**Конспект**

1. ОС - основни понятия. Поколения ОС.

Определение за ОС: Набор от програми, които управляват всички ресурси на компютъра и осигуряват интерфейс с потребителя, се нарича ОС. Осигуряват достъп до хардуера. Linux: по-свободен достъп до хардуера от Windows. Тенденция на нарастване: по-мощен хардуер - по-големи ОС.

Компютърната система съдържа хардуер, системни програми и потребителски програми. Ресурсите, това е хардуера.

ОС надгражда хардуера и го скрива от потребителите, и разширява ресурсите на хардуера. ОС дава по-удобни функции за работа.

Шина=Интерфейс:  
- интерфейс за данни;  
- адресен интерфейс;  
- управляващ интерфейс (шина);

Контролерът е междинно хардуерно устройство и се управлява от драйвер. Контролерът управлява входа и изхода, процесора и процесите, както и видовете памет. (контролер=канал);

Виртуалната памер се пази на твърд диск и е по-бавна от оперативната памет.

Главни групи от функции на ОС:

І. Грижи за управление на процесора и процесите, ако ОС е многозадачна.

ІІ. За управление на ОП и виртуална памет.

ІІІ. Управление на вход-изход (файловите системи). Windows поддържа дософски файлови системи. Linux поддържа почти всички файлови системи.

ІV. Потребителски интерфейс. Команден интерпретатор, който работи в текстов режим.Днес има и графичен интерфейс. Командния интерпретатор предоставя команди за работа в текстов режим при липса на графичен интерфейс.

Поколения ОС:

І. Първо поколение (1945-1955):

Компютри с вакумни лампи или релета, а ОС помагат да се зареди програма писана на машинен код с 0 и 1, или за стартиране на компютъра. Те се ползват предимно за числени пресмятания. Използват се от военните. Използват се за криптирания, декриптирания и др. Към 1950г. излизат перфокарти като начин за въвеждане на информация. ENIAC е първият компютър излязъл през 1944г.

ІІ. Второ поколение (1955-1965):

Компютри с ел.база транзистори, а ОС са за пакетна обработка. Програмата заедно с входните данни се въвежда наведнъж с перфокарти

и като дойде време за изпълнение, ОС я зарежда и програмата се изпълнява до края от процесора, като ОС дава необходимите ресурси. Друг вид работа е: Мултипрограмния режим - пуска се заданието, стигаме до входно-изходните операции. Ако зададем друго задание,за да се изпълни трябва да свърши изпълнението на първото. Използва се езика Fortran-една зала за машинната инсталация и една за климатичната. Машините са 2 вида: за числени пресмятания и за търговски цели. Появяват се първите компютри, които работят в реално време и ОС FMS (първата ОС).

ІІІ. Трето поколение (1965-1980):

Интегрални схеми с ниска и средна степен на интеграция. Ползва се

мулти-програмирането: изпълнява се задание, което ако спре за вход-изход се измества от друго задание (ОС CTSS, MULTICS). Използват се терминали за работа с много потребители. Появяват се миникомпютри от PDP1 до PDP11, като за PDP7 е направена ОС Unix (от Кен Томпсън) и езика С. През 1987 е направена ОС Minix за академично ползване и на нейна база е направен Linux.

ІV. Четвърто поколение (1980 – до сега):

Използват се големи интегрални схеми. Ползват се микропроцесори и

Микрокомпютри - работи един потребител, еднозадачни, нямат защита. През 1974г. Е направен Intel 8080, който е 8 битов. Аналог на него е Правец с ОС CP/M. През1980г. се прави 16 битов процесор на Intel 8086, а после се правят Пентиум 2,3,4. През 1995г. е направена ОС Windows NT, Windows 95. Проявяват се и компютърни мрежи за персонални компютри и системи за управление на бази от данни. С появата на компютърните мрежи отпада необходимостта от големи машини. Вече има персонални компютри.

1. Видове ОС, основни характеристики на МS Windows и UNIX/LINUX.

Видове ОС:

1. Mainframe (ОС за големи машини): ползват се за сървър и съдържат хиляди дискове,данни и усторйства за вход.
2. Операционни системи за сървъри.
3. Многопроцесорни ОС.
4. ОС за персонални компютри (Потребителски ОС).
5. ОС за реално време. Два вида за реално време: Hard Real Taim, Sort Real Time .
6. Вградени ОС (мобилни устройства).
7. Смарткарт ОС (за дебитни карти и идентификационни карти).

UNIX е проектирана като многопотребителска операционна система с времеделене. Тя поддържа многозадачен режим, а повечето от съвременните й версии включват и функции в реално време. Първата версия на UNIX е за PDP7 и е написана през 1969г. от Кен Томпсън на езика С. Днешните системи на UNIX могат да се разделят на 2 типа:  
- базирани на SVR4  
- базирани на BDS  
SVR4 е една от най-важните ОС, създадени досега. Включва по-важните  
черти на различните версии на UNIX и работи на различни компютри.

Linux е UNIX-подобна ОС, която днес е широко разпространявана. Първоначалната версия е написана от Линус Торвалдс (студент). Успехът на Linux дължи на предоставянето на свободни програмни пакети. Системата е високо модулна и лесно се конфигурира.

Windows е пример за новата тенденция в ОС за микрокомпютри, предназначени да използват значителната мощност на днешните процесори. Те са многозадачни системи. Две са предпоставките за въвеждане на многозадачност в ОС за персонални компютри, работните станции и сървърите. Първата е, че с нарастването на бързодействието на процесора и обема на паметта, и с въвеждането на виртуалната памет, приложенията стават по-сложни и взаимосвързани. В многозадачна среда потребителят може да отвори няколко приложения и да ги остави отворени. Втората предпоставка е нарастването на изчисленията „клиент/сървър“.

Windows работи на различни персонални компютри и работни станции. Първата версия на Windows NT е 3.1, представена през 1993г. Основната част на NT е написана на С, а част от графичната подсистема и потребителският интерфейс – на С++. Асемблерен език е използван само за машиннозависимите части или където се изисква бързодействие.  
 През 1998 г. се представя подобрението NT 5.0, което през 2000 г. излиза под името Windows 2000. През 2001 г. излиза Windows XP с вътрешен номер 5.1.

1. Процеси. Състояния, операции, блок за управление на процес.

Всяка ОС нарича процесите по различен начин. Можем да имаме потоци или нишки. В Windows управляемата единица е поток, който си изпълнява в рамките на един процес. Процесът е програма по време на изпълнение. Ако стартираме една програма два пъти, това са 2 различни процеса.

Определение: Последователен процес е работа, извършвана от последователен процесор при изпълнение на програмата с нейните данни.  
Процесът е активен обект докато програмата е пасивна. Процесът е двойката ”процесор-програма” при изпълнение.   
 Процес, това е програма по време на изпълнението й. Не се изпълнява наведнъж, а му е нужно време.



1) Избран е процес за изпълнение.

2) Изтекъл е квантът отделен на процеса.

3) Чакане на събитие или завършване на вход/изход.

4) Настъпило е събитие или е завършил вход/изход.

Квант от време: от 10 ms -300 ms (зависи от ОП)

По време на съществуването си всеки процес може да преминава през различни **състояния**.

За много ОС са характерни следните три състояния:

- Изпълняван: процесорът е предоставен на процеса.

- Готов: процесът би могъл да се изпълни, ако му се разпредели процесор.

- Блокиран: процесът не може да се изпълни, докато не получи сигнал,съобщение или ресурс.

Тъй като е възможно няколко процеса да са в състояние на готовност или блокирани, поддържат се опашка на готовите процеси и опашка на блокираните процеси. Обикновено, за по-добро управление, има и опашка на всички процеси в системата.

Операции:

1) Създаване на процес: Отделя се място за програмата в ОП или виртуалната памет.

2) Унищожаване на процес: Унищожаването I-во на дъщерния процес не е

проблем, проблем е унищожаването I-во на родителския.

3) Стартиране или избор: Процесът трябва да се подготви. Избира се от ОС по някакъв алгоритъм, кой да е следващият за изпълнение процес и той се стартира.

4) Изтичане на времето за изпълнение: Може да не е квант, може процеса да се махне от изпълнение преди да му е свършило времето по някакви причини.

5) Блокиране и деблокиране на процеса: При деблокиране процеса се вкарва в опашката на готовите процеси. Такава операция е унищожаване на процес. Когато програмата приключи с действието си ще трябва да се освободи ОП и ресурсите, заемани от процесите. Можем да имаме проблеми, когато се унищожават процеси - родителски и дъщерни. Проблемът се решава в Unix и Linux, като се прави с пренасочване.

6) Промяна на приоритет: Може да се променя по време на работа. Дали ще има или не приоритет зависи от ОС. В зависимост от различните алгоритми за изпълнение може да има статичен или динамичен приоритет или да няма приоритет.

Блок за управление на процеса (дeскриптор) съдържа:

1) Трябва да имаме уникално име на процеса: Обикновено се използват цели числа, като всеки следващ стартиран процес получава с 1 по-голям номер от изпълнявания. Ако програмата я стартираме втори път тя ще има друг номер, което е вече различен процес.

2) Трябва да се пази текущото състояние на процеса и да се знае къде е

разположен.

3) Приоритет на процеса.

4) Указатели към паметите на процеса (представляват адреси от ОП): Това са указатели съм кода, данните, стека на програмата.

5) Указарели към отделните ресурси за процеса: Да се поддържа информация за отворени файлове.

6) Област за съхранение на регистри: Използват се, когато на процеса му изтече кванта от време.

7) Да се защитят процесите един от друг.

1. Паралелни процеси. Взаимно изключване на процеси. Критерии за взаимно изключване.

Паралелни процеси:

Определение: Когато изпълнението на 2 и повече процеси се припокрива във времето се говори за паралелно изпълнение на процеси.

Те биват:

* Независими: не работят с общи данни и работата на единия процес не зависи от другия. Както и да работят два независими процеса техния резултат ще бъде един и същ. Те са лесни за изпълнение и ОС няма проблем с тях.
* Взаимодействащи: те могат да ползват общи памети, данни. Може да си изпращат сигнали за синхорнизация. По този начин изпълнението на единия процес зависи от другия и има значение кой е написан първи, и кой втори.

Взаимодействащите биват:

- Конкуриращи се: чакат за общи ресурси.

- Коопериращи се: ползват общи данни за изпълнението на една задача.

Взаимно изключване на процеси:

Въпреки че физическите и логическите ресурси могат да бъдат разделяни, обикновено във всеки момент от времето те са достъпни само на един процес. Такива ресурси се наричат *критични*. Ако няколко процеса искат да използват критичен ресурс, те трябва да съгласуват действията си във времето така, че в даден момент ресурсът да бъде използван само от един процес – останалите процеси трябва да чакат, докато се освободи ресурсът. Това е същността на понятието взаимно изключване на процеси.

Коректното решаване на проблема на взаимното изключване трябва да отговаря на следните критерии:

* Само един процес може да използва ресурса в даден момент.
* Ако няколко процеса едновременно желаят ресурса, той трябва да бъде предоставен на един от тях в крайно време.
* Ако процес получи ресурс, той трябва да го освободи в крайно време.
* Ако един процес не работи с критичен ресурс, той не трябва да пречи на останалите.

1. Взаимно изключване на 2 процеса. Семафори, събития.

Ако един ресурс трябва да бъде достъпен за един процес се нарича критичен. Ако два процеса искат да ползват критичен ресурс трябва двата процеса да се синхронизират. Процесите трябва да съгласуват действията си във времето, така че в даден момент ресурсът да бъде използван само от един процес ­- останалите трябва да чакат, докато се освободи ресурсът. Това е същността на понятието взаимно изключване на процеси. Ресурсите могат да бъдат физически и логически. Конкретното решаване на проблема на взаимното изключване трябва да отговаря на следните критери:

1. Само един процес може да ползва ресурса в даден момент.

2. Ако няколко процеса желаят едновременно да получат ресурс, то той трябва да бъде предоставен на един от тях в крайно време.

3. Ако един процес получи ресурс, ресурса трябва да бъде освободен в крайно време.

4. Процеси неработещи с ресурси, не трябва да пречат на други да ги ползват.

Частта от програмата, която се занимава с критичен ресурс е критична секция. За да се реши задачата на взаимното изключване, трябва да са изпълнени следните предположения:

1. Само един процес се намира в критична секция в даден момент.

2. Процес може да остане в критична секция крайно време.

3. Ако процес иска да влезе в критичната си секция, трябва да му се даде такава възможност в крайно време.

4. В частност, никакъв процес намиращ се извън критичната секция не може да пречи на други процеси да влязат в критичната си

секция.

Взаимно изключване на 2 процеса.

Предлагат се програмни решения:

І вариант: Само едни от процесите да е в критчната си секция, но се нарушава третото условие. Процесите стриктно се редуват при влизане в критичната си секция. Ако един процес блокира или завърши след излизане от критичната секция,то и с другите след него става същото. Тук обаче възниква проблем с твърдата синхронизация.

ІІ вариант: Помни се състоянието на всеки процес, но се губи време докато изпълнява цикъла на очакване.

ІІІ вариант: Всеки процес ще установява флага си, преди да влезе в цикъл, и само индицира, че може да влезе в критична секция, но един процес може да чака безкрайно, за да влезе в критичната секция. Тук възниква друг проблем. Възможно е и двата процеса да вдигнат флаговете си преди да влязат в цикъла. По-късно всеки от тях ще открие, че флагът на другия процес е установен и ще остане безкрайно

дълго в цикъла - *мъртва хватка.*

ІV вариант: За да се избегне мъртвата хватка, трябва да се разбият циклите във вариант три. Въвежда се нулиране на флага, но има друг проблем – безкрайното отлагане.

V вариант: Всеки процес установява флага си и съобщава на другия процес, че може да влезе в критична секция, ако желае.

**Семафори:**

Реализирането на критични секции става с помощта на въвеждането на две

примитивни операции P и V, който оперират над цели променливи, *наречени семафори*. Семафорите са средства за изключване от високо ниво. Различават се двойчни семафори, приемащи стойностти само 0 и 1, и общи семафори, приемащи цели, неотрицателни стойности.

Класическата дефиниция на P и V e:

1. P(s): while s = 0 do skip; s = s-1;

Извиквайки P(s), процеса чака при стойност на семафора s=0, докато друг процес не го освободи чрез V(s). Тогава стойноста на s се намалява и процеса продължава.

2.V(s): s:=s+1

Стойноста на семафора s се увеличава и процесът продължава. Освен това трябва да има инициализация на семафора s чрез операция:

s: =<цяла константа>.

P и V са критични секций спрямо s .

Семафорните променливи лесно могат да се реазлизират чрез команди

TS(wait,s), където wait и s са локална и глобална променливи (инициализирани съответно с true и false):

P(s): while wait do TS(wait,s);

V(s): s:=false;

Недостатъци са активното очакване, при което напразно се изразходват ресурси и инверсия на приоритетите на процесите, която може да доведе до безкрайно отлагане.

Решение: Когато процес изпълнява операция P(s) и открие, че s=0, той се блокира вместо да чака, след което се избира друг процес за изпълнение. Блокиращия процес може да бъде стартиран отново чрез изпълнение на V(s) от друг процес, при което блокираният процес става готов.

**Събития:**

Между процесите е необходимо не само да се изключват при работа с общи ресурси, но и да си разменят информация, за да синхронизират работата си във времето.Пример за такъв вид взаимодействие е, когато процесите се намират в отношение производител/потребител, т.е единият процес произвежда ресурс, а другият го изполва. Например, процесът-производител изаботва информация, която се записва в буфер. Паралелно с това процесът – потребител разпечатва съдържанието му. Тези действия се извършват асинхронно поради разликата в скоростите на двата процеса. Затова има опастност производителят да се опита да запише в пълен буфер или потребителят да чете от празен буфер. Само взаимното изключване на процеси при работа с буфера не е достатъчно, те трябва да си обменят информация, за да синхронизират действията си. Основно изискване при синхронизацията на процесите е: на единият процес да

се дава възможност да чака, докато друг процес не му съобщи за настъпването на определено събитие.

Първият процес чака, изпълнявайки операцията wait, а втория процес сигнализира за настъпване на събитие чрез операция signal. *Събитията се представят с блок за управление - дескриптор.* С помоща на семафори лесно се решава проблемът производител/потребител. Операция P-вместо wait и операция V вместо signal. Семафор може да получава цели неотрицателни стойности, тогава той е общ(броячен) семафор. Може да се направи още едно обобщение - да се разреши на семафора да получава отрицателни стойности, които ще показват броя на блокираните пред семафора процеси.

Обобщеният семафор може да се прилага за организация и отчитане на

ресурси. Процес произвежда единица ресурс и изпълнява V над свързания с ресурса семафор, а консумира ресурс чрез P. При инициализацията семафорът получава стойност, равна на броя на достъпните ресурси. Читатели и писатели: Данни могат да се използват от няколко процеса. Някой от процесите искат само да четат общите данни, докато

други искат да обновяват данните, т.е. могат да се различават два типа процеси, наречени Читатели и писатели. Няма проблеми двама читатели да четат обшите данни, докато други искат да обновяват данните т.е. могат да се различават два типа процеси, наречени Читатели и писатели. Няма проблем двама читатели да четат общите данни едновремено. Обаче, ако писател и друг процес (читател и писател) искат достъп до данните, може да настъпи хаос. Затова се изисква писателите да имат монополен достъп до ресурса.   
Първи вариант: Никакъв читател не трябва да чака други читатели да свършат, само защото писател чака за достъп до данните.

Втори вариант: Ако писател е чакащ за достъп никакъв нов читател не може да стартира.

1. Мъртва хватка. Условие за възникване на мъртва хватка.

В мултипрограмната среда може да настъпи безизходна ситуация, при която два или повече процеси чакат за условия, които никога няма да настъпят. Подобна ситуация се нарича **взаимна блокировка, мъртва хватка** (deadlock) и др.  
 Разпределението на ресурсите е една от главните функции на ОС. Когато ресурсите се разпределят между много потребители, всеки от които има монополно право за управление на отделния му ресурс, възможно е да настъпи мъртва хватка. Може да се различават постоянни и временни ресурси. Постоянни ресурси са устройствата, а временни – съобщенията.

Условия за възникване на мъртва хватка:

Кофман формулира необходимите условия, които трябва да се изпълнят  
едновременно, за да настъпи мъртва хватка:

* Взаимно изключване – Ресурсът може да се използва само за 1 процес в даден момент.
* Очакване на ресурси – Процесите държат вече отделените им ресурси, като в същото време очакват да получат допълнителни ресурси.
* Непреразпределение – Ресурси не могат да бъдат отнети. Те се освобождават само от процеса, който ги притежава, след като той изпълни задачата си.
* Кръгово (Циклично) очакване – Съществува кръгова верига от процеси, всеки от които чака за ресурс, държащ се от предшестващия го.

1. Стратегии за борба с мъртва хватка.

Методи – 1.Стратегия на предпазване 2.Стратегия на избягване 3.Стратегия на разпознаване

1.Стратегия на предпазване:

Хавандер доказва, че ако бъде нарушено поне 1 от правилата за възникване на мъртва хватка, то тя не може да възникне.Има 3 стратегии, които той предлага и които нарушават правило 2, 3 и 4.

За да се наруши ВТОРО условие: Всеки процес трябва да заяви всички необходими ресурси наведнъж, като при това не може да се изпълнява докато те не му бъдат предоставени. Алтернативно, процесът може да заявява ресурси само тогава, когато няма никакви ресурси.

НЕДОСТАТЪЦИ : 1. Снижава се ефективността на използване на ресурсите, тъй като много от тях могат да бъдат разпределени, но неизползвани дълго време.

2. Възможно е безкрайно отлагане.

ТРЕТО условие – Процес, притежаващ някакви ресурси и получил отказ на заявката за разпределение на допълнителни ресурси, е длъжен да освободи своите ресурси и при нобходимост да ги заяви заедно с допълнителните.

НЕДОСТАТЪЦИ : Възможно е с освобождаването на ресурси да се загуби извършената работа до момента. Може да се появи безкрайно отлагане.

МОДИФИЦИРАН ВАРИАНТ : Като процес заяви доп. Ресурси и те не са налице се проверява дали не се държат от друг процес, чакащ за ресурси. Ако да, то те се отнемат от чакащия процес и се дават на заявяващия процес.

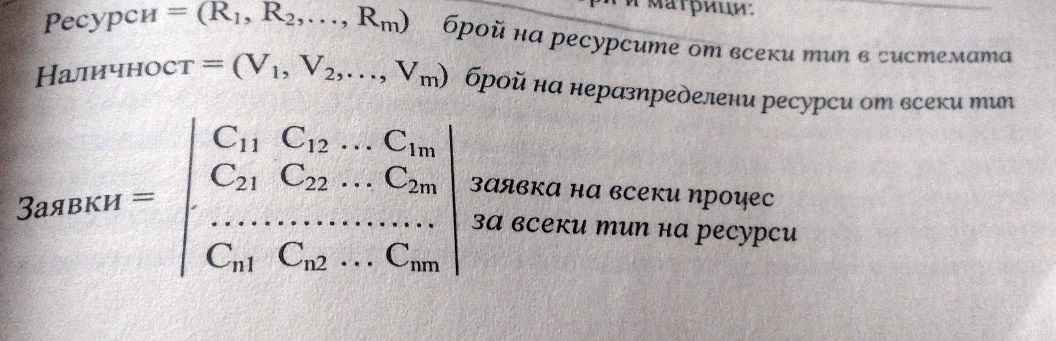
ЧЕТВЪРТО условие – Линейно се подреждат всички типове ресурси, като им се преписват уникални номера. Процесите са длъжни да заявяват ресурси в реда на нарастване на номерата (йерархия) . НЕДОСТАТЪЦИ : Загуба на ефективност, ако на процесите са необходими ресурси в друг ред.

2.Стратегии на избягване :

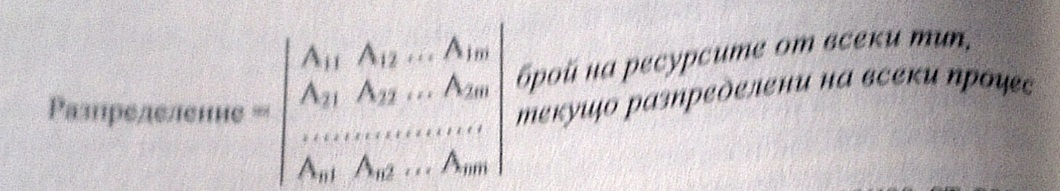
Достатъчно е да се разпределят ресурсите, придържайки се към определени правила

2.1.Отхвърляне на старта на процес

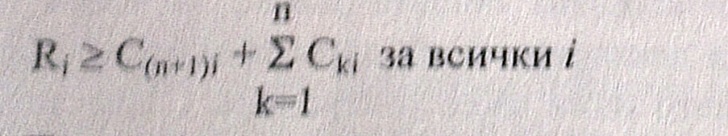
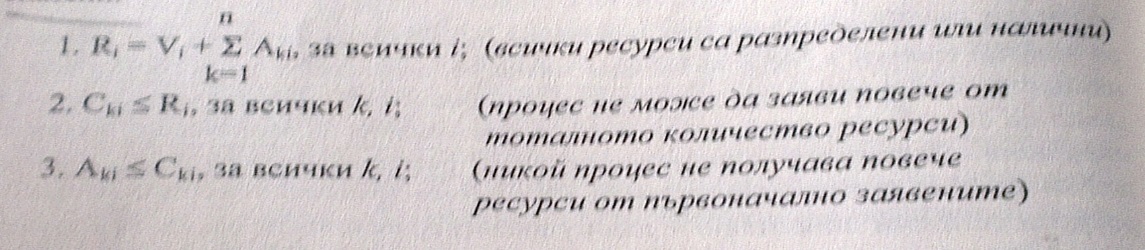
При този подход процес не се стартира ако изискванията му водят до мъртва хватка. Нека система има n процеса и m типове на ресурси. Дефинирани са :



Матрицата Заявки дава максималните нужди на вс. процес от от вс. тип на ресурс . Ред отговаря на процес (Тази инф. Трябва да е декларирана като процес предварително) .Разпределение дава текущото разпр. на ресурси от всеки тип за вс. процес. Да е спазено:



От тук може да се дефинира политиката за избягване на мъртва хватка – нов процес не се стартира ако изискванията му за ресурси могат да доведат до мъртва хватка. Нов процес се стартира само ако Процес може да се стартира само ако максималните претенции на вс. Процеси (+ новия) могат да бъдат удовлетворени.



2.2 Отхвърляне на заявка за разпределение на ресурс

Състояние на системата е текущото разпределение на ресурсите м/у процесите.Може да се каже че системата е в безопасно състояние ако съяествува поне 1 път за изпълнение на процесите ,който позволява да се обходи опасното състояние.Достатъчно е да се провери дали отделянето на ресурс не води до опасно състояние. За всяка заявка (ако е удовлетворена) трябва да се определи съществува ли сред общите заявки на вс. Процеси някаква последователност от заявки, която да доведе до опасно състояние. Ако да заявката се отклонява. (МИЛИ ДРУГАРЧЕТА ПРЕКРАСНИ СЛЕДВА ПРИМЕР ЗА НЯКАКЪВ СИ БАНКЕР КОЙТО Е 3 СТРАНИЦИ НЯМА ДА ГО ПИША АКО НЯКОЙ ГО ИСКА НА СТРАНИЦА 135-6 Е )

2.3.Друг алгоритъм за разпределение на ресурси

Когато се разпределят ресурси, където всеки клас ресурси включва само един елемент,може да се състави по-ефективен алгоритъм,който използва вариант на графа за разпределение на ресурсите,като допълнително се въвежда претендиращо ядро. То индицира (а у :ДД) че процес може да заяви ресурс някога в бъдещето и повтаря посоката на заявяващото но се отбелязва с прекъсната линия.Когато процес заяви ресурс, той може да бъде разпределен само ако конверсията на заявяващото в присвояващо ребро не води до възникване на цикъл в графа.АКо цикъл бъде открит системата ще изпадне в опасно състояние.

3.Откриване на мъртви хватки (Стратегия на разпознаване)

- Установяване на факта и определяне на процесите и ресурсите въвлечени в мъртва хватка. Важи за първите 3 условия. След това се определя дали има кръгово очаване. Загубите се състоят в задържане на процесите и понижаване на възможностите на системата.

3.1. Редукция на графите

Преобразуване на графа за разпределение на ресурсите във вида,който би имал ако процес завърши работата си и върне ресурсите на системата .Ако графът може да се редуцира за всички процеси значи мъртва хватка няма. В противен случай нередуцируемите процеси образуват групата процеси, въвлечени в мъртвата хватка.

3.2.Единични ресурси

Използва се вариант на графа за разпределение на ресурси наречен wait-for .Той се получава от началния граф чрез отстраняванена възлите на ресурсите и прекарване на съответните ребра.За да открие мъртва хватка системата трябва да поддържа такъв граф и периодично да стартира алгоритъма за търсене на цикъл онзачаващ че в системата има мъртва хватка

3.3.Обобщен алгоритъм

Използват се векторите Наличност и ресурси и матрицата Разпределение.Допълнително е дефинирана n x m матрица Заявки като всеки нейн елемент qij представлява кол. Ресурси от типа j заявени от процес I .Алгоритъма маркира процесите,които не са в мъртва хватка. Стъпки на алгоритъма:

3.3.1. за i=1,2…n маркира всеки процес, чийто ред в Разпределение има само нули с Край[i]:=true в противан случай процесът е немаркиран и Край[i]:=false

3.3.2. Инициализира вектор В със съдърж. Наличност

3.3.3. Намира индекс i,такъв че процес i е немаркиран и i-ият ред на Разпределение е п-малък или = на В. Ако няма такъв ред се преминава към 5

3.3.4. Ако има такъв ред i се маркира с Край[i]:=true и съответнипт ред на Разпределение се добавя към В.Преминава към 3

3.3.5.Ако някои стойности на i (1<=i<=n) с Край[i]:=false системата е в мъртва хватка.Всеки немаркиран процес е блокиран.

Мъртва хватка има ако само след като се заварши процеса има немаркирани процеси.

Смесени стратегии

* Ресурсите се групират в отделни класове
* Те (кл.) се подреждат линейно
* Във всеки клас се използва алгоритъм най-подходящ за класа

Като пример за илюстриране се разгл. Системата с ресурсите :

* Пространство за размяна – предварително разпределение
* Ресурси на процес – избягване може да се наложи
* Основна памет – предпазване чрез изместване (на процес)
* Вътрешни ресурси – Предп. Чрез подреждане на ресурсите

1. Структура на ОС. Основни групи от функции. Ядро. Примитиви.

1 и 2 . Работата на ОС се характеризира така – Нови процеси,които променят състоянието си и се унищожават.Нови и преразпределени ресурси.

Функциите на ОС са няколко.

ОС. – основната задача е *управлението на процесите*. Системата за управление на ресурсите се грижи за другите ресурси. *Трета група е свързана* с управлението на входа/изхода и файловете от данни

Обръщенията към функциите могат да бъдат явни(пр. чрез програмно прекъсване)

3.Ядро.

Елементите на ОС са транзитни (временно се преместват в паметта където могат да се пазят) и резидентни (в оперативната памет,нарича се ядро)

Ядрото се нарича базовия функционален набора,въз основа на който се строи останалата част на ОС и се осигурява работа на вс. Програми . Функциите и структурата на ядрото зависят от конкретната ОС .Традиционен подход ереализация на функциите чрез примитиви.

4.Примитиви.

Примитивите се извикват от процесите под форма на извикване на подпрограма.Изискващата и изисканата програми се изпълняват строго последователно. Изпълнение на примитивите е като изп. На сложни команди в рамките на процеса.Не изискват блок на управление и запис в опашката на готовите процеси.

Използването на спец. Команди SVC е по-разпространен начин за извикване на примитиви.

1. Структурна организация на ОС. Реализация и генерация на системата.

Структурна организация на ОС:

І. Монолитни ОС - цялата ОС е написана като една програма. Трудно писане на програмата. Трудно може да се направи някаква защита.

ІІ. С йерархичен подход - функциите на ОС са разделени на нива и много лесно може да се направи защита. Основните принципи за подобна организация на ОС са формулирани при описанието на системата THE. На нулевото ниво имаме ядро - управление на процесите и синхронизация.

Ниво 1: имаме управление на ОП и виртуална памет от страничен супервайзор.   
Ниво 2 - връзки с оператор: диалог с оператора и управление на конзолата.   
Ниво 3 - управление на устройствата.   
Ниво 4 - управление на заданията: приложни процеси и интерпретатор на езика за управление. С представянето на ОС във вид на йерархична структура са свързанидве идеи. Първата е, че на програмите, отнасящи се към по-високите нива, трябва да бъде предоставена възможност да ползват услугите на програмите, отнасящи се само към по-ниските нива.

Другата е,че при прехода към по-ниските нива, съответстващите им функции трябва да се изпълняват все по-бързо.

ІІІ. Виртуални машини:  
 Тя дублира реалната, тоест едновременна работа на ОС на една реална машина.

ІV. Машини със системно ядро:

Минимален набор от основни процедури. Само ядрото работи в супервайзорски режим, а всички останали части се обръщат към неговите

функции. Тези фунции се наричат интерфейс на системните изисквания. Структура: след инициализация паметта съдържа ядрото и вътрешен процес, който е основната ОС. При заявка от терминалита вътрешния процес може да създава процеси, които също могат да образуват процеси. Структурата на известната система Unix също се базира на принципа на ядрото.

V.Модел клиент-сървър:

ОС е разделена на части. Всяка изпълнява някаква нейна функция. В този модел ядрата асигурява само комуникацията между процесите. Обикновено голяма част от функциите на ОС се реализират като потребителските процеси. Когато му е необходимо обслужване от ОС, потребителски процес наречен процес-клиент изпраща заявка към процес-сървър на ОС, който изпълнява заявката и връща отговор.

**Реализация на ОС**

Трудностите,свързани с използването на асемблерните езици за целите на системното програмиране карат конструкторите на ОС да проявят инетерес към езиците на по-високо ниво .(Вероятно първата ОС на не асемблерен език е МСР) . Предимствата от използването на такива езици от високо ниво е същото като при приложните програми – по=компактно и бързо програмиране,по-лесни за разбиране и настройка програми. Недостатък – скоростта и обема на паметта.Увеличаването на скоростта идва от добре подбраните данни и алгоритми.

**Генерация на ОС**

Предполага се,че ОС ще работят на машини, които могат да се разполагат на различни места с различна конфигурация на устройствата. Системата трябва да бъде конфигурирана или генерирана за всяко специфично място . Процесът е известен като **генерация на системата .**За генериране е необходима специална програма.(тя изисква от оператора – какъв CPU е използван,обем памет,устройства на разположение,какви опции на ОС се желаят). Генерация на системата се свежда до създаване на таблици за описание на системата.

1. Защита на ОС. Домейни. Матрици за достъп.

Процесите в ОС трябва да бъдат защитени един от друг при тяхната работа с различните ресурси – апаратни (процесор, памет, устройства) и логически (програми, таблици, файлове). Всеки ресурс трябва да се изролзва само от процесите, получили съответните права от ОС. За това съществуват разнообразни механизми – напр. управлението на достъпа до файловете в UNIX позволява на потребителя да определя кой и как да има достъп до неговите файлове. Апаратурата и програмните средства за адресиране осигуряват изпълнение на процеса само в неговото адресно пространство – напр. при обръщане към сегмент или страница се проверява в таблицата на сегмените (страниците) за правилност на обръщане от страна на процеса. Входно/изходната система не разрешава на потребителя директно да управлява устройствата и така тя се предпазва от неефективност и възможност за достъп на един от потребителите към данни на друг. Използва се таймер за прекъсване на изпълнението на текущия процес.

Ролята на защитата в компютърната система е да даде механизъм за налагане на политиките, управляващи използването на ресурсите. Тези политики могат да бъдат установени по различни пътища – някои са фиксирани при проектирането на системата, а други са формулирани от мениджмънта на системата. Достъпът до средствата за защита от страна на приложния програмист предоставя възможност да бъдат запазени срещу неправилно използване ресурсите, създавани от приложната подсистема.

Защитата може да повиши надеждността чрез откриване на скрити грешки в интерфейсите между компонентите на подсистемите. Незащитен ресурс не може да противостои на използването (или неправилното използване) от неоторизиран или некомпетентен потребител.

**Домени на защита**

Елементите на компютърната система, които трябва да се защитават, се наричат обекти. Обектите могат да бъдат както апаратни (като процесор, области на памет, устройства), така и програмни (като програми, файлове, справочници, семафори). Елемент, който желае да получи достъп до обекти, се нарича субект. Типичен субект е процесът или процедурата. Възможните операции зависят от обекта. Например процесорът може да бъде използван за изпълнение на програми, над файловете се извършват операциите четене, запис, отваряне, затваряне и др. Субект и обект могат да си разменят ролите – напр. два процеса могат да установят защита един от друг и тогава те действат или като обект, или като субект.

Очевидно е, че трябва механизъм, който да разрешава достъпа на всеки субект само до обекти, за които е упълномощен да има достъп.

За да могат да се обсъдят различните защитни механизми, въведено е понятието домен (област на защита). Всеки домен дефинира множество от обекти итипове операции, които могат да се изпълнят над всеки обект. Възможността да се изпълни операция над обект е право на достъп. Така доменът може да се представи като съвкупност от подредени двойки (име на обект, множество от права).

Доменът определя контекста на защитата, в който оперира дадена субект. Вместо да се свързват правата за достъп със субектите (процесите), сега те се свързват с домените. Всеки процес оперира в контекста на защитата, определен от някакъв домен (който специфицира обектите, до които има достъп процеса). Други процеси могат да се ползват със същите права, ако се изпълняват в същия домен. По време на изпълнението си процесът може да преминава от един домен в друг, променяйки статуса на защитата си – правилата за привключване зависят в голяма степен от системата. Разделянето на статуса на защита на домени позволява обръщане с права на достъп, независимо от субектите в системата и осигурява гъвкаво средство за съвместно използване на механизмите на защитата.

Не е необходимо домените да бъдат отделни – те могат да споделят права за достъп.

**Матрици за достъп**

Статусът на защитата на дадена система може да бъде описан с прост модел. Във всеки момент има множество от домени *D={d1, d2,…, dn}* и множество от обекти *O={o1, o2,…,om}.* Статусът на защитата се описва с изображението *F:D×O →R*, където множеството *R={R1, R2,…,Rk}* представялва множество от атрибути – права на достъп (като четене, четене и запис, изпълнение).

Ако изображението *F* зависи само от *d* и *o,* статусът може абстрактно да се изобрази като матрица, наречена матрица на достъп. Редовете в матрицата представляват домените, а колоните – обектите.

1. Файлови системи.

Файлът е съвкупност от данни, обединени с общо име и предназначение.

Файловата система се грижи да намери определен файл и да го даде като цяло. Занимава се с правата за достъп до файловете и ефективното използване на заеманите от тях ресурси. Файловете могат да съхраняват първични програми, обектни програми, текст, цифрови данни и др.

Типове файлове:

* По съдържание: програмни файлове, файлове - данни, текстови.
* По форма представяне: символни, двоични, смесени.
* По метод на достъп: последователни, директни, индексни и др.

В описанието на файла (в неговия дескриптор) се съхранява типът на файла, като съвкупност от няколко атрибута. Определена последователност от данни (байтове) свързано с някакво име,

атрибути.

Когато се създава файл, системата трябва да му разпредели пространство във външната памет, да оформи блок за управление и да запише дескриптора в справочник. Потребителят се обръща към конкретен файл с името му. Имената на файловете могат да бъдат записани в папки и поддиректории. В повечето случаи поддиректориите се организират като файлове. Всяка поддиректория си има име и съдържание. Структората е дървовидна. Всеки файл си има дескриптор на файла – информация за собственост, дата, час и др. При завършване на работата с файла, трябва да се извикат процедурите за затваряне на файла. Тъй като на разположение е ограничено дисково пространство, необходимо е да се използва и освободеното от унищожените файлове. Затова системата трябва да води списък на свободното пространство. При създаването на нов файл се търси в списъка и се отделя пространство за файла. Това пространство се изважда от списъка. Когато файлът се унищожи, неговото пространство се добавя в списъка.

Разпределение на свободното пространство:

Основни методи за разпространение на дисковото пространство:

1. *Непрекъснато разпределение* – При този метод всеки файл заема една непрекъсната област от дисковото адресно пространство. В директориите трябва да се пази само началния адрес и дължината на файла.

Това разпределение разрешава последователен и директен достъп.

При последователен достъп се помни адресът на последния прочетен блок и веднага може да бъде прочетен следващият.

При директен достъп до файла лесно се изчислява адресът на блок, след

като се знае къде е началото на файла.

1. *Разпределение чрез свързване в списък* - Файловете не заемат

свързани области на вторичната памет. Всеки файл може да се организира като свързан списък на блокове от данни, които могат да бъдат разпръснати произволно в дисковата памет. В директорията се записва указател към I блок. Всеки блок съдържа, освен данните, указател към следващия блок. Може да се използват 1 или 2 указателя за ускороние на търсенето. Краят на файла се отбелязва с Nil или в директорията може да се запише указател към последния блок или да се окаже броя на блоковете. Не е необходимо да се декларира дължина на файла при създаването. Лесно е и унищожаването - освободените блокове просто се връщат в списъка на свободното пространство.

1. *Разпределение с индексен блок* – Този метод за разпределение

поддържа директния достъп до файлове. Проблемите, възникващи при използването на свързан списък, са решени чрез събиране на всички указатели в последователна таблица, наречена индексен блок. Всеки файл има собствен индексен блок. Директорията съдържа адреса на индексния блок. Достъп до i-тия блок на файла става като се използва указател в i-тия елемент на индексния блок. При създаването на файл всеки указател е Nil. Когато блок от данни се записва за пръв път, отделя се блок от списъка на свободните блокове и адреси и се записва в съответния елемент на индексния блок.

Карта на файловете – Вкл. и изкл. на блокове може да се улесни, като се състави единна карта на всички файлове, в които вместо указатели се използват номера на блокове. Съответният елемент на картата съдържа номера на следващия блок от файла, а в директорията се намества номера на I блок на файла. Картата може да заема един или повече блокове. В елементите на картата съответстващи на последния блок, се записва Nil.

Файлови системи: FAT 16/32, NTFS, ex fat и други.  
 Файловите разширения идват от DOS, за улеснение на потребителите.

Файловете винаги се записват като цяло число в клъстери.

***(клъстер = блок на файла)***

Средство за сигурност е дублирането на FAT таблицата.

Файлови операции: създаване, изтриване, отваряне, затваряне, четене, писане и други.

1. Процеси и потоци.

Определения за процес във въпрос 3.

**Потоци**

Ако трябва едновременно да се обслужват няколко заявки,тогава някои нови системи имат възможност да поддържат няколко потока (нишки) в един процес. Потоците се нар. Олекотени процеси, а процесите с много потоци-задачи. Потоците работят в рамките на един процес,ползват ресурсите и паетта на процеса, в който участват. Ако имаме процес стигнал до вход/изход той се блокира до изпълнението.Потока няма собствена памет и ресурс.Той ползва някакъв дескриптор.Заемат някаква част от паметта на процеса.Когато работи процес може да се превключват потоци,единият спира, а другият се пуска.

**Начини за упр. на потоци:**

I-ви начин-Ние си управляваме потоците в рамките на процеса.ОС не ги вижда и не отделя квант време.

II-ри начин- Потоците да се упр. От ОС (т.е. да се виждат от диспечера, който упр. Процесите) На всеки поток се задава квант време.и потоците се превключват от ОС.

Единицата за ресурси е процес (задача), а единицата за планиране се означава с поток. Всеки процес има свои: брояч на командите, регистри и стек, собствено адресно пространство. В много отношения потоците приличат на процесите. Потоците, подобно на процесите, се планират и делят процесора, но усилията за превключване на контекста на два потока са много по-малки в сравнение с превключване на контекста на два процеса. Потоците в един процес не са така независими, както отделните процеси. Т еделят едно и също адресно пространство (имат и едни и същи глобални променливи), множество отворени файлове, таймери, сигнали и т.н.

Състояния на процеса

По време на съществуването си всеки процес може да преминава през различни състояния. За много ОС са характерни следните три състояния:

* Изпълняван – процесорът е предоставен на процеса.
* Готов – процесорът би могъл да се изпълни, ако му се разпредели процесор.
* Блокиран – процесът не може да се изпълни, докато не получи сигнал, съобщение или ресурс.

Тъй като е възможно няколко процеса да са в състояние на готовност или блокирани, поддържат се опашка на готовите процеси и опашка на блокираните процеси. Обикновено, за по-добро управление, има и опашка на всички процеси в системата.

1. Планиране на процесора.

Процесорът е най-важният ресурс на системата, подлежащ на разпределение. В определени моменти трябва да се реши дали процесът да продължи да изпълнява текущо развиващият се процес или да избере друг процес за изпълнение. Планирането се извършва на високо, ниско и междинно ниво.

*Планиране на високо ниво* се извършва с паметния режим на работа и е свързано предимно с пакетния режим на работа. То се изпълнява от системен механизъм, които избира задание от множеството постъпили в системата задания, създава процес за неговото изпълнение и го записва в опашката за обслужване от процесора.

Програмата за системен вход чете пакета задания от системното устройство и формира вътрешна системна опашка от задания. Програмата за планиране на високо ниво преглежда опашката и избира задание за изпълнение, като отчита текущото състояние и свойствата на наличните процеси в системата и съпоставя характеристиките на заданията в опашката. Планиращата програма управлява броя на процесите в паметта. Когато то е стабилно средните скорости на пристигане и завършване на заданията са еднакви.

*Планиране на ниско ниво*. Средствата на това ниво определят кой от готовите процеси да бъде избран за изпълнение и за колко време процесът се разполага в процесора, след което предоставят процесора на избрания процес. За разлика от планирането на високо ниво, което се изпълнява еднократно от гледна точка на всяко задание, планирането на ниско ниво се изпълнява постоянно - един процес може да получи процесора многократно. Превключването на процесора между заданията изисква съхраняване на състоянието на стария процес и зареждане на

състоянието на новия. В някой ОС има часовник или интервални таймери, за да се генерира прекъсване в определен момент, след което се избира нов процес за изпълнение.

*Структура на планиращата програма*. На най-ниското ниво се разполага кратка програма – диспечер, която само взема избран процес от опашката и му предоставя процесора. На следващото ниво е програма за краткосрочно

планиране. На най-високо ниво е програмата за дългосрочно планиране, която прави сложни изменения в състоянията на процесите.

*Дисциплини за планиране*: Планиране без изместване, когато не може да се

отнеме процесор от изпълнявания процес в противен случай дисциплината е с изместване. Между тези две крайности лежи избирателно изместване. Например с всеки процес може да се свърже двойка битове, които определят:

1) Дали процеса може да измести друг процес.

2) Дали процеса може да бъде изместен от друг процес.

*В много ОС, планирането се извършва на три нива.*

*Междинно ниво на планиране -*  организира се междинна опашка, в която заданията се поставят от планирана на високо ниво програма, а диалоговите попадат в нея при заявка от терминалите. Планираната на междино ниво програма обработва тази опашка и прехвърля процеси в опашката на готовите, той се обработва от диспечера. Функциите на междинното планиране могат да бъдат извиквани през фиксирани интервали от време или възникване на определени ситуации.

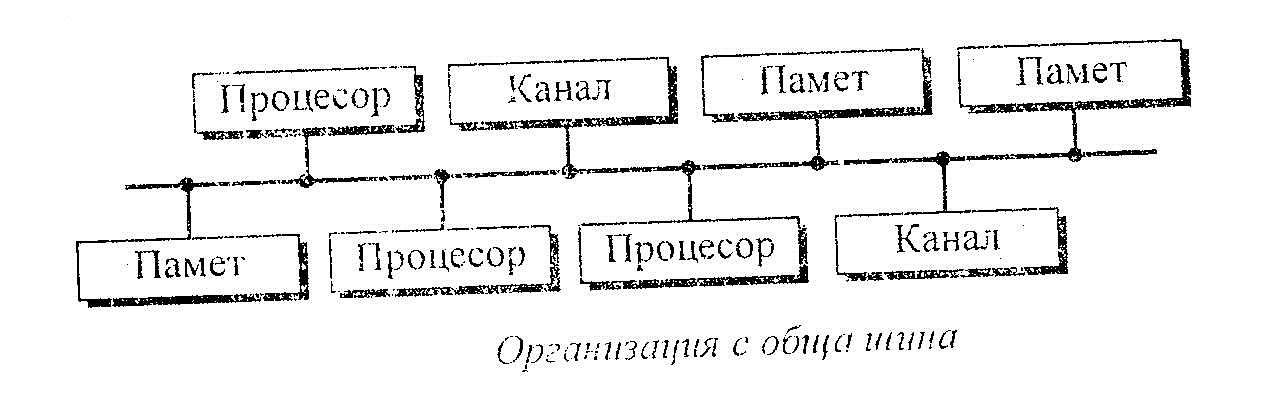
1. Многопроцесорни ОС.

В мултипроцесорните системи процесорите споделят обща основна памет и такт, образувайки единен изчислителен комплекс. В тях комуникацията се основава главно на общата памет. Данните в основаната памет са достъпни за всички процесори, а едни и същи програми могат да с еизпълняват от няколко процесора.

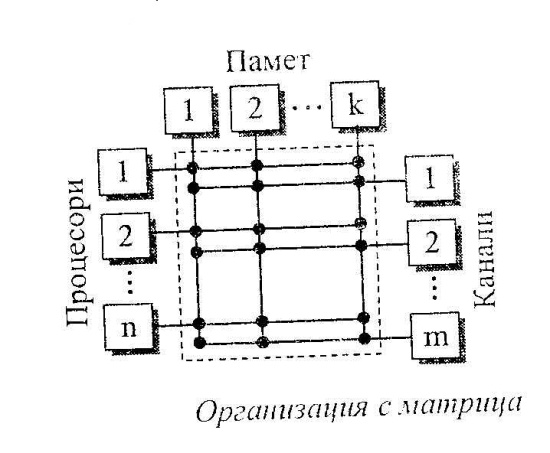
Главните предимства на мултепроцесорните системи са повишаване на производителността и надеждността. Съществуват много конфигурации на мултипроцесорни системи. В „чист вид” тези системи включват повече от един процесори с приблизително еднаква производителност, които споделят обща памет и периферни устройства, и се управляват от една ОС.

**Организация на апаратурата**

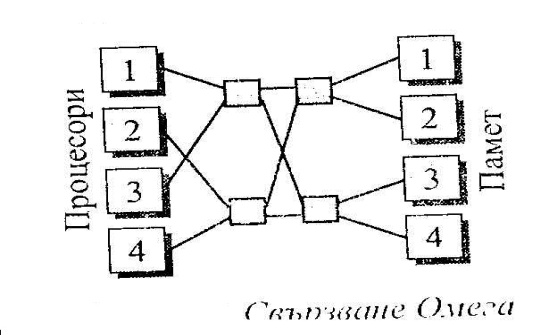
Най-простият метод е да се свържат всички функционални модули към обща шина, по която те обменят данни. Шината включва линия за адреси, данни и управление. Тя е пасивно устройство и обменът на данни се извършва под управлението на самите модули. Шината е общ ресурс и трябва да бъде предвиден механизъм за определяне на реда на нейното използване. Тази организация е проста и гъвкава – към нея лесно се включват нови модули. Главните и недостатъци са: цялата система става неработоспособна при повреда на шината, общата скорост на предаване на данни се ограничава ат пропускателната способност на шината, състезанията за достъп до шината могат да доведат до деградация на производителността.



В големите системи шината става тясно място и затова трябва да се използват много шини. Разпространен подход е да се използва матрица за координатна комутация, която осигурава физическа връзка между произволен модул на паметта и произволен процесор. Възможно е да се прави едновременно обръщение към всички модули на паметта. Тази организация позволява максимална гъвкавост на обмена на даннии минимизира състезанията за достъп до ресурси, което осигурява по-висока пропускателна способност и производителност в сравнение с другите методи.



За да се опрости комутаторът, се използват и други методи за комутация – мрежа за превключване, наречена омега. В примера тя включва само 2х2 превключвателя, всеки с два входа и изхода. Всеки превключвател може да свърже произволен вход и произволен изход – така всеки процесор има достъп до всеки модул на паметта.



При използването на многопортова памет всеки процесор има собствена вхадна линия до всяка памет. Броят на пресечните точки практически е като на матрицата с координатна комутация, но превключването е разпределено между модулите на паметта. Обикновено портовете на паметта получават постоянни приоритети за разрешаване на конфликтни ситуации. Независимо от начина на реализация, броят на съединенията е по-висок в сравнение с другите организации, но могат да се постигнат високи скорости на обмен на данни.

