

GYARMATI PÉTER

INFORMATIKAI ELEMEEK



TCC COMPUTER STÚDIÓ

Gyarmati Péter

Informatikai elemek

GYARMATI PÉTER

INFORMATIKAI ELEMEEK

VISSZATEKINTÉS A KÖZELMÚLTBA



TCC COMPUTER STÚDIÓ

2010

A könyvben található grafikák a Microsoft
Médiatárból származnak,



míg az írásokhoz kapcsolódó ábrák a szerző
alkotásai.



Más képeknél a forrás, illetve a jog egyenként van
megjelölve.

www.tcc.at.hu
www.gyarmati.tk
www.gyarmati.dr.hu

ISBN 978-963-06-8599-3
Copyright © Gyarmati Péter, 2010

Ajánlom e könyvet

- *mindenekelőtt azoknak, akik Neumann Jánost tekintik tudós és emberi példaképüknek,*
- *a kortárs számítástechnikusoknak, akikkel együtt vettünk részt az elemek alkotásában,*
- *az ifjú informatikusoknak, akik a miértre is kíváncsiak, vagy, hogy felébresszem kíváncsiságukat,*
- *a kaliforniai barátaimnak, az egyetemi kollégáknak,*
- *a számítástudomány iránt érdeklődőknek,*
- *mindenkinek, hogy kideríthessék magukról, érdekli-e őket a számítástudomány,*
- *és végül Rabár Ferenc emlékének.*

~~~~~

*Köszönöm a családomnak és a barátaimnak a biztatást, amely nélkül ez az írás nem született volna meg.*

~~~~~

Sose feledem az „akkor ki fog dolgozni” kifejezést, amellyel majdnem kettétörték tudományos pályafutásomat. Hát, nem rajtuk múltott!

~~~~~







# Informatikai elemek

## TARTALOMJEGYZÉK

|                                              |     |
|----------------------------------------------|-----|
| BEVEZETÉS.....                               | 9   |
| AZ INFORMATIKA HAJNALA.....                  | 13  |
| THE COMPUTER.....                            | 19  |
| Overviev.....                                | 19  |
| Computers at work, applications.....         | 24  |
| Types of computers.....                      | 28  |
| Parts of a computer system.....              | 31  |
| History of the computer.....                 | 45  |
| Virtual reality.....                         | 52  |
| Multimedia.....                              | 54  |
| The infomation superhighway.....             | 57  |
| Artificial intelligence, expert systems..... | 59  |
| The future of computers.....                 | 61  |
| Careers in the computer field.....           | 64  |
| NÉHÁNY VÁRAKOZÓSoros MODELL.....             | 67  |
| A várakozósor leírása.....                   | 68  |
| A modell leírása.....                        | 73  |
| Modellek egy kiszolgáló egységgel.....       | 78  |
| Elsőbbségi, prioritásos modellek.....        | 89  |
| Összefoglalás.....                           | 92  |
| Függelék.....                                | 92  |
| RÁDIÓKOMMUNIKÁCIÓS HÁLÓZATOK.....            | 97  |
| Bevezetés.....                               | 97  |
| Alapvető forgalomszervezési típusok.....     | 98  |
| A véletlen-elérésű forgalom modellje.....    | 102 |
| Kiegészítési és bővítési lehetőségek.....    | 109 |
| Összefoglalás.....                           | 111 |
| Függelék.....                                | 112 |



|                                               |     |
|-----------------------------------------------|-----|
| OPERÁCIÓS RENDSZEREK ADAPTIV VEZÉRLÉSEL.      | 113 |
| Elvek (ADIOS). .....                          | 114 |
| Optimalizáló adaptív szabályozások. ....      | 120 |
| Egyparaméteres szabályozási példa .....       | 123 |
| A mérési funkció. ....                        | 127 |
| A döntési funkció. ....                       | 132 |
| A beavatkozási funkció. ....                  | 135 |
| Erőforrások dinamikus elosztása. ....         | 136 |
| A heurisztikus algoritmus és realizálása ..   | 137 |
| A döntési funkció kiegészítése. ....          | 142 |
| Összefoglalás. ....                           | 150 |
| Függelék. ....                                | 151 |
| AZ ANTIGRAVITÁCIÓ, AVAGY A LEBVACSKA.....     | 157 |
| Hipotézis. ....                               | 157 |
| Bizonyítás. ....                              | 159 |
| Thesis. ....                                  | 159 |
| További vizsgálatok és a jövő feladatai. .... | 160 |
| Megjegyzés. ....                              | 161 |
| BELIEVES ABOUT VIRUSES.....                   | 163 |
| Security ideas. ....                          | 163 |
| The complexity of detecting viruses.....      | 168 |
| Summary.....                                  | 174 |
| PAGE RANKING ALGORITHM .....                  | 175 |
| Bevezetés .....                               | 175 |
| A hálózat .....                               | 179 |
| A kapcsolati mátrix .....                     | 181 |
| A fontossági vektor.....                      | 183 |
| Összefoglalás .....                           | 191 |
| DATA ACQUISITION ÉS A PDA. ....               | 193 |
| a <i>MOBI-X</i> .....                         | 196 |
| a <i>PREFIX-2000</i> .....                    | 197 |
| a <i>Personal IC Memory Card</i> .....        | 198 |
| AZ INFORMATIKA NÉHÁNY NAGY TÉVEDÉSE. ....     | 199 |
| UTÓSZÓ.....                                   | 205 |



*"A fejlődés ellen nincs gyógymód, a tudomány a jövőben inkább a szabályozás és vezérlés, a programozás, az adatfeldolgozás, a kommunikáció, a szervezés és a rendszerek problémáival törődik majd."*  
Neumann János

## Bevezetés

Csodálatos világban élünk a Földünkön, a világ-mindenségünkben.



Ez a világ ontja ránk, mindenkire, egymásra kinézetét, milyenségét, viselkedését, szépségét, - az információit - hol nyíltan, hol rejtőzködve. Az élet, a lét alapvető feltétele, hogy ezeket az

információkat megszerezzük, értékeljük, hogy hasznosíthassuk céljainknak megfelelően.

Információt szerez a virág az esőről, hogy időben becsukhassa szirmait és az oroszlán is, hogy széllel szemben támadhasson, hogy ő érezze a szagot és ne őt „szagolják ki”.

Az ember ősi vágya, ahogy mindent, így az információval kapcsolatos dolgait is, másokkal, mással végeztesse. Így létrejött az emberek közötti kapcsolat, a munkamegosztás, a tudás-



átadás, az emlékezés, a tanítás, vagyis „informatikául” a nyelv, a rajz, az írás, a jelek sokasága. Azután a mérő- és számoló eszközök, valamint az írás különböző tároló eszközei is.

Innen már nem volt, nincs megállás, mert a technika fejlődése új és újabb lehetőségeket ad, amelyeket az evvel foglalkozók felhasználtak, felhasználnak. Fogalomalkotó szokásaink alapján nevezzük ezt *információtechnikának, kommunikációnak, számítástechnikának, számítógépkalkulációnak, algoritmizálásnak, stb., legújabban informatikának.*

Talán ebből is látszik, hogy sokrétű, szerteágazó tudást igénylő dologról van szó, interdiszciplinárisnak mondjuk, amely folytonosan változik, fejlődik, és egyre jobban „szétterül” a különböző tudományok területén.

Vajon a mérnök lesz közgazdász, vagy fordítva, vajon az informatikus válik orvossá vagy az orvos informatikussá a gének kutatásában? A sort ugyancsak folytathatnánk. Azt hiszem merő időpocsékolás ennek vitatása. *Mindenki tegye a dolgát, használja a legújabb ismereteket, eszközöket saját munkájának végzésében.*

Ez a megoldás!

Az informatika fogalmát számosan igyekeztek meghatározni, de, mint a fentiekből is látszik eléggé reménytelen feladat. Elégedjünk meg annyival, hogy az emberiség fejlődésének adott időszakában és problémakörében adunk egy meghatározást, amely megmutatja, hogy az adott helyzetben mit értünk alatta és mely területen fogunk vele dolgozni.



Szóval úgy tűnik sikerült az olvasót eléggé elkeseríteni. Nos, jól van, ez volt a szándékom!

Azért, mert aki *informatikára „adja a fejét”* tudja meg, hogy azonos útra lépett *a jó pappal, aki holtig tanul!*



Vonatkozik ez mindazokra, akik akár az informatika eszközeit fejlesztik, gyártják, kezelik, akár az informatika módszereit, alkalmazásait kutatják, tervezik, használják.

Arra is gondolnunk kell, hogy az ember - tehát az informatikus is - gyarló, hibáktól nem mentes lény, hát még az alkotása.

Ezért a jó informatikus, miként a matematikus, nem hisz, hanem ellenőriz, bizonyít. Tudnunk kell, ahogy a mondás tartja: *a program nem a terveink, hanem az utasításaink szerint működik.*

Ajánlom tehát ezt a művet az ifjabb informatikusoknak ismerkedésül és azért, hogy a tehetségesek bővítsék az informatikai elemek gyűjteményét.

Kelt Szentendrén, 2009 decemberében.

~~~~~


SOME
THOUGHTS
CONCERNING
Education.



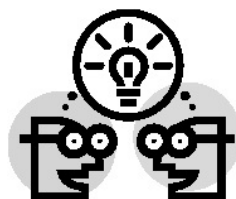
LONDON,

Printed for *A. and J. Churchill,*
at the *Black Swan* in *Pater-*
noster-row, 1693.

„Az informatika egy átfogó társadalmi folyamat kiszolgálója, - az információ alkalmazása a társadalmi szükségletek kielégítésére – amelynek háttere a számítógépes és távközlési technológia.”
Gömbös Ervin, ENSZ

Az informatika hajnala.

A társadalomtudósoknak régóta vizsgálati tárgya az emberek közötti kapcsolatok, azok értelme, oka, célja, módszerei.



A természettudósok a természetes folyamatok, dolgok mesterséges utánpótlását tűzték zászlójukra, mert munkájuk során rengeteget kell számolni, mert emlékezni kell mindenféle megjegyezhetetlen dologra, mert az eredményeiket közölni akarják a többiekkel és a jövővel.

A művészek az általuk ihletett formákat, alakzatokat, képeket, színeket, hangokat, stb. el szeretnék juttatni az emberekhez minél szélesebb körben, hogy ismerjék meg és szeressék alkotásaikat; ismétlődő igényű alkotásaikhoz ismétlődő technikákat alkalmaznak.

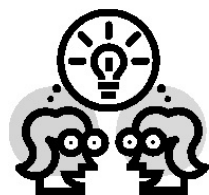
A jelentős különbségek dacára, van ezekben olyan közös rész, amely lényeges, mégpedig az *emberi kapcsolatok analízisének és szintézisének szándéka.*

Az *analízis* magától értetődő, mert a vizsgálat arra irányul, hogy feltárjuk a sokféle kapcsolat közös, jellemző folyamatait, módozatait, megbízhatóságát, hatását, hibáit, egészen az emberi agy ebbéli működéséig.

A *szintézis* alatt olyan eszközök, technikák, módszerek előállítását értjük, amelyek alkalmassak lesznek az analízis során feltártak realizálására.

A tevékenységnek a különálló, eltérő felfogású kutatás, az erők dekoncentráltága szabtak határt és csak egy-egy területen jött létre említésre méltó fejlődés.

Az előrelépést az *interdiszciplináris* közelítés kialakulása - számos terület hasonló problémáinak közös megoldására irányuló szándék - jelentette. A teljesség igénye nélkül álljon itt néhány, a legfontosabbak közül: a könyvtárak tudományos tájékoztatási elmélete, a hadászati célkövetési számítások, a titok kódolás,



népszámlálási adatok feldolgozása, vasúti menetrend készítése, meteorológiai számítások, „esőcsinálás”, csillagászati és más számolás igényes műveletek, a hírközlés csatorna elmélete és matematikai módszere.

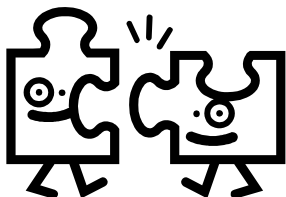
Ílymódon született az új tudományág és egyszerre neve is lett, az *információ* és a *matematika* szavak kombinációjaként, az INFORMATIKA, gyakori rövidítéssel az IT.

Einstein, a speciális relativitáselméletéről szóló

művében írja: *"Egy elmélet legszebb sorsa, ha maga mutat utat olyan többet felölelő elmélethez, amelyben ő maga, mint határeset él tovább."*

Hát ez történt a „hajnali informatikával” is! A kommunikáció matematikai elméletéből, a logikai algebrából, az áramköri kapcsolókból, az elektronikai áramkörökből, kijelzőkből, megjelenítőkből, lyukkártyákból és floppykból, stb. egy magasabb szint alakult ki, *a tárolók-, az adatkezelés elmélete, a kapcsolatok-, a folyamatvezérlés-, a programozás elmélete, a hálózatelmélet*, és ezek újabb szimbiózisaként *a viselkedés vizsgálata az emberek és a gépek együttesében, sőt ezek hálózatának hatalmas és összetett szövevényében.*

Létejt az informatika olyan mellékterméke is, mint a *titkosság és biztonság* eszközei és módszerei a rosszszándékúak- és a szándéktalan rossz tevékenységének kivédésére.



Izgalmas kérdés az informatika hatása a jelen kora és még izgalmasabb a jövőre, utódainkra gyakorolt hatása.

Tudnunk kell mit okozunk mai tevékenységünkkel a jövőnek!

Aztán még ott van *az informatika saját belső fejlődése* is, amely az eddigi tapasztalatok szerint mindig okozott meglepetéseket, nem várt újdonságokat. Joggal feltételezhetjük ennek a tendenciának a folytatódását is.

Tehát a feladatunknak eléggé az elején járunk, van még mit feltárni és a hatásokat elemezni a következmények megismerése érdekében.

Mint már említettem felelősségünk is van, mert az informatikai tudás minden eddigi tudományos eredménynél közvetlenebbül és gyorsabban, hat az emberre, a társadalomra.

Kiemelt felelőssége van az evvel foglalkozóknak, ezért a saját tudományos munkásságuk mellett nagyon fontos azok hatásának vizsgálata is!

Minden próbálkozásnál jobban fejezi ki ezt az összefüggést - korát megelőzve - egyik szellemi nagyságunk, Bólyai János beadványa, amelyet Ferenc József császár és királyhoz intézett „a közjó gyors előmozdítására”: ...*"amint a fizikai világban egy erő, vagy gép értékét csak hatása, vagy teljesítménye szerint jogosult az ember megbecsülni, az erkölcsi világban is alapja és joga van az intézmények értékét azon tömeg és jók szerint itélni meg, amellyel a boldogság birodalmát, vagy egy tökéletes államot és a mindenoldalú boldogságot közelebb hozzák."*

A magam szerény eszközeivel igyekeztem és igyekszem hozzájárulni az informatika tudományához. Ebben a könyvben bemutatok néhány olyan *informatikai elemet*, amelynek megalkotásához valamilyen módon hozzájárultam. Ezen írások némelyike hosszas kutató munka beszámolóiból készült, előadások, ismertetőik anyagai, mások népszerűsítő, ismeretterjesztő szándékkal készültek. Emiatt olvashatóságuk jelentősen különböző, azonban remélem, hogy

az olvasmányosabbak felkeltik eléggé az érdeklődést a nehezebben tanulmányozhatókkal való megbírkózáshoz. A dolgot kissé bonyolítja, hogy néhány írásom, hazai előadása hiján, angolul olvasható, de hát ez ma az informatika „hivatalos nyelve”.

Nyelve a matematika is és talán eljuthatunk még egyszer oda, hogy a matematikai kifejezéseket tartalmazó szöveg nem fekete folt lesz sokak szemében, hanem pusztán egy másik nyelv kifejezései, amelyet éppúgy használunk, mint bármelyik nyelvet.



Egmont Colerus írja művében a matematika népszerűsítésére, hogy *„Belopózott azonban a matematika hétköznapi nyelvünkbe is. Az újságok „középértékekről” írnak, „átlag-hőmérsékletet”, „maximális teljesítményt”, „görbék kritikus pontjait” és „erőtereket” emlegetnek: csupa olyan kifejezés, melyet a köznyelv a matematikától és a matematikai fizikától kölcsönzött. Felesleges, hogy az ilyen szavakat értelmetlen zörejként hallgassuk végig, sőt hatásukra magunkat kisebbértékűnek, esetleg műveletlennek tartsuk. Nagyszerű az ilyen szavak jelentése s ugyanannyira jelképes is, de felfogható és megtanulható.”*

Szerény amatőr matematikusunk ha tudná, hogy mára már hová jutottunk: *meridián, kontextus, korreláció, szignifikancia, kaotikus, mainstream, halmazok, scénárió* és még hosszasan sorolhatnám a kifejezések tömegét, amelyek használatosak a mindennapi életünkben és

vajon mit tudnak, értenek belőle a hétköznapi emberek, sőt - tisztelet a kivételnek – azok is, akik mondják, használják.



Hát ezért feladatunk a népszerűsítés! Lassú, nehéz munka, „nincs királyi út”, ismerjük egy mondából, de ettől még....

Az új kifejezések sokasága informatikai származék. Csak annyit tennék hozzá ezen bölcsességekhez, hogy *könyörgöm, ne emeljék egy újabb „fekete folttal” a művelt emberek és a művelődni vágyók a matematikai alacsonyabbrendűségi komplexusukat!*

Vajon miért büszkék az emberek arra, hogy a kisiskolás gyerekük, unokájuk is jobban ért a számítógéphez?

Kelt Szentendrén, 2009. szeptemberében.

~~~~~



# The Computer.

## A SHORT STORY OF COMPUTERS 1997.

*Von Neumann, John (1903-57), U.S. mathematician, born in Budapest, Hungary; left to U.S. 1930, became citizen 1937; research professor of mathematics at Institute for Advanced Study, Princeton, N.J. Served also the Atomic Energy Commission, 1954-57; did much pioneering work in the areas of logical design of computers, methods of programming, the problem of designing reliable machines using unreliable components, machine imitation of randomness, and the problem of constructing machines that can reproduce their own kind. He must be the ideal for all scientist as his adaptivity, helpfulness, moderation and also his knowledge. This inaugural lecture offered to his memory.*

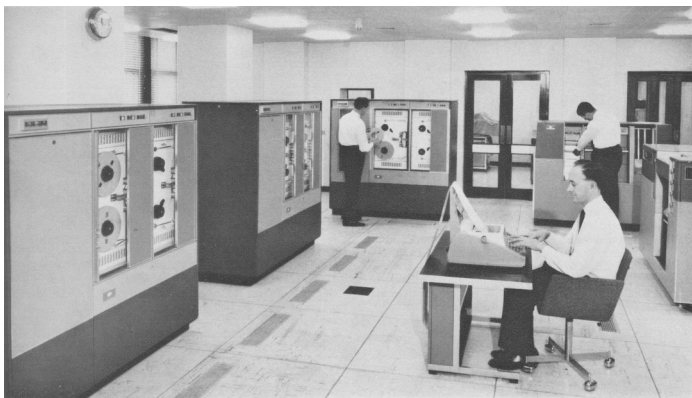
### *Overview.*

Generally, a computer is any device that can perform numerical calculations even an adding machine, an abacus, or a slide rule. Currently, however, the term usually refers to an electronic device that can use a list of instructions, called a program, to perform calculations or to store, manipulate, and retrieve information.

Today's computers are marvels of miniaturization. Machines that once weighed 30 tons and occupied warehouse-size rooms now may weigh as little as 1-2 kilograms and can be carried in a suit pocket. The heart of today's computers are integrated circuits (ICs), sometimes called microchips, or simply chips.



These tiny silicon wafers can contain millions of microscopic electronic components and are designed for many specific operations: some control an entire computer (CPU, or central processing unit, chips); some perform millions of mathematical operations per second (math coprocessors); others can store some ten million characters of information at one time (memory chips).



(ICL 190x by courtesy of DP Journal)

In 1953 there were only about 100 computers in use in the entire world. Today hundreds of millions of computers form the core of electronic products, and more than 100 million programmable computers are being used in homes, businesses, government offices, and universities for almost every conceivable purpose.

Computers come in many sizes and shapes. Special-purpose, or dedicated, computers are designed to perform specific tasks. Their



operations are limited to the programs built into their microchips. These computers are the basis for electronic calculators and can be found in thousands of other electronic products, including digital watches (controlling timing, alarms, and displays), cameras (monitoring shutter speeds and aperture settings), and automobiles (controlling fuel injection, heating, and air conditioning and monitoring hundreds of electronic sensors).

General-purpose computers, such as personal computers and business computers, are much more versatile because they can accept new sets of instructions. Each new set of instructions, or program, enables the same computer to perform a different type of operation. For example, one program lets the computer act like a word processor, another lets it manage inventories, and yet another transforms it into a video game.

Although some general-purpose computers are as small as pocket radios, the smallest class of fully functional, self-contained computers is the class called notebook computers. These usually consist of a CPU, data-storage devices called disk drives, a liquid-crystal display (LCD), and a full-size keyboard all housed in a single unit small enough to fit into a briefcase.

Today's desktop personal computers, or PCs, are many times more powerful than the huge, million-dollar business computers of the 1960s and 1970s. Most PCs can perform from 16 to 66 million operations per second, and some can even perform more than 100 million. These



computers are used not only for household management and personal entertainment, but also for most of the automated tasks required by small businesses, including word processing, generating mailing lists, tracking inventory, and calculating accounting information.

Minicomputers are fast computers that have greater data manipulating capabilities than personal computers and can be used simultaneously by many people. These machines are primarily used by larger businesses to handle extensive accounting, billing, and inventory records.

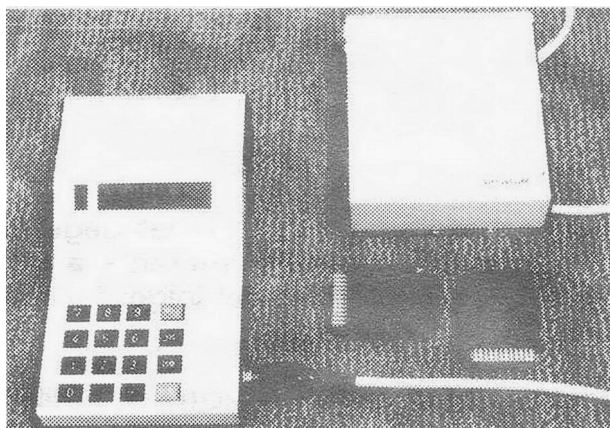
Mainframes are large, extremely fast, multi-user computers that often contain complex arrays of processors, each designed to perform a specific function. Because they can handle huge databases, can simultaneously accommodate scores of users, and can perform complex mathematical operations, they are the mainstay of industry, research, and university computing centers.

The speed and power of supercomputers, the fastest class of computer, are almost beyond human comprehension, and their capabilities are continually being improved. The most sophisticated of these machines can perform nearly 32 billion calculations per second, can store a billion characters in memory at one time, and can do in one hour what a desktop computer would take 40 years to do. Supercomputers attain these speeds through the use of several advanced engineering techniques. For



example, critical circuitry is supercooled to nearly absolute zero so that electrons can move at the speed of light, and many processors are linked in such a way that they can all work on a single problem simultaneously. Because these computers can cost millions of dollars, they are used primarily by government agencies and large research centers.

Computer development is rapidly progressing at both the high and the low ends of the computing spectrum. On the high end, by linking together networks of several small computers and programming them to use a language called *Linda*, scientists have been able to outperform the supercomputer. This technology is called parallel processing and helps avoid hours of idle computer time. A goal of this technology is the creation of a machine that could perform a trillion calculations per second, a measure known as a teraflop.



(MOBI-X Copyright © P.G.Gyarmati, 1981)



On the other end of the spectrum, companies like Apple and Compaq are developing small, handheld personal digital assistants (PDAs) based on a hungarian copyright. The Apple Newton, for example, lets people use a pen to input handwritten information through a touch-sensitive screen and to send mail and faxes to other computers. Researchers are currently developing microchips called digital signal processors, or DSPs, to enable these PDAs to collect and recognize analog signals. This development will make possible more professions to use a computer quickly and easily. It is a promise to lead to a revolution in the way humans collect, and transfer information.

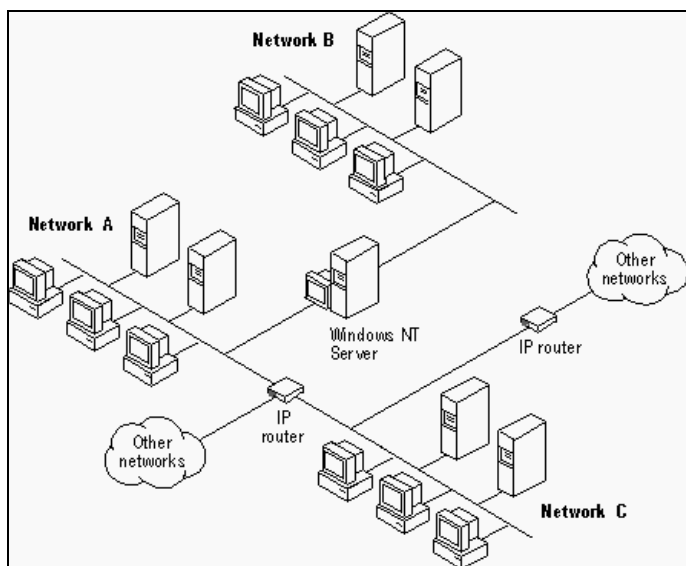
### *Computers at work, applications.*

#### *Communication.*

Computers make all modern communication possible. They operate telephone switching systems, coordinate satellite launches and operations, help generate special effects for movies, and control the equipment in all phases of television and radio broadcasts. Local-area networks (LANs) link the computers in separate departments of businesses or universities, and larger networks, such as the Internet, permit modems telecommunication devices that transmit data through telephone lines to link individual computers to other computers, or LAN's anywhere in the world - called wide-area-networks (WAN).



Journalists and other writers now use word processors to write books and articles, which they then submit to publishers on magnetic disks or through telephone lines. The data may then be sent directly to computer-controlled typesetters, some of which actually design the layout of printed pages on computer screens.



(Example inter network with server and router)

### *Science and research.*

Computers are used by scientists and researchers in many ways to collect, store, manipulate, and analyze data. Running simulations is one of the most important applications. Data representing a real-life system is entered into the computer, and the computer manipulates the data in order to show



how the natural system is likely to behave under a variety of conditions. In this way scientists can test new theories and designs or can examine a problem that does not lend itself to direct experimentation. Computer-aided design, or CAD, programs enable engineers and architects to design three-dimensional models on a computer screen. Chemists may use computer simulation to design and test molecular models of new drugs. Some simulation programs can generate models of weather conditions to help meteorologists make predictions. Flight simulators are valuable training tools for pilots.

### *Industry.*

Computers have opened a new era in manufacturing and consumer-product development. In the factory, computer-assisted manufacturing, or CAM, programs help people plan complex production schedules, keep track of inventories and accounts, run automated assembly lines, and control robots. Dedicated computers are routinely used in thousands of products ranging from calculators to airplanes.

### *Government.*

Government agencies are the largest users of mainframes and supercomputers. The United States Department of Defense uses computers for hundreds of tasks, including research, breaking codes, interpreting data from spy satellites, and targeting missiles. The Internal Revenue Service uses computers to keep track of tens of millions of tax returns. Computers are



also essential for taking the census, maintaining criminal records, and other tasks.

### *Education.*

Computers have proved to be valuable educational tools. Computer-assisted instruction, or CAI, uses computerized lessons that range from simple drills and practice sessions to complex interactive tutorials. These programs have become essential teaching tools in medical schools and military training centers, where the topics are complex and the cost of human teachers is extremely high. Educational aids, such as some encyclopedias and other major reference works, are available to personal-computer users either on magnetic disks or optical discs or through various telecommunication networks.

### *Arts and Entertainment.*

Video games are one of the most popular applications of personal computers. The constantly improving graphics and sound capabilities of personal computers have made them popular tools for artists and musicians. Personal computers can display millions of colors, can produce images far clearer than those of a television set, and can connect to various musical instruments and synthesizers. Painting and drawing programs enable artists to create realistic images and animated displays much more easily than they could with more traditional tools. „Morphing” programs allow photographers and filmmakers to transform photographic images into any size and shape



they can imagine. High-speed supercomputers can insert life-like animated images into frames of a film so seamlessly that movie-goers cannot distinguish real actors from computer-generated images. Musicians can use computers to create multiple-voice compositions and to play back music with hundreds of variations. Speech processors even give a computer the ability to talk and sing.

### *Types of computers.*

There are two fundamentally different types of computers analog and digital. (Hybrid computers combine elements of both types.) Analog computers solve problems by using continuously changing data (such as pressure or voltage) rather than by manipulating discrete binary digits (1s and 0s) as a digital computer does. In current usage, the term computer usually refers to digital computers. Digital computers are generally more effective than analog computers for four principal reasons: they are faster; they are not as susceptible to signal interference; they can convey data with more precision; and their coded binary data are easier to store and transfer than are analog signals.

### *Analog computers.*

Analog computers work by translating continuously changing physical conditions (such as temperature, pressure, or voltage) into corresponding mechanical or electrical quantities. They offer continuous solutions to



the problems on which they are operating. For example, an automobile speedometer is a mechanical analog computer that measures the rotations per minute of the drive shaft and translates that measurement into a display of km per hour. Electronic analog computers in chemical plants monitor temperatures, pressures, and flow rates and send corresponding voltages to various control devices, which, in turn, adjust the chemical processing conditions to their proper levels.

### *Digital computers.*

For all their apparent complexity, digital computers are basically simple machines. Every operation they perform, from navigating a spacecraft to playing a game of chess, is based on one key operation determining whether certain switches, called gates, are open or closed. The real power of a computer lies in the speed with which it checks these switches anywhere from 1 million to 4 billion times, or cycles, per second.

A computer can recognize only two states in each of its millions of circuit switches on or off, or high voltage or low voltage. By assigning binary numbers to these states 1 for on and 0 for off, for example and linking many switches together, a computer can represent any type of data from numbers to letters to musical notes. This process is called digitization.

Imagine that a computer is checking only one switch at a time. If the switch is on, it symbolizes one operation, letter, or number; if



the switch is off it represents another. When switches are linked together as a unit, the computer can recognize more data in each cycle. For example, if a computer checks two switches at once it can recognize any of four pieces of data one represented by the combination off-off; one by off-on; one by on-off; and one by on-on. The more switches a computer checks in each cycle, the more data it can recognize at one time and the faster it can operate. Below are some common groupings of switches (each switch is called a binary digit, or bit) and the number of discrete units of data that they can symbolize:

4 bits = 1 nibble (16 kind of data);

8 bits = 1 byte (256 kind of data);

16 bits = 1 word (65 536 kind of data).

32 bits = 1 double word (4 294 967 296 kind of data).

A byte is the basic unit of data storage because all characters, numbers, and symbols on a keyboard can be symbolized by using a combination of only eight 0s and 1s.

Each combination of ons and offs represents a different instruction, part of an instruction, or type of data (number, letter, or symbol). For example, depending on its context in a program, a byte with a pattern of 01000001 may symbolize the number 65, the capital letter A, or an instruction to the computer to move data from one place to another. The all possible variations of these patterns called codes represents the whole set of symbols, letters, numbers, etc. what a computer could process.



### *Parts of a computer system.*

A digital computer is a complex system of four functionally different elements a *central processing unit*, *input devices*, *memory-storage devices*, and *output devices* linked by a communication network, or bus. These physical parts and all their physical components are called hardware.

Without a program, a computer is nothing but potential. Programs, also called software, are detailed sequences of instructions that direct the computer hardware to perform useful operations.

Computers may also have other communication facilities, called either LAN, local area network, or WAN, wide area network, or internetwork.

### *Hardware.*

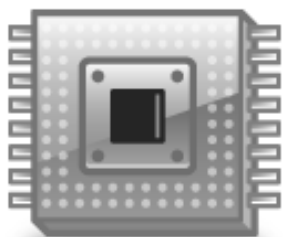


### *The central processing unit, or CPU.*

The central processing unit, or CPU, is the heart of a computer. In addition to performing



arithmetic and logic operations on data, it times and controls the rest of the system. Mainframe CPUs sometimes consist of several linked microchips, each performing a separate task, but most other computers require only a single microprocessor as a CPU.



Most CPU chips and microprocessors have four functional sections: (1) arithmetic and logic unit, which performs arithmetic operations (such as addition and subtraction) and logic operations (such as

testing a value to see if it is true or false);

(2) temporary storage locations, called registers, which hold data, instructions, or the results of calculations;

(3) the control section, which times and regulates all elements of the computer system and also translates patterns in the registers into computer activities (such as instructions to add, move, or compare data); and

(4) the internal bus, a network of communication lines that links internal CPU elements and offers several different data paths for input from and output to other elements of the computer system.

### *The input devices.*

Input devices let users enter commands, data, or programs for processing by the CPU.



Computer keyboards, which are much like typewriter keyboards, are the most common input devices. Information typed at the keyboard is translated into a series of binary numbers that the CPU can manipulate. Another common input device, the mouse, is a mechanical or optomechanical device with buttons on the top and a rolling ball in its base. To move the cursor on the display screen, the user moves the mouse around on a flat surface. The user selects operations, activates commands, or creates or changes images on the screen by pressing buttons on the mouse. Other input devices include joysticks and trackballs. Light pens can be used to draw or to point to items or areas on the display screen. A sensitized digitizer pad translates images drawn on it with an electronic stylus or pen into a corresponding image on the display screen. Touch-sensitive display screens allow users to point to items or areas on the screen and to activate commands. Optical scanners „read” characters on a printed page and translate them into binary numbers that the CPU can use. Voice-recognition circuitry digitizes spoken words and enters them into the computer.

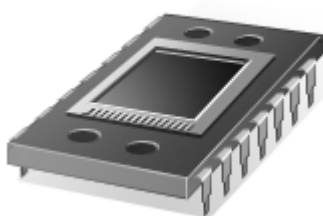
#### *The memory-storage devices.*

Most digital computers store data both internally, in what is called main memory, and externally, on auxiliary storage units. As a computer processes data and instructions, it temporarily stores information internally, usually on silicon random-access memory, or



RAM, chips often called semiconductor memory. Usually mounted on the main circuit board inside the computer or on peripheral cards that plug into the board, each RAM chip may consist of as many as 16 million switches, called flip-flop switches, that respond to changes in electric current. Each switch can hold one bit of data: high voltage applied to a switch causes it to hold a 1; low voltage causes it to hold a 0. This kind of internal memory is also called read/write memory.

Another type of internal memory consists of a series of read-only memory, or ROM, chips. The switches of ROM chips are set when they are manufactured and are unchangeable.



The patterns on these chips correspond to commands and programs that the computer needs in order to boot up, or ready itself for operation, and to carry out basic operations. Because read-only memory is actually a combination of hardware (microchips) and software (programs), it is often referred to as firmware.

Other devices that are sometimes used for main memory are magnetic-core memory and magnetic-bubble memory. Unlike semiconductor memories, these do not lose their contents if the power supply is cut off. Long



used in mainframe computers, magnetic-core memories are being supplanted by the faster and more compact semiconductor memories in mainframes designed for high-speed applications. Magnetic-bubble memory is used more often for auxiliary storage than for main memory.

### *The auxiliary storage units.*

Auxiliary storage units supplement the main memory by holding parts of programs that are too large to fit into the random-access memory at one time. They also offer a more permanent and secure method for storing programs and data.

Four auxiliary storage devices floppy disks, hard disks, magnetic tape, and magnetic drums store data by magnetically rearranging metal particles on disks, tape, or drums. Particles oriented in one direction represent 1s, and particles oriented in another direction represent 0s. Floppy-disk drives (which „write” data on removable magnetic disks) can store from 140,000 to 2.8 million bytes of data on one disk and are used primarily in laptop and personal computers. Hard disk drives contain nonremovable magnetic media and are used with all types of computers. They access data very quickly and can store from 10 million bytes (10 megabytes) of data to a few gigabytes (billion bytes).

Magnetic-tape storage devices are usually used together with hard disk drives on large computer systems that handle high volumes of



constantly changing data. The tape drives, which access data very slowly, regularly back up, or duplicate, the data in the hard disk drives to protect the system against loss of data during power failures or computer malfunctions.

Magnetic-drum memories store data in the form of magnetized spots in adjacent circular tracks on the surface of a rotating metal cylinder. They are relatively slow and are rarely used today.

Optical discs are nonmagnetic auxiliary storage devices that developed from compact-audio-disc technology. Data is encoded on a disc as a series of pits and flat spaces, called lands, the lengths of which correspond to different patterns of 0s and 1s. One removable 4  $\frac{3}{4}$ -inch (12-centimeter) disc contains a spiral track more than 3 miles (4.8 kilometers) long, on which can be stored nearly a billion bytes (gigabyte) of information. All of the text in this encyclopedia, for example, would fill only one fifth of one disc. Read-only optical discs, whose data can be read but not changed, are called CD-ROMs (compact disc-read-only memory).

Recordable CD-ROM drives, called WORM (write-once/read-many) drives, are used by many businesses and universities to periodically back up changing databases and to conveniently distribute massive amounts of information to customers or users.

### *The output devices.*

Output devices let the user see the results of the computer's data processing. The most



common output device is the video display terminal (VDT), or monitor, which uses a cathode-ray tube (CRT) to display characters and graphics on a television-like screen.

#### *The modems (modulator-demodulators)*

Modems are input-output devices that allow computers to transfer data between each other. A modem on one computer translates digital pulses into analog signals (sound) and then transmits the signals through a telephone line or a communication network to another computer. A modem on the computer at the other end of the line reverses the process.

#### *The printers.*

Printers generate hard copy, a printed version of information stored in one of the computer's memory systems. The three principal types of printers are daisy-wheel, dot-matrix, and laser. Other types of printers include ink-jet printers and thermal printers.

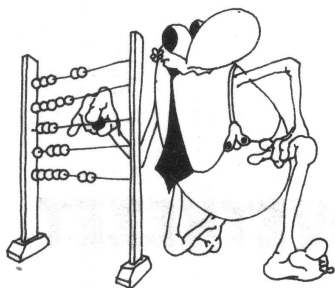
#### *Software.*

#### *The operating system.*

A computer's operating system is the software that allows all of the dissimilar hardware and software systems to work together. It is often stored in a computer's ROM memory. An operating system consists of programs and routines that coordinate operations and processes, translate the data from different input and output devices, regulate data storage in memory, allocate tasks to different



processors, and provide functions that help programmers write software.



Computers that use disk memory-storage systems are said to have disk operating systems (DOS). MS-DOS is the most popular micro-computer operating system. UNIX, a powerful operating

system for larger computers, allows many users and many different programs to gain access to a computer's processor at the same time.

Visual operating systems called GUIs (graphical user interfaces) were designed to be easy to use, yet to give UNIX-like power and flexibility to home and small-business users.

Future operating systems will enable users to control all aspects of the computer's hardware and software simply by moving and manipulating their corresponding „objects,” or graphical icons displayed on the screen.

### *The built-in programs.*

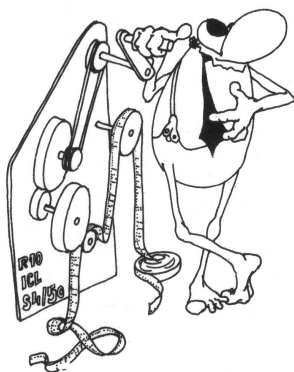
Sometimes programs other than the operating system are built into the hardware, as is the case in dedicated computers or ROM chips. Most often, however, programs exist independently of the computer. When such software is loaded into a general-purpose computer, it automatically programs the computer to perform a specific task such as



word processing, managing accounts and inventories, or displaying an arcade game.

### *The programming.*

Software is written by professionals known as computer programmers. Most programmers in large corporations



work in teams, with each person focusing on a specific aspect of the total project. The eight programs that run each craft in the Space Shuttle program, for example, consist of a total of about half a million separate instructions

and were written by hundreds of programmers. For this reason, scientific and industrial software sometimes costs much more than do the computers on which the programs run.

Generally, programmers create software by using the following step-by-step development process:

- (1) Define the scope of the program by outlining exactly what the program will do.
- (2) Plan the sequence of computer operations, usually by developing a flowchart (a diagram showing the order of computer actions and data flow).
- (3) Write the code the program instructions encoded in a particular programming language.



- (4) Test the program.
- (5) Debug the program (eliminate problems in program logic and correct incorrect usage of the programming language).
- (6) Submit the program for beta testing, in which users test the program extensively under real-life conditions to see whether it performs correctly.

Often the most difficult step in program development is the debugging stage. Problems in program design and logic are often difficult to spot in large programs, which consist of hundreds of smaller units called subroutines or subprograms. Also, though a program might work, it is considered to have bugs if it is slower or less efficient than it should be. (The term bug was coined in the early 1940s, when programmers looking for the cause of a mysterious malfunction in the huge Mark I computer discovered a moth in a vital electrical switch. Thereafter the programmers referred to their activity as debugging.)

*The logic bombs, viruses, and worms.*

In an effort to sabotage other people's computers, some computer users (sometimes called hackers) create software that can manipulate or destroy another computer's programs or data. One such program, called a logic bomb, consists of a set of instructions entered into a computer's software. When activated, it takes control of the computer's programs. A virus attaches itself to a program, often in the computer's operating system, and



then copies itself onto other programs with which it comes in contact. Viruses can spread from one computer to another by way of exchanged disks or programs sent through telephone lines. Worms are self-contained programs that enter a computer and generate their own commands. Logic bombs, viruses, and worms, if undetected, may be powerful enough to cause a whole computer system to crash.

### *The programming languages.*

On the first electronic computers, programmers had to reset switches and rewire computer panels in order to make changes in programs. Although programmers still must „set” (to 1) or „clear” (to 0) millions of switches in the micro-chips, they now use programming languages to tell the computer to make these changes.

There are two general types of languages low-level and high-level. Low-level languages are similar to a computer’s internal binary language, or machine language. They are difficult for humans to use and cannot be used interchangeably on different types of computers, but they produce the fastest programs. High-level languages are less efficient but are easier to use because they resemble spoken languages.

A computer „understands” only one language patterns of 0s and 1s. For example, the command to move the number 255 into a CPU register, or memory location, might look like this: 00111110 11111111. A program might consist of thousands of such operations. To



simplify the procedure of programming computers, a low-level language called assembly language was developed by assigning a mnemonic code to each machine-language instruction to make it easier to remember and write. The above binary code might be written in assembly language as: MVI A,0FFH. To the programmer this means „MoVe Immediately to register A the value FF (hexadecimal for 255).” A program can include thousands of these mnemonics, which are then assembled, or translated, into binary machine code. High-level languages use easily remembered English-language-like commands (such as PRINT, OPEN, GOTO, INCLUDE, and so on) that represent frequently used groups of machine-language instructions. Entered from the keyboard or from a program, these commands are intercepted by another separate program an interpreter or compiler that translates the commands into the binary code the computer uses. The extra step, however, causes programs to run more slowly than do programs in low-level languages.

The first commercial high-level language was called FLOW-MATIC and was devised in the early 1950s by Grace Hopper, a U.S. Navy computer programmer. In 1954, as computers were becoming an increasingly important scientific tool, IBM began developing a language that would simplify the programming of complicated mathematical formulas. Completed in 1957, FORTRAN (Formula Translator), became the first comprehensive high-level



programming language. Its importance was immediate and long-lasting, and it is still widely used today in engineering and scientific applications.

FORTRAN manipulated numbers and equations efficiently, but it was not suited for business-related tasks, such as creating, moving, and processing data files. COBOL (Common Business-Oriented Language) was developed to address those needs. Based on FORTRAN, but with its emphasis shifted to data organization and file-handling, COBOL became the most important programming language for commercial and business-related applications and is widely used today.

A simplified version of FORTRAN, called BASIC (Beginner's All-purpose Symbolic Instruction Code), was developed in the 1960s by two professors at Dartmouth College. Considered too slow and inefficient for professional use, BASIC was nevertheless simple to learn and easy to use, and it became an important academic tool for teaching programming fundamentals to non-professional computer users. The explosion of microcomputer use in the late-1970s and 1980s transformed BASIC into a universal programming language. Because almost all microcomputers are sold with some version of BASIC included, millions of people now use the language, and tens of thousands of BASIC programs are now in common use.

Lots of computer programming languages (or language variants) exist today. PASCAL is a



highly structured language that teaches good programming techniques and therefore is often taught in universities. Another educational language, LOGO, was developed to teach children mathematical and logical concepts. LISP (list processing), developed to manipulate symbolic lists of recursive data, is used in most artificial-intelligence programs. C, a fast and efficient language used for operating systems and in many professional and commercial-quality programs, has recently evolved into the computer world's most powerful programming tool C++. This object-oriented programming (OOP) language lets programs be constructed out of self-contained modules of code and data, called classes, that can be easily modified and reused in other products.

### *The software publishing.*

Software is the term used to describe computer programs. Since 1979 when VisiCalc, one of the most successful early programs, was introduced, software publishing has grown even faster than the computer industry itself. In the late 1980s there were more than 14,000 companies producing about 27,000 products.

The range of software programs has expanded dramatically to include accounting, book-keeping, foreign languages, office management, school subjects, games, engineering and drafting, graphics design, inventory records, legal services, medical records, real estate management, travel reservation systems, library acquisitions, word processing, and much more.



Individual computer owners often use programs for desktop publishing. These programs make it possible to combine word processing with illustrations on a single page.

Marketing for software is very similar to that for books. The software itself is on floppy discs of various sizes, and these are packaged along with instructional material and encased in clear plastic wrapping or in boxes. The packages are displayed on shelves, much the same way books are. Some of the larger producers sell their software directly to companies.

### *History of the computer.*

The ideas and inventions of many mathematicians, scientists, and engineers paved the way for the development of the modern computer. In a sense, the computer actually has three birth dates one as a mechanical computing device, in about 500 BC, another as a concept (1833), and the third as the modern electronic digital computer (1946).

### *Calculating Devices*

The first mechanical calculator, a system of strings and moving beads called the abacus, was devised in Babylonia around 500 BC. The abacus provided the fastest method of calculating until 1642, when the French scientist Blaise Pascal invented a calculator made of wheels and cogs. When a units wheel moved one revolution (past ten notches), it moved the tens wheel one notch; when the tens wheel moved one revolution, it moved the



hundreds wheel one notch; and so on. Improvements on Pascal's mechanical calculator were made by such scientists and inventors as Gottfried Wilhelm Leibniz, W.T. Odhner, Dorr E. Felt, Frank S. Baldwin, and Jay R. Monroe.

*Beyond the Adding Machine.*

The concept of the modern computer was first outlined in 1833 by the British mathematician Charles Babbage. His design of an „analytical engine” contained all of the necessary elements of a modern computer: input devices, a store (memory), a mill (computing unit), a control unit, and output devices. The design called for more than 50,000 moving parts in a steam-driven machine as large as a locomotive. Most of the actions of the analytical engine were to be executed through the use of perforated cards an adaptation of a method that was already being used to control automatic silk-weaving machines called Jacquard looms. Although Babbage worked on the analytical engine for nearly 40 years, he never actually constructed a working machine.

Herman Hollerith, an American inventor, spent the 1880s developing a calculating machine that counted, collated, and sorted information stored on punched cards. When cards were placed in his machine, they pressed on a series of metal pins that corresponded to the network of potential perforations. When a pin found a hole (punched to represent age, occupation, and so on), it completed an electrical circuit and



advanced the count for that category. First used to help sort statistical information for the 1890 United States census, Hollerith's „tabulator” quickly demonstrated the efficiency of mechanical data manipulation. The previous census took seven and a half years to tabulate by hand, but, using the tabulator, the simple count for the 1890 census took only six weeks, and a full-scale analysis of all the data took only two and a half years.

In 1896 Hollerith founded the Tabulating Machine Company to produce similar machines. In 1924, after a number of mergers, the company changed its name to International Business Machines Corporation (IBM). IBM made punch-card office machinery the dominant business information system until the late 1960s, when a new generation of computers rendered the punch card machines obsolete.

In the late 1920s and 1930s, several new types of calculators were constructed. Vannevar Bush, an American engineer, developed the differential analyzer, the first calculator capable of solving differential equations. His machine calculated with decimal numbers and therefore required hundreds of gears and shafts to represent the various movements and relationships of the ten digits.

In 1939 the American physicists John V. Atanasoff and Clifford Berry produced the prototype of a computer based on the binary numbering system. Atanasoff reasoned that binary numbers were better suited to



computing than were decimal numbers because the two digits 1 and 0 could easily be represented by electrical circuits, which were either on or off. Furthermore, George Boole, a British mathematician, had already devised a complete system of binary algebra that might be applied to computer circuits. Developed in 1848, Boolean algebra bridged the gap between mathematics and logic by symbolizing all information as being either true or false.

### *Electronic Digital Computers.*

The modern computer grew out of intense research efforts mounted during World War II. The military needed faster ballistics calculators, and British cryptographers needed machines to help break the German secret codes.

As early as 1941 the German inventor Konrad Zuse produced an operational computer, the Z3, that was used in aircraft and missile design. The German government refused to help him refine the machine, however the computer never achieved its full potential.

A Harvard mathematician named Howard Aiken directed the development of the Harvard-IBM Automatic Sequence Controlled Calculator, later known as the Mark I an electronic computer that used 3,300 electromechanical relays. Completed in 1944, its primary function was to create ballistics tables to make Navy artillery more accurate.

The first fully electronic computer, which used vacuum tubes rather than mechanical relays, was so secret that its existence was not revealed



until decades after it was built. Invented by the English mathematician Alan Turing and in operation by 1943, the Colossus was the computer that British cryptographers used to break secret German military codes. Messages were encoded as symbols on loops of paper tape, and the 2,000-tube computer compared them at nearly 25,000 characters per second with codes that had already been deciphered. Because Colossus was designed for only one task, the distinction as the first modern general-purpose electronic computer properly belongs to ENIAC (Electronic Numerical Integrator and Calculator), designed by two American engineers, John W. Mauchly and J. Presper Eckert, Jr., and ENIAC went into service at the University of Pennsylvania in 1946. Its construction was an enormous feat of engineering the 30-ton machine was 18 feet (5.5 meters) high and 80 feet (24 meters) long, and contained 18 000 vacuum tubes linked by 500 miles (800 kilometers) of wiring. ENIAC performed 100,000 operations per second, and its first operational test included calculations that helped determine the feasibility of the hydrogen bomb.

To change ENIAC's instructions, or program, engineers had to rewire the machine. The next computers were built so that programs could be stored in internal memory and could be easily changed to adapt the computer to different tasks.

These computers followed the theoretical descriptions of the ideal „universal” (general-



purpose) computer first outlined by Turing and later defined by John von Neumann, a Hungarian-born mathematician.

The invention of the transistor in 1948 brought about a revolution in computer development. Hot, unreliable vacuum tubes were replaced by small germanium (later silicon) transistors that generated little heat yet functioned perfectly as switches or amplifiers.

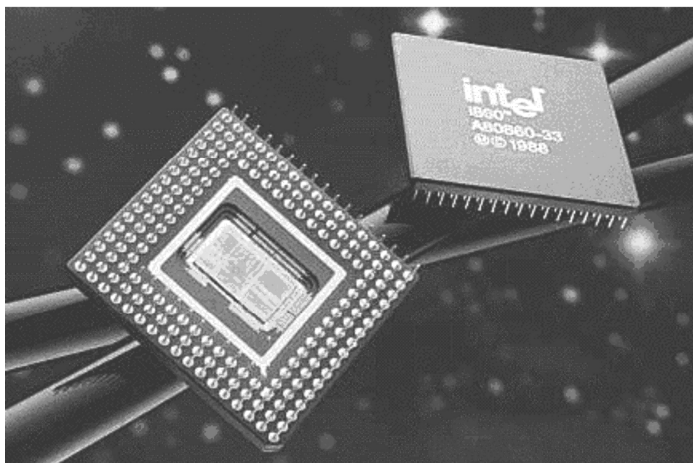
The breakthrough in computer miniaturization came in 1958, when Jack Kilby, an American engineer, designed the first true integrated circuit. His prototype consisted of a germanium wafer that included transistors, resistors, and capacitors the major components of electronic circuitry. Using less expensive silicon chips, engineers succeeded in putting more and more electronic components on each chip.

The development of large-scale integration (LSI) made it possible to cram hundreds of components on a chip; very-large-scale integration (VLSI) increased that number to hundreds of thousands; and engineers project that ultra-large-scale integration (ULSI) techniques will allow as many as 10 million components to be placed on a microchip the size of a fingernail.

Another revolution in microchip technology occurred in 1971 when the American engineer Marcian E. Hoff combined the basic elements of a computer on one tiny silicon chip, which he called a microprocessor.



This microprocessor the Intel 4004 and the hundreds of variations that followed are the dedicated computers that operate thousands of modern products and form the heart of almost every general-purpose electronic computer.



(INTEL© chip by courtesy of INTEL)

### *PCs and other Revolutions.*

By the mid-1970s, microchips and microprocessors had drastically reduced the cost of the thousands of electronic components required in a computer.

The first affordable desktop computer designed specifically for personal use was called the Altair 8800 and was sold by Micro Instrumentation Telemetry Systems in 1974.

In 1977 Tandy Corporation became the first major electronics firm to produce a personal computer. They added a keyboard and CRT to their computer and offered a means of storing



programs on a cassette recorder.

Soon afterward, a small company named Apple Computer, founded by engineer Stephen Wozniak and entrepreneur Steven Jobs, began producing a superior computer.

IBM introduced its Personal Computer, or PC, in 1981. As a result of competition from the makers of clones (computers that worked exactly like an IBM-PC), the price of personal computers fell drastically.

Today's personal computer is 400 times faster than ENIAC, 3,000 times lighter, and several million dollars cheaper. In rapid succession computers have shrunk from tabletop to lap-top and finally to palm size.

With some personal computers, called pen-pads, people can even write directly on an etched-glass, liquid-crystal screen using a small electronic stylus, and words will appear on the screen in clean typescript.

### *Virtual reality.*

As personal computers became faster and more powerful in the late 1980s, software developers discovered that they were able to write programs as large and as sophisticated as those previously run only on mainframes. The massive million-dollar flight simulators on which military and commercial pilots trained were the first real-world simulations to be moved to the personal computer.

Flight simulators are perfect examples of programs that create a virtual reality, or a computer-generated „reality” in which the user



does not merely watch but is able to actually participate. The user supplies input to the system by pushing buttons or moving a yoke or joystick, and the computer uses real-world data to determine the results of those actions.

For example, if the user pulls back on the flight simulator's yoke, the computer translates the action according to built-in rules derived from the performance of a real airplane. The monitor will show exactly what an airplane's viewscreen would show as it begins to climb. If the user continues to climb without increasing the throttle, the „virtual plane” will stall (as would a real plane) and the „pilot” will lose control.

Thus the user's physical actions are immediately and realistically reflected on the computer's display. For all intents and purposes, the user is flying that is, the „plane” obeys the same laws of nature, has the same mechanical capabilities, and responds to the same commands as a real airplane.

Virtual reality programs give users three essential capabilities *immersion, navigation, and manipulation*. People must be immersed in the alternate reality, not merely feel as if they are viewing it on a screen.

To this end, some programs require people to wear headphones, use special controllers or foot pedals, or wear 3-D glasses. The most sophisticated means of immersing users in a virtual reality program is through the use of head-mounted displays, helmets that feed slightly different images to either eye and that actually move the computer image in the



direction that the user moves his or her head. Virtual reality programs also create a world that is completely consistent internally. Thus one can navigate one's way through that world as „realistically” as in the real world.

For example, a street scene will always show the same doors and windows, which, though their perspective may change, is always absolutely consistent internally. The most important aspect of a virtual reality program is its ability to let people manipulate objects in that world. Pressing a button may fire a gun, holding down a key may increase a plane's speed, clicking a mouse may open a door, or pressing arrow keys may rotate an object.

Many amusement parks now have rides and attractions that use virtual reality principles for creating exciting alternate realities for their audiences for example, a simulated ride in a spaceship, complete with near collisions and enemy attacks. Acceleration and deceleration are simulated by pitching and moving seats, all computer-controlled and cleverly coordinated with stereo sound effects and wrap-around video displays.

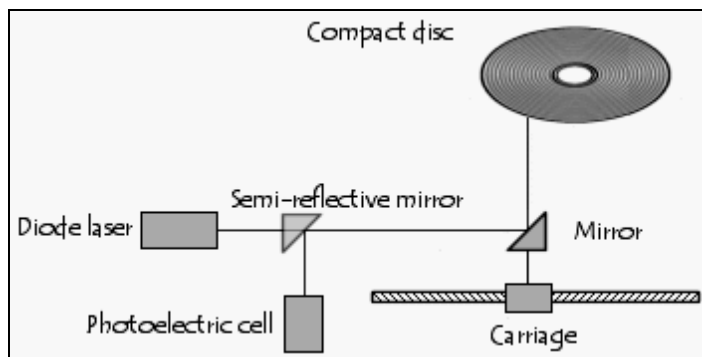
### *Multimedia.*

In the early 1990s, manufacturers began producing inexpensive CD-ROM drives that could access as many as 650 megabytes of data from a single disc.

This development started a multimedia revolution that may continue for decades. The term multimedia encompasses the computer's



ability to merge sounds, video, text, music, animations, charts, maps, etc., into colorful, interactive presentations, a business advertising campaign, or even a space-war arcade game.



CD layout by courtesy of ...)

Audio and video clips require enormous amounts of storage space, and for this reason, until recently, programs could not use any but the most rudimentary animations and sounds. Floppy and hard disks were just too small to accommodate the hundreds of megabytes of required data. The enormous storage potential of CD-ROM changed all that.

Driving simulations, for example, can now show actual footage of the Indianapolis Speedway as the user plays the game. The manufacturer first digitizes video scenes of the speedway and records the real sounds of the racers as they circle the track.

Those images and sounds are then stored on a CD-ROM disc with the driving program itself. When a user runs the simulation and turns his



computerized car, for example, the program senses the „turn” and immediately flashes the corresponding real sounds and scenes on the screen. Likewise, when a driver’s car approaches another car, a video image of a real car is displayed on the screen. By using simultaneous clips from several different media, the user’s senses of sight, sound, and touch are merged into an astonishingly real experience.

Faster computers and the rapid proliferation of multimedia programs will probably forever change the way people get information. The computer’s ability to instantly retrieve a tiny piece of information from the midst of a huge mass of data has always been one of its most important uses. Since video and audio clips can be stored alongside text on a single CD-ROM disc, a whole new way of exploring a subject is possible.

By using hyperlinks a programming method by which related terms, articles, pictures, and sounds are internally hooked together material can be presented to people so that they can peruse it in a typically human manner, by association.

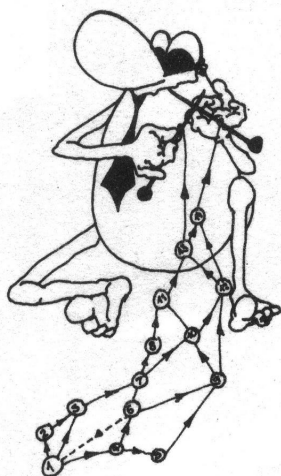
For example, if you are reading about Abraham Lincoln’s Gettysburg Address and you want to read about the battle of Gettysburg, you need only click on the highlighted hyperlink „battle of Gettysburg.” Instantly, the appropriate text, photos, and maps appear on the monitor. „Pennsylvania” is another click away, and so on. Encyclopedias, almanacs, collections of reference books, interactive games using movie



footage, educational programs, and even motion pictures with accompanying screenplay, actor biographies, director's notes, and reviews make multimedia one of the computer world's most exciting and creative fields.

### *The information superhighway.*

A computer network is the interconnection of many individual computers, much as a road is the link between the homes and the buildings of a city. Having many separate computers linked on a network provides many advantages to organizations such as businesses and universities.



People may quickly and easily share files; modify databases; send memos called E-mail, or electronic mail; run programs on remote mainframes; and get access to information in databases that are too massive to fit on a small computer's hard drive. Networks provide an essential tool for the routing,

managing, and storing of huge amounts of rapidly changing data.

The Internet is a network of networks: the international linking of tens of thousands of



businesses, universities, and research organizations with millions of individual users. It is what *United States Vice-President Al Gore* first publicly referred to as the *information superhighway*.

What is now known as the Internet was originally formed in 1970 as a military network called *ARPAnet* (*Advanced Research Projects Agency network*) as part of the Department of Defense.

The network opened to non-military users in the 1970s, when universities and companies doing defense-related research were given access, and flourished in the late 1980s as most universities and many businesses around the world came on-line.

In 1993, when commercial providers were first permitted to sell Internet connections to individuals, usage of the network exploded. Millions of new users came on within months, and a new era of computer communications began.

*Most networks on the Internet make certain files available to other networks.* These common files can be databases, programs, or E-mail from the individuals on the network. With hundreds of thousands of international sites each providing thousands of pieces of data, it's easy to *imagine the mass of raw data available to users.*

The Internet is by no means the only way in which computer users can communicate with others.

Several commercial on-line services provide connections to members who pay a monthly



connect-time fee. CompuServe, America OnLine, Prodigy, Genie, and several others provide a tremendous range of information and services, including on-line conferencing, electronic mail transfer, program downloading, current weather and stock market information, travel and entertainment information, access to encyclopedias and other reference works, and electronic forums for specific users' groups such as PC users, sports fans, musicians, and so on.

### *Artificial intelligence, expert systems.*

The standard definition of artificial intelligence is „the ability of a robot or computer to imitate human actions or skills such as problem solving, decision making, learning, reasoning, and self-improvement.”



Today's computers can duplicate some aspects of intelligence: for example, they can perform goal-directed tasks (such as finding the most efficient solution to a complex problem), and their performance can

improve with experience (such as with chess-playing computers). However, the programmer chooses the goal, establishes the method of operation, supplies the raw data, and sets the



process in motion. Computers are not in themselves intelligent. It is widely believed that human intelligence has three principal components:

- (1) consciousness,
- (2) the ability to classify knowledge and retain it, and
- (3) the ability to make choices based on accumulated memories.

Expert systems, or computers that mimic the decision-making processes of human experts, already exist and competently perform the second and third aspects of intelligence.

INTERNIST is a computer system that diagnoses 550 diseases and disorders with an accuracy that rivals that of human doctors.

PROSPECTOR is an expert system that aids geologists in their search for new mineral deposits. Using information obtained from maps, surveys, and questions that it asks geologists, PROSPECTOR compares the new data to stored information about existing ore deposits and predicts the location of new deposits.

As computers get faster, as engineers devise new methods of parallel processing (in which several processors simultaneously work on one problem), and, as vast memory systems (such as CD-ROM) are perfected, consciousness the final step to intelligence is no longer inconceivable.

English scientist Alan Turing devised the most famous test for assessing computer intelligence.



The „Turing test” is an interrogation session in which a human asks questions of two entities, A and B, which he or she can't see. One entity is a human, and the other is a computer. The interrogator must decide, on the basis of the answers, which one, A or B, is the human and which the computer. If the computer successfully disguises itself as a human and it or the human may lie during the questioning then the computer has proven itself intelligent.

### *The future of computers.*

Research and development in the computer world moves simultaneously along two paths hardware designs and software innovations. Work in each area alternately influences the other.

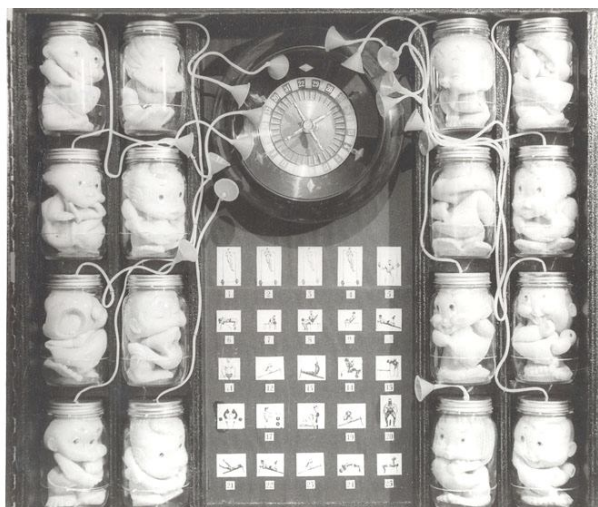
Many hardware systems are reaching natural limitations. RAM chips that can store 64 megabits (millions of 0s or 1s) are currently being tested, but the connecting circuitry is so narrow that its width must be measured in atoms. These circuits are susceptible to temperature changes and to stray radiation in the atmosphere, both of which could cause a program to crash (fail) or lose data.

Newer microprocessors have so many millions of switches etched into them that the heat they generate has become a serious problem. For these and other reasons, many researchers feel that the future of computer hardware might not be in further miniaturization, but in radical new architectures, or computer designs (low power). Almost all of today's computers process



information sequentially, one element at a time. Aspirations are for massively parallel computers consisting of even hundreds of small, simple, but structurally linked microchips break tasks into their smallest units and assign each unit to a separate processor. With many processors simultaneously working on a given task, the problem can be solved much more quickly. One design, called the Thinking Machine, uses several thousand inexpensive microprocessors and can outperform many of today's supercomputers.

Some researchers predict the development of biochips, protein molecules sandwiched



between glass and metal, that would have a vastly greater storage capacity than current technology allows.

Several research labs are even now studying the



feasibility of bio-computers that would contain a mixture of organic and inorganic components. Several hundred thousand computer-controlled robots currently work on industrial assembly lines in Japan and America. They consist of four major elements: sensors (to determine position or environment), effectors (tools to carry out an action), control systems (a digital computer and feedback sensors), and a power system. As computers become more efficient and artificial intelligence programs become more sophisticated, robots will be able to perform more difficult and more human-like tasks.



Special purpose robots are being built by researchers to use in scientific explorations too dangerous for humans to perform, such as descending into active volcanoes or exploring nuclear sites in which radiation leakage has



occurred.

As exciting as all of the hardware developments are, they are nevertheless dependent on well-conceived and well-written software. Software controls the hardware, uses it efficiently, and forms an interface between the computer and the user.

Software is becoming increasingly user-friendly easy to use by non-computer professional users and intelligent able to adapt to a specific user's personal habits. A few word-processing programs now learn their user's writing style and offer suggestions; some game programs learn by experience and become more difficult opponents the more they are played. Future programs promise to adapt themselves to their user's personality and work habits so that the term personal computing will take on an entirely new meaning.

### *Careers in the computer field.*

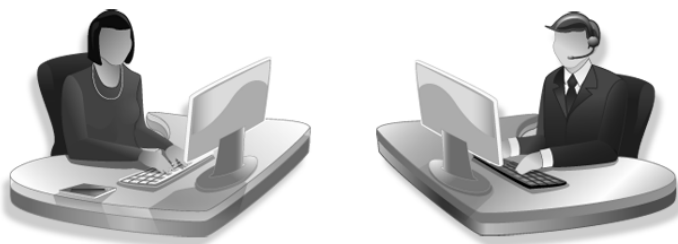
Computer-related jobs are among the most rapidly growing employment segments. Economic studies project that computer equipment will represent about one-fifth of all capital expenditures by businesses in the 1990s. Hundreds of thousands of people will be needed to manufacture, operate, program, and manage new equipment. The most sought-after computer specialists will probably be systems analysts, programmers, and operators.

Systems analysts develop methods for computerizing businesses and scientific centers. They and computer consultants also improve



the efficiency of systems already in use. Computer security specialists will be in great demand to help protect the integrity of the huge information banks being developed by businesses and the government.

Programmers write the software that transforms a machine into a personal tool that not only is useful for increasing productivity but also can enlighten and entertain. Applications programmers write commercial programs to be used by businesses, in science centers, and in the home. Systems programmers write the complex programs that control the inner-workings of the computer. Many specialty areas exist within these two large groups, such as database and telecommunication programmers.



As more small- and medium-sized businesses become computerized, they will require more people to operate their systems. Computer operators will need to handle several types of computers and be familiar with a diversified range of applications, such as database managers, spreadsheets, and word processors. Other important careers in this rapidly



expanding field include computer scientists, who perform research and teach at universities; hardware designers and engineers, who work in areas such as microchip and peripheral equipment design; and information-center or database administrators, who manage the information collections developed by businesses or data banks.

Various support careers also exist. Technical writers, computer-based training specialists, and operations managers do not need extremely technical backgrounds to work in their fields; they need only an expertise in their original fields, a knowledge of computers, and a desire to share their knowledge with others.

~~~~~


Néhány várakozósoros modell.

Először az IBM Systems Journalban, 1975-ben jelent meg, Some elements of queuing theory for system designers címmel, majd 1976-ban a KSH Statisztikai Közleményekben, Néhány gyakoribb várakozósoros modell rendszertervezéshez címmel, majd a SZÁMOK nemzetközi rendszer-szervező képzésének tankönyveként 1976-tól több kiadás-ban.

Az írás aktualitását a mindennapokat átható kapcsolati, divatos szóval élve *kommunikációs, rendszerek működésének tervezése, számítása, ellenőrzése* adja. A közlekedés, a számítógépek, a telefonok, a hálózatok kapacitása, sebessége és ezáltal használhatósága felveti a folyamata-tukat leíró rendszer feltárását és alapos tervezé-sének igényét. A várakozósoros modellek elég jól kidolgozott matematikája lehetőséget ad a gon-dos tervezéshez.

A sorbanállási elmélet, - amelyet telefon hálóza-tok tervezéséhez A. K. Erlang dán matematikus indított el - a számítógépes rendszerek terve-zésének és vizsgálatának hasznos és nélkülöz-hetetlen eszközévé vált a ráfordított adaptációs és fejlesztési munka eredményeként.

Az elmélet segítségével előre meghatározhatók például a számítógép teljesítőképességének jel-lemző adatai, például a várakozási idő egy on-line terminálnál, vagy a tárolóigények üzenet-közvetítő rendszerekben, vagy a prioritás hozzá-rendelések hatása, stb.

Ez a tanulmány ilyen és más hasonló kérdések tárgyalásához szükséges elméleti eszközök bemutatásával - a gyakorlati alkalmazás síkján - kíván foglalkozni.

A sor a kiszolgálásra várakozó igények tárolója például egy kommunikációs-, vagy számítástechnikai rendszerben, míg a sorbanállási elmélet ezek kezelésének a tanulmányozására, tervezésére alkalmas eszköz.

A feldolgozásra várakozó igények egy rendszerben sorokat alkotnak. Az ilyen kiszolgálásra várakozó sorok további sorokat alkothatnak, mint például az input-output igények sora egy feldolgozási programban.

A várakozósor leírása.

Az elméletben az alapelem az *igénylő*, aki, vagy ami igényli és élvezi a kiszolgálást, ezért az ilyen rendszereket - legyen az kommunikációs, számítástechnikai, informatikai, vagy kereskedelmi, pénzügyi stb. természetű - *kiszolgálási rendszereknek* nevezzük és a tervezéshez modellezzük.

Az igénylő által keltett igénylések sorban állnak és várakoznak a kiszolgálásra, ezért *várakozósoros modellnek* nevezzük.

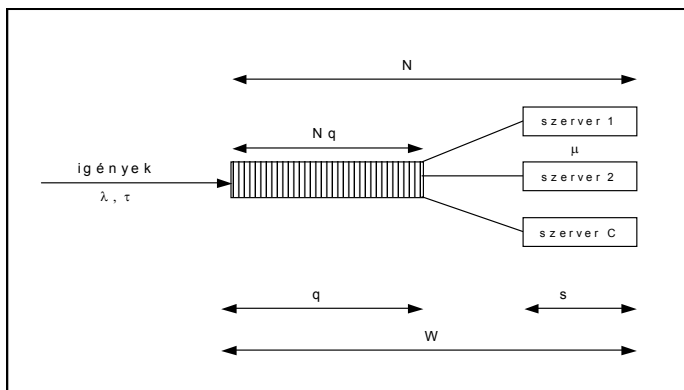
Az informatikai rendszerekben az igénylő lehet például egy átviendő üzenet, egy feldolgozási program, egy választ váró kérdés, egy adatkérés, egy eredményközlés, vagy ezek bármilyen ismétlése, kombinációja, stb.

Az igénylő kiszolgálást kér - üzenet átvitelét egy csatornán, egy program utasításainak végre-

hajtását, egy kifejezés értelmezését, egy adat megkeresését, adat kérését, átvételét más forrásokból, eredmények közlését, stb. - egy erre alkalmas kiszolgáló -, szolgáltató forrásból.

A kiszolgáló forrás egy vagy több kiszolgáló egységből áll, amelyek a kívánt szolgáltatást nyújtják az igénylőknek. Itt csak *univerzális kiszolgáló*król beszélünk, amelyek bármelyike alkalmas bármelyik szolgáltatás elvégzésére.

Ha az összes kiszolgáló egység foglalt, amikor az igénylő belép a rendszerbe, akkor az igény egy várakozó sorba kerül és várakozik addig, ameddig egy kiszolgáló egység a számára fel nem szabadul.



A várakozósoros model jelölései

Az ábrán látható az általános modell a használatos kifejezésekkel, a változókkal és jelentésükkel.

A sorbanállási rendszert, vagy modellt matematikai tanulmányozásához az alábbi módon írhatjuk le.

A forrás.

A forrás, ahonnan az igények származnak, illetve maguk az igények. Az igények forrása lehet véges, vagy végtelen.

Véges forrású rendszerben nem lehet a kiszolgálásra várakozó sor tetszőlegesen hosszú és az igények száma a rendszerben befolyásolja a beérkezési arányt.

Extrém esetben, ha minden igény vár, vagy kiszolgálás alatt van, akkor a beérkezési arány nulla értékű lesz.

Ha a források száma véges, de nagy, akkor végtelen forrást feltételezhetünk az alkalmazott matematikai módszer kezelhetősége érdekében.

Az igények beérkezése.

Feltételezzük, hogy az igények egymásután $t_0 < t_1 < t_2 < t_3 \dots < t_k \dots$ időben érkeznek a rendszerbe. A beérkezések közötti időt a $\tau \approx t_k - t_{k-1}$ (ahol $k \geq 1$) valószínűségi változóval írjuk le, amelyről feltételezzük hogy független és azonos eloszlású valószínűségi változók sorát képezi.

Tetszőleges beérkezések közötti idő esetén a τ jelet használjuk. A beérkezéseket a beérkezések közötti idő eloszlásfüggvényével $A(t) = P(\tau \leq t)$ jellemezzük.

Ha a beérkezési eloszlásfüggvény exponenciális, - a következő igény beérkezésének valószínűsége az idő múlásával exponenciálisan növekszik - tehát minden τ -ra $P(\tau \leq t) = 1 - e^{-\lambda t}$, akkor annak a valószínűsége, hogy bármely t időben n igény érkezik be: $e^{-\lambda t} (\lambda t)^n / n!$, ahol $n=0,1,2,\dots$ és

λ az átlagos beérkezési arány. Ebben az esetben a beérkezések a Poisson eloszlást követik. Gyakran használatos még az Erlang- k és a konstans eloszlás.

A használatos eloszlásfüggvények leírásai megtalálhatók az [1., 2.] irodalomban.

A kiszolgálási idő.

Legyen s_k a k -adik igény kiszolgálásához szükséges idő és feltételezzük, hogy az s_k egy független, azonos eloszlású valószínűségi változó. (Tetszőleges kiszolgálási idő esetén az s jelölést használjuk.)

1. A kiszolgálási idő eloszlásfüggvénye legyen $W_s(t) = P(s \leq t)$.

2. Az átlagos kiszolgálási arányt μ -vel jelöljük.

3. Véletlen, vagyis exponenciális kiszolgálás esetén az eloszlásfüggvény: $W_s(t) = 1 - e^{-\mu t}$, ha $t \geq 0$.

A gyakoribb eloszlásfüggvények még az Erlang- k és a konstans eloszlás. Leírásuk az [1., 2.] irodalomban található.

Egy paraméter.

Egy hasznos statisztikai paraméter az $A(t)$ és $W(t)$ eloszlásfüggvények jellemzésére az u.n.

négyzetes variációs együttható: $C_x^2 = \frac{\text{var}(x)}{E(x^2)}$.

Ha x konstans eloszlású, akkor $C_x^2 = 0$,

ha x exponenciális eloszlású, akkor $C_x^2 = 1$,

ha x Erlang- k eloszlású, akkor $C_x^2 = 1/k$.

Következtetések.

Ha C_τ^2 megközelítőleg nulla, akkor a beérkezéseknek szabályos rendje van, ha közel van az egyhez, akkor a beérkezések véletelenszerűnek jellemezhetők és végül, ha nagyobb egynél, akkor a beérkezések csoportosulására lehet következtetni.

Ha C_s^2 megközelítőleg nulla, akkor a kiszolgálási idő megközelítőleg állandó, nagy értékei pedig a kiszolgálási idő nagy mértékű változásait jelzik

A várakozósoros rendszer kapacitása.

Egyes rendszerekben a sor kapacitását *végtelen*-nek feltételezik, vagyis minden igény addig várakozhat, amíg kiszolgáláshoz nem jut.

Más rendszerekben a sor kapacitása *nulla*, vagyis, ha minden kiszolgáló egység foglalt amikor egy igény beérkezik, akkor az igény visszautasításra kerül.

Megint más rendszerek *véges* kapacitással rendelkeznek, vagyis csak bizonyos mennyiségű igény felvételére képesek.

A kiszolgáló egységek száma.

A legegyszerűbb várakozósoros rendszer *egy* kiszolgáló egységgel rendelkezik, egy időben egy igényt szolgál ki.

A *több* kiszolgáló egységű rendszerek C számú azonos kiszolgáló egységgel rendelkeznek és egy időben C számú igényt elégítenek ki.

Végtelen számú kiszolgálási egységű rendszerben minden igény azonnal kiszolgálásra kerül.

A sorbanállási rend.

A sorbanállási rend, vagy kiszolgálási rend az a szabály, amely szerint kiválasztásra kerül a kiszolgálásban sorrakerülő következő igény.

A leggyakrabban alkalmazott ilyen szabály az „első be - első ki”, a *FIFO* (first-in, first-out), vagy másnéven *FCFS* (first-come, first-served). Ilyen a mindennaposan használt sorszám.

Egy másik gyakori rend az „utolsó be - első ki”, a *LIFO* (last-in, first-out), vagy más néven *LCFS* (last-come, first-served). Ilyen modell például a közismert veremautomata.

Megemlíthető még a „véletlen kiválasztás”, az *RSS* (Random Selection for Service, más néven *SIRO* (Service In Random Order) és a „fontossági sorrend”, a *PRI* (priority service).

A rövidítések kombinálásával további modellek is létrehozhatók, amelyeknek a mi vizsgálataink szempontjából kevésbé van jelentősége.

Emlékeztetek még a fogyasztói társadalom szélsőértékeként emlegetett „közgazdasági modellre, a *GIGO*-ra (garbage-in, garbage-out)”. Ennek tárgyalása nem tárgya jelen tanulmányunknak.

A modell leírása.

A továbbiakban az eddig bemutatott alapfogalmak szerinti sorbanállási, vagy várakozó-soros rendszerek gyakorlati használatra is megfelelő modelljét mutatjuk be.

A rendszerek rövidített jelölése.

A sorbanállásos rendszereket az u.n. *Kendall jelöléssel* szokás rövidített formában leírni az

alábbi módon:

$A/B/c/K/m/Z$, ahol

- A a beérkezések közötti idő eloszlása,
- B a kiszolgálási idő eloszlása,
- c a kiszolgáló egységek száma,
- K a rendszer kapacitása (a tárolható igények száma),
- m a források száma,
- Z a sorbanállási rend.

Gyakran csak a rövidebb $A/B/c$ jelölést használjuk, ilyenkor a várakozósornak nincs felső korlátja, a források száma végtelen, a várakozási rend *FIFO*.

Az A és B helyén a következő eloszlásokat használjuk a leggyakrabban:

- D konstans eloszlás
- $E-k$ Erlang- k eloszlás
- Gi általános, független beérkezések közötti idő,
- G általános kiszolgálási idő a függetlenség feltételezésével,
- M exponenciális eloszlás

Például az $M/E4/3/20/\infty/SIRO$ rendszer

- exponenciális eloszlású beérkezések közötti idővel, három kiszolgáló egységgel, azonos Erlang- k kiszolgálási idő eloszlással, 20 igény (ebből 3 kiszolgálás alatt, 17 a várakozósorban) kapacitással, végtelen számú igényforrással és véletlen kiválasztásos (minden várakozó igény u.a. valószínűséggel kerül sorra) - kiszolgálási rendet ír le.

A rendszer forgalma.

A rendszer forgalma a forgalom intenzitásával - (u) - jellemezhető, ami a kiszolgálás várható értékének - $E(s)$ - és a beérkezések közötti idő várható értékének - $E(\tau)$ - az aránya.

Ez az arány a sorbanállásos rendszer egyik legfontosabb paramétere és az alábbi formulával írható le:

$$u = \frac{E(s)}{E(\tau)} = \lambda E(s) = \frac{\lambda}{\mu}$$

A forgalom intenzitása (u) határozza meg a kiszolgálási egységek minimális számát, amely szükséges a rendszer egyensúlyához a beérkező igények számát tekintve.

Ha például $E(\tau) = 10$ sec és $E(s) = 15$ sec, akkor legalább két kiszolgáló szükséges, mivel:

$$\frac{E(s)}{E(\tau)} = \frac{15}{10} = 1,5 \rightarrow 2 .$$

Az u egységét *erlang*nak nevezték el A.K. Erlang után.

A kiszolgálási egység foglaltsága.

Egy másik fontos paraméter a kiszolgálási egységenkénti intenzitás: ρ , ami u/c , ha a forgalom egyenletesen oszlik meg az egységek között.

A kiszolgálási egységek foglaltsága annak a valószínűsége, hogy az adott egység foglalt. A nagy számok törvénye értelmében a ρ annak az

időnek a megközelítő része, amikor minden egység foglalt.

Egy valószínűség,

amikor t időben n igény van a rendszerben:

Ez a valószínűség $p_n(t)$ nemcsak t -től függ, hanem a rendszer iniciális állapotától is, vagyis a kiszolgálás indulásakor már fennálló igények számától.

A legtöbb működő rendszer esetében t növekedésével $p_n(t)$ közelíti és eléri a p_n értéket, ami független t -től és így az iniciális állapottól. Az ilyen rendszert *kiegyensúlyozott rendszernek* nevezzük.

Ebben a tanulmányban csak kiegyensúlyozott rendszerekkel foglalkozunk, mivel az időfüggő és tranziens megoldások általában túl bonyolultak a gyakorlati használatra. De a legfontosabb, hogy a legtöbb esetben - amelyekről itt szót ejtünk - kiegyensúlyozott rendszer kialakítása a célunk. A nem kiegyensúlyozott, vagyis az időfüggő, illetve tranziens rendszerek tervezésével, illetve az ilyen jelenségek felismerésére, elkerülésére irányuló módszerekkel például az Adaptív irányítások (a [6., 9.]-ben) foglalkoznak.

További jellemzők.

A sorbanállási elmélet a rendszer teljesítményének statisztikus mérését teszi lehetővé és így segíti a rendszertechnikust, hogy az igényelt kiszolgálási szinthez szükséges minimális ráfordítású rendszert tervezhessen, alakíthasson ki.

Ezek a statisztikus mértékek és változataik többek között az alábbiak:

- a sorban eltöltött idő várható értéke: Wq ,
- a rendszerben eltöltött idő várható értéke: W ,
- a sorban álló igények várható értéke: Lq ,
- a rendszerben lévő igények várható értéke: L .

Ezek a mértékek nem függetlenek egymástól, ezért valamelyik ismeretében a többi már kiszámítható. A kapcsolatok az alábbi formulákban fejezhetők ki:

1. $u = \frac{E(s)}{E(\tau)} = \lambda E(s) = \frac{\lambda}{\mu}$
2. $\rho = \frac{u}{c}$
3. $W = q + s$
4. $W = E(W) = E(q) + E(s) = W_q + W_s$
5. $N(t) = N_q(t) + N_s(t)$
6. $N = N_q + N_s$
7. $L = E(N) = \lambda W = E(N_q) + E(N_s)$
8. $L_q = E(N_q) = \lambda W_q$

Ha például Wq -t ismerjük, akkor a többi már számolható:

$$Lq = \lambda Wq, \quad W = Wq + Ws, \quad L = \lambda W.$$

További hasznos teljesítési mérték a rendszer válaszadási idejének értéke, illetve statisztikusan az u.n. 90%-hoz tartozó érték, a $\pi_w(90)$. Ez az az idő, amelynél nem többet tölt el a rendszerben a beérkező igények 90%-a. Formálisan

a $p(w \leq \pi_n(90)) = 0,9$ egyenlettel fejezhető ki. A 90%-os időérték a várakozósortra is definiálható ugyanígy, jele $\pi_q(90)$.

Modellek egy kiszolgáló egységgel.

Ebben a fejezetben néhány gyakrabban előforduló modell leírását és formuláit - a levezetések mellőzésével - adjuk meg gyakorlati használatra.

Az M/M/1 modell.

A leggyakrabban alkalmazott modell az M/M/1 típus, az egyszerű forma és a fontosabb változók eloszlásának pontos meghatározhatósága miatt.

A modell exponenciális eloszlású beérkezések közötti- és kiszolgálási idővel rendelkezik.

Jelentősen különbözik ez a modell azoktól, amelyeknél - a legtöbb modell ilyen - a fontosabb változók átlagos, vagy várható értéke, esetleg még a szórása számítható „csak” ki. Egy másik előnye, hogy gyakran célszerű véletlenszerű beérkezésekkel számolni, ugyanakkor a véletlenszerű kiszolgálás feltételezése a gyakorlati szokásokkal találkozók.

A CPU típusú kiszolgálási idő eloszlásoknál a szórás sokkal nagyobb lehet a várható értéknél és a modell esetenként túl optimisztikus következtetésekre vezethet.

Az alábbiakban az M/M/1 modell kiegyen-

súlyozott rendszerekre vonatkozó formuláit adjuk meg a levezetések mellőzésével:

- a rendszer foglaltsága

$$\rho = u = \frac{E(s)}{E(\tau)} = \lambda E(s) = \frac{\lambda}{\mu} \quad \text{mert } c = 1$$

- annak valószínűsége, hogy a rendszerben n darab igény van

$$P_n = P(N = n) = (1 - \rho)\rho^n \quad n = 0, 1, 2, \dots$$

- annak valószínűsége, hogy a rendszerben n -nél több igény van

$$P(N \geq n) = \rho^n \quad n = 1, 2, \dots$$

- a rendszerben lévő igények várható száma

$$L = E(N) = \frac{\rho}{1 - \rho}$$

- a rendszerben lévő igények szórásnégyzete

$$\sigma_N^2 = \frac{\rho}{(1 - \rho^2)}$$

- a rendszerben tartózkodási idő eloszlásfüggvénye (annak a valószínűsége, hogy az igény legfeljebb t időt tölt el a rendszerben)

$$W(t) = P(w \leq t) = 1 - e^{-\mu(1-\rho)t} = 1 - e^{-t/E(w)}$$

- a rendszerben eltöltött idő várható értéke

$$W = E(w) = \frac{E(s)}{1 - \rho} = \frac{1}{\mu(1 - \rho)}$$

- a rendszerben eltöltött idő szórásnégyzete

$$\sigma_w^2 = E(w)^2$$

- a várakozósor hosszának várható értéke

$$L_q = E(N_q) = \frac{\rho^2}{1-\rho} \quad \text{ha } N_q \geq 0$$

$$= \frac{1}{1-\rho} \quad \text{ha } N_q > 0 \quad (\text{nem üres sor})$$

- a várakozósor hosszának szórásnégyzete

$$\sigma_{N_q}^2 = \frac{\rho^2(1+\rho-\rho^2)}{(1-\rho)^2} \quad \text{ha } N_q \geq 0$$

$$= \frac{\rho}{(1-\rho)^2} \quad \text{ha } N_q > 0 \quad (\text{nem üres sor})$$

- a várakozósorban eltöltött idő eloszlásfüggvénye (annak a valószínűsége, hogy az igény legfeljebb t időt tölt el a várakozósorban)

$$W_q(t) = P(q \leq t) = 1 - \rho e^{-\mu(1-\rho)t} = 1 - \rho e^{-t/E(w)}$$

- a várakozósorban eltöltött idő várható értéke

$$W_q = E(q) = \frac{\rho E(s)}{1-\rho} = \frac{\rho}{\mu(1-\rho)} \quad \text{ha } q \geq 0$$

$$= \frac{E(s)}{1-\rho} = \frac{1}{\mu(1-\rho)} \quad \text{ha } q > 0 \quad (\text{nem üres sor})$$

- a várakozósorban eltöltött idő szórásnégyzete

$$\sigma_q^2 = \frac{\rho(2-\rho)(E(s))^2}{(1-\rho)^2} = \frac{\rho(2-\rho)}{\mu^2(1-\rho)^2} \quad \text{ha } N_q \geq 0$$

$$= \left(\frac{E(s)}{1-\rho} \right)^2 = \left(\frac{1}{\mu(1-\rho)} \right)^2 \quad \text{ha } N_q > 0$$

(nem üres sor).

- az r százalékos idő a rendszerben

$$\pi_w(r) = \frac{E(s)}{1-\rho} \log\left(\frac{100}{100-r}\right) = E(w) \log\left(\frac{100}{100-r}\right)$$

- a 90%-os idő: $\pi_w(90) = 2,3E(w)$

- a 95%-os idő: $\pi_w(95) = 3E(w)$

- az r százalékos idő a várakozósorban

$$\pi_q(r) = E(w) \log\left(\frac{100\rho}{100-r}\right) = \frac{E(q)}{\rho} \log\left(\frac{100\rho}{100-r}\right)$$

- a 90%-os idő: $\pi_q(90) = E(w) \log(10\rho)$

- a 95%-os idő: $\pi_q(95) = E(w) \log(20\rho)$

A valószínűségi változók többsége az M/M/1 modellben közismert formájú:

- az igények száma a rendszerben (N) geometrikus eloszlású,

- a rendszer válaszadási ideje (w) exponenciális eloszlású,

- a várakozósorban eltöltött idő (q) vegyes eloszlású

- (diszkrét belépéskor: $P(q=0)=1-\rho$, folytonos később).

Kiegyensúlyozott rendszerben a várakozósorban eltöltött idő eloszlása az alábbi:

$$W_q(t) = 1 - \rho^{-t/E(w)} \quad \text{ha} \quad t \geq 0$$

A modell formuláiból kitűnik néhány változó erősen nem-lineáris függése a rendszer

foglaltságától (ρ). Ilyen a rendszer különböző részeiben eltöltött idő. A non-linearitás azt mutatja, hogy magas rendszerfoglaltság esetén a várakozási idő nagyon megnövekszik és minden határon túl nő $\rho \rightarrow 1$ esetén. Mintegy $\rho=0,8$ értéknél a foglaltság egészen kis növekedése dramatikusan lerontja a rendszer átbocsájtó képességét.

Ennek elkerülésére az igények prioritásának bevezetésére van szükség. Egy alkalmasan megtervezett prioritásos rendszer magas foglaltság esetén is jól működik. Erre a későbbiekben még visszatérünk.

Az M/M/1 modell alkalmas az u.n. *léptékhatás* bemutatására:

Ha az átlagos beérkezési idő aránya λ és az átlagos kiszolgálási idő arány μ megkétszereződnek (változatlan $\rho=\lambda/\mu$ mellett !), az eltöltött idő várható értéke úgy a várakozó-sorban $E(q)$, mint a rendszerben $E(W)$ egyaránt megfelelődik. Az igények számának várható értéke mindkét helyen változatlan marad.

Általában ha az új rendszerben λ helyett $n\lambda$ és μ helyett $n\mu$ áll, akkor a lépték-viszony az alábbi lesz:

$$\frac{E(q)_{új}}{E(q)_{rég}} = \frac{\left(\frac{\rho}{(1-\rho)n\mu} \right)}{\left(\frac{\rho}{(1-\rho)\mu} \right)} = \frac{1}{n}$$

és

$$\frac{E(w)_{ij}}{E(w)_{\text{régi}}} = \frac{\left(\frac{1}{(1-\rho)n\mu} \right)}{\left(\frac{1}{(1-\rho)\mu} \right)} = \frac{1}{n}$$

A léptékhatás egy következtetésre ad lehetőséget. Ha egy nagyszámítógép terhelését egyenlően elosztjuk n darab olyan kisebb számítógép között, amelyek sebessége a nagy gép $1/n$ -szerese, akkor a válaszadási idő nem változik és a felhasználók alkalmasabban elhelyezhető és üzembiztosabb rendszerhez jutnak.

Erről részletesebben a [12., 14.]-ben olvashatnak.

Az $M/M/1/K$ modell.

($K \geq 1$ és az igények száma $n \leq K$)

Ez a modell az $M/M/1$ modelltől annyiban különbözik, hogy a rendszer igényfelvételi kapacitása véges és legfeljebb K lehet ($n \leq K$).

Ilyen rendszereknél az igények visszautasításra kerülnek, ameddig $n=K$ -val, vagyis a rendszer teljes igénytároló kapacitása foglalt.

A modell az alábbi formulákkal jellemezhető a levezetések mellőzésével:

- a rendszer foglaltsága $\rho = (1 - p_k)u$

- a valószínűség, hogy a rendszerben n darab igény van

$$P_n = \frac{(1-\mu)u^n}{1-u^{K+1}} \quad \text{ha } \lambda \neq \mu \quad \text{és} \quad n = 0, 1, 2, \dots, K$$

- a valószínűség, hogy a beérkező igény visszautasításra kerül

$$P_k = P(n = K)$$

- a tényleges beérkezési arány

$$\lambda_a = (1 - p_k)\lambda$$

- a rendszerben lévő igények várható száma

$$L = E(n) = \frac{u(1 - (K + 1)u^K + Ku^{K+1})}{(1 - u)(1 - u^{K+1})} \quad \text{ha } \lambda \neq \mu$$

$$= \frac{K}{2} \quad \text{ha } \lambda = \mu$$

- a rendszerben eltöltött idő várható értéke

$$W = E(w) = \frac{L}{\lambda_a} = \frac{E(n)}{(1 - p_k)\lambda}$$

- a várakozó sor hosszának várható értéke

$$L_q = E(N_q) = L - (1 - p_0)$$

- a várakozó sorban eltöltött idő várható értéke

$$W_q = E(q) = \frac{L_q}{\lambda_a} \quad \text{ha } q \geq 0$$

$$= \frac{W_q}{1 - p_0} \quad \text{ha } q > 0$$

Az M/M/1/K modell bizonyos szempontból jobb, mint az M/M/1 modell, amint az a formulákból is látszik a rendszer stabil marad még akkor is, ha a beérkezési arány λ meghaladja a kiszolgálási arányt μ , mivel a

rendszer telítettsége esetén a beérkező igények visszautasításra kerülnek.

Az M/G/1 modell.

Ennél a modellnél általában nem nyerhetők az N , Nq , W , q eloszlási függvényei úgy, mint az M/M/1 esetben és így csak a várható értékek kiszámítására szorítkozhatunk.

Ha a kiszolgálási idő várható értékének első három momentuma - $E(s)$, $E(s^2)$, $E(s^3)$ - ismert, akkor több valószínűségi változó várható értéke és szórása számítható.

A modell az alábbi formulákkal jellemezhető:

- a várakozósor hosszának várható értéke

$$L_q = E(N_q) = \frac{\lambda^2 E(s^2)}{2(1-\rho)} = \frac{\lambda^2 \sigma_s^2 + \rho^2}{2(1-\rho)}$$

- a várakozósorban eltöltött idő várható értéke

$$W_q = E(q) = \frac{L_q}{\lambda} \quad \text{ha } q \geq 0$$

$$= E(q | q > 0) = \frac{W_q}{\rho} \quad \text{ha } q > 0$$

$$= E(q^2) = \frac{\lambda E(s^3)}{3(1-\rho)} + \frac{1}{2} \left(\frac{\lambda E(s^2)}{1-\rho} \right)^2$$

- a várakozósorban eltöltött idő szórása

$$\sigma_q^2 = E(q^2) - W_q^2$$

- a rendszerben lévő igények számának várható értéke

$$L = E(N) = L_q + \rho$$

- a rendszerben eltöltött idő várható értéke

$$W = E(w) = \frac{L}{\lambda}$$

$$E(w^2) = E(q^2) + \frac{E(s^2)}{1 - \rho}$$

- a rendszerben eltöltött idő szórása

$$\sigma_w^2 = E(w^2) - W^2$$

- a rendszerben lévő igények számának a szórása

$$\sigma_N^2 = \frac{\lambda^3 E(s^3)}{3(1 - \rho)} + \left(\frac{\lambda^2 E(s^2)}{2(1 - \rho)} \right)^2 + \frac{\lambda^2 (3 - 2\rho) E(s^2)}{2(1 - \rho)} + \rho(1 - \rho)$$

Az $E(q^2)$, σ_q^2 , $E(w^2)$, σ_w^2 , σ_N^2 csak akkor számítható, ha a kiszolgálási idő első három momentuma szerinti értékek ismertek.

A kiegyensúlyozott rendszerre vonatkozó L és W értékek egyenletei az irodalomban *Pollaczek-Hincsin* egyenletekként ismertek.

Adott $E(s)$ esetén az L , Lq , W és Wq értékek akkor minimálisak, ha a kiszolgálási idő szórása nulla. (σ_s^2), azaz a kiszolgálási idő *konstans*. Ilyenkor a momentumok kiszámíthatók és ez az $M/D/1$ modellhez vezet.

Az M/D/1 modell.

Az $M/D/1$ modell az $M/G/1$ speciális esetének tekinthető, amikor

- a kiszolgálási idő: $E(s) = konstans$,

- a momentumok: $E(s^2) = E(s)^2$ és $E(s^3) = E(s)^3$,

- a szórás természetesen nulla ($\sigma_s^2 = 0$).

A modell formulái az alábbiakra egyszerűsödnek:

- a várakozó sor hosszának várható értéke

$$L_q = E(N_q) = \frac{\rho^2}{2(1-\rho)}$$

- a várakozó sorban eltöltött idő várható értéke

$$W_q = E(q) = \frac{L_q}{\lambda} = \frac{\lambda E(s)}{2(1-\rho)} \quad \text{ha } q \geq 0$$

$$= E(q \mid q > 0) = \frac{W_q}{\rho} = \frac{E(s)}{2(1-\rho)} \quad \text{ha } q > 0$$

- a várakozó sorban eltöltött idő szórása

$$\sigma_q^2 = \frac{\rho E(s)^2}{3(1-\rho)} + \frac{\rho^2 E(s)^2}{4(1-\rho)^2}$$

- az igények számának várható értéke

$$L = E(N) = L_q + \rho = \frac{\rho^2}{2(1-\rho)} + \rho$$

- a rendszerben eltöltött idő várható értéke

$$W = E(N) = \frac{L}{\lambda} = W_q + E(s) = \frac{\rho E(s)}{2(1-\rho)} + E(s)$$

- a rendszerben eltöltött idő szórása

$$\sigma_w^2 = \sigma_q^2$$

- az igények számának a szórása

$$\sigma_N^2 = \frac{\rho^4}{4(1-\rho)^2} + \frac{\rho^3}{3(1-\rho)} + \frac{\rho^2(3-2\rho)}{2(1-\rho)} + \rho(1-\rho)$$

Az M/Ek/1 modell.

Az M/Ek/1 modell egy fontos modell, mert számos esetben a kiszolgálási idő jól közelíthető az Erlang eloszlással. További előnye az Erlang eloszlás alkalmazásának, hogy a momentumok ismertek és így a q , w , N változók várható értékei és a szórás számíthatók. Ezek ismeretében a 90%-os és 95%-os értékek is becsülhetőkké válnak. Ez a modell végülis az M/G/1 modell speciális eseteként is tekinthető. A kiszolgálási idő eloszlási függvényének a momentumai az alábbi módon számíthatók ki:

$$E(s^2) = \frac{(k+1)}{k} (E(s))^2$$

$$E(s^3) = \frac{(k+1)(k+2)}{k^2} (E(s))^3$$

Ezek behelyettesítésével az M/G/1 modell formuláiba az $E(q^2)$, L_q , σ_q^2 értékek már kiszámolhatóak.

A q , w , N százalékos valószínűségei becslésére gamma-disztribúciós táblázat alkalmazását ajánljuk, mivel az Erlang-k a gamma-disztribúció speciális esete, ahol a k paraméter diszkrét egész értékű. A táblázatok az u..n.

négyzetes variációs együttható: $C_x^2 = \frac{\text{var}(x)}{E(x^2)}$

különböző értékeihez adják meg a π -t.

Ha x Erlang- k eloszlású, akkor $C_x^2 = 1/k$.

Elsőbbségi, prioritásos modellek.

Az elsőbbségi, vagy prioritásos modellek lényege, hogy az igényeket, vagy forrásaikat számozott osztályokba soroljuk 1-től n -ig.

Az alacsonyabb sorszámúnak elsőbbsége van a magasabb számúval szemben, azaz az i osztályba tartozónak elsőbbsége van a j osztálybeli előtt, ha $i < j$, továbbá az 1. osztályba tartozó, ha $i=1$ az összes előtt van.

Az azonos osztályba tartozók kiszolgálása FIFO, beérkezési sorrend szerint történik.

Az i igény a beérkezésekor a már várakozó i igények mögé kerül, ha ilyen nincs, akkor a j igény elé kerül mert $i < j$.

A j kiszolgálásának befolyásolásától függően kétféle taktika lehetséges:

1. *Kivárásos rendszer, non-preemptive, vagy HOL (head of line)*

i csak akkor kap kiszolgálást, amikor j kiszolgálása befejeződött.

2. *Megszakításos rendszer, preemptive*

j kiszolgálása azonnal megszakad és i kiszolgálása megkezdődik.

Ezután a visszatérés a j kiszolgálásához kétféle módon lehetséges:

- megszakítás újraindítással, preemptive-repeat:
a j kiszolgálása teljes egészében újraindul,
- megszakítás folytatással, preemptive-resume:
a j kiszolgálása a megszakítástól folytatódik.

Az M/G/1 prioritásos modell főbb összefüggései.

Általános összefüggések:

- az átlagos beérkezési arány

$$\lambda = \lambda_1 + \lambda_2 + \dots + \lambda_n$$

- a kiszolgálásban eltöltött idő várható értéke

$$E(s) = \frac{\lambda_1}{\lambda} E(s_1) + \frac{\lambda_2}{\lambda} E(s_2) + \dots + \frac{\lambda_n}{\lambda} E(s_n)$$

$$E(s^2) = \frac{\lambda_1}{\lambda} E(s_1^2) + \frac{\lambda_2}{\lambda} E(s_2^2) + \dots + \frac{\lambda_n}{\lambda} E(s_n^2)$$

- a rendszer forgalma

$$u_j = \lambda_1 E(s_1) + \lambda_2 E(s_2) + \dots + \lambda_j E(s_j) \quad \text{ahol}$$

$$j = 1, 2, \dots, n$$

$$u_n = u = \lambda E(s)$$

Nonpreemptive esetben:

- a várakozó sorban eltöltött idő várható értéke

$$E(q_j) = \frac{\lambda E(s^2)}{2(1 - u_{j-1})(1 - u_j)} \quad \text{ahol } j = 1, 2, \dots, n$$

és $u_0 = 0$

$$E[q] = \frac{\lambda_1}{\lambda} E[q_1] + \frac{\lambda_2}{\lambda} E[q_2] + \dots + \frac{\lambda_n}{\lambda} E[q_n]$$

- a rendszerben eltöltött idő várható értéke

$$E[w_j] = E[q_j] + E[s_j] \quad \text{ahol } j = 1, 2, \dots, n$$

$$E[w] = E[q] + E[s]$$

- a várakozó sor hosszának várható értéke

$$L_q = E[N_q] = \lambda E[q]$$

$$L = E[N] = \lambda E[w]$$

Preemptive-resume esetben:

- a várakozó sorban eltöltött idő várható értéke

$$E[q_j] = E[w_j] - E[s_j] \quad \text{ahol } j = 1, 2, \dots, n$$

$$w_q = E[q] = \frac{\lambda_1}{\lambda} E[q_1] + \frac{\lambda_2}{\lambda} E[q_2] + \dots + \frac{\lambda_n}{\lambda} E[q_n]$$

- a rendszerben eltöltött idő várható értéke

$$E[w_j] = \frac{1}{1 - u_{j-1}} \left[E[s_j] + \frac{\sum_{i=1}^j \lambda_i E[s_i^2]}{2(1 - u_j)} \right] \quad \text{ahol}$$

$$j = 1, 2, \dots, n; \quad u_0 = 0$$

$$W = E[w] = E[q] + E[s]$$

- a várakozó sor hosszának várható értéke

$$L_{q_j} = E[N_{q_j}] = \lambda_j E[q_j] + E[s_j] \quad \text{ahol } j = 1, 2, \dots, n$$

$$L_q = E[N_q] = \lambda E[q] = \lambda w_q$$

$$L = E[N] = \lambda E[w] = \lambda w$$

Összefoglalás.

Az itt összefoglalt technika a számítóközpontok üzemének tervezéséhez használatos módszerek egyik készlete. Segítségével tervezési és döntési értékű válasz nyerhető - a számítógépekkel kapcsolatos elvárások alapján - a rendszer konfigurációs adatainak meghatározására.

Ez az összeállítás megjelent a *KSH Rendszer-technikai Közlemények* sorozatában. Az ott ismertetett módon került sor ennek a technikának az alkalmazására a KSH számítóközpontja legújabb gépének konfigurálására, valamint az Üzemeltetési Kézikönyv és -Szabályzat elkészítésére.

A szerző köszönetet mond munkatársainak a tanulmány elkészítéséhez nyújtott segítségükért, értékes munkájukért.

Budapest, 1976.

Függelék.

Az alkalmazott jelölések és fogalmak.

- $A(t)$ -a beérkezések között eltelt idő eloszlásfüggvénye, vagyis $A(t) = P(\tau \leq t)$
- C -az azonos kiszolgálók száma
- D -állandó eloszlás (beérkezés, vagy kiszolgálás)
- E_k -Erlang-k eloszlás (beérkezés, vagy kiszolgálás)

- $E(N_q | N_q > 0)$ -a nem-üres várakozó sor hosszának várható értéke, vagy átlaga
- $E(q | q > 0)$ -a nem-üres várakozó sorban eltöltött idő várható értéke, vagy átlaga
- FCFS* -first-come, first-served, lásd még *FIFO*
- FIFO* -first-in, first-out, beérkezési sorrend szerint
- G* -általános eloszlás (kiszolgálás)
- GI* -általános, független eloszlás (beérkezés, vagy kiszolgálás)
- GIGO* -garbage-in, garbage-out
- K* -a rendszer kapacitása: legfeljebb ennyi igény lehet a várakozó sorokban és a kiszolgálókban együttesen
- L* - $E(N)$, a rendszerben lévő igények számának várható értéke
- L_q - $E(N_q)$, a várakozó sorban lévő igények várható száma, vagy átlaga
- LCFS* -last-come, first-served, lásd még *LIFO*
- LIFO* -last-in, first-out, az utoljára beérkezett következik (verem automata)
- λ -az átlagos beérkezési idő
- M* -exponenciális eloszlás (beérkezés, vagy kiszolgálás)
- μ -az átlagos kiszolgálási idő
- $N(t)$ -valószínűségi változó, a rendszerben lévő igények várható száma t időpontban
- N* -valószínűségi változó, a rendszerben lévő igények számának várható értéke
- $N_q(t)$ -valószínűségi változó, a várakozó sorban lévő igények várható száma t időpontban
- N_q -valószínűségi változó, a várakozó sorban

- lévő igények számának várható értéke
- $N_s(t)$ -valószínűségi változó, a kiszolgálásban lévő igények várható száma t időpontban
- N_s -valószínűségi változó, a kiszolgálásban lévő igények számának várható értéke
- $P_n(t)$ -annak a valószínűsége, hogy t időpontban n igény van a rendszerben
- P_n -annak a valószínűsége, hogy n igény van a rendszerben
- PRI -elsőbbségi kiszolgálási szabály
- q -valószínűségi változó, az igénynek a várakozósortban eltöltött ideje a kiszolgálás megkezdése előtt
- RSS -Random Selection for Service, véletlenszerű kiszolgálás lásd még SIRO
- ρ -a kiszolgáló kihasználtsága: $\rho = \frac{\lambda}{c\mu}$
- s -valószínűségi változó, a kiszolgálási idő leírására
- $SIRO$ -Service In Random Order, minden igény ugyanolyan valószínűséggel kerül kiszolgálásra, lásd még RSS
- τ -valószínűségi változó, a beérkezések között eltelt idő leírására
- u -rendszer forgalom: $u = \frac{E(s)}{E(\tau)} = \lambda E(s) = \frac{\lambda}{\mu}$
- w -valószínűségi változó, a rendszerben eltöltött idő leírására
- $W(t)$ -a w (a rendszerben eltöltött idő) eloszlásfüggvénye: $W(t) = P(w \leq t)$
- W - $E(w)$, az igénynek a rendszerben eltöltött ideje

$Wq(t)$ -a w_q (a várakozó sorban eltöltött idő) eloszlásfüggvénye:

$Wq - E(q)$, a várakozó sorban eltöltött idő várható értéke

$Ws(t)$ -a w_s (a kiszolgálásban eltöltött idő) eloszlásfüggvénye:

$Ws - E(s)$, a kiszolgálásban eltöltött idő várható értéke.

Irodalomjegyzék.

1. G.A. Korn: Matematikai kézikönyv műszakiaknak. Műszaki, 1975.
2. Prékopa A.: Valószínűségelmélet, Műszaki, 1974.
3. P. Denning: The working set model of program behaviour. Comm. ACM., 1968.
4. E.G. Coffman: Analysis of two time-sharing algorithms designed for limited swapping. Journal of ACM., 1968.
5. L. Kleinrock: A continuum of time-sharing algorithms. Proc of AFIPS., 1970.
6. Gyarmati P.: Dinamikus erőforrás elosztás vegyes üzem esetén. SZTAKI Közl., 1975
7. A. Brandwajn: A model of time-sharing virtual memory system using equivalence and decomposition method. Acta info., 1974.
8. T. Beretvas: A simulation model representing OS/VS2 CP. Proc of OS., 1974.
9. P. Gyarmati: On the adaptive control of Operating Systems., 1975.
10. KSH Rendszertechnikai Közlemények /3. Szerk.: Gyarmati Péter., 1974.

11. G.S. Shedler: A queuing model of multiprogrammed computer with a two level storage system. Comm. ACM., 1973.
12. L. Kleinrock: Queuing systems vol.1 theory., 1975.
13. W. Chang: Single-server queuing processes in computer systems. IBM Sytems Journal., 1970.
14. D. Gross and C. M. Harris: Fundamentals of Queuing Theory. -The McMillan Co., 1974.

~~~~~



# Rádiókommunikációs hálózatok.

*Készült az NJSZT Számítógéphálózat modellek Tavaszi Iskola előadás-sorozataihoz. 1977-1980.*

*In this paper we show a somewhat new networking method: the broadcasting type of communication for computer networking. The paper introduce the basic technics together with their design and evaluation method. The second part shows the packet-switching model with the analysis of the collisions, the channel throughput, together with the stability. The last part shows some method to improve and enhance the capacity of such broadcasting packet swithing network.*

*This work was written by the author at IBM Research, Poughkeepsie and offered for the Spring School Series in Networking of NJSZT.*

## *Bevezetés.*

A technikai fejlődés, a kiépítés egyszerűsége és a relatív olcsósága miatt az utóbbi években előtérbe került a rádiókommunikációs rendszerek alkalmazása számítógépes hálózatok kiépítése céljából is.

Az alkalmazás elve lényegében az, hogy nagy sebességű rádiókommunikációs csatornán (vagy csatornákon) valamilyen többszörös elérési séma alapján üzenetközvetítők (felhasználók) osztoznak.

Számítógépek ilyen hálózati kapcsolatban történő alkalmazásához nagy segítség a rádió-kommunikáció már kialakult, tapasztalatokkal rendelkező működése, függetlensége a korláto-



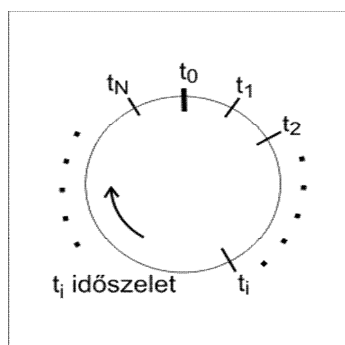
zott kiépítésű vezetékes hálózatoktól, valamint az a tény, hogy a felhasználók számát nem korlátozza technikai körülmény (hol van vezeték és mennyi, stb.).

Korlátlan bevezetésének azonban bizonyos hátrányos tulajdonságok határt szabnak. Ilyen tulajdonság az üzenetek ütközésének (interferencia) esete, amikor két-, vagy több adó-állomás egyszerre küldi üzenetét. Ennek a problémának a feloldási lehetőségeiről később még lesz szó. Tekintsük át először röviden a legfontosabb forgalomszervezési megoldásokat.

### *Alapvető forgalomszervezési típusok.*

Az üzenetközvetítés lebonyolításának számos megoldása lehetséges egy csatornán, amelyet *csatornaforgalomnak* nevezzünk. Példaként bemutatunk néhány ilyen forgalomszervezési megoldást, amelyeknek mindegyike valamilyen többszörös elérési sémát (multi-access) alkot.

#### *- Időosztásos forgalom.*



A más területről jól ismert időszeleteléses megoldás, amikor az  $i$ -edik felhasználó kommunikálhat (az időszeletek allokációja esetleg dinamikusan is változtatható).

Működése a time-sharing módszerekkel

elemezhető és tervezhető. A rendszer felső



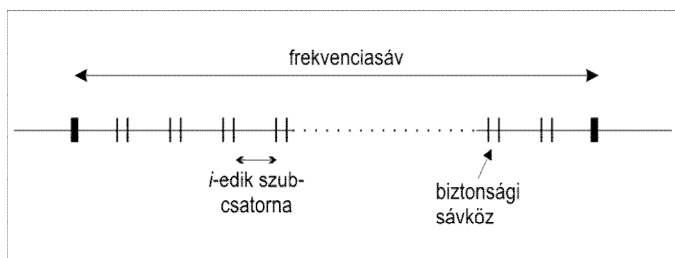
korlátját – a felhasználók maximális száma - a teljes időciklusból adódó várakozási idő  $\sum t_i$  elfogadhatósága határozza meg.

Az időszeletek dinamikus változtatása kiegészítő hálózati felszerelést igényel.

- *Frekvenciosztásos forgalom.*

A kommunikációs csatorna frekvenciasávokban szubcsatornákra van felosztva bizonyos biztonsági sávközök kihagyásával.

Az  $i$ -edik szubcsatorna tartozik az  $i$ -edik felhasználóhoz (a szubcsatorna alokációk dinamikusan is változtathatóak).

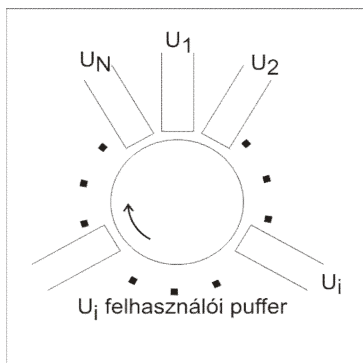


A felhasználók maximális számát a szükséges szubcsatorna sávszélesség határozza meg. Az alokáció dinamikus változásával tovább növelhető a felhasználók száma, azonban a várakozási idő a váltások számával többszöröződik (a dinamikus változtatás kiegészítő kommunikációs felszereléseket igényel).

Elemzése, tervezése a klasszikus módszerekkel lehetséges.



- *Polling rendszerű forgalom.*



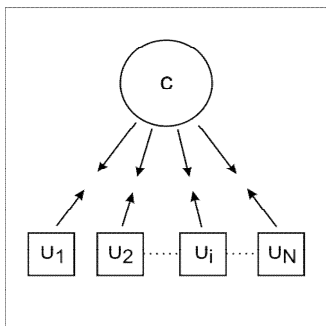
A központi állomás valamilyen, előre meghatározott sorrendben lekérdezi a felhasználókat, hogy van-e átviendő üzenet. Üres felhasználói puffer esetén azonnal továbblép.

Lényegében az időosztásos rendszer to-

vábbfejlesztésének tekinthető, tervezési feltételei avval azonosak.

A lekérdezés miatt adódó overhead árán a kommunikációs csatorna maximális kihasználását teszi lehetővé. A várakozási idő nem a felhasználók számától, hanem azok aktivitásától függ (az egyszerű időosztásos forgalomhoz képest kiegészítő kommunikációs felszereléseket igényel).

- *Véletlenelérésű forgalom.*



A forgalom elve az, hogy amint egy üzenet elkészült az átvitel azonnal elkezdődik. A központi állomás az üzenet vételét egy válaszcsatornán nyugtázza.

Várakozási idő ebben a rendszerben nincs, azonban az egyes termi-



nálok egymástól független üzenetei „ütközhetnek” a közös átviteli csatornában, ami az átvitel meghíúsítását jelenti az összes egyidejű, ütköző - üzenetre. Emiatt az átvitel megismétlésére kell számítani, ami végül is valamilyen statisztikusan kiértékelhető várakozási időhöz vezet.

A rendszer analízise és tervezése a szokásostól eltérő módszerű, amelyre a későbbiekben visszatérünk.

Egyszerű kiegészítő felszereléssel a rendszerbe bizonyos szinkronizálást lehet bevinni, amelynek révén az ütközések száma jelentősen csökkenthető. A szinkronizáció lényege, hogy az üzenetátvitel csak az „üzenethossznak” megfelelő időintervallumok kezdetekor indíthatók.

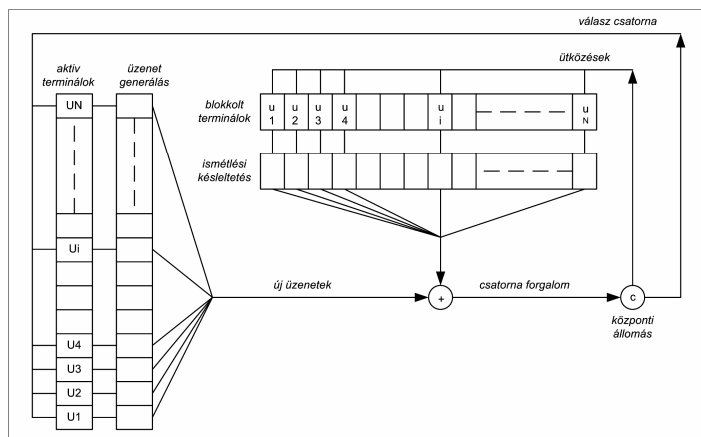
A fenti forgalmi rendszerek mellett számos más forgalmi megoldás is elképzelhető, amelyek azonban általában visszavezethetők az itt ismertetettekre, illetve ezek valamilyen kiegészítései, kombinációi.

Az ismertetett forgalmi rendszerek közül a továbbiakban a véletlen-elérésű forgalommal fogunk foglalkozni, mivel a tervezése, elemzése eltérő a többiekétől és a hagyományos módszerekkel nem követhető. A közismert ALOHA rendszer, illetve annak különböző továbbfejlesztései mind ezt a sémát követik és több éves működésük számos gyakorlati tapasztalattal bizonyították a véletlen-elérésű forgalom előnyeit.



## A véletlen-elérésű forgalom modellje.

### A modell.



A működés vizsgálata és a forgalom tervezhetősége érdekében a későbbi kiterjeszthetőség figyelem előtt tartásával az alábbi módon fogalmazzuk meg a modellt:

- egy meghatározott frekvencián tetszőleges számú terminál üzeneteket küld és fogad egy központi állomással (forgalom kiszolgáló);
- egy másik frekvencián a központi állomás az üzenet vételét nyugtázza;
- az üzenetek fix hosszúságúak és az átvitelük zajmentes csatornán történik (vagyis a véletlen zaj okozta hibák elhanyagolhatóak az ütközésekhez képest);
- ha egy terminálhoz véges időn belül nem érkezik nyugtázás a küldött üzenetre, akkor ütközést kell feltételezni;



- ütközés esetén az összes érintett üzenet átvitelét hibásnak feltételeztük és a terminálok az üzenetük átvitelének megismétlésére kényszerülnek;
- az átvitel megismétlésére egy bizonyos késleltetés után kerülhet sor –a késleltetési idők véletlenszerűen különböznek az egyes terminálokban, az ismételt összeütközések valószínűségének csökkentése érdekében;
- az átvitel sikeres befejezésével a terminálon újabb üzenet készítése kezdődhet.

### *A működés vizsgálata.*

A működés vizsgálatához feltételezzük, hogy az egyes üzenetek (az új és az ismételt együtt) közötti időközök eloszlása független és exponenciális.

A számításhoz az alábbi jelöléseket vezetjük be:

$\tau$ : egy üzenet átviteléhez szükséges idő

$k$ : az aktív terminálok száma

$\lambda$ : üzenetgenerálási ráta terminálonként

$r$ : üzenetgenerálási ráta a rendszerben.

Akkor  $r = k\lambda$ , általánosabban  $r = \sum_{i=1}^k \lambda_i$

Az átviteli csatorna terhelése az  $r\tau$  sorozattal jellemezhető, amelynek értéke maximálisan 1 lehet. Ez az érték azonban nem érhető el, mert mint láttuk üzenetütközések is előfordulnak, ismétlés is van, amely a csatorna forgalmát a hasznos üzenetek kárára növeli. Be kell vezetnünk tehát a csatorna forgalom fogalmát is, ez  $R\tau$ , ahol  $R > r$ .



A csatornaforgalom  $-R\tau$  - tehát a csatornán ténylegesen áthaladó üzenetközvetítés mértéke, míg a csatornaterhelés  $-r\tau$  - ebből csak a hasznos, a központi állomás által átvett és nyugtázott üzenek mértéke. A különbséget a modell ábrája is mutatja.

A feladat tehát meghatározni az eloszlásra és a késleltetésre vonatkozó - a modellben meghatározott - feltételek mellett a csatornán átvihető hasznos üzenetek várható mennyiségét.

Ennek alapján a rendszer további jellemzői is meghatározhatók lesznek.

A modell alapján az alábbi valószínűsések fennállnak:

$$1. \quad p(\text{T intervallumban nem lesz start}) = e^{-RT}$$

$$2. \quad p(\text{ismétlés előfordul}) = 1 - e^{-2R\tau}$$

ahol  $T = 2\tau$  könnyen belátható, mivel egy üzenet egy őt megelőző, vagy egy őt követő üzenettel találkozhat csak.

Ennek alapján az ismétlések átlagos értéke:

$$R = (1 - e^{-2R\tau}) \quad \text{és akkor} \quad R = r + R(1 - e^{-2R\tau}),$$

innen a csatornaterhelés:

$$r\tau = R\tau e^{-2R\tau}$$

Ha ezt az összefüggést - a csatornaterhelést a csatornaforgalom függvényében - megvizsgáljuk, akkor azt találjuk, hogy a függvénynek az

$$r\tau = \frac{1}{2e} \quad \text{helyen szélső értéke van, amit az alábbi}$$

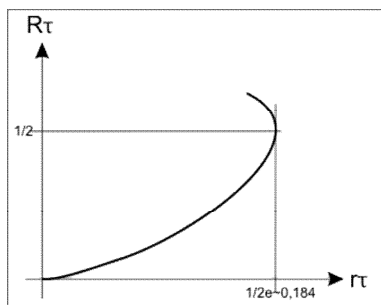
ábrán bemutatunk.

Ez azt jelenti, hogy a csatornán átvihető



hasznos üzenetek száma várhatóan csak a csatorna teljes kapacitásának  $\frac{1}{2e}$ -ed része lehet (kerekben 18 %-a).

A tényleges csatorna forgalom a kapacitás 50 %-át teszi ki ebben az esetben.



Amennyiben tovább növelnénk az új üzenetek számát, akkor az ütközések száma is növekedne, nő a csatornaforgalom, de – amint az ábrán is látszik – csökken a hasznos terhelés és a rendszer instabillá válik (az ismétlések száma minden határon túl nőhet).

Megállapíthatjuk tehát, hogy a rendszert a rendelkezésre álló kommunikációs csatorna kapacitásának  $\frac{1}{2e}$ -ed részére lehet csak tervezni.

A továbbiakban nézzük meg, hogyan használható fel ez az összefüggés az egyidejűleg aktív terminálok maximális számának meghatározásához.

A számításhoz tételezzük fel, hogy a rendel-



kezésre álló kommunikációs csatorna sebessége és egy üzenet mérete alapján az egy csomag átviteli ideje:  $\tau = 34\text{ms}$ .

Tételezzük fel továbbá, hogy az összes terminálon azonos „sebességgel” dolgoznak és percenként egy üzenetet készítenek a felhasználók, tehát  $\lambda = \frac{1}{60}$ .

A feltételezésünk szerint:

$$r\tau = k\lambda\tau = \frac{1}{2e} \quad \text{és innen}$$

$$k_{\max} = (2e\lambda\tau)^{-1}$$

A felvett értékeket behelyettesítve  $k_{\max} = 324$  értéket kapjuk, vagyis egyidőben maximálisan 324(!) terminál lehet aktív, ami nem kis érték a gyakorlatban előforduló terminálhálózatokat és átviteli sebességeket tekintve.

Az összefüggés felhasználható a központi állomás terhelésének a megállapítására is.

Tételezzük fel, hogy központi állomás maximálisan  $24\text{ kB}$  sebességű csatornát képes kiszolgálni (ez megegyezik a példa kommunikációs csatornájának a sebességével), akkor az állomás terhelése:  $24000/2e \approx$  azaz  $4000\text{ bit/sec}$ .

Tehát jelentős kapacitás marad a központi állomás multiplexerében a szükséges adminisztráció és további feladatok elvégzésére még ilyen lassú gép esetén is.



## Stabilitás

A csatornaterheléssel nő az ütközések - és evvel az ismételések - száma és ez tovább növeli a csatornaforgalmat, ami ismét csak növeli az ütközések valószínűségét és így tovább.

Ha egy ilyen - kumulatív folyamat - helyzet kialakul, akkor szinte bizonyosra vehető, hogy a rendszer telítésbe megy át és végül is nem lesz hasznos átvitel, csak ütközés.

Fayolle, Gelenbe, Labetuelle [2] kimutatták, hogy az ilyen állapot elkerülhető, a rendszer stabilizálható az ismételések valamilyen szabályozásával.

A továbbiakban - a bizonyítás bemutatásának mellőzésével - tekintsük át a stabilitást biztosító szabályozás módját. Vegyük fel az alábbi valószínűségeket:

1.  $f(n) = p(\text{ismétlés indult egy blokkolt terminálról})$
2.  $c_i = p(i \text{ új üzenet átvitele kezdődik egyszerre})$
3.  $g_i(n) = p(i \text{ blokkolt terminálról ismétlés van és } n \text{ a blokkolt terminálok száma})$

Kimutatható, hogy

$$g_i(n) = \binom{n}{i} (f(n))^i (1 - f(n))^{n-i}$$

Feltételezzük, hogy az új üzenetek száma független a blokkolt terminálok számától, akkor a csatornaterhelés  $n$ -re az alábbi feltételes valószínűséggel fejezhető ki:

$$\begin{aligned} S_n(f) &= p(\text{sikeres átvitel} | n \text{ blokkolt terminál}) \\ &= c_0 g_1(n) + c_1 g_0(n) \end{aligned}$$



A feladatunk a csatornaterhelés -  $S_n(f)$  - maximalizálása az ismétlések függvényében. Ez a feladat az  $f(n)$  alkalmas meghatározásával érhető el.

Eszerint, a részletes számítások mellőzésével

$$\max S_n(f) = c_0 \left( \frac{n-1}{n-a} \right)^{n-1}, \quad \text{ahol} \quad a = \frac{c_1}{c_0},$$

$$\text{ha} \quad f(n) = f^*(n) = \frac{c_0 - c_1}{nc_0 - c_1} = \frac{1-a}{n-a},$$

vagyis (visszahelyettesítve  $S_n(f)$  kifejezésébe):

$$S_n(f) = c_0 \left( 1 - \frac{1-a}{n-a} \right)^{n-1} \text{ ha } n \rightarrow \infty = c_0 e^{a-1}$$

Ha az új startokra Poisson eloszlást feltételezünk

$$\left\{ c_i = \frac{\lambda^i}{i!} e^{-\lambda}, \quad a = \lambda \right\} \text{ akkor}$$

$$\max S_n(f) \rightarrow e^{-\lambda} e^{\lambda-1} = e^{-1} = \frac{1}{e} \approx 0,368$$

Végül, ha az ismétlések vezérlése  $f^*(n)$ , akkor a teljes csatornaterhelés:

$$S(f) = \sum_{i=0}^{\infty} p(i \text{ blokkolt terminál}) = S_i(f) \geq e^{-1}$$

Tehát az  $f^*(n)$  vezérlés biztosítja mindenkor a hasznos átvitelek – csatornaterhelés – maximális értékét és így a rendszer stabil marad. A rendszer stabilitását az ismétlések indításának a korlátozásával értük el.

A fenti vezérlés mellett kiszámítható még az  $n$



blokkolt terminál okozta ismétlésekből származó kiegészítő csatornaforgalom is. Ez a számítások mellőzésével az alábbi:

$$nf^*(n) = n \frac{1-a}{n-a} < 1-a$$

### *Kiegészítési és bővítési lehetőségek.*

Mint láttuk a véletlen-elérésű forgalmi megoldású kommunikációs hálózat az egyszerű topológia és a nagy megbízhatóságú kommunikációs hordozó által adott előny alacsony csatorna-kihasználással működik, sőt az instabilitás veszélye miatt a rendszer várakozási ideje exponenciálisan növekszik a terheléssel.

A fenti hátrányok elkerülésére, illetve hatásuk csökkentésére --további feltételek bevezetésével, illetve kiegészítő berendezések segítségével-- különböző megoldási lehetőségek vannak. Az alábbiakban röviden áttekintjük ezeket.

### *Részleges szinkronizáció.*

A kommunikációs berendezés kiegészítésével a központi állomás  $\tau$  hosszúságú időszakonként a válaszcsatornára szinkronjelet bocsát ki, amelyet a terminálok vesznek az üzenet átvitel megkezdésének szinkronizálására. Könnyen belátható, hogy ezáltal az egy üzenetre vonatkozó ütközési valószínűség időtartama a korábbi  $2\tau$  helyett  $\tau$ -ra csökken. Evvel a lehetséges csatornaforgalom a kétszeresére,  $\frac{1}{e}$ -re (kb.36%) növekszik. (SLOTTED ALOHA).



### *Csatorna foglalás.*

A megoldás lényege az, hogy a működés bizonyos időszakára a csatornát meghatározott felhasználók kizárólagosan vehetik igénybe. A lefoglalás szabályozása tetszőleges algoritmus alapján, vagy ötletszerűen is lehetséges. Gyakorlati értelme például nagyobb adatmennyiségek összefüggő egységben történő átvitele céljából van. A megoldás tervezése visszavezethető az időosztásos rendszer esetére.

### *Ütközési valószínűség csökkentése*

Különböző kiegészítő kommunikációs berendezésekkel csökkenteni lehet az ütközések számát. Ennek többféle megoldása lehet:

- A vivő frekvencia figyelése, amikor a terminál az üzenetet csak akkor indítja el, ha a csatornában nincs éppen adás – nem „hallható” rajta vivőfrekvencia. Ez a megoldás csak esetleges és jól láthatóan a terminálkezelő „türelmétől” függ. Mindenesetre – a várakozási idő rovására – az ütközések számának csökkenésére lehet számítani.

- Hasonló megoldás a központi állomás által kiadott foglaltsági jelzés figyelése is.

- Az adóteljesítmény, illetve az „adás érthetőség” megkülönböztetése révén lehetővé válik, hogy a központi állomás az ütközött üzenetek közül a nagyobb teljesítményűt – ha még érthető – vegye és ezzel ennek ismétlésére nem lesz szükség. A megoldás hátránya, hogy a kedvezőbb adási körülményű terminálok többnyire nem kívánt előnyös helyzetbe kerülnek a többiek rovására.



- A terminálok adóteljesítményének tudatos, szándékos szabályozásával, változtatásával azonban a terminálokat különböző módon csoportosítani lehet. Ilyen módon például egy prioritásos rendszer kialakítása is lehetséges.

Ezekkel a módszerekkel a hasznos csatorna-forgalom tovább növelhető (50 % fölé is). A megoldás módja, a rendszer tervezése és stabilitásának biztosítása eltér az eddig tárgyaltaktól.

*A kommunikációs csatorna, illetve a központi állomás bővítése.*

A terminálok számának, vagy aktivitásának növelése, a várakozási idő csökkentése elképzelhető a kommunikációs csatornák számának a növelésével, illetve ennek megfelelő mértékben a központi állomás multiplexerében várakozó sor kialakításával. A bővítésnél külön kell vizsgálni a terminál adócsatornák számát és a terminálok közötti kiosztásának módját, a válaszcatornák számát, illetve a multiplexer várakozó sorának hosszát és kezelését.

A megoldás vizsgálata és tervezése, valamint a stabilitás biztosítása az eddigiekhez képest további kiegészítéseket igényel.

*Összefoglalás.*

A rádiókommunikációs hálózatok egyszerű topológiájuk, egyszerű technológiájuk, könnyű tervezhetőségük és néhány más előnyük révén jelentős elterjedésre tarthatnak számon.

A rendszer komoly hátrányának tekinthető az alacsonyfokú csatorna-kihasználás és a várako-



zási idő exponenciális növekedése. Ezek a határnyok azonban kiküszöbölhetőek, illetve jelentőségük csökkenthető – különösen a mikroprocesszorok megjelenésével - a különböző kiegészítések, bővítések célszerű alkalmazásával. Ennek vizsgálata és tervezése a későbbi előadások témája lesz.

### *Függelék.*

Az ALOHA rendszer, Hawai Egyetem tapasztalatai és az elméleti továbbfejlesztések révén a rádiókommunikációs hálózatoknak ma már kiterjedt irodalma van. Ezek közül néhány:

1. Abraham, N.: The throughput of a Pocket Broadcasting Channels. IEEE COM-25, (1977), 117-128.
2. Fayolle, G., Gelenbe, E., Labetouelle, L.: Stability and Control of Pocket Switching Broadcast Channels. Journal of the ACM, vol. 24. (1977), 375-386.
3. Kleinrock, L., Lam, S.: Pocket Switching in a Multiaccess Broadcast Channel. IEEE COM-23, (1975), 410-422 and 891-904.
4. Metzner, J.J.: On improving Utilization in ALOHA Networks. IEEE COM-24, (1976), 447-.
5. Gyarmati, P.: The ALOHA layout as a resource sharing. Hawaii University Papers, MENE-8, 1974, 24-27.
6. Gyarmati, P.: An extension for the pocket switching network. The  $(m,d,L)$ ALOHA modell. Hawaii University Papers, MENE-10, 1977, 77-.

~~~~~


Operációs rendszerek adaptív vezérléssel.

Az eredeti munka a 70-es években az IBM System/360 rendszer nagyobb gépei operációs rendszerének, az OS-360-nak a kiegészítését szolgálta. Eredményeként született az OS-VS/360, majd a VS/370 rendszer, a VS1 és VS2 erőforrás elosztási és vezérlési rendszere. Ugyanez épült be a szocialista országok számítógéprendszere, az ESZR nagyobb gépeinek, R-55/65, operációs rendszerébe is. Ez a tanulmány egy rövidített változat, egy azonos című előadás írott anyaga és alapja egy PhD disszertációnak.

Ma már talán furcsának tűnik, hogy egy számítógép működéséhez ilyen komplex, sok területre kiterjedő matematikai apparátus szükségeltetik. Ennek megértéséhez figyelembe kell vennünk azt a hatalmas növekedést, amelyet az elektronika és a számítástechnika produkált az elmúlt években. Ezek az eredmények nem jöhettek volna létre az előbb említett jelentős eszközök felhasználása nélkül.

A bonyolult számítások eredményei megmutatták az utat az ideálisabb, gyorsabb, nagyobb sáv szélességű, céltudatosabb, összetettebb utasításkészletű gépek kialakítására. Kialakultak az egységes, szabványnak elfogadott eljárások és nem utolsósorban a gépek közötti kapcsolatok alapvető normái. Mindezek tartalmazzák az egyes részletek optimális vezérléséhez szükséges eljárásokat.

Manapság könnyeden beszélünk memória managementről, GUI-ról, cache-ről, routerről, wifi-ről, stb. ezek mindegyike működésében tartalmazza az itt bemutatott elveket, hogy végsősoron megbízható, üzembiztos gépeken végezhessük napi feladatainkat.

A vezérlés elemeinek alaposabb megértéséhez ajánlom ezt a fejezetet, minden informatikus számára, akiknek feladata számítógépek, rendszerek megfelelő működésének biztosítása, és nem utolsósorban *az alkalmazó pénztárcájának kímélése*, a meglévő eszközök minél teljesebb kihasználása és akadálymentes működtetése révén.

Elvek (ADIOS).

A számítógéprendszerek működésének irányítását az operációs rendszerek látják el. A technika fejlődésével a különféle irányítási feladatok egyre bonyolultabbá válnak. Általában azonban nem rendelkezünk elegendő apriori ismeretekkel a működés feltételeiről, ezért a változatlan struktúrájú és többnyire állandó paraméterekkel leírható irányítási analízis és szintézis módszerek igen nehézkesek, sőt gyakran egyáltalán nem alkalmazhatók. Az automatikus irányítási rendszerek felépítésének alapvető eleme az eltérés alapján történő irányítás, a negatív visszacsatolás elve.

Mivel a *hibajel* különféle ellenőrizhetetlen hatásokra vonatkozó információt foglal magába, ezért éppen a hibajel tekinthető *univerzális mértéknek* abban a vonatkozásban, hogy a

rendszer állapota mennyiben tér el a megkívánt üzemmódtól.

Ennek szokásos módú felhasználása azonban több okból kifolyólag is nehézségekbe ütközik. Tárolók, késleltetők és egyéb nem lineáris elemek jelenléte rendszerben korlátozza egy univerzális mérték felhasználását, mivel az egyébként csekély jelentőségű paraméterek és non-linearitások lehetséges hatásait nem ismerve, könnyen megszeghetjük a *stabilis működés* szabta határokat.

Emellett figyelembe kell vennünk azt a tényt is, hogy az irányított rendszer jellemzői még az *idő függvényében is változnak*. Jól érzékelhető a problémák súlya, különösen, ha célunk nem csupán az irányítás, hanem *optimális irányítás* létrehozása.

Mindezek miatt elvileg más felépítésű irányítási rendszert kell kialakítani, olyant, amely képes tanulási feladatokat is megoldani.

Az ilyen *tanuló és adaptív* irányításnak a rendszerről szerezhető információ mérése és feldolgozása alapján úgy kell változtatnia struktúráját és paramétereit, hogy működés közben javítsa saját minőségi jellemzőit, végeredményben bizonyos szempontokból optimális működésre törekedve.

Az adaptív szabályozás esetén tehát az apriori ismeretek hiányát a mért információ célszerű feldolgozásával kompenzáljuk.

Bármely adaptív irányítási rendszer működésének alapját iterációs algoritmusok képezik. A tanuló algoritmusok *olyan folyamatokat képez-*

nek, amelyek valamely átlagolt célfüggvény szélső értékét állapítják meg, egyben ez a célfüggvény nem más, mint a tanulásnak, a megismerésnek a célja.

Az algoritmusok egy osztálya a kereső típusú algoritmusok, amelyek alkalmazására olyan esetekben van szükség, amikor az átlagolt célfüggvény analitikusan nem adott. Számítógépes rendszerek működésének irányítása legtöbbször csak ilyen adaptív kereső típusú algoritmusok segítségével lehetséges.

A cél tehát a működést irányító operációs rendszerek módosítása, kiegészítése ilyen típusú algoritmusokkal a rendszer hatékonyságának javítása érdekében.

Az operációs rendszer adaptív vezérléssel való kiegészítéséhez alapvető feladat az elérendő cél, nyereség kitűzése és az optimalizáló jellemző meghatározása.

A szabályozás feladata kiterjedhet az egyszerűbb szélsőséges működést megakadályozó megoldásoktól a több paraméteres, komplex statisztikai modellek bevezetésére is.

Nézzük meg egy számítógépes rendszer azon területeit, ahol adott esetben adaptív vezérlésre lehet szükség.

A szélsőséges működés (trashing) megelőzése.

Ezeknél a megoldásoknál általában az érintett erőforrás terhelését vizsgáljuk és az alkalmasan megválasztott kritériumnak megfelelően a terhelést korlátozzuk, vagy csökkentjük. A

továbbiakban nézzünk meg néhány ilyen esetet:

Az I/O terhelés (a virtuális tár kivételével).

Ha az input-output csatorna terhelése egy meghatározott küszöbértéket meghalad, akkor a legnagyobb terhelést okozó programot vissza- tesszük a várakozósorba (swap-out).

A szabályozás magyarázata az, hogy túlzott I/O terhelésnél a CPU és a memória kihasználása alacsony, mégis jelentőssé válhat a CPU-ra várás a parallel futó I/O utasítások kezdésének, befejezésének egybeesése miatt. Csökkenő programszámmal ez a várás kevesebb lesz, javul tehát az átfutási idő, ugyanakkor a belépő új programmal esetleg kedvezőbb I/O igény megoszlás alakul ki, esetleg nagyobb CPU igényű program érkezik.

A CPU terhelése.

A CPU túlterhelését jelenti, ha meghatározott időtartam alatt volt olyan program, amelyik nem kapott vezérlést és a CPU nem volt várakozó állapotban. A túlterhelés bekövetkeztekor visszatesszük a várakozósorba azt a programot, amelyik a legnagyobb CPU terhelést okozta (swap-out).

A CPU alacsony kihasználását jelenti, ha a várakozó állapot egy meghatározott küszöbszintet túllép. Ilyenkor a várakozósorból a legnagyobb, ha nem ismert, akkor a sorban elől álló, CPU terhelést okozó programot kiszolgálásra visszük (swap-in).

Pufferek, sorok, segéd-tárak telítettsége.

A különböző várakozó sorok, segéd-tárak a memóriában meghatározott page-eket foglalnak le, amelyek fixen le vannak kötve (nem vesznek részt a lapváltásban). Amennyiben ezek mennyisége egy bizonyos értéket meghalad, akkor a rendszer teljesítménye jelentősen csökkeni kezd, mivel lecsökken a lapváltásra rendelkezésre álló page-ek száma. Ennek megelőzése új programok kiszolgálására való belépésének megakadályozásával oldható meg. Jelzése küszöbérték meghatározásával és a lapváltási ráta figyelésével lehetséges.

A központi tár foglaltsága.

A központi tár terhelése részben a lapváltási rátával, részben a page-ek hivatkozási aktivitásával jellemezhető. Szélsőséges működést jelent, ha a túlzott lapváltás miatt megnő a CPU várakozási idő.

Megakadályozható a programok számának csökkentésével, ha a lapváltás egy előre megadott küszöbszintet elér. Egy másik megelőzési mód, - amennyiben állandóan visszatérő rendszerkomponensről van szó - hogy a lapváltást okozó programot rezidenssé tesszük a központi tárban. Abban az esetben, amikor a page-ek fel vannak osztva a programok között és mindegyik program a saját page-eit váltogatja, akkor bizonyos nagyságú page-lopást megengedve csökkenthetjük a lapváltási rátát (page-lopás: egy program a többi program page-iből a

legrégebben hivatkozottat lefoglalja, LRU algoritmusnak nevezzük).

A lapváltási ráta.

A lapváltás megengedhető legnagyobb sebességét a háttértár működése határozza meg. Ez a háttértár általában mágneslemez, amelynek jellemzője a fejmozgatás. A fejmozgatás szempontjából bizonyos optimalizálási lehetőség áll fenn oly módon, hogy adott mennyiségű lapváltási igényt összegyűjtünk, majd ezeket a minimális fejmozgatási időnek megfelelően sorbaállítva elégitjük ki. Ezzel a megoldással tulajdonképpen a megengedhető lapváltási rátát növeljük. A szabályozási feladat itt annak a meghatározása, hogy mennyi ideig gyűjtsük a lapváltási igényeket, hogy az még ne, vagy csak minimálisan növelje a többi kiszolgáló várakozási idejét.

A CPU és I/O egyensúly.

A programok általában különböző arányban igényelnek CPU, illetve I/O kiszolgálást. Célunk, hogy mindkét kiszolgálót minél jobban terheljük és minimális legyen az egymást kölcsönösen kizáró várakozás. Ennek megoldása a programok relatív, egymáshoz viszonyított prioritásának dinamikus változtatásával lehetséges. A programokat a CPU felhasználás mennyiségének megfelelően sorba állítjuk és a legkisebb felhasználásúnak adjuk a legnagyobb prioritást és így tovább, csökkenő sorrendben. A sorbaállítást és a prioritás hozzárendelést

meghatározott időközönként megismételjük. A megoldás minimalizálja a CPU várakozási időt.

A szabályozás akkor a leghatásosabb, ha a kölcsönös - CPU és I/O - várakozási idő mérhető és az átrendezést ettől függően indítjuk.

Egyéb területek.

A felsoroltakon kívül még más optimalizálandó rendszer komponens is elképzelhető (például a felhasználói és rendszer állományok elhelyezkedése a háttértárolón, az I/O eszközök száma, prioritása, központi tár helyfoglalása, stb). Ezeket azonban többnyire statikusan is meg lehet oldani. Mégis szükséges erről itt említést tenni, mivel az optimalizálás módszere azonos az adaptív szabályozással, csak hosszabb időbeli ciklussal történik, naponta, hetente, stb.

Optimalizáló adaptív szabályozások.

Az adaptív szabályozás magasabb fokának tekinthető, amikor a szélsőséges működés elkerülése mellett optimális működésre is törekszünk. Ilyen esetekben a szabályozandó cél meghatározásán kívül szükséges egy alkalmas költségfükcionál felállítása is. Számítógépes rendszer esetében a költségfükcionálnak olyannak kell lennie, hogy az erőforrások maximális terhelését és ugyanakkor az átfutási idő minimalizálását jelentse.

A szabályozó rendszert ennek megfelelően kell

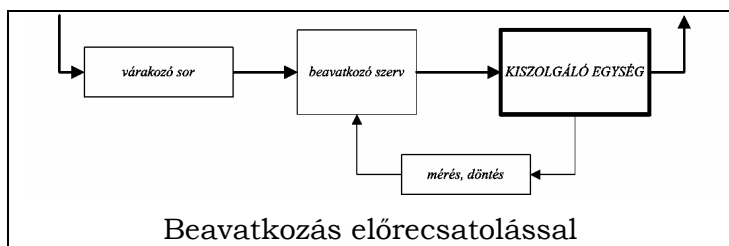
felépíteni:

- a mérendő rendszerparaméter jellemezze a költségfüncionált;
- a döntési funció az optimális működés irányába hasson;
- a beavatkozás paramétere a költségfüncionál jellemző változója legyen;
- az egész szabályozás okozta terhelés minimális legyen olymódon, hogy az általa hozott nyere-ségből csak annyit vegyen el, hogy a megkívánt eredményt meg hozza.

A fentieknek megfelelően többféle szabályozási mód és szabályozandó paraméter alakítható ki.

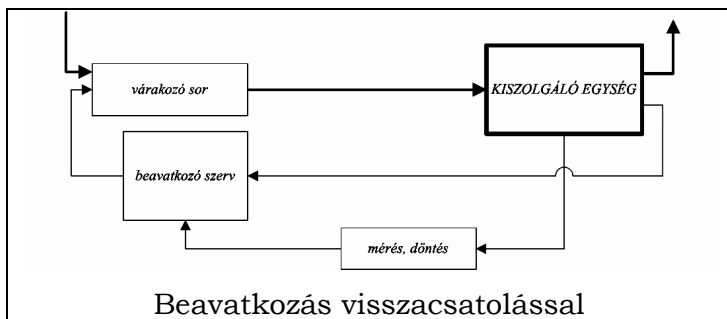
A szabályozás módjára a teljesség igénye nélkül az alábbi megoldásokat javasoljuk, amelyek tulajdonképpen csak a beavatkozás módjában térnek el egymástól.

Beavatkozás előrecsatolással.



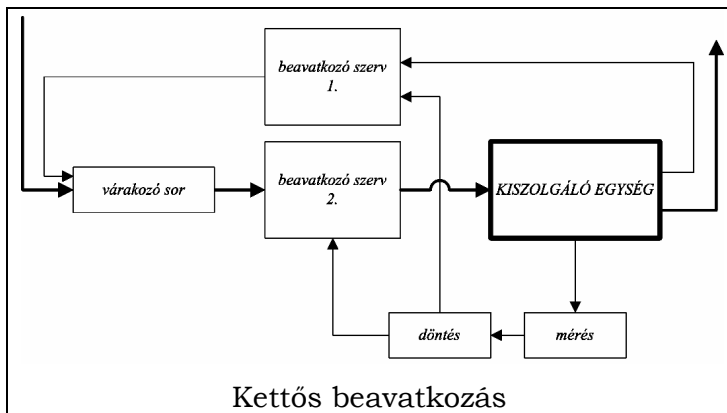
A mérési adatok alapján „megjósoljuk” a következő időszakra vonatkozó legkedvezőbb munkaösszetételt.

Beavatkozás visszacsatolással.



Általában a szélsőséges működés megelőzésére szolgáló szabályozások esetében alkalmazzuk. A mérési eredmények alapján a szélsőséges működést okozó munkákat visszahelyezzük a várakozósorba.

Kétparaméteres beavatkozás.



Ez a módszer – az előrecsatlakozásos és vissza-

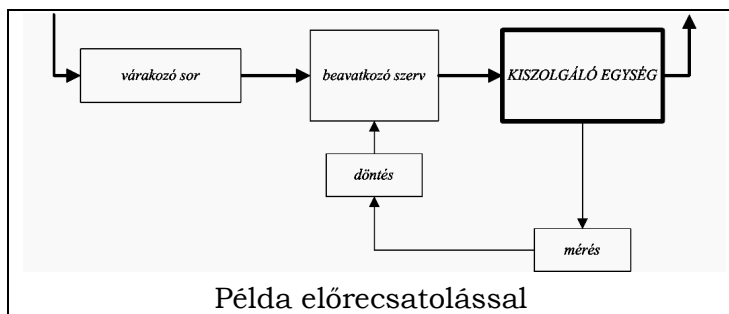
csatolósos beavatkozás - kombinációja. Előnye, hogy a kettős beavatkozási lehetőséggel a munkaösszetétel szélesebb skálán változtatható. Alkalmazására a következő fejezetben látunk példát.

- Szabályozás a kiszolgáló egységen belül

További szabályozási módokra nyílik lehetőség a kiszolgáló egység működésétől függően. Ilyen megoldásoknál a munkaösszetétel változatlan hagyása mellett a kiszolgáló egység erőforrásait valamilyen adaptív szabály szerint rendeljük hozzá az egyes programokhoz. Ezt a módszert alkalmazza például Blevins a [16]-ban időosztásos rendszerű gépen az időszetek hosszának meghatározására. Hasonló megoldás a már említett dinamikus prioritás változtatás is.

Egyparaméteres szabályozási példa

Az adaptív vezérlési módszernek ez az alkalmazása az alábbi modell sémáját követi:



A mérés feladata a költségfüggvény pillanatnyi értékének meghatározásához szüksége paraméterek mérése és a pillanatnyi érték meghatározása.

A döntés feladata a pillanatnyi optimális multiprogramozás (programok száma) fokának megállapítása.

A beavatkozás feladata a döntésnek megfelelően újabb programot a várakozósorból a kiszolgáló egységbe beengedni.

A továbbiakban nézzük meg a modell felépítését és működését.

- *A költségfüggvény meghatározása*

Legyen

- $[t_{i-1}, t_i]$ időintervallum

- n programok száma a kiszolgáló egységben

- K az erőforrások száma

- t_{skj} a kiszolgálási idő a j -edik program számára a k -adik erőforrásból a $[t_{i-1}, t_i]$ -ben.

Az összes kiszolgálás, amit a j -edik program kapott:

$$\sum_{k=1}^K t_{skj}$$

A kiszolgáló egység munkája a $[t_{i-1}, t_i]$ időintervallumban:

$$(1) \sum_{j=1}^n \sum_{k=1}^K t_{skj}$$

ami az összes program számára teljesített

összes kiszolgálás.

A kifejezést normalizálhatjuk a vizsgálati időintervallumra vonatkozólag és ha csak egy program van a kiszolgáló egységben ($n=1$), akkor a kifejezés értéke

$$(2) \quad \frac{\sum_{k=1}^K t_{skj}}{t_i - t_{i-1}} \leq 1$$

Amennyiben a programok száma a kiszolgáló egységben n , akkor a kiszolgálás, amit a felhasználók kapnak, a t_{skj} meghaladja a $t_i - t_{i-1}$ értéket az erőforrások párhuzamos használati lehetősége miatt.

Ílymódon a kiszolgáló egység munkája az időintervallumra normalizálva legfeljebb n nagyságú

$$(3) \quad \frac{\sum_{j=1}^n \sum_{k=1}^K t_{skj}}{t_i - t_{i-1}} \leq n$$

Nyilvánvaló, hogy a gép kihasználása annál jobb, minél nagyobb a kiszolgáló egység munkája. Ezért célként tűzhető ki a kifejezés maximálása és (3)-ból következően n maximálása. Választhatjuk tehát a kialakítandó költségfükcionál paraméterének n értékét.

Alakítsuk át ezt a kifejezést a következő formára:

$$(4) \quad \sum_{k=1}^K \frac{\sum_{j=1}^n t_{skj}}{t_i - t_{i-1}}$$

Jól látható, hogy az első szumma alatti kifejezés éppen a k -adik erőforrás kihasználását adja az i -edik időintervallumban n fokú multi-programozás mellett.

Jelöljük ezt a kifejezést $\alpha_i^k(n)$ -el. Nyilvánvaló, hogy minél nagyobb $\alpha_i^k(n)$ értéke, annál jobb az erőforrás kihasználása.

Ennek alapján meghatározható az erőforrás-kihasználás szempontjának megfelelő költség-funkcionál, amelynek maximálása a kitűzött cél. Bevezetve az egyes erőforrások súlyozására vonatkozó c_k szorzót a költségfünkcionál az alábbi lesz:

$$(5) \quad x_i(n) = \sum_{k=1}^K \alpha_i^k(n) \cdot c_k$$

Ez a költségfünkcionál akkor alkalmas az optimalizálásra, ha ezzel egyben a felhasználói szempontot is optimalizáljuk, vagyis $x_i(n)$ értékének növelésével a felhasználó kiszolgálásban eltöltött ideje csökken, rövidül a válaszadási idő.

Ennek fennállása az alábbi módon bizonyítható. Tekintsük ismét a (3) alatti kifejezésünket:

$$\frac{\sum_{j=1}^n \sum_{k=1}^K t_{skj}}{t_i - t_{i-1}} \leq n$$

Alakítsuk át a kifejezést a következő formára:

$$(6) \quad \sum_{j=1}^n \frac{\sum_{k=1}^K t_{skj}}{t_i - t_{i-1}}$$

Jól látható, hogy az első szumma alatti kifejezés –jelöljük β_i^j -vel-- éppen a j -edik program kiszolgálási „sebessége” n fokú multi-programozás mellett. Nyilvánvaló, hogy ennek minél nagyobb az értéke, annál rövidebb a válaszadási idő (növekszik a kiszolgálás sebessége).

Mivel a (3) és a (6) kifejezések növekedése együttesen következik be, ezért ha $x_i(n)$ maximális, akkor β_i^j is maximális. Tehát $x_i(n)$ alkalmas költségfünkcionál, paraméterként az n felhasználásával, a vezérlési feladat megoldására.

A mérési funkció.

A mérés az $x(x_1, x_2, \dots, x_i, \dots)$ mintasorozatot adja, ahol

$$x_i = \sum_{k=1}^K \frac{\sum_{j=1}^n t_{skj}}{t_i - t_{i-1}} \cdot c_k$$

amint azt az előzőekben megállapítottuk.

A kifejezésből látható, hogy a tulajdonképpeni mérendő mennyiség a t_{skj} , a többi állandó mennyiség. Az x_i értékét ezekből a kijelölt műveletek elvégzésével kaphatjuk meg. Ismerve azonban az operációs rendszereket különböző egyszerűsítésekre van lehetőség. A legtöbb operációs rendszerben, így az ESZR gépeken

alkalmazottaknál is a $\sum_{j=1}^n t_{skj}$ érték minden

egyes erőforrásra nézve már mérésre kerül. Az időintervallum az eltelt idővel egyszerű kivonás után rendelkezésre áll. Ílymódon újabb mérés bevezetése nélkül egyszerű számolással az x_i értékek nyerhetők.

A számolás az alábbi alakra egyszerűsödik:

$$x_i = \frac{\sum_{k=1}^K c_k \cdot t_k}{t_i - t_{i-1}}$$

ahol c_k erőforrás-súlyozási együtthatókat alkalmas táblázatban tarjuk, a t_k értékeket az operációs rendszer meglévő számlálóiból kivonással állítjuk elő: $t_{CPUi} - t_{CPUi-1}$ és ugyanígy képezzük az időintervallum értékét az eltelt idő (elapsed time) számlálóiból.

A mintavételezést alkalmasan beállított óragenerátor, eltelt idő számláló valamelyik kimenete jelzésekor, vagy új program várakozó-sorba való beérkezésekor, vagy egy program futásának befejezésekor (program távozik a kiszolgáló egységből) aktualizáljuk.

Az időintervallum az utóbbi két feltétel miatt rövidebb is lehet, ezért van szükség az eltelt idő alapján történő számolására. Ezen feltételek szerinti aktualizálásra azért van szükség, mert ezekben az esetekben a rendszer állapota oly mértékben megváltozik, hogy a kvázistacionárius feltétel nem tartható. Sőt, mint később látni fogjuk optimalizálásra is szükség lesz. A mintavételezéssel előállított x_i értékek alapján a feladat tehát a folyamat alkalmas becslése. A becsléshez a legegyszerűbben az exponenciális súlyozású eljárást használhatnánk:

$$\overline{x_{\text{exp}}} = \alpha x_i + (1 - \alpha) \overline{x_{\text{exp}-1}}$$

és

$$\sigma_{\text{exp}}^2 = \overline{x_{\text{exp}}^2} - (\overline{x_{\text{exp}}})^2$$

ahol

$$\overline{x_{\text{exp}}^2} = \beta x_i^2 + (1 - \beta) \overline{x_{\text{exp}-1}^2}$$

Ez a számítás azonban bonyolult eljáráshoz vezet, mivel

$$M(\overline{x}) \neq \overline{M(x)} \quad \text{és} \quad M(\sigma^2) \neq \overline{M(\sigma_{\text{exp}}^2)}$$

és normalizálásra szorul.

A várható érték esetében ez a normalizálás még egyszerű egy λ faktorial, ahol

$$\lambda_I = \frac{1}{1 - (1 - \alpha)^i}$$

mert egyes esetekben a λ_i értéke az egyhez olyan közel álló lehet, hogy a normalizálás akár elhagyható, vagy konstans értékke tehető.

A szórás számításánál azonban sokkal bonyolultabb kifejezés áll elő, amely i értékével analitikusan nem kézben tarthatóan változik. Ezért a normalizálást minden i -re el kellene végezni. A részlete levezetés mellőzésével a végső számítás az alábbi módon nézne ki:

$$\sigma_i^2 = \frac{1}{1 - \alpha^2 \lambda^2} \cdot \alpha^2 \left[\frac{1}{1 - (1 - \beta)^2} \overline{x_{\text{exp}}^2} - \lambda_i^2 \left(\overline{x_{\text{exp}}} \right)^2 \right] + (1 - \alpha) \sigma_{i-1}^2$$

Nyilvánvaló, hogy ilyen bonyolult normalizálási eljárás az alapfeltételeknek nem tesz eleget és ezért nem alkalmazható. Inkább az ugyan bonyolultabb számítású és kisebb súlyozású „utolsó N esemény” számítási módszert célszerű követni, ahol normalizálásra nincsen szükség.

Az átlag számítása tehát, (1)

$$\overline{x}_N = \frac{1}{N} \sum_{k=i+1-N}^i x_k ;$$

a szórás pedig , (2)

$$\sigma_N^2 = \frac{1}{N-1} \sum_{k=i+1-N}^i x_k^2 - (\overline{x}_N)^2 = \overline{x_N^2} - (\overline{x}_N)^2$$

alakú lesz.

A külső állítási lehetőséget az N értékének változtatásával oldhatjuk meg.

N értékének meghatározása becsléses, a centrális határeloszlás tétel alkalmazásával lehetséges az aszimptotikusan normális eloszlás feltételezésével (igaz ugyan, hogy a mérési adatok függetlensége nem áll fenn, viszont induló becslésnek, a valós működtetés során lehetséges változtatás biztosításával megfelel).

Ílymódon elég nagy N esetén [13] gondolatmenetét követve

$$P\left(\left|\overline{x_N} - M(x_i)\right| < \lambda \frac{\sigma}{\sqrt{N}}\right) \approx 2\Phi(\lambda) - 1$$

Ha tehát előírt p_o esetén λ -t meghatározzuk olymódon hogy

$$1 - p_o = 2\Phi(\lambda) - 1$$

majd előírt d -hez N értékét oly nagyra

választjuk, hogy $\lambda \frac{\sigma}{\sqrt{N}} \leq d$, akkor azt kapjuk,

$$\text{hogy } P\left(\left|\overline{x_N} - M(x_i)\right| < d\right) \geq 1 - p_o$$

Ehhez N -nek ki kell elégítenie az alábbi feltételt:

$$N \geq \lambda^2 \frac{\sigma^2}{d^2}$$

Amennyiben a relatív hibát be kívánjuk állítani $1-p_o$ valószínűséggel, akkor

$$P\left(\left|\frac{\overline{x_N} - M(x_i)}{M(x_i)}\right| < \varepsilon\right) \geq 1 - p_o$$

és innen az N értéke értelemszerűen így módosul:

$$N \geq \frac{\lambda^2 \sigma_N^2}{\varepsilon^2 (\overline{x_N})^2}$$

Ílymódon a folyamat becslésére figyelembe veendő utolsó N esemény meghatározásához két, a valós működés során is állítható paraméter p_o és ε - áll rendelkezésre.

Tehát az x becslés p_o valószínűséggel lesz ε -nál kisebb relatív hibájú, amennyiben a Gauss-eloszlású közelítés fennáll.

A gyakorlatban p_o és ε értékeit alkalmas táblázatba foglalva a valós működés során validálni, pontosítani lehet.

A döntési funkció.

A mérés eredményeként rendelkezésünkre állnak x_N , σ_N és a hozzájuk tartozó n multiprogramozási értékek. A döntési funkciót aktiváljuk az óragenerátor minden N -edik jelzése után, vagy amikor egy program futása befejeződik (program távozik a kiszolgáló egységből). Ez utóbbi két aktiválási feltételre azért van szükség, mert ilyenkor n értéke változik, illetve változhat.

A szükséges identifikációt az alábbi módon végezzük el:

- tekintsük az utolsó három különböző multiprogramozási értékhez tartozó mérési adatokat és legyen ezek közül a legnagyobb n_o és a hozzátartozó becslések x_{No} és σ_{No} .
- válasszuk ki a tárolt becslések közül az n_o-1 és n_o-2 értékekhez tartozókat és ezeket jelöljük rendre : x_{N1} és σ_{N1} , illetve x_{N2} és σ_{N2}
- a szélsőérték keresés logikájának megfelelően az alábbi szabályokat követjük:

(1) ha $\overline{x_{N2}} < \overline{x_{No}}$ és $\overline{x_{N1}} < \overline{x_{No}}$, akkor n értékét növeljük

(2) ha $\overline{x_{N2}} < \overline{x_{No}}$ és $\overline{x_{N1}} > \overline{x_{No}}$, akkor n értékét csökkentjük

(3) ha $\overline{x_{N2}} > \overline{x_{No}}$, akkor n értékét csökkentjük

(4) ha $\overline{x_{N2}} \approx \overline{x_{No}}$, akkor n értékét csökkentjük

- a szórás figyelembevétele a szélsőérték keresésben a normális eloszlású közelítés alkalmazásával történik (a 3.4.3 fejezetben foglaltaknak megfelelően)
- az előbbi összehasonlítások ekkor így alakulnak:

(1) ha $\overline{x_{N2}} - \overline{x_{No}} + \varepsilon \sqrt{\sigma_{N2}^2 + \sigma_{No}^2} < 0$,
akkor $\overline{x_{N2}} < \overline{x_{No}}$

(2) ha $\overline{x_{No}} - \overline{x_{N2}} + \varepsilon \sqrt{\sigma_{N2}^2 + \sigma_{No}^2} < 0$,

akkor $\overline{x_{N2}} > \overline{x_{No}}$

(3) különben $\overline{x_{N2}} \approx \overline{x_{No}}$

továbbá

(4) ha $\overline{x_{N1}} - \overline{x_{No}} + \varepsilon \sqrt{\sigma_{N1}^2 + \sigma_{No}^2} < 0$,

akkor $\overline{x_{N1}} < \overline{x_{No}}$

(5) ha $\overline{x_{No}} - \overline{x_{N1}} + \varepsilon \sqrt{\sigma_{N1}^2 + \sigma_{No}^2} < 0$,

akkor $\overline{x_{N1}} > \overline{x_{No}}$

- az ε értéke függ a közelítés jóságától és a megkívánt pontosságtól és külső paraméterként használható.

Az identifikáció elvégzése után és eredménye alapján meg kell határozni az n pillanatnyi optimális értékét.

Ez a következő módon történik:

- legyen n_0 az optimális érték, mivel n maximuma a célunk és ez a mérések legnagyobb értéke

- a döntési funkció minden egyes aktiválásánál n_0 értékét módosítjuk oly módon, hogy hozzáadunk, vagy levonunk belőle egyet az identifikáció eredményének megfelelően (azért egyet, mert ezzel teszünk eleget a stabilitás minimum feltételének)

- az így keletkezett új n_0 értéket tekintjük a következő időszakra az optimális multiprogramozási értéknek.

Az n_0 meghatározásának speciális esetei van-

nak, amikor kivételes eljárást kell követni. Ezek az alábbiak:

(1) ha $n_0=3$, akkor levonás műveletet nem végzünk, mivel $n_0 - 2 \geq 1$, mint minimális multi-programozási szám

(2) hasonló okokból a rendszer indításakor $n_0 \geq 3$ értéket kell beállítani.

Az ily módon számított n_0 érték lesz tehát alkalmas arra, hogy a következő időszakra beállítsuk a legkedvezőbb működési állapotot. Egyes esetekben ez az állapot nem az optimális, de mindenképpen annak irányába ható érték lesz az optimum keresés folyamat jellegéből és a stabilitási feltételből következően.

A beavatkozási funkció.

A beavatkozási funkció a várakozósor és a kiszolgáló egység között foglal helyet. Feladata a várakozósorból programot beengedni kiszolgálásra a döntési funkció által meghatározott n_0 optimális multiprogramozási arányszámnak megfelelően.

Az optimalizálási funkció a működésének befejezésekor mindig aktualizálja a beavatkozást, mert n_0 értéke változik.

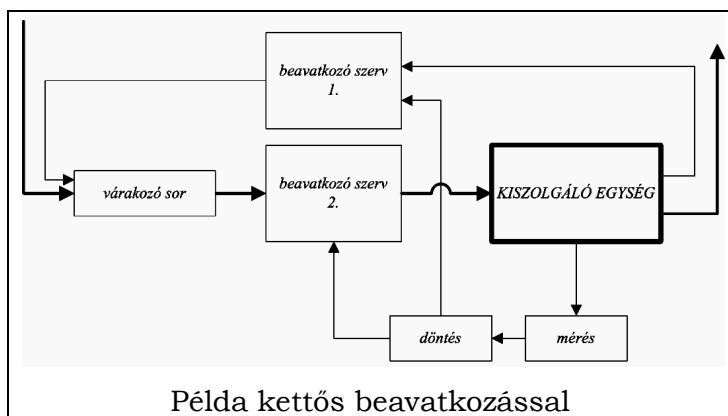
A beavatkozási funkció – SW - akkor és csak akkor enged egy(!) újabb programot a nem üres várakozósorból kiszolgálásra, ha a legutolsóként mért érték kisebb, mint az optimalizálási funkciótól kapott n_0 értéke:

$$SW = \begin{cases} 1, & \text{ha } n < n_0 \text{ és } Q > 0 \\ 0, & \text{különben} \end{cases}$$

Stabilitási okokból mindig csak egy újabb programot engedünk a kiszolgáló egységbe, még akkor is, ha $n_o > n+1$. Ennek az oka nyilvánvaló, mert ha több programot engednénk be és beállna a trashing, akkor már nem tudnánk beavatkozni, hiszen a programok száma a kiszolgáló egységben ennél a modellnél nem csökkenthető (a trashing itt például azt jelenti, hogy jelentősen növekszik a CPU overhead, vagy a lapváltás, stb).

Erőforrások dinamikus elosztása.

Az adaptív vezérlési módszernek ez az alkalmazása az alábbi modell sémáját követi:



A modell, mint látjuk a nyílt hurkú (előrecsatolás) és a zárt hurkú (visszacsatolás) beavatkozást vegyesen alkalmazza (a szabályozás ettől függetlenül mindig zárt hurokban

történik!).

Az egyik paraméterként általában globális - a teljes rendszer működésére - jellemzőt, míg a másik paraméterként a rendszer egy specifikus - a működést legjobban befolyásoló, legérzékenyebb - jellemzőjét használjuk fel.

A jelen példában a [31]-ben ajánlott és több helyen bevezetett [33, 43, 46] dinamikus szabályozási algoritmust és annak továbbfejlesztését mutatjuk be.

A továbbfejlesztés célja egyrészt az, hogy a működési tartományt szélesítsük olyan értelemben, hogy a rendszer szélsőséges terhelési körülmények között is optimálisan működjék, másrészt csökkentsük az apriori ismeretek szükséges mennyiségét. Ezt a célt újabb adaptív szabályozás beiktatásával fogjuk elérni.

A továbbfejlesztés nyeresége különböző beállításokra és feladatösszetételekre a [33, 40, 41, 43] irodalomban és más, [48, 49] belső tájékoztatókban került kimutatásra.

A heurisztikus algoritmus és realizálása

Az algoritmus feladata a programok oly módon történő elrendezése, hogy mindig a legjobban „rászorult” programok jussanak kiszolgáláshoz.

A folyamat a következő:

(1) Egy időintervallum végén a program által kapott - mérés útján ismert - kiszolgálás és az apriori meghatározott - a futás során nem változó - kiszolgálási igény különbözőségéből állapítjuk meg a program pillanatnyi

„rászorultságát”.

(2) A programokat a „rászorultság” nagysága szerint, előjel helyesen, csökkenő sorba rendezzük.

(3) A kiszolgáló egységből eltávolítjuk a legkevésbé „rászorultakat” és helyükre a jobban „rászorultakat” töltjük be, vagyis a sorban elől állók, a legjobban rászorultak, maradnak, illetve kerülnek kiszolgálásra.

Ennek a rendezésnek a paramétere az egyes programok központi tár igénye, tehát annyi program kerül kiszolgálásra, amennyi „befér a központi tárba”.

Mint látjuk az algoritmus két paraméterrel működik. Az egyik a kiszolgálásra jellemző mérték, míg a másik a központi tár igény.

- *A mérési funkció*

Legyen a_j az i -edik időintervallumban a j -edik program által nyert összes kiszolgálás:

$$S_j = c_1 S_j^{CPU} + c_2 S_j^{PD} + c_3 S_j^{FD}$$

ahol a CPU , FD , PD indexek rendre a központi egységet, a virtuális tárat (page device), és az input-output egységet (file device) jelzik, míg c_1 , c_2 , c_3 az egyes erőforrások súlyozó paramétere.

Az egyes kiszolgálásokat mérés alapján az alábbi módon határozzuk meg:

$$(1) \quad S_j^{CPU} = \frac{t_{CPUj}}{t_i - t_{i-1}}$$

ahol t_{CPUj} a j -edik program által felhasznált CPU

idő, a $[t_i, t_{i-1}]$ időintervallumban.

$$(2) \quad S_j^{PD} = \frac{t_{CPUj}}{t_i - t_{i-1}} \cdot p_j$$

ahol p_j a j -edik program számára a (t_i, t_{i-1}) időintervallum végén aktiv page-k száma.

$$(3) \quad S_j^{FD} = \frac{1}{t_i - t_{i-1}} \cdot \sum_{k=1}^K u_{jk}$$

ahol u_{jk} a j -edik program számára a (t_i, t_{i-1}) időintervallumban teljesített I/O utasítások száma, ha K az összes I/O egység.

(4) A c_1, c_2, c_3 súlyozó paraméterek az alábbi módon határozhatók meg:

- tekintsük az egyes kiszolgálók (CPU, PD, FD) által teljesíthető maximális kiszolgálást

$$S_{\ddot{o}}^{CPU} = 1$$

$$S_{\ddot{o}}^{PD} = N$$

ahol N a rendszerben lévő összes page-k száma

$$S_{\ddot{o}}^{FD} = \sum_{k=1}^K u_{k \max}$$

ahol $U_{k \max}$ a k -adik I/O egység által időegység alatt végrehajtható utasítások száma

A fentieknek megfelelően és ahhoz, hogy a maximális kiszolgálás egységenként legfeljebb 1 legyen a súlyozó paraméterek az alábbiak lesznek:

$$c_1 = 1; \quad c_2 = \frac{1}{N}; \quad c_3 = \frac{1}{\sum_{k=1}^K u_{k \max}}$$

Gyakorlati tapasztalat alapján további c_k szorzók még bevezethetők.

A döntési funkció.

Legyen adott $\underline{r}(r_1, r_2, \dots, r_i, \dots, r_m)$ m elemű sor, amelynek egyes elemei az s kiszolgáláshoz hasonló mértékű küszöbszintek olymódon, hogy

$$r_l < r_{l+1} \quad \forall l - re$$

Az $\underline{r}(r_1, r_2, \dots, r_i, \dots, r_m)$ sor meghatározása a rendszerről szerzett ismeretek alapján tapasztalati úton történhet. A felhasználó feladata, hogy programjához hozzárendelje a megfelelő r_i értéket, amelyet a rendszert üzemeltetők által kiadott tájékoztatás, vagy utasítás alapján tesz meg (a tájékoztató is tapasztalati alapokon készül).

Az optimalás a (t_i, t_{i-1}) időintervallum lefutásakor a következő módon történik:

(1) Elvégezzük a $q_j = r_j - s_j$ kivonást minden programra, amely a kiszolgáló egységben van; a programok q_j értéke szerint sorba rakhatók.

(2) Tekintsük az alábbi három osztályt:

$q_j > 0$ a teljesített kiszolgálás alacsonyabb szintű az igényelnél (a kiszolgálás KÉSIK!)

$q_j \approx 0$ a teljesített kiszolgálás megfelel az igénynek

$q_j < 0$ a teljesített kiszolgálás magasabb szintű

az igényeltnél (a kiszolgálás KÉSLELTETHETŐ!).

(3) A kiszolgáló egységben határozzuk meg az összes $q_j < 0$ osztályú programot, ha ilyen nincs, akkor a $q_j \approx 0$ osztályúakat, ha ilyen sincs, akkor ez a rendszer általános túlterheltségét jelenti és külső beavatkozás szükséges.

(4) Állapítsuk meg a kiszolgáló egységben ezen programok által elfoglalt page-k számát

$$N_F = \sum_j p_j$$

ahol p_j a kiszolgáló egységben a (3) szerint meghatározott j -edik program által elfoglalt page-k száma.

(5) A várakozó sorban az elől álló programok közül válasszunk annyit, hogy

$$\sum_k p_k \approx N_F$$

ahol p_k a várakozó sorban éppen sorra kerülő program legutolsó futásakor igényelt page-inek a száma (a \approx jel azt jelenti, hogy a legközelebb álló, de N_F -nél kisebb érték).

(6) Ha a várakozó sorban kevés program van, vagyis

$$N_F > \sum_k p_k \quad \text{az összes } k\text{-ra}$$

akkor a (3) alapján kiválasztott $q_j < 0$ osztályú programokat sorban vegyük vissza, vagyis az N_F értéket csökkentsük egészen addig, amíg

$$N_F' \approx \sum_k p_k$$

(a \approx jel azt jelenti, hogy a legközelebb álló, de nem kisebb N_F érték).

Erre a lépésre azért van szükség, hogy alacsony terhelés esetén se következzen be trashing (a programok fölösleges mozgatása a várakozósor és a kiszolgáló egység között).

A beavatkozási funkció.

Ezek után a programok tényleges mozgatása következik:

- a (3) alapján kiválasztott és a (6) után is megmaradó programokat a kiszolgáló egységből eltávolítjuk és a várakozósor végére helyezzük (swap-out);
- az (5) alapján a várakozósorból kiválasztott programokat a kiszolgáló egységbe bevisszük (swap-in);
- a rendszerbe újonnan érkező programok a várakozósor végére kerülnek.

A programok eltávolítása egyszerűen a lekötött page-k felszabadítását jelenti, míg a programok bevitele a szükséges page-k számának lekötésével történik. Az egyes page-k tartalmának betöltését a lapváltási algoritmus az igény szerinti alapon elvégzi. A rendszerbe érkező új programok egy tapasztalati úton meghatározott állandó page számot kapnak.

A döntési funkció kiegészítése.

A túlterhelési állapot szabályozása.

A számítógép az algoritmus működésével együtt is túlterhelési állapotba kerülhet, amelyet a (3)-

ban már jellemeztünk. Nevezetesen, ha a kiszolgáló egységben olyan programok vannak, amelyek $q_i > 0$ osztályúak, akkor minden program futása késik. Ilyenkor az algoritmus már nem működik ($N_F = 0$) és további beavatkozás szükséges, amellyel a késések vagy késleltetések elosztását szabályozni lehet.

A módszer a következő:

(1) legyen w_i az a mérték, amely megadja, hogy az i -edik időintervallumban az összes kiszolgálási igény hányszorosa a teljesített kiszolgálásnak

$$w_i = \frac{\sum_j r_j + \sum_k r_k}{\sum_j s_j}$$

(2) tekintsük a $\underline{w}(w_1, w_2, \dots, w_n)$ n elemű sort, ahol az egyes elemek küszöbszintek olymódon, hogy

$$w_l < w_{l+1} \quad \forall l\text{-re}$$

a \underline{w} sor meghatározása a rendszerről szerzett ismeretek alapján, tapasztalati úton történik.

(3) az (1) alapján kiszámított w_i érték a sor l -edik értékével helyettesíthető, ha

$$w_{l-1} < w_i \leq w_l$$

(4) legyen \underline{L} egy $m \times n$ elemű mátrix, amelynek elemei $0 \leq l_{jk} \leq 1$ értékűek és $l_{j1} \equiv 1 \quad \forall j\text{-re}$ (az első oszlop minden eleme egyes)

(5) tegyük fel, hogy a rendszer az i -edik idő-

pontban a w_l terhelési állapotban van;

-tartozzék w_l legkisebb terhelési állapothoz a korábban felvett $\underline{r}(r_1, r_2, \dots, r_i, \dots, r_m)$ m elemű vektor, amely a lehetséges kiszolgálási igény küszöbszinteket tartalmazza; az i -edik időpontban akkor úgy tekintjük, mintha $\underline{r}^{(l)}$ kiszolgálási igény lenne oly módon, hogy $\underline{r}^{(l)} = \underline{r}^{(1)} \cdot \underline{l}_l$ ezáltal a terhelésnek megfelelően az \underline{L} mátrix szerint osztottuk el a késleltetéseket; a terhelés változtatásával tehát \underline{r} értékei változnak és a legkisebb terhelés esetén az eredeti értéket veszik fel, mivel a w_l -hez tartozó vektor elemei mind egységnyi értékűek;

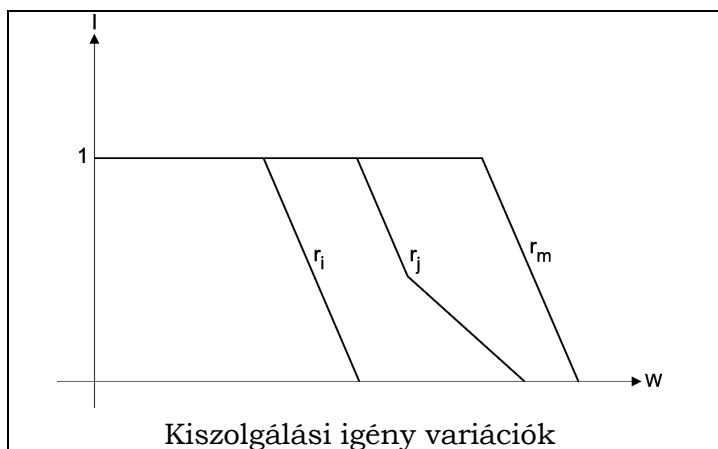
(6) a programokat átrendező döntési algoritmus ezek után az itt kiszámított $\underline{r}^{(l)}$ értékkel hajtható végre

\underline{L}	w_1	w_2	...	w_l	...	w_n
r_1	1	l_{12}	...	l_{1l}	...	l_{1n}
r_2	1	l_{22}	...	l_{2l}	...	l_{2n}
...
r_j	1	l_{j2}	...	l_{jl}	...	l_{jn}
...	
r_m	1	l_{m2}	...	l_{ml}	...	l_{mn}

Az L kiszolgálási és terhelési mátrix

A (4) és (5) pontok mátrixa és a kapcsolódó művelet képletesen azt fejezi ki, hogy növekvő terhelés, - például w_l - esetén az előre meghatározott, - például r_j - igénylési szinten belépett program az i -edik időintervallumban csak

kevesebb kiszolgálást - $\underline{r}^{(l)} = \underline{r}^{(1)} \cdot \underline{l}_l$ - igényelhetnek.



Az \underline{L} mátrix elemeinek értéke a rendszerről szerzett ismeretek alapján tapasztalati úton határozható meg.

Az apriori ismeretek kiváltása.

Azaz helyettesítés adaptív szabályozással. Az előzőekben bemutatott szabályozás a rendszer működését számottevően javítja és megakadályozza trashing jelenségek kialakulását.

Ennek azonban feltétele, hogy a rendszer működéséről információk álljanak rendelkezésre, úgymint

(1) $\underline{r}(r_1, r_2, \dots, r_i, \dots, r_m)$ monoton növekvő sor a kiszolgálási igényszintek sora, amelynek alapján a programok kiszolgálást igényelnek;

(2) $\underline{w}(w_1, w_2, \dots w_n)$ monoton növekvő sor a rendszer terhelésének állapotváltozást is jelentő küszöbszintjeinek sora;

(3) \underline{L} mátrix az igényszintek terheléstől függő csökkentési faktorai.

A problémát tulajdonképpen az okozza, hogy ezek az ismeretek csak a tényleges működés során, - gyakorlati tapasztalatok útján - szerezhetők meg az időbeli függés miatt és az üzemeltetők részéről rendszeres ellenőrzést igényel az esetleges beavatkozás felfedezésére (az időbeli függés miatt, az eltérő viszonyok között eseteleg szükség lehet vektorok, illetve a mátrix cseréjére, egyes értékeinek megváltoztatására).

Ez lényegében azt jelenti, hogy a szabályozással a rendszer csak egy *szűkebb tartományban* lesz adaptív a változó terhelésekre. Ebből a tartományból kilépve önmaga által indukált működési helyzetbe kerül, ami már nem a kitűzött célt szolgálja.

A probléma feloldására, az apriori ismeretek szükségességének csökkentésére, a gyakorlati tapasztalatok beépítésének automatizálására, a működési tartomány szélesítésére további adaptív szabályozást vezettünk be.

A módszer lényege, hogy a programok kiszolgálási igény szintjeit a pillanatnyi állapotoknak megfelelően az optimum irányába változtatjuk.

A programok tehát mindig az általuk igényelt legmagasabb kiszolgálási szinten legyenek, amelyeket azonban a rendszer általános terhelése korlátoz.

(1) Legyen s_j az i -edik időintervallumban a j -edik program által nyert összes kiszolgálás ugyanúgy, ahogy azt az előzőekben definiáltuk.

(2) Legyen adott $\underline{r}(r^1, r^2, \dots, r^l, \dots, r^m)$ tetszőleges elemszámú sor, amelynek egyes elemei s_j -vel egyező mértékű küszöbszintek oly módon, hogy

$$r^l < r^{l+1} \quad \forall l\text{-re.}$$

(3) Az új programok a várakozó sor élére kerülnek (LCFS), hogy minél előbb bekerüljenek a kiszolgálási igényszintet meghatározó folyamatba. Az \underline{r} sorból egy átlagos értéket rendelünk hozzá, amelyet a belépéskori kiszolgálás alapján határozzunk meg:

$$\text{ha } r^{l-1} < \frac{1}{n} \sum_{j=1}^n s_j \leq r^l, \text{ akkor } r_j = r^l$$

Az iniciális page-k száma $p_{j_0} = \frac{N}{n}$ lesz az egy

programra jutó átlagos page-szám.

(4) Interaktív programok a terminálról való visszatéréskor természetesen a korábbi \underline{r} sorbeli rangjuknak megfelelő értékkel és a várakozó sor végére térnek vissza (mivel futtatásuk a korábbi állapotnak megfelelő folytatást jelenti).

(5) Az identifikáció a már leírtakkal azonos módon történik: $q_j = r_j - s_j$ minden kiszolgálási egységben lévő programra és ugyanúgy képezzük a három osztályt a q_j értékeknek megfelelően.

(6) A döntés a három osztályra az alábbi, feltéve,

hogy $r_j = r^l$ az i -edik időintervallum végén:

ha $q_j < 0$, akkor $r_j = r^{l+1}$

ha $q_j \approx 0$, akkor $r_j = r^l$

ha $q_j > 0$, akkor $r_j = r^{l-1}$

(7) A döntés második lépésében a Döntési funkció rész (3), (4), (5), (6) pontjaival azonos módon járunk el.

(8) A beavatkozási funkció is azonos a Döntési funkció részben meghatározottakkal.

A folyamatból jól látható, hogy a programok rangja (r_j) a kiszolgálási igényük teljesítése alapján változik.

- Ha kevesebb kiszolgálást kapott ($q_j > 0$), akkor kiszolgáláson marad, de most már alacsonyabb igényszinten.

- Ha több kiszolgálást kapott ($q_j < 0$), mint amit a pillanatnyi rangja jelez, akkor valószínűleg magasabb kiszolgálást igényel és ezért a rangja növekszik. Igaz ugyan, hogy a terheléstől függően ehhez csak később jut hozzá (a várakozó sorba került).

- Az átlagos kiszolgálást ($q_j \approx 0$) nyert programok a terheléstől függően (nagyobb terhelésnél nincs $q_j < 0$) tovább futnak, vagy késleltetésre kerülnek.

- Amennyiben minden program a $q_j > 0$ osztályba kerül, akkor ez még nem befolyásolhatatlan túlterhelési állapot, mint az eredeti algoritmusnál, Döntési funkció rész (3)-ban), mivel a prog-

ramok igénylési rangja csökken. Ezáltal a következő ciklusban, vagy később biztosan kerülnek programok a másik osztályokba és ismét léphetnek kiszolgálásra programok a várakozósorból is.

A döntési funkció fentieknek megfelelő módosításával tehát a rendszer működése adaptívá vált a különböző terhelési viszonyokra.

A terheléstől függő késleltetéseket az egyes programok között olymódon osztja el, hogy azok késnek többet, amelyek korábban több kiszolgálást kaptak. A szabályozás ilyen megoldása szükségtelenné teszi az eredeti algoritmushoz igényelt vektorok és mátrixok apriori ismeretét, illetve létrehozásukat egyáltalán. Szükség van ugyan egy \underline{r} vektor kialakítására a kiszolgálási igényszintek jellemzésére, értékei azonban nem kritikusak. A szabályozás határciklusa az optimum ($q_j=0$ minden j -re), mivel ilyenkor minden program az igénye szerinti kiszolgálásban részesül ($r_j=s_j$). A határciklus megfelelő érzéketlenségi sávval ($|q_j| \leq \varepsilon$) beállítható oly módon, hogy

$$\underline{r}^{(l+1)} - \underline{r}^{(l)} \leq 2\varepsilon \quad \forall l\text{-re}$$

Ez a beállítás tulajdonképpen nem kritikus, mivel a gyakorlatban a határciklus csak közelíthető a programok befejeződése és az új programok belépése miatt.

A szabályozás stabil, az optimum irányába hat, mivel a határciklusa felé konvergál, vagyis

- ha $q_j < 0$, akkor r_j növekszik
- ha $q_j > 0$, akkor r_j csökken és

- r_j változásának mértéke $\geq 2\varepsilon$.

A szabályozás általában nem optimális, csak közelíti azt, de feltétlenül az optimális működés irányába hat (az optimális szabályozás a folyamat jellege miatt mindig, és itt is, tendencia jelleggel érvényesül).

Összefoglalás.

(1) Összefoglalva tehát megállapítható, hogy az adaptív szabályozás bevezetésével a gép kihasználása javul, ugyanakkor rövidül az átfutási idő is.

(2) Az általános javulás oly módon volt elérhető, hogy az adaptív folyamat révén a működés rendjét mintegy „hozzaigazítottuk” az időben változó felhasználói igényekhez.

(3) Minden esetben a szabályozás bevezetésével a rendszerben tartalékok is keletkeztek, pedig a szabályozás nélküli esetekben a rendszer túlterhelésben, sőt szélsőséges (trashing) működési állapotban volt.

(4) Az adaptív szabályozással működő rendszer is kerülhet túlterhelési állapotba, ez azonban nem okozhat szélsőséges működést, amint azt kimutattuk.

(5) Felmerülhet, hogy a túlterhelési állapotban az adott szabályozás már nem kielégítő és a rendszer bővítésére, vagy magasabb szintű adaptivitás bevezetésére - a túlterheltség felismerésére és annak alapján alkalmas beavatkozás elvégzésére - van szükség.

Ennek a kérdésnek a vizsgálata további kutatások tárgya.

Függelék.

- [1] Y. Z. CIPKIN: Optimization, adaptation and learning in automatic systems. Computer and Information Sciences. 15-32. oldal Academic Press, 1967, New-York
- [2] Y. Z. CIPKIN: Principles of dynamic adaptation in automatic systems. IFAC 5th World Congress, Paris, 1972.
- [3] J. KIEFER-J. WOLFOWITZ: Stochastic Estimation of the maximum of a regression function. Ann. Math. Stat. 462-466- oldal, 1952.
- [4] H. ROBBINS-S. MONROE: A stochastic approximation method Ann. Math. Stat. 400-407. oldal, 1952.
- [5] B. WIDROW: A statistical theory of adaptation. Pergamon Press, 1963.
- [6] V. KAZAKEVICS: Az extrémális szabályozásról. Automatika és telemechanika. 1966. No 12.
- [7] CSÁKY F.: Irányítástechnikai kézikönyv. Műszaki 1977.
- [8] CSÁKY FD.: Lineáris szabályozási rendszerek analízise és szintézise. Műszaki 1977-1978.
- [9] AMBRÓZY A.-JÁVOR A.: Mérésadatok kiértékelése. Műszaki 1976.

- [10] PRÉKOPA A.: Valószínűségelmélet. Műszaki 1974.
- [11] KÓSA A.: Optimumszámítási modellek. Műszaki 1979.
- [12] G. A. KORN: Matematikai kézikönyv műszakiaknak. Műszaki 1975.
- [13] P. R. BLEVINS: Aspects of a dynamically adaptive operating systems. Ph.D dissertation. University of Austin. 1972.
- [14] W. A. WOLF: Performance monitors for multiprogramming systems. 2nd symposion on operating system principles 175-181. oldal Princeton. 1969.
- [15] V. RAMAMOORTY: The analytic design of a dynamic lookahead and programming segmenting sístems for multiprogrammed computers. ACM National Conference 229-239. Washington, 1966.
- [16] P. DENNING: The working set model for program behaiour. ACM Communications 323-333. vol.11. 1968.
- [17] G. ESTRIN, D. HOPKINS, B. COGGAN, S. D. CROCKER: SNUPER COMPUTER. A computer in instrumentation automation. Proc. AFIPS 1967. Conference 615-656. 1967.
- [18] R. G. HAMLET: Efficient multiprogramming resource allocation and accounting. AXCM Communications 337-343. vol.16. 1973.
- [19] S. SHERMAN, F. BASKETT, J. C. BROWNE: Trace driven modeling and analysis

of CPU in a multiprogramming system. ACM SIGOPS workshop in Cambridge, MA. 1971.

[20] S. MARSHALL: Dynamic calculation of dispatching priorities under OS/360 MFT. Datamation 93-97. vol.15. 1960.

[21] F. STEVENS: On overcoming high priority paralisis in multiprogramming system --a case history. ACM Communications. 539-541- vol.11. 1968.

[22] K. D. RYDER: A heuristic approach to task dispatching. IBM System Journal 189-198. vol.8. 1970.

[23] G. COFFMAN: Analysis of two time-sharing algoritms designed for limited swapping. Journal of ACM. vol.15. 1968.

[24] A. J. BERNSTEIN, J. C. SHARP: A policy driven scheduler for a time-sharing system. ACM Communications. 74-78. vol.14. 1971

[25] L. KLEINROCK: A continuum of time-sharing algoritms. Proc. AFIPS. 453-458- vol.36. 1970.

[26] A. WOLLISZ: A detailed system overhead including description of priority algoritms for real-time scheduling. Operációs rendszerek elmélete, Téli iskola, Visegrád. SZTAKI Közlemények 61-86. vol.18. 1977.

[27] TÓKE P.: Kötegelt és kollektív felhasználást támogató dinamikusan adaptív vezérlés. Operációs rendszerek elmélete, Téli iskola, Visegrád. SZTAKI Közlemények 87-98. vol.18. 1977.

- [28] GYARMATI P.: Dinamikus erőforrás elosztás vegyes üzem esetén. Operációs rendszerek elmélete, Téli iskola 1975. SZTAKI Közlemények 201-209. vol.18. 1977.
- [29] G. BERGHOLZ.: Zur analyse des Echtzeitbetriebe von Prozessorrechnersystemen. Operációs rendszerek elmélete téli iskola 1976. SZTAKI Tanulmányok 141-158- vol.69. 1977.
- [30] GYARMATI P.: Feedback controls in the operating systems. Operációs rendszerek elmélete, Téli iskola, Visegrád. 1976.
- [31] P. R. BLEVINS, D. V. RAMAMOORTHY: Aspects of a dynamically adaptive operating system. IEEE Transactions on computer 713-735. vol.25. 1976.
- [32] J. LEROUDIER, PARENT: Discrete event simulation modeling of computer systems for performance evaluation. IRIA Laboria, rapport de recherche 1. 77. 1976.
- [33] Y. BARD: Experimental evaluation of systems performance. IBM Systems Journal. 302-314. 1973/3.
- [34] A. BRANDWAJN: A model of time-sharing virtual memory system solved using equivalence and decomposition method. Acta Informatica 11-48. vol.4. 1974.
- [35] P. J. COUTOIS: Decomposability, insubabilities and saturation in multiprogramming systems. ACM Communicatins 371-377. vol.18. 1975.
- [36] M. BADEL, E. GELENBE, J.

LEROUDIER, D. POTIER: Adaptive optimization of a time-sharing systems performance. Proc. of IEEE. 958-965. vol.63. 1975.

[37] A. BRANDWAJN, J. A. HERNANDEZ: A study of a mechanism for controlling multiprogrammed memory in an interactive systems. Ecole nationale superieure des telecommunications. Paris. 1978.

[38] Y. BARD: The modelling of some scheduling strategies for an interactive system. Symp. on Performance measurement and evaluation. 113.117. Yorktown Heights. 1977.

[39] A. GECK: Performance improvement by feedback control of the operating system. Informatik Rechnerabteilung Universitat Karlsruhe. 1978.

[40] T. STOREY, S. TODD: Performance analysis of large systems. Software practice and experience. 363-369. vol.7. 1977.

[41] P. DENNING, J. BUZEN: Operational analysis of queuing networks. Measueing, modelling and evaluating computer systems. North-holland Publishing Co. 151-172. 1977.

[42] T. BERETVAS: A simulation model representing the OS/VS2 rel.2. control program. Proc. of a symposium on Operating systems. Springer Verlag. 1974.

[43] P. GYARMATI: On the adaptiv control of operating systems. Operációs rendszerek elmélete, Téli iskola, Visegrád. 1977.

[44] ed.GYARMATI P.: KSH Rendszertechnikai Közlemények. KSH Rendszertechnikai Osztály.

[45] ed. GYARMATI P.: KSH Üzemeltetési szabályok IBM 370. KSH Rendszertехnikai Osztály.

[46] J. Y. BABANNEU, et.al.: Automativ and general solution to the adaptation of programs in paging environment. Proc. of 6th ACM symposium on operating systems. 109-116. nov. 1977.

[47] E. G. COFFMAN, T. A RYAN: A study of storage partitioning using a mathematical model of locality. Comm. of ACM. March. 1972. vol.15. 185-190.

[48] G. S. SHEDLER: A queuing model of multiprogrammed computer with a two-level storage system. Comm. of ACM. Jan. 1973. vol.16. 3-8.

[49] Proceeding of the IBM seminar on virtual machines. Uithoorn, 1972. Belgium. ed: G. Kjellin, IBM Public Sector Industrial Center, Brussels.

~~~~~



# Az Antigravitáció, avagy a Lebvacska.

*Legújabb tudományos eredmények, sorozat.*



*HÍR: Felfedezték a gravitációnak ellenálló Lebvacsát, amelynek segítségével az illetékesek szerint lehetséges lesz mindenki számára a tömegmentes tömegközlekedés. A bevezetésével még némi problémák vannak, ezeket azonban EU támogatással a kormányzati erők oda-hatása révén kiküszöbölni látszik a tudós társadalom. Részletesen...*

## *Hipotézis.*

Az antigravitáció létezik, meghozzá itt a Földön!

### *Axióma 1.*

Ha leejtünk egy szelet vajaskenyeret, akkor az a vajas felére esik!

### *Axióma 2.*

Ha kidobjuk a macskát, akár magasról is, mindig a talpára érkezik.

### *Tétel.*

Legyen egy komplex rendszer, figyelemmel a fenti axiómákra, az alábbi:

- erősítsünk a macska hátára vajaskenyeret a vajas felével felfelé.

A komplexumot dobjuk ki az ablakon és figyeljük meg, mi fog történni?



### *Lemma 1.*

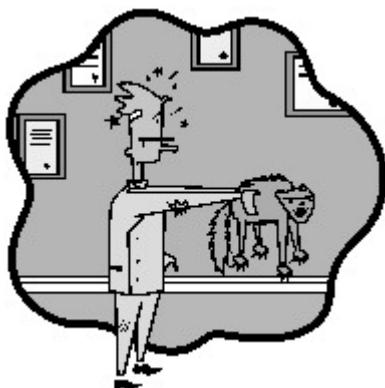
A komplexum mindig a talpára esik.

A Lemma 1. cáfolata: ha a macskának lenne igaza, akkor a vajtan sérül, - nevezetesen, a vajnak kell előbb a földre érnie.

### *Lemma 2.*

A komplexum a vajas felére esik.

A Lemma 2. cáfolata: ha a vajaskenyérnek van igaza, akkor a macskaság sérül, - nevezetesen, hogy a macskalábnak kell először érintenie a földet.



### *Corollárium 1.*

A rendszerünk, a - vajaskenyérmacskával - „natura paradoxont” képez, mivel mindkét tételt megcáfoltuk, vagyis semmiképpen sem érhet földet a diszkutált komplexum.

### *Általánosítás.*

Legyen a kísérlet tárgyát képező komplexum egy *precedens*, amelyet az Axióma 1. és az Axióma 2. szerinti *minták* alkotnak. Tekintsünk most el az őket összekapcsoló, rögzítő elemektől.



(*Segédétel* és később bizonyítandó, hogy erre a komplexumunk invariáns).

Mivel a mintákra semmiféle kikötést nem tettünk, ezért feltesszük, hogy a kísérlet azoknak a természetben előforduló bármelyik reprezentánsára érvényesek.

### *Bizonyítás.*

A vajaskenyérség és a macskaság létezése óta nem fordult még elő az Axióma 1.-től és az Axióma 2.-től eltérő eset (hiszen ezért axiómák azok). Továbbá a fenti minták a földön bárhol létezhetnek.

Mindezekből következik, a

### *Corollárium 2.*

Eszerint bármelyik minta alkothat a precedens szerinti komplexumot.

A Corollárium 1. és a Corollárium 2. következtében az alábbi predikáció mondható ki bármelyik, a precedens szerinti, komplexumra:

### *Thesis.*

A - macska vajaskenyérrel a hátán - rendszer kidobva, leejtve nem esik le, lebegve marad. Ellenáll a gravitációnak és avval azonos nagyságú, de ellentétes irányú antigravitációt kelt!

Az ilyen rendszert mostantól tudományos jelentősége miatt önálló névvel illetjük és legyen a neve: LEBVACSKA!



## *További vizsgálatok és a jövő feladatai.*

Következmények és további kérdések a téma kutatásához:

### *Postulatum 1.*

Az antigravitáció számítása: Alapvetően egyensúlyi számításra van szükség, mert a Lebvacska két eredendő erő-rendszer összességéből tevődik össze. Ezek:

1. Macska-forgató erők, amelyek a macska lábprioritású földet érését határozzák meg;
2. A kenyér-tekerő és vaj-gravitáló erők együttese, amelyek a vajprioritást okozzák.

### *Postulatum 2.*

A hypothesis szerint a Lebvacska olyan magasan helyezkedik el a föld felett, ahol ezen erők éppen kiegyenlítik egymást!

Szerkesztendő egy Lebvacsckára szerelhető és távvezérelhető eszköz, amely ezeket a funkciókat ellátja és ezáltal a Lebvacsckát kívánság szerint mozgatni lehessen!

### *Postulatum 3.*

Szélső értékek:

Ha túl magasra kerül a Lebvacska, akkor az erős napsütés leolvaszthatja a vajat és a komplexum felbomlik (lezuhan, lásd Ikarosz!?).

Ha a macska megéhezik, megeheti vajaskenyeret. Megoldás lehet például olyan macska alkalmazása, amelyik nem szereti.

### *Postulatum 4.*

Vizsgálandó a Lebvacska működése a macska élettartama, a kenyér száradása és a vaj avasod-



dása esetekben is.

*Postulatum 5.*

Alkalmas eljárás kidolgozása szükségeltetik a biztonság és megbízhatóság (SS) kérdéskörében, ahogy a művelt *internetikus* mondaná:

*We received thru Spam the challenging question about the surviving of our specie –Lebvacska- on the dotnet. An RSS says, e-Trade Lebvacska in web2 only and behind the Vista firewall otherwise the specimen would be hit by worms. Diverse worms are enemies for either part, i.e. the dangerousity range is wider.*

*Megjegyzés.*

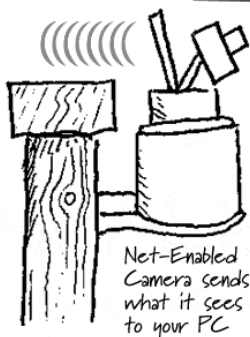
Az alkalmazott segédétel bizonyítása hiányában némi bizonytalanság érzékelhető még, amely azonban minden határon túl epszilonizálható, azaz a predikátum probabilitása egyszerűen maximalizálható (tart az 1-hez, amint epszilon a 0-hoz), ha a lemma szerinti corollárium komplexe a minták cedált primitívje. És ez számunkra megnyugtató.

Q.E.D.



# Web Watcher

You won't BELIEVE what you will SEE!



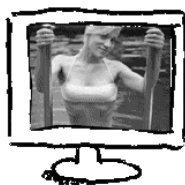
Keep an Eye  
on the Pool



Your Crazy  
Kids!



You neighbors dog  
making your yard  
his OWN!



Your PC

You SEE What your camera SEES!



Receiver gets wireless  
signal from 100 feet away!



**NEW!**



You, seeing what  
your cameras see  
from anywhere  
in the world!!!  
Can't beat that!



## *Believes about viruses.*

*The problem nearly as old as the computer networking, the data exchange among computers or even only among computer-people by using data carriers, tapes, floppies, etc. Now we are on this and shall try to explore and understand the situation and also try to glimpse to the future.*

*This artikel was first published as a white paper and for the conference: The Truth About Computer Security Hysteria, 2002.*

Theoretically you have many possibilities to secure your computer against viruses. Here are some of them for your convenience:

1. Create a hideous system, heavily find vulnerable places,
2. Watch flowing of data and find the pattern of viruses in the communication channels and in files, programs, etc.;
3. Keep under control your sytem, to reveal vulnerabilities and fix them soon.
4. Designing secure, flawless software.
5. Isolate environment from attackers, etc.

Now let's see how look these like in the real life today. Everyone already understood that viruses are a big role of our increasing networked world. But how big is indeed?

## *Security ideas.*

Most people believes that Linux systems would suffer the same frequency of virus problems as Microsoft Windows if they were as popular as



Windows is now. Such a comment ignores several factors that make up the vulnerability profile of Windows with regard to viruses.

The explanation is in the so called *security by obscurity*, which is a principle in security engineering. A system relying on security through obscurity may have theoretical or actual security vulnerabilities, but its owners or designers believe that the flaws are not known, and that attackers are unlikely to find them.

A closed source system looks like such a solution, because the would-be attacker does not have access them. But how true is this? Let's see the facts!

So we must understand that Linux-based systems and other open source OSes actually use the *security through visibility* approach. The security through visibility idea taken by popular open source software projects and includes two *sub-principles* of software security: security through transparency and security through popularity.

The *transparency* means that the source code will be more simple not containing hideous code for obscurity. The idea coming from the experiment that the reverse-engineering is much reliable tool than understanding the source-code, so the source code may be freely spread among people. This will not produce more vulnerability.

The *popularity* has twofold meaning: the more used software has more enemy, but also have



more expert and user who wants to have secure software. By the open source even more experts work with and the vulnerabilities will be recognized soon, and the solution for fixing nearly immediate. Against this is the closed system, where only a few expert are engaged such job, and also the way to recognizing is undetermined. So fixing vulnerabilities are long process.

Microsoft even doesn't fix virus vulnerabilities, they let this work for antivirus companies. This is quite a business, but what price for the users!

A virus is malicious code carried from one computer to another by some kind of medium - often an "infected" file. Once on a computer, it's executed when that file is "opened" in some meaningful way by software on that system. When it executes, it does something unwanted. This often involves, among other things, causing software on the host system to send more copies of infected files to other computers over the network, infecting more files, and so on. In other words, a virus typically maximizes its likelihood of being passed on, making itself contagious.

All of this relies on security vulnerabilities that exist in software running on the host system. For example, some of the most common viruses of the last decade or so have taken advantage of security vulnerabilities in Microsoft Office macro capabilities. Infected files that were opened in a text editor such as Notepad would



not then execute their virus payload, but when opened in Office with its macro execution capabilities would tend to infect other files and perhaps even send copies of themselves to other computers via Outlook. Something as simple as opening a macro virus infected file in Wordpad instead of Microsoft Word or translating .doc format files into .rtf files so that macros are disabled was a common protective measure in many offices for a while.

Macro viruses are just the tip of the iceberg, however, and are no longer among the most common virus types. Many viruses take advantage of the rendering engine behind Internet Explorer and Windows Explorer that's also used by almost every piece of Microsoft software available to one degree or another, for instance. Windows viruses often take advantage of image-rendering libraries, SQL Server's underlying database engine, and other components of a complete Windows operating system environment as well.

Viruses in the Windows world are typically addressed by antivirus software vendors. These vendors produce virus definitions used by their antivirus software to recognize viruses on the system. Once a specific virus is identified, the software attempts to quarantine or remove the virus - or at least inform the user of the infection so that some kind of response may be made to protect the system from the virus.

This method of protection relies on knowledge of the existence of a virus. This means that a virus



against which you are protected has, by definition, already infected already at least one computer or network and done its damage. Will it be you the martyr suffering from the damage to save others or the lucky fellow receiving the new virus definition to your antivirus software.

The life even worse than that, though. Each virus exploits a vulnerability - but they don't all have to exploit *different* vulnerabilities.

In fact, it's common for hundreds or even thousands of viruses to be circulating "in the wild" that, among them, only exploit a handful of vulnerabilities.

This is because the vulnerabilities exist in the software and are not addressed by virus definitions produced by antivirus software vendors.

These antivirus software vendors' definitions match the signature of a given virus — and if they are really well-designed might even match similar, but slightly altered, variations on the virus design. Sufficiently modified viruses that exploit the same vulnerability are safe from recognition.

Viruses can exploit the same vulnerability without being recognized by a virus definition, as long as the vulnerability not stopped by the vendor of the vulnerable software.

This is a key difference between open source operating system projects and Microsoft Windows. Microsoft leaves dealing with viruses to the antivirus software vendors, but open



source operating system projects generally fix such vulnerabilities immediately when they're discovered. So they are much less use antivirus software on an open source system. The only exception are the mail server and such as they communicate potentially virus-laden files among computers.

Very easy to produce a new virus that will slip past antivirus software vendor virus definitions, but in the open source software world the many expert always busy with open code will discover and patch a whole new vulnerability with high probability than a hacker.

### *The complexity of detecting viruses.*

There is another question about detecting viruses is the complexity so the time required to that job.

Lets consider the time of detecting a virus in some executable file. First, time depends on a set of possible variants of virus body. The more possible variants there are in the virus body, the more time needed to iterate them while checking files. This way is the simplest one, and it was chosen by viruses, when they morphed into crypt-, polymorphic-, permutating- and metamorphic- ones.

For sure, while checking file, all these variants are iterated very seldom (when their amount is small enough), and in most cases there performed special processing of code and then comparing with some mask, or more complex



algorithmic detection -- in this case we consider complexity of such checking, instead number of possible body variants. In case of poly-encrypted virus, this complexity should be increased by a complexity of polymorphic decryptor emulation. In case of simple poly decryptor, it can be the object of detection, instead of virus itself.

Well, at second, the time of detecting a virus depends on a set of possible virus locations (or entrypoints) in file. For example, if virus hooks entrypoint, it will be enough to perform a checking for this virus only at that address.

But, if virus inserts its body into random location in the file, then time of detection for this virus is multiplied by size of this file. (this is true for poly decryptors in which next instructions works depending on result of previous ones).

Let

C - complexity of checking file for some virus;

C<sub>i</sub> - complexity of checking virus in file at some specified address, including poly-decryptor emulation and checking for all possible variants of virus body;

I - number of possible addresses in file, where execution of virus body (or part of this body) can be started at.

Then  $C = C_i * I$

This trivial formula shows as an algorithm of checking a file for being infected by some



complex virus: for each possible address  $I$ , where virus can be located, perform checking for presence of this virus ( $C_i$ ).

The algorithm looks as follows:

```
for (i = 0; i < I; i++)      I
{                             *
    init_emul();
    emul(codeblock[i]);      Ci
    check_virus();           /
}
```

This tells us that

- bigger files are better targets to be infected,
- virus should be smaller but more complex,
- polymorphic decryptor should work longer,
- that number of possible virus locations in file should be maximal.

Lets consider now such case,

- when polymorphic decryptors consists of  $N$  parts,
- which are located in  $N$  different places of a file, and
- that this decryptor works only when all these  $N$  blocks are executed consecutively.

In this case,

- if antivirus does not emulate all the program,
- but only combinations of its blocks,
- possibly containing virus decryptor.

The complexity of such checking will look like:



$$C = C_i \frac{I!}{(I - N)!}$$

For example, if there are 4 suspicious blocks in the program, but there should be only 2 consecutively executed virus blocks, there will be checked 12 variants:

|       |       |       |
|-------|-------|-------|
| {0,1} | {0,2} | {0,3} |
| {1,0} | {1,2} | {1,3} |
| {2,0} | {2,1} | {2,3} |
| {3,0} | {3,1} | {3,2} |

So virus checking will look as following:

```

for(i1 = 0; i1 < I; i1++)
for(i2 = 0; i2 < I; i2++)
if (i2 != i1)
for(i3 = 0; i3 < I; i3++)
if (i3 != i1)
if (i3 != i2)
...
for(iN = 0; iN < I; iN++)
if (iN != i1)
if (iN != i2)
if (iN != i3)
...
{
init_emul();
emul(codeblock[i1]);
...
emul(codeblock[iN]);
check_virus();
}

```

*I!*  
 -----  
 (I-N) !  
 /  
 /  
 /  
 /  
 /  
 /  
 \*  
  
*Ci*  
 /  
 /



And now there appears a question: how could we know sequence of program instruction execution to insert virus blocks into such places, that they will be executed in some specified order?

This can be achieved by means of tracing the program. And tracing can be performed using *win32 debug api*. The selected program should be traced for some small (but random) period of time (seconds). While tracing process, the list of executed instructions should be build for each (or only main) thread.

Very important : we're tracing not program, but process, i.e. EXE file and some DLL files. As such, there appears possibility to insert these N decryptor blocks into different files, but with known execution order.

On small local parts of code all this work can be done without tracing, but by means of statical analysis of code instructions. For big multithreaded programs, consisting of lots of different files, and also in case of big distances between viral blocks, statical analysis (i.e. disassembling) is not enough, and tracing (or emulation) is required.

Now the flow of this work is:

1. Select some program.
2. Perform static analysis (parsing on instructions, e.g.: Mistfall and ZMist)
3. Perform dynamic analysis (tracing, e.g.: Tracer32), but taking into account the results of previous disassembling. For



example, breakpoint (INT3) should be inserted into beginning of each subroutine, to avoid losing control on callback functions, called from non-tracing DLLs.

4. Perform static analysis once again, with more quality, because taking into account instruction list, obtained while previous tracing.
5. Retry from step 3 if needed.
6. Combine all obtained data.
7. Select some random places of the program, which are executed in some known order, to insert decryptor parts into.
8. Integrate decryptor parts and encrypted body into code section.
9. Build new program.

As a result, by means of synthesing two known technologies, there appears possibility to make virus detection much more harder, and, as such, to hammer another one nail into kaspersky's bottom.

At last some suggestion.

Viral blocks should be of minimal length (few instructions), but N should be big enough. The sequence of executing these N blocks should be complex, including big amount of iterations. This can be achieved by finding cycles in program while tracing it, and then integrating viral blocks into these cycles.



## *Summary.*

What I hope showed that viruses need not simply be a “fact of life” for anyone using a computer. Antivirus software is basically just a dirty hack used to fill a gap in your system’s defenses left by the negligence of software vendors who are unwilling to invest the resources to correct certain classes of security vulnerabilities.

The truth about viruses is simple, but it’s not pleasant. The truth is that you’re being taken to the cleaners — and until enough software users realize this, and do something about it, the software vendors will continue to leave you in this vulnerable state where additional money must be paid regularly to achieve what protection you can get from a dirty hack that simply isn’t as effective as solving the problem at the source would be.

Also keep in your mind the curious question: *who are the „bad-boys” and why are they „bad” and are they „bad”?*

~~~~~


Page ranking algorithm

avagy, a weblap „fontosságának” megállapítási módjáról.

Bevezetés

Évekkel a kidolgozás, majd a sikeres bevezetés, óta időszerű, hogy áttekintsük, mi is ez a híres rendszer, amely meghódította szinte a teljes 3w-t (www, world-wide-web, világháló, internet-nek is mondjuk) és előidézője több más jó eredménynek. Az alapgondolat, mint előidéző, ma is érvényes, nevezetesen, hogy az akár-mennyire is megnövekvő hálózat, egyenjogú és személytelen maradjon, legyen független szervezetektől, kormányoktól, stb.

Legyen tehát egy olyan rendszere az embereknek, az egész világra kiterjedően, amelyből bárki tetszőlegesen információkat szerezhethet be – abból, ami a hálózaton található - és amelyen az ember a józan emberi mivoltából fakadóan tetszőlegesen információt közölhet.

Tudom ez az óhaj számos kérdést vet fel, amelyre most nem válaszolunk, azonban engedjék meg, csak itt és most, ez a szabadoság a teljes jószándék feltételezésével. Meglátjuk, az eredmények igazolni fogják. Lássuk tehát:

A 3w, mint logikai, virtuális hálózat egy bizonyos értelemben egységes – mindenki által érthető, értelmezhető - szerkezetű írásokból, a weblapokból, vagy röviden lapokból, és az

ezeket elérni szándékozók kereséséből származó elérésekből, *hivatkozásokból* áll.

A fizikai rendszer természetesen tartalmaz a valódi hardware hálózatban számos egyéb segédeszközt, a különböző célú kiszolgálókat, stb. ezek azonban akkor működnek jól, ha a mi rendszerünkben észrevétlenek maradnak!

Nyilvánvaló, hogy a különböző lapok más és más információkat tartalmaznak és más a hozzáférésük. Egyeseket többször, gyakrabban keresnek fel, mint másokat. A lapoknak ezáltal különböző fontosságot tulajdoníthatunk: *kimondhatjuk, egy lap annál fontosabb a hálózat szempontjából, minél több a rá való hivatkozás.* Vagyis egy keresésnél a fontosabb lapokhoz fordulunk először, hiszen már bizonyították, hogy a keresett információ inkább rajtuk található! *Ezt az elvet kívánjuk megvalósítani!* Nézzük hogyan!

Miért is kell egyáltalán, miért is lényeges ilyen megkülönböztetést tenni?

A hálózatban rengeteg lap van, például a 3w-n milliárdos nagyságrendben, ha minden keresésnél végig kellene menünk a lapokon, amíg meg nem találjuk a kívánt információt tartalmazót, akkor nagyon lassú és ezáltal használhatatlan lenne a hálózat. Ennek feloldására számos megoldás készült, nem egy az emberek sokéves gyakorlatából származtatja magát: például eszünkbe sem jut alma vásárlásakor a ruha-boltba menni. Mit is teszünk tulajdonképpen? Felteszünk egy rendező elvet, amely mentén fogjuk a keresést végezni, ezáltal lerövidítjük a műveletet. A fenti példában szakcsoportokat

képeztünk és a keresésben először azt állapítjuk meg, majd csak azok között „lapozunk”. Jó megoldás, de például a multidiszciplináris esetek itt határesetek, nem eldönthetőek.

Számos más megoldás is elképzelhető, sőt más-más hálózatokon ma is alkalmaznak különböző technikákat. Mi most egy olyan algoritmust szeretnénk bemutatni, amely az eredményét tekintve nem túl bonyolult eljárással meghatározható, adaptív a változó hálózatra, képes tetszőleges nagyságú hálózatban a keresést optimalizálni, de legalább elfogadható sebességen tartani.

Ha tudnánk, hogy az egyes információkra mennyire van szüksége a keresőknek, vagyis, ha tudnánk az információk ilyen értelmű értékét, akkor csak annyit kellene tennünk, hogy az információk helyeit eszerint sorba rakjuk. Sajnos minden próbálkozás erre nézve sikertelen és előbb-utóbb szaknévsorokká válnak. Így például a könyvtárak, az információs központok, a telefon és más kommunikációs elérések, mind valahogyan téma szerint csoportosulnak és azon belül abc és más nem logikus sorrendekben találhatóak. Ráadásul ezeknek még az is feltétele, hogy az egyes témacsoportok megegyezzenek abban, hogy mit ajánlanak (ha mindkettő adja nem nagy baj, de, ha egyik sem?!). Látjuk, tudjuk, mindennap találkozunk ezekkel!

Vajon lehetséges-e olyan megoldást találni, ahol a szolgáltatók függetlenek egymástól, ahol a keresést nem korlátozza a hálózat tömege, amely a változó igényekhez adaptálható, amely

egyaránt hasznos a tartalmi keresés esetében is. Nos erre született, a 3w hálózat gigantikus méretéhez, a bevezetőben elmondott gondolat a *Stanford Egyetemen*. A hálózatban a weblapokat jellemezhetjük a rá való hivatkozásokkal! Ez az adat, akár ismerjük értékét, akár nem, létezik a hálózatban! Ha feltesszük, hogy ezt ki tudjuk számítani és eszerint a lapokat „sorba állítani”, akkor megoldottuk a feladatot, hiszen nyertünk egy olyan keresést, amely pontosan a hálózaton nyüzsgők kívánsága szerint működik! Ráadásul, ha ezt a sorrendet időnként felülvizsgáljuk, akkor eleget teszünk az időben változó követelményeknek is! Továbbá, az egyes lapok teljesen függetlenek egymástól, azaz semmiféle megállapodásra, információ elosztásra nincs szükség. A hálózat méretére sem tettünk semmiféle kikötést!

Tehát a feladat ennek a hálózatban létező számnak a meghatározása!

Legelőször nevezzük el: legyen ennek a mérőszámnak a neve *lap fontossági szám*, annak a mértéke, hogy a környezetében lévő oldalak hányszor hivatkoztak rá. A hivatkozott lap szempontjából sokszor *visszafelé hivatkozásnak* fogjuk hívni.

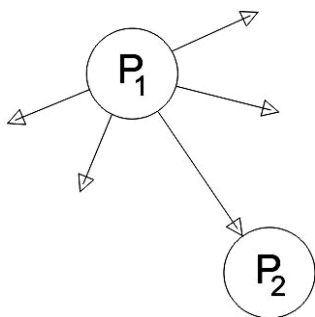
Minél nagyobb ez a szám, annál fontosabb az oldal, annál nagyobb a valószínűsége, hogy a keresett információ ott található. Ha a sikeres találatok számát rögzítjük és eszerint újra-rendezzük a keresési sorrendet, akkor az adott helyzetre pontosíthatjuk, aktualizáljuk a keresést.

Szükségünk lesz tehát egy olyan rekurzív

eljárásra, algoritmusra, amely ezt a lapfontosságot előállítja a hálózat teljes egészére, minden lapra. A fontosság kezelésének és a keresés, vagyis a kulcsszavak és kifejezések kapcsolatainak technikájáról most nem beszélünk. Feladatunk „csupán” annak a matematikai módszernek a feltárása, amelynek segítségével a lapfontosság mérhető, megállapítható tetszőleges nagyságú és struktúrájú hálózatban.

A hálózat

A hosszú bevezető után térjünk a lényegre és nézzük először, hogyan is néz ki a fontosság a hálózat saját szerkezetében:



Legyen egy hálózat egy irányított gráf az alábbi értelemben:

1. a gráf egy pontja a hálózat egy lapja;
2. a gráf egy éle hivatkozás egy lapra más lapról, minden lapról külön él fut minden külön lapra

való hivatkozáshoz.

3. a gráf él iránya a hivatkozás iránya, a nyíl a hivatkozott lapra mutat.
4. A P_1 lapnak k_1 számú hivatkozása van különböző lapokra, köztük egy a P_2 -re.

Ilyenkor azt mondjuk, hogy a P_1 $\frac{1}{k_1}$ fontosságot ad P_2 -nek!

Definíció:

Legyen P_i a hálózat egy lapja, amelyhez feltételezzük, hogy csatlakozik a $P_1; P_2; \dots; P_n$ lap, rendre $k_1; k_2; \dots; k_n$ csatlakozással, akkor a

P_i fontossága: $f_i = \frac{f_1}{k_1} + \frac{f_2}{k_2} + \dots + \frac{f_n}{k_n}$, vagyis a P_i

lap fontossága a ráhivatkozó lapok fontosságának összege!

Tulajdonságok:

- A lap tevékenysége *invariáns* a saját fontosságára nézve.
- A lap fontossága *visszamenőleges* szám, azaz az őt keresők határozzák meg.
- Egy „új” hálózatban a lapok fontossága *definiálatlan* és a hivatkozások hozzák létre, azok útján terjed szét a hálózatra! A fontosságnak csak a már működő hálózatban van értelme (említettük, hogy a hálózatban *létező mérőszámról* van szó).
- Új belépő lap, vagy nem hivatkozott lap fontossága nem megállapítható, *definiálatlan*!
- Regulárisnak* nevezzük azt a hálózatot, amelynek bármely lapjáról el lehet jutni bármely másik lapjára.
- Tisztának* nevezzük azt a hálózatot, amelyben minden lapnak van hivatkozása, vagyis létezik a fontossági szám.
- Monogámnak* nevezzük azt a hálózatot,

amelynek egy kereső rendszere van.

A kapcsolati mátrix

Egy tetszőleges hálózatot leírhatunk táblázatos formában, ahol a sorfejek és az oszlopfejek rendre a hálózat lapjai, míg a tábla belseje a lapok közötti súlyozott kapcsolatot tartalmazza: egy n elemű hálózat esetében n sorból és n oszlopból álló táblázatot kapunk.

P	1	2	3	...	i	...
1						
2						
3						
...						
j					$1/k_j$	
...						

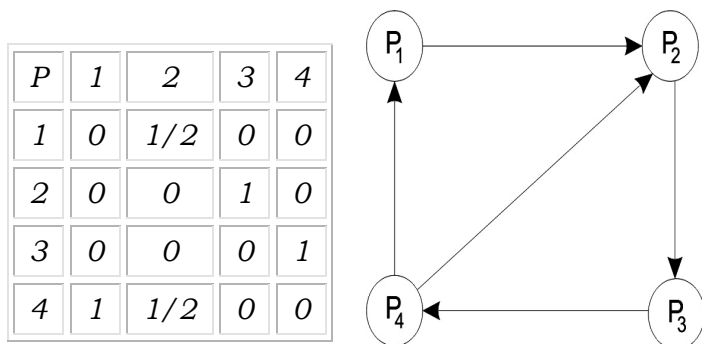
Az i -edik oszlop és j -edik sor kereszteződésében lévő tábla elem a P_i lapnak P_j laptól való hivatkozására vonatkozik:

= 0, ha P_j -nek nincs hivatkozása P_i -hez;

= $1/k_j$, ha van hivatkozás és k_j a P_j összes hivatkozása.

A táblázat tartalmából négyzetes mátrixot képezhetünk, amely pontosan egy n elemű hálózat kapcsolati rendszerét írja le, ezért *kapcsolati mátrixnak* nevezzük.

Egy példán mutatjuk be ezt a hálózatileírási technikát. Legyen egy hálózat az alábbi:



Például a második elem az első sorban $\frac{1}{2}$, mert P_2 -re van hivatkozás P_1 -ről és az összes P_2 -re hivatkozás $k_2=2$.

Akkor a mátrix:

$$K = \begin{bmatrix} 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & \frac{1}{2} & 0 & 0 \end{bmatrix}$$

Láthatjuk, hogy az oszlopok összege mindenütt 1, mivel az, a súlyozott fontosságok összege.

A fontossági vektor

Tegyük fel, hogy ismerjük minden egyes lap fontossági mértékét: a $P_i \Rightarrow f_i$, akkor az összeset sorbagyűjtve a teljes hálózat fontosságát kapjuk:

$$F = (f_1; f_2; \dots; f_i; \dots; f_n)$$

Az F egy n elemű vektor és a *hálózat fontossági vektorának* nevezzük.

Ha most visszatérünk egy pillanatra a kapcsolati mátrixhoz, akkor ennek analógiájára megállapíthatjuk, hogy a mátrix sorai rendre az egyes lapok fontossági sorrend vektorai.. Azt is mondhatnánk, hogy a teljes mátrix a teljes hálózat fontossági mátrixa!

A feladat tehát az $F = (f_1; f_2; \dots; f_i; \dots; f_n)$ vektor kiszámítása, azaz az egyes P_i lapok f_i fontossága, amely szerint a keresési sorrendet meghatározzuk. A legfontosabb lap tartalmazza a legtöbb keresett információt, hiszen így állítottuk fel a fontossági kritériumot!

Amennyiben a hálózat változik – a keresések átmennek jellemzően más lapokra – a kapcsolati mátrixot és a fontossági vektort újra elő kell állítani! Ennek módszere másik feladat.

Nézzük most, hogyan tudjuk meghatározni a fontossági vektort?

A feladatot már részben elvégeztük az előbbieken egy példa kapcsán azáltal, hogy felvételeztük a kapcsolati mátrixot! Ugyanis, ha feltesszük, hogy ismerjük az F fontossági vektort, akkor $K \times F = F$ azonosságnak fenn kell állnia a hálózat méretétől összetételétől

függetlenül, annak értelmében, ahogy azokat definiáltuk.

Hogyan?

A K mátrixunk $n \times n$ elemű;

Az F vektor n elemű;

$K \times F$ szorzat is n elemű;

Hogyan keletkezik $K \times F$?

A K mátrix *első sorának* elemeit rendre összeszorozzuk az F vektor elemeivel (elsőt az elsővel, másodikat a másodikkal, ... , az utolsót az utolsóval), majd ezeket összeadjuk: ez lesz a szorzat első eleme.

A fenti műveletet elvégezzük a K mátrix *második sorával*: ez lesz a szorzat második eleme.

A műveleteket az összes sorral elvégezzük: így megkapjuk a szorzatvektort.

Mellesleg ezt a műveletet a lineáris algebrában *lineáris kombinációnak* nevezzük.

Most már csak azt kell belátnunk, hogy ez a szorzatvektor azonos az F vektorral.

Emlékezzünk, hogyan építettük fel a K mátrixot: az első sor a P_1 lapra vonatkozó hivatkozásokat tartalmazza: - ha az adott P_i lapról nincs hivatkozás, akkor nulla, egyébként $1/k_i$; ahol k_i a P_i lap összes hivatkozása. Tehát az első elem 0 , vagy $1/k_1$; a második 0 , vagy $1/k_2$;...; az utolsó 0 , vagy $1/k_n$.

Akkor a szorzatvektor első eleme 0 , vagy f_1/k_1 ; a második 0 , vagy f_2/k_2 ;...; az utolsó 0 , vagy f_n/k_n . Ha ezeket összegezzük, akkor azt látjuk, hogy a szorzatvektor első eleme az összege azon fontosságoknak, amelyet a rá hivatkozó oldalaktól „kapott”, vagyis a saját fontossága!

Tehát nem más, mint f_i , ami az F vektor első eleme!

Az összegzést megismételve minden elemre, azt látjuk, hogy a szorzat-vektor éppen F , tehát $Kx F = F$.

Matematikai értelmezésben az ilyen tulajdonságú vektort a *mátrix sajátvektorának* nevezzük.

Pontosan azt mondjuk, hogy egy V vektor az M mátrix sajátvektora s sajátértékkel, ha az MxV szorzat eredménye az sV vektor (a skalár s szorzás azt jelenti, hogy a vektor minden eleme meg van szorozva s -értékével).

Ha a sajátérték $s=1$, akkor $MxV=V$, minden más esetben s -el való súlyozásról beszélünk.

Megjegyzendő, hogy a zérus vektor a fenti feltételeket mindig kielégíti, hiszen $0xM=0$, de semmit nem mond a hálózatról ezért érdektelen a számunkra.

Tehát a feladat mindig a *nem-nulla sajátvektor meghatározása*!

A példánkban feltételeztük, hogy $s=1$ és a mátrixalgebra felhasználásával

$F = (1/7; 2/7; 2/7; 2/7)$ adódott.

Értékeljük a vektort egy pillanatra: az a meglepő eredmény látszik, hogy az első lap kevésbé fontos, mint az összes többi, amelyek azonos fontosságúak!

Akárhogyan nézegetjük a hálózatunkat ez nem látszik rajta, úgy mondanánk nem-triviális!

Tehát már jól jártunk, mert kiderítettünk olyan hálózati tulajdonságot, amelyről nem tudtunk, nem is sejtettük! (ne feledjük a lap fontossága

jellemzően „nem saját tulajdonság”, ezért lehetségesek meglepő eredmények).

Manapság a hálózatok elég nagyok, sok eleműek, például a $3w \cdot 10^9$ körüli nagyságú és egy ilyen mátrix sajátvektorainak kiszámítása nem kis feladat. Ezért a gyakorlatban egyéb trükkök-höz és közelítő számításokhoz kell folyamodni. Ez legyen a programozók feladata, az elmélettel azonban még korántsem végeztünk!

Az elv, amit az eddigiekben felállítottunk több dolgot feltételez a hálózatról, amit matematikailag, mint az elmélet tulajdonságai felsoroltunk: eszerint a módszerünk *a relativ statikus, reguláris, tiszta hálózatok fontossági értékeinek meghatározására* alkalmas.

A fentiek mellett még azt is fel kell tennünk, hogy a hálózatunk *monogám*, azaz csak egy, a saját fontosság-mérő rendszerével, kereső rendszerével üzemel.

Fordítsuk le ezt a hálózat nyelvére:

Statikus. A hálózat „lassan” változik: a kapcsolati mátrix változása a fontossági vektort befolyásolja. A kérdés, hogy „milyen változás” esetén szükséges az újraszámolás, vagyis a fontossági sorrend újbóli megállapítása.

Reguláris. A hálózat bármely lapjáról el lehet jutni bármely lapjára a hivatkozások mentén. Ez a reguláris hálózat! A valóságban mindig vannak olyan csoportok, szigetek, amelyeknek nincs hivatkozása csoporton kívüli lapokkal. Azt is mondhatnánk, hogy ezek önálló hálózatok.

Tiszta. A hálózat minden lapjára van

hivatkozás. A valóságban számos olyan lap is létezik, amelyik részt vesz a hálózaton keresésben, azonban órá nincs hivatkozás! Mivel ezek kapcsolati mátrixbeli elemei mind nullák, a fontossága sem állapítható meg, pedig akármilyen aktivitással részt vehetnek a hálózatban!

Monogám. A hálózat minden hivatkozása ebben a rendszerben történik. A valóságban a lapokat másik kereső rendszerekből és közvetlen eléréssel is használják!

A *3w* hálózat azonban nem-statikusan, nem-reguláris, nem-tiszta és nem-monogám, tehát bajunk van, pedig az eddig felállított elmélet olyan jónak látszik. Ha ki tudnánk küszöbölni a „nem”-eket, akkor az elméletünk alkalmazhatóvá válna.

Nézzük meg hogyan segíthetünk ezen a „bajon”:

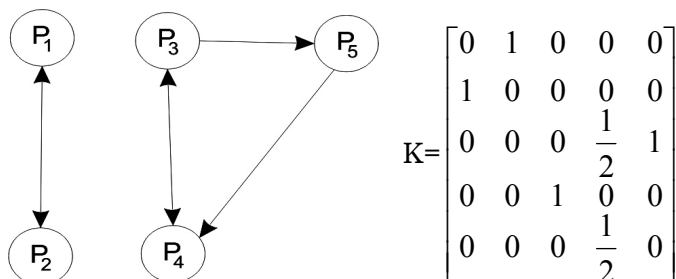
Nem-statikusan. Elméletünk nincs rá, de szerencsére a valóságos hálózatok nem sietik el a dolgukat, ami persze várható is tőlük. A kisebb változások programtechnikai módszerekkel elvégezhetők a keresők találati valószínűségei alapján.

Ha nagy kiesések vannak, akkor bizony vannak sebességi problémák, amíg a helyettes lap „bejáratozik”, azaz a helyettesállító „ugyanolyan paraméterekkel” be tudja üzemelni a pótlást. A *3w* például a Google-nál, nem-hivatalos adatok szerint, havonta egyszer futtatja újra a kapcsolati mátrixot.

Nem-reguláris hálózatoknál elméleti trükkhöz

folyamodunk, amelyet az alábbi példával mutatunk be.

Legyen egy hálózat adott az alábbi módon:



Ennek két sajátvektora van, amelyek nem egymás többszörösei, például a $(1; 1; 0; 0; 0)$ és a $(0; 0; 2; 2; 0)$. (Ha több megoldás van, akkor nem egyértelmű a dolog, ahogy azt vártuk is).

Ha a mátrixot két sub-mátrixra bontjuk, ahogy az a valós hálózaton is van, akkor mindkét mátrixnak van sajátvektora 1 sajátértékkel:

$$K_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; \text{ és } K_2 = \begin{bmatrix} 0 & \frac{1}{2} & 1 \\ 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \end{bmatrix};$$

akkor $F_1 = (1; 1)$ és $F_2 = (2; 2; 1)$

Ha kiegészítjük nullákkal a teljes elemszámra, rendre visszkapjuk az eredeti mátrix két sajátvektorát, tehát nem követtünk el hibát.

Lehetséges lenne tehát két független hálózatként kezelni, azonban ez egyrészt nem elegáns,

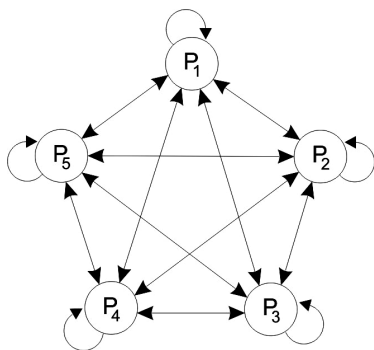
másrészt bármely pillanatban megváltozhat a dolog és a két subnet össze-kötődik.

Ha nem tudsz mást, hát folyamodj trükkökhöz: íme egy ilyen trükk, a súlyozott kombinálás.

Alkossunk egy új B mátrixot, amelyet az eredetivel összekombinálunk az alábbi módon:

$(1-m)K+mB$, valamely $m=0$ és 1 közötti számra, ahol K az eredeti, míg B egy olyan kapcsolati mátrix, ahol minden kapcsolódik mindennel, önmagával is és pont annyi eleme van, mint K -nak.

Talán így bonyolultnak tűnik, rajzoljuk fel, hát a legutóbbi példánkhoz a B mátrixot:



K elemszáma 5 , tehát ez is annyi és minden lapról él megy minden másik lapra és önmagára is, vagyis ennek a kapcsolati mátrixában minden egyes elem $1/5$.

Alkalmazzuk a fenti lineáris kombinációt:

-a K minden elemét megszorozzuk $(1-m)$ -el, míg a B -t m -el, majd összeadjuk rendre a kettő elemeit, képezve a kombinációs mátrixot. A Google nem adja közre csodaszámait, de azok előbb-utóbb kitudódnak: a mi példánkban az $m=0,15$ értékkel számolunk, mint a legutóbb kitudódtal.

Íme a kombinációs mátrix:

$$\text{Az eredmény: } \begin{bmatrix} 0,03 & 0,88 & 0,03 & 0,03 & 0,03 \\ 0,88 & 0,03 & 0,03 & 0,03 & 0,03 \\ 0,03 & 0,03 & 0,03 & 0,455 & 0,88 \\ 0,03 & 0,03 & 0,88 & 0,03 & 0,03 \\ 0,03 & 0,03 & 0,03 & 0,455 & 0,03 \end{bmatrix} ;$$

Ennek a mátrixnak egyértelmű sajátvektora van, $s=1$ sajátértékkel és ez az alábbi:

$(0,43895; 0,43895; 0,51331; 0,51066; 0,28287)$.
Tehát a trükkel definiáltuk az egyértékű fontossági vektort.

Nézzük meg a teendőket a *nem-tiszta* hálózat esetén, vagyis amikor van olyan lap, amelyikre nincs hivatkozás, de ő használja a hálózatot. Miről is van szó? Az ilyen lapnak nincs fontossági száma, nem is tud „átadni” fontossági értéket az általa hivatkozott lapoknak, ezért azok fontossága alacsonyabb lesz, mint várnánk. A megoldáshoz *nincs elméletünk*, de számos programtechnikai trükk bevethető, amelyek valamilyen valószínűségi értékeken nyugszanak.

Ilyen például, amikor ezeket a lapokat egy kezdeti fontossági értékkel látjuk el, számítva a rendszerbe való beépülésükre. Aztán a felülvizsgálati ciklusban elnyerik „saját-értékeiket”.

Más a helyzet az olyan lapokkal, amelyek csak „leszívják a hálózatot”, de maguk nem tartalmaznak értéket, vagyis később sem lesz rájuk hivatkozás! Ezekhez tapasztalati értékeket rendelünk, hogy minden keresésük a találat

lapok fontosságát növeljék! Például egy rendezés után a nulla fontossági értékű lapoknak egységszintű értéket adunk, amivel részt vesznek a hálózatban. Minél aktívabbak annál több fontosságot adnak át, tehát elértük a célunkat!

A *3w nem-monogám* hálózat, azaz akárhány kereső rendszer is működhet rajta! A valóság ennél sokkal korlátozottabb, hiszen érvényesül a járt utat járatlanért el ne hagyd elv. Nem könnyű új olyan kereső rendszert bevezetni, amely jobb eredményekkel kecsegtet, mint a már beváltak! *Habár ki tudja, hiszen a mostaniakra is igaz volt ez egykor!*

Összefoglalás

Egy biztos: nem törekedtünk „abszolút jó” megoldásra, talán nincs is, csak kerestünk egy „legendően jó” megoldást. Mindenesetre sikeres disszertáció született!

Megérte? Ebből lehetett milliárdos céget építeni, hát megérte!

A tudományos érdem a kitalálóké, a kidolgozóké, az őket támogatóké. A milliárdok a bevezetőké!

Így van ez jól a mai társadalmi rendszereinkben !?

~~~~~







## Data Acquisition és a PDA.

*MOBI-X, hordozható adatgyűjtő készülék, a személyi digitális segéd – a Personal Digital Assistant - elődje.  
Ma inkább a telefonnal, GPS-el kombinált változatai (iPod, iPad) divatosak.*

*Az adatgyűjtés – data acquisition – egy eljárás, amelynek során a valós világ valamilyen állapotról mintát veszünk és azt feldolgozható formába szerkesztve tároljuk.*

Például egy meteorológiai állomáson elhelyezett hőmérőt, nedvességmérőt, szélmérőt, stb. rendszeres időközönként leolvassák és rögzítik; régebben maga a meteorológus, manapság a kihegyezett számítógép.

Vagy egy másik tipikus példa a pénztáros tevékenysége, amikor a befizetett pénz tételeket egy papíron egymás alá írja, hogy a nap végén az így keletkezett számoszlopot összeadva meghatározza a napi bevételt, amellyel a pénztár tartalmának egyeznie kell.

- Megkülönböztetünk tehát *automatikus és kézi adatgyűjtést*, amelyek korlátait egy adott helyzetben a technikai fejlettség, az eszközök rendelkezésre állása – mennyi pénzünk van rá – határoznak meg.

- Egy másik fontos megkülönböztetés, hogy az *adatok fizikai forrásból, vagy társadalmi tevékenységből származnak*. A fizikai forrású adatokat mindig valamilyen mérőeszközzel mérjük és annak értékét rögzítjük. A társadalmi



források, mint az előbb említett pénztáros, vagy a búzatábla termés hozama, vagy a népesség adatai, a szállítólevelek, fogyasztási mennyiségek, egy verseny pontozási értékei, stb. legtöbbször emberek által leolvasott adatok, amelyeket emberi kezelésű eszközökkel rögzítünk.

Követelmény minden esetben az *adatok helyességének, teljességének ismerete*, amely a mérőeszközök esetében az eszközök pontossága, működésének megbízhatósága. Az ember által végzett adatgyűjtés esetében pszichológiai tényezők kerülnek előtérbe, amelyek nehezen kezelhetők, befolyásolhatóak.

A hibák csökkentésének egyik legnyilvánvalóbb módja, ha *az adatokat a keletkezésükhöz képest térben és időben minél közelebb rögzítjük.*

- A mérőkészülékek esetében ez általában megvalósul, mivel azokat a mérendő objektum közvetlenségében helyezzük el. Problémát ebben az esetben az adatok átvitele jelent.

- A személyi adatrögzítés esetében, azon túlmenően, hogy az embernek el kell mennie az adat keletkezésének forrásához, még ott van a számos emberi tényező, valamint az így feljegyzett adatok szállítása, majd egy későbbi, már a feldolgozáshoz közeli újbóli rögzítése, végre már feldolgozásra alkalmas formába.

Látjuk ennek bonyolultságát, következőképp számos hibaforrását, amely akár az egész feldolgozást értelmetlenné teheti. Nagyon fontos lenne tehát, hogy az adatokat a keletkezési helyükön azonnal feldolgozásra alkalmas formában rögzíthessük, amint azt a mérő eszközök esetében tehetjük.



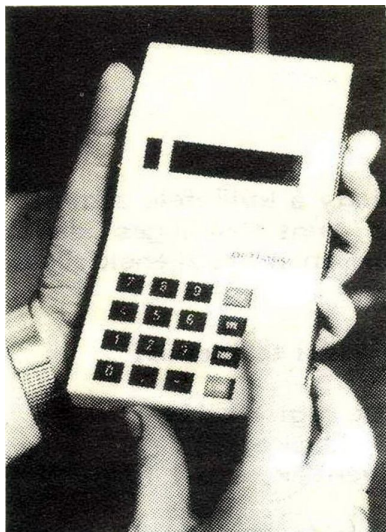
A fejlődő számítástechnika lehetővé tette, hogy ilyen készüléket lehessen készíteni. A hetvenes évek végét, a nyolcvanas évek elejét írjuk, amikor megszületett a személyi számítógép, vagyis az olyan computer, amely már nem közös használatú eszköz, hanem minden személy külön-külön használhat egyet. Ebben a pillanatban feborult a megszokott rend, hiszen mindenki tudott feldolgozni, feltéve, hogy volt ehhez megfelelő programja, írt egyet például. Ez mind rendben működött volna, azonban mit dolgoz fel? *Honnan szerez adatokat, megfelelőt, szükségeset, kellően pontosat, stb.?*

A nagy számítóközpontok fenntartottak önálló hálózatokat, személyeket, intézményeket, amelyek begyűjtötték, adathordozóra rögzítették, ellenőrizték, és végül feldolgozásra alkalmas formába tették az adatokat. A PC-s feldolgozó egységek esetén nem tarthatunk fenn mindegyikhez ilyen apparátust! A közös megoldás sem túl jó, hiszen *azért született a PC, hogy ezek a bonyolult közösségek szűnjenek meg és a tevékenység atomizálódjon*. Felmerült tehát az az igény, hogy az adatokat továbbra is a keletkezési helyükön lehessen beszerezni, ott rögzíteni, tárolni és eljuttatni a feldolgozó PC-hez. Nyilvánvaló követelmény, hogy ezt a PC nagyságrendjében lehessen végezni, vagyis kellene egy olyan készülék és rendszer, amely kisebb, egyszerűbb, mint a PC, de ahhoz csatlakoztatható, hasonló kezelésű és megbízhatóságú.

Ennek a követelménynek *először egy magyar megoldás* felelt meg, amely végül jó magyar



szokás szerint évekkel később, 1984-ben nyert szabadalmi oltalmat.



Időközben Magyarországon, Németországban és Ausztriában már széleskörűen használták és az oltalom csak papír maradt. Meg kell azért jegyezni, hogy a megoldás kitalálói, a kezdeti fejlesztésben, gyártásban részt vettek és ők terjesztették el, azonban jogaikat nem védte semmi a lassú bürokráció

miatt. A bemutatott mintapéldányok és az ötlet alapján külföldön több elektronikai cég kezdett el, másolt operációs rendszerrel gyártani és csak egy-egy cég ismerte el a forrást.

Ilyen körülmények közepette született meg

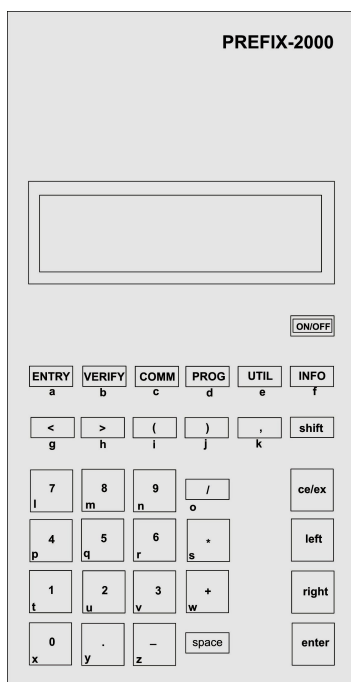
### *a MOBI-X*

A szabadalmi oltalom részesei: *Gyarmati Péter* és *Pék József*. A szabadalom három elemű, magára a „hordozható adatrögzítő készülék”-re, a „cserélhető adattár”-ra, valamint a „vezérlő mobil-operációs rendszer”-re vonatkozott.

A gyártásban, a Budaörsi Mgtsz Elektronikai Szakága, a forgalmazásban a Datakoord, majd a Trigon jeleskedett. Használták többek között mezőgazdasági összeíráshoz az állami gazda-



ságok és termelőszövetkezetek, a MÁV a vagonok ellenőrzéséhez, a Díjbeszedő Vállalat a mérők állásának rögzítéséhez. Módosított



változata már Nyugat-Németországban készült és kapott szabaddalmi oltalmat. A BMW és a VW autógyárak gyártósoraiknál használták. Alkalmazta többek között a DBB német vasútársaság és az ÖBB osztrák vasutak.

Ezek a készülékek már a MOBI-X-et követő MOBI-85, és az újabb igényekhez igazodó továbbfejlesztés,

*a PREFIX-2000*

voltak, németországi szerzői joggal, amelynek birtokosai:

*Ernst Blasits és Peter Gyarmati.*

Ez a készülék szabványos analóg és digitális mérőcsatlakozásokkal, vonalkódolvasási lehetőséggel és nyomtató csatlakozással rendelkezett.

A *cserélhető memória* a gyűjthető adatok mennyiségét növeli, valamint lehetővé teszi az adatok off-line tárolását, szállítását és átvitelét a számítógépbe. Az adatátvitelt a készülékbe beépített soros port is lehetővé tette kábeles, vagy modemes úton. Ezek egyben az adatgyűjtő és



ellenőrző program betöltését is lehetővé tették, vagyis az adatrögzítési előírást tartalmazó programot a számítógépen lehetett elkészíteni. Ehhez u.n. *Cross-Assembler* is készült: „kereszt fordító” program, amely az egyik gépen készült programot egy másik gép nyelvére fordítja.

Magyarországon is közismert utódok, a *PSION* készülékei, az USA-ban a *TIQIT*, amelyet jelenleg is használnak és windows kompatibilis. A modern PDA-k a technika boszorkányos gyorsaságú fejlődése révén már lényegesen több funkciót magukba foglalnak és tenyérnyi PC-nek tekinthetők, az operációs rendszerük is már a nagyobb testvéreik Windows-os változata.

A cserélhető adattár koncepció alapján készült  
*a Personal IC Memory Card*



Például a Mitsubishi BEE CARD-ja, vagy a *font cartridge* nyomtatókhoz ROM, PROM, RAM változatokban, vagy fényképezőgépekben, MP3 lejátszóknak,

stb.. Manapság mindenki a zsebében hordja másik mai változatát a *PenDrive*-ot.

Elmondhatjuk tehát, hogy a PDA őse *magyar, sőt magyarországi fejlesztés*: a MOBI-X és a PREFIX-2000, illetve ezek változatai és „melléktermékei”, a cserélhető adattár, és a mobil operációs rendszer koncepció.

~~~~~


Az informatika néhány nagy tévedése.

Minden ember vágya, hogy dolgai „örökkévalóvá” váljanak. Kevés embernek adatik meg azonban a halhatatlanság, különösen a száguldo technikájú korunkban van ez így.

Néhány éve még úgy hittük, a programozók minden alkotása örök, aztán jött a gyorsabb gép, a grafikus kijelző, stb. és minden addigi munka „elfelejtődött”, újak jöttek, még újabbak, nincs megállás. Voltak, vannak köztük jók, rosszak. A jót használjuk, arra építünk a fejlesztésben, a rosszat kicseréljük és közben persze a jó mellett újabb rosszakat is alkotunk. Álljon itt néhány példa, a teljesség igénye nélkül, az informatika baklövéseiből.

A felsorolás előtt, az író jogán, szeretném a legnagyobb bajra felhívni a figyelmet és ez *az informatika üzletté válása*, a profit érdekeltség megjelenése ezen a területen is, mint annyi máson. Ezt nem az informatika okozta, sokkal inkább a benne rejlő üzleti lehetőségek, amelyet a tőkések felismertek!

Milyen gyorsan elfelejtette a világ Neumann János szándékát és tetteit, hogy a számítógép és technikája legyen közkincs, senki se nyereszkedhessen vele.

A gátlástalan versenynek eszköze *a versenytárs alkotásának lerontása*, amely eredményeként elterjedtek a vírusok és társaik. Több pénzért lehet emiatt informatikához jutni, mint

amennyit az emberek, különösen a diákok, meg tudnak fizetni! Terjednek tehát a szerzői jog védelmét kikerülő megoldások. Ezért sok energia elmegy az ezek elleni védekezésre, ez újabb pénzbe kerül! *Hol a megállás?*

Ha nem lenne, az élet is egyszerűbb lenne, talán sivárabb? Mindenesetre a baj azért inspirál is, hajt a továbblépésre! No és a várható profit!

Nézzünk hát néhány tévedést, „nagyot”, egyetemeset és magyar vonatkozásút is!

1. Y2K, avagy a 2000. év mumusa.

A történet sci-fi-be illő, de nem alaptalanul, mert a gépekbe épített időmérő csak az évszám utolsó két számjegyét tartalmazta, ezért az 2000-ben 00 lesz, vagyis kisebb, mint az előző év 99-e, tehát az így vezérelt új évi adat régibb lesz, mint a tavalyi! Ennyi, ilyen egyszerű, csak át kellett írni minden ilyen programot! Milliókat kerestek egyesek, érdelemmel, érdeemtelenül?

Azonban semmi sem történt, nem jött a mumus, nem esett szét a világ, a bankok sem nullázták a számlákat.

De, vigyázzunk! Az új hardware-számlálók csak 2035-ig vannak felkészítve, az utódaink is kaszálhatnak egy nagyot! És még a mumus is jöhet!

2. MP3

Tetszenek tudni, a zene(le)játsszók szabványa. A bevezetése hatalmas szerzői jogi botrányt kavart, amelyből szerencséjére, a Linux világ ki-maradt. Az MP3 a zenei konverter programok egész sorának létrejöttét okozta, amely a

használókat már-már az örületbe kergeti. No és sokaknak ügyes üzleti lehetőség, csak győzzük fizetni, vagy „trükközni”. Ma már minden technikai „cucc” tartalmazza a jogot, vagyis jogdíjat fizetünk érte, akár használjuk, akár nem! Ráadásul az MP3-nál sokkal jobb tárolási formák vannak, szerzői jogi zöngék és jogdíj nélkül is!

3. Vista

Mindenki tudja, a Microsoft újabb „távlati”, operációs rendszere. Ez a valaha készített legrosszabb operációs rendszerek bajnoka! Mégis, tessék elhinni, a Microsoft a többszörösét kereste annak, mint amit belefektetett! Hja kérem, ez a divat mítosza! Egyesek mindig megve-szik a legújabbat, függetlenül minden más szemponttól! Hogy lassúbb, hibásabb, érthetetlenebb, inkompatibilisebb, ez mind nem számít! „Végre” itt van már az öszvér (XP+Vista), a Windows 7, amely egyesek szerint – már megint az a fránya üzlet – a valaha legjobb, a többség azonban tüntet az XP megmaradása mellett. De hát ez a Microsoft végét jelentené!

4. WordPerfect

Néhányan még bizton emlékeznek erre az egykor kiváló szövegszerkesztőre, amelyet - az egyébként kiváló rajzolóprogramot forgalmazó - Corel volt olyan szíves felvásárolni és hagyni, hogy a Microsoft az Office-al leradírozza! Talán, mert a szintén kiváló táblázatkezelővel (Framework), másik tulajdonos miatt, nem egyesülhetett? Vajha nem rontják el az üzletet, hol lenne

ma a Microsoft Office? Na, és perfekt lenne a Perfect?

5. IPv6

Tetszenek tudni az Internet Protokol tartalmazza és kezeli az IP címeket. Ezt valósítja meg az IPv4 szabvány! Egyszer csak – üzlet(?) – felröppent a hír: elfogynak a hálózati címek és akkor mi lesz! Nosza azonnal meg is jelent az új szabvány, az IPv6, mely majd ezt megoldja, csak mindenki vegye meg a „cuccot” és térjen át, tegye ezt, meg ezt. Tehát csak az a szerver lesz jó, amelyik „támogatja”, így mondják, az IPv6-ot. Mára már a szerverek is jók erre, mindenkinél ott az új szerver-OS, mely tartalmazza, de nem kell! Nem kell, mert NINCS cím elfogyási probléma! Van elég! Megint egy Y2K ügy?

Használjuk, vagy ne? Lassúbb, fölösleges, hibás is lehet, bonyolultabb, szóval?

6. CD, azaz kompakt lemez-lemez.

Azért írtam így, mert így nevezitek, „cédé-lemez” nem? Mondhatnátok, hogy C-lemez, vagy csak egyszerűen CD, de nem, nektek ez olyan, mint a tesztvizsga!

A szépreményű alkotóinak sikerült egy olyan adattárolót kitalálniuk, amely a legjobban kedvez a szerzői jogdíjat élvezőknek, még az ős gramofonlemezen is „alulatesz” rossz emlékezőtehetségével. Veszünk egyet, megírjuk, talán párszor olvasni is tudjuk és máris dobhatjuk el, mert „elhomályosodik”, mint a rossz a tükör. Még addig sem jó, mint az a „vacak” mágnesszalag! Merthogy abban van a mérnöki hókuszpókusz, hogy a lemez felét javítókéddal töltik meg és hibatűró algoritmust használnak a meg-

bízható technika helyett. Ezért a működése során tömeges olvasási hibát produkál, amelyet kiszűr ugyan az algoritmus, ameddig több hiba nem születik. Az viszont gyorsabban következik be, mint az eddig kitalált összes adattároló esetében! A DVD sem különb, csak azt már nem nagyon használjuk adattárolásra, pusztán szórakoztató izékre, amelyeknél egyenesen jó, ha mielőbb tönkre jut (ez magánvéleményem). Okos informatikus szilárdtest tárolót használ (PenDrive és társai).

7. Mesh network, avagy mindenki hálózata.

A vezeték nélküli hálózat (WiFi) egy bizonyos elterjedtségen túl lefedi az egész Földet és ezáltal mindenki részesévé válik a „mesh network”-nek. Így szólt a terv, a jóslat. De nem ez történt. Pedig igazi nagy ötletnek látszott, piszkálták is a hálózati kiszolgáltatókat és azok használóit, hogy alakítsák át a saját hálózataikat erre, hogy „kiterjesszék” a net-et (minden gép a mesh-hálózat egy-egy eleme, amely részt vesz a kapcsolat-tartásban).

Nagy ötlet volt, grandiózus álom, de a nép, az internet közössége nem valósította meg, egyszerűen saját biztonsága érdekében, óvatosságból! Még, hogy nyilvánossá váljék a vállalat belső rendszere, mindenki szabadon nyúlkaíjon a szomszéd irodából, az utcáról az adatbázisunkban, akár a versenytárs, vagy az adóhatóság is? A telefontársaságok is minden igyekezetükkel ezen voltak, mert a mobil olyan, mint a wifi(?!). No de, vajon ki tartaná állandóan bekapcsolva a mobilját és költene energiára és

mire még ennek érdekében! Megjegyzem, a saját rendszerüket a telefonosok sem „wifizik”!

8. ESZR, (nem) Egységes Számítógép Rendszer. Soha sem volt egységes, csak a neve, vagy még az sem! Volt benne „saját” alkotás és sok nyugati utánczat, másolat, több, kevesebb sikerrel. Valójában sikerült a szocialista gazdaságot eléggé „információtlanítani” az ESZR segítségével és az ezáltal ki is múlt. A gazdasággal a politikai rendszer is és így az ESZR is. Még annyi sem maradt belőle, mint az a bizonyos szobor-park.

9. A magyar helyzet.

Sajnos kiderült, hogy a magyar, információ-technika is kimúlt az ESZR-el és ami maradt az „elkelt” a rendszerizelő privatizáláson. Hol van például a KFKI, a VIDEOTON, az EMG, a TERTA és társaik? Pedig volt, van abból a korból nem is kevés magyar eredmény, amelyre épülhetett volna egy eredményes jelen! Minden eredmény külföldre „szökött”!?

Ma Magyarországon az informatikát külföldi eszközökön gyakorolják, az evvel foglalkozó szakemberek és alkotásaik, ha egyáltalán van ilyen, a külföldieket táplálják. A külföldön tevékenykedő néhány magyar számottevőbb eredményt ér el, mint itthon mindösszesen! Kár, nem?

~~~~~



## Utószó

Ha valaki könyvírásra adja a fejét, akkor azt hiszi, hogy csupa olyanokat fog írni, amelyeket habzsol az olvasó. Vajon van-e, egyáltalán lesz-e olvasója a könyvemnek? A kérdés akkor merül fel, amikor elkészül az elképzelés szerinti tartalom, vagyis későn ahhoz, hogy megakadályozzon a kiadásban. Tehát az írás elindul a nyomtatás és terjesztés kalandos útjára: *habent sua fata libelli*.

A számítástudomány és számítástechnika merőben hasonlít a mérnöki tevékenységekhez hiszen bizonyos szabályok törvényszerűségek megtartásával létre kell hozni, alkotni kell valamiféle eljárást, programot, amely a kitűzött feladatot akárhányszor elvégzi. Okosan és gépiesen és ismételten is helyesen, ugyanúgy!

Ésszerűnek tartom tehát a mérnöki módszerek alkalmazását itt is! Vezessük be *a gépelemek mintájára az „informatikai elemek” fogalmát*.

Ezek olyan módszerek, megoldások, amelyek minden számítógépnek része kell, hogy legyen, máskülönben a gép nem működik, nélkülük a gép nem gép. Ezek nem teljesen azonosak a gépiparban alkalmazott szabványokkal, hiszen többnyire lényegesen bonyolultabbak, ezért megfogalmazásuk sem lehet egy „sima, mezei program”. Tehetjük viszont, hogy a fontosabb, most már használjuk a szót, *elemeket* a maguk



teljességében összefoglaljuk. Az informatika kialakulása és fejlődése során számos elemet írtunk le. A hálózatokat, az internetet például az u.n. RFC-kben (Request For Comments).

Számos tudományos dolgozat hozta létre az egykori time-sharing rendszereket, amelyek elemei nélkülözhetetlen részei a ma PC-jének. Nem létezhet számítógép interrupt, verem-automata, szemafor, asszociációs tár, erőforrás-kezelés, driverek, hibajavító kódok, jelölési rendszerek nélkül, hogy csak néhányat említsek. Azután a proceszor IC-k megszületésével, többek között a mikroprogram alkotta utasítás-készlet, az egységesség, a kompatibilitás jegyében, a programnyelvek értelmezését lehetővé tevő formális nyelvek, vezérlő protokollok, grafikus leképezések és megjelenítők mind-mind elmaradhatatlan elemek!

Ezen írásomban néhány ilyen módszer, eszköz leírását adom. Olyanok találhatók itt, amelyeknek kialakításában valamilyen módon részt vettem, részese lehettem.

*Ez a gyűjtemény egy kezdeményezés, amely remélem jó példa lesz a folytatásra, míg nem összegyűlik a legtöbb elem. Talán mások már elindítottak ilyen összeállítást, például számos munka foglalkozik az algoritmusok elméletével. Ajánlom ezt a munkámat csatlakozásként azokhoz.*

Fontosnak tartom ezek összegyűjtését, hiszen a legtöbb esetben az új informatikus generációk csak összefoglaló fogalmakkal találkoznak, mint például Kernel, vagy GUI, stb. Ezeket a valódi



tartalmuk, értelmük szerint már nem is ismerik, csak mint egy fekete dobozt! Azonban egy sáv-szélesség változtatáshoz, vagy cache méret meghatározáshoz, új eszköz létrehozásához szükség lehet az érintett elem teljes matematikájára. Hát ezek az elemek!

Remélem mások is tollat ragadnak, hogy mihamarabb a legtöbb elem gyűjteménybe kerüljön! Talán nekem is lesz még erőm újabb elemek közreadására. Talán lesz egyszer valaki, aki felsorolja az informatika teljes elemkészletét, ha ez egyáltalán lehetséges!

Az egyes fejezetek függelékeket is tartalmaznak a részletekben elmélyedni kívánók számára, amelyekhez kiegészítésül felsorolok itt néhány internetes helyet.

Remélem ez az írás nem jut nagyon sok más írás sorsára: *Graeca sunt non leguntur* (görögül van, nem olvassuk).

Sokkal inkább, mivel magyarul és angolul van, - minden informatikával foglalkozó számára olvasható - *TOLLE, LEGE (vigye és olvassa)!*

~~~~~


Végezetül még néhány internetes címet ajánlok szíves felhasználásra:

www.gyarmati.dr.hu

www.szalon.tk

www.ams.org/meetings/erdos-lect.html

www.siam.org/activity/

www.sul.stanford.edu/

www.acm.org

www.nrich.maths.org

www.thefreedictionary.com

www.scientific-computing.com/news

www.corvinuslibrary.com/idoszeru/katalog.htm.

Karl Steinbuch:

<http://citeseer.ist.psu.edu/cache/papers/cs2/334/http:zSzzSzhelios.informatik.uni-kl.dezSzeuology.pdf/unknown.pdf>

Wikipédia:

[http://en.wikipedia.org/wiki/Informatics_\(academic_field\)](http://en.wikipedia.org/wiki/Informatics_(academic_field))

http://en.wikipedia.org/wiki/Computer_science

http://en.wikipedia.org/wiki/Peter_G._Gyarmati

Számítástudományi tanulmányok:

<http://www.lecturefox.com/computerscience>

http://videlectures.net/Top/Computer_Science

http://www.dmoz.org/Computers/Computer_Science//

A számítástudomány oktatásáról:

<http://portal.acm.org/citation.cfm?id=971303&dl=ACM&coll=#opub>

Informatikai definíciók:

<http://www.dai.ed.ac.uk/homes/cam/informatics.shtml>

<http://www.inf.ed.ac.uk/publications/online/0139.pdf>

<http://www.rae.ac.uk/pubs/2006/01/docs/f23.pdf>

~~~~~



Ettől a szerzőtől nemrégiben megjelent könyvet ajánlunk,  
**NYERESÉGES VÁLSÁG?**  
címmel, amelynek rövidesen megjelenik második kiadása.

GYARMATI PÉTER

NYERESÉGES VÁLSÁG?



TCC COMPUTER STUDIO

A tartalomból:

- Bölcs Ezopusnak oktató beszédi, LXXXI.
- Rettegek...
- A XXI. század küszöbén.
- Beszélgetés a miniszterelnökkel.
- Magyarország ma, az antalli-örökség.
- Nem lehet Magyarország független?
- Válság és valóság 2008.
- Pénzügyletek, derivatives.
- K. Lorenz víziója (recenzió).
- Intermezzo.
- Szavak.
- A számítógépről.
- Bölceletek.
- Versek.
- Bölcs Ezopusnak oktató beszédi, XCVIII.

Vélemények a könyvről:

*... Ha vannak vesztesek, akkor nyerteseknek is kell lenniük! Miért ne éppen Ön lenne az? Talán a receptet tartja most a kezében! Hiszi? Nem? Fontolja meg, gondolja meg! Olvassa!*

J. G. OLAH, Stanford, USA

*... A nemzeti felemelkedéshez... rámutat néhány területre, ahol a kitörés lehetséges, ahol cselekedetre van szükség és lehetőség van, mint például az árvozek hasznosítása.*

*... arra a megállapításra jut, hogy „találjuk ki Magyarországot”, sürgősen és áthatóan.*

*Könyvét saját verssel zárja...*

LOSONCI MIKLÓS, művészettörténész, Szentendre



*... fontos olvasmányként ajánlom mindazok számára, akik meg szeretnék érteni a válság lényegét, a folyamatokat, néhány fontos tudományos hátterét.*

**Úgy vélem, ezt a könyvet minden egyetemistának a kezébe kellene nyomni.**

MAGYARÓDY SZABOLCS, a HOLLÓ szerkesztője,  
Kanada, Hamilton

~~~~~

Terveink között szerepel a
SZENTENDRE SZALON
közérdeklődésre számító előadásaiából készülő

SZALON ESTÉK

sorozat kiadása

~~~~~

Könyveink megvásárolhatóak

- a LIRA könyvesbolt-hálózatban;
- a Kráter Kiadó könyvesboltjaiban;
- a kiadónál : [www.tcc66@hotmail.com](mailto:tcc66@hotmail.com) utánvéttel és postaköltség(250-1200Ft);
- ugyanott előreutalással, postaköltséggel együtt,
- Magyarországon: MKB 10300002-10330962-49010013,
- külföldre: MKB 10300002-10330962-48810030;
- a nyomda üzletében: Óbudán, Bécsi út 89-93, CORBIS DIGITAL;
- Szentendre könyvárusainál.



Kiadja a TCC COMPUTER STÚDIÓ, az 1988-ban alapított TCC-GROUP tagja, 2000 Szentendre, Pannónia u. 11., tel: +36-26-314-590, e-mail: [tcc66@hotmail.com](mailto:tcc66@hotmail.com). A kiadásért felel a Stúdió igazgatója, felelős szerkesztő Terjék Aranka.

Tervek, fedélterv, művészeti szerkesztés a szerző munkája.

Nyomta és kötötte a CORBIS DIGITAL – digitális és offszet nyomtatás, 1033 Budapest, Szentendrei út 89-93. PP Center, III.

épület (rózsaszín) , I.emelet tel: +36-1-999-6593, e-mail: [corbiskft@gmail.com](mailto:corbiskft@gmail.com), [www.corbis-digital.hu](http://www.corbis-digital.hu), nyomtatásért felel a nyomda igazgatója.

Megjelent 13,25 (A/5) ív terjedelemben.

ISBN 978-963-06-8599-3



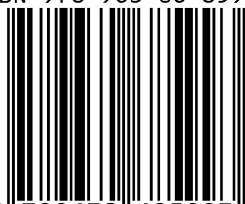


GYARMATI PÉTER (1941) villamosmérnök és matematikus, a BME Budapest, az ELTE TTK Budapest egykori növendéke, a brit Manchester University és a kaliforniai Stanford University professzora. Ez az összeállítás néhány korábbi, de most is időszerű dolgozatát ötvözi egybe az utóbbi években született előadásaival. Mint nyilatkozataiban visszatérően halljuk, hogy szerinte az „idősödő tudósok” kötelezettsége a tudományuk „közművelésre” bocsátása, törekvés a szerzett tudás és tapasztalat továbbadására minél szélesebb körben.

Sajnos manapság egyre kevesebb az olvasásra fordított idő, ezért kiemelt fontosságúvá vált az olyan fórumok szervezése, ahol előadások, viták formájában „közművelni” lehet. Ezek sorában említhetjük a szerző közreműködésével létrejött SZENTENDRE SZALON-t is.

Ez a gyűjtemény a szerző számítástudományi tevékenységéből és kapcsolódó előadásaiból készült, amelyek egy része angol nyelven íródott. Elsősorban számítástechnika iránt érdeklődők részére ajánljuk.

ISBN 978-963-06-8599-3



9 789630 685993