcount--;     //Отброява текущия читател като неактивен
      if (rcount == 0) V(db); //Ако читателя е единствен, разблокира писател
      V(mutex); // Операция V със семафора mutex. Разблокира друг читател
   }
}
```

Код на читател.

Когато читателят е единствен, блокира писателите, разрешава на другите читатели да четат и прочита. След последния читател се разблокира един писател.

Важно е да се обърне внимание на значението на брояча rcount. На практика неговите стойности 1 и 0 определят състоянието на семафора db, който блокира и разблокира писателите. Първият активен читател (за който rcount=1) блокира писателите, а последният активен читател (след който rcount=0) разблокира писателите.

ВТУМУ, курс Администриране на БД, Електронно. О. Железов

Участъците, в които се оперира с брояча rcount са критични за да не се допуска състезание между читателите. Достъпът до критичните участъци – променяне на rcount – се контролира със семафора mutex.

```
writer () { //важи за всички процеси-писатели
   while (1) //Безкраен цикъл за писател
   {
      produce_data(); //Създава данни за записване
      P(db);          // Операция P със семафора db – блокира другите писатели
      write_data();   // Записва данни
      V(db);          // Операция V със семафора db – разблокира един писател
   }
}
```

Код на писател

Достъпът на писателя до критичната секция – операция запис - се контролира със семафора db.

Да предположим, че първият процес, който се изпълнява е читател. Тогава rcount става 1, нулира се брояча на db и след това започва четенето. Ако в този момент започне да се изпълнява писател, той ще се блокира в P-операцията на db. Ако започне да се изпълнява читател, той ще увеличи rcount и няма да проверява семафора db. След приключване на четенето, всеки читател намалява брояча rcount и последният читател изпълнява V-операция върху семафора db. Това води до събуждане на блокиран писател по db или до увеличаване на брояча на db.

Да предположим, че първият процес, който се изпълнява е писател.Тогава той нулира брояча на семафора db и започва писането.Ако в този момент започне да се изпълнява читател, той ще се блокира в P-операцията на db. Докато не бъде събуден, следващите читатели ще се блокират по семафора mutex. Ако започне да се изпълнява писател, той ще се блокира в P-операцията на db. След приключване на писането, писателят ще събуди блокиран писател или читател по семафора db или ще увеличи брояча на db. Ако е събуден читател, той ще изпълни V-операция на mutex, с което ще събуди друг блокиран читател, ако има такъв и т.н.

Всички досега разгледани методи за взаимно изключване и синхронизация чрез семафори или обща памет не са безопасни, в смисъл че отговорността за правилното прилагане на алгоритмите принадлежи на програмиста.

http://www.bg-informatika.com/2011/07/17_21.html Информатика - лекции и материали
http://www.bg-informatika.com/2011/07/22.html 22. Междупроцесни комуникации. Взаимно изключване. Алгоритъм на Декер. Алгоритъм на Питерсън.

4.8. Комуникация между процесите

Изпълнението на процесите и предаването на данни при тяхното завършване почти винаги изискват обмен на информация между тях. Основните схеми за осъществяване на комуникация са две: комуникация чрез обща памет и комуникация чрез съобщения.

4.8.1. Комуникация между процесите чрез съобщения.

Ще разгледаме още един механизъм за междупроцесна комуникация – така наречения метод на съобщенията.

Той е значително по-безопасен от по-горе разгледаните, тъй като отговорността за коректна комуникация между процесите се поема от операционната система.

За да могат два или повече процеса да обменят съобщения, между тях трябва да се създаде комуникационна връзка. Също така са необходими системни примитиви чрез които се обменят съобщения. Най-общо те трябва да изглеждат по следния начин:

```
send (destination, message) //Системен примитив за изпращане на съобщение
receive (source, message).  // Системен примитив за получаване на съобщение
```

Чрез send се изпращат съобщения - аргументът *destination* определя процеса, който трябва да получи съобщението, аргументът *message* е самото съобщение.

Чрез receive се получават съобщения – аргументът *source* определя от кой процес се чака съобщение, аргументът *message* определя къде да се запише съобщението.

По този начин за да се предаде съобщение от един процес към друг процес, първият трябва да изпълни send, а вторият трябва да изпълни receive. В общия случай тези два примитива се изпълняват асинхронно.

При конкретна реализация на механизъм за съобщения между процеси трябва да се решат следните въпроси относно логическите характеристики на комуникационната връзка:

1. как се установява връзка между процесите, които участват в комуникацията;
2. колко процеса могат да комуникират чрез една връзка;
3. колко комуникационни връзки могат да се създават между два процеса;
4. еднопосочна или двупосочна е комуникационната връзка;
5. какъв е капацитетът на комуникационната връзка. (дали може да съхранява съобщения).

Размерът и структурата на съобщенията касаят физическата организация на връзката.

Адресиране на съобщенията

Отговорът на първите четири въпроса зависи от това как се адресират получателя и изпращача в примитивите send и receive.

Директна комуникация

За да се предаде съобщение от процес P към процес Q, процесът P трябва да изпълни send (Q, message), а процесът Q трябва да изпълни receive (P, message), т.е. в примитивите send и receive в явен вид се указва на кого се изпраща съобщение и от кого се получава съобщение. При директната комуникация отговорите на дадените по-горе въпроси са следните:

1. комуникационната връзка се създава автоматично между двата процеса, но всеки от тях трябва да знае идентификатора на другия;
2. комуникационна връзка може да се създаде само между два процеса;
3. между два процеса може да съществува точно една комуникационна връзка;
4. комуникационната връзка е двупосочна – например, възможно е изпращане на потвърждение за получено съобщение от процес P към процес Q

send (Q, message); receive (P, message);
receive (Q, message); send (P, “acknowledgement”).

Косвена комуникация (комуникация с пощенска кутия)

При косвената комуникация процесите обменят съобщения посредством някаква пощенска кутия. В даден момент може да има много пощенски кутии, така че те трябва да имат идентификатори, за да са различими.

На мястото на source и destination в примитивите send и receive трябва да стои идентификатор на пощенска кутия. Изпращане на съобщение от процес P към процес Q през пощенската кутия A става така:

```
процес P              процес Q
send (A, message);    receive (A, message).
```

При косвената комуникация отговорите на дадените по-горе въпроси са следните:

1. комуникационна връзка се установява когато процесите използват една и съща пощенска кутия;
2. възможно е повече от два процеса да комуникират през една пощенска кутия, стига да знаят нейния идентификатор;
3. възможно е два процеса да комуникират през повече от една пощенски кутии;
4. връзката може да е еднопосочна или двупосочна.

При реализация на този метод трябва да се определи как се създават и унищожават и как се идентифицират пощенските кутии. Освен това трябва да се реши въпросът за собственика на пощенската кутия и затова какви са правата на процесите при използване на пощенската кутия. Една възможна реализация е следната: създаването и унищожаването на пощенска кутия става чрез системни примитиви.

Собственик на пощенската кутия става процесът, който е изпълнил системния примитив за създаване, той определя какви ще са правата на останалите процеси по отношение на пощенската кутия и само той може да унищожава пощенската кутия.

Буфериране на съобщенията

Отговорът на петия въпрос (какъв е капацитетът на комуникационната връзка) зависи от това дали се буферират съобщенията или не.

Комуникация без буфериране: При нея изпратените, но неполучени съобщения не се съхраняват. Това означава, че ако send се изпълни преди receive, то процесът, който изпраща съобщението трябва да се блокира, докато процесът, който трябва да получи съобщението не изпълни receive. С

ВВМУ Курс Администриране на БД лектор доц. О. Железов

други думи, комуникиращите процеси синхронизират работата си в момента на предаване на съобщението. Предимството на този механизъм е лесната реализация, недостатъкът е, че се ограничава свободата на процесите.

Комуникация с буфериране: При нея изпратените, но неполучени съобщения се съхраняват в буфер.
Този буфер може да поеме ограничен брой (обем) неполучени съобщения. Ако send се изпълни преди receive и буферът не е пълен, то процесът, който изпраща съобщение не се блокира, а записва съобщението в буфера. Ако receive се изпълни преди send и буферът не е празен, то процесът, който трябва да получи съобщение не се блокира, а чете съобщението от буфера. По този начин се дава по-голяма свобода на процесите. Един недостатък на този метод е, че процесът, който е изпълнил send не може да е сигурен, че съобщението му е получено.
В случаите когато това може да доведе до проблеми, процесите, които получават съобщение трябва да изпращат обратно потвърждение.

Ще разгледаме накратко две реализации на механизма на съобщенията – в ОС MINIX и UNIX/LINUX.

Операционната система MINIX поддържа съобщения. Използва се директна комуникация и механизъм без буфериране. Има няколко типа съобщения, като всички са с фиксиран размер. Съобщенията в MINIX се използват за служебни цели от ядрото. Те са необходими, тъй като голяма част от ядрото в MINIX работи като отделни процеси.

В UNIX/LINUX също могат да се използват съобщения. Там е реализирана косвена комуникация с автоматично буфериране.
Пощенските кутии в термините на UNIX/LINUX се наричат опашки на съобщенията (message queue). Всяка опашка (message queue) има външен идентификатор, който е цяло число и се нарича ключ. Всеки процес, който знае идентификатора на някакъв message queue и има права да го използва може да изпраща или да получава съобщения през него. Това означава, че е възможна комуникация и между неродствени процеси.

Message queue се създава (отваря) чрез системния примитив int msgget (key_t key, int flag). Ако съществува message queue с външен идентификатор key, то процесът, изпълнил msgget, получава неговия вътрешен идентификатор, който се използва от функциите за работа с message queue. Ако не съществува message queue с външен идентификатор key, то се създава нов message queue. Негов собственик е процеса, изпълнил msgget, а правата на останалите процеси се определят от аргумента flag. В младшите 9 бита на

flag са записани правата на собственика, групата и останалите. Право r означава, че процес може да получава съобщения, право w означава, че процес може да изпраща съобщения. Примитивът msgget връща -1 при грешка.

Съобщения се изпращат със системния примитив msgsnd, а се получават със системния примитив msgrcv. За извършване на някакви други операции върху message queue, както и за унищожаване на message queue се използва системния примитив msgcntl. Право да унищожава message queue има само неговия собственик. Ако собственикът завърши, то message queue не се унищожава – той може да се използва и от други процеси, които имат права.

Структурата на съобщенията в UNIX/LINUX е следната:

struct msgbuf { long mtype; char mtext[N]; }.

където:
Полето mtype съдържа тип на съобщението – цяло положително число.
Полето mtext съдържа самото съобщение (N е неговата максимална дължина).

Системният примитив msgrcv имат аргумент, който посочва какъв тип съобщение ще се получава. Ако този аргумент е 0, то типа на съобщението няма значение при получаването.
Например, по този начин може да се реализира двупосочна комуникация между два процеса P и Q чрез един message queue – съобщенията от P към Q имат тип 1, съобщенията от Q към P имат тип 2.

Да отбележим, че програмните канали могат да се разглеждат като механизъм за съобщения с косвена адресация и автоматично буфериране. В програмните канали, обаче, между отделните съобщения няма ясни граници.

http://www.bg-informatika.com/2011/07/24_21.html Съобщения.

4.9. Нишки (POSIX threads)

Многозадачните ОС са изградени върху концепцията за процесите. Всеки процес се дефинира с:

1) Брояч на инструкции

2) Регистри
3) Стек
4) Собствена адресно пространство

Освен това процесите при изпълнението си използват и други системни ресурси, които в някои случаи трябва да се използват паралелно и да се поделят. Така в един процес могат да се изпълняват паралелно няколко потока от инструкции, които е прието да се наричат нишки (threads).

Нишките са много полезни тогава, когато процесът трябва да изпълнява няколко независими задачи, особено в случаите, когато изпълнението на някоя от задачите може да бъде блокирано, а останалите трябва да продължат да се изпълняват.

Например в текстообработваща програма, една от нишките във фонов режим да извършва проверка на граматиката и правописа, докато друга нишка очаква въвеждане от клавиатурата, трета зарежда изображения от дисковото устройство, и четвърта автоматично съхранява резервни копия на документа.

Друг пример за многонишкова програма е WEB сървърът – Изпълнението на много нишки позволява много заявки да се обслужват едновременно, вместо да се изпълняват заявките последователно или да се стартира отделен процес за всяка заявка (което би изисквало много повече ресурси от ОС).

Нишките са сравнително нова концепция в операционните системи. Без използване на нишки всеки процес определя строга последователност от действия. С въвеждането на нишките процесът вече не е последователен, а включва няколко нишки, които могат да се изпълняват асинхронно. Процес, който има повече от една нишка става конкурентен сам със себе си.

Всяка нишка се дефинира с:
1) Брояч на инструкции
2) Регистри
3) Стек
4) Няма собствено адресно пространство (използва това на процеса)

От сравнението на дефиницията на процес и нишка се вижда основната разлика: нишките нямат собствено адресно пространство. Всички останали ресурси на процеса (освен изброентите по-горе) са общи за всички негови нишки. От това следва, че нишките са по-“икономични” за създаване и поддържане от процесите и превключването между тях се осъществява по-лесно. Нишките дават възможност да се реализират ефективни конкурентни приложения.

Важно е да се запомни, че използването на адресната област от потоците не се контролира от ОС. Този контрол е задължение на програмиста.

Управление на нишките

Съществуват два подхода при управление на нишките: управление чрез системна библиотека извън ядрото и управление непосредствено от ядрото на ОС. В съответствие с това нишките се класивицират като нишки на ядрото (kernel thread) и потребителски нишки (user thread).

Потребителски нишки: управление чрез системна библиотека извън ядрото

Управлението на потребителските нишки се прави от приложението. Ядрото не знае за съществуването на тези нишки.

Този подход има като недостатък неравномерно разпределение на процесорното време между нишките. Така ако един процес има само една нишка, той ще и предостави цялото процесорно време, с което разполага. Ако друг процес има две нишки, той ще им предостави по половината от своето процесорно време. Така при еднакъв приоритет на двата процеса, нишките на втория процес ще разполагат с два пъти по-малко процесорно време.

Нишки на ядрото: управление непосредствено от ядрото на ОС

Ядрото може да разпредели множество нишки от един процес на различни процесори или процесорни ядра. Така по-ефективно се използват съвременните многоядрени процесори. Ако една нишка в един процес е блокирана, ядрото може да разпредели за изпълнение друга.

Управлението от ядрото на ОС дава по равно време на всяка нишка. Негов недостатък е по-голямото време за превключване на нишките, което използва в тоя случай ядрото. Съотношението на времето за превключване за двата подхода е от порядъка на 1:10. Това увеличено време за превключване се компенсира от по-точното разпределение на процесорното време.

Може да се каже, че при по-малко на брой процеси и нишки по-добър е първия подход, докато при по-сложни задачи с по-голям брой паралелни процеси и нишки по-добър е втория подход.

Изпълнение на нишки в ОС Windows

Нишките могат да се изпълнят на всеки от процесорите, но приложението може да ограничи този ефект.

Процесорен афинитет

Процесорният афинитет (Processor affinity или CPU pinning), дава възможност за обвързване или необвързване на процес или нишка към определен процесор или група процесори, така, че процесорът или нишката да се изпълнява само на този процесор или група процесори. Тази планировка може да се разглежда като модификация на алгоритъма, при който се използва опашка от задачи, които се разпределят на процесорите в симетрична многопроцесорна ОС. Всяка задача в опашката има атрибут, който задава предпочитаният процесор. При разпределение на задачите, всяка задача се разпределя приоритетно на предпочитаният процесор. Могат да се определят два вида афинитет

Слаб афинитет (Soft Affinity)
Разпределителят (dispatcher)опитва да присвои готова нишка на същият процесор от последното изпълнение. Това подпомага преизползването на данни, които са кеширани в паметта на процесора от последното изпълнение на нишката.
Твърд афинитет (Hard Affinity)
Приложението изисква изпълнението на нишките да бъде на определен процесор.

Различните операционни системи предлагат специфични програмни средства за задаване на процесорен афинитет на процесите и нишките. Например в ОС Windows за задаване на афинитет се използват системните функции SetThreadAffinityMask и SetProcessAffinityMask. В ОС Linux афинитет може да се зададе със системната функция sched_setaffinity или, при стартиране на програма от командния ред да се използва командата taskset.

http://www.bg-informatika.com/2011/07/25-posix-threads.html Нишки (POSIX threads).
http://littledaemons.wordpress.com/2009/01/22/cpu-affinity-and-taskset/ CPU Affinity and taskset
http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/4_Threads.html Threads

4.10. Мъртва хватка (deadlock). Предотвратяване на дедлок. Заобикаляне на дедлок – алгоритъм на банкера. Откриване и възстановяване от дедлок.

Мъртвата хватка на процеси (дедлок) е неприятно събитие, което може да се случи в многопроцесни системи, когато няколко процеса се състезават да използват общи ресурси.

Казваме, че едно множество от процеси се намира в дедлок, ако всички процеси са в състояние блокиран и всеки от процесите чака настъпване на събитие, което може да бъде предизвикано само от друг процес в множеството. Аналогично се дефинира дедлок и при нишки. Ясно е, че ако системата не предприеме мерки, процесите, които се намират в дедлок ще останат вечно блокирани. Обикновено събитието, което чакат процесите е предоставяне на ресурс от системата.

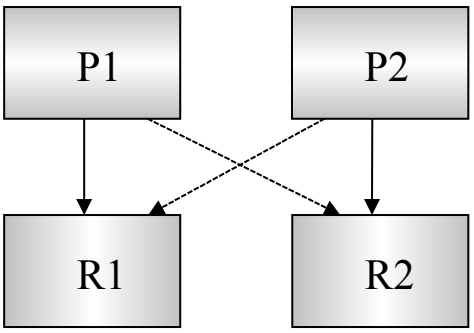
Условия за възникване на мъртва хватка

Кофман формулира четири условия, които са необходими за възникване на дедлок в системата:

1)	Взаимно изключване	Процесите заявяват монополно управление на ресурсите, т.е. заявен ресурс се използва само от един процес в даден момент.
2)	Очакване на ресурси	Процесите задържат вече отделените им ресурси и очакват да получат нови.
3)	Непреразпределение	Ресурси не могат да бъдат отнети, тъй като те се освобождават от процеса само след изпълнение на програмата.
4)	Кръгово очакване	Съществува кръгова верига от процеси, всеки от които очаква ресурс, задържан от предшестващ процес.

Ще дадем един прост пример, в който се получава дедлок. Нека имаме два процеса P1 и P2 и два ресурса R1 и R2. Предполагаме, че ресурсите R1 и R2 се използват монополно от процесите.

ВВМУ „Н. Й. Вапцаров" Администриране на ИС Лектор доц. О. Железов



Фиг. 1 Взаимна блокировка на два процеса

Нека в някакъв момент P1 е получил ресурса R1 и P2 е получил ресурса R2. Нека в някакъв следващ момент, P1 поиска достъп до ресурса R2. Тогава P1 ще бъде блокиран, тъй като R2 все още се използва от P2, при това монополно. Нека след това процесът P2, преди да е освободил ресурса R2, поиска достъп до ресурса R1.

Тогава P2 се блокира, тъй като R1 все още се използва от P1, въпреки че P1 се намира в състояние блокиран. Така P1 и P2 са блокирани и чакат събитие, което може да се предизвика само от P2 и P1 съответно, което означава че P1 и P2 се намират в дедлок.В проблема дедлок участват процеси, които се състезават да използват един или няколко типове ресурси. Типът на ресурсите не е от значение – те могат да са устройства, системни структури, файлове, части от файлове и др. Възможно е да има по няколко ресурса от даден тип. Съществено е, че при проблема дедлок ресурсите са обекти, които се използват *монополно* от процесите.

Работата на един процес с даден ресурс се разделя на три етапа:

1) заявка за ресурса (request);
2) използване на ресурса (use);
3) освобождаване на ресурса (release).

Тези три стъпки могат да са отделни системни примитиви, но може да са реализирани и чрез един системен примитив. Това зависи от типа на ресурса. Ако един процес направи заявка за ресурс, който не може в дадения момент да се осигури от системата, той бива блокиран и се събужда едва когато друг процес, който използва въпросния ресурс го освободи.

Предпазване от мъртва хватка (дедлок) чрез нарушаване на условията за възникването и.

За да се предпази системата от дедлок трябва да се осигури нарушаване на някое от изброените по-горе четири условия за получаване на такава взаимна блокоривка на процесите:

Първото условие (*взаимно изключване*) не е удачно да се нарушава, тъй като едновременното използване на ресурси по правило създава проблеми.

Второто условие (*очакване на ресурси*) за възникване на дедлок може да се наруши ако всички необходими за даден процес ресурси се заявяват наведнъж при неговото стартиране. Това обаче води до много неефективно използване на оперативната памет и до по-големи изчаквания за получаване на ресурси. Модификация на този подход е стъпково задаване на ресурси, като числото на стъпките се свежда до възможния минимум. В тоя случай обаче при недобро планиране може да се стигне но безкрайно отлагане.

Третото условие (*непреразпределение*) за възникване на дедлок може да се наруши ако процесът, получил ресурс, който не използва, го освободи за ползване от друг процес. Така се избягва мъртвата хватка, но се увеличава процесорното време за управление на ресурсите.

Четвъртото условие (*кръгово очакване*) може да се наруши при подреждане на ресурсите по зададени номера в нарастващ ред. Процес, заявил даден ресурс може да заяви друг само ако той е с по-голям номер. Процес може да освободи ресурс само ако е освободил преди това заетите от него ресурси с по-малки номера. Този метод обаче много усложнява писането на програми, понеже в процеса на изпълнение състоянието на ресурсите се променя и планирането може да се окаже невъзможно.

Предпазване от мъртва хватка (дедлок) на етап планиране на мултипрограмен режим.

Използват се два алгоритъма за избягване на ситуациите, довеждащи до мъртва хватка (дедлок):

1) Отхвърляне на процеса при неговото стартиране поради недостатъчни ресурси:

   Ако ОС определи, че свободните системни ресурси са недостатъчни за изпълнението на даден процес, тя може да го отхвърли преди стартирането му. Така ще се избегне възможността след стартиране той да се окаже блокиран.

2) Отхвърляне на заявки за разпределение на ресурс:

   Ако съществува поне един път за изпълнение на процеса, ОС е в безопасно състояние. В противен случай състоянието се счита за опасно и нови заявки за изпълнение на процеси не се приемат. При този метод се приема, че ако ресурсите са достатъчни за изпълнение на всички процеси, ОС ще избегне мъртвата хватка.

   Недостатък на този алгоритъм е неизменният брой ресурси и процеси в даден момент. При всяка промяна трябва да се правят нови разчети. Освен това освобождаването на ресурси в реално време не се отчита от алгоритъма.

Заобикаляне на дедлок – алгоритъм на банкера

При този метод първите три условия на Кофман (*взаимно изключване, очакване на ресурси и непреразпределение*) не се нарушават. Това означава, че дедлок принципно е възможен, но се заобикаля по подходящ начин. Ако при дадена заявка на процес за ресурс системата прецени, че има опасност от дедлок, вследствие изпълнение на заявката, то процесът се блокира и ресурсът не му се предоставя.

Алгоритмите за заобикаляне на дедлок се базират на информация за това как процесите ще използват ресурси по време на своето изпълнение. Всеки процес трябва да осигури на системата тази информация преди да започне своето изпълнение.

Най-популярният алгоритъм за заобикаляне на дедлок е така наречения алгоритъм на банкера. Той е предложен от Дейкстра за един тип ресурс и по-късно обобщен от Хаберман за произволен брой типове ресурси. Информацията, нужна на този алгоритъм е следната:

1. Максимален брой ресурси от всеки тип за всеки процес – предоставя се от процесите преди да започнат изпълнението си.

2. Брой свободни ресурси от всеки тип – в началото се предоставя от системата, а след това се поддържа в зависимост от разпределението на ресурсите.

3. Брой разпределени ресурси от всеки тип за всеки процес – поддържа се във всеки момент от системата. Този брой никога не надвишава заявеният максимален брой от процесите.

4. Брой ресурси от всеки тип, които процесът може да поиска – получава се като от максималния брой ресурси, заявен от процеса извадим броя на разпределените ресурси за този процес.

Естествено, максималният брой ресурси за всеки процес трябва да е по-малък от броя на наличните ресурси в системата.

Информация, нужна за изпълнение на алгоритъма на банкера

Казваме, че системата се намира в надеждно състояние (safe state) на разпределение на ресурсите, ако съществува наредба P1, P2, …, Pn на процесите в системата, такава че за всеки тип ресурс и за всеки процес Pi, броят на ресурсите, които Pi може да поиска е по-малък от сумата на броя свободни ресурси, намалена с броя разпределени ресурси за процесите P1, P2, …, Pi-1. С други думи, ако процесите се изпълняват в тази последователност, то системата може безопасно да удоволетвори заявките на всеки процес. Когато за текущото разпределение на ресурсите не съществува такава последователност, казваме че системата се намира в ненадеждно състояние (unsafe state).

Алгоритъмът на банкера анализира всяка заявка и я удоволетворява само ако системата ще премине в надеждно състояние. В противен случай, процесът се блокира и чака освобождаване на ресурси.

5. Структура на операционната система

Една операционна система се състои от две основни части:

- Ядро (на английски: Kernel)
- Обвивка (на английски: Shell)

Ядрото се грижи за абсолютно всички процеси, които се изпълняват, както и за комуникацията с наличните устройства. То осигурява работата на обвивката и на приложните програми.
Обвивката служи за връзка между потребителя и ядрото. Тя може да бъде както графична, така и команден ред. ОС използва и друг вид системен софтуер, който обаче не е част от самата операционна система — драйверите. Те служат за връзка между ядрото на операционната система и съответните физически устройства. Самата ОС има вградени драйвери за определени устройства като процесор, временна памет, твърд диск и др., които осигуряват нейната работа.

5.1. Примитиви и процеси

Критерият за различаване на процесите, които принадлежат на ядрото от тези, които принадлежат на обвивката е дали те се изпълняват последователно или паралелно. Процесите, принадлежащи на ядрото се наричат *примитиви* и се изпълняват строго последователно, докато процесите, чрез които се изпълняват като системни или потребителски програми се изпълняват паралелно.

Примитивите се извикват от процесите така, както се извикват подпрограми, но със съхраняване на съдържанието на регистрите на процеса. Извикващата и извикваната програма се изпълняват строго последователно. Изпълнението на примитивите може да се разглежда като изпълнение на “сложни” команди в рамките на процеса.

Най-разпространеният начин за извикване на примитиви е изпълнението на команда “софтуерно прекъсване”, при което процесът съхранява състоянието си в стековата памет и след това управлението се предава на извикания примитив. След изпълнението на примитива се изпълнява

процесорната команда “връщане от прекъсване” при което се възстановява състоянието на регистрите на процеса и той продължава изпълнението си.

Потребителските програми не могат да извикват директно примитиви (от съображения за сигурност). Достъпът до примитивите става чрез системните функции на обвивката на ОС.

Освен синхронните извиквания на функции на ядрото чрез софтуерно прекъсване, функции на ядрото могат да се извикват и асинхронно, например чрез таймери и външни сигнали за прекъсване.

5.2. Управление на периферните устройства

За осигуряване на работата с наличните периферни устройства, за всяко едно от тях, в състава на ОС има специална програма, наречена драйвер. Или - драйвер е специализирана програма за конкретно периферно устройство, която синхронизира и управлява работата на компютъра с външното устройство. Тя се инсталира задължително при включване на всяко периферно устройство към КС. В процеса на зареждане някои съвременни ОС имат възможност сами да установят факта, че в компютърната система е инсталирано ново периферно устройство и подканят потребителя да избере подходящ драйвер за управлението му.

5.3. Управление на данните

Организирана съвкупност от потребителски данни, на която е дадено име, се нарича файл. Имената на файловете играят ролята на ключ за достъп до техните данни. Потребителите се освобождават от грижата да познават физическото устройство на дисковете и как се разполагат данните върху тях.

Тази част от ОС, която организира съхранението, достъпа, разделянето и защитата на файловете, се нарича файлова система на ОС. В компютъра се съхраняват стотици файлове. Затова файловете се обединяват в логически групи, наречени справочници, каталози, директории или папки. Тези четири думи са синоними в информатиката за обозначаване на група от файлове, които изграждат файловата структура. Например в Windows се използва папки.

Или, за да може файловата система правилно да организира достъпа до файловете, ОС дава възможност файловете да се групират в справочници/каталози/. Всеки справочник може да съдържа и подсправочници.

Файловете и каталозите на потребителите, разположени на един диск, образуват йерархична дървовидна структура. Тя има следните свойства:

1) Във всеки диск съществува само един главен каталог, който не е включен в никой друг от каталозите на диска;
2) Всеки файл/подкаталог участва в един-единствен каталог, като даден каталог не може да съдържа два файла/подкаталога с едно и също име;

ВВМУ , курс
Администриране на БД
лектор доц. О.Железов

6. Операционна система MS-DOS

DOS (Disk operating system) е най-разпространената и използвана някога програма. Голяма част от потребителите на КС не я възприемат като самостоятелна програма, а като вградена в компютъра система от управляващи команди. DOS е най-използвана операционна система за PC. Разработена е от компанията Microsoft, а по известните версии са DOS 3.1; DOS 3.3; DOS 5.0; DOS 6.0; DOS 7.

DOS като сравнително проста операционна система изисква много малко компютърни ресурси за разлика от съвременните операционни системи (Windows), за чиято работа се изразходват огромни ресурси (памет). Освен това DOS е система, която е с по-малка обвивка е и по-близо по функционални възможности до управлението на хардуерните елементи. Това означава, че за системните програмисти и специалистите по поддръжка на КС, понякога DOS е по-лесният (или единственият) начин за разрешаване на някои проблеми с хардуера на КС. Така например, когато трябва да се обнови BIOS системата на някои от елементите, най лесният начин е това да стане от DOS.

6.1. Структура на DOS 6.5 :файлова система, входно-изходни обработки, команден блок, конфигурациони файлове

DOS (Disk operating system) е най-разпространената и използвана някога програма. Голяма част от потребителите на КС не я възприемат като програма, а като вградена в компютъра система от управляващи команди. DOS е най-използвана операционна система за PC. Разработена е от компанията Microsoft, а по известните версии са DOS 3.1; DOS 3.3; DOS 5.0; DOS 6.0; DOS 7.

6.2. Функции на MSDOS.

DOS изпълнява три основни функции:

- Управлява ресурсите на ОС
- Осигурява интерфейса между потребителите и компютърната система (КС).
- Изпълнява инструкции за управление на информацията

6.3. Ядро на MSDOS.

Под ядро (kernel) се разбира основната, програмна част на операционната система. Ядрото на DOS се състои от две, а в последните версии от три компоненти, всяка от които се намира в отделен файл. За системите MS DOS тези файлове са с атрибути скрит (hidden), системен (system) и само за четене (read only) и имат наименования IO.SYS, MSDOS.SYS и DBLSPACE.BIN. В DOS системите на други фирми, наименованията на тези файлове са други. Например, за системите на IBM наименованието на файловете от ядрото на системата са IBMBIO.COM и IBMDOS.COM. Обикновено тези файлове се разполагат непосредствено след основния каталог в дисковото пространство на системния диск. Независимо от вида на дисковия носител (дискета или твърд диск) тези файлове трябва да бъдат записани на фиксирани места след началния (BOOT) сектор, който съдържа програмата за начално зареждане на DOS.

Файлът IO.SYS съдържа расширение на базовата входно-изходна система (BIOS).

Файлът MSDOS.SYS съдържа програмите за обработка на прекъсвания.

Файлът DBLSPACE.BIN обезпечава на MS-DOS достъп до компресирани дискове.

6.4. Управление на оперативната памет

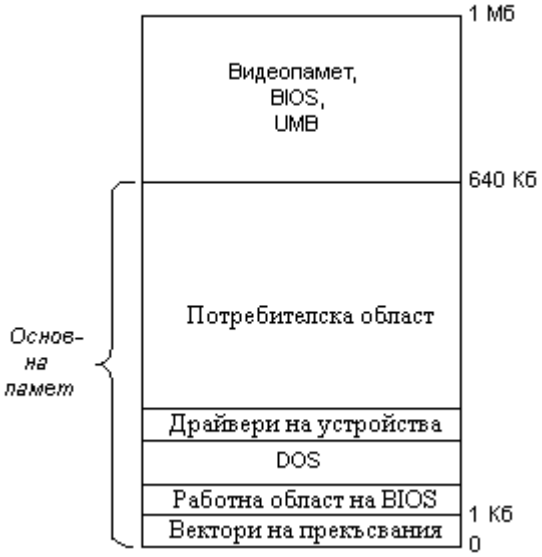
MS-DOS е операционната система, работещ в реален режим на процесори i86. Това предполага използването на адресно пространство с размер само от 1 Mb RAM. В действителност, на компютри IBM горна част от адресното пространство е заета с BIOS и видеопаметта. Тази горна част от адресното пространство включва и разпръснати блокове оперативна памет, обозначавана като UMB (Upper Memory Blocks “горни блокове памет).

Адресът в реален режим се записва във формат [сегмент : отместване], като в случая сегментът не е селектор, адресиращ ред в таблицата на сегментите, а просто номер на параграф от паметта с размер 16 байта. Поради това може да се счита, че в MS-DOS се използват само физически адреси.

По принцип, програмите, работещи под MS-DOS, могат да получат достъп до памет над границата от 1 мегабайт, но за това е необходим специален драйвер за разширена памет.

Тъй като MSDOS е еднозадачна операционна система, в нея не се извършва разпределение на паметта между различни процеси, както това се

прави в многозадачните ОС като Windows и Unix/Linux. Разпределението на паметта в ОС MSDOS е показано на следващата фигура:



Фиг. 2 Разпределение на оперативната памет в ОС MSDOS

6.5. Файлова система на DOS

Всеки файл в DOS се идентифицира с определено име, което е комбинация от символи и цифри. Името съдържа две части: основна част, която представлява символен низ с не повече от 8 символа и разширение – низ с не повече от 3 символа. Разширението не е задължително. Между основната част и разширението се поставя знак ‘.’ – точка. В по-голяма част от случаите разширението на файла идентифицира типа му. Така например, изпълнимите файлове (програми, които се изпълняват) се идентифицират с разширение ‘.exe’; програмните файлове на C++ – с ‘.cpp’; файловете с текст – ‘.doc’ или ‘.txt’ .

DOS се обръща към дисковите устройства като към отделни томове (книги). Те се идентифицират с букви от латинската азбука: A, B, C, D, E, F…. Обикновено A и B са означения за флопидисковите устройства; C, D и т.н. – за твърди дискове (или логически дялове от твърд диск) и след това се означават CD-ROM устройствата. Разделянето на твърдия диск на логически дялове беше въведено поради ограниченията в ранните версии на DOS за адресиране на прстранството за съхраняване на информацията. Нарастването на обема на твърдите дискове се извършваше с по-големи темпове отколкоте развитието на операционната система.

DOS операционната система се съхранява на диск (дискета или твърд диск) и е ориентирана за работа с дисковите устройства. В DOS основната

единица за съхраняване на информацията се явява секторът. Секторът съдържа 512 байта информация. Секторът може да има и друг размер, но винаги кратен на някаква степен на 2 по-голяма от 9 (по-голям от 512 байта). За по-големи сектори от 512 байта е необходим специална програма (драйвер). Физическото разделяне на информацията на сектори се извършва от контролера на твърдия диск. Контролът на отделните сектори и номерирането им се извършва от драйвера на диска. Функцията на DOS е да превърне обръщението към даден файл (по име) в обръщение към конкретен сектор в логическата последователност от 0 до N (брой на секторите). Контролерът и BIOS превръщат този номер в логическата последователност във физически адрес (четяща глава, цилиндър, сектор).

Кеширането на диска означава използване на буфер, разположен между диска и RAM-паметта. В него временно се съхраняват данни от диска, до които има вероятност да бъде поискан достъп. Тъй като външната памет е много по-бавна от използваната за кеш, то кеширането на диска позволява да се ускори достъпа до данни върху него.

6.6. Физическа структура на диск

Физическата структура на магнитен диск за съхраняване на информацията е показана на следващата фигура (П. Нортон, стр 221).

```
                        Disk Editor
   Object  Edit  Link  View  Info  Tools  Help
            Description          Boot Record Data     DOS Reports
Physical Sector: Cyl 0, Side 0, Sector 1

                                OEM ID: MSWIN4.1
                      Bytes per sector: 512                    0
                   Sectors per cluster: 1                      0
          Reserved sectors at beginning: 1                     0
                           FAT Copies: 2                       0
                 Root directory entries: 224                   0
                  Total sectors on disk: 2880                  0
                  Media descriptor byte: F0 Hex
                        Sectors per FAT: 9                     0
                    Sectors per track: 18
                                Sides: 2
                 Special hidden sectors: 0
            Big total number of sectors: (Unused)
                  Physical drive number: 0
          Extended Boot Record Signature: 29 Hex
                   Volume Serial Number: 2D028441 Hex
```

Отделните области върху повърхността на диска са секторите. Съвкупността от всички сектори, разположени на едно и също разстояние от

оста на въртене на диска се нарича *пътечка* или писта (track). Броят на секторите на една пътечка е различен за различните дискове. В съвременните дискови устройства информацията се записва от двете страни на диска. Твърдите дискове имат по няколко диска, които образуват пакет от дискове. За всяка повърхност е комплектована отделна магнитна глава за запис и четене (Head) и повърхностите и, съответно, магнитните глави са номерирани поред от горе надолу, като се започне от 0. Всички пътечки, намиращи се едно и също разстояние от оста на въртене върху всички повърхности образуват цилиндър (Cylinder). Цилиндрите се номерират от периферията към вътрешността, като се започва от 0.

За да може да се използва един диск за съхраняване на информация, той трябва да е форматиран. На току що произведен магнитен диск не съществува никаква информация. Такъв диск не може да се използва директно от никоя операционна система. За да стане подходящ за употреба с диска трябва да се извършат две или три последователни операции.

- Първата от тях е физическо форматиране на диска. При тази операция на новия диск се маркират отделните пътечки и се поставят специални отметки за обозначаване на началото на всеки сектор върху диска. Всеки сектор съдържа идентификационен участък, указващ физическия адрес (положението) на сектора върху повърхността на диска (номер на пътечка, номер глава и номер на сектор) и спецаилен маркер, отбелязващ края на участъка за данни на сектора. Между отделните сектори се оставят интервали.
- Втората стъпка при подготовката на диска за употреба е логическото форматиране (за твърдите дискове това е третата стъпка). В процеса на тази операция се създават записи върху диска, съдържащи специална информация, необходима на DOS за работа с диска. Тази информация е оформена в три таблици:

  1) Таблица за разположението на файловете (FAT – File Allocation Table)
  2) Резервно копие на FAT (backup FAT).
  3) Основен каталог (Root directory).

  Също така в първия сектор на диска се записва BOOT информацията, необходима за стартиране на операционната система, ако дискът е системен.
- За твърдите дискове между физическото и логическо форматиране съществува една междинна операция – създаване на таблица с дяловете (Partition table). Тази операция е необходима, дори и да не се разделя дискът на дялове. Тъй като производителите на дискове произвеждат унифицирани изделия за много широк диапазон на използване с различни операционни системи и различни изисквания за разделяне на дисковото

пространство, логическото форматиране и разделянето на твърдите дискове на логически дялове е оставено да се извършва от потребителите или от фирмите, които асемблират компютърните системи.

Логическата структура на магнитен диск форматиран с DOS файлова система, има вида, показан на фиг. 3.3.8.

http://bg.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0
http://tuj.asenevtsi.com/ST2008/ST009.htm Програмно осигуряване © Христо Тужаров, 2008 Структура на операционната система
http://www.shtrakov.net/Phys_CPU_Arch/Lect_11.pdf Операционна система DOS.

6.7. BOOT сектор

BOOT секторът съдържа необходимата информация за достъп до отделните области от паметта на дадено запомнящо устройство. DOS създава този сектор по време на форматиране на диска. BOOT секторите винаги имат една и съща структура и винаги са разположени в сектор 0 (най-външния цилиндър).

Терминът ‘BOOT’ се използва, защото DOS се стартира от този сектор (‘’boot’ – ва се т.е обува се). DOS не се съхранява в паметта на компютъра (ROM), той се зарежда от диск при стартиране на компютърната система. След включване на компютъра, BIOS поема инициализацията на системата. След първоначалните тестове на компонентите, той зарежда физиеския сектор 0 от флопидиск или от твърдия диск в паметта (BIOS работи с физически адреси, защото операционната система и необходимите драйвери още не са заредени). Голяма част от BOOT сектора заема програма, използвана за стартиране на компютъра (ако дискът е системен). След зареждане на BOOT сектора, BIOS предава управлението на работа на заредената BOOT програма. От нея се прочита определена конфигурационна информация и се преминава към ‘Boot Start’. Тази програма управлява зареждането и стартирането на DOS ядрото.

В BOOT сектора се съдържа така нареченият блок параметри на BIOS, с някои много важни за работата на DOS данни. Тази информация съдържа броя на байтове в един сектор, общия брой на сектори върху диска, типа и броя на копията на FAT (File Allocation Table), броя сектори, заемаща FAT, броя сектори за основния каталог и някои други параметри на диска.

6.8. Дялове върху твърдия диск. Master Boot Record

Основната разлика между флопидиска и твърдия диск се явява възможността при твърдия диск да се формират повече от един логически дял (флопидискът има само един логически дял). Върху твърдия диск могат да се формират отделни части, които биват: първични логически томове (primary logical volumes), скрити (hidden) или разширени (extended) дялове с много логчески томове. Всеки от тези раздели може да бъде с нулев размер, но BIOS предполага, че всички те са определени, и преди да започне работа с диска, търси информация за тяхното определяне. Това се извършва като в първия сектор на всеки твърд диск се разполага една проста програма и информационен файл със стандартен формат. Тази програма заедно с информационния файл образуват така наречения Master Boot Record (MBR) – главен запис за зареждане. Информационният файл на този запис съдържа така наречената Partition table (таблица с дяловете). В тази таблца са описани дяловете, на които е разделен диска, какво пространство заемат (от сектор до сектор) и е отбелязан дялът, който съдържа операционната система. По-долу е показана таблица с дяловете за твърд диск с 20 GB памет, разделен на два дяла (PRI DOS) – том с наименование C и съдържащ операционната система (Status ‘A’ – Active) и допълнителен дял (EXT DOS) – наименование D.

```
Display Partition Information
Current fixed disk drive: 1
Partition Status Type Volume Label Mbytes System Usage
C: 1 A PRI DOS 10001 FAT32 51%
2 EXT DOS 9539 49%
Total disk space is 19540 Mbytes (1 Mbyte = 1048576 bytes)
```

Основни първични (primary), разширени и логически дялове

По-голямата част от BIOS системи могат да зареждат операционната система само от логическия дял на твърдия диск, означен с буква C. Изключение правят КС, използващи твърди дискове с контролери от типа SCSI. Само основни първични дялове (primary partitions) могат да бъдат зареждащи за операционните системи DOS и Windows. Другите дялове обикновено се използват за съхраняване на данни или за зареждане на други операционни системи. Основният раздел на DOS може да съдържа само един логически диск. Освен това DOS разпознава само един допълнителен раздел (DOS Extended), но в него могат да бъдат разположени няколко логически диска с последователни наименования от латинската азбука: D,E,F,H…Върху диска могат да бъдат формирани и дялове за работа под друга операционна система (Linux, UNIX). Тези дялове са с друг тип организация и въобще не се виждат от DOS или Windows. Някои операционни системи (OS/2, Windows

NT, Linux) използват собствени файлови системи, но могат да работят и с файловата система на DOS.

6.9. Конфигурационни файлове на MS-DOS

Конфигурационните файлове CONFIG.SYS и AUTOEXEC.BAT се създават автоматично при инсталирането на DOS или от потребителя като се редактират с който и да е текстов редактор. При всяка промяна на тези файлове трябва да се рестартира компютърът, за да се приемат промените.

Конфигурационен файл Autoexec.bat

AUTOEXEC.BAT (от англ. Automatic execution — автоматично испълнение и англ. Batch — пакет, група) — системен пакетен файл (файл, съдържащ последователност от команди на езика на командния интерпретатор на MS-DOS COMMAND.COM или неговите клонинги като 4DOS), който се съдържа в главната директория на системното дисково устройство (Boot disk). За пръв път файлът AUTOEXEC.BAT се появява в операционната система MS-DOS. Името му (в превод съкратено от "Автоматично изпълнение") описва неговото предназначение — автоматично изпълнение на команди при начално стартиране на системата.

Конфигурационен файл CONFIG.SYS

Файлът CONFIG.SYS е текстов файл, който съдържа команди, които конфигурират хардуерните компоненти (памет, клавиатура, мишка, принтер и т.н.) на MSDOS. Когато се стартира MSDOS, той приема най-напред командите на CONFIG.SYS файла.

В CONFIG.SYS се задават две групи команди:

•Команди, които изпълняват специални функции;
•Команди DEVICE и DEVICEHIGH за зареждане на външен на DOS софтуер като драйвер на устройства;

Съществуват 16 команди, които се използват в CONFIG.SYS. Най-често използваните команди, които изпълняват специални функции са:

BREAK=ON|OFF	управлява честотата на проверка на сигнала за

	прекъсване Ctrl+Break
DEVICE [d:][path] име на програма	инсталира драйверни устройства и друг външен софтуер във файла CONFIG.SYS
DEVICEHIGH [d:][path] име на програма	

Файлът CONFIG.SYS имеет свой специален синтаксис. Той съдържа директиви вида команда=стойност (или същото, но без знак за равенство – например, numlock off). По-долу е даден списък от най-често използваните команди на CONFIG.SYS:

;	ред с коментари
break	Задава поведението на системата при натискане на комбинация Ctrl + C при изпълнение на програма
buffers	Резервира място за зададен брой дискови буфери
country	Задава регионални настройки (формат дата и време, наименование на валута, ред на сортировка и т.н.)
device	Зарежда драйвер
devicehigh	Зарежда драйвер в UMB – горният блок памет над 640 KB
dos	Параметри на зареждане на DOS (к примеру, пренасяне на части от ядрото в HMA)
fcbs	Задава макс. брой контролни блокове FCB , които могат да се отварят едновременно.
Files	Задава макс. брой файлове, които могат да се отварят едновременно.
Install	Зарежда резидентна програма (обикновена изпълнима програма не в формат на драйвер)
installhigh	Зарежда резидентна програма в UMB
lastdrive	Задава последната буква, достъпна за задаване на дискови устройства
numlock	Задава началното състояние на режима Num Lock
rem	Начална дума за ред с коментари
set	Задава начална стойност за променлива от обкръжението на командния интерпретатор
shell	Задава команден интерпретатор, различен от command.com, и/или опции при стартирането му
stacks	Резервира място за стекове при обработка на апаратни пркъсвания
switches	Допълнителни опции за зареждане

Предвиден е също режим за потвърждаване от оператора на изпълнението на командите. За целта непосредствено след командата може да се постави въпросителна (‘?’).

6.10. Команди на MSDOS

DOSKEY MS-DOS помощна програма, която да съхранява вече въведените команди. Това позволява на потребителя да използва въведените вече команди повторно без те да се въвеждат отново.
Използва се в:
Всички версии на MS-DOS и Windows

COPY Команда на MSDOS за копиране на файлове. Синтаксис:
COPY source destination
където:
source е спецификация на файл-източник
destination е спецификация на файл приемник

ATTRIB Вътрешна команда на MSDOS за променяне атрибутите на файлове и директории. Синтаксис: ATTRIB ±r ±s ±h spec
където:
„+” означава включване, а „-” означава изключване на опция
±r включва или изключва опция “read only” (само за четене)
±s включва или изключва опция “system” (системен)
±h включва или изключва опция “hidden” (скрит)

FDISK Външната команда на MSDOS FDISK дава възможност за създаване и изтриване на дялове на твърдия диск.
Използва се в:
Всички версии на MS-DOS
Windows 95
Windows 98
Windows ME

BACKUP Създава архивно копие (backup) на файлове и директории
Синтаксис:
BACKUP [Source:\Path\Filename] [Target:] [/s] [/m] [/a] [/d:date] [/t:time] [/f:size] [/L:LogDrive:\Path\Log]
Използва се в:
MS-DOS 2.x to MS-DOS 5.00a

ВВМУ курс Администриране на БД лектор ДИТ Ожелезов

Във версии MS-DOS 6.0, 6.2, 6.21, and 6.22 вместо backup се използва командата msbackup

Параметри на командата backup

Source:\Path\Filename	Спецификация на източника (файлове или директории, за които ще се създава резервно копие)
Target:	Спецификация на backup файла
/s	Създава резервно копие не само на файловете от текущата директория, но и от всички поддиректории.
/m	Създава резервно копие на всички файлове, които са били променени след последното архивиране.
/a	Добавя нов backup файл към съществуващите вместо да ги надписва.
/d:date	Архивира файловете, които са били създадени или променени след зададената дата.
/t:time	Архивира файловете, които са били създадени или променени след зададеното време.
/f:size	Създава backup файлове с размер, не по-голям от зададения.
/L:LogDrive:\Path\Log	Създава log файл за архивирането в директория със зададена спецификация.

RESTORE Възстановява файлове от резервно копие. Синтаксис: RESTORE d: [d:][path]filename [/P][/S][/B:mm-dd-yy] [/A:mm-dd-yy] [/E:hh:mm:ss] [/L:hh:mm:ss] [/M][/N][/D]

6.11. Пакетни (.bat) файлове на MSDOS

Пакетният файл (англ. batch file) е текстов файл в MS-DOS, OS/2 или Windows, съдържащ последователност от команди, предназначени за изпълнение от командния интерпретатор. След стартирането на пакетния файл, програмата-интерпретатор (по правило COMMAND.COM или cmd.exe)

чете съдържанието на файла ред по ред и последователно изпълнява командите. Пакетният файл е аналог на скриптовите файлове за командния ред (shell script) в Unix-подобните операционни системи.

Пакетните файлове в DOS имат расширение .BAT; за други операционни системи те могат да имат и други расширения — например, .CMD в Windows NT и OS/2 или .BTM в 4DOS.

История

Пакетните файлове са били включени в ОС MS-DOS още при нейното създаване. Командните интерпретатори на тази система (а в последствие и на Windows) предлагат два режима на работа: интерактивен (когато потребителят въвежда команди в командния ред и ги изпълнява) и пакетен (когато потребителят стартира предварително записана последователност от команди). Концепцията на двата режима е била взаимствана от интерфейсите на командния ред предшестващи ОС (напр. CP/M) и командните интерпретатори на Unix.

Командният интерпретатор в MS-DOS (и по наследство и в семейство Windows 9x) е с наименование COMMAND.COM. Най-известният пакетен файл в тези ОС е файлът AUTOEXEC.BAT, който автоматично се изпълнява от COMMAND.COM при стартирането на операционната система.

Семейството ОС Windows NT (2k, XP и следващи) нямат в основата си MS-DOS и включват интерпретатора cmd.exe, който е частично съвместим с COMMAND.COM. Някои стари възможности на COMMAND.COM не са включени в него, но вместо тях са добавени нови.Все пак COMMAND.COM е включен в NT-подобните системи за обезпечаване на по-добра обратна съвместимост.

Съществуват и другие командни интерпретатори, разработени от други фирми, предоставляющи расширен набор от команди за пакетно программиране — например 4DOS.

Освен това, съществуват компилатори на пакетни файлове (например Bat To Exe Converter), които преобразуват пакетните файлове в самостоятелни испълними програми.

Специфични команди на пакетните файлове

Цикъл FOR

Цикълът FOR се използва за изпълнение на команди над група файлове от зададен списък със спецификации. Командата има следния синтаксис

FOR %%A IN (list) DO command [parameters]

където
list е списък от елементи, разделени с празни интервали.
command Може да бъде коя да е вътрешна или външна команда
parameters – съдържа параметри на командния ред за за MSDOS командата.
При всяко изпълнение на цикъла FOR параметърът %%A се замества с поредната спецификация на файл от списъка list, Да разгледаме един пример:

FOR %%A IN (*.txt *.bat) DO dir %%A

Цикълът FOR ще изпълни командата dir за всеки файл от текущата директория, който съответства на спецификациите, зададени в списъка. В примера това са всички файлове които има разширение .txt или .bat .

! Важно е да се отбележи, че при директно изпълнение (не в пакетен файл) преди името на променливата се поставя само един символ „%”.

Допълнителни възможности на командата FOR за Windows 2000 и следващи версии

С параметър /L в list се задава числова последователност в следният формат:

(start, step, end)

като променливата като %%A получава последователно стойностите от *start* до *end* със стъпка *step*. Пример:

FOR /L %%A IN (0,2,10) DO echo %%A

При изпълнение на командата ще се изведат последователно числата 0,2,4,6,8,10.

Команда за условно изпълнение IF

Командата IF е вътрешна команда, която предоставя възможност за изпълнение на MSDOS команди при изпълнение на зададено условие <condition>.

Когато <condition> е истина , тогава <command> се изпълнява , в противен случай не се изпълнява нищо. Ако след командата е зададена клауза ELSE, тогава, ако условието не се изпълнява, MSDOS изпълнява командата след ELSE.

Командата IF има три различни варианта в зависимост от вида на условието:

a) При проверка на ниво на грешка от последната изпълнена команда

IF [NOT] ERRORLEVEL number command

където:

ERRORLEVEL number — Връща логическа истина (true) ако последната изпълнена програма е върнала изходен код , който е равен или по-голям от зададеният параметър number.

b) При сравняване на символни низове

IF [NOT] string1==string2 command

където:

string1==string2 — Връща логическа истина (true) ако двата низа (string1 и string2) съвпадат.

c) При проверка за съществуване на файл

IF [NOT] EXIST filename command

където:

EXIST filename — Връща логическа истина (true) ако зададеният файл съществува.

За всички варианти параметрите NOT и command имат следното значение:

NOT — (Логическа инверсия) Задава че командата се изпълнява ако условието връща логическа неистина (false).

command — Задава командата, която ще се изпълни ако се изпълнява зададеното логическо условие. След командата може да се зададе клауза ELSE и след нея команда, която да се изпълнява ако условието не се изпълнява. Командата IF и клаузата ELSE (ако е зададена) трябва да са записани на един ред.

!Трябва да се има предвид, че клаузата ELSE не се поддържа от всички версии на MSDOS.

ВВМУ, курс Администриране на БД лектор доц. О. Железов

Да разгледаме няколко примера:
Пример1:

```
IF EXIST C:\autoexec.bat type C:\autoexec.bat
```

Тази команда извежда съдържанието на файла autoexec.bat ако той съществува в главната директория на устройство C.

Пример2:

```
IF "%MyName%"=="John" echo Hello John
```

Тази команда сравнява променливата MyName с низа "John" и ако съдържанието им е еднакво се извежда текста Hello John. Обърнете внимание на това, че името на променливата и символния низ са ограничени с двойни кавички, а преди и след името на променливата е добавен символа %.

http://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D0%BA%D0%B5%D1%82%D0%BD%D1%8B%D0%B9_%D1%84%D0%B0%D0%B9%D0%BB Пакетный файл

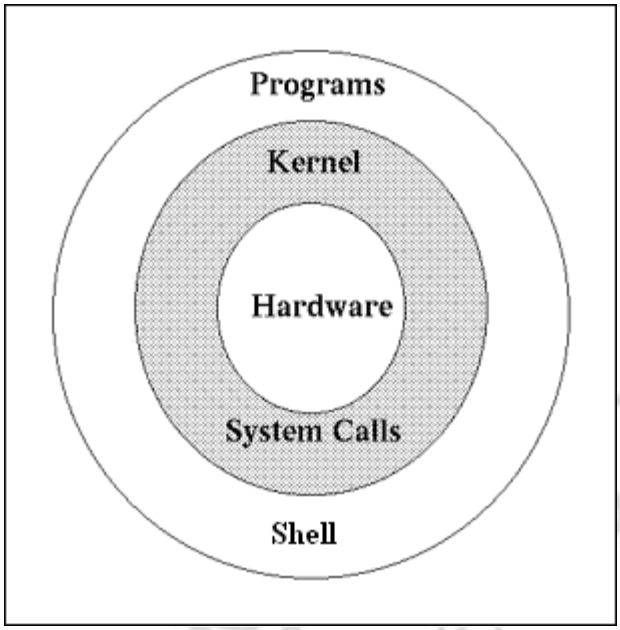
7. Операционна система Unix

ОС Unix е проектирана като многопотребителска система с времеделене. Тя поддържа многозадачен режим и функции за реално време. Официалното признаване на ОС Unix е през 1971г. , когато тя е внедрена в патентния отдел на AT&T. Разделянето на основателите на ОС Unix на две групи дава началото на два основни клона на ОС – SRV и BSD.

7.1. Структура на ОС UNIX

Операционната система UNIX е структурирана на две нива: ядро и обвивка. Ядрото взаимодейства с апаратния слой и предоставя на обвивката системни функции, с които я изолира от непосредствените характеристики на компютърната реализация.

ВВМУ Курс Администриране на мрежови ресурси лектор доц. Ст. Железов



Взаимодействието между ядрото (Kernel) и обвивката (Shell) е унифицирано и се нарича системен интерфейс.

Ядрото осигурява базовото функциониране на UNIX като създава и управлява процесите, разпределя оперативната (ОП) и виртуална (ВП) памет, осигурява достъп до файловата система и периферните устройства.

Подсистеми на ядрото

Файлова подсистема(ФПС): Осигурява унифициран интерфейс за достъп до данните, разположени на различните видове дискови устройства (HD,FD,CD, Ram Dosk).

Подсистема за вход-изход (В/И): Изпълнява заявките от ФПС и контролира достъпа до периферните устройства (ПУ). Осигурява буфериране на данните, взаимодейства с драйверите и осигурява обслужване на заявките, постъпили от тях.

Подсистема “Управление на процесите (ПУП)”: Управлява създаването и унищожаването на процесите, разпределянето на системните ресурси, синхронизацията и взаимодействието между процесите. Като допълнителни задачи: изпълнява алгоритми за избягване на мъртва хватка, определя приоритетите на опашките.

Системни извиквания (System Calls): Всички системни извиквания са стандартизирани в POSIX.1 и POSIX.1b. Те се изпълняват синхронно и управлението се предава на извикващия процес след приключване на заявката.

7.2. Потребителски интерфейс

Потребителските програми се извикват за изпълнение с командния интерпретатор Shell. Unix предоставя възможност за създаване на командни файлове, чрез които могат да се извършват сложни операции чрез командите на командния интерпретатор.

7.3. Управление на процесите в ОС UNIX

При изпълнението на програма ОС създава един или няколко процеса, които се изпълняват паралелно. Така че между програма и процес не може да се постави знак за равенство. Друга съществена разлика между програма и процес е това, че докато изпълнението на програмите се управлява от оператора, то изпълнението на процесите се управлява от ОС.

Видове процеси в ОС UNIX. Състояние на процесите

Най-общо процесите в ОС UNIX могат да се разделят на три вида: системни процеси, фонови процеси (демони) и приложни процеси.

Системни процеси те са част от ядрото и винаги са разположени в ОП. Нямат съответстващи им изпълними файлове, и се стартират при инициализация на ядрото.
Фонови процеси Това са процеси, които се изпълняват без непосредствена връзка с оператора. Като пример могат да се посочат обслужването на принтера, системата за достъп до мрежата и др.
Приложни процеси Всички останали процеси в ОС UNIX са приложни. По правило те са свързани с изпълнение на потребителски заявки или обслужване на потребителя. За най-важен процес от този тип може да се посочи командният интерпретатор (за всеки потребител се стартира по един). Чрез него потребителят се регистрира и изпълнява команди и стартира потребителси програми.

Състояние на процесите в ОС UNIX

В операционната система има много процеси, но централният процесор в един момент изпълнява точно един от тях. В най-общия модел, всеки процес се намира в едно от тези три състояния:
- текущ (running) – процесът в момента се изпълнява от централния процесор;

- готов (ready) – процесът логически може да се изпълнява, но централния процесор е зает с друг процес;
- блокиран (blocked) – процесът чака настъпването на някакво събитие, най-често завършването на входно-изходна операция.
Естествено, във всеки момент точно един процес е текущ.

Описаният модел на състоянията на процесите е прекалено прост – например в него не е ясно как работи ядрото.

В модела на процесите в UNIX, всеки процес е в едно от следните девет състояния:

1) текущ в потребителска фаза (user running)	процесът е текущ и централният процесор изпълнява потребителската програма, свързана с него;
2) текущ в системна фаза (kernel running)	процесът е текущ и централният процесор изпълнява код на ядрото от името на процеса; така по-голямата част от ядрото се изпълнява от процесите в системна фаза;
3) готов в оперативната памет (ready in memory)	процесът е готов за изпълнение и се намира в паметта;
4) блокиран в оперативната памет (blocked in memory)	процесът е блокиран (изчаква събитие) и се намира в паметта;
5) готов на диска (ready, swapped)	процесът е готов за изпълнение, но се намира на диска в специална swapping област, тъй като за него няма място в паметта;
6) блокиран на диска (blocked, swapped)	процесът е блокиран (изчаква събитие) и се намира на диска; естествено, има смисъл на диска да са точно блокираните процеси;
7) преразпределен (preempted)	процесът е бил текущ, но след това е бил насилствено свален от планировчика;
8) новосъздаден (created)	това е преходно състояние при създаване на всеки процес; процесът все още не е напълно работоспособен;
9) зомби (zombie)	процесът е завършил, но все още не е изхвърлен от системата – неговият баща може все още да получи информация за него; това е крайното състояние на всеки процес.

Адаптирано за БД
Лектор доц. О. Железов

Всеки процес се създава чрез системния примитив *fork* и влиза в системата в състояние новосъздаден. След това преминава в състояние готов в паметта или на диска, в зависимост от наличните ресурси. След известно време процесът ще стане текущ, където ще довърши своята част от fork.
След това процесът може да премине в състояние текущ в потребителска фаза и да започне да изпълнява своята потребителска програма. Когато завърши обработката на прекъсването процесът може или да се върне в потребителска фаза или да премине в състояние преразпределен, ако планировчикът реши да го свали.

Атрибути на процеса в ОС UNIX

Атрибутите на процеса позволяват на операционната система да следи и управлява всеки процес. Най-важните от тях са:

1) Идентификатор на процеса (Process Identifier = PID): Това е уникален номер, който се присвоява на процеса при неговото създаване. Номерата на процесите се задават от ОС в нарастващ ред. Чрез PID операционната система получава достъп до точно определен процес.
2) Идентификатор на родителски процес(Parent process ID = PPID): В ОС UNIX в този параметър всеки процес съдържа идентификатор на процеса, който го е породил (това е неговият т.н. „родителски процес“). Ако родителският процес приключи работа и бъде унищожен от ОС, тогава в PPID на процесите, които той е породил (неговите процеси - наследници) се записва PID на родителския процес на унищожения процес. Чрез тази организация в ОС UNIX информацията за дървовидната структура на процесите (родител-наследник) се съдържа в самите процеси.
3) Приоритет на процеса(Nice Number = NN):: Това е относителен (статичен) приоритет, който е присвоен на процеса. Той по правило се различава от динамичния (фактическия) приоритет, който процесът получава при неговото изпълнение.
4) Реален идентификатор (Real ID = RID): Това е идентификаторът на потребителя, който е стартирал процеса.
5) Ефективен идентификатор (Effective ID = EID)

Системен сегмент на процеса в ОС UNIX

Системните данни, които ядрото използва при управление на процесите и системния стек образуват т.н. системен сегмент, влизащ в адресното пространство на процеса. Съдържанието на системния сегмент се поддържа от ядрото на ОС, Системните данни включват информация за:

1) Състоянието на регистрите на процеса.
2) Таблицата на отворените файлове
3) Текущия и главните каталози
4) Състоянието на системните извиквания
5) Текущите данни (състояния на флагове, прекъсвания от таймери, квоти за изпълнение и др.
6) Атрибути на процеса

http://www.bg-informatika.com/2011/07/17_21.html Процеси. Модел на процесите. Състояния и диаграма на преходите.

7.4. Файлова система на UNIX

Файловата система е логическа колекция от файлове на дисково устройство или дял от него. Дялът е контейнер на информация върху дисковото устройство, форматиран за определена файлова система. Общата файлова система дава възможност за логическа организация и поддръжка на съхраняваните данни.
Всичко в ОС Unix се счита за файл, в това число физическите устройства като оптически дискови устройства (DVD-ROM), USB устройства, печатащи устройства и др..Файловата система има един коренен каталог - бележи се с /. В този каталог се монтира определено устройство (обикновено един от дяловете на твърдия диск). Файловата система на това устройство се нарича коренова (на английски: root). По този начин в този каталог се вижда съдържанието на това устройство. В някой каталог от това устройство могат да бъдат монтирани и други устройства и по този начин и тяхното съдържание да стане достъпно. Обикновено работата по монтирането и демонтирането на устройствата се извършва автоматично. Съществуват програми, които позволяват монтиране (mount) и демонтиране (umount) на файлови системи. При повечето UNIX-подобни системи подвижните носители (дискети и CD), флаш-памети и други външни устройства за данни се монтират в каталог /mnt, /mount или /media.

Една примерна UNIX файлова система може да изглежда така:

```
/usr
   /bin
        /arch
        /ls
        /raw
```

```
   /lib
      /libhistory.so.5.2
      /libgpm.so.1
/home
   /lost+found
      /host.sh
   /guest
      /Pictures
         /example.png
      /Video
         /matrix.avi
         /news
            /lost_ship.mpeg
```

Сравнение между файловите системи на MSDOS и UNIX

	UNIX, LINUX	MSDOS
1.	Имена на файлове	До 14с. или 255с., може с няколко разширения. Прави се разлика между малки и главни букви.
		До 8с. първа част и до 3с. разширение, без разлика между малки и главни букви.
2.	Типове файлове	Обикновени, каталози, специални и др., като имената им са вградени в структурата на ФС
		Обикновени, ката-лози, специални, но имената на специалните не са вградени в структурата на ФС.
3.	Структура на ФС	ЙерархичнаЙерархична
4.	Няколко имена на файл	
	Да - твърди и символни връзки	
	Не	

7.4.1. Твърди и символни връзки

Операционна система UNIX поддържа два типа връзки: твърди и символни.

Символна връзка — това е специален файл във файловата система,който съдържа само един текстов ред с указател. Този текстов ред се интерпретира като път към файл, който трябва да бъде отворен при обръщение към дадената символна връзка (файл). В ОС MSDOS няма аналог на символните връзки в UNIX/Linux. В ОС Windows аналог на символните връзки са препратките.
Символната връзка заема точно толкова място във файловата система, колкото е необходимо за запис на нейното съдържание (нормалният файл заема минимум един клъстер).

Администриране на БД Електронен курс доц. Железов

Цел на символната връзка може да бъде всеки обект — например, друга символна връзка, файл, каталог, или даже несъществуващ файл (в тоя случай при опит той да бъде отворен се извежда съобщение за липсващ файл). Символната връзка, която указва несъществащ файл, се нарича висяща.

На практика символните връзки се използват за по удобна организация на файловата структура на компютъра, тъй като позволяват на един файл или каталог да има няколко имена, различни атрибути и свободни от някои ограничения, присъщи на твърдите връзки (те действат само в рамките на един раздел и не могат сочат към каталози). Ако се изтрие оригиналния файл, символичната връзка става висяща (не сочи към никакъв файл).

Символна връзка в UNIX/Linux се създава с командата

ln -s име_на_файл име_на_връзката

Например с командата ln –s prog.1.1 prog се създава символична връзка с име prog към файл prog.1.1

Твърда връзка – Това е запис в каталог, който представя съществуващ файл под друго име. Твърдата връзка е равноправна на оригиналния файл, което означава , че ако се изтрие оригиналния файл, (което на практика означава да се изтрие оригиналния запис в каталога за него), то файла продължава да съществува и да бъде достъпен чрез твърдата връзка към него. За разлика от символичните връзки, твърдите връзки могат да сочат само към файл, но не и към каталог.
Твърда връзка в UNIX/Linux се създава с командата

ln -име_на_файл име_на_връзката

Както се вижда, разликата между тази команда и командата за създаване на символична връзка е само в това, че липсва параметъра “s”. Например с командата ln –prog.1.1 prog.1.1.2 се създава твърда връзка с име prog.1.1.2 към файл prog.1.1

7.4.2. Типове файлове в ОС UNIX

Regular File Обикновен потребителски файл
Directory каталог (списък от имена на файлове

Special Device File Специален файл на устройство
Named Pipe Именуван канал или FIFO
Socket Гнездо за външна връзка

Regular File: Съдържа данни в някакъв формат. За ОС това е само последователност от байтове. Как те ще бъдат интерпретирани се определя от приложната програма.
Directory: Каталозите формират логическото дърво на файловата система. За ОС каталогът е файл, който съдържа имената на файловете и каталозите, съдържащи се в каталога. Каталозите определят мястото на файла в дървото на каталозите (в дескриптора на файла такава информация не се съдържа).
Special Device File: Специалното файлово устройство осигурява достъп до физическо устройство (по-скоро до неговия драйвер). В ОС UNIX СФУ се класифицират според начина на обмен на данни на два вида – символни и блокови.

http://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D1%81%D1%8B%D0%BB%D0%BA%D0%B0 Символьная ссылка
http://www.tutorialspoint.com/unix/unix-file-system.htm Unix - File System Basics

ВВЕДЕНИЕ В КУРС Администриране на UNIX лектор доц. О.Жекезов

8. Операционна система Windows

ОС Windows е създадена от Microsoft като алтернатива на ОС MS-DOS. За разлика от MS-DOS Windows е създадена като многозадачна операционна система

През 1991г. излиза първата версия на многозадачната ОС Windows NT 3.1, която е създадена като надстройка на MSDOS. Windows NT 4.0, като изцяло самостоятелна ОС, се появява през 1995/96.

8.1. Архитектура на Windows NT 4.0

Windows NT 4.0 е изградена на модулен принцип. Това позволява да се добавят и премахват подсистеми без да се нарушава функционирането на ОС. Основният модул е подсистемата WIN32, която е в ядрото на Windows NT. Тя управлява всички входни и изходни устройства. Всички DOS-приложения се управляват от 32 битова виртуална машина VDM (Virtual DOS Machine).

16-битовите DOS приложения подават заявки към подсистемата WIN-16, която няма директен достъп до хардуера, а се обръща към "абстрактен хардуерен слой" HAL (hardware abstract layer) под управление на Win32.

Win32 използва *изпреварваща* многозадачност. Затова комуникацията между ядрото и приложенията е чрез съобщения. Формират се опашки от съобщения и има възможност процес да бъде отстранен ако блокира опашката.

8.2. Файлови системи, поддържани от Windows NT 4.0

Windows NT поддържа три вида файлови системи:

FAT (File Allocation Table) Това е традиционният DOS формат. Този формат не разполага добре големите файлове върху дисковите устройства. Освен това, за разлика от NTFS не се поддържа възстановяване на файлове при дискови грешки (т.н. "толериране" на грешките). FAT обаче е необходим за начално зареждане на Windows NT и по тази причина един от дяловете на HD е с FAT файлова система.

CDFS (CD Rom File System) Тази файлова система се използва за CD дисковете.

NTFS (New Technology File System) NTFS е разширяема файлова система, проектирана да поддържа допълнителни атрибути. Най-полезната

допълнителна характеристика на NTFS е т.н. “толериране” на грешките. Постига се чрез системата за запис на файлове RAID и чрез огледален дял на ФС.

8.3. Основни характеристики на файлова система NTFS

NTFS се възстановява след пропадане на ОС и след дискови грешки. В този случай се използва моделът за обработка на транзакциите и реконструиране на томовете на HD. Всяка транзакция, извършвана по време на повредата се довършва или се отхвърля (така се използват транзакциите и в системите за управление на бази от данни). Критичните данни при модифицирането се помнят в специална област на HD (както е в Linux). При необходимост те се използват за възстановяване на ОС. Възможно е също дублиране на данни на отдалечени КС при разпределените файлови системи.

Поддържа се толеранс към грешки. NTFS не гарантира пълно възстановяване на потребителските файлове. Но ако се използва допълнителна опция “отказоустойчив драйвер” и масиви от HD в SCSI-интерфейс, възстановяването на файловете е 100%.

Поддържат се файлове и HD с голям обем – 2000 TB (тера байта) за HD и 250 000 MB за файл.

Възможност за множествени потоци от данни – Всеки файл има набор от атрибути: (име на файл, собственик, права за достъп, количество данни и др.). Всеки атрибут се съхранява като отделен поток от байтове, т.е. за един файл са възможни няколко потока от информация.

Индексиране – Атрибутите на файловете в дисковия том могат да се търсят чрез индексна таблица, с което се ускорява търсенето (тук отново може да се направи аналогия с базите от данни, в които индексирането също е средство за ускоряване на търсенето).

8.4. Развитие на Windows NT

През 1998г. Microsoft представя Windows NT 5.0. След отстраняване на грешки и неправилни алоритми тя излиза през 2000-ната година като Windows 2000. Продуктът има няколко нива: Professional, Server, Advanced Server и Data Center Server. Структурната им организация, базирана на микроядро и изпълнителна част, използва структурата на Windosw NT. Добавени са обаче и доста нови модули: поддържане на Plug and Play устройства, шини USR и Fire Ware, управление на захранването и др. В областта на мрежовата комуникация системата Active Directory поддържа разпределена ФС – Dfs, която позволява споделяне на файлове с множество потребители от мрежата и поддържане на множество терминални връзки.

Включени са емулатори на различни микропроцесорни системи, но се работи само с процесор Pentium и МП се емулират върху 64-битова шина.

8.5. Структура на Windows 2000

Операционната система Windows 2000 представлява подобрена версия на Windows NT 4.0 с интерфейс на Windows 98. Тя осъществява пълна поддръжка на plug-and-play устройства, на шината USB, стандарта IEEE 1394 (FireWire), IrDA (Infrared Data Association – стандарт за инфрачервено предаване на данни и извеждане на печат, разработен от ассоциацията IrDA), управление на захранването. Освен това, са добавени нови функции, които не са включвани дотогава в други операционни системи на корпорация Microsoft:

1) каталогова служба Active Directory;
2) система за безопасност Kerberos;
3) поддръжка на смарт-карти;
4) инструменти за мониторинг системата;
5) по-добра интеграция на преносими и desk-top компютри;
6) инфраструктура за системно администране;
7) интернационализация (при инициализация на системата и даже за всеки потребител може да се избере език, който ще се използва при работа със системата);
8) Операционната система MS-DOS е заменена с нова 32-разрядна програма, включающа функционалността на MS-DOS с добавени нови функции;
9) добавена е нова функция на файловата система NTFS, при използването на която два потребителя ще могат заедно използват един свързан файл; когато един от потребителите започне да записва в този файл, автоматично се создават копия на файла.

Операционната система Windows 2000, въпреки допълнителните мерки за преносимост на програмите, притежава като цяло по-малка преносимост от Windows NT 4.0. Тя работи само на две платформи: Pentium и Intel IA-642.

Как и предните версии на Windows NT, Windows 2000 се предоставя в няколко версии: Professional, Server, Advanced server и Datacenter Server. Различията между версиите обаче са незначителни, тъй като се използва един и същ изпълним двоичен код. При инициализиране на системата типът

ВВМУ Курс Администриране на ПС Лектор доц. О.Железов

на продукта се записва във вътрешна база от данни (системен списък). При зареждане операционната система проверява съдържанието на списъка и определя версията на програмния продукт. Формално разликата във версиите се управлява в програмите само от две променливи, прочитани от списъка: ProductType и ProductSuite. Изменението на техните стойности обаче се регистрира като нарушение на лицензa.

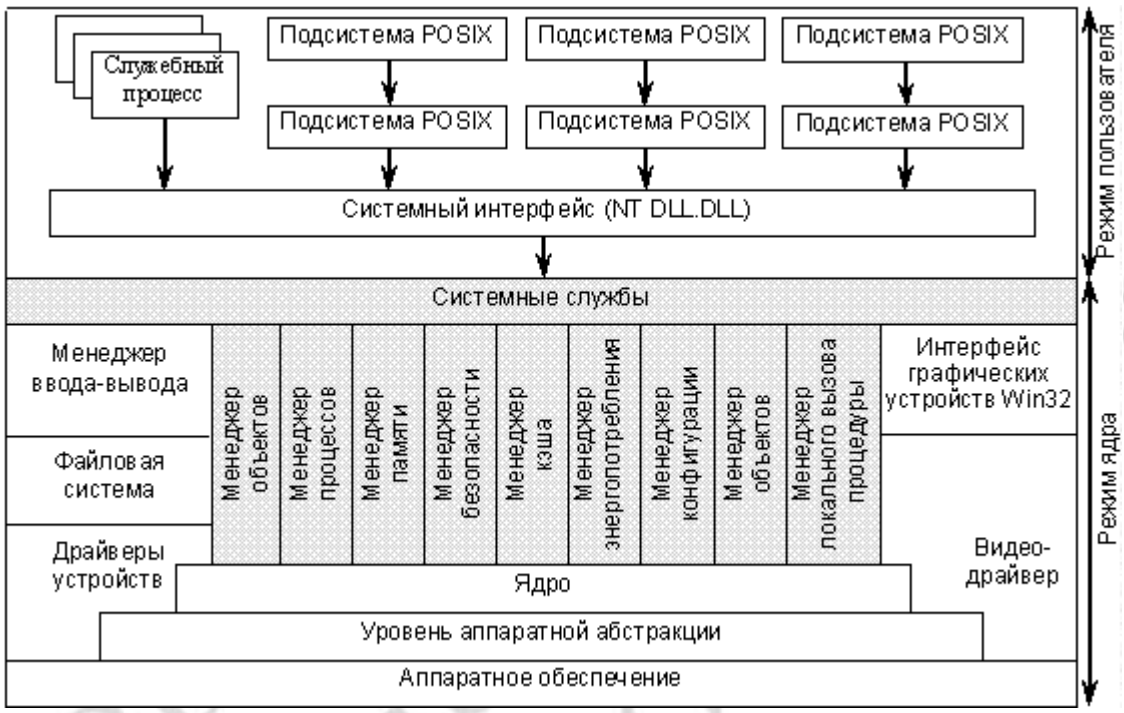
Освен основната операционна система, Microsoft предоставя няколко помощни програми за напреднали потребители: Support Tools, Software Development Kit, Driver Development Kit и Resource Kit. Те предоставят набор от функции за настройка и мониторинг на системата. Включени са в инсталационния диск на Windows 2000 в каталога \support\tools и се инсталират със самостоятелна инсталационна програма setup.exe съдържаща се в същия каталоге.

Операционната система 2000 состоит от две основни части: модул на операционната система, работещ в режима на ядрото, и подсистеми на обкръжението, работещи в режим на потребителя. Подсистемите на обкръжението представляват отделни подпрограми, даващи възможност на потребителя да изпълнява определени функции.

Едно от многото усовершенствания на операционната система Windows NT е в нейната модулна структура – сравнително неголямо ядро, работещо в привилегирован режими, и няколко сървърни процеси, работещи в потребителски режим. В Windows NT първоначално потребителските процеси са взаимодействали със сървърните процеси в съответствие с модела «клиент-сървър»: клиент изпраща на сървъра съобщение, а сървърът извършва определена работа и връща на клиента резултата в сообщение-отговор. Такава модулна структура опростява пренасянето на системата на другие платформи. В резултат ОС Windows NT е била успешно перенесена на платформи с процесори, различни от процесорите на Intel, напр. Alpha на DEC, Power PC на IBM и MIPS на фирма SGI. Освен това, такава структура защищава ядрото от грешки в кода на сървърите. Обаче за увеличаване на производителността, от версии Windows NT 4.0, по-голямата част от операционната система (например, управление системните извиквания и цялата екранна графика) са били върнати в ядрото. Таза схема е запазена и в Windows 2000.

Системата се подразделя на няколко нива, всяко от които използва функциите на по-ниското ниво (рис. 2.18). Изпълняващата подсистема е представена от системни услуги, разделени на модули, всеки от които изпълнява определени функции и има определен интерфейс за взаимодействия с другими модули. Двете най-ниско стоящи нива на

програмното обезпечение – нивото на апаратните абстракций (HAL, Hardware Abstraction Layer) и ядрото – са написани на C и асемблер и са частично машинно-зависими. Горните нива са написани само на езика C и са напълно машинно-независими. Драйверите са написани на език C или C++.



Опростена структура на Windows 2000: – изпълняваща система

8.6. Ниво “Апаратни абстракции” (HAL)

Една от целите, които са си поставили разработчиците на Windows 2000 и Windows NT е преносимостта на операционната система на други компютърни конфигурации. На теория за използване на ОС на друг вид компютър е достатъчно кода на ОС да се прекомпилира с подходящ компилатор. Може да се осигури пълна преносимост на горните нива на операционната система, тъй като те работят основно с вътрешни структури от данни. Ниските нива на операционната система обаче работят с регистрите на устройствата, с прекъсвания, с контролера за непосредствен достъпа до паметта и други апаратни устройства, които могат много да се различават на различните компютърни системи. Въпреки че по-голямата част от кода на ниско ниво е написан на езика C, той не може просто да бъде пренесен от процессор от един тип на процессор от друг тип. Кодът трябва да може да се промени и перекомпилира заради разликите между процесорите.

Фирмата Microsoft е решила да скрие много от апаратните различия на нивото на апаратните абстракций (HAL, Hardware Abstraction Layer). Предназначението на това ниво е да предоставя на останалата система интерфейс към абстрактни апаратни устройства, които не зависят от индивидуалните отличителни особености на апаратното обеспечение. Тези устройства се представят във вид на машинно-независими услуги (процедурни извиквания и макроси), които могат да се използват от останалата част от операционната система и драйверите. Тъй като драйверите и ядрото ползват услугите на HAL и не се обръщат непосредствено към устройствата, се изискват значително по-малко изменения в тях при пренасяне на ОС на друга платформа. Пренасянето на нивото на HAL се улеснява от това, че целия машинно-зависим код е концентрирован на едно място. В нивото HAL са включени услуги, които зависят от набора схеми на дънната платка и се променят за различните компютърни конфигурации в разумни граници.

Нивото HAL не предоставя абстракции или служби за специфични устройств за вход-изход (клавиатура, мишка, дискове), а също блокове за управление на паметта. Тъй като нивото HAL е машинно-зависимо, то трябва да съответства на компютърната система, на която се инсталира, поради това на инсталационния диск Windows 2000 се съдържа набор различни версии. При инсталирането на системата се избира подходяща версия и се копира на системния диск в каталога \winnt\system32\ като файл hal.dll. Въпреки че ефективността на нивото е висока, за мултимедийни приложений корпорация Microsoft предоставя допълнително библиотека DirectX, расширяваща функционалността на нивото HAL с допълнителни процедури и предоставяща на потребителските процеси непосредствен достъп до апаратното осигуряване.

8.7. Ниво “Ядро на Операционната система”

Над нивото на апаратните абстракции се располага нивото на ядрото и драйверите на устройства. Значителна част от ядрото представлява машинно-зависима програма, но по-голямат и част е написана на езике С, с изключение на модулите, в които е било необходимо да се постигне максимално възможната производителност. Една от най-важните функции на ядрото е предоставянето на абстрактен модел апаратната реализация на по-високо ниво. Поради това част от ядрото постоянно се намира в оперативната памет, и чрез задаване на съответния приоритет решава, допустимо ли е прекъсване от устройств входно-изходните устройства или не.

Предназначението на ядрото е да направи останалата част от операционната система апаратно независима и лесно преносима на други

платформи. Ядрото получава достъп до апаратните устройства чрез нивото HAL. То е надстройка на службите от ниско ниво на HAL, като формира от тях абстракции на високи нива. Например, нивото HAL свързва прекъсванията с процедури за тяхната обработка и със задаване на техните приоритети, а ядрото предоставя пълният механизъм за преключване на контекста: съхраняет всички регистри на централния процесор, променя таблиците за страниците, съхранява кеш паметта на централния процесор и т. н. Когато всички действия по съхраняване на контекста се изпълнят, състоянието на изпълнявания до тогава поток се оказва напълно съхранено в таблиците, расположени до тогава в памятта. След това ядрото настройва картата на памятта на новия поток и го зарежда в регистрите на процесора, с което новият поток е готов за работа.

Друга важна функция на ядрото е планирането на потоците. Когато настъпи моментът, в който трябва да се провери готов ли е за изпълнение новият поток, например, след като изтече отделеният на потока квант време или при завершване на процедурата за обработка на прекъсване от входно-изходни устройства, ядрото избира поток и изпълнява преключване на контекста за да стартира неговото изпълнение.

Трета ключева функция на ядрото е предоставянето на поддръжка на два класа обекти – управляващи обекти и обекти за диспетчеризация, които са вътрешни обекти, на основата на които испълняващата система създава потребителски обекти. Управляващие обекти са Обекти за управление на системата, в това число примитивни Обекти за процесите, Обекти за прерываниятa, Обект DPC (Deferred Procedure Call), отложено извикване на процедури; обект APC (Asynchronous Procedure Call), асинхронно извикване процедури.

Обектът за отложено извикване (DPC) се използва, за да се отдели частта от процедурата за обработка на прекъсвания, за която времето е критично, от тази част , за която времето не е критично. Като правило, процедурата за обработка на прекъсвания съхранява няколко апаратни регистри, свързани с входно-изходното устройство, което генерира прекъсването, така че съдържанието на регистрите да може да бъде възстановено и да продължи изпълнението на процедурата, с която се обработва прекъсването. През това време регистрите не трябва да се ползват от други функции на ядрото, следователно тази част от процедурата за обработка на прекъсвания е критичен участък. Изпълнението на останалата част от процедурата може да бъде отложено във времето. Формира се опашка от DPC отложени за доизпълнение процедури, които следва да се изпълнят по-късно.

Обектът за асинхронно извикване (APC) се отличава с това, че асинхронното извикване на процедурите се изпълнява в контекста на определен процес.

Към обектите за диспечеризации се отнасят: семафори, мютекси, събития, таймери и други обекти, чието изменение на състоянието може да се следи и очаква от потоците. Тези обекти са непосредствено свързани с планирането на потоците, и поради това частично се обработват от ядрото.

8.8. Ниво "Изпълняваща система"

Изпълняващата система (наречена супервизор или диспечер) се разполага в най-горната част на операционната система над ядрото и драйверите на устройствата. Тя е написан на език С, не зависит от архитектурата и може (с малки корекции) да се пренася на други машини. Изпълняващата система състоит от 10 компоненти, всеки от които представлява набор от процедури, работещи съвместно за изпълнения някаква задача, в това число управляващи системи за:

1) Обекти управлява всички обекти на ОС, в това число процеси, потоци, файлове, каталози, семафори, устройства за вход-изход, таймери и др
2) Вход-изход
3) Процеси
4) Памет
5) Сигурност
6) Кеширане на дискови блокове
7) plug-and-play устройства
8) Енергопотребление
9) Конфигурация (регистър)
10) Извикване на локални процедури

8.9. Интерфейс за графичните устройства

Изпълняващият модул Win32 GDI (Graphic Device Interface) първоначално се е разполагал в адресното пространство на ползвателя, но във версията на Windows NT 4.0 е бил пренесен в пространството на ядрото за увеличаване на производителността. Win32 GDI управлява графичните изображения за монитора и печатащите устройства. Той съдържа прозоречен менажер и драйвер за дисплея и предоставя системни извиквания, предоставящи на потребителските програми апаратно независим интерфейс за извеждане на данни на монитора и принтерите.

8.10. Ниво на системните служби

Нивото се разполага над изпълняващата система. Предназначението на нивото е да предоставя интерфейс към изпълняващата система – приема системни извиквания на Windows 2000 и извиква за други части от изпълняващата система.

При зареждане на операционната система Windows 2000, тя се зарежда в паметта като набор от файлове. Основната част от операционната система, състояща се от ядро и изпълняваща система, се съдържа във файла ntoskml.exe. Нивото HAL представлява библиотека за общ достъп, расположена в отделен файле hal.dll. Интерфейсът Win32 и интерфейсът на графичните устройства се съхраняват заедно във файла win32k.sys. След зареждане на ядрото на операционната система и изполняющите модули се зареждат драйверите на устройствата, повечето от които имат расширение .sys.

8.11. Драйвери на устройства.

Всеки драйвер может да управлява едно или няколко устройства за вход-изход, да шифрира потока от данни или да предоставя достъп до структури с данни на ядрото. При инсталиране на драйвера в системата той се добавя в регистра и след това се зарежда динамично при всяко зареждане на системата. Съществуват драйвери за физически устройства за вход-изход (дискове, принтери) и за внутрешни устройства и микросхеми, освен това, файловите системи също се представят от драйвери на устройства.

Обекты Windows 2000 представляют собой однородный и непротиворечивый интерфейс ко всем системным ресурсам и структурам данных: процессам, потокам, семафорам и т. д. Доступ к Обектам предоставляется при помощи дескрипторов Обектов и осуществляется через менеджера Обектов. Поэтому все проверки, связанные с защитой, могут быть размещены в одном месте, с гарантией, что ни един процесс не сможет обойти их. Исполняемый Обект представлява собой набор последовательных слов (структуру данных) в памяти (в виртуальном адресном пространстве ядра). Файл на диске не является Обектом, хотя для файла при его открытии создается Обект – структура данных в виртуальном адресном пространстве ядра. При перезагрузке (или сбое) системы Обекты теряются. Когда Операционната система загружается, Обектов нет, кроме бездействующих системных процессов, чьи Обекты жестко прошиты в файле ntoskml.exe. Все остальные

Обекты создаются при загрузке системы, во время работы различных программ инициализации и пользовательских программ.

Каждый Обект содержит заголовок с определенной информацией, общей для Обектов всех типов. Поля заголовка включают: имя Обекта; каталог, в котором Обект «живет» в пространстве других Обектов; информацию защиты (при открытии Обекта выполняется определенная проверка); список процессов, у которых есть открытые дескрипторы к данному Обекту (если установлен определенный флаг отладки).

Каждый заголовок Обекта содержит поле цены квоты, представляющей собой плату, взимаемую с процесса за открытие Обекта. Если файловый Обект стоит един пункт, а процесс принадлежит к заданию, у которого есть 10 пунктов квоты, то суммарно все процессы этого задания могут открыть не более 10 файлов. Таким образом, для Обектов каждого типа могут реализовываться ограничения на ресурсы.

Обекты занимают участки виртуального адресного пространства ядра, поэтому, когда в них нет необходимости, они должны быть удалены, а их адресное пространство возвращено системе. Для этого в заголовке каждого Обекта содержится счетчик ссылок на Обект, который увеличивается на единицу, когда Обект открывается, и уменьшается на единицу при закрытии Обекта. При открытии или освобождении Обекта компонентом исполняющей системы използва второй счетчик. Когда оба счетчика уменьшаются до 0, это означает, что Обект не използва пользователем и ни одним исполняющим процессом, т. е. его можно удалить, а его память освободить.

Часто элементам исполняющей системы бывает нужно динамически получать на время участки памяти. Для этого исполняющая система содержит два пула[6] в адресном пространстве ядра: для Обектов и динамических
структур данных. Един пул является выгружаемым, другой – невыгружаемым (фиксированным в памяти). Обекты, к которым обращения частые, хранятся в невыгружаемом пуле; Обекты, к которым обращения редкие (например, ключи реестра; информация, относящаяся к безопасности) хранятся в выгружаемом пуле. Когда памяти не хватает, этот пул может быть выгружен на диск и загружен обратно по страничному прерыванию. Обекты, которые могут понадобиться при выполнении критического участка программы, когда подкачка не разрешается, должны храниться в невыгружаемом пуле. Когда требуется небольшое количество памяти,

страница может быть получена от любого пула, а затем разбита на мелкие участки размером от 8 байт.

Обекты подразделяются на типы. Тип Обекта определяется указателем на Обект типа (рис. 2.19).

http://edu.dvgups.ru/METDOC/ITS/STRPRO/ASY/METOD/UP/frame/2_4.htm

8.12. Синхронизация на потоците

Средствата за синхронизация са реализирани като диспечерски обекти на ядрото. Обектите за синхронизация на потребителско ниво получават тази опция като включват в себе си диспечерски обект. Поток може да се синхронизира чрез:

Събитие известява за събитие, свързано със синхронизация

Взаимно изключване (mutex) прилага се в критични секции или за В/И устройства, така че само един поток се освобождава.

Семафор прилага се при синхронизиране на потоци от различни процеси. В Windows семафорите са от броячен тип.

Таймер Обекът Таймер сигнализира изтичане на зададено време. Едно типично приложение на таймера: повторение на въвеждането на символ при задържане на бутон от клавиатурата: на всеки 100ms се генерира нов символ.

8.13. Windows XP

Most likely your SP3 installation was not successful. The GetLogicalProcessorInformation was first introduced in Windows XP with Service Pack 3 and was not present on older Windows XP installations.

9. Операционна система Линукс

Линукс (Linux) или GNU/Linux е общото название, което се дава на всички операционни системи, основаващи се на ядрото „Линукс“ и системните инструменти и библиотеки от проекта GNU. Различните такива операционни системи се наричат Линукс дистрибуции, като те се различават

по това с какъв друг софтуер пристигат. Линукс е флагман и един от най-известните представители на свободния софтуер.

Проектът и движение GNU, чиято цел е създаване на нова операционна система свободен софтуер е основан от Ричард Столман през 1984 г. Системата съдържа голям брой инструменти и програми, например компилатори, текстови редактори и сървъри. Софтуерът се разпространява с лиценза GNU GPL, което гарантира бъдещата му свободна достъпност. През 1991 г. към почти завършената операционна система е добавено ядрото Линукс, написано от Линус Торвалдс. С Линукс ГНУ става напълно работеща операционна система и това спомага за бързото ѝ разпространение.

В момента ОС Linux е достигнала параметрите на UNIX. Поради гъвкавото конфигуриране на ОС Linux тя е предпочитана операционна система за Web сървъри.

9.1. Подсистеми на ОС Linux

В ОС Linux са включени следните четири основни компонента: ядро, обвивка, файлова структура и обслужващи програми.

1) Ядрото е базовият набор от компоненти. Разработени са доста версии на ядрото на Linux, като за 2013г. последната стабилна версия е 3.10.6.
2) Обвивката (Shell) е интерфейсът на ядрото с потребителя. В команден режим това е на практика интерфейсът на командния интерпретатор. Shell на Linux включва в набора си от команди както команди от MSDOS, така и команди от UNIX. Най-използваният интерпретатор е BASH (Bourne Again Shell), но Linux предлага и други интерпретатори като Korn Shell, C Shell.
3) Файловата структура много прилича на файловата структура на UNIX, но има и заемки от DOS. В последните версии основната директория root (\) съдържа осем поддиректории: /bin , /dev , /etc , /lib, /sbin , /sys, /proc и /tmp.
4) Обслужващите програми са текстови редактори (vi, emax, ed, ex и др.) , мрежови програми, управляваща програма Midnight Commander, и др.

9.2. Ядро на ОС Linux.

Ядрото на повечето ОС е монолитно и се стартира като единичен процес. Всяка промяна изисква всички модули и процедури отново да се свържат и преинсталират, след което системата се стартира отново за да се проявят промените. ОС Linux, за разлика от други операционни системи, е изградена от относително самостоятелни блокове, наречени презареждаеми

модули, които могат автоматично да се зареждат в ядрото и да се извеждат от него при заявка. Те се оформят като обектни файлове (има опция и за изтегляне от Интернет). Изпълняват се като процес в системен (т.е. защитен) режим.

Въпреки че има "мини ядро", което не може да се извади от състава на ОС, основната част от ядрото са зареждаемите модули, които изпълняват повечето му функции:

Динамично свързване: Могат да се зареждат и извеждат зареждаеми модули от състава на ядрото по време на активни действия на ядрото.

Стекова организация: Модулите са подредени йерархично. Отделните модули могат да играят ролята на библиотека за модули от по-високо ниво. В състава на модулите влизат:

Системни извиквания
Сигнали
Файлова подсистема
Процеси и планиране
Виртуална памет
Управление на основната памет
Драйвери
Прекъсвания

9.3. Файлова подсистема на ОС Linux

Linux 2.6.X поддържа повече от 40 ФПС, които могат да се интегрират в ядрото или да се заредят като модули, както следва: ФПС с общо предназначение(ext2, FAT), списъчни ФПС (ext3, IBM, JFS), мрежови ФПС (NFS, Coda)

ВВМУ, курс Администриране на ОС
лектор доц. О.Железов

Изтриване на директория – команди rd и rmdir

Командите rd и rmdir са вътрешни командо на MSDOS, с които се изтрива зададена празна директория. За изтриване на директория, която не е празна (съдържа файлове и поддиректории) може да се използва командата deltree или, за ОС Windows 2000 или Windows XP, да се изпозва параметър /S на командите rd иrmdir.

Availability

The rd and rmdir commands are internal commands and are available in the below Microsoft operating systems.

All Versions of MS-DOS
Windows 95
Windows 98
Windows ME
Windows NT
Windows 2000
Windows XP
Windows Vista
Windows 7

Syntax

Removes (deletes) a directory.

RMDIR [drive:]path
RD [drive:]path

Windows 2000 and Windows XP Syntax.

RMDIR [/S] [/Q] [drive:]path
RD [/S] [/Q] [drive:]path

/S Removes all directories and files in the specified directory in addition to the directory itself. Used to remove a directory tree.
/Q Quiet mode, do not ask if ok to remove a directory tree with /S.

Examples

rmdir c:\full

ВВМУ , курс Администриране на БД доктор доц. О.Железов

If a directory contains files or folders when attempting to delete the directory you will receive "The directory is not empty." error message. If you want to delete directories that are full, use the deltree command or if you're using Windows 2000 or later use the below example with the /s switch.

rmdir c:\test
Remove the test directory, if empty.

rmdir c:\test /s
Windows 2000, Windows XP and later versions of Windows can use this option with a prompt to permanently delete the test directory and all subdirectories and files. Adding the /q switch would suppress the prompt.

MS DOS For Loop

Syntax
When used in a batch file the syntax is as follows

FOR %%variable IN (set) DO command parameters

When used on the DOS command line the syntax is as follows

FOR %variable IN (set) DO command paramaters

Excerpt from Special Edition Using MS-DOS 6.22 3rd edition -- Jim Cooper.

Microsoft DOS goto command

Извършва преход на зададен етикет

Синтаксис: GOTO label

```
GOTO END
ECHO SKIPPING THIS
:END
ECHO DONE
```

Microsoft DOS ping command

ВВМУ, курс Администриране на БД
лектор доц. О. Железов

Helps in determining TCP/IP Networks IP address as well as determine issues with the network and assists in resolving them. See the ping definition for a full description

ping [-t] [-a] [-n count] [-l size] [-f] [-i TTL] [-v TOS]
[-r count] [-s count] [[-j host-list] | [-k host-list]]
[-w timeout] destination-list

Options:

-t	Pings the specified host until stopped.
To see statistics and continue - type Control-Break;	
To stop - press Ctrl + C.	
-a	Resolve addresses to hostnames.
-n count	Number of echo requests to send.
-l size	Send buffer size.
-f	Set Don't Fragment flag in packet.
-i TTL	Time To Live.
-v TOS	Type Of Service.
-r count	Record route for count hops.
-s count	Timestamp for count hops.
-j host-list	Loose source route along host-list.
-k host-list	Strict source route along host-list.
-w timeout	Timeout in milliseconds to wait for each reply.

Examples

ping –n 5 localhost

http://www.cutepdf.com/products/cutepdf/writer.asp
http://rudocs.exdat.com/docs/index-417837.html?page=4 Операционните системи за работа в реално време

ВВМУ, курс Администриране БД лекции доц. Стелзов