MŰSZAKI SZEMLE
33. szám, 2006.

Magyar nyelvű szakelőadások
a 2003–2004. és a 2004–2005. tanévekben

Kiadja / Editor
Erdélyi Magyar Műszaki
Tudományos Társaság – EMT
Societatea Maghiară Tehnico-Ştiinţifică
din Transilvania
Ungarische Technisch-Wissenschaftliche
Gesellschaft in Siebenbürgen
Hungarian Technical Scientific Society
of Transylvania

Felelős kiadó / Managing Editor
Dr. Köllő Gábor

A szerkesztőség címe / Address
Romania
400604 Cluj, Kolozsvár
B-dul 21. Decembrie 1989., nr. 116.
Tel/fax: 40-264-590825, 594042
Levélcím: RO – 400750 Cluj, C.P. 1-140.

Nyomda / Printing
Incitato Kft.

ISSN 1454-0746

CNCSIS által elismert folyóirat
Revistă acreditată de CNCSIS

www.emt.ro

emt@emt.ro

Tartalomjegyzék – Cuprins – Content

A kiadvány megjelenését támogatta

Illyés Közalapítvány – Budapest

Oktatási és Kutatási Minisztérium – Bukarest

Communitas Alapítvány – Kolozsvár

Kritériumrendszer programozási nyelvek összehasonlító elemzésére

Criteria-system for Comparative Analyses of the Programming Languages

KOVÁCS Lehel István

Babeş-Bolyai Tudományegyetem, Kolozsvár
Számítógépes Rendszerek Tanszék

Abstract

60 years ago, in 1946, John von Neumann formulated the principles of large-scale computing machines. The machine-code was the only modality of programming the first generation of electronic computers constructed between 1946 and 1955. Between 1954 and 1958 appear the first high-level programming languages (FORTRAN, ALGOL). Using these programming languages the communication between man and computers evolves to the human side. The programs can be written in user-friendly mode, and translated to machine-code with compilers. We can speak about a very quick and spectacular evolution of programming languages the number of these exceeds 1000.

The aim of this article is to formulate a criteria-system for the classification and comparative analyses of the programming languages. This criteria-system can be a good methodological base for teaching the programming languages and searching the answer for the following question: which programming language provides the ideal elements and tools for solving the given problem.

Összefoglaló

60 évvel ezelőtt, 1946-ban Neumann János kidolgozta a korszerű számítógépek megépítésének alapelveit. 1946–1955 között megépültek az első generációs elektronikus számítógépek. Ezeket a kezdetleges számítógépeket a gépi kód közvetlen felhasználásával lehetett programozni. 1954–1958 között megjelentek a magas szintű programozási nyelvek (FORTRAN, ALGOL), amelyek segítségével emberközelibb formában lehet a programokat megírni, és azokat gépi kódra lefordítani. Ezt követően a programozási nyelvek gyors fejlődésnek indultak. Napjainkban több ezer programozási nyelvről beszélhetünk, és számuk növekszik.

Jelen cikkben a programozási nyelvek osztályozásához és összehasonlító elemzéséhez próbálunk felállítani egy kritériumrendszert, módszertani alapként az oktatáshoz és kutatáshoz, azon kérdés megválaszolására, hogy egy adott feladat megoldásához melyik programozási nyelv biztosít ideális eszközöket, elemeket.

Kulcsszavak: nyelvleírások, nyelvosztályok, történet, kapcsolatok, fordítóprogram, lexikális elemek, grammatikák, változók, konstansok, kifejezések, típusok, utasítások, kivételkezelés, programegységek, modulok, absztrakciós szintek

2000 Mathematics Subject Classification: **68N20**
1998 CR Categories and Descriptors: **D.3. [PROGRAMMING LANGUAGES]**

Bevezetés

A programozási nyelv egy jelölésmód, amelynek segítségével számítási folyamatokat írhatunk le, kódolni tudjuk az algoritmusokat ahhoz, hogy a számítógép megértse és végre tudja hajtani a megoldáshoz szükséges lépéseket. A programozási nyelv olyan számítógép által értelmezhető utasítások sorozata, mely adott feladat elvégzésének módját közli a számítógéppel. A számítógép által történő értelmezés nem direkt úton történik, hisz a számítógép csak a gépi kódot tudja értelmezni. Ez az indirekt út a *fordítóprogramok* jelenlétét teszi szükségesé, amelyek képesek az egyes programozási nyelvekben megfogalmazott feladatokat gépi kódra lefordítani.

A programozási nyelvek gyors és dinamikus fejlődést jártak be rövid – mintegy 60 éves – történelmük során, melyre az állandó megoldáskeresés, a feladat megfogalmazásának vagy megoldási lépéseinek lehető legjobb (egyszerűbb, egyértelműbb, optimális) kódolási módjának megtalálása volt jellemző.

Nem csoda, hogy e gyors fejlődés a programozási nyelvek hatalmas számához vezetett, számos speciális és általános nyelv alakult ki. A múlt századvég és az új századelő feladata ezeket a nyelveket osztályozni, kategorizálni, összehasonlító elemzések és szintézisek segítségével javaslatokat tenni, hogy a különböző programozási feladatok megoldására ki lehessen választani az ideális programozási nyelvet. Mindezek ellenére kevesen tették meg ezt a lépést, kevés összehasonlító tanulmány jelent meg, ezek inkább speciális esetekre, részletekre koncentrálnak, mintsem általánosan használható kritériumrendszert állítanának fel.

Kiindulva néhány alapmunkából: [1.], [2.], [3.], [4.], [5.], [6.], [7.], [8.], [9.], [10.], [11.], [12.], ezen cikk célja egy olyan kritériumrendszer felállítása, amely az osztályozási szempontok felállításán túl keretet biztosít az összehasonlító elemzések elkészítéséhez szükséges lépések, módszerek összefoglalására.

I. Programozási nyelvek osztályozása

Programozási nyelveket több szempont szerint is osztályozhatunk, különféle metszeteket készíthetünk, különböző nyelvosztályokat állíthatunk fel, de az egyes jellemzők közé éles határ nem húzható. *Hibrid nyelvekről* akkor beszélünk, ha az adott nyelv egy osztályozási szempont szerint több osztályba tartozik. Napjaink programozási nyelveinek többsége hibrid.

1.1. Amatőr és professzionális nyelvek

Az *amatőr programozási nyelvekre* az interaktivitás, a sok nyelvi elem, a gyors nyelvi fejlődés jellemző. Ezekben a nyelvekben a programok szerkezete egyszerű, és ezek speciális gépi tulajdonságokra épülnek rá (pl. *BASIC*, *Pascal*).

A *professzionális nyelvekre* a modularitás, a magasfokú stabilitás és a kevés nyelvi elem a jellemző. Ezen nyelvek igen hatékonyak, sok lehetőséggel bírnak és gépfüggetlen kódot generálnak, vagy a kód átvihető más architektúrájú gépekre is (pl. *C*, *Java*).

1.2. Emberközeliség

A *gépi nyelvek* használatával minden hardverlehetőség kihasználható, azonban a memóriacímeket, a memória-kiosztást és a programkódot önerőből kell megvalósítani (a memóriában lévő utasításkódokat közvetlenül a programozó adja meg). A megírt programok gépközeliek, közvetlenül a processzor utasításkészletére épülnek (pl. *Gépi kód*, *Assembly*).

Az *alacsony szintű nyelvek* géporientált nyelvek ugyan, de megjelennek a szimbolikus utasítások, azonosítók és címkenevek, megjelenik a feltételes vezérlésátadás fogalma, az eljárások és a visszatérések. Az adatokat deklarálni, definiálni lehet és a tárhely is ennek függvényében foglalódik le. Megjelennek a makrók és a direktívák. A közvetlen kódok helyett rövid, könnyen megjegyezhető szavakat alkalmazunk (mnemonikok) a könnyebb megjegyzés, a jobb átláthatóság kedvéért. Jobban áttekinthetők a címzési módok, a programokba megjegyzéseket szúrhatunk be, külön fordítható egységekkel dolgozhatunk (pl. *C*, *Assembly – Makró*).

A *magas szintű nyelvek* már feladatorientáltak, megjelenik a típus és a változó fogalma, kifejezések kiértékelésével komoly számításokat lehet elvégezni egyszerűen, megjelennek a ciklusok, elágazások. A nyelvek eljárásokat, függvényeket tudnak használni és komoly paraméterátadó mechanizmusokkal vannak felruházva (pl. *Pascal*, *Java*).

A *metanyelvekre* azért van szükségünk, hogy segítségükkel más nyelveket tudjunk leírni, ezáltal kizárható a különböző deklarációkban meghúzódó többértékűség (pl. *EBNF*, *BNF*).

1.3. Típusok használata

A *nem típusos nyelvek* esetében ha létezik is a változó fogalma, ez nincs kötve semmiféle típushoz (pl. *BASIC*, *JavaScript*).

A *típusos nyelveknél* megjelenik a típus fogalma, amely meghatározza, hogy a változó milyen értékeket vehet fel, mekkora memóriatartományra van szüksége, milyen műveletek végezhetők el vele stb. (pl. *C*).

A *szigorúan típusos nyelvek* esetében szigorú szabályok írják elő a típusok közötti átalakításokat, konverziókat (pl. *Pascal*).

1.4. Alapelvek szerint

Az *imperatív nyelvek* (I) osztályába a Neumann architektúrához szorosan kötődő algoritmikus nyelvek tartoznak, amelyeknél fő programozási egység az utasítás, és ezek egymásutánisága vezérli a processzort. A tár bizonyos területén lévő értékeket módosíthatjuk, így változókról beszélhetünk. A programozó mondja meg, hogy mit és hogyan kell csinálni (pl. *C*, *Pascal*).

Az *alkalmazás háttérnyelvek és makrónyelvek* (A) nagyobb alkalmazások hátterében feladatokat valósítanak meg, segítségükkel testre szabható, könnyebben, gyorsabban vezérelhetők lesznek az alkalmazások (pl. *Visual Basic for Application*, *AutoLisp*).

A *procedurális, strukturált nyelvek* (S) egy adott probléma megoldásának algoritmusát írják le. (pl. *BASIC*, *Algol*).

A *deklaratív nyelvek* (D) osztályába azok a matematikai logikára, vagy függvényhasználatra épülő, nem algoritmikus nyelvek tartoznak, amelyeknél a programozó csak a megoldandó feladatot írja le, a megoldást magát a rendszer végzi el. Ezeknél a nyelveknél nem létezik utasításfogalom, a tárhely értékeit nem lehet módosítani, nem léteznek adatok, vagy ezeknek teljesen más a szerepük (pl. *Prolog*, *Lisp*, *Miranda*, *Haskell*, *SML* – funkcionális, logikai nyelvek).

Az *applikatív nyelvek* (App) függvények változókra történő alkalmazásaival operálnak. Nincs mellékhatás (pl. *ML*)

A *funkcionális nyelvek* (F) magas szintű függvények használatára és operátor definíciókra épülnek. Az operátorok függvényeket manipulálnak, mintha azok egyszerű adatok lennének. (*Clean*, *Haskell*, *Miranda*, *SML*, *Hope*, *FP*).

A *definíciós nyelvek* (Def) olyan applikatív nyelvek, amelyeknél a megfeleltetések (értékadások) definíciókként vannak értelmezve (pl. *DAML*).

Az *egyszeres megfeleltetésű nyelvek* (SSL) esetén egy változó a láthatósági területén csak egyszer fordulhat elő a bal oldalon, egyszer vehet fel értéket (pl. *DAML*).

Az *adatfolyam (dataflow) nyelvek* (DF) az adatfolyam architektúrák programozási nyelvei (pl. *FOOD*, *TDFL*).

A *logikai nyelvek* (L) predikátumokra és relációkra épülnek. Tényekből szabályok segítségével következtetéseket tudnak levonni általában rezolúció-kalkulust használva (pl. *Prolog*).

A *megkötésorientált nyelvek* (C) a megoldandó feladatot megkötések sorozataként fogalmazzák meg és oldják meg (pl. *OCL*, *DAML+OIL*, *Eclipse*, *GUIdeLA*).

A *konkurrens nyelvek* (P) segítségével a párhuzamos, konkurrens, osztott, többszálas programokat tudjuk megfogalmazni (pl. *NESL*, *Orca*, *mpC*, *BLAZE*, *PARLANSE*).

A *folyamatszálas vagy veremalapú nyelvek* (T) alacsony szintű, konkrét feladatok megoldására szakosodott programozási nyelvek, melyek segítségével hatékonyan programozható a verem, vagy különböző grid-alapú osztott rendszerek (pl. *Vcode*, *Zero*).

A *matematikai vagy szimulációs nyelvek* (M) konkrét feladatosztályok megoldására szakosodtak. Többnyire matematikai számítások, szimulációs folyamatok végezhetők el velük igen nagy hatékonysággal. Kísérletek, folyamatok eredményeinek előrejelzését segítő eszközök (pl. *Mathematica*, *Matlab*, *Mathcad*).

A *szimbólum-feldolgozó nyelvek* (Sim) nagy mennyiségű szöveges információk, hosszú listaadatok értékelésére, elemzésére kidolgozott célnyelvek. Felhasználási területük az információkutatás, a matematikai kutatások. (pl. *Lisp*, *SPSS*)

A *formulakezelő nyelvek* (For) nagy pontosságot igénylő, bonyolult matematikai, műszaki számítások számítógép által történő elvégzéséhez biztosítanak hatékony segítséget, eszközöket (pl. *LotusScript*).

Az *objektumorientált nyelvek* (O) esetén magas absztrakciós szinten lévő osztályok és ezekből példányosított objektumok segítségével fogalmazhatjuk meg és oldhatjuk meg a feladatot. Az osztályok zárt egységnek tekintik az adatokat és az őket kezelő eljárásokat. Az objektumok egymással kommunikálnak (pl. *Java*, *Delphi*, *Sather*, *Modula*, *Smalltalk*, *Eiffel*, *Oberon*).

A *vizuális nyelvek* (V) esetén grafikus eszközök sokasága vehető igénybe az algoritmus leírására, megadására (pl. *ProGraph*).

A *negyedik generációs nyelvek* (4GL) nagyon magasszintű nyelvek, melyek segítségével a megoldandó feladat természetes nyelven, vagy diagrammok használatával fogalmazható meg. A fordítóprogram választja ki a megfelelő adatszerkezeteket vagy algoritmusokat (pl. *UML*, *RAD*).

A *lekérdező nyelvek* (QL) az adatbázis-programozás fő kommunikációs eszközei, interfészei (pl. *SQL*).

Az *adatbázis és szövegfeldolgozó nyelvek* (DB) segítségével adatokat, információkat strukturálhatunk, dolgozhatunk fel, módosíthatjuk, karbantarthatjuk az adatbázisban tárolt adatokat (pl. *SQL*, *ODL*, *TeX*, *HTML*).

A *specifikáló (leíró) nyelvek* (Spec) a szoftver vagy hardver tervezésének formális leírását szolgálják (pl. *VHDL*).

Az *assembly nyelvek* (Asm) a gépi kód szimbolikus jelölésére szolgálnak egy adott számítógép architektúrán (pl. *TASM*, *MASM*).

A *rendszer- és fordítóprogramok írására specializálódott nyelvek* (Sys) hatékonyan támogatják az alacsony szintű programozást és az operatív tár közvetlen kezelését, lehetővé teszik a bitenkénti műveletek végzését, de mégis rendelkeznek a magas szintű nyelvek előnyeivel (pl. *Gnu C*).

A *köztes nyelveket* (Int) a fordítóprogramok használják mint ábrázolás rendszert. Lehetnek szöveges vagy bináris formátumúak (pl. *Java ByteCode*, *OBJ*).

A *parancssor vagy szkriptnyelvek* (Com) segítségével operációs rendszer közeli feladatok írhatók le (pl. *BAT*, *Shell*).

A *kiterjeszthető nyelvek* (Ex) esetében a programozási feladat megoldásához a nyelv csak egy minimális alapot definiál. A hiányzó eszközöket a programozó maga állíthatja elő a már létező elemek felhasználásával, kombinálásával (pl. *XML*).

A *metanyelvek* (ML) más nyelvek deklarálására szolgálnak (pl. *BNF*, *EBNF*).

Az *egyéb, vagy más alapelvekre épülő nyelvek* (nemkonvencionális nyelvek – NCL) képezik az utolsó nagy nyelvosztályt. Ilyenek a különböző párhuzamos, adatfolyam, szisztolikus működést leíró nyelvek, vagy minden olyan nyelv, amelyet egy bizonyos speciális probléma megoldására tervezvtek.

1.5. Generációk szerint

Az elektronikus számítógépek nagy generációi tulajdonképpen meghatározták a programozási nyelvek generációit is. Ilyen értelemben beszélhetünk *első (1GL), második (2GL), harmadik (3GL), negyedik (4GL)* és *ötödik (5GL) generációs programozási nyelvekről*.

Az *első generációs nyelvek*et (kb. 1946–1955) a teljes mértékű processzorfüggőség jellemezte. Az utasítások bitsorozatok voltak, amelyeket a gép előlapján lévő kapcsolókkal lehetett megadni. Az első programozási nyelv a gépi kód volt. Ennek a nyelvnek az utasításait a számítógép képes volt közvetlenül, minden átalakítás nélkül végrehajtani. A nyelv erősen gépfüggő volt, hiszen minden gépen más és más utasításokat használt, az adott problémát minden géptípus esetén másképpen kellett leírni. A gépi kód nagy előnye a gyorsaság és az egységesség. A generáción belül komoly előrelépést jelentett az *Assembly* nyelvek megjelenése. A gépi kódú utasításokhoz egy-egy *mnemonikus* kódot rendeltek hozzá, a tárcímeket pedig a memória kezdetéhez viszonyított relatív címekkel számították. Az egyes memóriacímeket egy-egy szimbolikus névvel lehetett helyettesíteni.

A *második generációs nyelvek* megjelenése (kb. 1955–1963) tulajdonképpen a számítógépek alkalmazási területe kibővülésének eredménye. Felmerült az igény, hogy a programokat minél gyorsabban és hibamentesebben írják meg a programozók. A gépi kód és az assembly nehézkessége, géphez igazodása miatt nem volt erre alkalmas. A 60-as évek elején jelentek meg az első magas szintű programozási nyelvek, melyek már nem a számítógép sajátosságaihoz, hanem a problémához igazodtak. Egyetlen magas szintű programnyelvi utasítás több gépi kódú utasítást jelent. Ekkor jelennek meg a fordítóprogramok (*compiler*) és a szerkesztők (*linker*). Az első magas szintű programozási nyelv a *FORTRAN* volt.

A *harmadik generációs nyelvek* (kb. 1963–1973) már magas szintű programozási nyelvek, olyan nyelvek, amelyek nem egy konkrét feladatosztály megoldására specializálódtak, mint a második generációs nyelvek, hanem általánosak, univerzálisak voltak. Az új korszakot a procedurális programozási szemlélet és az adatok struktúrájának hangsúlyozása jellemezte. Ekkor jelennek meg az objektumorientált nyelvek is. A harmadik generációs nyelvek egyaránt alkalmasak az adatfeldolgozási és a számítási problémák megoldására, rendszerprogramozásra, emellett a program szerkezete egyszerű és követhető, formai előírásai nem túl szigorúak. A program tartalmaz egy főprogram részt, amely külön blokkokban több alprogramot foglalhat magába, az alprogramok egymásba ágyazhatók. Strukturált programozásról beszélhetünk (pl. *Algol*, *C*, *Pascal*, *BASIC*).

A *negyedik generációs nyelvek* (kb. 1973–) napjaink programozási eszközei. Bonyolult lekérdező nyelvek, programkód generátorok, interaktív fejlesztői környezetek, melyek már túl vannak a magas szintű nyelvek osztályán. A bonyolult információs rendszerek fejlesztése, központilag vezérelt számítógép-hálózatok programozása, döntéshozást támogató rendszerek megvalósítása túlmutatott a hagyományos programozási nyelvek által biztosított eszközök lehetőségein. Gyors alkalmazásfejlesztésre, vizuális paradigmára támaszkodó objektum- és komponensorientált kódgenerátorokra, magas hardverfüggetlenséget támogató nyelvekre van szükség. Sajnos, a program hatékonyságának növekedésével egyenes arányban nő a program hardver – elsősorban memória, tárhely – igénye is, a lefordított program mérete is (pl. *UML*, *Delphi*, *RAD*).

Az *ötödik generációs nyelvek* (1981–) alternatív irányvonalat képviselnek, valójában két fogalmat takarnak: egy gép közelebbi, de magas szintű nyelvet, amely tulajdonképpen a számítógép operációs rendszerét jelenti, és egy természetes nyelvet, amely során az ember és gép közötti kommunikáció („programírás") megvalósul (pl. *Prolog*, *KL1*, *természetes nyelvek*).

1.6. Számítási modellek szerint

Azon absztrakt modelleket követve, amelyeknek alapján az algoritmusokat végre kell hajtani, a feladatot meg kell oldani, a következő nagy paradigmákat különböztethetjük meg:

- *Az imperatív paradigma (IP):*
  - egyszerű operációs paradigma (SOP)
  - a Neumann-féle paradigma (NP)
  - az automata feldolgozás paradigmája (AP)
  - az adatbázis-kezelés paradigmája (DBP)
- *A deklaratív paradigma (DP)*
  - a funkcionális paradigma (FP)
  - a logikai paradigma (LP)
- *A párhuzamos és osztott paradigma (PDP)*
- *Az objektumorientált paradigma (OOP)*
- *A vizuális paradigma (VP)*
- *Az ötödik generációs paradigma (5GP)*
- *Alternatív paradigmák (AP)*

II. Imperatív programozási nyelvek elemzési szempontjai

Ha elemezni szeretnénk egy programozási nyelvet, vagy összehasonlító elemzéseket szeretnénk végezni, célszerű az alábbi kritériumrendszer szerint körbejárni a témát.

2.1. Nyelvleírások, könyvészet

Az elemzés első lépésében általánosságában szeretnénk megismerkedni a programozási nyelvvel:

- A nyelv céljai és specifikációja
- A nyelv rövid jellemzése
- Milyen feladatok megoldására specializálódott?
- Mennyire elterjedt a nyelv?
- Honnan lehet hozzáférni?
- Honlapok
- Létező fordítóprogramok
- Milyen dialektusokkal rendelkezik a nyelv?
- Jellemző példaprogram

2.2. Milyen nyelvosztályokba sorolható a nyelv?

Az előző fejezet alapján felállítjuk azokat a fő nyelvosztályokat, alosztályokat, amelyekbe besorolható a nyelv.

- Nyelvosztályok és alosztályok
- Hasonló nyelvek
- Paradigmák
- Hibrid nyelv-e vagy sem?

2.3. Története

Egy programozási nyelv története, létrehozásának körülményei sokat elárulhat a nyelv céljáról, specifikációjáról, fontosabb verziószámai pedig a fejlődéséről, korszerűsítéséről. Például igen érdekes az *Ada* nyelv története: 1975-ben az Amerikai Védelmi Minisztérium finanszírozásával megindult egy olyan komplex programozási nyelv elméletének kidolgozása, amely a kor legújabb kihívásait megoldotta. Az új kívánalmaknak megfelelő nyelv vázlatát STRAWMAN-nak (szalmabáb) nevezték el. Ezt felülvizsgálva az új változat a WOODENMAN (fabáb) nevet kapta. További vizsgálatok eredménye lett a TINMAN (ónbáb), majd az IRONMAN (vasbáb) jelentések. Ekkor versenyfelhívást tettek közzé, hogy ki tud egy olya nyelvet tervezni, ami a legközelebb áll az IRONMAN-ben szereplő leíráshoz. A négy induló közül a győztes a GREEN (zöld) csapat lett, amely a francia Cii-Honeywell Bull csoportja volt, amit Jean Ichbiah vezetett. A legújabb követelményeket STEELMAN-nak (acélbáb) nevezték el, és az ebből származó nyelvet *Ada* névre keresztelték Ada Augusta Byron (1815–1852) „az első programozó” tiszteletére. Az *Ada* potenciálisan a legfejlettebb nyelv lett a 80-as évek közepére, de szerepe ma messze nem akkora, mint várták volna.

- Kik tervezték?
- Mikor tervezték?
- Mi volt a terv neve?
- Miről kapta nevét a nyelv?
- Fontosabb verziószámok, bővítések
- Utolsó módosítás ideje
- A nyelv ősei
- Érdekességek a nyelvvel kapcsolatosan

2.4. Kapcsolat az operációs rendszerrel és a számítógéppel

A számítógép architektúrájával, az operációs rendszerrel fennálló kapcsolatokat vizsgáljuk:

- Architektúrafüggő-e a nyelv?
- Operációs rendszer függő-e a nyelv?
- Platformfüggetlen vagy átvihető, hordozható?
- Létezik-e köztes kód? Mi a szerkezete? Van-e virtuális gép?
- Milyen futtatható állományokat tud generálni (EXE, COM stb.)?
- Hogy veszi át a paramétereket a parancssorból?

Például a *Pascal* és az *Ada* csak függvények segítségével *(ParamCount, ParamStr*, illetve *Ada.Command_Line.Argument_Count, Ada.Command_Line.Argument)* tud operálni a parancssoron, a *C* és a *Java* a *main* függvény paraméterei által.

A *Pascal* átvihető nyelv például DOS és Linux között, a forráskódot kisebb-nagyobb módosításokkal le lehet fordítani (a Linux nem ismeri a CRT egységet), a *C* nyelv hordozható, a forráskód módosítás nélkül, vagy minimális módosítással lefordítható, a *Java* nyelv platformfüggetlen, a tárgykód lefut a különböző operációs rendszerek alatt a különböző architektúrákon.

2.5. A fordítóprogram

A hatékony tárgykódot, az interaktív, sőt feltételes fordítást a nyelv fordítóprogramja biztosítja. A fordítóprogram elemzésével a következő kérdésekre keresünk válaszokat:

- Parancssoros fordítóprogrammal rendelkezik-e?
- Milyen paraméterekkel kell meghívni?
- Milyen direktívákkal rendelkezik?
- Van-e pre- vagy posztprocesszálás (előfeldolgozás, utófeldolgozás)?
- Hány menetes fordítóprogramról beszélhetünk?
- Van-e külön szerkesztő (linker)?
- Optimalizál-e a fordítóprogram (Ha igen, akkor milyen elvek alapján)?
- Rendelkezik-e környezettel?
- Milyen tulajdonságokkal van a környezet felruházva?
    - Szövegszerkesztője
    - Fordítórendszere
    - Szerkesztőrendszere (linker)
    - Futtatórendszere

    - Súgó, kódkiegészítők, sablonok
    - Varázslók, kódgenerátorok
    - Tervezőfelület
    - Debugger, nyomkövető
    - Szimbólumkövető
    - Adatbázis-tervező
    - Támogatja-e a csoportprogramozást?
    - Más környezeti eszközök, beágyazott lehetőségek
- Fordítóprogram, értelmező, átalakító
- Hogyan kezeli a hibákat a fordítóprogram?
- Létezik-e formális helyességbizonyító?
- Milyen önellenőrző mechanizmusokkal rendelkezik?
- Mennyire gyors a fordító?

Például a *FoxPro* vagy *Logo* értelmezővel (interpreter) rendelkezik, amely értelmezi és végrehajtja a beírt utasításokat, programokat. A *Pascal* fordítóprogrammal (compiler) rendelkezik, a programokat elemzés után futtatható exe állománnyá fordítja, majd azt lehet futtatni. A *Java* átalakítóval (transzlátor) rendelkezik, a forráskódból köztes kódot állít elő, majd ezt a köztes kódot értelmezi a *Java Virtuális Gép* az adott architektúrán, operációs rendszeren.

Az *Assembly* parancssoros fordítóval rendelkezik, a *Pascal* parancssoros fordítóval, a 6.0-ás verziótól TurboVision környezettel, a *Delphi* fejlett környezettel rendelkezik. A *Java* fordítóhoz több környezet is létezik. Ezeket a környezeteket *IDE*-nek *(Integrated Development Environment)*, *beágyazott fejlesztési környezetek*nek nevezzük.

2.6. Lexikális elemek

A lexikális elemek összessége (kulcsszavak, azonosítók, konstansok, műveletek, speciális szimbólumok stb.) azt az eszköztárat képezi, amellyel a programozó direkt operál a programozás során. A következő kérdéseket kell megválaszolnunk:

- Milyen karakterek használhatók a nyelvben?
- Melyek a határoló jelek (szeparátorok)?
- Fehér karakterek
- Kis- és nagybetűk használata
- Azonosítók
    - Milyen karakterek használhatóak az azonosítók leírására?
    - Mi az azonosító szintaxisa?
    - Van-e hosszúsági megkötés az azonosítókra?
    - Vannak-e kulcsszavak?
    - Van-e különbség kulcsszó és az előre definiált szó között?
    - Vannak-e írásra vonatkozó konvenciók?
- Értékkonstansok
    - Milyen numerikus értékkonstansok vannak?
    - Milyen más alapok vannak a 10-esen kívül?
    - Mi az egész, illetve valós értékkonstansok szintaxisa?
    - Hogyan határoljuk a sztring értékkonstansokat?
    - Többsoros sztring értékkonstansok megengedettek-e?
    - Vezérlőkaraktereket használhatunk-e sztring értékkonstansokban, és hogyan?
- Megjegyzések
    - Van-e és hogyan használhatóak?
    - Megjegyzés a sor végéig?
    - Teljes sor megjegyzéssé alakítása?
    - Többsoros megjegyzés?
    - Egymásba ágyazható megjegyzések?
    - Dokumentációs megjegyzés?

Például az egész és valós konstansok esetén egyes nyelvek, pl. *Ada*, *Perl*, *Eiffel*, megengedik az aláhúzásjel („_”) használatát is százas elhatárolóként: *123_456, 1_000_000*. A *C*, *C++*, *Java*, *C#* nyelvekben a konstans után írt *l*, *L* vagy *u*, *U* betű tipizálja a számkonstanst: az `5L` *long* (hosszú egész) lesz, a `10u` *unsigned* (előjel nélküli) lesz. *Pascal*ban a valós szám kitevős alakjából elhagyható a tizedespont: *1E+2* a *100.0* valós

számot jelenti. *BASIC*-ben a tizedespont elé nem kell kitenni a nullát: *.5* a *0.5*-öt jelenti. Az *Ada* nyelvben egészeket is megadhatunk exponenciális alakban, ekkor a kitevő pozitív kell, hogy legyen: *1E3 = 1000*. Eiffelben nem szükséges a tizedespont mindkét oldalára számjegyet írni: *-1.* a mínusz egyes valós konstansot jelenti. *C++*-ban a *d*, *D* vagy *f*, *F* utótaggal pontosíthatjuk a valós konstans típusát (*double* vagy *float*). A *Java* nyelv definiálja a *NaN* konstanst (*Not a Number*), valamint a pozitív és negatív végteleneket jelölő *POSITIVE_INFINITY* és *NEGATIVE_INFINITY* konstansokat. Ezeknek véges szám hozzáadása vagy kivonása nem változtatja meg értéküket, szorzatuk a *NEGATIVE_INFINITY*-t eredményezi, összegük nem definiált. A nulla konstansnak előjelt is adhatunk: *+0.0* vagy *-0.0*. Az *1.0 / 0.0* eredménye a *POSITIVE_INFINITY*, míg az *1.0 / -0.0* eredménye a *NEGATIVE_INFINITY*. A *0.0 / 0.0* a *NaN*-t eredményezi.

2.7. Milyen grammatikákkal írható le a nyelv?

A fordítóprogram elméleti hátterére engednek következtetni az itt feltett kérdések:

- Lexikális elemek leírása
- Szintaktika
- Szemantika
- Speciális konstrukciók
- Ortogonalitás
- Egységesség
- Tömörség

Az *ortogonalitás* tulajdonsága azt jelenti, hogy minél több egymástól független elemet tartalmazzon a szintaktika, és ezeket bármely kombinációban lehessen alkalmazni (pl. a következő *C++* deklaráció: ***unsigned long int*** *valtozo;*). A nyelv néhány alaptulajdonsággal rendelkezik, ezen tulajdonságok mindegyike külön-külön is érthető, de együttes használatukkor is értelmes kifejezést kapunk. Így a nyelv könnyebben tanulható lesz, hisz csak kevés elemet kell megtanulni és ezeket kombinálni lehet, hátulütője viszont az, hogy a fordítóprogram logikailag zavaros, vagy kevéssé hatékony kombinációkat is le kell tudjon fordítani.

Az *egységesség* megköveteli, hogy a szemantikailag egységes elemek hasonló fogalmakat takarjanak, a *tömörség* pedig azt, hogy egy elem legyen szemantikailag használható több fogalom értelmezésére, különböző kontextusokban (pl. a „+” operátor stringekre, egészekre, valós számokra, halmazokra.)

2.8. Változók, szimbolikus konstansok

A tárhelyek címzésére, használatára szolgálnak a változók és a szimbolikus konstansok. A programozási nyelv módszereket kell hogy biztosítson mind az egyszerű, mind az összetett típusok konstansainak, változóinak a megadására, ezek későbbi használatára.

- Kell-e deklarálni a változókat?
- Vannak-e globális, lokális változók?
- Hogy lehet deklarálni a konstansokat?
- Léteznek-e speciális megkötések a nevekre?
- Írásra vonatkozó konvenciók
- A deklarációk szintaxisa
- A változóknak adhatunk-e kezdőértéket?
- Hogyan adjuk meg az összetett konstansok értékeit?
- Vannak-e dinamikus változók?
- Vannak-e közös referenciájú változók?
- Vannak-e automatikus változók, statikus változók?
- Regiszterekben tárolt változók
- Létezik-e a perzisztencia fogalma, milyen perzisztenciáról beszélhetünk?

2.9. Kifejezések

A *kifejezések* olyan programelemek, amelyek felhasználásával leírható a számítási folyamat. Szerkezeti és szintaktikai szempontból egy kifejezés *operátorok*ból és *operandusok*ból áll. Egy kifejezés kiértékelésének célja a számítási folyamat elvégzése, mely legtöbbször egy adott változó értékének a kiszámításával ekvivalens.

- Kifejezések szintaxisa
- Létezik-e mellékhatás?
- A logikai kifejezéseket hogyan értékeli ki (teljes, részleges)?

- Milyen műveleteket használhatunk?
- Vannak-e bitműveletek?
- Milyen típusokat téríthet vissza a kifejezés?
- A műveletek sorrendje és a kiértékelés iránya

Logikai kifejezéseknél gyakori a *nem teljes kiértékelés*. Ha egy logikai kifejezés csak *és* műveletekből áll, és az egyik operandusa *hamis*, vagy ha csak *vagy* műveletekből áll és az egyik operandusa *igaz*, illetve az előbbiek értelemszerű kombinációja a *tagadás* művelettel esetén, a kifejezést nem kell teljesen kiértékelnünk, megállhatunk az első olyan operandusnál, amelynél már egyértelmű lesz a kifejezés eredménye. Ilyen esetben vigyázni kell a különféle mellékhatásokkal, mert pl. a meg nem hívott függvények ezeket nem tudják kifejteni. Számos programozási nyelv erre lehetőséget biztosít, ekkor a *complete boolean evaluation* direktívát kell kikapcsolni. Pl. a (1 *és* 0) *és* ((1 *és* 1) *vagy* (0 *és* 1)) logikai kifejezés kiértékelése megállhat az első *és* utáni 0-nál.

Az *infix* jelölési módot a matematikából kölcsönöztük. A bináris operátor a két operandusa között állhat: $a_1\ o^2\ a_2$, pl. $x + y$. Ennek a jelölésmódnak az a hátránya, hogy a kiértékelése nem egyértelmű. Például az $x + y * a - b$ kifejezés esetén, ha semmiféle háttérismeretünk nincs, nem tudjuk eldönteni, hogy a műveleteket milyen sorrendben végezzük el. A háttérismeret, amire szükségünk van, a *művelet prioritása* vagy *precedenciája*, amely az elvégzés sorrendjét határozza meg. Hasonlóan szükségünk van az *asszociativitás* fogalmára is, amely megmondja, hogy több azonos prioritású művelet esetén melyiket kell elvégezni. A matematikai jelölésmódhoz hasonlóan, a prioritás és az asszociativitás megváltoztatására zárójeleket használhatunk. Például az (5 - 1) / (4 - 2) / (8 - 6) kifejezés esetén egyáltalán nem mindegy az asszociativitás. Ha jobbról, vagy balról végezzük el először az osztást 4 vagy 1 eredményhez jutunk!

2.10. Típusok

A *típus* egy olyan absztrakció, amely összefoglalja bizonyos entitások közös tulajdonságait: *kódolás, méret, szerkezet, szemantika*. Egy entitás típusa definiálja azt a halmazt, amelyből az entitás, mint változó, értékeket vehet fel, a memóriában lefoglalt hely méretét, és ugyanakkor definiálja azokat a műveleteket is, melyek az entitással elvégezhetők (szemantikai szinten). Az első programozási nyelvek típusként a primitív hardver típusokat használták, de támogattak néhány összetett típust is, azonban nem volt egy egységes ábrázolási mód kialakulva. A használt típuskonstrukciók függtek a hardvertől, a számítógép felépítésétől. Ezek a különbségek elsősorban a lebegőpontos számábrázolásban nyilvánultak meg.

Csak az 1960-as évek végén kezdték a típusokat absztrakcióként értelmezni, *C. Strachey* és *T. Standish* munkája hatására.

Egy típushoz *konstruktorok*at, *szelektorok*at és *predikátumok*at rendeltek. Ekkor születtek meg többek között a *Simula* és az *Algol68* nyelvek, amelyek már magasfokú típusszemlélettel bírtak: általános, nagy kifejezőerejű típusfogalmat használtak, szigorú *típusellenőrzés* mellett. Már ekkor kidolgozták és implementálták a *típusmegfelelési* (*kompatibilitási*, *compatibility*), *típuskiterjesztési* (*extension*) és *típusátalakítási* (*konverziós*, *conversion*) szabályokat.

Az *Ada83* nyelv újabb fontos mérföldkő volt a típusok terén. Definiálta a *típusértékhalmaz* és a *típusművelet* fogalmakat. Ezeken kívül lehetővé tette a típussal való paraméterezhetőséget, megszüntette a biztonsági réseket, megpróbálta szabályozni és explicitté tenni a szemantikailag szabálytalan programozási eszközöket.

A típusok elemzésekor a következő kérdésekre keresünk választ:

- Mit jelent a nyelvben az adattípus?
- Adattípusok szemantikája
- Írásra vonatkozó konvenciók
- Mik a beépített adattípusok?
  - Milyen elemi típusokkal rendelkezik a nyelv?
  - Milyen összetett típusokkal rendelkezik?
  - Szintaxis
- Szigorúan típusos-e a nyelv?
- Vannak-e a típusoknak egyéni jellemzői?
- Vannak-e beágyazott típusok?
- Rendezett típusok:
  - Kezdőértékük 0 vagy 1?
  - A logikai típus felsorolási típus-e, vagy önálló?
- Az egész és valós típusok számát a nyelv definíciója vagy az implementáció szabja meg?
- Milyen valós típusokat implementál a koprocesszor?
- Mutatók

    - Vannak-e típus nélküli pointerek?
    - Mire kellenek a mutatók?
        - Hogy ne kelljen nagy adatszerkezeteket átadni?
        - Dinamikus adatszerkezetek építéséhez?
        - Objektumorientált funkciókhoz?
    - Mutatóaritmetika
        - Mit jelent a „+” operátor mutatókra?
        - Hogyan történik a mutató és a mutatott objektumok értékadása?
        - Van-e, és mit jelent az egyenlőségvizsgálat?
    - Van-e mutató dereferencia művelet?
    - A szemétgyűjtés automatikus, vagy manuális?
    - Van-e sehová sem mutató mutató (nil, null)?
    - Lehet-e automatikus változóra pointer?
    - Lehet-e több mutató egy objektumra?
    - Van-e alprogramra mutató mutató?
- Lehet-e felhasználói típust deklarálni?
    - Szintaxis
    - Van-e altípus-definiálás?
    - Elérhetők-e az alábbi típuskonstrukciók, és ha igen, milyen tulajdonságokkal?
        - Tömb
            - Mi lehet az indexe?
            - Mi lehet az eleme?
            - Csak ugyanolyan típusú elemei lehetnek?
            - Van-e indextúlcsordulás-ellenőrzés?
            - Van-e kezdőérték-adás?
            - Van-e egyben értékadás?
            - Vannak-e dinamikus tömbök?
            - Vannak-e konstans tömbök?
            - Mikor dől el a mérete, a helyfoglalása?
            - Van-e többdimenziós tömb?
            - Van-e altömb (szelet) képzés?
            - Vannak-e speciális tömbök?
        - Rekord (direktszorzat)
            - Van-e kezdőérték-adás?
            - Van-e egyben értékadás?
            - Van-e rekord konstans?
            - Hogyan működik a kiválasztás művelet?
            - Vannak-e speciális rekordok?
        - Variáns rekord (unió)
            - Meg lehet-e állapítani, hogy a rekord melyik változat szerint volt kitöltve?
            - Ki lehet-e olvasni a kitöltésitől különböző változat szerint?
            - Vannak-e speciális variáns rekordok?
        - Halmaz
        - Állomány
            - Milyen állománytípusok vannak?
            - Milyen műveletek végezhetők állományokkal?
            - Vannak-e speciális állományok?
    - Vannak-e speciális típuskonstrukciók?
    - Vannak-e absztrakt adattípusok, típussablonok?
- Léteznek-e névtelen típusok?
- Létezik-e Variant típus?
- Mikor ekvivalens két típus?
    - Strukturális ekvivalencia esetén
        - A rekordok mezőneveit is figyelembe vették, vagy csak a struktúrájukat?
        - Számít-e a rekordmezők sorrendje?

        - Tömböknél elég-e az indexek számosságának egyenlőnek lenni, vagy az indexhatároknak is egyezniük kell?
    - Név szerinti ekvivalencia esetén
        - Deklarálhatók-e egy típushoz típusok, amelyekkel ekvivalens?
        - Névtelen tömb- illetve rekordtípusok ekvivalensek-e valamivel?
- Mikor kompatibilis két típus?
- Mi történik túlcsorduláskor?
- Típuskonverziók
    - Van-e, és hogyan működik
        - az automatikus konverzió?
        - az identitáskonverzió?
        - a bővítő konverzió?
        - a szűkítő konverzió?
        - a *toString* konverzió?

Érdekes a *Delphi Variant* típusa. A *Variant* típus olyan értékek tárolására szolgál, amelyeknek típusa nem ismeretes fordítási időben. Egy ilyen típusú változó tehát bármilyen értéket felvehet. A *Variant* típus 16 byte nagyságú helyet foglal a memóriában, ezen a helyen egy típusdeszkriptort és egy értéket, vagy egy mutatót egy értékre tárol. A *Variant* típus segítségével egész számokkal indexelhető dinamikus tömböket is létre lehet hozni. A tömbök elemei tetszőleges típusúak – akár tömbök is – lehetnek.

2.11. Utasítások, vezérlési szerkezetek

Az *utasítások* a program legalapvetőbb, algoritmikus részei. Az eredmény eléréséhez szükséges műveleteket – algoritmusokat – írják le. Az utasításokat általában kulcsszavak alkotják.

- Írásra vonatkozó konvenciók
- Egyszerű utasítások
    - Az értékadás egyszerű utasítás, vagy kifejezés utasítás?
    - Van-e többszörös értékadás?
    - Ki kell-e írni az üres utasítást?
    - Hogyan valósul meg az eljáráshívás?
    - Van-e ugróutasítás?
- Összetett utasítások, vezérlési szerkezetek
    - Van-e eseményvezérelt programozás?
    - Szekvencia
        - Terminátor vagy utasításelválasztó-e a „;” vagy más karakter?
    - Lehet-e blokkutasítást létrehozni és hogyan?
        - Lehet-e a blokk üres?
        - Elhelyezhető-e a blokkutasításban deklaráció?
        - Mi történik blokkból való kilépéskor?
    - Van-e makró-szubsztitúció, kódblokkok értelmezésére lehetőség?
    - Elágazás
        - Van-e kétirányú elágazás?
        - Hová tartozik az „else”?
        - Van-e aritmetikai elágazás?
        - Van-e többirányú elágazás?
        - Többirányú elágazás
            - Mi lehet a szelektor típusa?
            - Fel kell-e sorolni a szelektortípus minden lehetséges értékét?
            - Mi történik, ha fel nem sorolt értéket vesz fel a szelektor?
            - Rácsorog-e a vezérlés a következő kiválasztási ágakra is?
            - Diszjunktnak kell-e lennie a kiválasztási értékeknek?
            - Meg lehet-e adni intervallumot a szelektorértéknek?
            - Mi állhat a kiválasztási feltételben a következők közül?
                - egy érték
                - értékek felsorolása
                - intervallum
                - más is

- Ciklus
  - Vannak-e változó lépésszámú ciklusok?
    - Van-e elöl tesztelős ciklus?
    - Van-e hátul tesztelős ciklus?
    - A feltétel ciklusának logikai értéknek kell lennie, vagy más típusú is lehet?
    - Kell-e blokkot kijelölni a ciklusutasításnak?
  - Van-e fix lépésszámú ciklus?
    - A ciklusváltozó mely jellemzője állítható be a következők közül?
      - alsó érték
      - felső érték
      - lépésszám
    - Mi lehet a ciklusváltozó típusa?
    - Biztosított-e a ciklusmagon belül a ciklusváltozó változtathatatlansága?
    - A ciklushatárokat lehet-e dinamikusan változtatni?
    - Mi a ciklusváltozó hatásköre, definiált-e az értéke kilépéskor?
    - Lehet-e a ciklusutasításban ciklusváltozót deklarálni?
  - Van-e iterátor ciklus?
  - Van-e ciklusváltozó-iterátor?
  - Van-e általános ciklus, és hogy néz ki?
  - Léteznek-e a következő vezérlésátadó utasítások?
    - break
    - continue
    - kivételek
- Van-e hivatkozás utasítás?
- Vannak-e más, speciális utasítások?
- Az utasítások szintaxisa

Az üres utasítás jelölésére *COBOL*ban a ***NEXT SENTENCE***-t, *Pascal*ban a „;”-t, *Pyton*ban a ***pass*** kulcsszót használjuk akkor, ha szintaktikailag szükség van egy utasításra, de a programban nem kell semmit sem csinálni.

C++-ban a blokk fogalma sokkal többet fed, mint *Pascal*ban. A *Pascal* blokkdefinícióján kívül a következő elemeket tartalmazza: egy blokkon belül deklarált változó lokális az illető blokkra nézve; egy blokkból való kilépés alkalmával automatikusan meghívódik az összes blokkon belül használt objektum destruktora. *Ada*ban minden blokk elején újabb változókat deklarálhatunk, ezeknek külön cikkely van fenntartva, amit a ***declare*** kulcsszó vezet be.

Az elágazási utasítások valósították meg először a futás pillanatában történő döntést bizonyos feltételek függvényében. Ennek a megvalósításnak köszönhető, hogy ugyanaz az algoritmus különböző bemeneti értékek illetve részeredmények alapján önmagából más-más lineáris utasítássorozatot hajtson végre. Ettől az újítástól vált a lineárisan programozható algoritmust végrehajtó gép számítógéppé. Ez a megvalósítás Neumann Jánosnak tulajdonítható. Az első magas szintű nyelvben megjelent elágazás a *FORTRAN*-beli aritmetikai ***IF***: ***IF*** *(AritmetikaiKifejezés) E1, E2, E3*. Az elágazás az *AritmetikaiKifejezés* értékétől (negatív, nulla, pozitív) függ, és ennek alapján a programban az *E1, E2* vagy *E3*-as címkékre történik ugrás.

COBOL-ban a ciklusmag számára külön blokkot, alprogramot (paragrafust) kellett írni, és ezt a ***PERFORM*** utasítással lehetett meghívni. A *C#* bevezeti az *iterátor ciklust* is. Ezáltal lehetőség van olyan számlálásos, ciklusváltozóval ellátott ciklus megszervezésére, ahol a ciklusváltozó rendre felveszi egy előre megadott, felsorolható halmaz elemeinek értékeit (***foreach***). A *Pyton* érdekessége még, hogy a ciklus utasításoknak lehet egy ***else*** águk is. Ez az ág akkor hajtódik végre, ha a ciklus végighaladt a listán (***for*** esetén), illetve ha a feltétel hamissá vált (***when*** esetén), de nem hajtódik végre, ha a ciklust a ***break*** utasítással szakítottuk meg.

2.12. Kivételkezelés

A kivételek (*exception*) olyan hibás események, amelyek megszakítják az alkalmazás szabályszerű futását. Ilyenkor a vezérlés a kivételkezelőnek adódik át. A kivételkezelés nem egyszerű feladat, hiszen alkalmazásunk minden egyes forrássora potenciális hibaforrás is egyben. Ha már egy hibával szembekerültünk, célszerű azt kezelnünk, vagyis olyan tevékenységeket végeznünk, amelyekkel a hibák hatását eltüntethetjük vagy legalább „enyhíthetjük”. Ha egy hibát nem sikerül kezelnünk, szeretnénk annak helyéről, körülményeiről mindent tudni.

- Hiba- vagy kivételkezelést ad-e a nyelv?
- Milyen beépített kivételek vannak?
- Definiálhatunk-e saját kivételt?
- Milyen kivételkezelők vannak?
    - Ha kivétel lép fel, akkor...
    - Mindenképpen el kell végezni...
- Kivételkezelők szintaxisa
- Milyen programelemekhez köthető a kivételkezelő?
- Milyen hatáskörrel, élettartammal rendelkezik a kivételkezelő?
- Többszörös kivételek
- Hogyan folytatódik a program kivételkezelés után?
- Vannak-e beágyazott kivételkezelők?
- Van-e általános kivételkezelő?
- Van-e automatikus kivételkezelő?
- Párhuzamos környezetben vannak-e speciális kivételek?

2.13. Programegységek

A programozási nyelvek lehetőséget biztosítanak a programok bizonyos egységekre (*fordítási egységek*) való felosztására, klasszifikálására. Az utasításokat, műveleteket és adatokat tehát nem ömlesztve tartalmazza egy-egy forrásszöveg-állomány, hanem ezek valamilyen logikai vagy a programozó által meghatározott sorrendet követve rendezhetők a nyelv szintaxisának megfelelő állományokba. Ezek az állományok lehetnek moduláris egységek, vagyis külön-külön is van értelme mindegyiküknek, egymástól független egységek, vagy lehetnek olyan egységek, amelyek egymagukban semmit sem jelentenek, csak közös fordítás és láncolás után lesz meg az igazi értelmük.

- A program felépítése, hogy néz ki a főprogram?
- Minimális főprogram
- Milyen moduláris egységek léteznek, és ezek hogy néznek ki?
- Minimális egységek
- Az egységek szintaxisa
- Írásra vonatkozó konvenciók
- Létezik-e átlapoló egység (Overlay)?
- A vizuális elemek hogyan kötődnek az egységekhez?
- Lehet-e forrásszöveget inkludolni?
- Létrehozhatók-e DLL-ek, és hogyan?
- Lehet-e erőforrásokat (Resource) használni?
- Lehet-e külső OBJ állományt a programhoz szerkeszteni?
- Lehet-e más programozási nyelvben megírt alprogramokat használni?
- Vannak-e speciális egységek?

Az átlapoló egységek egymástól függetlenül végrehajtható programrészeket tartalmaznak. A memóriába egyszerre csak egy átlapoló rész töltődik be, és végrehajtás után felszabadul. A magasabb szintű programozási nyelvek megengedik az átlapoló egységek megírását. Lássuk, hogy valósul ez meg Borlad Pascalban: Az átlapoló egységek írása az Overlay unit (OVERLAY.TPU) használatával történik. Ez az egység tartalmazza az átlapolást kezelő függvényeket, eljárásokat, szimbólumokat. A {$O} direktíva engedélyezi vagy letiltja az átlapolásos kód generálását. A {$O EgységNév} direktíva pedig egy egységet egy overlay ágba irányít. Az OVR állományt az EXE állományhoz lehet másolni a copy DOS paranccsal (copy /b nev.exe+nev.ovr nev.exe), ha az Options / Debugger menüpontból a Standalone Off állapotban van és az OvrInit paramétere a ParamStr(0), vagyis az EXE állomány teljes elérési útvonala. Overlayt használó programot csak lemezre lehet fordítani.

2.14. Absztrakciós szintek

Az absztrakciós szintek a nyelv modularitását, strukturálhatóságát, a procedurális absztrahálás megvalósíthatóságát célozták meg. A procedurális absztrahálás egyike a legrégibb programozási eszközöknek, Charles Babbage már 1840-ben azt tervezte, hogy lyukkártyák egy csoportját fogja használni nagyobb számítások gyakrabban használt részeinél.

- Vannak-e alprogramok (eljárások, függvények)?
- Vannak-e függvények?
- Van-e különbség eljárás és függvény között?
- Írásra vonatkozó konvenciók
- Ki kell-e írni az üres paraméterlistát határoló jeleket?
- Hívási konvenciók
- Verem felépítése
- Paraméterátadás sorrendje
- Lehet-e alprogramokat egymásba ágyazni?
- Mik a láthatósági és a beágyazási területek?
- Hány belépési pontja lehet egy alprogramnak?
- Vannak-e korutinok?
- Engedélyezettek-e a mellékhatás eljárások, függvények esetén?
- Rekurzív hívások
- Megadhatók-e elő- és utófeltételek?
- A függvényeknek milyen visszatérési értékeik lehetnek, és hogyan jelöljük a visszatérést?
- Milyen paraméterátadási módokkal rendelkezik a nyelv?
- Vannak-e alapértelmezett értékek?
- A paraméterlista mérete lehet-e változó?
- Jó-e és meghatározható-e a formális-aktuális paraméterek közötti információáramlás?
- Az alprogram neve vagy szignaturája azonosítja ezt?
- Definiálhatók-e operátorok, ezek átlapolhatók-e?
- Léteznek-e sablonok?
- Lehet-e alprogrammal paraméterezni?
- Használhatók-e az alprogramok változókként?
- Lehet-e típussal paraméterezni?
- Van-e lehetőség generikus programozásra?
- Hogyan néznek ki a ki/bemeneti (I/O) műveletek?
- Van-e beágyazott assembly?
- Van-e beágyazott gépi kód értelmező?
- Kapcsolat az API-val
- Vannak-e más beágyazott lehetőségek?
- A kód újrafelhasználhatósága
- Ha hibrid nyelv, hogyan keverhetők a különböző paradigmák?
    - Imperatív
    - Objektumorientált
    - Funkcionális
    - Logikai
    - Párhuzamos és osztott
    - Vizuális
    - Ötödik generációs

Számos programozási nyelv nem tesz különbséget eljárás és függvény között. Például *C*, *C++*, *Java*, *C#*, ezekben a nyelvekben minden alprogram függvény – ha a visszatérési érték típusa üres (*void*), ezek eljárásoknak tekinthetők, de a nem üres típusú visszatérési értékű függvények is hívhatók egyszerű eljárásként. Más nyelvekben (pl. *Ada*, *Pascal*) éles a különbség az eljárás és a függvény között, olyanannyira, hogy külön kulcsszóval kell deklarálni őket.

Egyes programozási nyelvek esetében (pl. *C*, *C++*, *Java* stb.) az üres paraméterlistát határoló zárójeleket is ki kell tenni az alprogram neve után, más nyelvekben (*Pascal*, *Ada*) ezt nem szabad, vagy nem feltétlenül kell (*PL/I*) kitenni.

A koritunok olyan speciális alprogramok, amelyek szakaszosan adhatják át egymásnak a vezérlést. Egy korutin meghívhat egy másik korutint saját maga befejezése előtt, ekkor mindkettő szakaszosan fog futni. Másodszori meghívásnál ott fogja folytatni tevékenységét, ahol először abbahagyta – így a párhuzamosság látszatát kelti. A korutinok tehát tetszés szerint adogathatják át egymásnak a vezérlést, nincs külön hívási vermük, ezért a korutinok hívását inkább *folytatás*nak (*resume*) szokás nevezni. Kevés nyelv támogatja a korutinokat, pl. *SIMULA*, *Modula-2*. Korutinokat általában a ***coroutine*** kulcsszóval lehet deklarálni, és élet-

ciklusukban létezik két fontos pillanat: az első a *detach*, mikor az új korutint leválasztjuk a régiről (***detach***), a második a *transfer*, amikor átadjuk vagy visszaadjuk a vezérlést (***transfer***).

2.15. Végkövetkeztetések

Az elemzés utolsó szakaszában a megismert programozási nyelvről keltett benyomásainkat összegezzük:

- Megoldatlan problémák
- Vélemények a nyelvről
- Erőssége
- Gyengesége
- Továbbfejleszthetőség
- Megbízhatóság
- Általánosság

Könyvészet

[1.] Fischer, Alice E.; Grodzinsky, Frances S.: *The anatomy of programming languages*, Prentice-Hall, 1993.
[2.] Horowirtz, Ellis: *Fundamental Concepts of Programming Languages*, Computer Science Press, division of W.H. Freeman, New York, 1983.
[3.] Horowitz, Ellis (ed.): *Programming Languages: A Grand Tour*, Computer Science Press, Rockville, Maryland, 1984.
[4.] Horowitz, Ellis: *Magasszintű programnyelvek*, Műszaki Könyvkiadó, Budapest, 1987.
[5.] Pârv, Bazil; Vancea, Alexandru: *Fundamentele limbajelor de programare*, Ed. Albastră, Kolozsvár, 1996.
[6.] Sammet, J.: *Programming Languages: History and fundamentals*, Prentice Hall, Englewood Cliffs, NJ, 1969.
[7.] Scott, M.K.: *Programming Languages Pragmatics*, Morgan Kaufmann, San Francisco, 2000.
[8.] Sethi, R.: *Programming Languages: Concepts and Constructs*, Addison-Wesley, Reading Mass., 1996.
[9.] Bíró Ernő; Kovács Lehel: *A programozási nyelvek alapjai*, Komp-Press, Kolozsvár, 1997.
[10.] Kovács D. Lehel István: *Programozási nyelvek összehasonlító elemzése – Az objektumorientált paradigma*, Presa Universitară Clujeană, Cluj-Napoca, 2001.
[11.] Nyékyné Gaizler, Judit (ed.): *Programozási nyelvek*, Kiskapu Könyvkiadó, Budapest, 2003.
[12.] Kovács D. Lehel István: *Programozási nyelvek összehasonlító elemzése – A programozási nyelvek anatómiája*, Presa Universitară Clujeană, Cluj-Napoca, 2000. ISBN: 973-8095-63-8. p. 264., 2004.

Szekvenciális és párhuzamos algoritmusok

Sequential and Concurrent Algorithms

Dr. KALLÓS Gábor

Széchenyi István Egyetem, Győr
Számítástechnika Tanszék

Abstract

In this paper we present a relatively new research field: the theory of algorithms for concurrent machines. We analyze how classical algorithms and programming methods can be transformed into the concurrent environment, and how the totally new ideas can be applied. Among some interesting exercises which are included, the motivated reader can study further following the references.

Összefoglalás

A cikkben egy érdekes és viszonylag új kutatási területtel, a párhuzamos gépekre kifejlesztett algoritmusokkal foglalkozunk, elsősorban elméleti eredmények bemutatásával.

Először áttekintjük három különböző fejlett programozási módszer (rekurzió, dinamikus programozás, mohó algoritmusok) párhuzamosítási lehetőségeit. Ezután néhány klasszikus matematikai probléma megoldásának felgyorsítását mutatjuk be párhuzamos környezetben, majd a cikk végén röviden foglalkozunk a teljesen új elveken alapuló párhuzamos algoritmusok megalkotásához használható ötletekkel.

Az anyag önálló gyakorlását feladatok segítik. A további tanuláshoz, kutatáshoz a cikkben angol és magyar nyelvű szakirodalmi hivatkozásokat helyeztünk el, itt a bemutatott problémák egy része részletesebben is áttekinthető, ill. további érdekes anyagrészek és feladatok is találhatók.

A szerző e-mailben szívesen válaszol a felmerülő kérdésekre.

Az algoritmusok hatékonyságának mérése

Az algoritmus olyan pontosan definiált számítási eljárás, amely egy meghatározott feladatot elvégezve bemeneti (input) adatokból kimeneti (output) értékeket állít elő. Egy algoritmust a feldolgozandó elemek függvényében a lépésszámmal (és így végeredményben a futási idővel) jellemezhetünk. Ez a költségfüggvény elméletileg minden algoritmus esetében meghatározható.

Az algoritmusok programnyelvtől független megadásához és elemzéséhez elméleti gépmodelleket használunk, amelyek egy processzort és írható-olvasható memóriát tartalmaznak. A processzor lépésenként (szekvenciálisan) végrehajtja az algoritmust, eközben használhatja a memóriát. Az algoritmust a gép számára valamely általános algoritmus-leíró nyelvben adjuk meg. Ilyen modellek például a Turing-gép és a RAM gép (random-access machine).

A Turing-gép a gépi számítások legrégibb és legismertebb modellje. Turing angol matematikus még 1936-ban vezette be. Egy korlátos központi részből és egy végtelen tárból áll. Előnye, hogy bármely számítás elvégezhető rajta, amelyet akár előtte, akár utána bármilyen más gépmodellen el tudtak végezni. Lényeges hátránya ugyanakkor, hogy a távoli memóriablokkokat csak szekvenciálisan lehet elérni. Mivel erősen eltér a valós gépektől, így főleg elméleti vizsgálatokra alkalmas.

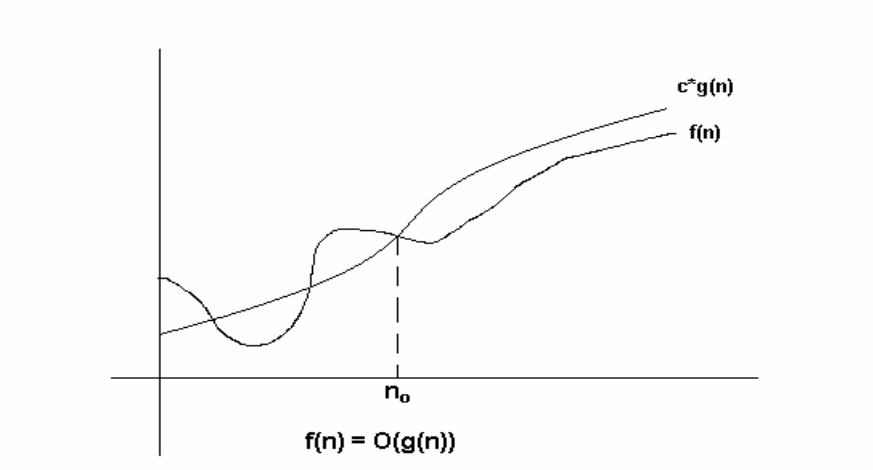
A RAM gép a memória bármely részét egy lépésben el tudja érni, ezért inkább tekinthető a valódi számítógépek leegyszerűsített modelljének, mint a Turing-gép. Ezen gép memóriája szintén korlátlan, és minden memóriarekeszben tetszőlegesen nagy egész szám tárolható.

Ezeknél a modelleknél a műveletek költségét a szükséges memória-hozzáférések száma határozza meg.

Def.: Azt mondjuk, hogy az f(n) függvény eleme az Ordó(g(n)) halmaznak, ha van olyan c konstans és n_0 küszöbérték úgy, hogy valamely $n>n_0$-ra mindig

$$0 \le f(n) \le c \cdot g(n).$$

Sok esetben az „f(n) eleme Ordó(g(n))” helyett az „f(n) = Ordó(g(n))” vagy az „f(n) Ordó(g(n))”-es írásmódot is használjuk. Rövidítés: O(g(n)).



Az Ordó(g(n)) f egy felső korlátja, amely lehet éles, illetve nem éles. A tovább már nem finomítható korlátot éles korlátnak nevezzük.

Például n^2 = Ordó(n^2) éles korlát, n^2 = Ordó(n^3) nem éles felső korlát. A továbbiakban az Ordó jelölést külön említés nélkül éles korlátként használjuk.

Megj. Ez a megkötés egyszerűsítéshez vezet, ugyanis így nem tudjuk megkülönböztetni az éles és a nem éles korlátot. Pontosabb mérést tesz lehetővé a „Théta” jelölés bevezetése (ld. [Co]).

Sok klasszikus probléma, illetve algoritmus esetében már régóta ismert, hogy a megoldások költségfüggvénye milyen Ordó(g(n)) függvényosztályba tartozik. Ugyanazon probléma esetén azokat a megoldásukat tartjuk hatékonynak, amelyek szintén ilyen költségűek.

F: Vizsgáljuk meg, hogy milyen éles korlátot tudunk adni a következő klasszikus algoritmusok műveletigényére (n elemű tömböket használunk). Foglalkozzunk külön a legrosszabb, a legjobb és az átlagos esettel:

Szekvenciális keresés, logaritmikus keresés; buborékos rendezés, kiválasztásos rendezés, beszúrásos rendezés, gyorsrendezés. (3)

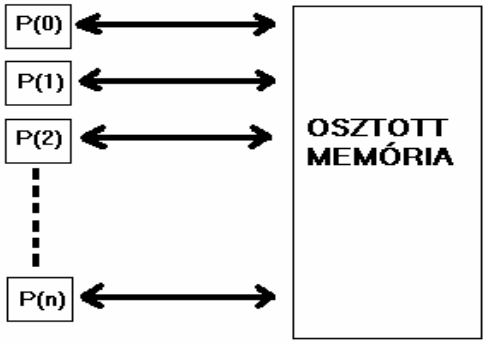
Megj. A feladatok utáni számok az adott feladat nehézségi szintjét adják meg.

Egy párhuzamos gépmodell

Bár már az 1940-es években készült első számítógépek is rendelkeztek bizonyos párhuzamosítási lehetőséggel (pipeline technika), az igazi többprocesszoros számítógépek csak 1-2 évtizede kezdtek elterjedni. Ezzel megnőtt az érdeklődés a párhuzamos algoritmusok iránt is, amelyekben egyidejűleg több művelet is futhat. Ma már sok olyan probléma megoldására is léteznek párhuzamos algoritmusok, amelyeket azelőtt közönséges, szekvenciális algoritmussal oldottunk meg.

A párhuzamos algoritmusok megadására és jellemzésére használt elméleti modell a párhuzamos véletlen hozzáférésű gép (PRAM gép, paralell random-access machine).

A PRAM modellben a processzorok az osztott memóriához csatlakoznak. Az osztott memóriában bármely szót minden processzor egységnyi idő alatt ér el. A PRAM gép processzorai egymással párhuzamosan dolgozni képes RAM gépeknek tekinthetők.



A PRAM modellben az algoritmus hatékonyságát a párhuzamos memória-hozzáférések számával mérjük. Ez a közönséges RAM modellnél használt azon feltételezés közvetlen általánosítása, miszerint a memória-hozzáférések száma (aszimptotikusan) jó közelítést ad a futási időre.

Megj. A valóságban a processzorok számának növekedésével nő az elérésük ideje is.

Egy párhuzamos algoritmus futási ideje az algoritmust végrehajtó processzorok számától is függ. Ezért amikor egy PRAM algoritmust vizsgálunk, a feladat bonyolultsága mellett a processzorok számát is figyelembe kell vennünk, ellentétben a szekvenciális algoritmusokkal. Egy algoritmus futási ideje és az általa használt processzorok száma között általában szoros kapcsolat áll fenn, amennyiben az egyik paraméter csak a másik kárára javítható.

PRAM gépekre írt egyes algoritmusokkal a cikk végén még részletesen foglalkozunk.

Rendezések és rekurzió

A klasszikus rendezések – buborékos, kiválasztásos, beszúrásos – Ordó(n^2)-es algoritmusok, hiszen a megoldáshoz két egymásba ágyazott ciklust használunk fel. Az Ordó-ban „elbújtatott” konstansok eredményezhetnek ugyan jelentős sebesség-különbséget egyes algoritmusok (például a hagyományos és a javított buborékos rendezés) között, de a különbség csak *konstansszoros*.

A gyorsrendezés

Az Ordó(n^2)-es algoritmusoknál ismerünk hatékonyabb rendező eljárásokat is. A legjobb általánosan is használható módszer a gyorsrendezés, amelyet C. A. Hoare mutatott be 1961-ben. Az algoritmus rekurzív, a teljes sorozat rendezését visszavezeti két félsorozat, majd négy negyedsorozat, …, rendezésére. A rendező eljárás meghív egy felosztó rutint, amely egy ún. „könyökelem” kiválasztása után átcserélgeti az eredeti tömb elemeit úgy, hogy a tömb felső részében ne maradjon a könyökelemnél kisebb, alsó részében pedig nála nagyobb elem.

Az algoritmus pszeudokódja:

```
gyorsrendezés(A, p, r)
      ha p < r akkor
            q := feloszt(A, p, r)
            gyorsrendezés(A, p, q)
            gyorsrendezés(A, q+1, r)
      elágazás vége
eljárás vége

feloszt(A, p, r)
      x := A[p]
      i := p - 1
      j := r + 1
      ciklus amíg i < j              # while ciklus, bennmaradási feltétel
            ciklus                   # repeat-until ciklus
                  j := j - 1
            amíg A[j] <= x           # kilépési feltétel
            ciklus
                  i := i + 1
```

```
            amíg A[i] >= x
            ha i < j akkor csere(A[i], A[j])
        ciklus vége
        return(j)
eljárás vége
```

A gyorsrendezés hatékonyságát alapvetően az határozza meg, hogy a felosztó eljárás hogyan vágja szét az „A” tömböt kisebb részekre, illetve, hogy milyen stratégia szerint választjuk ki a „q” könyökelemet.

A legszerencsésebb esetben a felezés után a részsorozatok elemszáma közel megegyezik, és ez a tulajdonság a rekurzió során végig megmarad.

Jelöljük „T(n)”-nel „n” elem rendezésének költségét. Ekkor

$$T(n) = 2 \cdot T(n/2) + \text{Ordó}(n),$$

ahol Ordó(n) a szétválogatás költsége (feloszt eljárás), „T(n/2)” pedig az „n/2” elem rendezésének költsége. A képletet kifejtve

$$T(n) = 2 \cdot T(n/2) + O(n) = 4 \cdot T(n/4) + 2 \cdot O(n) = \ldots = n \cdot T(1) + \lg n \cdot O(n) = O(n \cdot \lg n).$$

Megj. A képlet kiszámolható rekurzívan kifejtve illetve az ún. Mester tétel (ld. [Co]) alkalmazásával. Ugyanez a tétel használható a cikkben található többi rekurzív képlet kiszámítására is. A technikai részletekkel nem foglalkozunk.

A legrosszabb felosztás esetén az egyik félsorozat elemszáma mindig 1. Ekkor

$$T(n) = T(n - 1) + O(n) = T(n - 2) + 2 \cdot O(n) = \ldots = n \cdot O(n) = O(n^2).$$

A gyorsrendezés gyakorlati alkalmazhatóságát erősen korlátozná, ha egy véletlen számtömb esetén sokszor kapnánk a legrosszabb esetet. Szerencsére – mint a következőkben láthatjuk – ez nem így van.

A gyorsrendezés átlagos futási ideje sokkal közelebb áll a legjobb, mint a legrosszabb futási időhöz. Vegyük például azt az esetet, amikor a felosztás mindig 9:1 arányban történik. Ekkor

$$T(n) = T(9n/10) + T(n/10) + O(n) = \ldots = O(n \cdot \lg n)$$

Ugyanezt kapnánk minden állandó arányú felosztáskor is, hiszen a rekurziós fa mélysége ekkor Ordó(lgn), és a költség minden szinten Ordó(n). A futási idő tehát Ordó(n·lgn) bármilyen állandó arányú felosztáskor. Ezeket a felosztásokat ezért kiegyensúlyozott felosztásoknak is nevezzük. A gyakorlati tapasztalatok azt mutatják, hogy általános tömb esetében a rossz felosztások viszonylag ritkán fordulnak elő, általában valamilyen kiegyensúlyozott felosztást kapunk. További igazolás nélkül megemlítjük, hogy a gyorsrendezés futási teljesítménye „n” elem összes permutációit tekintve csak néhány esetben éri el vagy közelíti meg a legrosszabb esetet (részletes igazolás: [Co]).

Tovább javítható a rendezés várható hatékonysága a könyökelem véletlen kiválasztásával, illetve az elemek véletlen átrendezésével az algoritmus alkalmazása előtt.

Összefoglalva a fenti elemzést:

A gyorsrendezés futási ideje átlagos esetben (gyakorlatilag szinte mindig) Ordó(n·lgn).

F: Elemezzük, hogyan viselkedik a fenti algoritmus teljesen rendezett és fordítva rendezett tömb esetén. (2)

Rekurzív algoritmusok párhuzamosítása

A klasszikus rendező algoritmusok nem párhuzamosíthatók kézenfekvően, a gyorsrendezés esetén azonban több processzor használatával rövidebb futási időt érhetünk el az egymástól független lépések egyidejű végrehajtásával. Két processzorral rendezhetjük például az első rekurziós lépés után kapott két félsorozatot, majd négy processzorral a második lépés utáni négy negyedsorozatot, és így tovább. Végeredményben, ha van „n” illetve „n/2” darab processzorunk, akkor a teljes időigény csak

$$O(n) + O(n/2) + O(n/4) + \ldots + O(1) = O(n),$$

figyelemben tartva azt, hogy a teljes műveletigény továbbra is T(n) = O(n·lgn). Természetesen ez az eredmény elméleti jellegű, mert nem foglalkoztunk azzal, hogy mily módon lehet a részfeladatokat az egyes pro-

cesszoroknak kiosztani, és az eredményt tőlük begyűjteni. Ez a gyakorlatban a teljes végrehajtás lelassulását eredményezi.

Hasonló párhuzamosítási lehetőség adódik bármely olyan rekurzív algoritmusnál, ahol egymástól független részfeladatokat kell végrehajtani.

Dinamikus programozás

A dinamikus programozás a rekurzióhoz hasonlóan az eredeti feladatot részproblémákra osztással oldja meg, de a részproblémák most nem (teljesen) függetlenek egymástól, sőt egyes részproblémák megoldása ismétlődően előfordulhat. Emiatt a már megoldott részfeladatok eredményét táblázatban tároljuk, és amennyiben újból szükség van rájuk, visszakeressük a megoldást. Erről a táblázatos megadásról nevezték el magát a módszert is. A dinamikus programozást optimalizálási problémák megoldásához használjuk, ilyenkor a feladatra található egy vagy több optimális megoldás, és nekünk egyet elő kell állítani, illetve az értékét meg kell határozni. Az előállítás lépései a következők:

1. Jellemezzük az optimális megoldás szerkezetét.
2. Rekurzívan definiáljuk az optimális megoldás értékét.
3. Kiszámítjuk az értéket alulról felfelé.
4. Megszerkesztünk egy optimális megoldást (amennyiben ez is szükséges).

A rekurzióhoz hasonlóan a dinamikus programozással megoldható feladatok esetében is gyorsulást eredményezhet a párhuzamos megoldás. Nagyon lényeges azonban a processzorok közötti jó kommunikáció, hiszen szükséges, hogy a már kiszámított részeredmények a többi processzor számára is hozzáférhetőek legyenek.

Mohó algoritmusok

Hasonlóan a dinamikus programozáshoz, a mohó stratégiát is optimalizálási problémák megoldásához használjuk – van több optimális megoldás, egyet előállítunk. A mohó stratégia kevesebb lépéssel dolgozik, mint a dinamikus programozás. Mindig az adott lépésben optimálisnak látszót választja , feltételezve, hogy a lokális optimum globális optimumhoz vezet. Mivel ez a feltételezés sok problémára nem teljesül, ezért a mohó stratégia nem alkalmazható minden esetben.

Azok a problémák oldhatóak meg a mohó stratégiával, amelyekre teljesül a következő két feltétel:

a) mohó választási tulajdonság (lokális optimumot választunk, az aktuális választás függhet a korábbi, de nem függhet a későbbi választásoktól);
b) optimális részproblémák tulajdonság (optimális megoldás építhető a részproblémák optimális megoldásából).

Klasszikus mohó stratégiával megoldható feladat például a pénzkifizetési probléma „normál” pénzrendszer esetén, illetve a Huffman-kód előállítása karaktersorozat optimális kódolására.

A mohó stratégia szekvenciális jellege miatt nem párhuzamosítható, ezért nem foglalkozunk vele részletesebben.

Megj: A rekurzió, a dinamikus programozás és a mohó algoritmusok részletes bemutatása (szekvenciális környezetben) megtalálható a [Co] könyvben. Ugyanitt kidolgozott példákat is találhatunk, feladatokkal együtt.

Párhuzamos technikák, párhuzamos algoritmusok

A párhuzamos számítások lehetősége az algoritmusok szervezésében sokkal nagyobb szabadságot biztosít, mint a szekvenciális megközelítés. Ezt az alábbi hasonlattal szemléltethetjük:

- a szekvenciális algoritmus egy dimenziós konstrukció, euklideszi térben;
- a párhuzamos algoritmus több (változó) dimenziós konstrukció, változó térben.

Megj. A második pont arra utal, hogy a párhuzamos program lehetséges végrehajtásait egy bonyolult nagy fával szemléltethetjük, amelynek egyes ágait a szinkronizáció miatt levágjuk, és a struktúra annál bonyolultabb, minél több processzor áll a rendelkezésünkre. Ráadásul a környezet alapvetően nemdeterminisztikus (részletesen ld. [Bu]).

Jelenleg még főleg szekvenciális gépeken dolgozunk, ezért a párhuzamos algoritmusok főként a szekvenciális algoritmusokban amúgy is meglevő párhuzamosítási lehetőségeket használják. A párhuzamosítás alapvetően négyféle módon képzelhető el:

a) triviális párhuzamosítási lehetőségek (nem feltétlenül triviális klasszikus algoritmusokban, pl. Gauss elimináció);
b) klasszikus algoritmusok alkalmas részeinek vagy egészének párhuzamos végrehajtással való felgyorsítása (pl. mátrix-vektor szorzás, aritmetikai kifejezések párhuzamos kiértékelése);
c) új elemek beépítése klasszikus algoritmusokba egyes részfeladatok párhuzamos megoldására (pl. aritmetikai kifejezések párhuzamos kiértékelése);
d) teljesen új elveken alapuló párhuzamos algoritmusok kifejlesztése.

Megj. Teljesen új elveken alapuló párhuzamos algoritmusokat egyszerűbb esetektől eltekintve nehéz kifejleszteni, és a közeljövőben nem várható, hogy ilyenek tömegesen megjelennek. De előfordulhat, hogy néhány évtized múlva ez lesz a számítástechnika fő fejlődési területe (további részletek: [Mo]).

Az előző alfejezetekben megnéztük, hogy milyen párhuzamosítási lehetőségek vannak a rekurzív illetve a dinamikus programozással megoldható algoritmusokban. Most néhány fontos matematikai probléma megoldásának párhuzamos felgyorsítását tekintjük át, a fenti a)-c) pontok módszerei szerint. A cikk végén bemutatunk néhány d) ponthoz tartozó egyszerűbb példát is.

A Gauss-elimináció

A Gauss-elimináció a lineáris egyenletrendszerek megoldására használt klasszikus módszerek egyike. Általános esetben kezdetben adott „n" darab ismeretlen és „m" darab egyenlet a következő elrendezésben:

$$\begin{aligned} a_{11}\cdot x_1 + a_{12}\cdot x_2 + \ldots + a_{1n}\cdot x_n &= b_1 \\ a_{21}\cdot x_1 + a_{22}\cdot x_2 + \ldots + a_{2n}\cdot x_n &= b_2 \\ a_{31}\cdot x_1 + a_{32}\cdot x_2 + \ldots + a_{3n}\cdot x_n &= b_3 \\ \vdots \qquad\quad \vdots \qquad\quad & \vdots \\ a_{m1}\cdot x_1 + a_{m2}\cdot x_2 + \ldots + a_{mn}\cdot x_n &= b_m \end{aligned}$$

A megoldás során az ismeretlenek fokozatos eliminálásával a teljes együtthatómátrixból felső háromszög mátrixot hozunk létre, amelyből aztán a megoldások leolvashatók, illetve visszahelyettesítéssel meghatározhatók. Az elimináció során megengedett műveletek a következők:

- valamely sort egy nem nulla konstanssal megszorozhatunk;
- valamely sorból kivonhatjuk egy másik sor konstansszorosát;
- (ha szükséges) átjelölhetjük a még nem eliminált ismeretleneket.

1. lépés: a_{i1}-et „lenullázzuk" $i < 1$-re, azaz az i-edik sorból kivonjuk az első sor a_{i1}/a_{11}-szeresét.

Ezzel az első sort követő minden sorban az első oszlop lenullázódik. A következő lépés során a második sort követő minden sor második oszlopa is lenullázódik, stb. Az elimináció végén „jó esetben" az első sorban „n" tag összege fog szerepelni, a másodikban „n-1", …, az utolsóban pedig csak egy ismeretlen lesz. A „jó eset" akkor fordul elő, ha a rendszeren belül ugyanannyi független egyenlet található, mint ahány ismeretlen. Egyéb esetben lehet, hogy nem kapunk megoldást (ellentmondásos egyenletrendszer), illetve végtelen megoldás is lehet (egy vagy több ismeretlen értéke szabadon megválasztható). Ezen esetek pontos meghatározásával most nem foglalkozunk, ez bármely egyetemi/főiskolai lineáris algebrai kurzus anyagában szerepel.

A lépésszám és a műveletigény négyzetes mátrixra hagyományosan Ordó(n^3). Ha van n darab processzorunk, és ki tudjuk köztük osztani a részfeladatokat úgy, hogy párhuzamosan dolgozzanak (például 1-1 sort egymástól függetlenül számoljanak végig), akkor a lépésszám Ordó(n^2) lesz (ugyanakkor a műveletigény továbbra is Ordó(n^3) marad). Tovább gyorsítható az eljárás n^2 processzorral, ekkor minden mátrixelemhez rendelünk egy processzort, és az azzal dolgozik. Így a lépésszám Ordó(n) lesz. Megjegyzendő, hogy a gyakorlatban a processzorok közötti kommunikációra fordított idő a fenti elméleti határokhoz képest jelentősen lelassítaná a feladat megoldását.

F: Gyorsítható-e a végrehajtási idő újabb processzorok rendszerbe állításával? (1,5)

Mo: A probléma részeredményei tovább már nem függetleníthetők egymástól, tehát ha ennél több processzort használunk, akkor már nem érhető el sebességnövekedés.

Megj. Néhány fontos gyakorlati problémával nem foglalkoztunk, amelyek szintén komoly lassulást okozhatnak a megoldás során. Ilyen például az eliminációra sajátosan jellemező ún. együttható-növekedési probléma: az egymás utáni lépésekben egyre több jegyű számokkal (gyakran egyre hosszabb törtekkel) kell

dolgozni. Ennek részbeni kivédésére javasolhatnánk a közelítő számításokat, de ez sem oldja meg a gondot: eleve elveszítjük a pontos megoldás lehetőségét, ráadásul a közelítések hibái a lépések során összegződnek, így rossz esetben nem is kapunk megoldást a számolás végén! Csak ügyes egyszerűsítések alkalmazásával háríthatók el a fenti problémák általános esetben (részletek: [Ge]).

Polinom-faktorizáció

A polinomok szorzattá alakítása általános esetben bonyolult probléma.

Példa: Bontsuk fel $A(x, y) = y^2 + 2xy - 3y + x^2 - 3x - 4$-et

Erre a feladatra első közelítésben nem látszik használható megoldási módszer.

Az alfejezetben feltételezzük, hogy az olvasó tisztában van a többváltozós polinomokkal kapcsolatos fontosabb fogalmakkal. A szakszerű megalapozáshoz a következő ismeretek, definíciók szükségesek (csak felsorolásszerűen bemutatva):

Z: egész számok gyűrűje (+-ra és *-ra);
Q: racionális számok teste (+-ra és *-ra);
Z[x]: polinomgyűrű;
Z[x, y]: értelmezhető mint Z[x][y] vagy Z[y][x] polinomgyűrű;
Z[x, y, z] hasonlóan, rekurzív módon értelmezhető;
$Z_p[x] = Z[x]$ mod p test, ha p prím.
A fenti tartományokban van egyértelmű faktorizáció és egyértelműen létezik a legnagyobb közös osztó is.

A következő ötletet használjuk:

a) Az eredeti problémát redukáljuk egy sokkal egyszerűbben kezelhető tartományba, és ott oldjuk meg. Ez az egyszerűsítés lehet például a következő:

$$Z[x, y, z, ..., w] \rightarrow Z[x] \rightarrow Z_p[x].$$

b) Ezután megoldjuk a problémát $Z_p[x]$-ben, ahol egy polinom már viszonylag egyszerűen szorzattá alakítható.

c) Utolsó lépésben a kapott megoldást visszaképezzük az eredeti tartományba:

$$Z_p[x] \rightarrow Z[x] \rightarrow Z[x, y, z, ..., w]$$

Példa folyt. $A(x, y)$ Z[x]-ben $x^2 - 3x - 4$ lesz, $Z_3[x]$-ben pedig $x^2 – 1$.
Ez a polinom már egyszerűen szorzattá alakítható. A felbontások:

$Z_3[x]$-ben $(x + 1)(x - 1)$;

Z[x]-ben pedig $(x + 1)(x - 4)$.

Ez a részeredmény már csak abban tér el az eredeti felbontástól, hogy az y-os tagok hiányoznak. Ez visszaképezéssel megkapható. A teljes megoldás Z[x, y]-ban:

$$A(x, y) = (x + 1 + y)(x - 4 + y).$$

A megoldandó részfeladatok elemzése

a) Egyszerűsítés. Ez könnyen végrehajtható rész, adott esetben párhuzamosítható is.

b) A probléma megoldása $Z_p[x]$-ben vagy Z[x]-ben.

Bár a polinom szorzattá alakítása nyilván sokkal egyszerűbb probléma valamely redukált tartományban, intuitív módon (mint a fenti egyszerű példában) általában még most sem lehet célhoz érni. A szorzattá alakító algoritmusok túlnyomó többsége a legnagyobb közös osztó (lnko) meghatározásán alapul. Egy lehetséges ötlet a következő:

lnko(p, p') az eredeti polinom egy valódi osztója, ahol p' a derivált polinom.

Példa: Legyen $p = A \cdot B^2 \cdot C^3$, ahol A, B és C tovább már nem bontható faktorok.
ekkor $p' = A' \cdot B^2 \cdot C^3 + A \cdot 2B \cdot B' \cdot C^3 + A \cdot B^2 \cdot 3C^2 \cdot C'$

(de például $Z_3[x]$-ben csak $p' = A' \cdot B^2 \cdot C^3 + A \cdot 2B \cdot B' \cdot C^3$, ez egyszerűbb)
$lnko(p, p') = B \cdot C^2$, ez egy valódi osztó

F: Mikor nem használható ez az ötlet? (2)

Más felbontó eljárások is ismeretesek. Bonyolult (magas fokú) polinom esetén célszerű lehet (egyszerre) több eljárást is kipróbálni, hogy hamarabb célhoz érjünk. Mivel lényegében az összes felbontó eljárás használ lnko számításokat, ezért szükségünk van ezek hatékony kivitelezésére. Erre az euklideszi algoritmust használjuk, amely $Z_p[x]$-ben és $Z[x]$-ben egyaránt végrehajtható.

Az euklideszi algoritmus a maradékos osztás ismételt alkalmazásával határozza meg lnko(a, b)-t.

A maradékos osztás adott a, b elemekre a következő módon hajtható végre:

$a = b \cdot q + r$, ahol $|r| < |b|$ vagy $r = 0$ (1) (polinomokra a |...| norma a polinom foka)

áll: $lnko(a, b) = lnko(b, r)$ (ennek igazolása nyilvánvaló)

Ezután választhatjuk: új a := b; új b := r; majd újra végrehajtjuk az (1) lépést. Az utolsó nem 0 maradék az eredeti számok lnko-ja.

c) A megoldás visszaképezése az eredeti többváltozós polinomgyűrűbe.

Megjegyezzük, hogy ez a legnehezebb lépés, itt csak a legfontosabb ötleteket tekintjük át.

Nyilvánvaló, hogy egy $Z_p[x]$ vagy $Z[x]$-beli kép sokkal kevesebb információt hordoz, mint ami az eredeti felbontás előállításához kell, ezért további adatokra van szükségünk a feladat megoldásához.

Két megközelítés használható:

(i) sok $Z_p[x]$-beli képet állítunk elő (több különböző p-re), vagy több $Z[x]$-belit;

(ii) csak egy képet használunk, de van még valami plusz információnk.

(i) eset

1. Példa: Tudjuk a Z[x, y]-beli u-ról hogy

$u(x, 0) = - 9x – 21$,

$u(x, 1) = - 3x + 20$,

$u(x, 2) = 5x – 36$.

Ez alapján polinom interpolációval

$u(x, y) = (- 9x - 21) + (6x + 41)(y - 0) + x(y - 0)(y - 1) = xy^2 + 5xy + 41y - 9x - 21$

2. Példa: Tudjuk valamely u számról, hogy

$u = 2 \pmod 5$ és $u = 3 \pmod 7$.

Mennyi u valójában?

Mo: nyilván

$5x + 2$ alakú (lehet: 2, 7, 12, 17, 22, 27, 32, ...) és

$7x + 3$ alakú (lehet: 3, 10, 17, 24, 31, 38, 45, 52, ...)

egyszerre, így lehet 17, 52, ..., de mod 35 egyértelmű a megoldás.

A példák alapján láthatjuk, és általánosan is igazolható, hogy

$Z[x, y] \rightarrow Z[x]$ invertálható polinom interpolációval (y foka + 1 kép kell);

$Z[x] \rightarrow Z_p[x]$ invertálható kínai maradék algoritmussal.

Gondot jelent azonban, hogy összességében nagyon sok kép kell (végeredményben exponenciális), és nem jól párhuzamosítható a folyamat.

(ii) eset

A szükséges plusz információ lehet például a következő:

$F(v, u) = a(x, y, w, ..., z) - v \cdot u$; (2) (két faktorra vágtuk a-t)

és

$u(x, y, w, ..., z) = u_0(x)$; (3) ($Z_p$ [x]-ben)

$v(x, y, w, ..., z) = v_0(x)$. (4) ($Z_p$ [x]-ben)

Ezután $u_0(x)$-et és $v_0(x)$-et felemeljük fokozatosan

Z[x]-be, Z[x, y]-ba, Z[x, y, w]-be, ..., Z[x, y, w, ..., z]-be

úgy, hogy (2)-(4) érvényes maradjon.

A faktorok felemelése egymástól lényegében független (egy szinten belül), így a módszer jól párhuzamosítható. Még többet nyerhetünk egy teljes felbontással:

$F(u_1, u_2, ..., u_n) = a(x, y, w, ..., z) - u_1 \cdot u_2 \cdot ... \cdot u_n$ (5)

– ahol (2)-(4) megfelelői érvényesek – majd ennek a felemelésével. Az (i) és az (ii) esetekben használható részletes algoritmusok (pontos elméleti igazolással együtt) megtalálhatók a [Ge] könyvben.

F: Írjunk olyan programot, amely szorzattá alakít általános többváltozós polinomot (reális fokkorláttal). Ajánlott irodalom: [Ge].

Összefoglalva, a polinomok szorzattá alakítása bonyolult, komoly számításigényű probléma. Mivel ezen számítások jelentős része párhuzamosan is végrehajtható, többprocesszoros gépen a végrehajtás jelentős felgyorsulását érhetjük el.

PRAM algoritmusok

Ebben az alfejezetben néhány egyszerűbb párhuzamos algoritmust mutatunk be, amelyeket PRAM gépekre dolgoztak ki. Az alapvető számítástechnikai algoritmusok közül sok tömbökön, listákon, fákon és gráfokon dolgozó szekvenciális algoritmus felgyorsítható a PRAM modell segítségével.

A PRAM modellben a lehetséges algoritmus-típusok a következő módon csoportosíthatók:
(i) EREW: kizárásos olvasás és írás (exclusive read and exc. write)
(ii) CREW: egyidejű olvasás és kizárásos írás (concurrent read and exc. write)
(iii) ERCW: kizárásos olvasás és egyidejű írás (exclusive read and conc. write)
(iv) CRCW: egyidejű olvasás és írás (concurrent read and conc. write)
változat: P-CRCW ill. P-ERCW (prioritásos modell).

A prioritásos modellben egyidejű írás esetén valamilyen stratégia szerint ki kell választani a „győztest” (vagy győzteseket), aki(k)nek az akarata érvényesül. Erre a következő módszereket használhatjuk:

- a legkisebb sorszámú dönt;
- a közös akarat érvényesül (csak akkor van eredmény, ha ugyanazt írják);
- véletlenszerűen választunk;
- adott képlet alapján döntünk (pl. összeadjuk az eredményeket vagy azok átlagát, maximumát vesszük).

A modellek közül az EREW és CRCW a leggyakoribb.

Példa: Határozzuk meg n darab processzorral, hogy két n hosszú 0-1 sorozat közül melyik a nagyobb!

P-CRCW modellben elég O(1) lépés:

A processzorok egyidejűleg sorszámot írnak, hogy melyik sorozat a nagyobb, a legkisebb helyen levő eltérés dönt.

F: EREW modellben elég O(log n) lépés.

A példában láthattuk, hogy a modellek közötti időkülönbség nem „túl nagy”. Ez általánosan is igazolható:

Áll. Ha egy probléma megoldható P-CRCW modellben p processzorral t idő alatt, akkor megoldható EREW modellben $O(p^2)$ processzorral $O(t \cdot \log p^2)$ idő alatt.

Határozzuk meg egy lista elemeinek a lista végétől való távolságát!

Nem párhuzamos megoldás esetén O(n) lépés szükséges, minden elem a következőtől megkapja a partner távolságadatát, a sajátja ennél eggyel nagyobb lesz.

A párhuzamos megoldásnál n darab processzort használunk, minden elemhez pontosan egyet rendelünk. Így EREW modellben egy Ordó(log n)-es megoldást állíthatunk elő (részletek és további példák: [Co]).

Irodalom

[Bu] ALAN BURNS, GEOFF DAVIES, *Concurrent Programming,* Addison-Wesley Publishing Company, 1994.

[Co] THOMAS H. CORMEN, CHARLES E. LEISERSON, RONALD L. RIVEST, *Algoritmusok,* Műszaki Könyvkiadó, Budapest, 1998.

[Ge] KEITH O. GEDDES, STEPHEN R. CZAPOR, GEORGE LABAHN, *Algorithms for Computer Algebra,* Kluwer Academic Publishers, 1991.

[Mo] JAGDISH MODI, *Paralell Algorithms,* Oxford University Press, 1986.

Mobil eszközök objektumorientált programozása, *a Java2 Micro Edition*

Object-oriented Programming Language for Mobile Devices – *J2ME*

VARJASI Norbert

Széchenyi István Egyetem, Győr
Számítástechnika Tanszék

Abstract

The spreading of Java enabled devices in everyday life and the effective object-oriented softwares written for these devices can mean new directions and possibilities in IT training in the future. In my presentation I am going to give an overview of the structure of Java Micro Edition, which has been created for the development of mobile applications, and the application models based on it.

Keywords: MIDlets, wireless programming, MIDP programming

Összefoglaló

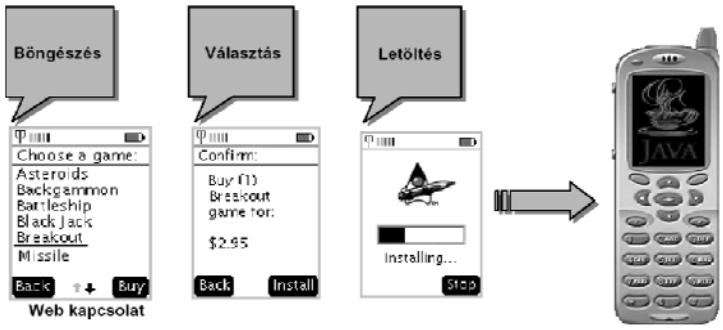
A Java-képes eszközök mindennapos elterjedése és ezen eszközökre írt hatékony objektum-orientált programok (a továbbiakban mobil-oo, MIDlet) az informatikus-képzés új irányait és lehetőségeit jelenthetik a jövőben. Az alábbiakban bemutatom a mobil alkalmazások fejlesztésére megalkotott Java Micro Edition felépítését, az erre épülő alkalmazás-modelleket.

Kulcsszavak: MIDletek, vezetéknélküli fejlesztés, mobilprogramozás, MIDP

A mobil eszközök

Napjainkban az elterjedt mobil szolgáltatások és lehetőségek többféleképpen csoportosíthatók. Egy lehetséges összeállítás szerint a mobil eszközök az alábbi területeken érik el a szolgáltatásokat:

- mobil kommunikáció (email, SMS stb.)
- mobil kereskedelem (aukciók, bank, utazás)
- mobil információtartalom (helyfüggő szolgáltatások, hírek, időjárás)
- mobil szórakozás (zene, kép, játék)
- mobil vállalati szolgáltatások



1. ábra
Mobil szolgáltatás igénybevétele

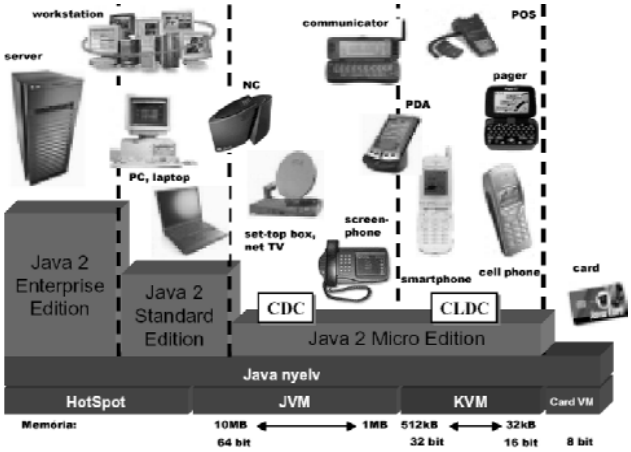
A Java2 Micro Edition fejlődése és architektúrája

A mobiltelefonok jelenlegi fejlődési ütemét felismerve a Sun Microsystems 2000 nyarán bejelentette a MIDP (Mobil Information Device Profile) 1.0-s verzióját [i]. Ennek keretében elsőként fogalmazott meg egy szabványos, platformfüggetlen és skálázható mobiltelefon szabványt. Ezzel sikerült a fejlesztők és gyártók érdeklődését visszaterelni a Java megoldások felé, és szinte teljes mértékben meghódította a piac ezen szegmensét, hiszen mára már többmillió készülékbe implementálták a Java szoftvert. Az első verzió megjelenésével eleinte csak egyszerűbb alkalmazásokat, játékokat és segédprogramokat fejlesztettek (mint a számológépek, e-mail olvasók, a tetris, az aknakereső stb.), de fellelhetők komolyabb algoritmusokat megvalósító útvonaltervező, tőzsdei elemző és MP3 lejátszó programok is.

A MIDP 2.0 fejlesztésébe már egyre több mobilgyártó cég bekapcsolódott [ii], megalakítva a Java Community Processt (JCP). Így 2002 novemberére, a 2.0-s verzió bejelentésekor már 49 gyártó cég volt tagja a JCP-nek. A hardver és szoftver fejlesztések összehangolásával a mobil eszközök grafikai és audió képességei maximálisan kihasználhatóvá váltak, és sokat fejlődött az alkalmazások biztonsága is.

A Java2 ME fejlődése a konkurens termékekkel szemben a nyílt forráskódnak, a szabványosított fejlesztői felületnek köszönhető, mert az egyedi készülékek egyedi megoldásaival szemben szabványos és hordozható alkalmazások fejleszthetők, hiszen a Java alapelvei szerint a megírt szoftver bármilyen készüléken futtatható, függetlenül attól, hogy milyen operációs rendszer és milyen processzor működik alatta.

Maga a Java2 ME nem önálló szoftver, hanem a „kisméretű elektronikai cikkek” piacára tervezett technológiák és szabványok gyűjteménye. A platform magját a központi Java könyvtárak és jelenleg két – különböző termékekhez fejlesztett – konfiguráció adja.



2. ábra
A Java2 kiadásai és a céleszközök

A Java2ME rétegei

3. ábra
A J2ME szabvány logikai szerkezete.
Az eszköz operációs rendszere felett futó virtuális gép (VM)
az erre épülő konfiguráció (CDC/CLDC), majd a profil réteg (MIDP/PDAP)

A konfigurációs réteg

Ezek a Connected Device Configuration (CDC) és a Connected Limited Device Configuration (CLDC) [iii]. A CDC-t a nagyteljesítményű hordozható készülékekhez (kommunikátorok, Net TV dobozok), míg a CLDC-t a kisebb teljesítményű mobiltelefonokhoz, személyhívókhoz tervezték. A CDC a hagyományos Java Virtuális gépet használja, míg a CLDC a K Virtuális gépet, melynek neve arra utal, hogy a mobil eszközökben a rendelkezésre álló erőforrások néhány 10 kbájtra korlátozódnak. A KVM olyan 16/32 bites RISC/CISC processzorokkal működik együtt, melyek 128 kbájtban tárolják a virtuális gépet az osztálykönyvtárakkal és további 32 kbájt áll a futásidejű adatok rendelkezésére.

A felhasználók számára ez a réteg nem látható, a profil réteg implementálásában kap szerepet, összességében meghatározza a virtuális gép minimális tulajdonságait és az elérhető java osztálykönyvtárakat.

A profil réteg

A konfigurációkon alapulnak, és ezekre épülnek az egyes készülék-kategóriákat meghatározó profilok. Jelen pillanatban a CLDC-re épülő MIDP van használatban, de már fejlesztés alatt áll a PDA Profil is.

Összegezve, ez a „látható" réteg, az API olyan minimális halmaza, amely elérhető az adott eszközcsalád részére, és ez a réteg biztosítja a hordozhatóságot az adott profilt támogató eszközök között.

A program-elnevezési tradíciók szerint a CLDC-t és MIDP-t használó Java alkalmazásokat MIDleteknek nevezik. A MIDletek céleszközei tehát olyan mobil eszközök, amelyek minimum 96 x 54 pixeles kijelzővel, billentyűzettel vagy érintőképernyővel, vezetéknélküli összeköttetéssel és minimum 160 Kbájt memóriával rendelkeznek. A MIDletek „jar" fájlokba csomagolhatók és szabványos leírófájlok vezérlik a helyes működésben. A mobil eszközre feltöltött MIDletek mindegyike rendelkezik egy-egy startApp(), pauseApp() és destroyApp() metódussal, melyek az alkalmazás életciklusát vezérlik.

Alkalmazásfejlesztés

A szakmai fórumokon és a fiatal, már a mobil-kultúrában nevelkedett generációkban élénk az érdeklődés a mobil programozás és a MIDletek iránt.

Az ingyenesen beszerezhető Java vezetéknélküli fejlesztőcsomag (Java Wireless Toolkit [iv]), és a hozzá letölthető emulátorok [v] segítségével olyan alkalmazások készíthetők, amelyek:

- magas absztrakciós szintet képviselnek,
- eseményvezéreltek,
- előterében áll a funkcionalitás,
- hatékony algoritmusokra ösztönöz,
- figyelmes és pontos tervezést igényelnek,
- alkalmasak hálózati, multimédiás működésre,
- nagyfokú biztonsággal ruházhatók fel,
- hordozhatók az egyes eszközök között.

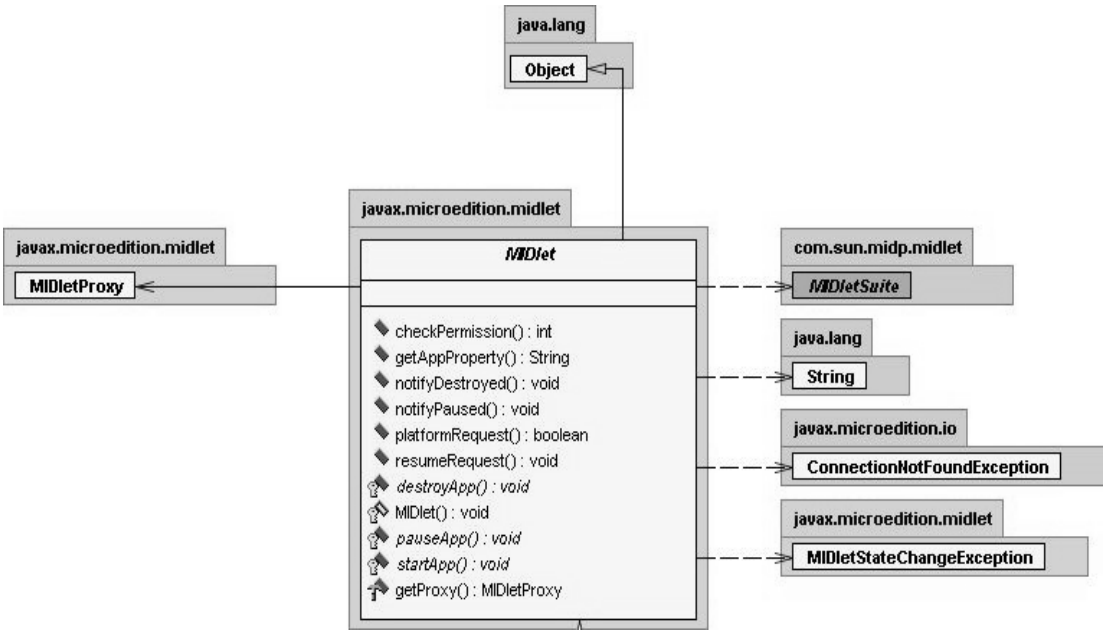
A fejlesztésben felhasználható szabványos osztályok a `javax.microedition.midlet, lcdui, io, media, rms` és a `javax.wireless.messaging` csomagokban kaptak helyet.

Az egyes mobil eszközökhöz letölthetőek a fejlesztőkörnyezethez írt emulátorok, amelyek tartalmazzák a hardverspecifikus osztálykönyvtárakat is.

4. ábra
A Sun vezetéknélküli fejlesztőcsomagja és emulátorok

A használt osztályok

A MIDP profil alaposztálya a MIDlet. Az emulátorban illetve a mobil eszközökben futó Java kódok belépési pontja a MIDlet osztály konstruktora.



5. ábra
A MIDlet osztály

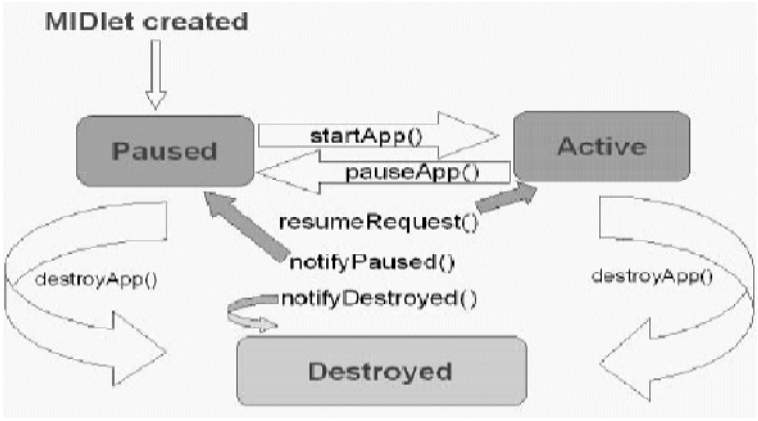
A csomag további osztályai definiálják a megjelenítést és az eseménykezelést megvalósító osztályokat. Ezek közül a legfontosabbak az alábbiak:

`Command` — eseménykezelés információi,
`Display` — képernyő és billentyűzet magas szintű kezelése, vezérlése,
`Screen` — absztrakt osztály, minden megjelenő elem őse,
`Canvas` — alacsonyszintű grafika és eseményvezérlés,

`Graphics`	—	2D ábrák osztálya,
`Image`	—	képek, grafika kezelése,
`Form`	—	adatelemek együttes megjelenése, kezelése,
`Alert`	—	egyszerű információkijelző ablak,
`ChoiceGroup`	—	választható elemek csoportja (egyszeres, többszörös) List, TextBox, Font, Gauge, Ticker stb.

A fejlesztés során figyelemmel kell kísérni, hogy a mobil eszközök csak korlátozott hibakezelést végeznek. Ezért már a fordítás során egy „preverifier" ellenőrzi a kód integritását és az erőforrás használatot. Az elkészített alkalmazás feltöltésekor a céleszköz a lefordított kódot ellenőrzi, („verifier") majd letárolja. Amennyiben az eszköz nem tudja az adott kódot értelmezni, abban az esetben nem engedi a mobil eszköz interpreterének sem futtatni azt.

Egy MIDlet futása során csak meghatározott állapotokban állhat. Ezen állapotokat szintén a fent említett MIDlet osztály definiálja.



6. ábra
Egy MIDlet életciklusa[2]

Összegzés

A XXI. század mindennapos eszközeivé váltak a mobiltelefonok, és a felnövekvő generációk érdeklődéssel fordulnak nemcsak használata, hanem a fejlesztések felé is. A hatékonyabbá és olcsóbbá váló készülékek elterjedésével a mobil eszközök terén otthonosan mozgó szakemberekre lesz szükség. A könnyen átlátató és elsajátítható osztályhierarchia, a rendelkezésre álló szoftverkörnyezet új irányokat és lehetőségeket nyújt az objektumorientált technológiák elsajátítása és a fejlesztések előtt. Az erőforrások korlátozottsága nem jelent hátrányt, mert a kitűzött feladatok és célok csak hatékony és kompakt megoldásokkal oldhatóak meg.

A mobil eszközök szerepe napról-napra nyílik meg az újabb lehetőségek, mint a kliens-szerver kommunikáció, a nagy sávszélességű és megbízható kapcsolatok, az on-line szolgáltatások előtt, és a piaci érdeklődés ebben a szektorban nagyon jelentős.

Irodalom

[i] J2ME White Paper on KVM and the Connected, Limited Device Configuration(CLDC), Sun Microsystems, Inc. May 19.2000.

[ii] Mobile Information Device Profile, http://java.sun.com/products/midp/

[iii] J2ME Step by Step http://www.ibm.com/developerWorks/

[iv] Java2 Platform Micro Edition, Wireless Toolkit, http://java.sun.com/products/j2mewtoolkit

[v] Nokia Mobile Toolset (http://www.forum.nokia.com/)

Drótnélküli hálózatok és szolgáltatások minősége (QoS)

Wireless Networks – Quality of Service Issues

Dr. SEBESTYÉN György
Kolozsvári Műszaki Egyetem
Számítástechnika és Automatizálás Kar

Abstract

The significant technological advances achieved in the last years, in the field of wireless communications, opens new design opportunities and also new challenges for application developers. Wireless networks assure greater mobility and flexibility, but in the same time they require solutions for some new issues, which are not present in legacy LANs. The efficient use of limited bandwidth and the control of services' quality are two of the issues that need better solutions. This paper presents an overview on the wireless technologies used today, emphasizing the possibilities and limitations. The paper analyzes different methods used to control and guarantee the quality of services (QoS) in wireless networks.

Kivonat

Az utóbbi években a drótnélküli kommunikációs technológiák terén tapasztalt, robbanásszerű fejlődés új lehetőségeket nyújt, de ugyanakkor új kihívásokat is jelent a fejlesztők számára. A drótnélküli hálózatok nagyobb mobilitást és flexibilitást biztosítanak, viszont olyan feladatok megoldását is követelik, amelyek nem jellemzők a hagyományos helyi hálózatokra. Ezek közé tartozik a kommunikációs médium hatékonyabb kihasználása vagy a szolgáltatások minőségének ellenőrzése és garantálása. A dolgozat átfogó képet nyújt a drótnélküli hálózatok mai helyzetéről, és elemzi a jelenleg használt protokollok lehetőségeit és korlátait. A tanulmány főleg a szolgáltatások minőség-ellenőrzési feladataira fektet hangsúlyt.

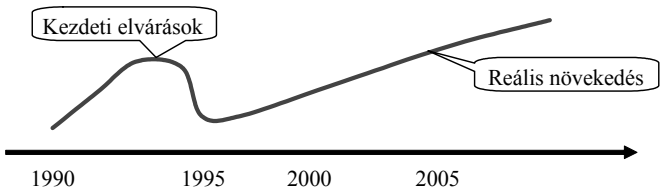
1. Bevezető

Ha a történelmi adatokra tekintünk, azt tapasztaljuk, hogy már a 19-ik század végén (pontosabban 1896-ban) Guglielmo Marconi bebizonyította a drótnélküli kommunikáció elvi lehetőségét, amikor rádióhullámok által megvalósított egy telegráf jellegű kapcsolatot. Az első években a fejlődés kis lépésekben történt, mivel hiányzott a szükséges elektronikus technológiai háttér. A két világháború felgyorsította a fejlesztést, mivel nagyobb mobilitásra és gyorsabb kapcsolatteremtésre volt szükség. Kezdetben a rádió- és a későbbi tévéadások ugyancsak drótnélküli csatornán működtek.

A 60-as évek elején *bocsátották* fel az első telekommunikációs műholdakat, amelyek interkontinentális kapcsolatot és információközvetítést tettek lehetővé. A műholdas rádiókapcsolat áthidalta a hatalmas távolságokat, és kapcsolatot teremtett a drótos kapcsolatra kevésbé alkalmas területek között is.

A 90-es évek elején robbanásszerűen fejlődött a mobil telefonhálózat, elsőként Amerikában és mindjárt azután a többi kontinensen is. Ma már több mint egy milliárd felhasználót szolgálnak ki ezek a hálózatok.

A drótnélküli számítógépes hálózatok főleg az utóbbi 4-5 évben kerültek előtérbe, egyrészt a hordozható számítógépes eszközök (laptop-ok, PDA-k, intelligens telefonok, stb.) fejlődésének köszönhetően, másrészt a fokozott mobilitást igénylő mai társadalom kielégítése érdekében. A 90-es évek nagy reményei után ma már egy egyenletes fejlődési trendet tapasztalunk. Az 1-es ábra tükrözi ezt a fejlődést, ami máskülönben a legtöbb új technológiára jellemző.

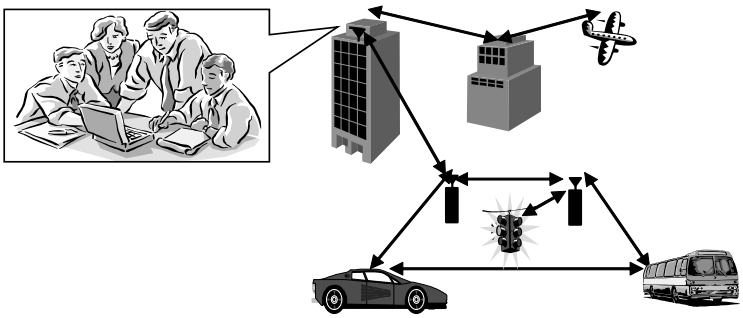


1. ábra
A drótnélküli hálózatok fejlődése

Ma a drótnélküli kommunikáció fő célja az emberek és gépek közti kötetlen (ang. ubiquitous) kapcsolat megvalósítása, a „*kapcsolat bárhol és bármikor*” elvnek megfelelően. Előnyei közé tartoznak a következők:

- nincs szükség fizikai infrastruktúrára
- ideiglenes (ad-hoc) kapcsolatok vagy akár hálózatok könnyű és aránylag olcsó megvalósítása
- kommunikálási kompatibilitás különböző eszközök között (laptop, PDA, telefon, orvosi készülékek, stb)
- mobil számítástechnikai eszközök és mozgásban levő felhasználók Internethez való bekapcsolása

Vannak viszont bizonyos hátrányok is, mint például: a korlátolt sávszélesség, az alacsonyabb megbízhatósági szint vagy sok esetben a térbeli korlátozottság. A 2-es ábra a különböző drótnélküli kapcsolattal megvalósítható kommunikálási lehetőségeket tükrözi.



2. ábra
Drótnélküli kapcsolatok

2. Tervezési és megvalósítási feladatok

A rendelkezésre álló korlátozott sávszélesség és az ebből következő alacsony logikai csatornaszám a legkritikusabb tervezési gondokhoz tartoznak. A hálózati kommunikációra csak bizonyos szűk sávok állnak rendelkezésre; ezek közül egyes sávokat más célokra is alkalmaznak (pld. ipari, vagy háztartási célokra). Az alkalmazott kommunikációs protokollok különbözőképpen próbálják minél hatékonyabban kihasználni a rendelkezésre álló sávszélességet.

Az állandóan változó hálózatkonfiguráció (gépek, felhasználók, adathozam változások) újabb tervezési komplexitást okoz. A kommunikációs protokoll automatikusan fel kell ismerje az ideiglenes hálózatban résztvevő aktív szereplőket és az igényeiknek megfelelő sávszélességet kell, hogy biztosítson.

Azt is szem előtt kell tartani, hogy sok mobil eszköz korlátozott energiaforrással rendelkezik, ami megszabja a maximális közvetítési távolságot. Ugyanakkor egyes mobil eszközök adatfeldolgozási sebessége és tárolási kapacitása ugyancsak korlátozott, ami meggátolja a bonyolult protokollok beépítését.

A nyitott közvetítési környezet, amelyben a drótnélküli kommunikáció történik, biztonsági gondokat okoz; egyrészt biztosítani kell a hibamentes közvetítést és ugyanakkor védeni kell a hálózatot idegen beavatkozás ellen. Ismert tény, hogy a drótnélküli hálózatokban sokkal nagyobb a hiba-megjelenési gyakorisága mint a dróton vagy optikai szálon alapuló hálózatokban. A hibás üzenetek újraközvetítése csökkenti az úgyis keskeny gyakorlati sávszélességet. Ezért nagyon fontos olyan minőség-biztosító eljárásokat beépíteni a protokollba, amelyek egy előre megszabott minőségi szintet tudnak garantálni.

3. Drótnélküli rendszerek

A drótnélküli rendszerek kategóriájába a következő rendszerek sorolhatók:

- a sejt- (mobil-) rendszerek (ang. Cell Systems)
- a drótnélküli helyi-hálózatok (ang. WLAN Wireless LAN)
- műholdas rendszerek (ang. Satelite Systems)
- rövid üzeneteket közvetítő rendszerek (ang. Paging Systems)
- rövidtávú rádiókapcsolatok (Bluetooth)

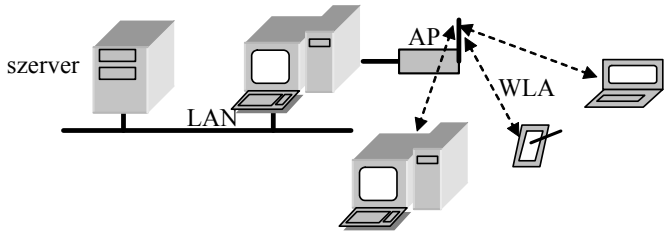
A sejtrendszereket főleg telefonkapcsolatokra alkalmazzák. Ugyanakkor viszont az aránylag új GPS és GPRS protokollok lehetővé teszik a digitális adatok közvetítését és a mobil eszközök révén az Internet hálózatba való gyors bekapcsolását. A sejthálózati technológia több változáson ment keresztül: az első generáció (1G) analóg eszközökre és sávmegosztásra alapszik; a második generáció (2G) digitális alapokra épül és sávszélesség-, idő- és kódmegosztást alkalmaz a csatornák meghatározásához; a 3-dik generáció (3G) egyszerre áramkör- és csomagkapcsolási technikákat alkalmaz, ami azt mutatja, hogy alkalmas mind telefon-, mind számítógépek közti kapcsolatra.

A műholdas közvetítés kevésbé függ össze technikai szempontból a számítógépekközti kommunikációval. Ezért a továbbiakban főleg a WLAN és a Bluetooth hálózatokról lesz szó.

A drótnélküli helyi-hálózatok (WLAN) a LAN típusú hálózatok feladatait próbálják helyettesíteni olyan esetekben, amikor a hagyományos közvetítési eszközök (rézdrót, optikai szál) alkalmazása nem lehetséges vagy nem gazdaságos. A leggyakrabban előforduló esetek, amelyekben WLAN technológiát alkalmazunk, a következők:

- helyi-hálózatok bővítése
- épületek közti kapcsolat megteremtése
- mobil eszközök hálózatba való bekötése
- „Ad-hoc” (alkalmi) hálózatok megvalósítása

Az első esetben a hagyományos LAN alkotja a hálózat gerincét, amelyhez hozzáférési pontok által (ang. AP - Access Points), WLAN interfésszel rendelkező számítógépek kapcsolhatók. A kapcsolat lehet ideiglenes, mint például a laptopok vagy PDA-k esetében vagy állandó, az irodai gépek (PC-k) esetében. A 3-as ábra tükrözi ezt az esetet.



3. ábra
LAN bővítés WLAN kapcsolatokkal

Két épület között drótnélküli kapcsolatot alkalmazunk abban az esetben, amikor hagyományos kapcsolatot bizonyos okok miatt (pl. adminisztratív szabályzatok) nem lehet vagy nagyon költséges létrehozni. Általában ezek a kapcsolatok ideiglenek.

A mobil, vagy „nomád” kapcsolatokat iskolákban, könyvtárakban vagy más információs központokban alkalmazhatjuk. Jellemző az ilyen esetben egyrészt az azonnali hozzáférési igény, másrészt a kapcsolat ideiglenes jellege.

Az „Ad-hoc” hálózatok olyan esetben jönnek létre, amikor egy megbeszélés alatt a résztvevők laptopjaik által operatív információt cserélnek egymás között anélkül, hogy egy hagyományos LAN hálózatba bekötnék őket. Ebben az esetben hiányzik a hozzáférési pont (AP) koordináló szerepe.

Minden esetben különféle stratégiát kell alkalmazni a mobil eszközök hálózatba való bekapcsolásához. Különbözik a mobil eszközök felismerése és nyilvántartása, az adatok közvetítése (pl. központi vagy elosztott kontroll) és a biztonsági eljárások. Ugyanakkor viszont a felhasználók számára fontos, hogy ugyanazt az interfészt több fajta hálózatban lehessen használni, kompatibilitási gondok nélkül.

4. WLAN hálózatok

A közvetítési médium alapján a WLAN hálózatok három csoportra oszthatók: infravörös sugaras LAN-ok (IR-LAN), szórt spektrumú LAN-ok és szűksávú mikro-hullámos LAN-ok. Az infravörös sugaras közvetítés lehet irányított sugaras vagy mindenirányú sugaras; a hálózat egy szobában levő eszközöket kapcsol össze azzal a feltétellel, hogy adóvevők egyenes vonalban lássák egymást. Gyakran a plafonra helyeznek el egy központi adóvevő készüléket, amely közvetítőként dolgozik. A különböző hőforrások (pl. napsugár, kályha) befolyásolhatják a közvetítés minőségét.

A szórt spektrumú LAN bizonyos hullámhossz-tartományban levő rádióhullámokat (5GHz, 2,5GHz) alkalmaz közvetítésre. Az adatok kódolásához egy különleges modulációs eljárást alkalmaznak (ang. DSSS - Direct Sequence Spread Spectrum), amely magas zajszint mellett is biztosítja a hibamentes adatközvetítést. Ahhoz, hogy egy korlátolt sávszélességben több logikai csatornát biztosítson, a protokoll az úgynevezett frekvenciaugrás-módszert alkalmazza (ang. FHSS -Frequency Hopping Spread Spectrum).

A hordozó jel pontos időközönként, látszólag véletlenszerűen frekvenciát vált; az adó és a vevő egy előre meghatározott terv szerint, egyszerre vált frekvenciát.

Ez a módszer megvédi a hálózatot idegen beavatkozások ellen.

A szabványosítás szempontjából a különböző változatú drótnélküli hálózatokat az IEEE 802.11x vagy az IEEE 802.15 (Bluetooth) szabvány írja le. Ezek a változatok a sebesség, a hordozójel frekvencia-tartomány, az alkalmazott kódolási módszer vagy a hálózat-hozzáférési technika szempontjából különböznek. Az IEEE 802.11a 5GHz-en működik és 6-54MBps adatsávot biztosít; az IEEE 802.11b jelenleg a leghasználtabb protokoll, a 2,5 GHz-es sávot használja és gyakorlatilag 5Mbps adatátvitelt biztosít. Az IEEE 802.11g változat eléri az „a" változat sebességét a „b" változat frekvenciasávjában. A Bluetooth elnevezésű szabvány lényegesen különbözik az előbbiektől, mivel más célokat szolgál. Ez a hálózat a különböző periferikus eszközök számítógéphez való kapcsolását biztosítja. Ezért az alkalmazott hálózat-hozzáférési módszer master-slave típusú; a számítógép, mint „master" egység rendre lekérdezi a környezetében levő „slave" eszközöket.

5. Szolgáltatásminőség biztosítása (ang. QoS) WLAN hálózatokban

Amikor a drótnélküli hálózatok szabványait tervezték, az eredeti elképzelés az volt, hogy a funkcionalitás szempontjából a WLAN hálózatok hasonlóképpen kell működjenek mint a hagyományos LAN hálózatok és ugyanazokat a szolgáltatásokat kell nyújtsák a felsőbb szintű protokolloknak. Viszont a LAN hálózatok mai változatai (100 MHz, 1GHz) aránylag nagy sávszélességet biztosítanak, ami fölöslegessé teszi a minőség-ellenőrző eljárásokat. Ezért egy helyi-hálózaton belül nem szükséges olyan eljárásokat beépíteni, amelyek különbséget tesznek a különböző típusú adatfolyamok között és ezáltal biztosítják az idő szempontjából kritikus adatok minőségi közvetítését.

Az Internet-környezetben változik a helyzet. Itt a sávszélesség még mindig nem elégíti ki az igényeket. Több olyan kezdeményezés létezik, amely próbálja megoldani a szolgáltatások minőségének biztosítását főleg hálózati szinten (IP-szinten). Az új IP v6 változat tartalmaz olyan módszereket, amelyek részben lehetővé teszik az idő-kritikus multimédia adatfolyamok közvetítését egy előre megszabott minőségi szinten.

A WLAN hálózatokban a jóval alacsonyabb sávszélesség miatt a szolgáltatások minőségének biztosítása kritikus feladat. Az első WLAN szabványok, a LAN hálózatok modellje alapján, kevésbé foglalkoztak ezzel a gonddal. Viszont a mára már egyre gyakoribb multimédia közvetítések ennek a feladatnak a megoldását követelik.

IP (Internet Protocol) szinten a minőség-ellenőrzési feladatot két féle megközelítéssel próbálják megoldani. Az IntServ (Integrated Services) protokoll szerint garantálni lehet bizonyos adatfolyamok minőségét, amennyiben ezek paraméterei előre ismertek. A közvetítés egy adó és egy vevő állomás között történik, és az adatok elküldése előtt le kell foglalni a közvetítési pálya mentén a szükséges erőforrásokat.

A DiffServ (Differentiated Services) protokoll különböző szolgáltatási szintet oszt ki a különböző felhasználóknak vagy csoportoknak. A közvetítési sávot megosztja a csoportok vagy osztályok között, és különböző minőségi szintet biztosít ezeknek. A minőségi szintet az adatok típusának megfelelően szabja meg; az üzeneteket a tartalmuk alapján (pl. audió-, videó- vagy számítógép-adatok) osztályozzák és jelölik. A DiffServ nem igényel plusz egyeztetési időt, és csak a kapcsolat két végében levő állomásban (esetleg routerben) alkalmazzák. Ezzel szemben az IntServ az útba eső routerek működését is befolyásolja. Mikor egy üzenet a DiffServ doméniumba kerül, akkor a kapott jelző alapján kezelik a routerek.

A WLAN hálózaton belül nem használnak IP szintű protokollt; ezért az IntServ és a DiffServ jellegű mechanizmusokat alacsonyabb szinten, pontosabban a MAC szinten szükséges beépíteni. Az eredeti IEEE 802.11 szabvány tervezői gondoltak az idő-kritikus és minőséget követelő közvetítésekre, és ezért bevezettek

egy olyan hálózat-hozzáférési módszert is, amely nem az ismert ütközéses (CSMA) módszert alkalmazza, hanem egy master-slave jellegű, ütközés nélküli eljárást. Elvben egy szuper-keretben (ang. superframe) két periódus létezik, az első, amelyben a hálózathoz való hozzáférést szigorúan a hozzáférési pont (AP Access Point) irányítja, és egy második, amelyben bármely állomás a CSMA/CA protokoll alapján igyekszik elfoglalni a közvetítési sávot és elküldeni az adatokat.

Az első periódust az idő-kritikus közvetítésekre szánták. Mivel a közvetítést az AP állomás központilag szabályozza, feltételezhető, hogy egy olyan sávmegosztást lehet elérni, ami megfelelő szintű minőséget biztosít. A második periódusban történő kommunikációra nem lehet egy bizonyos minőségi szintet igényelni és garantálni. Az üzenetek késleltetése a hálózat pillanatnyi terhelésétől függ. Az ilyen fajta közvetítés a véletlenszerű adat-tömbök esetében előnyös, viszont nem felel meg a multimédia típusú adatok esetében, ahol az üzenetek periódusát szigorúan be kell tartani.

A felmérések alapján kiderült, hogy az első periódusban történő közvetítés sem biztosít egy garantált minőségi szintet. Ez azzal magyarázható, hogy a központilag irányított kommunikáció időveszteséggel jár (lásd a szükséges kontroll-üzeneteket), és az AP állomás szintjén egy szűk keresztmetszet keletkezik. Mivel a második periódusban történő üzenetek hossza nem korlátozott, előfordulhat, hogy a következő szuper-keret később kezdődik, és ezáltal a periodicitás nincs betartva.

Ezek következtében egy új protokoll-javaslat született (az IEEE 802.11e), amely az IntServ meg aDiffServ elveit a MAC-szintre igyekszik beépíteni. Az új protokoll kétféleképpen próbálja megoldani a minőség-ellenőrzési kérdést: az első az EDCF (Enhanced Distributed Coordination Function) rövidítésként ismert módszer, a másik meg a HCF (Hybrid Coordination Function). Az EDCF módszer prioritásokat ad a különböző üzeneteknek annak alapján, hogy mennyire kritikus a késleltetési idejük. Természetesen a multimédia típusú adatok magasabb prioritással rendelkeznek. A különböző prioritású üzenetek más várakozási pufferekbe kerülnek, és bizonyos időparaméterek beállításával a nagyobb prioritással rendelkező pufferek előbb szolgálja ki a rendszer. Egy szolgáltatás minőségét a paraméterek pontos és helyzethez adaptált beállítása biztosítja.

A HCF módszer lehetővé teszi, hogy ütközés-mentes periódusokat lehessen használni nemcsak a szuper-keretben erre a célra fenntartott periódusban, hanem a második periódusában is. Az üzenetek hossza és a szuper-keret periódusa korlátozott. Ezáltal egy hálózatban ki lehet dolgozni egy stratégiát, amely az üzenetek maximális késleltetési idejét tudja garantálni. A végzett szimulációk bizonyították, hogy a javasolt új módszerek lényegesen feljavítják a WLAN hálózat QoS tulajdonságait.

6. Minőségbiztosítás ipari környezetben

Mivel a WLA technológiákat egyre gyakrabban alkalmazzák ipari környezetben is, feltevődik a kérdés, hogy a létező szabványok mennyire elégítik ki ennek a környezetnek az igényeit.

A legtöbb ipari alkalmazásban (irányítás, felügyelet, szabályozás) az idő egy kritikus paraméter. Az automatizálási rendszer helyes működése az időkorlátok betartásától is függ. Amennyiben drótnélküli hálózatokat alkalmazunk, szükséges, hogy korlátozzuk az üzenetek maximális késési idejét.

Míg a multimédia alkalmazásoknál az üzenetek késése csak a minőség csökkentését jelenti, az ipari alkalmazásokban egy elkésett üzenet komoly anyagi károkat vagy akár emberi áldozatokat is okozhat.

Sajnos a WLAN szabványok tervezői nem vették figyelembe az ipari alkalmazások igényeit. Ezért szükséges, a multimédiára alkalmas eljárásokat sokkal szigorúbb helyzetekre adaptálni.

Egy ipari alkalmazásban három típusú adatközvetítést lehet meghatározni:

- véletlenszerű kritikus üzenetek
- periodikus üzenetek
- véletlenszerű nem-kritikus üzenetek

Az első kategóriához tartoznak az olyan üzenetek, amelyek egy kritikus eseményt jeleznek (pl. egy komponens meghibásodása). Megjelenési idejük nem ismert, viszont a maximális közvetítési idő adott. Az üzenet rövid és általában valami új eljárást indít el.

A második kategóriához tartoznak azok az üzenetek, amelyek biztosítják az irányított folyamat adatainak begyűjtését és a kontrolljelek kiosztását. Az üzenetek periódusának betartása nagymértékben befolyásolja a vezérlés minőségét, tudniillik a legtöbb szabályozási függvény időtől függ.

A harmadik csoporthoz tartoznak az olyan üzenetek, amelyek nagy adattömböket közvetítenek, de amelyek nincsenek időhöz kötve. Ilyenek például a konfigurálási adatok, a hosszabb időre tárolt adatok (logged files), vagy akár egy program kódjának új verziója.

Az utolsó két üzenetkategóriát aránylag könnyen lehet egy WLAN környezetben közvetíteni. Amennyiben a periodikus üzenetek időparaméterei ismertek, ki lehet dolgozni egy ütemezési stratégiát, amely garantálni tudja a ha-

táridők betartását. Az IEEE 802.11e szabványon belüli HCF módszer alkalmas erre a célra. A harmadik kategóriában levő üzeneteket a megmaradt időben, CSMA/CA módszerrel lehet közvetíteni. Ilyen módon maximálisan ki lehet használni a létező adat-sávot és ugyanakkor nem zavarjuk a periodikus közvetítést.

A véletlenszerű kritikus üzenetek közvetítését viszont nehéz garantálni, amennyiben a határidő kisebb mint a szuperkeret periódusa. Ezekre az üzenetekre az ütközés-mentes periódus volna alkalmas, csakhogy ezt az AP állomás irányítja. Ezért a kritikus esetek állapotát ugyancsak periodikusan pooling módszerrel kérdezheti le az AP állomás. Ez a megoldás lényegesen csökkenti a hálózat hatékonyságát, mivel minden lehető esetet szükséges lekérdezni. Egy sokkal jobb megoldást a szabvány módosításával lehetne elérni. Például a kritikus üzenetek rövidebb üzenetközti várakozási időt kellene használjanak. Emiatt ezek az üzenetek kapnák a legmagasabb prioritást. Egy másik lehetséges megoldás a szuperkeretben egy olyan különleges periódus fenntartására, amelyben csak a kritikus üzeneteket lehet elküldeni. Feltételezve, hogy egy bizonyos időn belül a kritikus üzenetek száma korlátozott (ami megfelel a meghibásodási modelleknek), az esetleges ütközések száma alacsony.

7. Következtetések

A drótnélküli hálózatok új kommunikálási lehetőségeket nyújtanak, főleg a kötetlen kapcsolatok területén. A mobil számítástechnikai eszközök fejlődésével szükségessé vált egy olyan kommunikálási környezet, amely megfelel a „kapcsolat bármikor és bárhol elvnek.

A helyi-hálózatok modelljére épített WLAN hálózatok részint teljesítik ezt az elvet. Viszont a rendelkezésre álló korlátozott sávszélesség miatt ezek kevésbé teljesítik a minőségi elvárásokat. Ezért szükséges beépíteni olyan minőség-szabályozó és garantáló módszereket, amelyek a multimédia adatfolyamok kérelmeit is kielégítik. A legújabb javaslatok, amelyek az IEEE 802.11e szabvány, változatban jelennek meg, lehetőséget nyújtanak a feladat megoldására.

Ugyanakkor viszont a minőség szempontjából sokkal igényesebb ipari környezet kérelmeit az új szabvány sem fedi teljesen. Az idő-kritikus üzenetek határidőn belüli közvetítése még mindig nehezen garantálható a jelenlegi WLAN protokollok által. Ezért olyan típusú üzenet-ütemezési módszereket kell beépíteni, amelyek hasonlítanak az ipari hálózatokban alkalmazott, prioritáson alapuló stratégiákhoz. Ugyanakkor az üzenetek hosszúságának korlátozásával jobb reagálási időt lehet elérni, és a periodikus közvetítések frekvenciáját növelni lehet. A jelenleg folyamatban levő kutatások ezeket a feladatokat próbálják rövid időn belül megoldani.

Irodalom

[1] Braden R., Clark D., Shenker S., "RFC1633 - Integrated Services in the Internet Architecture: an Overview", 1994

[2] Blake, Black D., Carlson M., Davies E., Wang Z., Weiss W., "RFC2475 - An Architecture for Differenciated Services", 1998

[3] Chung S., Piechota K., , "Understanding MAC protocol architectural implementation of 802.11 QoS amendments: A guide to 802.11e technology", Technical report, S3 - Silicon Software Systems, (www.s3group.com), 2003

[4] IEEE, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification", IEEE Standard 802.11, 1999

[5] IEEE, 802.11TGe, "Hybrid Coordinatin Function (HCF)", Proposed Updates to Normative Text, 2001

[6] Grilo A., Macedo M., Nunes M., "A service Discipline for Support of IP QoS in IEEE 802.11 networks", IFIP Personal Wireless Conference, Finland, 2001

[7] Quing N., Ropmdhani L., Turletti T., Aad I., "QoS Issues and Enhancements for IEEE 802.11 Wireless LAN, Technival report no. 4612, INRIA, 2002

[8] Mercier A., Minet P. George L., "Introdicing QoS support in Bluetooth Piconet with a Class-Based EDF Scheduling", Technical report No. 5054, INRIA (www.inria.fr), 2003

Hálózati biztonság az IPv6 protokoll tükrében

Network Security and the IPv6 Protocol

SOMODI Zoltán

Kolozsvári Műszaki Egyetem,
Számítástechnika és Automatizálás Kar

Abstract

This paper presents one of the most important features of the IPv6 protocol: security. Since it was developed in order to solve the address space shortage of the current IP protocol, it provides solutions for many security problems detected in IPv4. In the first part of the paper we present some of the most "popular" internet attacks. They could be grouped in a few categories. Many of them are based on IP spoofing. Others have the goal to inhibit some internet services. For each attack type there is a special solution, a particular protection mechanism. But there is no general solution to prevent IP Spoofing. In the second part the IPSec protocol is presented as a protection against attacks based on IP Spoofing. Because IPSec is mandatory in IPv6, the new protocol provides better security than its predecessor. The fragmentation mechanism of IPv6 offer protection against fragmentation attacks. Although, IPv6 is not a panacea for security problems, security provided by IPv6 is a great improvement over IPv4, and is a good reason to deploy IPv6 in our network.

1. Bevezetés

Az Internet immár több mint 30 éves története alatt nagyon sok támadást jegyeztek fel a statisztikákba. Az idő folyamán a támadások egyre sűrűbbek és egyre változatosabbak lettek.

Az Internetre kötött eszközök számának rohamos növekedése szükségessé tette az IP címtartomány kibővítését. Ez egy új Internet Protokoll, az ún. IPv6 bevezetésével valósult meg. Az IPv6 tervezésénél figyelembe vették az eddigi tapasztalatokat, és kijavították ez észlelt hibákat, hiányosságokat. Így, az IPv6 megoldást nyújthat több támadástípus ellen is.

Ebben a cikkben bemutatunk néhány gyakori támadástípust, valamint a jelenlegi védekezési módszereket, majd áttekintjük az IPv6 azon tulajdonságait, amelyek védelmet nyújtanak ezen támadások ellen.

2. Biztonsági hibák az IPv4-ben

Az internetes támadásokkal foglalkozó szakemberek arra a következtetésre jutottak, hogy a támadásokat négy fő csoportba lehet besorolni [1]. Ezek a támadástípusok a következők:

- Szolgáltatás megtagadás
- Információlétrehozás
- Információtörlés vagy módosítás
- Elektronikus fülelés

Minden támadástípus az IPv4 egy-egy gyenge pontját, hiányosságát használja ki. Az alábbiakban bemutatjuk ezeket a hiányosságokat, kiemelve azokat, amelyeken a szolgáltatás-megtagadásos támadások alapulnak, és ismertetjük a jelenlegi védekezési módszereket.

A szolgáltatásmegtagadásos támadás (röv: *DoS - Denial of Service*) jelenleg az egyik legnehezebben kivédhető támadás-típus. Ilyen esetekben a támadó megpróbálja úgy leterhelni a rendszert vagy a hálózatot, hogy egy bizonyos szolgáltatásnak a működése lehetetlenné váljon. Általában kiszolgáló gépek a célpontok, pl. webszerverek, DNS-szerverek, FTP-szerverek, email-szerverek stb. Sok megoldás létezik arra, hogy hogyan lehet megbénítani egy szolgáltatást, de majdnem mindenik módszer közös vonása, hogy hamis IP címeket használ. A jelenlegi internet protokoll egyik nagy hátránya, hogy lehetővé teszi a címhamisítást (ang.: *IP Spoofing*), és a hamis címekkel való hálózati kommunikációt. Ez az oka annak, hogy nagyon nehezen lehet védekezni a DoS típusú támadások ellen. Hamis címeket két okból használhatnak a támadók. Egyrészt, természetesen azért, hogy elrejtsék valós azonosságukat. Másrészt, a hamis címek használata a támadási mechanizmus részét képezheti, ahogy ezt a továbbiakban látni fogjuk.

A DoS támadásoknak több változata ismert, ezekből fogunk az alábbiakban bemutatni egy néhányat.

A Smurf támadás. A támadó *ICMP echo request* típusú csomagokat küld broadcast IP címekre. Ez azt jelenti, hogy több gép a helyi hálózaton kap egy-egy ilyen csomagot, amire *ICMP echo reply* csomaggal válaszol. Ezek a csomagok megnövelik a hálózati forgalmat, torlódáshoz vezetve. Általában a támadó nem a saját címét használja, hanem forráscímként egy másik gép IP-jét használja, amely ellen tulajdonképpen irányul a támadás. Tehát az összes ICMP válasz-csomag erre a csomópontra irányul, ami megbéníthatja az illető gép működését.

Az ilyen támadások elleni védekezés lehet, ha letiltjuk a broadcast címekre irányuló ICMP forgalmat [2]. Így megakadályozhatjuk a támadás továbbterjedését. De ha ez nem történik meg, a célgépet nagyon nehezen tudjuk megvédeni.

TCP SYN elárasztás (ang.: *flooding*)**.** A támadás a TCP kapcsolatok kialakításának módját használja ki. Ahhoz, hogy egy ilyen kapcsolat létrejöjjön egy kliens és egy szerver között, három lépésre van szükség. Először a kliens egy SYN csomagot küld, amire a szerver SYN-ACK csomaggal válaszol. Végül a kliens egy ACK csomaggal fejezi be a kapcsolatteremtést. A támadás abból áll, hogy egy „kliens" elárasztja a kiszolgálót SYN csomagokkal, de nem válaszol a SYN-ACK csomagokra. Így a szerver erőforrásai elhasználódnak a rengeteg félig nyitott kapcsolat nyilvántartására, és nem lesz lehetőség a valódi kliensek kiszolgálására. Természetesen a támadó hamis, és változó forráscímeket használ, így nem lehet kiszűrni a támadást [3].

TCP ACK vihar. Minden gép, amelynek élő TCP kapcsolata van más gépekkel, nyilván kell tartsa a fogadott csomagok sorszámát. A támadás lényege, hogy megzavarja ezt a nyilvántartást, hamis csomagokat küldve a kapcsolatban álló állomásoknak. Ez olyan forgalmat eredményez, amely megbéníthatja a megtámadott gépek működését [4].

LAND támadás. Egyes TCP/IP implementációk sebezhetők olyan SYN csomagokkal, amelyeknek forrás- és célcíme, illetve portja azonos. A LAND ezt a hiányosságot használja ki, hamis csomagokat küldve a célgépre [5]. A IP protokoll jelenlegi változatával lehetetlen kiszűrni ezt a támadást.

Könnycsepp-támadás (ang.: *Teardrop attack*)**.** A TCP/IP protokoll-együttes egyik fő feladata a csomagok tördelése. Ha az útvonalon a következő router nem képes kezelni egy adott méretű csomagot, akkor azt kisebb darabokban fogja neki továbbküldeni a szomszédja. Minden töredékben található egy ofszet, amely a következő töredékre mutat. A támadó olyan hamis csomagokat gyárt, amelyben az ofszet helytelen. Ez megzavarhatja a töredékek összeillesztésének folyamatát, amely a rendszer összeomlásához vezethet.

Halálos ping (ang.: *ping of death*)**.** Ebben a támadásban is a csomagok méretét manipulálja a támadó. A legnagyobb megengedett IP csomag 65.536 bájt hosszú lehet. Ennél nagyobb csomagokat a hálózat nem továbbít. De küldhetünk több olyan csomagrészt, amelyeknek összmérete nagyobb a megengedettnél. Bizonyos operációs rendszerek nem tudnak mit kezdeni ezekkel a túlméretezett csomagokkal (amelyek az összeillesztés után keletkeznek) és ez a rendszer lefagyásához, összeomlásához vagy újraindulásához vezethet.

3. Az IPv6 biztonsági rendszere

A biztonsági mechanizmus fő részét az IPSec adja az IPv6-ban. Az IPSec egy biztonsági szabványrendszer, amelyet eredetileg az IPv6-hoz fejlesztettek ki. Mivel nagy szükség volt a biztonságra a jelenlegi IP-ben is, alkalmazták az IPv4-re is. Mégis, az IPv4-ben az IPSec opcionális szolgáltatás és nagyon gyártófüggő, több nem teljesen kompatibilis implementáció létezik. Az IPv6-ban viszont az IPSec kötelező, végpont-végpont biztonságot nyújtva.

Az IPSec a hitelesítési (authentication) fejléc (AH) és a titkosítási fejléc (ESP) segítségével valósítja meg a biztonsági szolgáltatásokat. Ezek a fejlécek használhatók külön-külön, kombinálva a szükséges biztonság függvényében. Mindkét fejléc két módban használható:

- *transzport mód*: a felsőbb szintű protokollok (TCP, UDP, ICMP), azaz az IP csomag adatmezejének a védelmére szolgál [6]. A transzport mód tipikusan két host (IP kommunikációs szereplő) közti végpont-végpont kapcsolatokban használatos.
- *tunnel mód:* az egész eredeti csomagot egy másik IP csomag belsejébe helyezik (IP-IP tunnelezés), így biztosítva, hogy az egész eredeti csomag a (publikus) hálózaton való áthaladás közben változatlan marad. A tunnel módot leginkább két biztonsági gateway (pl. tűzfal vagy router) között használják, amely esetben a két gateway között egy VPN-t (Virtual Private Network, virtuális magánhálózat) hoznak létre.

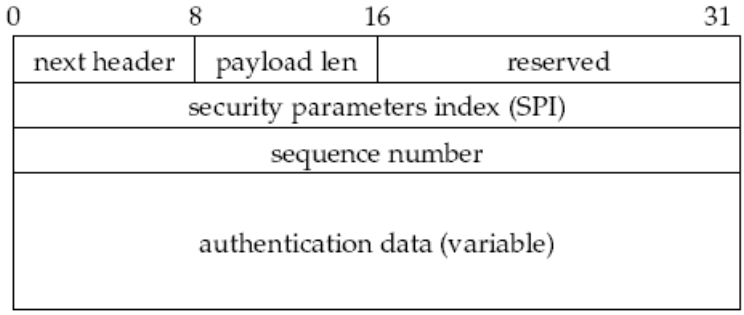
Az IPSec egyik kulcsfogalma a *Security Association* (SA – Biztonsági kapcsolat). A biztonsági fejlécekben található *Security Parameters Index* mező a cél IP címmel és a biztonsági protokollal (AH vagy ESP) együtt egyértelműen meghatároz egy SA-t. Ez egy egyirányú kapcsolat a kommunikáló partnerek között. Álta-

lában tartalmazza a hitelesítéshez vagy titkosításhoz szükséges kulcsot, valamint a használt hitelesítési, illetve titkosítási algoritmust. Az IKE (Internet Key Exchange) leírja azt a folyamatot, amely által létrehozható egy új SA.

3.1. Az AH protokoll

A hitelesítési fejléc (Authentication Header, AH) az egyes IP csomagok hitelesítése, a bennük lévő adat sértetlenségének az ellenőrzésére és ún. replay támadások elleni védelemre nyújt módot. A hitelesítés lehetővé teszi, hogy a csomag címzettje megbizonyosodjon az IP csomag feladójáról. Az adatsértetlenség ellenőrzésével a címzett igazolhatja, hogy a csomagot annak feladása óta egy közbülső ponton nem módosították.

A *1. ábrán* az AH fejléc szerkezete látható, amelyet az alábbiakban ismertetünk részletesebben.



1. ábra
Az AH fejléc

A Next Header (következő fejléc) mező adja meg az AH fejlécet közvetlenül követő fejléc típusát. Az AH fejléc teljes hosszát a Payload Length tartalmazza. A Security Parameters Index mezőben találhatjuk azt a számot, amely azonosítja az SA-t. Egy monoton növekvő számláló értékét tartalmazza a Sequence Number, amelynek – ahogy azt a későbbiekben látni fogjuk – a replay támadások elleni védelemben van szerepe. A változó méretű Authentication Data tartalmazza azt az ICV (Integrity Check Value) értéket, amelyet a csomagok hitelesítésére és sértetlenségének ellenőrzésére használunk. Az ICV kiszámítása során az IP csomag következő részeit veszik figyelembe: az IP fejléc azon mezői, amelyek vagy nem változnak meg az út során, vagy amelyeknek a célpontbeli értéke a feladó számára kiszámítható. A megváltozó és nem kiszámítható mezők tartalmát az ICV kiszámításához a feladó és a címzett oldalon egyaránt zérusokkal helyettesítik; az egész AH fejléc úgy, hogy a hitelesítési mező nullákkal van kitöltve; az egész felsőbb szintű protokoll adat, amelyről feltesszük, hogy az út során nem változik meg.

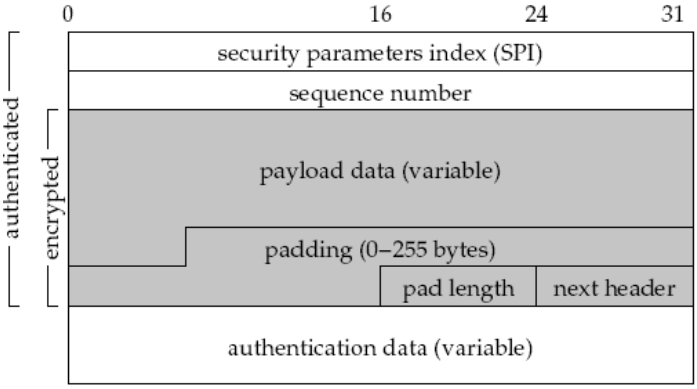
A küldő a fenti logikai algoritmus alapján számolja ki az ICV értékét, amelyet azután a hitelesítési adatmezőben helyez el. A fogadó szintén ezek alapján számolja ki az ICV értékét és veti össze az eredményt a hitelesítési mezőben kapott értékkel. Ha a kettő megegyezik, a csomag feladójának a hitelesítése és a csomag integritásának az ellenőrzése sikeres volt.

3.2. Az ESP protokoll

Az ESP (Encapsulating Security Payload) titkosítási szolgáltatást nyújt, védve az üzenet tartalmát a lehallgatások ellen, valamint korlátozott védelmet tud nyújtani a forgalmi adat-analízisen alapuló támadások ellen. Az ESP opcionálisan az AH-hoz hasonló hitelesítési szolgáltatásokra is képes.

Az ESP fejléc szerkezetét a *2. ábrán* mutatjuk be, az alábbiakban a mezők jelentését írjuk le.

Ez a fejléc is tartalmaz egy SPI mezőt, amely azonosítja az SA-t, valamint egy Sequence Number-t, ugyanazzal a jelentéssel, mint az AH-ban. A titkosított adatot az adatmezőben (Payload Data) mezőben találhatjuk, ezt egy helykitöltés (padding) követi, amely a titkosító algoritmusok helyes működéséhez szükséges. A helykitöltés hosszát a Pad Length adja meg, az ezt követő Next Header pedig a következő fejlécet azonosítja.



2. ábra
Az ESP fejléc

Az *adatmező*, *helykitöltés*, *helykitöltés hossza* és *következő fejléc* mezőket az ESP az SA-ban tárolt szimmetrikus titkosító algoritmussal titkosítja. Az ESP először titkosítja a *hasznos adat*, *helykitöltés*, *helykitöltés hossza* és *következő fejléc* mezőket, majd hitelesíti az ESP csomagot. Az ESP csomagot tartalmazó IP csomag fejléc-elemeinek a hitelesítését az ESP – az AH-val ellentétben – nem végzi el, ezért az ESP NULL titkosítás melletti autentikációs szolgáltatása nem teljesen azonos az AH szolgáltatásával.

3.3. Támadások elleni védelem IPSec-kel

Amikor az IPv6-ot tervezték, az előző fejezetben bemutatott támadások mind ismertek voltak. Így bizonyos védelmi mechanizmusokat be lehetett építeni az új protokollba.

Amint említettük, az IPv6-ban a biztonsági szolgáltatásokról az IPSec protokoll gondoskodik. Az IPSec *opcionálisan* az IPv4-gyel is használható, az új protokollban viszont *kötelező* szolgáltatás. Lényeges különbség van a két megközelítés között. Az IPv4 esetében használhatjuk a biztonsági protokollt, de csak akkor, ha a kapcsolat másik végén levő csomópont is ismeri a szolgáltatást. Egy IPv6-os hálózaton viszont, biztosak lehetünk abban, hogy minden csomóponton működik az IPSec. Ez megoldást nyújt a legtöbb IPv4-ben tapasztalt biztonsági hiányosságra. Ebben a részben főleg azokat a védelmi mechanizmusokat ismertetjük, amelyek a DoS típusú támadásokat hiúsítják meg.

A különböző támadások ismertetéséből kiderült, hogy a legtöbbjük az IP csomag tartalmának meghamisításán alapul. Sok esetben a forrás- illetve célcímet hamisítják a támadók, de az is előfordul, hogy más mezőt változtatnak meg. Ezt a támadási módszert, amelyet az angol szakirodalom IP Spoofing-nak nevez, az új Internet protokoll lehetetlenné teszi. Az AH vagy az ESP fejléc használatával egy hálózati csomópont mindig biztos lehet abban, hogy a fogadott csomag arról az IP címről jött, amelyet a csomag tartalmaz, hogy nem változott meg útközben az információ tartalma, és hogy a kommunikáló partnereken kívül senki más nem látta a csomag tartalmát.

A DoS támadásoknál a támadónak két célja lehet az IP csomag meghamisítására. Egyrészt, minden támadó el szeretné rejteni valós azonosságát, ezért megváltoztatja a forráscímet. Másrészt, ha a csomag egyes mezőit egy megadott formában módosítja, bizonyos működési zavarokat okozhat a célgépen, vagy az egész hálózaton. Például a LAND támadásoknál a forrás- és célcím, illetve forrás- és célport azonos kell legyen ahhoz, hogy hatásos legyen a támadás. Ezek a támadási formák mind lehetetlenné válnak, ha az IP csomagokat azonosítjuk. Ha a fogadott csomag kiszámolt ICV értéke nem egyezik meg a csomagban levővel, akkor azt a csomagot vissza kell utasítani.

Vannak viszont olyan DoS támadások, amelyek címhamisítás nélkül is megvalósíthatók. Ilyen például a TCP SYN elárasztás. Ha a támadó a saját valós címéről küldi a SYN kéréseket, és nem válaszol a SYN-ACK csomagokra, egy idő után pont úgy megbéníthatja a szerver működését, ahogy azt az előző fejezetben láttuk. Ebben az esetben viszont, be lehet építeni olyan védekezést, amely nem enged be a hálózatba csomagokat olyan címről, amely már több fél-nyitott kapcsolatot kezdeményezett. Ugyanakkor, a támadót (vagy legalább annak gépét) azonosítani lehet a címe alapján, és ez elbátortalanítja a rossz szándékú felhasználókat.

A Smurf, TCP ACK vihar és LAND támadások ellen is védelmet nyújt az AH protokoll sértetlenséget ellenőrző szolgáltatása. Ha egy csomag megváltozott tartalommal érkezik egy csomópontra, ezt a célgép ész-

leli, és visszadobja a csomagot. Így nem történhet meg az, hogy egy hamis sorozatszámú csomag megbontsa a kommunikáló partnerek közötti szinkront.

Bemutattunk két olyan DoS támadást is, amely nem csomaghamisításon alapszik, hanem a TCP/IP protokoll-együttes tördelési mechanizmusának hiányosságait használja ki. A „könnycsepp" és a „halálos ping" támadásokról van szó. Igaz, ezekben az esetekben is szükség van egy módosított tartalmú csomag küldésére, de ezt a módosítást elvégezheti a támadó is az ICV érték kiszámítása előtt. Így az ICV-t egy hibás csomagra számolja ki a biztonsági protokoll, és a célgép nem fogja észlelni, hogy valami nincs rendben. De az IPv6-nak van egy másik fontos tulajdonsága is, amely megakadályozza az ilyen típusú támadásokat. Az új protokollban csak a végpontokon levő állomások tördelhetik szét és állíthatják ismét össze a csomagokat, a közbeesők nem. A forrás állomás meghatározza a maximálisan átvihető méretű egységet (MTU), és ha kell, tördelést végez. Ekkor az összeillesztéshez szükséges információkat a Fragmentation Header nevű kiegészítő fejlécben tárolja el, mely csak ebben az esetben van jelen a csomagban.

Kutatócsoportunk méréseket végez, amelynek célja, megvizsgálni az IPv6 stabilitását tördelési támadások esetén valós hálózatokban.

4. Következtetések

Az IPSec jelenleg az egyik legjobb biztonsági szolgáltatást nyújtó protokoll. Kötelező szolgáltatásként jelenik meg az IPv6, míg az IPv4-nél csak opcionális volt. Az IPSec jelenléte miatt, az IPv6-os hálózatok biztonságosabbá válnak a címhamisításos, illetve szolgáltatás-megtagadásos támadások és a lehallgatásokkal szemben.

Habár az IPv6 által nyújtott biztonsági szolgáltatások nagy fejlődést jelentenek az IPv4-hez képest, mégsem tekinthetjük ezt sem csodaszernek, neki is megvannak a saját biztonsági rései és hátrányai. Ezek közül íme néhány [1]:

- Az export törvények miatt, a használható titkosítási algoritmusok erőssége korlátozott.
- Az IPSec a Public Key Infrastructure-re (PKI) alapul, amely még nem teljesen szabványosított.
- További fejlesztés szükséges az IKE protokoll illetve a DoS és az elárasztásos támadások elleni védekezés területén.

Az IPv6 megold néhány fontos biztonsági problémát, de ugyanakkor újabb támadási lehetőségek is adódnak [7]:

- A DoS típusú támadások általi sebezhetőség, mivel a titkosítási eljárások processzor-igényes folyamatok.
- A tűzfalak részére lehetetlenné válik a forgalmi analízis, ha használjuk az ESP titkosítási protokollt. Tunnel módban használva, létre lehet hozni kapcsolatot egy nem megbízható állomással egy privát hálózaton belül, mivel az IP cím láthatatlan a tűzfal számára.
- Az IPv6 autokonfigurációs követelményei lehetőséget nyújtanak bizonyos DoS támadásokra, mivel a rendszer sebezhetővé válik hamisított alhálózat maszkokat tartalmazó csomagokra.

Mindezek ellenére elmondhatjuk, hogy az IPv6-ot még akkor is érdemes bevezetni hálózatunkba, ha van elég szabad IP cím. Az új protokollnak, a kibővített címtartomány mellett, más fontos tulajdonságai is vannak: biztonság, szolgáltatásminőségi garanciák, autokonfiguráció stb.

5. Könyvészet

[1] S. Hagen, IPv6 Essentials, O'Reilly, Sebastopol, CA, pp. 77-88, 2002

[2] "CERT Advisory CA-1998-01 Smurf IP Denial-of-Service Attacks", 1998, http://www.cert.org/advisories/CA-1998-01.html

[3] "CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks", 1996, http://www.cert.org/advisories/CA-1996-21.html

[4] D. Anderson, B. Teague, "ACK Storms and TCP Hijacking", http://www.owlnet.rice.edu/~bteague/papers/comp527.pdf

[5] "I-019: Tools Generating IP Denial-of-Service Attacks", in CIAC Information Bulletin, December 19, 1997, http://ciac.llnl.gov/ciac/bulletins/i-019.shtml

[6] TIPSTER6, Testing Experimental IPv6 Technology and Services in Hungary, 2001**,** http://tipster6.ik.bme.hu/tipster6_ener6_en.html

[7] Y. Chauhan, D. Sanghi, "Security in the wake of IPv6", A Term Paper Report for Advanced Computer Networks, Indian Institute of Technology Kanpur, 2003

e-kereskedelem, e-vásárlás

e-sales, e-purchase

Dr. BORBÉLY Endre

Budapesti Műszaki Főiskola
Kandó Kálmán Villamosmérnöki Főiskolai Kar
Híradástechnika Intézet

Abstract

In the past few decades, years, the forms of shopping have greatly changed all over the world. More and more people would do their shopping through the Internet rather than go intovarious shops and stores.

In Hungary, too, several shoppers prefer sitting in an armchair and doing the purchases through e-stores. In 2002 over HUF 4.5 billion were spent by use of e-purchase. A wide range of products, various benefits, mail service, discount prices are the advantages that increase the volume of e-sales. Furthermore, aspects of economics, market, policy, regulations, consumer protection, as well as the rights of customers and the duties of salespeople are considered as matters of great importance.

Az utóbbi években, évtizedekben a vásárlási szokások az egész világon sokat változtak. Egyre többen nem az utcai boltokban, hanem a világháló adta lehetőségeket használják beszerzéseiknél.

Magyarországon is sokan karosszékből intézik vásárlásaikat. 2002-ben több mint 4.5 millárd forintért vásároltak az e-boltokban.

A széles termékválaszték, a különböző kedvezmények, a postai szállítás, fizetési kedvezmények is fokozzák az e-boltok forgalmát. A gazdasági, a piaci, a politikai, a jogi, fogyasztóvédelmi, a vásárló jogai, az eladó kötelességei is fontos kérdések.

Széles termékválaszték

Világosan látszik, hogy a piaci kezdetekre jellemző szűkös termékválaszték után, ma már szinte nincs olyan termékkategória, amely ne lenne megtalálható az interneten is.

Az online boltok túlnyomó része száz és kétezer közötti terméket forgalmaz, de itt is jellemző a kínálat koncentrációja, néhány nagy webáruház adja a teljes termékkínálat több mint felét.

Kedvezmények

Az üzletek 15%-a nyújtott állandó árkedvezményt termékeire, amelynek jellemző mértéke 6% volt. Az állandó kedvezmények mellett időszakos árkedvezményt is adnak a vásárlás növelése érdekében.

Postai szállítás, készpénzes fizetés

Az online boltokban leggyakrabban a postai szállítást választhatjuk (78%), de lehetőség van futárszolgálat igénybevételére (29%) és néhány esetben (22%) a boltok is vállalják a kiszállítást. A boltok átlagosan egy hetes szállítási határidőt vállalnak. Bankkártyával egyelőre csak a boltok alig egytizedében fizethetünk.

Javuló tájékoztatás

Magyarországon is terjed a megfelelő szintű tájékoztatás gyakorlata, azonban ezen a téren még van mit fejlődnie a magyar e-boltoknak. Már található súgó, és ugyanekkora az aránya azoknak a boltoknak is, ahol a fizetési feltételekről tájékozódhatunk a vásárlást megelőzően. A garanciális feltételekről és az adatvédelemről is olvashatunk. A marketing célú adatgyűjtés nem jellemző. A vásárlástól való elállás jogáról olvashatunk. Az e-boltok ügyfélszolgálata elérhető e-mailen keresztül és kérdéseinkre általában már néhány órán belül pontos tájékoztatást kaphatunk. Az üzle-

tek működtetnek telefonos ügyfélszolgálatot. Mindent figyelembe véve a széles termékkínálat, az árkedvezmények, vásárló számára biztonságot nyújtó szállítási és fizetési feltételek, valamint a tájékoztatás és a szolgáltatás minősége alapján elmondható, hogy az online beszerzés akár otthonról, a karosszékből is kényelmesen megoldható.

Online vásárlás

Az internet felhasználói táborának növekedésével párhuzamosan a hagyományos internetes tevékenységek mellett – mint például az elektronikus levelezés vagy a világháló böngészése – az elmúlt években Magyarországon is fejlődésnek indult az online vásárlás. A világhálón értékesített termékek és szolgáltatások köre igen széles: a kézzelfogható, fizikai termékek dominálnak, de megtalálhatók a szolgáltatások, valamint a digitális, információs javak is. Az online áruházak a világháló által nyújtott lehetőségeket használják ki, melyek mind a vásárlók, mind az eladók számára számos előnyt jelentenek. Ilyen előny például az eladó számára, hogy nem kell bolthelyiséget fenntartania, a vásárló szempontjából pedig a helytől és nyitvatartási időtől független, kényelmes vásárlási lehetőség.

E-boltok Magyarországon

A hazai elektronikus boltok száma és a boltok által lebonyolított forgalom évről-évre nő. Jelenleg megközelítőleg 300 e-bolt működik Magyarországon. A GKI Gazdaságkutató Rt. által végzett felmérés szerint az összes bolt 2002-ben 4,5 milliárd forintos forgalmat bonyolított le, az online vásárlók aránya az internethasználók körében 6%-ra emelkedett.

Az Egyesült Államokban, ahol az elektronikus vásárlás kultúrája igen fejlett, az online vásárlók általában az alábbiakat jelölik meg az internetes vásárlás indokaként:

- az utazási idő megtakarítható,
- nincs kötött nyitvatartási idő,
- az ünnepi tumultus elkerülhető,
- megvan a lehetőség arra, hogy kedvezőbb árakat találjunk,
- könnyebb megtalálni a termékeket a világhálón,
- fizikai boltban nem árusított termékek is megvásárolhatók online.

Magyarországon az internetezők az alábbi szempontokat tartják a legfontosabbnak:

- biztonság,
- árkedvezmények,
- gyorsaság,
- kényelem,
- széles körű termékinformációk,
- 24 órás elérhetőség.

Üzleti modellek

Az online boltokat több csoportra oszthatjuk aszerint, hogy rendelkeznek-e hagyományos üzlettel, illetve kereskedelmi háttérrel:

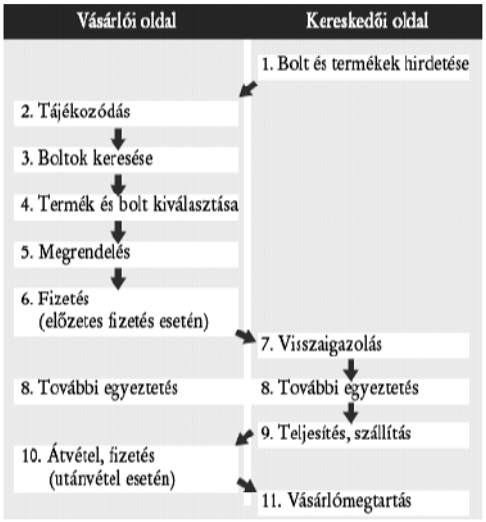
- *offline háttérrel rendelkező e-bolt:* Ebbe a kategóriába azok az e-boltok tartoznak, amelyek hagyományos, offline áruházi jelenléttel és kereskedelmi háttérrel rendelkeznek.
- *tisztán internetes bolt:* A tisztán internetes boltok kizárólag az interneten értékesítik termékeiket, hagyományos bolttal nem rendelkeznek.
- *önálló e-bolt:* Az önálló, független e-boltok kizárólag saját termékeiket kínálják online áruházukban.
- *elektronikus bevásárlóközpont:* Ebbe a kategóriába azok az online áruházak tartoznak, amelyek több kereskedő számára biztosítanak internetes értékesítési felületet.

Termékkínálat

A magyar online boltok kínálata gyakorlatilag a hagyományos áruházakban kapható legtöbb termékkörre kiterjed. A legnagyobb kínálattal rendelkező termékkategóriák a következők:

- könyv,
- kép- és hanghordozó (CD, DVD, audió- és videókazetta, hanglemez),

– hardver, szoftver.
– A felsorolt három termékkör adja a boltok kínálatának több mint négyötödét.



Az internetes vásárlás folyamata

Az elektronikus boltokban történő vásárlás – akárcsak a telefonos rendelés – a termék kiválasztását és megrendelését, a kiszállítást és az áru átvételét foglalja magába. Ebben a folyamatban a vásárló és a kereskedő, illetve a kereskedővel kapcsolatban lévő egyéb piaci szereplők – mint például futárszolgálat, posta, bank – vesznek részt. Az online vásárlás folyamata több lépésre bontható, ezt szemlélteti az alábbi ábra:

Az online és offline vásárlás kapcsolata

A világhálón és a hagyományos boltokban történő információgyűjtés és vásárlás sok esetben egymásba fonódik, a vásárlók az internetet és a hagyományos boltokat egyaránt felkeresik, információt gyűjtenek mindkét helyről. Összetettebb termékek esetén nem ritka az alábbi, mind az elektronikus, mind a hagyományos boltokat érintő vásárlási folyamat. A vásárlás szakaszai:

Információgyűjtés az interneten

Az e-boltok jellemzője, hogy a termékek kiválasztásától a megrendelésig – és néhány esetben a fizetésig – a vásárlási folyamat a világhálón keresztül zajlik. Számos honlap van azonban, amelyen ugyan nem lehet a rendelést leadni, de a termék- és árinformációk begyűjthetők.

Azok a jó e-boltok, melyekben a következő módon lehet vásárolni:
– kosárral lehet vásárolni, vagyis a boltban katalógusból közvetlenül kiválasztható a termék, a kosárba tett termékek pedig együttesen megrendelhetők,
– a megrendelés teljes folyamata az interneten történik,
– bárki megkötés és korlátozás nélkül használhatja és vásárolhat,
– a honlap és az ott elhelyezett információk, adatok magyar nyelvűek és
– az üzlet termékválasztéka részben vagy egészben beleillik a tipikus karácsonyi termékek körébe.

Céginformációk

A vásárlás során a vevő és az eladó nem kerül személyes kapcsolatba egymással, ezért a vásárlói bizalom megteremtésében, a problémák, reklamációk kezelésében fontos szerepe van annak, hogy a vevő pontosan tudja kivel áll kapcsolatban. Információkérés vagy reklamáció esetén e-mailen vagy telefonon elérhető kell legyen az ügyfélszolgálat.

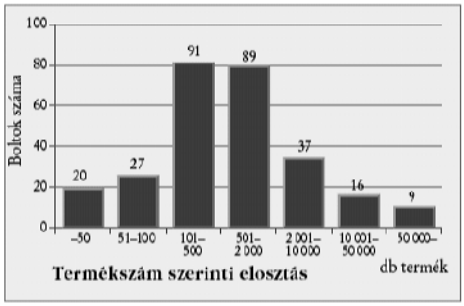
Üzletszabályzat, súgó

A vásárlás feltételeit, körülményeit a vevő jogait, a kapcsolódó fogalommagyarázatokat az e-boltok többsége üzletszabályzatok, tájékoztatók, általános tudnivalók formájában, összegyűjtve ismerteti a vásárlókkal, vagyis ezek az információk egy helyen megtalálhatók.

Termékinformációk

A vizsgálat eredményei igazolták, hogy az internet, az ott található információk mennyisége miatt ideális eszköz a vásárlást megelőző információgyűjtés egyszerű és hatékony lebonyolítására.

Termékválaszték



A virtuális üzletek termékkínálata változatos képet mutat. Az interneten egyaránt megtalálhatók a néhány terméket forgalmazó kis szaküzletek és a hatalmas áruválasztékkal rendelkező nagyáruházak.

E-boltok termékkínálata

Termékkör	Termékkör: Boltok száma (db)	Termékkör: %	Termék: Termékszám (db)	Termék: %	Termék: Átlagos termékszám
Könyv, folyóirat	78	13,15	768.080	51,29	9 847,2
Zenei cd, dvd, videó (kép, hanghordozók)	79	13,32	485.279	32,40	6 142,8
Hardver, szoftver	84	14,17	48.463	3,24	576,9
Irodaszer	31	5,23	41.834	2,79	1 349,5
Erotika	19	3,20	28.575	1,91	1 503,9
Élelmiszer	28	4,72	27.456	1,83	980,6
Szabadidő (hobbi, benne barkács)	32	5,40	21.203	1,42	662,6
Ruházati cikk (ruha, cipő)	25	4,22	10.685	0,71	427,4
Telefon és tartozékai	22	3,71	10.627	0,71	483
Játék	22	3,71	10.015	0,67	455,2
Szórakoztató elektronika	33	5,56	7.717	0,52	233,8
Sport cikkek, camping cikkek	20	3,37	6.683	0,45	334,2
Szépségápolás, kozmetika	20	3,37	5.845	0,39	292,3
Fotó	22	3,71	4.997	0,33	227,1
Bútor	18	3,04	4.572	0,31	254
Háztartási készülékek	19	3,20	4.548	0,30	239,4
Ajándék	19	3,20	4.357	0,29	229,3
Gépjármű, autóalkatrész	6	1,01	3.216	0,21	536
Óra-ékszer	11	1,85	1.994	0,13	181,3
Képzőművészeti alkotások	5	0,84	1.502	0,10	300,4
Összesen	593[5]	100,00	1.497.648	100,00	2.525,5

Kedvezmények

Az e-boltok egy része valamilyen konkrét kedvezménnyel próbálja vásárlásra ösztönözni a látogatókat.

A honlapok használhatósága

Amerikai kutatások szerint [7] az e-boltok felhasználóbarát kialakítása nagyban befolyásolja azok sikerességét, hiszen nem elég sok információt és gazdag áruválasztékot kínálni a látogatóknak, de a termékszámmal összhangban, áttekinthető.

Szállítás, fizetés

Az e-boltoknál a leggyakoribb kiszállítási mód a postai szállítás, és fizetésnél az utánvét.

Szállítási módok megoszlása

Szállítási mód	Gyakoriság: db	Gyakoriság: %[6]
Futár	85	29
Posta	226	78
Bolt szállítja ki	63	22
Helyben lehet átvenni	101	35
Nincs információ	26	9

Fizetési módok gyakorisága

Fizetési mód	Gyakoriság: db	Gyakoriság: %
Utánvét	252	87
Bankkártya	26	9
Készpénz, boltban	92	32
Átutalás	44	15
Csekk	12	4
Nincs információ	23	8

Fogyasztóvédelem az interneten

A fogyasztók védelméről szóló jogszabályok az online vásárlásra ugyanúgy vonatkoznak, mint a hagyományos kereskedelemre. A jótállási, garanciális és egyéb fogyasztóvédelmi jogok természetesen online vásárlás esetén is megilletik a vásárlót.

Kiegészítő szolgáltatások

Jellemző kiegészítő szolgáltatások

Szolgáltatás	Gyakoriság: db	Gyakoriság: %
Termékfigyelés	16	6
Rendeléskövetés	20	7
Termék összehasonlítás	4	1
Hírlevél	114	39

Szerződéskötés online vásárlás esetén

Internetes vásárlás során az eladó és a vásárló nincs közvetlen kapcsolatban egymással, a termékek megrendeléséről szóló szerződés távol lévő felek között jön létre. A szerződéskötés folyamata az alábbi szakaszokra bontható:

– Bolt ajánlata vagy felhívása
– Megrendelés
– Visszaigazolás

Az eladó kötelezettségei

Az alábbiakban röviden összefoglaljuk az internetes kereskedők néhány fontosabb kötelezettségét. Vásárlás előtt érdemes megnézni, hogy a bolt ezeknek a kötelezettségeknek eleget tesz-e:

– *Cégadatok*
– *Vásárlási tájékoztató*
– *Szerződési feltételek*
– *Visszaigazolás:*

Szállítási költségek

Honlap neve	Ára	Vállalt szállítási és csomagolási költség	Fizetett végösszeg	Különbözet
Alexandra ONLINE	2 890	734	3 245	–379
Cigarshop	990	1 125	2 115	0
eBolt Műszaki Áruház	6 756	1 000	7 756	0
Fotexnet	27 990	0	27 990	0
FotóMarket	25 500	0	25 500	0
Gift Rt.	4 857	1 180	6 248	211
Groby Élelmiszerdiszkont	1 199	475	1 674	0
Haldorádó	1 990	1 055	3 045	0
HáziPatika.com	5 490	900	6 390	0
Játék-Makett.hu	3 696	n.a. [12]	4 551	0
NetPiac Online Áruház	3 790	890	4 680	0
Office Depot	274	2 000	1 329	–945 [13]
Otto	1 990	935	2 925	0
Westel-Webshop	4 831 [14]	0	4 831	0

A vásárló jogai

Az alábbiakban a vásárlók három fontosabb, az internetes vásárlásra vonatkozó jogát mutatjuk be. Ezek a jogok minden vásárlót megilletnek tetszőleges vásárlás esetén.

– *Meg nem rendelt termékek*
– *Megrendeléstől való elállás joga*
– *Vásárlástól való elállás joga.*

Mire figyeljünk a vásárláskor?

Online vásárlásnál – akárcsak a hagyományos üzletekben történő vásárlásnál – érdemes odafigyelni néhány dologra

– *Nézzük meg a bolt adatait, elérhetőségét*
– *Olvassuk el az üzletszabályzatot*
– *A rendelés előtt tájékozódjunk a szállítási és fizetési feltételekről*
– *Győződjünk meg személyes adataink védelméről*
– *Mielőtt rendelésünket véglegesítjük, győződjünk meg a teljes költségről*
– *Őrizzük meg a rendelés adatait*

Praktikus tanácsok

– *Az internetes vásárlás nem feltétlenül olcsóbb, csak más előnyei vannak a hagyományos vásárláshoz képest.*
– *Ha információt gyűjtünk, a legjobb megoldás az internet*
– *Az interneten rengeteg termékinformáció található*
– *Nem a méret számít*
– *Ha valami nem világos, kérdezzünk*
– *Célszerű munkahelyi címet adni, hogy napközben kiszállíthassák a terméket.*
– *Rendeljük meg idejében a terméket*
– *Használjuk ki a kedvezményeket*

Fogalommagyarázat

– *adatkezelési nyilvántartási azonosító:* Az Adatvédelmi Törvény rendelkezései szerint minden adatkezelést be kell jelenteni az adatvédelmi nyilvántartásba, a nyilvántartási számot az adatkezelő (jelen esetben a kereskedő) ekkor kapja.

- *B2C (Business to Consumer):* A B2C rövidítés a fogyasztói elektronikus kereskedelmet, e-kiskereskedelmet jelöli. Termékek vagy szolgáltatások interneten történő értékesítése a fogyasztók felé. Az e-boltok, e-áruházak, netplázák a B2C piac szereplői.
- *bankkártyás fizetés:* A bankkártyás fizetés lényege, hogy a vásárló a megrendelés időpontjában, bankkártyája adatainak megadásával a megrendelt áruk ellenértékét kiegyenlíti.
- *cookie:* A cookie, vagy más néven süti, a kereskedő által a vásárló számítógépére küldött rövid adatsor, amely lehetővé teszi, hogy a későbbi vásárlások során az e-bolt a vevőt automatikusan felismerje és egyéb kényelmi szolgáltatást biztosítson számára. A cookie-kat csak az a honlap tudja olvasni, amelyik elhelyezte.
- *e-bolt:* Termékek értékesítésével foglalkozó internetes honlap. Az e-boltokban a vásárlás teljes folyamata az információnyújtástól a termék megrendeléséig az interneten keresztül valósul meg.
- *elállás joga:* A vásárló a megrendelt terméket a termék átvételétől számított 8 munkanapon belül visszaszolgáltathatja a kereskedőnek, a kereskedő pedig köteles a termék árát harminc napon belül visszatéríteni. A vásárlástól való elállást a vevőnek nem kell indokolnia. A termék visszaszolgáltatásának díját és a termék nem rendeltetésszerű használatából eredő károkat a vásárlónak kell állnia.
- *futárszolgálat:* Csomagok, küldemények kézbesítésére, célbajuttatására vállalkozó társaságok, akik jellemzően a postai kézbesítésnél rövidebb határidővel vállalják a szállítást.
- *hírlevél:* A boltok gyakran kínálják fel a vásárlóknak, hogy e-mailen rendszeresen tájékoztatják őket új termékeikről, akcióikról.
- *hűségprogram:* Több bolt is kedvezményeket biztosít gyakori vásárlóinak, törzsvásárlóinak. Egyes boltoknál a vásárlók hűségpontokat, lojalitás pontokat gyűjthetnek, amiket később termékek vásárlására fordíthatnak, máshol a vásárlók a boltnál elköltött összeg alapján különböző árkedvezményekre jogosultak.
- *kívánságlista:* Néhány e-bolt oldalán a vásárlók a termékeket a kosáron kívül saját kívánságlistájukra is helyezhetik. A kívánságlistába tett termékek adatait a bolt tárolja, így később bármikor megrendelhetők.
- *kosár:* Az internetes kosár lényegét tekintve hasonló a hagyományos bevásárlókosárhoz: online vásárlás során a kiválasztott termékeket lehet beletenni és kivenni belőle. A kosár folyamatosan jelzi a vásárló számára, hogy mit, milyen mennyiségben és milyen összegben válogatott össze a termékek böngészsése során, a vásárlás végén a kosárban található termékek alapján történik a megrendelés.
- *partnerprogram:* A partnerprogram keretében az áruház termékeit kisebb-nagyobb honlapok is hirdetik, amelyek bizonyos százalékban részesednek az általuk generált eladásokból.
- *regisztráció:* Számos e-boltnál alkalmazott megoldás, melynek lényege, hogy a vásárló csak egyszer adja meg adatait (szállítási, számlázási adatok, beállítások stb.), minek után kap egy azonosítót és egy jelszót. További vásárlások során pedig elég megadnia ezt az azonosító-jelszó párost.
- *rendeléskövetés:* A rendelés állapotának nyomon követésével a vásárló megtudhatja, hogy a feladott rendelése megérkezett-e az ügyfélszolgálathoz, a termék raktáron van-e, mikor szállítják.
- *termékfigyelés:* A boltok egy része a vásárló által megadott feltételek alapján értesítést küld új termékeiről.
- *utánvétel:* Magyarországon az elektronikus vásárlások során leggyakrabban alkalmazott fizetési mód. Lényege, hogy a vevő a termék átvételekor, utólag fizet a megrendelt áruért. A vételárat a terméket kiszállító kézbesítőnek, postásnak, futárnak kell átadni.

Irodalomjegyzék

[1] 17/1999. (II.5.) Kormányrendelet a távollévők között kötött szerződésekről
[2] 2001. évi CVIII. törvény az elektronikus kereskedelmi szolgáltatások, valamint az információs társadalommal összefüggő szolgáltatások egyes kérdéseiről (http://www.complex.hu/kzldat/t0100108.htm/t010 0108.htm)
[3] CyberAtlas: E-tailers Will See Green http://cyberatlas.internet.com/markets/retailing/ article/0,,6061_3105491,00.html
[4] GKI Gazdaságkutató Rt., Az online áruházak helyzete, 2002. IV. negyedév
[5] GKI Gazdaságkutató Rt., Az online kereskedelmi áruházak helyzete, 2003. II. negyedév
[6] NRC-TNS, E-kereskedelem és interaktív szolgáltatások, 2003.
[7] Jakob Nielsen: E-Commerce User Experience, 2001.
[8] Hírlevél 2. HTE 2004. február

Újdonságok a számítástechnika és az elektronika világában

News in Computer Technique and Electronics

Dr. BUZÁS Gábor

Babeş Bolyai Tudományegyetem
Kolozsvár

Abstract

Almost every major user in computer technique and electronics is currently applying a great number of new components. This rises a question: whether they really know these modern components and their mode of operation. The present paper is dealing with a relatively few number of new circuits, devices and technologies and it is also describing the main features and improvements they provide. An accurate presentation is not a major goal this time.

The paper evaluates the new features offered by the ASIC-type circuits, in-system programmable eXpanded Programmable Gate Arrays (ispXPGA), Function and Algorithm Specific Circuits (FASIC), Field Programmable Analog Arrays (FPAA), Instant Silicon Solution Platforms (ISSP), Systems on Chip (SOC), digital light processors, CMOS and nanotechnologies and some other components, ideas and features.

Az elektronika és a számítástechnika napjaink műszaki életének valamennyi területén alapvető szerepet kap. Szinte észrevétlenül jutottunk el oda, hogy ezek az ágazatok mindennapjaink szerves részévé váltak.

Ezek keretében az eddigi magyar terminológiai előadások nagyrésze egy-egy kutatási területet, vagy részterületet igyekezett átfogni terminológiai, de főleg szakmai szempontból. Értelemszerűen minden előadó leginkább a szűkebb kutatási területével kapcsolatos témákról beszél/beszélt. Így állhat elő az a helyzet, hogy néha akaratlanul is eltérünk az eredeti célkitűzéstől, azaz a műszaki élet magyar fogalomtárának a minél részletesebb bemutatásától.

Ennek a dolgozatnak az az elsődleges célja, hogy a teljesség igénye nélkül bemutasson néhányat az előadásokban nem szereplő, új és fontos fogalmak közül. Ugyanakkor nem tekinthetünk el e fogalmakhoz kapcsolódó rövid, célratörő magyarázatoktól sem. Ilyen összefüggésben e rövid fogalomtár figyelemfelkeltőként is értékelhető. Ezért kiindulópontként szolgálhat azok számára, akik a következőkben tárgyaltakat részletesebben és magasabb szinten kívánják megismerni. A minél átfogóbb bemutatás és a bemutatás mélysége közötti elkerülhetetlen ellentmondást e sorok szerzője tudatosan vállalja, bízva a hallgató/olvasó megértésében, hogy e néhány oldallal az enciklopédia szerepét nem lehetett és nem is lett volna célszerű felvállalni. E sorokat záró irodalomjegyzék tartalmaz magyar nyelvű szakszótár jellegű munkákat is.

Jelenleg szinte minden nap egy csokorravaló új szakkifejezés jelenik meg a műszaki élet számos területén, és sajnos (főként a szaklapokban) ezeket igen gyakran csak rövidítve használják, holott „friss" fogalmakról van szó, amelyeket csak viszonylag szűk réteg ismer. Arra is fel kell hívni a figyelmet, hogy az esetek többségében az általánosan elfogadott magyar terminológia még hiányzik.

Elsőrendű cél az anyanyelvi fogalomtár bővítése, de az idegen kifejezéseket is megismerni vágyókra gondolva az angol megnevezések is szerepelni fognak, márcsak azért is, mert a rövidítések ezekből származnak.

E dolgozat időszerűségét és célszerűségét abban látom, hogy reményeim szerint hozzájárulhat a gombamód szaporodó új szakkifejezések világában való jobb tájékozódáshoz.

Az említett érveket szem előtt tartva és a hiányosságokat vállalva, a következőkben újabb fogalmak és a hozzájuk kapcsoló rövid magyarázatok bemutatására kerül sor a számítástechnika (hardver), az elektronika és a vonatkozó gyártástechnológia területéről.

Az *alkalmazásorientált áramkörök* (ASIC – Application Specific Integrated Circuit) elterjedt típusai a *mezőprogramozható kapumátrixok* (FPGA – Field Programmable Gate Array) és ennek a *törölhető-újraírható* változata az EPGA (Erasable PGA). E területen a legújabb termék a rendszeren belül programozható kiterjesztett kapumátrix (ispXPGA – in system programmable eXpanded Programmable Gate Array). Ezek ún. *memó-*

riaalapú kapumátrixok, amely azt jelenti, hogy a bekapcsolás pillanatában az alkalmazási felületre telepített csak olvasható tárolóból az óhajtott konfigurációnak megfelelő tartalom az eszköz belső tárolójába töltődik. Ezt követően a belső tároló vezérli az összeköttetéseket létesítő logikát, vagyis a kapcsolómátrixokat. Az átprogramozás tehát a csak olvasható tároló újraírására korlátozódik. Az eljárás kényelmes, gyors és nagyfokú rugalmasságot biztosít.

Ugyanebbe az áramkörcsaládba tartozik az *alkalmazásorientált adó-vevő áramkör* (ASTARIC – Application Specific TrAnsmit and Receive Integrated Circuit), amelynek a megnevezése lényegében tarlamazza az alkalmazási területet is. Kifejlesztését a megnövekedett egyedi adó-vevő feladatokhoz való gyors alkalmazhatóság tette szükségessé.

A mezőprogramozható áramkörök időközben átlépték a digitális elektronika és a számítástechnika kereteit és nemrég megjelentek az első *mezőprogramozható analóg mátrixok* (FPAA – Field Programmable Analog Array). Ezeket különleges analóg igények kielégítésére kínálják, vagy például olyan műveleti erősítők megvalósítását teszik lehetővé, amelyek egyes paraméterei dinamikusan újrakonfigurálhatók akár szoftver eszközökkel is.

A jelfeldolgozás területén robbanásszerűen jelentkezett és terjedt el a *funkció és algoritmusorientált áramkör* (FASIC – Function and Algorithm Specific Integrated Circuit). Ez egy részben programozható, másrészt fix alegységeket tartalmazó integrált struktúra, amely egyes szaklapok szerint máris 52 %-ban tört be a jelfeldolgozási piacra (2003). Összehasonlításképp megemlítjük, hogy a hagyományos *digitális jelprocesszor* (DSP – Digital Signal Processor) 34 %-al van jelen az említett piacon. Lényegében ugyanezt a piacot célozta meg a RISC processzor (Reduced Instruction Set Computer) és a digitális jelprocesszor tulajdonságait ötvöző architektúra, az MSA (Micro Signal Architecture).

Tudni kell, hogy manapság annyira elszaporodtak az alkalmazásorientált áramkörök, hogy egyesek közülük szabványossá váltak (*szabványos alkalmazásorientált termék*, ASSP – Application Specific Standard Product).

Igen elterjedtedt, főként az általános célú alkalmazásokban (pl. kijelzők, televízió készülékek) a videó jelfeldolgozásra optimalizált videoprocesszorok (VISP – VIdeo/VIsual Signal Processor, VPU - Visual Processing Unit). Ezeket általános média processzorként is forgalmazzák.

Itt említhető meg a processzorok általános teljesítményére jellemző újabb paraméter, a *millió szorzás-akkumulátor ciklus* (MMAC – Million Multiply-Accumulate Cycle), ez részben a másodpercenkénti lebegőpontos műveletek millióit (MFLOPS – Million Floating Point Operations per Second) helyettesíti, illetve egészíti ki.

Az elmúlt két évben hallatlan gyorsasággal robbant be főleg az automatizálás világába az ún. *egytokos rendszer* (SOC – System On Chip). Az egytokos számítógép már régebb ismeretes volt, de ezúttal igen bonyolult, általában feladatorientált rendszereket integráltak egy tokba. Az egytokos rendszer felbecsülhetetlen értékű a *beágyazott rendszerek* (Embedded System) számára. A beágyazott rendszer a vezérlés „beágyazását" jelenti a feladatvégző berendezésbe. Az alkalmazások tekintetében a lehetőségek száma végtelen, különösen a beágyazott vezérlési feladatok kivitelezésében kínálkozik további útkeresés a közeljövőben.

Hasonló elképzelést valósít meg az *azonnali megoldásokat kínáló szilícium platform* (ISSP – Instant Silicon Solution Platform). A különbség az, hogy itt a rendszert egy „félkész" szilícium morzsán rövid idő alatt valósítják meg, nem pedig nulla szintről kiindulva mint az egytokos rendszer esetében. Még időszerűtlen az ISSP elterjedéséről beszélni és a tervezőket máris az foglalkoztatja, hogy a programozhatóság irányába fejlesszék tovább.

A fent említett berendezések (bátran használhatjuk a berendezések szót) nyilván nem nagy sorozatban kerülnek forgalomba. Ennek megfelelően a költségük elég magas, és esetükben számolni kell egy bizonyos *meg nem térülő*, vagy esetleg később részben megtérülő *mérnöki munkával* (NREC – Non Recurring Engineering Cost). Ezek ellensúlyozására igyekeznek többször (más tokokban is) felhasználható modulokat kialakítani, viszont így elkerülhetetlenül megjelennek redundáns elemek is.

Ugyancsak vezérlési feladatok ellátására fejlesztették ki a *személyesített vezérlő processzort* (CCP – Customised Control Processor). Megjegyzendő, hogy gyakran a fogalmak egybefolyásának vagyunk tanúi, azaz más-más munkákban, különböző szerzők különböző megnevezéssel illetik lényegében ugyanazt az alkatrészt, eszközt, vagy berendezést. Jelen esetben a személyesített vezérlő processzor többnyire egy *mikrovezérlő* (microcontroller), vagyis valamely vezérlési feladatra, feladatcsoportra optimalizált mikroprocesszor, de az is előfordul, hogy a felhasználó által megszabott egyedi termékről van szó.

Az egytokos rendszer gyakran *elektro-mechanikus mikrorendszer* (MEMS – Micro Electro-Mechanical System). E megnevezés azt jelöli, hogy tokon belül mechanikai komponenseket is létrehoztak. Ezek között ma már szinte minden elképzelhető. Akár hihetetlennek is tűnhet, hogy miket valósítottak meg a tervezők és kivitelezők e téren rendkívül rövid idő alatt. Ennek alátámasztására álljon itt példaként, hogy egy kongresszuson olyan tokot mutattak be, amelybe működőképes mikrokerékpárt integráltak kizárólag szilíciumból. Természetesen ez csak a lehetőségeket példázta, a megvalósítások ennél sokkal komolyabbak.

Az elektro-mechanikus mikrorendszer egyik igen jelentős képviselője a *digitális mikrotükör eszköz* (DMD – Digital Micromirror Device). Az eszköz tükrei szilícium mikrolapkák, amelyeket a piezoelektromos hatáson alapuló digitális vezérléssel lehet különböző irányba állítani. A kapcsolási ideje lényegesen jobb, mint a hagyományos optocsatolóké, ezért azt valószínűsítik, hogy hamarosan uralkodó szerepe lesz a száloptikán továbbított jelek csatolásában. Ugyancsak fontos szerepük lesz az *optoelektronikus integrált áramkörökben* (OEIC – OptoElectronic Integrated Circuit), legalábbis amennyiben beteljesül a rövid időn belüli elterjedésükre vonatkozó jóslat. Ez olyan, egyelőre különlegesnek mondható integrált áramkör, amelyben a logikai funkciókat fényimpulzusok segítségével valósítják meg. Mint minden optikai rendszer, e mikrorendszer is három alapvető összetevőből áll: a fényforrásból, a fénydetektorból és az átviteli közegből. Paradox módon, az optoelektronikus integrált áramkörök technológiájában nem a fényforrás és a detektor miniatürizálása okozza a legnagyobb gondot, hanem a fénysugár modulálása, amelynek hiányában bizonyos logikai feladatok nem valósíthatók meg. A modulálásra azonban ilyen szinten még nincs kikristályosodott optikai vagy elektronikai módszer. A másik számottevő probléma onnan ered, hogy a szilícium hordozón nehezen alakíthatók ki gallium-arzenid (GaAs) struktúrák, márpedig a nagysebességű optoelektronikai eszközök számára ez a legmegfelelőbb alapanyag. Ugyanakkor úgy tűnik, hogy a nagybonyolultságú processzorok számára egyelőre még a MOS technológia a legmegfelelőbb. Két külön struktúra kialakítása és összekötése szintén nehézségeket szülne. Mindezek ellenére a szupergyors és olcsó gépek létrehozására való törekvés, a külvilággal való jobb kapcsolat biztosítása, az új belső architektúrák létrehozása az optoelektronikus integrált áramkörökök látványos fejlesztését sürgeti.

Minthogy az imént az optoelektronika alkalmazásairól esett szó, feltétlenül említést érdemel, hogy az optoelektronika és a digitális feldolgozás ötvözeteként már létrejött a *digitális fényprocesszor* (DLP – Digital Light Processor).

A nagybonyolultságú áramkörök (pl. processzorok, feladatorientált áramkörök, stb.) teljesítményfelvétele rendszerint számottevően nagyobb, mint a hagyományos integrált áramköröké. Ma már minden ilyen áramkörtől, illetve az adó-fogadó eszközöktől megkövetelik az ún. „*zöld*” üzemmód (Green Idle) meglétét. Ez azt jelenti, hogy tevékenység hiányában minimálisra csökkenti a fogyasztást, ugyanakkor szükség esetén azonnal képes tevékenységét folytatni ugyanolyan hatásfokkal.

Az elektronika, a számítástechnika és a távközlés korszerű, népszerű és ennélfogva igen elterjedt megvalósítása a mobiltelefon (GSM – Global Satellite Mobile system). Ennek az átviteli sebesség tekintetében továbbfejlesztett változata az EDGE (Enhanced Data rate for GSM Evolution). Az EDGE átmenetet képez a GSM és az *általános csomagkapcsolt rádiótávközlési rendszer* (GPRS – General Packet Radio System) között. A GPRS csomagkapcsolt technológia a GSM hálózatok továbbfejlesztésére, amelynek főbb szolgáltatásai között a mobil adattovábbítás, a gyors internet, az e-mail és a mozgókép továbbítása szerepel. Az EGPRS (Enhanced GPRS) főleg a továbbítását tökéletesítette.

Itt kell megemlíteni a *drótnélküli helyi hálózatot* (WLAN – Wireless Local Area Network), amely már eddig is létezett, de széleskörű elterjedése most van kibontakozóban.

Ezek után tekintsük át az újabb alkatrészek néhány képviselőjét is.

Úgy tűnik, hogy a jólismert galliumarzenid-alapú *világítódiódákat* (LED – Light Emitting Diode) hamarosan a *szerves* (OLED – Orgnic LED), vagy *polimér* (PLED - Polymer LED) vegyületekből készült társaik fogják felváltani. Szerkezetük egyszerűbb, fogyasztásuk kedvezőbb az előállítás költsége pedig olcsóbb. Már készült ilyen típusú „óriás” LED, amely világításra is alkalmas, de az élettartama egyelőre nem megfelelő.

Új, véletlen hozzáférésű memóriatípus a *ferroelektromos RAM* (FRAM – ferroelelctric RAM), amelynél a tárolás belső domének irányítottságán alapszik. Kedvező tulajdonságai ellenére (pl. alacsony fogyasztás, integrálhatóság) egyelőre nem terjed.

Ezzel szemben lendületet kapott az integrált *felületi akusztikai hullámokat* (SAW – Surface Acoustic Wave) kihasználó eszközök létrehozása. Akusztoelektronikus eszköz (AED – AcoutoElectronic Device) alatt olyan elektronikus feladatot ellátó elemet értünk, amelynek a működése szilárd testbeni akusztikai hullám és elektromos töltések kölcsönhatásán alapszik. Az akusztoelektronikus eszközök a felületi akusztikai hullám és egy piezoelektromos hordozóra felvitt félvezető réteg vezetési elektronjai közötti kölcsönhatást hasznosítják. A legtöbb ilyen eszköz két rétegből áll, a piezoelektromos hangvezetőből, amelynek a felületén a rugalmas hullám terjed és az erre felvitt félvezetőből, amelyben külső tápfeszültséggel elektronáramot hoznak létre. A piezoelektromos hordozó leggyakrabban kvarc, vagy litium-nióbiumoxid ($LiNbO_3$), a félvezető pedig szilícium, vagy indium-antimonid (InSb). Az akusztoelektronikus eszközöket analóg feladatokra alkalmazzák, leggyakrabban szűrő, vagy erősítő funkciót látnak el, de jelfeldolgozó feladatokat is megvalósíthatnak. Az ilyen erősítők hatásfoka kicsi, de ezt a hátrányt kiegyenlítheti az az előny, hogy nemcsak jól elválasztják a bemeneti és kimeneti köröket, hanem széles sávban jelentős erősítést is nyújtanak.

Lássuk most, melyek az újdonságok az anyagok és a gyártástechnológia területén.

Először is, egyre nyilvánvalóbb, hogy a CMOS (Complementary Metal Oxide Semiconductor) technológia kezd uralkodó jelleget kapni, az eszközök 75–80 %-a erre alapul. A CMOS technológián belül is van már továbblépés, éspedig kidolgozták már a *továbbfejlesztett javított tulajdonságokkal rendelkező implantált CMOS* eljárást (EPIC – Enhanced Performance Implanted CMOS). Egyébként jelenleg legalább 4–5, egymástól kissé eltérő CMOS technológia van az elterjedés emelkedő szakaszában. A fejlesztések és módosítások főleg a gyors működés és a minél alacsonyabb tápfeszültség használatát tűzték ki célként.

A bipoláris és a CMOS eljárás ötvözése ugyanazon tokban (BiCMOS) ma már közhelynek számít.

Igen ígéretes és új alapanyag a *szilíciun karbid* (SIC – Silicon Carbide), de innen az is kiderül, hogy nem tudunk (egyes jóslatok szerint még jó ideig) lemondani a szilícium használatáról. Jelenleg nagyteljesítményű és nagyfrekvenciás alkalmazások számára készülnek alkatrészek szilíciumkarbid felhasználásával.

A korszerű integrált áramkörök akár többmillió belső alkatrészt tartalmaznak. Értelemszerű, hogy ennél az alkaltrészsűrűségnél megvalósításuk végett tervezőprogramokat és általában célorientált és magasfokú tervezés automatizálást kell alkalmazni. Ennek megnevezésére használják az *elektronikus tervezés automatizálást* (EDA – Electronic Design Automation) mint gyűjtőfogalmat.

Az elektronika mai eszközeit (pl. tranzisztor, integrált áramkör) a gyártástechnológia során egy félvezető tömbben hozzák létre, vagy egy tömbből „faragják" le.

Napjaink fizikájának egyik igen divatos ága a *nanotechnológia*. A nanotechnológia felhasználásával közelinek jósolják azt a pillanatot, amikor az alkatrészeket már nem egy tömbből hozzák létre, hanem atomokból, molekulákból valósítják majd meg minden eddiginél kisebb méretekben. A nanotechnológia egyelőre a szénre alapul, ezért is foglalkoznak a kutatók olyan behatóan a szilícium karbiddal.

Akár külön munkában is foglalkozhatnánk az alkatrészek, vagy bonyolult berendezések tesztelésével. Minthogy ez csak közvetve kapcsolódik a megfogalmazott célunkhoz, mindössze három jellegzetes és elterjedt, viszonylag új eljárásról lesz szó röviden a következőkben.

A párhuzamos aláíráselemzés (PSA – Parallel Signature Analysis) azon alapszik, hogy a tokozatlan integrált áramkör tesztpontjaiba (logikai csomópontok) tesztjeleket injektálnak, és más pontokban vizsgálják a kiváltott hatást. Üzemszerű működés esetén a rendszerbe juttatott jel annak minden pontjában egy jellemző többites konfigurációt (digitális áramköröknél), vagy jelalakot (analóg áramköröknél) eredményez. Ez az adott pontra jellemző „aláírás". Bármely részegység rendellenes működése, amely valamely logikai csomóponthoz kapcsolódik, a minta megváltozását, tehát hamis aláírást vált ki.

Egy tervezőgárda (JTAG – Joint Test Active Group) olyan módszert dolgozott ki, amely segítségével kész rendszerek/áramkörök üzemszerű működése ellenőrizhető szabványosított kivezetéseken keresztül.

A modellezés, szimulálás, bejáratás és ellenőrzés korszerű módszere a *hardver hurok* (HIL – Hardware In the Loop). Az eljárás azt jelenti, hogy valamely modellt, vagy kész hardver terméket addig tartanak egy folyamatos zárt ellenőrző hurokban, amíg a működési jellemzők és a megbízhatóság szempontjából tökéletesen kielégíti a megszabott feltételeket. Kidolgoztak egy feladatorientált nyelvet (UML – Unified Modelling Language), amely szoftver oldalról támogatja az említett eljárást.

Végezetül meg kell ismételni, hogy a számítástechnika és az elektronika napjainkban is igen dinamikusan fejlődik. Folyamatosan látnak napvilágot új elvek, megvalósítások és természetesen a hozzájuk kapcsolódó új fogalmak. Ezért e munka csak azt a szerepet töltheti be, hogy az említett területeken a közelmúlt néhány, valószínűleg szubjektíven megítélt újdonságához rövid magyarázatot fűzzön és ezáltal ezekre a figyelmet felhívja.

Irodalom

Horváth L., Pirkó J. (szerkesztők), *A számítógépes világ enciklopédiája* (Kiskapu Kiadó, Budapest, 2001)

Jodál E., *Informatikai alapszókincs* (Cédrus Kiadó, Budapest, 1993)

E. Schoitsch, *Embedded Systems*, ERCIM News, Jan. 2003, p. 10)

J.F.Groote, *Master Course Embedded Systems*, Eindhoven University of Technology, 2003

http://www.analog.com

A VHDL kódtól az FPGA-ba való ágyazásig

From the VHDL Code to the Implementation to FPGA-s

KIREI Botond Sándor
Kolozsvár

Abstract

The purpose of the VHDL hardvare describing language is to descibe the behaviour of digital circuits. This technology is suitable for designing, simulating and implementing digital circuits and systems. In order to make the design of the digital circuit successful, we have to be familiar with the design flow, which drives us to the digital circuit's netlist. This paper is briefly presenting the possibilities and the syntax of VHDL, and the design flow. In spite of the wide possibilities of the VHDL any VHDL code cannot be implemented to FPGAs, the limits are contoured by the synthesizing software, the size of the FPGA, or the required speed.

Bevezetés

A VHDL hardver leíró nyelv célja, hogy leírja a digitális áramkörök viselkedését. Ez a technológia alkalmas áramkörök és rendszerek tervezésére, szimulálására, illetve fizikai kivitelezésére. Viszont, hogy az áramkörök tervezése sikeres legyen, nagyon fontos, hogy ismerjük azokat a folyamatokat, melyek során a VHDL kódtól elindulva eljutunk az áramkör kapcsolási rajzához. A dolgozat röviden bemutatja a VHDL nyelv szintaxisát, illetve lehetősegeit, és a tervezés folyamatát. A VHDL nyelv szeleskörű lehetőségei ellenére nem minden VHDL kód ágyazható FPGA-ba. Határt szab az áramkör szintetizálását végző szoftver, az FPGA mérete vagy akár az általunk kívánt sebesség.

A VHDL mint leíró nyelv

Mindennapi életünkben számos elektronikai eszköz vesz körül minket, amelyek nem születhettek volna meg a VHDL nyelv nélkül.

Már a '70-és években megjelent az igény a digitális eszközök viselkedésének egyszerű írására, szimulációjára és szintézisére. A '80-as években megjelentek a VLSI (Very Large Scale Integration) integrált áramkörök. Az integrált áramkörökbe egyre több logikai kapu, regiszter vagy ROM fért el, ezért az áramkör által elvégzett feladatok bonyolultabbá váltak. A klasszikus tervezési eljárások elavultak (egy egyszerű szorzó áramkör sematikus úton való megtervezése több mint 30 oldalt vesz igénybe, míg egy szorzó VHDL kódja 6 sor), ezért szükség volt egy alternatív lehetőségre, amely leegyszerűsíti és meggyorsítja a tervezést. Ugyanakkor, megnövekedtet az igény a magasabb szintű tervezésre. A fejlesztők a digitális eszközök viselkedését leíró nyelvben látták ezt az alternatívát.

A VHDL 1981-ben jelent meg, de csak 1987-ben vált nemzetközi szabvánnyá. 1993-ban megjelent a VHDL bővített szabványa, amit 1999-ben és 2001-ben átjavítottak. Mint a nyelv neve is mutatja (VHSIC Hardware Description Language) alkalmas különböző digitális áramkörök, rendszerek és eszközök viselkedésének a leírására.

Szeretném kihangsúlyozni, hogy a VHDL nyelv nem egy klasszikus értelemben vett programozási nyelv, mint a Pascal vagy a Java, hanem egy eszkö,z amivel digitális rendszereket írunk le. Ezt a nyelvet elsősorban áramkörök szimulálására és szintézisére használjuk, nem pedig bonyolult algoritmusok megvalósítására (annak ellenére, hogy a nyelv szintaxisa alkalmas volna erre a feladatra is).

A VHDL nyelvnek voltak előfutárai, ilyen például az ABEL. Ugyanakkor „testvérei" is jelen vannak az ipari tervezésben, pl. a VERILOG leírási nyelv. A VHDL egyszerű szintaxisa, könnyű megérthetősége és kiterjedt lehetőségei miatt didaktikai szempontból nagyon jól bevált.

Röviden a VHDL szintaxisáról

Mivel a VHDL áramkörök vagy rendszerek leírására készült, többszintű absztraktizálást engedélyez. Az absztraktizálás megengedi, hogy egy adott szintű tervezés során, az adott szinten nem lényeges részletek elrejthetővé váljanak. Például így, egy digitális rendszer megtervezhetővé, leellenőrizhetővé, szimulálhatóvá válik anélkül, hogy az adott tervezőcsoport az idejét a rendszer megvalósításával töltené. Egy másik tervezési szinten a rendszer szerkezetét lehet megtervezni, finomítani, majd egy más szinten a rendszer tömbjeinek a megvalósítására lehet koncentrálni.

Innen következik, hogy a VHDL nyelv alapjában véve két féle leírást engedélyez: *strukturált* leírást, amely tömbök vagy áramkörelemek összekapcsolását írja le, és *viselkedési* leírást, amely tömbök működését írja le. A kétféle leírás együtt létezhet egy VHDL kódban.

Amikor kívülről vizsgálunk meg egy áramkört, mi nem látunk mást, mint bemenő és kijövő adatokat. Ennek a leírására használjuk az entitást. Az entitásban határozzuk meg, hogy a mi áramkörünknek milyen I/Ó portjai vannak. Az entitás leírásának a szintaxisa a következő:

```
entity Entitás_neve is
   port (
   <port_deklaraciok>
   );
end Entitás_neve;
```

Ami az áramkörön belül van, azt az áramkör arhitektúrájának nevezzük. Az adott entitásokhoz egy vagy több arhitektúrát is hozzárendelhetünk. Az arhitektúra szintaxisa a következő:

```
architecture Arhitektúra_neve of Entitás_neve is
   <deklaraciok>
begin
   <utasitasok>
end Arhitektúra_neve;
```

Az arhitektúra leírása során két fő modellel találkozunk. Az első a *viselkedés modell*, ahol egy logikai függvénnyel írjuk le az áramkör viselkedését. Egy egyszerű példa:

```
architecture Viselkedés_modell of Entitás_neve is

begin
      E<= (A AND B) OR (C AND D);
end Viselkedés_modell;
```

A második a *gerjesztés modell* (process), aminek a használatával lehetőségünk nyílik az áramkör viselkedésének a szekvenciális leírására. A gerjesztés modell akkor kerül végrehajtásra, amikor egy jel megváltozik a szenzitivitási listában (process (<szenzitivitási lista>)). A parancsok egymás után hajtódnak végre, akárcsak egy klasszikus program futtatásakor.

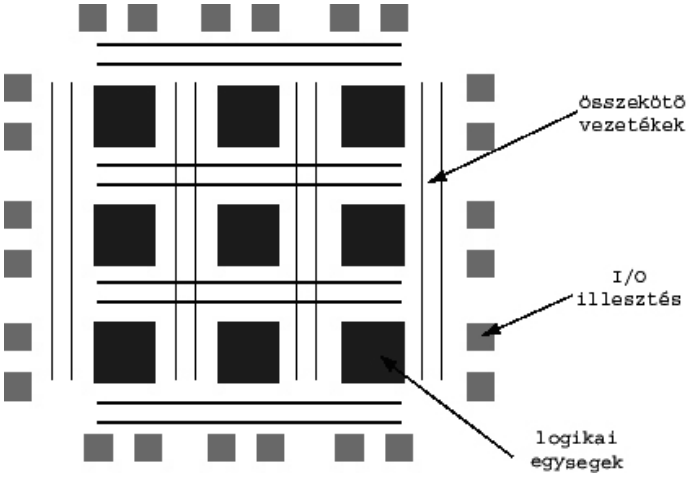
```
architecture gerjesztés_model of szamlaló is
begin
      process (clk)
      variable state:std_logic_vector(7 downto 0);
      begin
            if clk'event and clk='1' then
                  state:=state+'1';
                  szam<=state;
            end if;
      end process;
end gerjesztés_model;
```

Nagyon fontos tudni még a VHDL nyelvről, hogy az arhitektúrában leírt parancsok és gerjesztés modellek párhuzamos módon hajtódnak végre. Az FPGA-ba ágyazott algoritmusok igazi előnye, hogy az utasítások külön szálakon hajtódnak végre, ami igazán gyors adatfeldolgozást tesz lehetővé.

Az FPGA mint áramkör

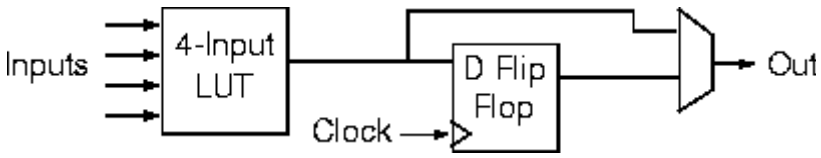
Az ipar fejlődése során egyre bonyolultabb rendszerekre volt szükség. Ezek a rendszerek egyre több logikai kaput, regisztert vagy ROM-ot tartalmaztak. Ekkor született meg Ron Cline[1] ötlete: adjunk a mérnökök keze alá egy olyan eszközt, amit kedvük szerint gyúrhatnak, kedvük szerint konfigurálhatnak. Carnough óta tudjuk, hogy bármilyen logikai függvény megvalósítható ÉS kapukból és VAGY kapukból. Ron Cline két programozható síkot képzelt el: az egyik síkban ÉS kapuk, a másik síkban VAGY kapuk vannak. A síkok összekapcsolásával szinte bármilyen ÉS-VAGY kombináció megvalósítható. Így született meg a SPLA, vagyis a Simple Programmable Logic Array. Ezek az áramkörök azonban csak kombinatorikus áramkörök megvalósítására alkalmasak.

Az FPGA arhitektúráját 1985-ben egy Freeman nevű úriember találta ki. A SPLA arhitektúrájától annyiban különbözik, hogy egyszerű logikai kapuk helyett logikai blokkokat (egységeket) használ. A 1. ábra az FPGA arhitektúrájának egy leegyszerűsített modellje. Az I/O illesztések kapcsolják össze az FPGA-ban található összekötő vezetekéket a külvilággal. Ezek az illesztések teszik lehetővé, hogy az FPGA chip-ek 10-15 gyártási technológiával kompatibilisak. Az összekötő vezetékek segítségével kapcsoljuk össze a logikai egységeket vagy blokkokat.



1. ábra
Az FPGA arhitektúrájának egyszerűsített modellje

Mint már említettük, egy logikai egység nem egy egyszerű logikai kapu, hanem sokkal bonyolultabb arhitektúra. Ezeket a logikai egységeket használjuk a különböző logikai függvények megvalósításához. A 2. ábrán láthatjuk egy logikai egység egyszerűsített arhitektúráját.



2. ábra
A logikai egység egyszerűsített modellje

A LUT vagyis Look Up Table működése hasonlít egy memória működéséhez, és segítségével megvalósítható bármilyen négyváltozós bináris függvény. A D bistabilt a szinkron áramkörök esetében latch-nek, a szekvenciális automaták esetében pedig tárolóelemként használjuk. Megemlítendő, hogy a szekvenciális automaták elkészítése elképzelhetetlen tárlóelemek nélkül. Ezért az FPGA nemcsak kombinatorikus áramkörök, hanem szekvenciális automaták elkészítésére is alkalmas.

[1] Ron Cline a Signetics mérnöke, amit később a Xilinx felvásárolt

A VHDL kód FPGA-ba való ágyazása

Az a tervezési folyamat, amely során a VHDL kódtól eljutunk egy áramköri rajzhoz, amelyet be lehet ágyazni az FPGA chipbe, három lépésből áll.

Leírás

Az első lépés során meghatározzuk, hogy a tervezett áramkör tulajdonképp milyen feladatot kell ellásson. Ennek a lépésnek a végeredménye egy táblázat, amelyben az áramköri kapcsolás van tárolva (milyen logikai kapuk, bistabilok vagy ROMok vannak az áramkörbe és ezek az alkotó elemek hogyan vannak összekötve).

Két dolgot tehetünk. Vagy megépítjük az áramkört logikai kapukból, regiszterekből és ROMokból, vagy leírjuk az áramkört VHDL segítségével. Amennyiben VHDL segítségével írjuk le az áramkört, akkor az áramköri kapcsolást egy szintézis során kapjuk meg. Ezt a szintézist elvégzik helyettünk az erre a célra kifejlesztett szoftverek.

Ellenőrzés

Ebben a lépésben elvégezzük a szimulációkat, és kijavítjuk az esetleges hibákat. A szimulátor is egy szoftver, amely segítségével megkereshetjük a szemantikai hibákat, így egyszerűbbé válik a kód (vagy kapcsolási rajz) kijavítása. Nagyon fontos tudni, hogy az a VHDL kód ami a szimulátoron kifogástalanul viselkedik, sokszor nem implementálható az FPGA-ba.

Implementáció, vagy a chipbe való ágyazás

Ebben a lépésben a leírás eredményeképpen kapott áramköri kapcsolást megvalósítjuk az FPGA-ban található logikai egységek segítségével. Ez egy bonyolult folyamat, mert nemcsak a logikai kapukat kell összekötni, hanem az I/O portokat is meg kell határozni, majd az I/O portokat hozzá kell rendelni az áramköri kapcsolás megfelelő be- és kimeneteleihez, és végül be kell tölteni az adott FPGA-ba. Ezek a lépések már nem platform függetlenek, például figyelembe kell venni az FPGA-ban található logikai blokkok számát. De nem kell megijedni, mert ezt is számítógép végzi el helyettünk.

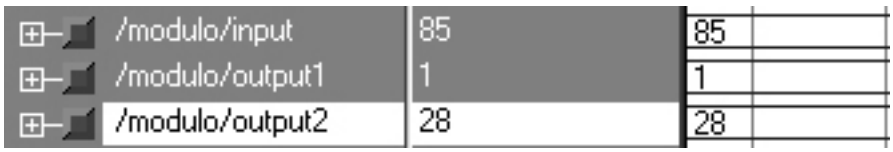
Minden VHDL kód FPGA-ba ágyazható?

A kérdésre a válasz: nem. Csak olyan VHDL kódod implementálhatunk, ami szintetizálható, vagyis biztosan létre lehet hozni olyan áramkört, ami elvégzi a VHDL kód által leírt feladatot.

Egy egyszerű példa:

```
entity moduló is
    Port ( input : in std_logic_vector(7 downto 0);
           output1, output2 : out std_logic_vector(7 downto 0));
end modulo;
architecture Behavioral of moduló is
begin
process (input)
variable szam1,szam2:integer range 0 tó 255;
begin
    szam1:=conv_integer(input);
    szam2:=szam1;
    szam1:=szam1 mod 3;
    szam2:=szam2/3;
    output1<=conv_std_logic_vector(szam1,8);
    output2<=conv_std_logic_vector(szam2,8);
end process;
end Behavioral;
```

A VHDL nyelv megengedi a moduló és az osztás használatát. A fenti program egy olyan digitális eszközt ír le, ami beolvas egy számot, kiszámítja hárommal való osztás hányadosát és maradékát, majd az eredményeket a kimenetre irányítja. Ezt a kódot le lehet szimulálni. A 3. ábrán láthatjuk a szimulálás eredményét.



3. ábra
A szimuláció eredménye

A problémák akkor kezdődnek, amikor megpróbáljuk a kódot szintetizálni. Az általam használt szintetizátor nem enged osztani csak 2 hatványaival. Ahhoz, hogy egy bármilyen számmal való osztást megvalósítsunk, egy szekvenciális automatát (state machine) kell használjunk, ami elbonyolítja a kódot. Valószínű, hogy a komolyabb szintetizáló programok meg tudnak birkózni a problémával.

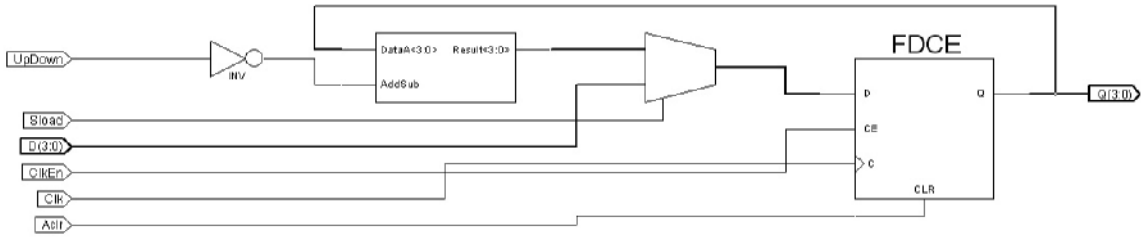
Amikor VHDL kód segítségével tervezünk egy áramkört, mindig arra kell gondoljunk, hogy a kód számára kell léteznie egy megfelelő áramkörnek. Az ilyen típusú tervezésnek van egy pár szabálya, amit jó ha betartunk, különben a szintetizátor nem tudja a kódot „lefordítani" fizikai áramkörbe, vagy pedig hibás áramkört készít.

A következőkben bemutatunk egy VHDL kódot, ami egy tipikus számlálót ír le, majd megvizsgáljuk a szintetizátor által előállított áramkör elemeit.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity szamlalo is
 port ( CLK: in STD_LOGIC;
 RESET: in STD_LOGIC;
 CE, LOAD, DIR: in STD_LOGIC;
 DIN: in STD_LOGIC_VECTOR(3 downto 0);
 COUNT: inout STD_LOGIC_VECTOR(3 downto 0)
 );
end szamlalo;
architecture Behavioral of szamlalo is
begin
process (CLK, RESET)
begin
   if RESET='1' then
      COUNT <= "0000";
   elsif CLK='1' and CLK'event then
      if CE='1' then
         if LOAD='1' then
          COUNT <= DIN;
         else
            if DIR='1' then
               COUNT <= COUNT + 1;
            else
               COUNT <= COUNT - 1;
            end if;
         end if;
      end if;
   end if;
end process;
end Behavioral;
```

Láthatjuk, hogy a számláló viselkedését a gerjesztés modellel írjuk le. Általában a gerjesztés modellek egy clock-ot engednek meg, amit az FPGA-ban tálalható logikai cellák felépítésével magyarázunk. A logikai cellák belső felépítésében találhatunk egy D bistabilt, amelynek a következő pinjei vannak: CLK, RESET, DATA, Q. A kódban a RESET jel mint egy aszinkron jel szerepel, és arra számítunk, hogy ez a jel a bistabil RESET jeléhez lesz kötve. A CLK jel a kódból mint szinkron jel jelenik meg, és valószínűleg a bistabil CLK jelére lesz majd kötve. Az elsif ágon belül található utasítások kombinatorikus áramkörökkel lesznek megoldva, de a kimeneten csak akkor jelennek meg, amikor a bistabil vált.

Most nézzük meg, hogy a szintetizátor milyen áramkört javasol. A 4. ábrán láthatjuk a szintetizátor által előállított áramkört.



4. ábra
A szintetizátor által előállított áramkör kapcsolási rajza

Mint azt láthatjuk, elég jól meg lehet saccolni a szintetizátor által előállított struktúrát, de ehhez természetesen szükséges az FPGA belső felépítésének az ismerete.

Összefoglalás

Amikor egy implementálható kódot akarunk írni, figyelembe kell vegyük az FPGA belső felépítését, mert különben nem leszünk eredményesek. Érdemes megtanulni egy pár VHDL programozási sémát, mert így elérhetjük, hogy gyorsan és könnyen tervezzünk áramköröket.

Sok algoritmus van, amely párhuzamos számítást igényel. Amennyiben ezeket az algoritmusokat sikerül FPGA-ba ágyazni, látványos sebességet érhetünk el. Az, amit a számítógép több ezer művelet alatt végez el, az FPGA chipekben csak pillanatok műve...

Bibliográfia

[1] Hosszú, G., F. Kovács, L. Varga, V. Gajodi (1998): "VHDL based circuit synthesis using language transformations", XI. Polish National Conference, Application of Microprocessors in Automatic Control and Measurement, Warsaw, October 13-14, 1998 pp. 42-48.
[2] J. Birkner: "A very-high-speed field-programmable gate array using metal-tometal antifuse programmable elements," Microelectronics Journal, v. 23, pp. 561-568
[3] Ulrich Heinkel: "The VHDL Reference", John Wiley And Sons, 2000
[4] Karen Parnell, Nick Mehta: "Programmable Logic Design Quick Start Handbook", Xilinx Coorporation, June 2003
[5] Sorin Hintea: "Tehnici de Proiectear cu Arii Logice", UTPress, 2003