

Operating systems

ELTE IK.

Dr. Illés Zoltán

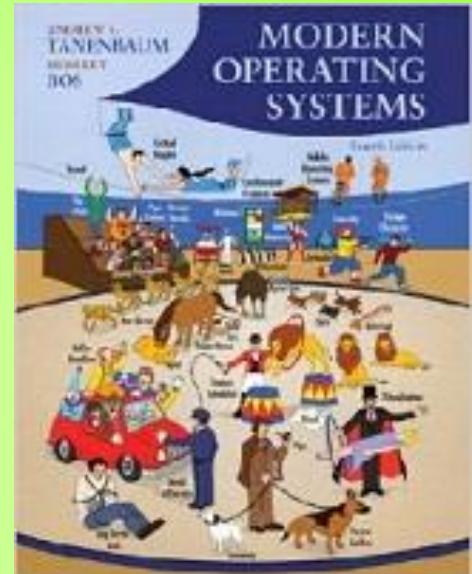
zoltan.illes@elte.hu

Base Information

- ▶ The „Operating systems” homepage is:
<http://oprendszer.inf.elte.hu>
- ▶ Schedule of subject: 2+1(+1)
- ▶ Practice: 45 min/ week or 90 min/2 weeks
- ▶ The main result of the subject: one (final) mark
 - Both of lectures and practices
- ▶ Requirements of grade:
 - 2 homework
 - 2 exams(one theoretical,one practical).
 - 1. exam time: appr. 9.week (lectures)
 - 2. ZH időpontja: last but one week (practical)

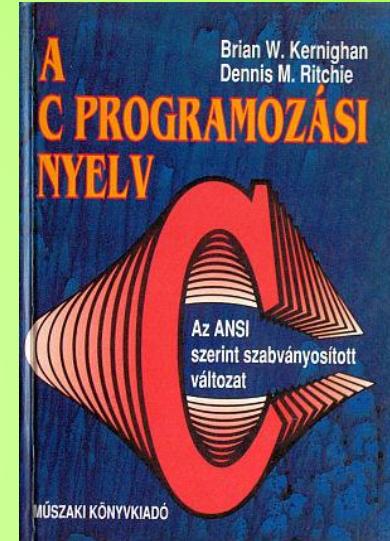
Literature I.

- ▶ Andrew S. Tanenbaum,
Herbert Boss: Modern
Operating systems.
4. edition, 2014.



Literature II.

- ▶ Brian W.Kernighan, Dennis M. Ritchie:
The C programming language
- ▶ Brian W.Kernighan, Rob Pike: Unix operating systems



Some of the illustrations are based on the figures of the above given literature.

The goal of subject

- ▶ Make more clear the role of operating system.
- ▶ Review questions, base problems, solutions during a planned, examined operating system.
- ▶ Choosing the right, appropriate operating system on demand.
- ▶ Knowledge of the operating system core base API functions.

Prerequisites

- ▶ Fundaments of computers
 - Unix shell script programming
- ▶ Fundaments of Programming
 - C, C++ language, base algorithms.
 - C#
- ▶ Programming languages I. (C++)
 - ...

Preview of chapters

- ▶ Introduction, reviews (Foundation of computers, architecture of computers, API)
- ▶ Operating system definitions, it's evolution, user interfaces
- ▶ Files, file systems, directories, disk management
- ▶ Processes, scheduling of processes
- ▶ Input–output, resources, deadlock handling
- ▶ Memory management
- ▶ Real–Time Operating system features

Preview of practices

- ▶ Setup of your workspace.
 - We use the oprendszerek.inf.elte.hu server!
 - Your login id is: Your Neptun code
 - Password... passwd command.
 - Using a text editor: vi, mcedit, joe
 - Local text editing
 - Moving local file to server via ssh based FTP.
 - Using the winscp editor.
 - Modifying PATH environment variable.

Compiling on Server

- ▶ The server has (SLES 11 RTE SP1) GNU C, C++ compiler.
 - /usr/bin/gcc-4.3 Default C, C++ compiler, at .c C, at .cpp C++ mode.
 - cc, gcc – easier usage
 - Man gcc
 - Compiling: cc apple.c
 - Result: a.out
 - Compiling: cc -o apple -Wall apple.c
 - Result: apple, and shows all warnings
 - /usr/bin/g++-4.3 Default C++ compiler
 - it may be used as gpp

**Thanks for your
attention!**

zoltan.illes@elte.hu

Operating systems

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

We have talked about...

- ▶ Evolution of operating systems
- ▶ Notions of Operating system, structures
- ▶ Files,directories, filesystems
- ▶ Processes
- ▶ Classical IPC problems
- ▶ Scheduling of processes
- ▶ I/O, deadlock problem
- ▶ Memory management
- ▶ Real-time systems

What comes today...

- ▶ Modern operating system environments.
- ▶ Which typical solutions are there in a modern operation system?
 - Linux (Android)
 - Windows

Unix-Linux world – keywords

- ▶ Bell Labs, Multics descendent system, at the beginning of the 1970th, UNIX, PDP7,PDP11, Ken Thompson, Brian Kernigham,Dennis Ritchie
- ▶ Portable Unix (C language), Berkeley Unix(BSD)
- ▶ Two different tendencies, BSD, System-V (SVID, AT&T), at the end of the 80th
- ▶ Standardization, IEEE 1003.1 (ANSI, ISO standard), 1988.
- ▶ LINUX– Linus Torwalds, v0.01–1991, v1.0–1994
 - Lots of distributions (SUSE,Red Hat,Debian,etc.)
 - www.distrowatch.com

Linux features

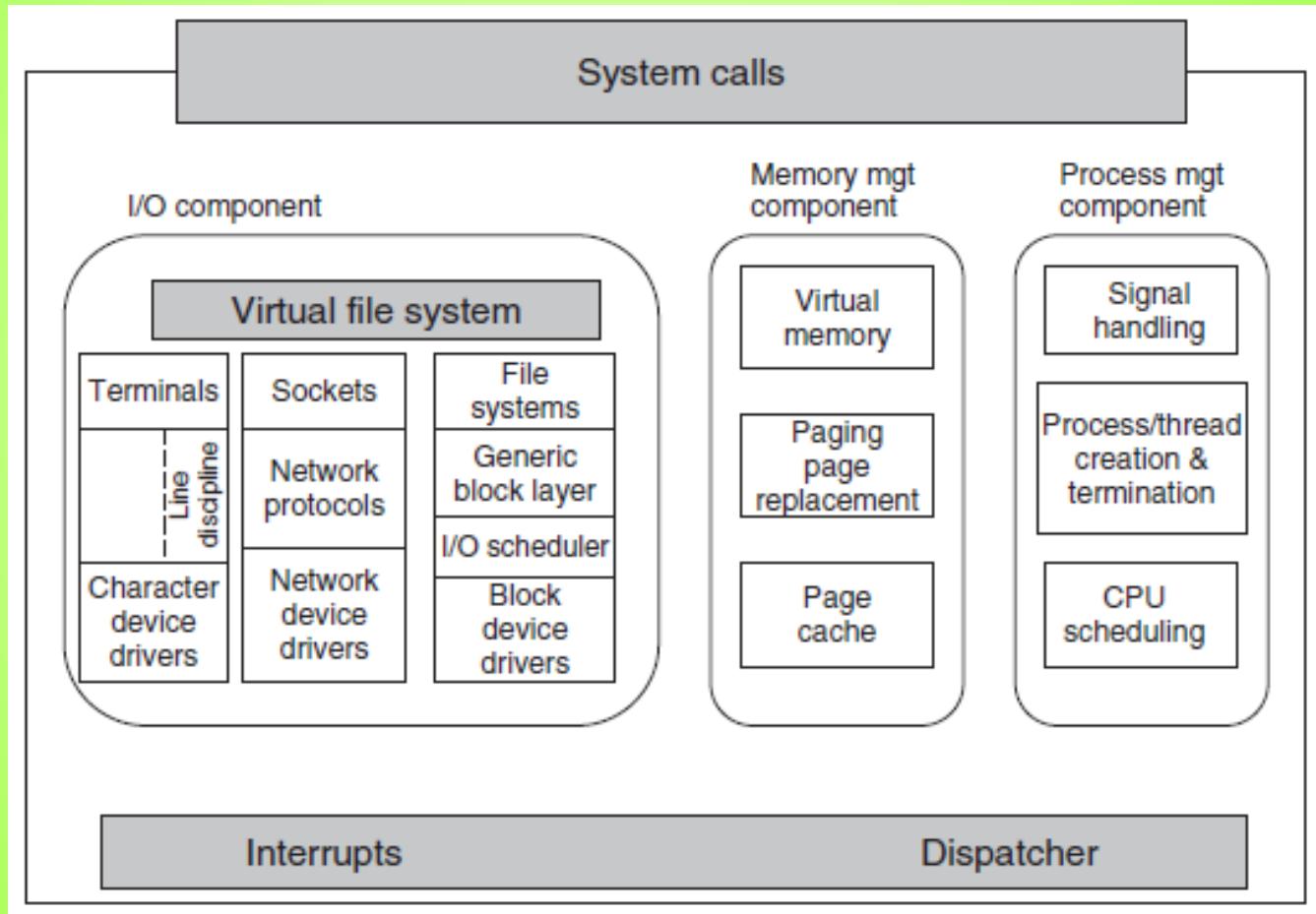
- ▶ Classical terminal mode+X-Windowing System (X11)
- ▶ System version: uname -a

```
illes@oprendszer:~> uname -a
Linux oprendszer 2.6.33.7-rt29-0.5-rt #1 SMP PREEMPT RT 2010-08-
25 19:40:23 +0200 x86_64 x86_64 x86_64 GNU/Linux
illes@oprendszer:~>
```

- ▶ Classical layered structure
 - 1.Hardware layer
 - 2.Kernel (Process, memory,I/O, file system) layer
 - 3.Standard library (open,close, fprintf, fork, exec...)
 - 4.Standard programs (shell, gcc, etc..)

Kernel structure

► User's applications



Hardware

Processes

- ▶ Processes – background processes (daemons)
- ▶ Process tree (pstree)– PID, root element is init, PID=1

```
illes@oprendszer:~> ps -ef|grep init
root      1  0  Jan16 ?        00:00:04 init [5]
root    3558  1  0  Jan16 ?        00:00:09 /usr/sbin/sshd -o
PidFile=/var/run/sshd.init.pid
root    3841  1  0  Jan16 ?        00:00:00 /usr/sbin/xinetd -
pidfile /var/run/xinetd.init.pid
illes   13317 12666  0 09:36 pts/0    00:00:00 grep init
illes@oprendszer:~>
```

- ▶ Message passing (pipes, signals,...)

Processes, threads

- ▶ A Linux process often calls as task!
 - A task_struct is used to represent any execution context!
 - Main fields: scheduling parameters, registers, signal masks, file descriptor table, memory image, etc.
- ▶ POSIX.1 defines pthreads kernel threads
 - See: man pthreads
- ▶ Clone – system call, creating a thread in the original process, or in a new process
 - The original process can share some of it's data, like address space, file descriptors, signal handling, file system parameters, etc

Scheduling in Linuxban I.

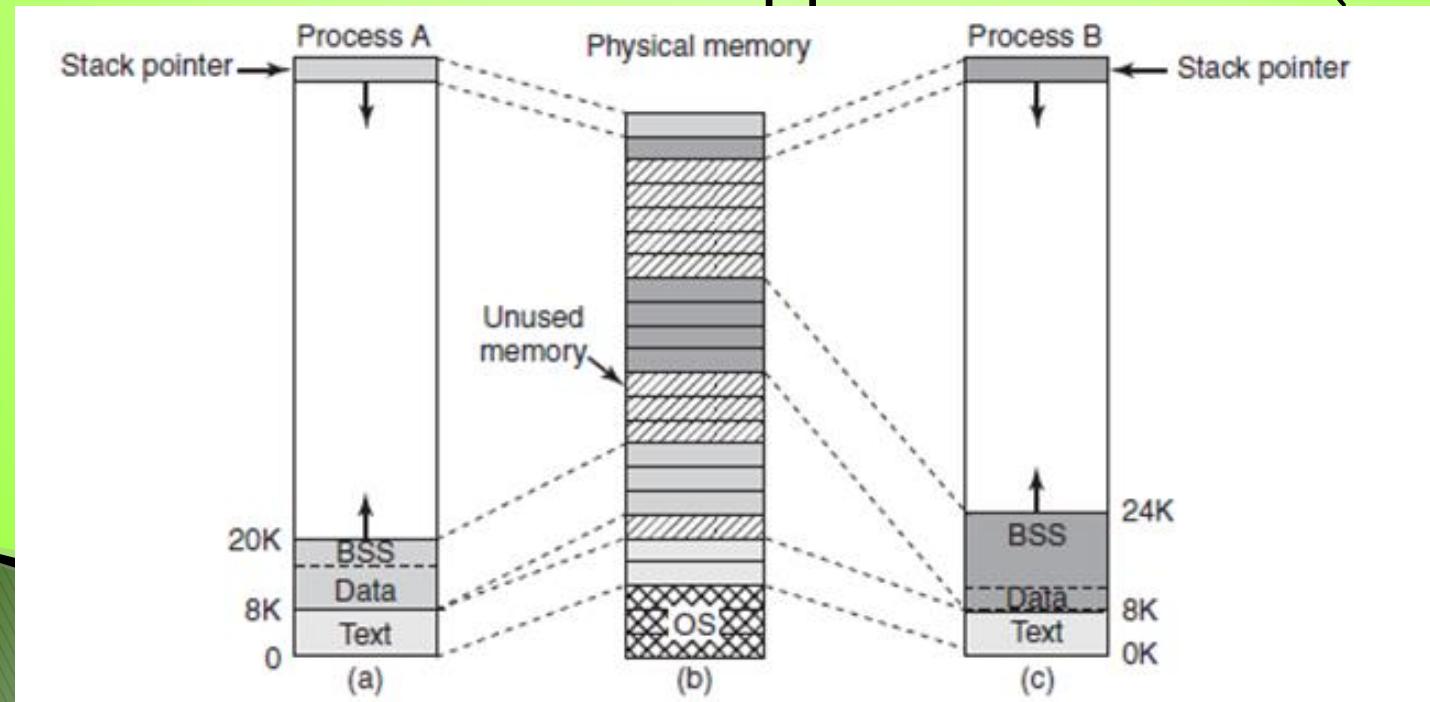
- ▶ After POSIX1003.4 standard (real-time extensions to UNIX) to the 2000th two different type of scheduling support appeared!
 - Preemptive time-sharing, priority scheduling (classical, 0–39 priority class interval from -20 till +19, 0 is the default value, nice instruction, -20 is the biggest one!)
 - Real-time scheduling (0–99 priority classes, 0 is the biggest one, 99 is the smallest priority class, but each of them has got bigger priority than the classical 0–39 classes)

Scheduling in Linuxban II.

- ▶ Static (original, base), dynamic (permanently changing) priority
 - Modification of priority (boost priority)
- ▶ Classical timesharing scheduling:
 - CFS (Completely Fair Scheduling, Ingó Molnár, Red–Black tree, virtual time slice)
- ▶ Real–Time scheduling
 - There is a RUNQUEUE for each priority class!
 - SCHED_RR (Round–Robin)
 - SCHED_FIFO
- ▶ Chrt command

Memory management

- ▶ A process virtual address space contains : code (text), data and stack block
- ▶ In the example A and B process shares his code block because A and B same application (e.g:vi), processes can use a mapped file as well!(mmap)



Linux Memory Management

- ▶ 32 bit system: 3GB virtual memory size, 1GB kernel data
- ▶ 64 bit system: used only 48 bit,(256TB, virtual memory space, 128TB for process, 128TB for kernel data)
- ▶ Memory allocation:
 - 1. Buddy algorithm (Dividing in half of memory while the size is uncorrect!) Memory pages: 2^n
 - 2. If smallest one is too large, take a smaller unit and manage it!
 - 3. Use vmalloc!

Page replacement

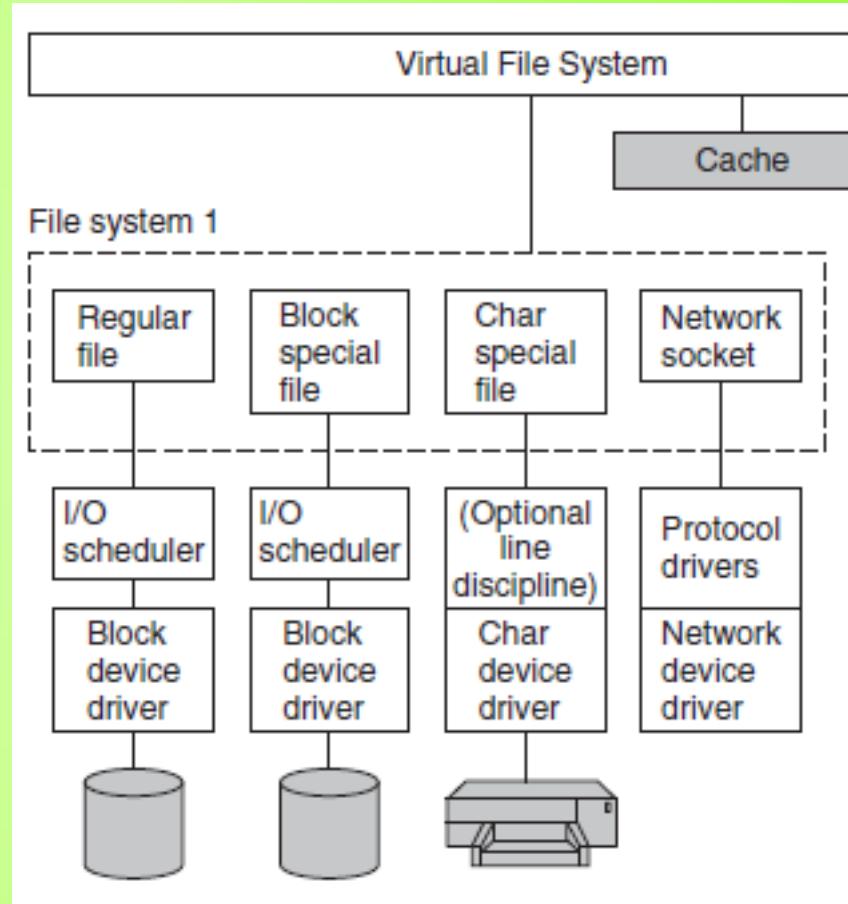
- ▶ Page size: 4 kb (in 64 bit system 2MB page system also supported)
- ▶ Process ID 2: page daemon, the swapper process
- ▶ Page Frame Reclaiming Algorithm (kswapd)
 - 4 types of pages: unreclaimable(pinned), swappable (must be written back before reclaiming), syncable (must be written back if dirty bit is set), discardable (can be reclaimed immediately)
 - PG_active, PG_referenced bits
 - Clock-like algorithm, from the easiest point: first take the discardable, unreferenced pages
 - Active, inactive page list

Input/Output in Linux

- ▶ Character special files (terminal connections)
- ▶ Block device files (file systems)
- ▶ Networking
 - Berkeley design „socket”
 - If a socket is created we can choose for transmissions a reliable byte stream protocoll (TCP)
 - Or for an unreliable packet-oriented transmission we can choose UDP!
 - Once a connection is ready between source and destination processes, it functions analog to a pipe!

Linux I/O architecture

- ▶ To reduce repetitive disk-head movements, there is used block I/O scheduler
- ▶ It is similar as „SSTF” schedules (File systems)
 - It can lead to starvation!
 - 2 lists,
 - 1 is ordered by address of sector of disk request
 - 2 is the deadline list (anti starvation)



Linux file systems

- ▶ Typical Unix style file system based on index tables (i-node tables)
- ▶ Shared lock, exclusive lock for an entire file, or for a part of file!
- ▶ Virtual File System(VFS) has 4 object
 - Superblock, dentry(directory), i-node, file(in memory file representation)
- ▶ Most popular Linux file system: Ext2FS
- ▶ Dentry cache- for quick search
- ▶ File descriptor table does not directly map to a file(i-node)
 - A particular file descriptor corresponds to an open file descriptor table element-contains the file position, R/W mode,i-node

Ext3FS, Ext4FS

- ▶ However the most general FS is Ext2FS, typically that successor (Ext3FS) is used!
 - Check your FS: df -T

Filesystem	Type	1K-blocks	Used	Available	Use%	Mounted on
/dev/sda2	ext3	101139356	22144884	73856888	24%	/
devtmpfs	devtmpfs	8028468	92	8028376	1%	/dev
tmpfs	tmpfs	8028468	88	8028380	1%	/dev/shm

- ▶ Ext3FS is a journaling file system.
 - Basic idea: maintain a „journal”(log), which describes all operations in sequential order!
 - Benefit: At any unexpected case the system recognise it and apply the file system changes based on journal log!

Ext4FS – supporting extents, gives faster support at larger file and file system size!

Security in Linux

- ▶ Same as in a general UNIX
 - Base security is a 3x3 rwx control!
 - Additional: SETUID,SETGID and STICKY BIT.
- ▶ Particural rights: setfacl, getfacl
- ▶ User ID stored in /etc/passwd file
 - A user's password stored in /etc/shadow file in hashed format!
 - Manually to check a password:
 - 1. Enter the pw as a text.
 - 2. Get the user's shadow entry.
 - 3. Getting the „salt”
 - 4. Crypt the pw.
 - 5. Compare it with the existing shadow one!

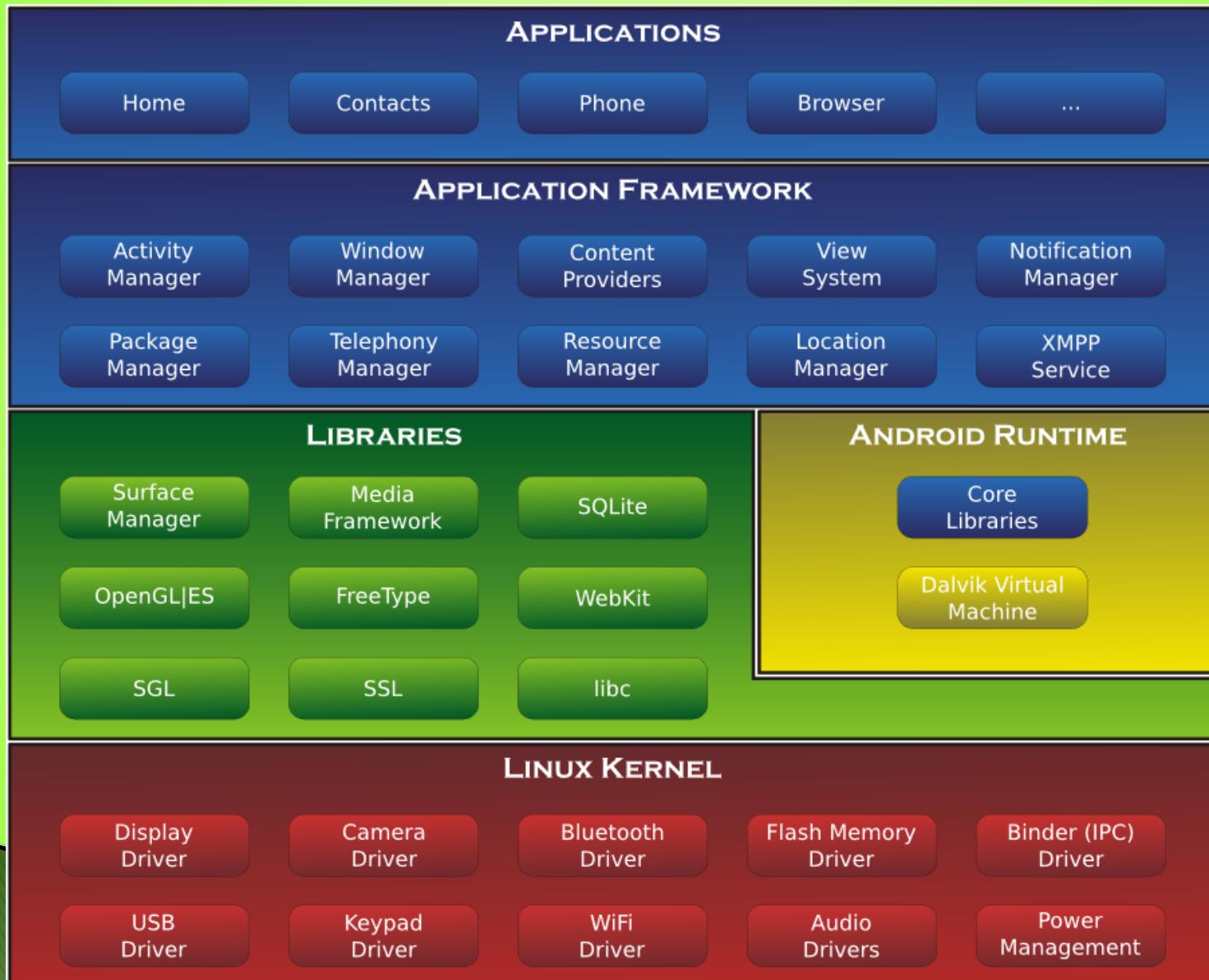
Android

- ▶ New operating system, designed to run on mobile devices.
- ▶ Kernel and most of low level libraries written in C, C++.
- ▶ Generally all other part is developed in Dalvik Java. (Quicker VM, more efficient memory usage)
- ▶ In 2005 Google acquired Android Inc.
- ▶ Today the most popular mobile (phone, tablet, TV, etc.) system.

Android extensions

- ▶ Wake lock – manage how the system goes to sleep
- ▶ Out-of-Memory Killer – replaces the traditional Linux one, it is more aggressive
- ▶ Binder IPC – transaction based RPC
- ▶ Android application – package, .apk, contains not only code, but resources, manifest, etc.
- ▶ Application activity – application part which interacts with user via UI(event-handler)
- ▶ Application Sandboxes – a new UID for an app according security reason.
- ▶ Process model – every process is created and managed by Dalvik's zygote module

General Android architecture



Android today

- ▶ Ver.3 is for tablets!

Version	Name	Date	API level	Distribution
6.0	Marshmallow	5th October 2015	23	0.3%
5.1-5.1.1	Lollipop	9th March 2015	22	10.1%
5.0.0-5.0.2		3rd November 2014	21	15.5%
4.4	KitKat	31th October 2013	19	37.8%
4.3	Jelly Bean	24th July 2013	18	4.1%
4.2.x		13rd November 2012	17	13.9%
4.1.x		9th July 2012	16	11.0%
4.0.3-4.0.4	Ice Cream Sandwich	16th December 2011	15	3.3%
2.3.3-2.3.7	Gingerbread	9th February 2011	10	3.8%
2.2	Froyo	20th May 2010	8	0.2%

Android's future

- ▶ However Android is a general full purpose operating system, but it is running mostly on a mobile device.
- ▶ Android – general computer , maybe soon.
- ▶ ...
- ▶ On other hand, there are other Linux successor mobil distribution too: e.g:
Mobile Ubuntu
- ▶ ...

Windows world

- ▶ MS-DOS 1.0 – 1981
- ▶ Windows 3.0,3.1 – 1990,1992
- ▶ Windows 95, NT – 1995
 - There are two tends: MS-DOS based (client) and NT based (server)!
- ▶ MS-DOS based (client) systems:
 - Windows 98, Windows ME – 1998, 2000
- ▶ NT based (client, server) systems:
 - Windows 2000 (client, server)
 - Windows XP (client)-2002, Windows 2003(server)
 - Windows Vista, 7(2006,2009), Windows server 2008
 - Windows 8,8.1, 10(2012–2015),Win2012 server

Windows programming elements

- ▶ MinWin approach, from win8.1, win10
 - Same the most of the core, binaries for all version including WP!
- ▶ Win8.1 removes POSIX support!
- ▶ New WinRT (Runtime) API (replaces Win32)
- ▶ NT namespace holds OS names, objects (e.g. device objects) created during boot, stored in kernel's virtual memory.
 - This part is marked as permanent!
- ▶ API features
 - Unicode, WoW(Windows on Windows) functions, ACL, journaled NTFS,I/O subsystem manage, GUI functions, etc.

Windows registry

- ▶ During boot the NT namespace is created.
 - How it is created? Where are the configuration parameters stored?
- ▶ The registry holds these information!
 - Registry files (hives) stored in directory Windows\system32\config
 - System – HKLM\System
 - SAM – HKLM\SAM (Security Access Manager)
 - Etc.
 - In earlier Windows versions this parameters was stored in .ini files, same as UNIC config files!
 - To explore registry we can use API calls or regedit.exe, or PowerShell!
 - Over time of Windows versions the size, the disorganization was evolved, so be carefull to make any modification in it!

System structure

- ▶ 1. Hardware layer (CPU, memory, devices, BIOS, etc)
- ▶ 2. Hypervisor layer (if exist, not necessary), every OS runs in its virtual machine.
- ▶ 3. HAL – Hardware Abstraction Layer, holds abstracts low level hw details, registers, timers,DMA,etc.(hal.dll)
- ▶ 4. NTOS Executive layer(contains I/O manager, process manager, memory manager, notification, etc)
- ▶ 5. NTOS kernel layer (Deferred Procedure calls, ISR, APC)
 - ntoskrnl.exe

Processes in Windows

- ▶ Processes can be grouped – called job! (batch processing feature)
- ▶ Each process has a user mode data: PEB (Process Environment Block)
 - It includes loaded modules, environment data, etc.
- ▶ Every process starts with one thread! Process acts as a „thread container”!
 - Later new threads can be created dynamically as needed!
 - TEB (Thread Environment Block) – user data for thread

Interprocess communication, synchronization

- ▶ IPC functions:
 - Pipe, named pipes
 - Byte and message type pipes!
 - Sockets
 - Mailslots (for OS/2 compatibility)
 - Shared files (memory)
 - RPC
 - There are no significant differences to Unix.
- ▶ Syncronization
 - Semaphores, mutexes, critical regions, notification events, synchronization events.

Scheduling in Windows

- ▶ It uses a priority based scheduling avoiding starvation! (Higher priorities first!) (Dynamic Fair-Share Scheduling)
- ▶ There are 32 priorities in Windows!
 - 0 – Zero page thread
 - 1–15 User priorities
 - 16–31 System priorities (Real time class)
 - A thread has a base priority and a current priority! ($\text{current} \geq \text{base}$)
 - Windows maintains 32 lists of threads!
 - Avoiding starvation and for other reasons the kernel boosts the base priority of a thread! (Actual priority always <16)
 - Priority inversion – A lower and higher thread priority will be changed avoiding unnecessary semaphore waits!

Memory management

- ▶ In 32 bit environment each virtual address space is 4GB. (In 64 bit longer, depending OS version)
 - Typically 2GB User space+2GB shared OS system calls,etc.
 - Pagefile.sys, on system volume.
 - Normally 4kb page size (it can be max. 2MB)
- ▶ Memory manager focuses to processes!
 - For a process MM creates a Virtual Address Descriptor(VAD) entry, with 4 data (range of mapped address, backing store region, backing store map, permission)
 - VAD is organised as a balanced tree (like B-R tree)

Paging – Page faults

- ▶ On demand paging based on page faults!
 - A page table entry is a 64 bit long field!
 - Global page, Dirty bit,present bit, Accessed bit,etc
- ▶ 5 categories of page faults
 - The referenced page is not committed (invalid operation)
 - Access to a page is restricted by permission.
 - A shared copy-on-write page was modified.
 - The stack needs to grow.
 - The referenced page is committed and not mapped in! (This is the normal page fault!)

Page replacement algorithm

- ▶ Based on working set concept!
 - Each process needs a min-max page number for ideal work.
- ▶ 3 working set manager activity:
 - Lots od memory available
 - Memory getting low
 - Memory low (reducing working sets)
 - Working set manager runs appr. every second.
- ▶ 4 physical memory reference list:
 - Free, Modified, Standby, Zeroed

I/O operations

- ▶ Supporting automatic device discovery.
 - Some devices no needs specific driver!
- ▶ DDK – for creating new device driver
 - Driver verifier
 - Windows Driver Foundation (WDF)
 - UMDF – User-mode driver framework
 - KMDF – Kernel mode driver framework
- ▶ See books about WDF for more details!

NT File System

- ▶ Supports 255 long file names, 32767 long full path!
- ▶ Each NTFS volume a set of blocks (def block size 4kb)
- ▶ Index table based structure- main table is MFT(Master File Table)
 - MFT contains 16 records.
- ▶ Journaled file system, supports encryption,compression, soft RAID functionality.

**Thanks for your
attention!**

zoltan.illes@elte.hu

Operating systems

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

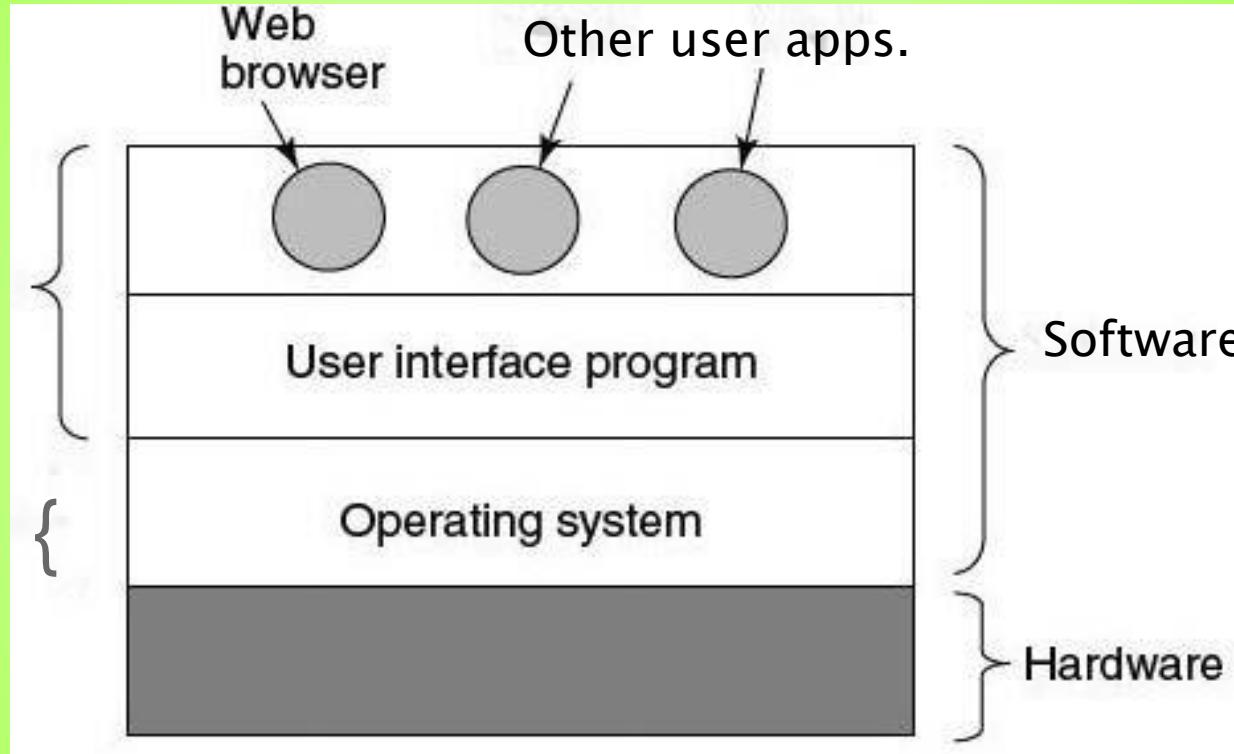
Introduction

- ▶ Review (Introduction to Comp. Technology)
- ▶ Computer Architecture I.(HW)
- ▶ Computer Architecture II. (SW)
- ▶ Definitions of the „Operating systems”
- ▶ History, evolution of Operating systems
 - Past, Present, Future?
- ▶ Notions, names, conventions in an operating systems
- ▶ System calls
- ▶ Structures of operating systems

An overview of a computer

User mode

Kernel mode



Review

- ▶ As we finished „Introduction Computer technology” ...
- ▶ Script programs
 - Best friend of the system administrator
 - Shell script
 - PowerShell
- ▶ Client–server architecture
 - HW differences
- ▶ Client–server services
 - Administration
 - SW differences

Computer Architecture (HW)

► Computer components

◦ Hardware side

- Stored programs, commands, data are stored (binary, why?) in the memory, same way.
- Central processing Unit (CPU), arithmetical-logical unit (ALU) are controls the commands execution and base arithmetical instructions.
- Input/Output handling, this is the communication between CPU and outside world.
- Neumann principals.

◦ Base parts: Processor, Memories, Pheripherals, data storage

- Connecting emlement: Bus (data, address, control)

Processor commands

- ▶ All parts are intelligent, no direct communication, main role: processor
- ▶ Registers: special memories inside of the processor
 - Register groups (general, segment, status,...)
- ▶ Groups of proc. commands
 - Data movement commands (registers-memories)
 - Jump commands (absolute-relativ)
 - I/O port read/write,
 - Interrupt handling etc.

Processor execution levels

- ▶ Intel 80286 – one level (real mode, 20 address lines– protected mode – not often used)
- ▶ Intel 80386 – starts in real mode– switch to protected mode
 - In protected mode there are 4 level (level 0, level 1, level 2, level 3)
 - Level 0 – kernel mode
 - Level 1, 2– not used
 - Level 3 – user mode
- ▶ **Level 0 functionality**
 - Interrupt handling (IDT)
 - I/O port management
 - Memory management
- ▶ **Sofware interrupt handling (trap) is same as a hardware interrupt handling**
Interrupt masking – NMI

Using processor commands

- ▶ Commands, data are in the memory, the CPU executes the commands
 - Mov al, 'F'
 - Mov ah,'T'
 - Mov bl,'C'
 - etc.
- ▶ Is it fun?
 - Sure--- (FTC:)... //FTC is a football team ☺
 - ...

Computer Architecture (SW)

- ▶ **Execution levels, sw levels**
 - Logical circuits – no sw
 - CPU, microprogram, microarchitecture level
 - Assembly, hw devices
 - **Operating system**
 - **System application (on user level too)**
 - Written in low level, in assembly
 - Written in high level language (usually in c,c++)
 - **User applications**
 - Like awk, Pasians etc.

Operating system definitions

- ▶ Operating system: Such a program (group) which gives to the user an easy to use workplace, hiding the computer(system) elements.
- ▶ Op. system as a virtual machine
 - Do not care how to copy, how to play a movie (film), only i need player or a „copy machine”.
- ▶ Op. System as a resource manager
 - Printing spool manager
 - Memory manager
- ▶ Kernel mode
- ▶ User mode
- ▶ Special controlled mode-embedded systems

Operating system main task

- ▶ Giving an easy to use, efficient user interface!
 - 0. generation: spec. Hw. Switching table
 - Early systems (1940–70): Special terminals
 - The architecture was same as today..
 - Beginning the age of 80th évek eleje: microcomputers (ZX81 etc), Basic
 - PDP compatible TPA1140, serial terminals
 - MS DOS character terminal
 - Unix_X Window system, Xerox, MacOS
 - Windows 3.1, 95,98,Mill,2000,XP, Win7
- ▶ Are they good user interfaces?

Peripheral communication

- ▶ **Continuous checking (polling)**
 - I/O port continuous reading
 - Often used technic, typically used in synchronic software calls.
- ▶ **Interrupt usage**
 - There is not continuous reading, the program waits for an event (external, data is ready), the handler will read the data.
 - Asynchronous method
- ▶ **DMA, direct memory access**
 - eg. direct memory addressing: 0xb800:0

Application libraries

► Computer command levels:

- Machine code
 - Pl:intel x86, mov ax, ‘F’, mov eax, ‘T’, jmp cím
- **Normal user API, Application Programming Interface**
 - C64 ROM Basic
 - DOS (IBM, MS) , IO.sys, msdos.sys, interrupt table
 - Windows 98,...Windows 7, Win32 API
 - **Unix–Linux system libraries, C language**
- **Script programming (BASH, PowerShell)**
 - We saw it in 1th semester!

User API's

- ▶ Layered architecture
- ▶ Divided into 2 groups:
 - Kernel level calls
 - Peripheral communication
 - User level calls
 - Wide library support
- ▶ Which programming language is supported?
- ▶ Sure, the C language! And more? C++ – etc...😊
- ▶ Compatibility!

What is POSIX?

- ▶ **POSIX = Portable Operating System Interface for uniX**
- ▶ **Official standard: IEEE 1003 – ISO 9945**
- ▶ **The POSIX is a minimal , standard API**
- ▶ **POSIX 1, 1a, 1b,1c ...versions**
- ▶ **ANSI C compatibility**
- ▶ **Every OS has POSIX compatibility**
- ▶ **Windows compatibility**
 - **Windows Services for Unix**

Main POSIX API features

- ▶ File, directory management
- ▶ Process control
- ▶ Signals
- ▶ Pipes
- ▶ Standard C library
- ▶ Clocks, Timers
- ▶ Semaphors
- ▶ Synchron, asynchron I/O
- ▶ Threads
- ▶ etc.

Function groups

- ▶ Math functions: pl. sin, cos, tan, atan, atan2, log, exp etc.
- ▶ File management functions: pl. creat, open, fopen, close, read, write, unlink etc.
- ▶ Directory management functions: pl. opendir, closedir, mkdir, rmdir, readdir stb.
- ▶ Character, string management: strcpy, strlen, strcmp, strcat, strchr, strstr stb.
- ▶ Memory management: malloc, free, memcpy stb.
- ▶ Communication functions: msgsnd, msgrcv, shmat, semop, signal, kill, pipe stb.

How can we use in practice?

- ▶ Our Op. system: Suse Linux Enterprise server
 - oprendszer.inf.elte.hu
- ▶ Text editor: vi, mcedit
 - Or using locally a graphics editing, and ftp.
- ▶ Help: man
 - E.g.: man exit, man strlen
- ▶ Compile: cc -c also also.c
 - Try to resolve warnings too!

Operating systems API's

- ▶ So much API's as Op. systems
- ▶ Typical API's
 - Open VMS
 - OS/400
 - System V, BSD , közös rész: POSIX
 - Win32 API
 - Mac OS API
 - Windows Mobile, CE API
 - Palm OS
 - Nokia S40, S60, S80 API
 - Android x, WP7,8, etc.

Firmware – Middleware

- ▶ We saw: Hardware – Software
- ▶ Hardware is not only the physical device
 - Eg: HDD firmware.
 - BIOS calls.
- ▶ Firmware: An integrated software is built into hw by manufacturer
- ▶ Middleware: A system layer is above Op. System.
 - E.g.: JVM

Operating systems generations I.

- ▶ Historical generation: Charles Babbage (1792–1871)
 - Fully mechanics machine, no op.system
 - Operator job
 - Later, one of programmer was Ada Lovelace (Lord Byron's (poet) daughter) (Ada language)
- ▶ First generation, 1940–1955, switching table, reles, electrical vacuumtube
 - Neumann János, Institute for Advanced Studies, Princeton
 - Customised comp. machines
 - Machine code, simple math calculations
 - Punched cards as a data, command holder

Operating systems generations II.

- ▶ Second generation 1955–1965, transistors systems
 - Trusty, better computer components
 - Mainframes, computer centers
 - Distinct planning, manufacturing, programming, operation
 - Punched cards,tapes systems, batch systems
 - Fortran language
 - Op. system
 - FMS, Fortran monitor system
 - IBM 7094 machine, 1401 input – 7094 evaluation–1401 output unit

Operating systems generations III.

- ▶ Third generation, 1965–1980, appears of integrated circuits
 - IBM 1401 and 7094 next one: System/360 family
 - Same architecture, construction, compatibility
 - Developing OS/360 , Fits to all System/360 HW, it results a big, tricky, complicated op. system.
 - Multiprogramming, multitask
 - There are more tasks in the memory at the same time.
 - Spooling, time sharing systems
 - No general on-line access, work

Operating systems generations III.

- ▶ First time sharing system: M.I.T-en CTSS (Compatible Time Sharing System)
- ▶ MULTICS, Multiplexed Information and Computing System
 - AT&T Bell labs, General Electric support
 - PL/1 language
- ▶ Bell Labs, Ken Thompson, Simplifying Multics , PDP 7->UNIX
- ▶ Two main developing stream:
 - Berkeley University – Berkeley Software Distribution
 - AT&T Bell Labs, System V Unix

Operating systems generations IV.

- ▶ Since 1980 till now, personal computers, MS Windows
- ▶ LSI (large scale integration) circuits, CPU development
- ▶ Z80 – CP/M (Control Program for Microcomputers)
 - ZX-81, ZX-Spectrum – Basic
- ▶ Intel x86 family, IBM PC – DOS, MS DOS
 - Command line desktop
- ▶ GUI – X Window, Mac OS X, MS Windows
- ▶ Network, distributed systems

MINIX 3

- ▶ From the beginning the UNIX source was free, open by AT&T .
- ▶ UNIX – not open, free from AT&T 7. version
- ▶ MINIX – MINI Unix, open source code
 - A.Tanenbaum, Vrije Univ. Amszterdam
 - Written in C language
- ▶ Linus Torvalds, ,Tanenbaum's student'
 - MINIX modification, 1994, LINUs uniX->LINUX
 - Open source
 - LAMP-Linux-Apache-Mysql-Php

System calls

- ▶ **System call:** An operating system library call which connects the user level call to the kernel level.
- ▶ **Two category:**
 - Task or process handling call
 - File management call
- ▶ **The best friend of the programmer:** man, ...

Process management

- ▶ **Process-** a program loaded into the memory and executing it
 - Own address space
 - **Process table**
 - Address, register, workfile datas, metadatas
 - **Process start, stop**
 - Shell, child processes
 - **Process pause**
 - Memory map + process table save
 - **Process communication**
 - Signals

File management

- ▶ Only one directory, /
 - Tree structure
 - Two registry entry: file, directory
- ▶ Dir. operations: create, copy, delete, open, read, write
- ▶ Access rights: rwx, – denied the given right
 - SETUID, SETGID, Sticky bit
- ▶ File system connecting (mount), disconnecting (unmount)
- ▶ Specific files:
 - Character, block files, /dev directory
- ▶ Special file: pipe

Major process handling calls

fork – To create a new process

exec – To execute a new program in a process

wait – To wait until a created process completes its execution

exit – To exit from a process execution

getpid – To get a process identifier of the current process

getppid – To get parent process identifier

nice – To bias the existing priority of a process

brk – To increase/decrease the data segment size of a process

Major signal handling functions

`sigaction` – Define action to take on signals

`sigreturn` – Return from a signal

`sigprocmask` – Examine or change the signal mask

`sigpending` – Get the set of blocked signals

`sigsuspend` – Replace the signal mask and suspend the process

`kill` – Send a signal to a process

`alarm` – Set the alarm clock

`pause` – Suspend the caller until the next signal

Major file management functions

open – Create a file or open an existing file

close – Close a file

read – Read data from a file

write – Write data to a file

lseek – Move the file pointer

stat , fstat – Get various file attributes

fcntl – File locking

Major directory management functions

`mkdir` – Create a new directory

`rmdir` – Remove an empty directory

`link` – Create a link to a file

`unlink` – Delete the link

`mount, umount` – Mount/ unmount a filesystem

`chdir` – Change the current working directory

Operating system structures

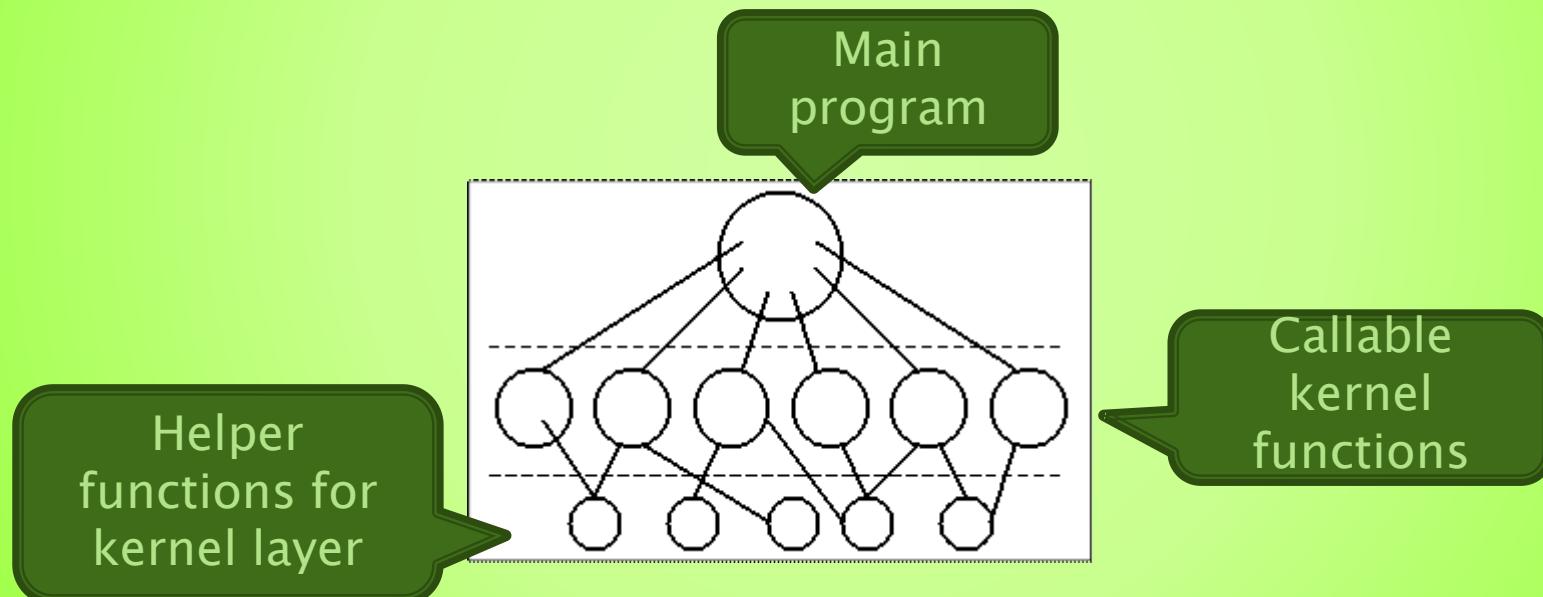
- ▶ **Monolithic systems**
- ▶ **Layered systems**
- ▶ **Virtual machines**
 - Exokernelek
- ▶ **Client – Server model**

Monolithic systems

- ▶ Generally: no defined structure, but...
- ▶ System library is one big system, so every process can use everything.
 - No information hide.
- ▶ Exist software moduls, planning modul groups
 - The predefined entry points(functions) can be called!
- ▶ During a system call the execution level is often switched to kernel mode (CPU execution level 0)
 - Parameters in registers
 - Trap

Monolithic system model

- ▶ Monolithic system: typically a 2 layers support



Layered architecture

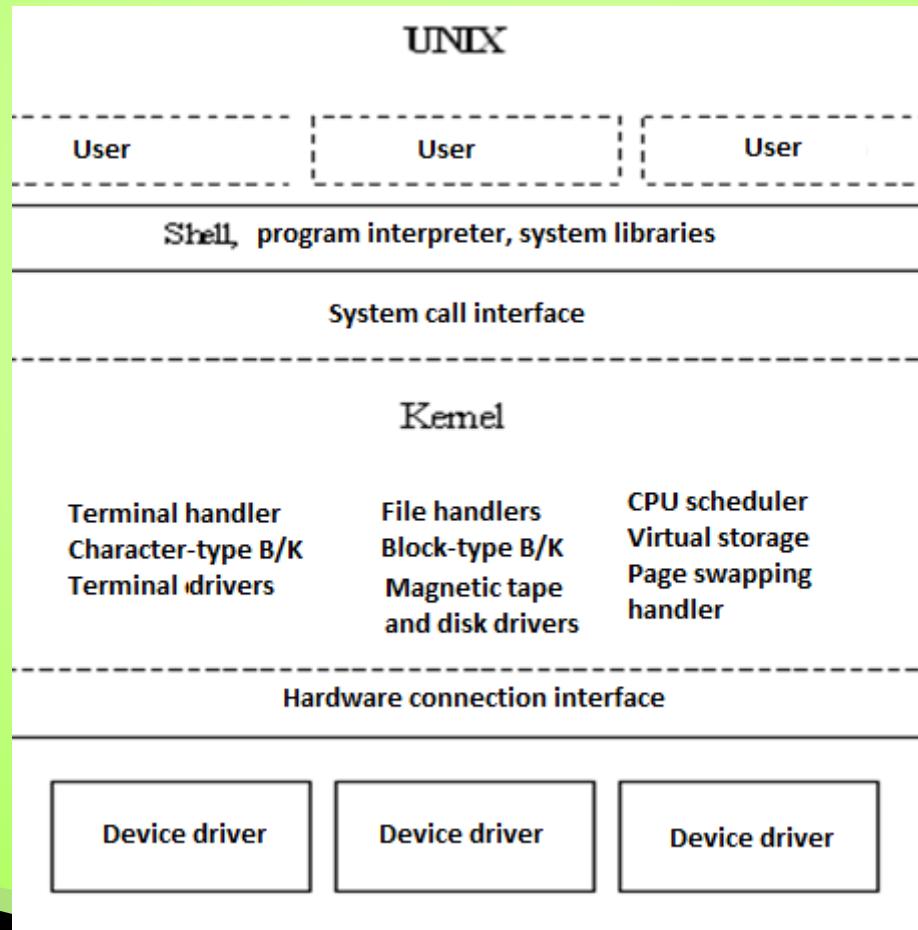
- ▶ Planned by E.W. Dijkstra : THE (1968)

5.	The operator
4.	User applications
3	I/O management
2	Machine-process
1	Memory management
0	CPU management and multiprogramming

- ▶ In MULTICS more generally
 - Round, ring architecture

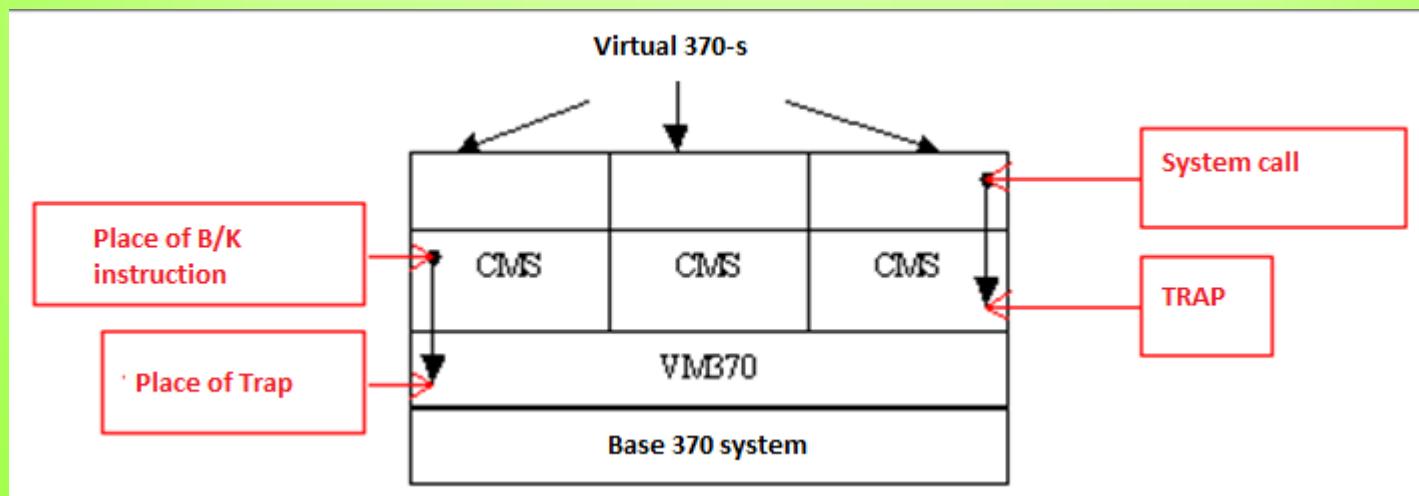
Typical layered structure

- ▶ The UNIX typical layered (round) structure.



Virtual machines

- ▶ The idea comes from IBM
- ▶ It was implemented on VM/370 system
- ▶ Virtual machine Monitor: gives a „copy” of the hardware

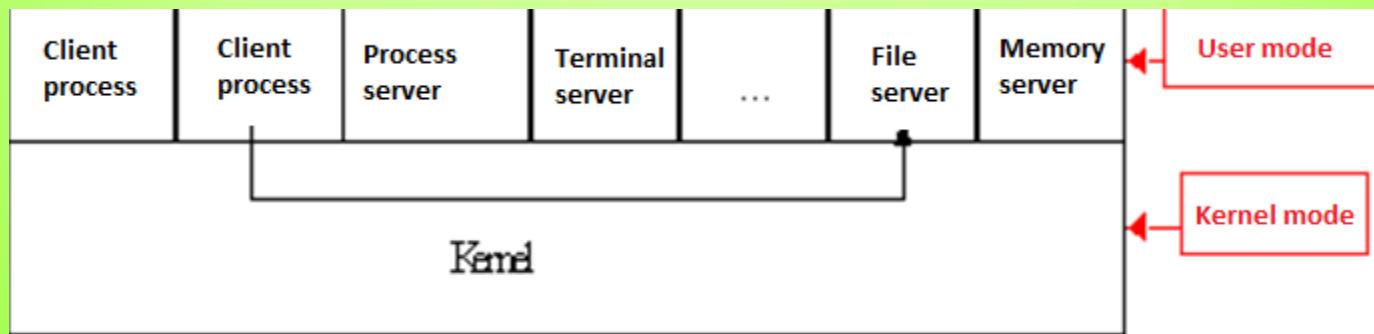


Virtual machines today

- ▶ **VMWare – under Unix– Linux platform**
 - Under Windows too
- ▶ **MS Virtual Server, Virtual PC**
 - Todays processors supports virtualization
- ▶ **Microsoft– Hyper-V**
- ▶ **LINUX – XEN–KVM**
- ▶ **Other virtualization:**
 - JVM
 - .NET

Client–Server model

- ▶ Improvement of the idea of vm/370
 - To divide the task to much more parts.
- ▶ User application: client program
- ▶ Service application: server program
- ▶ Both runs in user mode
- ▶ Less and less functions stays in the kernel



Operating system expectations I.

- ▶ **Efficiency**, an efficient management of the real resources
- ▶ **Reliability**, Build a reliable system
 - Working all time without problems
 - Archive datas
 - 3–4 nine... reliability (99.9%, 99.99%)
 - **Failover systems**
 - Redundant systems(both HW and SW too), Server Cluster

Operating system expectations II.

- ▶ **Security**
 - Safe work from outside „attacks”, firewall
 - Data security
- ▶ **Compatibility**
 - Data, program exchange between systems.
 - Role of standards (POSIX)
- ▶ **Low energy occupation**
 - Not only for mobile devices

Operating system expectations III.

- ▶ **Flexibility**
 - Flexibil resource management(memory, processor, today cloud technology)
- ▶ **Manageability**
 - On support and user level too.
- ▶ **Can we offer all expectations?**
 - All manufacturers answer: yes....😊
- ▶ **We'll see later!**

**Thanks for your
attention!**

zoltan.illes@elte.hu

Operating systems

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

We have talked about...

- ▶ **Operating system generations**
 - Typical hw constructions
- ▶ **Op. Systems definitions**
- ▶ **Notions:**
 - Files, directories, processes
- ▶ **System calls**
- ▶ **System structures**
 - Today: Client-server model, with layered features

What comes today...

- ▶ **Data storage types**
 - Magnetic tape, magnetic disk, optical storage
- ▶ **Formatting,**
- ▶ **MBR, partitions**
- ▶ **Boot process**
- ▶ **Disk access, features, schedulers, efficiency**
- ▶ **Redundant data storage, RAID**

Data storage types

- ▶ Magnetic storage type
 - Magnetic tapes
 - Magnetic drives
 - Hard disk
 - Floppy disk
- ▶ Optical storage type
 - CD, DVD, Blu-Ray, laser, appr. 5xDVD capacity
 - It is based on light(laser, from red to blue) reflection (pit–land)
- ▶ Semiconductor principle
 - USB, memory card
 - SSD(Solid State Drive/Disk) disk

Data storage types tomorrow

- ▶ Holographic
 - GE communications, 2011, 500GB, holographic storage system, a bit is a hologram
- ▶ Biology
- ▶ Nano architecture
- ▶
- ▶ Moore's law, ... "...but..."

Physical architecture of magnetic tapes

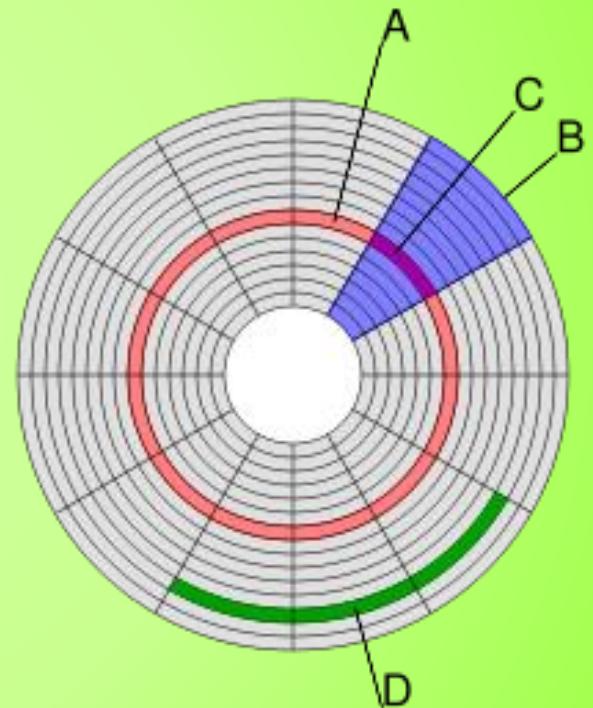
- ▶ Magnetic tape – linear storage, architecture
 - Linear Tape File System, XML based structure
 - 9 bits frames (8 bit + 1 bit parity)
 - Frames organized into records
 - Between records: record gap
 - A file is a set of records, between files there is a file gap
 - Beginning of the tape is stored the dir. structure
- ▶ Typical usage
 - Security archive (backup)
 - For storing „big data”
- ▶ Not so cheap
- ▶ Typical size today: DLT (Digital Linear Tape), LTO (Linear Tape-Open) 4 Ultrium 800/1600 GB, LTO5 1.5TB/3TB, LTO6 2.5TB/6.25TB

Architecture of magnetic disks I.

- ▶ FDD – Floppy Disk Drive
 - Typically one disk
- ▶ HDD – Hard Disk Drive
 - Typically more disk (or one)
- ▶ Disk is round – divided into stripes
- ▶ Stripes are divided into sectors – intersection is a block
 - cluster – more block
- ▶ If the drive has more disks – the stripes above each other : cilinder
- ▶ The disk is logically a rage of blocks (0-xxx)
- ▶ Firmware: hides physical operations

Architecture of magnetic disks II.

- ▶ A: stripe (red)
- ▶ B: sector (blue)
- ▶ C: block, 512 byte
- ▶ D: cluster, n is decided by OS. $D=n \times C$, where $n=1..128$.(green: $n=3$)
- ▶ Cylinder: stripes above each other (red)



Addressing of magnetic disk

- ▶ CHS address (Cylinder- Head- Sector)
 - Example: 1.44 MB FD
 - No. Of Cilinders: 80 (0–79)
 - No. Of heads: 2 (0–1)
 - No. Of sectors on a track: 18 (1–18)
 - Total size: $80 \cdot 2 \cdot 18 = 2880$ szektor * 512byte
- ▶ LBA address (Logical Block Addressing)
 - Earlier 28 bits, up to 137GB.
 - Now 48 bits, up to 144 PB (Petabájt), (144 000 000 GB)

$$A = (c \cdot N_{\text{heads}} \cdot N_{\text{sectors}}) + (h \cdot N_{\text{sectors}}) + s - 1$$

Optical storage

- ▶ A 8 or 12 cm (diagonal) optical disks
 - CD – Compact Disc, DVD –Digital Versatile Disc
 - Size: 650MB – 17 GB között
 - Speed: 1x = 150 KB/sec
- ▶ Working principle: Laser light reflection differences.
 - The time the reflected light from a pit is longer than from the land (pit–land range)
 - Writable disks: writing process: laser beam (heat) changes (increase) the light refraction coefficient (pit).

Device driver

- ▶ A program which makes communication between user application and hardware.
- ▶ Part of the kernel.
- ▶ During disk read or write there is used DMA (big data)
 - Interrupt message: there is finished the read or write activities.
 - I/O ports are used to setup device parameters.
- ▶ Layered architecture

Formatting magnetic disks

- ▶ Creating the tracks–sectors architecture
- ▶ Typical block size is 512 byte
- ▶ Disks are ready to use by manufacturers
- ▶ Quick format– Normal format
 - A normal format finds bad sectors on the disk.
- ▶ Block (Sector) content= Head information+datablock+footer inf.
 - Block inf.: track no., head no., sector number
 - Footer inf.: error corr. block
- ▶ Low level format: Creating sectors, done by manufacturer.

Logical formatting

- ▶ Creating partitions
 - Allowed max. 4 on a hard disk.
- ▶ 0. sector– MBR (Master Boot Record)
 - Contains 2 parts, like every blocksize is 512 byte
 - First part: bootloader code, max. 446 byte
 - Second part: Max. 4 partitions data ($4 \times 16 \text{ byte} = 64 \text{ byte}$)
 - Record close 2 byte: 0x55,0xAA
 - Primary partition– op. system can boot
 - Extended partition– there are more logical drive
 - Swap partition
- ▶ Task of formatting: Creating the necessary data structure on the partition (volume)

MBR architecture

Address			Description		Size (byte)
Hex	Oct	Dec			
0000	0000	0	Loader program code		440 (max. 446)
01B8	0670	440	Optional disk code		4
01BC	0674	444	Typically: 0 x 0000		2
01BE	0676	446	Primary partition table data (4 16 byte long part, IBM partition table schema)		64
01FE	0776	510	55h	MBR closing: 0 x AA55	2
01FF	0777	511	AAh		
446 + 64 + 2 =					512

A partition entry description

- ▶ 1. byte: Partition status (80=activ, 0=nem boot)
- ▶ 2–3–4. byte : Partition begin block CHS address
 - 0–5. bit: No. Of heads
 - 6–15. bit: No. Of cilinders
 - 16–23. bit: No. Of sectors
- ▶ 5. byte: Partition type (primary, extended)
- ▶ 6–7–8. byte : Partition end block CHS address
- ▶ 9–10–11–12. byte: Partition begin LBA addr.
- ▶ 13–14–15–16. bájt: No of sectors
 - 4 bájt: $2^{32} * 512 \sim 4 \text{ GB} * 512 = 2 \text{ TB}$

The Boot process

- ▶ From a boot device, ROM-BIOS loads the „boot program” from MBR at 7c00h address.
- ▶ Only one primary partition can be active.
- ▶ Boot program loads the first block of active partition into memory.
- ▶ This is the operating system boot program,
eg:LILO, NTFS boot
- ▶ The boot program knows, which files can be loaded to make a „systemstart”
 - Multi layered process, vary from operating systems.

Interleave–Block calculation

- ▶ For reading-writing we have to calculate the blocks number.
 - Number of heads, sectors
 - Assume we have 4 head (2 or 4 disks)
 - Lets divide one track into 7 sectors
- ▶ Because of the rotation speed the blocks logical order is not same with the physical order! (interleave)
 - 1:2 interleave, every second sector gives the log. order”

	1 sector	2 sector	3 sector	4 sector	5 sector	6 sector	7 sector
1 head.	1	17	5	21	9	25	13
2 head.	2	18	6	22	10	26	14
3 head.	3	19	7	23	11	27	15
4 head.	4	20	8	24	12	28	16

Physical parameters of disk access

- ▶ Rotation speed (typical 5400,7200,10000 or 15000 rpm)
 - How many times rounds in a minute.
- ▶ Head speed
 - Inside a cilinder the head must not to move.
- ▶ The task of disk I/O scheduler is to choose an efficient, quick service order
 - Decrease the time of the disk access
 - Increase the data(read–write) bandwith

Reading-Writing access

- ▶ We need for a system call(read):
 - The number of block(s) to read.
 - The address space into write.
 - Number of bytes
- ▶ We have more processes...more read-write request
 - Main question: Which one is the first?

Scheduling of disk access

- ▶ Low level parameters(kernel)
 - Type of request (read-write)
 - A block begin address, (track, sector, head)
 - DMA address
 - Number of bytes to read-write
- ▶ Disk is used by every process
 - Who will be served first?
 - We'll calculate of head position

FCFS scheduler

- ▶ First Come – First Service order
- ▶ Simple strategy, as the requests comes, they will be served in the same order.
- ▶ We sure, all requests will be served!
 - No starve (perish with hunger)
- ▶ Do not care about head position.
- ▶ Not so efficient.
- ▶ Low read–write bandwith.
- ▶ General, average service time, low deviation.

SSTF scheduler

- ▶ Shortest Seek Time First – SSTF
- ▶ We will service that request which is more closer to actual head position.
- ▶ Average waiting time is low.
 - But the deviation is high.
- ▶ High read–write bandwidth
- ▶ There is real starve danger (perish with hunger)

Scan scheduler

- ▶ SCAN (LOOK) method
- ▶ The head is moving all time from inside to outside and back and service the requests of call.
- ▶ The head turns back if there are no requests at that direction or at rich the end position.
- ▶ The service of bad timing requests is after a „round”.
 - Waiting time is medium, high deviation
- ▶ Deviation of access tracks located at the center of disk is low.

C-Scan scheduler

- ▶ Circular SCAN, C-SCAN
- ▶ Modifying of SCAN, reading-writing service is only in one direction.
- ▶ Quicker head movement
- ▶ Higher bandwidth
- ▶ Average waiting time, low deviation.
 - No real bad request

Improve scheduler efficiency

- ▶ FCFS method: if we have a request near from actual head position, lets to serve it! (Pick up)
- ▶ Usually a file data is located in a block. If we ask to read the first part, in that case the system will read the second part too.
 - Scheduler in advance
- ▶ The access of the middle of the disks is very efficient!

Improve scheduler efficiency using memory

- ▶ DMA (also memory)
- ▶ Memory buffer
 - Double using
 - Read: Scheduler writes information, user reads it.
 - Write: User writes information, scheduler reads it
- ▶ Disc cache
 - Reads or writes not only the asked data, but the closest integral data too.
 - Gives additional operating system task
 - Eg: Smartdrive

Which scheduler can we choose?

- ▶ The algorithms calculates only with the head position
- ▶ FCFS is used in a single user system.
- ▶ SSTF, There is real starve danger
- ▶ C-Scan , Big I/O bandwith, no starve
- ▶ Bild in schedules: eg. SCSI controller
 - Generally OS sends requests one after the other.

The main role of scheduler

- ▶ To give quick answer to requests.
- ▶ We can (OS too) improve it:
 - Defragmentation
 - Best bandwidth in the middle of disks.
 - Access of disks is more quick in the middle too (virtual memory)
 - Disk buffer in the memory.
 - Maybe data compression (more CPU load)

Disk efficiency–Redundant data storage

- ▶ Definition: Avoid data loss in case of disk error
- ▶ Operating system support
 - Dynamic volume– more than one disks gives a logocal volume. The volume size is the sum of disks.
 - Mirroring– Two disks works parallel as one volume. The volume size is one disk (less) size.
 - Bigger CPU load.
- ▶ Hardware support
 - Intelligent devices
 - SCSI devices supports (RAID)

Redundant drives

- ▶ RAID – Redundant Array of Inexpensive Disks
- ▶ First was used at SCSI devices
 - Small Computer System Interface
 - A definition, a standard for data exchange between computers and devices.
 - Usually was used to connect HDD's to computers.
 - Today SCSI rarely used, instead: SAS controller (Serial Attached SCSI)

RAID

- ▶ If supports it an operating system: SoftRaid.
- ▶ If supports it a hardware: Hardware Raid- or simply Raid system.
- ▶ However in the names inexpensive, but usually more expensive than a normal disk.
- ▶ Collect more disks, and gives as a single volume to the operating system.
- ▶ There are more type of raid's:RAID 0-6

RAID 0(striping)

- ▶ This is a Raid, but not redundant...
- ▶ Like OS supported dynamic volume, but this is supported by a RAID controller. Adds more disks into a collection (Raid Array) and
- ▶ It gives the summary of size of individual disks as a new logical disk.
- ▶ Striping, a (big) file parts is stored on more disks.
- ▶ Quick I/O execution.
- ▶ No error correction, no redundancy.

RAID 1 (Mirroring)

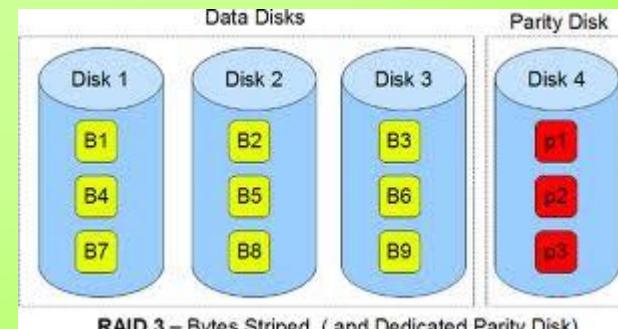
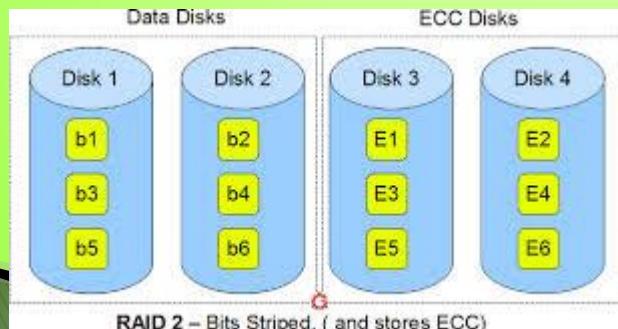
- ▶ Creates a logical volume from two independent disks.
- ▶ Every data is stored on both disks (mirror)
- ▶ The total storage capacity is half size.
- ▶ It is expensive. (2 disks, one size)
- ▶ Significant redundancy.
 - Occuring at one time some disk error on both disk, data lost.

RAID 1+0, RAID 0+1

- ▶ RAID 1+0: Creating a mirror using RAID 0 volumes.
- ▶ RAID 0+1: Creating a volumes using RAID 1 (mirrored) disks.
- ▶ Nowdays the cotrollers are usually supports them, but rarely used because of price!!
(mirror, 2 disk 1 size)

RAID 2,3,4

- ▶ RAID 2: It contains not only data bits, but error correction bits too. (ECC–Error Correction Code) Eg. 2 data disk, 2 ECC disk
- ▶ RAID 3: Only a Plus parity disk, $n+1$ disk, $\text{sum_size}=\sum n$
- ▶ RAID 4: RAID0 implementation with parity disk.
- ▶ RAID 2,3,4: Rarely used.



RAID 5

- ▶ No parity disk, the parity information is striped.
(stripe set)
- ▶ Data, files etc. also is striped!
- ▶ Intensive CPU load (controller has own CPU!!!)
- ▶ Redundant storage, occurring 1 disk error do not cause a data lost
 - 2 disk error at the same time occurs data lost!
 - How is it works? (From the parity and data bits we can recalculate, recover a lost bit!)
- ▶ N disks in a RAID 5 array($N \geq 3$), Total volume size is sum. $N-1$ disks.

RAID 6

- ▶ Add to RAID 5 parity block an error correction code.(+1 disk)
- ▶ More intensive CPU load.
- ▶ Two disks failure does not cause a „system crash”, a data lose!
- ▶ Relative expensive
- ▶ Capacity of N disk in a RAID 6 array, is same as N-2 disk capacity.
- ▶ In principle we can continue the error correction method (3 disk failure, etc...)

RAID summary

- ▶ Most often used RAID types: RAID 1,5
- ▶ RAID 6 controllers appeared last 1–2 years.
 - Raid– Inexpensive disk, but in reality the Raid disks more expensive than other, normal disks
 - Raid 6–2 disks spare, not effective used, a little bit more expensive!
- ▶ Hot-Swap RAID controllers– Hot-swap disks: during computer work(without switch off), we can safely change the failed disk.

Summary of data storage

- ▶ How can we support safe data storage?
- ▶ Multilayer result:
 1. Physical disks(HDD)
 2. Hardware RAID
 3. Partitions
 4. Software RAID
 5. Volume Manager in the operating system.
- ▶ It is not safe if:
 - eg: Power supply (redundant too) is off, operator mistake, etc.
 - Software attacks, viruses.
- ▶ How our data is organised on a „volume”?

**Thanks for your
attention!**

zoltan.illes@elte.hu

Operating systems

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

We have talked about...

- ▶ **Operating system generations**
 - Typical hw constructions
- ▶ **Op. Systems definitions**
- ▶ **Notions:**
 - Files, directories, processes
- ▶ **System calls**
- ▶ **System structures**
 - Today: Client–server model, with layered features
- ▶ **Data storage**
 - Types, efficiency, hw, sw

What comes today...

- ▶ **Files**
 - File types
- ▶ **Directories**
 - Directory architecture
- ▶ **File systems**
- ▶ **File system access scheduling**
- ▶ ...

Filesystem

- ▶ File: logical grouping a data set with a name (filename), with other parameters e.g. Size, date, etc.
- ▶ Directory: logical grouping of files and directories
- ▶ Filesystem: a method, how to organize this storage, data system on a physical disk.

Files

- ▶ A file: a base unit of information storage.
- ▶ Most common property: name.
- ▶ Stored on a storage device, HDD, SSD, SD card, etc.
 - The standard output, input are also a special file.
- ▶ Generally there are 3 types of files:
 - User files.
 - Temporary files
 - Administration, configuration files. Part of op.system.

File properties

- ▶ **Filename:** a string
 - There are restrictions for containing characters
 - Length, special characters are not allowed(*, \,&,etc.)
 - Restrictions are up to operating system.
- ▶ **Other attributes (additional information)**
 - Size of file, owner, last modification date, hidden or read only file, general access rights, etc...
- ▶ **Physical location on the drive(e.g. LBA address)**
 - Real file, link (hard), link (soft)

Directories

- ▶ A special entry, shows the place in the file system where the containing files and directory entry are recorded.
- ▶ Directory architectures
 - Systems without any catalogs, e.g. Early tape systems
 - One level, two level catalogs
 - Not used today
 - Multi level, hierarchical catalog structure
 - Tree structure
 - Effective search
 - Today typically is used.
- ▶ Absolute, relative names
 - PATH environment variable

Access rights

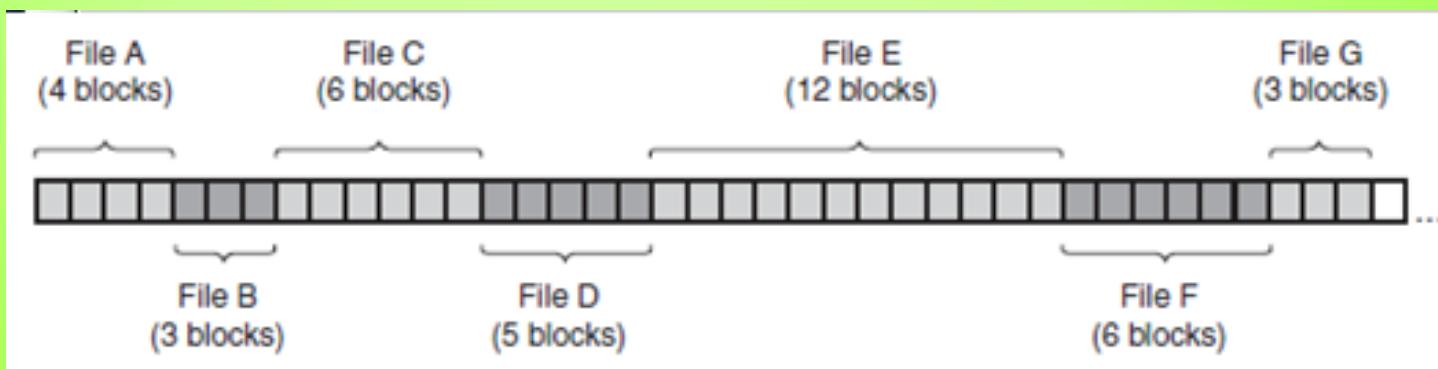
- ▶ There are no generally used security system
- ▶ Typical access rights:
 - Read
 - Write, Create, delete
 - Execution
 - Modify
 - Full control
- ▶ Recording of access rights
 - As Attributes
 - ACL (Access Control List), getfacl, setfacl, chacl

Partition description

- ▶ At the beginning of partition: a Superblock (e.g. FAT, the 0. block) describes the properties of file system.
- ▶ Next one is usually a used/free block registration (e.g. FAT, File Allocation Table)
- ▶ Next part is the directory structure (inode), file, directory entries, and file data
- ▶ Where can we store a new file?
- ▶ What kind of strategy can we choose?

Storage strategy of files I.

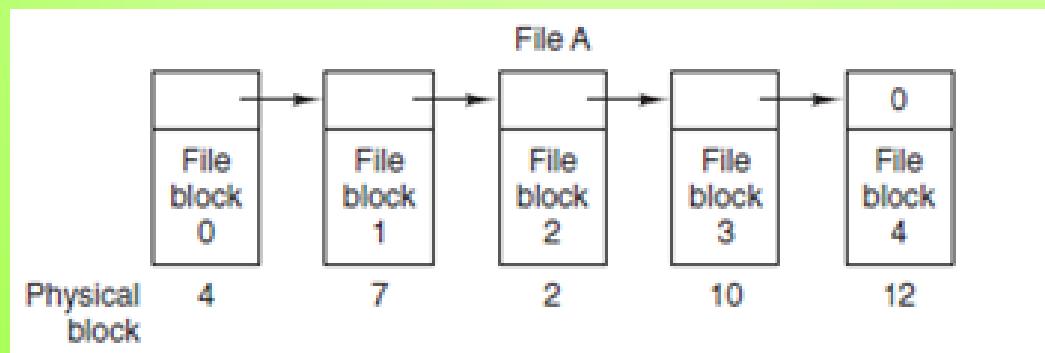
- ▶ Contiguous storage place reservation
 - First Fit
 - Best Fit
 - Worst Fit (Put the file in such a free disk gap, so the remaining place (free place) should be the biggest)
 - In all of above are lost (waste) place.



Storage strategy of files II.

► Chained storage

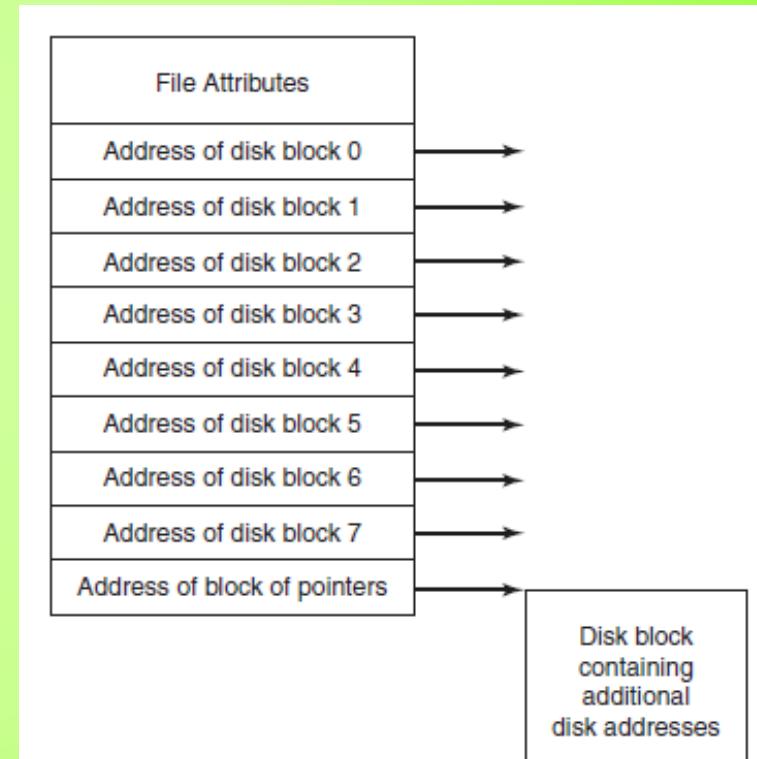
- No waste of place(only from the block size)
- The file data is stored as a chained list of blocks
 - The access of last blocks of files will be slow.
- Free- Reserved blocks : File Allocation Table,FAT
 - It might be big, and the FAT is always in the memory!



Storage strategy of files III.

▶ Indexable storage

- The directory catalog contains the address of file node
- The inode address gives the file data.



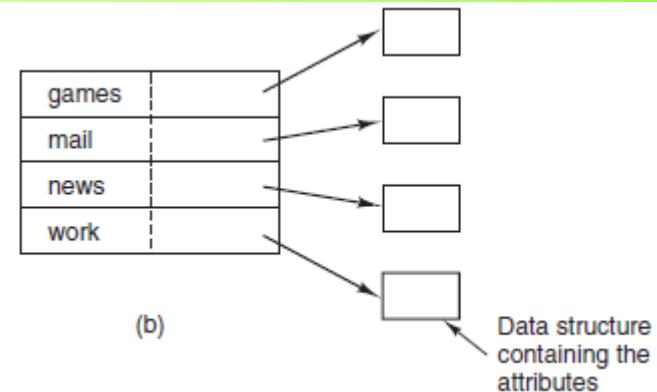
An example i-node.

Implementation of a directory

- ▶ Main function to bind names and locations searching for filenames (linear, hash table, using cash)
- ▶ A directory entry may contain:
 - The disk address of the entire file (contiguous file allocation)
 - The number of the first block (linked list scheme)
 - The number of the i-node.
- ▶ Where do it store the attributes? There are two typical methods:
 - Fixed length
 - In i-nodes

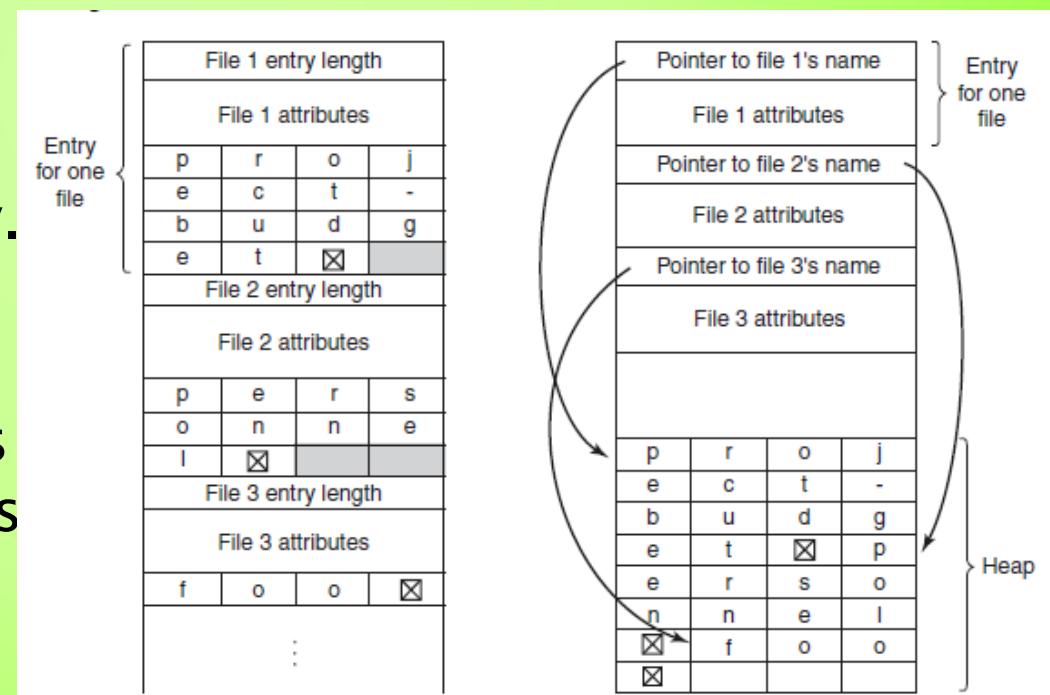
games	attributes
mail	attributes
news	attributes
work	attributes

(a)



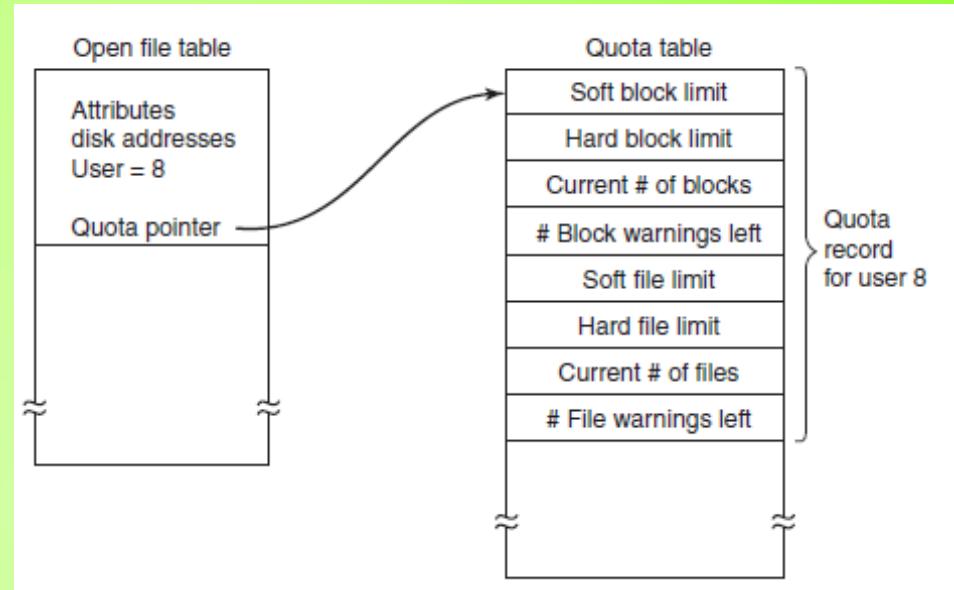
Filename implementation

- ▶ Recently – fix length (8+3)
- ▶ Nowadays – usually max 255 characters
 - How can we avoid place wasting?
 - With different length, at the first place storing the length of the entry.
 - With same length entries using pointers to various length of filenames



Disk quotas

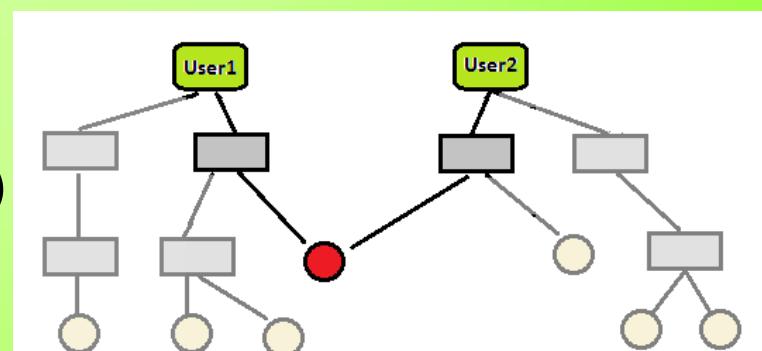
- ▶ If user opens a file
 - Opens file table (in memory)
 - Quota table (in memory)



- ▶ Adding new blocks -> change quota table
- ▶ If there is an attempt to log in
 - Over soft limit, counted warnings (reached max) – disabled
 - Over hard limit – log in is not permitted

Shared files

- ▶ How can the system manage to be able to see all of the changes the other user made?
- ▶ There are mainly two solutions
 - The file blocks are not listed in the directory entry, they are in a structure and directory entry point to it. (e.g. UNIX where the data structure is the i-node) So user1 and user2 points the same structure. (What happens if the owner want to delete it?)
 - With a new „link” file, a link to the original filename. (symbolic link) (Slow access.)

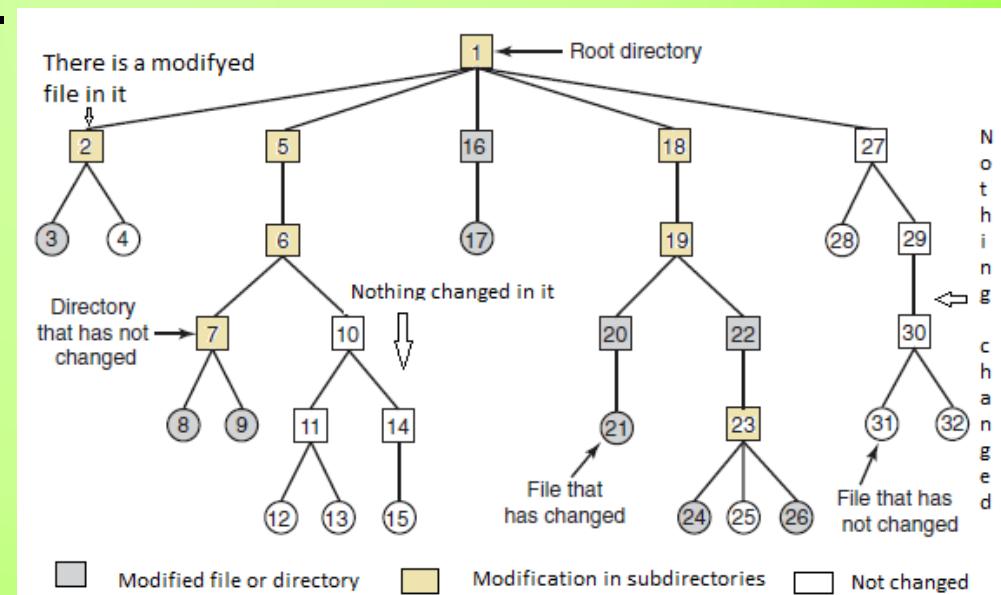


Backups

- ▶ Physical dump (copies everything from the 0th block)
 - Advantage (simple, quick)
 - Disadvantage (copy unused, failed blocks..)
- ▶ Logical dump (copies only the newer or modified files)
 - Advantage (may restore a required directory or file too, not only the entire disk)
 - Disadvantage (complicate algorithm)
- ▶ Mixed backups
 - Time to time a physical dump
 - Frequently a logical dump
- ▶ Restore (physical dump, logical dump1, ... last

Logical dump – algorithm

- ▶ 4 step algorithm (used in most UNIX systems)
 - Mark each directory+modified files
 - Unmark directory in which there are no modified files or directories.
 - Dump the marked directories with attributes.
 - Dump the marked files with attributes.



Questions in logical dumping

- ▶ Free block list is not a file – so it is not dumped (But it can be restored, while it is the complement of used ones.)
- ▶ Links – several directories may contain a link to a modified file. (Restore all.)
- ▶ A file may contain holes (e.g. seek+write) – Allocate a huge area and fill it with 0?? (No)
- ▶ What to do with not real files (e.g. named pipe) – do not dump.

File, directory functions

- ▶ File
 - Open a file
 - Functions: read, write, append
 - Close a file
- ▶ File data
 - Binary – a sequence of bytes
 - Text file – a sequence of chars
- ▶ File access: sequentially, random
- ▶ Directory functions
 - Creating, list table of contents, delete

Filesystem types

- ▶ Filesystem types on HDD:
 - FAT, NTFS, EXT2FS, XFS, etc.
- ▶ On tape systems, typically used on backup system
 - LTFS – Linear Tape File System
- ▶ CD, DVD, Magneto-opto Disc filesystem
 - CDFS, UDF (Universal Disc Format)
- ▶ RAM drives (today rarely used)
- ▶ FLASH memory (FAT32)
- ▶ Network drive
 - NFS
- ▶ Other pseudo filesystems
 - Zip, tar.gz, ISO

Journaled filesystems

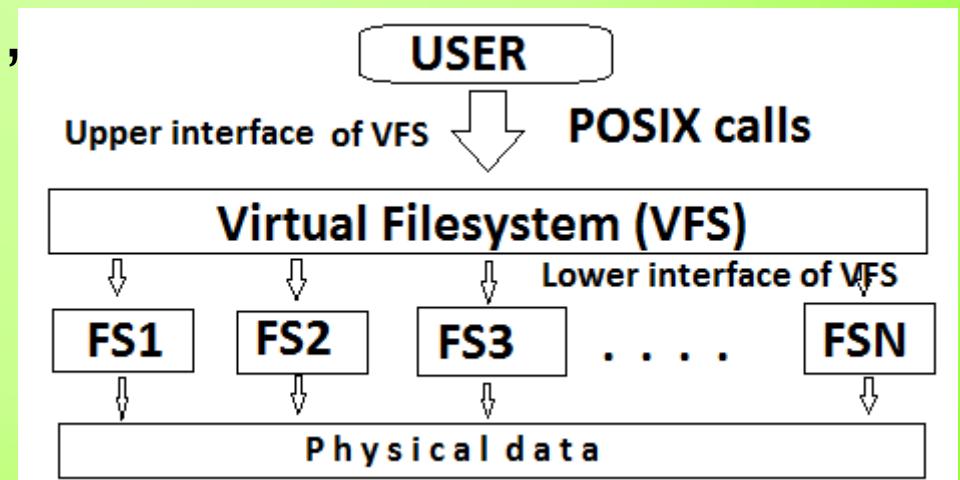
- ▶ On case of disk fails, power-cut, etc. The file system may be inconsistent, the storage content will be inaccessible.
- ▶ JFS often called as an LFS (Log-structured File System) too.
- ▶ As a database system, LFS based on a transaction system: task + log
 - On case of failure, the LFS repairs itself, file system check can run automatically too.
 - Log is stored too(sometimes on other artitions, disks)
- ▶ It needs more CPU load- but gives more safety!

Filesystem support

- ▶ Recent operating systems supports a lot of file systems.
 - E.g.: Linux 2.6 kernel supports more than 50.
- ▶ Mounting filesystems
 - Mount command, mounts all given in /etc/filesystems, during boot
 - Mount automatically (eg. USB drive)
 - Mount manually (Linux, mount parancs)
- ▶ Windows: Drive letters
 - A,B,C,...
- ▶ UNIX: uniform filesystem, one entry, /

Parallel usage of several filesystems on the same system

- ▶ Many different filesystems may be used on the same computer parallel
 - E.g. Windows: NTFS, FAT-32, UDF (DVD,CD) etc.
 - The systems uses them separately.
Which one is the actual? The drive letter decides it:
(c:, a:)
 - E.g.: UNIX: ext2, ext3,
 - The modern UNIX systems try to integrate them.
They use a virtual filesystem (VFS) –



Application- Disk relation

▶ Layered architecture

- User level
 - An application can access disk information via system libraries.
 - Text, binary operation
- Operating system level
 - Filesystem implementations
 - Access, rights
 - Volume manager
 - Device driver
 - Based on BIOS
- Hardware level
 - I/O device, IDE, SATA, SAS, etc.

FAT

- ▶ **File Allocation Table**
 - Rather old, but still alive filesystem!
- ▶ **The FAT table acts as a disc allocation map, a bit describe a block allocation, 0–free, 1–reserved**
 - E.g.: Fat12, FDD, Cluster size 12 bits.
 - For safety 2 FAT's
- ▶ **Chained storage**
 - In the directory a file entry contains the file name, date, etc, and the first block index, where file data is stored.
 - The last bytes of block shows to the next block.
 - The last bytes FFF, if no more data in the file.
- ▶ **One directory entry size is fixed, it's equal 32 bytes. (max. 8.3 name, size, last modification date, etc)**
- ▶ **System, Hidden, Archive, Read only, directory attributes**

FAT properties

- ▶ FAT16, 16 bits cluster description, 4 byte (2x2) the first block address
 - Max. partition size: 4 GB (using 64kb blocks), using 32kb and less blocks, the max size is 2 GB.
 - Reserved directory blocks (on FDD this is the track 0)
 - On FDD 512 directory entry
 - On HDD 32736 directory entry(16 bits, signed)
- ▶ FAT32 (from 1996)
 - 28 bits cluster size
 - File size maximum is 4 (2) GB.
 - 2 TB partition size (512 byte block size)
- ▶ 32MB-ig, 1 block = 1 sector(512bájt)
 - 64 MB, 1 block=1KB (2 sector), 128MB, 1 block=2KB
 - 1 block max. 64 KB lehet.
- ▶ Long file name support
- ▶ Defragmentation necessary.

UNIX directory structure

- ▶ Indexable directory structure
- ▶ After MBR, after partition boot sector follows a partition superblock
- ▶ Next is the free-reserved allocation part.
- ▶ i-node table, including root inode table
- ▶ Modular, lots of many table, quick access to file entry, this is the directory catalog structure.
- ▶ A file pointed by an i-node entry!
 - An i-node table entry contains 15 fields, the first 12 points to the file blocks.
 - If it is not enough, the 13. field points to a next i-node, that again +15 fields.
 - If it is not enough, the 14 field points to a next i-node, that again +15 fields, and repeat the first i-node architecture.

i-node chaining example

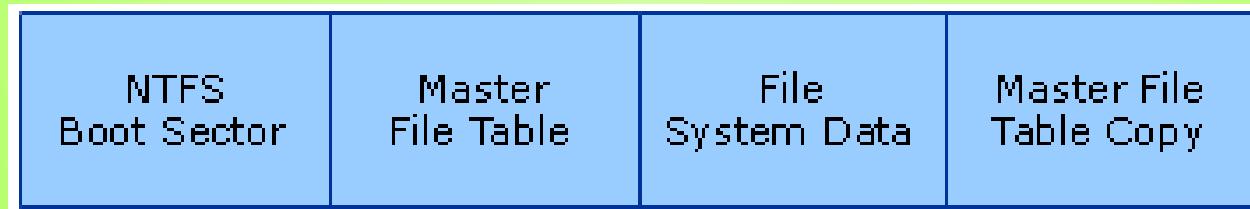
Root directory		The /usr directory i-node number 6	The 132. block the block of /usr directory	The /usr/ast directory i-node number 26	The 406. block is the block of /usr/ast directory
1	.				
1	..				
4	bin				
7	dev				
14	lib				
9	etc				
6	usr				
8	tmp				
Searching /usr gives the 6. i-node		In the 6. i-node there is the fact that the /usr is in the 132. block		In the 26. i-node there is: the /usr/ast is in the 406. block	The /usr/ast/mbox file i-node number is 60
		Mode size times	6 • 1 .. 19 dick 30 erik 51 jim 26 ast 45 bal	Mode size times	26 • 6 .. 64 grants 92 books 60 mbox 81 minix 17 src
		132		406	

NTFS

- ▶ New Technology File System
 - FAT-NTFS efficiency border: appr. 400 MB.
- ▶ A filename length max. is 255 chars, 8+3 secondary name
- ▶ Fine tuning security settings
- ▶ Defragmentation also necessary.
- ▶ Encrypted filesystem, eventing subsystem
- ▶ POSIX support
 - Hard link (fsutil command), time stamps, case sensitive (sometimes, sometimes not😊)
- ▶ Compressed file, directory, user quote support
- ▶ NTFS is cluster based, default 4kb, so 1 cluster is 8 sectors (8*512byte)

NTFS partition architecture

- ▶ A Master File Table is a table.
- ▶ A Files System Data too.



NTFS partition Boot sector

- ▶ Boot sector
 - JMP +0x52 (EB 52)
 - OEMID (8 byte, MSWINx.y)
 - BPB (Bios Parameter Block)
 - Bytes per sector (512)
 - Sectors per cluster (8)
 - Extended BPB
 - Total Sector number (8 byte-on tárolva)
 - LCN – Logical Cluster Number for MFT
 - Volume serial number
 - Loader code (loads ntldr.dll, and ntfs.sys,ntoskrnl.exe)
 - Sector end(0xAA55)

MFT

- ▶ NTFS partition begins with an MFT (Master File Table)
 - 16 attributes reserved for a file entry.
 - Each attributes is max. 1kb. If it is not enough, the last attr. Points to the next entry.
 - Each data is an attribut, an entry can contain more data. (eg: preview picture)
 - Logical file size max is 2^{64} byte
 - If file size is < 1kb, so the file content in the attribut, direct file.
 - No file size maximum.

NTFS table details

File
0 \$Mft - MFT
1 \$MftMirr - MFT mirror
2 \$LogFile - Log file
3 \$Volume - Volume file
4 \$AttrDef - Attribute definition table
5 \ - Root directory
6 \$Bitmap - Volume cluster allocation file
7 \$Boot - Boot sector
8 \$BadClus - Bad-cluster file
9 \$Secure - Security settings file
10 \$UpCase - Uppercase character mapping
11 \$Extend - Extended metadata directory
12 Unused
15 Unused
16 User files and directories

Reserved for NTFS
metadata files

**Thanks for your
attention!**

zoltan.illes@elte.hu

Operating systems

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

We have talked earlier...

- ▶ **Progress of Operating systems**
 - Computer – Op.system generations
- ▶ **Notions of Operating system, structures**
 - Client–server model, ...
 - System calls
- ▶ **Files, directories, filesystems**
 - Hardware architecture
 - Logical architecture
 - FAT, UNIX, NTFS,...

What comes today...

- ▶ **Processes**
 - Creating, ending
 - States of processes
- ▶ **Process communication**
 - Race situation
 - semaphores, mutexes, monitors
- ▶ **Classical IPC problems**

The model of processes

- ▶ Program – process differences
- ▶ A process is a running program in the memory (code+I/O data+state)
- ▶ How many processes are working at the same time?
 - Single Task – Multi Task
 - Real Multi Task?
- ▶ Sequential model
- ▶ Swithing between processes: multiprogramozás
- ▶ At the same time there is only 1 active process.
 - In case of more cores there are more active process

System model

- ▶ 1 processor + 1 system memory + 1 I/O device = 1 task execution
- ▶ In Interactive (windowing) systems, there are more than one process in the memory
 - Environment switching: Only the program located in foreground is running.
 - Cooperative system: the current process periodically offers the cpu control to other processes. Ask a question: Anybody wants the CPU? (Win 3.1)
 - Preemptive system: the kernel takes back the cpu after a predefined time, and gives it to the next process.
 - Real time systems

Creating processes

- ▶ Recently there are used preemptive systems (Linux, Windows)
- ▶ There are more than one active, living process.
- ▶ Reasons of process creation:
 - System initialisation
 - A process creator system call
 - Copy of origin process(fork)
 - Exchange of the origin(execve)
 - A user command (command&)
 - Etc.
- ▶ There are foreground processes
- ▶ Background processes (daemons)

Process relations

- ▶ Parent – child relation
- ▶ Process tree:
 - Every process has a parent
 - A process can have more than one child
 - Process subtree
 - E.g.: Init, /etc/rc script execution
 - The init id is 1.
 - Fork command...Be care using it
- ▶ Reincarnation server
 - Some services has such a feature.
 - In case of death of a process, the control service (reincarnation server) creates a new one instead of death process.

End of processes

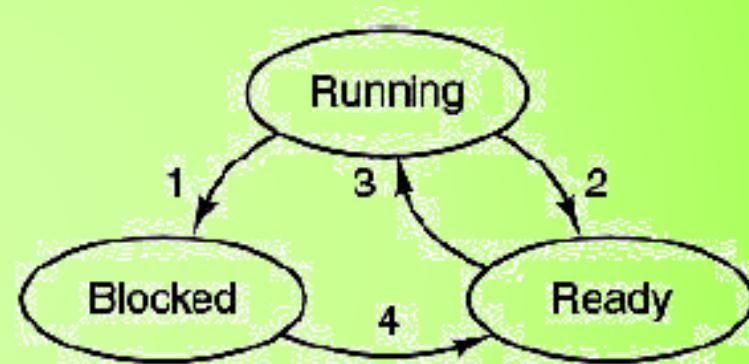
- ▶ After starting, the process finishes his task within a given time frame.
- ▶ Reasons of a process end:
- ▶ Voluntary end
 - Normal process end (exit, return commands.)
 - On error, the program ends his run(also execute a return command)
- ▶ Unintended end
 - Illegal command causes a process end, (divide by 0, illegal memory usage, etc)
 - Outside „partner” helps in finishing. Other process sends a signal.

State of processes

- ▶ A process: a program in the memory with instruction pointer, stack, etc.
- ▶ The processes usually are not independent.
 - Some process uses other results.
- ▶ A process can be one of the next 3 states:
 - Runing
 - Ready to run, temporary it waits for a CPU time frame.
 - Blocked , it can not continue his execution, eg the process needs an other process result to run! (cat Fradi.txt|grep Fradi|sort, grep and sort are blocked in the beginning...)

State transitions

1. Running -> Blocked
 - Have to wait something
2. Running -> Ready to Run
3. Ready to Run -> Running
 - Scheduler makes decision, the processes do not know about it.
4. Blocked->Ready to Run
Waiting data is arrived



Implementation of processes

- ▶ A CPU executes only the actual command, pointed by CS:IP
- ▶ Only one process is running at one time.
- ▶ The CPU does not know about processes:
 - During process change (switch) what can we store to properly continue the original process?
 - Mostly everything,...execution counter, registers, memory state, file descriptors, etc.
 - These informations are stored in the process table(process control block, CPU helps to manage this, State segment, etc)
- ▶ I/O interrupt table

When and how to change a process?

- ▶ When: During timer interrupt, special system call etc.
- ▶ Scheduler saves process information into process table.
- ▶ Load the next process state, and run it! (Set CPU registers back, etc).
- ▶ Can not save cache store
 - Often change – needs more resources
 - The perfect changing time interval is not an exact value (not obvious).

Process Control Block (PCB)

- ▶ It is build during system initialization
 - Contains process informations
- ▶ This is like an array (based on PID) – each array element a composed structure containing process parameters.
- ▶ Major parameters of a process:
 - Process IDentifire (PID), name (name og program)
 - Owner, Group ID
 - Memory occupation, register data
 - etc.

Threads

- ▶ General situation: One process – one instruction set – one thread
- ▶ Sometimes we need to create inside a process more threads.
 - Thread: It is a standalone instruction set (typically a function) inside a process.
 - Often called: „lightweight process”
- ▶ Threads: Independent instruction sets.
 - Inside a process can be only one
 - Inside a process can be more–If one of them is blocked, the complete process will be blocked!
 - Thread table – for keeping a record of threads
- ▶ The thread has the same memory space as his parent!

Process-Thread properties

- ▶ Only a process has:
 - Address space
 - Global variables
 - File descriptors
 - Child processes
 - Signal handling, wake up listeners
 - ...
- ▶ A thread also has:
 - Process counters (instruction pointer, etc.)
 - Registers copy, own stack

Thread problems

- ▶ Be careful using fork – If a parent process has more threads, the child also will have!
- ▶ File management – What to do if a thread closes a file while an another thread uses it?
- ▶ Error management – errno is a global variable
- ▶ Memory management – ...
- ▶ Relevant: A system call can handles threads (thread safe call)

Communication of processes

- ▶ IPC – Inter Process Communication
- ▶ We have three major problem during process collaboration:
 - A process cannot be interrupted during execution a „critical, sensitive instruction block”.
 - A serial execution of subtasks (joining). (We can print only after collecting printing information!)
 - How can a process send any information to an another one?
- ▶ All three problems fits to threads too, but the third one is not causes any problem because of same memory space!

Parallel systems

- ▶ Scheduler switches processes quickly, so it gives a „parallel” execution feeling.
- ▶ Multiprocessor systems
 - 2 or more processors
 - Higher computing power
 - Do not increase reliability
 - Clusters – Hardware parallelism
 - It is to increase reliability!
- ▶ Key question: How can we use common resources?

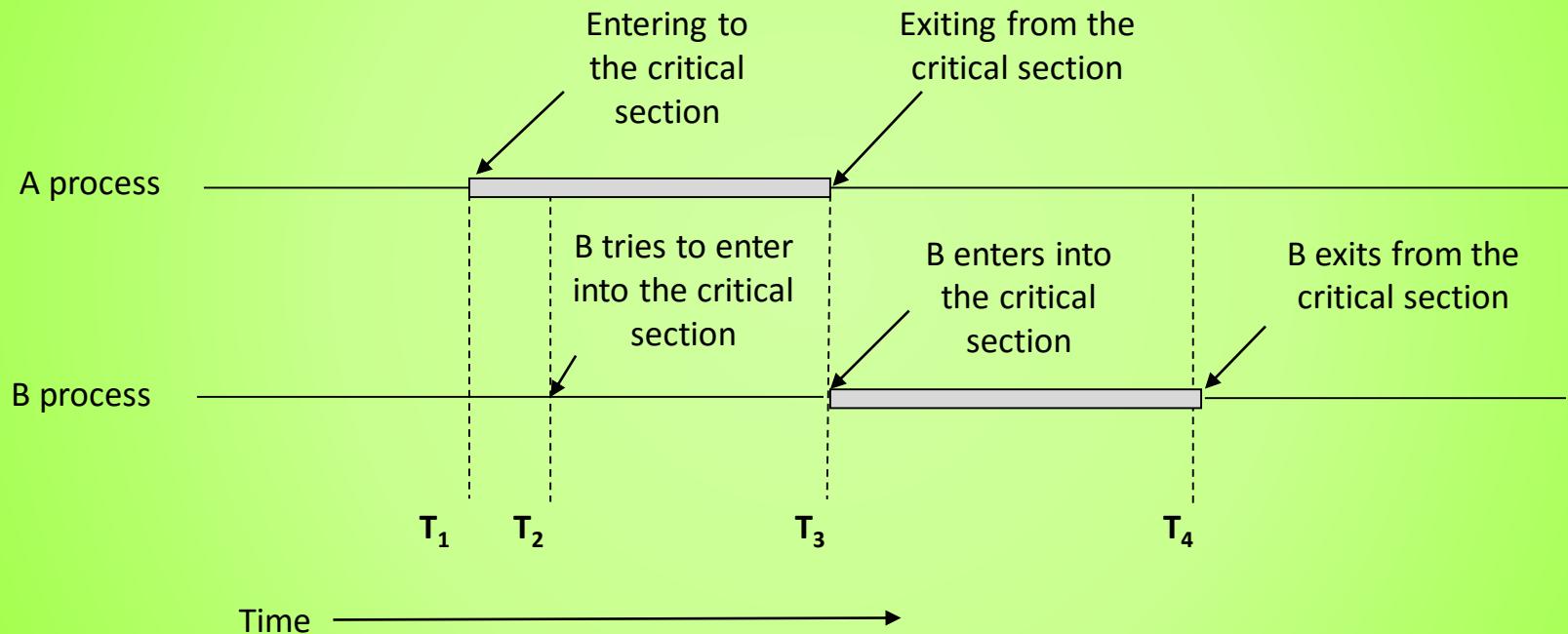
Common resources

- ▶ When 2 or processes want to use the same memory...
 - E.g.: 2 printing task, common printing queue
- ▶ Race condition – place: 2 or more processes wants to read/write common memory space, the result often is undetermined, it's related to the time moment!
 - This error is hard to discover often!
- ▶ Solution: We need a method which guarantees the common resource access only by 1 process at the same time!

Mutual exclusion

- ▶ Critical section: That code part which works with a resource(memory) and can be accessed by several processes.
- ▶ A good mutual exclusion suits the following requirements:
 - No two processes in the critical section at the same time.
 - It is not related to speed, CPU or other parameter.
 - Any process, outside the critical section, can not block another process.
 - A process does not wait forever to enter to the critical section.

The required behavior of the mutual exclusion



Mutual exclusion implementations I.

- ▶ Disabling interrupts (all)
 - After entering crit. section interrupts will be disabled (STI– set interrupt flag to 1)
 - On exit: Enable interrupts (CLI– Clear interrupt)
 - It is not perfect because a user level program can not disable interrupts.
- ▶ Common 2 state(enable-disable) variable
 - 0 (free) and 1 (reserved) critical section
 - 2 process might be in critical section!
 - 1 process enters, and before setting this variable to 1, kernel changes the running process!!.

Mutual exclusion implementations II.

▶ Strict changing:

- We can implement it for more than 2 processes.
- It does not grant the mutual exclusion requirement no.3.(Outside process cannot block an another process)! If ha process 1 is slow, in non critical section, and process0 quickly enters into critical section and leaves that, and finish its non critical section too, then process0 will be blocked by process1!
- Process0

Process1

```
while(1)
{
    while(next!=0) ;
    critical_section();
    next=1;
    not_critical_section();
}
```

```
while(1)
{
    while(next!=1) ;
    critical_section();
    next=0;
    not_critical_section();
}
```

G.L.Peterson's solution

- ▶ 1981, an improvement of strict changing
- ▶ Before critical section every process must call an enter function and after it call an exiting function.

```
#define N 2
int next;
int wants[N];
/* the modified process*/
while(1)
{
    enter_func(process);
    critical_section();
    exit_func(process);
    not_critical_section();
}
```

```
void enter_func (int proc)
{
    int other;
    other = 1 - proc; //N=2 so other proc is
                      // other =(proc+1) % N;
    wants [proc]=1; //proc 1 wants to run
    next =proc;
    while(next ==proc &&
          wants [other]);
}
void exit_func (int proc)
{
    wants [proc]=0; //outside from crit.
}
```

Little Peterson „modification” – big mistake

- ▶ Lets proc=0!
- ▶ At the signed line the scheduler changes, so proc=1 now!
- ▶ The wants[0] is 0, therefore process 1 enters into critical section!
- ▶ While process 1 is working in critical section, lets the scheduler changes running process again, now back to process 0! In this case the variable „next” value is 1, proc is 0 so the next==proc will fail therefore the process 0 also enters into critical section!

```
void enter_proc(int proc)
{
    int next;
    next=1-proc; //so N=2...
    // next=(proc+1) % N;
    next=proc; //2 lines change
    /* here changes the scheduler
    wants[proc]=1;//proc wants run
    while( next==proc &&
          wants[masik]);
}
void exit_func(int proc)
{
    wants[proc]=0; //false
}
```

Busy waiting, machine code

- ▶ TSL instruction – Test and Set Lock
 - Atomic instruction (it can not interrupt)

enter:

```
    TSL register, LOCK      ; LOCK value stored in the
                            ; register
                            ; and LOCK=1
                            ; during TSL,the CPU locks
                            ; memory address bus
    cmp register,0
    jne enter
    ret
;
```

exit:

```
    mov LOCK,0
    ret
```

Busy waiting : summary

- ▶ Both (Peterson and the TSL) solution works well, only they are waiting in a cycle.
- ▶ These solutions uses an „empty waiting cycle” – this method is called: Busy waiting!
- ▶ It wastes CPU time...
- ▶ It would be better to use a process blocking method instead of busy waiting! During process block, it do not use the CPU!!

Sleep – wakeup

- ▶ Busy waiting solution is not so effective!
- ▶ New solution: Using a block(sleep) method instead of busy waiting, and use a wake up method if needed!
 - sleep –wakeup, down–up keywords
 - These functions may have parameters too!
 - Tipical problem: Producer – Consumer problem

Producer-Consumer problem

- ▶ Well known problem as a confine (ranged) variable problem too.
- ▶ E.g: Baker–Bakery–Consumer „triangle”.
 - Baker is baking bread, until in bakery there is empty shelves(space).
 - A customer, consumer can buy a bread if bakery is not empty.(There is minimum 1 bread...)
 - If the bakery is full, than baker goes to sleep (rest) .
 - If the bakery is empty, than the customer will wait(sleep) for bread.

A Solution of the producer-consumer problem

► Baker(producer) Customer (consumer)

```
#define N 100
int place=0;
void baker()
{
    int bread;
    while(1)
    {
        bred=new_bread()
        if (place==N) sleep();
        to_shelf(bread);
        place++;
        if (place==1)
            wake_up(customer);
    }
}
```

```
void customer()
{
    int bread;
    while(1)
    {
        if (place==0) sleep();
        bread=ask_bread();
        place--;
        if (place==N-1)
            wake_up(baker);
        eat(bread);
    }
}
```

Baker-Customer problem

- ▶ The access of variable „place” is not restricted, this will cause a race situation.
 - The customer checks place, and the content is 0, after this checking the scheduler changes the running process. The baker increases the variable place, and the content will be 1. In this case baker sends a wakeup signal to customer. This signal will be lost, because the customer is not sleeping!
 - The customer gets back running state, and will sleep because he thinks the variable place is 0.
 - The baker will bake all N bread and he also will sleep!
 - So now both processes are sleeping!(deadlock)
- ▶ We can use a wakeup bit too, but the problem will remain!

Semaphores I.

- ▶ Suggested by E.W. Dijkstra (1965), this is a special „variable” type.
- ▶ It is an integer variable.
- ▶ A semaphore stops the process, if its value is 0.
 - The process will wait,sleep, will not use CPU, it stops before semaphore.
- ▶ If the semaphore >0 , the process can enter into critical section.
- ▶ The semaphore has 2 operation:
 - Down: Entering into critical section the value will be decreased.
 - Up: Exiting from the critical section the value will be increased!
 - These operations was called as P and V by Dijkstra!

Semaphores II.

- ▶ Atomic instruction: a semaphore value checking, modification, it can not be interrupted!
- ▶ This guarantees to avoid race situation.
- ▶ If a semaphore has only 2 values, the value 1 means the process (train) can continue his work, the 0 means the process (train) will stop!
 - Binary semaphore
 - This often called as MUTEX (Mutual Exclusion).

semaphore implementations

- ▶ Up, Down atomic instructions.
 - It can not be interrupted!
- ▶ How it is supported?
 - With kernel level system call, from user level can not.
 - At the begining of system call all interrupt will be disabled.
- ▶ semaphore instructions are in kernel level!
- ▶ It is accessed from development environment, supported by operating systems!

A Solution of the producer-consumer problem with semaphores I.

▶ Producer (baker) function

```
typedef int semaphore;
semaphore free=1; /*Binary semaphore,1 may go on, free sign*/
semaphore empty=N, tele=0; /* empty the shelf, this is a free sign*/
void baker()           /* N value „the size of bread-shelf” */
{
    int bread;
    while (1)
    {
        bread=baker_bake();
        down(&empty);          /* empty decrease, if before >0, go on*/
        down(&free);           /* May we change the shelf of bakery? */
        bread_to_shelf(bred); /* Yes, put on a bread*/
        up(&free);             /* Release the shelf of the bakery. */
        up(&full);             /* Sign to the customer, there is a bread. */
    }
}
```

A Solution of the producer-consumer problem with semaphors II.

► Consumer (customer) function.

```
void customer()          /* customer semaphore is full */  
{  
    int bread;  
    while (1)  
    {  
        down(&full);      /*full is decreased, if before >0, go on */  
        down(&free); /*May we modify the shelf of the bakery? */  
        bread=bread_polcról(); /* Yes, take away a bread. */  
        up(&free);       /* Release the shelf of bakery. */  
        up(&empty);      /* Sign to the baker, there is empty place,  
you may bake! */  
        bread_eating(bread);  
    }  
}
```

Summary of semaphore sample

- ▶ free: it safes the bread shelves, so it is accessed only by 1 process(the baker or the customer)
 - Mutual exclusion
 - Atomic instructions(up, down)
- ▶ full, empty semaphore: syncronisation semaphors, the producer stops if the shelves are full, or the consumer stops is the shop is empty.

semaphore functions,sample

► Unix environment:

- semget: creating semaphor(System V)
- semctl: semaphore control, read, write value
- semop: semaphore operation (up,down)
- sembuf structure
- On practice, you'll see samples!
- sem_open,sem_wait,sem_post,sem_unlink (Posix)

► C# sample.

- VS 2008.
- The implementation of baker–customer sample.

**Thanks for your
attention!**

zoltan.illes@elte.hu

Operating systems

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

We have talked earlier...

- ▶ **Progress of Operating systems**
 - Computer – Op.system generations
- ▶ **Notions of Operating system, structures**
 - Client–server model, system call
- ▶ **Files,directories, filesystems**
 - Hardware architecture
 - Logical architecture
 - FAT, UNIX, NTFS,...
- ▶ **Processes, process communications**
 - Race conditions, critical sectionmodel

What comes today...

- ▶ **Process communication**
 - Monitors
 - Message sending
- ▶ **Classical IPC problems**
 - Case of eating philosophers
- ▶ **Scheduling of processes**
 - Theories, implementations
 - Thread scheduling

Is there any problem with semaphores?

- ▶ No...but small mistyping might cause big problems.
 - E.g. Let's change 2 rows (red), when bakery is full, the baker is stopped by semaphor empty, at the same time the customer is stopped by free, therefore both processes are stopped! (deadlock)

```
void baker() /*original recipe
*/
{
    int bread;
    while (1)
    {
        bread=baker_bake();
        down(&empty);

        down(&free);
        to_shelf(bread);
        up(&free);
        up(&full);
    }
}
```

```
void baker() /*modified recipe*/
{
    int bread;
    while (1)
    {
        bread=baker_bake();
        down(&free); /*changed part*/
        down(&empty);
        to_shelf(bread);
        up(&free);
        up(&full);
    }
}
```

Do we have any problem with semaphors?

- ▶ Generally no! But as we saw, a little mistake causes program error! (Big error!).
- ▶ Also the changing two up and down instructions can cause same problem!
- ▶ Other reason: if we forget it...
- ▶ Do we have any better thing to do?
 - At Kernel level no.

Monitors

- ▶ Brinch Hansen (1973), Charles Anthony Richard Hoare (1974) suggested a higher level construction supported by languages.
- ▶ This construction is called: monitor
 - It is similar to a class definitions

```
Monitor critical_zone
    Integer shelf[];
    Condition c;
    Procedure baker(x);
    ...
    End;
    Procedure customer(x);
    ...
    End;
End monitor;
```

Monitor properties

- ▶ We can define inside a monitor procedures, data structures.
- ▶ At the same time only 1 process can enter into a monitor!
- ▶ This is supported by programming language compiler.
 - If a process calls a method inside a monitor, first it checks whether there is any other active process inside the monitor?
 - If yes, the process will be held, suspended.
 - If not, the process enters into the monitor.

Monitor implementation

- ▶ Using Mutex
- ▶ A customer has no concrete idea about implementations, but it is not needed.
- ▶ Result: a more safety mutual exclusion
- ▶ Only we have one problem: what about if a process can't continue his work inside a monitor?
 - E.g: a baker can't bake bread, because the bakery shelves are full!
- ▶ Solution: lets create a condition variable
 - We can do 2 operation on condition variable: wait, signal

A Solution of the producer-consumer problem with monitors I.

- ▶ N element

```
monitor Baker-Customer
    condition full, empty;
    int number;
    bread_to_shelf(bread elem)
    {
        if (number==N) wait(full);
        shelf(elem);
        number++;
        if (number==1) signal(empty);
    }
    bread bread_from_shelf()
    {
        if (number==0) wait(empty);
        bread elem=bread_from_shelf();
        number--;
        if (number==N-1) signal(full);
        return elem;
    }
end monitor
```

The process of Pék&Vásárló

```
baker()
{
    while(1)
    {
        bread new;
        new=bread_bake();
        Baker-Customer.bread_to_shelf (new);
    }
}
customer()
{
    while(1)
    {
        bread new_bread;
        new_bread=Baker-Customer.bread_from_shelf();
        eat(new_bread);
    }
}
```

Other solutions

- ▶ The previous example based on a theoretical language: Pidgin Pascal
- ▶ C has not monitor
 - C++ has monitor, also wait, notify
- ▶ Java:
 - Synchronized methods
 - No condition variable, but there are wait & notify
- ▶ C#
 - Monitor class
 - Enter,TryEnter,Exit,Wait,Pulse (this same as notify)
 - Lock keyword
 - Sample: VS2008 solution, monitor project.

Do we have any problem with monitors?

- ▶ Hm,...no, we have not!
- ▶ It is more safety than a simple semaphor.
- ▶ It works with one or more CPU, but only with a commom memory space!
- ▶ If a CPU has a standalone memory this solution does not fit, does not work!

Message sending

- ▶ Processes typically uses two general method:
 - Send(to, message)
 - Receive(from, message)
 - Forrás tetszőleges is lehet!
- ▶ These are system calls and not language constructions!
- ▶ Receipt message: If a sender wants to know about accepting message, the receiver process sends back a receipt!
 - If a sender does not get the receipt, it will send again the message!
 - If the receipt will be lost, the sender also will send again!
 - A message contain a number which holds the message number too to distinguish(mark) messages .

A Solution of the producer-consumer problem with message sending. I.

► The producer (baker) process:

```
#define N 100          // size of shelves
void baker()          // baker process
{
    int bread;        // „bread” variable, storage
    message m;        // message storage place
    while(1) // baker works all time
    {
        bread= bake_bread();
        receive(customer,m); // we wait a message from
                               //the customer
                               // empty message in m
        m=create_message(bread);
        send(customer,m); // send the bread to the customer
    }
}
```

A Solution of the producer-consumer problem with message sending. II.

► Consumer process :

```
void customer()          // consumer (customer) method
{
    int bread;           // „bread” variable
    message m;           // the place for the message
    int l;
    for(i=0;i<N;i++)
        send(baker,m);
                    //setup N empty space for bred
                    //sending to baker
    while(1) // the shopping is going on all the time too
    {
        receive(baker,m);      // buying a bread
        bread=message_outpack(m);
        send(baker,m); // sending back empty store
        eat_bread (bread);
    }
}
```

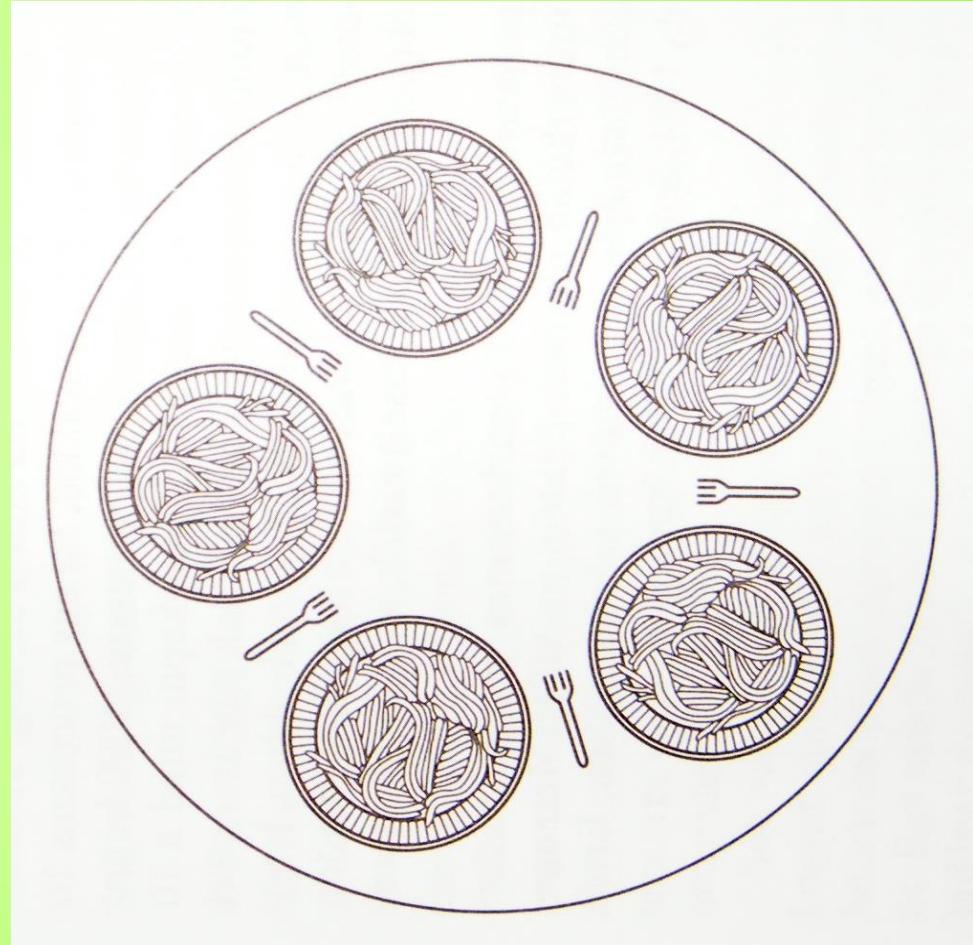
Summary of message sending

- ▶ We have to create temp place for messages on both places.(like mail store)
- ▶ We can omit this, in this case the receive statement will be blocked before send command!
 - Trysting strategy (randevú stratégia).
 - Minix 3 also uses this one!
 - Messages are same as pipes, but inside pipes there is not used any delimiter!
- ▶ The message sending is a general technic in parallel systems.

Classic IPC problems I.

▶ Eating philosophers case:

- They need 2 forks to eat spaghetti!
- All philosophers can access only forks near his own plate.
- They are thinking-eating...
- Lets create a program to solve this problem!



Solution I.

- ▶ It has a problem, all processes can be in deadlock if all gets the lefts fork and will wait for the rigth one!.
- ▶ If somebody puts back left fork and again try to get both also not good, because all processes do this!
(Starvation)

```
Void philosopher (int i)
{
    while(1)
    {
        thinking();
        need_fork(i); // left fork
        need_fork ((i+1)%N); //right
        eating();
        dont_need_fork (i);
        dont_need_fork ((i+1)%N);
    }
}
```

Solution II.

```
int s[5];           // values: eat, hungry, thinking
semaphore safe_s=1; //a sign for using s array
semaphore philo[5]={0,0,0,0,0}; //1 semaphore to each philosopher, 0=not
permitted
Void philosopher(int i)
{
    while(1) {
        thinking();
        down(safe_s); //only i modify s[]
        s[i]=hungry; //
        if (s[left]!=eat && s[right]!=eat) //whether the two neighbour's fork are free?
            { s[i]=eat; up(philo[i]); }; //i eat, philo[i] shows free
        up(safe_s); // other may access s[]
        down(philo[i]);
    // it is blocked, if there is no 2 forks, if the i. Philosopher is not eating
    spaghetti_eating();
    down(safe_s); // finished eating, again protect s because i modify it
    s[i]=thinking;
    if (s[left]==hungry && s[left--]!=eat) { s[left]=eat;up(philo[left]);}
    if (s[right]==hungry && s[right++]!=eat) { s[right]=eat;up(philo[right]);}
    up(safe_s);
    }
}
```

Solution III.

- ▶ Lets 5 forks semaphore[] for each fork.
- ▶ Max semaphore
- ▶ Sample to access restricted resources.

```
Int N=5;
semaphore forks[]={1,1,1,1,1}; //all free
semaphore max=4; //max 4 fork used in
the same time
void philozophus(int i)
{
    while(1)
    {
        thinking();
        down(max);
        down(forks[i]); // left fork
        down(forks[(i+1)%N]); //right
        eating();
        up(forks[i]);
        up(forks[(i+1)%N]);
        up(max);
    }
}
```

Readers–Writer problem

- ▶ The database can read by more than 1 process, but only 1 can write:

```
// writing process
semaphore database=1;
semaphore mutex=1;
int rc=0;
Void writer()
{
    while(1)
    {
        do_something();
        down(database); // critical
        write_to_database();
        up(database);
    }
}
```

```
Void reader()
{
    while(1)
    {
        down(mutex);
        rc++;
        if (rc==1) down(database);
        up(mutex);
        Read_from_database();
        down(mutex); // critical
        rc--;
        if (rc==0) up(database);
        up(mutex);
        working_on_data();
    }
}
```

Scheduling

- ▶ We have seen before that several process may run „parallel” to each other.
- ▶ Only 1 at a time can run.
- ▶ Which one should run?
- ▶ The scheduler makes the decision
- ▶ The decision is based on the scheduling algorithm

The I/O need of processes

- ▶ Typically a process does two types of work:
 - It is counting...
 - I/O need, wants to read or write data from/to the periphery
- ▶ CPU-bound process
 - It is working (count) a lot, it does not wait too much for I/O
- ▶ I/O-bound process
 - Works only for a small amount of time it is waiting for I/O long

When do we have to switch to a new process?

- ▶ There is a process (context) switch:
 - If a process ended
 - If a process' state becomes blocked (because I/O or semaphore)
- ▶ Usually there is a context switch:
 - A new process is created
 - I/O interrupt occurs
 - After an I/O interrupt, a blocked process, the process which waited for it typically may continue its running.
 - Timer interrupt
 - Preemptive scheduling
 - Non-preemptive scheduling

Groups of scheduling

- ▶ Representatives for each system:
 - Fairness – everybody can use CPU
 - The same rules are valid for everybody
 - Balance –Everybody should get the same loading
- ▶ Batched systems
 - throughput, turnaround time, CPU utilization
- ▶ Interactive systems
 - Response time, proportionality to the user's expectations
- ▶ Real-time systems
 - To keep deadlines, to avoid data loss, quality corruption

Scheduling in batch systems I.

- ▶ Ranking scheduling, it is non-preemptive
 - First Come First Served – (FCFS)
 - The process runs till it stops or becomes blocked.
 - If it blocks, it goes to the end of the queue.
 - The processes are in a fair, simple, chained list.
 - Disadvantage: I/O-bound processes are very slow.
- ▶ The shortest job runs first, non-preemptive (SJF)
 - We have to know the running time ahead
 - It is optimal, if everybody is known at the beginning

Scheduling in batch systems II.

- ▶ The process with the shortest remaining time runs
 - Preemptive, monitoring at each entering.
- ▶ Three level scheduling
 - Inlet scheduler
 - It lets the task into the memory in rotation.
 - Disk scheduler
 - If it lets in to much processes and memory becomes full then they all have to be written to the disk and back.
 - It runs rarely.
 - CPU scheduler
 - We may choose from the previously mentioned algorithms.

Scheduling in interactive systems I.

► Round Robin

- Everybody gets a time quantum, at the end of it or in the case of blocking comes the following process
- At the end of the quantum the next process in the round list will be the actual
- Fair, simple
- We may store the processes (features) in a list and we go round and round all the time.
- There is only one question: How big should be the quantum?
 - The process switch needs some amount of time.
 - Small quantum-> a lot of CPU time for switching
 - Too big quantum -> the interactive user feels it too slow (keyboard handling)

Scheduling in interactive systems II.

▶ Priority scheduling

- Importance, priority is shown
 - Unix: 0–49 → not preemptive (kernel) priority
 - 50–127 → user priority
- The process with the highest priority may run
 - Modifying: dynamical priority to avoid starvation
- Usage of priority classes
 - Usage of Round Robin among the same class processes
 - Must modify the priority of the processes, because low priority processes get CPU rarely.
 - Typically at each 100. quantum the priorities are recounted
 - Typically the high priority processes become lower level, then comes RR. At last the original classes will be built up again.

Scheduling in interactive systems III.

- ▶ Multiple queues
 - Also have priorities and uses RR
 - At the highest level each quantum gets 1 time quantum.
 - The next 2, then 4, 8, 16,32,64.
 - If the time of the highest process is used up it go down with one level.
- ▶ The shortest process next
 - Though we do not know the remaining execution time we have to estimate it!
 - Aging, weighted average for the time quantum.
 - $T_0, T_0/2+T_1/2, T_0/4+T_1/4+T_2/2,$
 - $T_0/8+T_1/8+T_2/4+T_3/2$

Scheduling in interactive systems IV.

- ▶ Guaranteed Scheduling
 - Each active process gets proportional CPU time.
 - Must be registered how many time was used up for a process and if somebody has got less time it „goes” forward.
- ▶ Lottery scheduling
 - Similar to the previous one, except the processes gets some lottery ticket. That process can run which has got the pulled out one.
 - It is easy to support proportional CPU time, useful e.g. at video servers
- ▶ Fair-share scheduling
 - Let's pay attention to the user as well! Similar to guaranteed one – only it refers to the users

Scheduling in real-time systems I.

- ▶ What is a real-time system?
 - The time is an important actor. We must guarantee to give a response within the deadline.
 - Hard Real Time, absolute, not modifiable deadlines.
 - Soft Real Time (tolerant), there are deadlines, but a small difference is tolerable.
 - The programs are divided into several smaller processes.
 - In the case of an outer event, have to give a response within the deadline.
 - Schedulable: if the response CPU time sum of n events is ≤ 1 .
- ▶ Unix, Windows are real-time systems?

Scheduling ideas, implementation

- ▶ Frequently there are child processes in the system.
- ▶ It is not sure that the priority of the parent and the child process must be the same.
- ▶ Typically the kernel uses priority scheduling (+RR)
 - It warrants a system call with which the parent may give the priority of the child
 - Kernel schedules – the user process modify the priority (nice)

Threadscheduling

► User level threads

- The Kernel does not know about it, the process gets a quantum, within the thread scheduler makes the decision who should run
- Quick switch between the threads
- It is possible to use application dependent thread scheduling

► Kernel level threads

- Kernel knows the threads, Kernel decides which process which thread should run.
- Slow switch, between the switch of two threads needs a full context switch
- This also noticed.

**Thanks for your
attention!**

zoltan.illes@elte.hu

Operating systems

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

We have talked earlier...

- ▶ Evolution of operating systems
- ▶ Notions of Operating system, structures
- ▶ Files,directories, filesystems
- ▶ Processes
 - process communications
 - critical sections, , semaphores
- ▶ Classical IPC problems
- ▶ Scheduling

What comes today...

- ▶ **Controlling input/output devices**
 - Type of I/O devices
 - Access of devices
 - Interrupt
 - DMA
 - I/O ports
- ▶ **Base concepts of I/O programs**
- ▶ **Problems of allocating resources**
 - Dead-locks
 - Base concepts
 - Recognize, prevent, avoid

Input–Output devices

► **Block-type devices**

- Store information in a block with given size.
- Block size is between 512 byte– 32768 byte.
- They can be written or read independently from each other.
- Addressable by blocks.
- Such devices are: HDD, CD, magnetic tape device,etc.

► **Character-type devices**

- Not addressable, characters (bytes) come and go one after the other

► **Timer: exception, not block-type and not character-type**

Model of I/O devices

- ▶ Though there are always some exceptions (e.g. timer), the device independent software-model of the operating system is based on this block-type, character-type model.
 - E.g. The filesystem is handling abstract block-type devices.
 - The magnetic tape is a block-type one, at the command to read the N-th block it rewinds back and then forward.
 - The device dependent parts are the device drivers. (DDK)

The speed of I/O devices

- ▶ Keyboard: 10 byte/sec
- ▶ Mouse: 100 byte/sec
- ▶ 56k modem: 7kbyte/sec
- ▶ Scanner: 400kbyte/sec
- ▶ 52xCD ROM: 8 MB/sec (1xCD, 150kb/sec)
- ▶ Firewire : 50 MB/sec
- ▶ USB2: 60 MB/sec
 - USB3 500 MB/sec (theoretical, 4.8GBPS)
- ▶ SATA: 200 MB/sec
- ▶ SCSI UW4: 320 MB/sec
- ▶ PCI bus: 528 MB/sec

Device controllers

- ▶ To connect the I/O device to the computer (system bus) we use device controllers or adapters.
 - Video controller
 - Serial-parallel controller
 - USB controller
 - HDD controller
 - IDE, SATA, SCSI
- ▶ In the case of mainframes I/O is implemented with special I/O machines.
- ▶ CPU-device controller communication
 - Memory-mapped I/O, interrupt, DMA

I/O gate, memory-mapped I/O

- ▶ CPU uses two different ways of data exchange towards the outside world
 - To read, write I/O gates
 - IN register, port; To read the port data into the register (8 or 16 bits long number)
 - OUT port, register ; To write the register to the port
 - The registers, the data area of the I/O device is inside the memory.
- ▶ Maybe there are such environments where
 - Only I/O gates are used. (IBM 360)
 - Only in memories are I/O gates (PDP-11)
 - Memórialeképezésű I/O
 - Mixed (Intel x86, Pentium)

Interrupts I.

- ▶ Interrupt – the notion itself is connected to I/O devices!
 - If an I/O devices is ready for data communication it is indicated with an interrupt signal!
- ▶ Software or Hardware interrupt
 - Software: a machine-code instruction (int x) is executed, in the case of multitask it is called „trap”!
- ▶ There is a number of the interrupt!
 - Interrupt vectors (real mode, from address 0)
 - Interrupt handling routine.
 - INTR, NMI

Interrupt handling

- ▶ Usually devices have a state-bit stating that the data is ready.
 - You may observe it, but it is not the perfect solution.
 - It is a busy-waiting, not efficient, rarely used.
- ▶ The process of interrupt handling (IRQ)
 - The HW device indicates an interrupt request (INTR)
 - The CPU before the execution of next instruction interrupts its activity! (Precise, imprecise)
 - To execute the routine given by an ordinal number.
 - To read the required data and to execute the tightly connected work.
 - To return to the state before the interrupt request.

Priority of interrupts

- ▶ INTR – maskable, non maskable NMI
 - INTR priority of interrupts
 - The same or lower priority interrupts have to wait if they are arrived during the actual interrupt process!
 - NMI – only one is served: the greatest priority one
- ▶ In PC world 15 different interrupts
 - 2x8 channel controller
 - Manual adapter IRQ setting earlier, with jumper, with sw.
 - BIOS automatic interrupt assigning (Plug&Play)

Direct memory access (DMA)

- ▶ Direct Memory Access
 - Contains: a memory address register, a register to indicate the direction of delivery, others to decide the amount of data and to control
 - They can be accessed through standard in,out ports.
- ▶ Typical steps of working:
 1. CPU sets the DMA controller. (Registers)
 2. DMA asks the disk controller to execute the given operation.
 3. After the disk controller reads the data from its buffer the data is read/written to/from the memory through the system-bus..
 4. The disk-controller acknowledge the execution of the request.
 5. DMA indicates with an interrupt that the operation is finished.

I/O software goals I.

▶ Device independence

- To be able to read e.g. a file from a HDD, CD, etc. with the same code, command.
- Standard name usage, the name is a parameter (seals) the actual (real) device.
- Logical mount
 - Unix: Floppy mounted to /home/fdd directory
 - Windows: Floppy mounted to a: name

▶ Error handling

- (should) Has to be handled on a hardware–close level
- Mainly it can be executed here, only in case of fatal errors might be handled on higher levels.

I/O software goals II.

- ▶ Synchronous (blocking), asynchronous (interrupting) transferring mode are supported.
 - In the synchronous mode it is easier to write user programs.
 - Operating system have to support asynchron mode as well.
- ▶ Buffering
 - Everybody use it e.g. keyboard buffer etc.
- ▶ Device usage in distributed or monopole mode
 - The disk can be used by several processes in the same time.
 - A magnetic tape or CD-writer is used in a monopol way.

The structure of I/O software system

- ▶ The structure consists of layers
 - Typically it is organized into 4 layers.
- ▶ Hardware device
 1. Interrupt handling layer
 - Handled on the lowest level of kernel.
 - Protected the critical section (or the whole) by semaphore block.
 2. Devicedriver programs
 3. Device independent operating system program
 4. User I/O device-using program

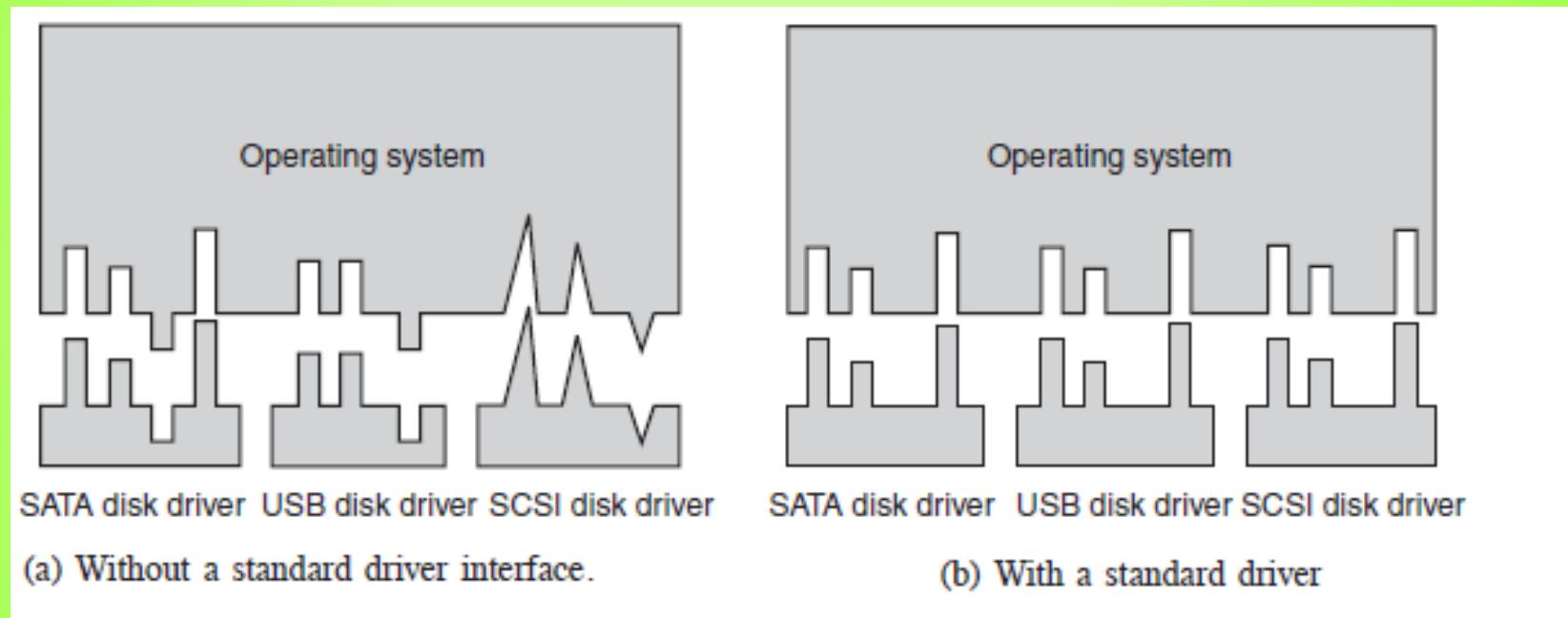
Device driver programs

- ▶ Device specific code – driver
- ▶ This knows punctually the features of the device
 - Mouse driver tells the shift, which button was pressed.
 - Disk driver knows the moving of the heads, the settings for a given sector, track.
 - Etc.
- ▶ Its task is to serve the abstract request arriving from the above level
 - Simple device, one request in the same time
 - Intelligent ones, can accept several demands simultaneously (scsi)
- ▶ Handles the device through I/O ports, using interrupt handling.
- ▶ Block-type – character-type devices

The task of a device independent I/O program I.

- ▶ To support a standard interface.
 - The same naming, calling conventions.
 - To connect the symbolic names of I/O devices to real drivers.
 - Major device number: identifier of the driver
 - Minor device number: + parameter, e.g.. Indicating write-read
 - Protection of I/O device
 - Applying filesystem-like permissions.

Standard interface – illustration.



The task of a device independent I/O program II.

- ▶ Buffering similar to adapters, general idea the user is independent from the device independent I/O program. (It writes quicker into the buffer than to the „device”.)
- ▶ Error handling
- ▶ To reserve and release monopol mode devices
- ▶ To form device independent block size.
 - Logical block size od the disks

User I/O programs

- ▶ Two categories of library I/O subroutines
 - Transmit parameters to the system call
 - `N=write(fd, buffer, db);`
 - Work some task then system call
 - `printf(„%d apple”, n);`
- ▶ Spooling
 - Handling mode of monopol devices (printer)
 - Special processes handle the storage device, spooling directories. Demons.

Usage of monopol mode resources

- ▶ As we saw formerly they are an important group of I/O devices.
- ▶ Handling of system internal table is of the kind (e.g. process table).
- ▶ Only one process can use it in the same time.
 - E.g: To print 2 files „parallel” is not the really.
- ▶ Race condition occurs not only in the case of monopol I/O devices.
 - They are at simple memory sections too, but typically at monopol I/O devices.
- ▶ Typical situation: two processes wait the same thing.
 - Two polite men before the elevator – the elevator go they stay...

Deadlock

- ▶ Two or more processes – during to allocate the same resource – blocks each other in further execution.
 - Punctual definition: A process set is in a deadlock if each of the process waits for an event which can be caused by another process.
- ▶ It is not connected only to I/O devices
 - Parallel systems
 - Databases
 - Etc.

Deadlock conditions

- ▶ Coffman E.G.: 4 conditions are needed for evolution of deadlocks
 1. Mutual exclusion: Each resource is assigned to 1 process or free.
 2. Holding and waiting. A process possessing a resource can ask for another one.
 3. No preemption. A resource can be released only by the process holding it.
 4. Circular wait. Two or more process-chain evolve in which all of the processes wait for a resource holding by another one.

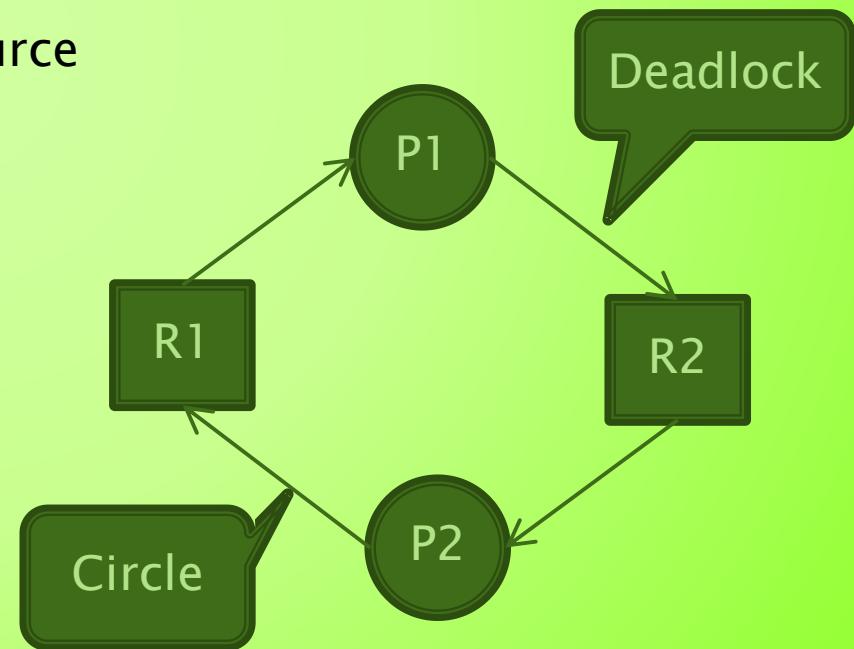
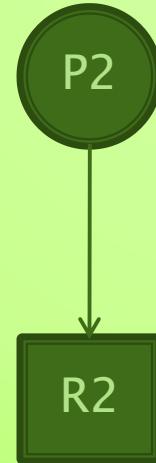
Graph model of deadlock

- ▶ Modeling deadlock conditions using graphs (Holt, 1972)
 - Process – circle
 - Resource – square
 - Finding a circle in the graph of resources, processes it means a deadlock.

Holding a resource



Asking for a resource



Evolving a deadlock, example

A

Request R
Request S
Release R
Release S

(a)

B

Request S
Request T
Release S
Release T

(b)

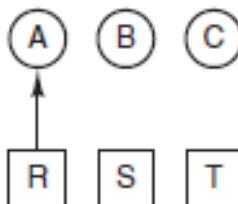
C

Request T
Request R
Release T
Release R

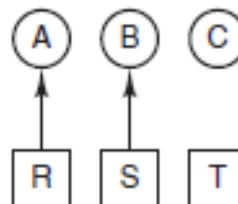
(c)

1. A requests R
 2. B requests S
 3. C requests T
 4. A requests S
 5. B requests T
 6. C requests R
- deadlock

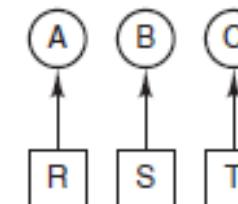
(d)



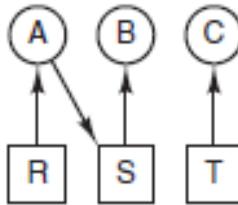
(e)



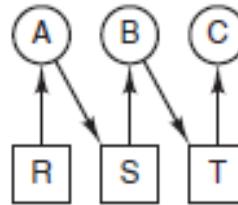
(f)



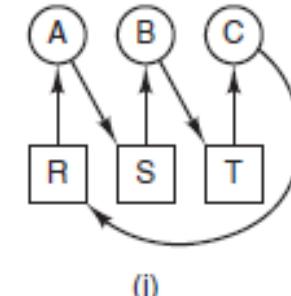
(g)



(h)



(i)

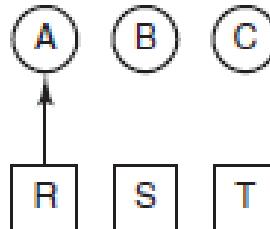


(j)

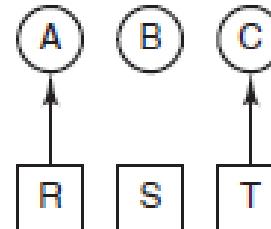
Avoiding deadlock, example

1. A requests R
2. C requests T
3. A requests S
4. C requests R
5. A releases R
6. A releases S
no deadlock

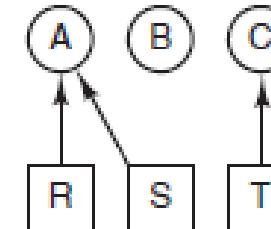
(k)



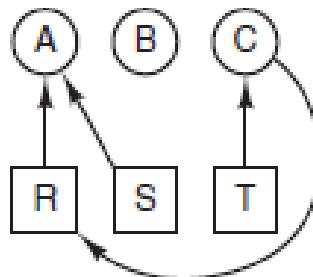
(l)



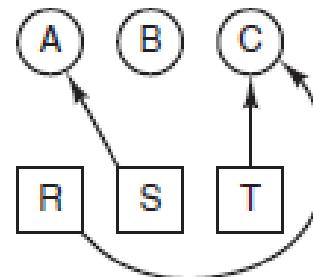
(m)



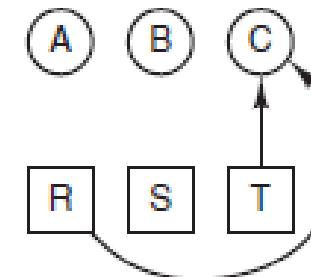
(n)



(o)



(p)



(q)

Deadlock strategy

1. Ignoring the problem.
 - We do not care with it hoping that it will not find us, if still...
2. Detection (recognizing and restoring).
 - We let to appear the deadlock (circle), we recognize it and take action.
3. Preventing. To disappoint one of the 4 needed conditions.
4. Dynamical avoidance. Resources allocation „carefully”.

1. Ignoring the problem

- ▶ This method is known also as „ostrich algorithm”.
- ▶ The question is, what does it mean, and how frequent this problem is?
- ▶ According to the studies the rate of deadlock problems and other crashes (compiler, operating systems, hw, sw error) is 1:250.
- ▶ The Unix, Windows too use this „method”.
 - The price is too big for the probable profit.

2. Recognizing and restoring

- ▶ We are monitoring the resource demands and releases continuously.
- ▶ We handle the resource-allocation graphs permanently.
 - If somewhere there is a circle we cease one of the processes from the circle.
- ▶ Other method: we do not pay attention to the resource graph at all, instead of if a process is blocked more than x (half an hour?) time, we terminate it.
 - This method is known in the case of mainframes.

3. Preventing

- ▶ Always there is a constraint for one of the 4 Coffman conditions.
 - Mutual exclusion. If a single resource is never assigned to alone 1 process, there is no deadlock!
 - It is uneasy, while the usage of a printer the printer demon solves the problem, but the printer buffer is a disk area where a deadlock may evolve.
 - If there is no such situation in which the process holding the resources wait for further resources there is no deadlock either. We can achieve this in two ways.
 - We have to know the resource request of the process forward.
 - If a process wants a resource it has to release all of its possessed ones.

3. Preventing (cont.)

- ▶ The third Coffman condition is the no preemption. To avoid this is rather difficult.
 - During a printing process it is not favored to give the printer to another process.
- ▶ The fourth condition, the cyclical waiting can be ceased easier.
 - A simple method: Each process can possess only 1 resource in the same time.
 - Other method: Assign ordinal numbers to the resources and processes may request the resources due to this ordering.
 - It is a good preventing method, but there is no proper order!

4. Dynamical avoidance

- ▶ Is there a method with which we can avoid the deadlocks?
 - Yes there is, if some informations (resource) are known forward.
- ▶ banker's algorithm (Dijkstra, 1965)
 - Similar to a small town banker's practice in trusting.
- ▶ Safe states, such situations in which there is a starting state range which result each process can allocate the demanded resources and finish!
- ▶ The banker's algorithm examines at the appearing of each request it checks whether the fulfillment of a request results a safe state or not?
 - If it is so, then the request is validated otherwise it is delayed.
 - Originally it was planned only for 1 resource.

Banker' algorithm, example states (1 resource)

	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

(a)

	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

(b)

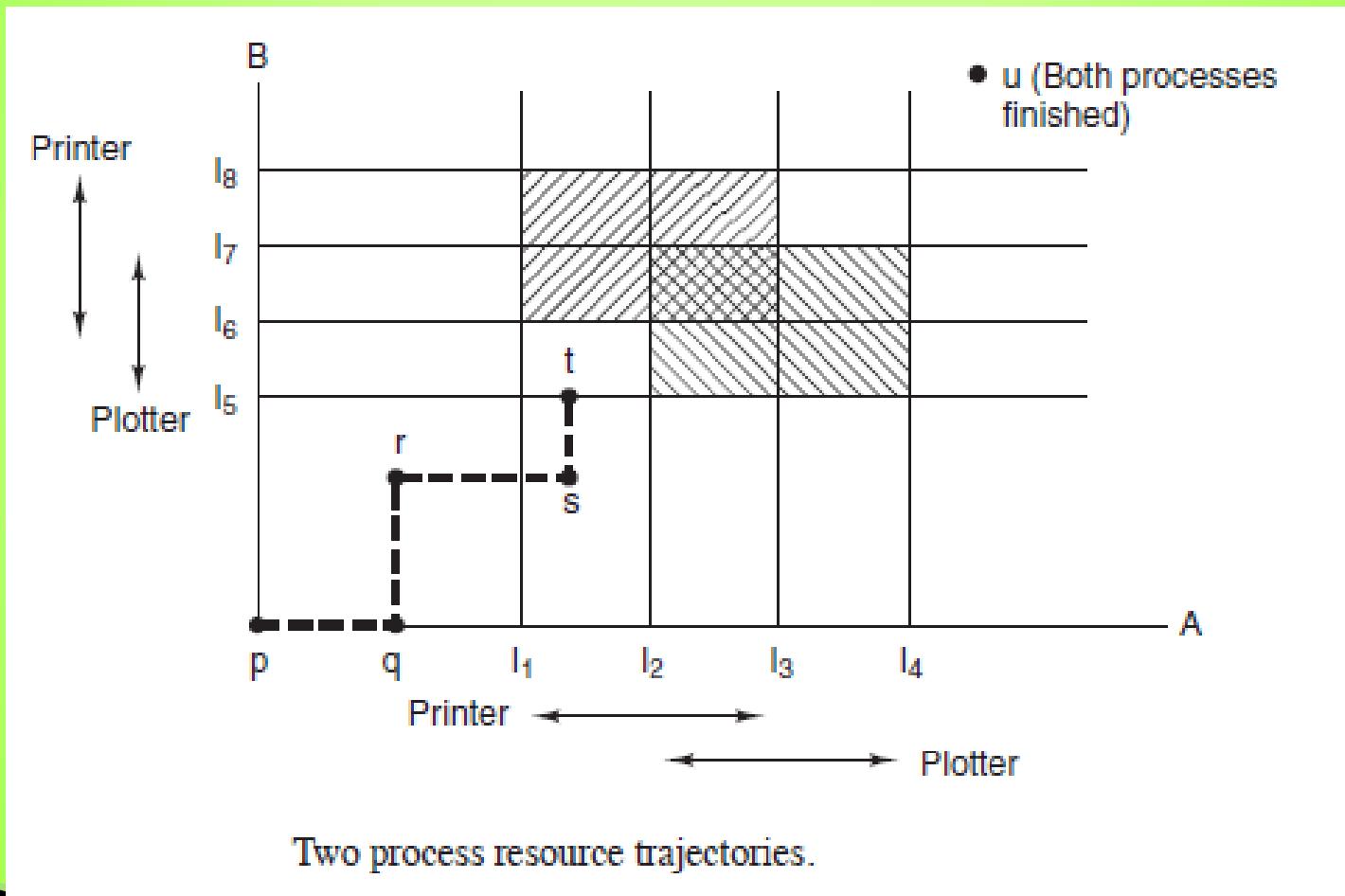
	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

(c)

Three resource allocation states: (a) Safe. (b) Safe. (c) Unsafe.

Two process resource trajectory



Banker's algorithm, in the case of several type resources

- ▶ We use 1 resource idea:
 - Notation: $F(i,j)$ the allocation of the i. of the process j. resource
 - $M(i,j)$ is the request of the i. process for the j. resource
 - $E(j)$ is all of the resources being at service.
 - $S(j)$ is the free resources being at service.
- 1. Search for i queue where $M(i,j) \leq S(j)$, if there is not such possibility then there is a deadlock, because no process can finish its running.
- 2. The i. process gets everything, it finishes then we add its allocated resources to $S(j)$.
- 3. Repeat 1,2 steps while they end or there is a deadlock.

Banker's example for several resources

	Process	Tape drives	Plotters	Printers	Blu-rays
A	3	0	1	1	
B	0	1	0	0	
C	1	1	1	0	
D	1	1	0	1	
E	0	0	0	0	

Resources assigned

	Process	Tape drives	Plotters	Printers	Blu-rays
A	1	1	0	0	
B	0	1	1	2	
C	3	1	0	0	
D	0	0	1	0	
E	2	1	1	0	

Resources still assigned

$$\begin{aligned}E &= (6342) \\P &= (5322) \\A &= (1020)\end{aligned}$$

B can allocate 1 printer this will be a safe state. (D can end then A,E,C,B.)

Banker's algorithm summary

- ▶ The previous preventing and this also asks such informations (the punctual resource requests, the number of processes), which are difficult to give forward.
 - Processes come into existance dynamically, resources modify dynamically.
- ▶ That is why it is used rarely in practice.
- ▶ ...

**Thanks for your
attention!**

zoltan.illes@elte.hu

Operating systems

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

We have talked earlier...

- ▶ Evolution of operating systems
- ▶ Notions of Operating system, structures
- ▶ Files,directories, filesystems
- ▶ Processes
 - process communications
 - critical sections, , semaphores
- ▶ Classical IPC problems
- ▶ Scheduling
- ▶ I/O, deadlock problem

What comes today...

- ▶ **Memory management**
 - Base memory handling
 - Swapping
 - Virtual memory
 - Page swapping algorithms
 - Designing paging systems
 - Segmentation

Memory handler

- ▶ Memory types
- ▶ Part of operating systems
 - Often in kernel
- ▶ Tasks:
 - To register memory which are free, occupied
 - To allocate memory for processes.
 - To free memory.
 - To control the swap between the RAM and the hard disk

Basic memory handling

- ▶ Two different group of algorithms:
 - There is a need to translate, change processes between the memory and the disk. (swap)
 - There is no need of it.
- ▶ If there is enough memory there is no need of swapping!
- ▶ Is there enough memory?
 - HT1080Z – 16 KB, C64 – 64 KB, IBM PC – 640 KB
 - Nowadays: PC 2–4–8 GB, a small server 4–16 GB

Monoprogramming

- ▶ One program in the same time.
 - There is no need of an „algorithm”. Type the command, execute it and wait for the next one.
 - Typical situations
 - Op.system is on the lower addresses, program is above it (it is rarely used today, formerly it was used at mainframes or at minicomputers).
 - On the lower addresses the program, the higher area the operating system in ROM (hand-held computer, embedded system)
 - Op.system is on the lower addresses, above the user programsm above the device drivers in ROM. (MS-DOS, ROM-BIOS)

Multiprogramming (Multitask)

- ▶ „Parallel” several programs are running.
Memory must be divided among the processes.
 1. To implement multiprogramming with fixed memory slices.
 2. Multiprogramming with memory swapping.
 3. Multiprogramming with the usage of virtual memory.
 4. Multiprogramming with segmentation.

1. Multiprogramming with fixed memory slices

- ▶ Monoprogramming is present today mainly in embedded systems (PIC)
- ▶ Nowadays typically there are several „running” processes in the memory in preemptive time sharing systems.
- ▶ Divide the memory into n slices (not equal) (Fix slices)
 - E.g. at system start it may be done
 - One common waiting queue.
 - Each slice has an own waiting queue.
 - Typical solution for batch systems.

Memory division

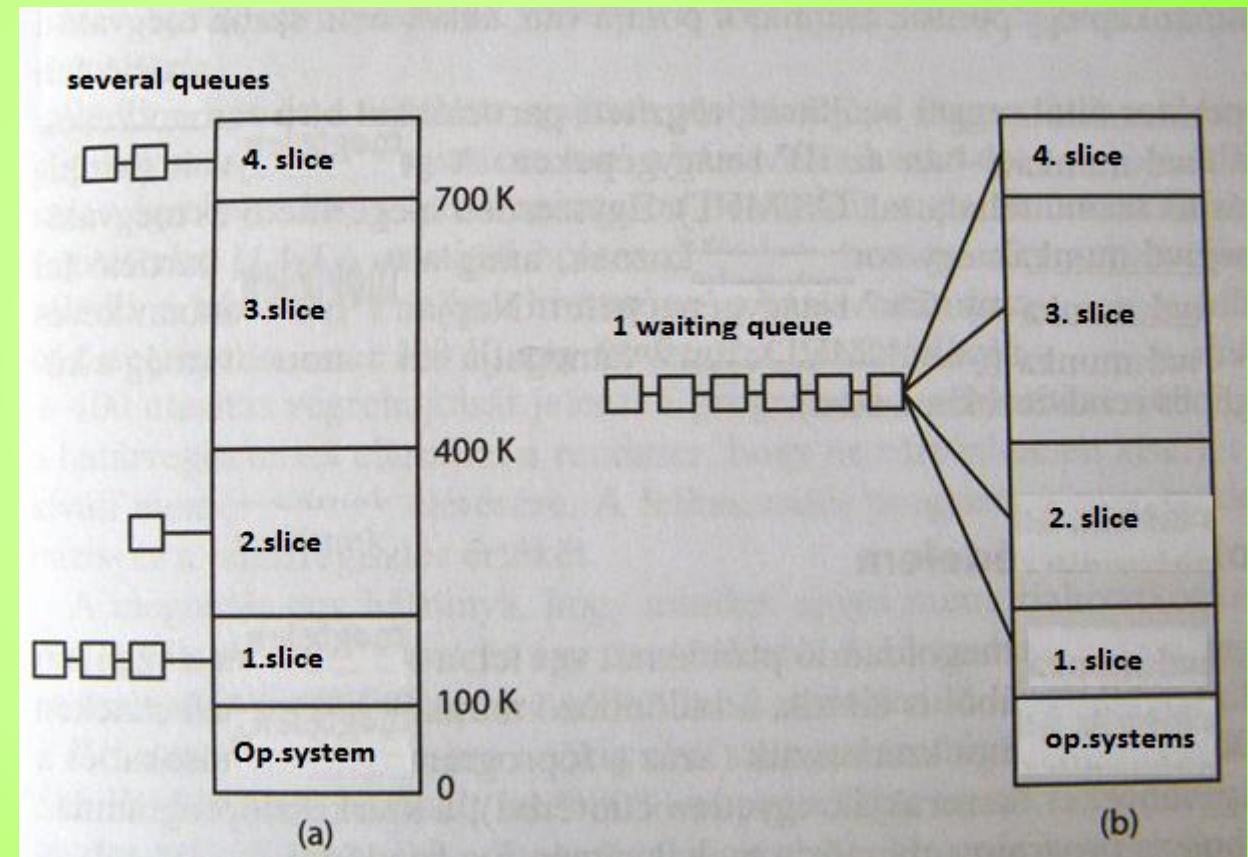
Several waiting queues:

- Not used partitions

One queue:

- If a partition becomes empty, the first from the queue get into it.

IBM used in OS/360 system: OS/MFT (Multiprogram Fix Task)



Relocation – protection

- ▶ We do not know where the process will be, so the memory addresses can not be compiled to fix ones.
 - OS/MFT program at loading fresh the relocated addresses.
- ▶ It is not desirable if a program can access the memory of another program! (Protect)
 - OS/MFT PSW (program status word, 4 bits long safety key)
- ▶ Other solution is: Base+limit register usage
 - Programs are not able to modify these.
 - Must be checked at each address reference: it is slow.

2. Multiprogramming with memory swapping

- ▶ Fixed memory slices are a typical solution for formerly used batch systems. (IBM OS/MFT)
- ▶ It is not proper for time sharing, graphical systems.
- ▶ To swap the whole process between the memory and the disk.
- ▶ There is no fixed memory partition, each of them changes dynamically due to how the operating system pack them to-and-fro.
- ▶ It will be dynamical, better memory usage, but the great number of swapping result holes!
 - Memory compression must do! (In several cases (during watching a football match) this time loss is not permitted.)

Dynamic memory allocation

- ▶ Generally it is not known forward how many dynamic data, stack area is needed for a program.
- ▶ The program code gets a fix slice, while the data and the stack gets changeable ones. They can grow and ease down.
 - If the memory runs out, the process stops and waits for carrying on or it is swapped to the disk to give the other running processes more memory.
 - If there is any waiting process in the memory it may be swapped too.
- ▶ How can we register the dynamic memory?

Registering the dynamic memory

- ▶ To define an allocation unit.
 - The size of it is questionable. If it is small there will not be so many holes but the resource (memory) demand for registering is big.
 - If it is big, then the loss we get from the remainder memory (not used) in a unit is too much.
- ▶ The implementation of registering:
 - Using a bitmap
 - Using chained list
 - If a process ends the memory holes being beside each other should be concatenated.
 - The holes and the processes have separate lists.

Bitmap, implementation of chained list

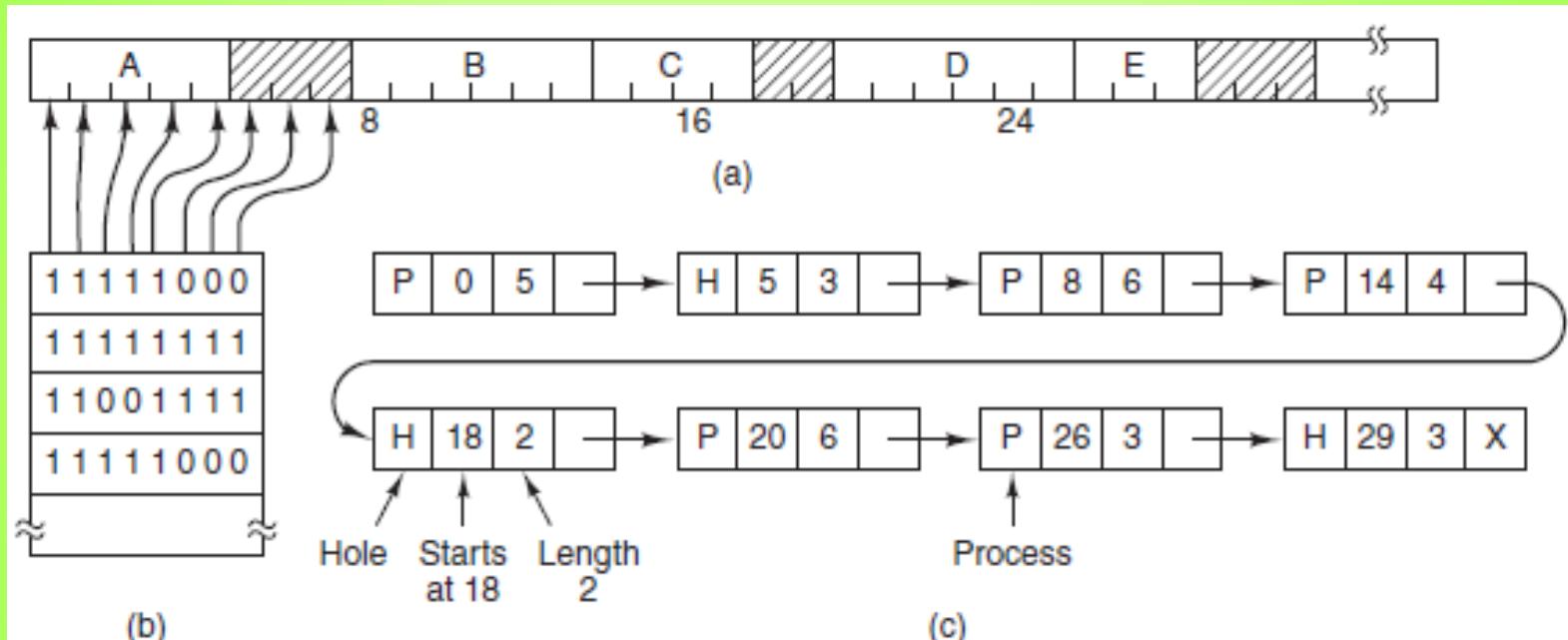


Figure 3-6. (a) A part of memory with five processes and three holes. The tick marks show the memory allocation units. The shaded regions (0 in the bitmap) are free. (b) The corresponding bitmap. (c) The same information as a list.

Memory allocation strategies

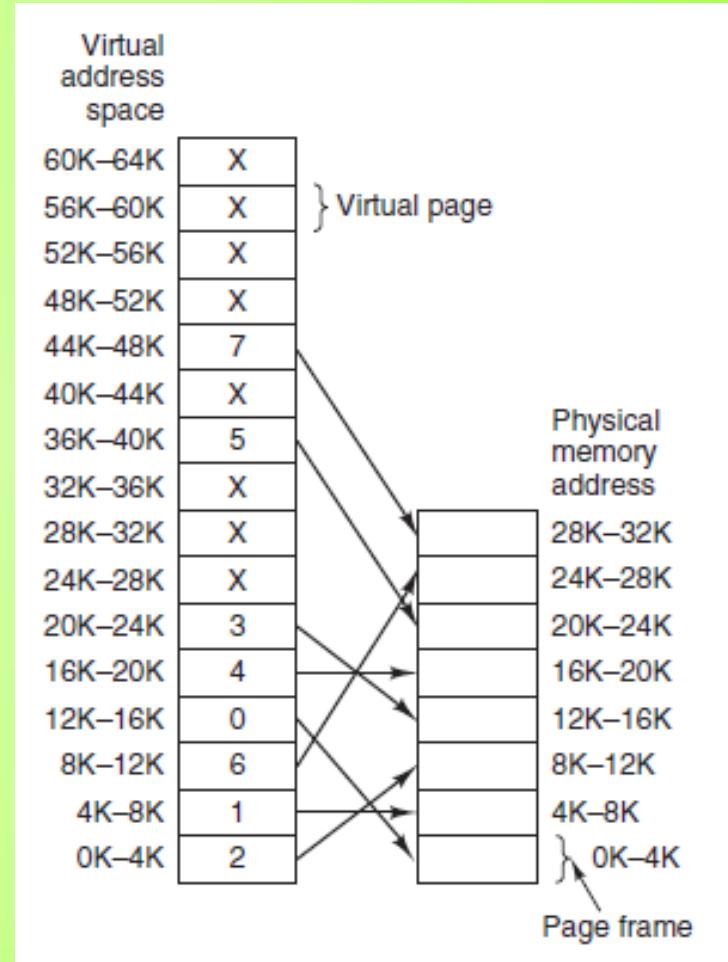
- ▶ To decide a memory location for a new (or a previously running one being now in the swap partition) there are several memory allocation algorithms. (similar to the ones at disks) :
 - First Fit (to the first place where it can fit, quickest, simplest method.)
 - Next Fit (the searching starts not from the starting point but from the last search finishing point, not so efficient as first fit)
 - Best Fit (it is slow, it produce a lot of small holes)
 - Worst Fit (will not be a lot of small holes, but it is not efficient)
 - Quick Fit (a list of holes ordered by their size, the concatenation of holes are expensive.)

3. Multiprogramming with virtual memory usage

- ▶ A program may use more memory than the available amount of physical memory.
 - The operating system keeps only the needed part in the physical memory.
 - A program is in the „virtual memory space”.
 - John Fotheringham (1961, ACM)
 - The IDE may be used in the case of a monoprogramming environment too.
 - In 1961 and later there were „single task” systems like (small) computers.

Memory Management Unit (MMU)

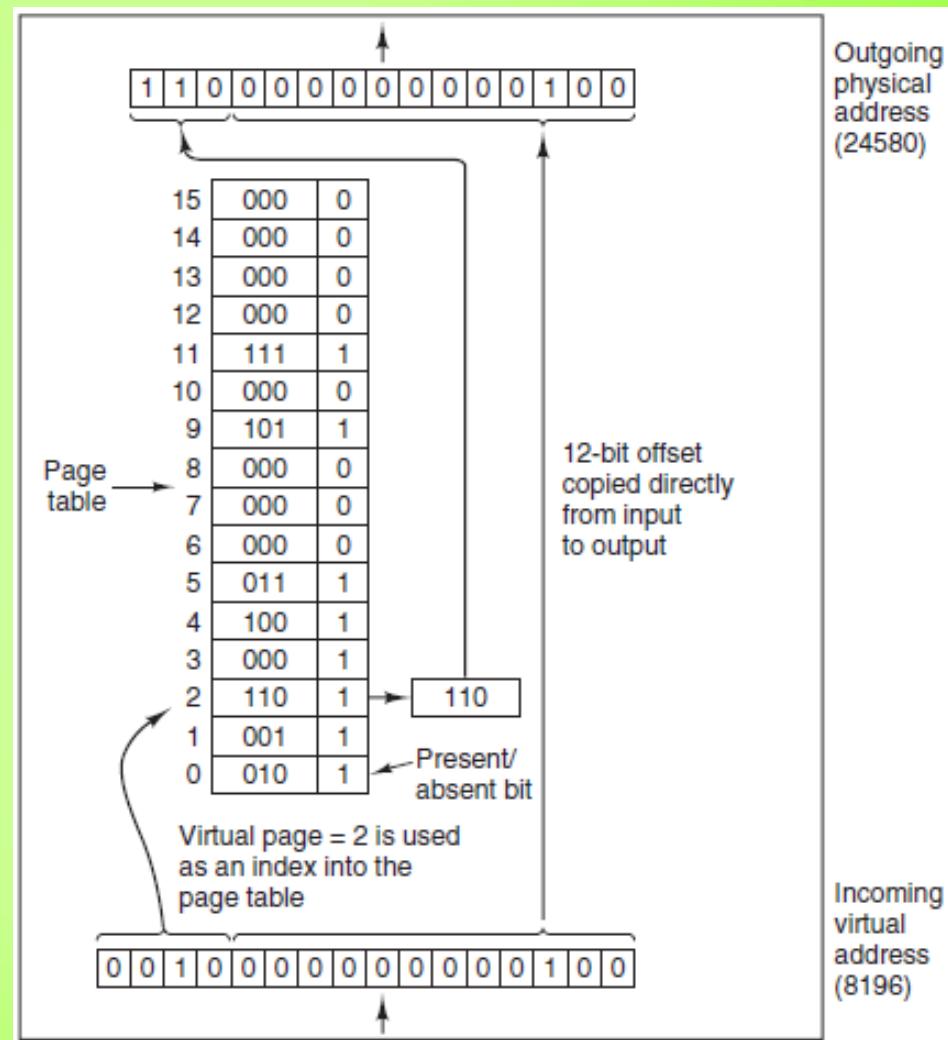
- ▶ The virtual address space is divided into pages. (it is called page-table)
- ▶ present, not present bit
- ▶ Binding of virtual-physical pages
- ▶ If MMU sees that a page is not in the memory, causes a page fault, creates a page frame and load the needed page.



MMU working

Example: 16 piece – 4kb page

- ▶ Size of pages are two on the power
- ▶ The first 4 bit from the 16 bits long virtual address is the page number, the remaining ones are the offset.
- ▶ 1 bit for signing the present or absence.
- ▶ 15 bites are going out to the physical address bus.



Page table problems

- ▶ Previous example (16 bit virtual address space, 4 bit page table, 12 bit, 4 KB, offset) simple, quick.
- ▶ A modern processor is 32 or 64 bit.
 - 32 bit virtual address space: at 12 bit (4kb) page size 20 bit is the size of page table. (1MB, kb. 1 million element)
 - Each process has an own virtual address space, and an own page table! Such size page table is unimaginable!
 - 64 bit virtual address space such page table size is not realizable!

Multi-level page tables

- ▶ Headache how to use a simple page table in the case of 32 bit virtual address space.
- ▶ Use a multi-level one:
 - Page table1 = 10 bit (1024 elements) – higher level page table
 - Page table2 = 10 bit – second level page table
 - Within a page Offset = 12 bit
 - Advantage: while it is needed for a process to keep the page table in the memory too, here we have to get only 4 tables with 1024 elements (LT1 plus the program, data and the stack) and not 1 million!
- ▶ We choose classical values for bit numbers for table and offset – we can modify them but will not be essential changings.
- ▶ We can change the two level to more levels! (More complicated!)

The structure of a table registration

- ▶ Contains the page frame number (physical memory and the offset size decide how many bits)
- ▶ present/absence bit
- ▶ Protect bit, if the value is 0 it is readable, in the case of 1 it is only writeable.
 - 3 protection bit: read, write, execution permission for each page.
- ▶ Dirty bit (modifying). If it is 1 then the page frame was modified, so in the case of writing out to the disk you have to write it really!
- ▶ Reference bit, if the value is 1 if the page is referenced (it is in use). If a page is used it is not able to be written out to the disk!
- ▶ Cache disabled bit. It is important in the case when a given area of the physical memory is at the same time the data area of any I/O device.

Instead of (besides) paging TLB (Translation Lookaside Buffer)

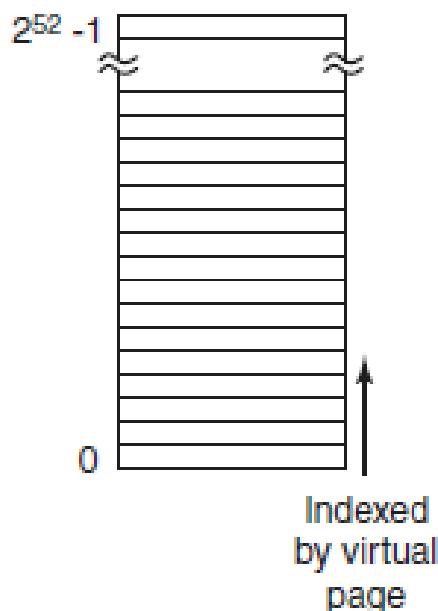
- ▶ The classical MMU possibility is slow, we need at least one table reference for each memory reference, so the memory operation time may decrease to the half of its minimum
- ▶ Insert a small HW unit (TLB – associative memory) to the MMU with a small amount of registry (not more than 64 at the beginning)
 - Today at Nehalem there is a two-level TLB, the TLB has 512 elements.
- ▶ Software handling of TLB
 - 64 elements are rather big to get only few TLB faults, so HW solution may be saved.
 - Such systems are e.g.. Sparc Mips, Alpha, HP PA, PowerPC

Inverted page tables

- ▶ Start from the size of the physical memory.
- ▶ The page table contains so many elements which is given by the physical memory.
 - It spares a lot of space but it is more complicated to get the physical memory address from the virtual one. (No possibility of using an automatic index!)
 - E.g.: 4 kb page size, 1GB RAM, 2^{18} number of reference.
- ▶ Out of the woods: Use the TLB, if it faults search the inverted page table and write the result into TLB.

Inverted table example

Traditional page table with an entry for each of the 2^{52} pages



1-GB physical memory has 2^{18} 4-KB page frames

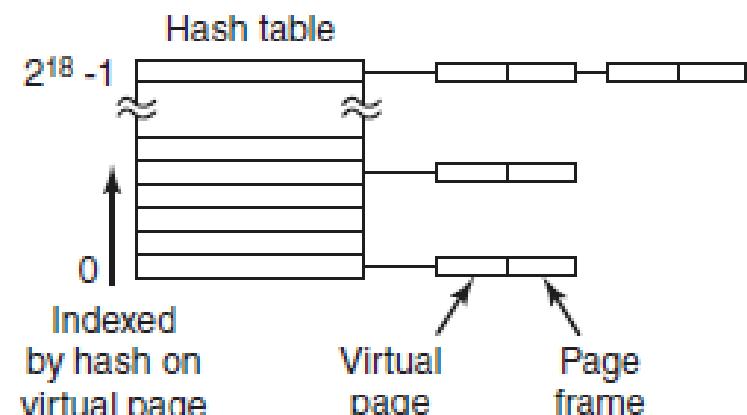
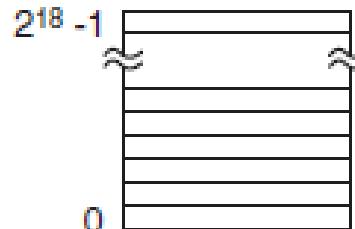


Figure 3-14. Comparison of a traditional page table with an inverted page table.

Page swapping algorithms

- ▶ If the virtual addressed page is not in the memory then a page must be throw away and translate this new page.
 - The question is how?
 - Randomly? It is better to move out thet page which was not used „recently”.
- ▶ The case is similar at the usage of the processor cache memory or at the local cache of a browser.

Optimal page swapping

- ▶ Sign each of the pages with a number which indicates how many CPU instructions was executed before this reference!
- ▶ Throw away that page in which this number is the least!
- ▶ There is a problem, it is impossible to implement!
- ▶ This method can be used at testing running twice the program!

NRU (Not Recently Used) algorithm

- ▶ Use the modify and the reference bit of the page table.
 - Set the reference bit to 0 time to time (at each clock interrupt at about 0.02 sec) with this signing whether it was used recently or not.
 - 0.class: not referenced, not modified
 - 1.class: not referenced, modified
 - Page can get here if the clock interrupt sets the reference bit to 0.
 - 2.class: referenced, not modified
 - 3.class: referenced, modified
- ▶ Choose randomly a page from the least not empty class.
 - Simple, not really efficient its implementation, it gives a proper result.

FIFO page swapping second chance algorithm

- ▶ Simple FIFO, it is known from other areas too, if we need a new page we throw away the oldest one!
 - The pages are in a list ordered by arriving time, a page arrives to the beginning of the list and leaves from the end.
- ▶ The correction of this is the second chance page swapping algorithm.
 - Almost the same as FIFO. The difference is if the page at the end of the list has got a reference bit with value 1 then it gets a second chance. It is moved to the beginning of the list and the reference bit is set to 0.

Clock page swapping

- ▶ Clock algorithm: like the second chance method but do not move the pages inside a list instead of it use a circled list and go around with a pointer.
- ▶ The pointer shows the oldest page.
- ▶ At page fault if the pointed page reference bit is 1 change it to 0 and move to the next page!
- ▶ If the examined page reference bit is 0 then we throw away!

LRU (Least Recently Used) algorithm

- ▶ Throwing away the least recently used page.
- ▶ How can we implement it?
 - HW or SW
 - HW1: Take a counter which increases with 1 at each reference. We can store this counter at each page table. At each page reference we write this counter into the page. At page fault we search for the page with the smallest counter!
 - HW2: LRU bitmatrix usage, n page, n x n bitmatrix. At a k. ōage frame reference set the k. row of the matrix to 1, whilst the k. column to 0.
 - At page fault the smallest row value indicates the oldest page!

NFU (Not Frequently Used) algorithm

- ▶ Give a counter to each page. At each clock interrupt give this the value of the page reference R bit.
- ▶ At page fault throw away the page with the smallest counter. (It is the least used)
- ▶ It is a problem, the NFU does not forget, the pages used frequently at the beginning of the program save their high values.
- ▶ Modify it: At each clock interrupt shift to the right the counter with one bit and put from the left the reference bit (shr). (aging algorithm)
 - It approximates well the LRU algorithm.
 - This page counter is implemented with n bits so it is not able to differentiate events before n time unit.

Viewpoints of designing page swapping I.

- ▶ Working set model
 - Load the demanded pages. At starting paging before.) To keep in the physical memory only those pages of a process which are used. This may alter with time.
 - We have to register the pages. If a page was not referenced in the last N time unit we throw it at a page fault.
 - Correction of clock algorithm: Examine if a page is a member of the working set or not? (WSClock algorithm)
- ▶ Local, global place allocation
 - If we examine all of the processes in the case of a page fault then it is global. If we examine only the pages of the same process then it is called local.
 - In the case of a global algorithm we can give a proper page due to its size which we can change dynamically.
 - Page Fault Frequency (PFF) algorithm, rate of page fault /sec, if there is a lot of page fault increase the page numbers of the process in the memory. If there is a lot of process we may write out the whole process to the disk. (load balance)

Viewpoints of designing page swapping

II.

- ▶ To decide a proper page size.
 - Small page size
 - The page loss is small, but we need a big page table for registration.
 - Big page size
 - Inversely, „page loss” is big, small page table.
 - Typically: $n \times 512$ byte the page size, XP, Linux-s 4KB page size. 8KB is used too (servers)
- ▶ Common memory
 - We can allocate a memory area which can be used by several processes.
 - Distributed common memory
 - To share memory between processes in the network.

Segmentation

- ▶ Virtual memory: one dimensional address space, from 0 till maximum address (4,8,16 GB, ...)
- ▶ Several programs have dynamical area which may increase with uncertain size.
- ▶ Create address spaces independent from each other these are called segments.
 - In this world an address consists of 2 parts: segment number and an address within it. (shift)
 - Segmentation makes possible to implement „simply” distributed directories.
 - A program can be split logically to program, data etc. segment...
 - To give a protection level to each segment.
 - The page sizes are fix sized, the segments are not.
 - Fragmentation of segments appears. This fragmentation can correct with concatenation.

Pentium processor – virtual address handling I.

- ▶ There can be a lot of segments, the address space of a segment: 2^{32} byte (4GB)
 - XP, Linux – use simple page swapping model. One process is one segment – one address space.
 - The number of segments: Pentium 16000, from TSS segment value, processor feature.
- ▶ LDT – Local Descriptor Table
 - Each process has got an own.
- ▶ GDT – Global Descriptor Table
 - There is only one from this, everybody uses it.
 - The system segments (operating system) is in it.
- ▶ Physical address: Selector + offset operation execution

Pentium processor – virtual address handling II.

- ▶ Achieving the segment: 16 bit segment selector
 - The low 0–1. bit of it gives the permission of the segment.
 - 2. bit=0=in GDT, 1=in LDT is the segment description.
 - High 13 bit is the index of the table. (Both of the two tables consist of 8192 elements)
- ▶ The LDT, GDT table elements are 32 bit long, from this we can get the base address to which we add the shift. (The result also 32 bit long!) This gives the linear address.
- ▶ If the paging is forbidden this is the real physical address. If it is not forbidden then two level page table usage: page size 4 KB, 1024 (10 bit) elements in page table.
- ▶ To get a page frame quicker TLB used too.

The protection levels of Pentium processor

- ▶ 0- Kernel level
- ▶ 1- System calls
- ▶ 2- Shared libraries
- ▶ 3- User applications
- ▶ To access data from lower levels is permitted.
- ▶ To access lower level (0–1 level) data from a higher level is forbidden.
- ▶ To call subroutines is permitted but only under controlling (call selector)
- ▶ User program can access data of shared libraries, but they do not modify them!

**Thanks for your
attention!**

zoltan.illes@elte.hu

Operating systems

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

We have talked earlier...

- ▶ Evolution of operating systems
- ▶ Notions of Operating system, structures
- ▶ Files,directories, filesystems
- ▶ Processes
- ▶ Classical IPC problems
- ▶ Scheduling of processes
- ▶ I/O, deadlock problem
- ▶ Memory management

What comes today...

- ▶ The rethinking of the operation system tasks
- ▶ What is missing from today solutions?
- ▶ What type of requirements appear in nowadays system environments?
 - From the users' side
 - From the devices of our environment
 - From industrial, economical side
- ▶ Appearing the importance of TIME!
 - (Real-Time Systems)

Features of nowadays operating systems

- ▶ Preemptive systems
 - Processes with priority
 - Typical scheduling, CFS or very similar to it.
- ▶ Multi task, similar file systems.
- ▶ Segmented, virtual memory management.
- ▶ Layered I/O, no supervisoring.
- ▶ No starvation, each process will be executed sooner or later!
 - Graphical UI

What is missing?

- ▶ On the whole nothing, everything is nice, everything is good ...
- ▶ Only one thing is missing from the previously mentioned system features!
- ▶ The role of time is subsidiary!
 - Everybody is equal (taking notice of priority)
 - Sometime everybody gets resource (CPU)!
- ▶ Is it enough, „sometime”?
- ▶ It is true, the efficiency of the computer systems continuously grow but we must rethink the role of time again!

Real-Time system

- ▶ From what becomes a system real-time?
- ▶ The system which takes care of the time during the execution of the processes is a real time system!
- ▶ With another words: the operating system guarantees that in the case of an event the process which waits for it gets to CPU!
- ▶ Why it is real?
 - If we assign to a task that it should be executed at 5 o'clock (maybe must finish at 5 o'clock) then we insist on the punctual time – not before or after it with 2 minutes!

Soft or Hard Real-Time

- ▶ Due to one of the wordings, an application must be scheduled as a response to an event!
 - We call response time the elapsed time between the event and the scheduling of the application!
- ▶ We call a system soft if we may differ from it „slightly”! (Soft Real-Time)
 - What does it mean „slightly” in this case? It depends on the application...
- ▶ If it is not permitted to differ from the required response time that it is called a hard real-time system!

Applying real-time systems

- ▶ We may speak about two different types of real-time systems:
 - Real-time operating systems (RTOS)
 - Real-time applications (RTApp)
- ▶ Where do we use them?
- ▶ Recently typically in industrial environment, their task was to control robots.
- ▶ Nowadays these kinds of tasks appeared in every day life too.
 - Paying with credit cards
 - Security systems
 - Card door lock system, barrier systems
 - etc.

Real-time systems – Past

- ▶ It is only the requirement of nowadays to support real-time tasks?
 - No
 - In the case of DOS there was not any barrier to write such applications!
- ▶ The operating system itself was not real-time, but typically they were not multi-task systems, so you might implement real-time applications!
 - E.g.: Timer preparing (1CH), to insert your own machine code, to use a custom made interrupt handling.

Real-time systems – Present

- ▶ Nowadays typically used operating systems (Windows, Linux) are not real-time ones.
- ▶ Though in Windows too appeared the possibility to set real-time priority for processes, but it is not an RTOS!
- ▶ Typically we can speak about two types of real-time systems.
 - Embedded systems
 - Full real-time operating systems

Embedded systems

- ▶ They are differ from general operating systems.
- ▶ Typical tasks are, to grant the working of industrial equipments, controllings, electronic devices!
 - e.g.: Digital cameras, GPS devices
 - Set-top box firmware,
 - Automotive Infotainment systems
 - etc.
- ▶ Windows CE systems (Windows Embedded Compact 7, 2013)
▶ ONX Neutrino

Full RTOS system(s)

- ▶ There is no such Windows based system!
- ▶ There are several Linux based RTOSs.
 - e.g.: RTLinux
 - Real-Time Linux for Debian
 - Suse Linux Enterprise Real-Time Extension
 - Etc.
- ▶ Typically there is a free version too without support!
 - Today the SLE 11 SP4 RT is the newest version!

System features – important themes

- ▶ Real-Time features
- ▶ Efficiency, CPU shield
- ▶ Scheduling
- ▶ RT interprocess communication
- ▶ Synchronization
- ▶ RT signals
- ▶ Clocks, timers

SLE RT features

- ▶ Processor shield
- ▶ Processor affinity
- ▶ Scheduling, priority changing
- ▶ I/O priority changing
- ▶ Posix RT Extensions
 - Memory resident programs
 - Process synchronization
 - Asynchron, synchron I/O
 - Signals, timers, messages

Processor shield

- ▶ In a time sharing, preemptive system each process is executed by a processor controlled by the scheduler!
- ▶ Let us dedicate one (or more) CPU core to the high priority (RT) processes.
 - Result of it: the shielded CPU-s grant the quick response time, interrupt handling and keeping of deadlines!
 - Not shielded cores will serve the other processes, devices of the system!

Shaping up of CPU sets

- ▶ Handling CPU sets: cset
 - cset shield --cpu=3
- ▶ Important subcommands:
 - Set
 - cset set -l # cpu set list
 - Shield
 - Cset shield -cpu=1,2,4-6 #1,2,4,5,6 CPU is shielded
 - Proc
 - Cset proc -exec command # run cmd in shield CPU set
- ▶ One can use this command only with administrator permission!
- ▶ Help: cset –help

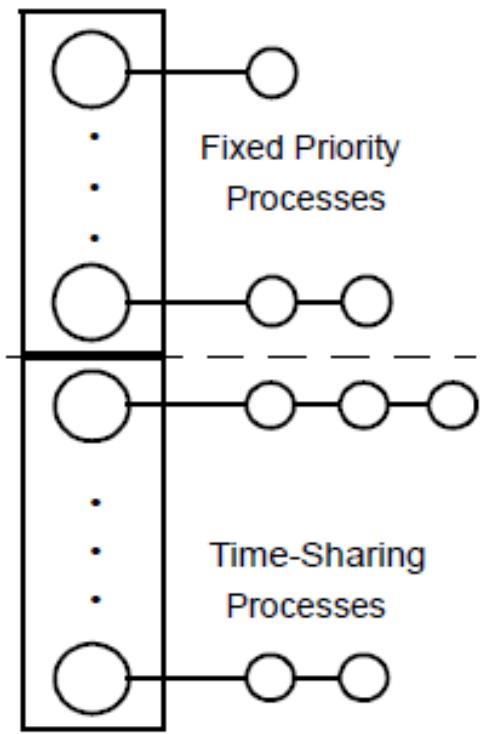
Processor affinity

- ▶ CPU affinity – taskset command
- ▶ Default behavior of the kernel: to keep a running process on the same CPU
- ▶ In the same time the kernel try to get a balanced CPU load! (load balance)
- ▶ During the scheuling sometimes may happen that a process is transferred from a CPU to an other one!
- ▶ Processor affinity prevents this!

Taskset command

- ▶ **Taskset --help # basic possibilities**
 - Taskset #does the same by using it without parameters!
- ▶ **Important commands:**
 - To have a look at the CPU affinity of a process:
 - Taskset -p pid
 - To set the CPU affinity of a process:
 - Taskset -p mask pid
 - To execute a process:
 - Taskset mask command
 - Instead of Mask you may write the ordinal number of the CPU
 - /proc/cpuinfo file

Schedulers in RT environment

User Priority	Global Priority	Scheduling Order	Policy-Specific Priorities	Scheduler Policies	Process Queues
99 · · · · 1	Highest 1  99	First  Last	Fixed Priorities SCHED_FIFO and SCHED_RR		Fixed Priority Processes
0 · · · · 0	100  139 Lowest		Time-Sharing Priorities SCHED_OTHER		Time-Sharing Processes

Setting of real-time attributes

- ▶ Chrt command – scheduler, priority changing
- ▶ Chrt -help # default command options
- ▶ Chrt -m # possible scheduling
algorithms
 - SCHED_OTHER (TS)- default CFS scheduler
 - SCHED_FIFO (FF)- RT FIFO scheduler
 - SCHED_RR (RR)- RT Round Robin scheduler
- ▶ Chrt --fifo -p 42 50234 # priority:42
pid: 50234

Chrt usage

- ▶ To set an RT priority you should have an admin permission!
- ▶ The nice command default meaning is to decrease the normal priority with 10!
- ▶ RT priorities are: 1–99

```
ilosz@oprendszer:~> ps -o pid,rtprio,comm,class,pri
   PID  TID RTPRIO COMMAND          CLS PRI
 5154  5154      - bash           TS  19
 5264  5264      - sleep          TS   9
 5267  5267      - ps             TS  19
ilosz@oprendszer:~> ps -o pid,rtprio,comm,class,pri
   PID RTPRIO COMMAND          CLS PRI
 5154      - bash           TS  19
 5269      - ps             TS  19
[1]+  Done                      nice sleep 15
ilosz@oprendszer:~> nice sleep 15&
[1] 5271
ilosz@oprendszer:~> ps -o pid,rtprio,comm,class,pri
   PID  TID RTPRIO COMMAND          CLS PRI
 5154      - bash           TS  19
 5271      - sleep          TS   9
 5272      - ps             TS  19
ilosz@oprendszer:~> chrt -m
SCHED_OTHER min/max priority : 0/0
SCHED_FIFO min/max priority : 1/99
SCHED_RR min/max priority  : 1/99
SCHED_BATCH min/max priority : 0/0
SCHED_IDLE min/max priority : 0/0
[1]+  Done                      nice sleep 15
ilosz@oprendszer:~>
```

SCHED_BATCH, SCHED_IDLE

- ▶ Besides SCHED_OTHER there are others we have seen it before!
- ▶ SCHED_BATCH
 - Similar to SCHED_OTHER default scheduler!
 - It can be used only with static priority! (There is only nice 0!)
 - It can be useful in the case of CPU intensive, not interactive tasks!
- ▶ SCHED_IDLE
 - It can be used only with 0 static priority, nice value does not affect it!
 - It is suggested for low priority background processes!
 - It has a lower priority then nice +19! (This is the 139 value of the table!)

I/O priority

- ▶ We have seen the features of classical I/O schedulers they are modified a bit in a modern RT environment.
- ▶ The system uses 3 priority class
 - Idle – 3 class (lowest), it is for not time critical processes, in it there is no nice level
 - Best Effort – 2 class, in it there are 8 levels (0–7), 0 is the highest. This is the default one. Ionice value fits to the nice value of the process!
 - Real Time – 1 class, 8 levels, this is the highest class, serving this is the first before anything else!

I/O priority usage

- ▶ Ionice command, man ionice
- ▶ -c parameter, setting of I/O priority class
 - -c1,-c2 or -c3 possible values
- ▶ -p parameter, process assigning
 - -p 5021 # process with pid number 5021
- ▶ -n parameter (you may omit it),setting ionice
 - -n 3 # to set ionice 3
- ▶ E.g.
 - ionice -c3 -p\$\$ # actual shell: idle
 - ionice -c1 -p5031 -n5 # process 5031 RT/n=5

Block device I/O scheduler

- ▶ Disk subsystem scheduler.
- ▶ We may set schedulers separately to each block-type device!
- ▶ As we have seen at the filesystems the main goal is to reduce the unwanted head movements and hereby to increase the bandwidth!
- ▶ Typical I/O block scheduling:
 - Noop – The alignment of default requirements, mainly used in RAID systems!
 - Deadline – To give a response before the deadline, it is used typically in RT systems.
 - Cfq – Completely Fair Queuing, it is the default.

Modifying the block I/O scheduler

- ▶ Device descriptors are in `/sys/block` directory! (they are links)
 - Sda – default disk
 - Fd01 – floppy 01
 - Etc.
- ▶ `/sys/block/device/queue`
 - The parameters, data of a given device
 - Cat `/sys/block/sda/queue/scheduler`
 - Noop deadline [cfq] # cfq the chosen
 - Sysfs command makes possible to set the parameters of block device scheduler.
 - These parameters are in the `/sys/block/sda/queue-iosched` directory!

Deadline I/O scheduling

- ▶ Goal: we must finish the I/O task before the deadline!
- ▶ The scheduler uses two lists:
 - In one of it there are the requires in block order.
(One after the other the „nearby” ones.)
 - In the other one the requests are sorted by the deadlines!
- ▶ The default serving is done by the block order except there is a deadline, in this case it will be executed before!

Real Time IPC – Queues

- ▶ Message Queue and Shared memory
 - Exists in System V too! (msgget, shmget...)
- ▶ Posix message queue: implemented as files in /dev/mqueue file system!
- ▶ System limits for Posix message queues:
 - Resides in /proc/sys/fs/mqueue directory
 - `Msg_max=10`, max. message number in each queue, limits by
`HARD_MAX=131072/sizeof(void*)` (appr: 32768)
 - `Msgsize_max=8192` (bytes) max message size
 - `Queues_max=256`, max number of queues

RT message queues

- ▶ Compile: -lrt
- ▶ Include: <mqueue.h>
- ▶ In a message queue each message has the same message slot size!
 - A message size may differ (less or equal) from slot size!
- ▶ Every message has a priority!
- ▶ The oldest, highest priority message is received first by a process!
- ▶ A message is a simple byte set! (char *)
 - Sending numbers, etc., it must cast!

RT message queue features

- ▶ Mq_open – opens a message queue
- ▶ Mq_send, mq_timedsend – send a message (char*) with a priority (and timespec time delay)
- ▶ Mq_receive, mq_timedreceive – receive a message (with a max timespec amount), this call blocks if queue is empty!
- ▶ Mq_notify – the calling process is registered for notification of the arrival a message!
- ▶ Mq_unlink – removes message queue.

Posix shared memory

- ▶ A storage object is defined as a named region and can be mapped by one or more processes!
- ▶ Shm_open – creates a shared memory object with zero size!
- ▶ Ftruncate – sets the size of memory object
- ▶ Mmap – maps to the virtual memory portion
- ▶ Fstat – gets the memory size
- ▶ Shm_unlink – delete shared memory

Memory mapping

- ▶ Establish memory mapping to a target process' address space!
 - Mmap – map a portion of memory to a /proc/pid/mem file and thus directly access the content of another process address space.
 - See man mmap
 - Usermap – an alternative way for mapping procedure!
 - See man usermap

Interprocess synchronization

- ▶ Reschedulling control
- ▶ Busy-Wait mutexes
- ▶ Posix semaphores
- ▶ Extensoins to Posix mutexes
- ▶ Condition synchronization

Rescheduling control

- ▶ It defers CPU scheduling for brief periods of time.
- ▶ Variables Rescheduling
 - Resched_cntl(cmd,arg) registers a variable, and the kernel examines it before rescheduling decision!
 - Resched_lock(v) – locks the variable (increase the number)
 - Resched_unlock(v) – unlocks the variable, if variable is zero or less than 0, preemption is enable!
 - Resched_nlocks(v) – returns the number of locks

Busy-Wait mutexes

- ▶ This is a very low overhead operation!
- ▶ Often called: spin lock
 - It uses the spin_mutex structure, <spin.h>
 - General interface functions
 - Spin_init
 - Spin_lock
 - Spin_trylock
 - Spin_islock
 - Spin_unlock
 - The spin lock often used in conjunction with rescheduling control variable!
 - Nopreempt_spin_mutex – this automatically uses a rescheduling control variable!

Posix semaphores

- ▶ Sem_init – initializes an unnamed semaphore
- ▶ Sem_open – creates, init, a named semaphore
- ▶ Sem_destroy – remove an unnamed semaphore
- ▶ Sem_unlink – remove a named semaphore
- ▶ Sem_wait – down the semaphor value (--)
- ▶ Sem_post – up the semaphor value (++)
- ▶ Sem_trywait, sem_timedwait, sem_getvalue
- ▶ Link: -Ipthread

Extensions to Posix mutexes

- ▶ Standard Posix mutex functionality:
 - Pthread_mutex...functions
- ▶ Robust mutexes – it can detect whether the previous owner is terminated while holding this mutex(errno=EOWNERDEAD). If the new cleanup of mutex can't be done the errno=ENOTRECOVERABLE.
- ▶ Priority inheritance – boost the mutex owner priority. A thread locks a mutex and an other higher priority thread goes to sleep for that mutex. In this case the priority of sleeper is temporarily transfers to the owner of mutex!

Condition synchronization

- ▶ These functions allows an easy to manipulate cooperating processes!
 - Postwait services – efficient sleep/wakeup/timer mechanism used between cooperating threads.
 - A thread identified with his UKID (Unified global thread Id).
 - Pw_getukid, pw_wait, pw_post,...
 - Server system calls
 - Server_block, server_wake1, server_wakevec

Posix clocks, timers

- ▶ `CLOCK_REALTIME` – the system wide clock, defined in `<time.h>`
- ▶ `CLOCK_MONOTONIC` – the system time measuring the time in seconds and nanosec since the system was booting. It can not be set!
- ▶ `Timespec`, `itimerspec` structures – see details in manual!
- ▶ `Clock_settime`, `clock_gettime`, `clock_setres`
- ▶ Timers: sets a value, and sends a notification when it expires!
 - `Timer_create`, `timer_delete`,`timer_settime`,`gettime`, `nanosleep`,etc.

Local timers, global timer

- ▶ Each CPU has a local (private) timer. It is used as a source of periodic interrupt to that CPU!
 - Cc 100 times per sec
 - Functionality:
 - CPU accounting(eg. For top command, etc)
 - Process time quantum for SCHED_OTHER and SCHED_RR
 - CPU balancing, rescheduling
 - Timing source for Posix timers
 - Local timers can be disabled via shield func!
- ▶ Global system wide timer (only one)- uses Int 0, cannot be disabled.

**Thanks for your
attention!**

zoltan.illes@elte.hu