ТЕМА 14: ПРПСП

МАШИННИ И ПРОЦЕСОРНИ АРХИТЕКТУРИ С ПАРАЛЕЛНА ОБРАБОТКА

- **Последователен модел.** Това е архитектурният модел на фон Нойман, като основната му характеристика е наличието на едно устройства за обработка (процесор), през което преминава целият входен поток от задачи, формиращ работното натоварване.
- **Конвейерен архитектурен модел**. Основава се декомпозиция на глобалната функция на изчислението на отделни фази, изпълнявани независимо в отделни функционални (процесорни) устройства. Ефектът се търси при обработка на множество еднотипни задачи, позволяваща едновременното натоварване на отделните устройства с различни фази от различни задачи (постъпващи последователно).
- **Паралелен архитектурен модел.** Съвкупност от независими процесорни устройства, работещи паралелно и съгласувано по обработката на единен (общ) или независим (множествен) поток от данни.

КЛАСИФИКАЦИЯ НА ФЛИН

Класификацията на Флин се базира не на структурата на компютъра, а на това как командите в компютъра се обвързват с обработката на данните. Флин въвежда понятието поток и го дефинира така: последователност от елементи (данни и команди) изпълнявани или обработвани от процесорите.

Във всеки компютър има два основни потока - единият **от команди**, а другият **от данни**. В съответствие с това дали потоците от данни или команди са единични или представляват множество, възникват и следните четири големи класа:

- **SISD** - Към класа SISD спадат обикновените фон Нойманвски компютри, които имат един поток команди и един поток данни, т.е. едно устройство за обработка на командите и едно изпълнително устройство (АЛУ).
- **SIMD** - Когато има само едно управляващо устройство (control unit) и всички процесори изпълняват синхронизирано една и съща инструкция, паралелната машина се класифицира като SIMD.
- **MISD** - Към този клас компютри се отнасят компютри с множество потоци от команди и един поток от данни, т.е. на всички конвейери. Единствената архитектура, която с някакво основание може да бъде отнесена към този клас е конвейерната и то при условие, че всеки етап от изпълнението на командата се счита за една отделна команда.
- **MIMD** - При MIMD машината всеки процесор има свое собствено управляващо устройство (control unit) и може да изпълнява различни инструкции върху различни данни. MIMD паралелните архитектури се състоят от множество процесори и от множество модули памет, свързани чрез комуникационна (свързваща) мрежа. Тези системи попадат в **две основни категории**: системи с обща памет – **shared memory** (процесорите обменят информация чрез тяхната централна споделена памет) или с обмен на съобщения - **message passing** (процесорите обменят информация посредством комуникационната мрежа).

Системи с обща памет (Shared memory)

При **системите със споделена памет** координацията между процесорите се осъществява чрез глобална памет споделена от всички процесори. Всеки процесор може да има регистри, буфери, кеш и локални части памет като допълнителни ресурси на паметта. В зависимост от комуникационната връзка (interconnection network) системите със споделена памет могат да бъдат класифицирани като:

- **UMA** - В системите UMA споделената памет е достъпна за всички процесори посредством комуникационна мрежа, по същия начин, по който единичен процесор достъпва неговата памет. Поради тази причина всички процесори имат еднакво време за достъп до всяко място от паметта.
- **NUMA** - В NUMA системите всеки процесор има част от общата памет. Паметта има едно адресно пространство, затова процесорите могат да достъпят всяка част от паметта директно, използвайки неговият реален адрес. Времето за достъп обаче зависи от разстоянието до процесора.
- **COMA** - Подобно на NUMA и в COMA системите всеки процесор има част от споделената памет. В този случай обаче споделената памет се състои от кеш памет, част от която може да се адресира отдалечено

Системи с обмен на съобщения (Message passing)

Системите с обмен на съобщения са клас мултипроцесорни системи, който предоставят алтернативни методи за комуникация и придвижване на данни между мултипроцесорите. При този тип системи няма глобална памет, всеки процесор има достъп до своя собствена локална и може да комуникира с другите процесори посредством комуникационната мрежа чрез изпращане и получаване на съобщения.

DATA FLOW (ПОТОКОВИ АРХИТЕКТУРИ)

При класическите фон Нойманови архитектури, програмата е последователност от инструкции – **control flow,** докато при Data flow архитектурите контролът се осъществява чрез данните. Представят се с общи потокови графики – Data flow diagrams.

ПРОЦЕСОРНИ АРХИТЕКТУРИ

Широк архитектурен клас от компютри. Пилагат се следните архитектури: **скларни, суперскаларни (RISC, CISC), процесорни – VLIW, векторни, суперконвейерни**

Броят на конвейрите в един процесор може да бъде различен и от тях зависи колко инструкции теоретично могат да се обработят за един такт.

- Процесор с един конвейер се нарича **скаларен**.
- По - развитите процесори, които включват няколко конвейера се наричат **супер скаларни**.

- **CISC** (**Complex Instruction Set Code)-** процесори със сложен набор команди - ***технологията CISC*** е свързана с традиционните процесори, при които се поддържат множество инструкции, изпълнявани за различно време.

- **RISC** (**Reduced Instruction Set Code**) - процесори със съкратен набор команди - разчита на повече на брой максимално опростени инструкции, всяка от които върши определена дейност. Тъй като въвеждането на конвейерната обработка, както и редица други подобрения изискват наличието на опростени инструкции, трябва да се намери начин да се опростяват макроинструкциите от CISC набора.

ПРОЦЕСОР С МНОЖЕСТВО ФУНКЦИОНАЛНИ УСТРОЙСТВА

Висока скорост на изчисление може да се получи и по начин различен от аритметичния конвейер, а именно чрез множество функционални устройства вътре в единичния процесор.

Синхронизация на апаратно ниво - Синхронизацията се реализира по време на изпълнение на програмата. По същество това е конвейер за команди, в който с цел намаляване на времето за изпълнение на командите, изпълнителното устройство е съставено от няколко паралелно работещи ФУ . Така става възможно да се реализира **функционален паралелизъм** заложен в програмата. Като правило

ФУ взаимодействат само с регистрите. Така сравнително лесно се решава един от проблемите със синхронизацията - зависимостта на времето за изпълнение на командата от местоположението на данните.

Синхронизация на програмно ниво - В този случай компилаторът е отговорен за насрочването на изпълнението на командите, т.е. за синхронизацията на множеството ФУ. Този подход води до архитектура известна под името процесор със свръхдълга команда - **VLIW** (**V**ery **L**ong **I**nstruction **W**ord).

- **VLIW** (**Very Long Instruction Word**) - процесор със свръхдълга команда. В известен смисъл **VLIW** е логическо разширение на **RISC** процесорите. При **VLIW** архитектурата, компилаторът пакетира няколко прости команди в една дълга дума на командите. Всяко поле от тази дълга дума управлява директно едно функционално устройство.

ВЕКТОРЕН ПРОЦЕСОР

Става въпрос за обработка на голям обем на данни, и то структурирани във вид на масиви. Такива приложни задачи са обработката на изображения, обработката на сеизмични данни, управлението на ядрените реактори, прогнозирането на климата и т.н. **Векторният процесор** осигурява паралелно изпълнение на операции с масиви от данни, вектори. Той се характеризира със специална архитектура, базирана на група паралелно работещи процесорни елементи.

ПАРАМЕТРИ НА ПАРАЛЕЛНАТА И РАЗПРЕДЕЛАНА ОБРАБОТКА, МЕТРИКА, МЕТОДИ ЗА АНАЛИЗ.

Развитието на паралелната обработка е свързано със следните принципи и закони.

- **Принцип на неограничения паралелизъм -** Ако съществува паралелен алгоритъм за изпълнение на дадена задача, то той се реализира в хипотетична паралелна КС, за която няма ограничения относно функционалните възможности.
- **Закон на Amdahl -** Всеки процес, подлежащ на разпаралелване, се състои от една част, която може да бъде изпълнена паралелно и от друга част, която трябва да се изпълни последователно. Системната производителност не може да надвиши определено ниво, определено от наличието на последователно-изпълнима част от програмата

Паралелната обработка е свързана с изпълнение на паралелна програма в подходяща парлелна КС (ПКС).Една задача може да бъде изчислена паралелно, ако може да се състави **паралелен алгоритъм**, допускащ реализация върху хипотетична паралелна КС.

Проектирането на ПА минава през следните фази:

- **Разделяне** (**partitioning**) – декомпозоция на проблема по данни или по функции.
- **Комуникации** (**communication**) – формулира информационните или контролните зависимости между отделните подзадания. Комуникациите са представят като канали и съобщения (т.е. данни и команди), които се предават по тези канали.
- **Форматиране** (**agglomeration**) – формулираните подзадания и прилежащите им комуникации, се групират в задания, при което се отчитат характеристиките на архитектурата на обработка – основно брой процесори/възли и комуникационен модел – и в резултат се постига оптимизиране по следните характеристики: грануларност и балансираност, евентуално репликиране на данни и подзадания, оптимизиране на комуникациите, евентуално запазване на линейност (скалируемост) и технологично оптимизиране.
- **Разпределение** (**mapping**) – незадължителна фаза, която се състои в разпределение на форматираните задания по обработващите възли на системата със кодиране на съответното решение.

Метрика и анализ на производителността

Сложността на последователните алгоритми се оценява като функция само на размера на проблемната област и следователно може да се оцени абстрактно от архитектурата. За разлика от тях обаче при паралелните алгоритми сложността е функция на архитектурата и на средата за паралелна обработка.

- **Степен на паралелизъм** – максималния брой операции, които могат да се изпълнят паралелно при обработката на алгоритъма – това е архитектурна величина.
- **Ускорение (acceleration)**
  - $S_p = \frac{T_1}{T_p}$, където T_1 е последователното време за изпълнение, а T_p е паралелното време за изпълнение
- **Ефективност (eficiency)**
  - $E_p = \frac{S_p}{p}$, където S_p е ускорението, а p е броят използвани компютри
- **Цена при обработка (cost)**
  - $C_p = p.T_p$ **,** където p е брой процесори, а T_p е време за изпълнение на една операция
- **Коефициент на използване (utilization)**
  - $U_p = \frac{O_p}{C_p}$, където O_p е брой на операциите с p процесора, а C_p е цената на обработка
- **Темп на обработка (execution rate)**
  - Представя се с няколко скали и изборът на скала зависи от типа ПА
- **Излишък (redundancy)**
  - $R_p = \frac{O_p}{O_1}$ **,** където O_p е брой на операциитес p процесора за паралелна обработка, а O_1 е брой на операциите с един процесор за последователна обработка

Коректността на даден паралелен алгоритъм е архитектурно независима, но неговата ефективност зависи от изпълнителната платформа, поради което е целесъобразно сложността му да се оценява и като функция на разпределението (**mapping**).

- **алгоритмичната сложност** - оценява времевите и пространствените характеристики на обработката
- **времева сложност** се задава като брой елементарни операции и комуникации (от който се получава времето за обработка в дадена архитектура),
- **пространствената сложност** - в брой алоцирани регистри и клетки памет.
- **процесорна сложност –** прилага се за по-обща оценка на ПА

МОДЕЛИ НА РАЗПРЕДЕЛЕНИТЕ СОФТУЕРНИ АРХИТЕКТУРИ

Софтуерната архитектура – представя/моделира програмния проект, като рапределен процес от софтуерни компоненти.

OBJECT-ORIENTED ARCHITECTURE

Обектно-ориентираният архитектурен стил е основан на разделянето на отговорностите на системата в индивидуални повтроно използваеми и независими обекти, всяка съдържаща данните и поведението, свързани с обекта. Тази архитектура вижда системата като група от сътрудничещи си обекти, вместо множество от инструкции. Ключовите принципи на този стил са:

- **Абстракция** - опростява сложната реалност чрез моделиране на класове подходящи за разрешаването на определен проблем.
- **Композиция** – обектите могат да се сглобят от други обекти и тези вътрешни обекти може да се скрият от други класове или да са достъпни чрез интерфейси
- **Наследственост** – обектите могат да наследяват други обекти и да използват техните функционалности или да им припишат ново

- **Капсулация** – обектите скриват вътрешните си детайли и могат да са достъпни само чрез
- **Полиморфизъм** – Това свойство дава възможност различни операции да имат едно и също име. Той позволява един метод на класа да реагира различно, в зависимост от използвания обект.

DATA FLOW SOFTWARE ARCHITECTURE STYLE

Този архитектурен стил вижда цялата система като последователност от трансформации на последователен набор от данни, където данните и операциите над тях са независими един от друг. Софтуерната система се декомпозира на обработваеми елементи, където данните насочват и контролират реда на изчислителния процес. Всеки компонент трансформира входните данни в съответните изходни данни.

- **Пакетна обработка**
  - Независимите програми биха обработили данните, предоставящи файл като изход
  - Този файл би станал вход за друга независима програма, която би го прочела, обработила и формирала друг изходен файл
- **Pipes and Filters** - Използва се за разделяне на големи задачи за обработква в последователност от малки, независими стъпки (филтри), които са свързани чрез канали (Pipes).

DATA-CENTERED STYLE

DATA STORE се намира в центъра на архитектурата и е достъпна често от други компоненти, които се обновяват, добавят, изтриват или модифицират по друг начин на данни в рамките на магазина. Контекстните архитектури насърчават integrability, което означава, че съществуващи компоненти могат да бъдат променени и да могат се добавят нови клиентни компоненти към архитектурата, без да се интересуваме от другите клиенти.

- **Хранилище (Repository)** - Подсистемите си комуникират единствено чрез хранилището. Сравнително независими са. Контролът върху поведението на системата може да се осъществява както от хранилището (при настъпването на някакво събитие в него), така и от подсистемите, които да освобождават ключове в хранилището и по този начин да дават зелена светлина на друга подсистема да ползва съответните ресурси.
- **Blackboard** - Тази архитектура има компонент „черна дъска", който се държи като централно хранилище, и множество други компоненти, които са зависими от общата структура данни, съхранявани в „черната дъска".

HIERARCHICAL ARCHITECTURE

Тази архитектура се характеризира с това, че вижда цялата система като йерархична структура. Системата е декомпозирана в логически модули на разлчини нива в йерархията. Модулите от различните нива са свързани чрез явни и неявни извиквания от методи. Т.е. модулите от по-ниските нива предоставят услуги на непосредствените модули от по-високо ниво, които извикват методи или процедури от ниското ниво.

- **Master-Slave** - В този архитектурен стил, „робите" предоставят услуги на своите „господари", и господарите избират определен резултат сред робите чрез различни стратегии. Робите могат да изпълнят същата задача с различни алгоритми и методи или чрез напълно различни функционалности.
- **Layered** - Тази архитектура се декомпозира на ниски и високи нива в йерархията;
- **Virtual Machine** - Виртуалната машина е изградена на базата на съществуващи системи и предоставя виртуална абстракция, множество от атрибути и операции. Обикновено разделя приложната среда от програмната.

ASYNCHRONOUS SOFTWARE ARCHITECTURE

Архитектури базирани на неявни обръщения между обслужващите процеси. Обемнът може да бъде online или offline – втория има буфер, докато първия няма. Активния процес генерира съобщения ,а пасивните ги получават и евентуално реагират.

- **Nonbuffered Event-Based** - Разделят системата на два дяла – източници на събития и слушатели на събития. Процеса на регистриран ена събития, свързва тези два дяла. Няма буфер между тях
- **Buffered Message-Based** - Разделя системата на три дяла: генератори на съобщения, консуматори на съобщения и услуга за асинхронен буфериран обмен на съобщения.

INTERACTION-ORIENTED SOFTWARE ARCHITECTURE

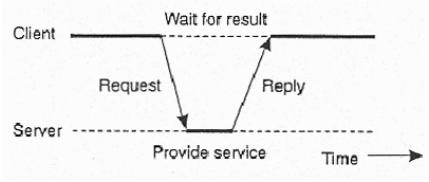
Разделя системата на три главни дяла: модул за данни, контролен модул и презентационен модул (изглед). Всеки модул си има свои отговорности. Модулът за данни предоставя абстракцията на данните и основаната бизнес логика за обработка на данните. Презентационния модул е отговорен за визуалното или звуково представяне на данни и може да използва потребителски интерфейси. Контролиращият модул установява потока на контрол, включващ избор, комуникация между модулите, изпращането на работа и инициализация на определени данни и действия за конфигуриране на действията.

ОРГАНИЗАЦИЯ НА РАЗПРЕДЕЛЕНИТЕ ПРИЛОЖЕНИЯ

КЛИЕНТ-СЪРВЪР, ДВУСЛОЙНИ АРХИТЕКТУРИ

При основния клиент-сървър модел процесите са обособени в две групи, които може да се припокриват.

- **Сървърът** е процес, който имплементира конкретна услуга (например сървър за база от данни).
- **Клиентът** е процес, който изисква някаква услуга от сървъра чрез изпращане на заявка и след това чака отговор от сървъра.



Взаимодействие между клиент и сървър

Комуникацията между клиента и сървъра може да се осъществи с прости протоколи без изграждане на връзка (connectionless protocoles – **UDP**) или чрез използване на протоколи, които установяват връзка (connection-oriented procotols – **TCP**).

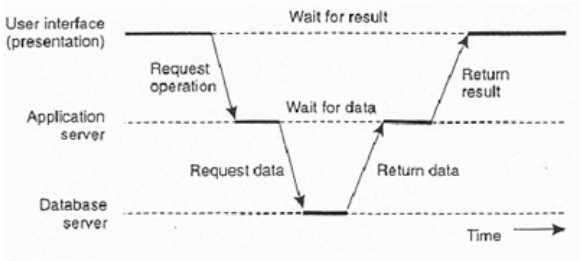
ТРИСЛОЙНА АРХИТЕКТУРА.

Голяма част от клиенти-сървър приложенията са ориентирани към предоставянето на достъп до някаква информация (база от данни). Имайки предвид това, може да разграничим три основни слоя:

- **потребителски интерфейс** – съдържа програми, които позволяват на потребителя да взаимодейства с приложението. Представя на потребителя информацията, получена от изпълнението на приложението.
- **слой с бизнес логиката** – съдържа логиката за обработка на информацията.

- **слой с данните** – съхранява информацията, с която приложението работи. Негова задача е да пази данните консистентни. Най-често съдържа база от данни.

В този случай слоят с бизнес логиката се намира на отделен сървър, който играе ролята и на клиент, когато взаимодейства със сървъра с данните.



Комуникация в трислойна архитектура

Причината за появяването на многослойните архитектури е разделянето на приложения на три слоя.

- **вертикално разпределение** - разпределената обработка означава организирането на клиент-сървър приложението в многослойна архитектура
- **хоризонтално разпределение** - клиентът или сървърът е физически разпределен, като всяка отделна логическа част оперира върху част от данни

КОМУНИКАЦИОННИ МОДЕЛИ В КЛИЕНТ-СЪРВЪР АРХИТЕКТУРИТЕ

RPC (**Remote Procedure Call**) – Отдалечено извикване на процедури. Идеята е извикването на отдалечена процедура да изглежда като извикване на локална процедура (да се постигне прозрачност).

RMI (**Remote Method Invocation**) – Този модел е развитие на RPC чрез използване на принципите на обектно-ориентираното програмиране. В този случай се извикват методи на разпределени обекти. Свързването с тях става като в адресното пространство на клиента се зарежда интерфейса на обекта (proxy – изпълнява ролята на клиентския стъб).

MOM (**Message-Oriented Middleware**) – Комуникация, базирана на обмяната на съобщения. Компютрите са свързани посредством комуникационна инфраструктура (комуникационни сървъри). Всеки клиент предоставя интерфейс към комуникационната система за изпращане на съобщения.

P2P

Peer to peer, представлява разпределена мрежова архитектура, съставена от участници, които отдават част от своите ресурси (като процесорна сила, дисково пространство, ширина на мрежовата връзка) директно на разположение на другите мрежови участници без необходимост от централна мрежова инстанция. Участниците са едновременно "снабдители" и "потребители" на ресурси в контраст на традиционния клиентско-сървърен модел, където само сървърите дават (снабдяват), а клиентите (clients) консумират.

СЪРВЪРИ ЗА ПРИЛОЖЕНИЯ И WEB-СЪРВЪРИ

Сървърът за приложения е сървър, който е специализиран в изпълнението на определени приложения. Сървърът за приложения представлява съвкупност от компоненти, които могат да бъдат достъпени чрез платформено независими интерфейси за програмиране (**API** – Application programming interface).

Web-сървърите са специализиран вариант на сървърите за приложения.

- Съдържанието, което се доставя, е основно HTML текст, но може да съдържа изображения, мултимедия, документи или други видове файлове, дефинирани съгласно MIME.
- Освен статичното съдържание, предоставят и интерфейс към сървърни web-приложения чрез някой от стандартните езици/интерфейси.
- Повечето web-сървъри предоставят и допълнителни системни функции като контрол на трафика, компресия/декомпресия на съдържанието, виртуален хостинг.

МЕТАСИСТЕМИ И ГРИД СИСТЕМИ

Разпределените системи, които се използват за извършване на сложни изчисления могат да бъдат разделени в две категории – **клъстерни системи** и **грид системи**.

- **Клъстерните системи** - състоят се от множество подобни компютри, които са свързани помежду си чрез високоскоростна локална мрежа.
- **Грид системи** - изградени са от различни компютри, с различни операционни системи, намират се в различни административни области, имат различни политики за сигурност.

СЕРВИЗНО-ОРИЕНТИРАНИ АРХИТЕКТУРИ

Сервизно-ориентираната архитектура (SOA) представлява съвкупност от принципи за изграждането и поддръжката на бизнес процеси в големи разпределени системи.

Базира се на три ключови концепции:

- **Услуги** - представляват софтуерни компоненти, които имат определен функционален смисъл (често енкапсулират бизнес концепция от високо ниво).
- **Сервизна шина** - инфраструктура, която позволява съвместната работа между разпределени системи. Улеснява разпределянето на бизнес процесите върху много системи, които използват различни платформи и технологии.
- **Слабото свързване** - концепция за намаляване на зависимостите в системата. Тъй като процесите са разпределени върху множество компютри, е важно да се намали ефектът от промени и грешки.

МОДЕЛНО-ОРИЕНТИРАНИ АРХИТЕКТУРИ

Моделно-ориентираната архитектура е софтуерна архитектура за автоматизирано проектиране на приложения.

- базира на принципите на моделното проектиране (**Model-driven engineering**) – изграждането на абстрактни модели, които да описват системата от гледна точка на конкретната област.
- цели на тази архитектура - разделянето на дизайна от конкретната имплементация.

АСПЕКТНО-ОРИЕНТИРАНИ АРХИТЕКТУРИ

Аспектно-ориентираната архитектура е софтуерна архитектура, която се базира на принципите на аспектно-ориентираното програмиране. Основаната цел е да се подобри структурата на приложението и да се улесни добавянето на нови функционалности. В общия случай разделяме отделните изисквания в отделни модули. В процеса на това разделяне се появяват много съображения, които не могат да бъдат адекватно обособени в отделни компоненти – нар. се **аспекти**. Аспектите представляват общи функционалности, които се прилагат в няколко модула.

СОФТУЕРНИ АГЕНТИ

Софтуерните агенти са автономни процеси, които могат да изпълняват някакви задания заедно с други агенти (които може да са отдалечени). Основните свойства на софтуерните агенти са:

- **автономност** – могат да извършват някакви задачи и да взимат решения без човешка намеса;
- **комуникативност** – могат да взаимодействат и да обменят информация с други агенти или с потребителя;
- **реактивност** – възприемат средата, която се намират и могат да отговарят на промените в нея;
- **проактивност** – могат да променят параметрите на средата, в която се намират;
- **мобилност** – миграция между различни възли. Това свойство не е характерно за всички софтуери агенти;
- **адаптивност** – възможност за приспособяване към средата (еволюция на реакциите при еднакви параметри на средата). Това свойство не е характерно за всички софтуерни агенти;
- **непреходност** – многократно изпълнение на едни и същи функции. Това свойство не е характерно за всички софтуерни агенти.

СИСТЕМИ ЗА РАЗПРЕДЕЛЕНА ОБРАБОТКА

УПРАВЛЕНИЕ НА ПРОЦЕСИ И НИШКИ

Управление на процеса

- За изпълнение на даден процес операционната система създава отделен виртуален процесор, който да изпълнява процеса.
- Операционната система се грижи процесите да нямат достъп до информацията за другите процеси и да не могат да променят тяхното поведение.
- Конкурентното изпълнение на процесите се постига прозрачно, но с цената на голямо системно свръхтоварване.

Нишките приличат на процесите по това, че изпълняват програма (или част от нея) на виртуален процесор.

- При тях няма такова високо ниво на конкурентност, тъй като това води до влошаване на производителността.
- За нишките се пази само минималното количество информация, което позволява споделянето на централния процесор от няколко нишки
- Друга важна особеност е, че при нишките няма защита на информацията както при процесите (една нишка може да достъпи данните на другите).
- Многонишковото програмиране се използва и при унипроцесорни приложения.

МИГРАЦИЯ НА КОД

В разпределените системи понякога е подходящо между отделните възли да се предават цели програми. Традиционно **миграцията на код** в разпределените системи е под формата на миграция на процеси. Основната причина е повишаване на цялостната производителност на системата – балансиране на натоварването на различните машини .

СИНХРОНИЗАЦИЯ И СИСТЕМНО ВРЕМЕ

В разпределените системи програмните компоненти са разположени на множество машини, които имат различни системни времена. Причината за разликата в системните времена е **десинхронизацията** (clock skew), породена от разликата в тактовата честота на осцилаторите. **Синхронизацията** на системното време може да бъде **централизирана** (с времеви сървър) и **разпределена** (схема от тип p2p).

- **Централизирана синхронизация** – алгоритъм на Christian и алгоритъм на Berkley.
  - алгоритъм на Christian - има един централизиран сървър. Всички компютри периодично изпращат запитвания за системното време до него.
  - алгоритъм на Berkley – при него времевият сървър е активен – периодично изпраща запитвания за локалното време на компютрите. След това той осреднява времето и го изпраща до другите компютри.
- **Разпределена синхронизация** - времето се разделя на интервали с фиксирана дължина.

ПРОТОКОЛИ ЗА ПОДРЕЖДАНЕ

В много ситуации е достатъчно всички компютри да съгласуват системното време. В този случай е подходящо да говорим за **логически часовници**. Всъщност важно е не процесите да съгласуват времето, а да съгласуват реда на настъпване на събитията.

Алгоритъм на Lamport - дава решение на проблема за задаване на време на настъпване на събитията, както и за синхронизацията на часовниците на комуникиращите си процеси. На всяко съобщение се задава времева марка на локалното логическо време на изпращащия процес. Ако получаващият процес има по-малка стойност на локалното логическо време от марката на изпратеното съобщение, той коригира своя логически часовник като към стойността на времевата марка добавя 1. Например, ако едно съобщение напусне даден възел във време 60 то може да бъде получено най-рано във време 61. Още една особеност е, че не може да има две събития с едно и също време.

РАЗПРЕДЕЛЕНИ ТРАНЗАКЦИИ

Транзакциите представляват механизъм за синхронизация на съвместната работа на процесите в системата. Тяхната роля е да предпазят даден споделен ресурс (най-често данни) от едновременен достъп от множество конкурентни процеси. Те позволяват даден процес да изпълни няколко операции като единична атомарна операция.

Транзакциите имат четири основни свойства (ACID):

- **атомарност** (**atomic**) – за външния свят транзакциите представляват неделима единица;
- **консистентност** (**consistent**) – транзакцията не нарушава състоянието на системата;
- **изолираност** (**isolated**) – конкурентните транзакции не взаимодействат помежду си;
- **устойчивост** (**durable**) – данните остават трайно променени след потвърждаване на транзакцията.

Има няколко типа транзакции: **блокови**, **вложени** и **разпределени**.

- **Блокови транзакции** – това са транзакции , които притежават четирите свойства (ACID).
- **Вложените транзакции** - състоят се от няколко подтранзакции. Главната транзакция може да стартира няколко транзакции от следващото ниво, които да се изпълняват паралелно на различни машини. Всяка една от тези транзакции може да изпълнява друга транзакция или да стартира подтранзакции.
- **Разпределените транзакции** - наподобяват блокови транзакции, които обработват разпределени данни от няколко машини. Характерно за тях е, че не може да се направи логическо разделяне. Основен проблем на тези транзакции е че трябва да се използват различни разпределени алгоритми за заключване на данните и потвърждаване на промените.