БУРГАСКИ СВОБОДЕН УНИВЕРСИТЕТ
ЦЕНТЪР ПО ИНФОРМАТИКА И ТЕХНИЧЕСКИ НАУКИ

ENGLISH FOR SPECIAL PURPOSES

Настоящият свитък предоставя информация относно извършената работа по проекта за обезпечаване на учебния процес по Английски език посредством създаване и реализиране на електронен курс - English for Special Purposes (ESP) в областта на техническите науки.

Отпечатването на този материал се финансира по научно-изследователски проект (НИП) на Бургаският свободен университет (БСУ) - № 3 за 2009 година.

Ръководител на проекта:
проф. д-р Димитър Юдов

Преподавателски колектив:
ас.Димитър Минчев, преп.Димитър Бинев

Участващи студенти:
Петя Димитрова, ИКН, 07311012
Мая Бошева, ИКН, 07311003
Галина Иванова, ИКН, 07311001
Георги Темелков, ИКН,07311014
Едуард Дионисиев, КСТ, 07311055
Илиан Найденов, КСТ, 06311183

Съдържание

Цели

Проекта ESP постави следните цели за изпълнение:

1. Да се повиши ефективността и качеството на обучението по специализиран английски език за студентите от ЦИТН посредством създаване на електронен речник от специализирани термини, преведени от английски на български език,и обратно, и краткото им обяснение и на двата езика.

2. Да се приложат интерактивни методи в обучението по английски език. Работата с компютър и запознаването с електронната платформа осигуряват по-високо ниво на усвояване на знания по английски език, по-висока мотивация за работа на студентите и моментален достъп до електронните материали съгласно мотото на Европейската програма Leonardo da Vinci – Long-Life Learning, „Веднага, тук и сега.";

3. Да се приложат алтернативни методи на обучение и оценяване на знания като: придобиване на умения за учене при съвместна работа в платформа за ЕО; прилагане на нови форми на изпити, например: електронен тест за проверка на усвоения материал и др.;

4. Да се осигури възможност за индивидуализация на обучението;

5. Да се проследи повишаване на нивото на знания на всеки един от обучаемите. Платформата за ЕО позволява наблюдаване работата на всеки един от студентите, например чрез преглеждане на лога за извършените действия и операции от съответния студент;

6. Да се използват възможностите на платформата за електронно обучение – Moodle

Ползи

Реализирането на електронен курс по специализиран технически Английски език - "English for Special Purposes" за нуждите на процеса на обучение в центъра по информатика и технически науки на Бургаският свободен университет ще даде положителни резултати както за обучаемите така и за обучаващите:

- **Студентите —** ще могат да получават полезна и коректна информация относно специализирани технически термини директно от сайта с учебни материали на университета: http://students.bfu.bg. Ще бъдат развити уменията им за работа в екип, изпълняване на задания във определен времеви интервал и проекта ще доведе до надграждане на знанията им в конкретна сфера на техническите науки.

- **Преподавателите —** ще могат да следят екипната работа на участниците по проекта, да наблюдават и контролират изпълнението на поставените задачи, както и да участват паралелно със студентите в разработката на електронния курс. Те също ще могат да се възползват от материалите подготвени в резултат от изпълнението на научно-изследователския проект в различни сфери на техническите науки.

Организация на курса

Електронният курс ESP е организиран в следните тематични направления:

- Компютърни Архитектури
- Компютърни Мрежи
- Програмиране
- Операционни системи
- Микропроцесорна техника

За всяко тематично направление са въведени следните дейности:

- Задания
- Речници
- Уики
- Форуми
- Чатове

Участващите студенти в научноизследователският проект са разпределени в групи по тематичните направления.

В края на своята работа по проекта, студенските екипи имат задачата да изготвят и изпратят текстов документ на Microsoft Word XP/2003, съдържащ материалите, които са качили в курса. Това включва: Уики и Речник.

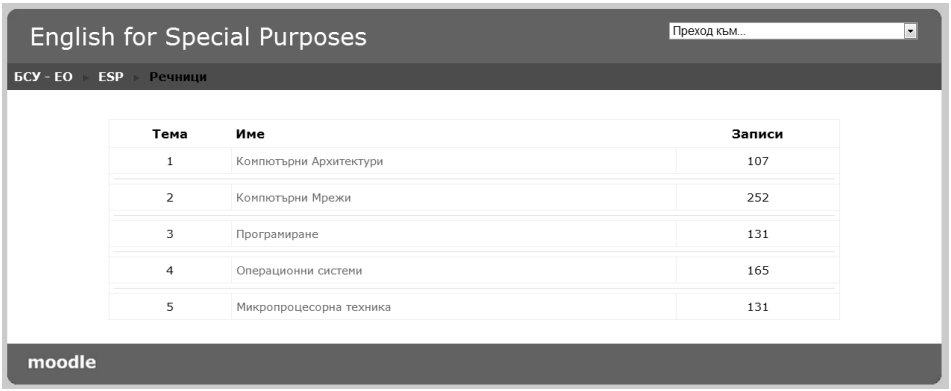
Преподавателския екип поема задачата да изготви и отпечата свитък с материали по проекта ESP на хартиен носител, който всеки от участниците ще получи.

Речници

Речниците в електронният курс ESP са организирани в следните темати чни направления:

Тема	Име	Записи
1	Компютърни Архитектури	107
2	Компютърни Мрежи	252
3	Програмиране	131
4	Операционни системи	165
5	Микропроцесорна техника	131

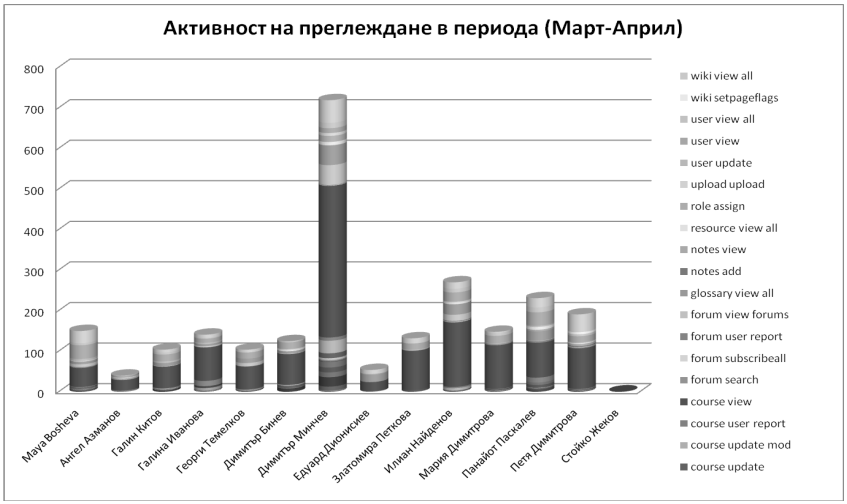
Таблица. Речници в курса English for Special Purposes



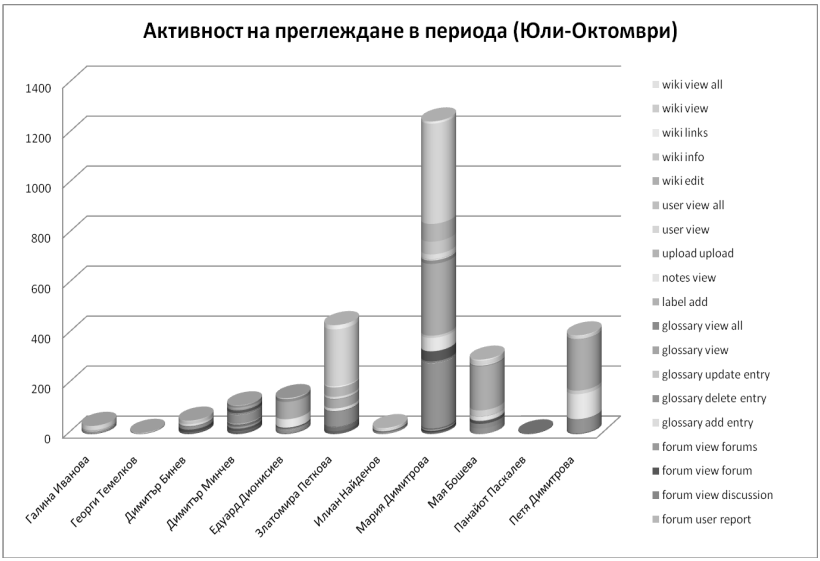
Фигура. Речници в курса English for Special Purposes

Забележка: Посочените брой думи в речниците е възможно да са повече след отпечатването на този материал, тъй като те непрекъснато биват допълвани с нови термини.

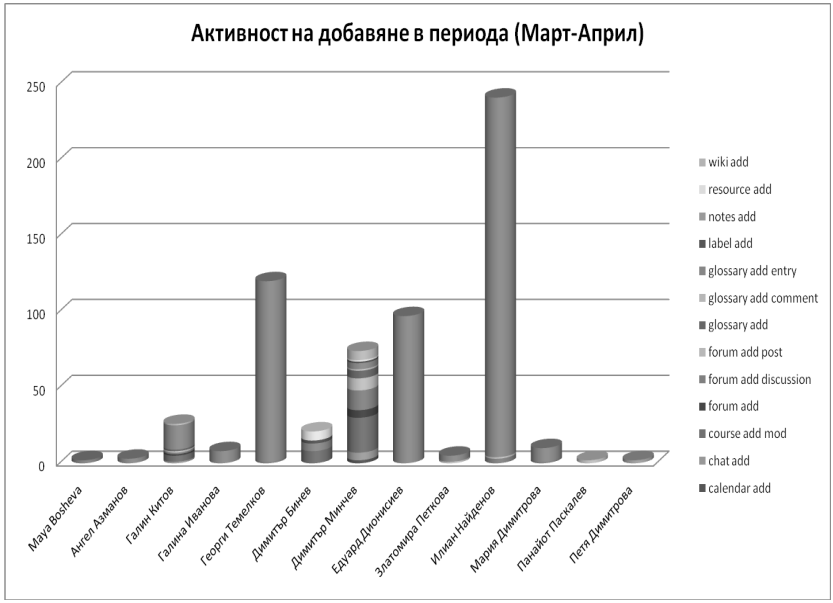
Активност на участниците



Графика: Активност на преглеждане в периода (Март-Април)



Графика. Активност на преглеждане в периода (Юли-Октомври)



Графика: Активност на добавяне в периода (Март-Април)

Графика: Активност на добавяне в периода (Юли-Октомври)

ЕЛЕКТРОННО ОБУЧЕНИЕ И КОЛАБОРАЦИЯ ПО СПЕЦИАЛИЗИРАН АНГЛИЙСКИ ЕЗИК РЕАЛИЗИРАНИ В РАМКИТЕ НА НАУЧНО-ИЗСЛЕДОВАТЕЛСКИ ПРОЕКТ "ENGLISH FOR SPECIAL PURPOSES" ПРОВЕДЕН В БУРГАСКИ СВОБОДЕН УНИВЕРСИТЕТ

ас. Димитър Минчев, преп. Димитър Бинев**
**Бургаски Свободен Университет, +359 56 900 477, +359 56 900 541, mitko@bfu.bg, binev@bfu.bg*

Резюме: В статията се показват резултатите от проект за електронно обучение "English for Special Purposes" изпълнен в Бургаския Свободен Университет през 2009 г. В нея се описват начина по който той бе проведен и се посочват предимствата на комбинирането на електронно обучение с преподаване насочено към самия студент. Получените резултати са добри и този начин на обучение може да се прилага в различни тематични области.
Ключови думи: електронно обучение, Мудъл, учене през целия живот, съвместно обучение, обучение насочено към самия студент.

E-learning and Collaboration in ESP Realized in the Scientific Project "English for Special Purposes" Implemented in Burgas Free University

Ass.Prof. Dimitar Minchev, Ass. Prof. Dimitar Binev**
**Burgas Free University, +359 56 900 477, +359 56 900 541, mitko@bfu.bg, binev@bfu.bg*

Abstract. This article is about the results obtained from the e-learning project "English for Special Purposes" implemented in Burgas Free University in 2009. It presents the way the project was done and stresses the advantages of the combination of students-centered teaching and e-learning. The outcome of the implementation was positive and could be applied in different fields of education.
Keywords: e-learning, Moodle, life-long learning, collaborative learning, students-centered teaching

Въведение
Съществуват емпирични доказателства, които доказват, че студентите които имат свободата да изследват области на базата на личните си интереси, и които са подкрепяни в техните усилия от преподавател (играещ ролята на модератор) не само постигат отлични академични резултати, но и показват социално развитие и личностно израстване. Въпреки това, обучението насочено към самия студент е по-взискателно по отношение на комуникация, организация, както и по предоставяне на учебни материали. Така, основната идея е да се комбинират този тип обучение и електронното обучение с цел да се използват предимствата на двата подхода. Ние наричаме този комбиниран стил: електронно обучение насочено към самия студент. Проучванията показват, че то има потенциал за намаляване на повишените изисквания на обучението насочено към студента в дългосрочен план, но като изцяло запазва всички негови предимства като: дълбочинен учебен процес, личностно израстване, социални умения и по-висока степен на гъвкавост, а и се запазва същността на идеята за учене през целия живот.

Реализация на електронното обучение
За успешната работа по проекта бе необходима подходяща платформа за електронно обучение. Интегрираната в Бургаския свободен университет платформа за учебни материали [8], базирана на Moodle [9], позволи провеждането на мероприятията по научния проект „English for Special Purposes". Платформата Moodle е продукт реализиран на езика PHP или така наречения хипертекст предпроцесор и се задвижва от сървър за база данни MySQL [11]. Moodle, PHP [10] и MySQL са продукти с отворен код, достъпни за свободно изтегляне от интернет и условията на техния лиценз GPL позволяват тяхната употреба за изследователски и научни цели без да се заплаща за софтуера или да се нарушават авторските права от употребата на нелегален софтуер. В рамките на платформата за електроно обучение бе създаден курс „English for Special Purposes", където са зададени права на преподавателите и студентите участващи в научния проект.

Проект „English for Special Purposes"
В БСУ е наша отговорност да подготвим студентите не само за настоящия, но и за бъдещия пазар на труда който ще се характеризира с още по-голяма мобилност на работната сила, още по-голяма интернационализация на националните предприятия и с неограничена комуникация на английски език. Вземайки на в предвид тези предизвикателства, ние сме изправени пред необходимостта да преоценим и променим обучението по специализиран английски език. Ето защо създадохме и реализирахме проект "English for Special Purposes". Основната цел на този проект е

създаване на електронен речник и уикипедия за нуждите на студентите от Центъра по информатика и технически науки в БСУ. Тук имаме пълната свобода и възможност да реализираме едно електронно обучение насочено към студентите.

На първата сбирка по проекта бе представена в детайли концепцията за обучение насочено към самия студент и студентите бяха помолени да попълнят въпросник. Той съдържаше въпроси свързани с техните интереси в различни области, очакванията им от проекта, настоящата им работа (ако имат такава), както и определяне на степента до която студентите са доволни от конвенционалните формати на обучение спрямо желанието им за участие в едно начинание каквото е обучението насочено към самия студент. Този въпросник ни послужи като ориентация и ни бе от помощ през цялото времетраене на проекта. Един интересен резултат бе тенденцията студентите да бъдат значително по-заинтересовани от изпробването на новия подход в сравнение със старите начини на преподаване. Това се подкрепя и от факта, че само 2 студенти не заявиха веднага своята готовност да участват в този начин на обучение. След това, на базата на интересите на самите студенти, се оформиха групи от по 2-3 души, които сами избраха в коя област да работят. Те обикновено се ориентираха към области (предмети), които изучават в момента, но някои си избраха такива, които мислят, че ще са от полза за бъдещото им професионално израстване. Бяха оформени групи, които ще работят по следните тематични направления: Компютърни мрежи [1], Компютърни архитектури [2], Програмиране [7], Операционни системи [3], Микропроцесорна техника [4] и Бази от данни [5]. Някои от студентите поставиха интересни идеи за размисъл и като цяло дискусията относно избора на тематични направления за работа обогати самия проект с много разнообразни и смислени теми, и по този начин се засили доверието в обучението насочено към самия студент. Преподавателите участващи в проекта предоставиха материали на студентите по които те да работят. Самите студенти трябваше, на база на предоставените им материали и натрупаните знания до момента, да започнат да въвеждат в електронната система Moodle термини, съкращения и обяснения от английски на български, и обратно, в областта в която си бяха избрали да работят. Тази база от данни ще е от неоценима помощ за настоящите и бъдещите студенти в Центъра по информатика и технически науки, тъй като всички ние знаем, че болшинството от информацията отнасяща се до новите и модерни технологии е на английски език.

На всяка следваща сбирка бе представяна и коментирана работата на студентите до момента. Те представяха същността на тематичната област в която са избрали да работят и обясняваха как биха могли да използват новите си знания в бъдещата си работа, както и говореха за положителните

страни на работата по този проект. Това имаше силно обогатяващ ефект за всички участници, тъй като те обсъждаха свободно и с ентусиазъм своите нови знания и виждания, казваха къде могат да се намерят добри уеб ресурси и разширяваха обхвата на знания на другите участници. В действителност, получаването на знания от самите колеги бе автентично и истинско, и даде възможност на всеки участник да се изяви като по този начин да се чувства важен в групата. Тук се наблюдаваше едно припокриване на информацията предоставена от отделните студенти, което засили факта за наличието на междупредметни връзки и всичко това правеше материала да изглежда по-малко теоретичен.
Ползата от работата по този начин е, че се използва по максимален начин наличието в Интернет на огромно количество материали по които да се работи. Той е особено подходящ за тип работа и обучение насочени към самия студент. Тук студентите могат да търсят и намират решение на задача, която сами са си поставили, или са избрали да работят, да решат проблем след консултация с преподавателя и др. Той дава възможност на студентите да си обменят информация по асинхронен начин без значение на тяхното местоположение [6].

Студентите показват склонност да са по открити, конструктивни, да сътрудничат помежду си и да са по-отговорни пред себе си, както и пред другите си колеги, и пред преподавателя. Тук тези студенти, които са по-тихи и затворени обикновенно проявяват голяма активност при онлайн дискусиите вътре в групата и между отделните групи. Всички са много дейни тъй като те изпълняват различни роли, като например на: автори на документ, кооординатори, на хора задаващи или отговарящи на въпроси, на оценители и др.

На последната сбирка по проекта всяка група представи обема от работа, който е извършила. Студентите също така споделят как са разпределили задачите и отговорностите вътре в самата група, демонстрират резултатите от своята работа както и си дават мнението и препоръките за бъдещо продължение на работата след приключване на проекта.

Резултати от проучването

• Студентите чувстват, че са научили определено повече в сравнение с конвенционалните курсове със същата продължителност;
• Студентите знаят кои аспекти на своята работа биха могли да подобрят в случай, че решат да я продължат;
• Студентите изпълняват поставените им задачи с удоволствие и изпитват удовлетворение от своята работа;
• Студентите знаят в кои области ще могат да прилагат знанията и уменията, които са получили;

• Всички студенти единодушно отчитат ползите от работата с Интернет, като в същото време желаят да имат повече права и възможности да изпълняват своите идеи, за което се нуждаят и от по-добро техническо подпомагане;
• някои студенти изявяват желание да присъстват или получават информация за допълнителни курсове или проекти от същия тип.

Изводи на преподавателите

- Студентите отделят много голямо време за работа по проекта;
- Студентите решават различни проблеми, предимно технически,сами;
- Някои студенти отпадат още в самото начало, но постоянно се информират за прогреса и работата на колегите си;
- Студентите участват в дискусии с колеги от другите групи и по-голяма част от тях успяват да управляват и разпределят работата си вътре в самия малък екип въз основа на индивидуалните си умения и знания;
- Работата по проекта отнема на студентите много повече време отколкото при конвенционалните курсове, като това важи и за преподавателя, и изисква повече комуникация;
- Този начин на работа води до добри междуличностни отношения със студентите.
- Преподавателите трябва да притежават разнообразие от умения, за да могат да ръководят групата към определената цел, да насърчават студентите, да могат да визуализират резултатите от процесите на решаване на проблеми от всякакъв вид и да познават основните медийни технологии от гледна точка на потребителя.

Бъдещето на ESP

Никога преди необходимостта от ESP не е била толкова голяма. Броят на хората, които използват английски език в работата, хобитата и като част от ежедневния си живот постоянно нараства. Крайната цел на обучението по ESP е не само и не толкова да предложи знания, а да развие умения за самостоятелно учене, за учене през целия живот и гъвкаво прилагане на опита в реални ситуации на работното място.

Използвана литература

1. Andrew S. Tanenbaum, Computer Networks - 4th Edition, Prentice Hall Education, 2003.
2. John L. Hennessy, David A. Patterson, David Goldberg, Computer Architecture. A Quantitative Approach, 3rd Edition, Morgan Kaufmann

Publishers, 2003.
3. William Stallings, "Operating Systems" with "Modern Operating Systems", Pearson Education, Limited, 2003.
4. Александър С. Атанасов, Основи на микропроцесорната техника, Страшен вълк, 2007.
5. Димо Д. Арнаудов, Бази от данни, Техника, 1992.
6. Holzinger A. and Motschnig-Pitrik R., Student-Centered Teaching, Meets New Media: Concept and Case Study, 2003.
7. http://www.cplusplus.com/
8. http://students.bfu.bg
9. http://www.moodle.org
10. http://www.php.net
11. .http://www.mysql.com

Програмиране / Уики

Програми
В днешно време компютрите са в състояние да изпълняват много различни задачи, от прости математически операции до сложни анимирани симулации. Но компютърът не създава тези задачи сам, те са представени след серия от предварително зададени инструкции, които отговарят на това което ние наричаме програма.
Компютърът не разполага с достатъчно изобретателност, за да изпълнява задачи, за които той не е бил програмиран, така че може само да следва инструкциите на програми, за които е бил програмиран да изпълнява. Онези, отговарящи за генериране на програми, за да могат компютрите да изпълнява нови задачи са известни като програмисти или кодировачи (coders), които за тази цел използват програмен език.

Програмни езици
Език за програмиране е набор от инструкции и редица лексикални конвенции специално проектирани да наредят на компютрите, какво да правят.
Когато се избира език за програмиране за създаването на проект, много различни съображения могат да бъдат взети в предвид. Първо, трябва да се реши какво е известно като нивото на програмния език. Нивото определя колко близо до хардуера е програмния език. В езиците на по-ниско ниво, инструкциите са написани мислейки пряко за взаимодействие с хардуера, а тези на "високо ниво" се пишат на един по-абстрактен или идеен код.
Като цяло, кода на високо ниво е по-портативен, това означава, че може да работи с повече различни машини с по-малък брой модификации, докато езикът на ниско ниво е ограничен от спецификата на хардуера, за който е бил написан. Независимо от това, предимството на ниското ниво на кода е, че той обикновено е по-бърз. Това се дължи се на факта, че той наистина е написан, като се ползват възможностите на определена машина.
По-високо или по-ниско ниво на програмирането се избира за определен проект в зависимост от вида на програмата, която се създава. Например, когато даден хардуерен драйвер е разработен за една операционна система, очевидно се използва много ниско ниво за програмиране. Докато при големите приложения обикновено се използва по-високо ниво, или комбинация от критични части написани на езици на ниско ниво и други които са на по-високо ниво.

Има езици които са ясно приемани за ниско ниво, като Асемблер, чиито инструкции са настроени за адаптиране към всеки машинен код за който са създадени, а други езици с наследено високо ниво, като Java, който е проектиран да бъде напълно съмостоятелен от платформата върху която ще работи. Езикът C++ е в средна позиция, тъй като може да взаимодейства директно с хардуера почти без никакви ограничения и може също да абстрактва по-ниските нива и да работи като един от най-мощните езици на високо ниво.

Защо C + +?
C + + има определени характеристики над други програмни езици. Най-забележителните от тях са:

Обектно-ориентирано програмиране
Възможността да се ориентира програмирането към обекти позволява на програмиста да проектира приложения от гледна точка по-скоро на комуникацията между обектите, отколкото на структурираната последователност на кода. В допълнение той позволява по-голяма възможна употреба на кода по логически и продуктивен начин.

Преносимост
Практически вие може да компилирате същия C++ код на почти всеки тип компютър и операционна система, без да правите никакви промени. C ++ е най-използвания и преносим програмен език в света.

Краткост
Кодът написан на C ++ е много кратък в сравнение с останалите езици, тъй като използването на специални знаци е предпочитано пред ключови думи, спестявайки някои усилия за програмиста (и удължава живота на нашите клавиатури!).

Модулно програмиране
Тялото на приложението в C ++ може да се изгради от няколко сорс код файлове, които са компилирани поотделно и след това са свързани заедно. Това спестява време, тъй като не е необходимо да се прекомпилира напълно приложението при извършване на единична промяна, а само файла който съдържа. В допълнение, тази характеристика дава възможност за свързване на C ++ кода с код, произведен в други езици, като например Асемблер или C.

C Съвместимост
C++ е обратно съвместим с езика C. Всеки код написан на C може лесно да бъдат включен в C++ програма, без да се правят някакви промени.

Скорост
Полученият код на C + + компилация е много ефективен, поради истинската си двойственост като език на високо и ниско ниво и намаления размер на самия език.

Programs
Nowadays computers are able to perform many different tasks, from simple mathematical operations to sophisticated animated simulations. But the computer does not create these tasks by itself, these are performed following a series of predefined instructions that conform what we call a program.

A computer does not have enough creativity to make tasks which it has not been programmed for, so it can only follow the instructions of programs which it has been programmed to run. Those in charge of generating programs so that the computers may perform new tasks are known as programmers or coders, who for that purpose use a programming language.

Programming languages
A programming language is a set of instructions and a series of lexical conventions specifically designed to order computers what to do.

When choosing a programming language to make a project, many different considerations can be taken. First, one must decide what is known as the *level* of the programming language. The level determines how near to the hardware the programming language is. In the lower level languages, instructions are written thinking directly on interfacing with hardware, while in "high level" ones a more abstract or conceptual code is written.

Generally, high level code is more portable, that means it can work in more different machines with a smaller number of modifications, whereas a low level language is limited by the peculiarides of the hardware which it was written for. Nevertheless, the advantage of low level code is that it is usually faster due to the fact that it is indeed written taking advantage of the possibilities of a specific machine.

A higher or lower level of programming is to be chosen for a specific project depending on the type of program that is being developed. For example, when a hardware driver is developed for an operating system obviously a very low level is used for programming. While when big applications are developed usually a higher level is chosen, or a combination of critic parts written in low level languages and others in higher ones.

Although there are languages that are clearly thought to be low level, like Assembly, whose instruction sets are adapted to each machine the code is

made for, and other languages are inherently high level, like the Java, that is designed to be totally independent of the platform where it is going to run. The C++ language is in a middle position, since it can interact directly with the hardware almost with no limitations, and can as well abstract lower layers and work like one of the most powerful high level languages.

Why C++?
C++ has certain characteristics over other programming languages. The most remarkable ones are:

Object-oriented programming
The possibility to orientate programming to objects allows the programmer to design applications from a point of view more like a communication between objects rather than on a structured sequence of code. In addition, it allows a greater reusability of code in a more logical and productive way.

Portability
You can practically compile the same C++ code in almost any type of computer and operating system without making any changes. C++ is the most used and ported programming language in the world.

Brevity
Code written in C++ is very short in comparison with other languages, since the use of special characters is preferred to key words, saving some effort to the programmer (and prolonging the life of our keyboards!).

Modular programming
An application's body in C++ can be made up of several source code files that are compiled separately and then linked together. Saving time since it is not necessary to recompile the complete application when making a single change but only the file that contains it. In addition, this characteristic allows to link C++ code with code produced in other languages, such as Assembler or C.

C Compatibility
C++ is backwards compatible with the C language. Any code written in C can easily be included in a C++ program without making any change.

Speed
The resulting code from a C++ compilation is very efficient, due indeed to its duality as high-level and low-level language and to the reduced size of the language itself.

Basics of C++ :

During the 60s, while computers were still in an early stage of development, many new programming languages appeared. Among them, ALGOL 60, was developed as an alternative to FORTRAN but taking from it some concepts of structured programming which would later inspire most procedural languages, such as CPL and its succesors (like C++). ALGOL 68 also directly influenced the development of data types in C. Nevertheless ALGOL was an non-specific language and its abstraction made it impractical to solve most commercial tasks. In 1963 the CPL (Combined Programming Language) appeared with the idea of being more specific for concrete programming tasks of that time than ALGOL or FORTRAN. Nevertheless this same specificity made it a big language and, therefore, difficult to learn and implement. In 1967, Martin Richards developed the BCPL (Basic Combined Programming Language), that signified a simplification of CPL but kept most important features the language offered. Although it too was an abstract and somewhat large language. In 1970, Ken Thompson, immersed in the development of UNIX at Bell Labs, created the B language. It was a port of BCPL for a specific machine and system (DEC PDP-7 and UNIX), and was adapted to his particular taste and necessities. The final result was an even greater simplification of CPL, although dependent on the system. It had great limitations, like it did not compile to executable code but threaded-code, which generates slower code in execution, and therefore was inadequate for the development of an operating system. Therefore, from 1971, Dennis Ritchie, from the Bell Labs team, began the development of a B compiler which, among other things, was able to generate executable code directly. This "New B", finally called C, introduced in addition, some other new concepts to the language like data types (char). In 1973, Dennis Ritchie, had developed the basis of C. The inclusion of types, its handling, as well as the improvement of arrays and pointers, along with the later demonstrated capacity of portability without becoming a high-level language, contributed to the expansion of the C language. It was established with the book "The C Programming Language" by Brian Kernighan and Dennis Ritchie, known as the White Book, and that served as de facto standard until the publication of formal ANSI standard (ANSI X3J11 committee) in 1989. In 1980, Bjarne Stroustrup, from Bell labs, began the development of the C++ language, that would receive formally this name at the end of 1983, when its first manual was going to be published. In October 1985, the first commercial release of the language appeared as well as the first edition of the book "The C++ Programming Language" by Bjarne Stroustrup. During the 80s, the C++ language was being refined until it became a language with its own personality. All that with very few losses of compatibility with the code with C, and without resigning to its most important characteristics. In fact, the ANSI standard for the C language published in 1989 took good part of the contributions of C++ to structured programming. From 1990 on, ANSI committee X3J16 began the development of

a specific standard for C++. In the period elapsed until the publication of the standard in 1998, C++ lived a great expansion in its use and today is the preferred language to develop professional applications on all platforms. C++ has been evolving, and a new version of the standard, c++09, is being developed to be published before the end of 2009, with several new features. C++ Language Tutorial Published by Juan Soulie Last update on Jan 11, 2009 at 9:44am UTC These tutorials explain the C++ language from its basics up to the newest features of ANSI-C++, including basic concepts such as arrays or classes and advanced concepts such as polymorphism or templates. The tutorial is oriented in a practical way, with working example programs in all sections to start practicing each lesson right away.

Structure of a program

Примерна програма "Здравей свят!" / "Hello World!" на езика C++

```
#include <iostream.h>
using namespace std;

int main()
{
 cout << "Hello World!" << endl;
 return 0;
}
```

Variables
In order to use a variable in C++, we must first declare it specifying which data type we want it to be.The syntax to declare a new variable is to write the specifier of the desired data type (like int, bool, float...) followed by a valid variable identifier.

For example:
int A;
float mynumber;

If you are going to declare more than one variable of the same type, you can declare all of them in a single statement by separating their identifiers with commas.

For example:
int a, b, c;

This declares three variables (a , b and c), all of them of type int , and has exactly the same meaning as:

int a; int b; int c;

For example, if we want to declare an int variable called a initialized with a value of 0 at the moment in which it is declared, we could write:

int a = 0;

The other way to initialize variables, known as constructor initialization, is done by enclosing the initial value between parentheses:
type identifier (initial_value) ;

int a (0);

Strings
Variables that can store non-numerical values that are longer than one single character are known as strings.
Both initialization formats are valid with strings:
string mystring = "This is a string" ;
string mystring ("This is a string") ;

Променливи
За да използваме променлива в C++, трябва първо да я декларираме като укажем типа данни, който искаме тя да ползва.

Например:
int A;
float mynumber;

Ако декларираш повече от една променлива от същия тип, можеш да декларираш всяка от тях в единно изложение чрез отделянето и от идентификатори със запетаи. Например:

int a, b, c;

Това декларират три променливи (a, b, c), всяка от тях е от тип int, и има точно същото значение както :

int a; int b; int c;

Например, ако искаме да декларираме int променлива означена със стойност " 0 " в момента в който е декларирана, бихме могли да напишем:

int a = 0;

Другият начин за означаване на променливи, познат както конструктор за означаване, се извършва от намиращата се първоначална стойност между скобите (()) :

int a (0);

Низове
Променливи, които могат да се съхраняват извън числени стойности, по-дълги от характерното известни като низове.

Data Types
Fundamental Data Types :

- **int -** integer ;
- **char** - character or small integer;
- **double** - real number, double precision floating point number;
- **bool** - Boolean value. She can take one of two values: true or false;
- **float** - real number, great number of valueless

Integer have two types - long and short.
The integers types can will be " with " or " without " symbol.

Типове данни
Фундаментални типове данни:

- **Int** (integer)- целочислен тип данни;
- **Char** (character) - представяне на единични симиволи или малки цели числа;
- **Double** -реални числа, с фиксирана или плаваща запетая;
- **Bool** (boolean) - булева стойност. Тя може да има една от две стойности: истина или лъжа ;
- **Float** - реално число, множество от стойности.

Integer има два типа - long и short.
Целите типове могат да бъдат "със " или " без "знак (signed/unsigned).

Constants
Constants are expressions with a fixed value.

- Literal constants can be divided in Integer Numerals, Floating-Point Numerals, Characters, Strings and Boolean Values.
- Defined constants (#define)
- Declared constants (const)

Константи
Константите са изрази с фиксирана стойност.

- Буквалните константи могат да бъдат разделени в Integer Numerals (цели числа), Floating-Point Numerals (числа с плаваща запетая), Characters (символи) , Strings (низове) and Boolean Values (булеви стойности).
- Определени константи (#define)

- Декларирани константи (const)

Operators
Operators in C++ are mostly made of signs that are not part of the alphabet but are available in all keyboards.

Assignment (=)
The assignment operator assigns a value to a variable.

a = 5;

The most important rule when assigning is the right-to-left rule: The assignment operation always takes place from right to left, and never the other way:

A = B;

This statement assigns to variable a (the lvalue) the value contained in variable b (the rvalue).

- Arithmetic operators (+, -, *, /, %)
- Compound assignment (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)
- Increase and decrease (++, --)
- Relational and equality operators (==, !=, >, <, >=, <=)
- Logical operators (!, &&, ||)
- Conditional operator (?)
- Comma operator (,)
- Bitwise Operators (&, |, ^, ~, <<, >>)
- Explicit type casting operator
- sizeof()

Оператори
Операторите в C++ са най-вече изградени от знаци, които не са част от азбуката, но са достъпни на всички клавиатури.

*Присвояване (=)

Посредством оператора присвояване се задава стойност на променлива.

a = 5;

Най-важното правило, при задаването е правилото отдясно-наляво : задаващата операция винаги се извършва отдясно наляво, и икога по обратния начин:

А = Б;

Това твърдение задава на променливата а (на lvalue) стойността, съдържащи се в променливо б (на rvalue).

Аритметични оператори (+, -, *, /,%)

- Съставно задание (+ =, -=, *=, / =,% =,>> =, <<=, & =, ^ =, | =)
- Увеличение и намаление (+ +, -)
- Релационни и оператори за равенство (==,! =,>, <,> =, <=)
- Логически оператори (!, & &, | |)
- Условно оператор (?)
- Оператор Запетайка (,)
- Двувалентени оператори (&, |, ^, ~, <<,>>)
- Конвертиращ оператор
- Sizeof ()

Object Oriented Programming
Object-oriented programming (OOP) is a programming paradigm that uses "objects" and their interactions to design applications and computer programs.

Обектно ориентирано програмиране
Обектно-ориентираното програмиране (ООП) е парадигма в компютърното програмиране, при която една програмна система се моделира като набор от обекти, които взаимодействат помежду си.

Templates
Classes or functions parameterized by a seet of types, values, or templates.

Шаблони
Класове или функции, параметризирани чрез набор от типове, стойности и шаблони.

Type Casting
Converting an expression of a given type into another type is known as type-casting .

We have already seen some ways to type cast:
- implicit conversion
- explicit conversion
- dynamic_cast

- static_cast
- reinterpret_cast
- const_cast
- typeid

Конвертиране
Преобразуване на израз от определен тип (данни) в друг тип е известен като type casting).

Ние вече наблюдаваме няколко операции за конвертиране:

- неявно преобразуване, по подразбиране;
- определено преобразуване;
- активно, динамично преобразование;
- неподвижно, пасивно преобразование;
- нестандартно преобразование;
- константно преобразование;

typeid връща псевдоним на обекта typeinfo

Функции

Функция на C++ е начин да се разделят блоковете код на части. Те предоставят на програмиста начин да раздели програмата си така, че да не трябва непрекъснато да се пише един и същи код, а просто да извика функцията. Чрез функциите кода се поддържа малък, чист и функционален.

Всяка функция има три части — прототип (незадължителен, но препоръчителен), заглавна част и тяло.

Прототипът - указва на компилатора, че определена функция съществува, но тялото на функцията е някъде другаде. Може и да не пишете прототип, като сложите заглавната част и тялото над всички извиквания на функцията, но в някои случаи това е невъзможно. Освен това, прототипа помага за изчистването и разбираемостта на кода, което е и главната цел на функциите. Прототипът има следния синтаксис:

<връщан тип> <име> (тип параметър1, тип параметър2,...);

Връщания тип може да бъде всеки тип променлива. Ако не искате функцията да връща тип, то в полето <връщан тип> трябва да напишете `void`. Името на функцията трябва да бъде съставено от символите `A-Z`, `a-z`,

0-9 и _ (долна черта), като не може да започва с цифра. Името трябва чисто и ясно да обяснява какво прави функцията, то не трябва да бъде нито прекалено дълго, нито прекалено късо. Разбира се, вие избирате името, така че функцията ви може да се казва и F(). Последната част от прототипа е описание на параметрите. Това е списък със стойности, които се предават на функцията. Те могат да бъдат колкото поискате, или може изобщо функцията да не приема параметри. Ако не искате да предавате стойности на функцията, тогава оставете мястото между скобите празно или просто напишете в тях void. Ако функцията приема повече от един параметър, то трябва да ги разделите със запетаи.

Ето няколко примера за прототипи:

```
double Square(double Number);
void ShowHelp(void);
double Area(double Width, double Height);
```

За да извикате някоя от тези функции, напишете например:

```
Square(5);
ShowHelp();
Area(5,10);
```

Заглавната част - се пише непосредствено преди тялото на функцията, за да укажете на компилатора коя функция ще описвате. Заглавната част има същия синтаксис като прототипа, но без ';' накрая. Например:

```
double Area(double Width, double Height)
```

Тялото на функцията - е този код, който всъщност се изпълнява; то е това, което прави функцията. Тялото започва с отворена фигурна скоба и завършва със затворена фигурна скоба ({ и }). Между тях трябва да напишете кода на функцията. Ето например реализацията на функцията Area:

```
double Area(double Width, double Height)
{
 double Ar=Width*Height;
 return Ar;
}
```

Чрез `return` указвате какво връща функцията, в случая лицето на правоъгълник с размери `Width` и `Height`. Тоест, ако напишете в `main` или в някоя друга функция следното: *cout<<Area(5,10)<<endl;* на екрана ще се изпише 50.

Приятелство и наследяване

Приятелски функции

Приятелски се наричат функциите, които не са член-функции на клас, но имат достъп до неговите private елементи. Приятелски функции се използват, когато е необходимо една функция да има достъп до private членовете на два или повече различни класа. Декларацията на приятелската функция може да се намира както в частта public, така и в частта private и се характеризира с ключовата дума friend, която се поставя в началото на декларацията. Една приятелска функция се дефинира като обикновена функция, която не е член на клас. Има два вида приятелски функции:

1. Независима приятелска функция: Тези функции се декларират във всеки от класовете като приятелски. Те не са член-функции на класа и затова те нямат достъп до елементите му. Поради тази причина е необходимо подаването на обект по явен начин като параметър на функцията (съответно извикването ще се извършва чрез името на функцията и списък от фактически параметри).

Пример: Дефинираме два класа представляващи марки автомобили с обща функция за определяне на по-бързият автомобил:

```
#include "stdafx.h"
#include "iostream.h"

// Предварителна декларация
class bmw;
class porsche;

class bmw{
private:
unsigned maxspeed;
public:
// Приятелска функция
friend int cmpspeed(bmw *, porsche *);
```

```
bmw(char *, unsigned speed);
char *model;
};

bmw::bmw(char *c, unsigned speed){
model = c;
maxspeed = speed;
}

class porsche{
private:
unsigned maxspeed;
public:
friend int cmpspeed(bmw *, porsche*);
porsche(char *, unsigned speed);
char *model;
};


porsche::porsche(char *c, unsigned speed){
model = c;
maxspeed = speed;
}

// Дефиниция на приятелската функция
int cmpspeed(bmw *a, porsche *b){
// Тя има достъп от елементите maxspeed въпреки че те са private!
return (a->maxspeed - b->maxspeed);
}

void main(){
bmw *x6 = new bmw("X6 xDrive 35i", 239);
porsche *cayenne = new porsche("Cayenne S", 270);
if (cmpspeed(x6, cayenne) < 0){
cout << cayenne->model;
cout << " is faster!" << endl;
}
else{
if (cmpspeed(x6, cayenne) > 0){
cout << x6->model;
cout << " is faster!" << endl;
}
else{
```

```
 cout << x6->model;
 cout << " and ";
 cout << cayenne->model;
 cout << " are at equal speeds";
 } } }
```

2. Член-функция на клас като приятелска за друг клас: При деклариране на приятелската функция трябва да се прецизира класът, на който тя е член-функция с помощта на оператор за принадлежност (::).

Пример: Отново условието от горният пример, но този път функцията cmpspeed е член-функция на класа bmw:

```
 #include "stdafx.h"
 #include "iostream.h"

 class porsche;
class bmw{
 private: unsigned maxspeed;
 public:
  bmw(char *, unsigned speed);
  char *model;
  // За BMW вече cmpspeed е член-функция
  int cmpspeed(porsche *);
};
class porsche{
private: unsigned maxspeed;
public:
// Извикваме функцията като приятелска
friend int bmw::cmpspeed(porsche*);
porsche(char *, unsigned speed); char *model;
};
bmw::bmw(char *c, unsigned speed){
model = c; maxspeed = speed;
}
// Дефиниция на самата функция
int bmw::cmpspeed(porsche *b){
 return (maxspeed - b->maxspeed);
}
porsche::porsche(char *c, unsigned speed){
 model = c; maxspeed = speed;
}

void main(){
```

```
bmw *x6 = new bmw("X6 xDrive 35i", 239);
porsche *cayenne = new porsche("Cayenne S", 270);
if (x6->cmpspeed(cayenne) < 0){
cout << cayenne->model;
cout << " is faster!" << endl;
}  else{
if (x6->cmpspeed(cayenne) > 0){
cout << x6->model;
cout << " is faster!" << endl;
} else{
cout << x6->model;
cout << " and ";
cout << cayenne->model;
cout << " are at equal speeds";
} } }
```

Задача: Дефинирайте класове успоредник, квадрат и правоъгълник и напишете член-функции за изчисляване на лицата и периметрите на фигурите.

Напишете: Приятелска за трите класа функция, която отпечатва на екрана фигурата с най-голямо лице и самото лице.
- Член-функция на класа успоредник, която е приятелска за другите два класа и връща числата 0, 1 или 2 в зависимост от това коя от фигурите е с най-голям периметър.

Полифоморфизъм

В рамките на една и съща област на действие не могат да се обявяват променливи с еднакви имена, но могат да се декларират едноименни функции.

Ако в една и съща програма трябва да се дефинират функции, които дават сходен резултат, например: функция за размяна на стойностите на две променливи от тип int, друга функция за размяна на стойностите на променливи от тип char, трета - за тип double и т.н., бихме могли да означим тези функции съответно с имената *swap_*int*, swap_*char и *swap_*double. При подобна ситуация на програмиста се налага да измисля множество "хитри" имена на своите функции, защото най-подходящото вероятно вече е използвано. Програмата се претрупва с имена на функции, което създава неудобство при използването им.

Езикът C++ дава възможност за обединяване на имена на функции, които имат сходно предназначение.

Полиморфизмът (много форми) е средство, позволяващо едно и също име да се свърже с множество функции с различни дефиниции, действащи в една и съща област.

Така например в една програма могат да се дефинират няколко глобални функции с име swap:

```
void swap(int&, int&);
void swap(char&, char&);
void swap(double&, double&);
```

За да разбере компилаторът коя от всички функции *swap* е извикана, те трябва да се различават по тип и/или брой на формалните си параметри.

Полиморфизъм – използване на различни функции, които имат едно и също име и са свързани с различен тип обекти.

При обръщение към някоя от функциите с еднакви имена, компилаторът сравнява броя и типа на фактическите параметри с броя и типа на формалните параметри във всички едноименни функции и изпълнява онази от тях, при която е регистрирано съответствие.

Въпроси и задачи:

1. Посочете областите на действие на обектите (константи, променливи, функции) в програмата:

```
#include<iostream.h>
const LIMIT = 10000;
int num1, num2;
void f1(int a, int& b);
void f2();
int main()
{
void f3();
f3();
f2();
cout<<num1<<' '<<num2<<endl;
return 0;
}
```

```
void f1(int x, double& y)
{
int a = 10;
y =double(x)/a;
}
void f2()
{
int a=num1;
double b=num2;
f1(a, b);
cout<<a<<' '<<b<<endl;
num2 = b;
}
void f3()
{
do
{
cout<<"num1=";cin>>num1;
cout<<"num2=";cin>>num2;
} while(num1>LIMIT || num2>LIMIT);
}
```

2. В дадена програма са дефинирани функции със следните прототипи:

```
double distance(double x, double y);
double distance(double x1, double y1, double x2, double y2);
```

Коя от тях ще се изпълни при обръщението:
```
d = distance(1.0, a);
```

3. Напишете програма, която дава възможност на потребителя да въведе 2, 3 или 4 реални числа и намира и извежда тяхната средна аритметична стойност. Използвайте полиморфизъм.

Шаблони

Класове или функции, параметризирани чрез набор от типове, стойности и шаблони.

Шаблоните помагат, когато искаме да напишем универсална функция, която да може да работи с всякакви типове данни. Синтаксисът е следния:

```
template <class <име-на-класа>>;
<декларация на функция>
```

Следният пример демонстрира как може да напишем една функция "max" връщаща по-голямото от две числа, като може да й подадем различни типове данни:

```
#include "stdafx.h"
#include <iostream>
using namespace std;

template <class T>
T max (T a, T b) {
T result = (a>b)?a:b;
return (result);
}

void main (){
int a=3, b=4;
int c = max<int>(a,b);
cout << "max(" << a << ",";
cout << b << ") = " << c << endl;

double d=4.6, e=2.18;
double f = max<double>(d,e);
cout << "max(" << d << ",";
cout << e << ") = " << f << endl;

// Може да сравнява и символи по ASCII код
char g='a', h='z';
char i = max<char>(g,h);
cout << "max(" << g << ",";
cout << h << ") = " << i << endl;
}
```

** Забележка: В примера типа данни в ъгловите скоби може да се пропусне при извикването на функцията, защото* променливите *които подаваме са от един и същ тип, както и очакваните входни параметри. Ако предаваме* променливи *от различен тип, то е задължително да укажем в какъв тип ще преобразуваме.*

Възможно е да създадете шаблон, който работи с повече типове:

```
template
A GetMin? (B x, C y) { ... }
```

Ето как можем да използваме шаблони върху класове и техните член-функции:

```
#include "stdafx.h"
#include <iostream>
using namespace std;

template <class T>
class myClass {
T a, b;
public:
myClass(T a, T b){
this->a=a;
this->b=b;
}

T getmax();
};

template <class T>
T myClass<T>::getmax (){
return ((a>b)?a:b);
}

void main () {
int x = 5;
int y = 6;
myClass <int> obj(x, y);
cout << obj.getmax() << endl;

double a = 8.12;
int b = 4;
// Ще преобразуваме данните в double
myClass <double> obj2(a, b);
```

```
cout << obj2.getmax() << endl;
// А сега ще ги преобразуваме в int
myClass <int> obj3(a, b);
cout << obj3.getmax() << endl;
}
```

Виждате, че използвайки шаблони ние можем да правим нашите функции и класове много по-универсални.

Пространства имена

Namespace е конструкция за езика C++, която представлява разширение на познатите досега класове, а именно - един namespace може да съдържа в себе си набори от класове. По този начин от една страна се получава естествено обединение на обектите, които имат сходна функционалност, а от друга се решава един основен проблем в дизайна на програмите. Представете си, че имате два header файла, които искате да използвате:

```
#include "header1.h"
#include "header2.h"
```

Нека обаче в тези файлове се съдържат класове с повтарящи се имена:

```
// header1.h
class myClass {
...
};
...

// header2.h
class myClass {
...
};
...
```

В такъв случай моментално в нашата програма ще се получи конфликт, понеже ще се опитаме да дефинираме myClass два пъти. Този проблем се решава, като използваме namespace *конструкцията:*

```
// header1.h
namespace header1 {
class myClass {
...
```

```
};
}

// header2.h
namespace header2 {
class myClass {
...
};
}
```

Когато искаме да използваме така създадените header файлове ние имаме два метода за работа с класовете в тях. Първият начин е да се извикват директно, чрез указване на namespace:

```
main(){
...
header1::myClass a;
header2::myClass *d = new header1::myClass;
...
}
```

Друг начин е да използваме ключовата дума "using" за един от класовете:

```
main(){
...
using namespace header1;
myClass a; // а ще е от класа в header1
header2::myClass *d = new header2::myClass;
...
}
```

Имайте в предвид, че ако укажете "using namespace header1" и "using namespace header2" отново ще се появи споменатият преди това конфликт. Затова при използване на namespace се препоръчва да се използва операторът "::". Ако дефинирате namespace, който няма име, то той по подразбиране ще бъде включен чрез ключовата дума "using" от компилаторът.

Интересна и често използвана възможност при писане на библиотеки са вложените пространства имена:

```
namespace root
{
class A{
public:
```

```
int var1;
};

namespace sub
{
class A{
public:
int var2;
};
}
}

int main () {
root::A a;
a.var1 = 5;

root::sub::A b;
b.var2 = 6;

return 0;
}
```

Най-често използваното и най-популярно пространство е “std”. Принципно “C++ Standards Committee” са пренаписали напълно стандартната библиотека на C++, като са сложили най-често използваните стандартизирани обекти в това пространство име.

Друго много важно обновяване на езика е въвеждането на нов стил header файлове. Досега не сме ги разглеждали, понеже целяхме да имаме максимална съвместимост за програми писани на C и стари компилатори. В кода на програмата всъщност няма особена разлика - просто не се добавя “.h” в името на файла при include:

Стар стил:

```
#include "iostream.h"
#include "fstream.h"
#include "string.h"
#include "math.h"
#include "stdlib.h"
```

Нов стил:

```
#include <iostream>
#include <fstream>
#include <string>
#include <math>
#include <cstdlib>
```

Въпреки, че имат известни разлики в имплементацията (например fstream.h добавя iostream.h автоматично, а fstream не; string.h използва C-стил низове, а string използва обектни низове; r тн) главната функционалност и най-често използваните функции остават същите. Препоръчително е при писането на програми за нови компилатори да използвате новата конвенция. Нещата, които трябва да запомните са:

1. Никога не "миксирайте", като вкарвате библиотеки от стар и нов стил едновременно.
2. Ако компилаторът поддържа новите библиотеки, то е добре да използвате тях, тъй като се счита, че функционалността и надеждността им е подобрена.
3. Не е изключено в близко бъдеще новите компилатори да не поддържат старите библиотеки.
4. Ако искате да използвате namespace std, то трябва да добавите поне една от новите библиотеки.

По принцип всички функции в новите стандартни библиотеки използват namespace std. Следният пример демонстрира това, като в допълнение показва още една новост за миксиране на глобални и локални променливи с едно и също име:

```
#include "stdafx.h"
#include
// Глобална променлива
int var = 5;

void main(){
// Локална променлива
int var = 10;
```

```
// Отпечатваме локалната променлива
std::cout << var << std::endl;
// Отпечатваме глобалната променлива
std::cout << ::var << std::endl;
// Добавяме namespace std
using namespace std;
// Вече може да използваме cout директно
cout << "Hello world" << endl;
}
```

Забележка: Вижте, че ако не сме добавили "using namespace std" при новия стил функции, то програмата няма да разпознае функциите ако не ги извикаме чрез "std::".

Програмиране / Речник

C string:

A C string is a literal value representing a variable number of characters. An example is "This is a test.". C strings are surrounded by double quotes ("). Please note that this is *not* the same as a C++ string.

С низа е буквално стойност представляваща променлив брой символи. Например: "Това е тест.". **С стринг** са оградени с двойни кавички ("). Моля, имайте предвид, че това не е същото като **С + + стринг**.

Cache:

A **cache** is a small amount of fast memory where frequently used data are stored temporarily.

Кеш е малък размер бърза памет, където често използваните данни се съхраняват временно.

Call instruction:

A **call instruction** is an *assembly language* instruction that is used to implement a *function call*. It saves the program counter on the *stack*, and then transfers execution from the *calling function* to the *called function*.

Call instruction е асемблеска инструкция, която се използва за изпълнение на функцията повикване. Тя пази program counter на стека, а след това прехвърля изпълнението от *calling function* на *called function*.

Called function:

A **called function** is a *function* that starts execution as the result of a *function call*. Normally, it will return to the *calling function* via a **return** *statement* when finished.

Called function е функция, която започва изпълнение в резултат от извикване на функция. Обикновено, тя се връща към повикващата функцията чрез оператора return, когато приключи своята работа.

False:

The keyword **false** is a predefined value, representing the result of a conditional expression whose condition is not satisfied. For example, in the conditional expression x < y, if x is not less than y, the result of the expression will be false. Also see bool.

Лъжа:

Ключовата дума **лъжа** е предварително дефинирана стойност, представляваща резултата от условен израз, чието състояние не е изпълнено. Така например, в условният израз X <Y, ако X е не по-малко от Y, резултата от условието ще бъде неверно (лъжа). Също така вижте bool.

Float:

A float is a type of *floating-point variable* that can represent a range of positive and negative numbers of magnitude from approximately 1.401298e-45 to approximately 3.40282e+38 (and 0), with approximately 6 digits of precision.

Плаваща запетая е тип плаваща променлива, която може да представлява набор от положителни и отрицателни числа с магнитуд от около 1.401298e-45 за около 3.40282e+38 (и 0), с около 6 цифри на прецизност.

Memory hierarchy:

A memory hierarchy is the particular arrangement of the different kinds of storage devices in a given computer. The purpose of using various kinds of storage devices having different performance characteristics is to provide the best overall performance at the lowest cost.

Йерархия на памет е специално подреждане на различни видове устройства за съхранение на даден компютър. Целта на използването на различни видове устройства за съхранение с различни характеристики е да се осигури най-доброто цялостно представяне на най-ниската цена.

Memory leak:

A memory leak is a programming error in which the programmer forgot to delete something that had been dynamically allocated. Such an error is very insidious, because the program appears to work correctly when tested casually. The usual way to find these errors is to notice that the program runs apparently correctly for a (possibly long) time and then fails due to running out of available memory.

Загуба на памет е програмна грешка, в която програмиста забрави да изтрие нещо, което е динамично разпределено. Такава грешка е много

коварна, защото програмата изглежда че работи правилно, когато се тества небрежно. Обичайният начин за намиране на тези грешки е да забележите, че програмата работи правилно очевидно, за вероятно дълго време, а след това не работи, поради изчерпване на наличната памет.

Modification expression:

A **modification expression** is the part of a **for** *statement* executed after every execution of the *controlled block*. It is often used to *increment* an *index variable* to refer to the next *element* of an *array* or a **vector**; see the entry for the **for** statement for an example.

Модификационния израз е част от **for** *инструкция* след изпълнена след всяко изпълнение на *контролният блок*. Той често се използва за инкриментиране(увеличаване) на индекс променлива, за да се отнасе към следващия елемент от масив или вектор; виж **for** инструкцията за пример.

Nanosecond:

A nanosecond is one-billionth of a second.

Наносекундата е една милиардна част от секундата

Native:

A native data type is one that is defined in the C++ language, as opposed to a user defined data type (class).

Прост тип данни е този, който е определен в езика C + +, като обратен на определения от потребителя тип данни (клас).

New:

The **new** operator is used to allocate memory for variables of the *dynamic storage class*; these are variables whose storage requirements aren't known until the program is executing.

Операторът new е използван за разпределение на памет за променливи на *динамичния клас за съхранение*; те са променливи, за които изискванията за съхранение не са известни, докато програмата се изпълнява.

Nondisplay character:

Nondisplay character; see *nonprinting* character.
Nondisplay знак, виж непечатаемите знак.

Nonmember function:

A nonmember function is one that is not a member of a particular class being discussed, although it may be a *member function* of another class.

Не член функция е тази, която не е член на даден **клас**, въпреки че може да бъде *член функция* на друг **клас.**

Object:

An object is a *variable* of a class type, as distinct from a variable of a *native* type. The behavior of an object is defined by the code that implements the class to which the object belongs. For example, a variable of type string is an object whose behavior is controlled by the definition of the string class.

Обект е променлива на тип клас, за разлика от променлива от *прост* тип. Поведението на даден обект се определя от код, който изпълнява класа, към който принадлежи обекта. Например, една променлива от тип низ е обект, чието поведение е контролирано от определението на низа клас.

Object code:

Object code; see *machine code*. This term is unrelated to C++ *objects*.

Обектен код, виж *машинен код*. Този термин не е свързана със C++ *обекти*.

Object code module:

An object code module is the result of compiling a source code module into object code. A number of object code modules are combined to form an executable program. This term is unrelated to C++ objects.

Модул на обектен код е резултат от съставянето на модул на сорс код в обектен код. Модулите на обектен код се обединяват за да формират изпълнима програма. Този термин не е свързана със C++ обекти.

Компютърни Архитектури / Уики

Конспект

1.Архитектура на съвременните процесори.

Архитектурата на съвременните процесори се дели основно на два тип - фон-нойманова и харвардска. По-голяма част от съвременните компютри работят с фон-нойманова архитектура.
Почти 50 години в света на процесорите царстваха принципите, заложени от Фон Нойман. Съвременните процесори са много по различни от своите предшественици. В тях са заложени нови архитектури и технологични решения, които съществено влият на функционалността им.

И в последствие се получават нови черти като:

- суперскаларна архитектура, която включва два или повече конвейера и позволява за един такт на процесора да бъдат изпълнени повече от една инструкция;

- предсказване на преходи в програмата, която се реализира чрез специални логически схеми, които определят точката на предаване на управлението в програмата и осигуряват предварителна подготовка за изпълнение на фрагменти от нея;

- Харвардска архитектура е разделяне на потока команди и данни с помощта на въвеждане на отделни вътрешни блокове кеш памет за съхранение на команди и данни, както и шини за тяхното предаване;

- динамично изпълнение на командите, реализиращо преразпределение на последователността от команди;

- използване на разширен регистров файл с преименуване на регистрите;

- двойна независима шина, съдържаща отделна шина за взаимодействие с кеш паметта L2 и системна шина за обръщения към паметта и външните устройстрва;

- наличие на вътрешни средства за самотестиране, настройка и мониторинг на производителността;.

2.Въведение в паралелната обработка.

Програми

В днешно време компютрите са в състояние да изпълняват много различни задачи, от прости математически операции до сложни анимирани симулации. Но компютърът не създава тези задачи сам, те са представени след серия от предварително зададени инструкции, които отговарят на това което ние наричаме програма.
Компютърът не разполага с достатъчно изобретателност, за да изпълнява задачи, за които той не е бил програмиран, така че може само да следва инструкциите на програми, за които е бил програмиран да изпълнява. Онези, отговарящи за генериране на програми, за да чогат компютрите да изпълнява нови задачи са известни като програмисти или кодировачи (coders), които за тази цел използват програмен език.

Програмни езици

Език за програмиране е набор от инструкции и редица лексикални конвенции специално проектирани да нарадят на компютрите, какво да правят.
Когато се избира език за програмиране засъздаването на проект, много различни съображения могат да бъдат взети в предвид. Първо, трябва да се реши какво е известно като нивото на програмния език. Нивото определя колко близо до хардуера е програмния език. В езиците на по-ниско ниво, инструкциите са написани мислейки пряко за взаимодействие с хардуера, а тези на "високо ниво" се пише на един по-абстрактно или идеен код.

Като цяло, кода на високо ниво е по-портативен, това означава, че може да работи с повече различни машини с по-малкия брой модификации, докато езикът на ниско ниво е ограничен от спецификата на хардуера, за който е бил написан. Независимо от това, предимството на ниското ниво на кода е, че той обикновено по-бърз. Дължи се на факта, че той наистина е написан, като се иполват възможностите на определена машина.

По-високо или по-ниско ниво на програмиране се избира за определен проект в зависимост от вида на програмата, която се създава. Например, когато даден хардуерен драйвер е разработен за една операционна

система, очевидно се използва много ниско ниво за програмиране. Докато при големите приложения обикновено се използва по-високо ниво, или комбинация от критични части написани на езици на ниско ниво и други които са на по-високо ниво.

Има езици които са ясно приемани за ниско ниво, като Асемблер,чиито инструкции са настроени за адаптиране към всеки машинен код за който са създадени, а други езици с наследено високо ниво, като Java, който е проектиран да бъде напълно съмостоятелен от платформата върху която ще работи. Езикът C + + е в средна позиция, тъй като може да взаимодействат директно с хардуера почти без никакви ограничения и може също да абстрактва по-ниските нива и да работи като един от най-мощните езици на високо ниво.

Защо C + +?
C + + има определени характеристики над други програмни езици. Най-забележителните от тях са:

Обектно-ориентирано програмиране
Възможността да се ориентира програмирането към обекти позволява на програмиста да проектират приложения от гледна точка по-скоро на комуникацията между обектите,отколкото на структурираната последователност на кода. В допълнение той позволява по-голяма възможна употреба на кода по по-логически и продуктивен начин.

Преносимост
Практически вие може да компилирате същия C ++ код на почти всеки тип компютър и операционна система, без да правите никакви промени. C ++ е най-използвания и преносим програмен език в света.

Краткост
Код написани на C ++ е много кратък в сравнение с останалите езици, тъй като използването на специални знаци е предпочитан пред ключови думи, спестявайки някои усилия за програмиста (и удължаване на живота на нашите клавиатури!).

Модулно програмиране
Тялото на приложението в C ++ може да се изгради от няколко сорс код файлове, които са компилирани поотделно и след това са свързани заедно. Спестява време, тъй като не е необходимо да се прекомпилира напълно приложението при извършване на единична промяна, а само файла който съдържа. В допълнение, тази характеристика дава възможност за свързване

на C ++ кода с код, произведен в други езици, като например Асемблер или C.

C Съвместимост
C + + е обратно съвместим със C език. Всеки код написан на C може лесно да бъде включен в C++ програма, без да се правят някакви промени.

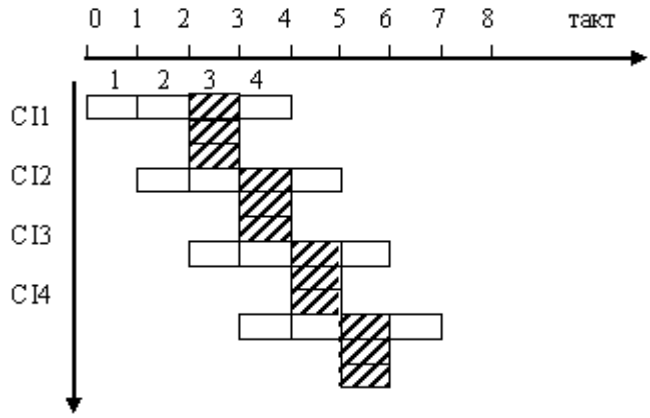
Скорост
Полученият код на C + + компилация е много ефективен, поради истинската си двойственост като език на високо и ниско ниво и намаления размер на самия език.

3. Основни принципи на работата на конвейра.

Целта на конвейра е посредством псевдо-паралено изпълнение на инструкциите да се намали средното време за обработка на инструкция. Повишаване на производителността на процесора се явява в резултат на повишаването на пропускателната способност на процесора за обработка на инструкцията.

Конвейра е реализиран чрез разлагане на инструкцията на отделни сегменти. Всеки един сегмент се изпълнява на отделно устройство в самия процесор. Всеки сегмент от инструкцията трябва да бъде отделен от следващия посредством регистри с цел синхронизация. Всеки сегмент би трябвало да се обработва в рамките на един такт от вътрешната тактова чистота на съответния процесор.

Обща схема на конвейер:



4. Конвейрно изпълнение на командите в процесора.
Конвейрерното изпълнение има няколко етапа, като по извесни са два: извличане и изпълнение. При изпълнение има интервал от време, когато няма обръщение към паметта. Тези интервали могат да са полезни и да се използват за избор на следващи команди и обработване текущо със следваща команда.

Схема на конвейерно изпълнение

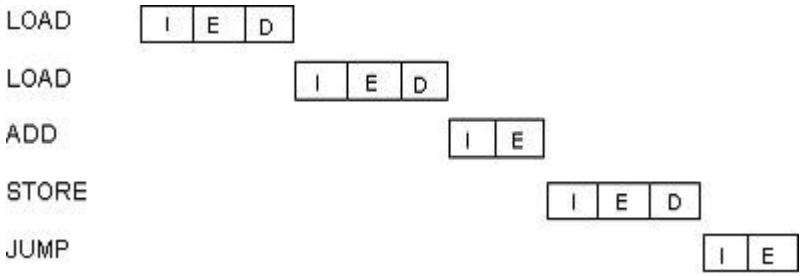
В такъв случай конвейера има две степени - първата изпълнява подфункцията извличане, а втората изпълнение. Също така първата степен осъществява избор на команда и буферизация. Когато втората степен е свободна, първата прехвърля буферираната команда към втората степен. По време на изпълнение на командата от втората степен, първата степен използва свободните цикли за обръщение към паметта за избор и буферизация на следващата команда. Този процес може да се нарича ***предварително извличане на данни***.

Конвейерното изпълнение на командите на процесора ще представим с програмен фрагмент на хипотетично асемблерски език:

LOAD A, M1
LOAD B, M2
ADD A, B
STORE M3, A
JUMP X

A и B са регистри, M1,M2,M3 са клетки от паметта, а X е етикет. Изпълнението на програмата ще свържем с RISC процесор, защото от една

страна е по - лесно конвейерното изпълнение на команди, а от друга страна RISC командите конвейеризират значително по лесно от CISC командите.



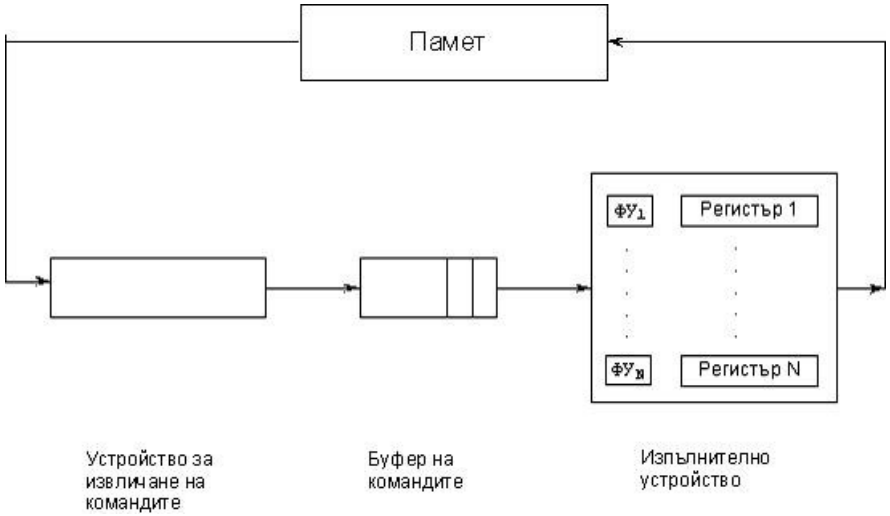
*Фиг: Не*конвейерно изпълнение *на примерна програма.*

I - Избор на команда;
E - Изпълнение на команда;
За изпълнение на операцията за четене от паметта са необходими три фази:
I - избор на командата;
E - изчисление на адреса на паметта;
D - обръщение към паметта;

5. Процесор с множество функционални устройства.
Процесора е основната част на електронно - изчеслителна машина, която изпълнява инструкции от програмното осигуряване. Често в контекста на изчеслителната техника се използва термина процесор. През 70-те, 80-те и 90-те на XX век, централните процесори по - често се използват като една интегрална схема и се наричат микропроцесор. Често пък микропроцесорите вкючват контролерите на паметта.
Централния процесор е най - големия чип на дънната платка. Той е сърцето на компютърната система, изпълнява всички инструкции и борави с данните. Едни от най - важните характеристики на процесора са:

- тип на процесора;
- скоростта, с която работи;
- размер и тип на вкючената в него кеш-памет;
- колко бита е шината за данни;
- колко битова адресна шина поддържа;
- допълнителни процесорни инструкции, които паддържа;
- тип на физическото свързване, което поддържа;

Освен чрез аритметични конвейри, висока скорост на производителност може да се постигне и чрез функционални устройства (ФУ), разположени в единичния процесор, като всяко ФУ се активизира от отделна команда. ФУ могат да са еднотипни по структура (универсални по функционалност) или разнотипни по структура (специализирани по функционалност). Не възниква конфликт при повтарянето на някои ФУ.



За разлика от векторния процесор, ФУ не изискват преобразуване на данните във векторен вид на ниво потребителска програма. Т.е. максимална производителност ще имаме, когато процесорът работи със скаларни операции. Същевременно, този тип система не изисква съществени издръжки, свързани с комуникацията между процесорите.

Естествено, и системата, съдържаща ФУ, има твърде важен проблем за решаване-синхронизацията между устройствата.

Синхронизация в конвейра се постига чрез самата структура на данните в операндите. Тук, обаче, от една страна различните команди могат да заемат различно време за изпълнение, а от друга страна времето за изпълнение се влияе от мястото, където са записани командите и от мястото, където ще бъде записан резултата. А и някои от входните данни за дадена команда могат да са резултат на друга команда, изпълнявана по същото време (съществува зависимост по данни). Проблемът за

синхронизацията е комплексен и се търсят два начина на разрешаването му - апаратен и програмен.

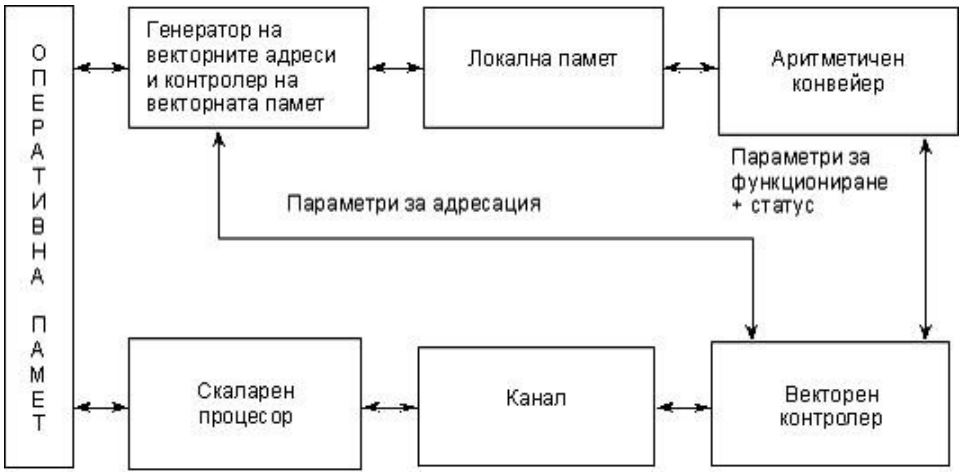
6. Векторни процесори.

Обикновенно векторния процесор (VP) се използва като допълнение в архитектурната организация, която включва и скаларен процесор (SP)

Съществуват два вида векторни процесори:

- **пълен векторен процесор** - тук е по-правилно да се говори за векторен компютър, защото става дума за интегрирано устройство предназначено както за обработка на вектори, така и на скалари;

- **присъединен векторен процесор** - специализирано устройство, което може да се включва в състава на скаларния компютър, като се разширява или по точно увеличава, възможностите за обработка на вектори и матрици. Връзката между двата векторни процесора и скаларния компютър става чрез канал.



Фиг.Обобщена структурна схема на векторен процесор, предложена от Cray

7. Процесорни матрици.

Процесорните матрици служат за обработка на числови масиви (например матрици). Архитектурата на процесора включва матрица от еднакви

процесорни елементи (64x64), работещи едновременно. Той още е синхронен паралелен компютър, който се състои от множество процесори, управлявани от едно единствено управляващо устройство. То извлича и декодира командите от паметта, размножава управляващите команди към всички процесори в матрицата, така всички процесори изпълняват една и съща команда по едно и също време.

12. Архитектура на паметта в паралелните компютри.
Задачата на паметта в компютъра е да съхранява данни или информация и да я предава в бъдещето.В днешно време има два подхода в организацията на паметта: вертикална и хоризонтална.

Вертикалната (йерархичната) организация на паметта
Е идеално структурна организация на паметта, т.е. процесора чете и записва данни само от една единствена памет.Тази организация се нарича още плосък модел на паметта.Но пък това се оказва невъзможно поради следните причини:

- -един бит бърза памет е по-скъп от един бит бавна памет;
- -колкото е по-голям обемът на паметта, толкова по-голямо време е необходимо за достъп до данните, които са съхранени на отделни адреси;

Поради тези причини паметта в съвременните компютри е стуктурирана във вертикала по следния начин: устройството за обработка в паралените компютри има непосредствена връзка с много малка, но бърза памет, достъп до която може да се реализира за един такт. Тази памет се нарича набор от регистри и се явява неотменна част от устройството за обработка.
На следващо ниво в йерархията се намира по-голяма памет, наричана оперативна. По време на изпълнение на задача в нея се записват основните части на програмата и базата от данни. Поради много голямата разлика в обемите на регистровата памет и основната памет, достигаща до няколко десетки порядъка, а също така и поради значителната разлика във времето за достъп (на няколко порядък) в паралелните компютри се включва кеш-памет, заемаща междинно място между регистровата памет и основната памет.
На най-ниското ниво в йерархията на паметта се намира относително бавната, но с изключително голям капацитет вторична (външна) памет. Тя е разположена на магнитни дискове. Обменът на данни между вторичната и оперативната памети е по блокове.

Хоризонтална организация
При тази организация на паметта, целта не е да се намали времето за достъп, както при вертикалната организация, а да се увеличи броят на

достъпите за единица време. Тази организация засяга само основната памет.
Съществуват различни начини за хоризонтална организация, най-простият от който се състои във физическо разбиване на паметта на две половини, достъпът до който може да бъде осъществен едновременно. Поместването на всички команди в едната половина, а данните в другата, увеличава средната скорост на достъп към паметта. Това по същество е Харвардската архитектура на компютъра и за първи път е реализиран в компютърът STRECH на IBM.
Тази идея развита понастоящем води до използуването на няколко модула памети, свързани към независими процесори с помощта на високоскоростна комуникационна мрежа, така че всички модули на паметта са еднакво достъпни за всички процесори.
Хоризонталната организация на паметта може да бъде изградена по два начина:

- пакетна обработка на множеството достъпи към паметта;
- конвейерна обработка на множеството достъпи към паметта

Паралелните компютри
Възможните класификации на паралелните компютри се базират на степента на свързаност на процесорите.
При първият клас компютри времената за обмен на информация между процесорите и средното време за изпълнение на машинните операции са съизмерими, докато във вторият клас времето за обмен на информацията между процесорите е по-голямо от времето за изпълнение на машинните операции. При силно свързаните системи, няколко процесора, чрез комуникационна мрежа директно комуникират с паметта и останалите устройства в системата. При слабо свързаните системи, паралелният компютър се състои от локални компютри, обединени в локална мрежа. Първите системи са познати още като компютри с обща памет – SMP (Symmetrical Multi Processors), а вторите като компютри с разпределена памет MMP (Massively Multi Processors).

14.Производителност на компютъра.Методи за определяне на производителността.

Понятието ПРОИЗВОДИТЕЛНОСТ на компютъра не може да се дефинира еднозначно. Като правило то се обвързва с характера на конкретните практически задачи които се решават с негова помощ и крайните срокове за тяхното изпълнение.

Производителността на компютъра зависи от конфигурацията и състоянието на хардуера, софтуера, квалификацията на потребителя и външни фактори.

- До 1990г. компютрите в масовите си приложения се занимаваха с аритметическа и логическа обработка на данни. Основен критерий за оценка на производителността е била тактовата честота. Колкото е по-висока тактова честота, толкова е по- голям броя изпълнени операции за секунда и съответно по - висока производителността на процесора.
- След 1990г. се появиха нови приложения, като мултимедията, многопроцесорните системи, многозадачните операционни системи и др., които изискват нов подход и критерии при оценка производителността на процесора.

Някои от факторите, оказващи влияние върху производителността са :

- **Разрядност на** процесора **-** колкото по-вече бита данни може да обработи процесорът едновременно, толкова е по-голямо бързодействието му.
- **Тактова честота -** колкото е по-висока тактовата честота на процесора, при равни други условия, толкова е по- висока производителността му. Основно, тактовата честота зависи от дебелината на силициевата подложка на процесорния чип.
- **Архитектура -** преминаването от стандартна към суперскаларна и многопроцесорна архитектури значително увеличи производителността на процесорите**.** В съвременните процесорни архитектури се използват техники като паралелна и конвейерна обработка, предсказване на преходи, двойна кеш памет и др., които натоварват процесорното време и значително ускоряват скоростта на обработка на данните.

Оценка на производителността

Оценка производителността на различните процесори и компютри е сложен процес и зависи от много фактори - производители, архитектура, компоненти и т.н. Съществуват различни методи за оценка, като те се променят с развитието на компютрите.

Компютърни Архитектури / Речник

ACL: Access Control List - Контролен списък на достъпа
ADSL:Asynchronous Digital Subscribers Line - Асинхронна цифрова линия
AI: Artificial Intelligence - Изкуствен интелект
AIT: Advanced Intelligent Tape
ALU: Arithmetic Logic Unit -Аретметично логическо устройство
ATAPI: Attachment Packet Interface
ATM: Asynchronous Transfer Mode - Асинхронен режим на предаване
BIOS: Basic Input Output System - Основна входно изходна система
BSB: Back Side Bus - Задна шина
CICS: Customer Information Control System
CIDR: Classless Inter-Domain Routing
CIS: Computer Information System
CISC: Complex Instruction Set Computing
CMM: Capability Maturity Model
COM: Component Object Model
CPU: Central Processor Unit - Централен процесор
CS: Code Segment - Кодов сегмент
CSS: Cascading Style Sheets
DAT: Digital Audio Tape
DDL: Data Definition Language
DDR: Double Data Rate
DLL: Dynamic Link Library - Динамична свързваща библиотека
DMA: Direct Memory Access - Директен достъп до паметта
DML: Data Manipulation Language
DRAM: Dynamic Random Access Memory - Памет с динамичен достъп
DRL: Data Retrieval Language
DS: Data Segment - Сегмент за данни
DWA: Device Wire Adapter - Жично адаперно устройство
DWH: Data Warehouse - Склад за данни
EDI: Electronic Data Interchange
EOF: End Of File - Край на файла
EOL: End Of Line - Край на реда или чертата
EPP: Enhanced Parallel Port
ES: Data Segment - Сегмент за данни
FAT: File Allocation Table - Таблица за разпределение на файловете
FDDI: Fiber Distributed Data Interface
FDMA: Frequency Division Multiple Access

FIFO: First In First Out - Първия влязъл първи излиза
FPU: Floating Point Unit - Устройство, работещо с плаваща запетая FS: Data Segment - Сегмент за данни
FSB: Front Side Bus - Предна шина
GDT: Global Descriptor Table - Глобална дескрипторна таблица
GS: Data Segment - Сегмент за данни
HDD: Hard Disk Drive
HDLC: High level Data Link Control
HGC: Hercules Graphics Card
HMA: High Memory Area
HTML: Hyper Text Markup Language
HTTP: Hyper Text Transfer Protocol
HWA: Host Wire Adapter
IDE: Integrated Drive Electronics
IDT: Interrupt Descriptor Table - Дескрипторна таблица на прекъсванията
IMAP: Internet Message Access Protocol
IP: Instruction Painter
IPC: Instructions Per Cycle
IPP: Internet Printing Protocol
IRQ: Interrupt Request
ISA: Instruction Set Architecture
ISO: International Organization for Standardization
ISP: Internet Service Provider
IT: Information Technology
JMS: Java Message Service
JSP: JavaServer Pages
LAN: Local Area Network - Локална мрежа
LDT: Local Descriptor Table - Локална дескрипторна таблица
LIFO: Last In Last Out
LLC: Logical Link Control
LRU: Least Recently Used - Последно използван
MAC: Media Access Control
MAN: Metropolitan Area Network
MIDM: Multiple Instruction Multiple DataStream
MMU: Memory Management Unit
MMX: MultiMedia eXtention
MRU: Most Recently Used
MSW: Machine Status Word
MTA: Mail Transfer Agent
MTBF: Mean Time Between Failures
NAT: Network Address Translation
NB: North Bridge
NIC: Network Interface Card или Networked Information Center
NNTP: Network News Transport Protocol

OLAP: Online Analytical Processing
OLTP: OnLine Transaction Processing
OSS: Open Source Software - Софтуер с отворен код
Paging: Странициране
PAP: Password Authentication Protocol
RAM: Random Access Memory
RISC: Reduced Instruction Set Computing
ROM: Read Only Memory
SB: South Bridge
SIMD: Single Instruction Multiple DataStream
SRAM: Static Random Access Memory
SS: Stack Segment
SSD: Solid State Drive
SSE: Streaming SIMD Extensions
STP: Segment Table Pointer
UMA: Unified Memory Access
UWB: Ultra-Wideband
VTOC: Volume Table Of Vontents
WTC: Write True Cashe
WUSB: Wireless USB
XML: Extensible Markup Language
ASP: Active Server Pages - Активни страници на сървър
АЛУ: Аритметико-логическо устройство
БУ: Блок за управление
ФУ: Функционално Устройство

Операционни системи / Уики

Microkernel
Микроядрото извършва малка част от операциите в по-съкратена форма: вътрешнопроцесна комуникация, ограничено управление на процесите и разписания, както и някои входно-изходни операции от ниско ниво. Микрокърнъла се явява по-малко обвързан с хардуера, защото много от системните специфики остават в потребителското пространство. Архитектурата на микроядрата основно е начин за резюмиране на детайлите на контрола на процесите, заделяне на памет и разположение на ресурсите така, че другият чипсет да изисква минимум промени.

Linux
Linux е пример за Unix съвместима операционна система, разработена от Линус Торвалдс. Има няколко версии на Linux (наречени дистрибуции), тъй като нейният автор е предоставил изходния код публично и свободно. Всеки може да свали ядрото на операционната система, добавя свой собствен софтуер и разпространява новия цялостен продукт. Някои дистрибуции са безплатни (плащате само дистрибуционните разходи) и компаниите печелят само от приходи от поддръжка и консултантски услуги (такива примери са Fedora и Mandrake Linux). Някои се плащат на основа компютър/потребител както други софтуерни продукти (вж. урока за лицензиране) – такива са RedHat Linux и SuSE Linux. Днес, тази система се счита за все по-значим конкурент на Microsoft Windows в полето на персоналните компютри, както и при сървърните платформи. Тя осигурява подобен графичен интерфейс, основан на прозорци, също и популярни офис приложения като текстови редактори и електронни таблици. Недостатък на Linux инсталацията е ниската поддръжка на компютърни игри, така че геймърите все още са верни на Windows или на игровите конзоли като Sony PlayStation, Nintendo или X-Box. Популярното лого на Linux е пингвин.
Multiprocessing - две или повече програми, които се изпълняват "видимо" конкурентно под контрола на операционната система. Програмите нямат никаква връзка помежду си освен факта, че се стартират и изпълняват едновременно.

Multitasking - Факт е, че един процесор може да изпълнява конкурентно само едно приложение в даден момент (или за предпочитане процес). Това може да е проблем, ако потребителят иска да работи едновременно с

текстообработваща система, електронна таблица и да играе любимата си игра с карти. Това не би било възможно, ако броят на процесорите е равен на този на приложенията.

Main memory - След процесора един от най-важните компоненти на всеки компютър е неговата памет. Паметта на компютъра е неговата работна област, където той временно съхранява всички файлове, които са му необходими, за да работи.

Multithreading - две или повече задачи, които се изпълняват "видимо" паралелно в рамките на една и съща програма. Понякога се наричат "леки" (lightweight) процеси

NetWare - NetWare е операционна система за сървърни платформи, разработена и продавана от Novell. Дълго време тя беше конкурент на Microsoft Windows NT сървърните системи. Тя винаги е била специализиран продукт, който е несъвместим с другите операционни системи на пазара – затова програмите, разработени за него, няма да се стартират на други системи. Понастоящем, Novell е пуснал в продажба нова версия на NetWare базирана на Linux ядро и SuSE Linux дистрибуция. Занапред, системата няма да е строго специализирана само за сървърни платформи, а ще поддържа и настолни компютри.

Operating system - Операционната система е специален вид софтуер, който се грижи за управлението на всички устройства, софтуерни и хардуерни компоненти в една компютърна система, както и за взаимодействието помежду им. Тя се грижи за разпределяне на ресурсите в системата, и за управление на достъпа до тях на отделните програми.

Processor - Централният процесор е основната част на електронно-изчислителна машина, която декодира и изпълнява инструкциите от програмното осигуряване. Процесорът е най-големият чип на дънната платка. Той е сърцето на компютърната система, изпълнява инструкциите и борави с данните. Представлява малка капсулирана силициева пластина с вградени микроелектронни елементи (транзистори).
Състои се от две основни части:

- Аритметико-логическо устройство (АЛУ)
- Контролно (управляващо) устройство (УУ)

Process - Процес Операционната система работи с един важен фундамент наречен процес. Той е единственият обект, който разпознава и използва. Когато стартирате приложение, системата създава процес и програмата се изпълнява в него. Всички ресурси, заемани от работещо приложение (стартиран процес), са локални за процеса и не могат да се достъпват от

други такива (освен ако самият той не го изисква). Това означава, че ако дадено приложение отвори файл и чете от него, никоя друга програма не може да пише в същия файл. Конкурентното четене и запис, освен ако не са синхронизирани, ще доведе до проблем, ако четящото приложение прочита данни, които веднага ще бъдат презаписани (и така никога не са съществували). Друг пример за сблъсък се появява когато приложение записва данни в RAM паметта. Вече споменахме, че цялата обработвана информация първо трябва да се зареди там. Представете си текстов редактор, намиращ се в RAM паметта. Сега друго приложение иска да стартира игра и се нуждае от паметта, за да зареди картини, карти и позицията на играча. Така играта би могла да пише върху паметта, използвана от редактора и потребителят би видял нещо странно на екрана. Затова паметта също е локална за всеки процес. Когато даден процес се прекъсне неочаквано – например неговите разработчици са допуснали грешка – някой файлове или част от паметта не са правилно освободени. Сега проблемът би бил, че файлът ще е недостъпен завинаги и също така паметта няма да е на разположение. Този проблем ефективно се предотвратява от операционната система – тя помни всички отворени файлове и заделената памет. Когато процесът завърши, всички ресурси се освобождават за бъдеща употреба.

RAM (Random Access Memory) е памет с произволен достъп. Нарича се "с произволен достъп" поради факта, че до всяко място от паметта може да се осъществи достъп толкова бързо, колкото и до произволно друго място.
Паметта служи като буфер между централния процесор и останалите компютърни компоненти.
Основните предназначения на RAM паметта са следните:

- Да съхранява копие от системните софтуеърни програми, които контролират базовите функции на компютъра. Това копие се зарежда в RAM – паметта, когато компютърът се включи и остава там през цялото време, докато той е включен;
- Временно съхранение на копие от приложни програми, чиито инструкции се извикват и изпълняват от централния процесор;
- Временно съхранение на данни, които се въвеждат от клавиатурата или други входни устройства, докато те бъдат съхранени за по-дълго време на устройствата за съхранение на данни или бъдат прехвърлени към централния процесор за обработка;
- Временно съхранение на данни, които са резултат от обработка, докато бъдат извикани от друг процес за обработка или бъдат прехвърлени към изходните устройства като екран, принтер или диск.

Видове RAM памет:

- **Статична RAM памет** (SRAM – Static RAM) – използва се в свръхбързодействащи буферни подсистеми (например като кеш-памет L2). Опакована е в DIL чипове или е вградена в CPU.

- **Динамична RAM памет** (DRAM - DynamicRAM) – основна системна памет, пакетирана като SIMM-ове или като DIMM-ове. За запомнящата клетка се използва кондензатор, който съхранява електрически заряд.

Resourses - Терминът означава всичко, което дадено приложение може да използва за работата си – памет, дисково пространство, процесорно време, мрежова карта и т.н. Тази глава засяга разпределението на ресурси към приложения, работещи под операционната система.

Unix
Unix е голямо семейство операционни системи, споделящи едни и същи стандартизирани концепции при дизайна. Системата е на пазара от десетилетия, но се специализира основно в тежки сървърни системи, изискващи висока наличност и надеждност. Така тя осигурява по-малък потребителски комфорт на неопитни администратори. Има няколко производителя, специализирани в операционните системи Unix - IBM, Sun Microsystems, Hawlett Packard, и т.н. Техните версии на системата не са безплатни и се разпространяват заедно "в кутията" със сървърен хардуер. Този хардуер може да бъде съвместим с обикновените компютри, или със специализирани платформи като IBM Mainframes. Програмите, предназначени за Unix системи (имащи така наречения POSIX стандарт), би трябвало да се изпълняват на всички тях. Част от тези приложения могат да се стартират дори на Microsoft Windows, но този набор е доста ограничен.

Secondary Memory
Това е спомагателна памет за компютри. Тя съхранява данни и програми, които не се използват в момента.Този тип памет е евтина и голяма.

Input/Output processor
Входно-изходен процесор - хардуер, предназначен да обработва входни и изходни операции, за да свали този товар от главния процесор. Например един процесор за цифрови сигнали може да извършва времеотнемащи, сложни анализи и синтез на звукови шаблони без допълнително натоварване на главния процесор.

Digital signal processor

Процесор за цифрова обработка на сигнали. Използва се за проектиране на системи за цифрова обработка на сигнали в комуникациите. DSP-тата могат да използват както инструкции с общо предназначение така и специфични такива, обезпечаващи необходимия брой операции за обработката на сигнали.

System bus
Служи за комуникация между процесора, главната памет и входно/изходните устройства.

MAR
Memory Address Register - съхранява адреса на мястото на паметта, където следващата инструкция ще се изпълни.Докато първата инструкция се изпълнява, адреса на следващата памет се съхранява от MAR.

MBR
memory buffer register - е регистърът в централния процесор, който съхранява данните, който се прехвърлят към и от прекия достъп на съхранението. Работи като буфер, който позволява на централния процесор и паметта да действат независими, без да бъдат афектирани от маловажни различия в операциите.

SPOOL = Simultaneous Peripheral Operations On Line
За да се съгласува високото бързодействие на процесора с бавната работа на някои периферни устройства (четец на карти, печатащо устройство) започва да се използва т.нар. спулинг (от SPOOL - Simultaneous Peripheral Operations On Line), който заменя автономните устройства. При спулинга се използва междинен носител (лента, диск, барабан), върху който предварително се записват подлежащите на обработка задания или получените резултати, но с помощта на процесора. В повечето системи се използват дискове, които отстраняват недостатъка на лентите - цялата лента трябва да се запише, след което се пренавива и започва четене (освен това са необходими няколко ленти вместо един диск). Процесорът непрекъснато управлява въвеждането на задания (нанесени на перфокарти) върху диска, избора и стартирането на задание, изчислението му, и разпечатването на резултатите. Всъщност това е първата форма на мултипрограмиране, като само една от 4-те програми е потребителска (тя чете данните си и записва резултатите от/върху диска).
Докато буферирането позволява припокриване на входно-изходните операции с изчисленията в заданието, спулингът позволява припокриване на входа/изхода на едно задание с изчислението на други. Така спулингът поддържа много по-голямо натоварване на процесора и устройствата, и се получава значително увеличение на производителността. Друго предимство

е възможността за планиране на изпълнението на заданията, разположени върху диска.

RTOS - ОС за реално време.
Друг тип ОС работят в реално време 42, 67, 78, 79, 83, 85, 88. Те се появяват в началото на 60-те години. Характеризират се с това, че действията им се управляват от външни събития, постъпващи в предварително неизвестни моменти. Обработката на всяко събитие трябва да бъде бърза и навременна - в рамките на точно определен интервал от време. Външните събития представляват заявки за обслужване на някакви асинхронно работещи устройства или за приемане на данни. Тези ОС най-често се проектират за управление на технологични процеси (затова още се наричат ОС за управление на процеси). Компютърът е снабден със специална периферия (датчици, аналого-цифрови и цифро-аналогови преобразуватели и др.), чрез която се свързва с управляваната система. ОС за реално време свеждат до минимум участието на човека в процеса на управление. Обикновено се предвижда вмешателство на оператора при непредвидени ситуации или при възникване на грешки.
Има две разновидности на системите за реално време. „Строгите" системи гарантират, че критичните реалновременни задачи завършват навреме. Това изисква закъсненията в системата (от четенето на данни до времената за обработка на заявки към ОС) да са съответно ограничени. Подобни ограничения по време диктуват чертите на строгите системи за реално време. Вторична памет обикновено липсва или се използва съвсем ограничено - данните се пазят в постоянна памет (ROM) или в бърза памет. Много от функциите на съвременните системи също липсват (например, виртуална памет почти винаги липсва). Причината е, че съвременните ОС имат за цел да създават удобства при работа с машината, изолирайки потребителя от апаратурата, което води до неприемливо нарастване на времето за изпълнение на операциите. Изискванията на строгите системи за реално време влизат в конфликт с тези на ОС с времеделене и затова съществуващите ОС с общо предназначение не поддържат функции за реално време от този вид.
По-малки ограничения имат „сметените" системи за реално време, при които критична реалновременна задача се изпълнява, като получава най-висок приоритет до завършването си. И тук закъсненията на системата трябва да бъдат ограничени до приемлива степен - реалновременна задача не може да чака неопределено дълго за изпълнението си. Смекчените системи не могат да се използват за управление на индустриални обекти, роботи и др., тъй като в тях лисват някои от важните механизми на ограничените системи (например, прецизно предпазване от мъртва хватка). Те са полезни, например, в мултимедиините приложения, научни проекти (като подводни изследвания), виртуална реалност и др. За разлика от ограничените системи, тези системи имат нужда от развитите функции на

ОС. Тъй като смекчените системи за реалновреме имат голямо приложение и не влизат в конфликт с другите типове ОС, техните функции са включени в повечето от днешните системи (UNIX, Windows NT, DEC VMS)

TS

Машинна команда. В много процесори има специални машинни команди за организиране на взаимно изключване. Най-често се използва команда от вида TS (Test and Set -провери и установи). Командата TS(a,b) има два параметъра от логически тип. Действието й се свежда до следното. Командата чете стойността на логическата променлива Ь, записва я в а (а:=Ь) и установява b:=true. Всеки процес има локална променлива, а обща е глобална (обща) променлива за всички процеси, конкуриращи се за някакъв ресурс, и е инициализирана с false. Процес (нека да е процес_1) влиза в критичната си секция в зависи!«ост от състоянието на локалната си променлива. Ако няма друг процес в критична секция, тогава обща-false. При изпълнението на TS в цикъла while, локална__1 получава стойност false, а обща - true. В резултат на това процес_1 влиза в критичната си секция, а другите процеси изпадат в цикъл на очакване (изпълнявайки оператор while). При напускане на критичната си секция процес_1 присвоява на обща стойност false, с което се разрешава на друг чакащ процес да влезе в критичната си секция.

ReiserFS

ReiserFS е журнална файлова система създадена специално за Linux от фирмата Namesys. Структурно ReiserFS е сходна както на класическите файлови системи, така и на някои бази данни. Отличава се с много високо бързодействие, особено при работа с голям брой малки файлове.

Поддържа се от Linux ядрото от версия 2.4.1 и е подразбираща се файлова система за някои дистрибуции като Elive, Xandros, Yoper, Linspire, GoboLinux и Kurumin Linux. Преди закупуването на SuSE от Novell бе подразбираща се и за тази дистрибуция.

Операционни системи / Речник

A

AAD: Advanced Applications Database
AAPI: Administrative Application Programmer Interface
AASP: ASCII Asynchronous Support Package
ABATS: Automatic Bit Access Test System
ABR: Area Border Router
AC: Accumulator
ACB: Access Control Block
ACC: Area Communication Controller
ACD: Active Configuration Directory
AFS: Acer Fast Filesystem
AHB: Advanced High-performance Bus
AMD: Advanced Micro Devices
AMD`:Active Matrix Display
APC: Asynchronous Procedure Call
API: Application Programming Interface
AR: Adress Register

B

BAT:BATch file
BCU: Bus Controller Unit
BFS: Be File System
BFT: Binary File Transfer
BIC: Bus Interface Chip
BIOS: Basic Input Output System
BIS: Boot Integrity Services
BIU: Bus Interface Unit
BNU: Basic Network Utilities
BOS: Base Operating System

C

CAD: Computer-aided design
CC: Condition code
CCR: Condition Code Register
CM: Cache Memory
CPU: Central Processing Unit
CS: Cache slot
CSR: Control and Status Registers

CTSS: Compatible Time-Sharing System

D

DA: Destination Address
DAF: Distributed Application Framework
DAL: Dedicated Access Line
DB: DataBase
DCB: Data Control Block
dip: Dual-In-line Package
DLL: Dynamic-Link Library
DLS: Data Loader System
DMA: Direct Memory Access
DPC: Deferred Procedure Call
DR: Data register
DSCB: Data Set Control Block
DSP: Digital Signal Processor
DTFS: Desktop File System

E

EAFS: Extended Acer Fast Filesystem
EMS: Expanded Memory Specification

F

FAT: File Allocation Table
FCB: File Control Block
FCFS: First-Come, First-Served
FCL: Framework Class Library
FEP: Firewall Enhancement Protocol
FFS: Fast File System
FOOL: Foundations of Object-Oriented Languages
FPDP: Front Panel Data Port
FPS: Frames Per Second

G

GDI: Graphics Device Interface
GTLB: Graphics Translation Lookaside Buffer
GUI: Graphical User Interface

H

HAL: Hardware Abstraction Layer
HDD: Hard Disk Drive
HDMI: High-Definition Multimedia Interface
HFS: Hierarchical File System
HPF: Highest Priority First
hpfs: high performance file system
HR: Hit Ratio

I

I/O: Input/Output

I/O AR: Input/Output Address Register
I/O BR: Input/Output Buffer Register
IC: Instruction Cycle
IDT: Interrupt Dispatch Table
IF: Instruction Fetch
ILS: Intelligent Library System
IMS: Information Management System
IPC: Interprocess Communication
IR: Instruction Register
ISP: Internet Service Provider
ISR: Interrupt service routine

J

JCL: Job Control Language

L

LAN: Local Area Network
LIFO: Last-In-First-Out
LPC: Local Procedure Call
LRU: Least-recently-used (пр. least-recently-used algorythm и т.н.)

M

MAC: Project MAC (Machine-Aided Cognition, or Multiple-Access Computers)
MacOS: Macintosh Operating System
MAR: memory address register
MBR: Master Boot Record
MCB: Memory Control Block
MM: Main memory
MP: Multiprogramming
MS-DOS: Microsoft Disk Operating System
MT: Multitasking
MTBCF: Mean Time Between Critical Failures
MTBF: Mean Time Between Failures
MTBSA: Mean Time Between System Aborts
MTBUR: Mean Time Between Unit Replacement
MTTF: Mean Time To Failure
Multics: Multiplexed Information and Computing Service

N

NFS: Network File System
NTFS: NT FILE SYSTEM

O

OBD: Object-Based Disk
OODB: Object-Oriented DataBase
OS: Operating system
OSDI: Operating Systems Design and Implementation
OSI: Open Source Initiative
OSS: Open Source Software

OSWD: Open Source Web Design
OT&E: Operational Test and Evaluation

P

PAD: Protocol Anomaly Detection
PAD`: Packet Assembler/Disassembler
PAS: Personal Access System
PC: Program counter
PC`: Personal Computer
PCB: Process Control Block
PID: Process IDentifier
PnD: Plug-and-Display
PnP: Plug-and-Play
PSN: Processor Serial Number
PSW: Program status word

Q

QFS: Quick File System

R

RAID: Redunant Array of Independent Disks
RAID`: Redundant Array of Inexpensive Disks
RAIL: Remote Assisted Instructional Learning
RAM: Random access memory
RAMDAC: Random Access Memory Digital-to-Analog Converter
RAPI: Remote Application Programming Interface
RAS: Remote Access Service
RDMA: Remote Direct Memory Access
RDMAP: Remote Direct Memory Access Protocol
Reg: Register
RIP: Raster Image Processor
RP: Reentrant procedure
RPC: Remote Procedure Call
RR: Round-Robin Scheduling
RRAS: Routing and Remote Access Service
RTOS: Real-Time Operating System
RTT: Real Time Task

S

SB: System bus
SF: Stack frame
SI: Stack Implementation
SJF: Shortest Job First
SJN: Shortest Job Next
SL: Spatial locality
SL`: Stack limit
SM: Secondary memory
SMP: Symmetric Multiprocessing

SP: Segment pointer
SPN: Shortest Process Next
SPOOL: Simultaneous Peripheral Operations On Line
SRT: Shortest Remaining Time
SVR4: System V Release 4

T

TCB: Task Control Block
TL: Temporal locality
TS: Test and Set
TSA: Target Service Agent

U

UCB: Unit Control Block
UFS: Unix File System
Unics: Uniplexed Information and Computing System
UP: Uniprogramming (single-program)
UUCP: Unix to Unix Copy Protocol
VDM:Virtual DOS Machine

W

WAN: Wide area network

Компютърни мрежи / Речник

AAA = authentication, authorization and accounting
AAUI = Apple Attachment Unit Interface
ABR = available bit rate
ADSI(1) = Analog Display Services Interface
ADSI(2) = Active Directory Service Interfaces
AES = Advanced Encryption Standard
AFP(1) = AppleTalk Filing Protocol
AFP(2) = Advanced Function Presentation
AMPS = Advanced Mobile Phone Service
ANS : Advanced Network Series
AODV : Ad Hoc On-Demand Distance Vector
AP : Access Point
ARPU : Average Revenue Per User
AUI : Attachment Unit Interface
AVI : Audio Video Interleave
BER : bit error rate
BERT : bit error rate test or tester
BPDU : bridge protocol data unit

buffer Underrun:

A common problem that occurs when burning data into a CD. It happens when the computer is not supplying data quickly enough to the CD writer for it to record the data properly. Recording data to a CD-R is a real-time process that must run nonstop without interruption of the signal. A computer will typically transfer the data to the CD-R faster than it is needed. The CD-R drive stores the incoming data in a buffer as a reserve of data waiting to be written so that minor interruptions or slowdowns in the data flow will not interrupt the writing process. The larger the buffer, the greater the chances of a successful transfer. A buffer underrun error occurs when the flow of data from the original source, such as a hard drive or a CD-ROM drive, was interrupted long enough for the recorder's buffer to empty. The writing action is stopped when this happens, and the recordable disc may be ruined during a write operation.

The most common causes of buffer underrun are out-of-date drivers or a system that does not meet the minimum requirements for CD burning. Before attempting to record onto a CD, check that your CPU and hard drive are fast enough to support CD recording and that your computer has enough RAM and

available hard disk space. It is also important to have the latest drivers for the CD-R drive, IDE or SCSI controller.

CBR : Constant Bit Rate
CCNA : Cisco Certified **Network** Associate
CD : Compact Disk
CD-ROM : Compact Disc-Read-Only Memory
CDN : Content Delivery Network
CDP(1) : Continuous Data Protection
CDP(2) : Cisco Discovery Protocol
CGI : Common Gateway Interface
CIDR : Classless Inter-Domain Routing
CMTS : Cable Modem Termination System
CSMACD : Carrier Sense Multiple Access Collision Detection
CSS : Cascading Style Sheets
DCF : Distributed Coordination Function
DDR : Dial-On-Demand Routing
DES : Data Encryption Standard

Directory Service:

A network service that identifies all resources on a network and makes them accessible to users and applications. Resources include e-mail addresses, computers, and peripheral devices such as printers. Ideally, the directory service should make the physical network topology and protocols transparent so that a user on a network can access any resource without knowing where or how it is physically connected.

There are a number of directory services that are used widely. Two of the most important ones are LDAP, which is used primarily for e-mail addresses, and Netware Directory Service (NDS), which is used on Novell Netware networks. Virtually all directory services are based on the X.500 ITU standard, although the standard is so large and complex that no vendor complies with it fully.

DLCI : Data Link Connection Identifier
DMCA : Digital Millennium Copyright Act
DMI : Desktop Management Interface
DMT : Discrete Multitone
DMTF : Distributed Management Task Force
DMZ : Demilitarized Zone
DOCSIS : Data Over Cable Service Interface Specification

Domain Controller :
As defined by Microsoft, in Active Directory server roles, computers that function as servers within a domain can have one of two roles: member server or domain controller. Abbreviated as DC, domain controller is a server on a Microsoft Windows or Windows NT network that is responsible for allowing host access to Windows domain resources. The domain controllers in your network are the centerpiece of your Active Directory directory service. It stores user account information, authenticates users and enforces security policy for a Windows domain.

DSLAM : Digital Subscriber Line Access Multiplexer
DSML : Directory Service Markup Language
DVD : Digital Versatile Disc Digital Video Disc
EAI : Enterprise Application Integration
ENI : Ethernet Networking Interface
FCIP : Fibre Channel Over Ip
FDM : Frequency Division Multiplexing
FHSS : Frequency-Hopping Spread Spectrum
FSMO : Flexible Single Master Operations
GAN : Global Area Network
GBIC : Gigabit Interface Converter
GBPS : Gigabits Per Second
GSM : Global System For Mobile Communications
HFC : Hybrid Fiber Coax
HPNA : Home Phoneline Networking Alliance
HSRP : Hot Standby Routing Protocol
HTML : Hypertext Markup Language
IAP : Internet Access Provider
IBM : International Business Machines
ICS : Internet Connection Sharing
IDF : Intermediate Distribution Frame

IS(1) : Information Systems
IS(2) : Information Services
ISP : Internet Service Provider
ITU : International Telecommunication Union
KBPS : Kilobits Per Second
LAT : Local Area Transport
LDAP : Lightweight Directory Access Protocol
LDCM : Landesk Client Manager
LMDS : Local Multipoint Distribution Services
LOM : Lan On Motherboard
LSI : Large-Scale Integration
LVDS : Low Voltage Differential Signaling
MACA : Multiple Access With Collision Avoidance
MAE : Metropolitan Area Ethernet
MBPS : Megabit Per Second
MDF : Main Distribution Frame
MDIX : Medium Dependent Interface Crossover

MegaTransfer :
A MegaTransfer is a unit of measurement that refers to the rate of signal on parallel I/O buses (like SCSI) where the data transfer rate depends upon the amount of data transferred in each data cycle, and is independent of the width of the bus. MegaTransfer is abbreviated at MT and is commonly seen written in the per second form; MT/s or MT/Sec.

member server:

As defined by Microsoft, in Active Directory server roles, computers that function as servers within a domain can have one of two roles: member server or domain controller. A member server is a computer that runs an operating system in the Windows 2000 Server family or the Windows Server 2003 family, belongs to a domain, and is not a domain controller. Member servers typically function as the following types of servers: file servers, application servers, database servers, Web servers, certificate servers, firewalls and remote-access servers.

MIB : Management Information Base
MIS : Management Information System Or Management Information Services
MMDS(1) : Multipoint Microwave Distribution System
MMDS(2) : Multi-Channel Multi-Point Distribution System
MP : Multilink Point-To-Point Protocol
MP3 : Moving Picture Experts Group-1 Audio Layer 3
MPEG : Moving Picture Experts Group

MTU : Maximum Transmission Unit
NAS : **Network**-Attached Storage
NDS : Novell Directory Services
NOC : Network Operations Center
PALMTOP : Джобен Компютър
PIO : Programmed Input/Output
PIX Firewall : Private Internet Exchange Firewall
RBGAN : Regional Broadband Global Area Network
SCSI : Small Computer System Interface

Shared Ethernet :
The traditional type of Ethernet, in which all hosts are connected to the same bus and compete with one another for bandwidth. In contrast, a switched Ethernet has one or more direct, point-to-point connections between hosts or segments. Devices connected to the Ethernet with a switch do not compete with each other and therefore have dedicated bandwidth.

Switched Ethernet :

An Ethernet LAN that uses switches to connect individual hosts or segments. In the case of individual hosts, the switch replaces the repeater and effectively gives the device full 10 Mbps bandwidth (or 100 Mbps for Fast Ethernet) to the rest of the network. This type of network is sometimes called a desktop switched Ethernet. In the case of segments, the hub is replaced with a switching hub.

Traditional Ethernets, in which all hosts compete for the same bandwidth, are called shared Ethernets. Switched Ethernets are becoming very popular because they are an effective and convenient way to extend the bandwidth of existing Ethernets.

Terminator :

A device attached to the end-points of a bus network or daisy-chain. The purpose of the terminator is to absorb signals so that they do not reflect back down the line. Ethernet networks require a terminator at both ends of the bus, and SCSI chains require a single terminator at the end of the chain.

UBR : Unspecified Bit Rate
VBR : Variable Bit Rate
XSP : Extensible Server Pages

Компютърни мрежи / Уики

Конспект

1. Компютърни мрежи. Класификация. Еталонен модел. Кратка характеристика на отделните нива.
2. Локални мрежи (LAN). Характеристика и особености. Топология на локалните мрежи. Реализация и стандарти. Характеристика на коменикационните среди. Модулация на сигналите.
3. Локални мрежи. Особености на мрежи с множествен достъп. Методи за достъп CSMA/CD . Особености. Оценка производителността на мрежи с метод за достъп CSMA / CD .
4. Локални мрежи. Формат на кадрите при мрежи IEEE 80 2 .1. Реализация на локални мрежи. Разлика между IEEE 80 2 . 3 и Ethernet .
5. Локални мрежи. Детирмистични методи за достъп. Стандарт 802.4. Формат на кадрите.
6. Локални мрежи. Кръгови топологии. Реализация на Token Ring . Формат на кадъра. Мениджмънт на кръга.
7. Локални мрежи. Мрежи от тип FDDI . Физически носещи среди. Характеристика на кабелните среди. Формат на кадъра. Особености на двойния кръг. Реализация на мрежи от тип FDDI .
8. Локални мрежи. Реализация на виртуални мрежи на канален слой.
9. Канален слой при глобалните комуникационни мрежи. ATM . Характеристика и особености.
10. Frame Relay . Характеристика и особености.
11. TCP/IP . Архитектурен модел. Протоколни тестове. TCP/IP – приложения. Модел от тип клиент/сървър.
12. Интернет протокол (IP). Основи на IP – адресирането. Особености на отделните класове адресиране.
13. Интернет протокол (IP). Служебни адреси. Мрежови маски. Предназначение на мрежовите маски.
14. Интернет протокол (IP). Подмрежи. Видове. Особености на разпределение на мрежовото пространство. Мрежови подмаски. Видове разпределение.
15. Интернет протокол (IP). Служебни адреси. Методи за предаване на данни. Множествени адреси. Лимитирани обръщения. Групово адресиране.
16. Интернет протокол (IP). Мрежови адреси за вътрешно ведомствени мрежи. Използване на вътрешно ведомствени мрежи.

17. Интернет протокол (IP). Маршрутизация. Видове. Особености и реализация. Маршрутизационни таблици. Основи на алгоритъм за маршрутизация.
18. Интернет протокол (IP). Безкласово рутиране между домейн. Реализация.
19. Интернет протокол (IP). Визуализация на IP – мрежата.
20. Интернет протокол (IP). Формат на кадъра. Служебни полета. Фрагментация на кадри.
21. Протоколи ARP и ICMP . Обща характеристика. Приложение. Прозрачно свързани мрежи.
22. Основи на IP – рутирането. Процес на рутиране. Автономни системи. Статично рутиране.
23. Основи на IP – рутирането. Дистанционно векторно рутиране. Проблемът с броенето до безкрайност.
24. Основи на IP – рутирането. Разцепен хоризонт. OSPF .
25. Основи на IP – рутирането. Протокол RIP . RIP – 2. RIP за IPv6 .
26. Основи на IP – рутирането. Външни протоколи за рутиране. EGP . BGP – 4.
27. Основи на TCP . Концепция. Особености на свързаната с информационния поток комуникация. Мониторинг на връзката при TCP.
28. Основи на TCP. Алгоритми за контрол на информацията. Метод "плаващ прозорец".
29. Домейни. Сървари за обслужване на имената. Йерархия на имената в Интернет. Разпределение на пространството.
30. Настройка на DNS . Видове записи. Видове бази данни. Администриране на сървър DNS .
31. Електронна поща. Формиране на имена. Сървъри за електронна поща. Разпределение на информацията. Настройка на сървър при Windows / NT . Особености при Unix / Linux .
32. Електронна поща. Протокол POP . Протокол IMAP . Особености при реализацията.
33. Трансфер на информация. Особености на протокол за файлов трансфер FTP . Реализация на FTP – сървър. Особености при настройката на FTP – сървър при Windows / NT и Linux .
34. Трансфер на информация. Клиенти за FTP реализация. Особености при различни платформи.
35. Реализация на тунели. Протоколи.
36. Реализация на модемни комуникации. Технически характеристики на модемите. Настройка на сървър и клиент.
37. Защита на информацията. Кодиране. Реализация на сигурни комуникации. SSH . Firewall. Реализация при Windows/NT и Unix/Linux .
38. Защита на информацията. Реализация на маскирани мрежи. Особености при реализацията при различните платформи.

39. Други приложения. Telnet . Особености. Графични терминали. Реализация.
40. Мрежови услуги. HTTP . Реализация на сървър и клиент. Особености при настройката. Виртуални сървъри.
41. Мрежови операционни системи. Реализация на разпределени компютърни системи.
42. Безжични комуникации. Реализация на радиомрежа. Реализация на сателитни комуникации.

Network

Компютърна мрежа - представлява група компютри и (или) други устройства, които са свързани помежду си за обмен на информация и за съвместно използване на ресурси .

Local Area Networks
Локална мрежа, най - често се реализира чрез Ethernet .

Fiber Distributed Data Interface
е технология за локални мрежи използваща метода за достъп до канала token passing и определяща оптични кабели като медия за пренос на данните. По спецификация FDDI функционира на скорост от 100 mbps, използва логическа топология кръг и физическа топология двоен кръг.

PAN
Персонални мрежи (Personal-Area Network) включват компютърни устройства използвани от един човек или в рамките на един офис. Обхвата им не превишава 10 m. В PAN се включват настолен компютър, лаптоп, PDA, принтер, телефон, факс или мултимедийни устройства.

LAN
Локални мрежи (Local - Area Network) включват компютри, мрежови адаптери, периферни устройства, среда за предаване на данни и други мрежови устройства. Те имат следните характеристики:

- Функционира в ограничена географска област;
- Притежава широка пропускателната способност;
- Предоставя постоянен достъп до ресурсите.

WAN
Глобалните мрежи (Wide-Area Networks) представляват комбинация от локални мрежи и допълнителни комуникационни връзки между тях. WAN съединяват потребители, разположени в обширни географски области.

MAN
Регионални (градски) мрежи (Metropolitan-Area Network) се разглеждат като WAN в сравнително малка географска площ-град, област.

VPN
Виртуална частна мрежа (Virtual Private Network) се нарича частна мрежа, получена в резултат на обединение на няколко териториално разделени LAN , с помощта на общодостъпни канали на глобални мрежи (например Internet).

SAN
Мрежите от хранилища данни (Storage-Area Networks) представляват специализирани високоскоростни мрежи, които свързват сървъри и хранилища на ресурси.

БЕЛЕЖКИ