Operacijski sustavi

Ljiljana Despalatović

Sveučilišni odjel za stručne studije
Sveučilište u Splitu

1. Uvod
   - Osnovni pojmovi
   - Osnovni dijelovi hardvera
   - Funkcije operativnog sustava
   - Vrste operativnih sustava
   - Zadaće operativnog sustava
   - Pokretanje operativnog sustava
   - Dijelovi operativnog sustava

Što je operativni sustav?

Operativni sustav je skup osnovnih sistemskih programa koji upravljaju hardverom i omogućuju izvršavanje aplikacija na računalu.

- ne postoji precizna definicija operativnog sustava

Operativni sustav je program koji se ponaša kao posrednik između korisnika i hardvera (Silberschatz et al).

Operativni sustav je sloj softvera čiji je posao upravljanje hardverom i osiguravanje sučelja preko kojeg aplikacije komuniciraju sa hardverom bez da znaju sve detalje o njemu.

Struktura računalnog sustava

- hardver
  - CPU, memorija, I/O uređaji, sabirnice
- operativni sustav
  - upravlja i koordinira rad korisnika i aplikacija sa hardverom
- aplikativni softver
  - word procesori, web preglednici, kompajleri, baze podataka, igrice
- korisnici

Osnovni dijelovi hardvera

- procesor
- memorija
- ulazno-izlazni uređaji
- sabirnice

Osnovni dijelovi hardvera - procesor

CPU (Central Processing Unit) – dobavlja instrukcije iz memorije i izvršava ih.
- *instruction set* – skup instrukcija koje procesor može izvršiti

Registri:

- programski brojač (*program counter*) – sadrži memorijsku adresu one instrukcije koja je na redu za izvršavanje.
- pokazivač na stek (*stack pointer*) – pokazuje na radni stek u memoriji
- PSW Program Status Word ili flag registar koji se sastoji od bitova koji opisuju razna stanja i događaje (error status field, interrupt enable/disable bit, supervisor/user mode bit).

Osnovni dijelovi hardvera - procesor

- Dva moda rada: user (korisnički) i kernel (jezgrin) mod rada.
- OS radi u kernel modu – ima pristup kompletnom hardveru.
- Korisnički programi rade u user modu.
- Prelazak iz user u kernel mode:
  - sistemski pozivi (system calls)
  - iznimke (exceptions)
  - prekidi (interrupt)

Osnovni dijelovi hardvera - memorija

- **RAM Random Access Memory** – vrlo brza memorija kojoj se može pristupati u proizvoljnom redoslijedu
- **Cache** – mala memorija koja se obično nalazi na ili u blizini procesora. Upravljana je od strane hardvera.
- **Registri** – još manja memorija na procesoru.
- **Hard disk** – spora memorija

Osnovni dijelovi hardvera - memorija

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

Osnovni dijelovi hardvera – ulazno izlazni uređaji

Sastoje se od dva dijela:

- kontroler (controller)
- uređaj (device)

Kontroler je čip ili skup čipova koji prima instrukcije od OS-a i dalje fizički upravlja uređajem.

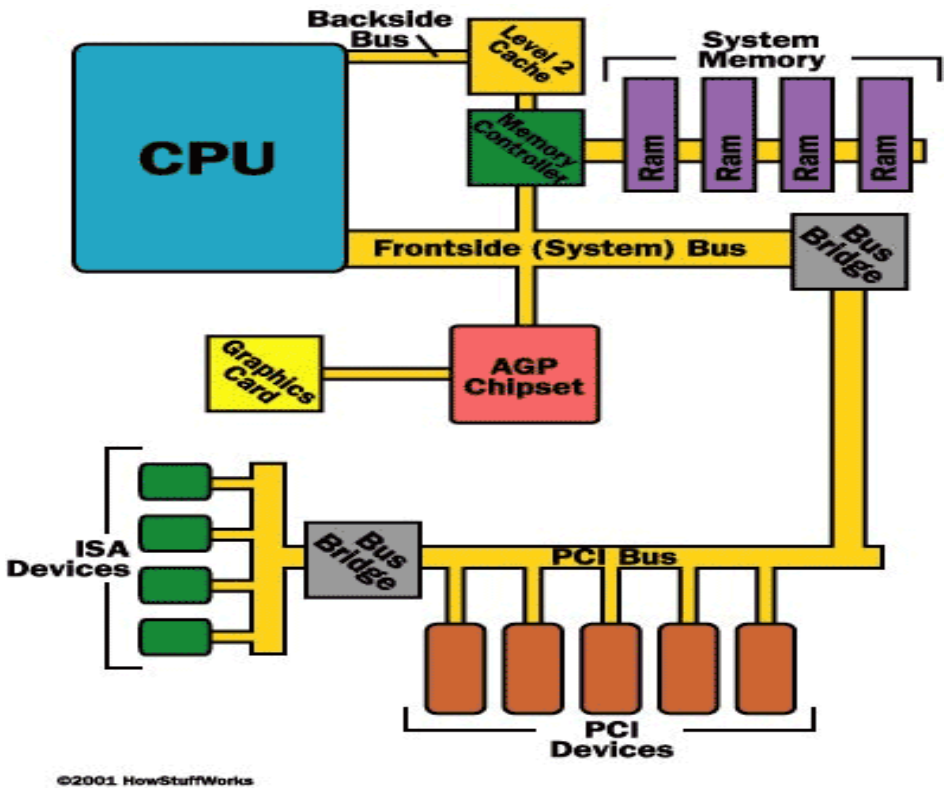
Softver koji komunicira s uređajem zove se **device driver**. Radi u kernel modu.

Osnovni dijelovi hardvera – sabirnice

Sabirnica (bus) je jedan ili više vodiča koji prenose podatke sa izvorišta do odredišta u računalu.
Intel Pentium sabirnice:

- cache, local ili sistem bus, memory, PCI (Peripheral Component Interconnect), SCSI, USB, IDE (*ATA) i ISA (Industry Standard Architecture)

Osnovni dijelovi hardvera – sabirnice



Funkcije operativnog sustava

Dvije su osnovne funkcije operativnog sustava (Tanenbaum: Modern Operating Systems):

- proširenje hardvera ili virtualna mašina
- upravljanje resursima

Operativni sustav kao proširenje hardvera

Operativni sustav krije detalje detalje o radu s hardverom od programera i korisnika (prekidi, upravljanje memorijom, file sistemom,...).

Primjer: za rad sa hard diskom korisnik ne mora znati kako se zapisuje podatak na disk. OS daje prikaz sadržaja diska kao file sistem i skup funkcija za rad sa diskom (čak i preko grafičkog sučelja).

Operatvni sustav kao upravljač resursima

Resursi (memorija, procesor, disk, mrežno sučelje itd.) su djeljivi između više programa, ali oni sami ne moraju odlučivati o tome kad će i kako trošiti resurse. O tome se brine **operativni sustav**.

Vrste operativnih sustava

Podjela prema funkcionalnim osobinama računalnih sustava:

- OS za osobna računala (*desktop systems*) – naglasak na izvođenju (performansama), a ne na raspodjeli resursa
- OS za velike računalne sustave (*mainframe systems*) – naglasak na raspodjeli resursa
- OS za radne stanice (*workstations*) – kompromis između performansi i raspodjele resursa (neke resurse dijele, a neke imaju samo za sebe)
- Ugrađeni OS (*embedded computers*) – od mikrovalne pećnice do rakete – samo jedna namjena – OS optimiziran i pouzdan
- OS za sustave sa dijeljenjem vremena (*time-sharing systems*) – online komunikacija sa poslom i OS-om - bitno vrijeme odziva

Vrste operativnih sustava

- OS za višeprocesorske sustave (*multiprocessor systems* ili paralelni sustavi) – dijele memoriju i sistemski sat
  - **Asimetrični** – jedan procesor za sistemske programe, ostali za ostale procese.
  - **Simetrični** – svi procesori rade sve.
- Distribuirani sustavi (*distributed systems*) – svako računalo ima svoju memoriju, komunikacija preko mreže. Distribuiraju se podaci, datoteke, printeri, ali i procesi.
- OS za ručne uređaje (*handheld*) – slabi hardverski resursi – mala memorija, slabi procesori, mali ekrani
- Operativni sustavi za rad u realnom vremenu (*real-time operating systems (RTOS)*) za upravljanje strojevima, instrumentima, industrijskim postrojenjima. Određeni zadaci uvijek se moraju izvršavati jednako dugo.

Vrste operativnih sustava

Podjela po broju korisnika i procesa:

- jednokorisnički, jednozadaćni OS (single-user single-task) – MS-DOS, PalmOS,
- jednokorisnički, višezadaćni OS (single-user, multitasking) – MS Windows 3.1, 95, 98, me
- višekorisnički, višezadaćni (multi-user, multitasking) – MS Windows XP i dalje, Unixoidi, Mac OS X

Zadaće operativnog sustava

- upravljanje procesorom – efikasnost, na višezadaćnim OS-ovima omogućavanje istovremenog izvršavanja programa
- upravljanje memorijom – odvajanje programa u zasebne memorijske prostore
- upravljanje ulazno izlaznim uređajima
- upravljanje uređajima za pohranu podataka
- API (Application Programming Interface)
- user interface (korisničko sučelje)

Pokretanje operativnog sustava

BIOS (basic input output system) – prvi program (firmware – ugrađen na BIOS chipu – Flash memory chip na matičnoj ploči) koji se pokreće.

- pokreće power-on self test (POST) – provjera hardvera, inicijaliziranje perifernih uređaja i memorije.
- aktivira disk i pokreće bootstrap loader.
- bootstrap loader pokreće specijalne boot programe koji učitavaju opertivni sustav.

Najpoznatiji boot loaderi su:

- LILO (Linux Loader)
- GRUB (Grand Unified Bootloader)
- MS Windowsi imaju svoj “proprietary” boot loader

Pokretanje operativnog sustava

Nakon pokretanja operativni sustav nalazi se u *idle* petlji. Budi se na sljedeće događaje:

- prekidi (*interrupts*) izazvani hardverskim uređajima
- iznimke (*exceptions*) izazvane od strane korisničkih programa
- sistemski pozivi (*system calls*)

Jedini program koji je cijelo vrijeme pokrenut je **kernel**.

Dijelovi operativnog sustava

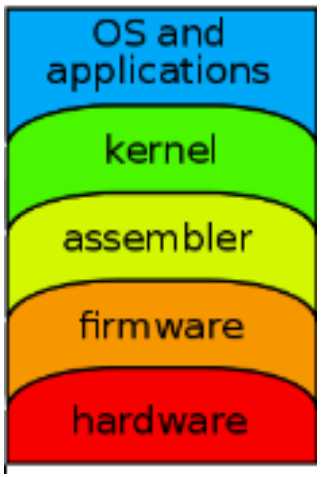
- Kernel ili jezgra operativnog sustava
- Korisničko sučelje
- File system (datotečni sustav)
- Sistemski programi
- API
- Networking

Kernel ili jezgra operativnog sustava

- dio operativnog sustava koji se prvi učita i stoji u zaštićenom dijelu memorije cijelo vrijeme dok kompjuter radi
- osnovne zadaće: upravljanje memorijom, procesorom, file systemom
- pisan za pojedini operativni sustav – MS Windows kernel pisan je za MS Windows OS (proprietary), Linux je kernel GNU/Linux OS-a i ima više verzija i svaka se može modificirati
- Mach kernel sa Carnegie-Mellon University razvijen s idejom da se može koristiti sa bilo kojim OS-om – osnova Macintosh OS X-a.

Kernel ili jezgra operativnog sustava

Postojanje kernela posljedica je dizajna računalnog sustava kao niza slojeva apstrakcije.



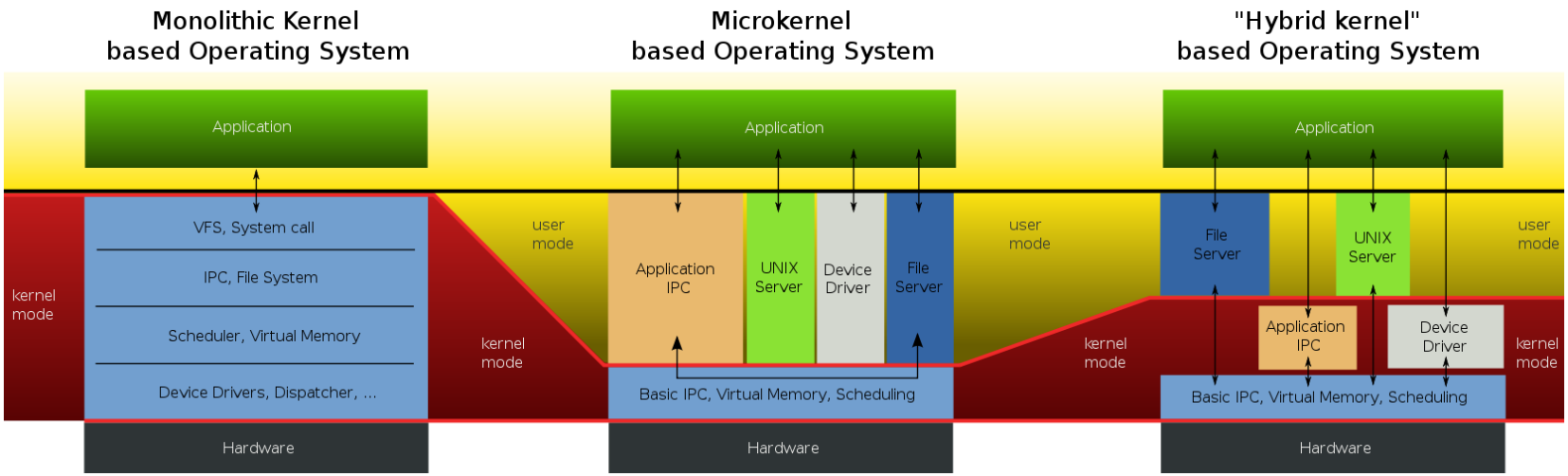
Glavne zadaće kernela

- osiguravanje sadržaja za raspoređivanje procesa (dispaching)
- komunikacija među procesima
- sinkronizacija procesa
- context switching
- manipulacija kontrolnog bloka procesa – process control block (PCB)
- obrada prekida
- kreiranje i uništavanje procesa
- prekidanje i ponovno pokretanje procesa

Vrste kernela

- **monolitni kernel** – glavne funkcije OS i device driveri implementirani su unutar kernela - unixoidi
- **mikrokernel** – kernel obavlja samo najosnovnije poslove (upravljanje memorijom, multitasking, međuprocesna komunikacija), lakši za implementiranje i održavanje, ali sporiji zbog velikog broja sistemskih poziva – Mach OS
- **hibridni kernel** – kombinacija – MS Windows
- **exokernel** – smanjivanje kernela na najmanji mogući, dozvoljeno korisničkim programima da izravno pristupaju hardveru – web poslužitelji

Vrste kernela



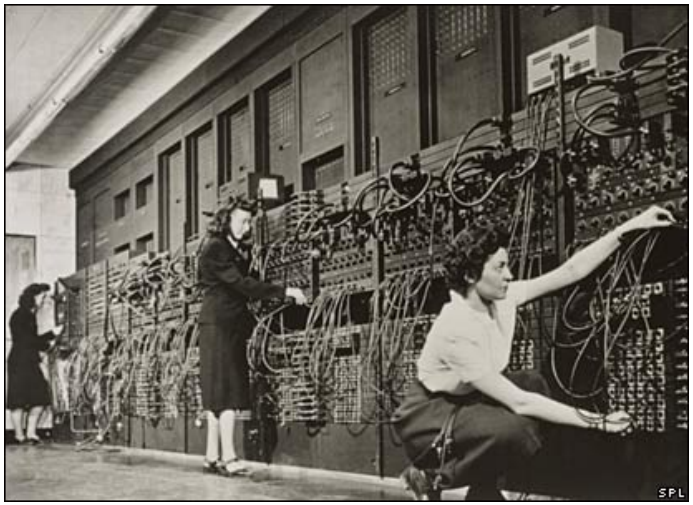
2 Kronologija

Praprapovijest 1930.-40.

- Z1 - Konrad Zuse - prvo elektro-mehaničko binarno programabilno računalo Z1 (Berlin 1936. - 1938.)
  - nastalo u dnevnom boravku
- Colossus - 1943. prvo električno programabilno računalo; Tommy Flowers, za potrebe kriptografije.
- ABC Atanasoff-Berry Computer (1937.-1942.) Iowa State College - prvo digitalno računalo.
  - za rješavanje linearnih jednadžbi
  - nije bilo programabilno

Prapovijest 1945.-1955.

ENIAC - Electronic Numerical Integrator and Computer



- 27 tona, 2.4 m x 0.9 m x 30 m, 167 m2, 150 kW, 18000 vakumskih cijevi
- za potrebe vojske - proračun balističkih tablica
- $500,000 (skoro $6 miliona u 2010)
- što sa OS-om?

1950.-te

- kompjuterski sustavi izvode samo jedan posao istovremeno
- jedan korisnik može zakupiti vrijeme računala, bez obzira da li troši sve resurse
- programi se unose mehaničkim switchevima
- kasnije, uvedene IOCS rutine (Input/Output Control System) - začetak OS-a
- single-user, single-job
- većina resursa je neiskorištena

Batch computing

General Motors Research Labaratories za IBM 701 mainframe

- grupirnje poslova koji se izvode jedan za drugim
- bušene kartice, poslovi odijeljeni kontrolnim karticama koje koriste Job Control Language (JCL)
- kontolne kartice govore računalu da li sljedeće kartice sadrže podatke ili programe, koji je programski jezik korišten, procjenjeno vrijeme izvršavanja

Prvi OS

1956. g. - prvi OS GM-NAA I/O napravio General Motors Research Division za IBM 704 računalo

- IBM 704 je prvi masovno proizvedeni kompjuter (123 komada u 5 godina) sa floating point aritmetikom.
- Za IBM 704 razvijeni su jezici Fortran i Lisp, prvi muzički program MUSIC i program za sintetiziranje govora (text-to-speech) (more info, filmic).

1960.-te

Vrijeme **timesharinga** i **multiprogramiranja**

- više poslova odjednom
- dok jedan posao čeka ulaz/izlaz, procesor može obraditi drugi posao (**multiprogramiranje**)
- zbog toga važna memorija
- programi su se vrtili satima/danima, u slučaju greške sve ispočetka

1959./60.

MAD (Michigan Algorithmic Decoder) baziran na Algolu.

- kompajliranje
- dijagnostika

Timesharing

- simultano korištenje resursa računala
- svaki korisnik dobije svoj odsječak vremena (npr. 2 ms) u kojem računalo obavlja posao za njega, a zatim predaje drugome
- koristi činjenicu da u vremenu u kojem korisnik obavi dvije radnje, računalo može izvršiti hrpu instrukcija za nekog drugog

Razlika multiprogramiranja i time sharinga

Kod **multiprogramiranja** se program izvršava dok ne dođe do neke točke zaustavljanja, npr. ulaz/izlaz, dok kod **time sharinga** svaki posao dobije unaprijed alocirani odsječak vremena.

1960-tih John McCarthy, Robert Fano and Fernando Corbato, MIT – implementiran prvi time-sharing sustav **CTSS** (Compatible Time-Sharing System)

Multics

1969 – nasljednik **Multics** (Multiplexed Information and Computing Service) - Bells Lab, MIT, General Electrics suradnja.

- PL/1 i assembler
- Ideja – sličnost sa mrežom električne energije
- Propao, ali u zadnja instalacija Multicsa ugašena 2000. godine

OS/360

1960-tih pokušaj da se napiše OS za cijelu seriju IBM System/360 mainframe računala.

- obećanje da će programi napisani za jedno računalo iz serije raditi i na ostalima
- budžet od $25 miliona i procjena troška $125 miliona popeo se na $500 miliona i 5000 čovjek-godina
- OS je kasnio godinu dana i imao je hrpu bugova, ali
- nastala cijela familija operativnih sustava i novo područje - softversko inženjerstvo
- Fred Brooks "Mythical Man-Month"
- "*It is very humbling experience to make a multi-million-dollar mistake, but it is also very memorable.*"
- **Brook's Law:** "*Adding manpower to a late software project makes it later.*"

1970.-te

- TCP/IP model (DARPA, 1970.) i Ethernet standard (Xerox PARC, 1973.), omogućili su komunikaciju računala (posebno vojnih i sveučilišnih)
- pitanje sigurnosti podataka i enkripcija
- sigurni OS-ovi su postali prioritet

UNIX

- Multics je dizajniran za mainframe: velik i kompleksan sustav, skup i težak za održavanje i razvoj, pa se Bell Labs povlače iz projekta.
- Ken Thompson i Dennis Ritchie sa vrlo malim budžetom na odbačenom DEC-ovom (Digital Equipment Corporation) minikompjuteru PDP-7 kreiraju prve verzije UNIX-a.
- Prvotno u assembleru, dakle ovisan o hardveru.
- Dennis Ritchie kreira programski jezik C da bi UNIX napisali u višem programskom jeziku i učinili ga portabilnim.
- Pisanje kôda operacijskig sustava u višem programskom jeziku je prekretnica i jedan od najznačajnijih poteza u razvoju operativnih sustava.

1970.-te

- AT&T, čiji je dio bio Bell Labs, do 1983. nije se smio tržišno natjecati sa kompjuterskim kompanijama, pa je UNIX ustupljen sveučilištima po nekoj nominalnoj naknadi i isporučavao se sa izvornim kodom.
- Uskoro razne verzije link.
- Vremenski se podudara sa razvojem mikroprocesora: 1971. Intel 4004, 4-bitni mikroprocesor, $1000

Mikroprocesori

- Prvo OS za računalo sa mikroprocesorom - Altair 8800 za hobiste: bez display-a, tipkovnice, sa malo memorije.
- Ed Roberts i Forrest M. Mims, MITS (Micro Instrumentation and Telemetry Systems) u garaži.
- Programiralo se u binarnom kodu, trzajući switcheve na prednjoj strani.
- Jedini znak izvršavanja programa bila su male neoske žaruljice.

Altair i BASIC

- 1975. Bill Gates i Paul Allen (Micro-soft) pišu BASIC za Altair u 6 tjedana, te ga isporučuju MITS-u, ali ga ne prodaju, već ga licenciraju (royalty).
- Prvi spor Micro-softa.
- Do 80-tih Microsoftov ključni produkt je BASIC interpreter (za ostale sustave, npr. Motorola 6800).

CP/M

- Gary Kildall - prvi OS za osobna računala bazirana na Intel procesorima.
- floppy disk
- osobna računala su jednostavnija - nema više poslova odjednom

Apple

- Apple 1, $666.66, prodano 600 komada

- Apple 2, US$1298] (4 kB RAM), US$2638 (maximum 48 kB RAM)

- Apple DOS operativni sustav, Microsoftov BASIC interpreter

1980.-te

- IBM-ov ulazak na tržište osobnih računala koristeći najbrže procesore u to vrijeme (Intel 16-bit 8088)
- IBM nudi Bill Gatesu da napiše OS za IBM PC.
- Microsoft kupuje QDOS (Quick and Dirt Operating System) od Seattle Computer Products Tim Patersona baziran na Killdalovom CP/M za $50000, mijenja ime u MS-DOS.
- Do 1983 IBM PC postaje standard.
- IBM PC sa 64 KB memorije i floppy diskom prodaje se za $2880.
- Po Time-u Man of the Year: the PC.

Apple Macintosh1984.

- U prosincu 1979. Steve Jobs odlazi u studijski posjet Xerox Palo Alto Researh Center (PARC), gdje se razvija Xerox Star računalo sa GUI (Graphics User Interface).
- 1984. izlazi Apple Macintosh sa GUI.
- Zatvorena arhitektura: ograničen razvoj aplikacija bez suradnje s Appleom. Microsoft (između ostalih) piše aplikacije za Macintosh.

Microsoft Windows

- 1981. kreće razvoj Microsoft OS-a sa GUI-em (nakon posjeta Bill Gatesa Apple-u).
- Windows 1.0 izlazi 1985. g. : 10000 naredbi, 18 programer/godina, $99, potpisana licenca s Applom za korištenje vizualnih karakteristika.
- Procesor Intel 80286 nedovoljno brz za GUI podršku.
- Sa sljedećom generacijom procesora Intel 386 i 486, izlaze Windows 2.0 (uz njih Word i Excel).
- Apple tuži Microsoft za kršenje licence (licenca je pokrivala samo Windows 1.0).
- Najveći rast softverske kompanije ikad.

GNU/Linux

- 1984. godine **Richard Stallman** započinje razvoj GNU operativnog sustava.
- GNU je rekurzivni akronim za "GNU nije Unix".
- Do 1991. godine većina OS je bila završena, ali sa kernelom (Hurd) developeri nisu bili zadovoljni.
- **Linus Torvalds**, 21-godišnjak iz Helsinkija započinje projekt koji će kasnije postati Linux kernel.

Linux kernel

Poruka Linusa Torvaldsa na Usenetu 25.8.1991.

> *Hello everybody out there using minix -*
>
> *I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).*
>
> *I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)*
>
> *Linus (torvalds@kruuna.helsinki.fi)*
>
> *PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.*

- Nakon toga, GNU i Linux programeri integriraju GNU komponente sa Linuxom i tako je nastao potpuni operativni sustav.

GPL

- Stallmanova motivacija za GNU bila je zaključavanje kôda Unixa.
- Njegova ideja je bila napraviti unix-like OS koji će se sastojati od potpuno slobodnog softvera.
- Da bi to omogućio kreira **General Public License (GPL)** - copyleft licencu.

Slobodni softver

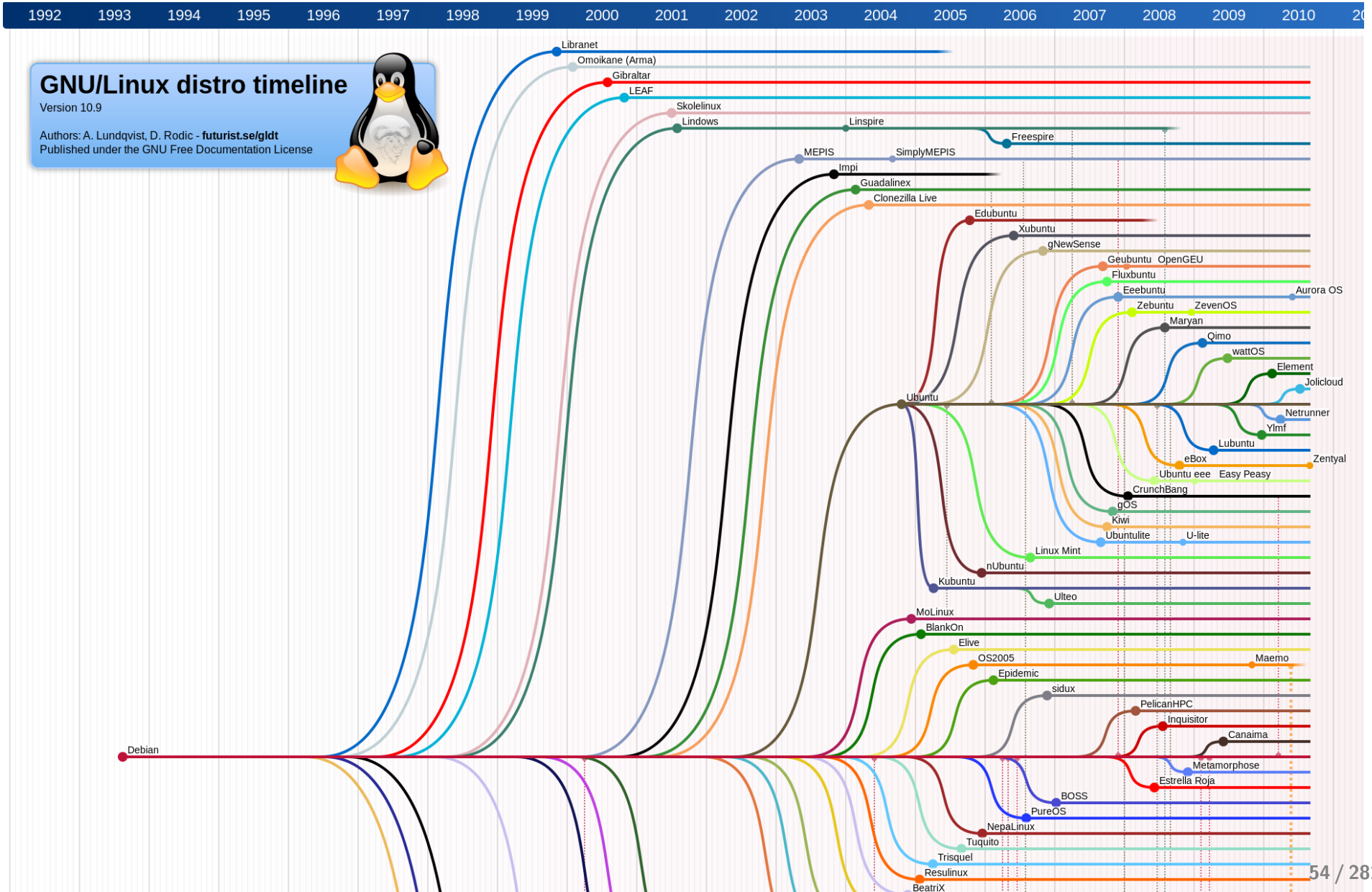
"Free as a freedom, not as free beer."

"Free software" is a matter of liberty, not price. To understand the concept, you should think of "free" as in "free speech", not as in "free beer". Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software. More precisely, it refers to four kinds of freedom, for the users of the software:

- *The freedom to run the program, for any purpose (freedom 0).*
- *The freedom to study how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.*
- *The freedom to redistribute copies so you can help your neighbor (freedom 2).*
- *The freedom to improve the program, and release your improvements to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this.*

Richard Stallman

GNU/Linux danas(cijela slika)



3. Korisničko sučelje
   - Graphical User Interface
   - Command Line Interface

Korisničko sučelje

- Graphic User Interface (GUI)
- Command Line Interface (CLI)

Graphical User Interface GUI - Grafičko sučelje

- GUI omogućava korištenje prozora, ikona, menija, i miša (pointing device) u radu s programima (**WIMP** paradigma).
- Douglas Engelbart je prvi patentirao miša (prototip 1964), a on se koristio na računalu koje je imalo GUI koji je sadržavao prozore (*windows*).
- Prozori se nisu mogli patentirati jer u to vrijeme nije bilo softverskih patenata
- Prva prezentacija prozora održana je 1968, trajala je 90 minuta, kompjuterski sistem je bio umrežen, imao je miša, *hipermediju* (hiperlinkove do teksta, slika i zvučnih datoteka), te video konferenciju.

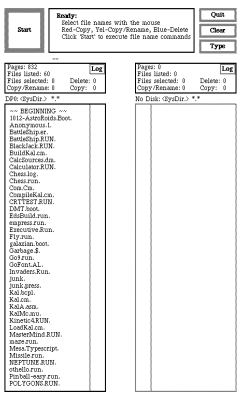
Graphical User Interface GUI - Grafičko sučelje

Graphical User Interface GUI - Grafičko sučelje

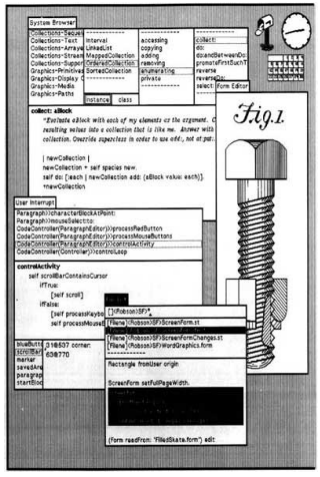
- Predhodnik današnjih modernih GUI-a je GUI na **Xerox Alto** kompjuterima (1973.) napravljen u Palo Alto Research Center (PARC). Xerox Alto je prvi koristio metaforu desktopa i GUI-a.
- U PARC-u je prvo izmišljen laserski printer. Xerox Alto je odgovor na potrebu da se dokumenti za printanje pripremaju u grafičkom obliku.
- Veličina displaya odgovarala je veličini i orijentaciji papira, a rezolucija 606x808.
- Smalltalk kao objektno orijentirani programski jezik i IDE nastao je tada iz potrebe da svi programi imaju sličan user interface.

Graphical User Interface GUI - Grafičko sučelje

Xerox Alto



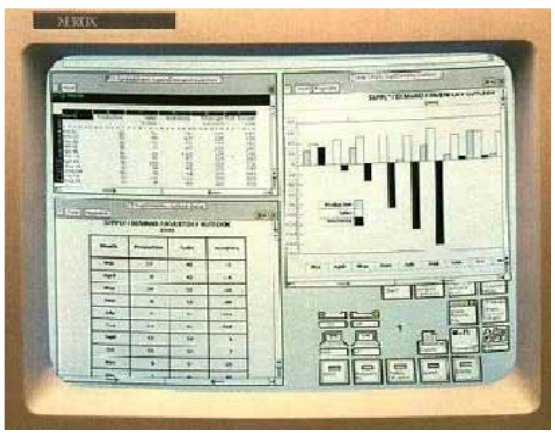
File Manager



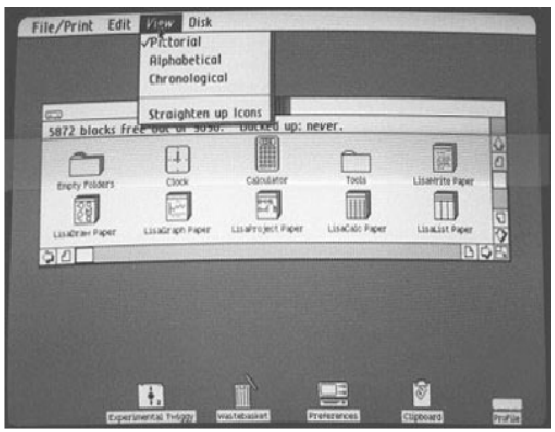
Smalltalk IDE

Graphical User Interface GUI - Grafičko sučelje

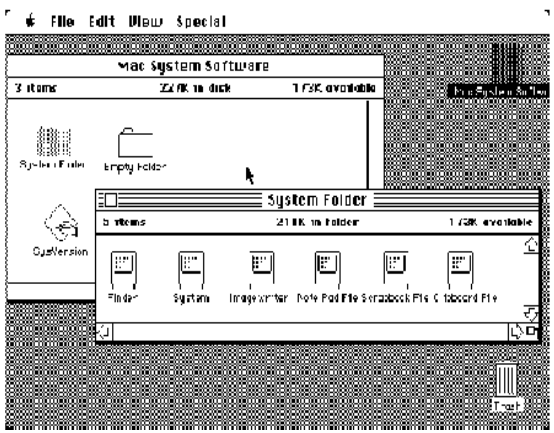
- Nakon njega Xerox Star 8010 (1981., prvi komercijalni kompjuter s GUI-em, US$17000), Apple Lisa (1983.), Apple Macintosh (1984.), Atari ST i Commodore Amiga (1985.)
- Danas još uvijek WIMP paradigma na PC, ali je prisutna i post-WIMP paradigma (handheldi, iPhone OS - više od jednog prsta, igre, govor, pokret, skica interface, Compiz, Desktop Windows Manager).



Xerox Star 8010



Apple Lisa



Apple Macintosh

Graphical User Interface GUI - Grafičko sučelje

- Microsoft Windows je GUI sa CLI "command" shellom.
- Apple Mac OS X ima "Aqua" GUI sa UNIX kernelom ispod sebe i njegovim shellovima.
- Linux/UNIX je CLI sa raznim dostupnim grafičkim sučeljima - Gnome, KDE, Xfce.

Graphical User Interface GUI - X Window System

Na UNIX i Linux operativnim sustavima:

- **X Window System** je konfigurabilan, cross-platform klijent-server sustav za upravljanje grafičkim korisničkim sučeljima (GUI) na računalu ili mreži računala.
- X Window System osigurava skup alatki (protokol) za implementaciju grafičkog sučelja, a programi za upravljanje prozorima (*windows managers*) kao što su KDE i Gnome, koristeći usluge X Windows Servera daju grafičko sučelje.
- X Window System razdvaja funkcije izračunavanja i prikaza (npr. program koji se izvršava na jednom računalu može koristiti ekran drugog računala).
- X Window System je neovisan o operativnom sustavu, ali i o aplikacijama.
- X su jedna od najuspješnijih tehnologija slobodnog softvera (uz Linux i TCP/IP).

Graphical User Interface GUI - X Window System

- Klijent-server odnos u X-ima je drugačiji od uobičajenog klijent-server pristupa (1 server kojem pristupa više klijenata). U X-ima svako lokalno računalo ima X server softver i može pristupiti udaljenim računalima na kojima se vrte X client aplikacije.
- **X server** je program koji upravlja pristupom grafičkoj kartici, displayu i ulaznim uređajima (tastatura i miš).
- **X klijent** je bilo koji aplikativni program. On se prikazuje pomoću X servera, ali je neovisan o njemu.
- Windows manageri i desktop environment su X klijenti.

Command Line Interface

- **Command Line Interface** je alfanumeričko korisničko sučelje za interakciju sa operativnim sustavom ili programima.
- **Command Line Interpreter** je sistemski proces koji prihvaća naredbe korisnika, interpretira ih i izvršava.
- Command Line Interpreter se pokreće u tekstualnim terminalima ili emulatoru terminala na nekom udaljenom računalu.
- Na unixoidima je uobičajeno da je više terminala vezanih uz jedno lokalno računalo (tty1,..,tty7).

Command Line Interface

- Windows i DOS - samo jedan interpreter (COMMAND.COM na MS-DOS i Win9x i cmd.exe na NT familiji Windowsa, poznatiji kao command prompt ili DOS prompt).
- Unix i Linux - više različitih interpretera, moguće preći iz jednog u drugi tokom rada (Bourne shell (/bin/sh), C shell(/bin/csh), Bourne-again shell (/bin/bash), Korn shell (/bin/ksh))
- Windows 7 Windows PowerShell

Trivia: prvi Unix shell zvao se Thompson shell, po Ken Thompsonu

Linux shell

- Lista svih shellova nalazi se u /etc/shells datoteci.
- U kojem sam shellu tj. koji je default shell? Info u /etc/passwd
- Prebacivanje iz jednog shella u drugi moguće je naredbom oblika ime_shella npr. tcsh.

Command Line Interpreter - CLI

- Naredbena linija (command line) je tekst unesen iza odzivnog znaka interpretera *prompta*, praćen pritiskom tipke Enter.
- Naredba može biti **interna**, tj. ugrađena u shell ili **eksterna** tj. poseban program.
- Naredba se sastoji od imena, opcija i argumenata.
- Eksterne naredbe koje su ustvari izvršne datoteke traže se na određenim mjestima u file systemu.
  - DOS i Windows - izvršne datoteke (.bat, .com i .exe) traže se u tekućem direktoriju i na sistemskoj putanji (PATH)
  - Linux - izvršne datoteke (datoteke s pravo pristupa x) traže se na sistemskoj putanji (PATH)
  - inače se ime naredbe mora zadati sa relativnom ili apsolutnom putanjom

Primjer Bourne-again shell BASH

- $ je odzivni znak za korisnika, # za root
- primjer: mkdir, rm, ls, cp, mv, date, wc
- redirekcija na stdin, stdout, stderr ili bilo koju datoteku se izvodi znakovima ¡, ¿, ¿¿ (dodavanje bez brisanja)
- pipeline povezivanje naredbi — - izlaz naredbe s lijeve strane je ulaz u naredbu s desne strane npr. $ ls -l . — wc -l
- koncept okruženja (environment) - naredbe set (lokalne varijable) i export (environment varijable)
- shell programiranje tj. korištenje command interpretera kao programski jezik

Primjer Bourne-again shell BASH

- alternativno ime naredbe **alias** $ alias ll="ls -al"
- ponavljanje naredbene linije **history**, ponavljanje zadnje naredbe !!, ponavljanje naredbe s brojem !broj
- kompletiranje imena naredbe <**Tab**>

4. Datotečni sustavi
   - Datoteka
   - Direktorij
   - Imenovanje datoteka i direktorija
   - Linkovi

Datoteka (**file**)

- Datoteka je imenovani skup podataka (zapisa) pohranjen na uređaju za trajnu pohranu podataka (*permanent storage*).
- Za svaku datoteku postoji zapis s nekim generalnim informacijama (*metadata*) koji se sprema zajedno s datotekom.
- Atributi datoteke su:
  - ime - niz znakova (string)
  - tip
  - lokacija - oznaka diska i putanja do mjesta gdje je datoteka spremljena
  - veličina
  - vlasnik
  - vrijeme zadnje promjene datoteke
  - informacija o pravima pristupa

  Najvažniji atributi datoteke su prava pristupa i lokacija podataka:

Datoteka

"On a UNIX system, everything is a file; if something is not a file, it is a process."
Što se tiče operativnog sustava, najmanje jedna vrsta datoteka mora postojati - **izvršna datoteka**. Naprimjer:

- UNIX: operativni sustav sve datoteke vidi kao slijedove bajtova. Operacije nad njima su pisanje i čitanje. Interpretaciju podataka vrše aplikacije. Kada je datoteka izvršna, aplikacija koja interpretira podatke je sam operativni sustav.
- Macintosh OS: izvršne datoteke imaju dva dijela: jedan sadrži kod, a drugi labele koje koristi user interface. Korisnik može promijeniti labele ne dirajući source. Primjer
- Windows NTFS: datoteke su skup atributa (atribut/vrijednost parova). Ime datoteke, security descriptori, pa i sami podaci su atributi datoteke. Mogu biti imenovani ili neimenovani. Primjer

Operacije nad datotekama

Najvažniji operacije nad datotekama su čitanje i pisanje:

- Open: omogućen pristup datoteci.
- Close: prepuštanje pristupa datoteci.
- Read: čitanje podataka iz datoteka, obično sa tekuće pozicije *current position*.
- Write: pisanje podataka u datoteku, obično sa tekuće pozicije *current position*.
- Append: dodavanje podataka na kraj datoteke.
- Seek: pomicanje tekuće pozicije na određenu poziciju u datoteci.
- Rewind: pomicanje na početak datoteke.
- Set attributes: postavljanje atributa, npr. o pravima korištenja.
- Rename: promjena imena datoteke.

Primjer - Promjena pozicije unutar datoteke

Da bi ostvarili slučajan pristup (random access) podacima u datoteci, koristimo "seek/ set file pointer" sistemski poziv.
To omogućava korisniku da se pozicionira na željeno mjesto u datoteci.
Windows API funkcija za pozicioniranje unutar datoteke je

```
DWORD SetFilePointer(HANDLE hFile,
     LONG lDistanceToMove,
     PLONG lpDistanceToMoveHigh,
     DWORD dwMoveMethod);
```

Parametar `dwMoveMethod` mora biti postavljen na:

- FILE_BEGIN – vrijednost 0, odnosno početak datoteke.
- FILE_CURRENT – trenutna vrijednost file pointer-a.
- FILE_END – vrijednost trenutnog kraja datoteke.

Primjer - Promjena pozicije unutar datoteke

Standardna C funkcija za pozicioniranje unutar datoteke:

```
int fseek(FILE *stream, long offset, int origin);
```

- postavlja trenutnu poziciju u datoteci,
- u slučaju neuspjeha vraća broj različit od nule.

Parametar origin predstavlja početnu točku:

- SEEK_SET – vrijednost 0, odnosno početak datoteke..
- SEEK_CUR – trenutna vrijednost file pointer-a.
- SEEK_END – vrijednost trenutnog kraja datoteke.

Direktorij

- Uređaj (npr. hard disk) možemo podijeliti na particije, koje možemo promatrati kao virtualne odnosno logičke diskove (volumes kod PC-a).
- Particije možemo podijeliti u direktorije.
- Direktorij može sadržavati kako datoteke tako i pod-direktorije.

Postoje različite operacije koje možemo vršiti nad direktorijima:

- pronalaženje datoteke unutar direktorija.
- izlistavanje sadržaja direktorija.
- kreiranje, brisanje, preimenovanje datoteke unutar direktorija.

Direktorij mapira imena datoteka u stvarne lokacije na disku.

Direktorij

Skup imena datoteka (i pod-direktorija) zajedno s pokazivačima na blokove na disku može se implementirati na različite načine u cilju efikasnijih izvršavanja operacija nad direktorijem.

Uobičajene strukture podataka su: povezana lista, hash tablica, B+ stablo (koriste ga NTFS, ReiserFS, XFS datotečni sustavi).

Direktorij

Neke Win32 funkcije za rad s diskovima, mapama i datotekama (za detalje pogledajte u MSDN):

- GetLogicalDrives – Vraća bitmasku koja predstavlja raspoložive diskove u sistemu (npr. ako je disk A postoji bit nula je postavljen).
- GetLogicalDriveStrings – Vraća string koji reprezentira raspoložive diskove
- GetVolumeInformation – Vraća osnovne informacije o specificiranoj particiji.
- GetDiskFreeSpace - Vraća dodatne informacije o specificiranoj particiji.
- GetCurrentDirectory – Vraća trenutni direktorij za proces koji se izvodi.
- SetCurrentDirectory – Postavlja trenutni direktorij za proces koji se izvodi.
- CreateDirectory – Kreira direktorij.
- RemoveDirectory – Briše direktorij (direktorij mora biti prazan).
- FindFirstFile/FindNextFile/FindClose – Skup funkcija koje koristimo za pretragu direktorija po zadanom imenu datoteke.
- SearchPath – Funkcija traži specificiranu datoteku.
- GetFileAttributes/GetFileTimes/GetFileSize/ GetFileAttributesEx – Funkcije za dohvat različitih atributa za datoteku ili direktorij.

Stablasta struktura direktorija

- Struktura se sastoji od čvorova i listova.
- Čvorovi predstavljaju direktorije, dok listovi predstavljaju datoteke.
- Svaki direktorij može sadržavati datoteke ili pod-direktorije.
- Svaki čvor ili list ima samo jednog roditelja.
- Različite datoteke mogu imati ista imena (sve dok su apsolutni putovi različiti).

Imena direktorija i datoteka

- Puno ime datoteke se zove još i pathname. Ono predstavlja "mapu" kroz strukturu direktorija do stvarne lokacije datoteke.

Pravila za imena datoteka, direktorija i pathname kod Win32:

- imena direktorija i datoteka u pathname moraju biti odvojena s backslash '\' znakom;
- pathname mora završavati s NULL;
- imena direktorija i datoteka ne smiju sadržavati '\' i karaktere od 1-31 (ASCII);
- imena direktorija i datoteka mogu sadržavati i mala i velika slova, ali kod pretraživanja nema razlike između njih (case-insensitive);
- kad se točka (.) koristi kao ime direktorija ona predstavlja trenutni direktorij;
- kad se dvije točke (..) koriste kao ime direktorija ona predstavlja roditeljski direktorij trenutnog direktorija;
- kad se točka (.) koristi kao dio imena direktorija ili datoteke ona služi za odvajanje komponenti imena (npr. file extensions - .EXE);
- imena direktorija i datoteka ne smiju sadržavati sljedeće karaktere : '<', '>', ':', '"', '|', '?', '*', '/', '\'

Imena direktorija i datoteka

Konvencije za imenovanje datoteka i direktorija:

- duga imena datoteka i direktorija se ne preporučuju iako u kombinaciji mogu biti duga do 255 znakova
- u nazivima se preporučuju alfanumerički znakovi uz specijalne znakove '-' i '_'
- ostali ne-alfanumerički znakovi su dopušteni ali se ne preporučuju
- nazivi datoteka uobičajeno imaju samo jednu ekstenziju (nastavak) ali mogu imati i više
- nazivi direktorija uobičajeno nemaju ekstenzije ali ih mogu imati

Imena direktorija i datoteka

Pravila za imena datoteka, direktorija i pathname kod Unix/Linux:

- imena direktorija i datoteka su case sensitive;
- imena mogu biti sastavljena od bilo kojeg znaka osim '/'
- preporučuje se korištenje malih i velikih slova, te znakova '.' i '_';
- imena direktorija i datoteka u pathname moraju biti odvojena s slash '/' znakom;
- kad se točka (.) koristi kao ime direktorija ona predstavlja trenutni direktorij;
- kad se dvije točke (..) koriste kao ime direktorija ona predstavlja roditeljski direktorij trenutnog direktorija;
- kad se točka (.) koristi kao dio imena direktorija ili datoteke ona služi za odvajanje komponenti imena, ali točka nije obavezna;
- imena direktorija i datoteka mogu sadržavati sljedeće karaktere, ali se ne preporučuje : '<', '>', '/', '&', ':', '—'.

Linkovi

- Datoteke mogu imati više imena.
- Postoje **hard linkovi** i **soft linkovi** (shortcut) slika.

5. Datotečni sustav (file system)
   - Što je file system?
   - Osnove datotečnog sustava
   - Vrste datotečnih sustava

File system (datotečni sustav)

Datotečni sustav

- Skup metoda i struktura podataka koje koristi operativni sustav za organiziranje i dohvaćanje datoteka na uređaju za spremanje podataka (*storage medium*).
- Sloj operativnog sustava koji transformira blokove podataka sa diska (ili nekog drugog uređaja) u datoteke i direktorije.

Dijelovi datotečnog sustava

- Upravljanje diskom - blokove na disku organizira u datoteke
- Imenovanje - sučelje za pronalaženje datoteka po imenu, a ne po blokovima
- Zaštita podataka
- Pouzdanost i trajnost sustava - oporavak/očuvanje datoteka u slučaju kršenja sustava, napada, problema sa medijem i sl.

File system (datotečni sustav)

Korisnički vs. sistemski pogled na datoteku

- Korisnički pogled
  - stabilne strukture podataka
- Sistemski pogled (system call interface)
  - Kolekcija bajtova (UNIX)
  - Nebitno koja vrsta strukture podataka se želi pohraniti
- Sistemski pogled (unutar OS-a)
  - Kolekcija blokova

  Blok je logička veličina, sektor je fizička veličina. Veličina bloka je uvijek veća ili jednaka veličini sektora (na UNIX-u veličina bloka je 4KB)

Osnove datotečnog sustava

- Da bi se na disku ili particiji na disku mogli pohraniti podaci treba na njima formirati datotečni sustav.
- Datotečni sustav čine:
  - zaglavlje,
  - metapodaci,
  - datoteke i direktoriji.
- Svaki se datotečni sustav treba aktivirati da bi se mogao koristiti. To se zove **mountanje**.
- Moguće je mountati cijeli logički disk (kod Unix/Linux ne postoje logički diskovi već samo direktoriji) ili drugi direktorij.

Virtualni file system VFS

- Objektno orijentiran način realizacije datotečnog sustava.
- Korisnik na isti način pristupa svim datotekama bez obzira kojem datotečnom sustavu one pripadaju.
- Korisnik se, putem sistemskih poziva tj. API-ja obraća virtualnom datotečnom sustavu.
- VFS omogućava formiranje jednog logičkog datotečnog sustava od više fizičkih diskova.

Zadaće datotečnog sustava

Datotečni sustav čuva sljedeće podatke:

- koji blokovi pripadaju kojoj datoteci
- u kojem poredku su blokovi koji formiraju datoteku
- koji su blokovi slobodni (za alokaciju)

Strukture podataka potrebne za realizaciju datotečnog sustava

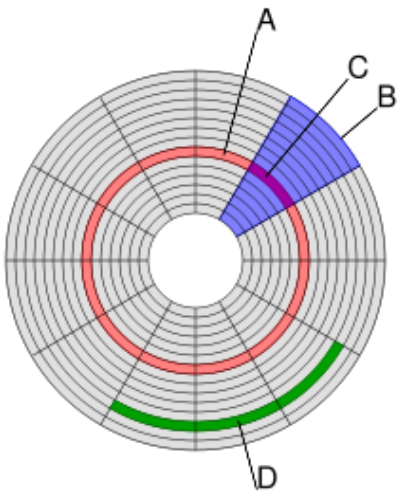
- MBR (**Master Boot Record**) je prvi sektor na disku (sektor 0). On sadrži tablicu particija tj. prvu i zadnju adresu svake particije i bootstrap loader.
- BCB (**Boot Control Block**) je prvi blok aktivne particije (ili boot particije). Sadrži informacije potrebne za podizanje operativnog sustava.
- Kontrolne strukture za dodjeljivanje datoteka (strukture kojima se određuju logički blokovi u kojima je sadržaj): Tabela indeksnih čvorova (**inode table**), FAT (**File Allocation Table)** tabela i MFT (**Master File Table)** - ovisno o datotečnom sustavu.
- Direktorijske strukture koje sadrže kontrolne blokove datoteka
- Kontrolni blokovi datoteka *file control blocks, FCB* koji sadrže atribute datoteka i opis prostornog rasporeda tj. pokazivače na blokove dodjeljene datoteci.

Osnovni pojmovi

- **Sektor** je dio diska (na slici (C)) tipične veličine 512B na magnetnim diskovima i 1024B na optičkim (iako noviji hard diskovi imaju veličinu sektora i 4096B tj. 4KB).
  - Sektor je najmanja jedinica podataka do koje operativni sustav može pristupiti.
- **Blok** ili **cluster** je niz od jednog ili više uzastopnih sektora. Veličinu definira kernel. Može biti veličine sektora ili veličine stranice (često 4KB).

```
sudo dumpe2fs /dev/sda7 | grep 'Block size'
```

  - Blok je najmanja veličina koja se može alocirati za datoteku.



Slika: Struktura diska: A-track, B-geometrical sector, C-track sector, D-cluster

Osnovni pojmovi

- Hardver blok: sector
  - osnovna hardverska jedinica, tipično 512 byte

```
sudo blktool /dev/sda7 sector-sz ili less /sys/block/sda/queue/hw_sector_size
```

- Filesystem blok: block (ili cluster)
  - alokacijska jedinica

```
sudo dumpe2fs /dev/sda7 | grep 'Block size'
```

- Kernel buffer cache: block
  - veličina buffera koji kernel koristi za keširanje sektora pročitanih sa blok uređaja (cluster size je višekratnik block size)
- Partition table block: cylinder
  - koristi samo tablica particija (partition table) i BIOS

Unix/Linux datotečni sustavi

Na Unixoidnim operativnim sustavima podržani su razni domaći (native) i strani (foreign) datotečni sustavi:

- **minix** - najstariji, i verovatno najpouzdaniji domaći UNIX datotečni sustav. Maksimalna veličina minix sustava datoteka je 64 MB, a imena datoteka ne mogu biti duža od 30 znakova;
- **xia** - modificirana varijanta minixa, ukida limite na veličinu sistema datoteka i broj znakova u imenu datoteke, ali ne donosi nove mogućnosti. Rijetko se koristi, ali se smatra da je pouzdan;
- **ext2** - Linux second extended, jako popularan datotečni sustav visokih performansi. Ovaj datotečni sustav je nekoliko godina predstavljao "Linux default";
- **ext3** - može se gledati kao "ext2 + journaling". Potpuno je kompatibilan sa prethodnom verzijom (ext2), tako da se nadogradnja ostvaruje jednostavnim kreiranjem dnevnika.
- **ext4**
- **ReiserFS** - journaling datotečni sustav solidnih performansi. U odnosu na ext3 brži je pri radu sa malim datotekama, ali je relativno nestabilan pri radu sa velikim datotekama.

Unix/Linux datotečni sustavi

Dodatno, na UNIX-u postoji podrška za nekoliko tipova stranih sistema datoteka čime je omogućena relativno laka razmjena datoteka sa drugim operativnim sustavima.

- **msdos** - omogućava razmjenu datoteka sa DOS i OS/2 FAT datotečnim sustavom. Može se aktivirati za čitanje i pisanje (read-write).
- **umsdos** - proširenje msdos sustava datoteka pod Linux-om. Dodata je podrška za duga imena datoteka, vlasništvo, prava pristupa, linkove i specijalne datoteke.
- **vfat** - proširenje FAT sustava datoteka sa većim kapacitetom poznato pod imenom FAT32. Većina Windows 9x/ME sistema koristi FAT32.
- **iso9660** - standard za CD-ROM sustave datoteka.
- **hpfs** - OS/2 High Performance File System
- **ntfs** - Windows NT datotečni sustavi.
- **nfs** - UNIX mrežni datotečni sustav.
- **jfs** (za IBM AIX)
- **xfs** (za SGI Irix)
- **HFS+** (za Mac OS X)

Windows datotečni sustavi

Windows:

- **FAT** File allocation table. FAT12, FAT16 (8.3 dužina imena datoteke), FAT32, exFAT ili FAT64 (nekompatibilan sa ranijim verzijama Windowsa).
- **NTFS** Podržava journaling, hard linkove, datoteke s prazninama (sparse files) itd.
- **WinFS** Windows future storage je datotečni sustav baziran na relacijskoj bazi podataka. Trebao je izaći sa Vistom, ali do danas nije izašao.
- **ReFS Re**silient **F**ile **S**ystem (kodno ime Protogon?) za sada na Windows Server 2012

Plan 9 from Bell Labs

Plan 9 (9P file system protokol) je još jedan jako bitan datotečni sustav. Razvili su ga i radili na njemu tvorci Unixa, C i C++-a od 80-tih. Plan 9 je ustvari distribuirani operativni sustav temeljen na konceptima:

1. Sve je datoteka (i network connections, i procesi i windowsi itd.)
2. pristup datotekama se obavlja preko protokola 9P,
3. korisnici stvaraju svoj pogled na sustav.

"Want to use a different machine's sound card? Import its /dev/audio. Want to debug processes that run on another machine? Import its /proc. Want to use a network interface on another machine? Import its /net. And so on."
Nije zaživio. "Worse is better".

Datotečni sustavi sa dnevnikom transakcija (**journaling**)

- Dnevnik transakcija prati aktivnosti vezane za promjenu meta-podataka, odnosno i-node tabele, i objekata sustava datoteka.
- Dnevnik (journal, log) se ažurira prije promjene sadržaja objekata i prati relativne promjene u sustavu datoteka u odnosu na posljednje stabilno stanje.
- Transakcija se zatvara po obavljenom upisu i može biti ili u potpunosti prihvaćena ili odbijena.
- U slučaju oštećenja, izazvanog npr. nepravilnim gašenjem računala, datotečni sustav se može lako rekonstruirati povratkom na stanje posljednje prihvaćene transakcije.
- ext3, ReiserFS, XFS, JFS, NTFS datotečni sustavi.
- U ext3 datotečnom sustavu prisutna su tri režima vođenja dnevnika transakcija: **journal, ordered i writeback**.

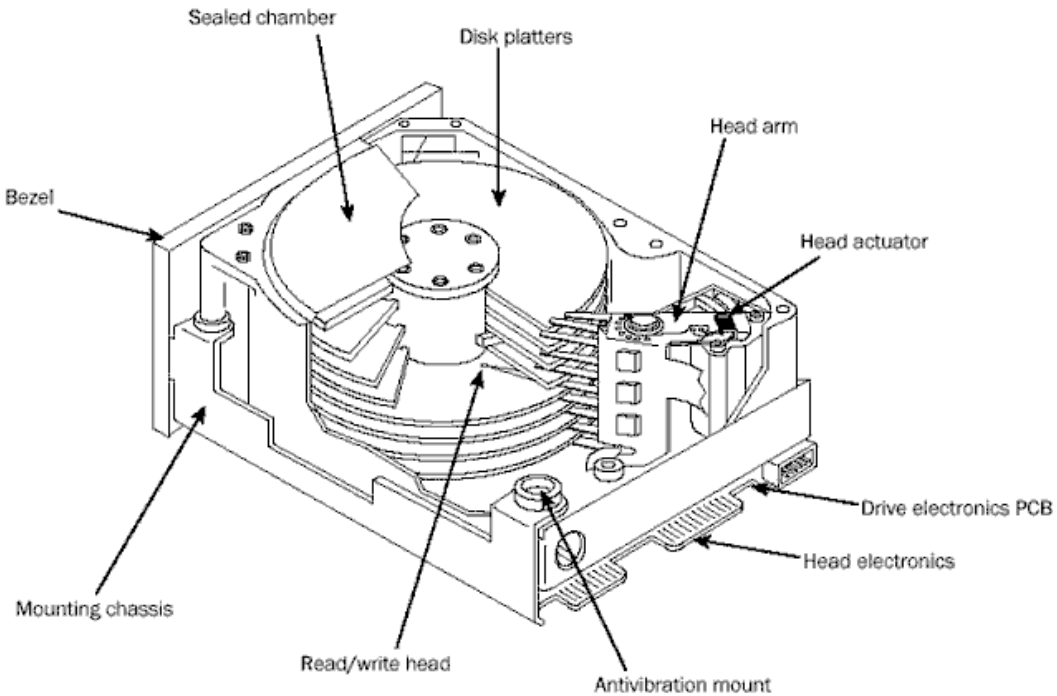
6 Organizacija rada diska

- Hard disk
- Algoritmi za dodjelu diska procesima
- Alokacija prostora na disku

Hard disk

- **Hard disk** ili magnetski disk je za dva reda veličine jeftiniji od RAM-a po byte-u i često dva reda veličine veći po dimenzijama.
- Vrline - jako velik i jeftin memorijski kapacitet.
- Mane - sporost pristupa podacima.
- Disk se sastoji od jedne ili više metalne ploče koje rotiraju brzinama 5400, 7200 ili 10800 okretaja u minuti.
- Magnetska glava za čitanje/snimanje kreće se na sličan način onome kod ručice gramofona od vanjske strane prema unutra.
- Površina diska je podijeljena na **staze (tracks)** koje čine koncentrične krugove s početkom na vanjskom rubu diska.
- Staze su podijeljene na **sektore**, svaki sektor obično sadrži 512 byte-a.
- Blok je najmanja jedinica koja se može alocirati za datoteku. Može biti veličine sektora do veličine stranice (često 4KB). Veličinu definira kernel.

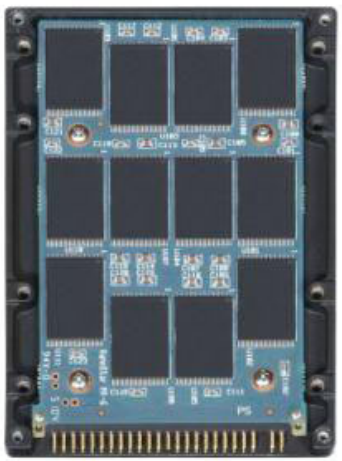
Hard disk



Slika: Tradicionalni hard disk

Hard disk vs Solid State

Traditional hard disk drive



Solid state hard drive

Solid state drive SSD

Vrline:

- Brzina pristupa (no latency).
- Nema pomičnih dijelova - otpornost.

Mane:

- Cijena
- Do izbrisanih podataka se ne može lako doći (vrlina i mana).

Organizacija rada diska

- Zapisivanje na disk povlači pozivanje operativnog sustava. Zahtjev za pisanje/čitanje s diska pošalje se *device driveru*.
- Ako disk ne radi ništa (*idle*) operacija može početi odmah.
- Ako je disk zaposlen, program (proces) koji treba I/O operaciju ide na čekanje.
- Ukupno vrijeme koje će program (proces) potrošiti je:
  - Vrijeme potrebno za prebacivanje iz user u kernel mode i obratno (*overhead time*).
  - Vrijeme potrošeno u redu čekanja (*queueing time*).
  - Vrijeme potrebno da disk dođe na pravu stazu i sektor (*latency time*).
  - Vrijeme potrebno za zapisati/pročitati podatke s diska (*transfer time*).

Algoritmi za dodjelu diska procesima

Kako programi (procesi) dolaze na red:

- **Shortest Seek Time First (SSTF)** U odnosu na trenutnu poziciju glave diska nađi najbližu poziciju tj. proces koji treba pisati/čitati najbliže trenutnoj poziciji glave. Ovaj algoritam je dobar, ali nije fer. Neki procesi mogu izgladnjivati *starving*.
- **First In First Out (FIFO)**. Programima (procesima) se dodjeljuje disk onako kako dolaze. Nema izgladnjivanja, ali je pristup disku slučajan.
- **Elevator Algorithm**. Odaberemo zadaću koja je najbliža u jednom smjeru. Kad dođe do kraja krene u drugom smjeru. Izbjegavao izgladnivanje, ali nije fer - zadaće koje traže I/O blizu sredine češće dolaze na red.
- **One-Way Elevator Algorithm**. Kad dođe do kraja diska glava se ne pomiče u suprotnom smjeru već se vraća na početak.

Primjeri dodjele diska procesima

Predpostavimo da se glava diska nalazi na sektoru 53. Gornjim algoritmima odredi udaljenost (**seek distance**)(tj. put koji pređe glava diska) ako su zahtjevi procesa zapisivanje na sektore: 95, 193, 45, 56, 29, 180, 220.

SSTF: Redoslijed pristupa disku 53, 56, 45, 29, 95, 180, 193, 220.

seekdistace = abs(56-53)+abs(45-56)+abs(29-45)+abs(95-29)+abs(180-95)+abs(193-180)+abs(220-193)=221

FIFO: Redoslijed pristupa disku: 53, 95, 193, 45, 56, 29, 180, 220.

seekdistace = abs(95-53)+abs(193-95)+abs(45-193)+abs(56-45)+abs(29-56)+abs(180-29)+abs(220-180)=517

Primjeri dodjele diska procesima

EA: Redoslijed pristupa disku: 53, 56, 95, 180, 193, 220, 45, 29.

seekdistace = abs(56-53)+abs(95-56)+abs(180-95)+abs(193-180)+abs(220-193)+abs(45-220)+abs(29-45)=358

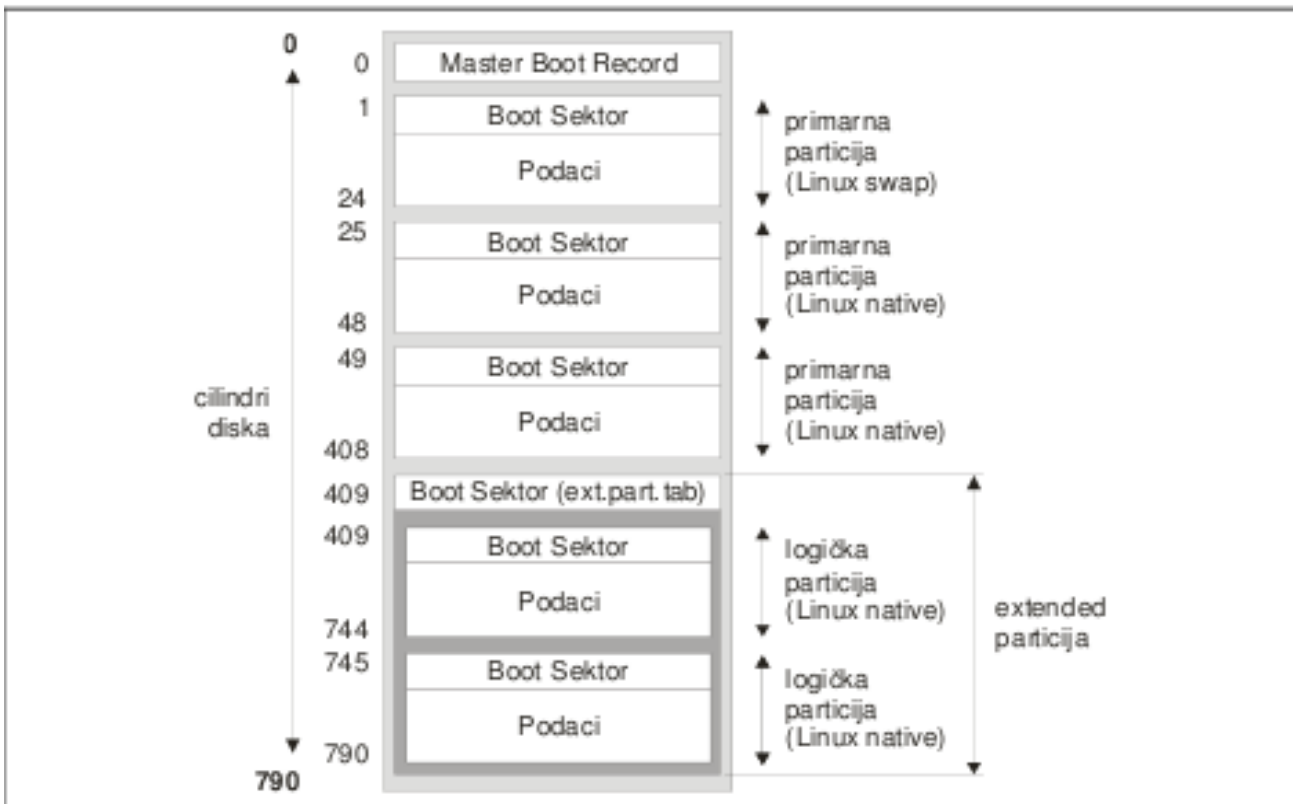
OWEA: Redoslijed pristupa disku: 53, 56, 95, 180, 193, 220, 29, 45.

seekdistace = abs(56-53)+abs(95-56)+abs(180-95)+abs(193-180)+abs(220-193)+abs(29-220)+abs(45-29)=374

Podjela diskova na particije

- Informacije o svim particijama čuvaju se u prvom logičkom sektoru tj. prvom sektoru prve staze sa prve površine diska **Master Boot Record MBR**.
- BIOS pristupa MBR tijekom bootanja i pokreće program koji očitava tablicu particija i očitava prvi sektor aktivne particije (boot sektor).
- U boot sektoru se nalazi mali program čijim pokretanjem počinje bootstrap tj. punjenje RAM memorije operativnim sustavom.
- Najviše 4 fizičke particije po jednom disku, ali jedna može biti *extended* tj. služi kao okvir u kome može biti nekoliko logičkih particija.

Podjela diskova na particije



Podjela diskova na particije na unixu/linuxu

- `fdisk` je program za rukovanje particijama.
- Svaka primarna, extended i logička particija predstavljena je specijalnom datotekom u direktoriju /dev.
- Brojevima 1-4 označavaju se primarne i extended, a većim od 5 logičke.

Uređaj	Primarne particije	Logičke particije
IDE Primary Master	/dev/hda[1-4]	/dev/hda[5-16]
IDE Primary Slave	/dev/hdb[1-4]	/dev/hdb[5-16]
IDE Secondary Master	/dev/hdc[1-4]	/dev/hdc[5-16]
IDE Secondary Slave	/dev/hdd[1-4]	/dev/hdd[5-16]
Prvi SCSI disk	/dev/sda[1-4]	/dev/sda[5-16]
Drugi SCSI disk	/dev/sdb[1-4]	/dev/sdb[5-16]
Treći SCSI disk	/dev/sdc[1-4]	/dev/sdc[5-16]
Četvrti SCSI disk	/dev/sdd[1-4]	/dev/sdd[5-16]

Interna i eksterna fragmentacija

- **Interna fragmentacija** je pojava kada je dio diska alociran, a neiskorišten za podatke.
- Na primjer datoteka počinje na početku clustera ili bloka, a dio između zadnjeg byte-a datoteke i zadnjeg byte-a clustera je neiskorišten.
- Taj dio se zove *slack space* ili *file slack*.
- **Eksterna fragmentacija** je pojava kada se disk s vremenom podijeli na manje dijelove.
- Događa se uslijed alokacije ili dealokacije prostora za datoteku. Slobodni ostaju mali dijelovi diska.

Upravljanje slobodnim blokovima

Struktura podataka koja predstavlja skup slobodnih blokova na disku:

- **Bit vektor**: za svaki blok čuva se jedan bit (1 - slobodan, 0 - zauzet). Primjer: slobodni blokovi 2, 5, 13, 14, 15, bit vektor 0010010000000111
  - Prednost - lako je pronaći slobodan blok, a prostor za predstavljanje bit vektora je zanemariv u odnosu na veličinu diska. Može se keširati u memoriji.
  - Mana - mora se često zapisivati natrag na disk.
- **Vezana lista**: svaki slobodni blok sadrži pokazivač na sljedeći blok.
  - Prednost - lako je doći do prvog slobodnog bloka.
  - Mana - kako bi se došlo do više slobodnih blokova, treba slijediti linkove po cijelom disku, što smanjuje performanse.

Alokacija prostora za datoteku

Osnovne napomene:

- većina datoteka mijenja svoju veličinu s vremenom
- za datoteke kojima se pristupa sekvencijalno trebalo bi alocirati i prostor na disku sekvencijalno (radi poboljšanja performansi)
- za datoteke kojima se pristupa slučajno (npr. baze podataka), trebalo bi isto tako osigurati pristojne performanse
- datoteke se stalno stvaraju i uništavaju - to može dovesti do eksterne fragmentacije diska
- pristup diskovima je jako spor - što više informacija treba biti keširano u memoriji

Alokacija prostora za datoteku - uzastopna alokacija

Datoteci se dodjeljuje određeni broj uzastopnih blokova na disku.
Prednosti:

- Lako za implementirati - pretražuje se lista slobodnih blokova i traži se odgovarajući broj slobodnih blokova. Kad su pronađeni, označe se kao zauzeti.
- Dobro podržava i sekvencijalni i slučajni pristup datoteci.
- Najefikasnija metoda što se tiče brzine operacija s diskom.

Mane:

- Eksterna fragmentacija.
- Kako unaprijed znati maksimalnu veličinu datoteke?
- Što kad datoteka premaši dodijeljenu veličinu? Kopiranje u dvostruko veći prostor uz oslobađanje starih blokova.

Alokacija prostora za datoteku - povezana alokacija

Svaki blok koji je dodijeljen datoteci sadrži pokazivač na sljedeći tj. vezana lista blokova

Prednosti:

- Izbjegnuta je eksterna fragmentacija.

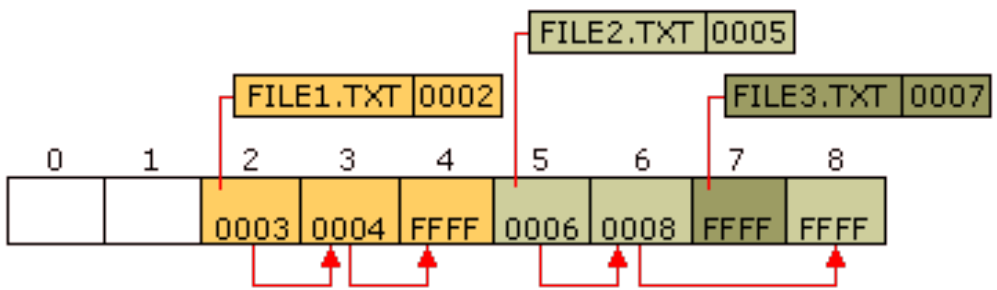
Mane:

- Loš pristup za slučajan pristup datoteci. Da bi se pristupilo slučajnom zapisu, treba proći sve blokove.
- Neotpornost na greške i nepouzdanost - ako je jedan pokazivač izgubi cijela datoteka je neupotrebljiva.

Alokacija prostora za datoteku - primjer - FAT tablica

FAT tablica sadrži pokazivače na svaki blok, te informaciju da li je blok slobodan ili zauzet.

- Broj sljedećeg bloka(clustera) u listi.
- Specijalni znak **EOC** (end of clusterchain) koji označava kraj liste.
- Oznaka za loš cluster (bad cluster).
- Oznaka za rezervirani cluster.
- Nula za cluster koji se ne koristi.



Slika: FAT tablica

Alokacija prostora za datoteku - primjer - FAT tablica

Prednosti:

- Alokacija se svodi na pronalaženje prvog bloka u FAT tablici koji je slobodan.
- Brisanje datoteke - prolazak kroz sve pripadajuće blokove u FAT tablici i označavanje da su slobodni.
- Tablicu se može brzo pretraživati kod slučajnog pristupa datoteci.

Mane:

- Da bi smanjili veličinu tablice, povećala se veličina bloka, a to vodi do interne fragmentacije.
- Ograničena je veličina datoteke.
- Kako se FAT tablica kešira u memoriju, problem s oporavkom u slučaju pada OS-a.
- Nema zaštite od neovlaštenog pristupa datotekama (svi podaci su u FAT tablici).

7 Struktura nekih datotečnih sustava
- UNIX datotečni sustavi
- Windows datotečni sustavi

UNIX datotečni sustavi

- UNIX sve sustave datoteka promatra na isti način, bez obzira na tip, da li su lokalni ili na mreži.
- Svaki datotečni sustav je nezavisna hijerarhijska struktura objekata (direktorija i datoteka) na čijem se vrhu nalazi root (/).
- Objektima se pristupa preko apsolutne ili relativne putanje i imena objekta.

Vrste datoteka

Objekti UNIX sustava datoteka su:

- **Regularne** datoteke.
- **Direktoriji** tj. datoteke (čvorovi stabla) koje sadrže datoteke ili direktorije. Direktoriji sadrže barem roditeljski direktorij (..) i tekući direktorij (.).
- **Linkovi** tj. hard linkovi i simbolički linkovi.
- Specijalne datoteke koje opisuju **uređaje** (`/dev/sda4`) ili **pseudouređaje** (/dev/random).
- Virtualne datoteke koje **postoje samo u memoriji** i sadržavaju informacije o sustavu (`/proc`).
- **Imenovani pipeline** (datoteke kojima mogu pristupiti dva procesa - jednosmjerna komunikacija FIFO) i **socketi** (dvosmjerna komunikacija) koriste se za interprocesnu komunikaciju.

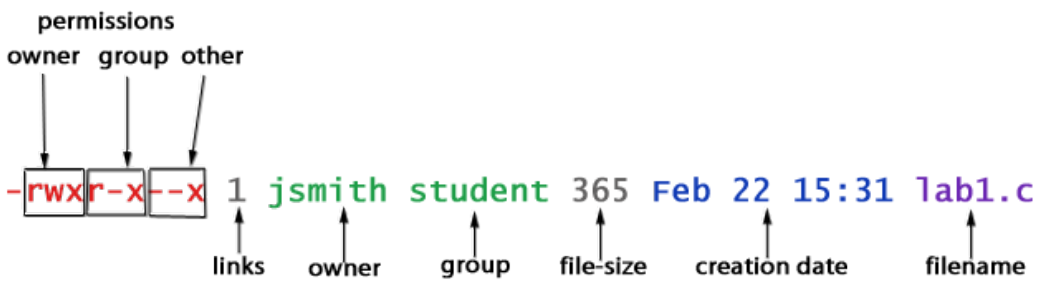
Vrste datoteka

oznaka	opis	primjer
-	regularna datoteka	`/etc/crontab`
d	direktorij (ili folder)	`/etc`
l	simbolički link	ln -s stvarna_datoteka simb_dat
c	character device	`/dev/console` ili `/dev/null`
b	block device	`/dev/sda4`
p	named pipe	`mkfifo moj_pipe`[1]
s	unix-domain socket	`/tmp/.X11-unix/X0` ili `/dev/log`

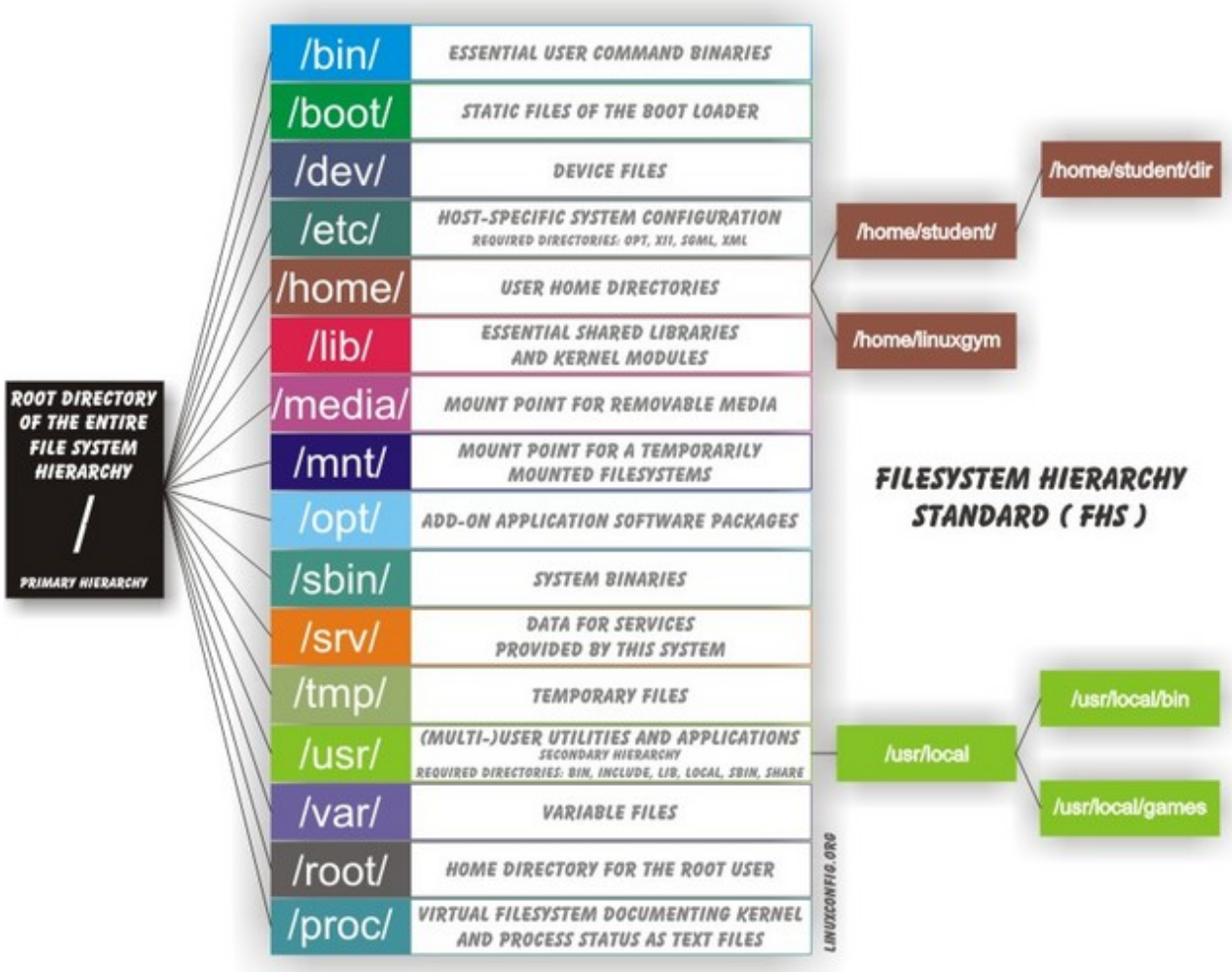
Prilikom izvršavanja naredbe `ls -l` (duži ispis direktorija) prvo slovo u ispisu označava tip datoteke.

[1] u jednoj konzoli napišite `ls -l > moj_pipe`, a u drugoj `cat < moj_pipe`

Prava pristupa



Aktivno stablo direktorija



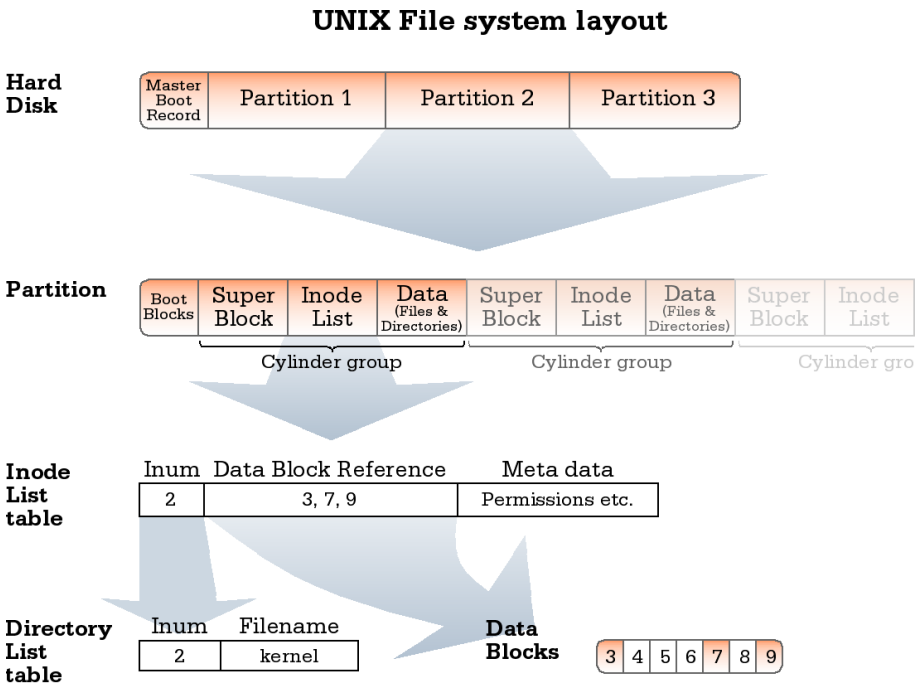
UNIX datotečni sustavi

UNIX datotečni sustav je kolekcija datoteka i direktorija sa sljedećim svojstvima:

- Ima root direktorij (/) koji sadrži sve druge datoteke i direktorije. Može imati i `lost+found` direktorij u kojem smješta sve fileove koji su ostali bez roditelja prilikom pada sustava (system crash).
- Svaka datoteka ili direktorij je jedinstveno određena svojim imenom, direktorijem u kojem se nalazi i jedinstvenim identifikatorom (do na particiju) koji se zove **inode** (`ls -i`).
- inode broj se koristi za pronalazak ostalih informacija o datoteci u i-node tablici.
- Samostalan je. Nema ovisnosti između dva sustava datoteka.

Zašto `/proc` i `/sys` imaju isti inode broj (`tree -L 1 --inodes /`)?
Zašto `/home` i `/boot` mogu imati isti inode broj?

UNIX datotečni sustavi



UNIX datotečni sustavi

UNIX datotečni sustav čine:

- zaglavlje ili superblock
- tabela indeksnih čvorova (i-node),
- data blok
  - blokovi s podacima (data blocks),
  - blokovi direktorija (directory blocks),
  - blokovi indirektnih pokazivača (indirection block).

UNIX datotečni sustavi

Superblock sadrži info o konfiguraciji sustava datoteka. Primjerice (detalji ovdje)

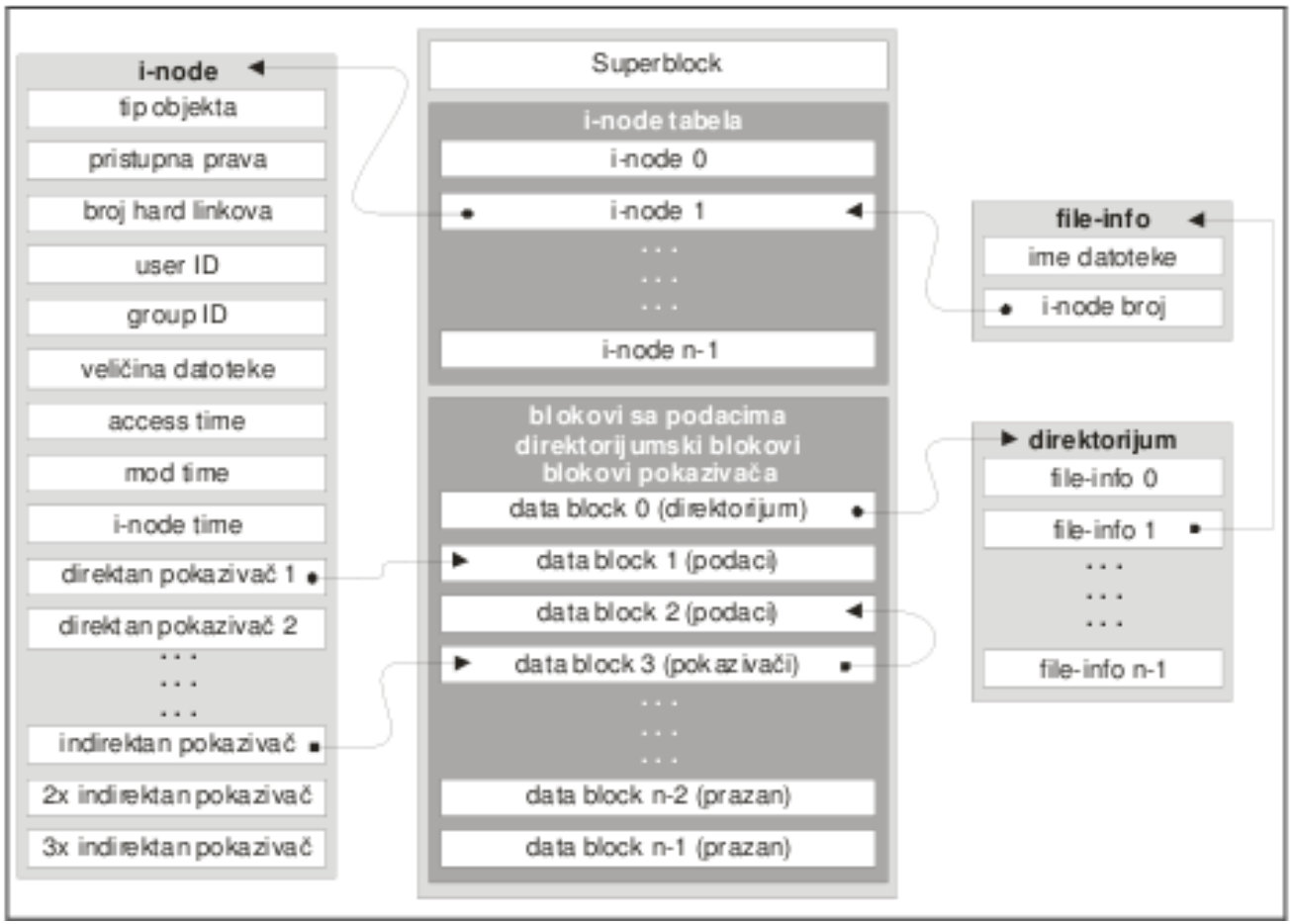
- ukupan broj inode-ova i blokova u datotečnom sustavu
- broj slobodnih blokova
- kada je sustav mountan, kada je modificirana
- koji OS je kreirao datotečni sustav

Superblock je drugi blok na disku (prvi je boot block, ali on nije dio datotečnog sustava).
Kopije superbloka nalaze se na više mjesta u datotečnom sustavu: `dumpe2fs /dev/sda4 | grep -i superblock`

UNIX datotečni sustavi

```
dumpe2fs /dev/sda4 | grep -i superblock
dumpe2fs 1.41.14 (22-Dec-2010)
Primary superblock at 0, Group descriptors at 1-1
Backup superblock at 32768, Group descriptors at 32769-32769
Backup superblock at 98304, Group descriptors at 98305-98305
Backup superblock at 163840, Group descriptors at 163841-163841
Backup superblock at 229376, Group descriptors at 229377-229377
```

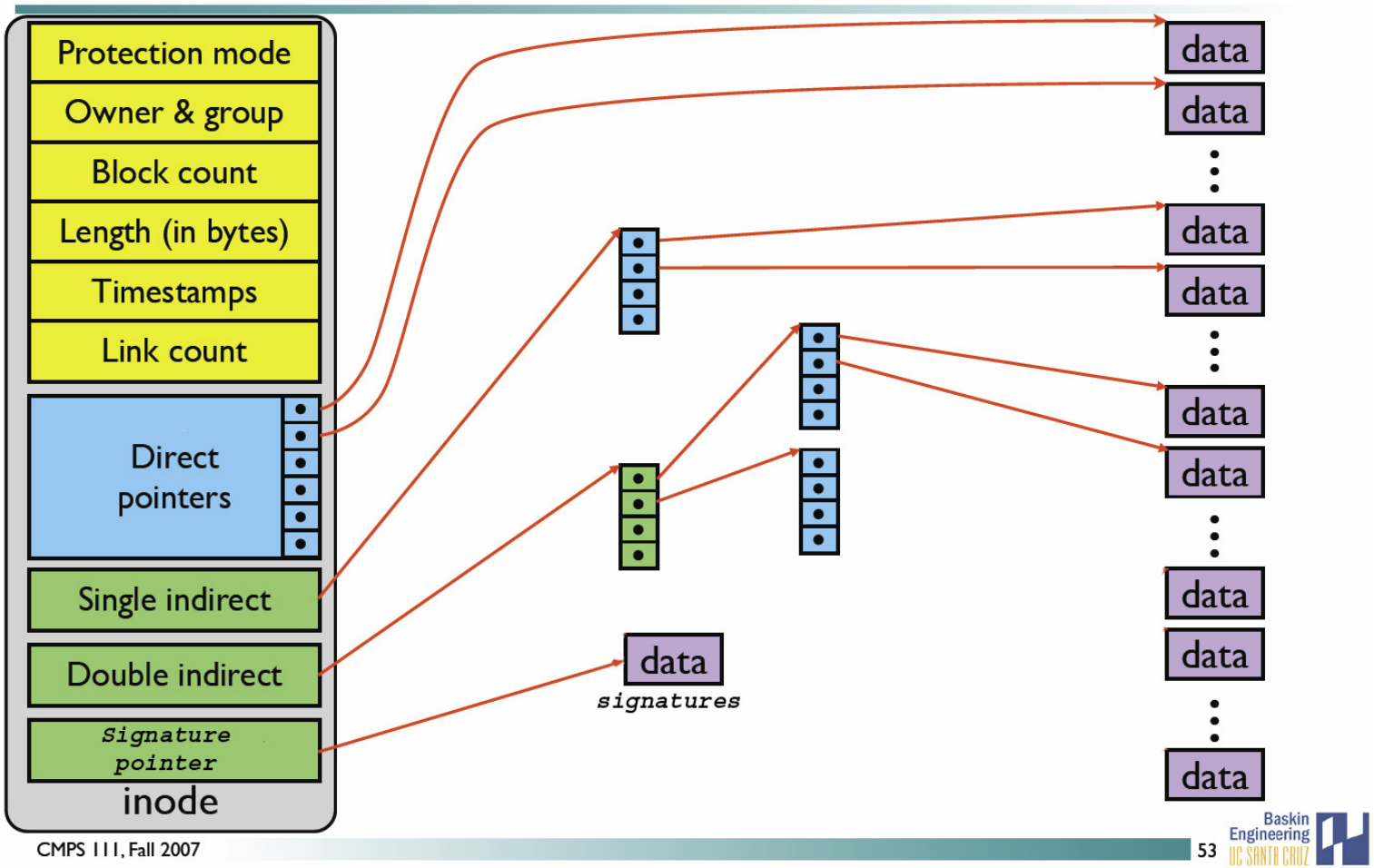
UNIX datotečni sustavi



I-node

- I-node (indeksni čvor) je osnovna struktura UNIX sustava datoteka koja u potpunosti opisuje jedan objekt.
- Dio je kontrolnog bloka datoteke (File Control Block FCB).
- I-node sadrži sve informacije o objektu osim imena.
  - tip objekta i pristupna prava,
  - broj hard linkova na dani objekt,
  - user ID, tj. ID korisnika kojoj objekat pripada,
  - group ID, tj. ID grupe kojoj objekt pripada,
  - veličinu objekta u byte-ovima,
  - vrijeme zadnjeg pristupa (access time),
  - vrijeme zadnje promjene objekta (modification time),
  - vrijeme zadnje modifikacije i-noda
  - listu direktnih pokazivača na blokove s podacima
  - listu indirektnih pokazivača (lista pokazivača na jednostruke, dvostruke i trostruke indirektne blokove)

I-node


Protection mode
Owner & group
Block count
Length (in bytes)
Timestamps
Link count
Direct pointers
Single indirect
Double indirect
Signature pointer
inode
data
signatures
data
data
data
data
data
data
data
data
data
data
CMPS 111, Fall 2007
53
Baskin Engineering
UC SANTA CRUZ

Maksimalna veličina datoteke

max file size = (# direktni pokazivači + jednostruki indirektni + dvostruki indirektni + trostruki indirektni) * veličina bloka

jednostruki indirektni pokazivači = veličina bloka / veličina pokazivača
dvostruki indirektni pokazivači = jednostruki indirektni pokazivači * jednostruki indirektni pokazivači
trostruki indirektni pokazivači = jednostruki indirektni pokazivači * dvostruki indirektni pokazivači

Veličina bloka na ext4 je tipično 4kB. Veličina pokazivača je 4B ili 8B.

ext4

- Veličina bloka:

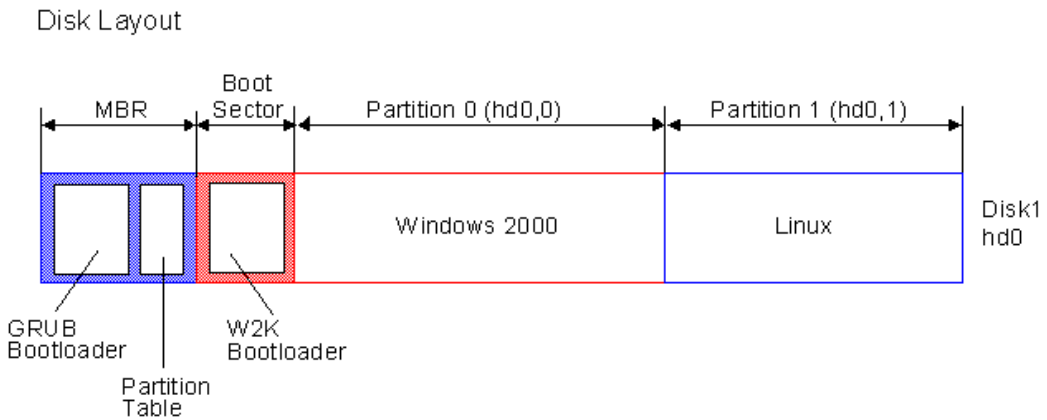
```
sudo dumpe2fs /dev/sda7 | grep 'Block size'
```

- Strukture
  - Direktoriji: vezana lista, B-stablo s hashiranjem
  - Loši blokovi (*bad blocks*): Tablica
  - Alokacija: Extents/Bitmap
- Limiti
  - Max file size 16 TB (for 4k block filesystem)
  - Max number of files 4 billion (specified at filesystem creation time)
  - Max filename length 256 bytes
  - Max volume size 1 EB (currently limited to 16TB because of mke2fs limitation)
- Dozvoljeni su svi znakovi osim NULL ('\0') i '/' u imenu datoteke.

Kreiranje i aktiviranje/deaktiviranje sustava datoteka na UNIX-u

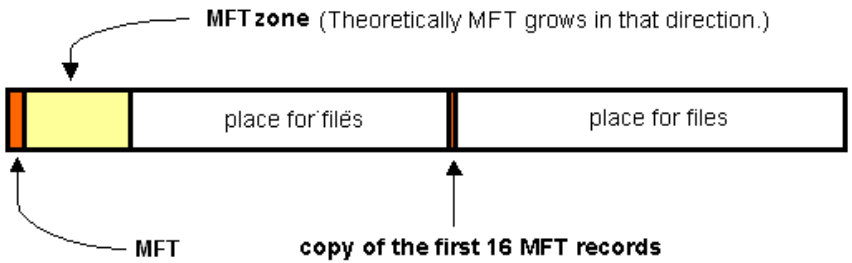
- Za kreiranje sustava datoteka koristi se program **mkfs** (ili mke2fs).
- Za aktiviranje(deaktiviranje) sustava datoteka koristi se program mount(umount).
- Mountiranjem sustava datoteka stvara se virtualni datotečni sustav koje čine svi aktivirani datotečni sustavi.
- Primjer: #mount /dev/sda1 /home.
- /etc/fstab ili file system table je konfiguracijska datoteka u kojoj se nalazi lista svih diskova i particija i opis kako će se mountati.
- Datotečni sustavi mora biti u statusu *clean* da bi ga se moglo mountati. Ako dođe do pada sustava, datotečni sustav postaje zaprljan *dirty*

Primjer izgleda diska za dual boot



- MBR (Master boot record), 512 byte sektor na početku hard diska sadrži naredbe nužne za bootanje. BIOS pokreće MBR. MBR uključuje tabelu particija, pomoću koje loada i pokreće boot record particije koja je označena active flagom.
- Više detalja ovdje.

NTFS



- **Master File Table MFT** je tablica koja sadrži informacije (metapodatke) o svim datotekama (ime, lokacija, veličina, dozvole) i direktorijima (imena datoteka i ID datoteka).
- Dvije kopije MFT - na početku particije i na sredini.
- MFT je podijeljena na zapise fiksne duljine (obično 1 KB) i svaki zapis odgovara jednoj datoteci.
- Prvih 16 podataka (metapodataka) su sistemski zapisi odgovorni za neki aspekt rada diska (housekeeping). Postoji kopija tih podataka (a prva tri zapisa su kopirana točno na sredinu diska).

NTFS MFT

Metadatoteke u MFT počinju znakom $:

$MFT	opisuje sam MFT
$MFTmirr	kopija MFT-a (na sredini diska)
$LogFile	koristi se oporavak od rušenja (journaling)
$Volume	housekeeping information (ime volume-a, verzija datotečnog sustava i sl.)
$AttrDef	lista standardnih atributa datoteka
$.	root direktorij
$Bitmap	bitmapa slobodnog prostora na volume-u
$Boot	boot sektor (bootable partition bootstrap kôd)
$Upcase File	tablica Unicode znakova za slova koja se pojavljuju u imenima datoteka (potrebna zbog sortiranja i pretraživanja).
file.ext	dio sa zapisima o datotekama

NTFS MFT

- Datoteka se sastoji od niza atributa, a atribut se satoji od parova: zaglavlje i vrijednost.
- Sadržaj datoteke je samo vrijednost jednog atributa ($DATA).
- Neki od atributa dani su u sljedećoj tablici:

ID	ime	opis
16	$STANDARD_INFORMATION	standardne informacije (vrijeme kreiranja, pristupa i modifikacije, vlasnik i sl.)
48	$FILE_NAME	ime datoteke (Unicode), vrijeme kreiranja, pristupa i modifikacije
64	$OBJECT_ID	ID datoteke ili direktorija (16 byte)
80	$SECURITY_ DESCRIPTOR	prava pristupa i sigurnosni atributi datoteke
128	$DATA	Sadržaj datoteke
192	$SYMBOLIC_LINK	Informacije o soft linkovima

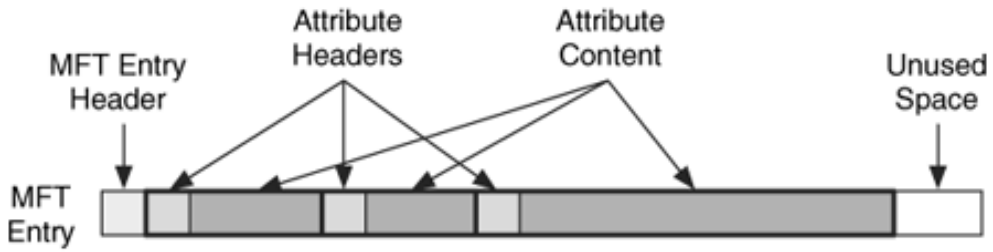
NTFS datoteka

- **Zaglavlje atributa** opisuje tip atributa, veličinu i ime. Sadrži i flagove da li vrijednost atributa komprimirana ili enkriptirana.
- **Vrijednost atributa** može biti bilo kojeg formata ili veličine (pa i veće od veličine MFT zapisa) npr. vrijednost atributa $DATA je sadržaj datoteke.
- Ako je datoteka veća od MFT zapisa, vrijednost atributa sa podacima se dohvaća preko adrese klastera van MFT tablice (extent).
  - Rezidentni atributi se nalaze u MFT tablici.
  - Nerezidentnim atributima se vrijednost zapisuje u klasterima van MFT tablice.
- Male datoteke se mogu potpuno zapisati u MFT tablici.[2]

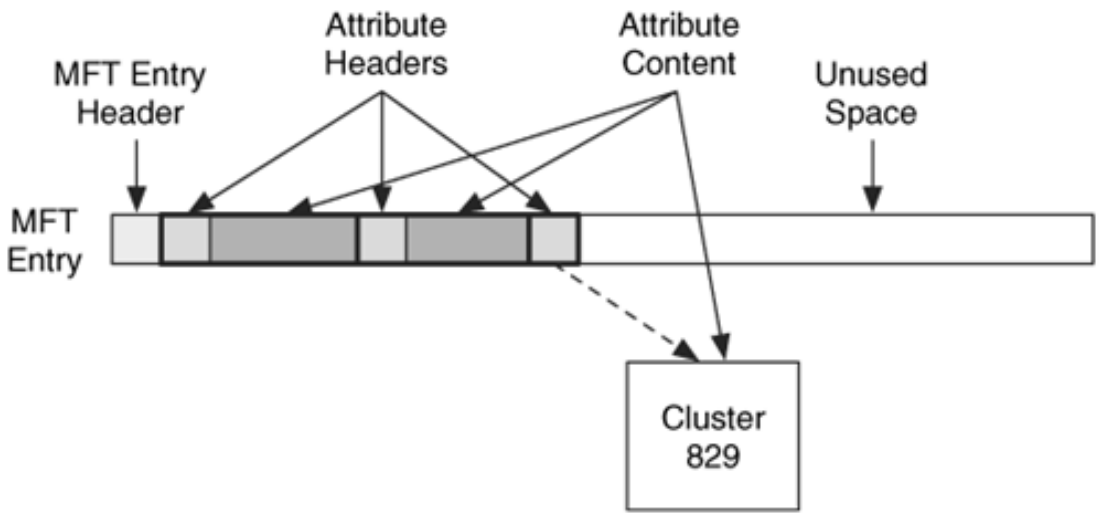
[2] Više detalja http://flylib.com/books/en/2.48.1.88/1/ i http://www.pcguide.com/ref/hdd/file/ntfs/files_Files.htm

NTFS MFT

MFT zapis za malu datoteku:



MFT zapis za malu datoteku:



NTFS

- Strukture:
  - Directory contents B+ tree[2]
  - File allocation Bitmap
  - Bad blocks $badclus
- Limits:
  - Max file size Theoretical: 16 EB minus 1 KB ($2^{64} - 2^{10}$). Implementation: 16 TB minus 64 KB ($2^{44} - 2^{16}$)
  - Max number of files 4,294,967,295 ($2^{32} - 1$)
  - Max filename length 255 UTF-16 code units
  - Max volume size $2^{64} - 1$ clusters in theory. MBR disks only support partition sizes up to 2 TB (2^{40})
- Allowed characters in filenames
  - In Posix namespace, any UTF-16 code unit (case sensitive) except U+0000 (NUL) and / (slash).
  - In Win32 namespace, any UTF-16 code unit (case insensitive) except U+0000 (NUL) and '<', '>', ':', '"', '—','?','*','/','\'

8 Korisnički (user) i jezgrin (kernel) mod rada procesora

Korisnički (**user**) i jezgrin (**kernel**) mod rada procesora

Procesor radi u dva različita moda:

- **Kernel** mode
  U kernel modu kod koji se izvršava ima pristup hardveru, tj. može izvršiti bilo koju procesorsku instrukciju i pristupiti bilo kojoj memorijskoj lokaciji.
  U kernel modu rade najniže (low level) funkcije operativnog sustava.
- **User** mode
  U user modu kod koji se izvršava ne može direktno pristupiti hardveru nego mora pozvati sistemske API funkcije da to naprave za njega.

Mod u kojem se nalazi procesor određen je jednim bitom u **Program Status Word** (PSW) registru.

Kernel mode

Kernel mode se često naziva supervisor mode, master mode ili privilegirani mode. Izvršavanje u kernel modu znači:

- može se izvršiti bilo koja instrukcija,
- može se pristupati bilo kojoj adresi u memoriji,
- može se postaviti (set) programski brojač,
- mogu se omogućiti ili onemogućiti prekidi izazvani s hardverske strane,
- može se direktno pristupiti hardveru.

U kernel modu najčešće rade upravljački programi *device drivers* i razni sistemski programi.

User mode

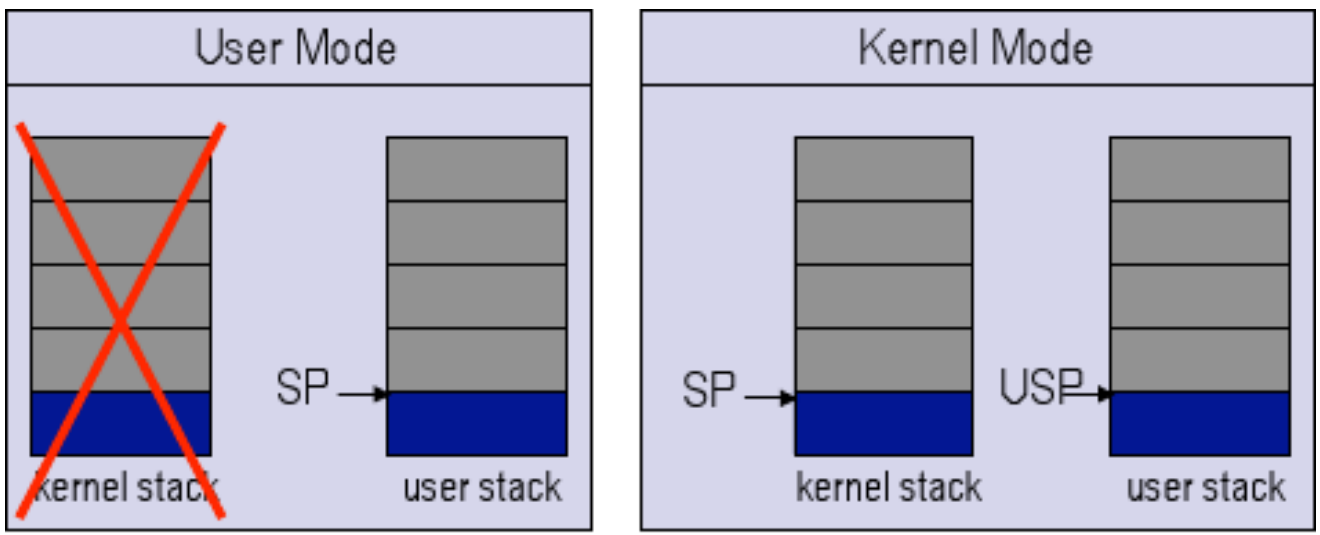
User (korisnički) mode se često naziva restricted mode ili slave mode. Kod koji se izvršava u user modu

- ima ograničen pristup resursima,
- ne može izvršavati sve instrukcije procesora.

U user modu se izvršavaju korisničke aplikacije i dijelovi operativnog sustava.

Userland

- Operativni sustav obično razdvaja virtualnu memoriju na **user space** i **kernel space**.
- **User space** je dio memorije koji pripada aplikacijama koje se izvršavaju u user modu.
- **Userland** je termin koji se koristi za aplikacije koje se izvršavaju u user space-u.



Minimalni kontekst procesora

- Procesor može biti prekinut u izvršavanju (npr. od strane hardvera ili pozivom određene instrukcije).
- Njegovo stanje tj. sadržaj svih registara koji treba biti pohranjen zove se **minimalni kontekst procesora**.
- Minimalni kontekst procesora smješta se na **stack** u kernel space-u.

Prelazak iz user u kernel mode

Postoje tri suštinska načina(događaja) kako se prelazi iz korisničkog u jezgrin mod:

- prekidi (**interrupts**)
- iznimke (**exceptions** ili **trap**)
- sistemski pozivi (**system calls**)

Kada se operativni sustav učita, on puni *exception table vector*, u kojem su pokazivači na kod koji će se izvršiti ukoliko se dogodi neka iznimka ili prekid.

Prekidi (**interrupts**)

Prekidi su asinkroni događaji koje izazivaju različiti hardverski uređaji da bi notificirali procesor o specifičnom događaju koji mora biti obrađen.
Nekad se zovu asinkrone iznimke (*exceptions*).
Primjer:

- pritisak tipke na tipkovnici
- pomicanje miša
- istek vremenskog intervala procesora
- završetak pisanja ili čitanja na disk

Prekidi(**interrupts**)

Što se događa prilikom obrade prekida?

- Procesor poziva OS na određenoj adresi (interrupt handler), postavlja se kernel (supervizor) bit procesora (tj. Program Status Word registra) i ulazi se u kernel mod.
- Sprema se minimalni kontekst procesora i povratna adresa na stack u kernel stack.
- Identificira se uređaj i određuje tip zahtjeva.
- Poziva se odgovarajuća procedura (interrupt handler) koja se treba izvršiti.
- Nakon što se procedura izvrši obnavlja se stanje korisničke aplikacije koja je prekinuta, odnosno sa steka se učitavaju registri i u program counter (PC) se sprema sačuvana povratna adresa (kod 80x86 se poziva IRET naredba).
- Resetira se kernel (supervizor) bit i vraća se iz jezgrinog moda u korisnički mod.
- Korisnički program se nastavlja točno na mjestu na koje je prekinut.

Iznimke(**exceptions**)

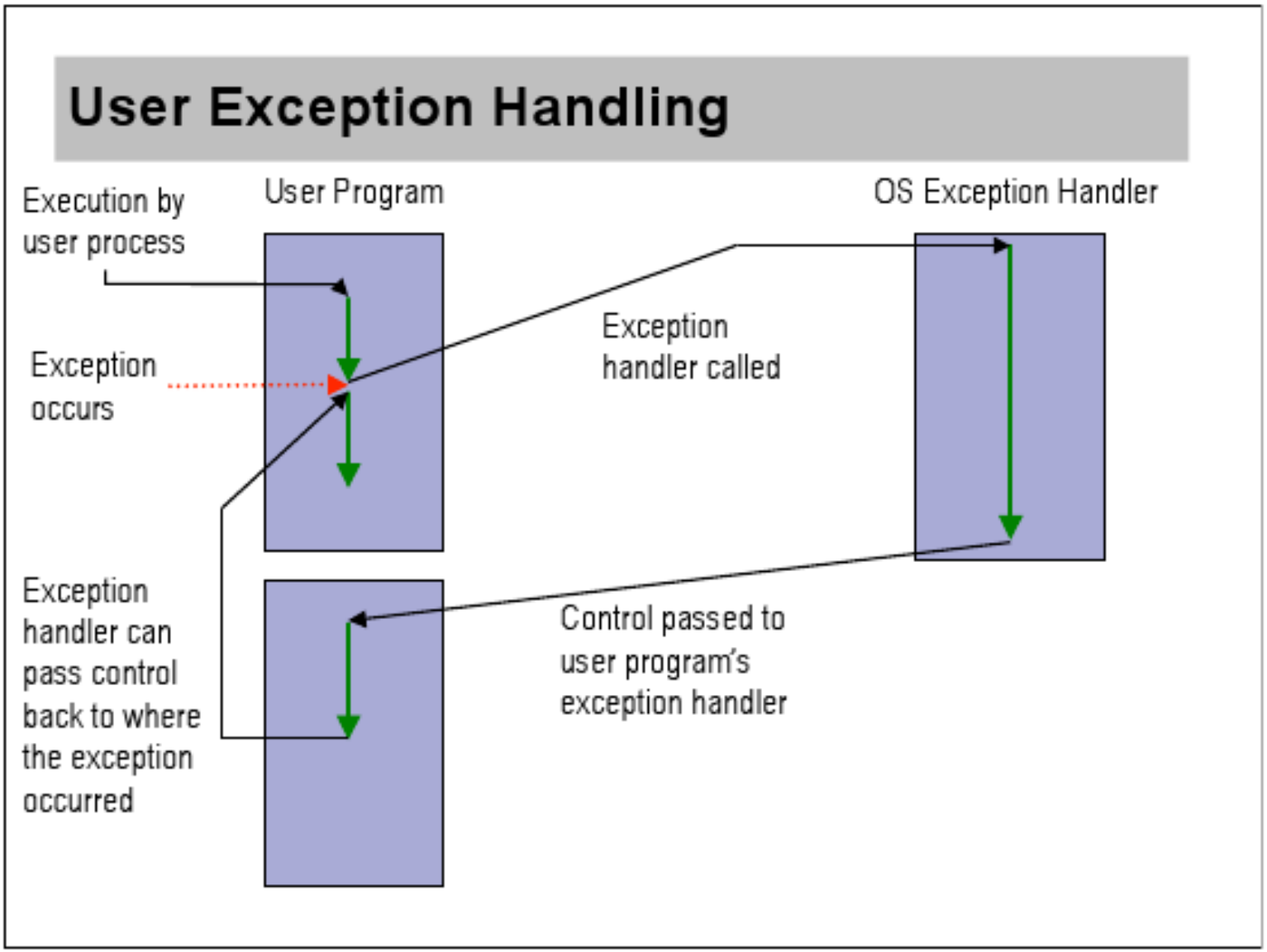
- Iznimke su sinkroni automatski izazvani događaji koji se javljaju kao posljedica nepredviđenog/izvanrednog izvršavanja programa.
- Npr. pokušaj dijeljenja broja sa nulom izazvati će odgovarajuću iznimku.
- Iznimke su sinkroni događaji zato što će se uvijek dogoditi prilikom izvršavanja neke aplikacije (naravno uz iste ulazne podatke).
- Tipični primjeri su:
  - prekoračenje steka (*stack overflow*)
  - prekoračenje maksimalne vrijednosti (*floating point overflow*)
  - nedozvoljen pristup memoriji (*memory access violation*)
  - nedozvoljena instrukcija (*invalide opcode*)
  - točka prekida (*debugger breakpoint*)

Iznimke(**exceptions**)

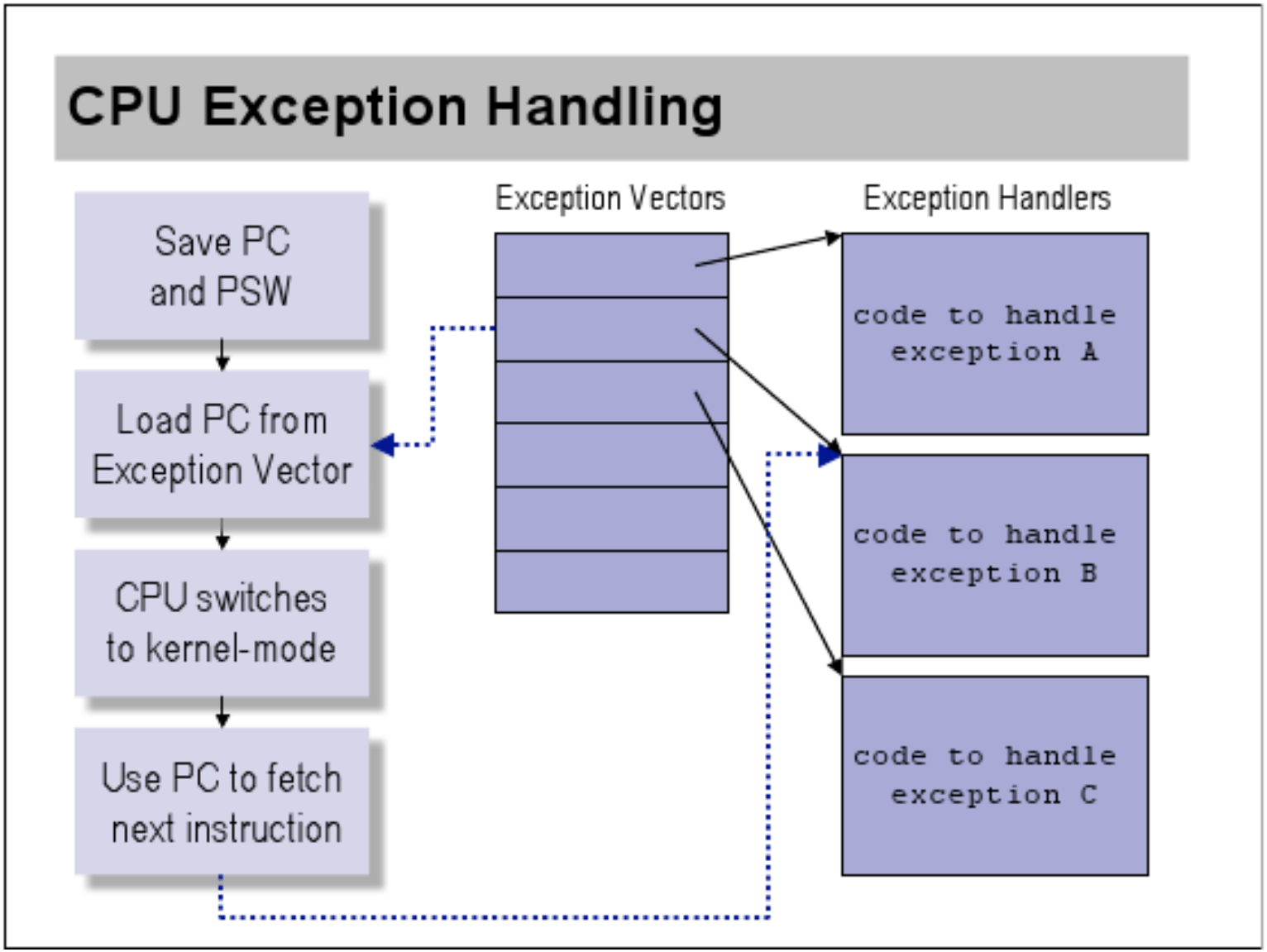
Što se događa prilikom obrade iznimke:

1. Postavlja se kernel (supervizor) bit u Program Status Word registru i procesor ulazi u kernel mod.
2. Minimalni kontekst procesora smješta se na kernel stack (PSW, PC i ostali registri).
3. CPU poziva OS exception handler. Identificira se uzrok iznimke (npr. djeljenje s nulom) i poziva handler za tu iznimku iz exception vectora..
4. Operativni sustav rukuje iznimkom:
   - Ako korisnički program ima ugrađeno rukovanje prepoznatom iznimkom (exception handling), onda OS prilagođava stanje korisničkog programa tako da on može zvati svoju proceduru za rukovanje iznimki (vrši se povratak u korisnički mod). Korisnički program može pokušati promjeniti uvjete koji su doveli do iznimke te ponovo izvršiti instrukciju koja je izazvala iznimku ili može potpuno promijeniti put izvršavanja.
   - Ako korisnički program nema specificirano rukovanje iznimkama, operativni sustav poziva odgovarajuće funkcije za rukovanje greškama ili zaustavlja izvršavanje programa.
5. Ako program nije zaustavljen procesor se vraća u user mode. Registri procesora se obnavljaju vrijednostima koje su bile smješte na kernel stacku.

Iznimke(**exceptions**)



Iznimke(**exceptions**)



Sistemski pozivi (**system calls**)

Sistemski pozivi su mehanizam po kojem korisnički programi pozivaju procedure kernela.

- Koriste se za I/O, pisanje/čitanje datoteka itd.

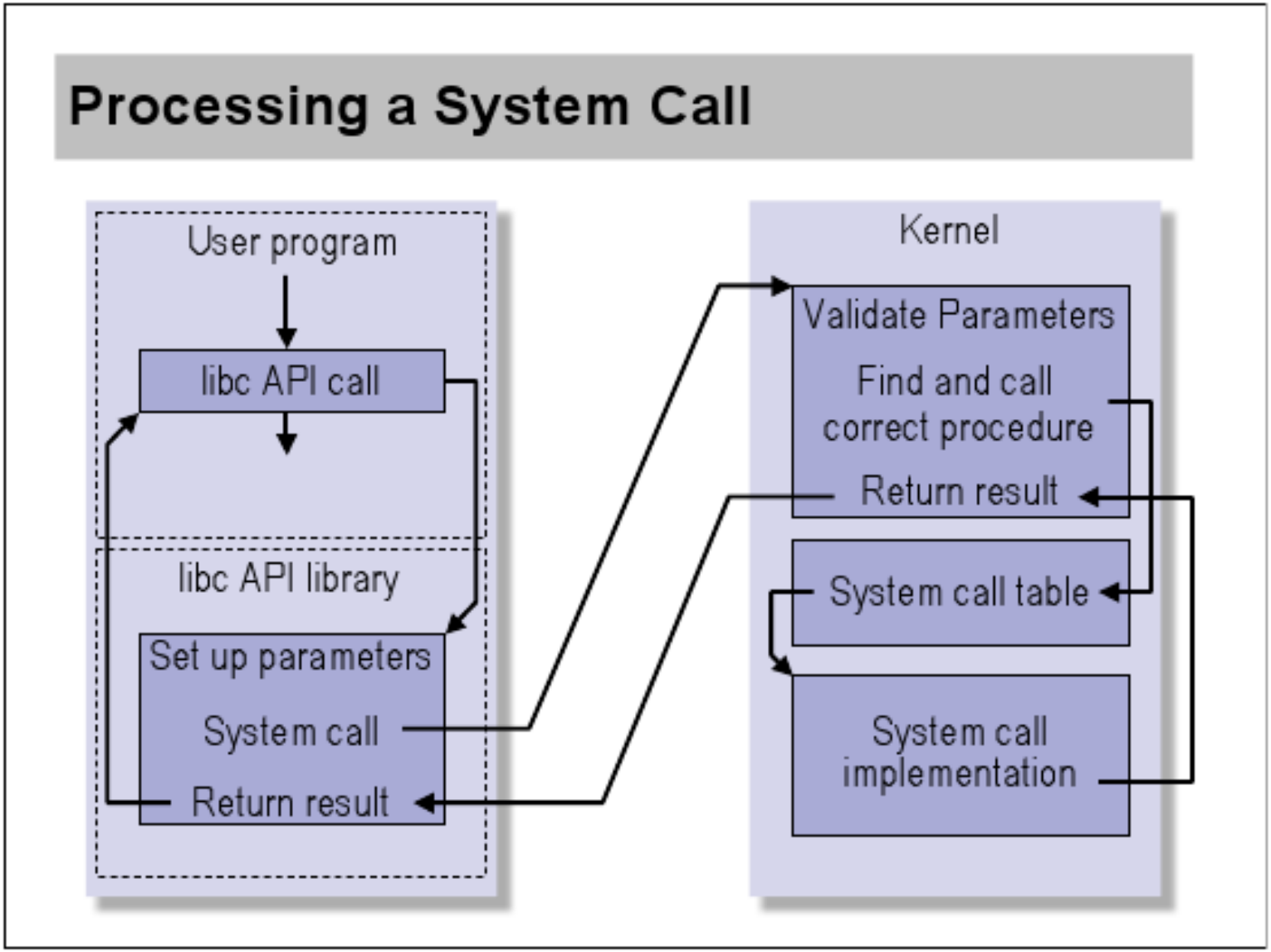
Osnovni mehanizam je sličan kao kod iznimaka i prekida.

Sistemski pozivi(**system calls**)

Što se događa prilikom poziva sistemske funkcije:

1. Korisnički program izaziva programski prekid (*software interrupt ili trap*) izvršavajući određenu instrukciju.
2. Procesor poziva OS na određenoj adresi, postavlja se supervizor bit i ulazi se u kernel mod.
3. OS identificira zahtjev i ulazne parametre.
4. Izvršava se zahtjevana operacija.
5. Postavljaju se određeni registri procesora tako da sadrže rezultate operacije.
6. Izvršava se određena instrukcija (IRET) i vraća se iz kernel moda u korisnički mod (resetira se supervizor bit).
7. Korisnički program prima rezultate i nastavlja s izvođenjem.

Sistemski pozivi(**system calls**)



Sistemski pozivi(**system calls**)

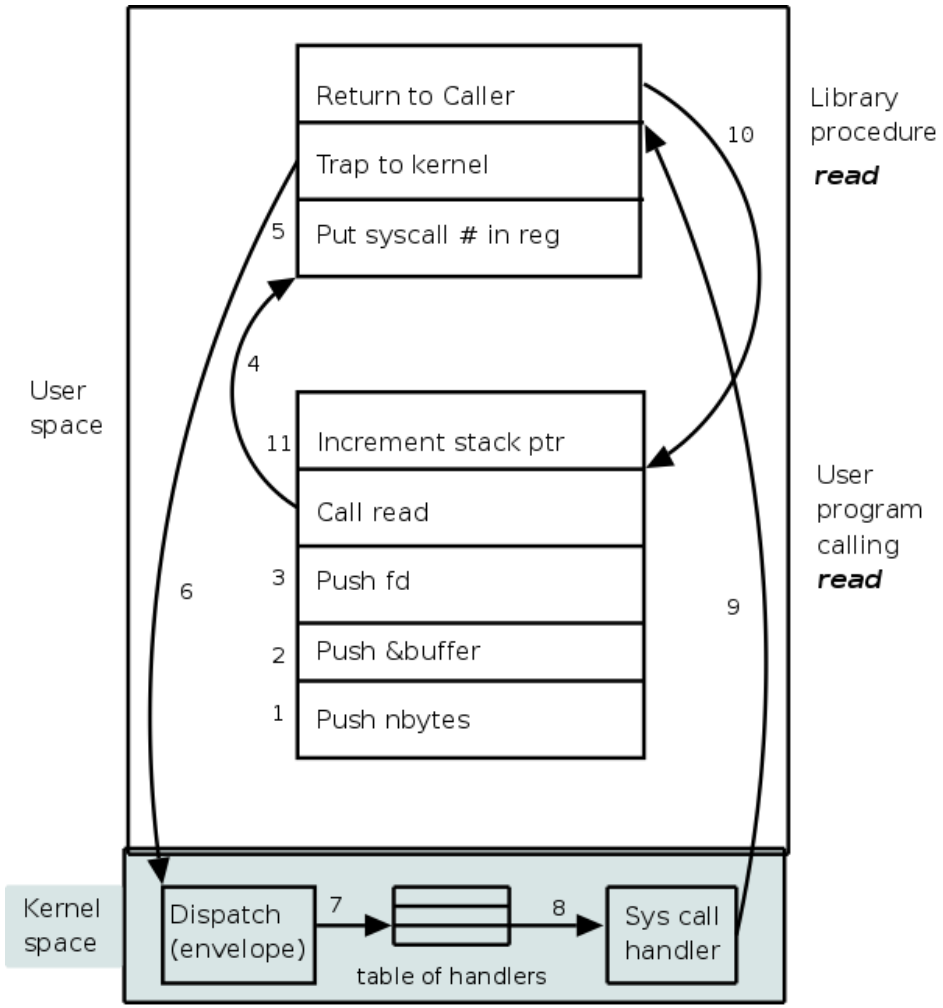
Primjer sistemskog poziva *read*:

- poziv iz c programa

```
count = read(fd, buffer, nbytes);
```

- *read* je sistemska funkcija koja ustvari izvršava `read` sistemski poziv.
- Program koji poziva funkciju *read* stavlja (*push*) parametre funkcije na stack, zatim poziva funkciju *read*.
- Funkcija *read* tipično stavlja broj sistemskog poziva na mjesto gdje to operativni sustav očekuje (registar).
- Zatim izvršava TRAP instrukciju koja prebacuje *user* u *kernel* mode i nastavlja izvršavanje na fiksnoj adresi u *kernel space-u*.
- Kernel kod koji se zatim izvršava proslijeđuje broj sistemskog poziva određenom rukovatelju sistemskih poziva *system call handler* (obično preko tabele sistemskih poziva).
- Kad system call handler završi, kontrola se vraća user-space funkciji na instrukciju iza instrukcije TRAP.
- User-space funkcija vraća vrijednost programu koji je pozvao.

Sistemski poziv (read)



POSIX i Windows API

- **POSIX** (**P**ortable **O**perating **S**ystem **I**nterface [for Unix]) je IEEE standard koji opisuje API (Application Programming Interface), zajedno sa shellom i drugim utility interface-ima za software kompatibilan sa Unix OS-om.
- Svaki operativni sustav napisan po POSIX standardu ima stotinjak funkcija preko kojih se realiziraju sistemski pozivi.
- **Windows API** (prije Win32 API) je skup funkcija za Windows familiju operativnih sustava koju koriste programeri za pristupanje servisima operativnog sustava.
- Na Windows OS razdvojene su API funkcije od sistemskih poziva.
- API funkcija ima na tisuće i neke od njih izvršavaju sistemske pozive (tj. izvodi ih kernel), a neke ne (tj. to su obične user-space funkcije).
- Nemoguće je odrediti koje su sistemske a koje user-space funkcije, jer od verzije do verzije Windowsa se to mijenja.
- Npr. na nekim Windowsima se upravljanje GUI-a izvršava u kernel modu, a na nekim ne.

Sistemski pozivi(**system calls**)

Unix vs Win32 System calls

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

9 Upravljanje procesima
- Uvod
- Procesi
- Threadovi

- Moderni računalni sustavi omogućuju da više od jednog programa bude učitano u memoriju i da se izvršavaju konkurentno (*concurrent*).
- Program u izvršavanju se naziva **proces**.
- Računalni sustav se sastoji od kolekcije procesa: procesa operativnog sustava (koji izvršavaju sistemski kôd) i korisničkih procesa (koji izvršavaju korisnički kôd).
- Svi ti procesi izvršavaju se konkurentno, dijeleći jedan ili više procesora.

Proces

- **Proces** je program u izvršavanju kojem je pridružen (virtualni)**adresni prostor**. **Adresni prostor** je dio memorije tj. lista memorijskih lokacija koje proces može čitati i po kojima može pisati.
  - Adresni prostor na Win32 je 4GB (obično je pola user space, pola kernel space), na Win64 8TB.
  - Adresni prostor na 32-bitnom linuxu je 4GB, 3GB za user space, 1GB za kernel space.
  - Te veličine su konfigurabilne.
- U tom adresnom prostoru nalaze se *instrukcije programa, njegovi podaci i stack*.
- Sa svakim procesom povezan je i neki *skup registara* uključujući *program counter* i *stack pointer*.

Proces - model

- Sav softver koji je pokrenut na računalu je organiziran u obliku nekog broja (sekvencijalnih) procesa.
- Iluzija da se procesi uzvršavaju u isto vrijeme se često naziva **pseudoparalelizam**.
- U stvarnosti procesor (CPU) se brzo prebacuje s procesa na proces.
- To se zove **multiprogramiranje**.
- Operativni sustav odlučuje kada će i koliko proces dobiti procesorskog vremena.
- Kada zaustavi jedan proces da bi procesor dodijelio drugom, operativni sustav mora sačuvati sve informacije o procesu.
  - Naprimjer, ako proces čita nekoliko datoteka, operativni sustav mora zapamtiti *current position* svake od njih.
- Informacije o procesu zapisuju se u tablici procesa (**process table**) koja je niz ili vezana lista struktura, po jedna za svaki proces.

Razlika programa i procesa

Tanenbaumova analogija sa kuharom koji radi rođendansku tortu za kćer.

- On ima recept za tortu i sastojke za nju.
- Recept je program, kuhar je procesor, a sastojci su ulazni podaci.
- Proces izrade torte se sastoji od čitanja recepta, dohvaćanja sastojaka i pravljenja torte.
- U nekom momentu dotrči njegov sin, vrišteći da ga je ugrizla pčela.
- Kuhar zapamti gdje je stao s tortom, dohvati knjigu o prvoj pomoći i počne slijediti upute iz nje.
- Dakle, procesor se prebacio sa jednog procesa (kuhanje) na proces višeg prioriteta (pružanje prve pomoći).
- Kad završi sa prvom pomoći kuhar se vraća torti, na ono mjesto gdje je stao prije.

Dakle, proces je aktivnost. Sastoji se od programa, ulaza, izlaza i stanja.

$\rightarrow$ Važno je naglasiti da ako se program pokrene dva puta to su dva procesa.

Kreiranje procesa

Postoji nekoliko načina da se kreira novi proces. Može ga kreirati korisnik, drugi proces ili operativni sustav.
Četiri su razloga koji uzrokuju kreiranje procesa:

1. Inicijalizacija sustava.
2. Izvršavanje sistemskog poziva za kreiranje procesa od strane nekog drugog aktivnog procesa.
3. Korisnički zahtjev za kreiranje procesa.
4. Pokretanje slijednih zadataka (obično na mainframeovima).

Kreiranje procesa prilikom pokretanja operativnog sustava

- Prilikom podizanja operativnog sustava nekoliko procesa se kreira.
- S nekima korisnik može komunicirati (**foreground procesi**).
- Neki se vrte u pozadini (**background procesi**) i nisu vezani uz korisnika, nego uz pojedinu funkciju.
- Npr. procesi koji rukuju aktivnostima poput e-maila, uglavnom spavaju; kad e-mail stigne se bude.
- Takvi procesi zovu se **daemons**.

Kreiranje procesa od strane nekog drugog procesa

Primjer: Treba dohvatiti veliki broj podataka preko mreže kako bi se s njima nešto računalo.

- Uobičajeno je da se kreira novi proces koji će dohvaćati podatke i spremati ih u neku zajedničku memoriju, dok bi prvi proces uzimao te podatke i računao s njima.

Kreiranje procesa od strane korisnika

Korisnik može kreirati proces na različite načine:

- unošenjem imena aplikacije iz komandne linije (u tom slučaju komandni interpreter tj. shell pokreće novi proces),
- preko grafičkog sučelja npr. klikom na ikonu aplikacije.

U oba slučaja kreira se novi proces. Na unixoidima koji vrte X-e proces preuzima prozor iz kojeg je pokrenut. Na Windowsima, kreirani proces nema prozor, ali ga može kreirati (i to uobičajeno radi).

Kreiranje procesa

U svakom od ovih slučajeva neki postojeći proces sistemskim pozivom kreira novi proces.

UNIX:

- Jedan jedini sistemski poziv za kreirannje procesa je `fork`.
- Ovaj poziv klonira proces koji ga je pozvao.
- Nakon `fork`-a dva procesa, roditelj i dijete, imaju istu memorijsku sliku, environment varijable i otvorene file-ove.
- Obično proces dijete dalje sa sistemskim pozivom `execve` ili sličnim poziva neki novi program.

Windows API:

- Koristi se `CreateProcess` sistemski poziv za kreiranje novog procesa.
- Taj poziv ima 10 parametara: program koji će se izvršiti, command-line parametri, nekoliko security atributa, bitovi koji određuju da li će se naslijediti otvoreni file-ovi, informacije o prioritetu, specifikacija prozora i pointer na strukturu u kojoj su informacije o novokreiranom procesu.
- Windows API ima još stotinjak funkcija za upravljanje i sinkronizaciju procesa.

Kreiranje procesa

- Nakon što se proces kreira i roditelj i dijete imaju vlastiti adresni prostor.
- Na UNIX-u je adresni prostor djeteta kopija adresnog prostora roditelja.
- Na Windowsima su to dva odvojena adresna prostora.

Završetak procesa

1. Normalni završetak (dobrovoljno).
2. Završetak zbog greške (Error exit) (dobrovoljno).
3. Nepopravljiva greška (Fatal error) (prisilno).
4. Završen (ubijen) od strane drugog procesa (prisilno).

Kreiranje procesa na UNIX-u, primjer

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main ()
{
   pid_t child_pid;
   printf ("the main program process ID is %d\n", (int) getpid ());
   child_pid = fork ();
   if (child_pid != 0)
   {
      printf ("this is the parent process, with id %d\n", (int) getpid ());
      printf ("the childs process ID is %d\n", (int) child_pid);
   }
   else
   {
      /* The argument list to pass to the ls command. */
      char* arg_list[] = {"ls","-l","/",NULL /* The argument list must end with a NULL.*/};
      printf ("this is the child process, with id %d\n", (int)getpid ());
      execvp ("ls", arg_list);
      /* The execvp function returns only if an error occurs. */
      fprintf (stderr, "an error occurred in execvp\n");
   }
   return 0;
}
```

Kreiranje procesa na UNIX-u

- Sistemskim pozivom `fork` zahtjeva se kreiranje novog procesa.
- Pokrenuti proces postaje "dijete" procesa "roditelja" koji ga je pokrenuo.
- Dijete dobije kopije memorijskog segmenta instrukcija i segmenta podataka od roditelja.

  $\rightarrow$ Kernel može uštediti vrijeme i memoriju tako da postavi segment instrukcija kao zajednički za oba procesa.
- `p_id fork(void);`
  - U ovaj sistemski poziv ulazi jedan proces, a iz njega izlaze dva odvojena procesa "dijete" i "roditelj" i dobivaju svaki svoju povratnu vrijednost.
  - Proces dijete dobija rezultat 0, a proces roditelj identifikacijski broj djeteta.
  - Ako dođe do greške, vraćena vrijednost je -1, dijete nije kreirano.
  - Dijete nasljeđuje većinu atributa iz segmenta sistemskih kao što su aktualni direktorij, prioritet i identifikacijski broj korisnika.
  - Dijete ne nasljeđuje: identifikacijski broj roditelja, file descriptore (dobija njihovu kopiju) i vrijeme izvođenja djeteta se postavi na nula.
- Dijete se može inicijalizirati novim programom (poziv `exec`) ili izvoditi poseban dio već prisutnog programa
- Roditelj može čekati da dijete završi ili paralelno raditi nešto drugo.

Kreiranje procesa na UNIX-u

Osnovni oblik upotrebe sistemskog poziva `fork` izgleda ovako:

```
if (fork() == 0)
{
  posao djeteta
  exit(0);
}
nastavak rada roditelja (ili nista)
wait(NULL);
```

Završetak procesa na UNIX-u

```
void exit(int status);
```

- Završava izvođenje procesa koji ga je pozvao (ubije ga).
- Prije završetka zatvore se sve otvorene datoteke.
- Za `status` se obično stavlja `0`, ako je sve uredno završilo, a `1` inače.
- Roditelj procesa koji završava pozivom `exit` prima njegov status preko sistemskog poziva `wait` ili `waitpid`.

Završetak procesa na UNIX-u

Postoje tri načina kako može završiti proces: pozivom `exit`, primitkom signala ili padom sustava (nestanak napajanja ili slično).

- Ako proces dijete završi, a roditelj ga ne čeka sa `wait`, on postaje **proces-zombi (zombie)**. Otpuštaju se njegovi segmenti u memoriji, ali se zadržavaju njegovi podaci u tablici procesa. Oni su potrebni sve dok roditelj ne izvede `wait` kada proces-zombi nestaje.
- Ako roditelj završi, a da nije pozvao `wait`, **proces-siroče (orphan)** dobiva novog roditelja, proces `init` s identifikacijskim brojem 1, koji će ga prikupiti sa `wait`. `init` je važan prilikom pokretanja sustava, a u kasnijem radu većinom izvodi `wait` i tako "prikuplja izgubljenu djecu" kada završe.

Kreiranje procesa na Windowsima, primjer

```
#include <windows.h>
#include <stdio.h>
#include <tchar.h>
int main( void )
{
   STARTUPINFO si;
   PROESS_INFORMATION pi;
   TCHAR tszCommandLine[500];

   // Initialize the startup info struct
   GetStartupInfo(&si);

   // Call CreateProcess(), mostly defaults. Note that
   // the lpCommandLine argument MUST NOT be a constant string.
   _tcscpy(tszCommandLine, _T("notepad.exe novi.txt"));
   CreateProcess(NULL, tszCommandLine, NULL, NULL, FALSE, NULL, NULL, NULL, &si, &pi);

   // Close our reference to the initial thread
   CloseHandle(pi.hThread);

   // Wait for the process to exit.
   WaitForSingleObject(pi.hProcess, INFINITE);

   // And close our reference to the process.
   CloseHandle(pi.hProcess);
   return 0;
}
```

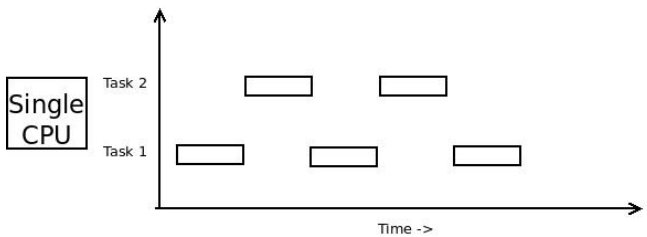
Kreiranje procesa na Windowsima

`CreateProcess` (nalazi se u zaglavlju Windows.h) ima 10 parametara:

1. Pokazivač na ime modula koji se želi izvršiti.
2. Pokazivač na string koji predstavlja komandu liniju koja će se izvršiti.
3. Pokazivač na strukturu za sigurnosnim atributima procesa.
4. Pokazivač na strukturu za sigurnosnim atributima za inicijalni thread.
5. Bit koji kazuje da li novi proces nasljeđuje handle-ove procesa koji ga je kreirao.
6. Različiti flagovi (npr. error mode, prioritet, i sl.)
7. Pokazivač na environment.
8. Pokazivač na radni direktorij novog procesa.
9. Pokazivač na strukturu koja postavlja inicijalne vrijednosti prozora za novi proces.
10. Pokazivač na strukturu koja vraća podatke o novo kreiranom procesu kao što su njegov process id i handle novog procesa.

Vraća 0 u slučaju greške.

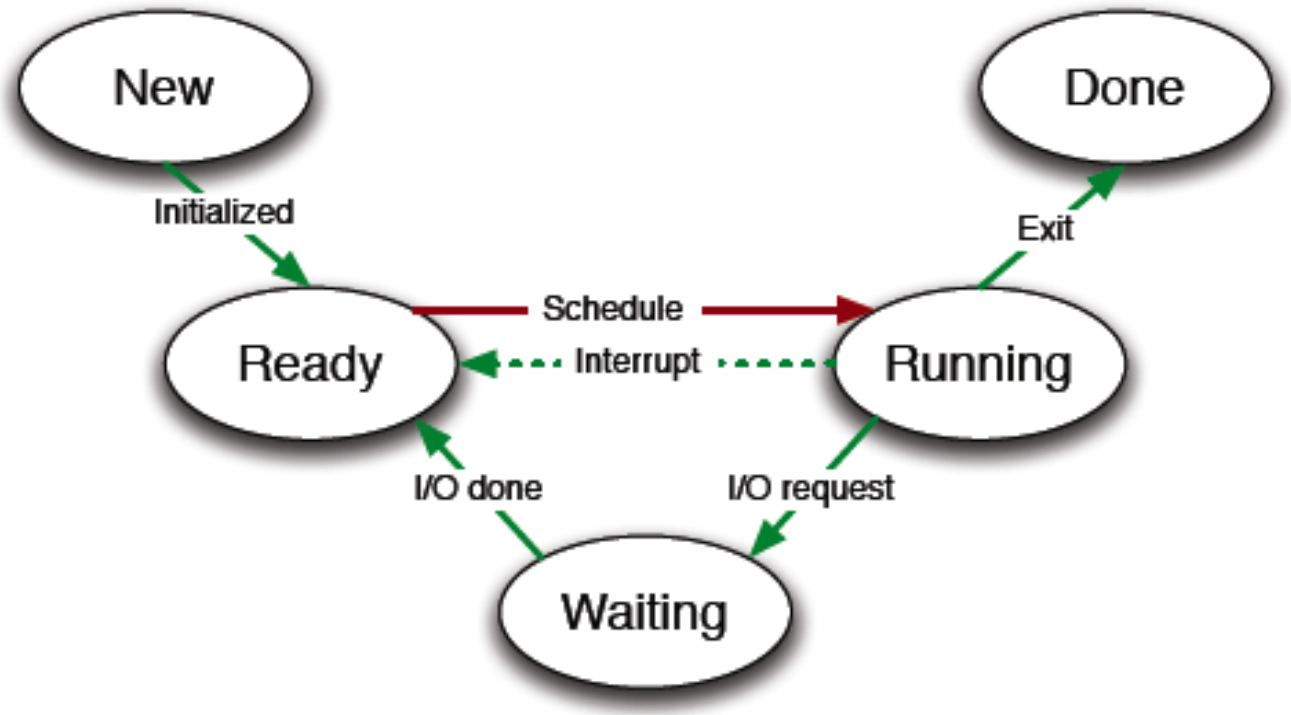
Istovremeno (*concurrent*) vs. paralelno izvršavanje procesa



(a) Two concurrent programs running sequentially

(b) Two concurrent programs running in parallel

Stanja procesa



Stanja procesa

Proces može biti u jednom od mogućih stanja:

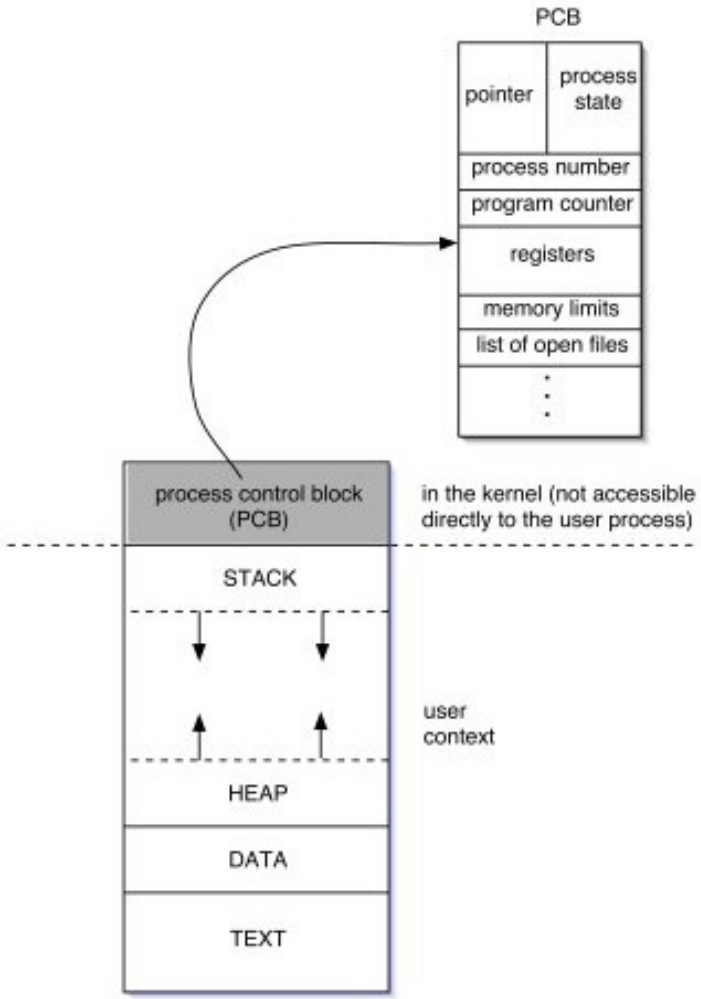
- New – novokreirani proces.
- Ready – proces čeka na izvršavanje kad procesor postane slobodan (neki drugi proces se trenutno izvršava).
- Running – proces izvršava instrukciju (za jednoprocesorski sustav može biti samo jedan proces u ovom stanju).
- Waiting – proces čeka na neki eksterni događaj (ne može se izvršavati dok se događaj ne izvrši).
- Done – proces je završio sa izvršavanjem.

Kontrolni blok procesa

Kontrolni blok procesa (**Process Control Block PCB**) čine podaci neophodni za upravljanje procesom.
Dio je radne memorije tj. to je memorijska struktura sa osnovnim informacijama o procesu zahvaljujući kojoj izvršavanje procesa se može zaustavljati i nastavljati više puta.
U informacije kontrolnog bloka spadaju:

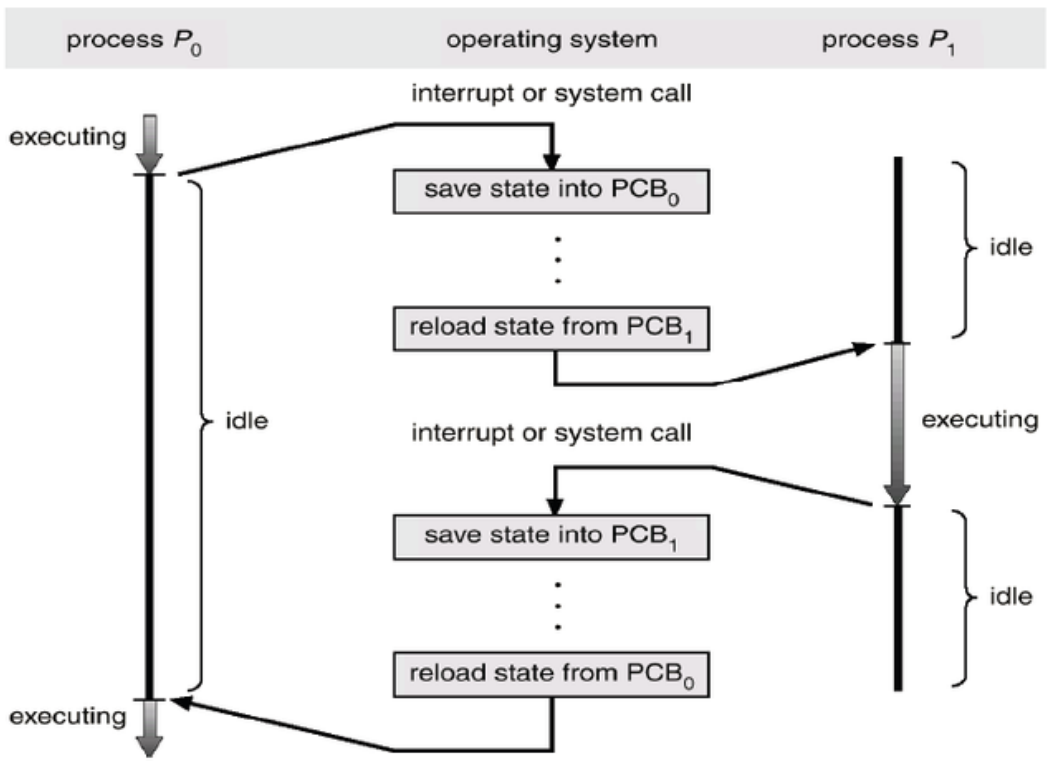
- ime ili jedinstveni identifikator procesa (PID)
- kontekst (okruženje) procesa
- prioritet procesa
- informacije o memoriji procesa
- lista otvorenih datoteka
- status zauzetih ulazno-izlaznih resursa
- trenutno stanje procesa

PCB


PCB
pointer
process state
process number
program counter
registers
memory limits
list of open files
process control block (PCB)
in the kernel (not accessible directly to the user process)
STACK
user context
HEAP
DATA
TEXT

Zamjena konteksta - context switch

Kontekst procesa čine podaci koji se čuvaju prilikom oduzimanja procesora, a njih generira sam hardver: programski brojač, vrijednosti registara i pokazivači na dio memorije koji proces koristi (njegov adresni prostor). **Zamjena konteksta** (*context switch*) je spremanje/učitavanje konteksta procesa (ili threada) do kojeg dolazi tijekom prebacivanja procesora na drugi proces ili thread.



Još o procesima

Dvije karakteristike procesa

- Proces kao jedinica vlasništva nad resursima
  - proces ima pripadajući virtualni adresni prostor
  - proces ima kontrolu nad nekim resursima (datotekama, ulazno-izlaznim uređajima)
- Proces kao jedinica raspoređivanja (*dispaching*)
  - proces se izvršava između drugih procesa (multitasking)
  - proces ima svoje stanje izvođenja i prioritet

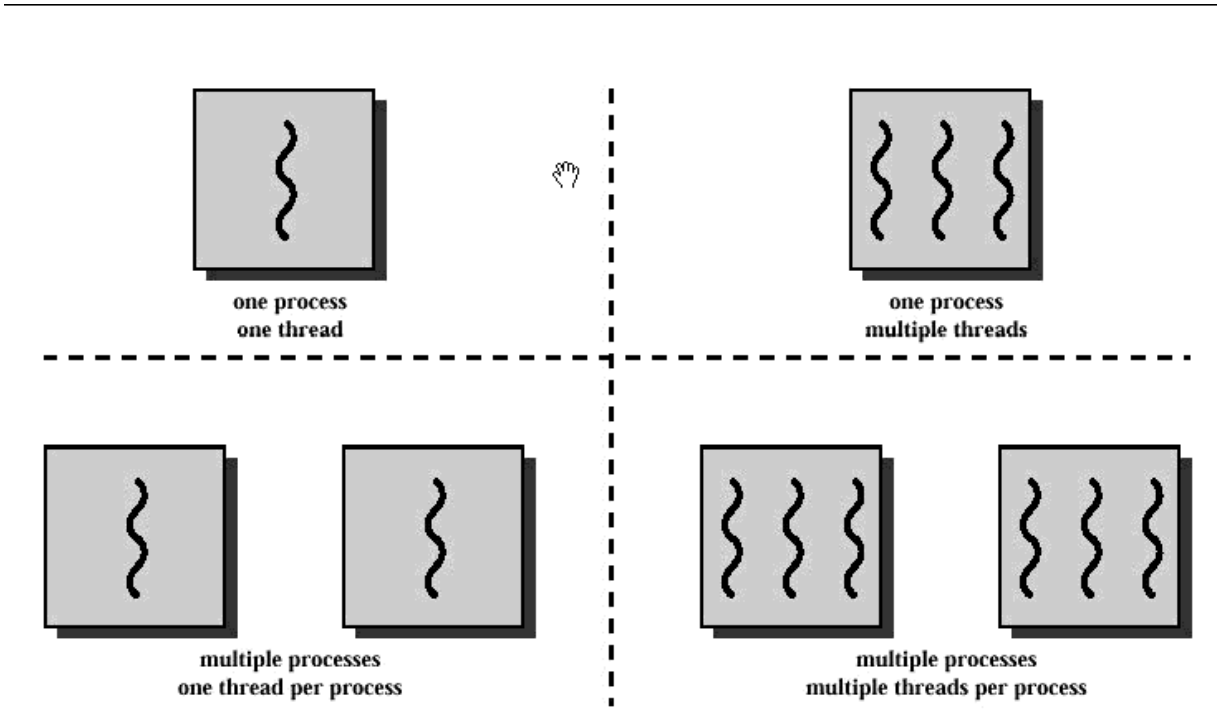
  U teoriji modernih operacijskih sustava ove dvije karakteristike su nezavisne, pa je jedinica vlasništva nad resursima **proces**, a jedinica raspoređivanja **thread**.

Thread ili nit

Thread (nit ili dretva) je najmanja jedinica koja CPU može konkurentno izvoditi tj. koju operacijski sustav može raspoređivati (*schedule*) u redu čekanja za procesor.

- Thread ima stanje izvršavanja (ready, running, waiting, itd.)
- Sprema se **kontekst threada** prilikom izlaska iz stanja *running*.
- Thread ima svoj *execution stack* i memoriju za statičke lokalne varijable.
- Thread ima pristup adresnom prostoru i resursima procesa kojega je thread dio.

Multithreading vs. Single threading



Multithreading vs. Single threading

- Multithreading omogućava većem broju threadova egistenciju unutar jednog procesa.
- Threadovi unutar jednog procesa dijele resurse, ali se izvršavaju nezavisno.
- Ako se različite funkcije unutar programa mogu izvršavati istovremeno i neovisno jedna o drugoj, ima smisla razmišljati o threadovima (npr. množenje matrica).
- Programiranje threadovima omogućuje jednom procesu da se izvršava paralelno na višeprocesorskim sustavima.

Odnos threadova i procesa

- Threadovi postoje unutar procesa i koriste resurse procesa.
- Imaju svoj vlastiti tok izvršavanja.
- Mogu međusobno dijeliti resurse procesa.
- Umiru ako umre proces roditelj.
- Threadovi se još nazivaju **lightweight procesi**.

Odnos threadova i procesa

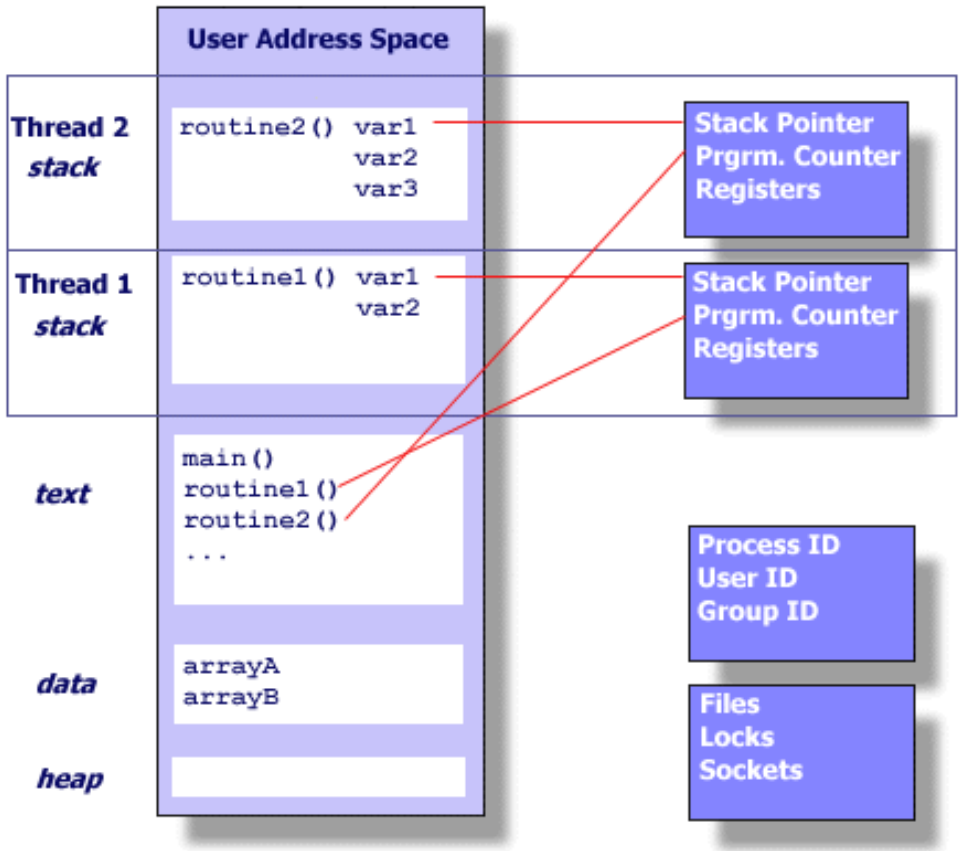
U donjoj tablici prikazano je što dijele, a što ne dijele threadovi unutar jednog procesa.

Svojstva zajednička svim threadovima unutar istog procesa	Svojstva pojedinog threada
Adresni prostor Globalne varijable Otvorene datoteke Procesi djeca Tekući alarmi Signali i rukovatelji signalima	Programski brojač Registri Stack Stanje threada

Prednosti multithreadinga

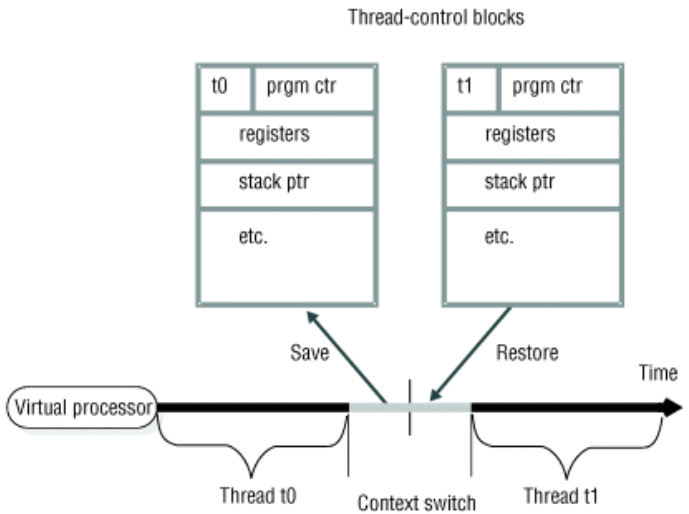
- brže se kreira novi thread od novog procesa, jer thread koristi postojeći adresni prostor procesa
- brže prebacivanje između threadova
- lakša komunikacija između threadova - threadovi dijele adresni prostor

Slika adresnog prostora procesa koji se sastoji od više threadova



Zamjena konteksta - context switch

Unutar svakog threada postoji **kontrolni blok threada** (*Thread control block*) u kojem se čuva kontekst threada tj. stanje registara prilikom zamjene konteksta, i stanje threada (waiting, ready i sl.) te prioritet threada.



Zamjena konteksta se odvija na sljedeći način:

- thread vraća kontrolu OS-u
- OS bira sljedeći thread iz reda čekanja
- sprema se stanje prvog threada
- učitava se stanje sljedećeg threada
- izvršava se sljedeći thread

Mogući problemi

- Promjene koje izvrši jedan thread nad resursima procesa (npr. zatvaranje datoteke), vide svi ostali threadovi unutar tog procesa.
- Pointeri sa istom vrijednošću unutar dva različite threada pokazuju na iste podatke.
- Kako mogu pisati i čitati iste memorijske lokacije, zahtjevaju od programera sinkronizaciju.

10 Raspoređivanje vremena procesora - CPU Scheduling

CPU Scheduling

- Da bi operacijski sustav ostvario bilo koji oblik višezadaćnosti mora postojati način da se raspodijeli vrijeme procesora među zadaćama koje se izvršavaju u danom vremenu.
- Ako je na raspolaganju samo jedan CPU, treba odabrati proces koji će se sljedeći izvršavati.
- Dio operativnog sustava koji odabire proces koji će se sljedeći izvršavati zove se **planer poslova** (**scheduler**).
- Algoritam za raspoređivanje poslova zove se **scheduling algoritam**.
- Fokusirati ćemo se na algoritme za raspoređivanje koji se odnose i na threadove i na procese.

CPU Scheduling

Kada treba odlučivati o raspoređivanju procesa?

- Prilikom kreiranja novog procesa treba odlučiti da li će se prvo izvršavati proces roditelj ili proces dijete (oba su u *ready* stanju).
- Prilikom završavanje jednog procesa, treba odlučiti koji će se proces izvršiti sljedeći. Ako nijedan nije u stanju *ready*, idle proces se vrti.
- Ako je proces blokiran od strane I/O uređaja.
- Ako završi odsječak vremena dodijeljen procesu.

CPU Scheduling

S obzirom na odsječak vremena dodijeljen pojedinom procesu, algoritmi za raspoređivanje procesa mogu se podijeliti na:

- **Nonpreemptive** ili **neiznuđeni** algoritmi koji odabiru proces i puštaju ga da radi sve dok se ne blokira.
- **Preemtive** ili **iznuđeni** algoritmi koji odabiru proces i puštaju ga da radi do nekog unaprijed određenog vremenskog odsječka.

U različitim okruženjima različiti algoritmi su optimalni. Tako se razlikuju okruženja:

- batch sustavi (rade dugo jedan posao - neiznuđeni algoritmi ili iznuđeni sa velikim vremenskim odsječkom),
- interaktivni sustavi (npr. serveri - iznuđenost je bitna u radu s puno korisnika),
- real-time sustavi.

CPU Scheduling

Ciljevi koje algoritmi za dodjelu procesora procesu trebaju postići:

- **pravednost** - svaki proces trebao bi dobiti pravedni dio procesora,
- **balans** - svi dijelovi sustava bi trebali biti zaposleni,
- što veću **propusnost** - broj procesa koji se mogu izvršiti u zadanom vremenu (bitno kod batch sustava),
- što manje **vrijeme kompletiranja procesa** ili **turnaround time** - vrijeme koje protekne od dolaska u red čekanja procesa do kraja izvršavanja,
- što manje **vrijeme čekanja** - vrijeme koje proces provede u redu čekanja,
- što manje **vrijeme odziva** ili **response time** - vrijeme koje protekne od dolaska u red čekanja, dok se proces ne počne izvršavati,
- **iskoristivost procesora** ili **CPU utilization** - održavanje procesora zaposlenim cijelo vrijeme.
- **izbjegavanje izgladnjivanja (starvation)** - stanje u kojem je proces beskonačno blokiran.

Algoritmi za dodjelu procesora

- First-Come First-Served ili FIFO,
- Shortest Job First,
- Shortes Remaining Time Next,
- Round Robin,
- Priority algoritam,
- Multilevel feedback queue (višerazinski redovi s povratnom vezom).

Algoritmi za dodjelu procesora - FCFS

First-Come First-Serve je najprostiji algoritam za raspoređivanje procesora - procesi dobijaju procesor onim redom kojim su došli u red čekanja.

- Red čekanja funkcionira po FIFO principu: objekt koji predstavlja proces ili thread (kontrolni blok procesa ili threada) koji je ušao u red čekanja stavlja se na kraj liste (tail), a procesor se dodjeljuje onom procesu koji je na početku liste (head).
- Primjer neiznuđenog (non preemptive) algoritma - proces drži CPU sve dok ne ode u stanje *waiting* ili *exit*.
- Prednosti:
  - broj zamjena konteksta je mali, pa je iskoristivost procesora dobra,
  - ne može doći do izgladnjivanja zbog čuvanja redosljeda procesa koji čekaju u redu,
  - algoritam se lako shvaća i implementira.
- Mane:
  - u slučaju da na red dođe proces koji obavlja CPU-intenzivnu zadaću koja dugo traje i ne pristupa I/O uređajima, ostali procesi mogu zapeti,
  - srednje vrijeme čekanja (waiting time) ovisi od dužine trajanja procesa i od rasporeda njihovog dolaska na red čekanja.

Algoritmi za dodjelu procesora - Shortest Job First

Za izvršavanje se odabire onaj proces koji će najbrže završiti.

- Za sve procese iz reda čekanja, procjenjuje se vrijeme koje je potrebno za izvršavanje.
- Na one procese kojima je procjenjeno vrijeme isto primjenjuje se FCFS algoritam.
- Primjer neiznuđenog (non preemptive) algoritma - proces drži CPU sve dok ne ode u stanje *waiting* ili *exit*.
- Iako je prosječno vrijeme odziva optimalno i iskoristivost CPU dobra može doći do izgladnjavanja, naime puno kratkih poslova koji se stalno kreiraju može spriječiti da se dugi poslovi izvršavaju.
- Najveći je problem kako predvidjeti budućnost, odnosno kako znati koliko će se neki proces izvršavati.
- Najčešće se koristi u batch sustavima kada se može predvidjeti koliko će pojedini zadatak trajati s obzirom da se isti zadaci obavljaju svakodnevno.

Algoritmi za dodjelu procesora - Shortest Remaining Time to Completion First

Iznuđena verzija STJ algoritma.

- Proces se može prekinuti ako je u redu čekanja proces kojemu manje treba da završi.

Algoritmi za dodjelu procesora - Round Robin

Round Robin je jedan od najjednostavnijih algoritama za dodjelu procesora koji dodjeljuje jednake odsječke vremena (**quantum** ili **time slice**) svakom procesu u poredku kako dolaze u red čekanja i bez prioriteta.
To je ustvari FCFS strategija sa iznuđenom višezadaćnošću (preemptive).

- Definira se kratak vremenski interval (time slice) reda veličine od 10 do 100 ms i kružni red čekanja na procesor.
- Proces koji pristigne u red čekanja ide na kraj liste,
- Operacijski sustav po isteku vremenskog intervala generira prekidni signal (timer interrupt), nakon čega se procesu oduzima kontorla nad procesorom i predaje slijedećem procesu iz kružne liste.

Algoritmi za dodjelu procesora - Round Robin

U okviru jednog vremenskog intervala moguće su sljedeće situacije:

- proces je **završio** aktivnost prije isteka vremenskog intervala - proces tada oslobađa procesor, a algoritam Round Robin uklanja proces iz kružne liste i predaje kontrolu slijedećem procesu;
- proces nije završio aktivnosti, ali mora prekinuti izvršavanje kad mu **istekne vremenski interval**; algoritam RR postavlja prekinuti proces na kraj reda čekanja, a kontrolu predaje slijedećem procesu iz liste;
- proces se **blokirao** zbog čekanja na ulazno-izlazne operacije - blokirani proces oslobađa procesor, a algoritam RR predaje kontrolu slijedećem procesu iz liste; proces prelazi u red čekanja na kraj RR liste tek nakon povratka u stanje READY.

Algoritmi za dodjelu procesora - Round Robin

Glavno pitanje je kako pravilno odabrati vremenski interval prekida.

- ako je preveliki vrijeme odziva je loše (drugi procesi dugo čekaju) - krajnji slučaj (beskonačan vremenski interval) zapravo je FCFS strategija,
- ako je interval premali iskoristivost procesora je loša (svo vrijeme se troši na zamjenu konteksta) - krajnji slučaj je da procesi izvršavaju samo jednu instrukciju u danom vremenskom intervalu.

U stvarnosti treba nam neki balans. Tipičan vremenski interval je 10-100 milisekundi, dok je tipično vrijeme za zamjenu konteksta oko 100-1000 CPU instrukcija, tako da se maksimalno 10% vremena troši za zamjenu konteksta. Odabir duljine vremenskog intervala ovisi o tipu operacijskog sustava, npr. kod Windowsa 2000 Server verzije iznosi oko 120 ms, dok kod XP iznosi 20 ms. Naime, ako je naglasak na interakciji sa korisnikom zanima nas što kraće vrijeme odziva, pa se odabire što kraći interval, a ako je naglasak na serverskim aplikacijama onda je iskoristivost procesora mnogo važnija pa odabiremo duži interval.

Algoritmi za dodjelu procesora - Round Robin

Prednosti Round Robin strategije:

- pravednost - svaki proces dobija isti vremenski interval, te uz pravilno odabranu dužinu intervala iskoristivost procesora je prilično dobra (obično 90% od maksimalne)

Mane:

- srednje vrijeme odziva može biti bitno lošije nego kod drugih strategija.
  Za više procesa čije je vrijeme izvršavanja različito, prosječno vrijeme odziva je dobro (kraći poslovi će završiti brzo, dok za duge vrijeme odziva nije puno gore od optimalnog).
  Problem se javlja kad poslovi zahtjevaju isto vrijeme za izvršavanje.

Algoritmi za dodjelu procesora - Prioritetni algoritam

- Svakom procesu možemo dodijeliti **prioritet**.
- Kod ove strategije uvijek se izvršavaju procesi **najvišeg** prioriteta.
- Tek kad oni svi završe tj. odu u stanje *waiting* (kad nema nijednog procesa u stanju *ready*) ide se na izvršavanje procesa nižeg prioriteta (naravno, onih koji su u stanju Ready).
- SRTCF spada u politike prioritetnog raspoređivanja vremena procesora, gdje je prioritet *predviđeno vrijeme do kraja izvršavanja*.
- Standardna implementacija je takva da se procesi najvećeg prioriteta izvršavaju u RR krugu (tako je implementirano kod Win32 operacijskih sustava).
- Kod Windows 2000 imamo 32 razine prioriteta (0-31) s tim da je 0 najniži prioritet.

Algoritmi za dodjelu procesora - Prioritetni algoritam

Navedimo standardne događaje koji izazivaju zamjenu konteksta:

- istek vremenskog intervala,
- završetak rada procesa,
- dobrovoljan odlazak procesa u stanje čekanja,
- proces većeg prioriteta (novokreirani ili onaj koji je prešao iz stanja *waiting* u *ready*) može prekinuti izvršavanje procesa nižeg prioriteta.

Zbog ovog zadnjeg možemo imati problem sa izgladnjivanjem jer procesi nižeg prioriteta čekaju da procesi višeg prioriteta završe i možda nikada ne dođu na red. Srećom pokazalo se da interaktivne aplikacije većinu vremena provedu u čekanju na neki događaj (obično akciju korisnika), pa kad OS detektira da svi poslovi najvećeg prioriteta čekaju, dodjeljuje CPU poslovima nižeg prioriteta.

Algoritmi za dodjelu procesora - Prioritetni algoritam

Kako bi mogli riješiti problem izgladnjavanja?

- Postoje različite varijante algoritama kod kojih kako vrijeme odmiče povećavamo prioritet procesa/threadova koji ne mogu doći na red za izvršavanje.

Windows 2000 primjenjuje slijedeću strategiju:

- Postoji poseban sistemski thread (balance set manager) koji svake sekunde provjerava da li postoje procesi u stanju *ready* koji predugo čekaju (taj interval izgladnjavanja je obično 3 sekunde).
- Ako se pronađe takav proces/thread privremeno mu se povećava prioritet.
- Nakon što taj proces izvrši svoj *quantum* povećani prioritet mu se vraća na staru vrijednost.

Algoritmi za dodjelu procesora - Multilevel feedback queue (višerazinski redovi s povratnom vezom)

Da bi riješile problem izgladnjavanja razne verzije UNIX-a koriste ovu strategiju.

- Postoji više redova sa procesima koji su u stanju *ready*, svaki sa **različitim prioritetom** i svaki sa **različitim vremenskim intervalom izvršavanja**.
- Za svaku razinu (red) prioriteta primjenjuje se Round Robin strategija.
- Izvršavaju se prvo poslovi najvišeg prioriteta i tek ako je red najvišeg prioriteta prazan prelazi se na sljedeći, niži nivo.
- Naravno, proces nižeg prioriteta može biti prekinut u izvršavanju (**preempted**) ako se pojavi proces višeg prioriteta.
- **Glavna značajka** ove strategije je da se pojedinom procesu može **dinamički mijenjati prioritet** (odnosno može biti smješten u različite redove).
- Round Robin vremenski intervali **povećavaju se** eksponencijalno kako se prioritet smanjuje.

Algoritmi za dodjelu procesora - Multilevel feedback queue (višerazinski redovi s povratnom vezom)

Tijekom vremena prioritet pojedinog procesa se prilagođava:

- proces koji tek ulazi u red čekanja dobija najveći prioritet,
- ako vremenski interval istekne, a proces nije završio, smanji mu se prioritet,
- ako "timeout izgladnjivanja" istekne, a posao još nije došao na red, povećaj mu se prioritet,
- ako proces dobrovoljno napusti procesor, prioritet mu ostaje isti.

Primjenom ove strategije interaktivni procesi (koji većinu vremena provedu u stanju čekanja) osciliraju oko najvišeg prioriteta, dok procesi koji intenzivno koriste CPU (serverski procesi, numeričke aplikacije) osciliraju oko najnižeg prioriteta.

11 Interprocesna komunikacija

- Međusobno isključivanje procesa
- Zastoj

Interprocesna komunikacija

Procesi mogu biti **nezavisni** ili **zavisni** (kooperativni).

- Ako se procesi izvršavaju usporedno i nezavisni su, među njima nema interakcije niti razmjene informacija.
- Ako se procesi izvršavaju zavisno (kooperativno), oni međusobno komuniciraju ili dijele zajedničke podatke i resurse.
  - U multitasking operativnim susutavima praktički nema nezavisnih procesa - svi dijele resurse.

Nezavisni procesi ne ovise o redoslijedu izvršavanja tj. raspoređivanju procesora. Determinirani su, tj. rezultat izvršavanja može se predvidjeti. Samim tim, mogu se reproducirati.
Zavisni procesi ovise o redoslijedu izvršavanja i preplitanju. Dakle, ovise o raspoređivanju. Nisu determinirani, tj. rezultat izvršavanja se ne može predvidjeti. Samim tim, ne mogu se reproducirati.

Primjer

Promatrajmo dva procesa, P1 i P2, koji žele privremeno sačuvati neku vrijednost na istoj memorijskoj lokaciji A. Neka se događaji odvijaju po slijedećem scenariju:

1. Proces P1 provjerava da li je memorijska lokacija A slobodna. Lokacija A je slobodna. Proces P1 je obaviješten da je lokacija A slobodna.
2. Proces P2 provjerava da li je memorijska lokacija A slobodna. Lokacija A je slobodna. Proces P2 je obaviješten da je lokacija A slobodna.
3. Proces P1 upisuje podatak na memorijsku lokaciju A.
4. Proces P2 upisuje drugi podatak na memorijsku lokaciju A.
5. Proces P1 čita POGREŠAN podatak s memorijske lokacije A.

Da bi se izbjegle slične situacije, operacijski sustav mora osigurati mehanizme za očuvanje konzistentnosti podataka, tj. mehanizme konkurentnog izvršavanja procesa.

Osnovni pojmovi

- **Kontekst procesa** čine podaci koji se čuvaju prilikom oduzimanja procesora, a njih generira sam hardver: programski brojač, vrijednosti registara i pokazivači na dio memorije koji proces koristi (njegov adresni prostor).
- **Zamjena konteksta** (*context switch*) je spremanje/učitavanje konteksta procesa (ili threada) do kojeg dolazi tijekom prebacivanja procesora na drugi proces ili thread.
- **Atomarna operacija**: Operacija koja se izvršava u potpunosti, bez prekida ili se uopće ne izvrši. Ne može se zaustaviti u sredini (nema zamjene konteksta).
- **Dijeljeno stanje**: Dio memorije koje više procesa/threadova može čitati ili pisati.
- **Kritični odsječak**: Dio koda koji pristupa (čita/piše) dijeljenim stanjima.
- **Problem kritičnog odsječka**: Osigurati da pristup svih kritičnih odsječaka skupu dijeljenih stanja bude (izgleda) atomaran. Naprimjer, thread A ne smije promatrati neko stanje dok thread B izvršava kritični odsječak u kojem mijenja to stanje.

Problem proizvođač-potrošač kao ilustracija zavisnih procesa

- Jedan proces puni (proizvodi), a drugi čita (troši) poruke.
- Neka dva procesa komuniciraju preko spremnika (*buffer*) ograničene veličine (`n` poruka).
- Proizvođač ne smije unositi poruke u pun spremnik.
- Potrošač ne smije čitati poruke iz praznog spremnika.
- Uvodi se varijabla `counter` koja broji poruke u spremniku. Inicijalno je sremnik prazan tj. `counter = 0`.

proizvođač

```
while (1)
{
    proizvedi poruku;
    while ( counter == n )
    /* buffer je pun, proizvodjac
      ceka da potrosac uzme nesto
      iz buffera */
    buffer[in] = poruka;
    in = in + 1;
    counter++;
}
```

potrošač

```
while (1)
{
    while ( counter == 0 )
    /* buffer je prazan, potrosac
      ceka da proizvodjac stavi
      nesto u buffer */
    poruka = buffer[out];
    out = out + 1;
    counter --;
    potrosi poruku;
}
```

Problem proizvođač-potrošač kao ilustracija zavisnih procesa

Problem: ovi procesi se mogu izvršavati istovremeno samo ako se operacije `counter++` i `counter--` izvršavaju **atomarno**, tj. nedjeljivo (bez prekidanja).
Po pravilu operacije `counter++` i `counter--` izvršavaju se na sljedeći način:

- vrijednost vraijable `counter` upisuje se u registar procesora,
- vrijednost registra se inkrementira ili dekrementira,
- vrijednost registra se spremi u varijablu `counter`.

counter++

```
register1 = counter
register1 = register1 + 1
counter = register1
```

counter–

```
register2 = counter
register2 = register2 - 1
counter = register2
```

Izvršava se šest strojnih instrukcija, ali u slučaju da globalne operacije koje ih sadrže nisu **atomarne**, potencijalni prekid i izbor procesa koji će nastaviti rad poslije prekida mogu dovesti do **preplitanja (interleaving)**, tj. do mješavine ovih instrukcija.

Problem proizvođač-potrošač kao ilustracija zavisnih procesa

Jedna vrst takvog preklapanja dana je u donjoj tablici, za početni vrijednost `countera` 5.

	izvodi	**naredba**	**stanje varijabli**
T1	proizvodjac	`register1 = counter`	register1 = 5
T2	proizvodjac	`register1 = register1 + 1`	register1 = 6
T3	potrosac	`register2 = counter`	register2 = 5
T4	potrosac	`register2 = register2 - 1`	register2 = 4
T5	proizvodjac	`counter = register1`	`counter` = 6
T6	potrosac	`counter = register2`	`counter` = 4

- Krajnja vrijednost zajedničkih podataka ovisi o slijedu instrukcija koje tim podacima pristupaju.
- Taj slijed naziva se **stanje trke (race condition)** i ovisi o prekidnim signalima i načinu raspoređivanja procesa.
- Krajnji rezultat slijeda je neizvjestan, što se treba izbjeći.

Interprocesna komunikacija

Osnovno pravilo usporednog (concurrent) programiranja je: "Rezultat (logička ispravnost) programa ne smije ovisiti o redoslijedu izvršavanja i preklapanju, tj. o raspoređivanju procesora."
Odnose među procesima koje ćemo analizirati možemo podijeliti u tri grupe:

- **međusobno isključivanje procesa** (mutual exclusion),
- **sinkronizacija procesa** (synchronisation),
- **uzajamno blokiranje - zastoj** (deadlock).

Međusobno isključivanje procesa

Međusobno isključivanje procesa je odnos među procesima koji ne dozvoljava istovremeno izvršavanje dva procesa u pojedinim njihovim dijelovima (**kritičnim sekcijama**), dok se u ostalim dijelovima procesi mogu nesmetano izvršavati.
Primjeri:

- procesi dijele neki nedjeljiv resurs npr. datoteka ili dio memorije. Primjer rezervacija avionskih karata.

Dio koda u kojem proces dijeli nedjeljivi resurs zove se **kritičan odsječak (critical section)**.
Zadatak operacijskog sustava je da prilikom korištenja nedjeljivog resursa tj. prilikom istovremenog izvršavanja kritične sekcije više procesa samo jednom procesu dozvoli izvršavanje.

Kritičan odsječak

- **Kritičan odsječak (critical section)** je dio koda u kome proces pristupa zajedničkim podacima kao što su memorijske varijable, tablice i datoteke – ili ih modificira.
- Svaki operacijski sustav može dozvoliti samo jednom procesu da bude u svom kritičnom odsječku.
- Dok je jedan proces u svom kritičnom odsječku, nijedan drugi proces ne smije ući u svoj kritičan odsječak.
- Prema tome, izvršavanje kritičkih odsječaka procesa međusobno je isključivo u vremenu (mutual exclusion).
- Da bi ovo osigurao, OS mora kontrolirati ulazak svakog procesa u svoj kritičan odsječak, odnosno proces mora prilikom ulaska u kritičan odsječak tražiti od OS-a pravo za tu operaciju.

Kritičan odsječak

Rješenje problema kritičnog odsječka mora zadovoljiti slijedeća tri zahtjeva:

1. **Međusobno isključivanje (mutual exclusion)**: ako jedan proces izvodi svoj kritičan odsječak tada nijedan drugi proces ne smije izvoditi svoj kritičan odsječak.
2. **Napredovanje (progress)**: ako ni jedan proces ne izvodi svoj kritičan odsječak, a postoje procesi koji čekaju da uđu u svoj kritičan odsječak, tada samo ti procesi mogu učestvovati u donošenju odluke koji proces će dobiti pravo da prvi uđe u kritičan odsječak. Ulazak u taj odsječak ne može biti odgađan unedogled.
3. **Ograničeno čekanje (bounded waiting)**: mora postojati vremensko ograničenje unutar kojega se mora omogućiti procesu T koji čeka da uđe u svoj kritičan odsječak da to učini (postoji gornja granica broja drugih procesa koji mogu ući u svoj kritičan odsječak prije procesa T).

Potrebno je pronaći rješenje kojim će se riješiti problem izvođenja kritičnog odsječka uvažavajući tri postavljena zahtjeva.

Kritičan odsječak

Prilikom razmatranja pretpostavka je da proces ima prikazanu strukturu:

```
do {
    entry section
       critical section
    exit section
      /* ostatak koda koji ne pripada kriticnom odsjecku */
} while(1);
```

- **Ulazni odsječak (entry section)** je dio koda u kojem proces traži od operativnog sustava ulazak u kritični odsječak.
- **Izlazni odsječak (exit section)** je dio koda kojim proces obavještava ostale procese da nije u kritičnom odsječku.

Kritičan odsječak

Određivanje dijela programskog koda koji predstavlja kritični odsječak zadatak je programera.
Kritični odsječak se može realizirati:

- softverski (algoritam striktne alternacije, Dekker-Petersonov algoritam, pekarski algoritam)
- hardverski (blokadom prekidnih signala – interrupt disable)
- pomoću semafora
- pomoću OS-a (sistemskim pozivima)
- višim programskim strukturama za sinkronizaciju (monitorima)

Semafori

- **Semafor** je cjelobrojna nenegativna varijabla koja štiti neki resurs i omogućava komunikaciju između procesa (mehanizam međusobnog isključenja ili sinkronizacije aktivnosti).
- Vrijednost semafora određuje da li je resurs koji taj semafor štiti slobodan.
- Ako je vrijednost semafora `val(s) = 0`, resurs je zauzet. Inače je resurs slobodan.
- Svaki semafor ima svoju početnu vrijednost, a nad njim se mogu izvršiti dvije atomarne (nedjeljive) operacije koje su radi jednostavnosti predstavljene kao `signal(s)` i `wait(s)`. Nedjeljivost ovih operacija ogleda se u slijedećim činjenicama:
  - signal i wait se ne mogu podijeliti na više ciklusa;
  - dva procesa ne mogu istovremeno izvršavati ove operacije nad istim semaforom.

  Simbolički, operacije `signal` i `wait` mogu se predstaviti na slijedeći način:

```
wait(s): when(s > 0) do decrement s;
signal(s): increment s;
```

Upotreba semafora

Problem kritičnog odsječka može se riješiti upotrebom binarnog semafora mutex (mutual exclusion), čija je inicijalna vrijednost 1.

```
semaphore mutex;    /* inicijalno, mutex = 1 */
do {
    wait(mutex);
    /* kriticni odsjecak */
    signal(mutex);
    /* ostatak koda */
} while(1);
```

Ulazni odsječak svih procesa riješava se operacijom `wait`. Samo jedan proces proći će semafor `mutex`, oboriti mu vrijednost na 0 i ući u svoj kritični odsječak. Svi drugi procesi čekaju na semaforu. Kada proces napusti kritični odsječak, osloboditi će semafor naredbom `signal(mutex)`.

Realizacija pomoću operacijskog sustava (proširena definicija semafora)

Osnovni nedostaci metoda za zaštitu kritičnog odječka su:

- ignorira se prioritet procesa
- mogućnost da proces bude zauzet čekanjem (**busy waiting**) (proces se vrti u `while` petlji i ne radi ništa korisno, nego samo provjerava vrijednost varijable kako bi saznao može li ući u kritičan odsječak). Npr. ako imamo 1000 procesa od kojih se jedan nalazi u kritičnom odsječku, a ostali čekaju da uđu u svoj, znači da jedan proces radi nešto korisno, a ostali samo opterećuju procesor.

Rješenje: za svaki semafor uvede se posebni red čekanja - **semaforski red**.
U proširenoj definiciji vrijednost semafora može biti negativna, što ukazuje na to da veći broj procesa čeka na semaforu. U takvim slučajevima, operacija signal po nekom kriteriju "budi" neki od "uspavanih" procesa.

```
typedef struct
{
    int value;
    struct proces *L;
}semaphore;
```

Proširena definicija semafora

Vrijednost semafora može biti:

- `val(s) > 0`, resurs je slobodan
- `val(s) = 0`, resurs je zauzet, ali je semaforski red prazan
- `val(s) < 0`, resurs je zauzet, u semaforskom redu postoje procesi koji čekaju na taj semafor

Operacija wait dekrementira vrijednost semafora i, ako je ona nakon toga negativna, stavlja proces u semaforski red i blokira proces:

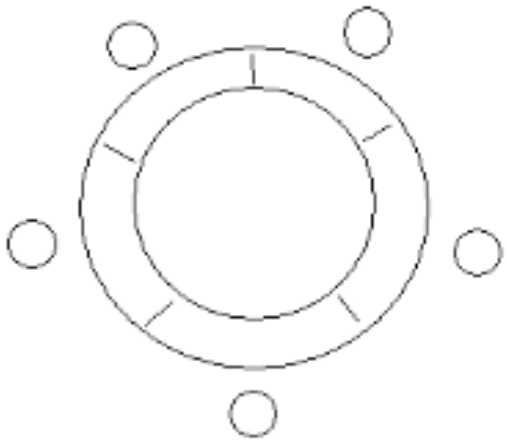
```
wait(S){
   S.value--;
   if (S.value < 0){
      /* dodaj proces u semaforski red */
      sleep();
   }
}
```

Operacija signal inkrementira vrijednost semafora i, ako je ona poslije toga negativna ili jednaka nuli, uklanja jedan od procesa iz liste (izvršava sistemski poziv wakeup i prebacuje proces u red čekanja za spremne procese).

```
signal(S){
   S.value++;
   if (S.value <= 0){
      /* ukloni proces iz semaforskog reda */
      sleep();
   }
}
```

Klasični problemi sinkronizacije i semafori

Problem večere filozofa: Zamislimo pet filozofa koji provode život samu u hranjenju i razmišljanju. U sredini sobe za ručanje nalazi se okrugli stol sa pet stolica. U sredini stola nalazi se velika posuda s hranom, ali ima samo 5 štapića za jelo.



- Kad filozof ogladni, sjeda na stolicu i uzima dva štapića koja su mu najbliža.
- Ako može uzeti oba štapića, onda može početi s jelom.
- Kad završi s jelom, odlaže štapiće i vraća se razmišljanju.

Problem gladnih filozofa

Svaki filozof predstavlja proces, a svaki štapić semafor. Definiramo zajedničke podatke – pet semafora kojima se štite štapići (stapici[i]), čija je inicijalna vrijednost 1.

`semaphore stapici[5];`

Kod koji rješava problem večere filozofa, za svakog filozofa posebno, glasi:

```
filozof(i)
  do{
     wait(stapici[i]);  /*jedan stapic */
     wait(stapici[(i + 1) % 5]);  /*drugi stapic */
        /* filozof jede */
     signal(stapici[i]);
     signal(stapici[(i + 1) % 5]);
       /* filozof razmislja */
  } while(1);
```

Zastoj je moguć ako svi filozofi ogladne odjedanput i uzmu štapić s lijeve strane.

Problem gladnih filozofa

Zastoj se može spriječiti ako se:

- dozvoli da najviše četiri filozofa sjednu za stol;
- dozvoli da filozof uzme štapiće samo ako su oba slobodna;
- koriste asimetrična rješenja – npr. da neparni filozofi (prvi, treći i peti) uzmu prvo štapiće s lijeve strane, a zatim s desne, a parni obratno.

Monitori

Monitori su konstrukcija visokog nivoa koja služi za sinkronizaciju.
Oni omogućuju programeru da resurse promatra kao objekte (struktura monitora podsjeća na instance klasa u objektno orijentiranom programiranju).
Svaki monitor se sastoji od:

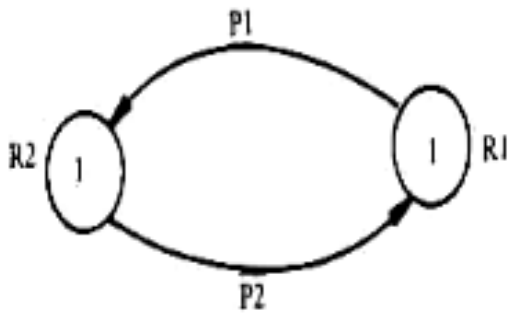
- varijabli koje opisuju dijeljeni resurs, tj. objekat, čije vrijednosti definiraju stanje monitora. Lokalne varijable su vidljive samo unutar monitora;
- skupa procedura i funkcija kojima se pristupa objektu, tj. varijablama monitora;
- dijela programa koji inicijalizira objekt, a koji se izvršava samo jednom, prilikom stvaranja objekta.

```
monitor ime_monitora{
   /* deklaracije dijeljenih varijabli */
   P1 (...) { /* definicija funkcije P1 */}
   P2 (...) { /* definicija funkcije P2 */}
   P3 (...) { /* definicija funkcije P3 */}
   { /* inicijalizacija monitora */}
}
```

Konstrukcija monitora rješava problem sinkronizacije, tj. dozvoljava da samo jedan proces bude aktivan u monitoru.

Uzajmno blokiranje procesa - zastoj

Pojavu deadlock-a je najlakše ilustrirati na primeru dva procesa p1 i p2, pri čemu svaki od njih "drži" resurs potreban onom drugom za nastavak izvršavanja.



Uzajamno blokiranje procesa - zastoj

Pristupi rješavanju problema zastoja su:

1. izbjegavanje zastoja (od procesa se traži da unaprijed specificira resurse koji su mu potrebni);
2. spriječavanje zastoja (resursi se numeriraju, proces može tražiti resurse samo u rastućem nizu rednih brojeva);
3. otkrivanje zastoja (prekidanje procesa koji su u zastoju i oduzimanje resursa);
4. zanemarivanje zastoja.

Još neki problemi sinkronizacije

- Problem uspavanog brijača:
  Brijačnica ima čekaonicu sa n stolica i radnu sobu sa brijačkom stolicom. Ako nema mušterija koje traže uslugu, brijač će zaspati. Ako mušterija uđe u brijačnicu i vidi da su sve stolice zauzete, izaći će napolje. U slučaju da je brijač zauzet, ali u čekaonici ima slobodnih stolica, mušterija će sjesti na jednu od njih i sačekati. Ako brijač spava, mušterija će ga probuditi. Napisati program kojim će aktivnosti barbarina i mušterija biti sinkronizirane.
- Problem pušača:
  U sustavu postoje tri procesa koja predstavljaju pušače i jedan proces koji predstavlja dobavljača. Svaki pušač u beskonačnoj petlji prvo mota cigaretu, a zatim je i pali. Da bi pušač smotao cigaretu, potrebni su mu slijedeći sastojci: duhan, papir i šibice. U početnom stanju, jedan od pušača ima papir, drugi duhan, a treći šibice. Dobavljač ima neograničene zalihe ovih sastojaka, ali im donosi samo po dva sastojka. Pušač koji ima preostali sastojak smotat će cigaretu i signalizirati dobavljaču da je završio cigaretu. Dobavljač će zatim ponovo donijeti dva sastojka, čime se ciklus ponavlja. Napisati program kojim će aktivnosti pušača i dobavljača biti sinkronizirane.

Problem paralelnog čitanja/pisanja

Na primjer: Dijeljena baza podataka (rezervacija mjesta u avionu, bankovni računi).
Dvije vrste korisnika:

- Čitači nikad ne modificiraju bazu podataka.
- Pisači čitaju i modificiraju bazu podataka.

Jednostavni monitor nas ne zadovoljava jer želimo omogućit pristup više čitača u isto vrijeme.
Uvjeti koje moramo zadovoljiti:

- Čitač može pristupiti bazi kad nema ni jedan pisač (uvjet uredu_čitati).
- Pisač može pristupiti bazi ako nema ni čitača ni pisača (uvjet uredu_pisati).
- Samo jedan thread može modificirati stanje varijabli (mutex).

12 Upravljanje memorijom

Osnovni pojmovi

Postoje razne "vrste" memorije: RAM, cache, fizička ili virtualna memorija, registri. Osnovni pojmovi vezani uz memoriju su:

- **Fizička memorija** ili **RAM** (random-access memory) je glavna radna memorija procesora. Da bi procesor mogao obraditi neki podatak, on se mora nalaziti u memoriji ili registru.
  - Memorija je niz byteova ili riječi. Svaki byte ima svoju **adresu**.
  - **Adresa** je pozitivan cijeli broj koji jednoznačno omogućava dohvat nekog podatka.
  - **Brzina memorije** (ili vrijeme pristupa) je vrijeme potrebno da procesor iz memorije dohvati traženi podatak.
- **Registri** su "memorija" koja se nalazi na samom chipu procesora i kao takva je jako brza, ali registara ima jako malo (obično ¡ 128).
- **CPU cache** je memorija koja je manja, brža i puno skuplja od RAM-a. U nju se spremaju najčešće korišteni podaci iz glavne memorije. Kada procesor treba pročitati nešto iz RAM-a prvo provjerava da li je taj podatak zapisan u *cache* memoriji.
  - Postoje razne *razine* cache memorije: L1, L2 i L3 (level 1, 2, 3) nalaze se na istom chipu kao i procesor. Višejezgreni chipovi obično sadrže po jedan L1 cache po jezgri, L3 dijele sve jezgre, a L2 cache može dijeliti više jezgri (npr. po dvije).

Još neki osnovni pojmovi

- **MMU** (memory management unit) je dio procesora koji prevodi virtualne adrese u fizičke. Sa stanovišta MMU, fizička memorija je podijeljena na *page frame-ove* fiksne veličine, obično 4kb ili 8kb (kod nekih arhitektura procesora je to konfigurabilno).
- **Page tables** su strukture podataka pomoću kojih MMU radi translaciju virtualnih u fizičke adrese.
- **Swap** je unaprijed rezervirani prostor na disku na koji OS sprema dijelove "neaktivnih" programa kada ponestane fizičke memorije.
- **Buffer cache** je dio RAM-a koji OS rezervira radi bržeg čitanja sa diska. Kada je potrebno dohvatiti sektor sa diska, OS prvo pogleda da li se taj sektor već nalazi u buffer cacheu i tek ako ga tamo nema, dohvaća se sa diska (koji je mnogo sporiji od RAM-a).

13 Mehanizmi upravljanja memorijom
- Virtualni adresni prostor
- Kontinuirana alokacija
- Diskontinuirana alokacija

Adresni prostor

Dva su problema o kojima treba voditi računa kod dodjeljivanja memorije procesima:

- korisnički programi ne mogu pristupiti svakoj adresi u memoriji,
- više korisničkih programa treba se istovremeno izvršavati.

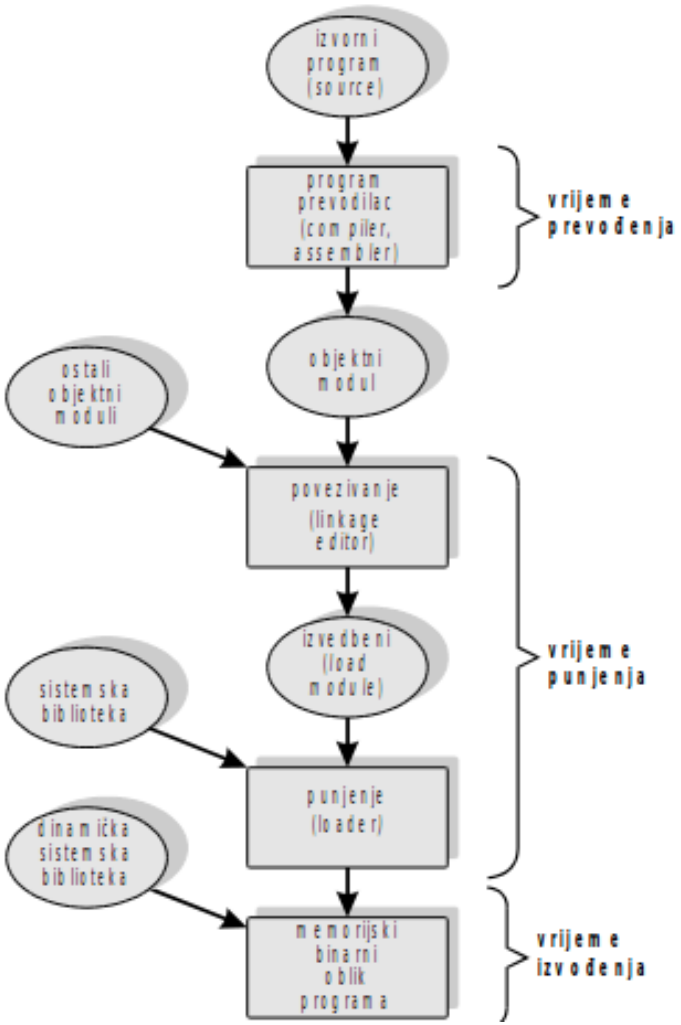
Adresni prostor je skup adresa koje proces može koristiti da bi adresirao memoriju.
Svaki proces ima svoj adresni prostor.

- Svaki proces u trenutku stvaranja dobiva 32-bitni ili 64-bitni adresni prostor.
- Proces ima *iluziju* da ima svu memoriju za sebe.
- Kada adresira memoriju, on zapravo adresira **virtualnu adresu**.
- MMU pretvara virtualnu adresu u fizičku adresu.
- Sukladno navedenim definicijama, adresni prostor koji koristi program prije punjenja u memoriju naziva se **logički (ili virtualni) adresni prostor** koji za vrijeme izvođenja (nakon punjenja u memoriju) prelazi u **fizički adresni prostor**. Tako se za vrijeme izvođenja, logički i fizički adresni prostori razlikuju.

Izvršavanje programa

- Programi su zapisani na disku ili nekoj drugoj neizbrisivoj memoriji u binarnom obliku (**executable**).
- Program se s diska upisuje u memoriju kako bi postao *proces* koje se može izvoditi.
- Ovisno o sustavu za upravljanje memorijom proces se može i tijekom obrade prebacivati iz radne memorije na disk i obratno (**swap**).
- Skup procesa na disku koji čekaju da budu upisani u memoriju naziva se **ulazni red** (input queue).
- Normalna procedura se sastoji u odabiru jednog od procesa iz ulaznog reda i upisa tog procesa u radnu memoriju.
- Tijekom izvođenja procesor dohvaća iz radne memorije naredbe i podatke te u nju upisuje rezultate obrade.
- Po završetku obrade memorijski prostor koji je proces koristio proglašava se **slobodnim**.

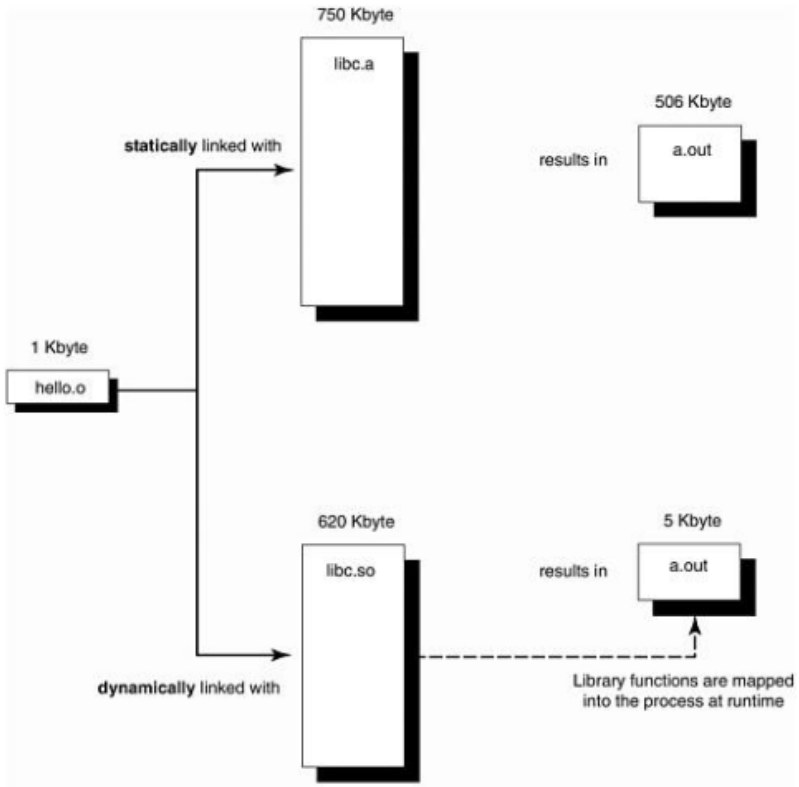
Generiranje adrese



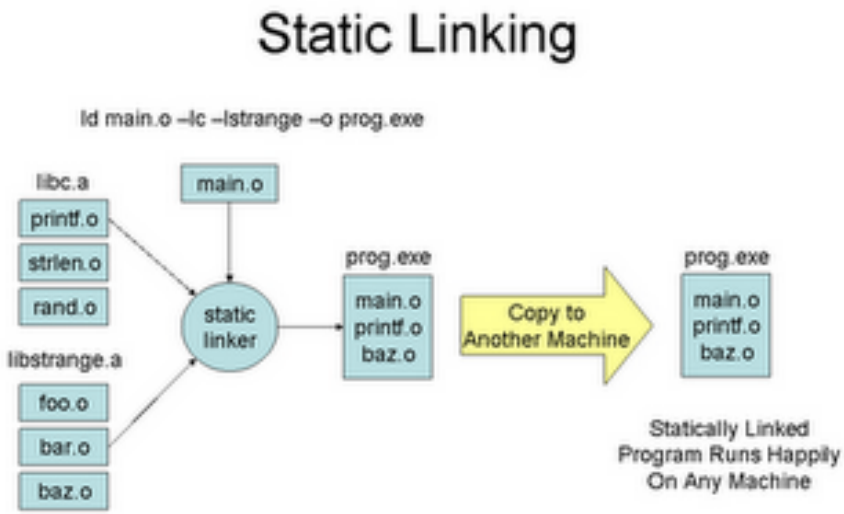
Dinamičko punjenje - dynamic loading

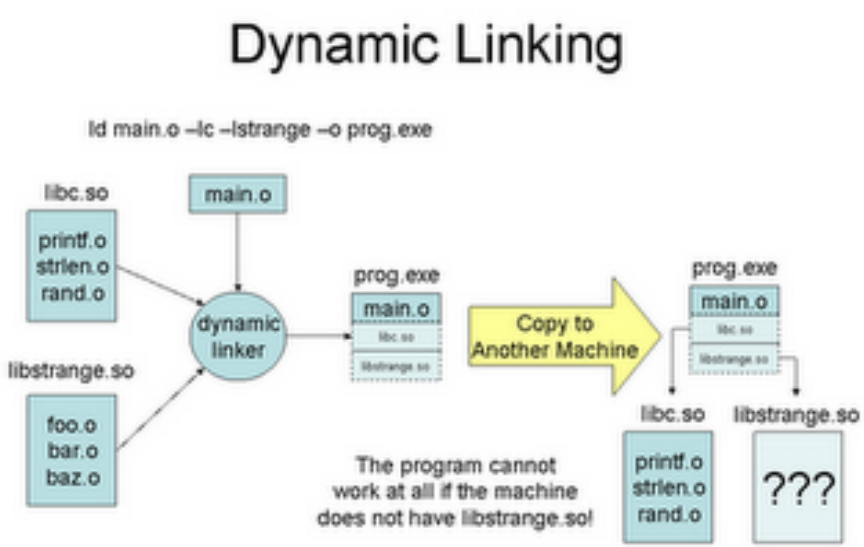
- Kod ovakvog načina punjenja programa u memoriju potprogrami ili rutine smještene su na disku, a upisuju se u memoriju tek nakon što ih glavni program pozove.
- Kod poziva potprograma, prvo se provjerava da li je potprogram u radnoj memoriji, a ako nije poziva se program za dinamičko punjenje i povezivanje (**relocatable linkage editor**) koji upisuje potprogram u memoriju te upisuje početnu adresu potprograma u tablicu adresa programa.
- Potom se izvođenje proslijeđuje na pozvani potprogram.
- Potprogrami ili procedure koje se nikad ne koriste ovim pristupom nikad neće biti upisane u memoriju što je prednost dinamičkog punjenja.

Dinamičko povezivanje - dynamic linking



Dinamičko povezivanje - dynamic linking





Dinamičko povezivanje - dynamic linking

Statičko linkanje

- Statičke biblioteke (libraries) (.lib na Windowsima, .a na unixoidima) koje program koristi povezuju se sa programom u samostalni (stand alone) izvršni file.
- Memorija za funkcije statičkih biblioteka dodjeljuje se prilikom izvršavanja programa (runtime).

Dinamičko linkanje

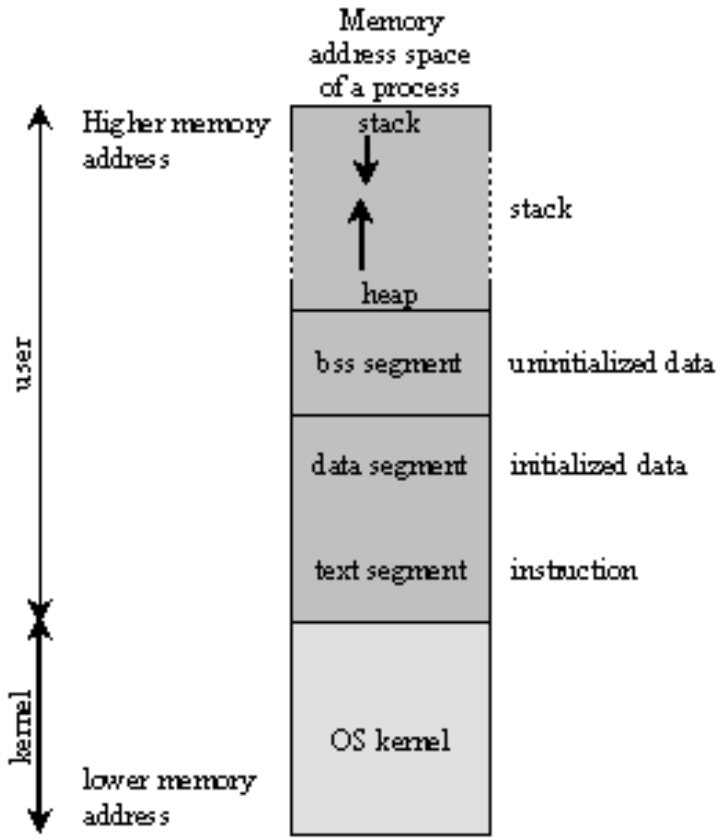
- Dinamičke biblioteke (.dll na Windowsima, .so na unixoidima) povezuju se sa programom koji ih koristi prilikom izvršavanja ili punjenja (loading) programa.
- Memorija za funkcije dinamičkih biblioteka dodjeljuje se kada program pozove funkciju, dakle, po potrebi.
- Dinamičko povezivanje zahtjeva uslugu operacijskog sustava.

Izvršavanje programa

C program se smješta u memoriju u nekoliko odvojenih segmenata unutar virtualnog adresnog prostora procesa:

- **Code segment** ili segment koda je dio u kojem je zapisan kod programa (**binary**), uključujući biblioteke koje koristi. Taj dio memorije je *read-only* i proces ga može dijeliti sa drugim procesima koje imaju isti kod ili koriste zajedničke biblioteke.
- **Stack segment** je dio u koji se spremaju vrijednosti stacka procesa. Veličina stacka definirana je kod pokretanja procesa i ne može se povećati u *runtimeu*.
- **Data segment** ili podatkovni dio zauzima prostor od stack segmenta do kraja virtualnog adresnog prostora. Ovaj segment može po potrebi mijenjati veličinu.

Memorija dodjeljena procesu



Stack i heap

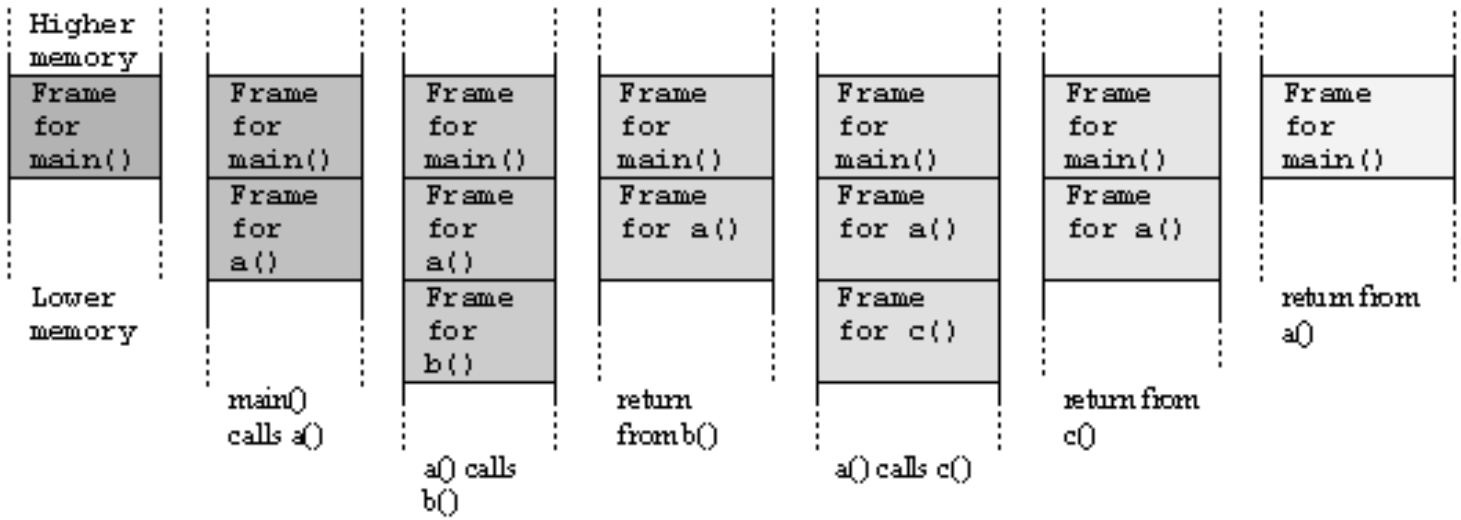
Stack procesa je dio memorije u kojem se nalaze lokalne (auto) varijable, parametri funkcije, privremene informacije (temporary information) i povratne adrese.

- Kad se pozove funkcija, **stack frame** se kreira i gurne (**push**) na vrh stacka. **Stack frame** sadrži adresu s koje je funkcija pozvana i gdje će se vratiti nakon što se funkcija izvrši, parametre funkcije, lokalne varijable i sve što funkciji treba za izvršavanje.
- Kad funkcija završi njen stack frame se skine (**pop**) sa stacka.
- Tipično stack se puni prema dolje, tj. prema nižim adresama (prema heap-u).
- Veličina stacka se ne može mijenjati u *runtimeu*.

Heap je dio memorije unutar adresnog prostora procesa pomoću kojeg vršimo dinamičku alokaciju proizvoljnih blokova memorije za potrebe naše aplikacije.

- Veličina heap-a je promjenjiva.
- Do podataka na *heapu* se može doći samo preko pointera.
- Heap je uvijek lokalan unutar jednog procesa, ali ga dijele threadovi unutar jednog procesa.
- Osim standardnog heapa, proces može kreirati više heapova unutar svog adresnog prostore.
- U c-u dinamička alokacija se izvodi pomoću `malloc()`, `calloc()`,

Memorija dodjeljena procesu



Transformacija iz logičkog (virtualnog) u fizički adresni prostor

- Programer ne može unaprijed odrediti fiksne memorijske lokacije za smještaj programa, i zato koristi **relativne** ili **simboličke** adrese (ovisno o programskom jeziku).
- Zadaća operacijskog sustava je da prevede **relativne (relocatable) adrese** u **fiksne** prilikom učitavanja programa u memoriju, tj. da veže adrese.

Prije samog izvršavanja, korisnički program obično prolazi kroz više faza, tokom kojih se memorijske adrese predstavljaju na različite načine.

- **Prevodioc (compiler)** prevodi **izvorni kôd** programa i vezuje (**bind**) simboličke adrese iz izvornog kôda za relativne adrese, koje **punioc (loader)** pretvara u apsolutne prilikom učitavanja programa u memoriju.

Npr. prevodioc vezuje varijablu `count` za lokaciju na adresi 14 u odnosu na početak programskog modula, koju punioc pretvara u apsolutnu adresu 74704.

- Vezivanje adresa se može shvatiti kao transformacija, tj. prevođenje sa "jezika" jednog adresnog prostora u drugi.

Transformacija iz logičkog (virtualnog) u fizički adresni prostor

Dodjeljivanje (vezivanje) adresa naredbama i podacima moguće je realizirati u bilo kojoj fazi pripreme programa za izvođenje:

- **Vrijeme prevođenja**: Ukoliko je za vrijeme prevođenja poznata adresa lokacije od koje će se upisivati program, tada se već za vrijeme prevođenja generiraju apsolutne adrese.
- **Vrijeme punjenja**: Kod suvremenijih sustava za vrijeme prevođenja obično nije poznato gdje će se upisati program. Tada program prevodilac generira relativne adrese, a apsolutne se generiraju u trenutku upisivanja (punjenja) programa u radnu memoriju. Ovakav pristup je fleksibilniji od prethodnog zato što je kod promjene položaja programa u memoriji (npr. prijenos programa s jednog računala na drugo) potrebno samo ponovo upisati program u memoriju, a program punjač automatski mijenja apsolutne adrese. Program nije potrebno ponovo prevoditi.
- **Vrijeme izvođenja**: U ovom slučaju moguće je da program mijenja svoje adrese i tijekom izvođenja, odnosno prebacuje se iz jednog u drugo memorijsko područje (uključujući i disk, ukoliko se koristi virtualna memorija). Posebna hardverska podrška potrebna je za realizaciju ovakvog načina rada.

Transformacija iz logičkog (virtualnog) u fizički adresni prostor

Funkciju preslikavanja (relokacija) iz logičkog u fizički adresni prostor realizira sklop za upravljanje memorijom (**memory management unit**).

- Jedan od načina za generiranje adresa je da se logičkim adresama, koje počinju od nula, dodaje veličina koja je zapisana u tzv. **relokacijskom registru**.
- Ovim pristupom logički adresni prostor, koji počinje na adresi 0, preslikava se u fizički adresni prostor, a koji počinje na adresi koju određuje relokacijski registar.
- Logički adresni prostor koji se nalazi u intervalu [0, max] se mapira u interval [R + 0, R + max], gdje je R vrijednost relokacijskog registra, tj. fizička adresa početka programa.

Zaštita memorije

Zaštita operacijskog sustava od korisničkih procesa i međusobna zaštita korisničkih procesa po pitanju pristupa memorijskim sekcijama može se realizirati pomoću dva registra:

- relokacijskog registra, koji sadrži najnižu adresu procesa;
- registra ograničenja, koji sadrži najveći opseg logičkih adresa procesa.

Jedinica za upravljanje memorijom (MMU) mapira svaku logičku adresu procesa dinamički, tako što provjerava da li je logička adresa manja od vrijednosti registra ograničenja, i ako je, dodaje vrijednost relokacijskog registra.

Dodjela memorije procesima

- Memorija se dijeli na najmanje dvije **particije**, od kojih je jedna (najčešće niži dio) namjenjena rezidentnom dijelu operativnog sustava (**kernel space**), a druga particija korisničkim programima (**user space**).
- U korisničkom adresnom prostoru nalazi se više procesa koji formiraju red čekanja na procesor.
- Također, postoje i procesi koji nastoje ući u red čekanja, obično sa diska.
- Da bi proces ušao u red čekanja na procesor, mora prvo dobiti potrebnu memoriju
- Tehnike dodjele memorije procesima mogu se podijeliti na dvije vrste:
  - **Kontinuirana alokacija** (contiguous allocation) - i logički i fizički adresni prostor procesa sastoje se od kontinuiranog niza memorijskih adresa.
  - **Diskontinuirana alokacija** (discontiguous allocation) - fizički adresni prostor nije kontinuirani niz; diskontinuirana alokacija obuhvaća metode **straničenja** (paging), segmentacije i straničenja sa segmentacijom.

Kontinuirana alokacija

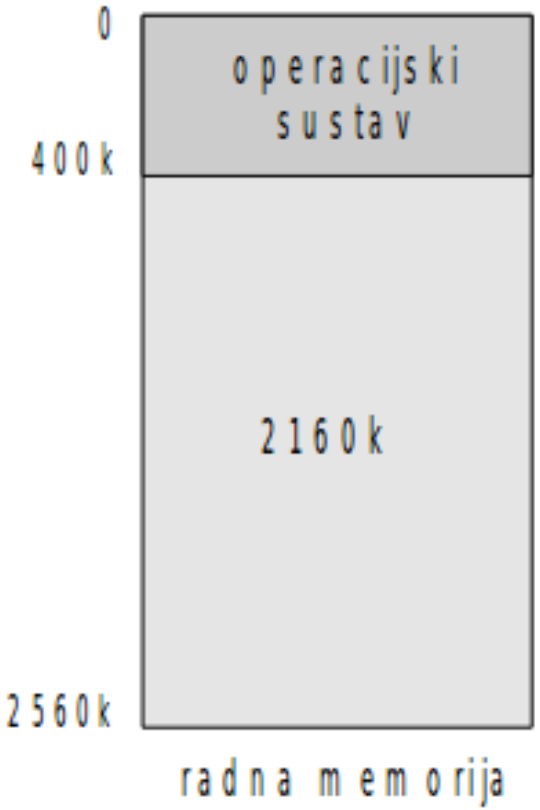
Metode kontinuirane alokacije su

- multiprogramiranje sa fiksnim particijama - svaka particija sadrži jedan proces i ima svoj red čekanja,
- multiprogramiranje sa particijama različite veličine - početno je cijeli memorijski prostor tretiran kao jedna particija, a proces koji dolazi dobiva onoliko prostora koliko mu je potrebno. Kada naiđe novi proces operativni sustav ispituje da li postoji dovoljno velik raspoloživi slobodni prostor. Ako neki proces završi s obradom, oslobađa se njegova memorija i može se dodijeliti drugom procesu.

Kontinuirana alokacija, multiprogramiranje sa particijama različite veličine, primjer

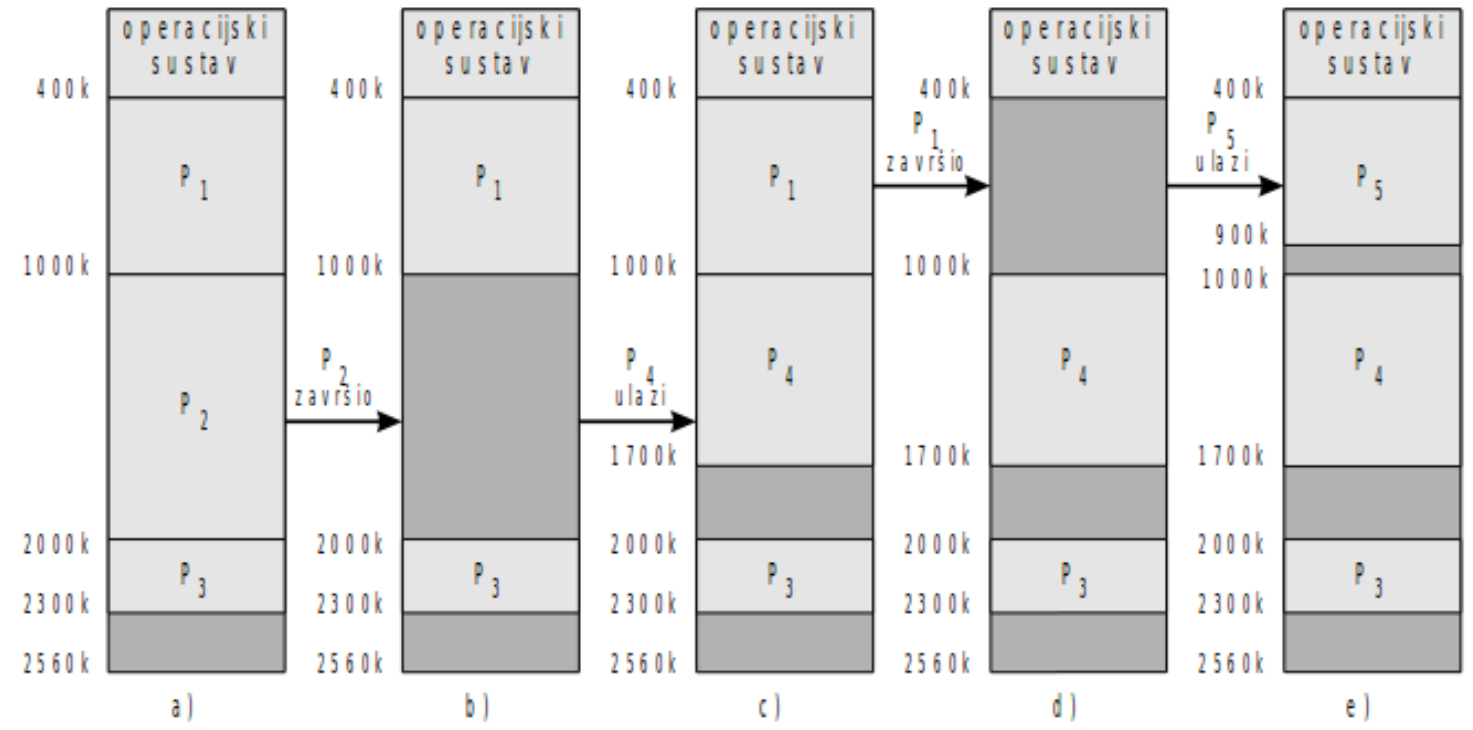
- Neka sustav ima na raspolaganju 2560k memorije od koje operacijski sustav koristi 400k.
- Korisniku je na raspolaganju ostalo 2160k memorije.
- Neka sustav prihvati na obradu procese P1 – P5, trajanja upisanog u tablici na slici i neka se za dodjelu procesora koristi Roun-Robin algoritam s vremenskim kvantom 1.
- Sustav za prihvat poslova na obradu u memoriju će prvo upisati proces P1 i započeti s njegovim izvođenjem.
- U tom trenutku zauzeto je 1000k memorije, a slobodno 1560k.
- Paralelno izvođenju procesa P1 upisati će se u slobodnu memoriju i proces P2 čiji zahtjevi na memoriju su manji od slobodnih kapaciteta.
- Sada je zauzeto 2000k memorije, a slobodno 560k.
- Slobodan prostor dovoljan je da se upiše proces P3 nakon čega ostaje slobodno svega 260k memorije što nije dovoljno da se upišu sljedeći procesi.

Memorija dodjeljena procesu



red procesa prihvaćenih na obradu		
proces	memorijski zahtjevi	trajanje
P_1	600k	10
P_2	1000k	5
P_3	300k	20
P_4	700k	8
P_5	500k	15

Memorija dodjeljena procesu



Fragmentacija memorije

- U ovakvom sustavu dodjele memorije privilegirani su manji programi.
- Naime, s vremenom u memoriji se nalazi veći broj manjih programa koji su neravnomjerno raspoređeni, a također i veći broj slobodnih mjesta manjeg kapaciteta, tzv. šupljina, koji su također razbacani po memoriji.
- Veći programi se ne mogu upisati u memoriju jer ne postoji kontinuirani slobodni prostor potrebne veličine iako je ukupna slobodna memorija dovoljna.
- Ovaj problem naziva se **eksterna fragmentacija memorije**.

Metode odabira slobodnih particija

Neka postoji više (m) slobodnih segmenata u koji se može upisati sljedeći proces. Postavlja se pitanje koji od njih odabrati. Moguća su sljedeća rješenja:

- Prvi koji zadovoljava (**first-fit**): Procesu se dodjeljuje prvi segment koji zadovoljava postavljene memorijske zahtjeve. Obično pretraživanje počinje od početka liste slobodnih segmenata ili se nastavlja od mjesta gdje je prethodno ispitivanje zaustavljeno.
- Najbolje poklapanje (**best-fit**): Procesu se dodjeljuje onaj segment koji nabolje odgovara njegovim memorijskim zahtjevima. Iako na prvi pogled ovim pristupom se najbolje iskorištava slobodna memorija, rezultat je stvaranje malih segmenata, šupljina, koji su posljedica razlike veličine segmenta i programa.
- Najlošije poklapanje (**worst-fit**): Procesu se dodjeljuje najveći slobodni segment. Ovaj algoritam ima za cilj stvaranje što većih šupljina, suprotno prethodnom algoritmu.

Interna fragmentacija

Kod dodjele memorije susreće se i slijedeći problem.

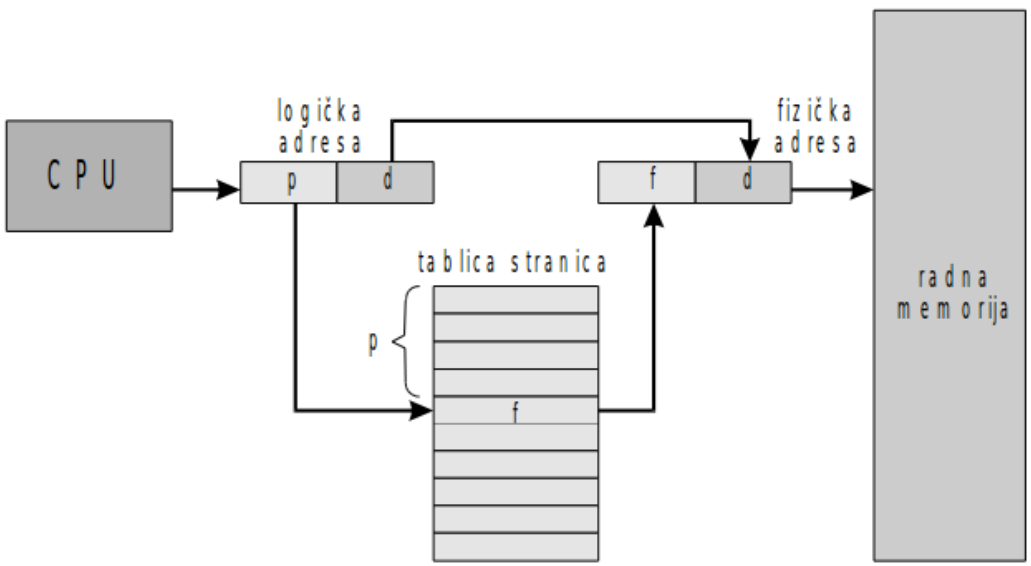
- Pretpostavimo da postoji slobodan memorijski blok veličine 18,464 byte, a program zahtjeva 18,460 byte.
- Tada je razumno dodijeliti programu cijeli blok, bez obzira što će 4 byte-a ostati neiskorišteno.
- Ovo je bolje nego da ta četiri byte-a tvore slobodan byte za koji je potrebno voditi evidenciju u operacijskom sustavu.
- Tako dodijeljena memorija može biti veća od memorije koju zahtjeva proces.
- Razlika u zahtjevanoj i dodijeljenoj memoriji naziva se **unutarnja (interna) fragmentacija**, koja je praktički zanemariva u usporedbi s vanjskom.

Diskontinuirana alokacija

- Radna memorija dijeli se na manje blokove fiksne veličine koji se nazivaju **okviri (frames)**.
- Logički adresni prostor programa također se podjeli na blokove iste veličine koji se nazivaju **stranice (pages)**.
- Kada se program upiše u memoriju stranice se upisuju u slobodne memorijske okvire.
- Radi jednostavnosti prebacivanja programa s diska u radnu memoriju i disk je podijeljen na okvire jednake veličine kao i okviri memorije.
- Tako se jedan okvir s diska upisuje u jedan okvir radne memorije.

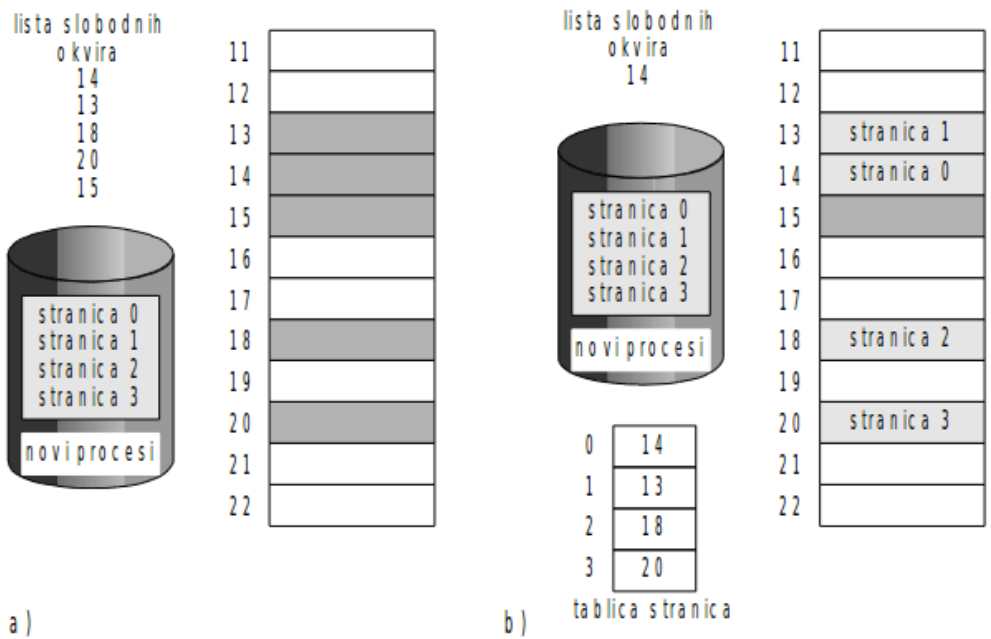
Diskontinuirana alokacija - straničenje

- Adresa koju generira procesor dijeli se na dva dijela: **broj stranice** p i **pomak** unutar stranice (page offset) d.
- Broj stranice je indeks (pokazivač) na redak tablice stranica.
- U tablici stranica upisane su početne adrese okvira u kojima je smještena stranica.
- Kombinacija početne adrese okvira i pomaka određuje fizičku adresu memorijske lokacije kojoj se pristupa.



Sustav za dodjelu memorije po stranicama djeluje na sljedeći način:

- Kad se program prihvati na izvođenje izračuna se potreban broj okvira i uspoređuje se s brojem slobodnih okvira u memoriji.
- Ukoliko je slobodan dovoljan broj okvira proces se upisuje u memoriju stranicu po stranicu.
- Istovremeno se za svaku stranicu u tablici stranica upisuje i broj okvira u koji je ona upisana. Ovaj proces prikazan je na slici.



14 Virtualna memorija

Virtualna memorija

- **Virtualna memorija** je strategija dodjele memorije koja dozvoljava da samo dio programa koji se izvodi bude u radnoj memoriji.
- Temeljna prednost ovakvog pristupa je da program može biti i veći od radne memorije.
- Tako korisnički program može poprimiti proizvoljnu veličinu, a sustav za upravljanje memorijom preslikava logički prostor korisnika u ograničeni prostor u radnoj memoriji.
- Ovakav sustav za upravljanje memorijom nije jednostavno realizirati. Loša implementacija ovakvog sustava može značajno smanjiti performanse cjelovitog računarskog sustava.
- U ovom poglavlju razmatrati će se realizacija straničenja na zahtjev (demand paging).

Virtualna memorija

Analize programa ukazuju da obično nije potreban cijeli program da bi se potrebna obrada izvela. Primjeri:

- Programi sadrže procedure za obradu slučajnih ili namjernih pogrešaka. Budući da se takvi slučajevi relativno rijetko dešavaju, program izvede potrebnu obradu bez poziva spomenutih procedura.
- Program za polja, liste, tablice, i slične statičke strukture obično rezervira više memorije nego je stvarno potrebno.
- Pojedine opcije programa relativno se rijetko koriste. Tako npr. pravnici kad pišu u Word tekst procesoru vjerojatno nikad neće koristiti Equation editor.

Čak i u slučajevima koji ne spadaju u navedene kategorije, činjenica je da cijeli program nije istovremeno potreban

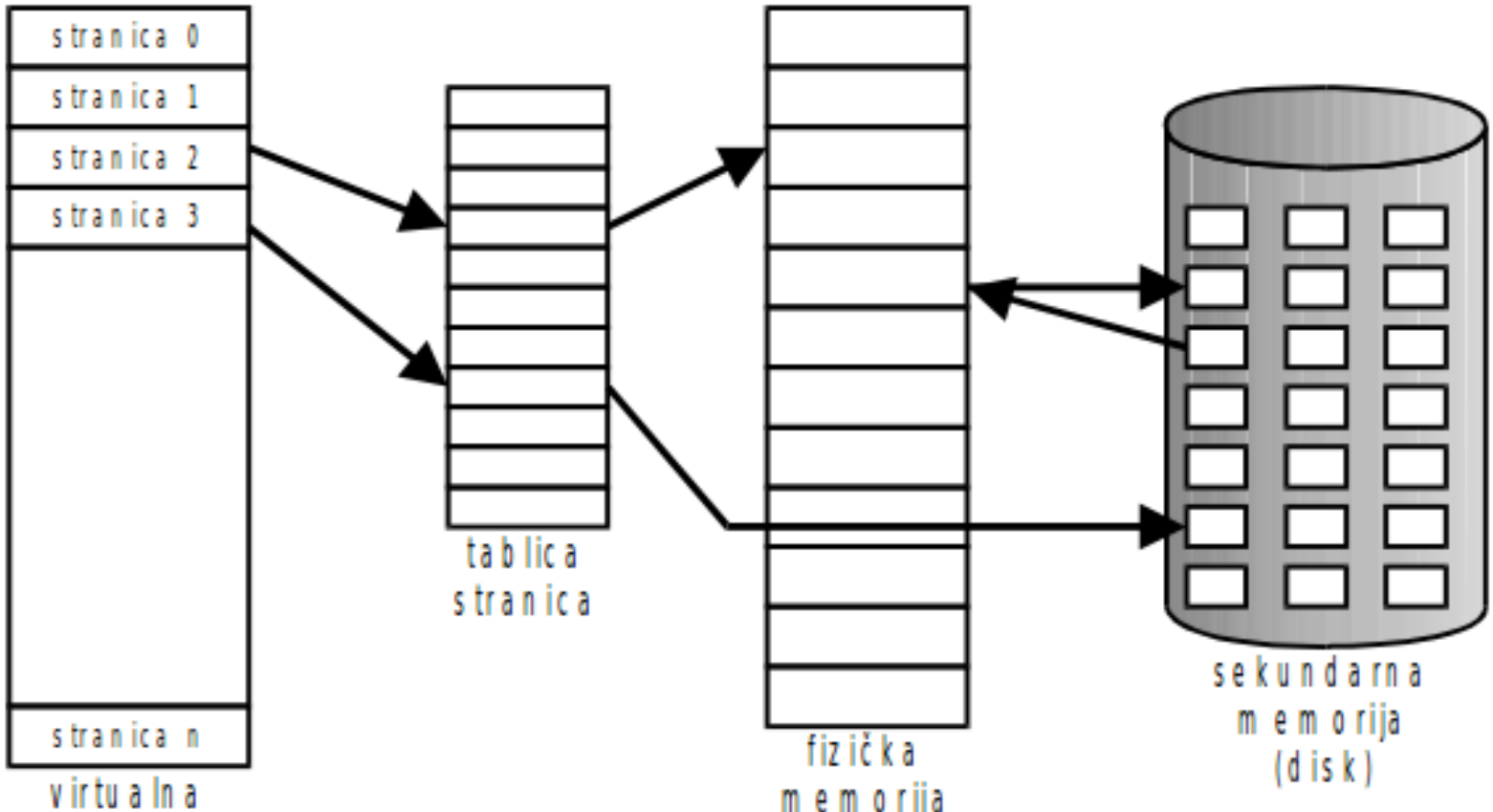
Virtualna memorija

Svojstvo da samo dio programa koji se izvodi se nalazi u memoriji ima niz prednosti:

- Veličina programa nije ograničena veličinom radne memorije. Programer može napisati programa kao da sustav raspolaže s neograničenom memorijom, odnosno programer raspolaže s neograničenim virtualnim logičkim adresnim prostorom. Ovo uveliko pomaže kod pisanja programa.
- Korisnički program može se izvoditi sa znatno manjom dodijeljenom fizičkom memorijom, što omogućava veći stupanja višeprogramskog rada. Time se povećava iskoristivost kao i propusna moć sustava.
- Manje U/I operacija potrebno je za prebacivanje korisničkih programa iz i u memoriju. Tako se korisnički programi brže izvode.

Virtualna memorija

Virtualna memorija je razdvajanje logičkog adresnog prostora koji vidi korisnik od fizičkog adresnog prostora u kojem se program izvodi.
Ovo razdvajanje omogućava programeru da raspolaže s neograničenim logičkim prostorom iako se program stvarno izvodi u relativno malom fizičkom adresnom prostoru.



Virtualna memorija

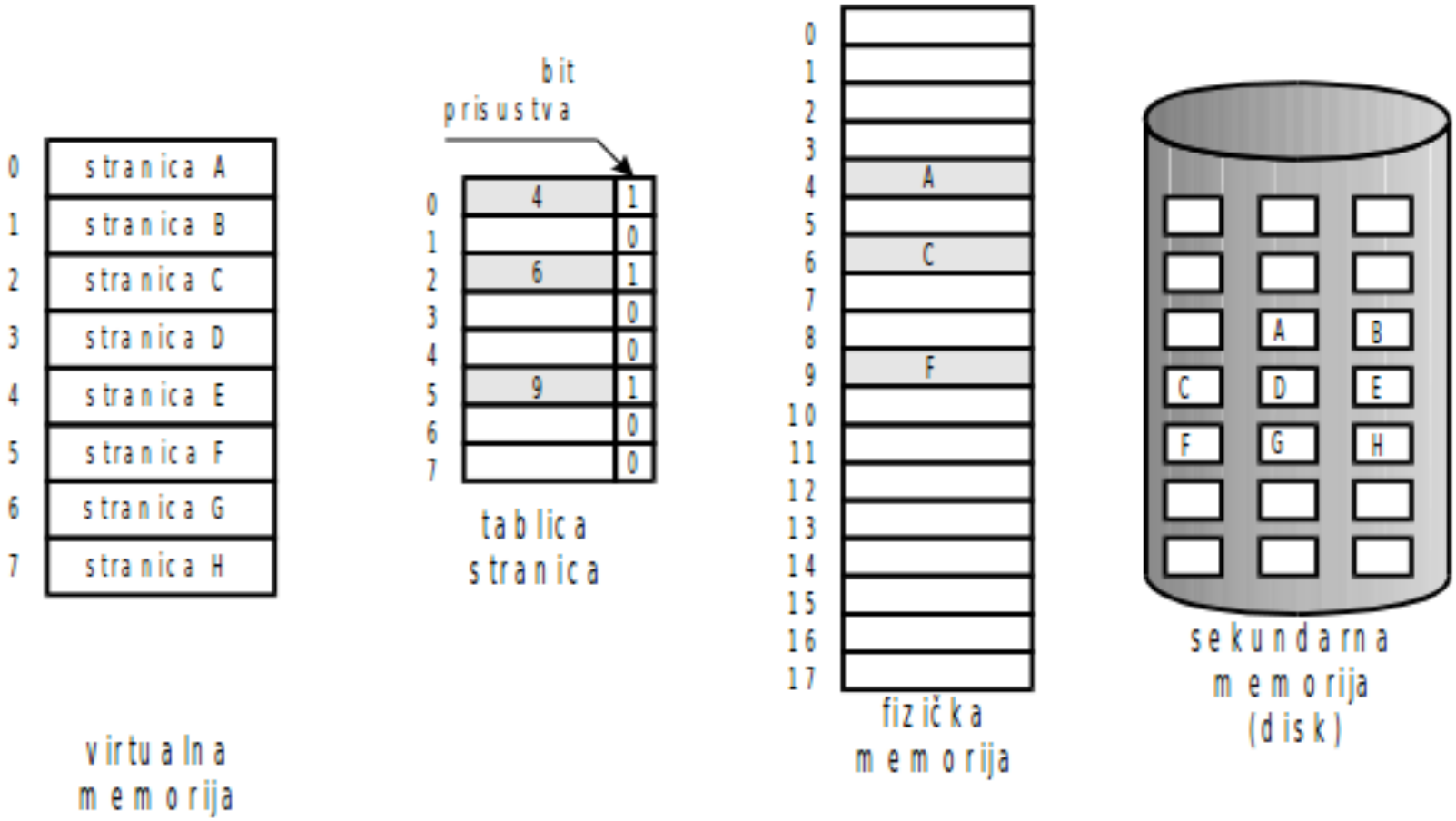
- Virtualna memorija obično se realizira kao straničenje na zahtjev (demand paging).
- Moguće ju je primijeniti u sustavima koji koriste podjelu memorije na segmente.
- Nekoliko sustava riješilo je virtualnu memoriju pomoću segmenata, gdje su segmenti podijeljeni na stranice.
- Tako korisnik vidi program podijeljen na segmente, a operacijski sustav dijeli segmente na stranice.
- Algoritmi zamjene segmenata su znatno složeniji od algoritama zamjene stranica budući su segmenti promjenjive veličine, a stranice fiksne.

Straničenje na zahtjev - demand paging

- Proces je pohranjen na sekundarnoj memoriji, obično disku.
- Kada se namjerava izvesti proces upisuje se samo jedan njegov dio u radnu memoriju.
- Kod upisivanja procesa u memoriju uobičajeno se koristi tzv. **lijeni prebacivač (lazy swapper)** koji upisuje stranicu u memoriju tek kada je ona potrebna.
- Termin swapper može se smatrati netočnim jer se on obično odnosi na prebacivanje cijelog procesa. Ispravnije je korištenje termina **pager** koji se odnosi na prebacivanje stranice.
- Proces se sada može promatrati kao niz stranica koji se prema potrebi upisuju u memoriju.

Straničenje na zahtjev - demand paging

potrebno je razlučiti koje su stranice upisane u memoriju, a koje se samo nalaze na disku.
Ovaj problem rješava se proširenjem tablice stranica **bitom prisutnosti** koji daje informaciju da li se stranica nalazi u radnoj memoriji ili ne.

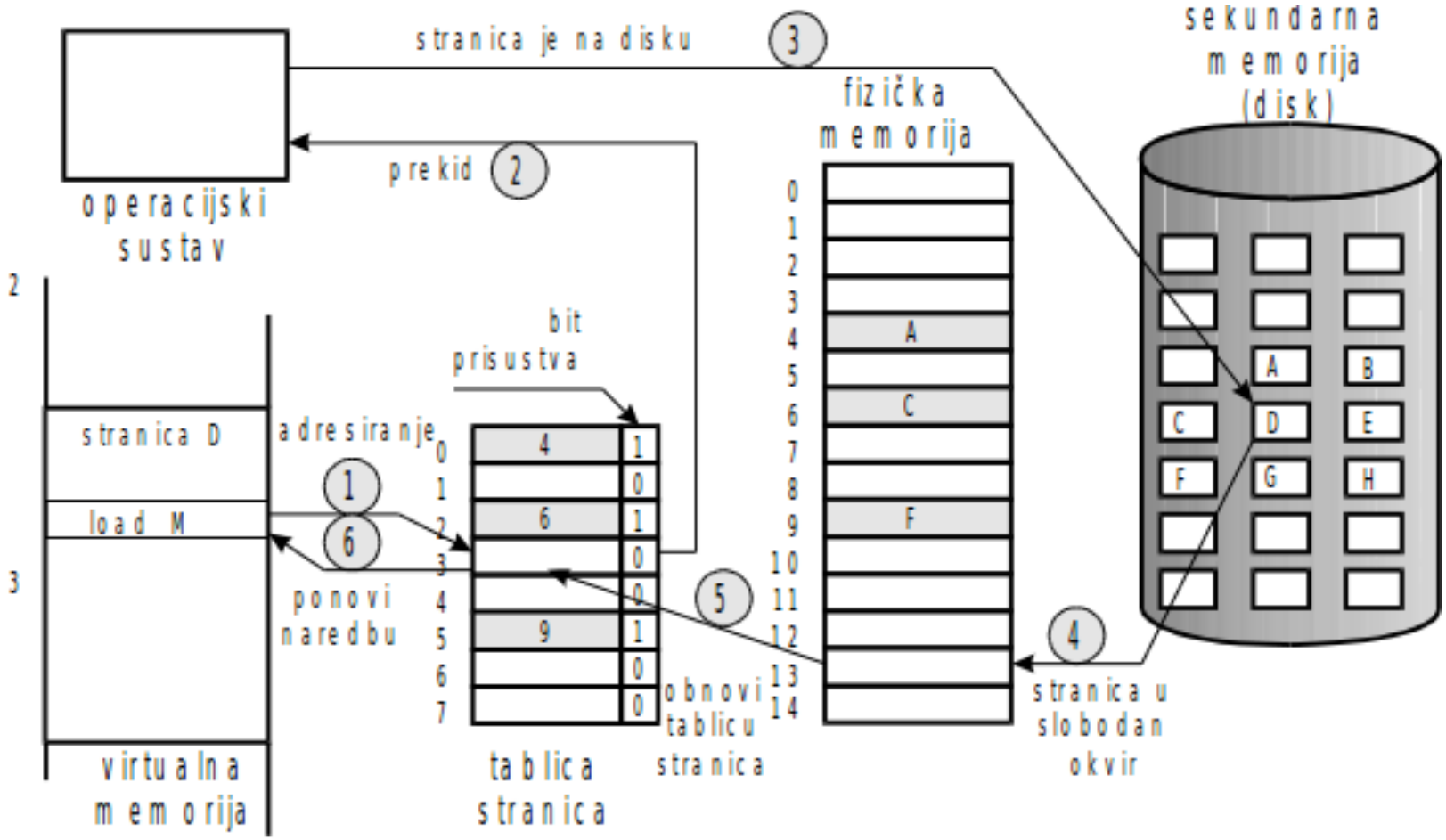


Straničenje na zahtjev - demand paging

Procedura pristupa stranici može se opisati na sljedeći način:

- Provjerava se bit prisustva adresirane stranice kako bi se odredilo da li je stranica u memoriji ili ne.
- Ukoliko stranica nije u memoriji (došlo je do tzv. **promašaja**) generira se prekid koji dojavljuje operacijskom sustavu da treba pronaći stranicu na sekundarnoj memoriji i prebaciti je u radnu memoriju. Obično promašaj rezultira prekidom prava korištenja procesora, te se proces prebacuje u red čekanja na U/I uređaj, u ovom slučaju disk.
- Operacijski sustav pronalazi slobodan okvir u radnoj memoriji (operacijski sustav vodi listu slobodnih okvira).
- Prebacuje se tražena stranica u odabrani okvir.
- Osvježava se tablica stranica procesa na način da se stranici pridružuje dodijeljeni okvir. Ovim je praktički proces pripravan da nastavi s izvođenjem.
- Prekinuta naredba se ponovo izvodi a stranici se pristupa kao da je ona oduvijek bila u memoriji.

Straničenje na zahtjev - demand paging



Problem zamjene stranica

- U slučaju promašaja operacijski sustav u memoriji nema uvijek slobodan okvir u koji će upisati traženu stranicu.
- Potrebno je osloboditi okvir za novu stranicu izbacivanjem neke od stranica koje su već u memoriji.
- Stranice koju će se izbaciti zovemo žrtva (victim).

Procedura obrade promašaja:

1. pronalaženje tražene stranice na disku,
2. odabir okvira u koji će se stranica upisati:
   a)ako postoji slobodan okvir njega se koristi,
   b)ako nema slobodnih okvira poseban algoritam odabire žrtvu i prebacuje prema potrebi odabranu stranicu na disk, ažurira tablicu stranica i oslobađa okvir,
3. upisuje traženu stranicu u oslobođeni okvir i ažurira tablicu stranica,
4. ponavlja prekinutu naredbu.

Algoritmi zamjene stranica

Literatura

Željko Vrba - Upravljanje memorijom u modernim operacijskim sustavima
www.fesb.hr/ emudnic/Nastava/OS