



Operativni sistemi

Uvod i pregled operativnih sistema

Prof. dr Dragan Stojanović

Katedra za računarstvo
Univerzitet u Nišu, Elektronski fakultet



Literatura

- ◆ *Operating Systems: Internals and Design Principles*, edition, W. Stallings, Pearson Education Inc., 7th – 2012, (5th -2005, 6th - 2008, 8th – 2014 , 9th – 2017)
 - <http://williamstallings.com/OperatingSystems/>
 - <http://williamstallings.com/OperatingSystems/OS9e-Student/>
- ◆ Poglavlje 2: Pregled operativnog sistema



Operativni sistem (OS)

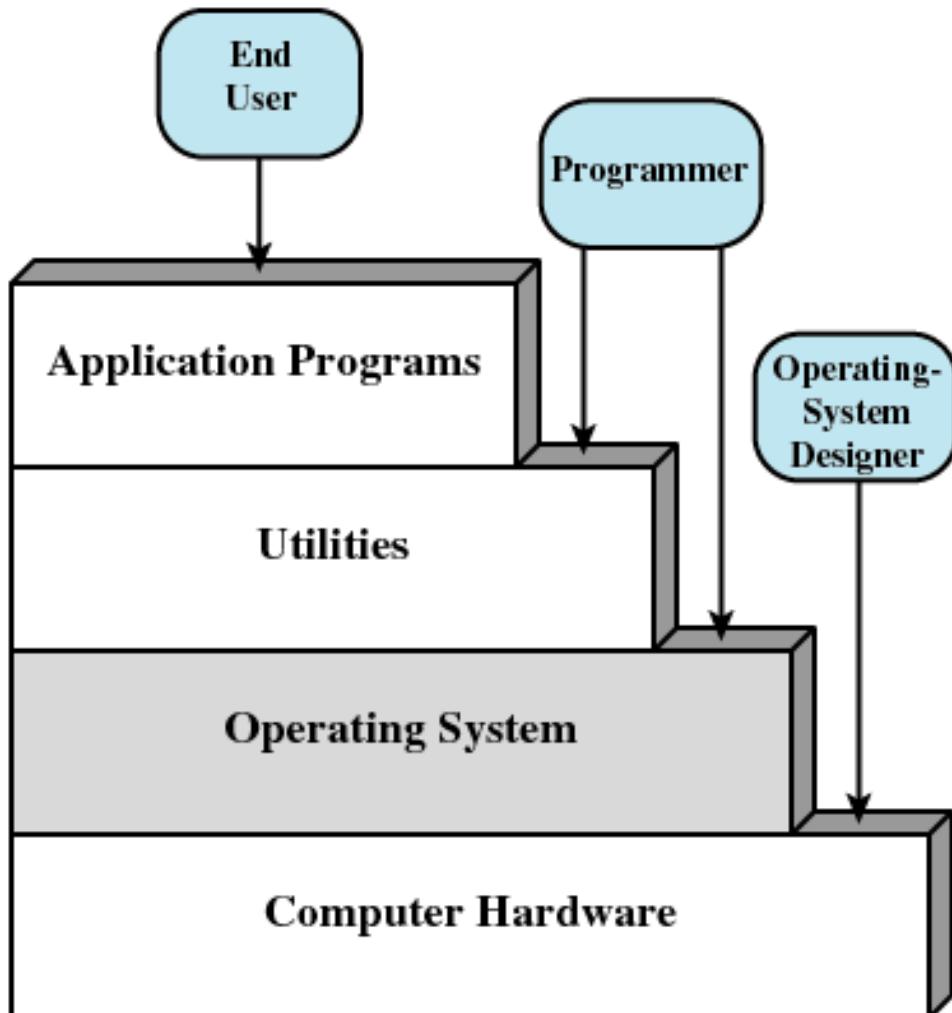
- ➊ Operativni sistem predstavlja organizovanu kolekciju programa koji upravlja izvršavanjem aplikativnih programa i služi kao interfejs između programa i hardvera računara
- ➋ Ciljevi i funkcije operativnog sistema:
 - ▣ **Pogodnost** - Da omogući lako i pogodno korišćenje računara
 - ▣ **Efikasnost** - Da obezbedi efikasno korišćenje i upravljanje resursima računara
 - ▣ **Mogućnost razvoja** - Da obezbedi osnovu za efikasan razvoj, testiranje i uvođenje novih funkcija sistema



OS kao interfejs između korisnika i računara

- ❖ Korisnik nije zainteresovan za detalje računarsog hardvera, već vidi računarski sistem kao skup aplikacija
- ❖ Korisnik interaguje sa OS-om pomoću komandi komandnog jezika (shell-a), ili preko grafičkog korisničkog interfejsa (GUI)
- ❖ Programer (*software developer*) pristupa računarskom sistemu i razvija aplikacije pomoću sistemskog softvera
 - Sistemski softver čine sistemski programi neophodni za razvoj i izvršenje aplikacija na računaru (uslužni programi) i upravljanje resursima računarskog sistema
 - Najvažniji sistemski softver je **operativni sistem**
 - OS sakriva detalje hardvera od programera i obezbeđuje mu jednostavan i prikladan interfejs za korišćenje računarskog sistema

OS kao interfejs između korisnika i računara

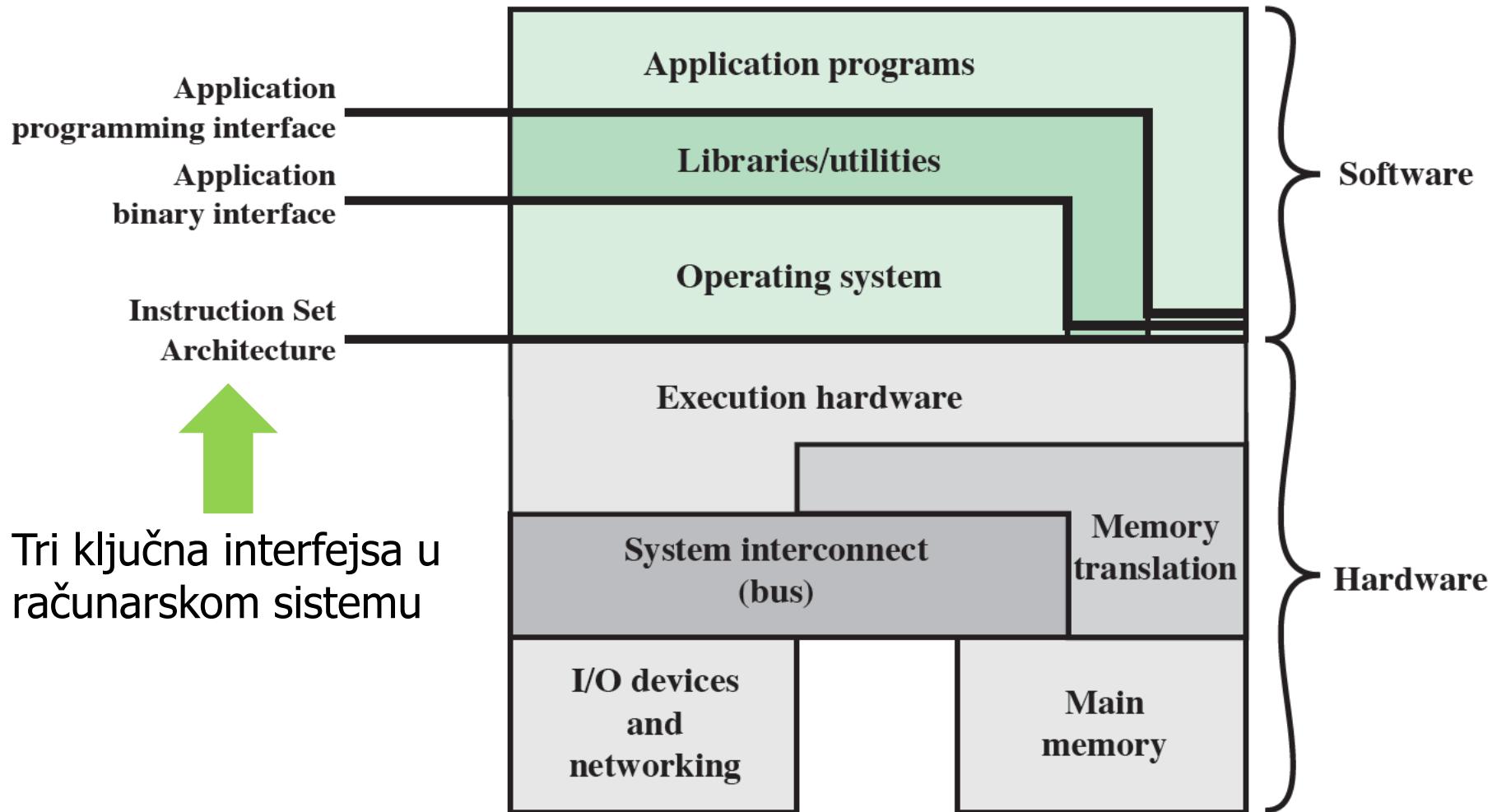




Servisi operativnog sistema

- ❖ Operativni sistem obezbeđuje servise u sledećim domenima:
 - ❖ Razvoj programa
 - ❖ Izvršavanje programa
 - ❖ Pristup U/I uređajima
 - ❖ Kontrolisan pristup datotekama
 - ❖ Pristup sistemu i upravljanje pristupom sistemskim resursima
 - ❖ Otkrivanje grešaka i odgovor na greške
 - ❖ Obračun korišćenja resursa sistema i nadgledanje performansi

Nivoi i pogledi računarskog sistema

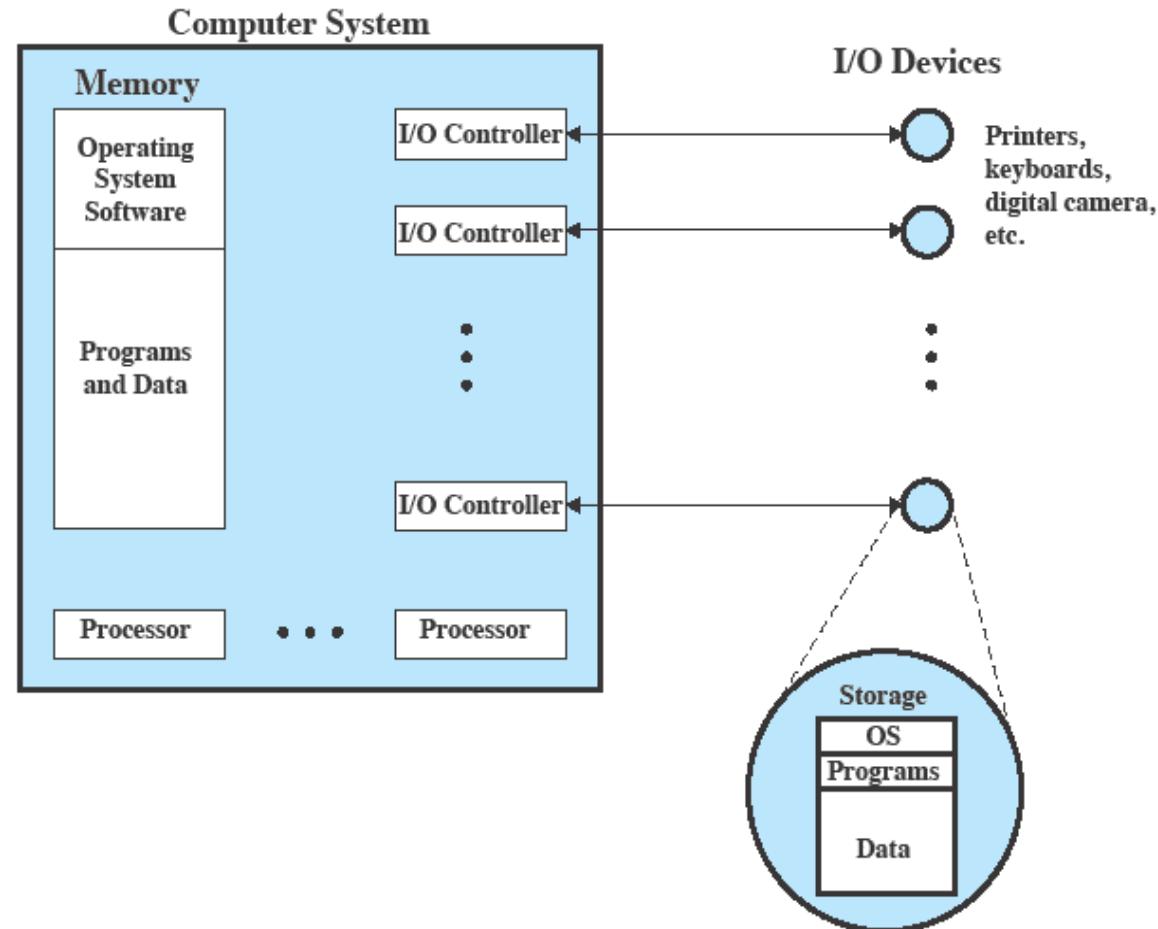


OS kao upravljač resursa

- ❖ Operativni sistem upravlja **resursima računarskog sistema**, a to su hardverski uređaji (procesor, memorija, štampač, disk, kamera,...) ili strukture podataka (datoteka, semafor, slog u bazi podataka, bafer poruka,...) koji su na raspolaganju korisnicima i programima.
- ❖ Operativni sistem je softver, skup programa koji se izvršavaju na procesoru
- ❖ OS se povremeno odriče izvršavanja na procesoru, i predaje procesor na izvršavanje korisničkog programa
- ❖ Kada se desi neki događaj u računarskom sistemu (prekid, trap), operativni sistem „preuzima procesor“ i izvršava se na procesoru da bi obavio odgovarajuće upravljačke funkcije nad resursima.

OS kao upravljač resursa

- ◆ Deo OS se nalazi u glavnoj memoriji
 - ◆ **Kernel** (jezgro) - sadrži najbitnije funkcije u okviru OS
 - ◆ Delovi OS koji se trenutno koriste
- ◆ Ostatak memorije sadrži korisničke programe i podatke
- ◆ OS upravlja dodelom procesora, memorije, U/I,... korisničkim programima





Lakoća evolucije OS

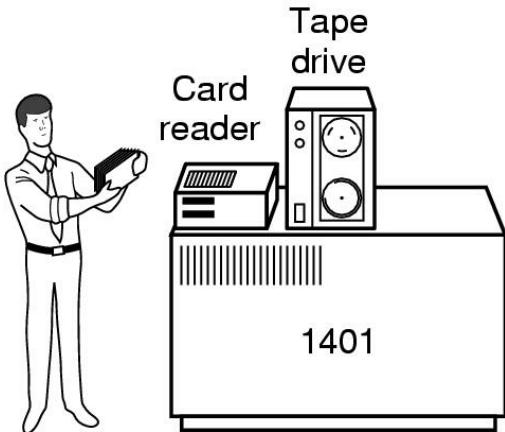
- ❖ Operativni sistemi moraju biti sposobni da evoluiraju tokom vremena iz sledećih razloga:
 - ❖ Nadogradnja hardvera i razvoj novih tipova hardvera
 - ❖ Novi servisi: kao odgovor na zahteve korisnika ili sistemskih administratora
 - ❖ Korekcija grešaka
- ❖ Operativni sistem mora biti modularne strukture sa jasno definisanim interfejsima između modula i dobro dokumentovan



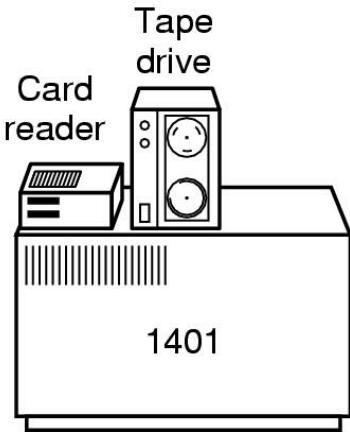
Razvoj operativnih sistema

- ❖ Serijska obrada (1945 – 1955)
 - ❖ Vakumske cevi, bušene kartice, mašinski jezik
 - ❖ **Nema OS-a**, programeri su pristupali direktno hardveru
- ❖ Jednostavni **sistemi paketne obrade (batch systems)** (1955 – 1965)
 - ❖ Tranzistori, mainframe računari, asemblerski jezik, FORTRAN, COBOL
 - ❖ **Monitor** – jednostavan OS (IBSYS – IBM OS za 7090/7094 računare)
- ❖ Multiprogramirani sistemi paketne obrade
- ❖ Sistemi sa deljenjem vremena (*time sharing*) (1965-1980)
 - ❖ Integrisana kola, mini računari i radne stanice, C, UNIX
 - ❖ **Multiprogramiranje, timesharing**
 - ❖ IBM System/360, *Compatible Time-Sharing System* (CTSS), UNIX, ...
- ❖ Personalni računari (1980 – danas)
 - ❖ LSI/VLSI, mikroprocesori, personalni računari (PC), mikroračunari
 - ❖ Windows, Apple Mac OS, UNIX, Linux, ...
- ❖ Distribuirani, paralelni, mobilni računari (1990 - danas)
 - ❖ Multiprocesorski sistemi, distribuirani sistemi, sistemi za rad u realnom vremenu, mobilni računari (pametni telefoni, tableti)
- ❖ *Cloud computing, Sveprisutno računarstvo, IoT (Internet of Things),...*

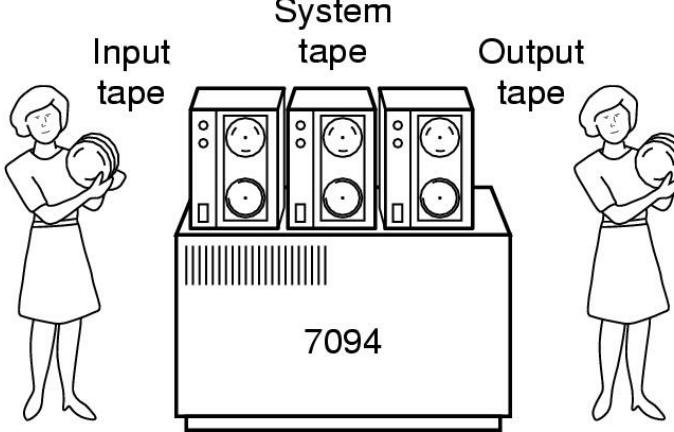
Sistemi paketne obrade (batch sistemi)



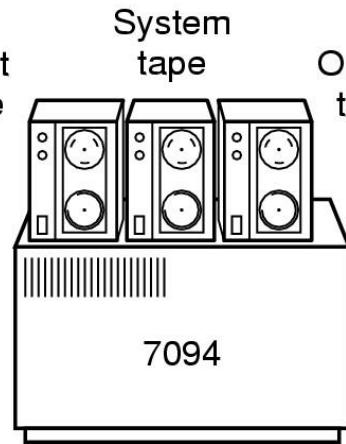
(a)



(b)



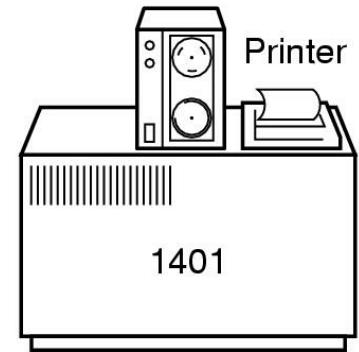
(c)



(d)



(e)



(f)

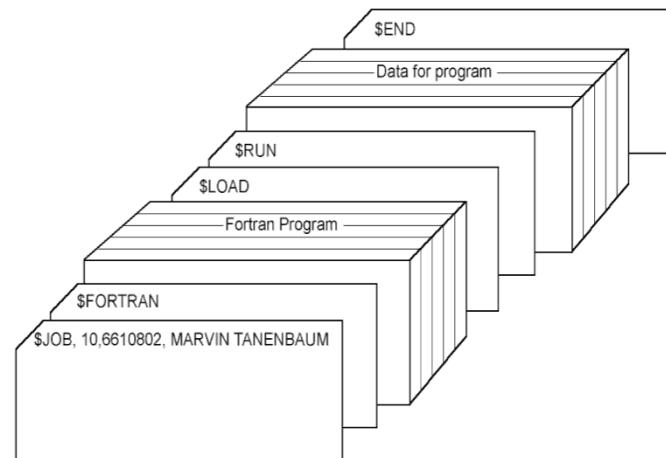
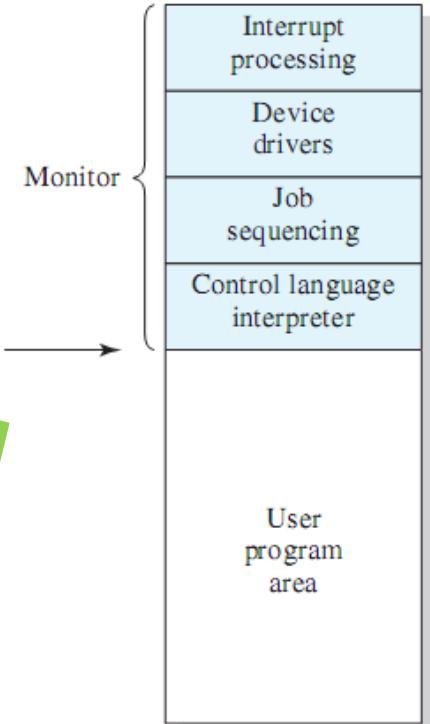
Stari mainframe računari

- Bušene kartice se unose u IBM 1401
- Čitanje sa kartica i snimanje na magnetnu traku
- Postavljanje trake na IBM 7094 koji obavlja obradu i rezultat snima na magnetnu traku (IBSYS operativni sistem)
- Postavljanje trake na IBM 1401 i štampanje

*Modern Operating Systems,
Tanenbaum, 2014*

Monitor u *batch* sistemu

- ❖ Struktura tipičnog posla (*job*) zadatog bušenim karticama
- ❖ Operativni sistem - **Monitor**
 - ❖ FMS (Fortran Monitor System)
 - ❖ IBSYS (IBM-ov OS za 7094 računar)
- ❖ **Monitor** je stalno smešten u glavnoj memoriji dostupan za izvršavanje (rezidentni monitor)
- ❖ Čita sa ulaznog uređaja jedan po jedan posao (job), smešta instrukcije i podatke u korisnički deo memorije i startuje izvršenje posla na procesoru.
- ❖ Po završetku, monitor učitava i izvršava sledeći posao (job).
- ❖ Instrukcije se monitoru zadaju preko *Job Control Language* (JCL)



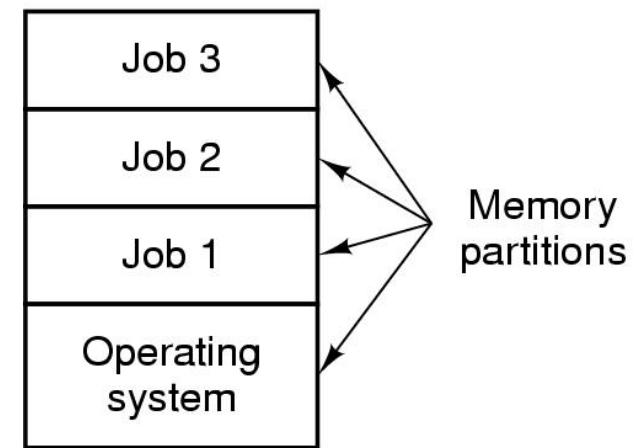
*Modern Operating Systems,
Tanenbaum, 2014*

Monitor – operativni sistem

- ❖ **Monitor** (OS sa paketnom obradom) je računarski program koji je smešten u deo glavne memorije i naizmenično se izvršava sa korisničkim programima
- ❖ Poželjna svojstva hardvera:
 - ❖ Zaštita memorije koju zauzima monitor
 - ❖ Tajmer
 - ❖ Privilegovane mašinske instrukcije – može ih izvršiti samo monitor
 - ❖ Prekidi
- ❖ Dva režima rada
 - ❖ **Kernel režim** (mod)
 - Monitor (operativni sistem) se izvršava u kernel modu
 - ❖ **Korisnički režim** (mod)
 - Korisnički programi se izvršavaju u korisničkom modu, koriste samo podskup iz skupa instrukcija i samo neke mogućnosti HW (generalno, instrukcije za U/I i zaštitu memorije su zabranjene u korisničkom modu)
 - Za ostalo korisnički programi pozivaju funkcije (servise) OS-a

Multiprogramske sisteme paketne obrade

- ❖ **Multiprogramiranje (multitasking)** – Operativni sistem istovremeno smešta u memoriju više poslova; u jednom trenutku samo jedan od poslova se izvršava na CPU, ukoliko se blokira izvršenjem U/I operacije (npr. čitanje podataka sa diska), aktivira se planiranje poslova
- ❖ **Planiranje poslova (Job scheduling)** – OS mora da iz skupa svih poslova izabere one koji će biti smešteni u memoriju i odrediti jedan koji će se izvršavati - planiranje CPU (CPU scheduling)
- ❖ **Dodatna svojstva hardvera**
 - ❖ U/I prekidi i DMA
 - ❖ Upravljanje memorijom
- ❖ **Operativni sistemi:**
 - ❖ OS/360
 - ❖ MULTICS
 - ❖ UNIX (System V, BSD)



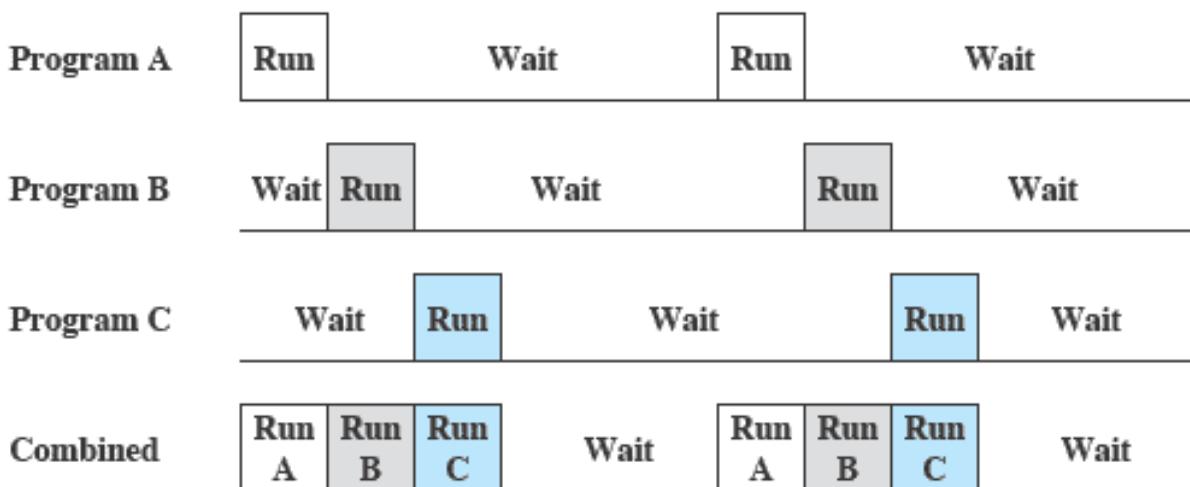
Multiprogramiranje

- ➊ Jednoprogramiranje - CPU mora da čeka dok se ne završi U/I instrukcija



Read one record from file	$15 \mu\text{s}$
Execute 100 instructions	$1 \mu\text{s}$
Write one record to file	$\underline{15 \mu\text{s}}$
TOTAL	$31 \mu\text{s}$
Percent CPU Utilization	$= \frac{1}{31} = 0.032 = 3.2\%$

- ➋ Multiprogramiranje sa tri aktivirana programa



Primer multiprogramiranja

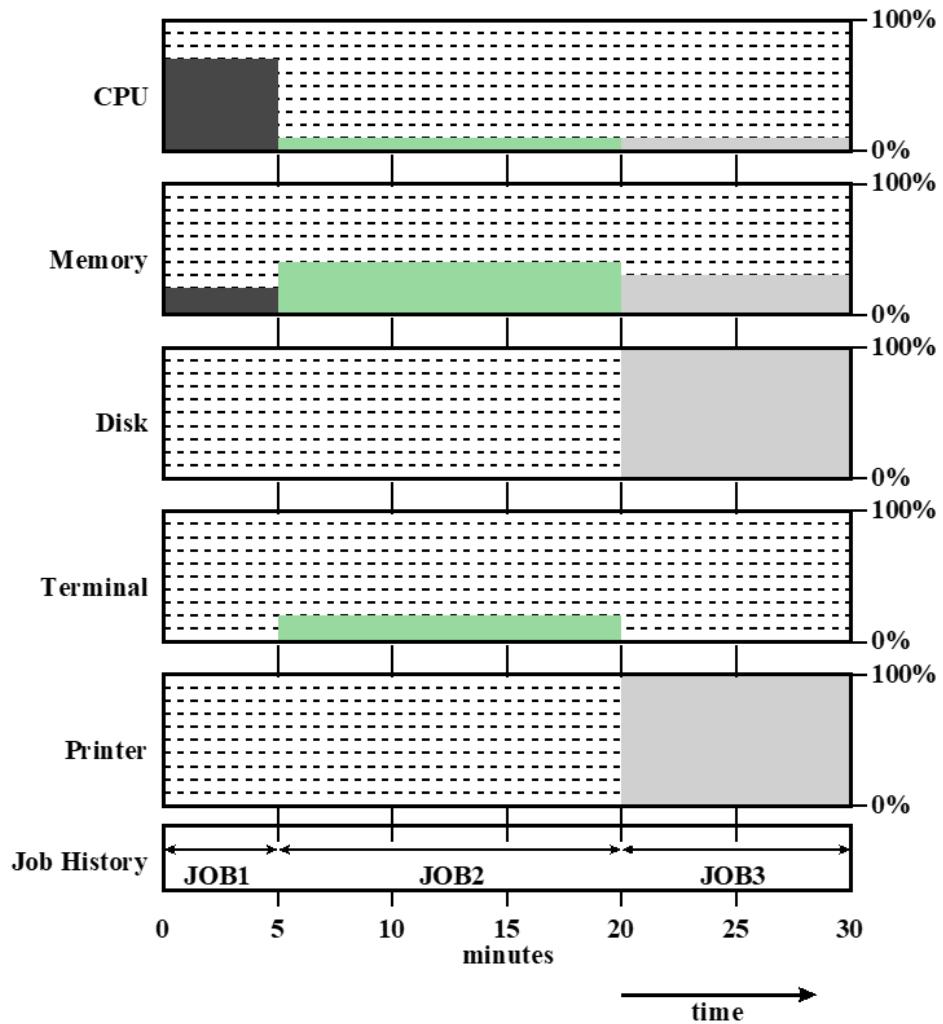
Primeri atributa za izvršavanje programa

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

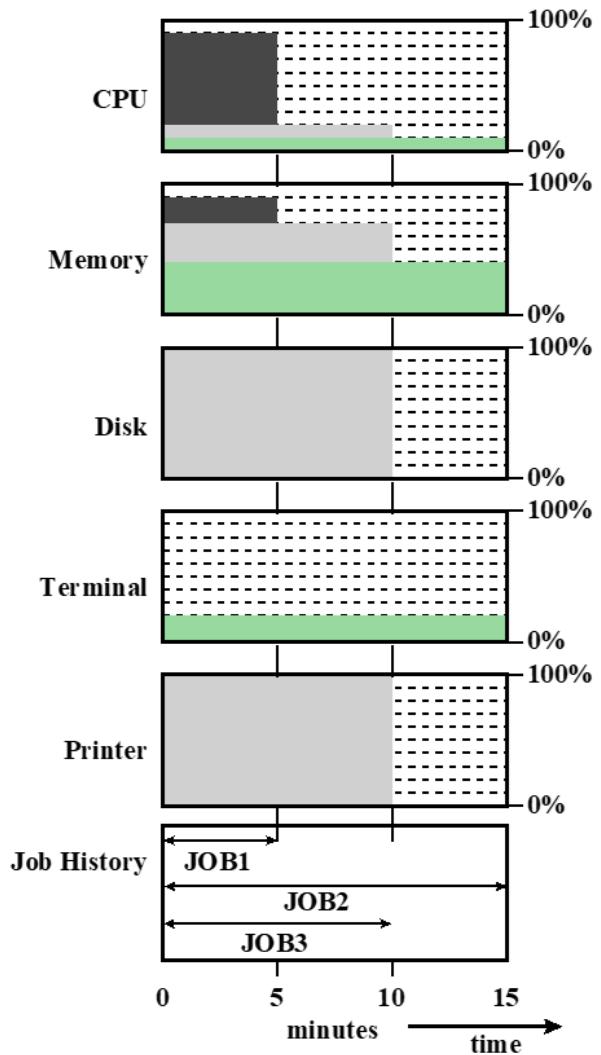
Efekti multiprogramiranja na iskorišćenje resursa

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

Histogram iskorišćenosti resursa



(a) Uniprogramming



(b) Multiprogramming

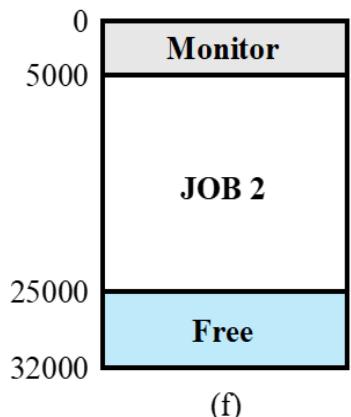
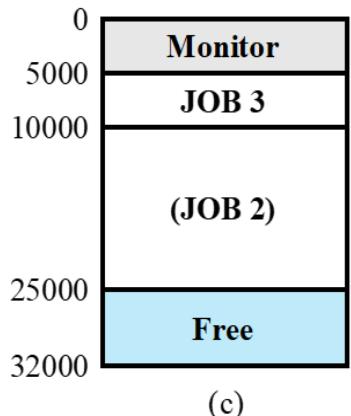
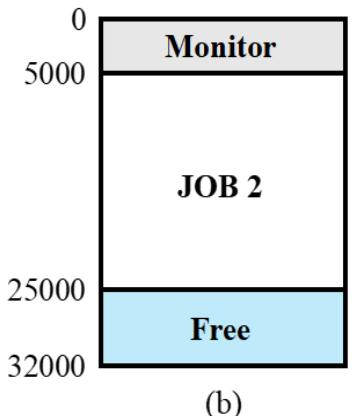


Sistemi sa deljenjem vremena

- ❖ **Time-sharing** – procesorsko vreme je podeljeno između više korisnika.
- ❖ Koristi multiprogramiranje za višekorisnički rad pri čemu svaki korisnik pristupa sistemu interaktivno putem terminala
- ❖ Svakom korisničkom programu se dodeljuje po jedan mali vremenski period (deo procesorskog vremena) za izvršavanje, pre nego što se pređe na drugi program.
- ❖ Jedan od prvih time-sharing OS je CTSS (*Compatible Time-Sharing System*) razvijen 1961. na MIT za IBM 709, a kasnije prenet na IBM 7094
 - ❖ Računar sa glavnim memorijom od 32000 36-bitnih reči, pri čemu monitor koristi 5000 reči

Primer rada CTSS

- JOB1: 15000
- JOB2: 20000
- JOB3: 5000
- JOB4: 10000



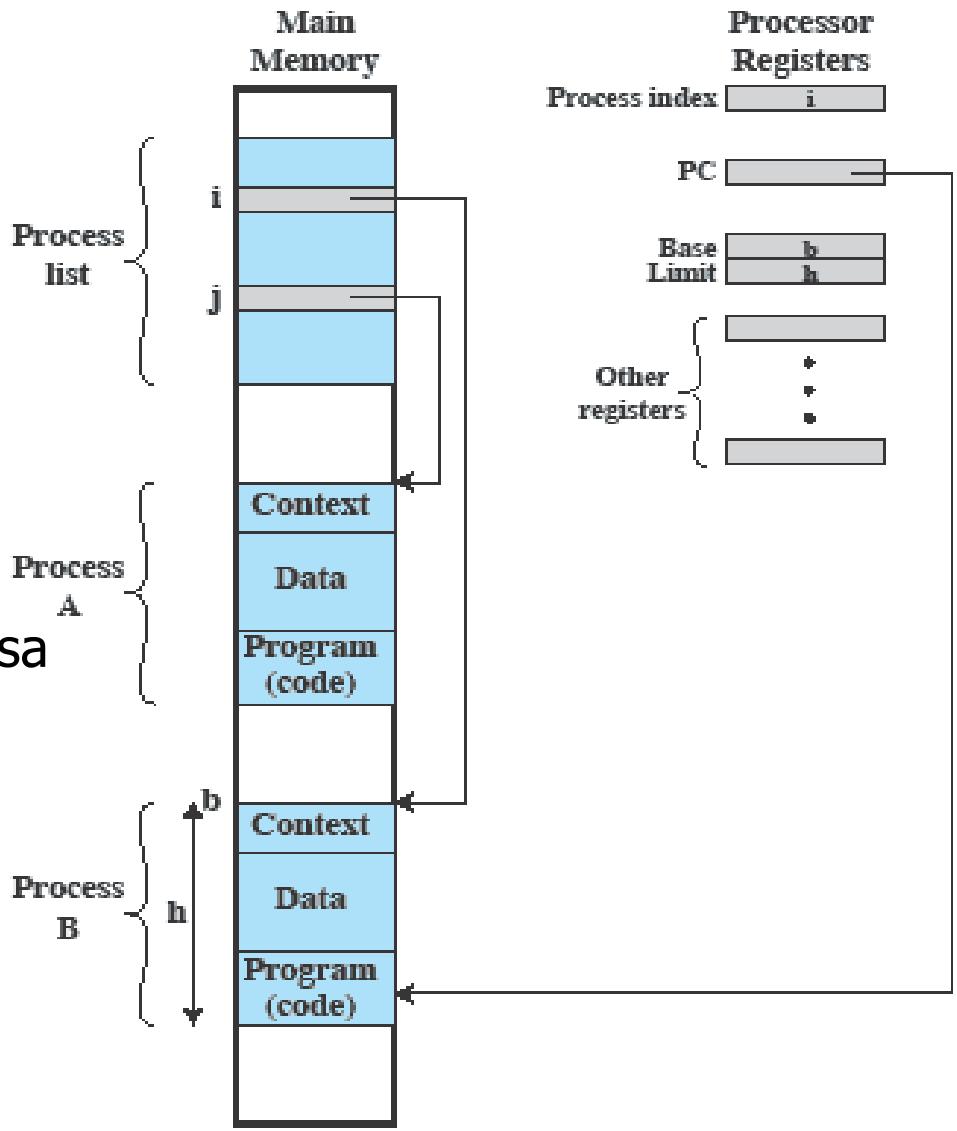


Glavna dostignuća u razvoju OS

- Procesi
- Upravljanje memorijom
- Zaštita i sigurnost informacija
- Planiranje i upravljanje resursima
- Struktura sistema

Procesi

- Proces je program u izvršenju
- Proces se sastoji od tri komponente
 - Izvršni program
 - Podaci koji se obrađuju u programu
 - **Kontekst izvršenja** procesa
- Tipična implementacija procesa prikazana na slici



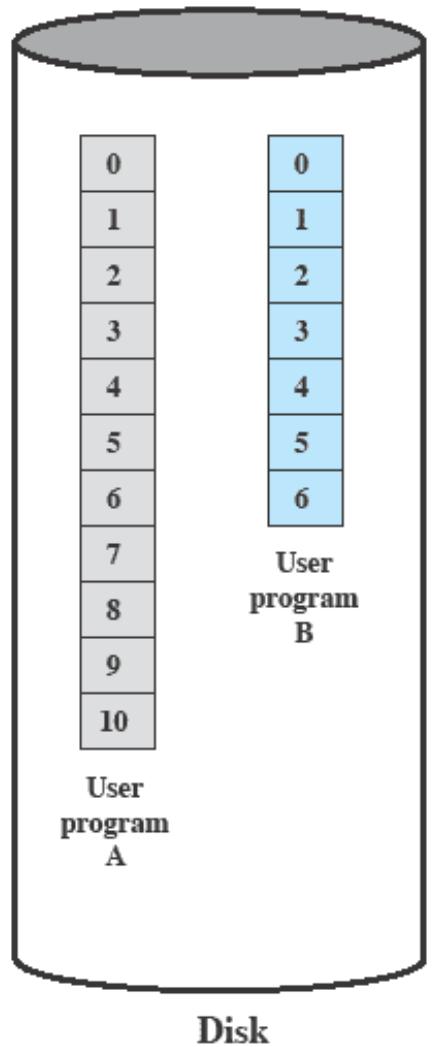
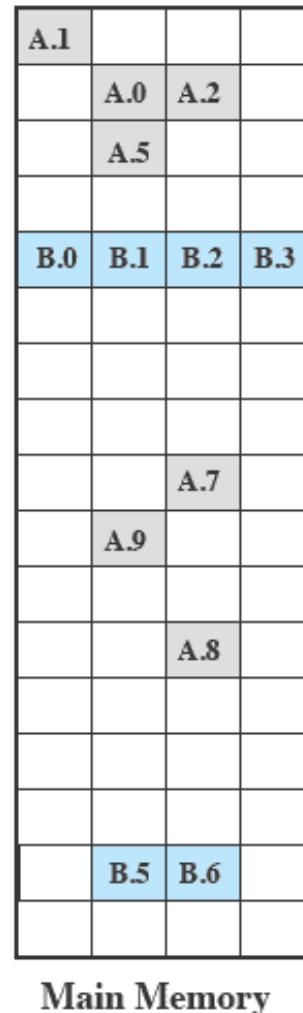
Upravljanje memorijom

Osnovne odgovornosti OS

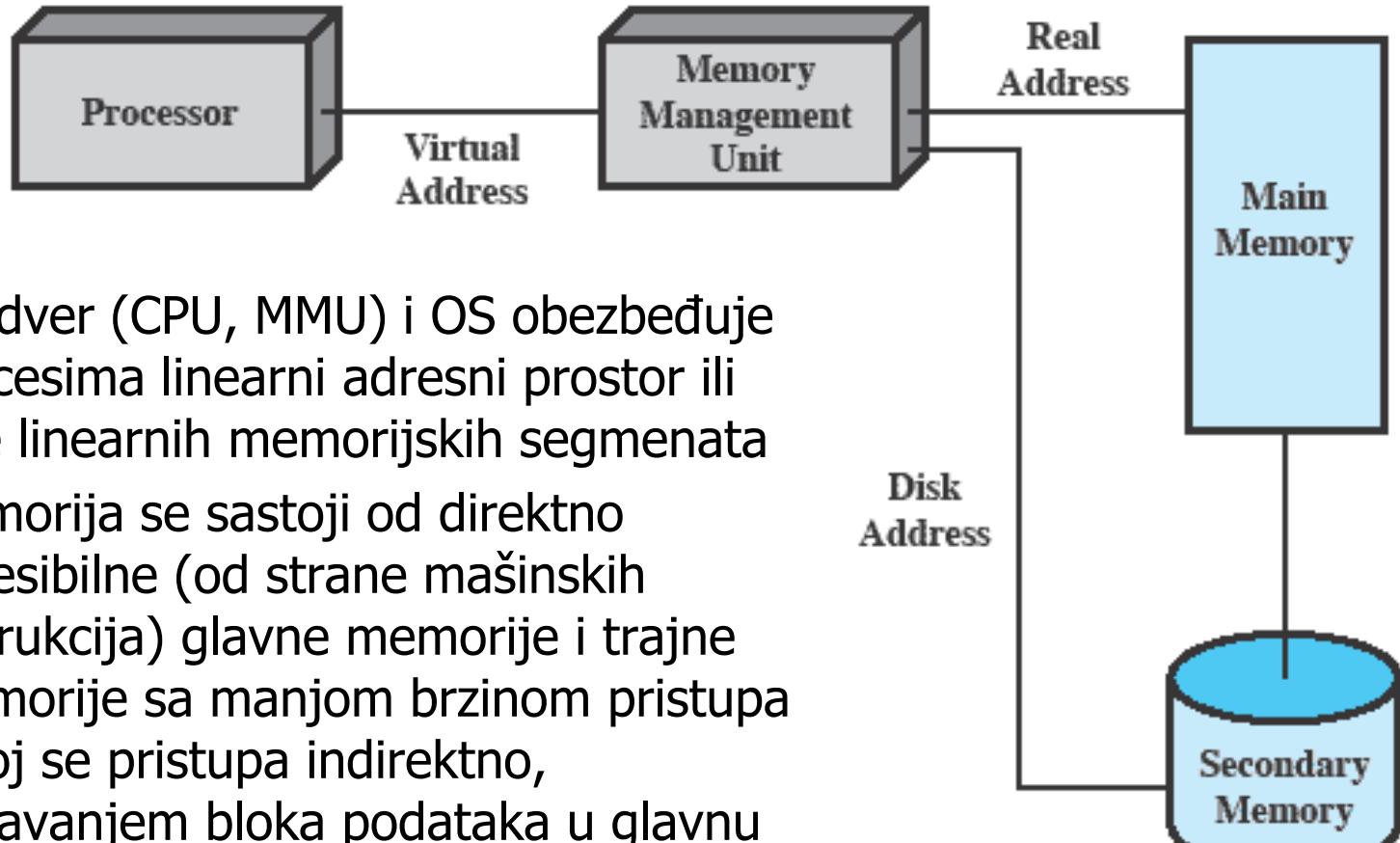
- Izolacija procesa
 - Automatsko dodeljivanje i upravljanje
 - Podrška za modularno programiranje
 - Zaštita i kontrola pristupa
 - Korišćenje dugotrajne memorije

- ➊ Koncept virtuelne memorije (straničenje - *paging*) i *file* sistema

- Virtuelna adresa
 - Realna (fizička) adresa u glavnoj memoriji



Adresiranje virtuelne memorije



- ❖ Hardver (CPU, MMU) i OS obezbeđuje procesima linearni adresni prostor ili više linearnih memorijskih segmenata
- ❖ Memorija se sastoji od direktno adresibilne (od strane mašinskih instrukcija) glavne memorije i trajne memorije sa manjom brzinom pristupa kojoj se pristupa indirektno, učitavanjem bloka podataka u glavnu memoriju



Zaštita informacija i bezbednost

❖ Raspoloživost

- ❖ Zaštita sistema od prekida funkcionisanja

❖ Poverljivost (tajnost)

- ❖ Zaštita podataka od neovlašćenog pristupa

❖ Integritet podataka

- ❖ Zaštita podataka od neautorizovane modifikacije

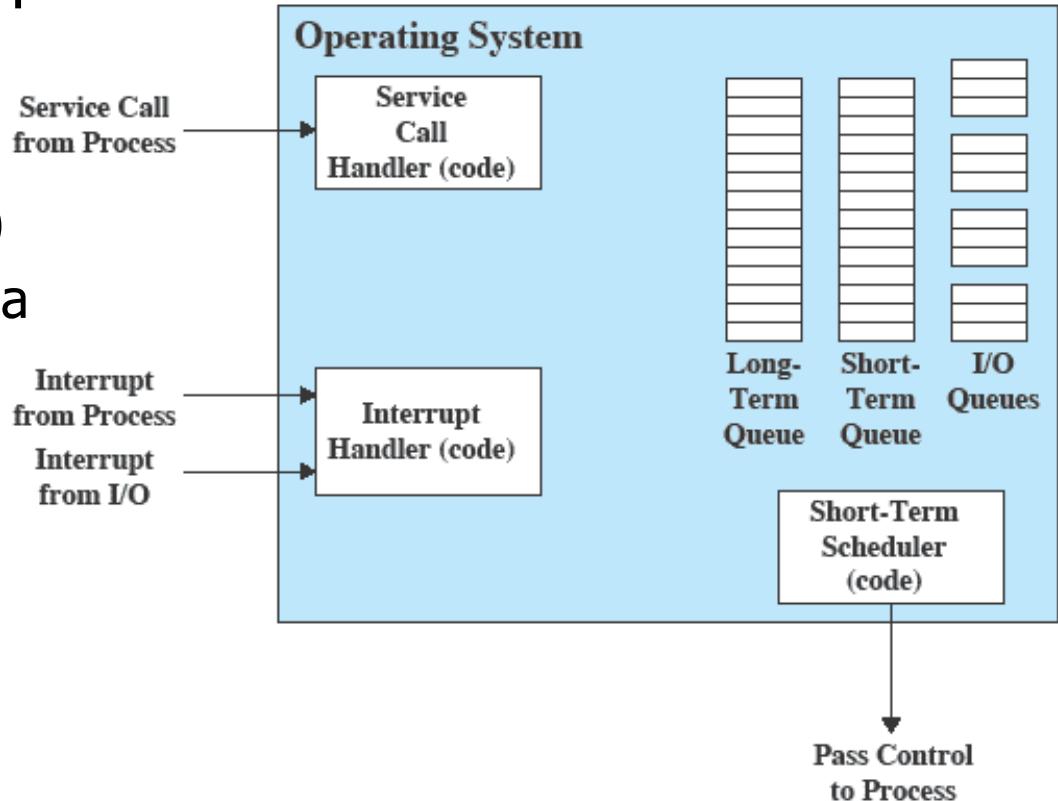
❖ Autentičnost

- ❖ Pogodna verifikacija identiteta korisnika i validnosti poruka i podataka

Raspoređivanje resursa i upravljanje

- Strategija raspoređivanja i dodelje resursa mora da uključi 3 faktora:

- Nepristrasnost (pravičnost)
- Različitost odgovora - Treba napraviti razliku između različitih klasa procesa
- Efikasnost
 - Maksimizovanje propusne moći (*throughput*),
 - Minimizovanje vremena odziva (*response time*) i
 - Opsluživanje što više korisnika (*time-sharing*)



Ključni elementi OS za multiprogramiranje

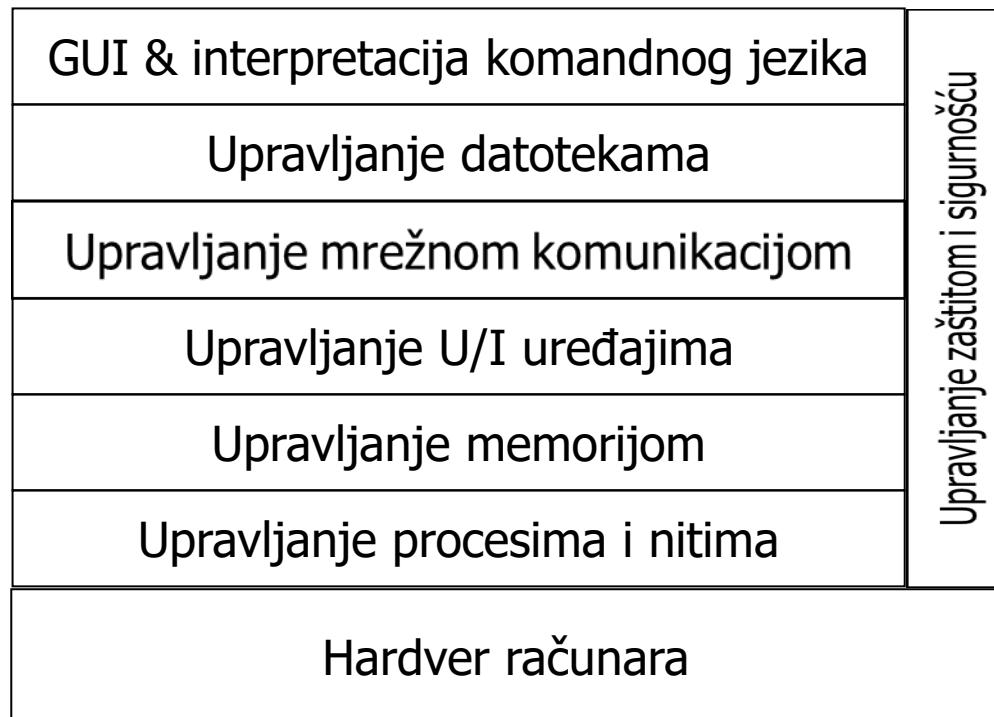


Struktura sistema

- ❖ Operativni sistem treba da bude modularne strukture sa jasno definisanim intrerfejsima između modula
- ❖ Struktura operativnog sistema kao skup hijerarhijskih slojeva (nivoa)
 - ❖ Svaki sloj izvršava odgovarajući podskup funkcija
 - ❖ Svaki sloj izvršenje svojih funkcija zasniva na sledećem nižem sloju u strukturi i njegovom izvršenju primitivnijih funkcija
 - ❖ Ovim se problem rastavlja na određeni broj lakših pod-problema



Struktura operativnog sistema





Hijerarhija dizajna OS

Level	Name	Objects	Example Operations
13	Shell	User programming environment	Statements in shell language
12	User processes	User processes	Quit, kill, suspend, resume
11	Directories	Directories	Create, destroy, attach, detach, search, list
10	Devices	External devices, such as printers, displays, and keyboards	Open, close, read, write
9	File system	Files	Create, destroy, open, close, read, write
8	Communications	Pipes	Create, destroy, open, close, read, write
7	Virtual memory	Segments, pages	Read, write, fetch
6	Local secondary store	Blocks of data, device channels	Read, write, allocate, free
5	Primitive processes	Primitive processes, semaphores, ready list	Suspend, resume, wait, signal
4	Interrupts	Interrupt-handling programs	Invoke, mask, unmask, retry
3	Procedures	Procedures, call stack, display	Mark stack, call, return
2	Instruction set	Evaluation stack, microprogram interpreter, scalar and array data	Load, store, add, subtract, branch
1	Electronic circuits	Registers, gates, buses, etc.	Clear, transfer, activate, complement

HW



Hardverski slojevi

● Sloj 1

- Elektronska kola
- Objekti su registri, memorijske ćelije i logička kola
- Operacije su brisanje sadržaja registra ili čitanje iz memorijske lokacije

● Sloj 2

- Instrukcioni set procesora
- Operacije poput: add, subtract, load, store,...

● Sloj 3

- Dodaje koncept procedure ili potprograma, kao i call/return operacije

● Sloj 4

- Prekidi (*Interrupts*)



Koncepti vezani za multiprogramiranje

• Sloj 5

- Proces kao program u izvršavanju
- Suspendovanje i nastavljanje procesa

• Sloj 6

- Sekundarni memorijski uređaji
- Transfer blokova podataka

• Sloj 7

- Kreiranje logičkog adresnog prostora za procese
- Organizovanje virtuelnog adresnog prostora u blokove



Upravljanje eksternim objektima

• Sloj 8

- Komunikacija informacija i porukama između procesa

• Sloj 9

- Podrška za trajno memorisanje imenovanih datoteka

• Sloj 10

- Obezbeđuje pristup eksternim uređajima korišćenjem standardizovanih interfejsa

• Sloj 11

- Održava asocijaciju između eksternih i internih identifikatora u okviru direktorijuma

• Sloj 12

- Obezbeđuje potpunu funkcionalnost za podršku procesima

• Sloj 13

- Obezbeđuje interfejs korisnika prema operativnom sistemu

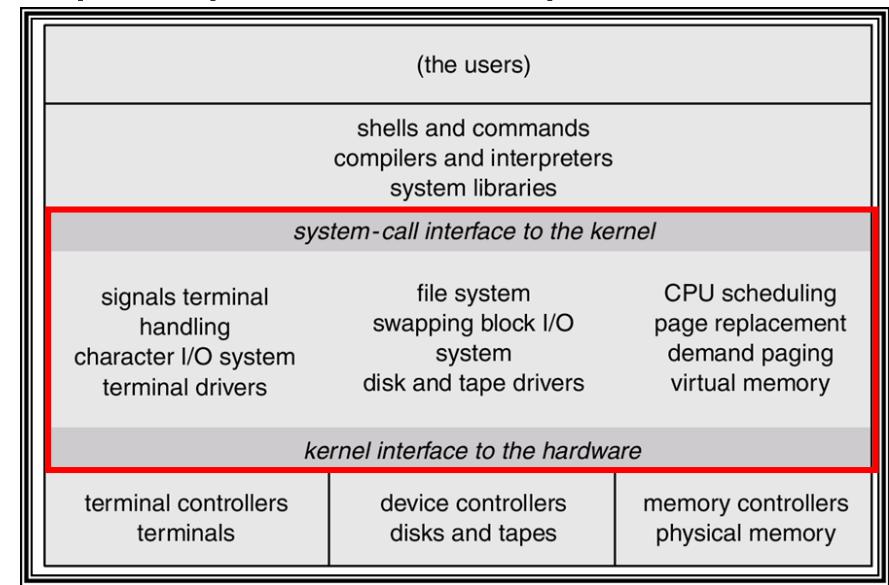


Pravci razvoja savremenih OS

- Mikrokernel arhitektura
- Višenitna obrada (*Multithreading*)
- Simetrično multiprocesiranje (*symmetric multiprocessing*- SMP) – na *multicore* arhitekturama
- Distribuirani operativni sistemi
- Objektno-orientisani dizajn i implementacija OS

Monolitna arhitektura OS

- ❖ Operativni sistem je kolekcija procedura. Pri čemu svaka može pozivati svaku poznajući njen interfejs (skup parametara i rezultat) i svaka procedura može pristupati deljivim podacima i strukturama podataka OS
- ❖ Prednosti:
 - Performanse i visok nivo zaštite od pristupa korisničkih procesa
- ❖ Nedostaci:
 - Loša proširljivost, održavanje
 - Loša zaštita između komponenti kernela
- ❖ UNIX - OS sadrži dva dela
 - Sistemski programi i kernel

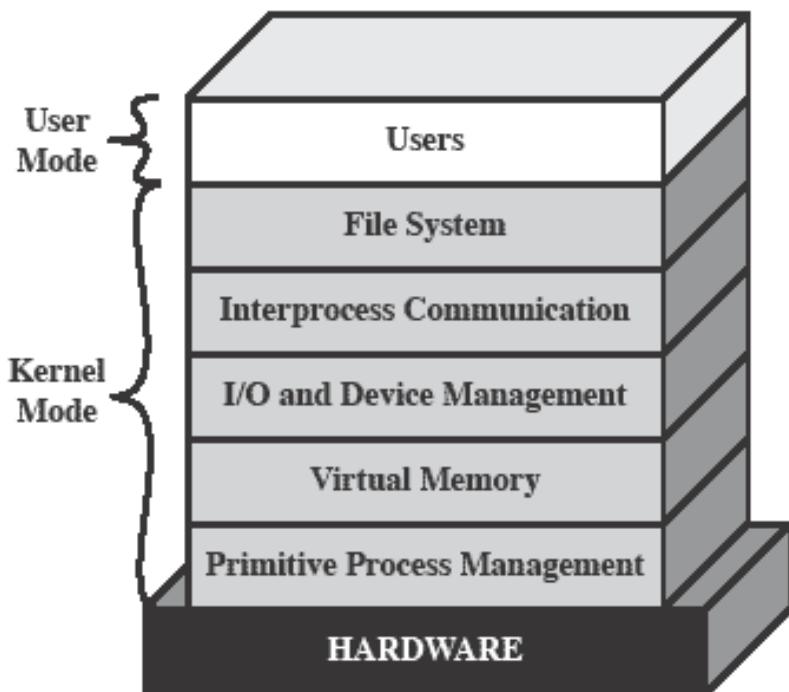


Mikrokernel arhitektura OS

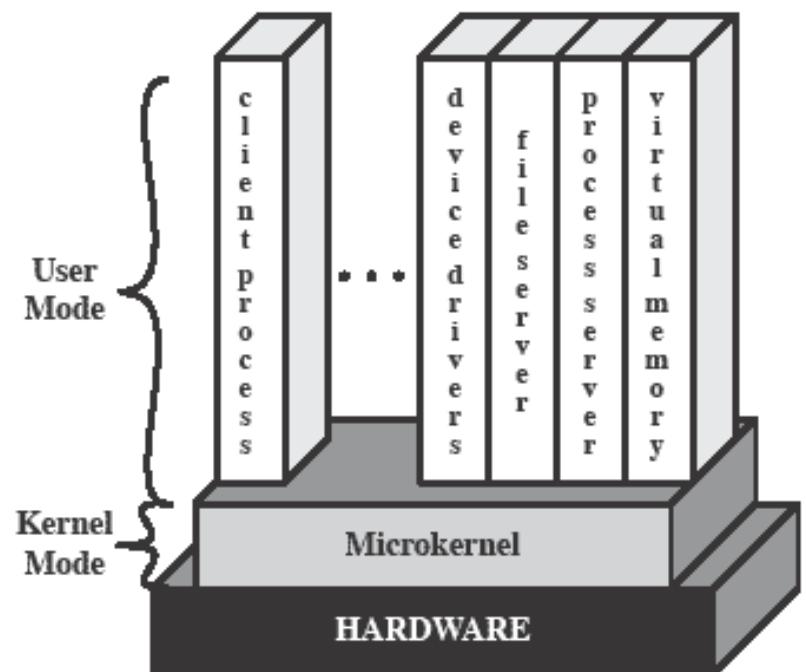
- ❖ Samo kritični OS procesi se izvršavaju u režimu kernela, npr. pristup U/I uređajima i U/I drajveri
- ❖ Ostale funkcije OS implementirane su kao servisi koji se izvršavaju u korisničkom režimu
- ❖ Prednosti
 - Jednostavno proširenje OS jer dodavanje novih servera ne zahteva modifikaciju kernela
 - Jednostavno portovanje OS sa jedne na drugu hardversku platformu
 - Pošto se svi serverski procesi izvršavaju u korisničkom modu, greška u nekom od njih ne uzrokuje pad OS
- ❖ Primeri:
 - Mach (*Carnegie Mellon University*, sredina 1980-ih)
 - Tru64UNIX (ranije Digital UNIX)
 - Apple MacOS X (Mach kernel + deo BSD kernela)
 - QNX, MINIX

Arhitektura kernela (poglavlje 4.3)

● Slojevita i mikrokernel arhitektura



(a) Layered kernel



(b) Microkernel



Višenitna obrada (*Multithreading*)

- ➊ Proces je podeljen u **niti** koje mogu da se izvršavaju konkurentno (paralelno)
 - **Nit (thread)**
 - Jedinica izvršenja koja se može planirati i rasporediti za izvršenje
 - Izvršava se sekvencijalno i može biti prekinuta i ponovo nastavljena
 - **Proces** je skup jedne ili više niti i pridruženih sistemskih resursa, poput memorije koja sadrži kod i podatke, otvorenih datoteka, i U/I uređaja
- ➋ **Višenitnost** je korisna u aplikacijama koje obavljaju više suštinski nezavisnih zadataka koji ne moraju serijski da se izvršavaju
 - Primer: Web server koji prihvata i opslužuje zahteve klijenata

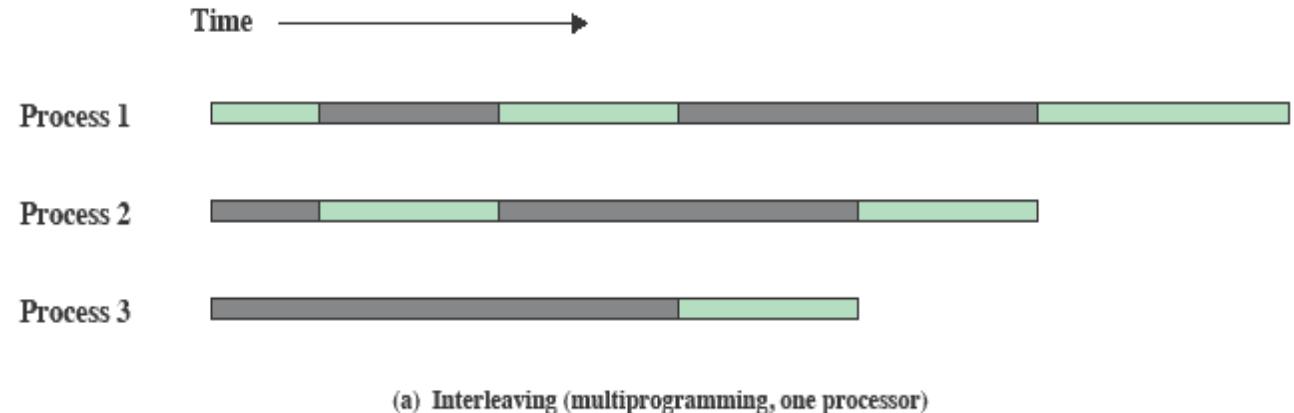


Simetrično multiprocesiranje (SMP)

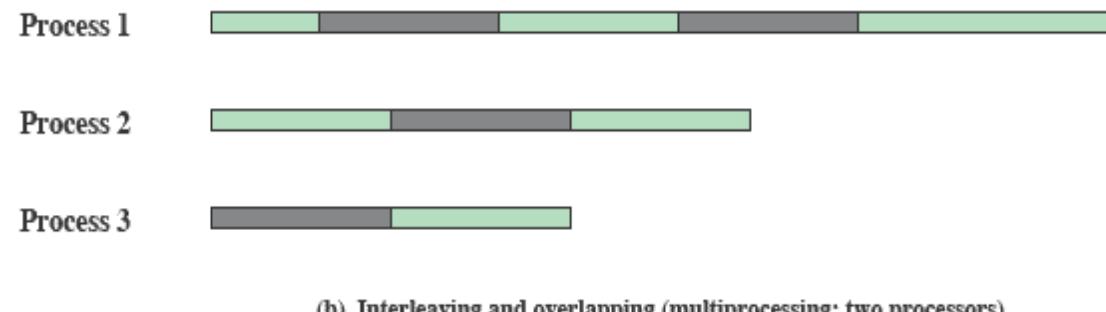
- ❖ Postoji više procesora u sistemu
- ❖ Ovi procesori dele istu glavnu memoriju i U/I resurse
- ❖ Svi procesori mogu izvršavati iste funkcije
- ❖ Prednosti:
 - ❖ Performanse
 - ❖ Raspoloživost u slučaju otkaza jednog procesora
 - ❖ Inkrementalno povećanje performansi dodavanjem dodatnih procesora
 - ❖ Skaliranje – može postojati više računarskih konfiguracija sa različitim brojem procesora sa različitom cenom i performansama

Simetrično multiprocesiranje (SMP)

- Multiprogramiranje (jedan procesor)



- Multiprocesiranje (dva procesora)



■ Blocked ■ Running

Uvod i pregled operativnih sistema
Operativni sistemi

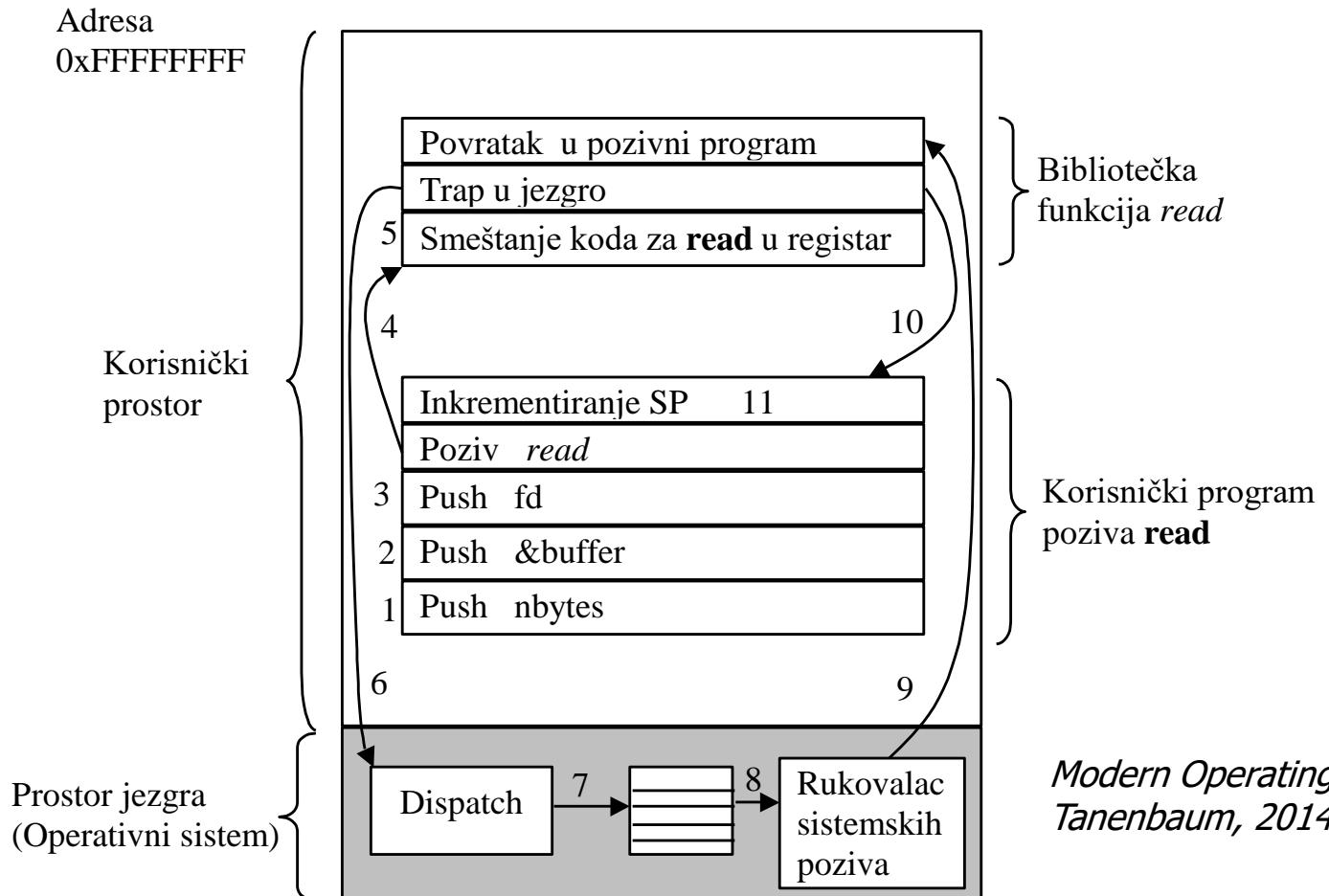


Sistemski pozivi

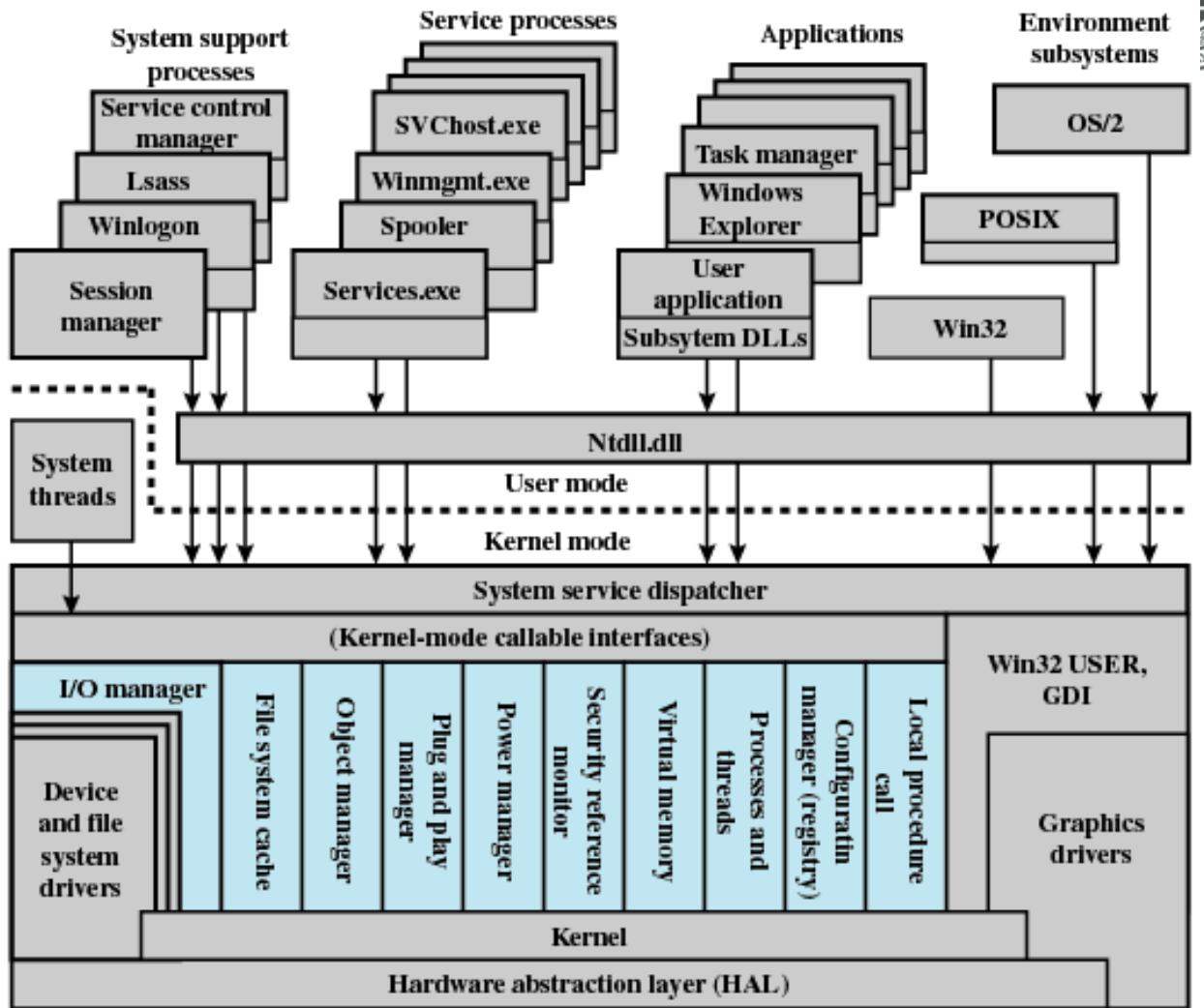
- ❖ **Sistemski pozivi** (*system calls*) obezbeđuju interfejs između aplikativnih/sistemskih programa i operativnog sistema
- ❖ Omogućuju pristup funkcijama operativnog sistema od strane korisničkih programa
 - ❖ Unix/Linux – POSIX 1003.1 standard
 - ❖ Windows - Windows API (*Application Programming Interface*)
- ❖ Sistemski poziv se obavlja u okviru korisničkog programa pozivom funkcije iz standardne biblioteke za odgovarajući programski jezik (API).
- ❖ U okviru ove funkcije se argumenti smeštaju na **stek**, i poziva **trap** instrukcija čiji je argument kôd sistemskog poziva.
- ❖ **Trap** instrukcija izaziva softverski prekid, OS čuva stanje prekinutog procesa, prelazi u mod kernela i poziva funkciju kernela (rutinu, *system call handler*) koja implementira sistemski poziv
- ❖ Postoje sistemski pozivi za upravljanje procesima, memorijom, datotekama, U/I uređajima, mrežnom komunikacijom, za dobijanje informacija o radu sistema, upravljanje GUI (Windows), itd.

Izvršenje sistemskog poziva

Praktikum - Poglavlje 1. Operativni sistem UNIX/Linux



Microsoft Windows



Lsass = local security authentication server

POSIX = portable operating system interface

GDI = graphics device interface

DLL = dynamic link libraries

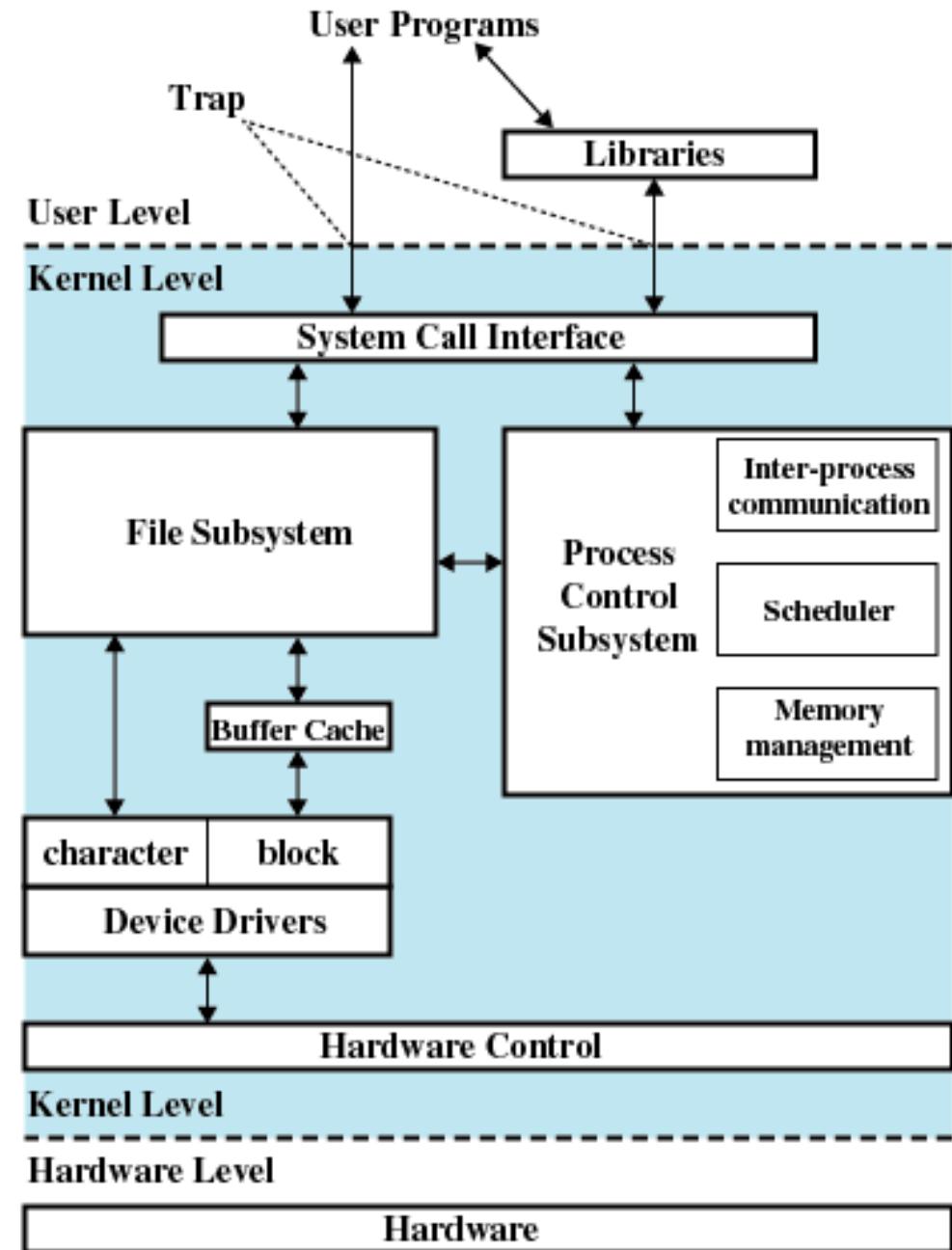
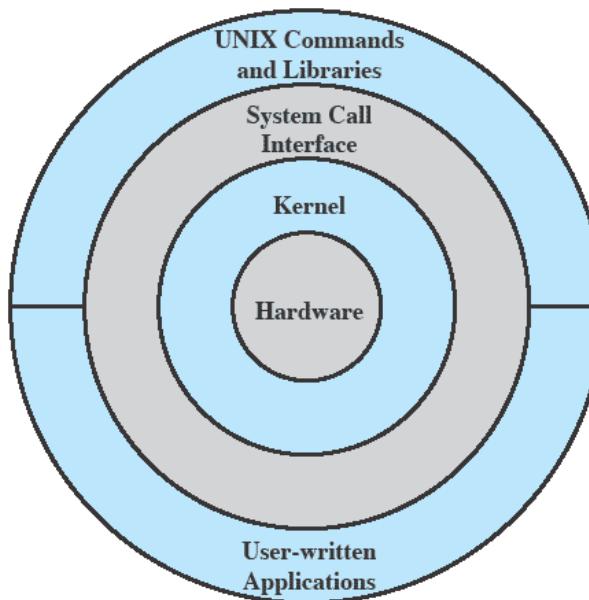
Colored area indicates Executive

UNIX

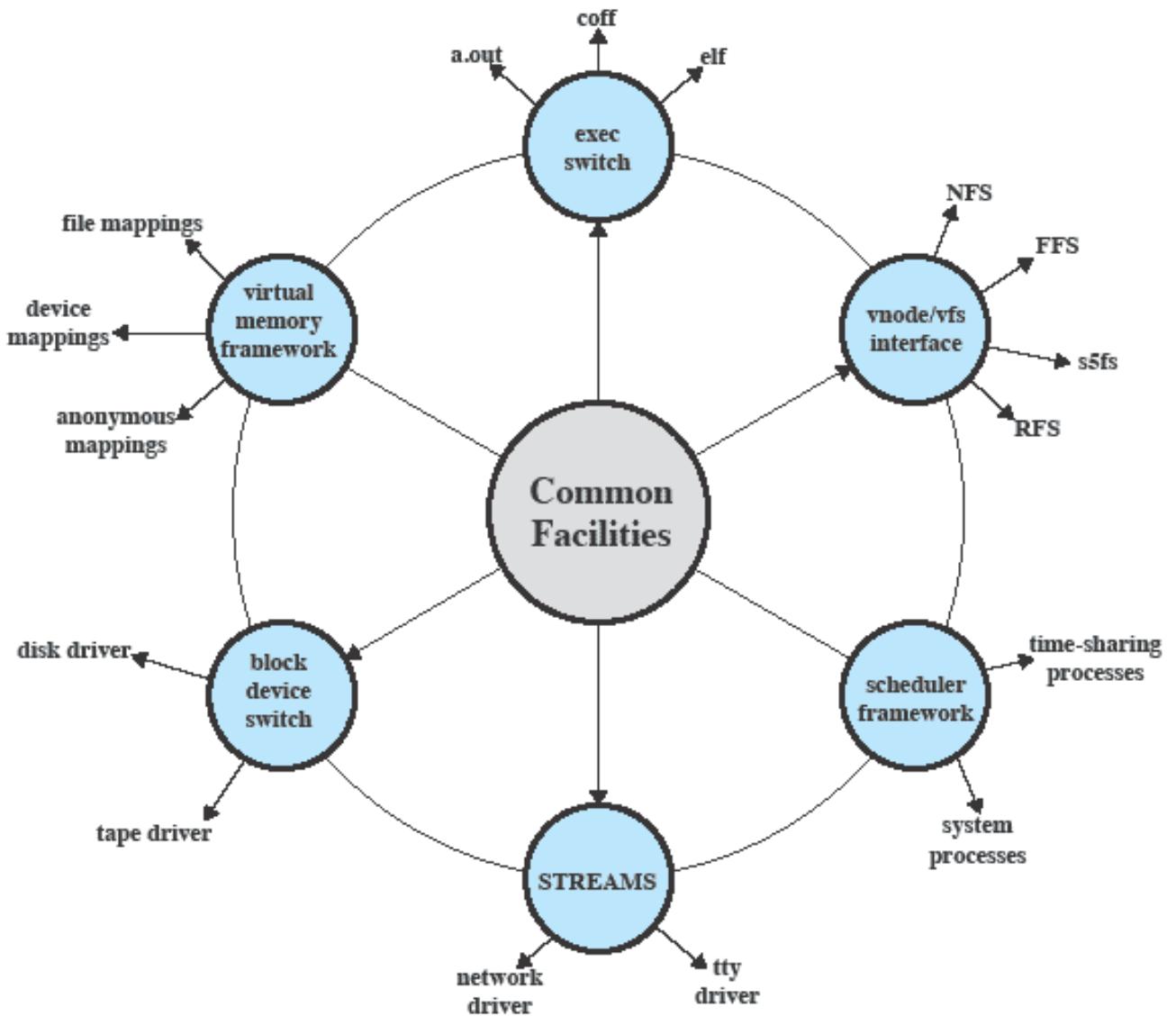
● Savremeni Unix

- System V R4 (SVR4)
- Solaris 10
- 4.4 BSD & FreeBSD

● Tradicionalni UNIX kernel



Savremeni UNIX kernel

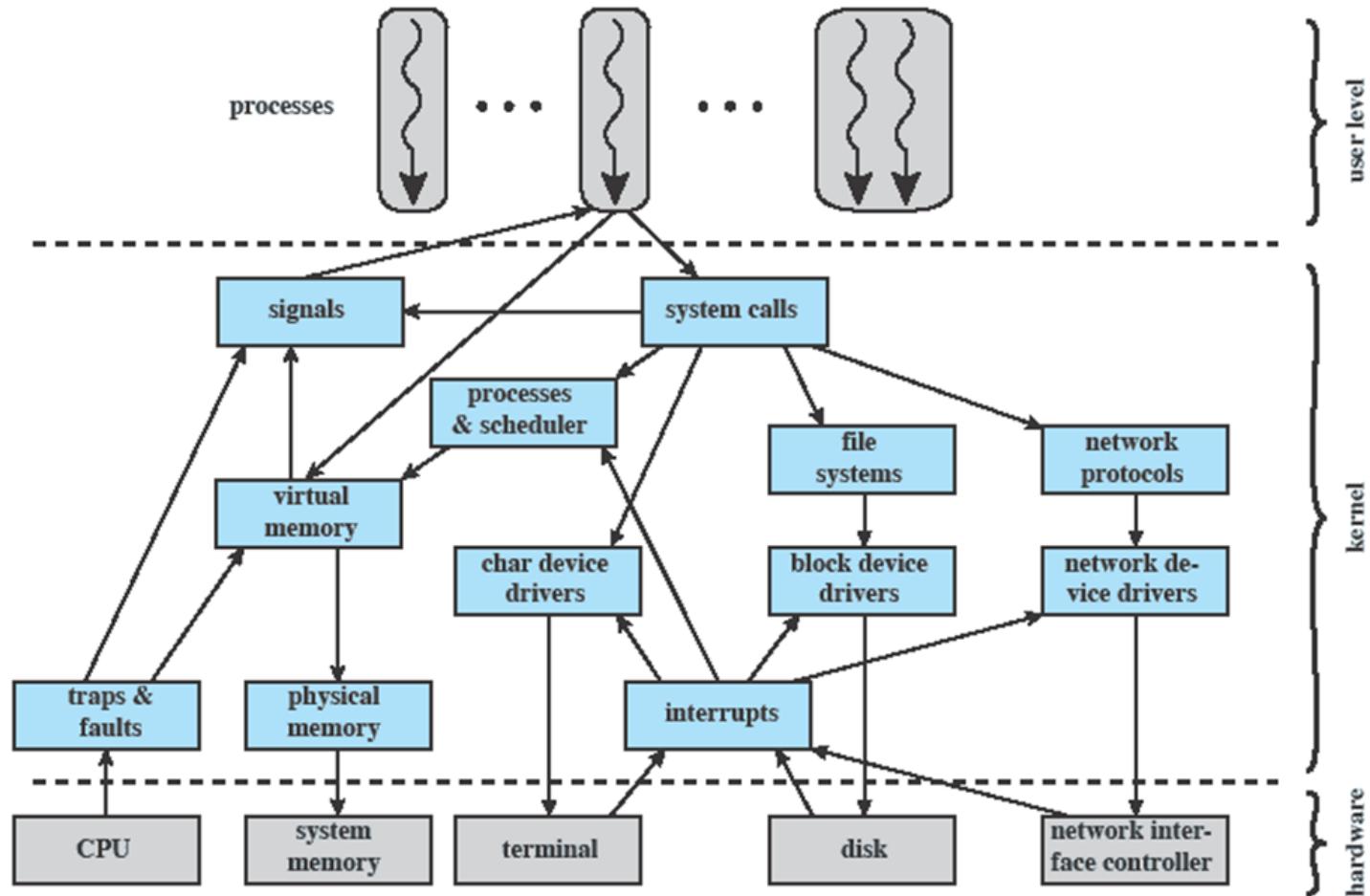


Uvod i pregled operativnih sistema

Operativni sistemi

Linux

● Komponente Linux kernela





Domaći zadatak

- Praktikum iz Sistemskog softvera
 - I. UNIX/Linux - Poglavlje 1. Operativni sistem UNIX/Linux
 - II. Windows - Poglavlje 1. Operativni sistem Windows 2000
- Pročitati poglavlja:
 - 2.7 Pregled Microsoft-ovog Windowsa
 - 2.8 Tradicionalni sistemi Unix
 - 2.9 Savremeni sistemi Unix
 - 2.10 Linux
- *Student resources*
 - <http://williamstallings.com/OperatingSystems/OS9e-Student/>
- *Animations*
<http://williamstallings.com/OS/Animation/Animations.html>
<https://apps.uttyler.edu/Rainwater/COSC3355/Animations/index.htm>



Operativni sistemi

Upravljanje procesima

Prof. dr Dragan Stojanović

Katedra za računarstvo
Univerzitet u Nišu, Elektronski fakultet



Literatura

- ◆ *Operating Systems: Internals and Design Principles*, edition, W. Stallings, Pearson Education Inc., 7th – 2012, (5th -2005, 6th - 2008, 8th – 2014 , 9th – 2017)
 - <http://williamstallings.com/OperatingSystems/>
 - <http://williamstallings.com/OperatingSystems/OS9e-Student/>
- ◆ Poglavlje 3: Opis procesa i upravljanje



Koncept procesa

- ❖ Svi multprogramske OS zasnovane su na konceptu procesa
- ❖ Operativni sistem mora da obezbedi:
 - ❖ Kreiranje procesa
 - ❖ Preplitanje izvršenja više procesa u cilju maksimalnog iskorišćenja procesora
 - ❖ Dodelu resursa procesima u skladu sa određenom politikom,
 - ❖ Sinhronizaciju i uzajamno isključivanje procesa i zaštitu resursa svakog procesa od strane drugih procesa
 - ❖ Komunikaciju između procesa



Osnovni pojmovi i koncepti

- ➊ Računarska platforma se sastoji od skupa hardverskih resursa
- ➋ Računarske aplikacije se razvijaju kako bi izvršile neki zadatak; prihvataju ulaz spolja, vrše obradu i generišu izlaz
- ➌ Nije efikasno pisati aplikacije direktno za datu hardversku platformu
- ➍ OS obezbeđuje interfejs između aplikacije i hardvera računara
- ➎ OS pruža apstrakciju resursa koje aplikacije može tražiti ili pristupati i upravlja njihovom upotrebom



Šta je proces?

- ❖ **Proces** je osnovni koncept operativnog sistema i predstavlja **program u izvršavanju na procesoru**
 - ❖ Proces se ponekad naziva **zadatak (task)**
 - ❖ Svi multiprogramske OS su izgrađeni oko koncepta procesa
- ❖ Gledano sa strane OS-a, **proces** je osnovna jedinica izvršavanja i najmanji entitet koji se može planirati, dodeliti i izvršavati na procesoru
- ❖ **Proces** je jedinica aktivnosti koju karakteriše izvršenje sekvence instrukcija, trenutno stanje i pridružen skup sistemskih resursa



Elementi procesa

◆ Proces se sastoji od:

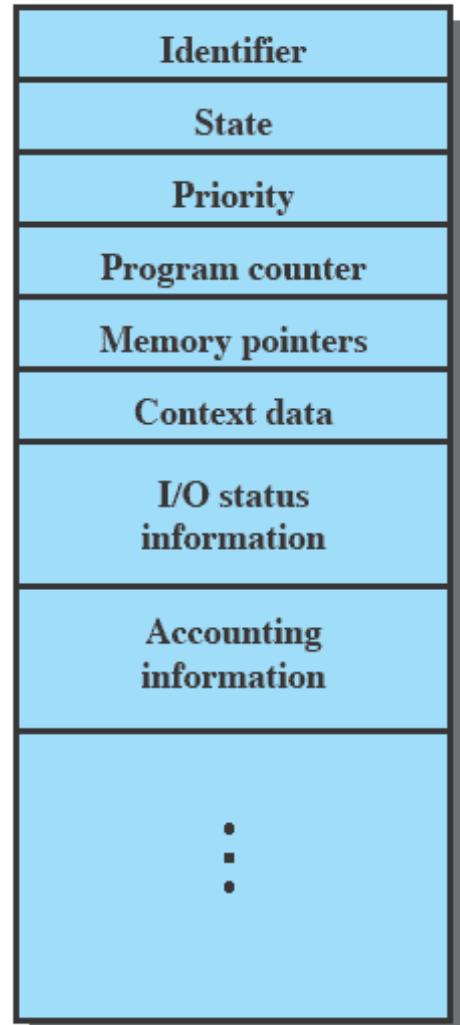
- Programskog kôda
- Skupa podataka nad kojima se izvršavaju instrukcije
- Steka
- Atributa koji opisuju stanje procesa i koji su smešteni u

Upravljački blok procesa:

- Identifikator
- Stanje
- Prioritet
- Programske adrese (Program counter)
- Pokazivač na memoriju u koju je smešten proces
- Kontekstni podaci (podaci smešteni u registrima procesora za vreme izvršenja procesa)
- Informacije o U/I statusu
- Informacije o obračunu korišćenja resursa
- ...

Upravljački blok procesa

- ❖ *Process Control Block* - PCB
- ❖ Sadrži atributе procesa
- ❖ Kreiran i upravljan od strane OS
- ❖ Obezbeđuje podršku za višestruke procese
- ❖ *Tabela procesa* predstavlja skup PCB-ova svih procesa

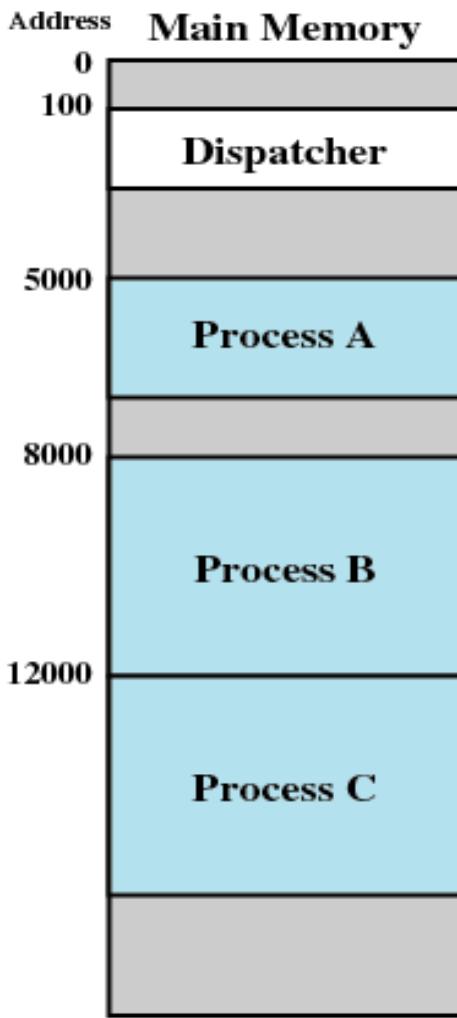




Praćenje (tok, *trace*) procesa

- Ponašanje individualnog procesa je prikazano sekvencom instrukcija koje se izvršavaju
- Ova lista instrukcija se naziva **praćenje** (tok, *trace*) procesa
- **Dispecer** (rasporedjivač, *dispatcher*) je program koji prebacuje (*switch*) procesor od jednog procesa drugom, i vrši zamenu aktivnog procesa

Izvršenje procesa



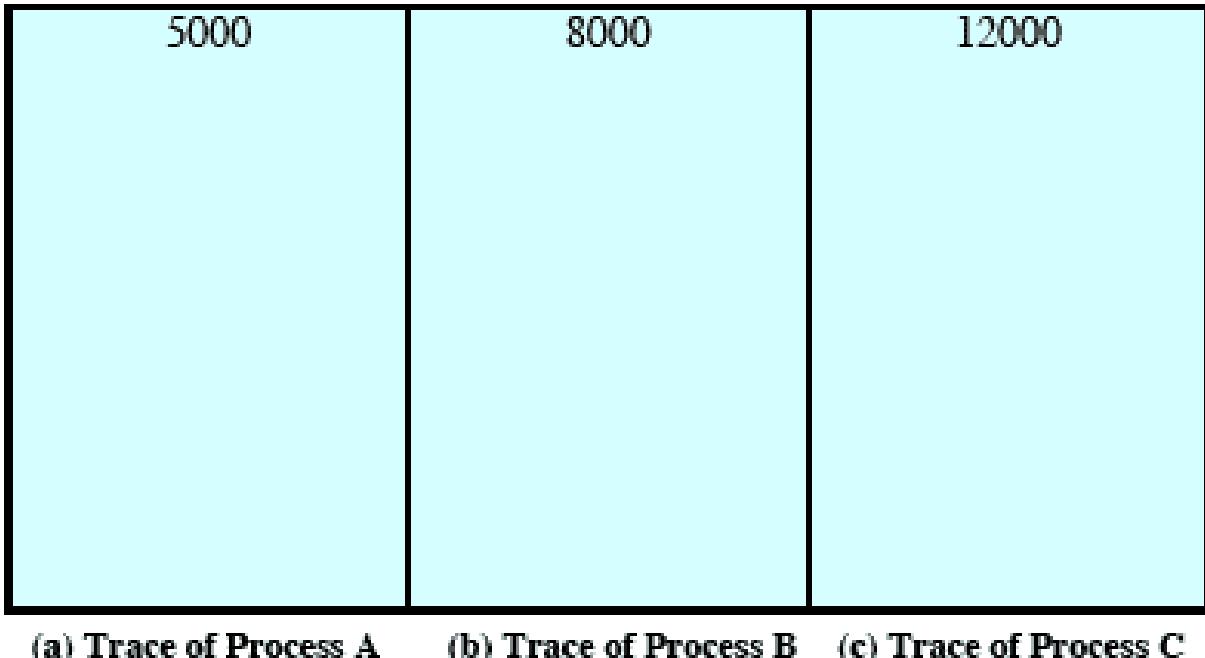
Program Counter

8000 

- ➊ Razmotrićemo tri procesa koji se izvršavaju
- ➋ Svi procesi su u glavnoj memoriji, uključujući i dispečer
- ➌ U ovom primeru ćemo ignorisati postojanje virtuelne memorije.
- ➍ Programski brojač u CPU sadrži adresu instrukcije koja treba da bude izvršena
- ➎ Kada se aktivira sledeći proces menja se sadržaj programskog brojača i njegova vrednost postavlja na adresu instrukcije koju treba izvršiti u novom procesu

Praćenje procesa sa stanovišta procesa

- ❖ Svaki proces se izvršava do svog završetka

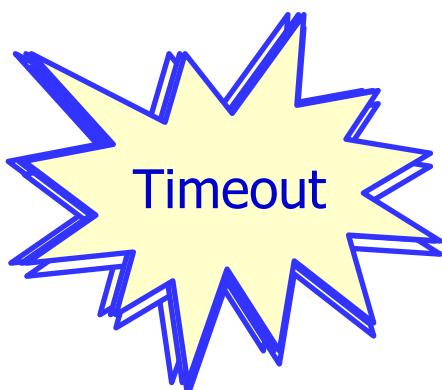
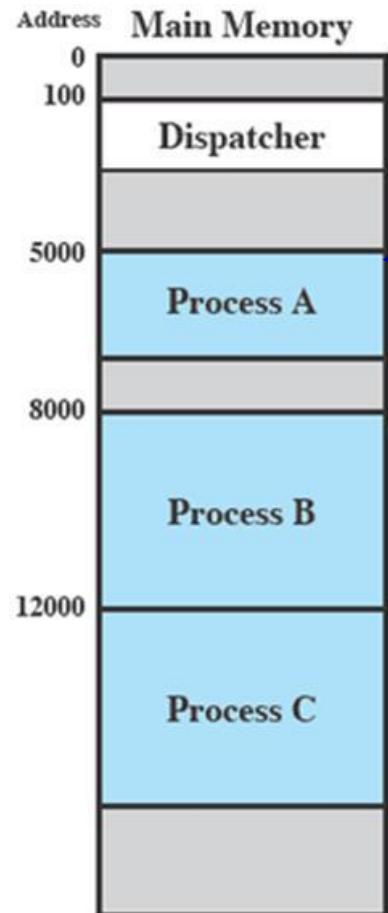


5000 = Starting address of program of Process A

8000 = Starting address of program of Process B

12000 = Starting address of program of Process C

Praćenje procesa sa stanovišta procesora



1	5000
2	5001
3	5002
4	5003
5	5004
6	5005
7	100
8	101
9	102
10	103
11	104
12	105
13	8000
14	8001
15	8002
16	8003
17	100
18	101
19	102
20	103
21	104
22	105
23	12000
24	12001
25	12002
26	12003

Timeout

27	12004
28	12005
29	100
30	101
31	102
32	103
33	104
34	105
35	5006
36	5007
37	5008
38	5009
39	5010
40	5011
41	100
42	101
43	102
44	103
45	104
46	105
47	12006
48	12007
49	12008
50	12009
51	12010
52	12011

Timeout

41	100
42	101
43	102
44	103
45	104
46	105
47	12006
48	12007
49	12008
50	12009
51	12010
52	12011

Timeout

Timeout

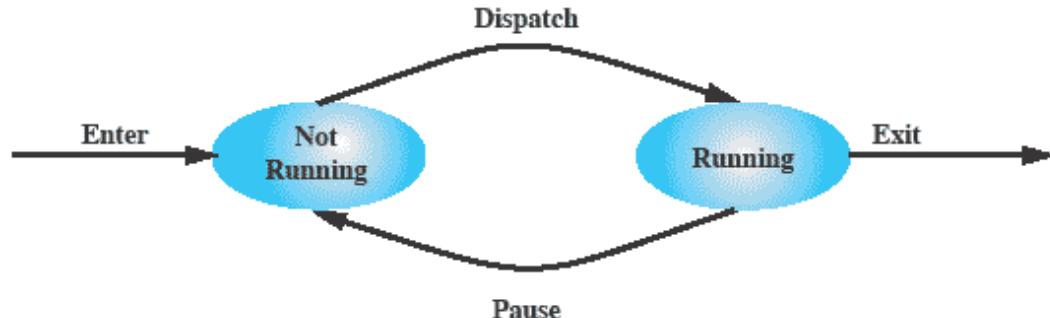
100 = Starting address of dispatcher program

Shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

Model dva stanja procesa

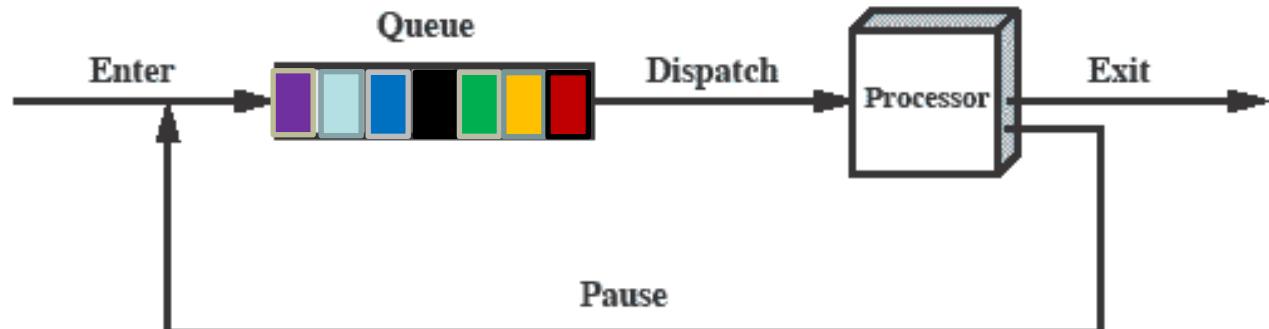
- Proces može biti u jednom od dva stanja

- Izvršava se
- Ne izvršava se



- Red procesa

- Procesi koji se ne izvršavaju se smeštaju u red (elementi su pokazivači na PCB), izvršavaju se podređeno vreme i vraćaju u red dok se ne završe





Upravljanje procesima u OS

◆ Osnovne funkcije:

- Kreiranje i završavanje procesa
- Suspendovanje i ponovo aktiviranje
- Planiranje izvršenja procesa i upravljanje procesorom/procesorima
- Obezbeđenje mehanizama za sinhronizaciju i komunikaciju između procesa
- Obezbeđenje mehanizama za upravljanje deadlock-om (uzajamno blokiranje, samrtni zagrljaj, zastoj)



Kreiranje (stvaranje) procesa

◆ Glavni razlozi za kreiranje procesa

1. Iniciranje nekog paketnog ("batch") posla
2. Interaktivna prijava (login) korisnika
3. Kreiran od strane OS za obezbeđenje nekog servisa
4. Kreiran od strane postojećeg procesa

◆ Roditeljski (*parent*) i proces dete (*child*, potomak)

◆ Aktivnosti OS pri kreiranju procesa:

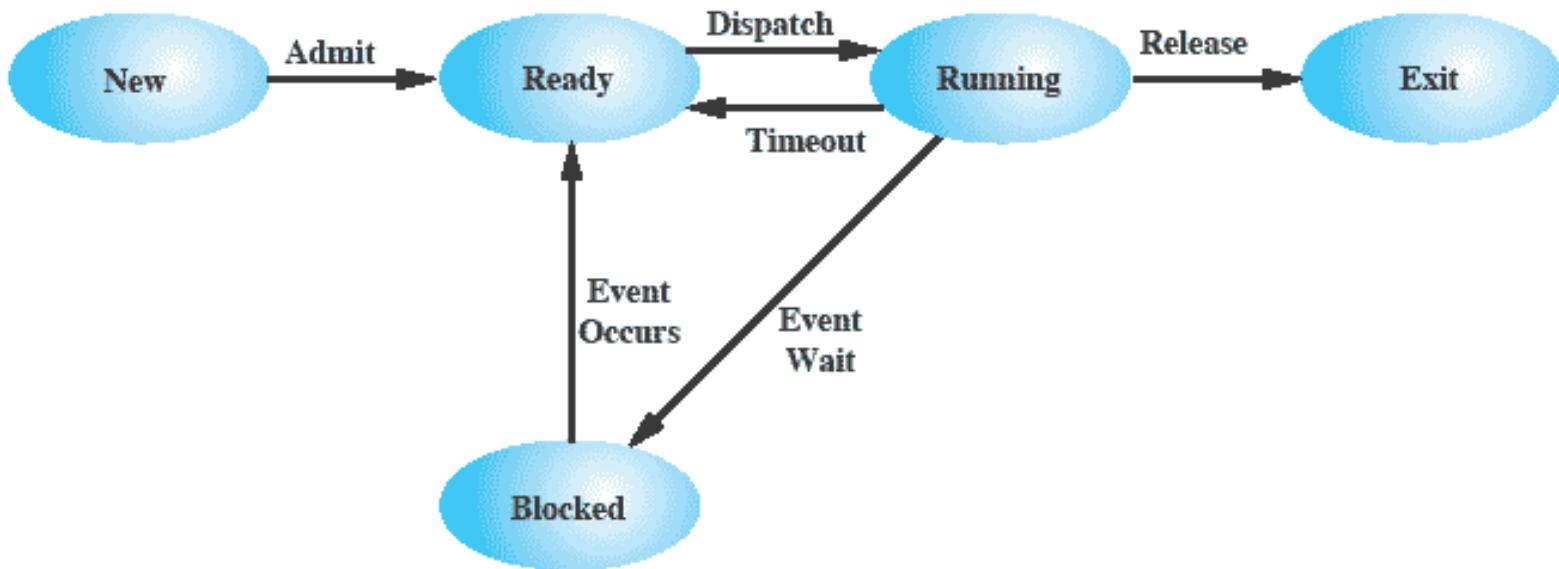
1. Dodela identifikatora procesu
2. Dodela inicijalnog prioriteta procesu
3. Kreiranje i unos PCB novog procesa u Tabelu procesa
4. Dodela početnih resursa procesu (memorija, otvorene datoteke, itd.)



Završetak (terminiranje) procesa

1. Normalni završetak
2. Istečlo dodeljeno vreme za izvršavanje
3. Nedovoljno raspoložive memorije
4. Greška usled narušavanja zaštite (*protection error*) u pristupu memoriji ili drugim resursima bez autorizacije
5. Aritmetička greška
6. Prekoračenje vremena čekanja na neki događaj
7. U/I greška, pogrešna instrukcija, privilegovana instrukcija, pogrešna upotreba podataka, itd.
8. "Ubijen" od OS ili operatera
9. Završetak roditeljskog procesa
10. Na zahtev roditeljskog procesa koji ima privilegije da završi procese decu

Model procesa sa pet stanja



- ➊ Izvršavanje (*Running*) – Proces koji se trenutno izvršava
- ➋ Spreman (*Ready*) - Proces koji je spreman za izvršavanje kada mu se pruži prilika
- ➌ Blokiran (*Blocked*) - Proces ne može da se izvršava dok se ne desi neki događaj, poput završetka U/I operacije
- ➍ Novi (*New*) – Proces tek kreiran, ali još nije primljen od strane OS u skup izvršnih procesa
- ➎ Završen (*Exit*) – proces izbačen iz skupa izvršnih procesa od strane OS



Prelazi između stanja procesa

- Null → Novi
 - Kreiranje procesa za izvršenje programa
- Novi → Spreman
 - OS prebacuje proces kada ima dovoljno resursa za njegovo aktiviranje
- Spreman → Izvršavanje
 - OS bira jedan od spremnih procesa za izvršavanje na CPU
- Izvršavanje → Završen
 - Trenutno aktivan proces se završava od strane OS iz nekog od razloga
- Izvršavanje → Spreman
 - Istečlo je vreme dodeljeno procesu za izvršavanje, ili je proces višeg prioriteta postao spremан

Prelazi između stanja procesa (2)

● Izvršavanje → Blokiran

- Proces je zahtevao resurs zbog koga mora da čeka (datoteka, memorijska sekcija, poruka od drugog procesa) ili U/I operaciju koja mora biti završena pre nastavka procesa

● Blokiran → Spreman

- Nastao je događaj na koji je proces čekao

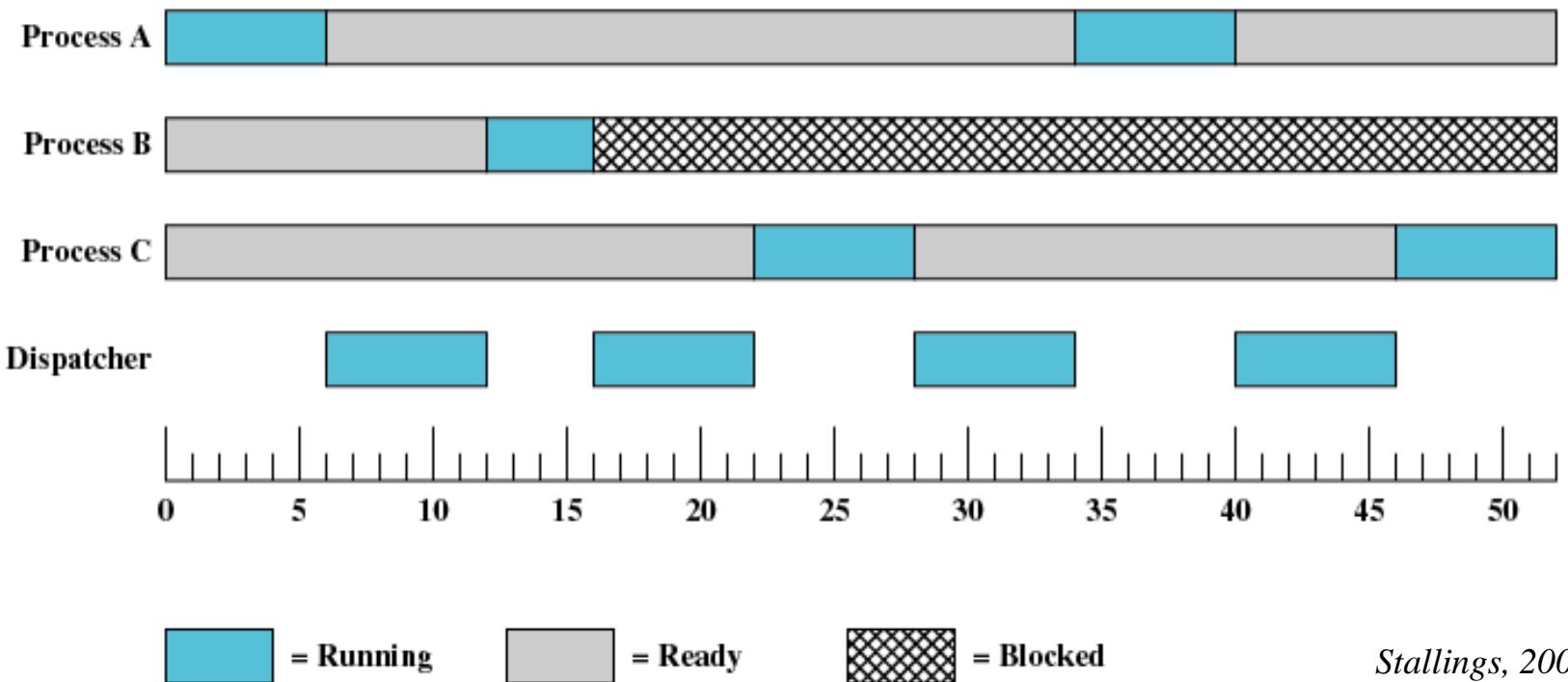
● Spreman → Završen

- Nije prikazano na dijagramu; u nekim sistemima roditeljski proces može završiti proces decu u bilo kom trenutku, pa i kad su u stanju spremni, a takođe završetkom roditeljskog procesa završavaju se i sva njegova deca procesi

● Blokiran → Završen

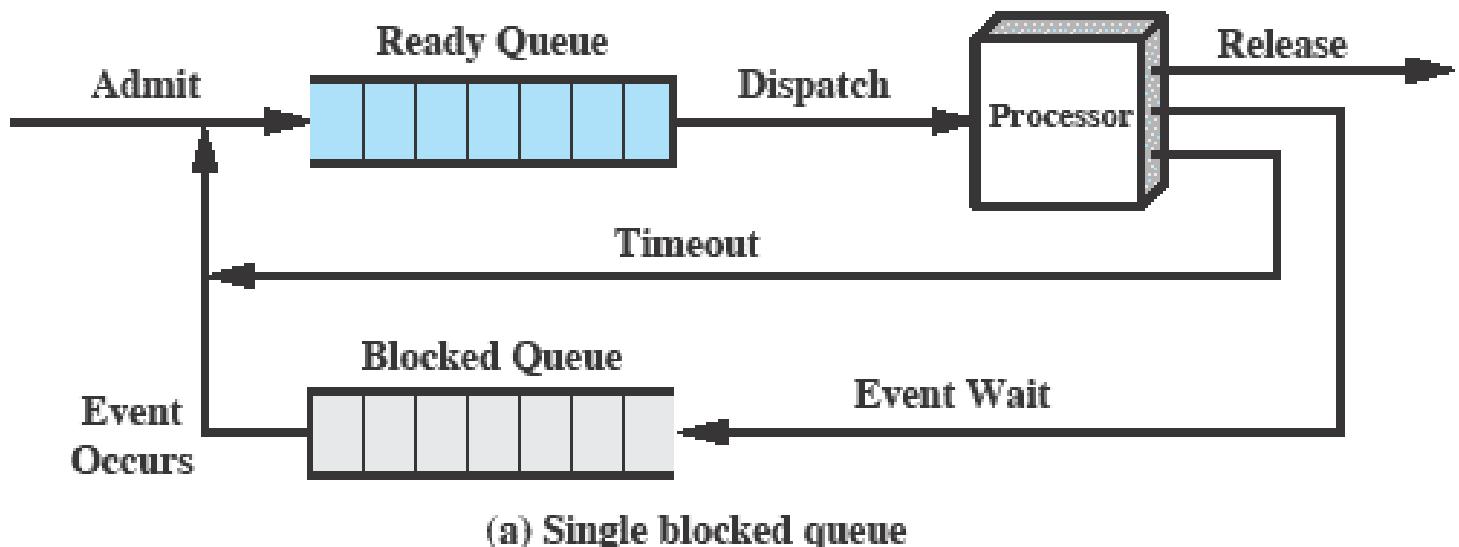
Stanja procesa

- ❖ Vremenski dijagram izvršenja procesa za prethodni primer

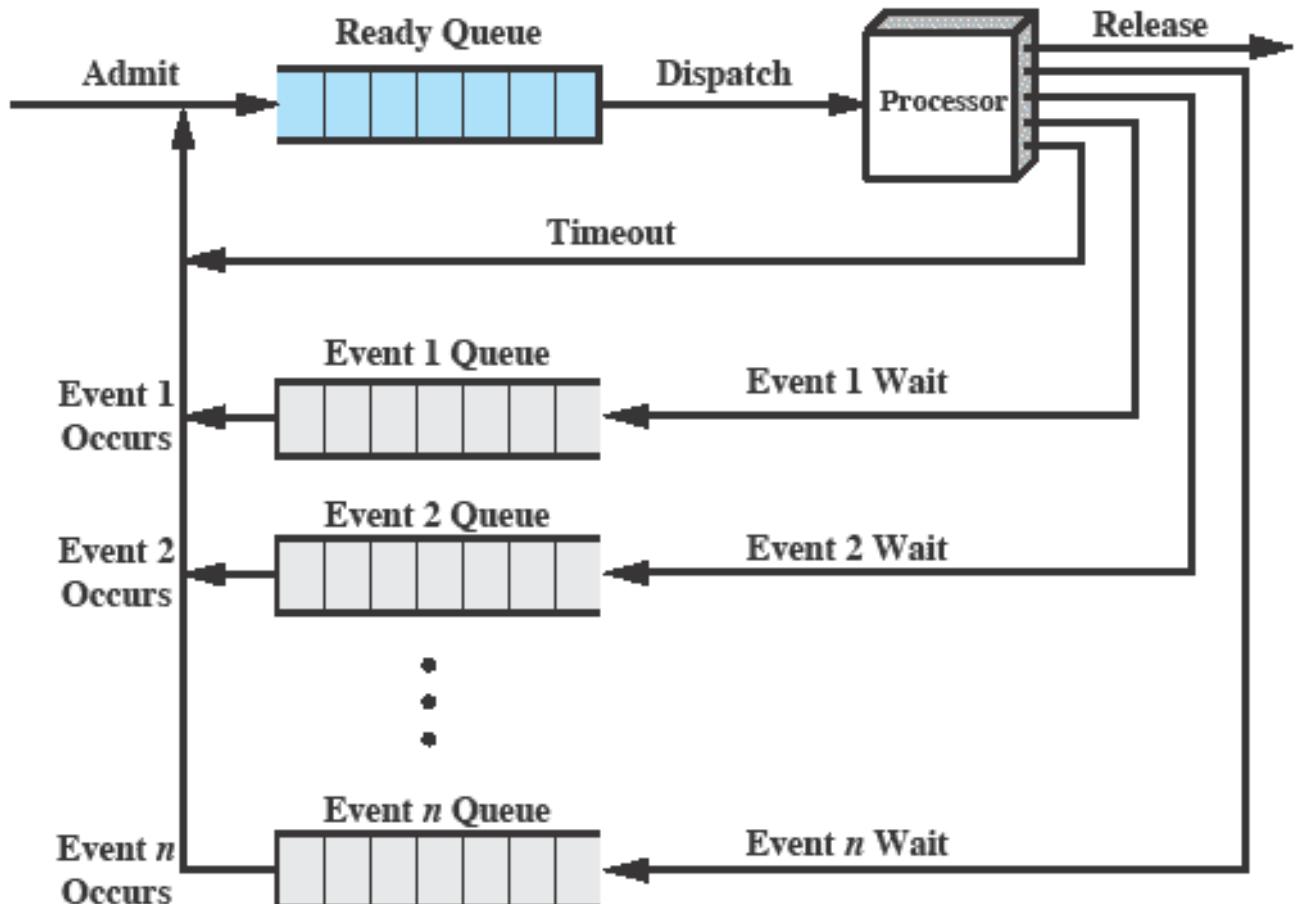


Dva reda procesa

- ➊ Jedinstveni red blokiranih procesa



Višestruki redovi blokiranih



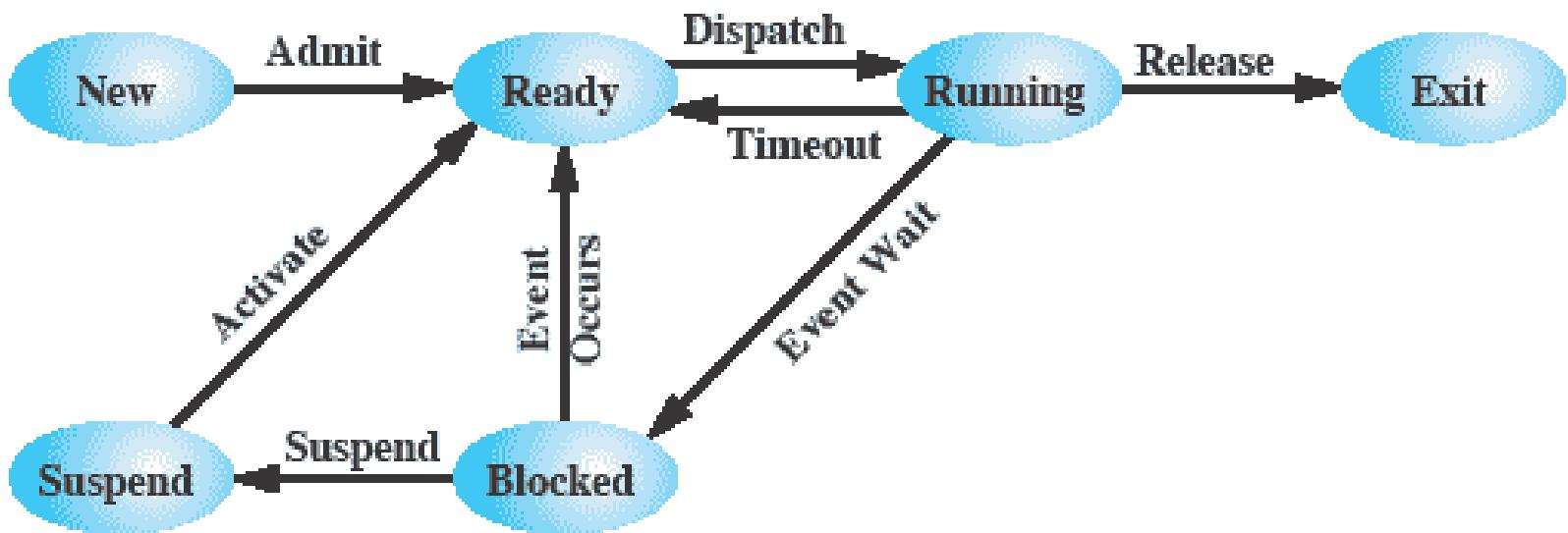
(b) Multiple blocked queues

Suspendovani procesi

- ➊ Procesor je mnogo brži od U/I uređaja, tako da može da se desi da mnogi (svi) procesi čekaju na završetak U/I operacija
 - ▣ Prebaciti (*Swap*) ove procese iz memorije na disk radi oslobođanja više memorije i koristiti procesor za aktiviranje novih procesa ili prethodno suspendovanih procesa
- ➋ Proces u stanju blokiran prelazi u stanje **suspendovan** kada se prebaci na disk
- ➋ Dva nova stanja
 - ▣ Blokirani/Suspendovan (*Blocked/Suspend*)
 - ▣ Spreman/Suspendovan (*Ready/Suspend*)

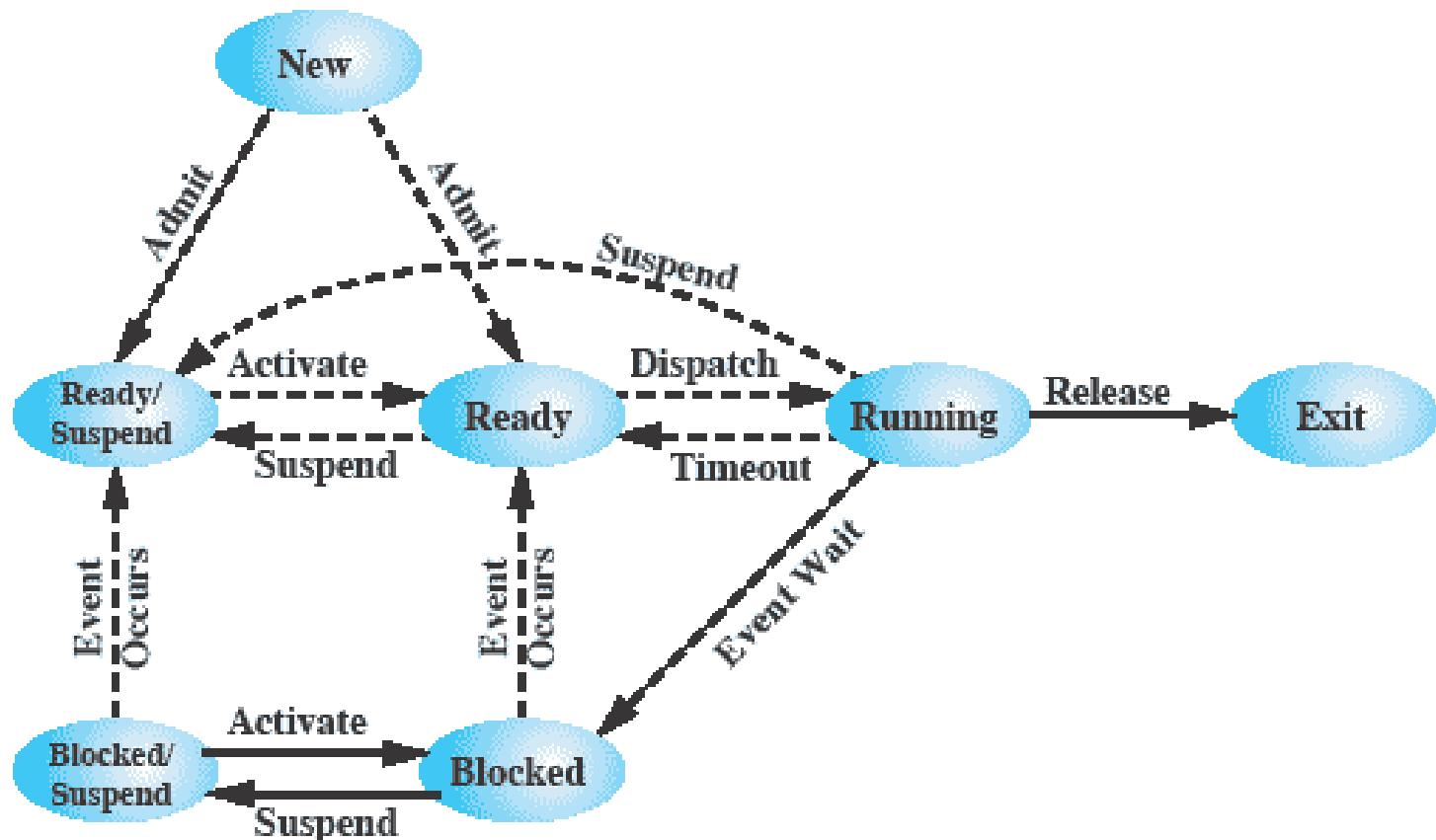
Dijagram prelaza stanja procesa

- Sa jednim suspendovanim stanjem



Dijagram prelaza stanja procesa

- Sa dva suspendovana stanja



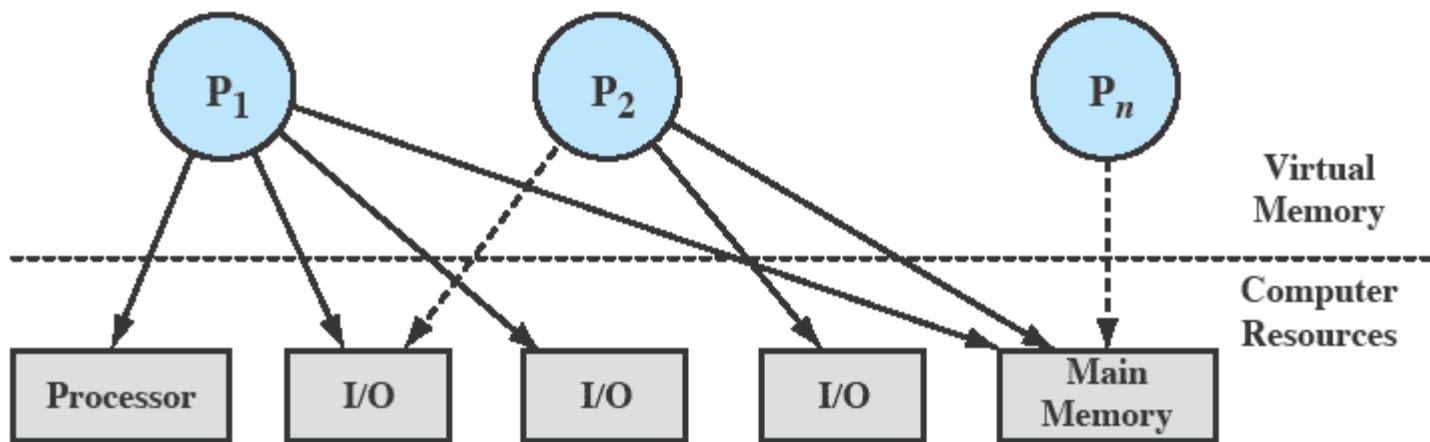


Razlozi za suspendovanje procesa

- ◆ **Prebacivanje** (swapping) – OS mora da oslobodi dovoljno glavne memorije da aktivira proces koji je spreman za izvršenje
- ◆ **Drugi OS razlozi** – OS može suspendovati proces za koji se sumnja da izaziva “probleme” tokom izvršavanja
- ◆ **Zahtev interaktivnog korisnika** – Suspendovanje procesa u toku debagiranja
- ◆ **Vremenski** – proces se izvršava periodično; između perioda može biti suspendovan
- ◆ **Roditeljski proces** može suspendovati izvršenje procesa-dece

Procesi i resursi

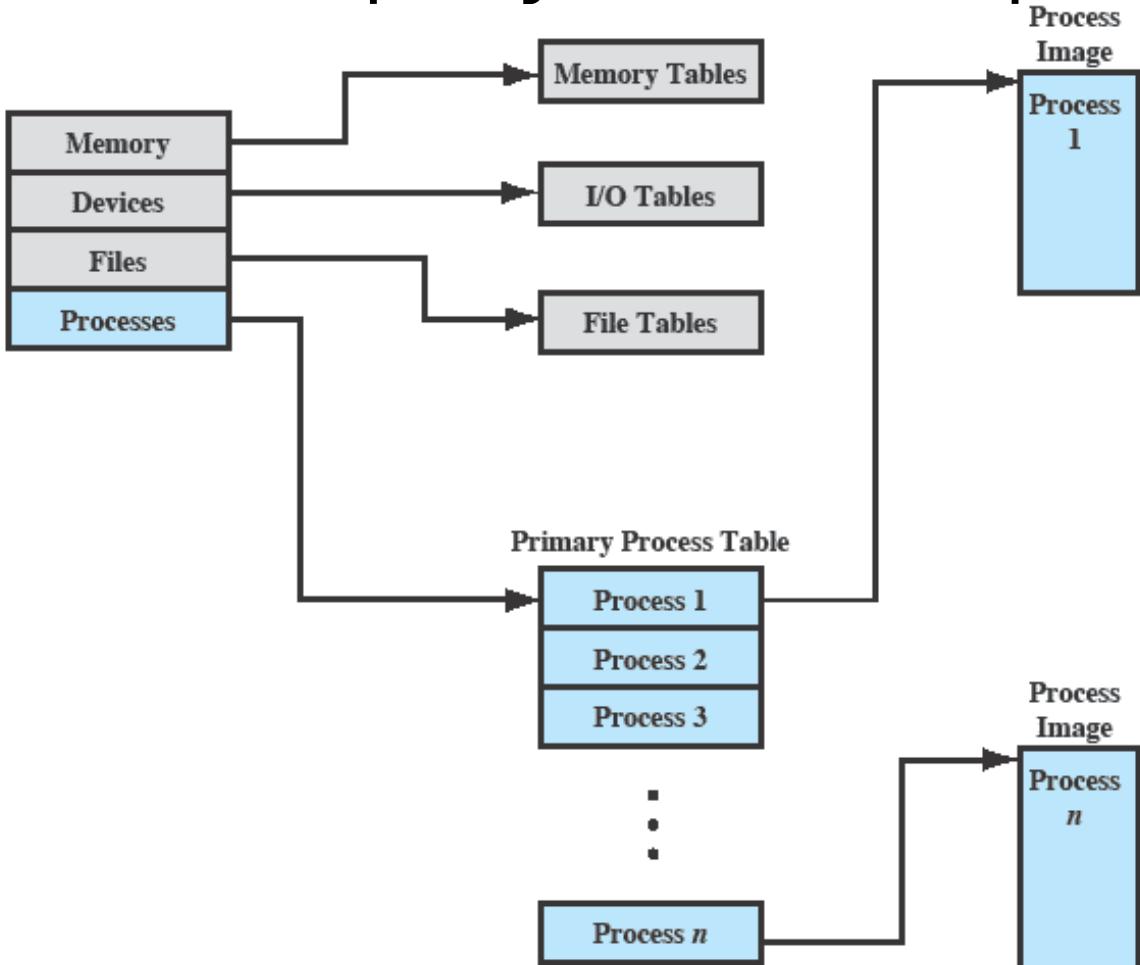
- Procesi i njima dodeljeni (alocirani) resursi u nekom vremenskom trenutku
- Da bi OS upravljaо procesima i resursima on mora imati informacije o trenutnom stanju svakog procesa i resursa.
- Za svaki entitet kojim upravlja OS kreiraju se odgovarajuće tabele



Upravljanje procesima
Operativni sistemi

Upravljačke tabele OS

- Generalna struktura upravljačkih tabela operativnog sistema



Upravljanje procesima
Operativni sistemi



Tabela procesa

- ➊ Da bi upravljaо procesima OS mora da u svakom trenutku zna detalje o svakom procesu
 - ▣ Trenutno stanje
 - ▣ Identifikator procesa (Process ID)
 - ▣ Lokacija u memoriji
 - ▣ Ostale atributi procesa
- ➋ **Upravljački blok procesa** (*Process Control Block – PCB*) – kolekcija atributa procesa (*Deskriptor procesa*)
 - ▣ Najvažnija struktura podataka u OS; sadrži sve informacije o procesima neophodne OS – skup PCB-a definiše stanje OS
- ➋ **Slika procesa** (*Process image*) predstavlja skup koji čine program, podaci, magacin (stek, *stack*), i atributi procesa (PCB) smešteni u glavnu memoriju (delom i na disku)



Atributi procesa

- ❖ Informacije u okviru upravljačkog bloka procesa (PCB) mogu se grupisati u tri kategorije:
 - ❖ Identifikatori procesa
 - ❖ Informacije o stanju procesora
 - ❖ Informacije za upravljanje procesom



Identifikatori procesa

- Svakom procesu je dodeljen jedinstveni numerički identifikator - (Process ID, PID)
- Svaki proces može da poseduje identifikatore roditeljskog procesa, grupe procesa korisnika koji je kreirao proces, itd.
- Sve druge tabele kojima upravlja OS mogu da koriste identifikator procesa da bi kros-referencirale PCB procesa



Informacije o stanju procesora

- ➊ Sastoje se od sadržaja registara procesora.
 - Registara opšte namene vidljivih od strane korisnika
 - Upravljačkih i statusnih registara (PC, uslovni kodovi, flagovi za dozvoljen/nedozvoljen prekid, mod izvršenja,...)
 - Stack pointera - LIFO strukture za smeštanje parametara, povratnih adresa i lokalnih promenljivih pri pozivima procedura i sistemskim pozivima
- ➋ *Program Status Word* (PSW)
 - Sadrži statusne informacije
 - Primer
 - EFLAGS register na Pentium procesorima
 - Itanium (IA-64) - psr register ima polje od dva bita: cpl (0 najviše privilegija, 3 najmanje privilegija)



Informacije za upravljanje procesom

- ❖ Informacije o stanju i planiranju (*scheduling*)
 - ❖ Stanje procesa
 - ❖ Prioritet
 - ❖ Informacije vezane za planiranje – zavisno od algoritma planiranja
 - ❖ Događaj na koji proces čeka
- ❖ Struktuiranje (povezivanje, ulančavanje) PCB-a
 - ❖ Procesi (PCB) su povezani u različitim strukturama (redovima, listama, itd.): roditelj-dete, red čekanja na U/I, itd.
- ❖ Međuprocesna komunikacija
 - ❖ Različiti flag-ovi, signali, poruke između procesa



Informacije za upravljanje procesom (2)

◆ Privilegije procesa

- Privilegije za pristup memoriji, izvršenje određenih tipova instrukcija, sistemskih servisa

◆ Upravljanje memorijom

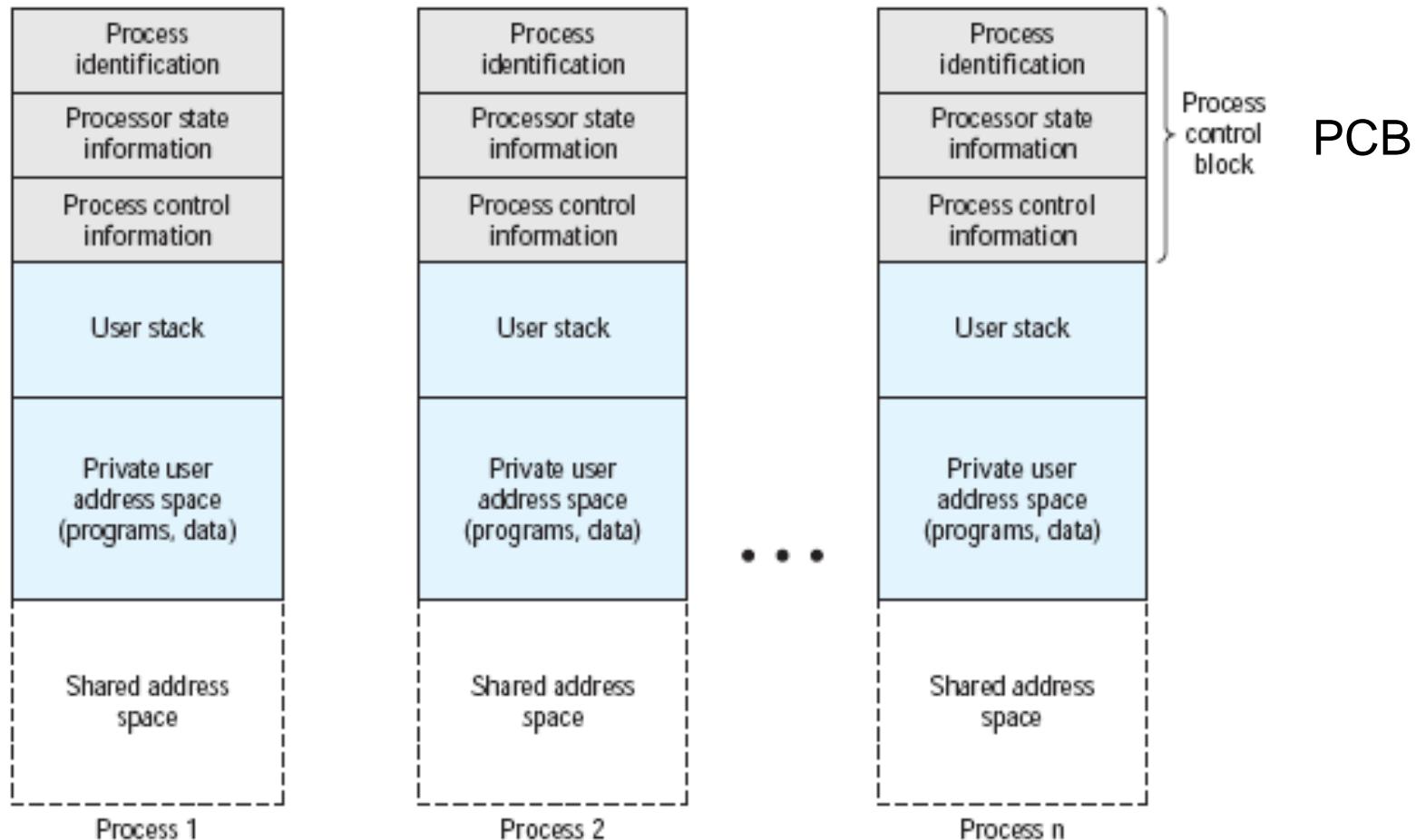
- Pokazivači (adrese) na tabele stranica/segmenata koje opisuju virtuelnu memoriju dodeljenu procesu

◆ Vlasništvo i korišćenje resursa

- Resursi kojima upravlja proces, poput otvorenih datoteka
- Takođe može biti uključena i istorija korišćenja procesora ili ostalih resursa za potrebe planiranja procesa

Slika procesa u virtuelnoj memoriji

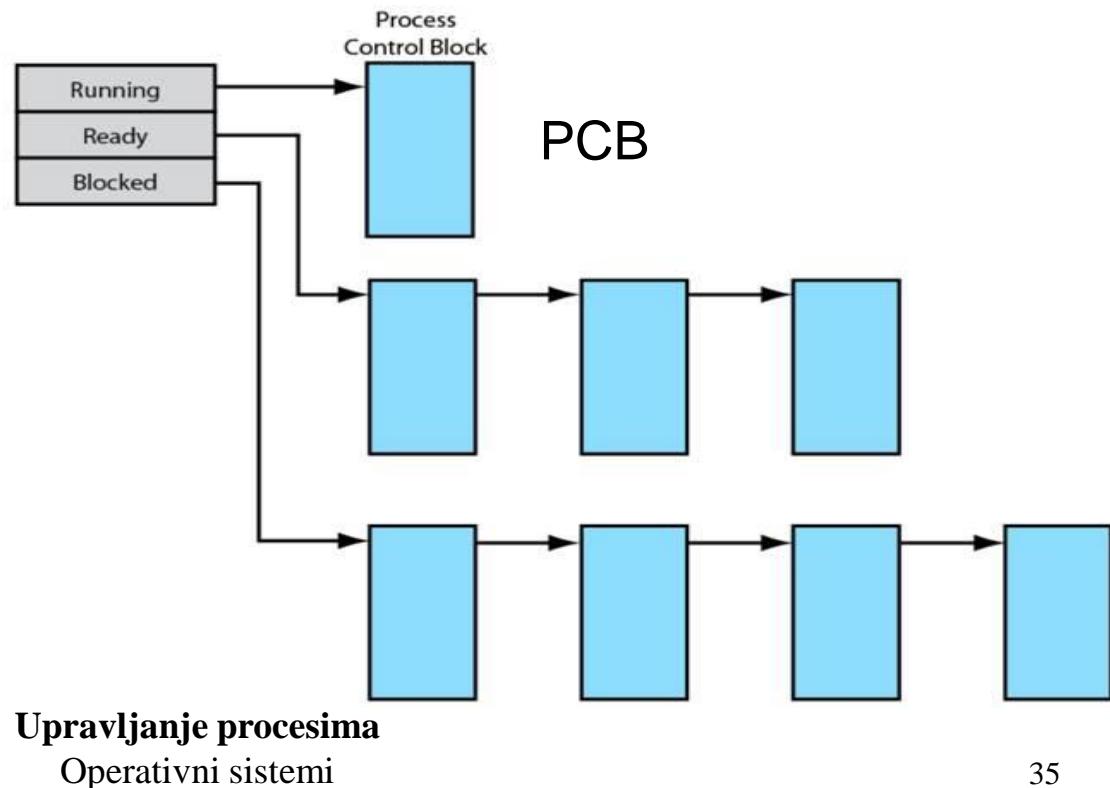
Struktura slike procesa u virtuelnoj memoriji

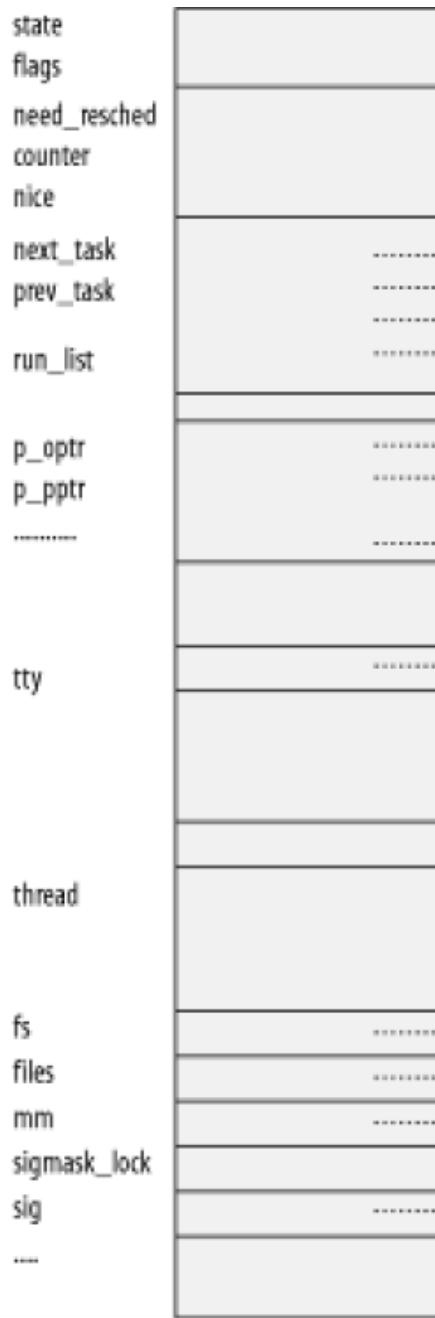


Upravljanje procesima
Operativni sistemi

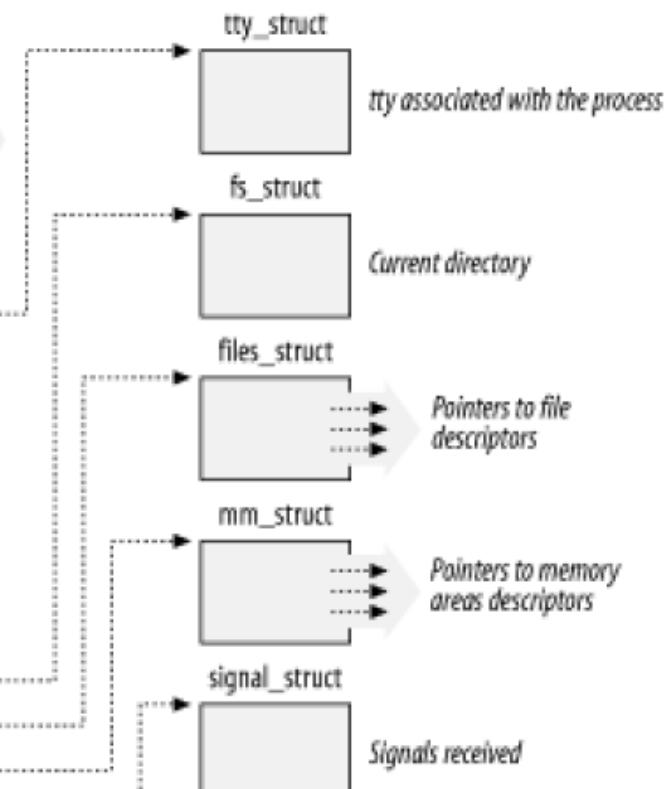
Struktura redova (listi) procesa

- PCB je najvažnija struktura podataka u operativnom sistemu (Deskriptor procesa)
- OS moduli (planiranje, alokacija resursa, obrada prekida, nadgledanje i analiza performansi, itd.) čitaju i/ili modifikuju PCB-ove
- PCB su povezani u različitim redovima i tabelama koji se često implementiraju kao lančane liste





Deskriptor procesa u Linux-u - Šematski prikaz



D. Bovet, M. Cesati, *Understanding the Linux Kernel.*
2nd Edition, O'Reilly 2002



Deskriptor procesa u Linux-u (2.4)

/include/linux/sched.h

```
struct task_struct {  
    volatile long      state; /*TASK_RUNNING, TASK_INTERRUPTIBLE, TASK_UNINTERRUPTIBLE,  
                             TASK_STOPPED, TASK_ZOMBIE */  
    long               counter; //brojač vremenskih taktova  
    long               priority; //prioritet procesa  
    unsigned           signal;  
    unsigned           blocked; /* bitmap of masked signals */  
    unsigned           flags;   /* per process flags, defined below */  
  
    struct task_struct *next_task, *prev_task; //pokazivač na naredni/prethodni proces u  
                                                listi  
    unsigned long      saved_kernel_stack;  
    unsigned long      kernel_stack_page;  
  
    int                pid;  
  
/* pointers to (original) parent process, youngest child, younger sibling,  
 * older sibling, respectively. (p->father can be replaced with  
 * p->p_pptr->pid) */  
//pokazivači na roditelja, decu i braću tekućeg procesa  
struct task_struct *p_opptr, *p_pptr, *p_cptr, *p_ysptr, *p_osptr;  
...  
}
```



Deskriptor procesa u Linux-u (2)

...

```
struct wait_queue    *wait_chldexit;
unsigned short      uid,euid,suid,fsuid;
unsigned short      gid,egid,sgid,fsgid;
unsigned long       timeout, policy, rt_priority;
long                utime, stime, cutime, cstime, start_time;
struct fs_struct   *fs;
struct files_struct *files;
struct mm_struct   *mm;
struct signal_struct *sig;
...
} //end task_struct
```



Upravljanje izvršenjem procesa

- ◆ Većina procesora podržava najmanje dva moda (režima) izvršavanja
- ◆ **Korisnički (*user*) mod (režim)**
 - Režim izvršavanja sa manje privilegija – zabranjen pristup svim memorijskim adresama i izvršavanje privilegovanih (U/I) instrukcija
 - Korisnički programi se izvršavaju u ovom režimu (modu)
- ◆ **Kernel (*system, supervisor*) mod (režim) – mod jezgra**
 - Privilegovan režim izvršavanja
 - U ovom režimu (modu) se izvršava kernel operativnog sistema
 - Bit(-ovi) u PSW označavaju trenutni režim izvršenja
- ◆ Intel Itanium procesor IA-64 ima statusni register procesora (*psr*) koji sadrži polje od dva bita **cpl**
 - Nivo 0 - najviše privilegija; nivo 3 - najmanje privilegija



Kreiranje procesa

Prilikom kreiranja novog procesa OS treba da:

- Dodeli jedinstveni identifikator procesa
- Dodeli (allocira) memorijski prostor za proces, tačnije za sve elemente slike procesa
- Inicijalizuje upravljački blok procesa (PCB)
- Uspostavi odgovarajuće veze novog procesa ulančavanjem njegovog PCB u odgovarajuće redove/liste
- Kreira ili proširi ostale strukture podataka
 - Na primer, za obračun korišćenja resursa, ili ocenu i analizu performansi



Zamena (*komutiranje*) procesa

- **Zamena** (*switch*) procesa predstavlja zamenu aktivnog procesa jednim od spremnih procesa
- Događaji koji zahtevaju izvršenje koda OS i mogu uzrokovati zamenu procesa
 - **Prekid** (*interrupt*) – eksterni događaj u odnosu na izvršenje procesa; prekid se javlja kao reakcija na eksterne, asinhrone događaje
 - Prekid generatora takta, U/I prekid, greška u referenci memorije
 - **Trap** – Odnosi se na izvršenje tekuće instrukcije procesa - ako nastane greška ili izuzetak koji je fatalan
 - **Sistemski poziv** (*supervisor call*) - Zahtev od strane procesa koji se izvršava za izvršenje U/I operacije (npr. pristup datoteci) kojim se aktivira odgovarajuća rutina OS.

Promena (*komutiranje*) režima

- ❖ Nakon izvršenja svake instrukcije, procesor proverava da li je nastao prekid (*interrupt signal*)
- ❖ Ukoliko je prekid nastao, procesor izvršava sledeće korake:
 - ❖ Postavlja programski brojač na početnu adresu rutine za obradu prekida (*interrupt handler*)
 - ❖ Prebacuje se iz **korisničkog** u **kernel režim**, tako da rutina za obradu prekida može uključiti i privilegovane instrukcije
 - ❖ **Kontekst** prekinutog procesa se čuva u njegovom PCB
 - Informacije o stanju procesora
 - ❖ Izvršava se **rutina za obradu prekida** (*interrupt handler*)
 - ❖ Ukoliko nastanak prekida **ne** zahteva zamenu (*komutiranje*) procesa i promenu stanja procesa, obnavljaju se informacije o stanju procesora (kontekst) prekinutog procesa
 - U suprotnom vrši se komutiranje procesa i promena njegovog stanja
 - ❖ Prekid kloka ili poziv U/I operacije uzrokuje **prebacivanje** prekinutog **procesa** u novo stanje (Spreman, Blokiran)



Promena stanja procesa

- ◆ Zamena procesa uzrokuje promenu njegovog stanja
- ◆ Koraci koje obavlja OS pri zamjeni procesa:
 1. Snimiti sadržaj registara procesora, programski brojač i statusni registar procesora u PCB procesa
 2. Ažurirati PCB procesa koji je do tada bio u stanju Izvršavanje (*Running*)
 3. Premestiti PCB u odgovarajući red – Spreman, Blokiran, Spreman/suspendovan i promeniti mu stanje
 4. Izabratи sledeći proces za izvršavanje
 5. Ažurirati PCB procesa koji je izabran u stanje Izvršavanje
 6. Ažurirati stukture podataka za upravljanje memorijom
 7. Obnoviti sadržaj registara procesora na vrednosti koje su postojale u trenutku kad je proces promenjen iz stanja Izvršavanje



Izvršavanje operativnog sistema

- ❖ OS radi na isti način kao i ostali računarski softver
- ❖ Ako je OS samo kolekcija programa i ako se ovi programi izvršavaju od strane procesora baš kao i bilo koji drugi programi, da li je OS proces?
- ❖ Ako jeste, kako se njime upravlja?
 - Ko (šta) upravlja njime?



Izvršenje OS-a

Kernel bez procesa

- Tradicionalni pristup u starijim OS
- Izvršenje kernela van bilo kog procesa
- OS kod se izvršava kao poseban entitet u glavnoj memoriji , sa svojim stekom, u kernel režimu (privilegovanom režimu)

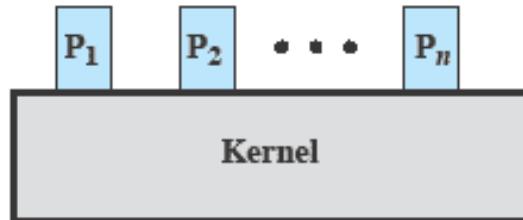
Izvršenje OS unutar korisničkih procesa

- Uobičajeno u OS na PC i radnim stanicama
- OS softver predstavlja kolekciju procedura koje korisnički proces može pozvati za izvršenje različitih funkcija i koje se izvršavaju unutar okruženja korisničkog procesa
- Slika procesa sadrži i delove za program, podatke i stek kernel programa
- Proces se izvršava u kernel režimu kada se izvršava kod OS-a

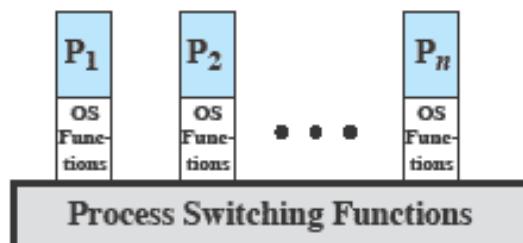
OS zasnovan na procesima

- OS implementiran kao kolekcija sistemskih procesa
- Korisno je u višeprocesorskom ili višeračunarskom okruženju

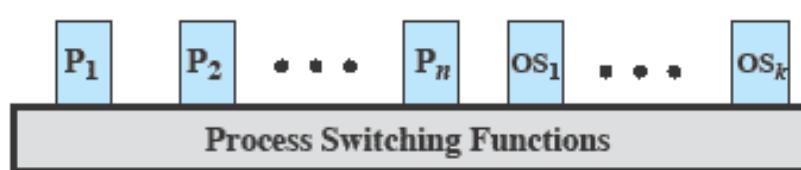
Odnos između OS i korisničkih procesa



(a) Separate kernel



(b) OS functions execute within user processes



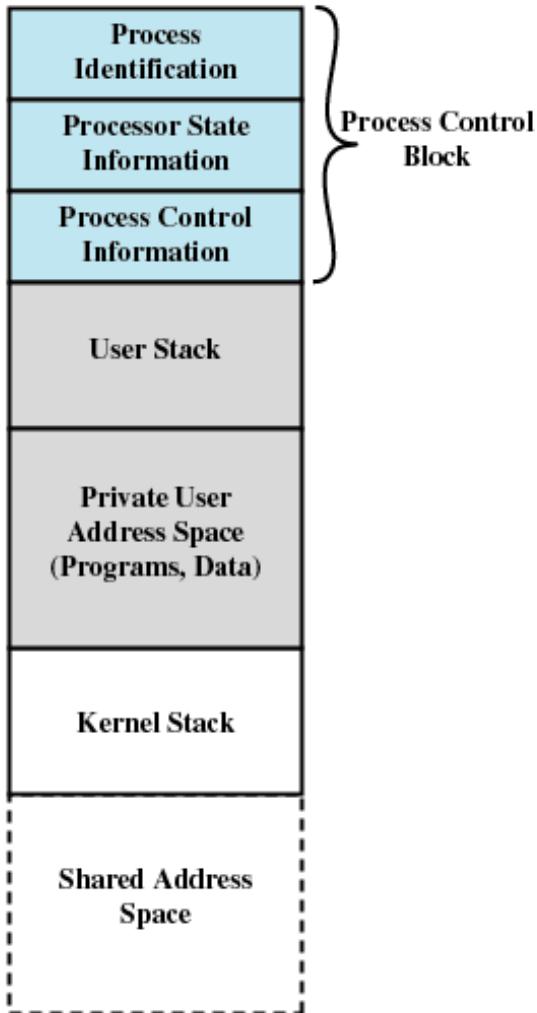
(c) OS functions execute as separate processes

- Zasebno jezgro

- Funkcije OS-a se izvršavaju unutar korisničkih procesa

- Funkcije OS-a se izvršavaju kao zasebni procesi

Slika procesa kada se OS izvršava unutar korisničkog prostora



- OS se izvršava unutar korisničkog procesa (adresnog prostora)
- Kernel stek se koristi za upravljanje pozivima procedura dok je proces u kernel režimu
- OS kod i podaci su u deljenom adresnom prostoru (deljeni su između svih korisničkih procesa)



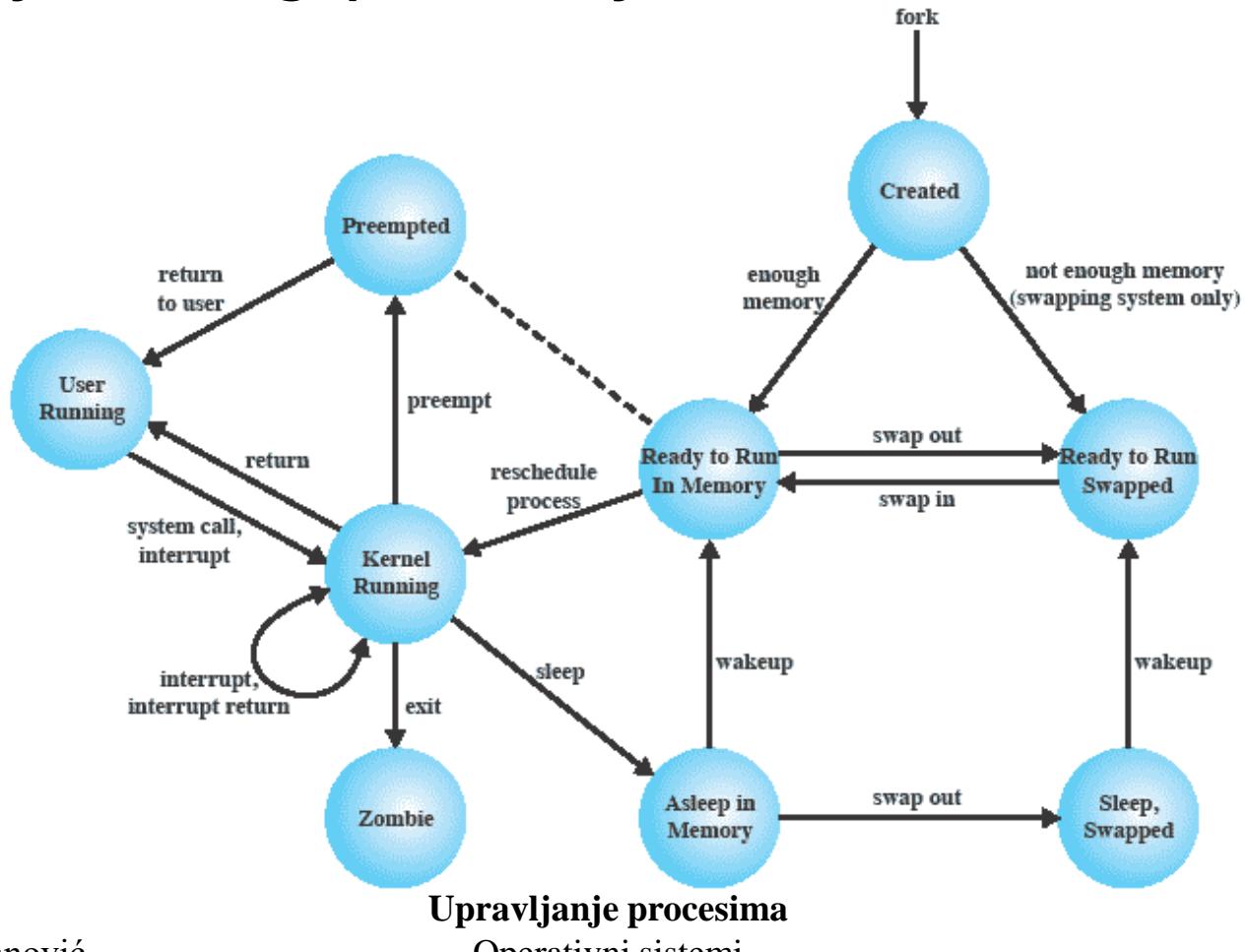
Unix SVR4

System V Release 4

- Veći deo operativnog sistema se izvršava u korisničkom procesu
- Sistemski procesi se izvršavaju isključivo u kernel režimu (modu)
- Korisnički procesi
 - Korisnički programi i funkcije se izvršavaju u korisničkom režimu.
 - U režimu kernela se izvršavaju funkcije koje pripadaju kernelu.

UNIX SVR4 - upravljanje procesima

- Stanja procesa i promene stanja (9)
- Dva stanja *Running* (izvršavanje u korisničkom i kernel režimu)





UNIX proces

- ❖ Proces u UNIX-u je kolekcija struktura podataka koje obezbeđuju OS-u sve informacije neophodne za upravljanje i raspoređivanje procesa.
- ❖ Slika procesa u UNIX-u sadrži sledeće elemente:
 - ❖ Kontekst korisničkog nivoa
 - Tekst programa, podaci, korisnički stek, deljena memorija
 - ❖ Kontekst registara
 - Programski brojač, statusni registar procesora, stek pointeri, registri opšte namene
 - ❖ Kontekst sistemskog nivoa
 - Ulaz (*entry*) u tabelu procesa – PCB procesa
 - U (*user*) oblast – neophodne kernelu kada se izvršava u kontekstu procesa
 - Tabela regiona – definiše mapiranje između virtuelnih i fizičkih adresa, kao i prava pristupa određenim regionima – koristi se od strane sistema za upravljanje memorijom
 - Kernel stek – za pozive/povratke iz kernel procedura



Kreiranje procesa

- Prednji (*foreground*) procesi
- Pozadinski (*background*) procesi – demoni, servisi
- Kako videti procese u sistemu?
 - Unix komanda [ps](#)
 - Windows - [TaskManager \(Ctrl+Alt+Del\)](#)
- Kako kreirati proces?
 - Proces koji se izvršava poziva sistemski poziv za kreiranje procesa
 - Unix (POSIX)
 - [fork](#) – proces kreira svoj klon; pravi se kopija slike procesa roditelja
 - [exec](#) (*execv*, *execve*, *execvp*, *execl*, *execle*, *execvp*) - definiše program koji će se izvršavati u novom procesu i okolinu novog procesa
 - Windows API
 - [CreateProcess](#) - kreira novi proces i puni ga novim programom
- Nakon kreiranja oba procesa imaju sopstvene slike u memoriji i adresne prostore



Sistemski pozivi za terminiranje

- Sistemski poziv kojim proces terminira sam sebe:
 - Unix (POSIX) - `exit`
 - Windows API - `ExitProcess`
- Sistemski poziv kojim jedan proces terminira drugi:
 - Unix (POSIX) - `kill`
 - Windows API - `TerminateProcess`



Sistemske pozive

- POSIX
- UNIX/Linux
- Windows API



Procesi u UNIX-u

Kreiranje procesa u UNIX-u

```
pid = fork( );           /* if the fork succeeds, pid > 0 in the parent */  
if (pid < 0) {  
    handle_error( );     /* fork failed (e.g., memory or some table is full) */  
} else if (pid > 0) {  
    /* parent code goes here. */  
}  
} else {  
    /* child code goes here. */  
}
```



Sistemski pozivi za upravljanje procesima u Unix-u

System call	Description
pid = fork()	Create a child process identical to the parent
pid = waitpid(pid, &statloc, opts)	Wait for a child to terminate
s = execve(name, argv, envp)	Replace a process' core image
exit(status)	Terminate process execution and return status
s = sigaction(sig, &act, &oldact)	Define action to take on signals
s = sigreturn(&context)	Return from a signal
s = sigprocmask(how, &set, &old)	Examine or change the signal mask
s = sigpending(set)	Get the set of blocked signals
s = sigsuspend(sigmask)	Replace the signal mask and suspend the process
s = kill(pid, sig)	Send a signal to a process
residual = alarm(seconds)	Set the alarm clock
s = pause()	Suspend the caller until the next signal

s je kod greške

pid je ID procesa

residual je preostalo vreme od prethodnog alarma



Sistemski pozivi za upravljanje poslovima, procesima, nitima i fiberima u Windows API

Windows API Function	Description
CreateProcess	Create a new process
CreateThread	Create a new thread in an existing process
CreateFiber	Create a new fiber
ExitProcess	Terminate current process and all its threads
ExitThread	Terminate this thread
ExitFiber	Terminate this fiber
SetPriorityClass	Set the priority class for a process
SetThreadPriority	Set the priority for one thread
CreateSemaphore	Create a new semaphore
CreateMutex	Create a new mutex
OpenSemaphore	Open an existing semaphore
OpenMutex	Open an existing mutex
WaitForSingleObject	Block on a single semaphore, mutex, etc.
WaitForMultipleObjects	Block on a set of objects whose handles are given
PulseEvent	Set an event to signaled then to nonsignaled
ReleaseMutex	Release a mutex to allow another thread to acquire it
ReleaseSemaphore	Increase the semaphore count by 1
EnterCriticalSection	Acquire the lock on a critical section
LeaveCriticalSection	Release the lock on a critical section

Upravljanje procesima

Operativni sistemi



Domaći zadatak

◆ The Linux Kernel archive & code reference

- <https://www.kernel.org/>
- <http://lxr.free-electrons.com/>
- <https://github.com/torvalds/linux>
 - /include/linux/sched.h

◆ FreeBSD Code reference

- <https://svnweb.freebsd.org/base/stable/>
- <https://github.com/freebsd/freebsd>
 - /sys/sys/proc.h

◆ Android

- <https://source.android.com/source/index.html>
- <http://androidxref.com/>



Operativni sistemi

Niti

Prof. dr Dragan Stojanović

Katedra za računarstvo
Univerzitet u Nišu, Elektronski fakultet



Literatura

- ❖ *Operating Systems: Internals and Design Principles*, edition, W. Stallings, Pearson Education Inc., 7th – 2012, (5th -2005, 6th - 2008, 8th – 2014 , 9th – 2017)
 - ❖ <http://williamstallings.com/OperatingSystems/>
 - ❖ <http://williamstallings.com/OperatingSystems/OS9e-Student/>
- ❖ Poglavlje 4: Niti



Procesi i niti

◆ Koncept procesa obuhvata dve karakteristike:

▣ **Vlasništvo nad resursom**

- Proces sadrži virtuelni adresni prostor u kome smešta sliku procesa
 - **Slika procesa:** program, podaci, stek, atributi definisani u Upravljačkom bloku procesa (PCB)
- Povremeno procesu može biti dodeljeno upravljanje ili vlasništvo nad resursima (glavna memorija, UI kanali, UI uređaji, datoteke)
- OS ima f-ju zaštite da ne bi došlo do neželjenog mešanja procesa

▣ **Raspoređivanje/izvršavanje**

- Izvršavanje procesa prati putanju izvršenja kroz jedan ili više programa
 - Ovo izvršavanje može biti isprepletano sa drugim procesima
- OS raspoređuje procese, **dodeljuje im resurse** i startuje izvršenje

▣ Ove dve karakteristike OS tretira nezavisno

Niti

Operativni sistemi

Niti (*threads*)

- ❖ Proces (*task*) predstavlja osnovnu jednicu za dodelu, vlasništvo i zaštitu u pristupu resursima
- ❖ Nit (*thread, lightweight process*) predstavlja entitet koji se raspoređuje i izvršava na CPU
- ❖ U tradicionalnom OS-u svaki proces ima sopstveni adresni prostor i **jednu nit izvršenja**
- ❖ U modernim OS-ima jedan proces može imati **više niti izvršenja** koje se izvode konkurentno ili paralelno
 - ▣ Sve niti pridružene jednom procesu dele program i resurse tog procesa
 - ▣ Svaka nit ima sopstveni magacin i stanje
- ❖ Nit omogućava podelu posla koji je dodeljen procesu na više niti izvršenja, tako da svaka nit preuzima deo tog posla

Niti

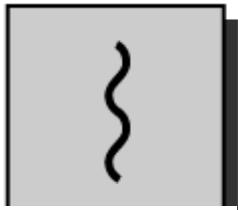


Višenitnost (*Multithreading*)

- **Višenitost** je sposobnost OS-a da podrži više niti izvršenja unutar jednog procesa
- Tradicionalni pristup – jedna nit po procesu – naziva se **pristup sa jednom niti** (*single threaded*)
- Niti u OS-ima:
 - ▣ MS-DOS podržava jednu nit
 - ▣ Tradicionalni UNIX podržava više procesa, ali samo jednu nit po procesu
 - ▣ JRE je primer jednog procesa sa više niti
 - ▣ Windows, Solaris, Linux, Mac OS, OS/2 podržavaju više niti

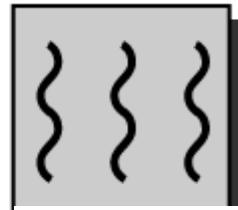
Niti i procesi

MS-DOS



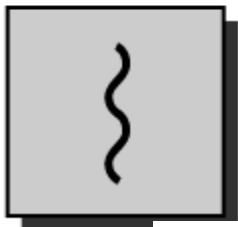
jedan proces
jedna nit

JRE

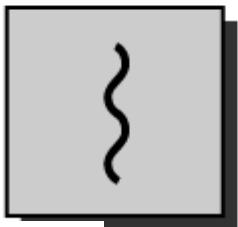


jedan proces
više niti

UNIX



više procesa
jedna nit



Windows
Linux
Solaris
Mac OS
OS/2

više procesa
više niti po procesu

{ = praćenje instrukcije

(Stallings, 2012)

Niti

Operativni sistemi



Proces

- ◆ Svakom procesu je dodeljen:

- Virtuelni adresni prostor gde se nalazi **slika procesa** – program, podaci, stek, kao i atributi definisani u Upravljačkom bloku procesa (PCB)
- Zaštićeni pristup procesoru, drugim procesima (IPC), datotekama i UI resursima (uredaji i kanali)

- ◆ Unutar procesa može postojati jedna ili više niti

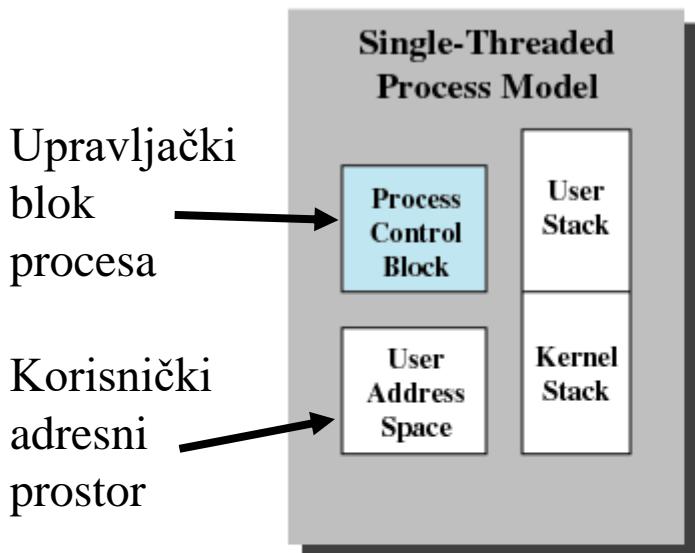
Nit

- ◆ Svaka nit poseduje:

- Stanje izvršenja niti (*Izvršava se, Spremna, itd*)
- Sačuvani kontekst niti kada se ne izvršava (programski brojač, registri procesora, stek pointeri)
- Stek (magacin) izvršavanja
- Statičku memoriju za lokalne promenljive niti
- Pristup memoriji i resursima svog procesa koje deli sa svim ostalim nitima procesa

Modeli jedno-nitnog i višenitnog procesa

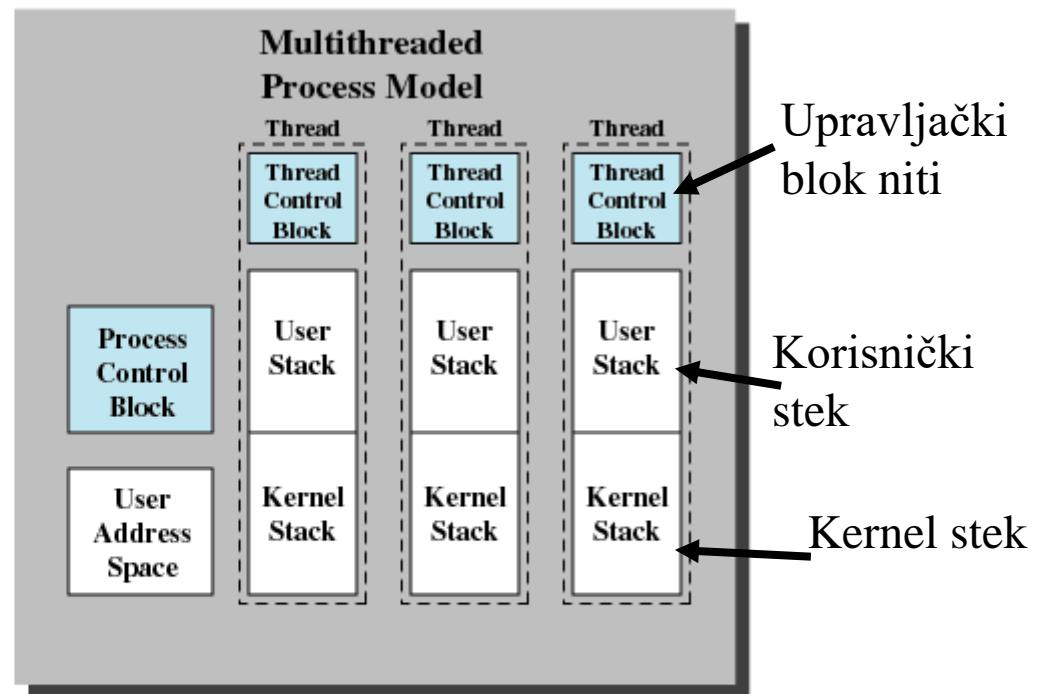
Model jednonitnog procesa



Upravljački
blok
procesa

Korisnički
adresni
prostor

Model višenitnog procesa



Upravljački
blok niti

Korisnički
stek

Kernel stek

Figure 4.2 Single Threaded and Multithreaded Process Models

Koristi od niti

◆ Osnovne koristi niti se zasnivaju na dobitku u performansama

1. Potrebno je mnogo manje vremena da bi se kreirala nova nit u postojećem procesu, nego da se kreira novi proces.
2. Potrebno je mnogo manje vremena za prekid niti nego prekid procesa.
3. Potrebno je mnogo manje vremena za prebacivanje (*switch*) izmedju dve niti unutar istog procesa nego za komutiranje procesa
4. Niti mogu komunicirati međusobno jer dele adresni prostor procesa i otvorene datoteke
 - Bez poziva OS kernela

Niti

Operativni sistemi



Korišćenje niti u jednokorisničkom multiprogramskom sistemu

- Čeoni (*foreground*) i pozadinski (*background*) poslovi
 - Program za tabelarna izračunavanja – jedna nit prikazuje GUI i prihvata korisnički unos, dok druga nit izvršava korisničke komande i ažurira tabele
- Asinhrona obrada
 - *Autosave* u programu za obradu teksta
- Brzina izvršavanja
 - Više niti može da se izvršava konkurentno/paralelno, dok je jedna niti blokirana na U/I druga može da se izvršava
- Modularna struktura programa



Niti

- Neke akcije u okviru OS odnose se na sve niti jednog procesa
 - OS mora da upravlja ovim akcijama na nivou procesa.
- Primeri:
 - Suspendovanje procesa uključuje suspendovanje svih niti procesa
 - Terminiranje procesa terminira sve niti u okviru procesa
- Slično procesima
 - Niti poseduju stanja izvršenja i mogu se sinhronizovati međusobno

Stanja niti

◆ Osnovna stanja niti:

- Izvršava se (*Running*)
- Spremna (*Ready*)
- Blokirana (*Blocked*)

◆ Prelazi između stanja su isti kao kod procesa

◆ Ne postoje suspendovana stanja jer sve niti unutar istog procesa dele isti adresni prostor

- Suspendovanje procesa povlači suspendovanje svih niti tog procesa

◆ Terminiranjem procesa, terminiraju se sve niti tog procesa



Operacije nad nitima

❖ Kreiranje (umnožavanje, *spawn*)

- ❖ Kreiranjem procesa kreira se nit tog procesa. Svaka nit procesa može kreirati novu nit u okviru istog procesa, obezbeđujući pointer na funkciju (instrukcije) i argumente nove niti. Nova nit dobija sopstveni kontekst (TCB), magacine i smešta se u red spremnih

❖ Blokiranje

- ❖ Kada nit čeka na događaj (npr. U/I) blokira se pri čemu se čuvaju vrednosti korisničkih registara, programskog brojača i stek pointera, i procesor započinje izvršavanje sledeće spremne niti u istom ili drugom procesu

❖ Deblokiranje

- ❖ Kada nastane događaj koji je čekala, nit se prevodi u stanje *spremna*

❖ Završetak

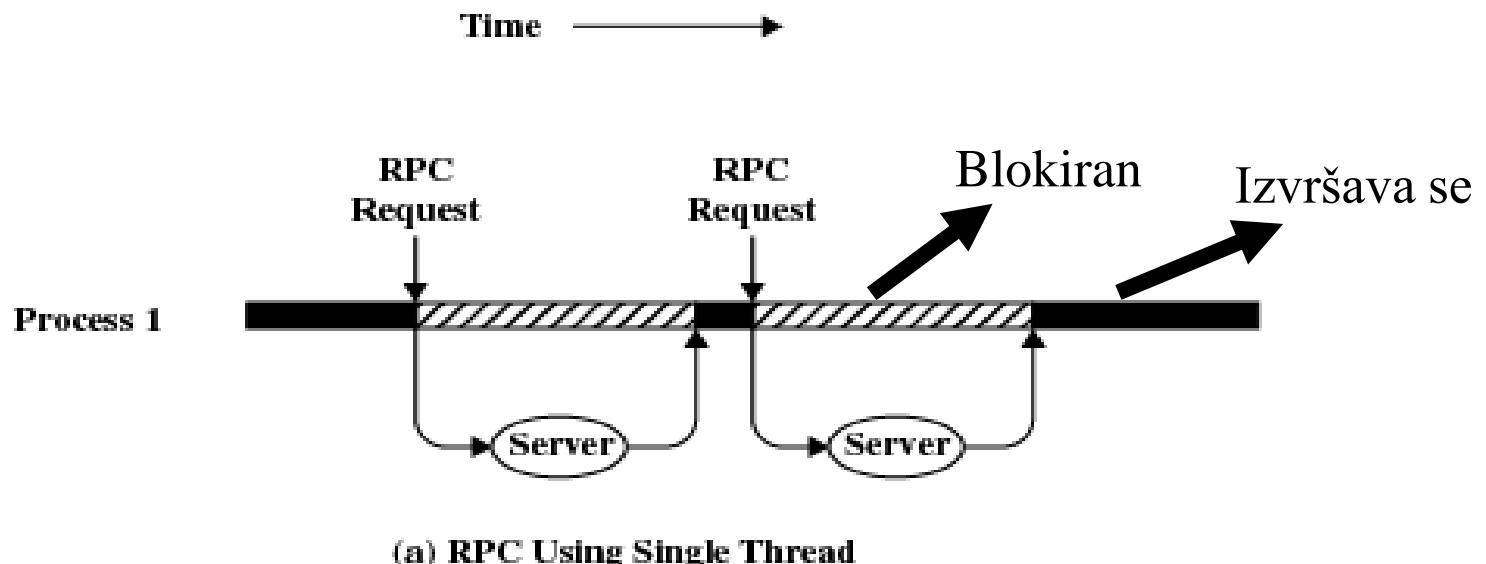
- ❖ Kad se niti završi, dealociraju se TCB i magacini

Niti

Operativni sistemi

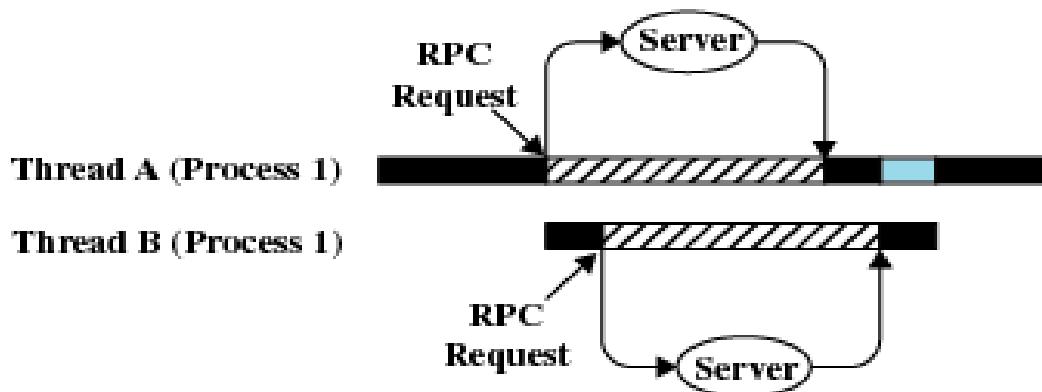
Primer: RPC korišćenjem jedne niti

- RPC je tehnika kojom dva procesa koja se izvršavaju na različitim računarima međusobno komuniciraju koristeći mehanizam poziva procedure
- Program vrši dva poziva udaljenih procedura (*Remote Procedure Call* - RPC) na dva različita računara



RPC korišćenjem više niti

- Program vrši dva poziva udaljenih procedure (RPC) na dva različita računara, i za svaki poziv koristi posebnu nit
- Ostvareno je poboljšanje u performansama

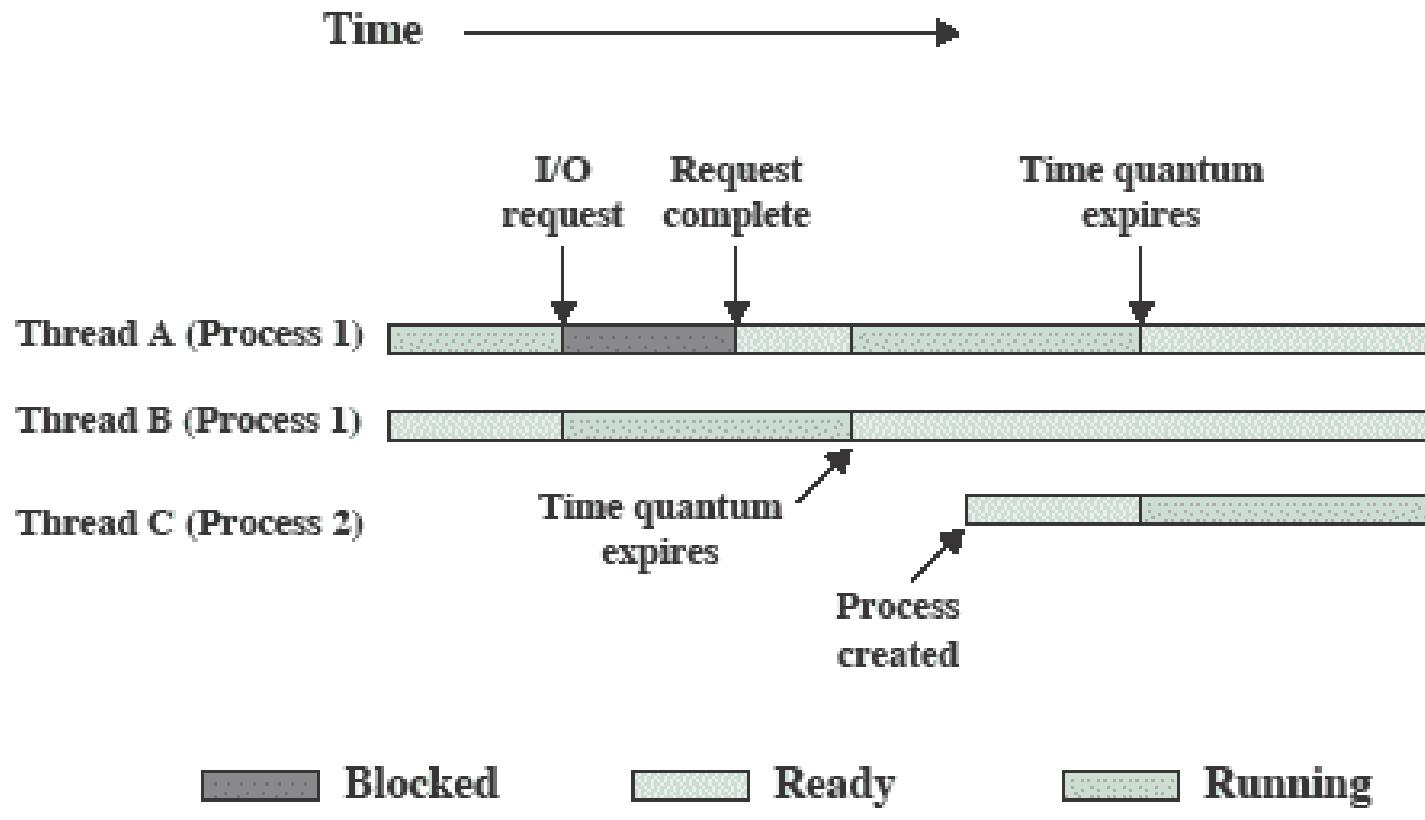


(b) RPC Using One Thread per Server (on a uniprocessor)

- **Blocked, waiting for response to RPC**
- **Blocked, waiting for processor, which is in use by Thread B**
- **Running**

Višenitnost na jednoprocесорском раčunару

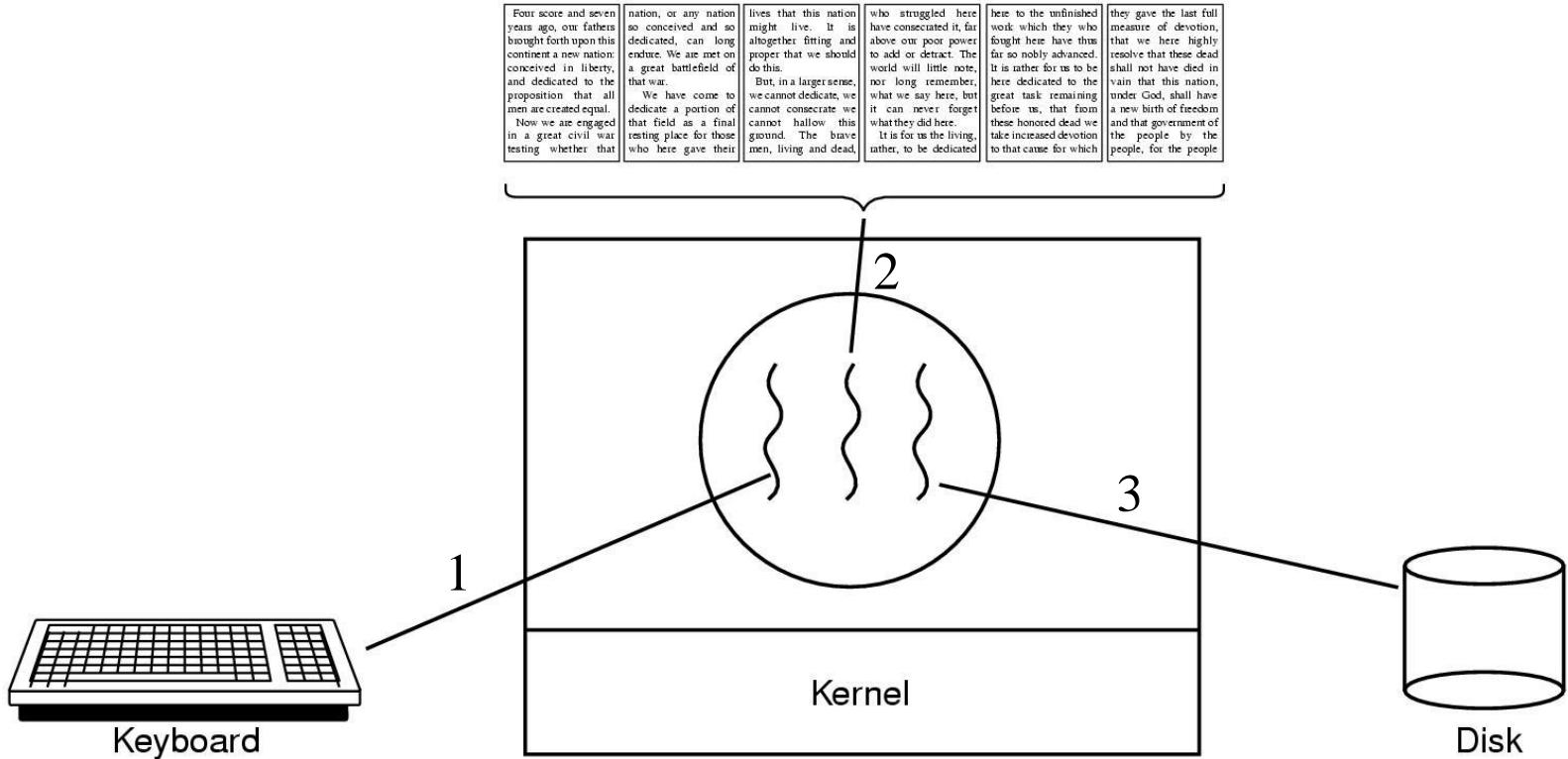
- Primer izvršenja više niti na jednoprocесорском
раčunару



Izvršavanje i sinhronizacija niti

- ❖ U **jednoprocesorskom** sistemu niti jednog procesa se izvršavaju konkurentno, slično procesima kod multiprogramiranja
- ❖ U **višeprocesorskom** sistemu niti jednog procesa se mogu izvršavati paralelno - na različitim procesorima
- ❖ Niti jednog procesa **nisu potpuno nezavisne**, jer se izvršavaju u jednom adresnom prostoru, što znači da dele globalne promenljive i otvorene datoteke
 - ▣ Svaka promena memorijskog resursa jedne niti utiče na druge niti koje koriste iste resurse
 - ▣ Neophodno je obezbediti sinhronizaciju više niti u pristupu istim memorijskim resursima
 - ▣ Tehnike za sinhronizaciju niti su identične tehnikama za sinhronizaciju procesa

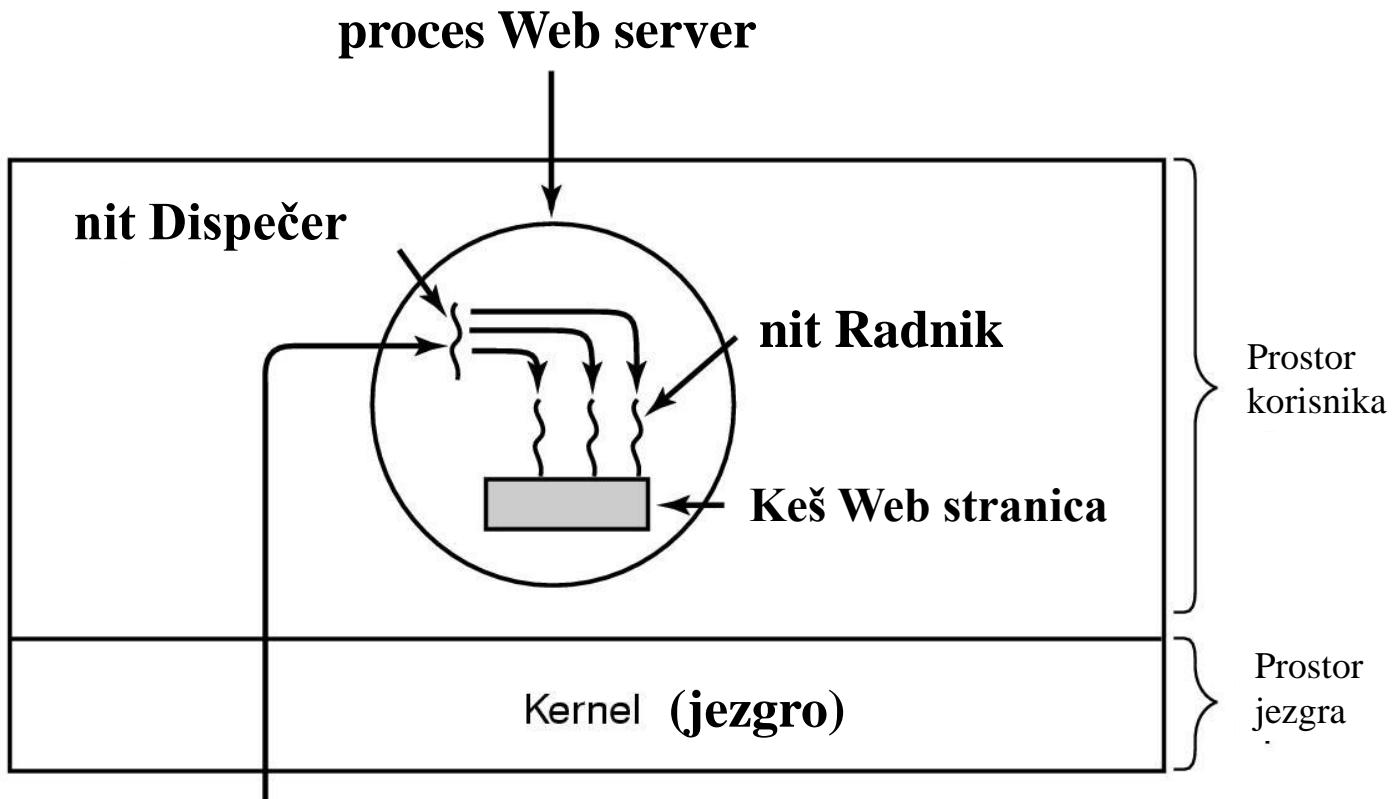
Primer 1: Višenitni Word procesor



Tanenbaum, 2014

Word processor sa tri niti: 1- interakcija sa korisnikom; 2- reformatiranje dokumenta; 3 - backup dokumenta

Primer 2: Višenitni Web server



Mrežna konekcija

Tanenbaum, 2014



Tipovi niti

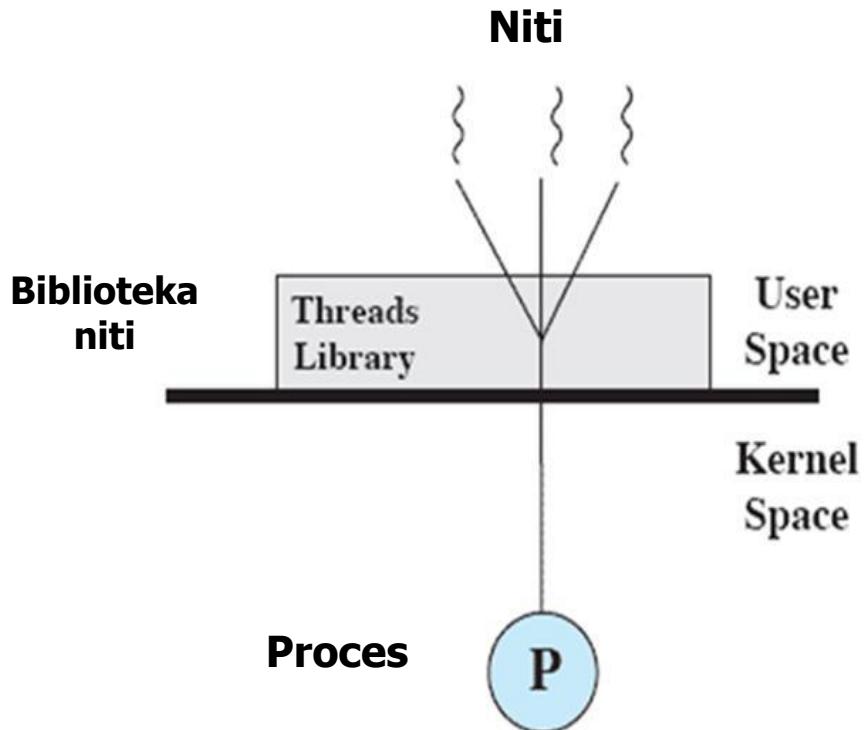
- ◆ Postoje dve kategorije implementacije niti:
 - Niti nivoa korisnika (**ULT** - *User Level Thread*)
 - Niti nivoa jezgra (**KLT** - *Kernel Level Thread*)
 - Kernelom podržane niti, *lightweight* procesi



Niti nivoa korisnika - ULT

- Kernel OS nije svestan postojanja niti
- Nitima upravlja aplikacioni program koristeći **biblioteku niti – skup funkcija za upravljanje nitima**
- Promena (komutiranje, *switching*) niti ne zahteva privilegije kernel moda, tj. nema promene moda
- Raspoređivanje niti je specifično za aplikaciju i obavlja se u okviru lokalne procedure u biblioteci niti, dakle **bez promene moda** (korisnički → kernel)
- Svaki proces može imati **sopstveni algoritam planiranja niti**
- Korišćenjem biblioteke niti bilo koja aplikacija se može implementirati kao višenitna

Niti nivoa korisnika - ULT



Podrška za niti nivoa korisnika

Biblioteka niti sadrži kod za:

- Kreiranje niti,
- Brisanje niti,
- Prenos poruka i podataka između niti,
- Raspoređivanje izvršenja niti,
- Pamćenje i restauriranje konteksta niti,...

Kontrolni blok niti (TCB) sadrži:

- Programski brojač,
- Pokazivač magacina,
- Registre procesora,
- Stanje,
- Prioritet,...

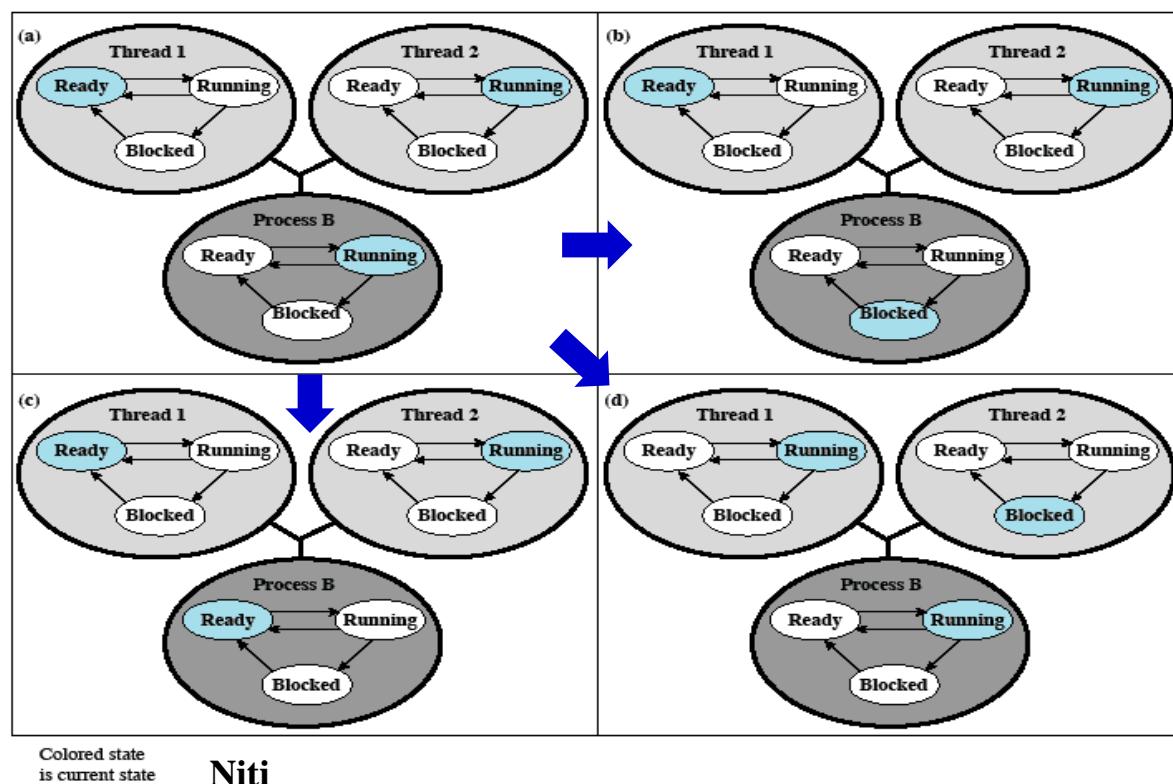


Niti nivoa korisnika - aktivnosti kernela

- Kernel ne vodi računa o aktivnostima niti, ali još uvek upravlja aktivnostima procesa
- Kada nit pozove neki sistemski poziv (pozove funkciju OS-a), ceo proces se blokira, ali za biblioteku niti ta nit je još uvek u stanju izvršenja
- Stanja niti su nezavisna od stanja procesa

Stanja ULT niti i procesa

- Proces B je u stanju *Izvršavanje* i izvršava se Nit2 (a)
- Kada se nit blokira, blokira se proces, dok je za biblioteku niti Nit2 još uvek u stanju *Izvršavanje* (b)
- Ukoliko istekne vremenski kvant proces je *Spreman*, a Nit2 i dalje u stanju *Izvršavanje* (c)
- Nit2 se *blokira* kada zahteva neku akciju od Niti1, koja prelazi u stanje *Izvršavanje*, dok je proces sve vreme u stanju *Izvršavanje* (d)





ULT - Prednosti i nedostaci

◆ Prednosti

- Promena sa jedne na drugu nit ne zahteva privilegije kernel moda, nije potrebno prebacivanje moda (korisnički-kernel, kernel-korisnički), pa se brže odvija
- Raspoređivanje može biti specifičan za proces; bira se najbolji algoritam
- ULT se može izvoditi nad bilo kojim OS-om. Jedino je potrebna biblioteka niti koja se može implementirati u OS-u koji ne podržava niti

◆ Nedostaci

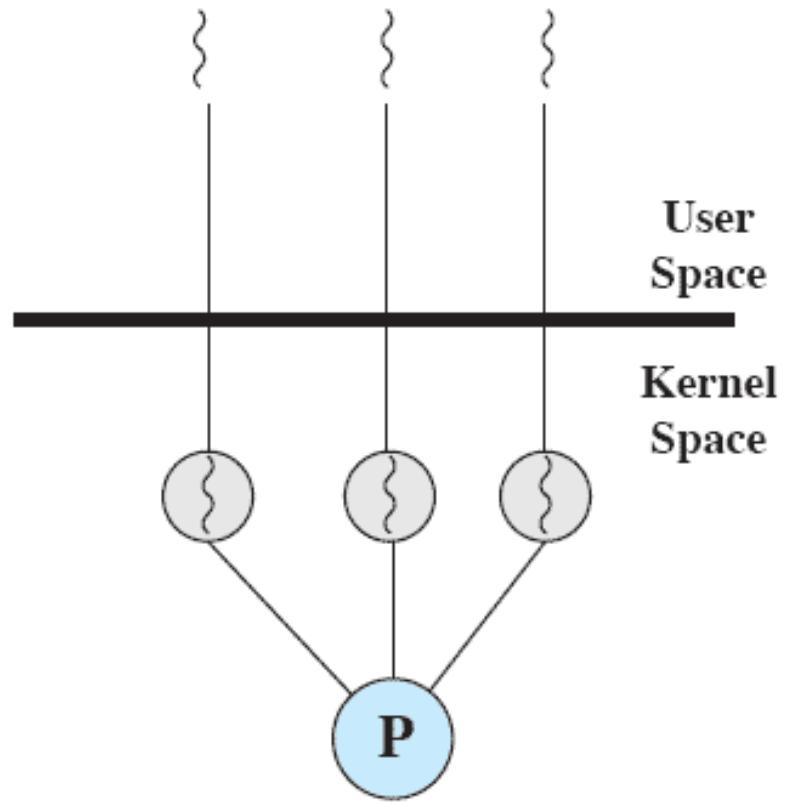
- Mnogi sistemski pozivi su blokirajući i kernel blokira procese. Ukoliko jedna nit pozove blokirajući sistemski poziv sve niti unutar tog procesa biće blokirane
 - Rešenje: korišćenje tehnike **omotavanja** (*jacketing*)
- Kernel može dodeljivati procesore jedino procesima
 - Niti istog procesa se ne mogu izvršavati istovremeno na različitim procesorima

Niti

Operativni sistemi

Niti nivoa kernela - KLT

- Kernel upravlja nitima
- Nema biblioteke niti, ali postoji API za rad sa nitima
- Kernel održava konteksne informacije za procese i niti
- Kernel vrši planiranje i raspoređivanje niti
- Kernel vrši promenu niti
- Primeri: Windows, Linux, Mac OS X



(b) Pure kernel-level

Niti

Operativni sistemi



KLT - Prednosti i nedostaci

◆ Prednosti

- Kernel može simultano planirati više niti istog procesa na više procesora
- Blokiranje se vrši na nivou niti - kada se nit blokira, jezgro bira spremnu nit istog ili drugog procesa
- Programi kernela mogu biti višenitni

◆ Nedostaci

- Promena niti unutar istog procesa zahteva promenu moda (kernel mod) i angažovanje OS
- Upravljanje nitima nivoa kernela sporije nego kod ULT-a

Niti

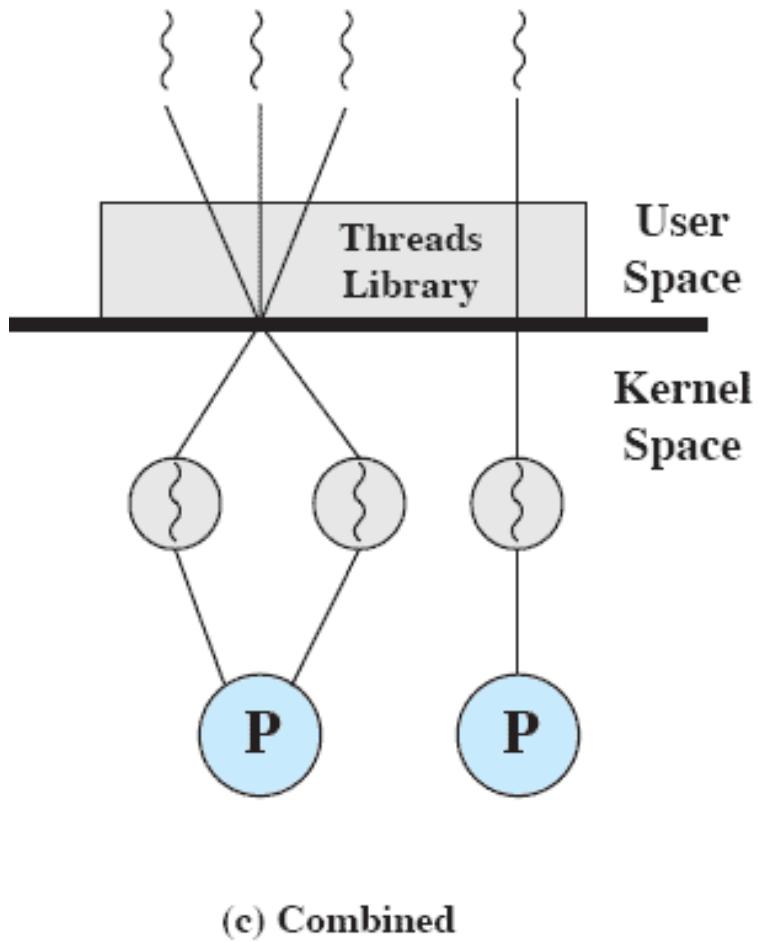
Operativni sistemi

ULT vs. KLT vs. Procesi

- ❖ Trajanje operacija nad nitima (ULT i KLT) i procesima
 - ❖ Kreiranje prazne niti/procesa
 - ❖ Sincronizacija niti/procesa zasnovana na signalima

Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal Wait	37	441	1,840

Hibridna implementacija



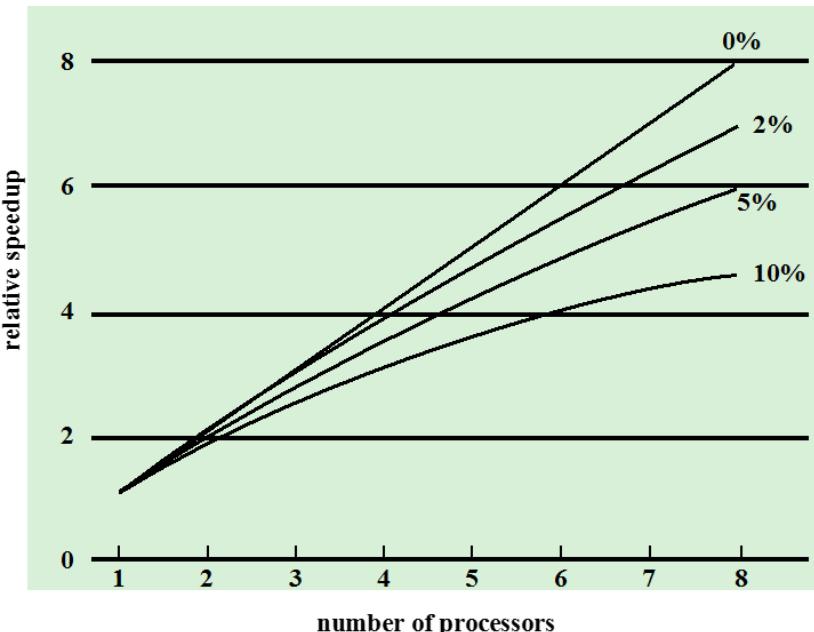
- Kreiranje niti obavlja se u prostoru korisnika
- Raspoređivanje i sinhronizacija niti se obavlja uglavnom na nivou aplikacije u prostoru korisnika
- Više ULT-a iz jednog procesa preslikavaju se u manji, ili jednak broj KLT-ova
- Programer može podešavati broj KLT-a za određenu aplikaciju
- Kombinuje dobre osobine ULT i KLT i minimizuje nedostatke
- Primer: Solaris

Performanse softvera na više jezgara (iz Stallings, 2012 - 7th edition)

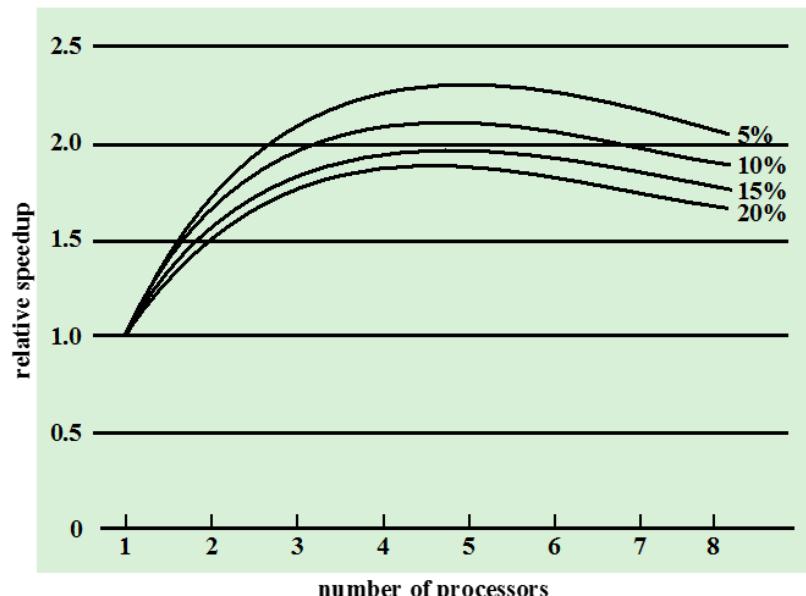
● Amdahl-ov zakon

- f – procenat koda koji se može paralelizovati na N procesora

$$\text{Speedup} = \frac{\text{time to execute program on a single processor}}{\text{time to execute program on } N \text{ parallel processors}} = \frac{1}{(1 - f) + \frac{f}{N}}$$



(a) Speedup with 0%, 2%, 5%, and 10% sequential portions



(b) Speedup with overheads



Niti u savremenim OS

- ◉ **Windows** – knjiga Stallings (Tanenbaum), praktikum
- ◉ **Solaris** – knjiga Stallings, praktikum
- ◉ **Linux/Unix** – knjiga Stallings, praktikum
 - ▣ POSIX - praktikum



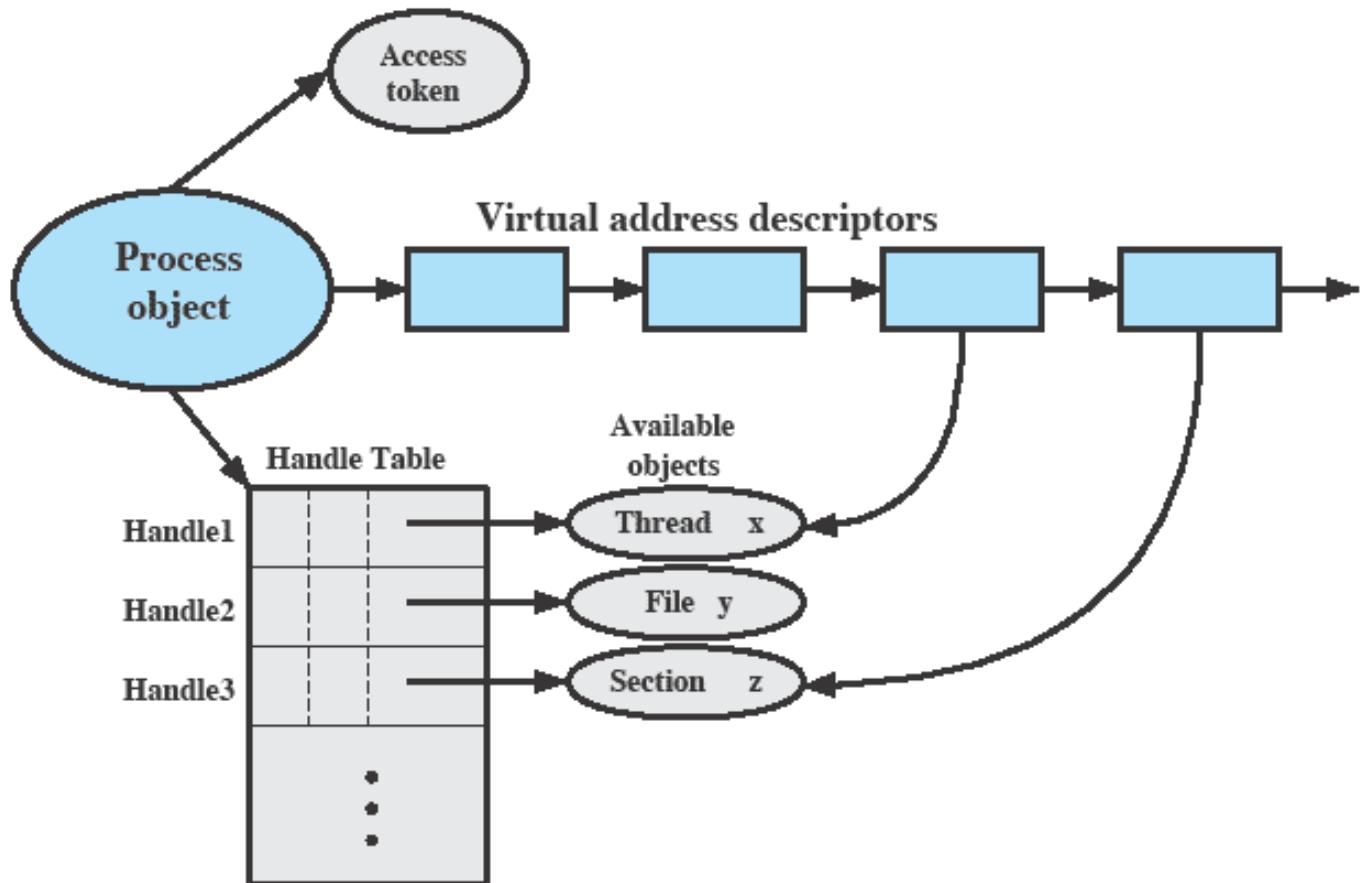
Windows procesi i niti

- ❖ Osnovni koncepti koji se koriste za upravljanje procesorom i resursima

Name	Description
Job	Collection of processes that share quotas and limits
Process	Container for holding resources
Thread	Entity scheduled by the kernel
Fiber	Lightweight thread managed entirely in user space

Veza između procesa i resursa

- Windows procesi i njihovi resursi

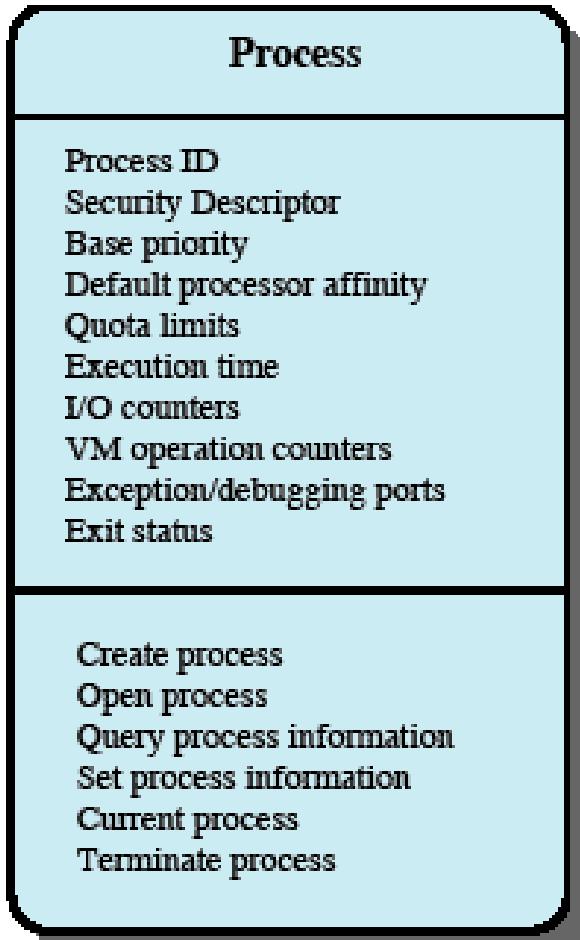


Niti

Operativni sistemi

Windows Proces i Thread objekti

Object Type



Object Body Attributes

Object Type

Object Body Attributes

Services

Thread

- Thread ID
- Thread context
- Dynamic priority
- Base priority
- Thread processor affinity
- Thread execution time
- Alert status
- Suspension count
- Impersonation token
- Termination port
- Thread exit status

- Create thread
- Open thread
- Query thread information
- Set thread information
- Current thread
- Terminate thread
- Get context
- Set context
- Suspend
- Resume
- Alert thread
- Test thread alert
- Register termination port

(b) Thread object

Niti

Operativni sistemi

Stanja Windows niti

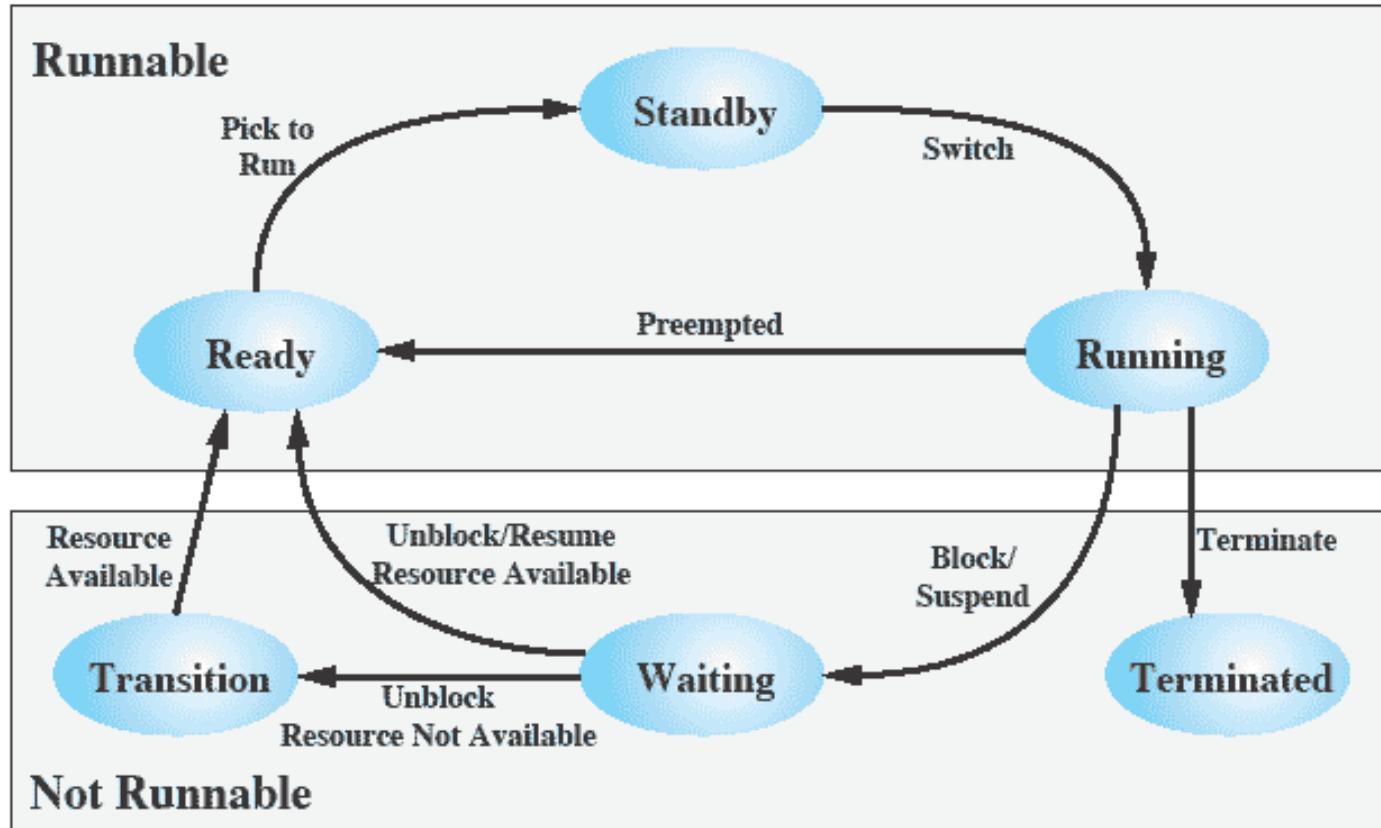


Figure 4.14 Windows Thread States



Windows API sistemski pozivi

Win32 API Function	Description
CreateProcess	Create a new process
→ CreateThread	Create a new thread in an existing process
CreateFiber	Create a new fiber
ExitProcess	Terminate current process and all its threads
→ ExitThread	Terminate this thread
ExitFiber	Terminate this fiber
SetPriorityClass	Set the priority class for a process
→ SetThreadPriority	Set the priority for one thread
CreateSemaphore	Create a new semaphore
CreateMutex	Create a new mutex
OpenSemaphore	Open an existing semaphore
OpenMutex	Open an existing mutex
WaitForSingleObject	Block on a single semaphore, mutex, etc.
WaitForMultipleObjects	Block on a set of objects whose handles are given
PulseEvent	Set an event to signaled then to nonsignaled
ReleaseMutex	Release a mutex to allow another thread to acquire it
ReleaseSemaphore	Increase the semaphore count by 1
EnterCriticalSection	Acquire the lock on a critical section
LeaveCriticalSection	Release the lock on a critical section

Neki Win32 pozivi za upravljanje procesima, nitima i fiberima

Niti

Solaris procesi i niti

- ➊ Solaris koristi 4 posebna koncepta vezana za niti
 - **Proces:** uključuje korisnički adresni prostor, stek i upravljački blok procesa
 - **Niti na korisničkom nivou:** Implementirane korišćenjem bilioteke niti u adresnom prostoru procesa; nevidljive za OS
 - **Procesi lake kategorije (*Lightweight*):** obezbeđuju mapiranje između ULT i kernel niti. Svaki LWP podržava više ULT koje mapira u jednu kernel nit.
 - **Kernel niti:** osnovni entiteti koji mogu biti raspoređeni i izvršeni na procesoru

Solaris višenitna arhitektura - primer

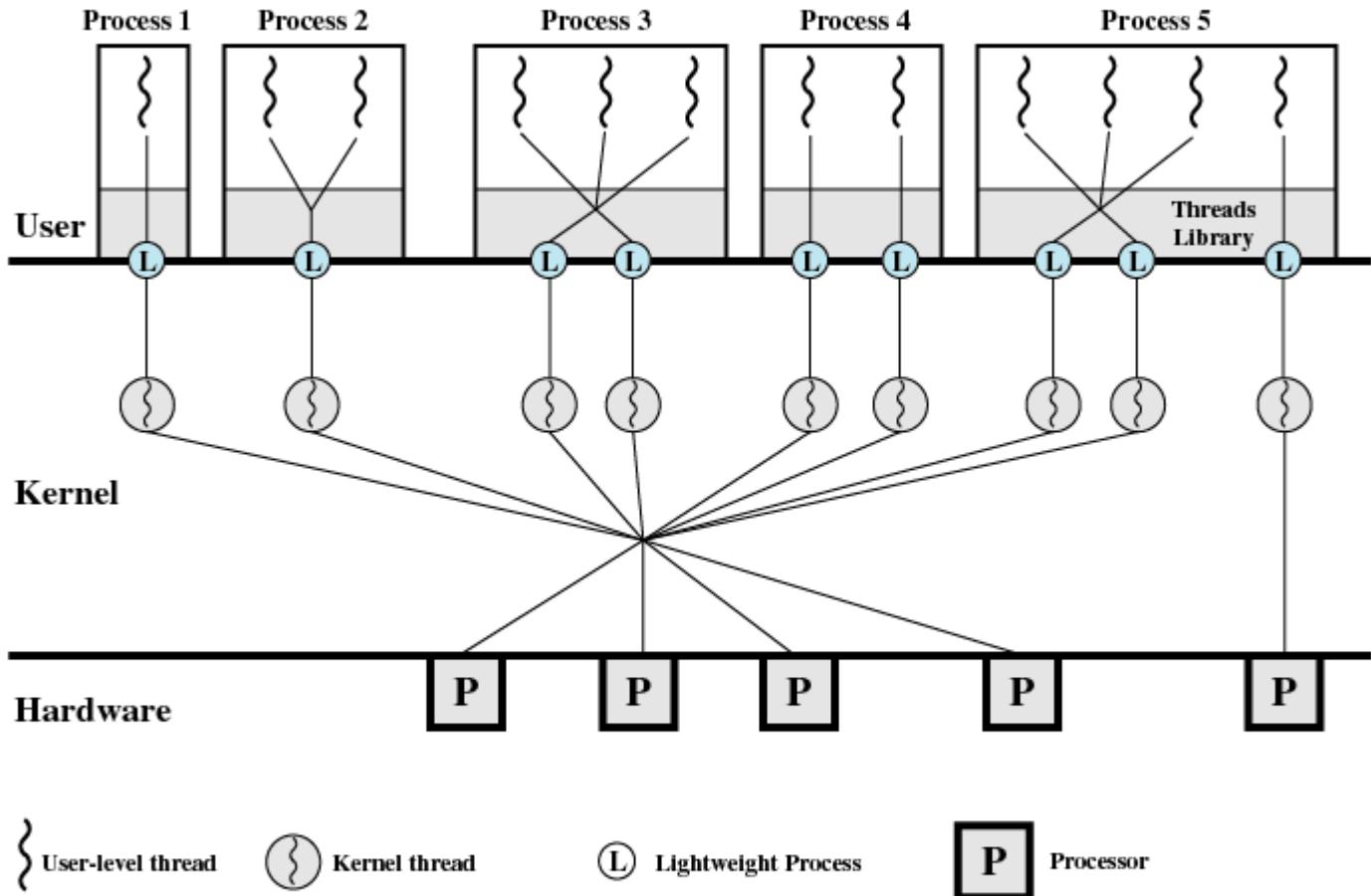


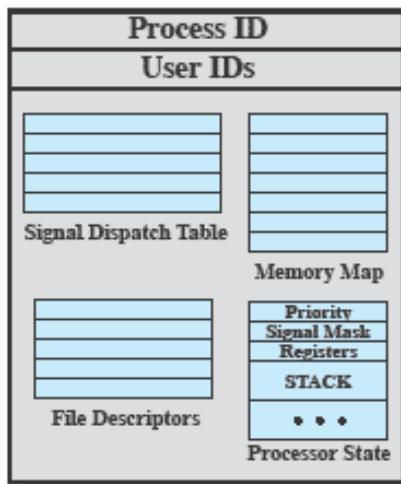
Figure 4.15 Solaris Multithreaded Architecture Example

Niti

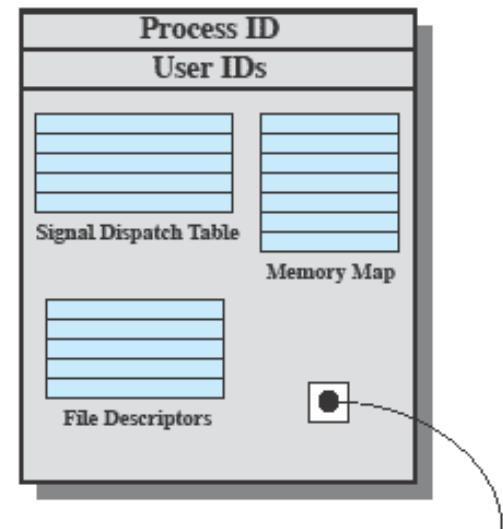
Operativni sistemi

Tradicionalni UNIX - Solaris

UNIX Process Structure



Solaris Process Structure



Solaris zamenjuje
blok sa stanjem
procesora listom sa
LWP-ima

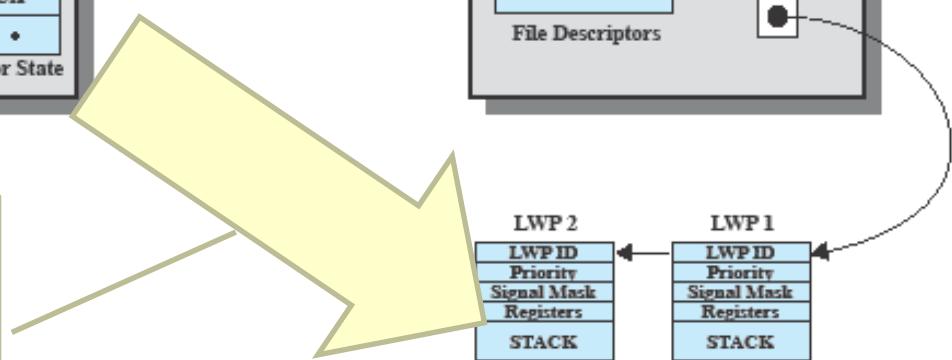


Figure 4.16 Process Structure in Traditional UNIX and Solaris [LEWI96]



UNIX/Linux niti (POSIX)

Thread call	Description
pthread_create	Create a new thread in the caller's address space
pthread_exit	Terminate the calling thread
pthread_join	Wait for a thread to terminate
pthread_mutex_init	Create a new mutex
pthread_mutex_destroy	Destroy a mutex
pthread_mutex_lock	Lock a mutex
pthread_mutex_unlock	Unlock a mutex
pthread_cond_init	Create a condition variable
pthread_cond_destroy	Destroy a condition variable
pthread_cond_wait	Wait on a condition variable
pthread_cond_signal	Release one thread waiting on a condition variable

Niti

Operativni sistemi



Kreiranje niti (POSIX primer)

```
#include <stdio.h>
→#include <pthread.h>

int glob_data = 5 ;

void *kidfunc(void *p) {
    printf ("Nova nit. Globalni podatak je: %d.\n", glob_data) ;
    printf ("Nova nit. ID procesa je: %d.\n", getpid()) ;
    glob_data = 15 ;
    printf ("Ponovo nova nit. Globalni podatak je: %d.\n", glob_data) ;
}

main ( ) {
→pthread_t kid ;
→pthread_create (&kid, NULL, kidfunc, NULL) ;
    printf ("Osnovna nit. Globalni podatak je: %d\n", glob_data) ;
    printf ("Osnovna nit. ID procesa je: %d.\n", getpid()) ;
    glob_data = 10 ;
→pthread_join (kid, NULL) ;
    printf ("Kraj programa. Globalni podatak je: %d\n", glob_data) ;
}
```

Linux proces/task

- ➊ Proces ili *task*, u Linux-u je predstavljen *task_struct* strukturom podataka (PCB)
- ➋ Ova struktura sadrži informacije svrstane u sledeće kategorije:
 - ▣ Stanje
 - ▣ Informacije za raspoređivanje (prioriteti)
 - ▣ Identifikatore
 - ▣ Interprocesnu komunikaciju
 - ▣ Linkove na druge *task_struct*
 - ▣ *File* sistem
 - ▣ Memorijski adresni prostor
 - ▣ Kontekst procesa
- ⌋ Nove verzije Linux, kao i Unix obezbeđuju KLT
 - ▣ Linux koristi specifično rešenje; ne pravi razliku između procesa i niti, već koristi *lightweight* procese, slično Solaris-u

Niti

Operativni sistemi

Stanja Linux procesa/niti

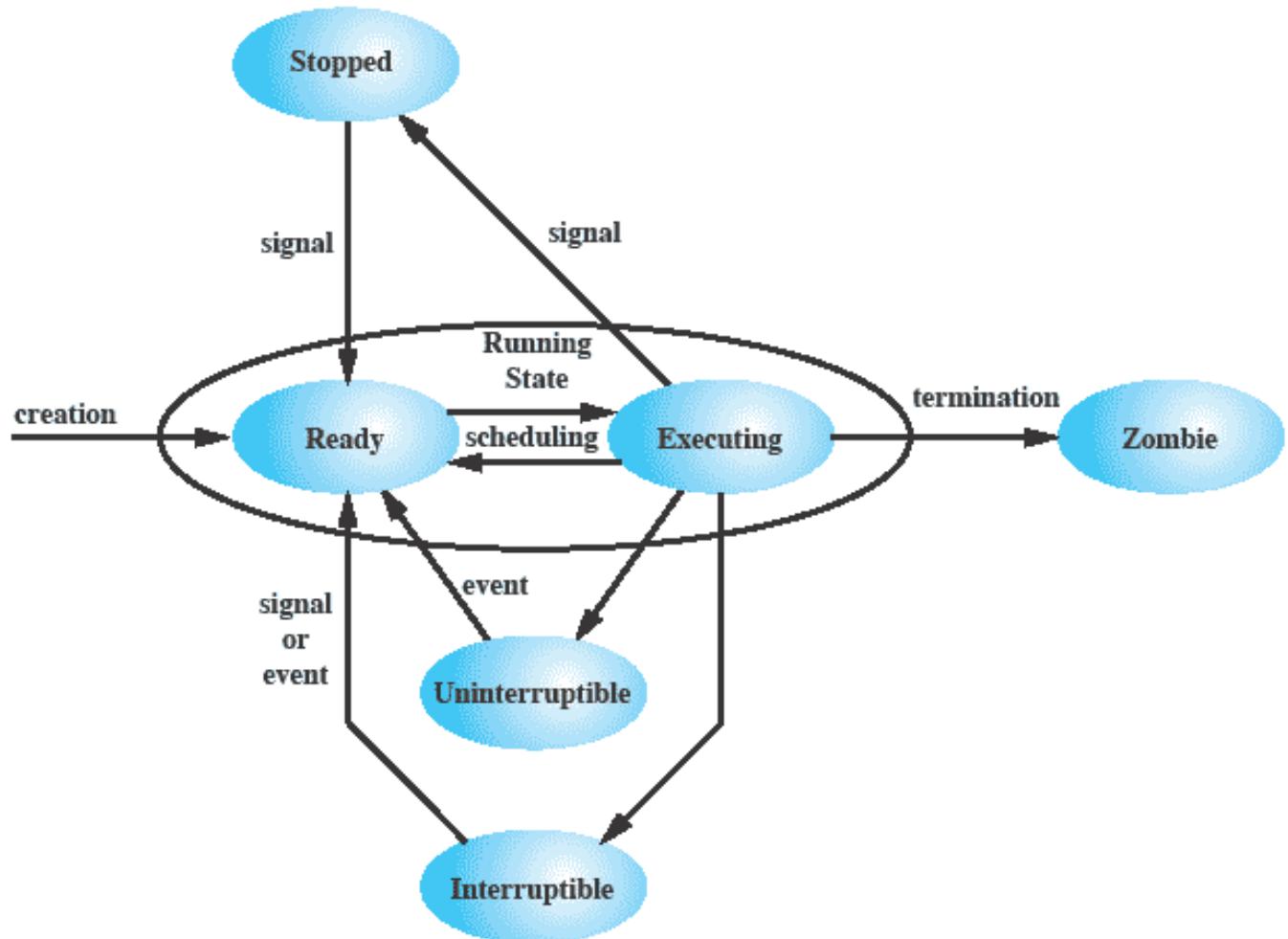


Figure 4.18 Linux Process/Thread Model

Niti

Operativni sistemi



Domaći zadatak

◆ Poglavlje 3 Opis procesa i upravljanje

- ❖ 4.10 Ključni pojmovi, kontrolna pitanja i problemi

◆ Poglavlje 4 Niti

- ❖ 4.10 Ključni pojmovi, kontrolna pitanja i problemi



Operativni sistemi

Konkurentnost: uzajamno
isključivanje i sinhronizacija

Prof. dr Dragan Stojanović

Katedra za računarstvo
Univerzitet u Nišu, Elektronski fakultet



Literatura

- ❖ *Operating Systems: Internals and Design Principles*, edition, W. Stallings, Pearson Education Inc., 7th – 2012, (5th -2005, 6th - 2008, 8th – 2014 , 9th – 2017)
 - ❖ <http://williamstallings.com/OperatingSystems/>
 - ❖ <http://williamstallings.com/OperatingSystems/OS9e-Student/>
- ❖ **Poglavlje 5:** Konkurentnost: uzajamno isključivanje i sinhronizacija
- ❖ **Dodatak A:** Teme o konkurentnosti

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi



OS i konkurentnost

- Centralna uloga OS je upravljanje procesima i nitima
 - Multiprogramiranje - Upravljanje više procesa unutar jednoprocesorskog sistema
 - Multiprocesiranje - Upravljanje više procesa unutar multiprocesora
 - Distribuirano procesiranje - Upravljanje više procesa koji se izvršavaju na distribuiranim računarskim sistemima
- Konkurentnost predstavlja značajni element u dizajnu OS, u okviru:
 - Dodela procesorskog vremena procesima
 - Podela resursa i nadmetanje za resursima
 - Synchronizacije izvršavanja više procesa
 - Komunikacije između procesa

Konkurentnost: uzajamno isključivanje i synchronizacija

Operativni sistemi



Kada se javlja konkurentnost?

Konkurentnost se javlja u tri različita konteksta:

- ❖ Višestruke aplikacije

- ❖ Multiprogramiranje obezbeđuje deljenje vremena obrade između aktivnih aplikacija

- ❖ Struktuirane aplikacije

- ❖ Aplikacije mogu biti programirane kao skup konkurentnih procesa

- ❖ Struktura operativnog sistema

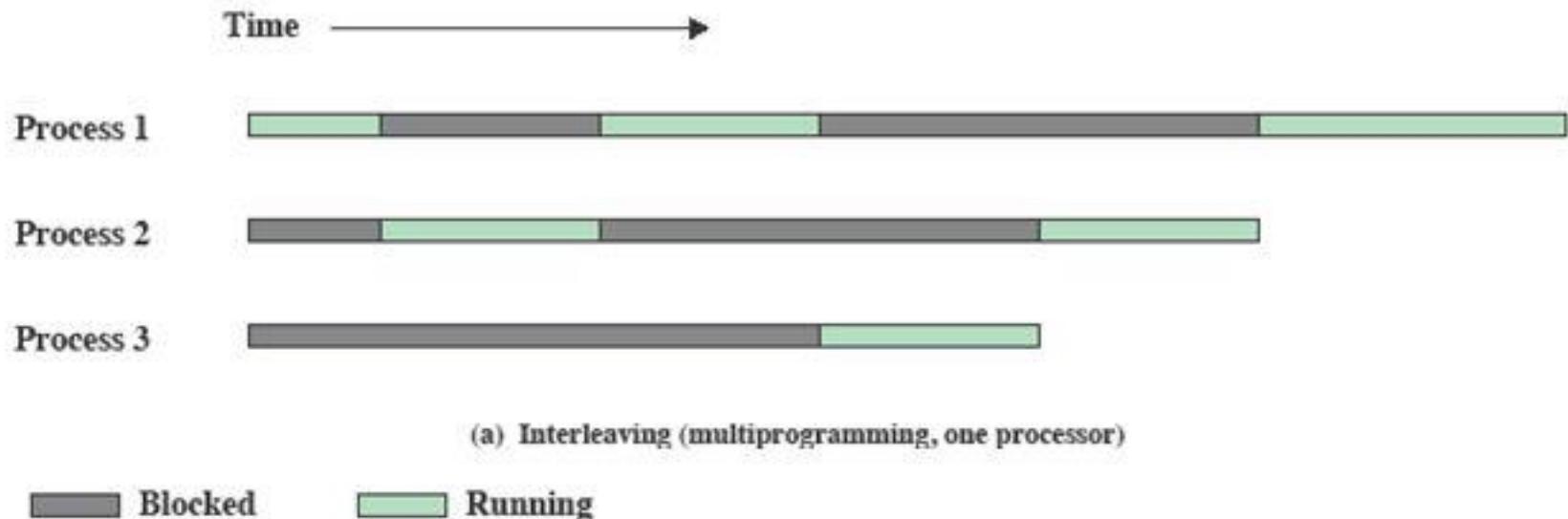
- ❖ Operativni sistem se implementira kao skup procesa ili niti

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi

Principi konkurentnosti

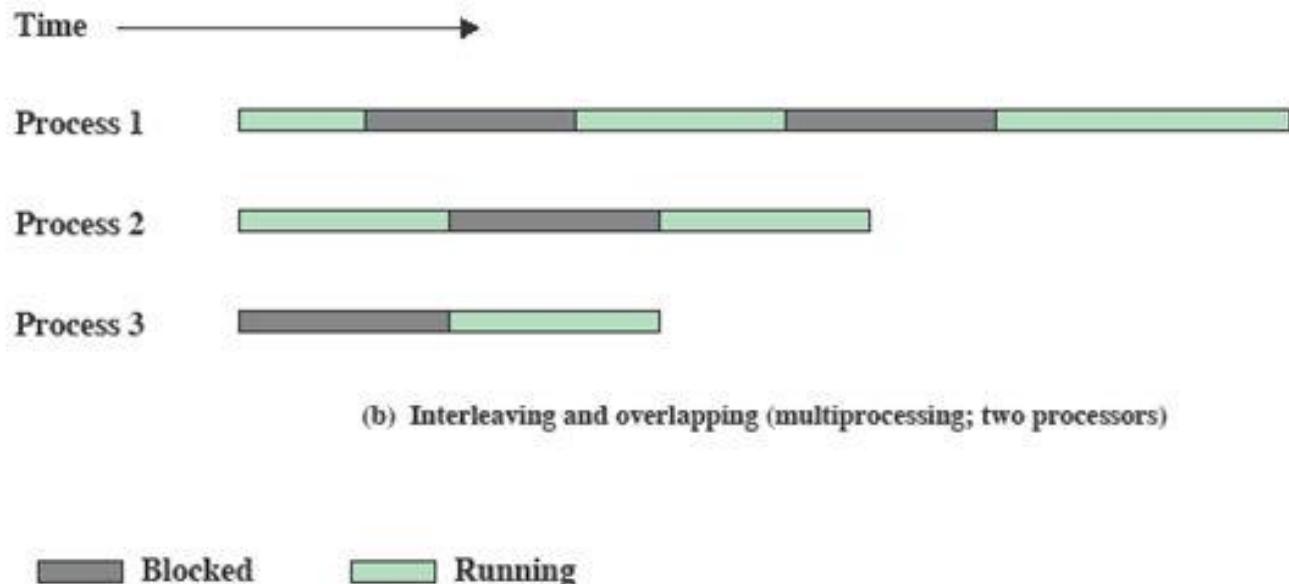
- Preplitanje izvršenja procesa na jednoprocesorskom sistemu



Konkurentnost: uzajamno isključivanje i sinhronizacija
Operativni sistemi

Pirncipi konkurentnosti (2)

- Preklapanje izvršenja procesa na multiprocesorskom sistemu (2 procesora)



Konkurentnost: uzajamno isključivanje i sinhronizacija
Operativni sistemi

Problemi sa konkurentnošću

- ◆ Problemi potiču od osnovnih karakteristika multiprogramske i multiprocesorskih sistema:
 - ▣ Relativna brzina izvršavanja procesa se ne može predvideti
- ◆ Problemi su:
 - ▣ Deljenje globalnih resursa
 - ▣ Operativnom sistemu je teško da optimalno upravlja dodelom resursa
 - ▣ Teško je otkrivanje grešaka u programiranju
- ◆ Rešenje za konkurentnost treba da obuhvati:
 - ▣ Komunikaciju među procesima
 - ▣ Deljenje resursa
 - ▣ Sinhronizaciju više procesa
 - ▣ Alokaciju vremena procesa za korišćenje resursa

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi



Ključni pojmovi koji se odnose na konkurentnost

◆ Kritična sekcija (*critical section*)

- Deo koda unutar procesa koji zahteva pristup deljenim resursima i ne može se izvršiti dok je drugi proces u odgovarajućem delu koda

◆ Uzajamno blokiranje, zastoj (*deadlock*)

- Situacija u kojoj dva ili više procesa ne mogu da nastave sa radom jer svaki čeka da jedan od preostalih nešto uradi

◆ Zaključavanje uživo (*livelock*)

- Situacija u kojoj dva ili više procesa konstantno menjaju svoje stanje kao odgovor na promene drugih procesa i pri tom ne rade ništa korisno

◆ Uzajamno isključivanje (*mutual exclusion*)

- Zahtev po kome nijedan proces ne može biti u kritičnoj sekciji koja pristupa bilo kom deljenom resursu kada je jedan proces u kritičnoj sekciji pristupio deljenim resursima

◆ Uslov trke (*race condition*, haotično stanje)

- Situacija u kojoj više niti ili procesa čita i upisuje stavke deljenih podataka, a krajnji rezultat zavisi od relativnog vremena izvršavanja

◆ Gladovanje (*starvation*)

- Situacija u kojoj dispečer neprekidno preskače proces koji je spreman za izvršavanje, iako je spreman za rad, taj proces nikada neće biti odabran

Konkurentnost: uzajamno isključivanje i sinhronizacija



Jednostavan primer konkurentnosti

```
void echo()
{
    chin = getchar();
    chout = chin;
    putchar(chout);
}
```

- ➊ Posmatraćemo jednoprocesorski multiprogramski sistem koji podržava jednog korisnika
- ➋ Svaki proces može ponavljati poziv ove deljene procedure kako bi prihvatao ulaz sa tastature i prikazivao ga na ekranu korisnika
 - ▣ Jedan primerak ove procedure se učitava u glavnu memoriju i zajednička je za sve aplikacije
 - ▣ Promenljiva *chin* je globalna
- ➌ Korisnik se može kretati između više aplikacija, ali su tastatura i ekran zajednički za sve aplikacije
 - ▣ Svaka aplikacija poziva jedanput, ili više puta ovu proceduru

Konkurentnost: uzajamno isključivanje i sinhronizacija



Jednostavan primer problema koje konkurentnost donosi

Proces P1

```
.  
chin = getchar();  
. . .  
chout = chin;  
putchar(chout);  
.
```

Proces P2

```
.  
. . .  
chin = getchar();  
chout = chin;  
. . .  
putchar(chout);
```

1. Proces P1 poziva proceduru **echo** i biva prekinut nakon prve instrukcije
2. Aktivira se proces P2 i poziva proceduru **echo** koja se izvršava do kraja
3. Nakon procesa P2 nastavlja se izvršavanje procesa P1
 - **Rezultat:** prvi znak je izgubljen, drugi znak se prikazuje 2 puta
 - **Srž problema** je deljena globalna promenljiva **chin**

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi



Jednostavan primer – rešenje problema

Proces P1

```
.  
chin = getchar();  
. .  
chout = chin;  
putchar(chout);  
.
```

Proces P2

```
.  
chin = getchar();  
chout = chin;  
. .  
putchar(chout);
```

Rešenje: Postaviti ograničenje da samo jedan proces u jednom tenutku može biti u proceduri **echo**

1. Proces P1 poziva **echo** i biva prekinuta nakon prve instrukcije (nakon unosa znaka)
2. Aktivira se proces P2 i poziva **echo**, ali kako je proces P1 u proceduri echo, proces P2 se blokira
3. Kada P1 nastavi sa radom, on nalazi svoj znak u **chin** i prikazuje ga na ekranu. Kada P1 napusti proceduru **echo**, uklanja se blokada sa P2 i on ulazi u proceduru **echo**

➊ **Rezultat:** oba znaka su uspešno prikazana

Konkurentnost: uzajamno isključivanje i sinhronizacija

Uslov trke

- ❖ Uslov trke (*race condition*) nastaje kada:
 - ❖ Više procesa ili niti čitaju ili upisuju deljene podatke
 - ❖ Oni to obavljaju na način da finalni rezultat zavisi od redosleda izvršenja procesa.
- ❖ Konačna vrednost deljenih podataka zavisi od toga koji proces završi "trku" poslednji
- ❖ Primer:
 - ❖ Globalne promenljive $b=1, c=2$
 - ❖ P3: $b=b+c$
 - ❖ P4: $c=c+b$
 - ❖ Ako je P3 prvi, rezultat je $b=3, c=5$
 - ❖ Ako je P4 prvi, rezultat je $b=4, c=3$



Problem

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include "common.h"
4 #include "common_threads.h"
5
6 static volatile int counter = 0;
7
8 // mythread()
9 //
10 // Simply adds 1 to counter repeatedly, in a loop
11 // No, this is not how you would add 10,000,000 to
12 // a counter, but it shows the problem nicely.
13 //
14 void *mythread(void *arg) {
15     printf("%s: begin\n", (char *) arg);
16     int i;
17     for (i = 0; i < 1e7; i++) {
18         counter = counter + 1;
19     }
20     printf("%s: done\n", (char *) arg);
21     return NULL;
22 }
23
24 // main()
25 //
26 // Just launches two threads (pthread_create)
27 // and then waits for them (pthread_join)
28 //
29 int main(int argc, char *argv[]) {
30     pthread_t p1, p2;
31     printf("main: begin (counter = %d)\n", counter);
32     Pthread_create(&p1, NULL, mythread, "A");
33     Pthread_create(&p2, NULL, mythread, "B");
34
35     // join waits for the threads to finish
36     Pthread_join(p1, NULL);
37     Pthread_join(p2, NULL);
38     printf("main: done with both (counter = %d)\n",
39             counter);
40     return 0;
41 }
```

```
prompt> gcc -o main main.c -Wall -pthread; ./main
main: begin (counter = 0)
```

```
A: begin
B: begin
A: done
B: done
main: done with both (counter = 20000000)
```

```
prompt> ./main
main: begin (counter = 0)
```

```
A: begin
B: begin
A: done
B: done
main: done with both (counter = 19345221)
```

Uzrok problema?

Konkurentnost: uzajamno isključivanje i sinhronizacija

Uzrok problema

counter = counter +1



```
100 mov    0x8049a1c, %eax
105 add    $0x1, %eax
108 mov    %eax, 0x8049a1c
```

OS	Thread 1	Thread 2	(after instruction)		
			PC	eax	counter
		<i>before critical section</i>	100	0	50
		mov 8049a1c, %eax	105	50	50
		add \$0x1, %eax	108	51	50
interrupt			100	0	50
	<i>save T1</i>		105	50	50
	<i>restore T2</i>		108	51	50
		mov 8049a1c, %eax	113	51	51
interrupt			108	51	51
	<i>save T2</i>		113	51	51
	<i>restore T1</i>				
		mov %eax, 8049a1c			

Konkurentnost: uzajamno isključivanje i sinhronizacija



Poslovi operativnog sistema u vezi konkurentnosti

- OS mora biti sposoban da prati razne procese
- OS mora dodeljivati i oduzimati razne resurse za svaki aktivan proces
 - Procesorsko vreme
 - Memoriju
 - Datoteke
 - U/I uređaje
- OS mora zaštiti podatke i fizičke resurse jednog procesa od nemamernog narušavanja od drugog
- Izlaz mora biti nezavistan od brzine izvršenja drugih konkurentnih procesa

Konkurentnost: uzajamno isključivanje i sinhronizacija

Uzajamno delovanje procesa

- ❖ Procesi nisu svesni postojanja drugih procesa
 - ❖ Nezavisni procesi za koje **nije predviđeno da rade zajedno**
 - ❖ OS vodi računa o njihovom **nadmetanju** za resurse
- ❖ Procesi su indirektno svesni postojanja drugih procesa
 - ❖ Procesi koji dele pristup nekom objektu (npr. U/I baferu)
 - ❖ Pri deljenu zajedničkog objekta pokazuju **kooperaciju preko deljenja**
- ❖ Procesi koji su direktno svesni postojanja drugih procesa
 - ❖ Procesi koji mogu komunicirati preko PID-a i rade zajednički na nekoj aktivnosti
 - ❖ Takvi procesi pokazuju **kooperaciju putem komunikacije**

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi



Uzajamno delovanje procesa

Stepen svesnosti	Odnos	Uticaj koji jedan proces ima na drugi	Potencijalni problemi upravljanja
Procesi nisu svesni postojanja drugih procesa	Nadmetanje	<ul style="list-style-type: none">• Rezultati jednog procesa nezavisni od akcije ostalih procesa• Može uticati na vreme izvršavanja procesa	<ul style="list-style-type: none">• Uzajamno isključivanje• Uzajamno blokiranje (resurs koji se može obnoviti)• Gladovanje
Procesi su indirektno svesni postojanja drugih procesa	Kooperacija deljenjem	<ul style="list-style-type: none">• Rezultati jednog procesa mogu zavisiti od informacija dobijenih od drugi procesa• Može uticati na vreme izvršavanja procesa	<ul style="list-style-type: none">• Uzajamno isključivanje• Uzajamno blokiranje (resurs koji se može obnoviti)• Gladovanje• Povezanost podataka
Procesi koji su direktno svesni postojanja drugih procesa	Kooperacija komunikacijom	<ul style="list-style-type: none">• Rezultati jednog procesa mogu zavisiti od informacija dobijenih od drugih procesa• Može uticati na vreme izvršavanja procesa	<ul style="list-style-type: none">• Uzajamno blokiranje (resurs koji se može obnoviti)• Gladovanje

Konkurentnost: uzajamno isključivanje i sinhronizacija



Nadmetanje procesa za resursima

- ❖ Konkurentni procesi se **nadmeću** za korišćenje istog resursa
- ❖ Pri nadmetanju procesa postoje tri problema upravljanja:
 1. **Uzajamno isključivanje** pri korišćenju nedeljivog resursa
 - Takav resurs nazivamo **kritični resurs**, a deo programa koji ga koristi **kritična sekcija**
 - Kritična sekcija
 - Samo jedan program u datom trenutku može biti u kritičnoj sekciji
 - **Primer:** samo jedan proces u jednom trenutku može slati komandu štampaču
 2. Uzajamno blokiranje
 3. Gladovanje

Konkurentnost: uzajamno isključivanje i sinhronizacija

Ilustracija uzajamnog isključivanja

- ❖ N procesa pristupa deljenom resursu u okviru svoje kritične sekcije
 - ❖ Funkcije zaključavanja/otključavanja (*entercritical/exitcritical*)

```
PROCESS 1 /*  
void P1  
{  
    while (true) {  
        /* preceding code */;  
        entercritical (Ra);  
        /* critical section */;  
        exitcritical (Ra);  
        /* following code */;  
    }  
}  
  
/* PROCESS 2 */  
void P2  
{  
    while (true) {  
        /* preceding code */;  
        entercritical (Ra);  
        /* critical section */;  
        exitcritical (Ra);  
        /* following code */;  
    }  
}...  
/* PROCESS n */  
void Pn  
{  
    while (true) {  
        /* preceding code */;  
        entercritical (Ra);  
        /* critical section */;  
        exitcritical (Ra);  
        /* following code */;  
    }  
}
```



Kooperacija procesa deljenjem i komunikacijom

- ❖ Konkurentni procesi (ili niti) često imaju potrebe da dele podatke (koji se održavaju u deljenoj memoriji, ili u datotekama) i resurse, ili razmenjuju poruke
- ❖ Ako ne postoji kontrolisani pristup deljenim resursima/podacima, tj. ukoliko nema uzajamnog isključivanja neki procesi će dobiti nekonzistentan pogled na podatke
- ❖ Kod kooperacije komunikacijom, nema uzajamnog isključivanja već je moguće uzajamno blokiranje i gladovanje
- ❖ Akcije koje izvode konkurentni procesi zavisi će od redosleda kojim se izvršenja prepliću/preklapaju –**uslov trke**
- ❖ Zbog toga procese treba **sinhronizovati**

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi



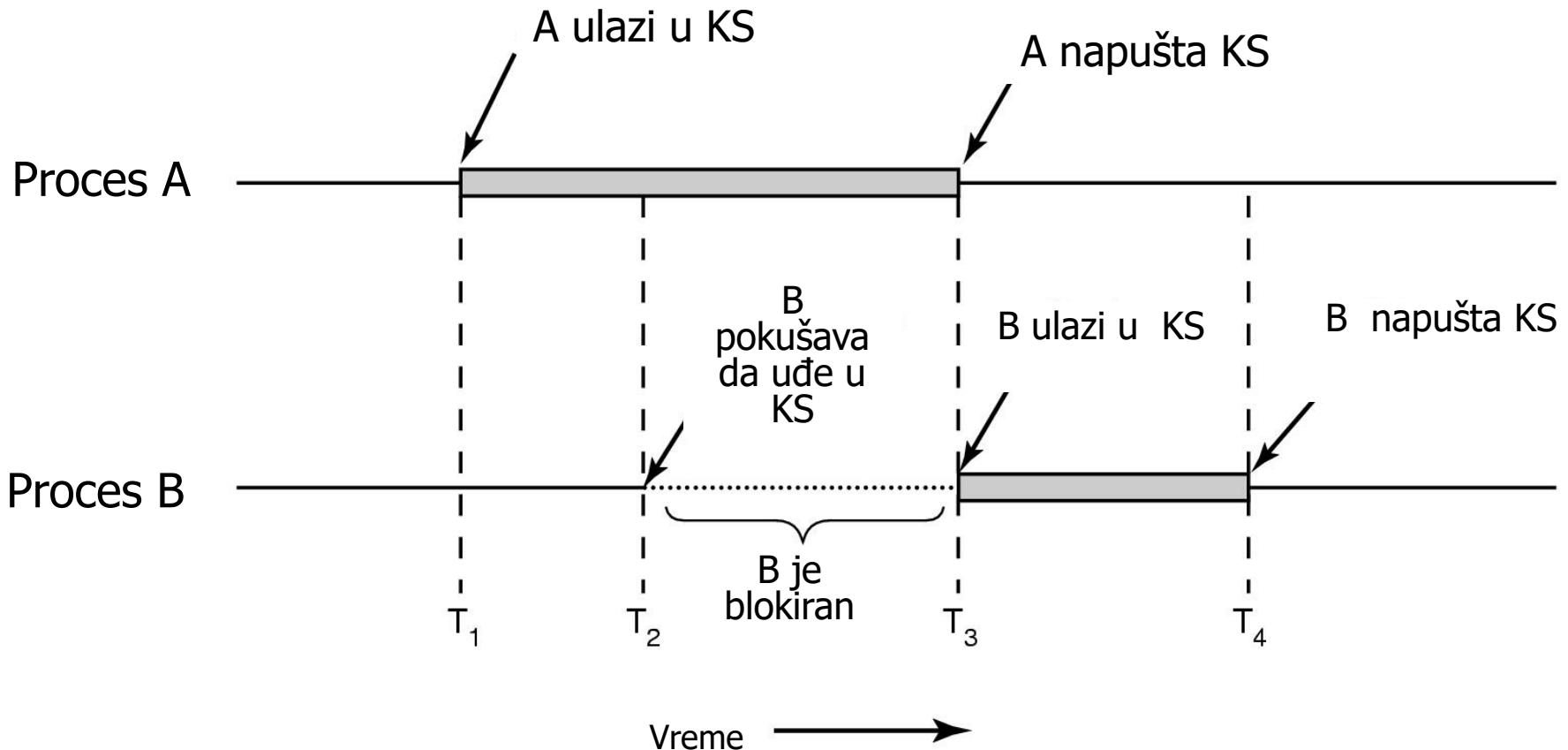
Zahtevi za uzajamno isključivanje

- ❖ Samo jednom procesu je dozvoljeno da uđe u kritičnu sekciju (KS) radi pristupa deljenom resursu
 - Dva i više procesa ne mogu biti istovremeno u kritičnoj sekciji
- ❖ Proces koji se zaustavi u sekciji koja nije kritična mora to uraditi bez uticaja na druge procese
- ❖ Proces koji zahteva ulazak u KS ne može biti beskonačno zadržan – nema zastoja i gladovanja
- ❖ Kada nijedan proces nije u KS, bilo kom procesu koji zahteva ulaz u KS to mora biti dozvoljeno
- ❖ Ne prave se pretpostavke o relativnoj brzini procesa niti o broju procesora
- ❖ Proces ostaje u kritičnoj sekciji samo konačno vreme

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi

Uzajamno isključivanje u izvršavanju kritične sekcije



Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi



Rešenje uzajamnog isključivanja

- SW rešenja (za samostalni rad, Dodatak A)
 - Softverski algoritmi imaju značajno dodatno procesiranje i rizik od nastanka logičkih grešaka
- HW rešenja
 - Vezana su za neke mašinske instrukcije i prevenciju prekida
- OS rešenja
 - OS obezbeđuje sistemske funkcije i strukture podataka koje programeri mogu koristiti za programiranje kritičnih sekacija



Uzajamno isključivanje: hardverska podrška

- ➊ Postoje dva pristupa:
 - ▣ Zabrana (onemogućavanje, *disable*) prekida
 - ▣ Specijalne mašinske instrukcije



Zabрана прекида

◆ На једнопроцесорском систему:

- Обећено узјамно искључивање, али ефикасност извршења деградирана.
- Док је процес у КС, ниједан други процес се не може извршавати

◆ На вишепроцесорском систему овај приступ не ради:

- Узјамно искључивање нје обезбеђено

◆ Генерално, решење нје прихватљиво

Proces Pi:

```
while(true)  
{
```

zabrana prekida
KRITIČNA SEKCIJA
dozvola prekida
OSTATAK

```
}
```



Specijalne mašinske instrukcije

- ◆ Na hardverskom nivou, pristup nekoj memorijskoj lokaciji isključuje ostale pristupe toj lokaciji
- ◆ Projektanti HW su predložili **mašinske instrukcije** koje nad istom memorijskom lokacijom izvode dve akcije **atomično** (nedeljivo, u jednom koraku koji ne može biti prekinut)
 - ▣ Na primer, čitanje i upis, ili čitanje i testiranje
- ◆ Izvršenje takve instrukcije je uzajamno isključivo (čak i kod multiprocesora)
 - ▣ Tokom izvršenja instrukcije, pristup memorijskoj lokaciji je blokiran za sve druge instrukcije koje referenciraju tu lokaciju
- ◆ Implementirane instrukcije:
 - ▣ *Test and Set* instrukcija
 - ▣ *Compare&Swap* instrukcija
 - ▣ *Exchange* instrukcija

Konkurentnost: uzajamno isključivanje i sinhronizacija



Test and Set instrukcija

- ❖ Instrukcija za testiranje i postavljanje (Test and Set Instruction)

```
boolean testset (int i) {  
    if (i == 0) {  
        i = 1;  
        return true;  
    }  
    else {  
        return false;  
    }  
}
```

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi



Exchange instrukcija

● Instrukcija razmene (*Exchange Instruction*)

```
void exchange(int register,  
             int memory) {  
    int temp;  
    temp = memory;  
    memory = register;  
    register = temp;  
}
```

● Instrukcija XCHG na procesorima Intel-32 (Pentium) i IA-64 (Itanium)

Konkurentnost: uzajamno isključivanje i sinhronizacija

Uzajamno isključivanje korišćenjem mašinskih instrukcija

```
/* program mutual exclusion */
const int n = /* number of processes */;
int bolt;
void P(int i)
{
    while (true)
    {
        while (!testset (bolt))
            /* do nothing */;
        /* critical section */;
        bolt = 0;
        /* remainder */
    }
}
void main()
{
    bolt = 0;
    parbegin (P(1), P(2), . . . , P(n));
}
```

(a) Test and set instruction

```
/* program mutual exclusion */
int const n = /* number of processes*/;
int bolt;
void P(int i)
{
    int keyi;
    while (true)
    {
        keyi = 1;
        while (keyi != 0)
            exchange (keyi, bolt);
        /* critical section */;
        exchange (keyi, bolt);
        /* remainder */
    }
}
void main()
{
    bolt = 0;
    parbegin (P(1), P(2), . . . , P(n));
}
```

(b) Exchange instruction

Figure 5.2 Hardware Support for Mutual Exclusion

Konkurentnost: uzajamno isključivanje i sinhronizacija

Compare & Swap instrukcija

```
int compare_and_swap (int
    *word,
    int testval,
    int newval)
{
    int oldval;
    oldval = *word;
    if (oldval == testval)
        *word = newval;
    return oldval;
}
```

```
/* program mutual exclusion */
const int n = /* number of processes */;
int bolt;
void P(int i)
{
    while (true) {
        while (compare_and_swap(bolt, 0, 1) == 1)
            /* do nothing */;
        /* critical section */;
        bolt = 0;
        /* remainder */;
    }
}
void main()
{
    bolt = 0;
    parbegin (P(1), P(2), ... ,P(n));
}
```

(a) Compare and swap instruction



Svojstva pristupa sa mašinskim instrukcijama - Prednosti

- Može se primenjivati na bilo koji broj procesa i na jednom da procesoru i na multiprocesoru gde procesori dele glavnu memoriju
- Jednostavno je i lako se proverava
- Može se koristiti za podršku više kritičnih sekcija; svaka KS može se definisati svojom promenljivom



Svojstva pristupa sa mašinskim instrukcijama - Nedostaci

- Zaposleno čekanje
- Moguće je gladovanje
 - Kada jedan proces napusti KS, a više čeka na ulaz u KS, izbor sledećeg procesa je proizvoljan
- Moguće je uzajamno blokiranje
 - Ako je proces nižeg prioriteta P1 u KS, prekinut je i procesor je dodeljen procesu višeg prioriteta P2 koji želi da uđe u KS
 - P1 se prekida, P2 dobija procesor
 - ako P2 želi da koristi isti resurs kao P1 neće mu biti dozvoljeno
 - P2 čeka i drži procesor, tako da P1 neće nikad biti raspoređen na procesor jer je nižeg prioriteta

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi



Semafori

- ➊ Definisani od strane E. Dijkstra, 1965
- ➋ Sinhronizacioni mehanizam koji obezbeđuje OS
- ➌ Semafor S je **integer promenljiva** nad kojom su definisane tri **atomične** operacije
 - ▣ **Inicijalizacija**
 - ▣ **semWait(S)**
 - alternativna imena **P** (Dijkstra, na holandskom *proberen*), **down** (Tanenbaum), **wait**
 - ▣ **semSignal(S)**
 - alternativna imena **V** (Dijkstra, na holandskom *verhogen*), **up** (Tanenbaum), **signal**
- ➍ Semafor može biti:
 - ▣ **Binarni** – uzima samo dve vrednosti 0 i 1
 - ▣ **Brojački (generalni)** – uzima vrednosti 0,1,2,...,n,

Konkurentnost: uzajamno isključivanje i sinhronizacija



Operacije semafora – definicija

- ❖ Proces koji izvede operaciju ***semWait(S)*** **blokira** sam sebe, umesto da čeka u petlji
- ❖ Proces koji je blokiran na semaforu **biće probuđen** kada neki drugi proces izvede operaciju ***semSignal(S)***
- ❖ Ove dve operacije moraju biti:
 - ▣ **Atomične** – moraju se izvoditi u celini, tj. bez prekidanja
 - ▣ **Uzajamno isključive** - na istom semaforu ne mogu se istovremeno izvoditi

semWait(S):

if ($S > 0$) **then**
 $S = S - 1$

else

{blokiraj_proces, aktiviraj_dispečer}

semSignal(S):

if (jedan ili više procesa čeka na S) **then**
probudi jedan od procesa koji su blokirani na semaforu S

else

$S = S + 1$



Operacije binarnog semafora

```
struct binary_semaphore {
    enum {zero, one} value;
    queueType queue;
};

void semWaitB(binary_semaphore s)
{
    if (s.value == one)
        s.value = zero;
    else {
        /* place this process in s.queue */;
        /* block this process */;
    }
}

void semSignalB(semaphore s)
{
    if (s.queue is empty())
        s.value = one;
    else {
        /* remove a process P from s.queue */;
        /* place process P on ready list */;
    }
}
```

Konkurentnost: uzajamno isključivanje i sinhronizacija



Semafor – implementacija

```
struct semaphore {
    int count;
    queueType queue;
};

void semWait(semaphore s)
{
    s.count--;
    if (s.count < 0) {
        /* place this process in s.queue */;
        /* block this process */;
    }
}

void semSignal(semaphore s)
{
    s.count++;
    if (s.count <= 0) {
        /* remove a process P from s.queue */;
        /* place process P on ready list */;
    }
}
```

Konkurentnost: uzajamno isključivanje i sinhronizacija

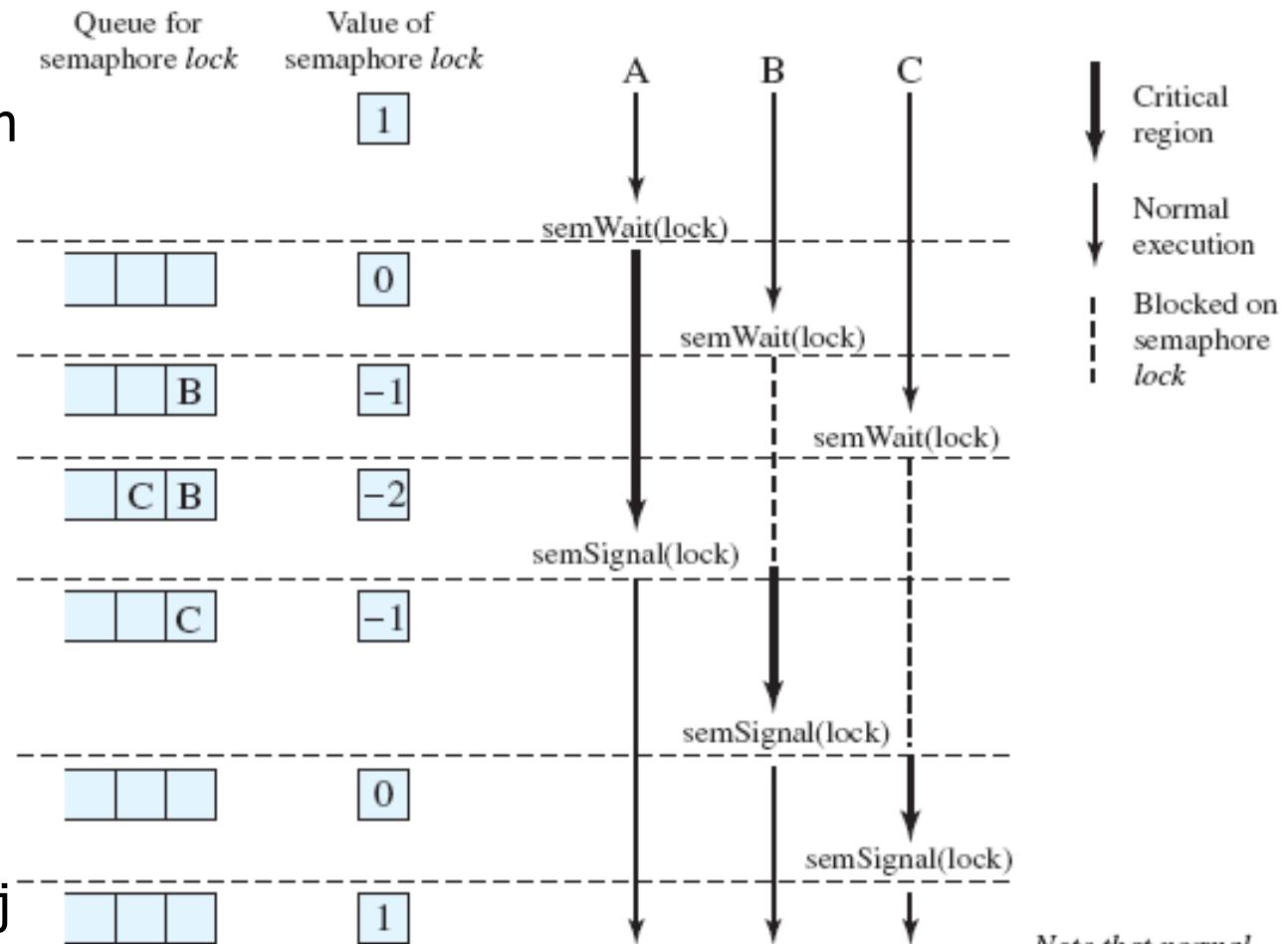


Uzajamno isključivanje korišćenjem semafora

```
/* program mutual exclusion */
const int n = /* number of processes */;
semaphore s = 1;
void P(int i)
{
    while (true) {
        semWait(s);
        /* critical section */;
        semSignal(s);
        /* remainder */;
    }
}
void main()
{
    parbegin (P(1), P(2), . . . , P(n));
}
```

Primer: Uzajamno isključivanje

- Tri procesa pristupaju deljenim podacima zaštićenim semaforom
- $s .count \geq 0$ – broj procesa koji može da izvrši operaciju `semWait(s)` bez blokiranja (ukoliko u međuvremenu nije izvršena `semSignal(s)`)
- $s .count < 0$ – broj procesa blokiranih u redu `s.queue`



Note that normal execution can proceed in parallel but that critical regions are serialized.



Korišćenje semafora za sinhronizaciju dva procesa

- ❖ Semafori se mogu koristiti **za sinhronizaciju dva procesa**
- ❖ Posmatramo dva procesa P1 i P2
- ❖ Naredba S1 procesa P1 treba da se izvede pre naredbe S2 procesa P2
- ❖ Koristi se binarni semafor **synch**
- ❖ Semafor **synch** se inicijalizira na 0

```
void P1()
{
    S1;
    semSignal(synch);
    ...
}

void P2()
{
    ...
    semWait(synch);
    S2;
}

void main()
{
    semaphore  sync =0;
    parbegin( P1,P2);
}
```

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi



Korišćenje semafora za sinhronizaciju procesa (2)

- ❖ Semafori se mogu koristiti za sinhronizaciju procesa tipa ***rendezvous***
- ❖ Kod tzv. **sastanka** (***rendezvous***) proces koji prvi dođe do KS mora čekati da drugi proces stigne do KS
- ❖ Koriste se dva binarna semafora **S1 i S2 koji se inicijalizuju na 0**

```
void P1()
{
    ...
    semSignal(s1);
    semWait(s2);
    ...
}

void P2()
{
    ...
    semSignal(s2);
    semWait(s1);
    ...
}

void main()
{
    semaphore s1=0, s2=0;
    parbegin( P1, P2 );
}
```

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi



Problem proizvođač/potrošač (Producer/Consumer)

- Jedan ili više proizvođača generiše podatke (slogove, znakove) i stavlja ih u bafer
- Jedan potrošač uzima podatke iz bafera jednu po jednu
- Sistem mora obezbiti da se operacije nad baferom ne preklapaju
- U datom trenutku samo jedan proizvođač ili potrošač može pristupati baferu
- Pretpostavimo da je bafer neograničen i sadrži linearan niz elemenata

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi

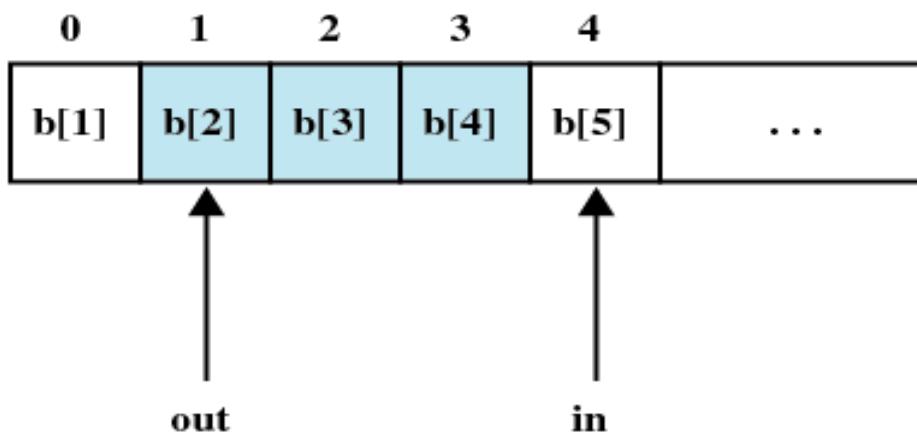
Proizvođač i potrošač

producer:

```
while (true) {  
    /*napravi stavku v*/;  
    b[in] = v;  
    in++;  
}
```

consumer:

```
while (true) {  
    while (in <= out)  
        /*ne radi ništa*/;  
    w = b[out];  
    out++;  
    /* potroši stavku w */  
}
```



Tamna polja predstavljaju elemente bafera (niza) sa podacima

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi

Rešenje korišćenjem binarnih semafora

- Neispravno rešenje problema proizvođač/potrošač sa beskonačnim baferom uz upotrebu binarnih semafora
- Broj elemenata u baferu $n = \text{in} - \text{out}$
- Semafor *s* obezbeđuje uzajamno isključivanje
- Semafor *delay* primorava potrošača da se blokira (izvršavanjem *semWait*) ukoliko je bafer prazan

```
/* program producerconsumer */
int n;
binary_semaphore s = 1;
binary_semaphore delay = 0;
void producer()
{
    while (true)
    {
        produce();
        semWaitB(s);
        append();
        n++;
        if (n==1)
            semSignalB(delay);
        semSignalB(s);
    }
}
void consumer()
{
    semWaitB(delay);
    while (true)
    {
        semWaitB(s);
        take();
        n--;
        semSignalB(s);
        consume();
        if (n==0)
            semWaitB(delay);
    }
}
void main()
{
    n = 0;
    parbegin (producer, consumer);
}
```

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi

Moguć scenario ovog rešenja

	Producer	Consumer	s	n	Delay
1			1	0	0
2	semWaitB(s)		0	0	0
3	n++		0	1	0
4	if (n==1) (semSignalB(delay))		0	1	1
5	semSignalB(s)		1	1	1
6		semWaitB(delay)	1	1	0
7		semWaitB(s)	0	1	0
8		n--	0	0	0
9		semSignalB(s)	1	0	0
10	semWaitB(s)		0	0	0
11	n++		0	1	0
12	if (n==1) (semSignalB(delay))		0	1	1
13	semSignalB(s)		1	1	1
14		if (n==0) (semWaitB(delay))	1	1	1
15		semWaitB(s)	0	1	1
16		n--	0	0	1
17		semSignalB(s)	1	0	1
18		if (n==0) (semWaitB(delay))	1	0	0
19		semWaitB(s)	0	0	0
20		n--	0	-1	0
21		semiSignlaB(s)	1	-1	0

Konkurentnost: uzajamno isključivanje i sinhronizacija



Korigovano rešenje

- ➊ Korektno rešenje problema proizvođač/potrošač sa **beskonačnim** baferom uz upotrebu binarnih semafora
- ➋ Dodavanje pomoćne promenljive **m** u okviru kritične sekcije

```
/* program producerconsumer */
int n;
binary_semaphore s = 1;
binary_semaphore delay = 0;
void producer()
{
    while (true)
    {
        produce();
        semWaitB(s);
        append();
        n++;
        if (n==1) semSignalB(delay);
        semSignalB(s);
    }
}
void consumer()
{
    int m; /* a local variable */
    semWaitB(delay);
    while (true)
    {
        semWaitB(s);
        take();
        n--;
        m = n;
        semSignalB(s);
        consume();
        if (m==0) semWaitB(delay);
    }
}
void main()
{
    n = 0;
    parbegin (producer, consumer);
}
```

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi



Rešenje korišćenjem brojačkog semafora

- Korektno rešenje problema proizvođač/potrošač sa **beskonačnim** baferom uz upotrebu brojačkog semafora
- Promenljiva **n** je sada semafor i predstavlja broj elemenata u baferu
- **Problem:** ako se semWait operacije nad **n** i **s** zamene

```
/* program producerconsumer */
semaphore n = 0;
semaphore s = 1;
void producer()
{
    while (true)
    {
        produce();
        semWait(s);
        append();
        semSignal(s);
        semSignal(n);
    }
}
void consumer()
{
    while (true)
    {
        semWait(n);
        semWait(s);
        take();
        semSignal(s);
        consume();
    }
}
void main()
{
    parbegin (producer, consumer);
}
```

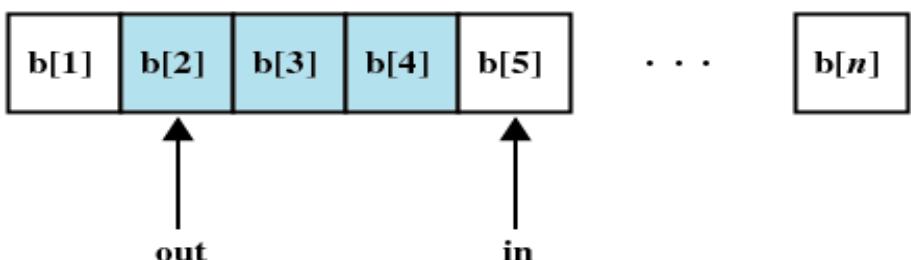
Problem proizvođač/potrošač sa kružnim baferom

producer:

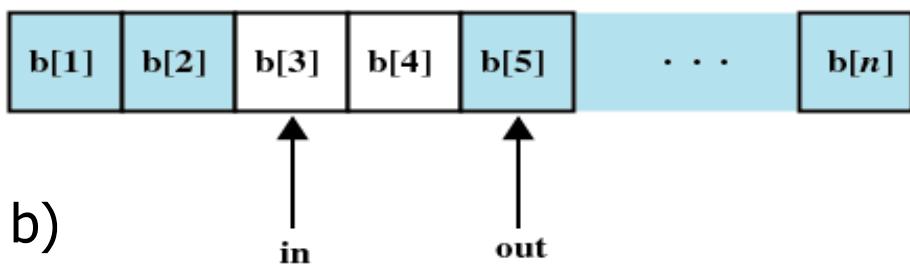
```
while (true) {  
    /* napravi stavku v */  
    while((in+1)%n==out)  
        /* ne radi ništa */;  
    b[in] = v;  
    in = (in + 1) % n  
}
```

consumer:

```
while (true) {  
    while (in == out)  
        /* ne radi ništa */;  
    w = b[out];  
    out = (out + 1) % n;  
    /* potroši stavku w */  
}
```



a)



b)

Rešenje sa ograničenim kružnim baferom

- ➊ Korektno rešenje problema proizvođač/potrošač sa **ograničenim** kužnim baferom uz upotrebu binarnog i brojačkog semafora

- Proizvođač se **blokira** kad hoće da upiše element u pun bafer
- Potrošač se **blokira** kad hoće da uzme element iz praznog bafera

```
/* program boundedbuffer */
const int sizeofbuffer = /* buffer size */;
semaphore s = 1;
semaphore n= 0;
semaphore e= sizeofbuffer;
void producer()
{
    while (true)
    {
        produce();
        semWait(e);
        semWait(s);
        append();
        semSignal(s);
        semSignal(n)
    }
}
void consumer()
{
    while (true)
    {
        semWait(n);
        semWait(s);
        take();
        semSignal(s);
        semSignal(e);
        consume();
    }
}
void main()
{
    parbegin (producer, consumer);
}
```

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi



Problemi sa semaforima

- ◆ Semafori nude moćan alat za obezbeđenje uzajamnog isključivanja i za koordinaciju procesa
- ◆ Ali *semWait(S)* i *semSignal(S)* su razbacani među procesima, pa je teško razumeti njihove efekte
- ◆ Korišćenje mora biti korektno u svim procesima
- ◆ Jedan loš proces može uticati na zastoj čitave kolekcije procesa

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi

Monitori

- Monitor je konstrukcija viših programskih jezika koja nudi funkcionalnost ekvivalentnu semaforima, ali je lakša za upravljanje
- Prvi put ga je formalno definisao Hoare, 1974
- Nalazi se u mnogim programskim jezicima
 - Concurrent Pascal, Pascal-Plus, Modula-2, Modula-3, Java
 - Takođe se može implementirati kao programska biblioteka

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi

Monitori (2)

Monitor je SW modul koji sadrži:

- Jednu ili više procedura/funkcija
- Inicijalizacionu sekvencu
- Lokalne promenljive

Karakteristike monitora:

1. Lokalnim promenljivama može se pristupati samo iz procedura monitora i nijednom spoljnom procedurom
2. Proces "ulazi" u monitor pozivom neke od njegovih procedura
3. U jednom trenutku samo jedan proces se može izvršavati u monitoru; svaki drugi proces koji poziva proceduru monitora se blokira i čeka da monitor postane dostupan

Prve dve karakteristike podsećaju na karakteristike objekata u OO jezicima

- Monitor se može implementirati kao objekat

Treća karakteristika obezbeđuje uzajamno isključivanje

Konkurentnost: uzajamno isključivanje i sinhronizacija



Monitori (3)

- ❖ Monitor osigurava **uzajamno isključivanje**
 - ❖ Nije potrebno programirati ovo ograničenje
- ❖ U jednom trenutku deljivim podacima u monitoru može pristupati samo jedan proces
 - ❖ Stoga su deljivi podaci zaštićeni njihovim smeštanjem u monitor
- ❖ Monitor takođe poseduje sinhronizacioni alat
 - ❖ To su **uslovne promenljive (condition)**
 - ❖ Sinhronizaciju procesa vrši programer korišćenjem uslovnih promenljivih

Konkurentnost: uzajamno isključivanje i sinhronizacija

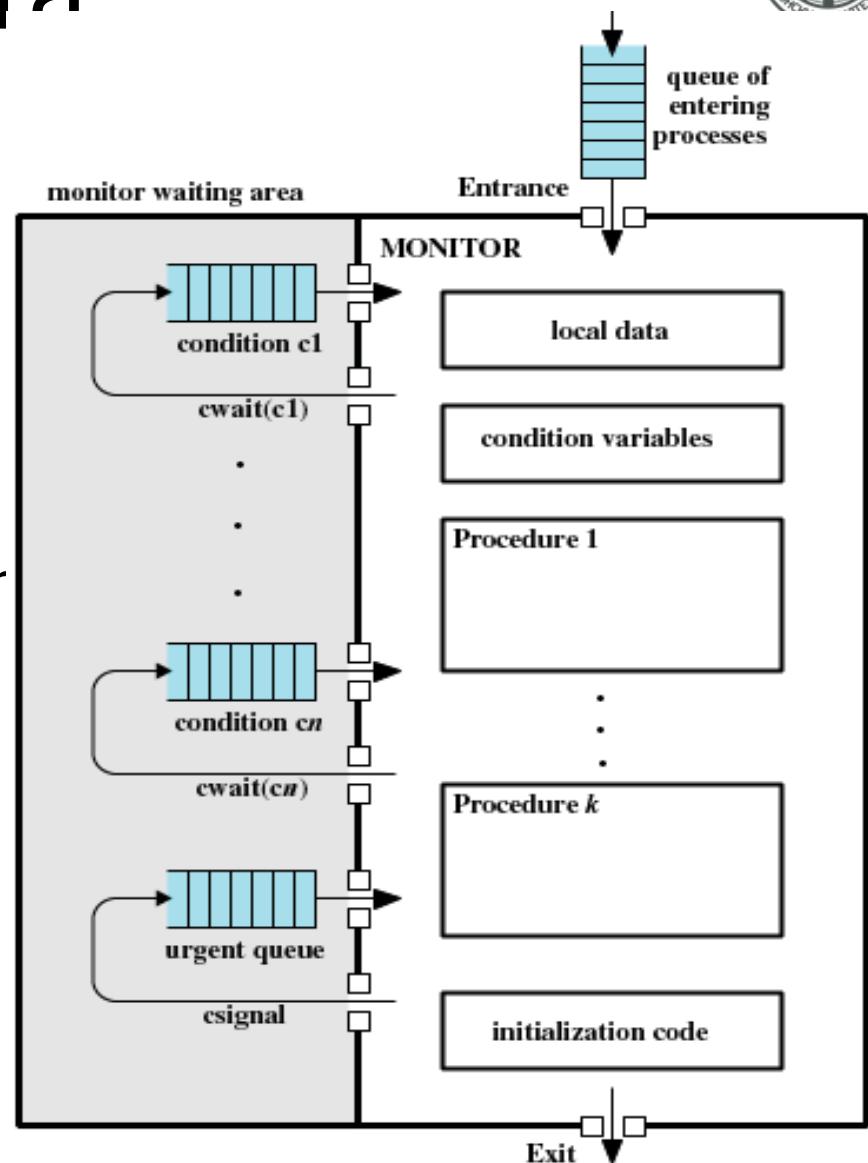
Operativni sistemi

Uslovne promenljive monitora

- ❖ **Uslovne promenljive** su lokalne u monitoru
 - ❖ Može im se pristupati jedino unutar monitora
- ❖ Uslovne promenljive su specijalnog tipa podataka u monitoru kojima se može pristupati jedino pomoću dve funkcije:
 - ❖ **cwait(a)**: blokira izvršenje procesa koji je pozvao ovu funkciju na uslovnoj promenljivoj **a**; monitor je sada na raspolaganju za bilo koji drugi proces
 - ❖ **csignal(a)**: budi proces koji je blokiran na uslovnoj promenljivoj **a**
 - Ako postoji više takvih procesa – bira jedan
 - Ako nema takvih procesa – ne radi ništa
 - ❖ Treba uočiti da su operacije **cwait** i **csignal** monitora različite od istih operacija kod semafora
 - Ako proces u monitoru izda signal i ako nijedan proces ne čeka na toj uslovnoj promenljivoj signal je izgubljen

Struktura monitora

- Monitor ima jednu ulaznu i jednu izlaznu tačku
- Proces koji je u monitoru može privremeno sam sebe blokirati na uslovnoj promenljivoj **a** korišćenjem operacije **cwait(a)**
- On se tada stavlja u red procesa koji pokušavaju da ponovo uđu u monitor kada se uslov promeni i tada nastavljaju sa izvršenjem sa naredbom koja sledi iza **cwait(a)**
- Ako proces koji se izvršava u monitoru detektuje ispunjenje uslova na kojem su blokirani procesi, poziva **csignal(a)** koja budi (deblokira) proces iz reda procesa te uslovne promenljive **a**





Primena monitora za rešavanje problema proizvođač-potrošač

- ❖ Monitor **boundedbuffer** upravlja baferom koji se koristi za smeštanje i preuzimaje znakova
- ❖ Monitor sadži dve uslovne promenljive **notfull** (kada postoji mesto u baferu da se doda barem jedan znak) i **notempty** (kada u baferu postoji barem jedan znak)
- ❖ Proizvođač može dodati znak u bafer samo preko procedure monitora **append**
 - ❖ Proizvođač nema direktni pristup baferu
- ❖ Potrošač može uzeti znak iz bafera samo preko procedure monitora **take**
 - ❖ Potrošač nema direktni pristup baferu

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi

Monitori: Proizvođač/potrošač

```
/* program producerconsumer */
monitor boundedbuffer;
char buffer [N];
int nextin, nextout;
int count;
cond notfull, notempty;
                           /* space for N items */
                           /* buffer pointers */
                           /* number of items in buffer */
                           /* condition variables for synchronization */

void append (char x)
{
    if (count == N)
        cwait(notfull);           /* buffer is full; avoid overflow */
    buffer[nextin] = x;
    nextin = (nextin + 1) % N;
    count++;
    /* one more item in buffer */
    csignal(notempty);         /* resume any waiting consumer */
}
void take (char x)
{
    if (count == 0)
        cwait(notempty);         /* buffer is empty; avoid underflow */
    x = buffer[nextout];
    nextout = (nextout + 1) % N;
    count--;
    csignal(notfull);          /* one fewer item in buffer */
                               /* resume any waiting producer */
}
{
    nextin = 0; nextout = 0; count = 0;           /* monitor body */
                                                /* buffer initially empty */
}
```

Lokalne
promenljive
monitorsa

procedura
append

procedura
take

inicijalizacija



Monitori: Proizvođač/potrošač

```
void producer()
char x;
{
    while (true)
    {
        produce(x);
        append(x);
    }
}
void consumer()
{
    char x;
    while (true)
    {
        take(x);
        consume(x);
    }
}
void main()
{
    parbegin (producer, consumer);
}
```

Konkurentnost: uzajamno isključivanje i sinhronizacija



Prenos poruka (*Message Passing*)

- ◆ To je metoda komunikacije među procesima
- ◆ Koriste se dve primitive:

send(odredište,poruka)

receive(izvor,poruka)

- ◆ **send** – proces šalje **poruku** drugom procesu koji je označen kao **odredište**
- ◆ **receive** – proces prima **poruku** od procesa koji je označen kao **izvor**



Prenos poruka (*Message Passing*)

● Problemi u prenosu poruka:

- Format poruke
- **Sinhronizacija** – procese koji komuniciraju treba na neki način sinhronizovati
- **Protokol** za komunikaciju
 - Primalac i pošiljalac moraju dogovoriti neki protokol za komunikaciju
 - Npr. primalac po prijemu poruke šalje potvrdu, a pošiljalac ako potvrda ne stigne za neko vreme ponovo šalje poruku
 - Poruke se numerišu čime se izbegava zabuna ako poruka kasni
- **Imenovanje** (adresiranje) procesa
 - Procesi koji komuniciraju moraju imati način da se međusobno referenciraju
- **Autentikacija** – da pošiljalac i primalac znaju da komuniciraju sa pravim procesom

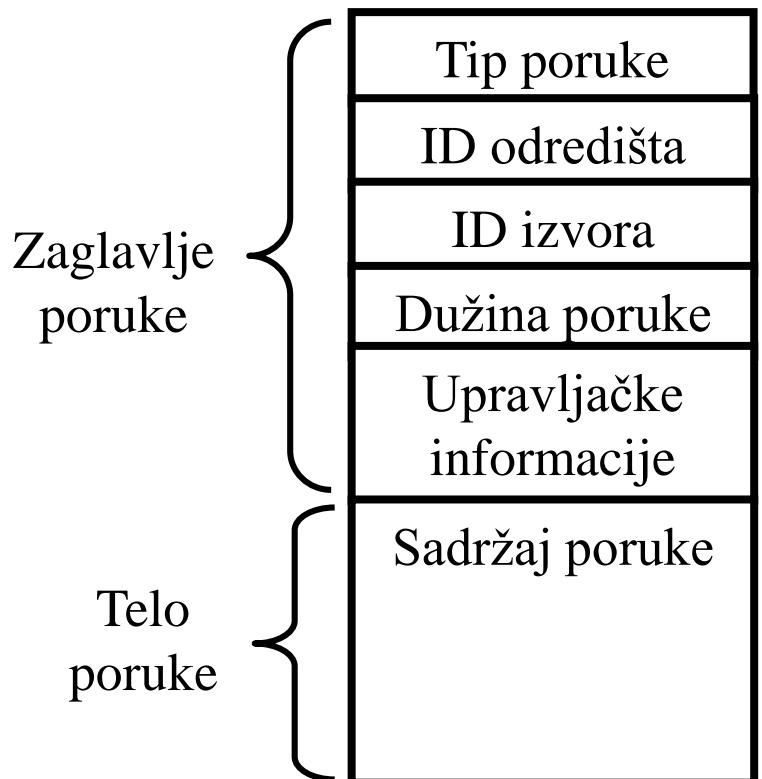
Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi

Format poruke

- Format poruke zavisi:
 - Od cilja sistema za prenos poruka
 - Od toga da li se sistem izvršava na jednom računaru ili distribuiranom sistemu
- Poruke mogu biti fiksne i promenljive dužine

- Opšti format poruke promenljive dužine



Sinhronizacija u prenosu poruka (1)

- Komunikacija dva procesa porukama podrazumeva neki nivo sinhronizacije između njih
 - Primalac ne može primiti poruku pre nego što je neki proces ne pošalje
- Važno je da se definiše šta se događa sa procesom nakon što pokrene **send** ili **receive** primitivu
- Proces primalac i proces pošiljalac mogu biti:
 - blokirani
 - neblokirani



Sinhronizacija u prenosu poruka (2)

◆ Moguće su tri kombinacije:

▣ Blokirajući **send**, blokirajući **receive**

- pošiljalac i primalac se blokiraju dok poruka ne bude isporučena
- Naziva se ***rendezvous***

▣ Neblokirajući **send**, blokirajući **receive**

- Pošiljalac nastavlja da se izvršava, a primalac se blokira dok ne stigne poruka
- Omogućava se pošiljaocu da pošalje 1 ili više poruka na jednu ili više destinacija
- Primer su serverski procesi

▣ Neblokirajući **send**, neblokirajući **receive**

- Ni pošiljalac ni primalac ne moraju da budu blokirani



Sinhronizacija u prenosu poruka (3)

- Za pošiljaoca: prirodno je da **ne bude blokiran** posle slanja poruke pozivom funkcije **send**
 - Može poslati nekoliko poruka na više destinacija
 - Ali pošiljalac obično očekuje potvrdu prijema poruke (za slučaj da je primalac neispravan)
- Za primaoca: prirodnije je **da bude blokiran** dok ne primi poruku pozivom funkcije **receive**
 - Primalac obično treba neku informaciju pre nego što nastavi obradu
 - Postoji opasnost da bude neograničeno blokiran ako proces pošiljalac završi pre nego što pošalje **send**

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi

Adresiranje procesa

➊ Direktno adresiranje

- Jedan proces direktno šalje poruku drugom procesu
- Pošiljalac eksplicitno specificira kome šalje poruku preko argumenta **odredište**
- Primalac eksplicitno specificira od koga prima poruku preko argumenta **izvor**

➋ Indirektno adresiranje

- Poruke se ne šalju direktno od pošiljaoca primaocu, već se šalju deljenim strukturama podataka koje se sastoje od redova koji mogu privremeno da čuvaju poruke
- Ovi redovi se nazivaju **poštanski sandučići (mailboxes)**
- Proces pošiljalac šalje poruku u određeni mailbox, a drugi proces primalac uzima poruku iz tog mailbox-a



Adresiranje poruka

- Kod indirektnog adresiranja **odnos između pošiljaoca i primaoca** može biti:
 - 1:1 (jedan-prema-jedan)
 - privatni komunikacioni link između 2 procesa
 - N:1 (više-prema-jedan)
 - koristan za klijent-server interakciju kada jedan proces prima poruke od većeg broja drugih procesa
 - U ovom slučaju se mailbox naziva **port**
 - 1:N (jedan-prema-više)
 - Postoji 1 pošiljalac i više primaoca
 - Pogodan je u slučaju kada jedan proces emituje poruke većem broju procesa
 - N:M (više-prema-više)
 - Dопушта да више serverskih procesa konkurentno nudi servis većem broju klijenata

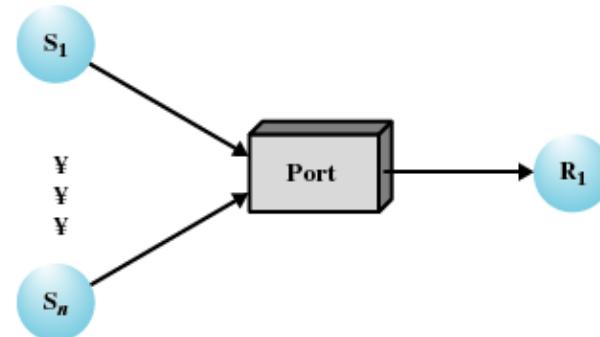
Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi

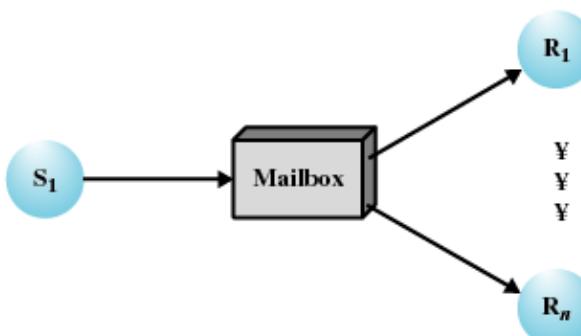
Indirektna komunikacija među procesima



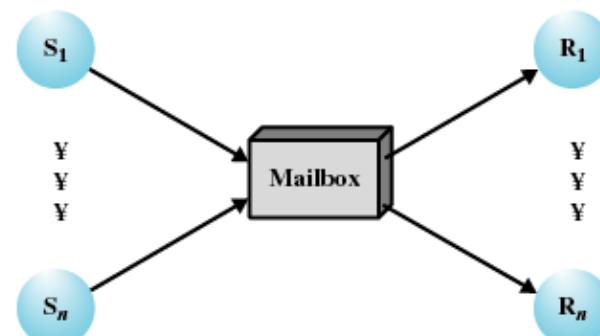
jedan:jedan



više:jedan



jedan:više



više:više

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi



Poštansko sanduče i port

- ❖ Poštansko sanduče (mailbox) može biti
 - ❖ Privatno za par pošiljalac-primalac
 - ❖ Isto poštansko sanduče može deliti više pošiljaoca i primaoca
 - OS mora obezbediti tip podataka *message*
- ❖ Port je poštansko sanduče povezano sa 1 primaocem i više pošiljaoca
 - ❖ Koristi se za klijent-server aplikacije
- ❖ Vlasnici portova i mailbox-ova
 - ❖ Port obično kreira proces primalac i on je njegov vlasnik
 - ❖ Port se uništava kad se proces primalac terminira
 - ❖ Mailbox kreira OS u ime nekog procesa koji postaje njegov vlasnik
 - ❖ Takav mailbox se uništava na zahtev vlasnika ili kada se vlasnik terminira

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi

Uzajamno isključivanje prenosom poruka

```
/* program mutualexclusion */
const int n = /* number of processes */;
void P(int i)
{
    message msg;
    while (true)
    {
        receive (mutex, msg);
        /* critical section */;
        send (mutex, msg);
        /* remainder */;
    }
}
void main()
{
    create_mailbox (mutex);
    send (mutex, null);
    parbegin (P(1), P(2), . . . , P(n));
}
```

Figure 5.20 Mutual Exclusion Using Messages



Uzajamno isključivanje prenosom poruka (2)

- ❖ Kreira se mailbox **mutex** deljiv za N konkurentnih procesa
- ❖ **send()** je neblokirajući
- ❖ **receive()** – proces se blokira kada je poštansko sanduče **mutex** prazno
- ❖ Inicijalizacija: šalje se prazna poruka **send(mutex,null);** null je poruka bez sadržaja
- ❖ Proces Pi koji želi da uđe u KS pokušava da primi poruku
 - ❖ Ukoliko je mailbox mutex prazan, proces se blokira
 - ❖ Inače proces Pi ulazi u KS
 - ❖ Po izlasku iz KS vraća poruku u mailbox
- ❖ Prvi proces Pi koji izvrši **receive()** će ući u KS. Ostali će biti blokirani dok Pi ponovo ne pošalje poruku

Proizvođač-potrošač sa prenosom poruka

- ❖ Koriste se dva mailbox-a: *mayproduce* i *mayconsume*
 - ❖ Proizvođač generiše podatke i šalje ih u mailbox *mayconsume*
 - ❖ Potrošač uzima poruke iz mailbox-a *mayconsume* dok ima bar jedna poruka
- ❖ Mailbox *mayconsume* je **bafer**
 - ❖ Bafer je kapaciteta K poruka (globalna promenljiva *capacity*)
- ❖ Mailbox *mayproduce* se inicijalno puni sa K praznih (null) poruka
 - ❖ Broj poruka u mailbox-u *mayproduce* se smanjuje sa svakom proizvodnjom, a povećava sa svakom potrošnjom
- ❖ Može podržavati više proizvođača i potrošača koji imaju pristup do oba mailbox-a
- ❖ Sistem može biti distribuiran:
 - ❖ proizvođač i mailbox *mayproduce* na jednom mestu, a potrošač i mailbox *mayconsume* na drugom

Konkurentnost: uzajamno isključivanje i sinhronizacija

Proizvođač-potrošač sa prenosom poruka

```
const int
    capacity = /* buffering capacity */ ;
    null =/* empty message */ ;
int i;
void producer()
{
    message pmsg;
    while (true)
    {
        receive (mayproduce, pmsg);
        pmsg = produce();
        send (mayconsume, pmsg);
    }
}
void consumer()
{
    message cmsg;
    while (true)
    {
        receive (mayconsume, cmsg);
        consume (cmsg);
        send (mayproduce, null);
    }
}

void main()
{
    create_mailbox (mayproduce);
    create_mailbox (mayconsume);
    for (int i = 1; i <= capacity; i++)
        send (mayproduce, null);
    parbegin (producer, consumer);
}
```



Klasični sinhronizacioni problemi

- Proizvođač – potrošač
- Čitaoci – pisci
- Večera filozofa
- Uspavani berberin

**Konkurentnost: uzajamno isključivanje i sinhronizacija
Operativni sistemi**



Problem čitaoci-pisci

The Readers and Writers Problem

- ❖ Objekti podataka (npr. fajl, blok memorije, grupa registara procesora) su deljivi za više konkurentnih procesa
- ❖ Neki procesi samo čitaju sadržaj deljivih objekata (to su **čitaoci**), dok drugi samo upisuju u deljivi objekat (to su **pisci**)
 - ❖ Ako dva čitaoca istovremeno čitaju deljivi objekat nema neželjenih efekata
 - ❖ Međutim, ako pisac i neki drugi proces (čitalac ili pisac) pristupaju simultano deljivom objektu nastupiće problem
- ❖ Moraju biti zadovoljeni sledeći uslovi:
 - ❖ Bilo koji broj čitalaca može istovremeno čitati deljivi objekat
 - ❖ U jednom trenutku samo 1 pisac može upisivati u deljivi objekat
 - ❖ Ukoliko pisac upisuje, nijedan čitalac ne može da čita
- ❖ Ovaj sinhronizacioni problem se naziva **problem čitaoci-pisci**

Problem čitaoci-pisci (2)

- ❖ Da bi se izbegli problemi **treba obezbediti da pisci imaju ekskluzivan pristup deljivom objektu**
- ❖ Ima više rešenja
 - ❖ **Čitaoci imaju prioritet** – nijedan novi čitalac se ne drži na čekanju, ako je neki čitalac već dobio dozvolu da koristi deljivi objekat; tj. nijedan čitalac neće čekati da neki čitalac završi čitanje bez obzira da li neki pisac već čeka
 - ❖ **Pisci imaju prioritet** – jedan pisac je spremjan da obavi upis čim bude to moguće; tj. ako neki pisac čeka za pristup objektu, nijedan novi čitalac ne može startovati čitanje
- ❖ Oba rešenja mogu dovesti do **izgladnjivanja** procesa:
 - ❖ Čitaoci imaju prioritet – gladuju *pisci*
 - ❖ Pisci imaju prioritet – gladuju *čitaoci*



Čitaoci imaju prioritet

```
/* program readersandwriters */
int readcount;
semaphore x = 1, wsem = 1;
void reader()
{
    while (true)
    {
        semWait (x);
        readcount++;
        if (readcount == 1)
            semWait (wsem);
        semSignal (x);
        READUNIT();
        semWait (x);
        readcount--;
        if (readcount == 0)
            semSignal (wsem);
        semSignal (x);
    }
}
void writer()
{
    while (true)
    {
        semWait (wsem);
        WRITEUNIT();
        semSignal (wsem);
    }
}
void main()
{
    readcount = 0;
    parbegin (reader, writer);
}
```

readcount – broj čitalaca

wsem – semafor za uzajamno isključivanje na deljivom objektu; dok pisac piše nijedan čitalac ili pisac ne mogu da koriste deljivi objekat

Ako neki čitalac čita ostali ne moraju da čekaju

Ako nijedan čitalac ne čita, prvi čitalac mora čekati na **wsem**

x – koristi se da bi se obezbedilo ispravno ažuriranje promenljive **readcount**

Konkurentnost: uzajamno isključivanje i sinhronizacija

```
/* program readersandwriters */
int readcount, writecount;
semaphore x = 1, y = 1, z = 1, wsem = 1, rsem = 1;
void reader()
{
    while (true)
    {
        semWait (z);
        semWait (rsem);
        semWait (x);
        readcount++;
        if (readcount == 1)
            semWait (wsem);
        semSignal (x);
        semSignal (rsem);
        semSignal (z);
        READUNIT();
        semWait (x);
        readcount--;
        if (readcount == 0)
            semSignal (wsem);
        semSignal (x);
    }
}
void writer ()
{
    while (true)
    {
        semWait (y);
        writecount++;
        if (writecount == 1)
            semWait (rsem);
        semSignal (y);
        semWait (wsem);
        WRITEUNIT();
        semSignal (wsem);
        semWait (y);
        writecount--;
        if (writecount == 0)
            semSignal (rsem);
        semSignal (y);
    }
}
void main()
{
    readcount = writecount = 0;
    parbegin (reader, writer);
}
```

Pisci imaju prioritet

rsem – sprečava sve čitaoce dok postoji bar jedan pisac koji želi da pristupi deljivom podatku

wsem – za uzajamno isključivanje na deljivom objektu; dok pisac piše nijedan čitalac ili pisac ne mogu da koriste deljivi objekat

x – semafor koji se koristi da bi se obezbedilo ispravno ažuriranje promenljive **readcount**

y – semafor koji se koristi da bi se obezbedilo ispravno ažuriranje promenljive **writecount**

z – semafor koji obezbeđuje da samo jedan čitalac može biti u redu na semaforu **rsem**, dok ostali ulaze u red na semaforu **z**



Čitaoci-pisci: rešenje prenosom poruka

```
void reader(int i)
{
    message rmsg;
    while (true) {
        rmsg = i;
        send (readrequest, rmsg);
        receive (mbox[i], rmsg);
        READUNIT ();
        rmsg = i;
        send (finished, rmsg);
    }
}

void writer(int j)
{
    message rmsg;
    while(true) {
        rmsg = j;
        send (writerequest, rmsg);
        receive (mbox[j], rmsg);
        WRITEUNIT ();
        rmsg = j;
        send (finished, rmsg);
    }
}
```

```
void controller()
{
    while (true)
    {
        if (count > 0) {
            if (!empty (finished)) {
                receive (finished, msg);
                count++;
            }
            else if (!empty (writerequest)) {
                receive (writerequest, msg);
                writer_id = msg.id;
                count = count - 100;
            }
            else if (!empty (readrequest)) {
                receive (readrequest, msg);
                count--;
                send (msg.id, "OK");
            }
        }
        if (count == 0) {
            send (writer_id, "OK");
            receive (finished, msg);
            count = 100;
        }
        while (count < 0) {
            receive (finished, msg);
            count++;
        }
    }
}
```

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi



Domaći zadatak

◆ Poglavlje 5 Konkurentnost: uzajamno isključivanje i sinhronizacija

- 5.9 Ključni pojmovi, kontrolna pitanja i problemi

◆ Animacije

- Mutual exclusion with a semaphore
<http://williamstallings.com/OS/Animation/Queensland/SEMA.SWF>
- Process Synchronization: Producer/Consumer problem
<https://apps.uttyler.edu/Rainwater/COSC3355/Animations/processync.htm>

Konkurentnost: uzajamno isključivanje i sinhronizacija

Operativni sistemi



Operativni sistemi

- Konkurentnost: uzajamno blokiranje i gladovanje -

Prof. dr Dragan Stojanović

Katedra za računarstvo
Univerzitet u Nišu, Elektronski fakultet



Literatura

- ➊ *Operating Systems: Internals and Design Principles*, edition, W. Stallings, Pearson Education Inc., 7th – 2012, (5th -2005, 6th - 2008, 8th – 2014 , 9th – 2017)
 - <http://williamstallings.com/OperatingSystems/>
 - <http://williamstallings.com/OperatingSystems/OS9e-Student/>
- ➋ **Poglavlje 6:** Konkurentnost: uzajamno blokiranje i gladovanje
- ➋ **Dodatak A:** Teme o konkurentnosti

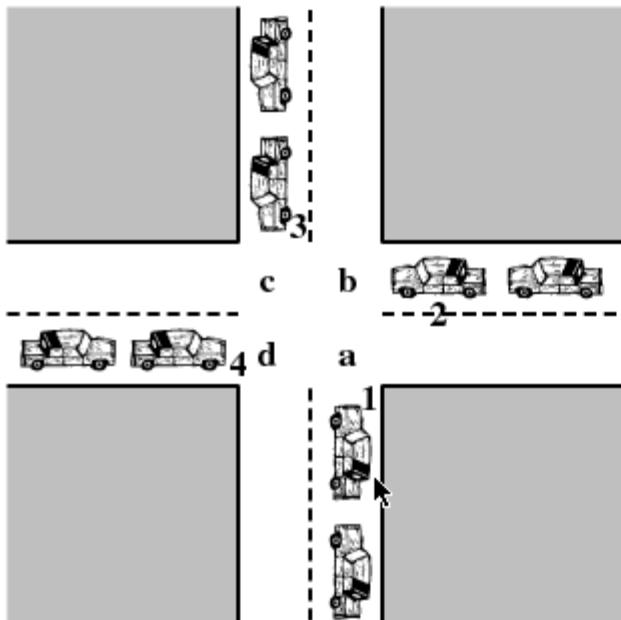


Uzajamno blokiranje

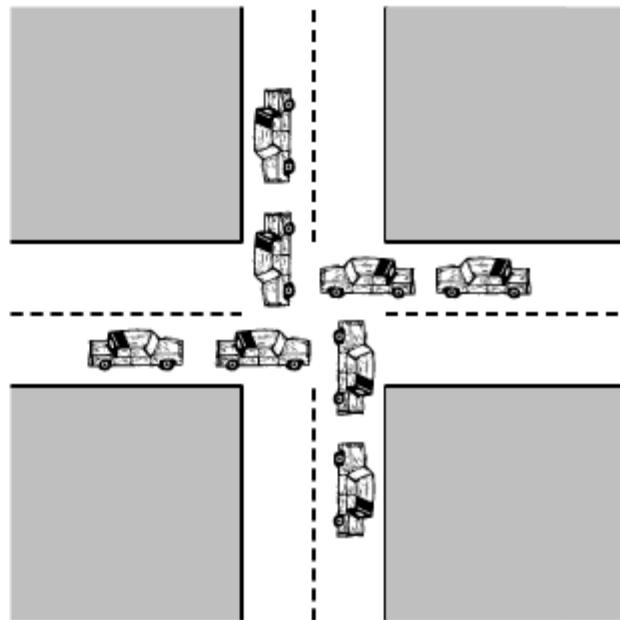
- ❖ Šta je uzajamno blokiranje?
 - ❖ **Trajno** uzajamno blokiranje skupa procesa koji se nadmeću za resurse sistema ili međusobno komuniciraju
- ❖ Skup procesa je **uzajamno blokiran** kada je svaki proces iz ovog skupa **blokiran** čekajući na događaj koji može aktivirati samo neki od procesa iz tog skupa
 - ❖ Uključuje procese koji se nadmeću za resurse iz istog skupa
- ❖ Ne postoji efikasno rešenje uzajamnog blokiranja

Ilustracija uzajamnog blokiranja

4 kvadranta raskrsnice su resursi koje treba kontrolisati



Potencijalno uzajamno blokiranje



Nastanak uzajamnog blokiranja

Primer uzajamnog blokiranja

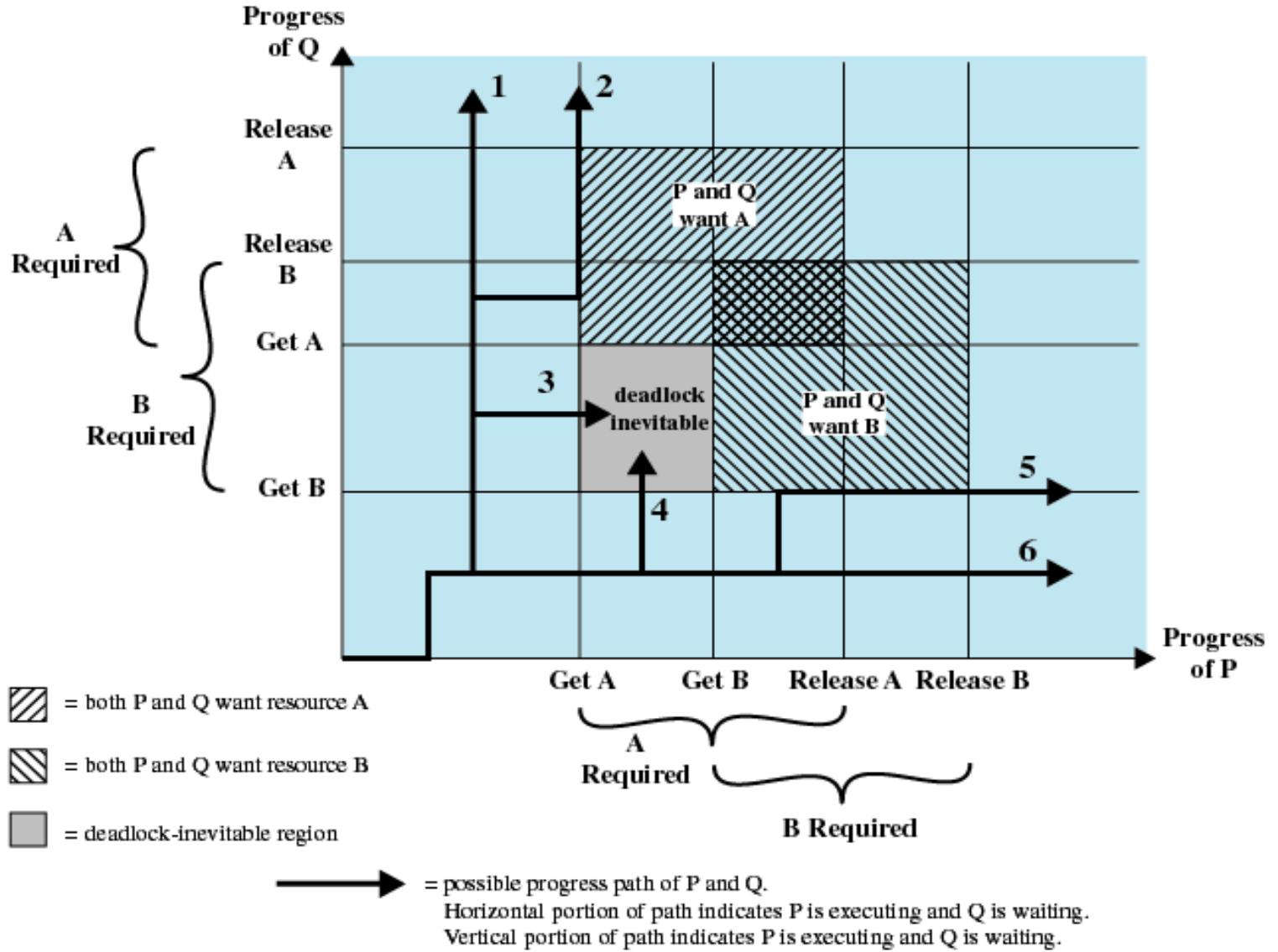
- ➊ Dva procesa P i Q zahtevaju isključiv pristup resursima A i B za određeni period vremena
- ➋ Dijagram zajedničkog izvršavanja (progrusa)

Process P	Process Q
...	...
Get A	Get B
...	...
Get B	Get A
...	...
Release A	Release B
...	...
Release B	Release A
...	...

Konkurentnost: uzajamno blokiranje i gladovanje

Operativni sistemi

Diagram zajedničkog napretka



Konkurentnost: uzajamno blokiranje i gladovanje

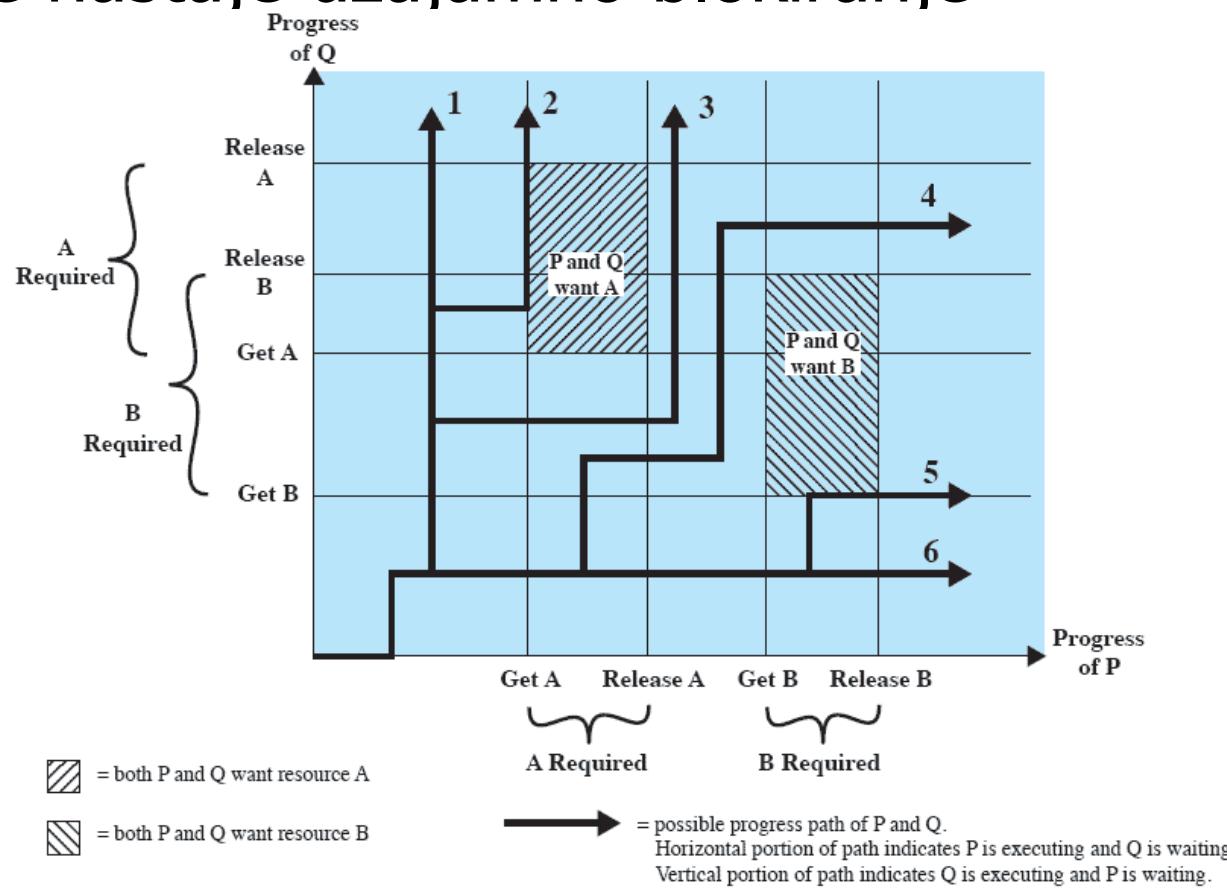
Operativni sistemi

Alternativni primer

- Prepostavimo da procesu P ne trebaju oba resursa istovremeno - ne nastaje uzajamno blokiranje

Process P
...
Get A
...
Release A
...
Get B
...
Release B
...

Process Q
...
Get B
...
Get A
...
Release B
...
Release A
...



Konkurentnost: uzajamno blokiranje i gladovanje

Operativni sistemi

Kategorije resursa

- ➊ Pod resursom ćemo podrazumevati bilo koji objekat koji se može dodeliti procesima
- ➋ Postoje dve kategorije resursa:
 - ➌ Ponovo upotrebljivi resursi
 - Mogu istovremeno biti korišćeni od strane samo jednog procesa i nakon njegovo korišćenja upotrebljivi su za drugi proces
 - ➍ Potrošni resursi
 - To su resursi koji se mogu **kreirati** (proizvesti) i **uništiti** (potrošiti)



Ponovo upotrebljivi resursi

- ❖ Procesi zauzimaju instance resursa, koriste ih i kasnije ih oslobađaju kako bi ih koristili drugi procesi
- ❖ Primer ponovo upotrebljivih resursa:
 - ❖ Procesori
 - ❖ Glavna i sekundarna memorija
 - ❖ U/I uređaji
 - ❖ U/I kanali
 - ❖ Strukture podataka, poput datoteka, baza podataka, semafora
- ❖ Uzajamno blokiranje nastaje kad svaki proces drži jedan resurs i zahteva drugi

Konkurentnost: uzajamno blokiranje i gladovanje

Operativni sistemi



Primer uzajamnog blokiranja na ponovo upotrebljivim resursima (1)

- Dva procesa P i Q se nadmeću za ekskluzivan pristup datoteci D i uređaju trake T
- Uzajamno blokiranje će se javiti ako svaki proces drži jedan resurs i zahteva drugi, npr. ako se ispreplete izvršavanje ovih procesa na sledeći način **p0 p1 q0 q1 p2 q2**

Process P

Step	Action
p ₀	Request (D)
p ₁	Lock (D)
p ₂	Request (T)
p ₃	Lock (T)
p ₄	Perform function
p ₅	Unlock (D)
p ₆	Unlock (T)

Process Q

Step	Action
q ₀	Request (T)
q ₁	Lock (T)
q ₂	Request (D)
q ₃	Lock (D)
q ₄	Perform function
q ₅	Unlock (T)
q ₆	Unlock (D)

Konkurentnost: uzajamno blokiranje i gladovanje



Primer uzajamnog blokiranja na ponovo upotrebljivim resursima (2)

- ➊ Dva procesa P1 i P2 se nadmeću za dodelu raspoloživog memorijskog prostora od 200 KB
 - ▣ Nastane sledeća sekvenca događaja

P1

...

Request 80 Kbytes;

...

Request 60 Kbytes;

P2

...

Request 70 Kbytes;

...

Request 80 Kbytes;

- ➋ Uzajamno blokiranje će se javiti ukoliko oba procesa u izvršavanju stignu do drugog zahteva

Potrošni resursi

- Kada proces potrošač zahteva takav resurs i dobije ga, resurs prestaje da postoji
- Ne postoji ograničenje u broju takvih resursa koje proces može napraviti
- Primeri potrošnih resursa:
 - Prekidi
 - Signali
 - Poruke
 - Informacije u U/I baferima
- Uzajamno blokiranje može nastati ako je operacija *Receive* poruke blokirajuća
- Retke kombinacije događaja mogu prouzrokovati uzajamno blokiranje

Konkurentnost: uzajamno blokiranje i gladovanje

Primer uzajamnog blokiranja

- Dva procesa **P1** i **P2** komuniciraju razmenjujući poruke
- Uzajamno blokiranje se može desiti ako je funkcija **Receive** blokirajuća

```
P1  
...  
Receive(P2);  
...  
Send(P2, M1) ;
```

```
P2  
...  
Receive (P1) ;  
...  
Send (P1, M2) ;
```



Model sistema

- ➊ Sistem se sastoji od određenog broja resursa koji se dodeljuju procesima
- ➋ Resursi su razvrstani u više tipova
 - ▣ Npr. štampači, memorija,...
- ➌ Svaki tip ima više identičnih instanci
 - ▣ Ako proces zahteva neki tip resursa, OS mu može dodeliti bilo koju instancu tog tipa resursa
- ➍ Proces može zahtevati proizvoljan broj resursa da bi obavio zadatak
 - ▣ Broj zahtevanih resursa ne sme prelaziti broj raspoloživih resursa



Graf dodele resursa

- ◆ **Graf dodele resursa** je orijentisani graf $G=(V,E)$ kojim se modelira dodela resursa
- ◆ Skup čvorova V se deli na dva podskupa:
 - **Čvorovi procesa** - Skup procesa u sistemu $P=(P_1, P_2, \dots, P_n)$
 - **Čvorovi resursa** - Skup tipova resursa $R=(R_1, R_2, \dots, R_m)$
- ◆ Skup grana E se deli na dva podskupa:
 - **Grane zahteva (zahteva, traži)** - Skup grana zahteva –par (P_i, R_j)
 - **Grane dodele (drži)** - Skup grana dodele – to je uređeni par (R_j, P_i)
- ◆ **Grana zahteva** (P_i, R_j) modelira pojavu da proces P_i zahteva instancu resursa R_j
- ◆ **Grana dodele** (R_j, P_i) modelira pojavu da je instance resursa tipa R_j dodeljena procesu P_i , tj. da proces P_i **drži** resurs tipa R_j

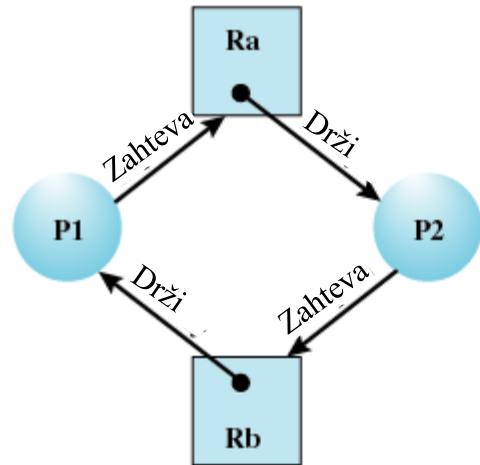
Graf dodele resursa (2)



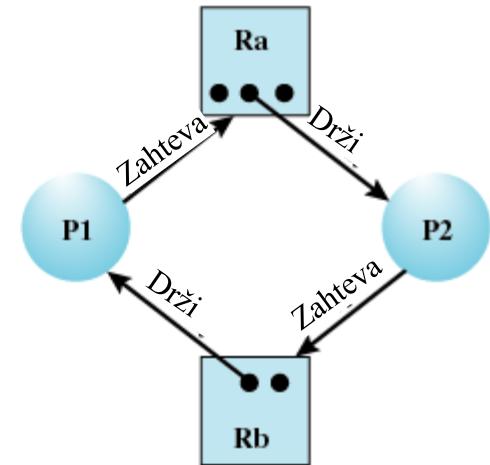
(a) Proces zahteva resurs



(b) Proces drži resurs



(c) Kružno čekanje



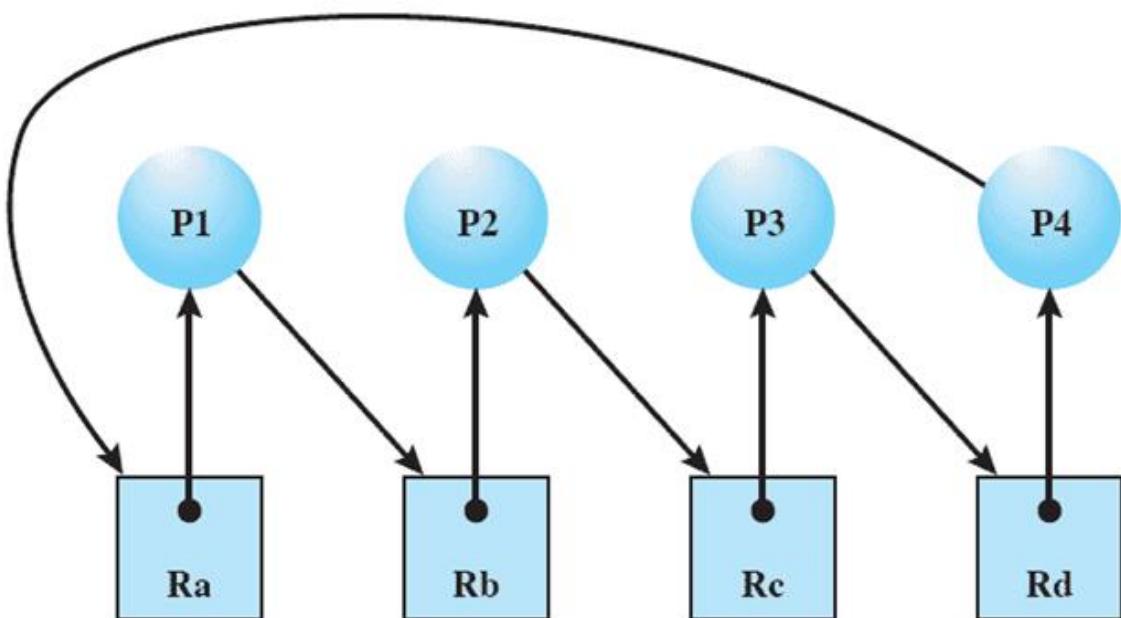
(d) Bez uzajamnog blokiranja

Konkurentnost: uzajamno blokiranje i gladovanje

Operativni sistemi

Graf dodele resursa (3)

- ❖ Graf dodele resursa za primer automobila na raskrsnici

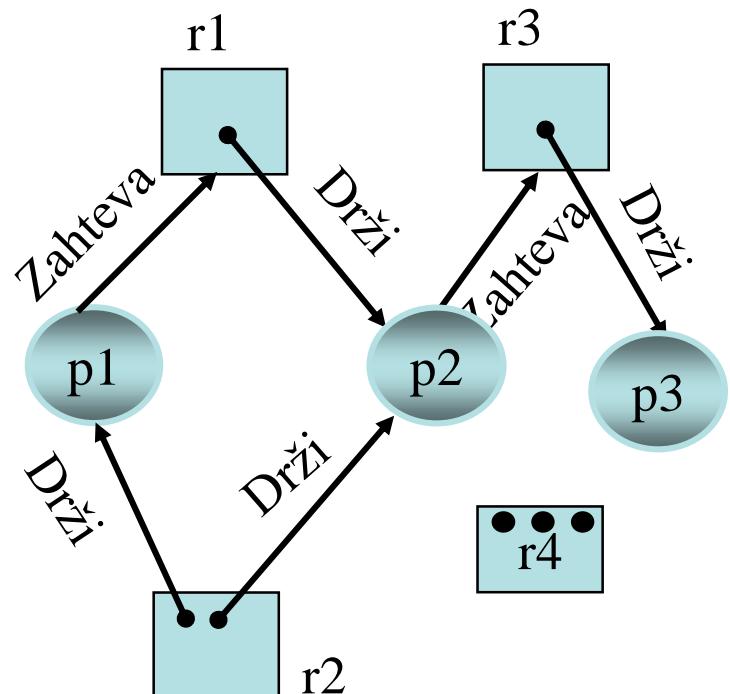


Konkurentnost: uzajamno blokiranje i gladovanje

Operativni sistemi

Primer grafa dodele resursa

- Procesi $P = (p_1, p_2, p_3)$
- Resursi $R = (r_1, r_2, r_3, r_4)$
- Instance resursa
 - r_1 ima 1 instancu
 - r_2 ima 2 instance
 - r_3 ima 1 instancu
 - r_4 ima 3 instance
- Stanje:
 - p_1 drži 1 instancu resursa tipa r_2 i zahteva 1 instancu resursa tipa r_1
 - p_2 drži r_1 i r_2 i zahteva r_3
 - p_3 drži r_3



Nema uzajamnog blokiranja



Modeliranje uzajamnog blokiranja pomoću grafa dodele resursa

- Ako graf dodele resursa **nema cikluse**, tada nijedan proces u sistemu nije u uzajamnom blokiranju
- Ako graf dodele resursa **ima cikluse**, tada **može** postojati uzajamno blokiranje
- Ako svaki tip resursa ima po **1** instancu:
 - Ciklus u grafu dodele resursa je **potreban i dovoljan uslov** za postojanje uzajamnog blokiranja
- Ako svaki tip resursa ima **više instanci**:
 - Tada postojanje ciklusa u grafu dodele resursa je **potreban**, ali **nije dovoljan** uslov za pojavu uzajamnog blokiranja



Uslovi za uzajamno blokiranje

Uslovi za moguće uzajamno blokiranje (potrebni)

1. **Uzajamno isključivanje** (*Mutual exclusion*)
 - U datom trenutku samo jedan proces može koristiti resurs
2. **Drži i čekaj** (*Hold-and-wait*)
 - Proces drži dodeljene resurse dok čeka dodeljivanje ostalih resursa
3. **Bez prekidanja** (*No preemption*)
 - Nijedan resurs se ne može nasilno oduzeti procesu koji ga drži

Uzajamno blokiranje nastaje kad su ispunjena prethodna tri uslova, i:

4. **Kružno čekanje** (*Circular wait*)
 - Postoji zatvoreni „lanac“ procesa, takav da svaki proces drži bar jedan resurs koji je potreban sledećem procesu u „lancu“
 - Ukoliko ima više instanci resursa ne mora da dodje do uzajamnog blokiranja

Konkurentnost: uzajamno blokiranje i gladovanje

Operativni sistemi



Potrebni i dovoljni uslovi za pojavu uzajamnog blokiranja

- ➊ Ako su prisutna samo prva 3 uslova **postoji mogućnost** za pojavu uzajamnog blokiranja –
 - ▣ Ovi uslovi se odnose na politiku dodele
- ➋ Da bi **nastalo** uzajamno blokiranje potrebno je da istovremeno budu prisutna sva 4 uslova
 - ▣ Kružno čekanje je okolnost koja zavisi od sekvence dodele i oslobođanja resursa od strane uključenih procesa



Metode za upravljanje uzajamnim blokiranjem

- ❖ **Sprečavanje** uzajamnog blokiranja
 - ❖ Onemogućiti nastanak nekog od 4 uslova za pojavu uzajamnog blokiranja
- ❖ **Izbegavanje** uzajamnog blokiranja
 - ❖ Donositi odgovarajuće dinamičke odluke o dodeli resursa u skladu sa trenutnim stanjem dodeli resursa
- ❖ **Otkrivanje** uzajamnog blokiranja i **oporavak**
 - ❖ Uдовoljiti zahtevu za resursom kada je to moguće, ali periodično proveravati da li je uzajamno blokiranje prisutno i tada oporaviti sistem od uzajamnog blokiranja



Sprečavanje uzajamnog blokiranja

- ➊ OS se projektuje tako da isključi mogućnost pojave uzajamnog blokiranja
- ➋ Dve klase metoda:
 - ▣ Indirektne metode sprečavanja uzajamnog blokiranja
 - Sprečavanje pojave jednog od 3 potrebna uslova
 - ▣ Direktne metode sprečavanja uzajamnog blokiranja
 - Sprečavanje pojave kružnog čekanja
- ➌ Ove metode vode neefikasnom korišćenju resursa i neefikasnom izvršenju procesa



Sprečavanje uslova “Uzajamno isključivanje”

- Ako nema resursa koji se dodeljuju ekskluzivno samo jednom procesu, nema uzajamnog blokiranja
- To nije moguće – **uzajamno isključivanje se mora zadržati za nedeljive resurse**
- Primer nedeljivog resursa: printer
- Primer deljivog resursa: *read-only* datoteka

Sprečavanje uslova "Drži i čekaj"

- ❖ Treba garantovati da kad proces zahteva neki resurs, ne sme držati nijedan drugi resurs
- ❖ Može se koristiti pristup kojim će se zahtevati od svakog procesa da pre izvršenja zatraži sve potrebne resurse
 - ❖ Proces se blokira dok mu se svi traženi resursi dodele
 - ❖ Procesi mogu dugo čekati udovoljenje svim zahtevima za resursima
 - ❖ Resursi dodeljeni procesu mogu ostati dugo neiskorišćeni iako bi mogao da ih koristi neki drugi proces
 - ❖ Proces bi morao da zna sve svoje zahteve za resursima
- ❖ Alternativa je da se koristi pristup po kome proces mora osloboditi sve resurse pre nego što zahteva dodatni resurs
- ❖ Oba pristupa imaju dva nedostatka:
 - ❖ **Iskorišćenje resursa** je malo kako je prikazano kod prvog pristupa
 - ❖ **Gladovanje procesa** – proces koji zahteva popularne resurse može beskonačno dugo čekati



Sprečavanje uslova "Bez prekidanja"

- **Pristup 1:** Ako proces drži neke resurse i zahteva drugi resurs koji mu se ne može odmah dodeliti (tj. proces mora da čeka), tada se svi resursi koje proces drži "otpuštaju" (oslobađaju) i proces će ih ponovo zahtevati sa dodatnim resursima
- **Pristup 2:** Kada proces P zahteva resurse koji su dodeljeni nekom drugom procesu Q koji takođe čeka na dodelu resursa oduzimaju se resursi od procesa Q koji čeka i dodeljuju se procesu P. Problem ako su procesi istog prioriteta.
- Kad god proces mora da oslobodi resurs koji je već koristio, stanje tog resursa se mora pamtitи za kasniji nastavak procesa
- Stoga, ovaj protokol je praktičan samo za resurse čije se stanje može lako pamtitи i restaurirati, kao što je procesor

Sprečavanje uslova "Kružno čekanje"

- ❖ Strategija za sprečavanje kružnog čekanja
 - ❖ Definisati **striktno linearno uređenje O() za sve tipove resursa**
 - ❖ Svakom tipu resursa se dodeljuje jedinstven ceo broj tako da se resursi mogu urediti. Npr:
 - R1: jedinica trake $O(R1)=2$
 - R2: jedinica diska $O(R2)=4$
 - R3: štampači $O(R3)=7$
 - ❖ **Proces može zahtevati resurse samo u rastućem redosledu njihovog uređenja**
 - Proces inicijalno traži određeni broj instanci nekog tipa resursa, R_i
 - Ako proces zahteva više instanci istog tipa resursa mora ih tražiti jednim zahtevom
 - Ako je procesu dodeljen resurs R_i , može zahtevati instance resursa R_j ako i samo ako je $O(R_j) > O(R_i)$
- ❖ Sprečavanje kružnog čekanja može biti neefikasno, usporava procese i nepotrebno odbija pristup resursima

Izbegavanje uzajamnog blokiranja

- ❖ Pažljivom dodelom resursa može se izbeći uzajamno blokiranje
 - ❖ Kada proces traži resurse, sistem proverava da li je bezbedno ili nebezbedno dodeliti tražene resurse
 - ❖ Ako je bezbedno, sistem može doneti odluku o dodeli
 - ❖ Ako nije bezbedno, sistem odbija dodelu resursa
- ❖ Pitanje je da li postoji algoritam koji omogućava sistemu da uvek napravi pravi izbor i da se na taj način uvek može izbeći uzajamno blokiranje
 - ❖ Potrebne su **unapred** informacije o tome kako će procesi zahtevati resurse
- ❖ Ova metoda se bazira na konceptu **bezbednih stanja** (*safe states*)

Stanje sistema

- **Stanje sistema** oslikava trenutnu dodelu resursa procesima

- $R = (R_1, R_2, \dots, R_m)$ - (*Resource*) ukupan broj svakog resursa u sistemu
- $V = (V_1, V_2, \dots, V_m)$ - (*Available*) trenutno raspoloživi resursi u sistemu
- $C = (C_{ij})$, $i=1, n$, $j=1, m$ - (*Claim*) max zahtevi procesa za resursima
- $A = (A_{ij})$, $i=1, n$, $j=1, m$ - (*Allocation*) trenutno dodeljeni resursi procesima (vrste predstavljaju procese, a kolone resurse)

- Važi sledeće:

1. Svi resursi su ili raspoloživi ili dodeljeni
$$R_j = V_j + \sum A_{ij} \quad | \quad i=1, n \quad \text{za svako } j=1, m$$
2. Nijedan proces ne može zahtevati više od ukupne količine resursa u sistemu
$$C_{ij} \leq R_j, \quad \text{za svako } i, j$$
3. Nijednom procesu nije dodeljeno više resursa od njegovih maksimalnih zahteva
$$A_{ij} \leq C_{ij}, \quad \text{za svako } i, j$$

Konkurentnost: uzajamno blokiranje i gladovanje

Operativni sistemi



Bezbedno stanje

- Stanje je **bezbedno** ako sistem može dodeliti resurse svakom procesu (do **max** broja resursa) u nekom redosledu i da pri tom izbegne uzajamno blokiranje
- Sistem je u bezbednom stanju samo ako postoji **bezbedna sekvenca procesa**
- Sekvenca procesa $\langle P_0, P_1, \dots, P_n \rangle$ je **bezbedna sekvenca** za tekuće stanje dodele resursa ako za svaki P_i , resursi koje proces P_i može još tražiti mogu biti zadovoljeni trenutno raspoloživim resursima i resursima koje drže svi procesi P_j , $j < i$, tj. uslov koji se mora zadovoljiti za proces P_i je
 $C_{ij} - A_{ij} \leq V_j$, za svako j
- Za svaki proces P_i i za svaki resurs R_j mora biti ispunjen uslov:

Broj zahtevanih – Broj dodeljenih \leq Broj raspoloživih resursa

Konkurentnost: uzajamno blokiranje i gladovanje

Operativni sistemi

Utvrđivanje bezbednog stanja

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

C - A

	R1	R2	R3
	9	3	6

Resource vector **R**

	R1	R2	R3
	0	1	1

Available vector **V**

Početno stanje

Procesu P2 su upravo dodeljeni 1 instanca R1 i 1 instanca R3

$R=(R_1, R_2, \dots, R_m)$ – ukupna količina svakog resursa u sistemu

$V=(V_1, V_2, \dots, V_m)$ - trenutno raspoloživi resursi u sistemu

$C=(C_{ij})$, za $i=1,n$, $j=1,m$ – maksimalni zahtevi procesa za resursima

$A=(A_{ij})$, za $i=1,n$, $j=1,m$ – trenutno dodeljeni resursi procesima

Utvrđivanje bezbednog stanja

	R1	R2	R3
P1	3	2	2
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

C - A

	R1	R2	R3
	9	3	6

Resource vector **R**

	R1	R2	R3
	6	2	3

Available vector **V**

Proces P2 se izvršava do kraja

$R=(R_1, R_2, \dots, R_m)$ – ukupna količina svakog resursa u sistemu

$V=(V_1, V_2, \dots, V_m)$ - trenutno raspoloživi resursi u sistemu

$C=(C_{ij})$, za $i=1, n$, $j=1, m$ – zahtev procesa za resursima

$A=(A_{ij})$, za $i=1, n$, $j=1, m$ – trenutno dodeljeni resursi procesima

Utvrđivanje bezbenog stanja

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	1	0	3
P4	4	2	0

C - A

	R1	R2	R3
	9	3	6

Resource vector **R**

	R1	R2	R3
	7	2	3

Available vector **V**

Proces P1 se izvršava do kraja

$R = (R_1, R_2, \dots, R_m)$ – ukupna količina svakog resursa u sistemu

$V = (V_1, V_2, \dots, V_m)$ - trenutno raspoloživi resursi u sistemu

$C = (C_{ij})$, za $i=1, n$, $j=1, m$ – zahtev procesa za resursima

$A = (A_{ij})$, za $i=1, n$, $j=1, m$ – trenutno dodeljeni resursi procesima

Utvrđivanje bezbednog stanja

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	0

C - A

	R1	R2	R3
	9	3	6

Resource vector **R**

	R1	R2	R3
	9	3	4

Available vector **V**

Proces P3 se izvršava do kraja, pa zatim i P4

$R=(R_1, R_2, \dots, R_m)$ – ukupna količina svakog resursa u sistemu

$V=(V_1, V_2, \dots, V_m)$ - trenutno raspoloživi resursi u sistemu

$C=(C_{ij})$, za $i=1, n$, $j=1, m$ – zahtev procesa za resursima

$A=(A_{ij})$, za $i=1, n$, $j=1, m$ – trenutno dodeljeni resursi procesima

Utvrđivanje nebezbednog stanja

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	1	0	2
P3	1	0	3
P4	4	2	0

C - A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
1	1	2

Available vector **V**

Početno stanje

$R=(R_1, R_2, \dots, R_m)$ – ukupna količina svakog resursa u sistemu

$V=(V_1, V_2, \dots, V_m)$ - trenutno raspoloživi resursi u sistemu

$C=(C_{ij})$, za $i=1, n$, $j=1, m$ – zahtev procesa za resursima

$A=(A_{ij})$, za $i=1, n$, $j=1, m$ – trenutno dodeljeni resursi procesima

Utvrđivanje nebezbednog stanja

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	2	0	1
P2	5	1	1
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	1	2	1
P2	1	0	2
P3	1	0	3
P4	4	2	0

C - A

	R1	R2	R3
	9	3	6

Resource vector **R**

	R1	R2	R3
	0	1	1

Available vector **V**

Proces P1 zahteva po jednu instancu resursa R1 i R3
i sistem je u nebezbednom stanju

$R = (R_1, R_2, \dots, R_m)$ – ukupna količina svakog resursa u sistemu

$V = (V_1, V_2, \dots, V_m)$ - trenutno raspoloživi resursi u sistemu

$C = (C_{ij})$, za $i=1, n$, $j=1, m$ – zahtev procesa za resursima

$A = (A_{ij})$, za $i=1, n$, $j=1, m$ – trenutno dodeljeni resursi procesima



Bankarov algoritam

- Predložio ga je Dijkstra (1965)
- Ime je dobio po tome što se princip koristi u bankama pri dodeli raspoloživog novca klijentima kako se ne bi došlo u situaciju da više ne mogu biti zadovoljene potrebe klijenata
- Proverava da li dodata resursa vodi u bezbedno stanje
 - Ako DA - vrši se dodata resursa
 - Ako NE – blokirati proces sve dok nije bezbedno dodeliti resurs (uslužiti zahtev)



Bankarov algoritam (pseudo kod)

● Potrebne strukture podataka:

- R – vektor ukupnog broja svakog resursa u sistemu
- V – vektor raspoloživih resursa
- A – matrica resursa trenutno dodeljenih procesima
- C – matrica maksimalnih zahteva procesa za resursima
 - $C[i,*]$ vektor maksimalnih zahteva procesa P_i
 - $C[i,j]=k$ znači da proces P_i traži maksimalno k instanci resursa R_j
- Q – vektor dodatnih zahteva procesa P_i za resursima– važi $Q = C_i - A_i$



Bankarov algoritam (pseudo kod)

1. **Ako** je $A[i, *] + Q[*] > C[i, *]$,
tada greška "process traži više od maksimuma"
inače preći na korak 2
2. **Ako** je $Q[*] > V[*]$
tada traženi resursi nisu raspoloživi i P_i mora da se blokira
Inače preći na korak 3
3. Simulira se dodela zahtevanih resursa $Q[i, *]$ procesu P_i i menja stanje dodele resursa:
$$V[i, *] = V[i, *] - Q[*] \quad // raspoloživi resursi$$
$$A[i, *] = A[i, *] + Q[*] \quad // dodeljeni resursi$$
4. Proverava se da li je novo stanje bezbedno (koristi se **algoritam ispitivanja bezbednosti** sa sledećeg slajda)
5. **Ako** jeste,
tada se procesu P_i stvarno dodeljuju traženi resursi;
inače proces P_i se blokira i čeka da mu se zahtevani resursi $Q[*]$ dodele, a staro stanje se restaurira;

Konkurentnost: uzajamno blokiranje i gladovanje

Operativni sistemi



Algoritam ispitivanja bezbednosti (pseudo kod)

- ➊ **Algoritam ispitivanja bezbednosti** proverava da li je sistem u bezbednom stanju

1. Inicijalizacija

$$TrenutnoRaspolozivi = V$$

$$Rest = \{svi procesi\}$$

2. Ponavljati korake 3 i 4
3. Naći Pi u $Rest$ za koji važi

$$C[i, *] - A[i, *] \leq Trenutno Raspolozivi$$

4. **Ako** takav Pi postoji **tada**

$$Trenutno Raspolozivi = Trenutno Raspolozivi + A[i, *]$$

$$Rest = Rest - \{Pi\}$$

inače preći na korak 5

5. **Ako** je $Rest = \{\}$ **tada** je sistem u bezbednom stanju
inače sistem je u nebezbednom stanju



Logika izbegavanja uzajamnog blokiranja – Bankarov algoritam

```
struct state {  
    int resource[m];  
    int available[m];  
    int claim[n][m];  
    int alloc[n][m];  
}
```

(a) global data structures

```
if (alloc [i,*] + request [*] > claim [i,*])  
    < error >; /* total request > claim */  
else if (request [*] > available [*])  
    < suspend process >;  
else {  
    < define newstate by:  
    alloc [i,*] = alloc [i,*] + request [*];  
    available [*] = available [*] - request [*] >;  
}  
if (safe (newstate))  
    < carry out allocation >;  
else {  
    < restore original state >;  
    < suspend process >;  
}
```

(b) resource alloc algorithm

Konkurentnost: uzajamno blokiranje i gladovanje



Logika izbegavanja uzajamnog blokiranja - Bankarov algoritam (2)

```
boolean safe (state S) {
    int currentavail[m];
    process rest[<number of processes>];
    currentavail = available;
    rest = {all processes};
    possible = true;
    while (possible) {
        <find a process Pk in rest such that
            claim [k,*] - alloc [k,*] <= currentavail;>
        if (found) {                                /* simulate execution of Pk */
            currentavail = currentavail + alloc [k,*];
            rest = rest - {Pk};
        }
        else possible = false;
    }
    return (rest == null);
}
```

(c) test for safety algorithm (banker's algorithm)

Otkrivanje i oporavak

- Sistem ne pokušava da spreči pojavljivanje uzajamnog blokiranja
 - Strategije sprečavanja i izbegavanja uzajamnog blokiranja ograničavaju pristup resursima i nameću ograničenja procesima
- Umesto toga, sistem dozvoljava da se uzajamno blokiranje pojavi, a onda uključuje mehanizme da otkrije uzajamno blokiranje i da oporavi sistem, tj. da raskine uzajamno blokiranje
- Provera uzajamnog blokiranja može se vršiti:
 1. Pri svakom zahtevu za resursima, ili
 2. Povremeno – zavisno od toga kolika je verovatnoća pojave uzajamnog blokiranja
- Prva varijanta je bolja u tome što se ranije otkriva uzajamno blokiranje i algoritam je jednostavan, ali česte provere troše značajno procesorsko vreme



Algoritam otkrivanja uzajamnog blokiranja

Resursi sistema - R
(R₁,R₂,R₃,...,R_m)

Matrica dodele

A ₁₁	A ₁₂	A ₁₃	...	A _{1m}
A ₂₁	A ₂₂	A ₂₃	...	A _{2m}
.
.
.
A _{n1}	A _{n2}	A _{n3}	...	A _{nm}

Raspoloživi resursi - V
(V₁,V₂,V₃,...,V_m)

Matrica dodatnih zahteva

Q ₁₁	Q ₁₂	Q ₁₃	...	Q _{1m}
Q ₂₁	Q ₂₂	Q ₂₃	...	Q _{2m}
.
.
.
Q _{n1}	Q _{n2}	Q _{n3}	...	Q _{nm}

R – vektor tipova resursa koji postoje u sistemu; R[j] broj instanci tipa resursa R_j

V – vektor raspoloživih resursa; v[j] broj raspoloživih instanci tipa resursa r_j

A – matrica tekuće dodele resursa; vrste procesi, kolone tipovi resursa

Q – matrica **dodatnih zahteva za resursima** Q = C - A

Konkurentnost: uzajamno blokiranje i gladovanje

Operativni sistemi



Algoritam otkrivanja uzajamnog blokiranja

- ➊ Algoritam označava sve procese koji **nisu** uzajamno blokirani
 - ▢ Inicialno svi procesi su **neoznačeni**
- ➋ Označiti svaki proces čija vrsta u matrici dodele A sadrži sve nule
- ➌ Inicijalizovati privremeni vektor W koji je jednak vektoru raspoloživih resursa V
- ➍ U matrici zahteva Q naći neoznačeni proces Pi koji zadovoljava uslov $Q_i \leq W$, $k=1,m$. Ako takav proces ne postoji, tada završiti algoritam
- ➎ Ako takav proces postoji, označiti proces Pi i vektoru W dodati i-tu vrstu matrice A ($W_k = W_k + A_{ik}$, $k=1,m$), i vratiti se na korak 3
- ➏ Uzajamno blokiranje postoji **ako i samo ako** postoje **neoznačeni procesi** po završetku algoritma

Konkurentnost: uzajamno blokiranje i gladovanje

Operativni sistemi



Algoritam otkrivanja uzajamnog blokiranja

- U koraku 3 traži se proces koji može da se izvršava – njegovim zahtevima za resursima sistem može odgovoriti.
- Takav proces se izvršava i nakon toga oslobađa resurse koje drži.
- Resursi se vraćaju u raspoložive, dodaju se u W (korak 4) - Proces se označava kao obrađen.
- Uzajamno blokiranje postoji ako i samo ako postoje **neoznačeni procesi** po završetku algoritma

Otkrivanje uzajamnog blokiranja

Primer otkrivanja uzajamnog blokiranja

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

Request matrix Q

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

Allocation matrix A

	R1	R2	R3	R4	R5	
Resource vector	2	1	1	2	1	
Available vector						
	R1	R2	R3	R4	R5	
Available vector	0	0	0	0	1	

1. Označava se P4 jer nema alociranih resursa
2. Postavlja se $W=(0\ 0\ 0\ 0\ 1)$
3. Zahtev Procesa P3 je \leq od W, označava se P3 i postavlja $W = W + (0\ 0\ 0\ 1\ 0) = (0\ 0\ 0\ 1\ 1)$
4. Nema neoznačenih procesa čiji su zahtevi $Q_i \leq W$ pa se algoritam završava

Pošto su P1 i P2 neoznačeni, uzajamno su blokirani



Oporavak iz uzajamnog blokiranja

1. Prekinuti sve procese koji su uzajamno blokirani
2. Vratiti sve uzajamno blokirane procese do neke prethodno definisane kontrolne tačke i tada restartovati sve procese
 - Ovo zahteva postojanje *rollback* i *restart* mehanizama ugrađenih u OS
3. Sukcesivno prekidati procese koji su uzajamno blokirani sve dok postoji uzajamno blokiranje
 - Redosled selektovanja procesa koji će biti prekinuti se zasniva na kriterijumu minimalnih troškova. Bira se proces koji ima:
 - najmanje korišćeno procesorsko vreme, najmanju količinu generisanog izlaza, najduže procenjeno preostalo vreme, najmanje alociranih resursa, najmanji prioritet, itd.
 - Nakon svakog prekidanja treba proveriti da li još uvek postoji uzajamno blokiranje algoritmom otkrivanja uzajamnog blokiranja
4. Selektivno oduzeti resurse procesima sve dok postoji uzajamno blokiranje (korišćenjem algoritma detekcije posle svakog oduzimanja).
 - Proces kome su oduzeti resursi se mora vratiti do tačke izvršenja pre dodele resursa

Konkurentnost: uzajamno blokiranje i gladovanje

Operativni sistemi



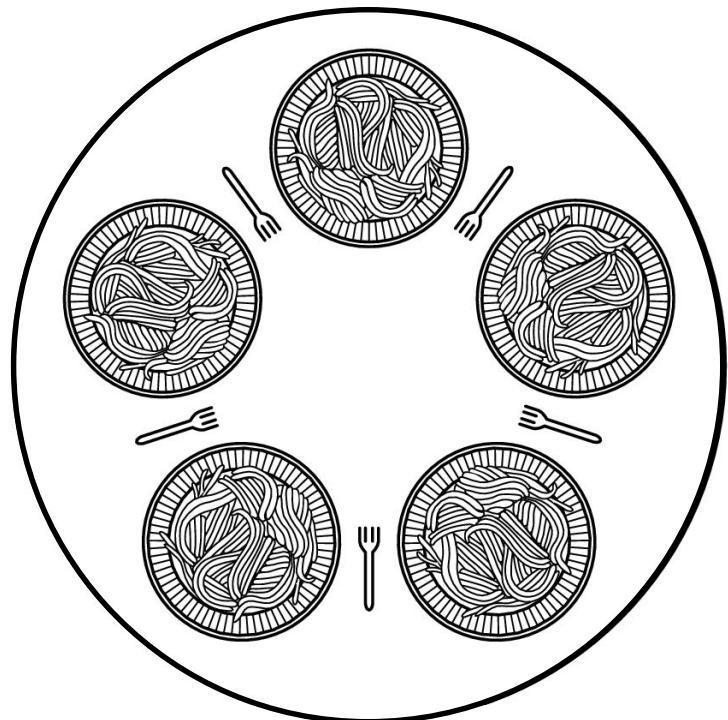
Strategije rešavanja uzajamnog blokiranja – rezime

Approach	Resource Allocation Policy	Different Schemes	Major Advantages	Major Disadvantages
Prevention	Conservative; undercommits resources	Requesting all resources at once	<ul style="list-style-type: none">• Works well for processes that perform a single burst of activity• No preemption necessary	<ul style="list-style-type: none">• Inefficient• Delays process initiation• Future resource requirements must be known by processes
		Preemption	<ul style="list-style-type: none">• Convenient when applied to resources whose state can be saved and restored easily	<ul style="list-style-type: none">• Preempts more often than necessary
		Resource ordering	<ul style="list-style-type: none">• Feasible to enforce via compile-time checks• Needs no run-time computation since problem is solved in system design	<ul style="list-style-type: none">• Disallows incremental resource requests
Avoidance	Midway between that of detection and prevention	Manipulate to find at least one safe path	<ul style="list-style-type: none">• No preemption necessary	<ul style="list-style-type: none">• Future resource requirements must be known by OS• Processes can be blocked for long periods
Detection	Very liberal; requested resources are granted where possible	Invoke periodically to test for deadlock	<ul style="list-style-type: none">• Never delays process initiation• Facilitates on-line handling	<ul style="list-style-type: none">• Inherent preemption losses

Večera filozofa

Dining Philosophers (1)

- ❖ Klasičan sinhronizacioni problem
- ❖ Pet filozofa jede (*eat*) i razmišlja (*think*)
- ❖ Da bi jeli potrebne su im dve viljuške (*forks*) - za stolom je samo 5 viljuški
- ❖ Algoritam mora zadovoljiti uzajamno isključivanje (dva filozofa ne mogu koristiti istovremeno istu viljušku) pri tom izbegavajući uzajamno blokiranje i gladovanje



Konkurentnost: uzajamno blokiranje i gladovanje

Rešenje problema večere filozofa

Prvo rešenje korišćenjem semafora

- Ukoliko svi procesi izvrše `wait(fork[i])` dolazi do uzajamnog blokiranja

```
/* program      diningphilosophers */
semaphore fork [5] = {1};
int i;
void philosopher (int i)
{
    while (true)
    {
        think();
        wait (fork[i]);
        wait (fork [(i+1) mod 5]);
        eat();
        signal(fork [(i+1) mod 5]);
        signal(fork[i]);
    }
}
void main()
{
    parbegin (philosopher (0), philosopher (1), philosopher (2),
              philosopher (3), philosopher (4));
}
```

Konkurentnost: uzajamno blokiranje i gladovanje



Rešenje problema večere filozofa

- Drugo rešenje uvodi brojački semafor koji dozvoljava da samo 4 filiozofa mogu da uđu u trpezariju

```
/* program diningphilosophers */
semaphore fork[5] = {1};
semaphore room = {4};
int i;
void philosopher (int I)
{
    while (true)
    {
        think();
        wait (room);
        wait (fork[i]);
        wait (fork [(i+1) mod 5]);
        eat();
        signal (fork [(i+1) mod 5]);
        signal (fork[i]);
        signal (room);
    }
}
void main()
{
    parbegin (philosopher (0), philosopher (1), philosopher (2),
              philosopher (3), philosopher (4));
}
```

Konkurentnost: uzajamno blokiranje i gladovanje



Rešenje korišćenjem monitora

- ➊ Koristi se vektor sa 5 uslovnih promenljivih **ForkReady**
 - Koriste se da bi omogućili filozofima da čekaju na raspoloživu viljušku
- ➋ Koristiti se vektor *boolean* koji registruje status viljuške
 - **true** viljuška slobodna
 - **false** viljuška zauzeta



Rešenje korišćenjem monitora

```
monitor dining_controller;
cond ForkReady[5];           /* condition variable for synchronization */
boolean fork[5] = {true};     /* availability status of each fork */

void get_forks(int pid)      /* pid is the philosopher id number */
{
    int left = pid;
    int right = (++pid) % 5;
    /*grant the left fork*/
    if (!fork(left))
        cwait(ForkReady[left]);          /* queue on condition variable */
    fork(left) = false;
    /*grant the right fork*/
    if (!fork(right))
        cwait(ForkReady[right]);          /* queue on condition variable */
    fork(right) = false;
}
void release_forks(int pid)
{
    int left = pid;
    int right = (++pid) % 5;
    /*release the left fork*/
    if (empty(ForkReady[left]))        /*no one is waiting for this fork */
        fork(left) = true;
    else                            /* awaken a process waiting on this fork */
        csignal(ForkReady[left]);
    /*release the right fork*/
    if (empty(ForkReady[right]))      /*no one is waiting for this fork */
        fork(right) = true;
    else                            /* awaken a process waiting on this fork */
        csignal(ForkReady[right]);
}
```

Konkurentnost: uzajamno blokiranje i gladovanje



Rešenje korišćenjem monitora (2)

```
void philosopher[k=0 to 4]           /* the five philosopher clients */
{
    while (true) {
        <think>;
        get forks(k);          /* client requests two forks via monitor */
        <eat spaghetti>;
        release forks(k);      /* client releases forks via the monitor */
    }
}
```

Konkurentnost: uzajamno blokiranje i gladovanje



UNIX mehanizmi konkurentosti

- ◆ UNIX obezbeđuje različite mehanizme za sinhronizaciju i komunikaciju procesa (pogledati detalje u praktikumu):
 - *Semaphore*
 - *Signals*
 - *Pipe*
 - *Message*
 - *Shared memory*



Linux Kernel mehanizmi konkurentnosti

- Linux uključuje sve mehanizme koji postoje u UNIX-u plus:
 - *Atomic operations*
 - *Spinlocks (Basic, Reader-Writer)*
 - *Semaphores*
 - Semafori na korisničkom nivou odgovaraju UNIX SVR4 semaforima
 - Semafori na nivou kernela samo za kernel procese: binarni, brojački i *reader-writer* semafori
 - *Barriers*
 - Da bi se obezbedio redosled instrukcija koje pristupaju memoriji, definišu se memorijske barijere



Solaris Thread sinhronizacione primitive

- Solaris dopunjuje mehanizme konkurentnosti u UNIX SVR4
 - *Mutual exclusion (mutex) locks*
 - *Semaphores*
 - *Multiple readers, single writer (readers/writer) locks*
 - *Condition variables*
- Ovi mehanizmi su dostupni nitima kernela i nitima korisničkih procesa



Windows mehanizmi konkurentosti

- Windows obezbeđuje sinhronizaciju između niti kao deo svoje objektne arhitekture.
 - Pogledati detalje u praktikumu (2. deo)
- Sinhronizacioni objekti koriste *Wait* funkciju
 - *WaitForSingleObject* blokira nit ukoliko kriterijum na kome se čeka nije ispunjen
- Sinhronizacioni objekti:
 - *Event, Mutex, Semaphore, Waitable timer, Process, Thread, ...*
- Objekti kritične sekcije - *critical sections*
- *Slim reader-writer locks, Condition variables.*



Domaći zadatak

◆ Poglavlje 6 Konkurentnost: uzajamno blokiranje i gladovanje

- 6.13 Ključni pojmovi, kontrolna pitanja i problemi

◆ Animacija

- Deadlocks (Resource Allocation Graph)

<https://apps.uttyler.edu/Rainwater/COSC3355/Animations/deadlock.htm>



Operativni sistemi

- Jednoprocesorsko
raspoređivanje -

Prof. dr Dragan Stojanović

Katedra za računarstvo
Univerzitet u Nišu, Elektronski fakultet



Literatura

- *Operating Systems: Internals and Design Principles*, edition, W. Stallings, Pearson Education Inc., 7th – 2012, (5th -2005, 6th - 2008, 8th – 2014 , 9th – 2017)
 - <http://williamstallings.com/OperatingSystems/>
 - <http://williamstallings.com/OperatingSystems/OS9e-Student/>
- Poglavlje 9: Jednoprocesorsko raspoređivanje
- Poglavlje 10:
 - 10.3 Raspoređivanje u sistemu Linux
 - 10.4 Raspoređivanje u sistemu Unix SVR4
 - 10.5 Raspoređivanje u sistemu Windows

Uvod u raspoređivanje (1)

- ❖ **Raspoređivanje (Planiranje)** je osnovna funkcija OS-a
- ❖ Gotovo svi resursi sistema se pre upotrebe planiraju
- ❖ **Planiranje CPU-a**, kao najvažnijeg resursa, ima centralno mesto u projektovanju OS-a
- ❖ CPU planiranje je osnova multiprogramske OS-a
- ❖ Prebacivanjem CPU-a među procesima, OS obezbeđuje efikasnije korišćenje CPU-a
- ❖ Cilj multiprogramiranja je da se svakog trenutka izvršava neki proces čime se obezbeđuje efikasno korišćenje CPU-a
- ❖ Kod jednoprocesorskih sistema u jednom trenutku se može izvršavati samo jedan proces, ostali čekaju spremni u memoriji da se CPU osloboodi i da ih OS izabere za izvršenje



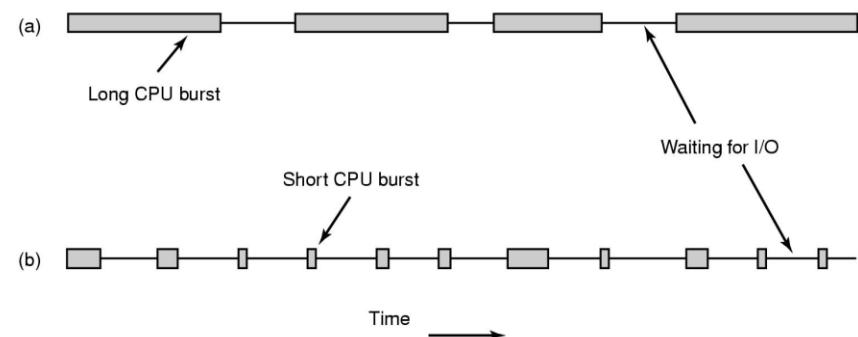
Uvod u raspoređivanje (2)

◆ Ideja multiprogramiranja je sledeća:

- Jeden proces se izvršava do trenutka kada treba da čeka na neki događaj (obično da se izvede neka UI operacija)
- Tog trenutka je CPU nezaposlen (*idle*)
- Kod multiprogramiranja se to vreme koristi
- OS oduzima CPU od procesa koji ga ne koristi i dodeljuje ga nekom od spremnih procesa
- Kojem procesu?
- Zavisno od algoritma planiranja

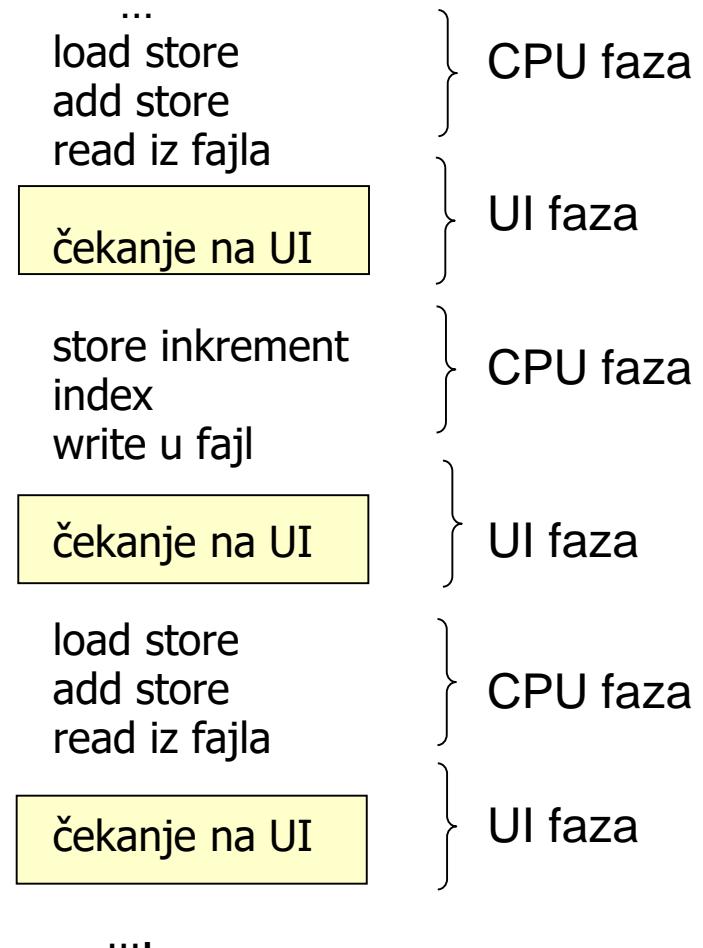
Ciklus CPU – UI (1)

- ◆ Procesi zahtevaju naizmenično korišćenje CPU-a i UI uređaja
- ◆ Izvršenje procesa je *ciklus* CPU izvršenje i UI čekanje. Procesi naizmenično menjaju ova dva stanja
- ◆ Svaki ciklus se sastoji od (obično veoma kratke) CPU faze (*bursta*) iza koga sledi (obično duža) UI faza (*burst*)
- ◆ Proces počinje i terminira se CPU fazom



- ◆ Postoje dve vrste procesa
 - (a) **Procesi orijentisani na CPU**
 - (b) **Procesi orijentisani na UI**
- ◆ Procesi (a) obično imaju duže CPU faze od procesa (b)

Ciklus CPU - UI (2)

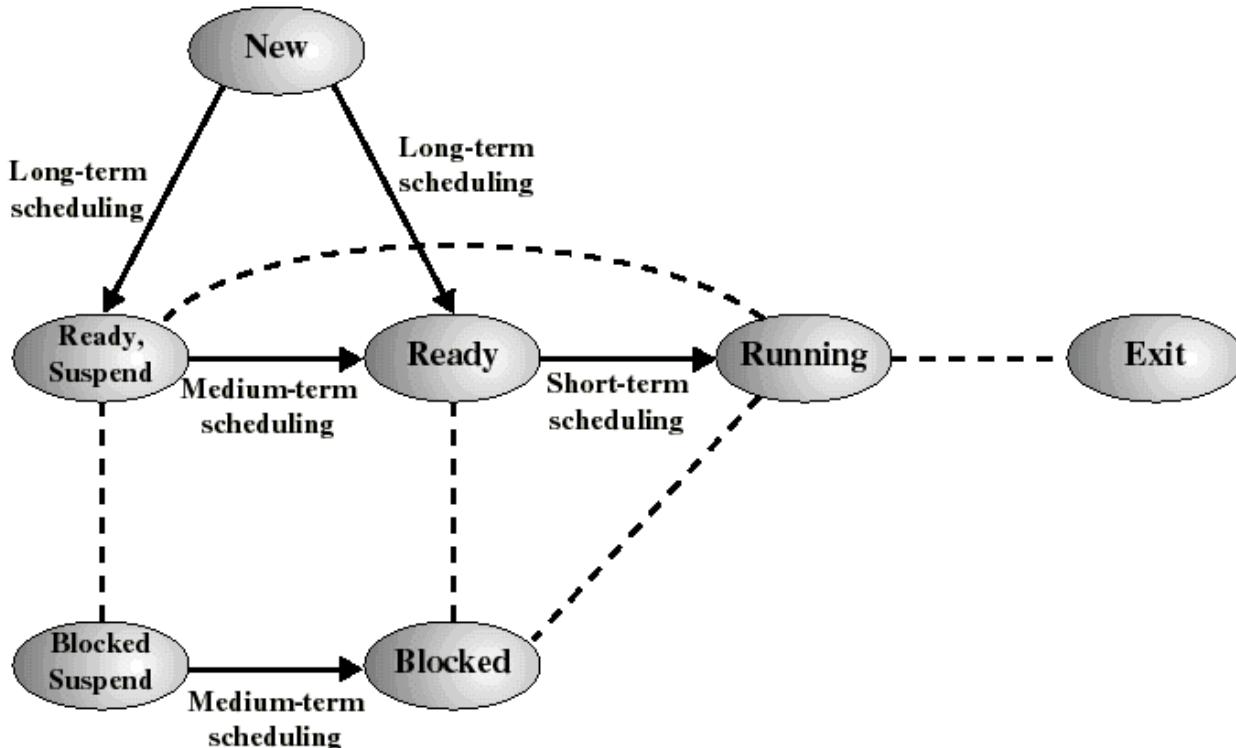




Raspoređivanje procesora

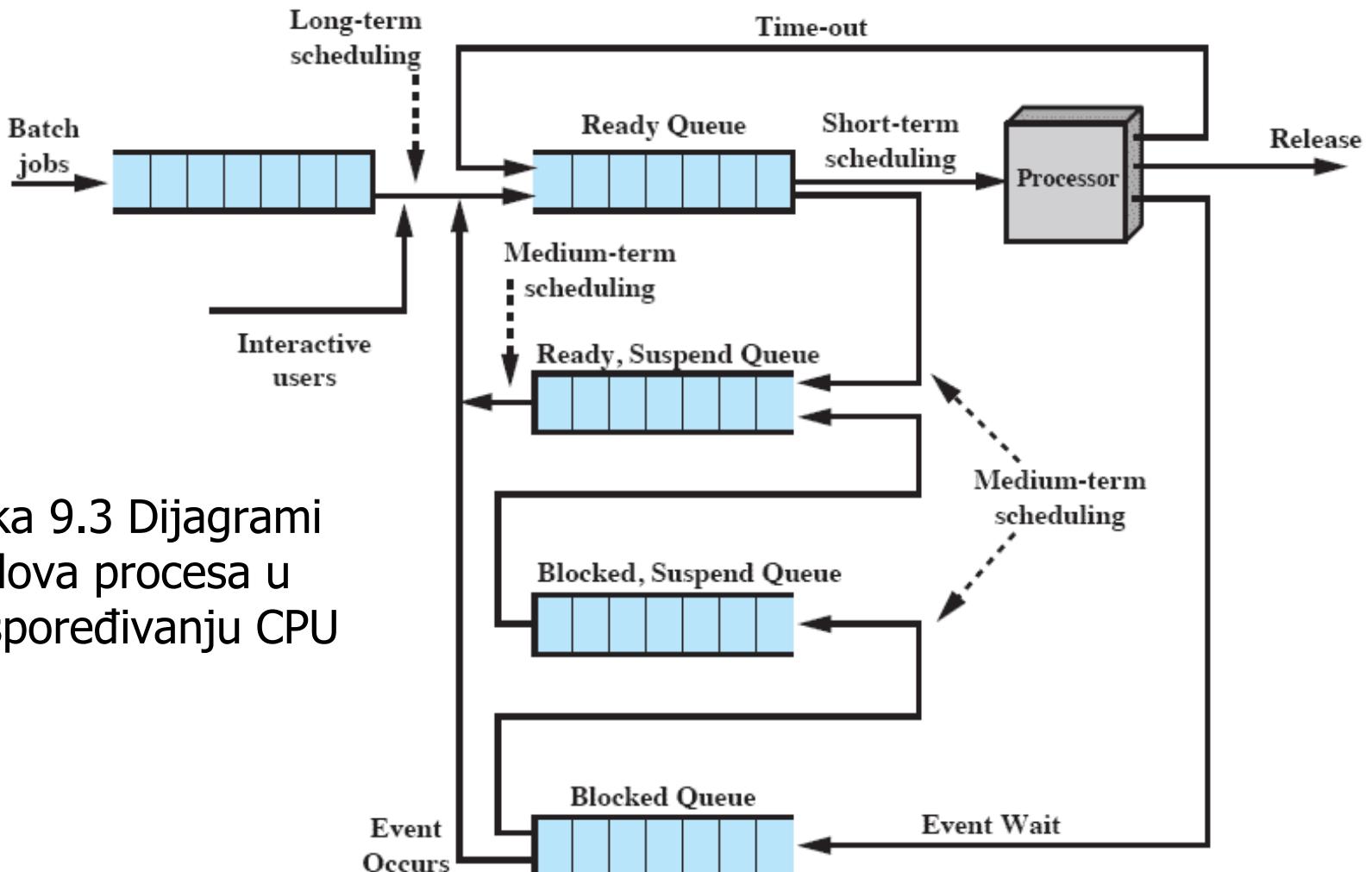
- ◆ Deo OS-a koji odlučuje kom će procesu iz liste spremnih procesa biti dodeljen CPU naziva se **kratkoročni raspoređivač** (*short-term scheduler*), **dispečer** (dispatcher), ili **CPU planer** (scheduler)
- ◆ Algoritmi koje CPU planer koristi nazivaju se **algoritmi raspoređivanja (planiranja)**
- ◆ Red spremnih procesa nije uvek implementiran kao FIFO red, i zavisno od algoritma planiranja može biti:
 - FIFO red, red po prioritetu, stablo, ili neuređena lančana lista
- ◆ Elementi reda spremnih procesa su upravljački blokovi procesa (PCB-ovi)
- ◆ Mi ćemo nadalje proučavati raspoređivanje kod **jednoprocesorskih** sistema

Tipovi raspoređivanja procesora



- **Dugoročno (Long-term):** koji proces pustiti u sistem - izvršava se prilikom kreiranja procesa
- **Srednjoročno (Medium-term):** koji proces preneti sa diska (swap područje) u glavnu memoriju
- **Kratkoročno (Short-term):** koji od spremnih procesa izabrati za izvršenje

Redovi raspoređivanja procesora



- Slika 9.3 Dijagrami redova procesa u raspoređivanju CPU

Dugoročno raspoređivanje

- ❖ Određuje koje kreirane procese pustiti u sistem na izvršenje
- ❖ Nakon izbora od strane dugoročnog planera, proces se dodaje u red kratkoročnog (u nekim OS – srednjoročnog) planera
- ❖ Kontroliše stepen multiprogramiranja
- ❖ Ako je više procesa aktivirano
 - ❖ Svaki proces dobija manji deo CPU vremena
 - ❖ Manja je verovatnoća da će svi procesi biti blokirani
 - Bolja iskorišćenost CPU
- ❖ Dugoročni planer pokušava da miksuje procese orijentisane na CPU i procese orijentisane na UI



Srednjoročno raspoređivanje

- Odlučuje koje procese preneti na disk ili koje uneti u memoriju sa diska za potrebe upravljanja multiprogramiranjem - upravlja funkcijom swapovanja (*swapping*)
- Saradjuje sa sistemom za upravljanje memorijom



Kratkoročno raspoređivanje

- ❖ Odlučuje koji će se proces sledeći izvršavati
- ❖ Kratkoročni raspoređivač se naziva još i **dispečer** (*dispatcher*)
- ❖ **Dispečer** se poziva kad god nastane događaj koji vodi blokiranju trenutno aktivnog procesa, ili koji obezbeđuje mogućnost da se prekine (*preempt*) trenutno aktivni proces u korist drugog procesa:
 - Prekid generatora takta
 - U/I prekidi
 - Sistemski pozivi operativnog sistema
 - Signali (na primer, od semafora)



Kada se vrši raspoređivanje?

- ➊ CPU raspoređivač donosi odluku u sledećim situacijama:
 - Kada se kreira novi proces
 - Kada se terminira neki proces
 - Kada se proces blokira na U/I ili semaforu (proces prelazi iz stanja aktivan u stanje blokiran)
 - Kada se javi U/I prekid (U/I je završen i proces prelazi iz stanja blokiran u stanje spremam)
 - Kada se javi prekid od generatora takta (kada proces prelazi iz stanja izvršenja u stanje spremam)



Kratkoročno raspoređivanje

- ➊ Kada OS izabere proces koji će se izvršavati neophodno je izvršiti promenu konteksta i aktiviranja izabranog procesa od tačke na kojoj je prekinut.
 - ▣ Prebacivanje konteksta podrazumeva čuvanje sadržaja registra CPU prekinutog procesa u okviru njegovog PCB i učitavanje prethodno zapamćenih vrednosti izabranog procesa u registre CPU
 - ▣ Prebacivanje u korisnički režim rada (*user mode*)
 - ▣ Prelaz na određenu lokaciju unutar izabranog programa koji je izabran od strane planera da bi se program restartovao od naredbe na kojoj je prethodno prekinut (istekom vremenskog kvanta), ili se sam blokirao
- ➋ **Dispečer** treba da je što je moguće brži jer se poziva pri svakoj promeni procesa
- ➋ Vreme koje dispečer potroši da bi stopirao jedan proces i startovao izabrani proces se naziva ***kašnjenje dispečerizacije (dispatch latency)***



Kriterijumi kratkoročnog raspoređivanja

◆ Orijentisani ka korisniku

- **Vreme odziva (Response time):** Za interaktivne procese to je vreme koje protekne od slanja zahteva do početka prijema odgovora
- **Vreme zadržavanja/prolaska (Turnaround time):** Vreme koje protekne od aktiviranja do kompletiranja procesa
 - Uključuje vreme izvršenja plus vreme koje proces provede čekajući na resurse uključujući i CPU

◆ Orijentisani ka sistemu

- **Propusna moć (Throughput):** broj kompletiranih procesa u jedinici vremena. Za duge poslove to može biti 1 proces na sat, za kratke transakcije deo procesa u sekundi
- **Iskorišćenost procesora (CPU utilization):** kreće se u opsegu 0 do 100%. Obično je 40% (lako opterećeni sistemi) do 90% (teško opterećeni sistemi)
- **Vreme čekanja (Waiting time):** vreme koje proces provede čekajući u redu spremnih



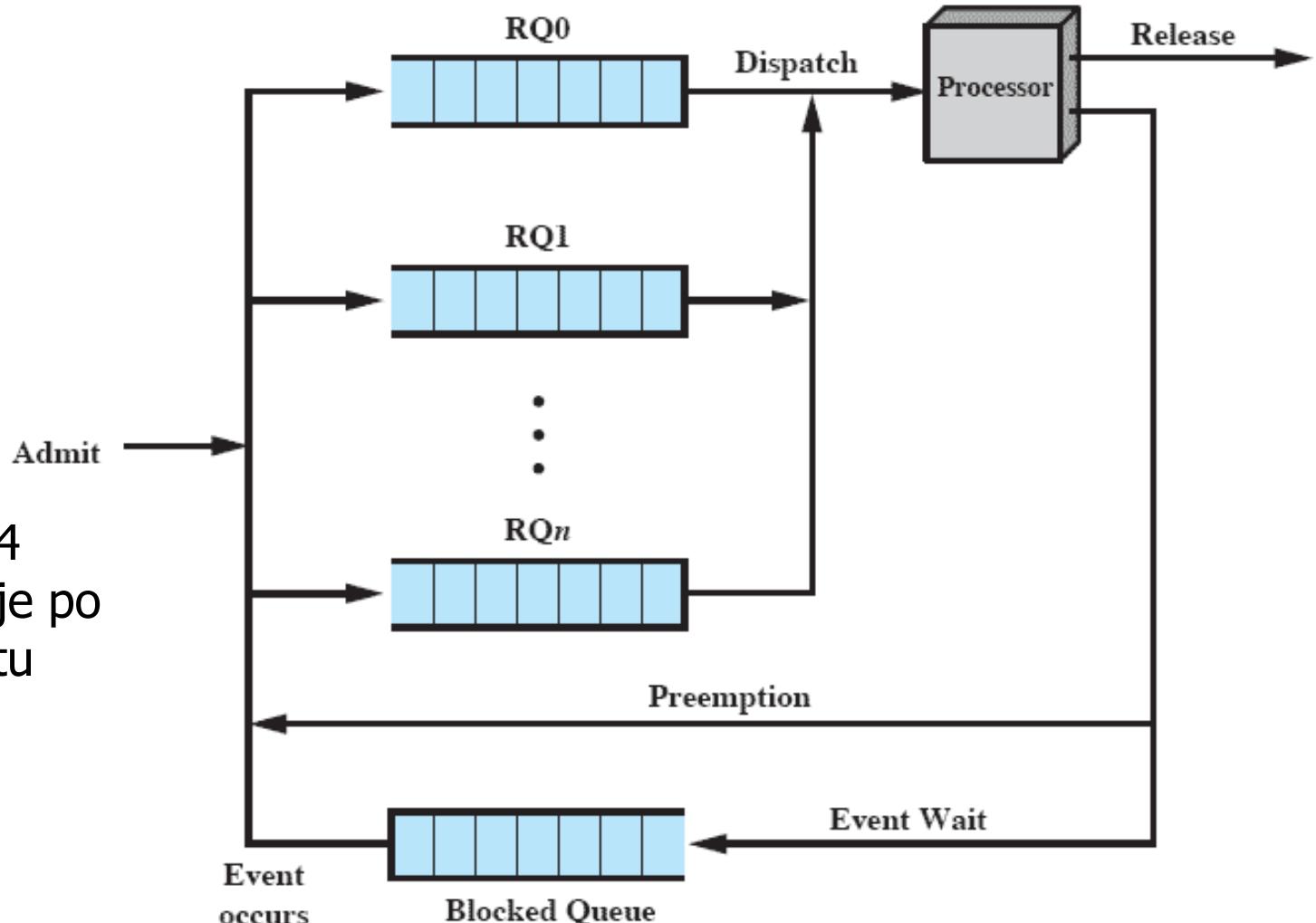
Ciljevi CPU raspoređivanja

- ➊ Maksimizirati iskorišćenost CPU i propusnu moć
- ➋ Minimizirati vreme zadržavanja, vreme čekanja i vreme odziva
- ➌ Pošto su ovi kriterijumi međusomo zavisni nije moguća njihova istovremena optimizacija
 - ▢ Na primer, obezbeđivanje brzog vremena odziva može zahtevati algoritam planiranja koji će često izvršavati promenu konteksta procesa, čime se povećava režijski rad OS (*overhead*) i smanjuje propusna moć
- ➍ Za interaktivne sisteme se predlaže minimiziranje variranja vremena odziva
 - ▢ Poželjno je da sistem ima predvidljivo vreme odziva

Raspoređivanje po prioritetu

- U mnogim sistemima svakom procesu je dodeljen prioritet i planer uvek bira proces višeg prioriteta u odnosu na proces nižeg prioriteta
- U tom slučaju održava se više redova spremnih procesa u opadajućem redosledu njihovih prioriteta
- Kada treba da se obavi selekcija, OS bira jedan proces iz prvog nepraznog reda (najvišeg prioriteta) po određenoj politici planiranja
- Procesi nižeg prioriteta mogu "gladovati" za CPU

Redovi po prioritetu



- Slika 9.4
Uređenje po
prioritetu

Jednoprocесорско rasporedivanje

Operativni sistemi

Algoritmi (politike) raspoređivanja

- ❖ **Funkcija selekcije:** određuje koji će proces iz reda spremnih procesa biti sledeći izabran za izvršenje
 - ❖ Može biti zasnovana na prioritetu procesa, zahtevima za resursima ili karakteristikama izvršenja procesa
- ❖ Ova funkcija se izražava koršćenjem tri veličine:
 - ❖ **w** = vreme provedeno u sistemu, u čekanju i izvršavanju
 - ❖ **e** = vreme provedeno u izvršavanju
 - ❖ **s** = totalno vreme zahtevano od strane procesa za izvršenje (vreme usluge) koje uključuje i *e*

Mod (režim) odluke

❖ **Mod odluke** definiše vremenski trenutak kada se izvršava funkcija selekcije

❖ **Bez prekidanja (nonpreemptive)**

- Kada je proces u stanju izvršenja, izvršava se sve dok se ne terminira ili dok se ne blokira zbog UI

❖ **Sa prekidanjem (preemptive)**

- OS može prekinuti proces koji se izvršava i premestiti ga u red spremnih procesa
- Omogućava bolje usluge jer nijedan proces ne može dugo monopolizovati procesor



Algoritmi raspoređivanja

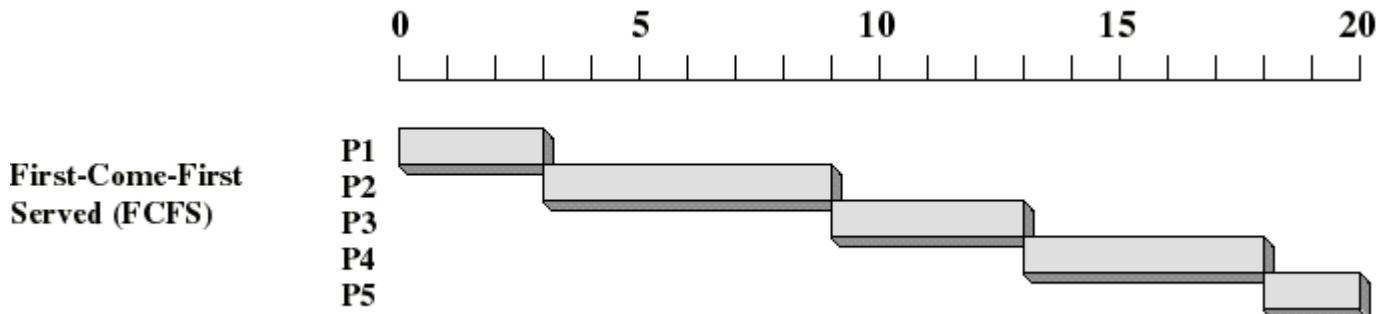
- *First Come First Served* (FCFS) – prvi došao prvi uslužen, bez prekidanja
- *Kružno planiranje* (Round-Robin)
- *Najkraći proces sledeći* (*Shortest Process Next* – SPN), bez prekidanja
- *Najkraće preostalo vreme* (*Shortest Remaining Time - SRT*), sa prekidanjem
- *Sledeći sa najvećim odnosom odziva* (*Highest Response Ratio Next* – HRRN)
- *Planiranje u redovima u više nivoa*, sa i bez povratne sprege (*Feedback*)

Primer koji će biti korišćen

Proces	Vreme pristizanja <i>(Arrival Time)</i>	Vreme obrade <i>(Service Time)</i>
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

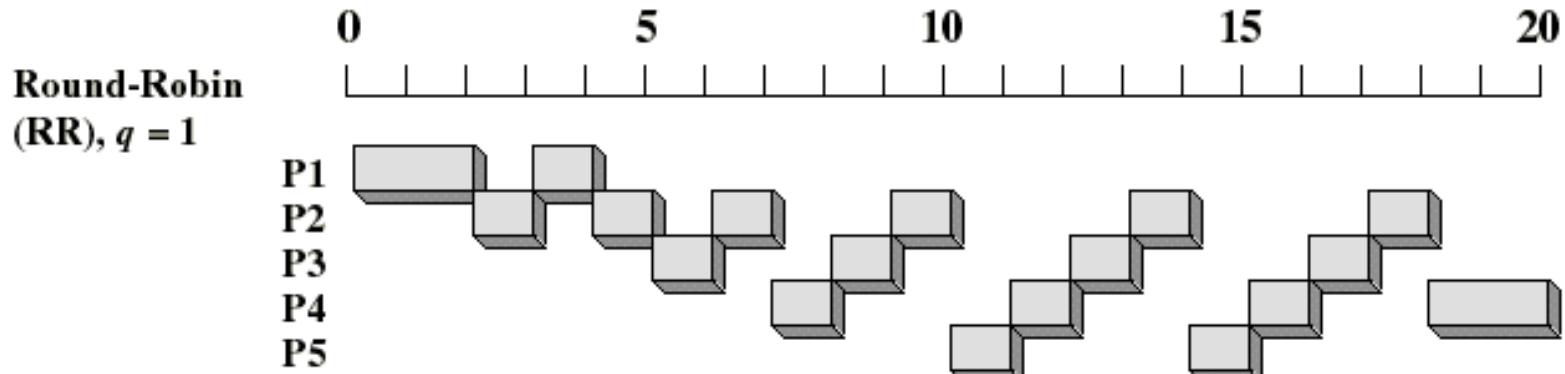
- Vreme obrade = ukupno procesorsko vreme potrebno u jednoj CPU fazi
- Procesi sa dugim vremenom obrade su procesi orijentisani na CPU i pominju se kao "dugi procesi"

Prvi došao prvi uslužen First Come First Served (FCFS)



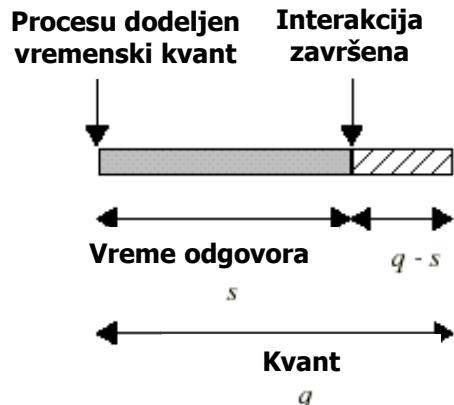
- **Funkcija selekcije:** proces koji najduže čeka u redu spremnih procesa (otuda, FCFS) - $\max[w]$
- **Mod odluke:** bez prekidanja (*nonpreemptive*)
 - Proces se izvršava dok se ne terminira, ili dok sam sebe ne blokira
- **Nedostaci:**
 - Proces koji nema U/I monopolisaće procesor
 - Favorizuje procese orijentisane na CPU
 - Procesi orijentisani na U/I čekaće čak i kada su završili U/I (loše iskorišćenje U/I uređaja)
 - Da bismo držali zauzetim U/I uređaje treba dati nešto veći prioritet procesima koji su orijentisani na U/I

Round-Robin (RR)



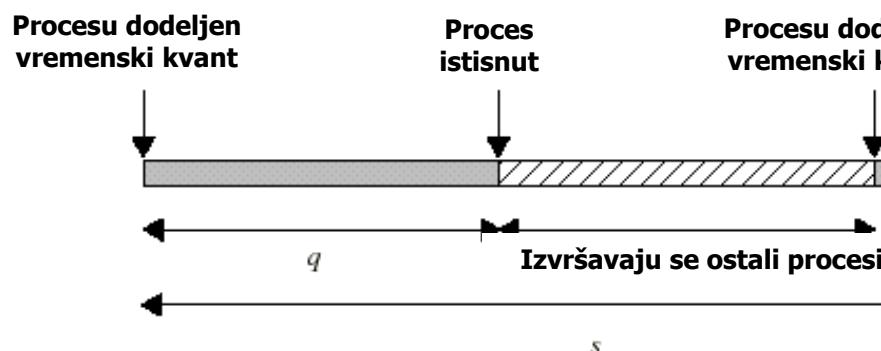
- ➊ **Funkcija selekcije:** ista kao kod FCFS
- ➋ **Mod odluke:** sa prekidanjem (*preemptive*)
 - ▣ Procesu se dopušta da se izvršava dok mu ne istekne dodeljeni vremenski period (kvant tipično 10 do 100 ms)
 - ▣ Tada se javlja prekidni signal od časovnika i proces se vraća u red spremnih procesa

Vremenski kvant za Round Robin



Vreme

Vremenski kvant veći od
tipične interakcije



Procesu dodeljen
vremenski kvant

Proces
istisnut

Procesu dodeljen
vremenski kvant

Interakcija
završena

Vremenski kvant manji od
tipične interakcije

- Vremenski kvant mora biti veći od vremena potrebnog za rukovanje prekidom od časovnika i promenu konteksta (*context switch*)
- Treba da je veći od tipične interakcije, ali ne mnogo veći da bi se izbeglo kažnjavanje procesa orijentisanih na UI



Nedostaci Round-Robin algoritma

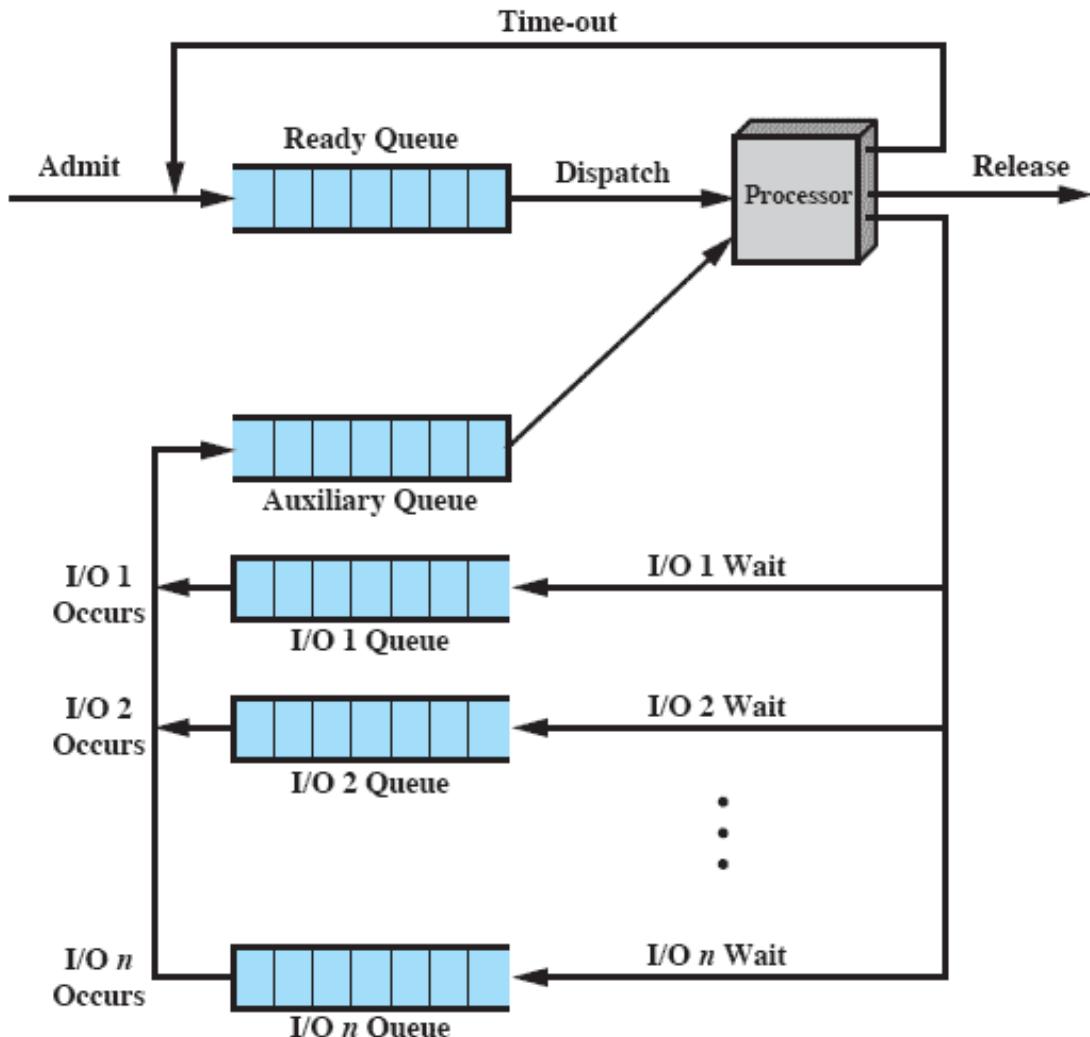
● Nedostaci:

- Još uvek se favorizuju procesi orijentisani na CPU
- Procesi orijentisani na UI koriste CPU kraće nego što je vremenski kvant i zatim se blokiraju čekajući na UI
- Procesi orijentisani na CPU izvršavaju se ceo vremenski kvant i vraćaju nazad u red spremnih procesa (tako da dolaze ispred blokiranih procesa)

● Rešenje: virtuelni Round-Robin (VRR)

- Kada se UI završi, blokirani proces se stavlja u pomoćni red spremnih procesa koji ima prednost u odnosu na glavni red spremnih procesa
- Proces izabran iz pomoćnog reda se izvršava onoliko dugo koliko mu je ostalo pre blokiranja zbog UI

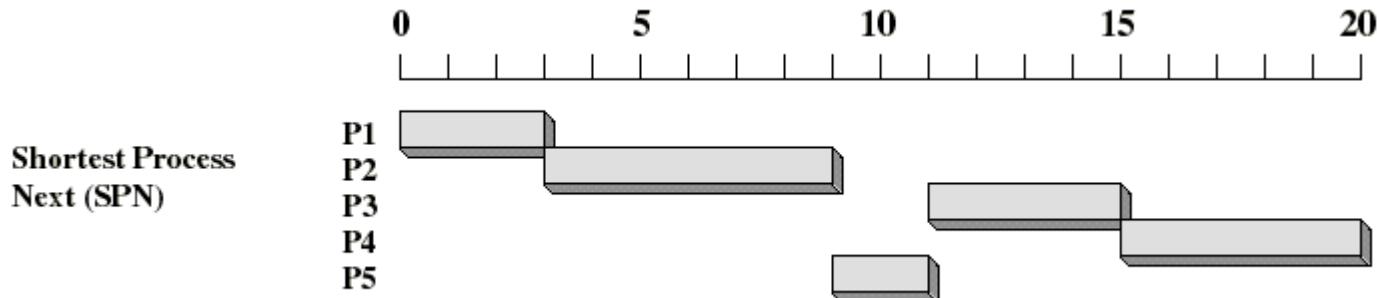
Virtuelni Round-Robin (VRR)



Jednoprocесорско rasporedivanje

Operativni sistemi

Najkraći posao sledeći Shortest Process Next (SPN)

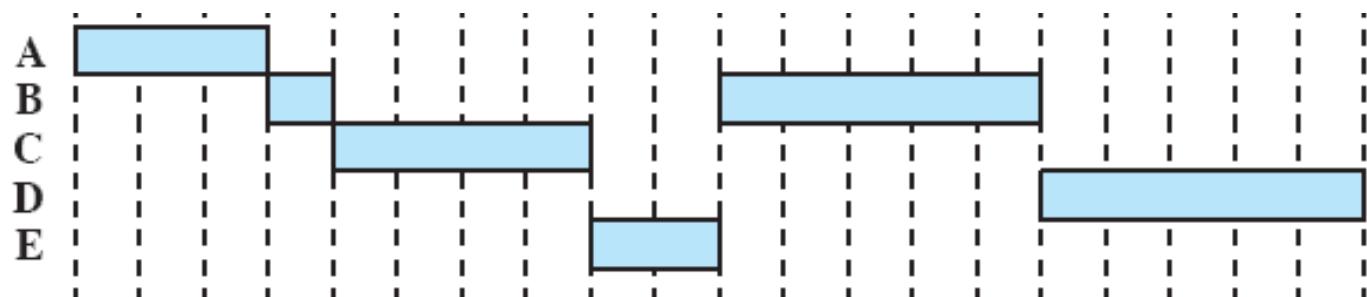


- ❖ **Funkcija selekcije:** proces sa najkraćim očekivanim vremenom obrade (*CPU burst time*) – **min[s]**
- ❖ **Mod odluke:** bez prekidanja
- ❖ Prvo se biraju procesi orijentisani na U/I
- ❖ Procena potrebnog vremena obrade $S_{n+1} = aT_n + (1-a)S_n$ ($0 < a < 1$)
- ❖ **Nedostaci:**
 - Moguće izgladnjivanje dugih procesa sve dok pristižu kraći poslovi
 - Nepostojanje prekidanja nije pogodno za sisteme sa vremenskom podelom
 - Procesi orijentisani na CPU implicitno imaju niži prioritet, dok bi procesi orijentisani na U/I mogli monopolizovati CPU ako stalno pristižu
 - SPN implicitno ugrađuje prioritet: prednost imaju najkraći poslovi

Najkraće preostalo vreme (*Shortest Remaining Time – SRT*)

- Verzija algoritma (politike) *Prvo najkraći posao sa prekidanjem*
- Treba proceniti vreme izvršenja i izabрати najkraće, pri čemu se trenutno aktivni proces može prekinuti (istisnuti) ako se pojavi proces sa manjim procenjenim vremenom izvršenja

Shortest Remaining Time (SRT)



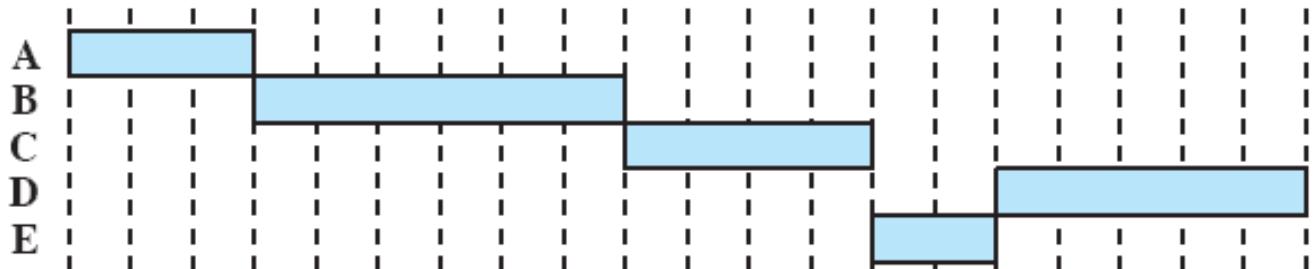
Sledeći sa najvišim odnosom odziva (HRRN)

- Bira sledeći proces sa najvišim odnosom (*Ratio*)

$$\text{Ratio} = \frac{\text{time spent waiting} + \text{expected service time}}{\text{expected service time}}$$

$$\text{Ratio} = \frac{w+s}{s}$$

Highest Response Ratio Next (HRRN)

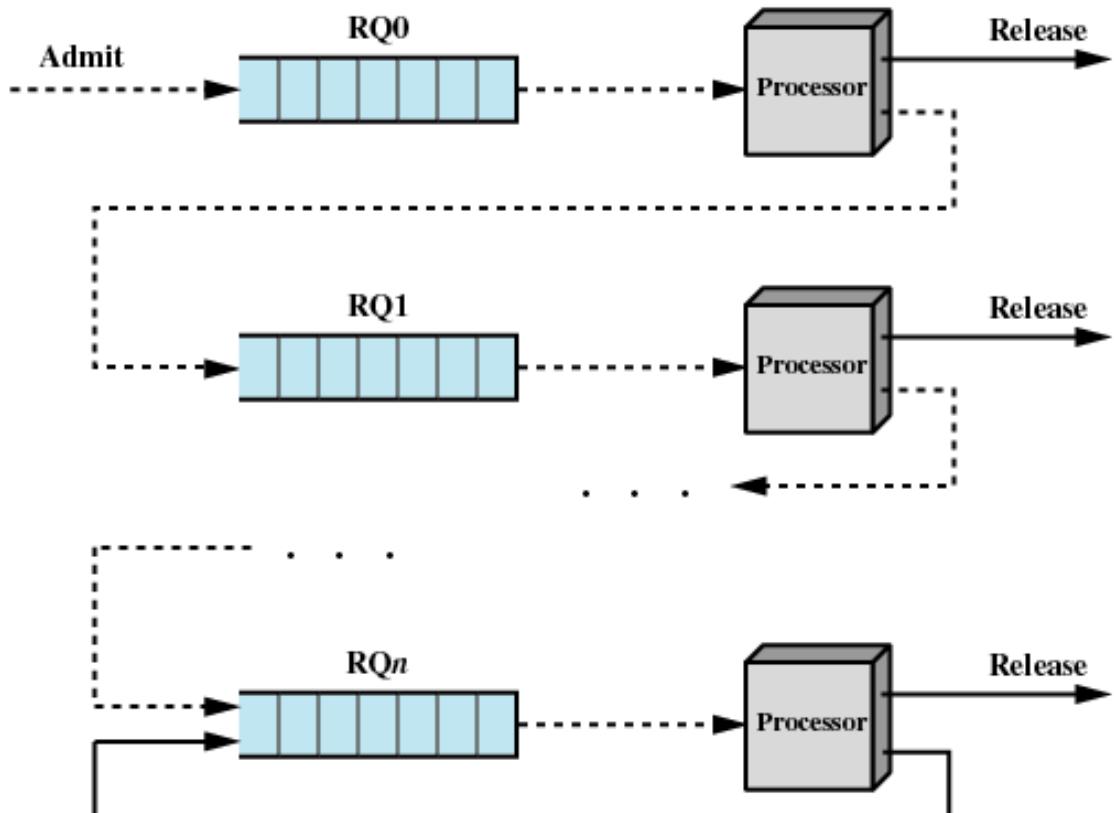




Planiranje u više redova sa povratnom spregom (Feedback)

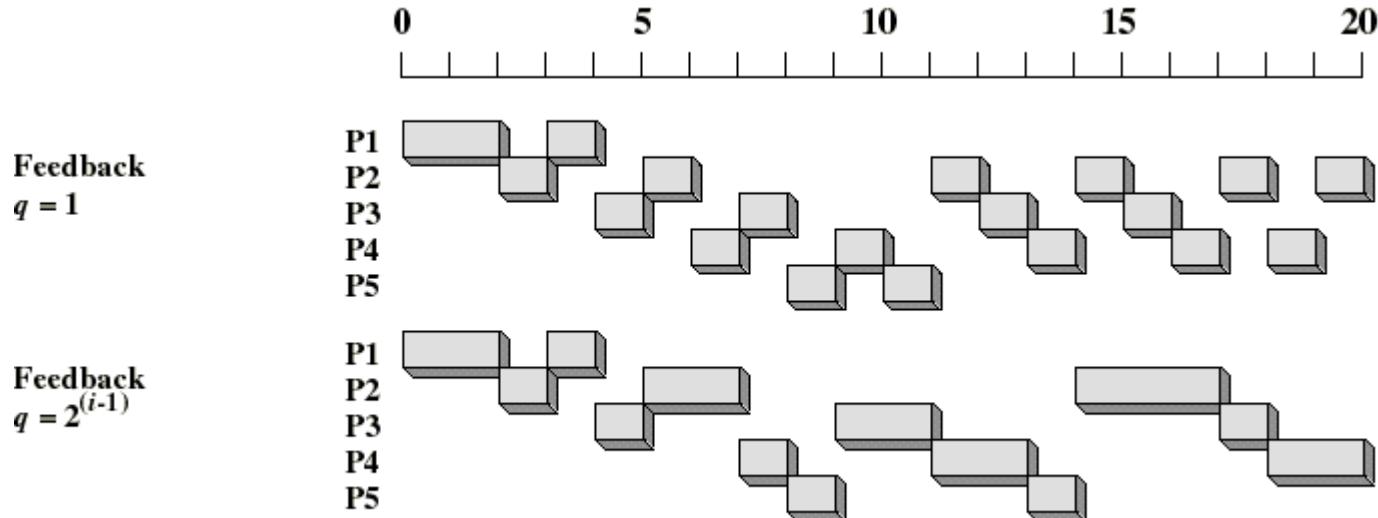
- To je raspoređivanje sa prekidanjem i dinamičkim prioritetima
- Postoji više redova spremnih procesa RQ_0, RQ_1, \dots, RQ_n (RQ - Ready Queue) različitog prioriteta:
 $P(RQ_0) > P(RQ_1) > \dots > P(RQ_n)$
- Novi proces se smešta u RQ_0
- Planer bira prvi proces iz RQ_0 i dodeljuje mu CPU na 1 vremenski kvant
- Kada istekne taj vremenski kvant, proces se premešta u RQ_1 , a zatim $RQ_2\dots$ dok ne dostigne RQ_n
- Procesi orijentisani na U/I ostaće u redu najvišeg prioriteta, dok će se procesi orijentisani na CPU pomerati u niže redove
- Planer za izvršenje bira proces iz RQ_i samo ako su prazni redovi RQ_{i-1} do RQ_0
- Obično je red RQ_n ciklični (round robin), pa proces tu kruži do terminiranja

Planiranje u više redova sa povratnom spregom (Feedback)



U svakom redu, osim u redu sa najnižim prioritetom, koristi se FCFS. U poslednjem redu koristi se Round Robin

Vremenski kvant za Feedback planiranje



- Sa fiksnom dužinom kvanta, vreme zadržavanja dužih procesa može biti alarmantno
- Da bi se kompenzovao ovaj nedostatak sa nivoom reda uvećava se vremenski kvant
 - Primer: vremenski kvant reda $RQ_i = 2^{i-1}$
- Duži procesi još uvek mogu patiti od "izgladnjivanja"
 - Moguće rešenje je posle izvesnog vremena prevesti proces u neki red višeg prioriteta



Karakteristike algoritama rasporedivanja

Table 9.3 Characteristics of Various Scheduling Policies

	FCFS	Round robin	SPN	SRT	HRRN	Feedback
Selection function	$\max[w]$	constant	$\min[s]$	$\min[s - e]$	$\max\left(\frac{w + s}{s}\right)$	(see text)
Decision mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Throughput	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
Response time	May be high, especially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
Overhead	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
Effect on processes	Penalizes short processes; penalizes I/O bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O bound processes
Starvation	No	No	Possible	Possible	No	Possible

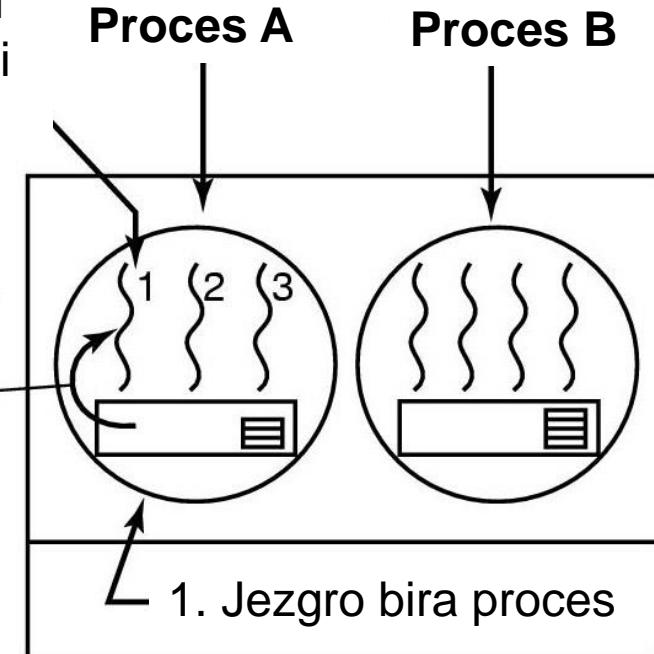
Jednoprocesorsko rasporedivanje

Operativni sistemi

Raspoređivanje niti na nivou korisnika

Redosled u
kome se niti
izvršavaju

2. Run-time
sistem bira
nit



Moguće: A1, A2, A3, A1, A2, A3

Nemoguće: A1, B1, A2, B2, A3, B3

Tanenbaum, 2014

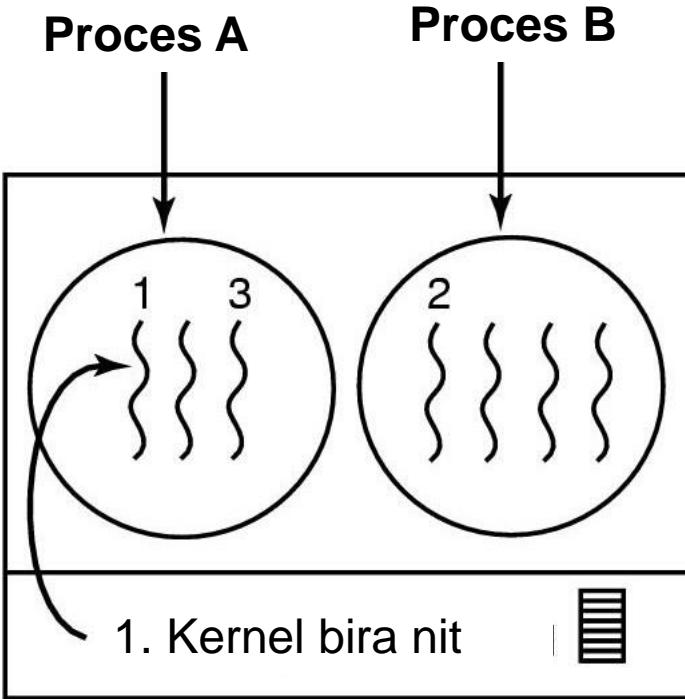
Moguće planiranje niti na korisničkom nivou

- ❖ 50-msec kvant procesa
- ❖ Niti se izvršavaju po 5 msec, jedna za drugom dok ne iskoriste ceo dodeljen kvant (CPU burst)

Jednoprocесорско raspoređivanje

Operativni sistemi

Raspoređivanje niti na nivou kernela



- Kernel bira nit unutar procesa koja će se izvršavati

Tanenbaum, 2014

Moguće: A1, A2, A3, A1, A2, A3

Takođe moguće: A1, B1, A2, B2, A3, B3

- Razlike u planiranju niti na nivou kernela i na nivou korisnika:
 - Performanse - Zamena korisničkih niti u okviru procesa je daleko brža od zamene niti na nivou kernela koje zahteva kompletну promenu konteksta
 - Algoritam planiranja – Za planiranje niti na nivou korisnika može se implementirati algoritam planiranja specifičan za konkretnu aplikaciju

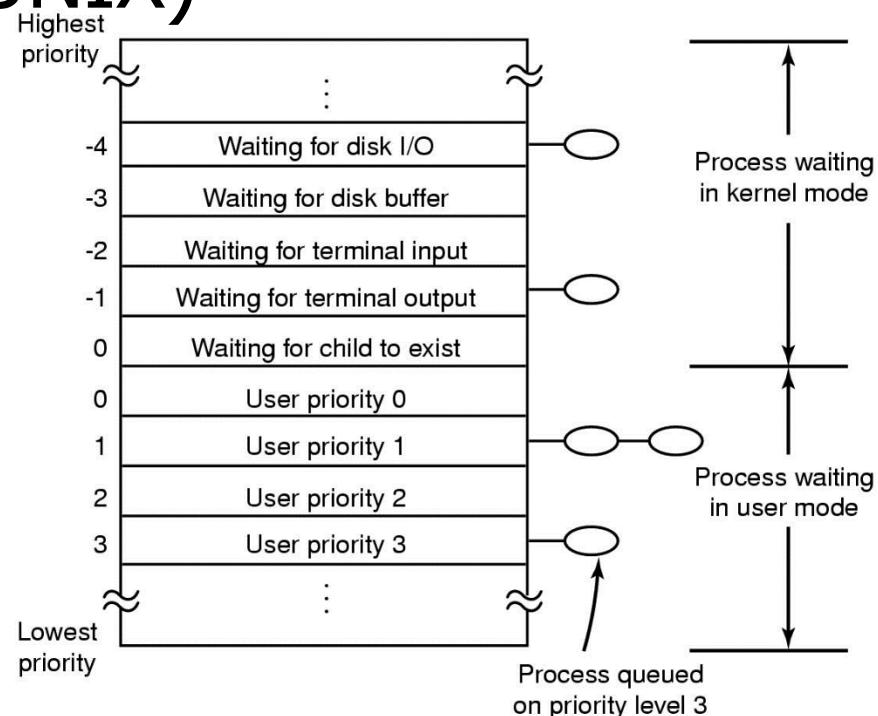
Jednoprocесорско raspoređivanje

Operativni sistemi

Raspoređivanje u UNIX

(System V R3, 4.3 BSD UNIX)

- ❖ Koristi se planiranje u **dva nivoa**:
 - ❖ Planer višeg nivoa premešta procese iz memorije na disk, i obrnuto
 - ❖ Planer nižeg nivoa (dispečer) bira proces kome će dodeliti CPU iz reda spremnih procesa
- ❖ Svaka verzija Unix-a ima nešto drugačiji algoritam nižeg nivoa, ali u suštini to je uvek **planiranje po prioritetu sa više redova**
- ❖ Procesi koji se izvršavaju u korisničkom režimu imaju pozitivan prioritet, a oni koji se izvršavaju u kernel modu negativan
- ❖ U redovima se nalaze samo spremni procesi koji su u memoriji
- ❖ Prioritet procesa se preračunava **svake sekunde**



- ❖ Planer pretražuje redove počev od reda najvišeg prioriteta i bira **prvi proces iz prvog nepraznog reda**
- ❖ Dodeljuje mu CPU na 1 kvant, nakon čega ga prekida, osim ako se proces nije u međuvremenu blokirao, i smešta na kraj reda odakle je izabran (RR planiranje)

Raspoređivanje u Unix (2)

- ❖ U redove prioriteta se smeštaju samo procesi spremni za izvršavanje.
- ❖ Planer pretražuje redove počev od reda najvišeg prioriteta.
- ❖ Aktivira se prvi proces iz prvog nepraznog reda i to za jedan vremenski kvant (100 ms-1s) ili dok se ne blokira, npr. na U/I zahtev.
- ❖ Kada završi sa dodeljenim kvantom proces se smešta na kraj istog reda
 - ❖ U okviru svakog reda se koristi Round Robin za izbor procesa
- ❖ Prioritet svakog procesa (*Priority*) se izračunava svake sekunde:
$$\text{Priority}(i) = \text{CPU}(i)/2 + \text{nice} + \text{Base}$$
 - ❖ $\text{CPU}(i)$ – mera korišćenja procesora od strane procesa u tokom poslednjeg vremenskog intervala i . Što je *CPU_usage* veći to je prioritet procesa manji.
$$\text{CPU}(i) = \text{CPU}(i-1)/2$$
 - ❖ *nice* – svakom procesu se dodeljuje ova vrednost (podrazumevano je 0). Sistemski poziv: *nice(value)* može se postaviti vrednost nice u opsegu 0-20 a administrator može dodeliti vrednost -20 – -1.
 - ❖ *Base* – Osnovni prioritet procesa
- ❖ U skladu sa izračunatim prioritetom proces se smešta u odgovarajući red ili prekida trenutno aktivni proces.

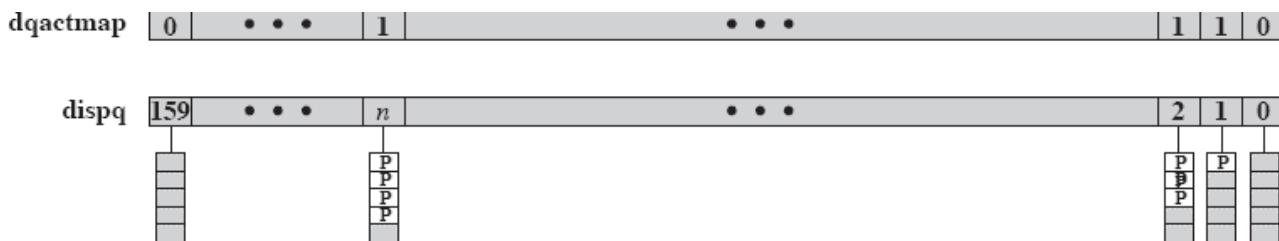


Raspoređivanje u Unix (3)

- ❖ Uloga *base* prioriteta je podela svih procesa u fiksne kategorije određenog nivoa prioriteta
- ❖ *CPU* i *nice* vrednosti su ograničene tako da proces ne može da menja svoju kategoriju
- ❖ Ove kategorije obezbeđuju optimizovani pristup sekundarnim memorijama (diskovima) i brz odgovor na sistemske pozive.
- ❖ U redosledu opadajućem po prioritetu, kategorije su:
 - Swapper
 - Upravljanje blok U/I uređajima
 - Manipulacija datotekama
 - Upravljanje znakovnim U/I uređajima
 - Korisnički procesi

Raspoređivanje u UNIX System V R4

- Najviši prioritet je dat *real-time* procesima, zatim *kernel-mod* procesima, a najniži korisničkim procesima (*time-shared* procesima)
- 160 nivoa prioriteta podeljenih u 3 kategorije
- Red za raspoređivanje (*dispq*) se pridružuje svakom nivou prioriteta, zajedno sa bitmap vektorom - *dqactmap*
- Unutar time-shared procesa prioritet se menja; ukoliko proces iskoristi ceo kvant, prioritet se smanjuje.
- Kvant time-shared proc. 10ms (0) -100ms (59)



Priority Class	Global Value	Scheduling Sequence
Real-time	159 • • • 100	first
Kernel	99 • • 60	
Time-shared	59 • • • 0	last

Jednoprocесорско raspoređivanje

Operativni sistemi



Raspoređivanje u Linux (verzija 2.4 i ranije)

- ❖ Linux ima planiranje niti (*light weight* procesi) - implementirane u jezgru
- ❖ Linux razlikuje tri klase prioriteta
 - **Real-time FIFO (najviši prioritet):** ne mogu se istiskivati, osim od novo- učitane RT FIFO niti (SCHED_FIFO)
 - **Real-time Round-Robin:** iste su kao RT FIFO, jedino što se mogu prekidati (SCHED_RR)
 - **Time sharing** (SCHED_OTHER)
- ❖ Nijedna od ovih klasa nije real-time u pravom smislu; nema specificiranja *deadline-a*, nema garancije za ispunjenje *deadline-a*.
- ❖ Razlog za uvođenje ovog termina je usklađivanje sa standardom P1003.4 (Real-time ekstenzije za UNIX)
- ❖ Podrazumevani prioritet niti je 20, ali se može promeniti sistemskim pozivom *nice(value)* , *value=[-20,19]* pri čemu prioritet dobija vrednost jednaku *20-value*, otuda prioritet niti u opsegu *1≤prioritet≤40*
- ❖ Svaka nit ima dodeljeni vremenski kvant za izvršavanje (vremenski takt na svakih 1 ms – *jiffy*, a kvant je određena multiplikacija takta)
- ❖ Funkcije kernela ***schedule()***, ***do_timer()***, ***switch_to()***



Raspoređivanje u Linux (2)

(verzija 2.4 i ranije)

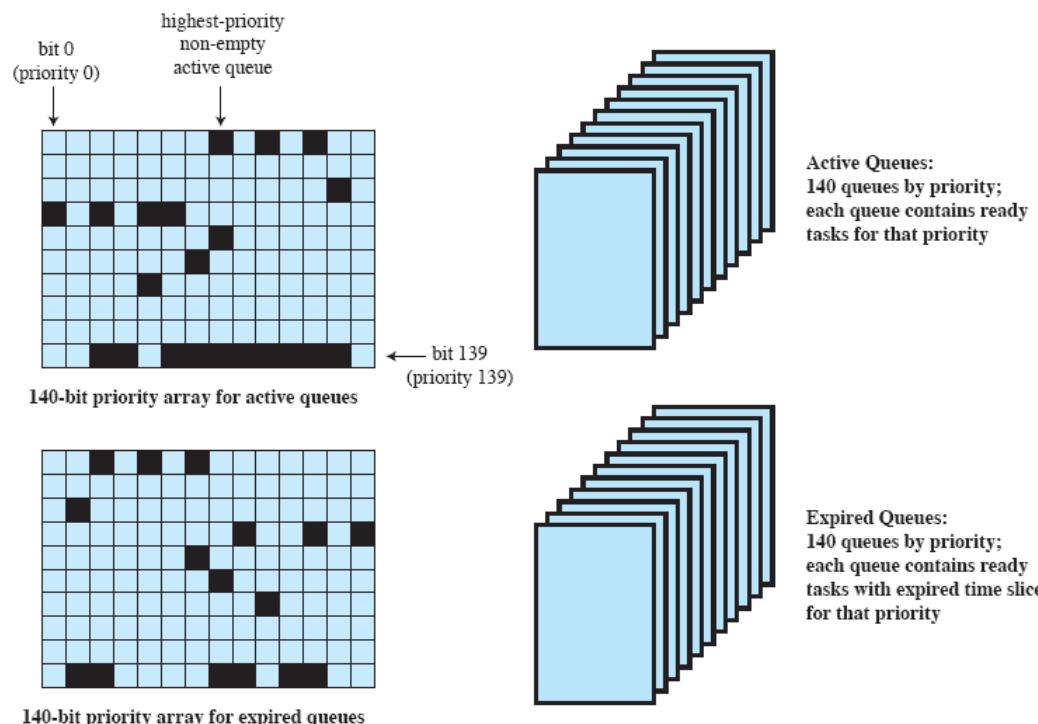
- ❖ Vremenski kvant je promenljiva *counter* u *task_struct* (10-200ms) i u izrazu je predstavljena kao *quantum*
- ❖ Planer najpre izračunava goodness:

```
if ((class == real_time) goodness = 1000 + priority;  
if ((class == time_sharing && quantum > 0) goodness = quantum + priority;  
if ((class == time_sharing && quantum == 0) goodness = 0;
```

- ❖ Planer bira nit koja ima najveći **goodness**. Sa svakim vremenskim taktom vremenski kvant aktivne niti se smanjuje za 1
- ❖ Nit se prekida ako bude ispunjen neki od sledećih uslova:
 - Kvant niti je postao 0
 - Nit se blokira na semaforu, I/O operaciji ili nečem drugom,
 - Prethodno blokirana nit sa većim goodness-om je postala spremna.
- ❖ Ako je kvant svih spremnih niti 0 (blokirane niti mogu imati kvant koji nije 0) planer resetuje kvantove svih niti:
$$\text{quantum} = (\text{quantum} / 2) + \text{priority}$$

Raspoređivanje u Linux (2.6+)

- Linux 2.4 planer za SCHED_OTHER klasu niti nije bio dovoljno skalabilan za povećan broj procesora (jezgara) i niti
- Linux 2.6 koristi novi planer zasnovan na prioritetu, poznat kao O(1) planer
- 140 nivoa prioriteta
 - 0-99 za RT i 100-139 za TS niti
- Za svaki procesor Linux planer održava dve strukture podataka za:
 - Aktivne redove
 - Istekle redove (prethodno iskoristili vremenski kvant)
- Prednost se daje nitima orijentisanim na U/I u odnosu na niti orijentisane na CPU



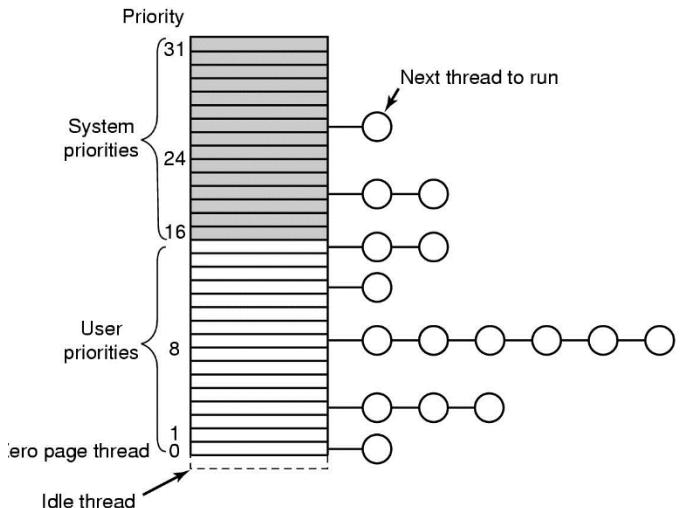


Rasporedjivanje u Linux

- ◆ Za svaki procesor planer bira prvu nit iz nepraznog reda najvišeg prioriteta; niti u redu se organizuju u round-robin stilu
- ◆ Niti se mogu premeštati iz reda jednog procesora u red drugog
- ◆ Prioritet *ne-real time* niti je u opsegu 100-139, inicijalno 120
- ◆ Dinamički prioritet se izračunava na osnovu prethodnog izvršavanja/čekanja niti
- ◆ *Real-time* niti nemaju dinamički prioritet
- ◆ Vremenski kvant je promenljiv – 10-200ms

Raspoređivanje u Windows

- ❖ Nema centralno planiranje niti
- ❖ Planiranje u više nivoa sa povratnom spregom
- ❖ Planer podržava 32 nivoa planiranja
- ❖ Niti u redovima po RR algoritmu
- ❖ Postoje 4 kategorije prioriteta:
 - **16-31 sistemski:** rezervisano za sistem i za niti kojima sistem administrator može dodeljivati prioritet
 - **1-15 korisnički**
 - **0 zero:** za potrebe menadžera memorije
 - **-1 idle:** *idle* nit se izvršava kad niko ne koristi CPU



- ❖ Planer pretražuje redove od 31 pa naniže i bira nit sa početka nepraznog reda i dodeljuje mu 1 kvant
- ❖ Pošto vreme istekne nit se vraća na kraj tog reda, a planer bira nit sa početka reda
- ❖ Ako nijedna nit nije spremna, izvršava se *idle* nit



Raspoređivanje u Windows (2)

- ❖ Planiranje se vrši na nivou niti.
- ❖ Ne postoji centralni planer (scheduler) procesa (niti) već je kod za planiranje razbacan po jezgru sistema.
- ❖ Kada trenutno aktivna nit ne može više da se izvršava ulazi u kernel mode i aktivira planer.
- ❖ Aktiviranje planera se vrši ukoliko je ispunjen neki od sledećih uslova:
 - ❖ Nit se blokira na semaforu, mutex-u, događaju (event)
 - ❖ Nit signalizira neki objekat i aktivira nit većeg prioriteta (npr. operacija na semaforu)
 - ❖ Istečao je vremenski kvant.
- ❖ Planer se poziva automatski u sledećim situacijama:
 - ❖ Završetak I/O operacije – proverava se da li trenutno aktivna nit može biti zamenjena.
 - ❖ Isteček perioda čekanja (timed wait)



Raspoređivanje u Windows (3)

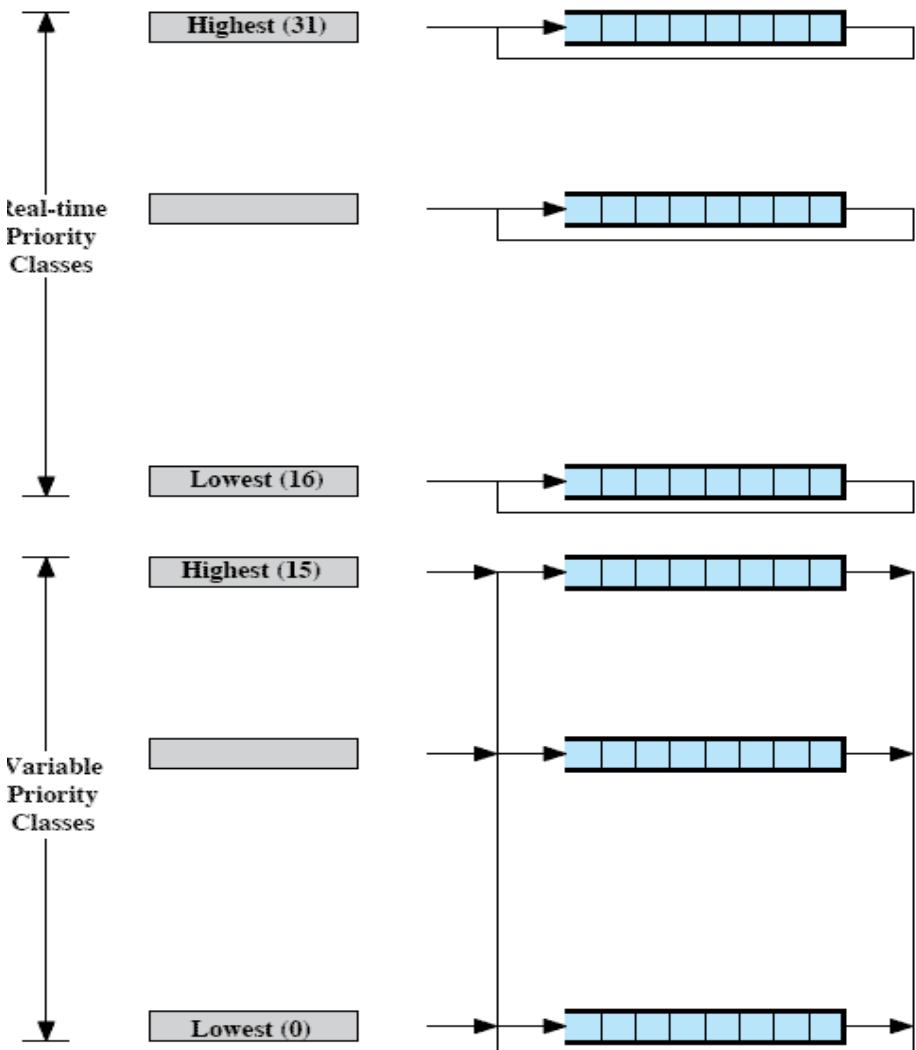
- ❖ Algoritam planiranja je **fully preemptive multi-level queue**.
- ❖ Za svaki red se koristi **Round Robin**.
- ❖ **Fully preemptive** – prekidanje niti se može desiti u bilo kom trenutku a ne samo na kraju vremenskog kvanta.
- ❖ Prioritet procesa i niti (42 moguće kombinacije):
- ❖ SetPriorityClass – postavlja prioritet svih niti procesa
 - ▣ Realtime
 - ▣ High
 - ▣ Above normal
 - ▣ Normal
 - ▣ Below normal
 - ▣ Idle
- ❖ SetThreadPriority – postavlja prioritet niti relativno u odnosu na druge niti istog procesa
 - ▣ Time critical
 - ▣ Highest
 - ▣ Above normal
 - ▣ Normal
 - ▣ Below normal
 - ▣ Lowest
 - ▣ Idle

Jednoprocесорско распоредување

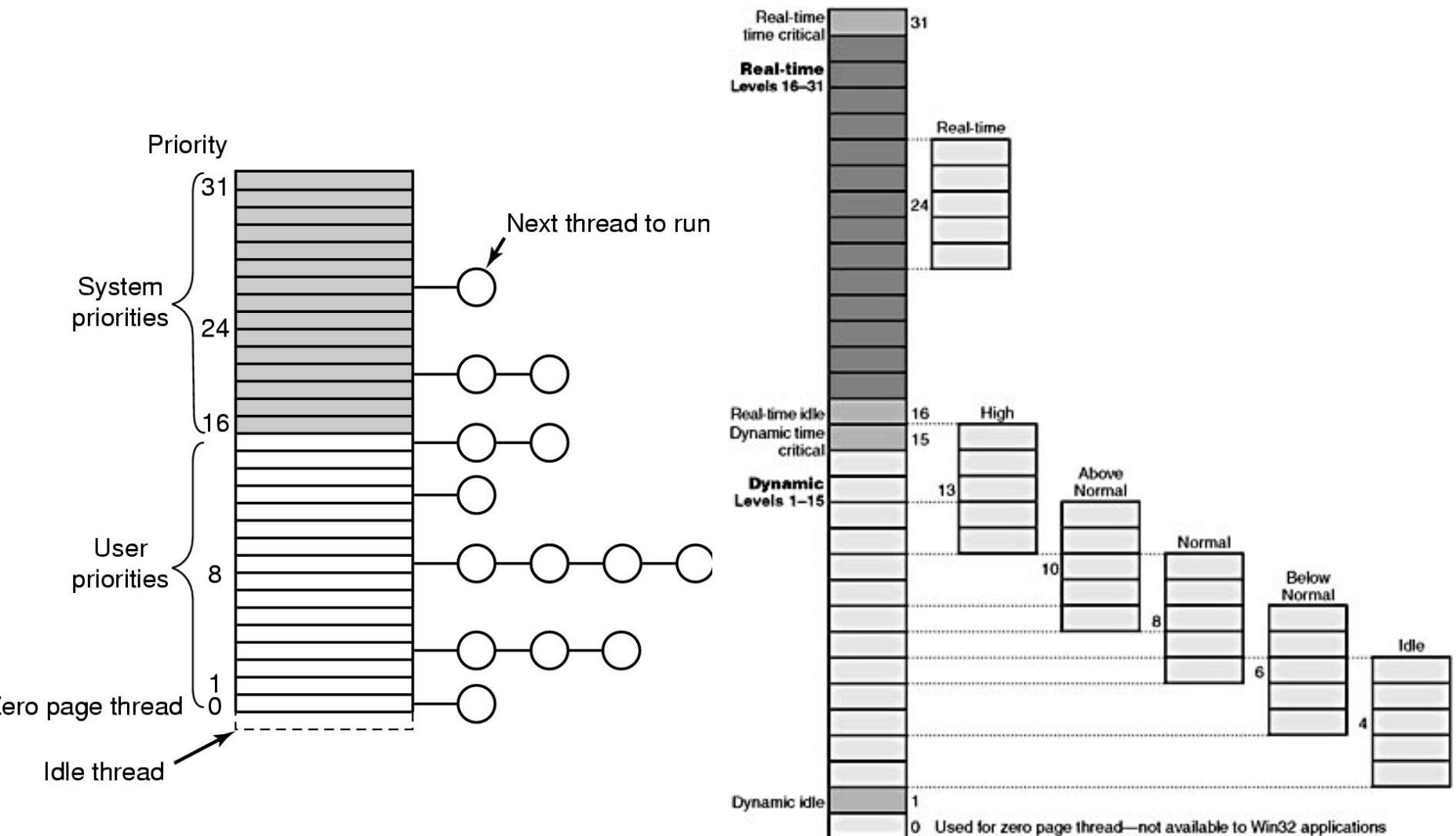
Оперативни системи

Prioriteti procesa i niti

		Win32 process class priorities					
Win32 thread priorities		Realtime	High	Above Normal	Normal	Below Normal	Idle
	Time critical	31	15	15	15	15	15
	Highest	26	15	12	10	8	6
	Above normal	25	14	11	9	7	5
	Normal	24	13	10	8	6	4
	Below normal	23	12	9	7	5	3
	Lowest	22	11	8	6	4	2
	Idle	16	1	1	1	1	1



Raspoređivanje u Windows (4)

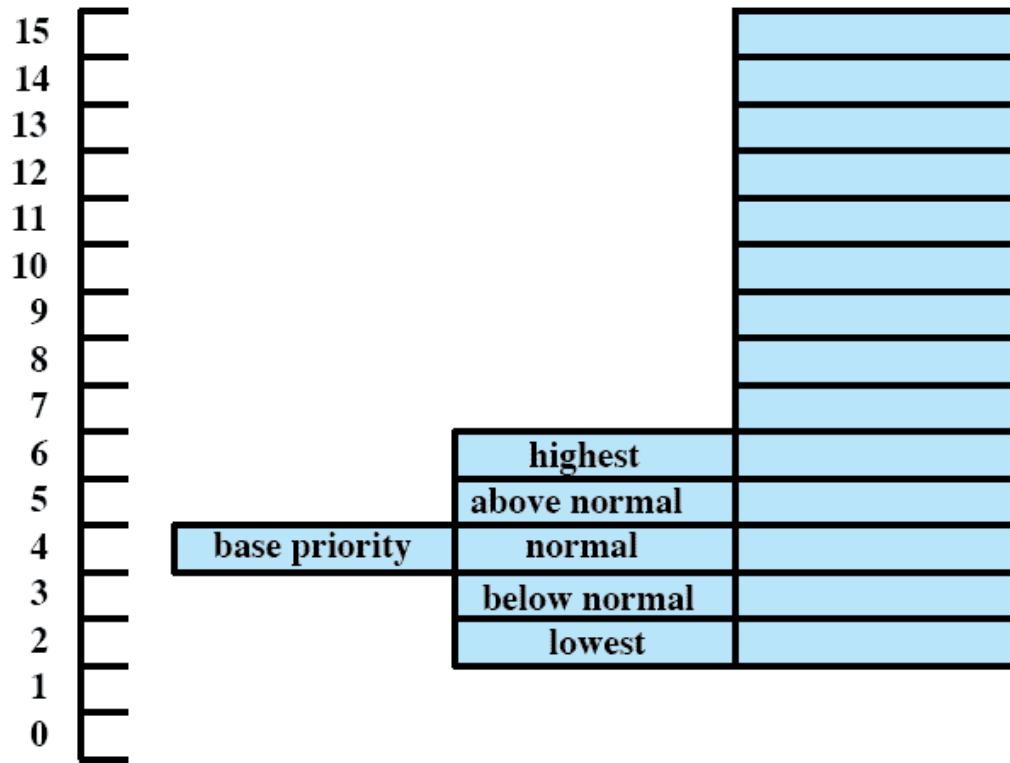


Raspoređivanje u Windows (5)

- ➊ *Zero page thread* je nit koja se izvršava u pozadini koristeći CPU vreme kada nema drugih niti koje su spremne za izvršavanje. Ima zadatak da ažurira stranice za upravljanje memorijom.
- ➋ Vremenom je osnovni algoritam modifikovan tako da nitima sa nivoa nižih od 15 prioritet može biti povećan po završetku I/O operacije. Time se postiže da se veoma brzo startuju I/O operacije čime se postiže da su I/O uređaji stalno aktivni.
- ➌ Ukoliko nit iskoristi čitav vremenski kvant prebacuje se u niz nižeg prioriteta.

Raspoređivanje u Windows (6)

Primer prioriteta procesa i niti



Process Priority Thread's Base Priority Thread's Dynamic Priority

Jednoprocesorsko raspoređivanje

Operativni sistemi



Raspoređivanje u Windows (7)

- ❖ Vremenski kvant:
 - ❖ Windows 2000 Professional – 20 milisekundi
 - ❖ Windows 2000 Server – 120 milisekundi
- ❖ Vremenski kvant se ručno može povećati: 2x, 4x, 6x u odnosu na default vrednost.
- ❖ Ako je nit čekala na nekom objektu za sinhronizaciju, nakon oslobođanja prioritet joj se povećava za 2 ako se radi o *foreground* niti (nit koja kontroliše prozor u fokusu) ili za 1 u ostalim slučajevima.
- ❖ Izmena osnovnog planera: kada prozor postane aktivan sve njegove niti dobijaju veći vremenski kvant.



Domaći zadatak

● Poglavlje 9 **Jednoprocesorsko raspoređivanje**

- 9.6 Ključni pojmovi, kontrolna pitanja i problemi

● CPU Scheduling animations

- <https://apps.uttyler.edu/Rainwater/COSC3355/Animations>
- [First Come, First Served Scheduling](#)
- [Shortest Job First Scheduling](#)
- [Priority Scheduling](#)
- [Round Robin Scheduling](#)



Operativni sistemi

- Upravljanje memorijom -

Prof. dr Dragan Stojanović

Katedra za računarstvo
Univerzitet u Nišu, Elektronski fakultet



Literatura

- *Operating Systems: Internals and Design Principles*, edition, W. Stallings, Pearson Education Inc., 7th – 2012, (5th -2005, 6th - 2008, 8th – 2014 , 9th – 2017)
 - <http://williamstallings.com/OperatingSystems/>
 - <http://williamstallings.com/OperatingSystems/OS9e-Student/>
- **Poglavlje 7: Upravljanje memorijom**



Sistem za upravljanje memorijom

◆ Osnovne funkcije:

- ❖ **Evidencija** slobodnih i zauzetih delova memorije u multiprogramskom sistemu
- ❖ **Dodela** (alociranje) memorije procesima po potrebi
 - Kontinualna dodatak memorije
 - Nekontinualna dodatak memorije
- ❖ **Oslobađanje** (deallociranje) memorije kada procesima više nije potrebna
- ❖ **Upravljanje zamenom (*swapping*)** sadržaja između glavne memorije i diska kada glavna memorija nije dovoljna da smesti sve procese



Zahtevi upravljanja memorijom

◆ Relokacija

- Proces se može smestiti počev od bilo koje memorijske adrese, po potrebi prenesti na disk i ponovo vratiti u memoriju počev od nove memorijske adrese. Memorijske reference moraju da se prevedu u stvarne, fizičke memorijske adrese.

◆ Zaštita memorije

- Zaštita pristupa memorijskom prostoru jednog procesa od strane drugog procesa, kao i zabrana pristupa memorijskom prostoru OS.

◆ Zajedničko korišćenje memorije

- Svaki mehanizam zaštite mora biti fleksibilan da obezbedi zajedničko korišćenje istog dela glavne memorije.

◆ Logička organizacija

- Linearni adresni prostor sastavljen od sekvence bajtova ili reči
- Podjela programa u module koji mogu biti kompajlirani nezavisno

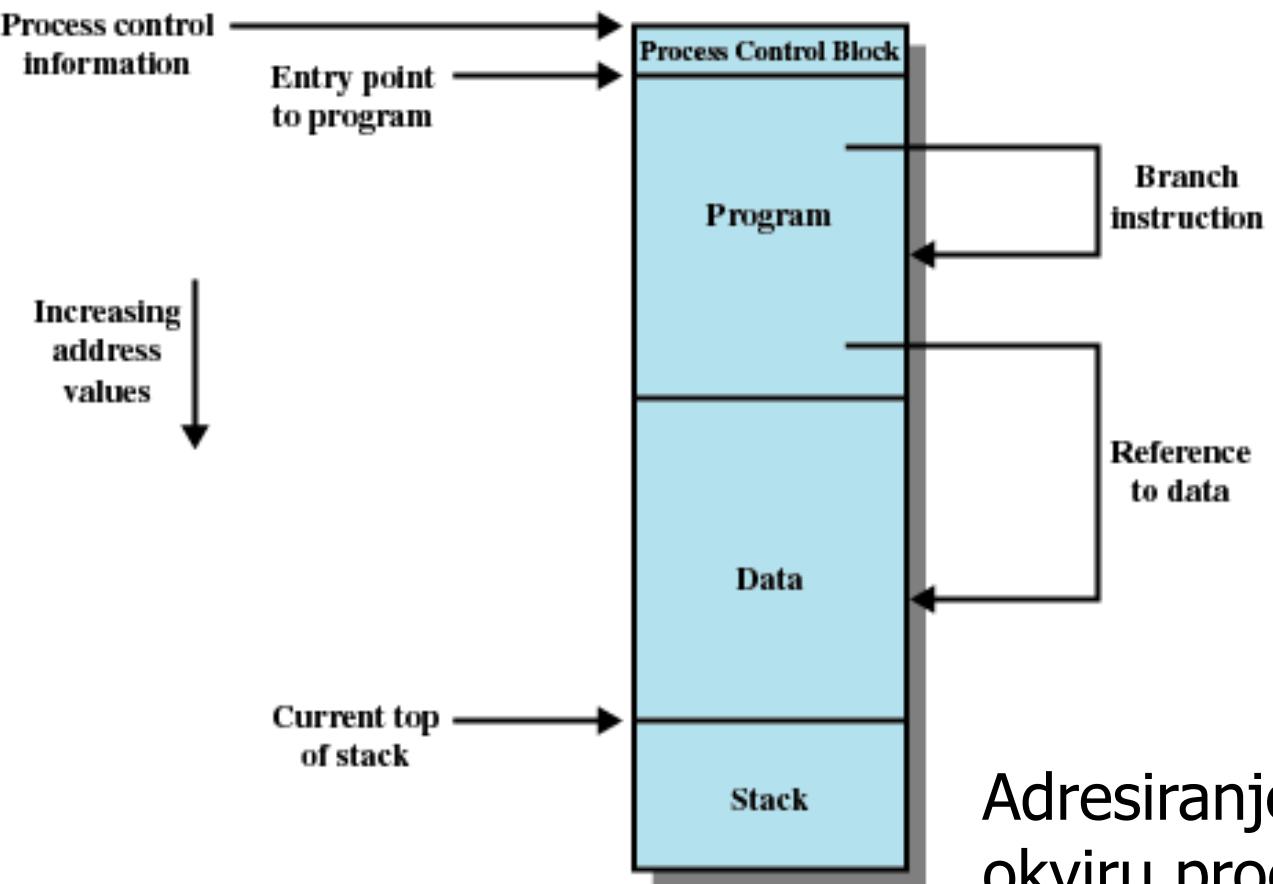
◆ Fizička organizacija

- Dva nivoa: glavna i sekundarna memorija

Upravljanje memorijom

Operativni sistemi

Memorijska slika procesa



- Programski kod
- Podaci
- Stek
- PCB

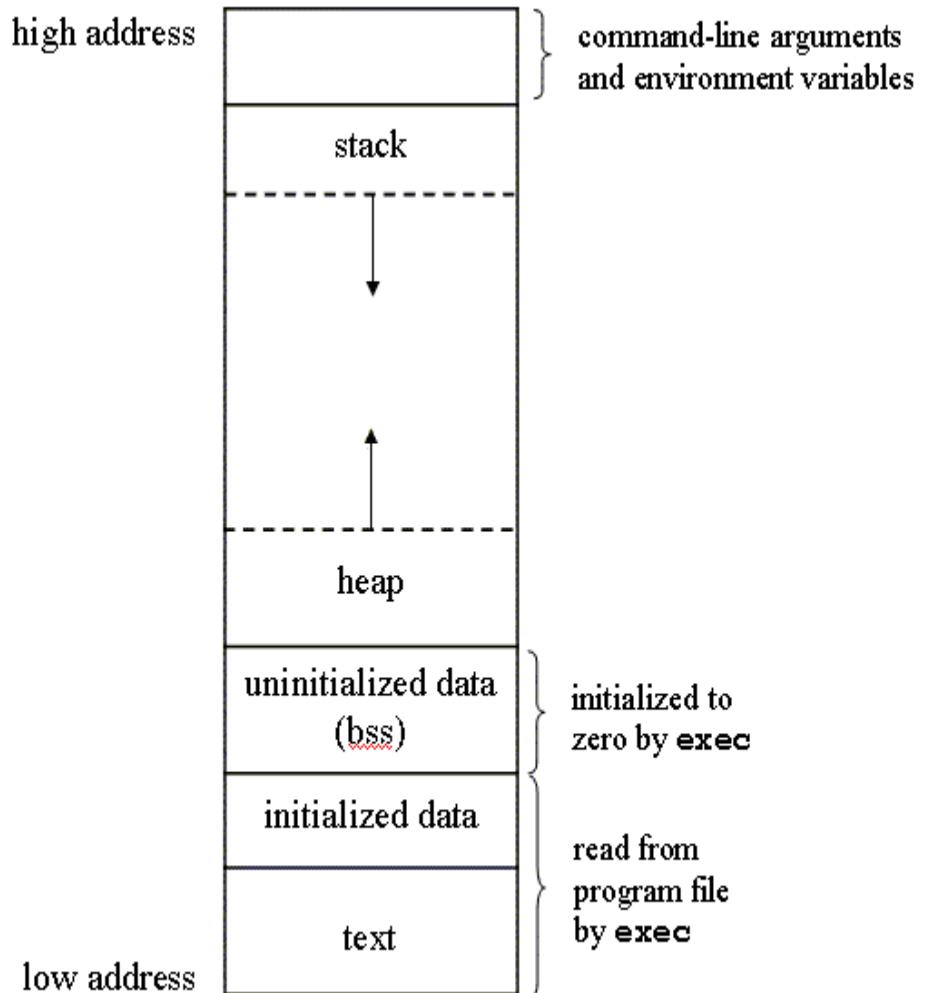
Adresiranje memorije u okviru procesa

Memorijska slika procesa (detaljnije)

- Izvršna datoteka se u određenom formatu procesa (*exe*, *obj*, *elf*,...) nalazi na sekundarnoj memoriji.
- Ova datoteka sadrži **programski kôd**, **podatke** koji su neophodni za izvršenje programa i informacije koje su neophodne OS-u da bi smestio taj program u memoriju i da bi ga izvršio.

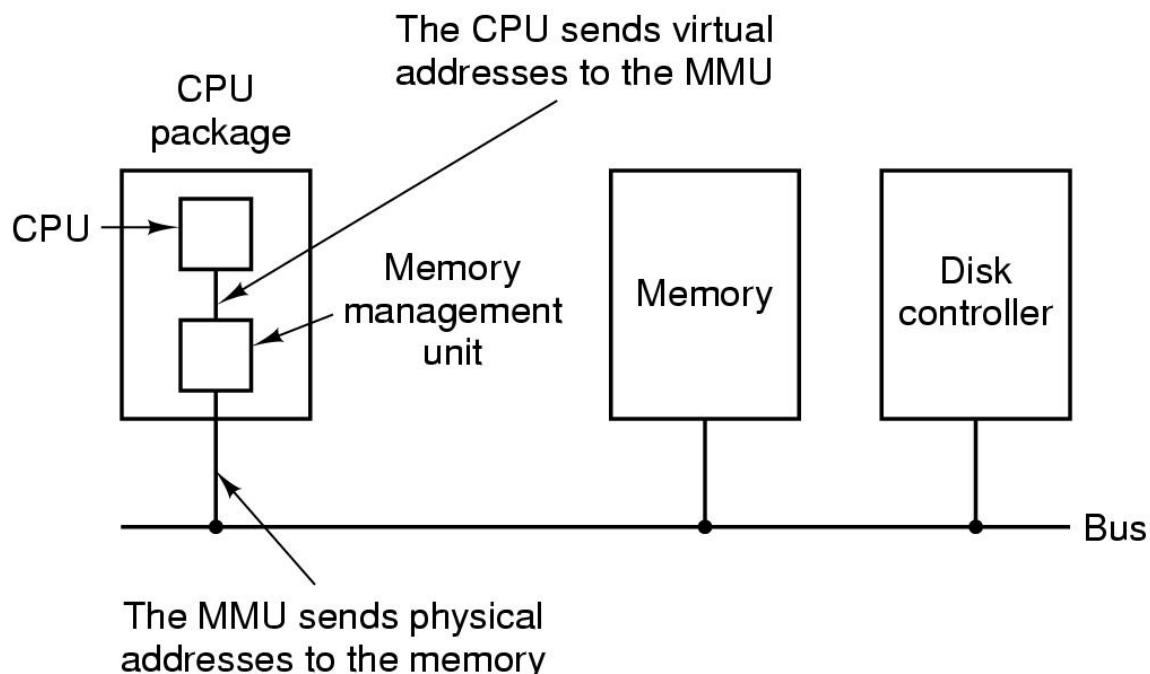
Memorijska slika procesa

- Programski kod (*code*, *text*)
- Podaci (inicijalizovani, neinicijalizovani, dinamički (*heap*))
- Stek



Memory Management Unit (MMU)

- **MMU** (*Memory Management Unit*) – hardverska komponenta, obično na CPU čipu koja vrši transformisanje logičke (virtuelne) adrese u fizičku adresu
- Korisnički procesi rade sa logičkim (virtuelnim) adresama, oni nikad “ne vide” realne, fizičke adrese



Proširenje memorije prostorom na disku

● *Overlay*

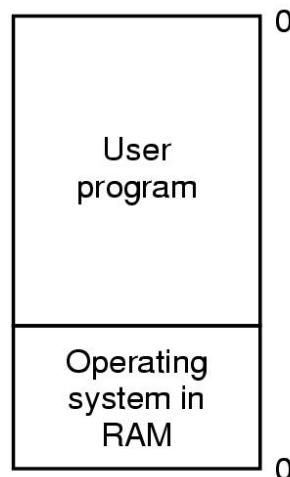
- Neophodan kada je proces veći od veličine memorije koja mu je dodeljena
- U memoriji su smešteni samo instrukcije i podaci procesa koji su neophodni u tekućem izvršavanju procesa
- Nepotrebni delovi procesa (overlay-i) se prebacuju na disk, a potrebni overlay-i se smeštaju u oslobođen deo memorije
- Aplikacioni programer je određivao delove programa koji će predstavljati overlay-e i redosled njihovog smeštanja u memoriju
- Primer: dvo-prolazni prevodilac (assembler)

● Swap-ovanje

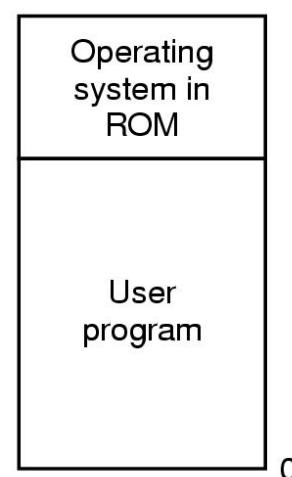
- Ceo proces može biti privremeno prebačen na disk i po potrebi opet smešten u memoriju da bi se nastavilo njegovo izvršavanje
- Mora biti obezbeđen direktni pristup swap području na disku
- Glavni problem je vreme utrošeno sa transfer memorijskih slika procesa
- Modifikovane verzije swapovanja mogu se naći na savremenim OS: UNIX, Linux, Windows, itd.

Monoprogramiranje

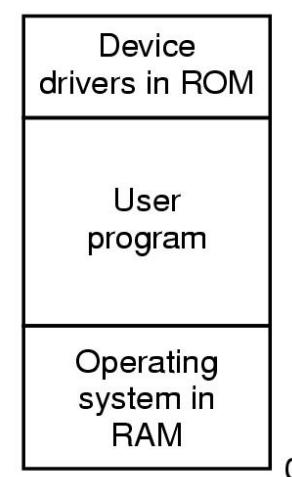
- Najjednostavnija šema za upravljanje memorijom
- Samo jedan proces (program) može da se izvršava i bude u memoriji i on deli memoriju sa operativnim sistemom
 - a) Korišćen na *mainframe* računarima i mini-računarima
 - b) Na nekim *palmtop* računarima i ugrađenim računarskim sistemima
 - c) Na prvim personalnim računarima (MS DOS) gde je deo OS bio smešten u ROM u obliku BIOS (*Basic Input Output System*)



(a)



(b)



(c)

Upravljanje memorijom
Operativni sistemi



Podela (particionisanje) memorije

◆ Raniji načini upravljanja memorijom

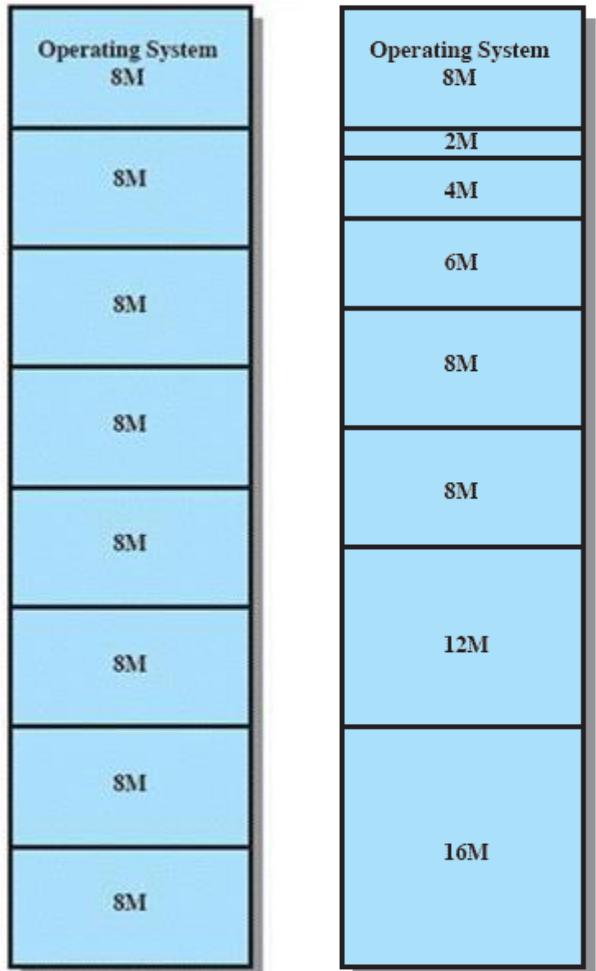
- Pre razvoja koncepta virtuelne memorije
- Danas se uglavnom ne koristi osim u ugrađenim/mobilnim računarima

◆ Dve tehnike

- Fiksna podela (particionisanje) memorije
- Dinamička podela (particionisanje) memorije

Fiksna podela memorije (1)

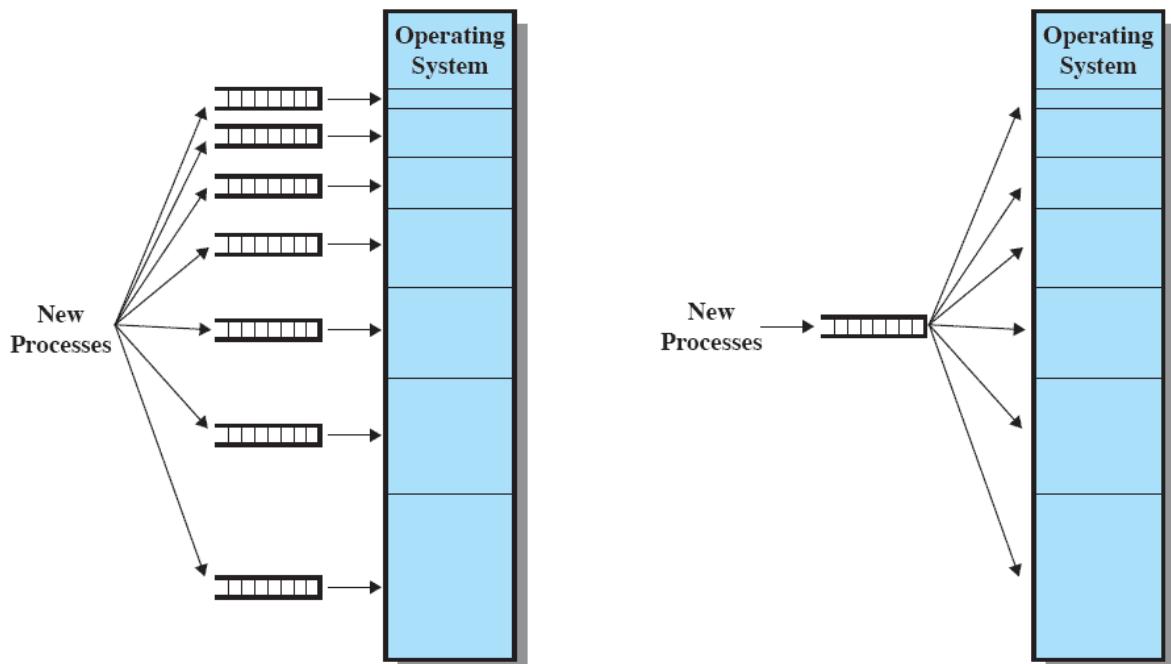
- Memorija je podeljena na **fiksne particije** pri startovanju sistema
- Particije mogu biti iste ili različitih veličina
- Procesima se dodeljuje memorijska particija veća od zahtevane
 - Proces veći od najveće dostupne particije mora da se deli u module i primenjuje tehnika **overlay**
 - Interna fragmentacija** - Preostali deo particije se ne evidentira kao slobodan i ne može se dodeliti nijednom procesu



Fiksna podela memorije (2)

Algoritam raspoređivanja

- Proces se smešta u red čekanja na odgovarajuću particiju u skladu sa zahtevom – nepovoljno ako veća particija bude oslobođena (a)
- Proces se smešta u jedinstveni red čekanja – čim se oslobodi particija biva dodeljena prvom procesu od početka reda kome odgovara (b)



Upravljanje memorijom
Operativni sistemi

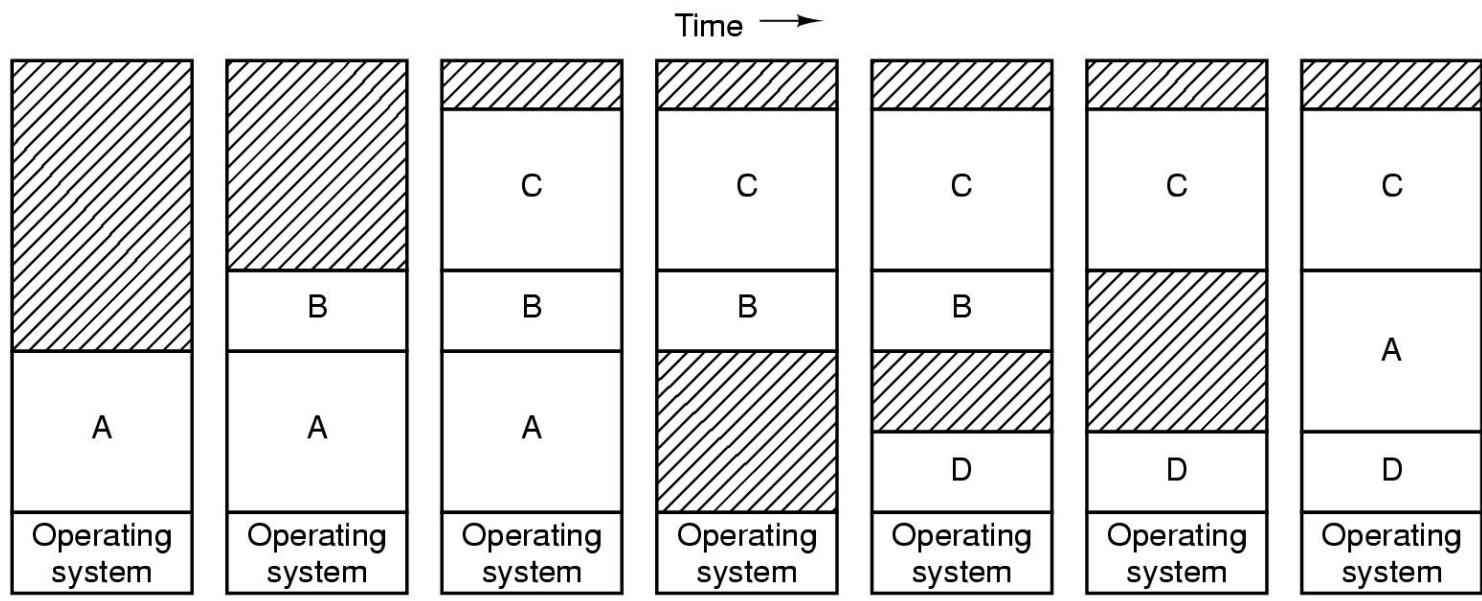


Problemi fiksne podele memorije

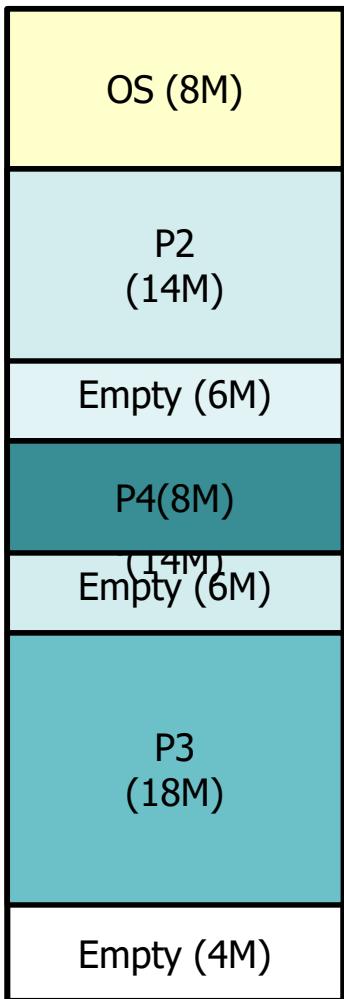
- Broj aktivnih procesa je ograničen sistemom
 - Ograničen pre-determinisanim brojem fiksnih memorijskih particija
- Veliki broj veoma malih procesa neće efikasno koristiti memoriju
 - U oba slučaja, sa jednakim i nejednakim particijama

Dinamičko deljenje na particije (1)

- Broj memorijskih particija, njihova veličina i stanje (slobodna, zauzeta) se menjaju sa aktiviranjem i završavanjem procesa – dinamički (IBM OS/MVT)
- Povećava iskorišćenost memorije, ali usložnjava dodelu i oslobođanje memorije, kao i evidenciju slobodnog i zauzetog memorijskog prostora



Dinamičko deljenje na particije (2)

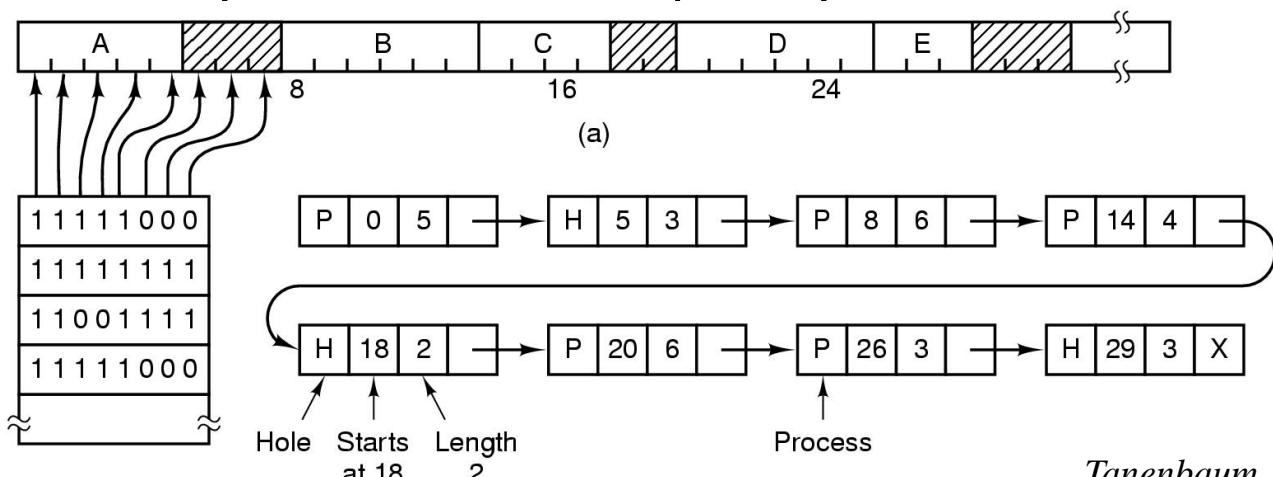


- **Eksterna fragmentacija** – Ukupna veličina slobodnih particija je dovoljna za smeštanje određenog procesa, ali je nekontinualna u memorijskom prostoru.
- Rešenje je **kompakcija** (sažimanje) – prebacivanje memorijskih delova procesa prema jednom kraju adresnog prostora
 - Zahteva mnogo procesorskog vremena

Dinamičko deljenje na particije (3)

◆ Evidencija slobodnih i zauzetih memorijskih particija

- **Bitmapa** – memorija je podeljena na alokacione jedinice od nekoliko reči do nekoliko KB: 0 – slobodna, 1 – zauzeta
 - Za proces koji zahteva k memorijskih jedinica, neophodno je pretražiti bitmapu za k suksesivnih 0
- **Lančana lista** – elementi lančane liste sadrže oznaku da li se radi o procesu ili slobodnoj particiji, početnu adresu, veličinu i link
 - Moguće su odvojene liste za proceze i slobodne particije ili ulančavanje samih slobodnih particija



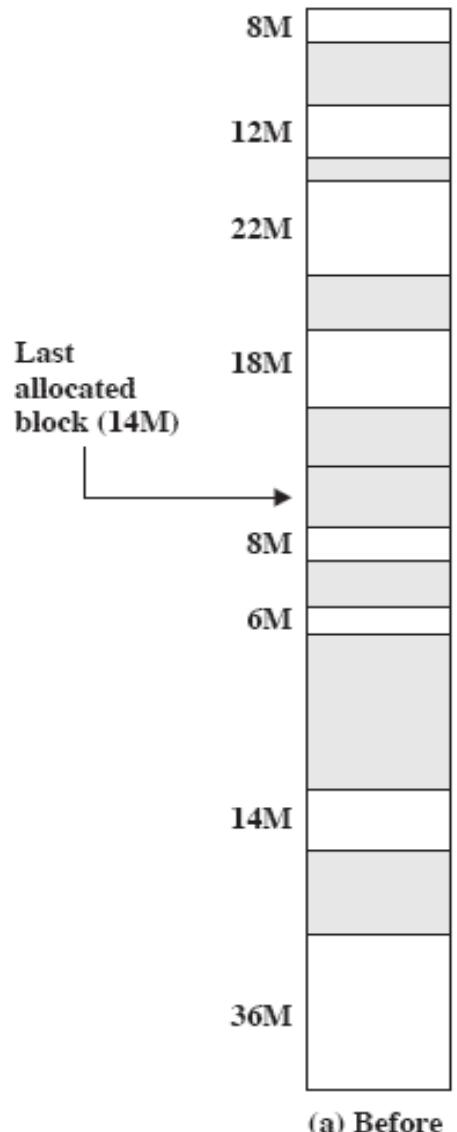
Tanenbaum, 2008

Algoritmi dodele dinamičkih particija

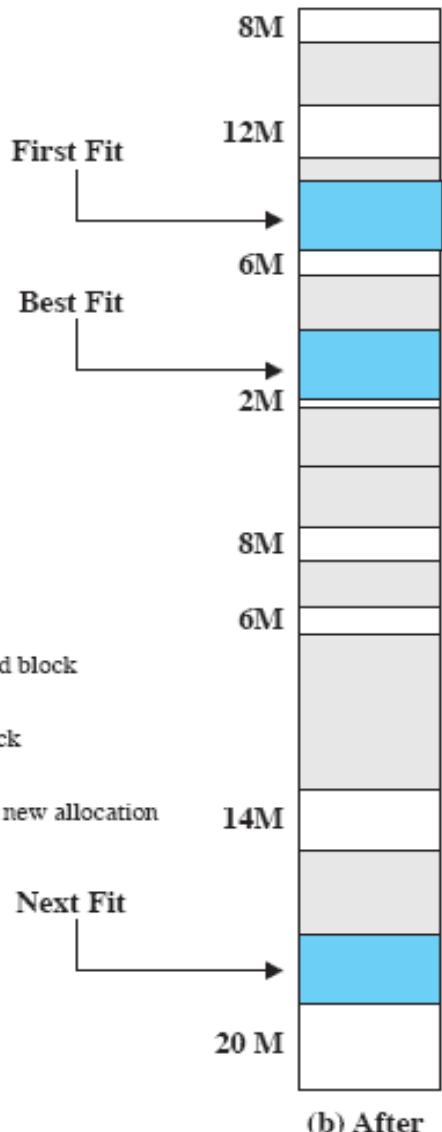
- ❖ **First fit** (Prvo poklapanje, prvi odgovarajući) – Pretražuje sve slobodne particije dok ne nađe dovoljno veliku, koja se deli na dva dela, jedan zauzima proces, a drugi deo ostaje kao slobodna particija
 - ❖ Jednostavno ukrupnjavanje susednih particija
- ❖ **Next fit** (Sledeće poklapanje, sledeći odgovarajući) – Kao i prethodni, ali pretrazivanje započinje od adrese gde je nadjena prethodna slobodna particija
- ❖ **Best fit** (Najbolje poklapanje, najbolje odgovarajući) – Pretražuje sve particije dok ne nađe najmanju odgovarajuću particiju
 - ❖ Sporiji algoritam, jer zahteva pretrazivanje cele liste osim ako je sortirana po veličini particija)
 - ❖ Generiše puno malih slobodnih particija (eksterna fragmentacija)
- ❖ **Worst fit** (Najgore poklapanje, Najgore odgovarajući) – Uzima najveću odgovarajuću particiju sa ciljem da preostali deo bude dovoljno velik da bi bio upotrebljiv
- ❖ **Quick fit** (Brzo poklapanje) – Organizuje posebne liste za particije određenih veličina (4KB, 8KB, 12KB, itd.) i eventualno binarno stablo u kome su čvorovi glave lančanih listi elemenata kojima se definišu particije odgovarajuće veličine

Alokacija dinamičkih particija

- Primer sadržaja memorije pre i nakon alokacije bloka od 16MB



(a) Before



(b) After



Sistem partnera (*Buddy system*)

- ➊ Celokupan raspoloživi memorijski prostor se tretira kao jedinstveni slobodni blok od 2^N
- ➋ Ako se pojavi zahtev za memorijom veličine s takve da je $2^{N-1} < s \leq 2^N$, celokupan blok se dodeljuje
 - ▣ U suprotnom blok se deli na dva jednakana bloka duplo manje veličine (partnera, *buddies*)
 - ▣ Ovaj postupak se nastavlja sve dok se ne dobije najmanji blok veći ili jednak zahtevu s

Sistem partnera - primer

1 Mbyte block



Request 100 K



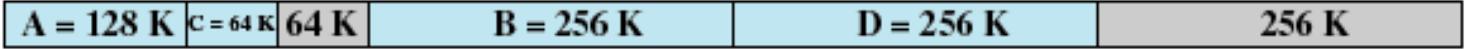
Request 240 K



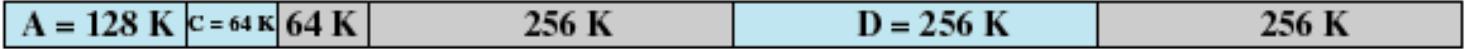
Request 64 K



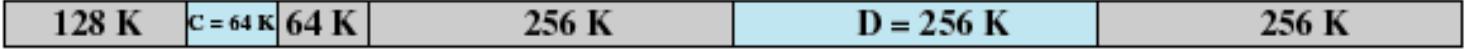
Request 256 K



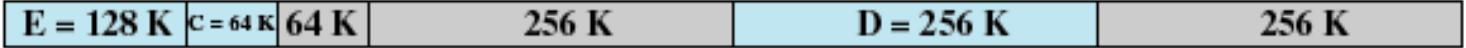
Release B



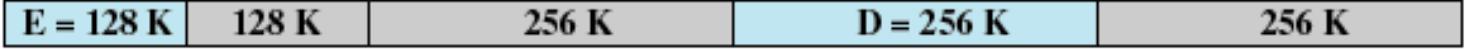
Release A



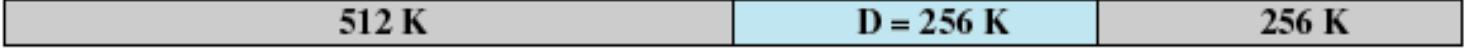
Request 75 K



Release C



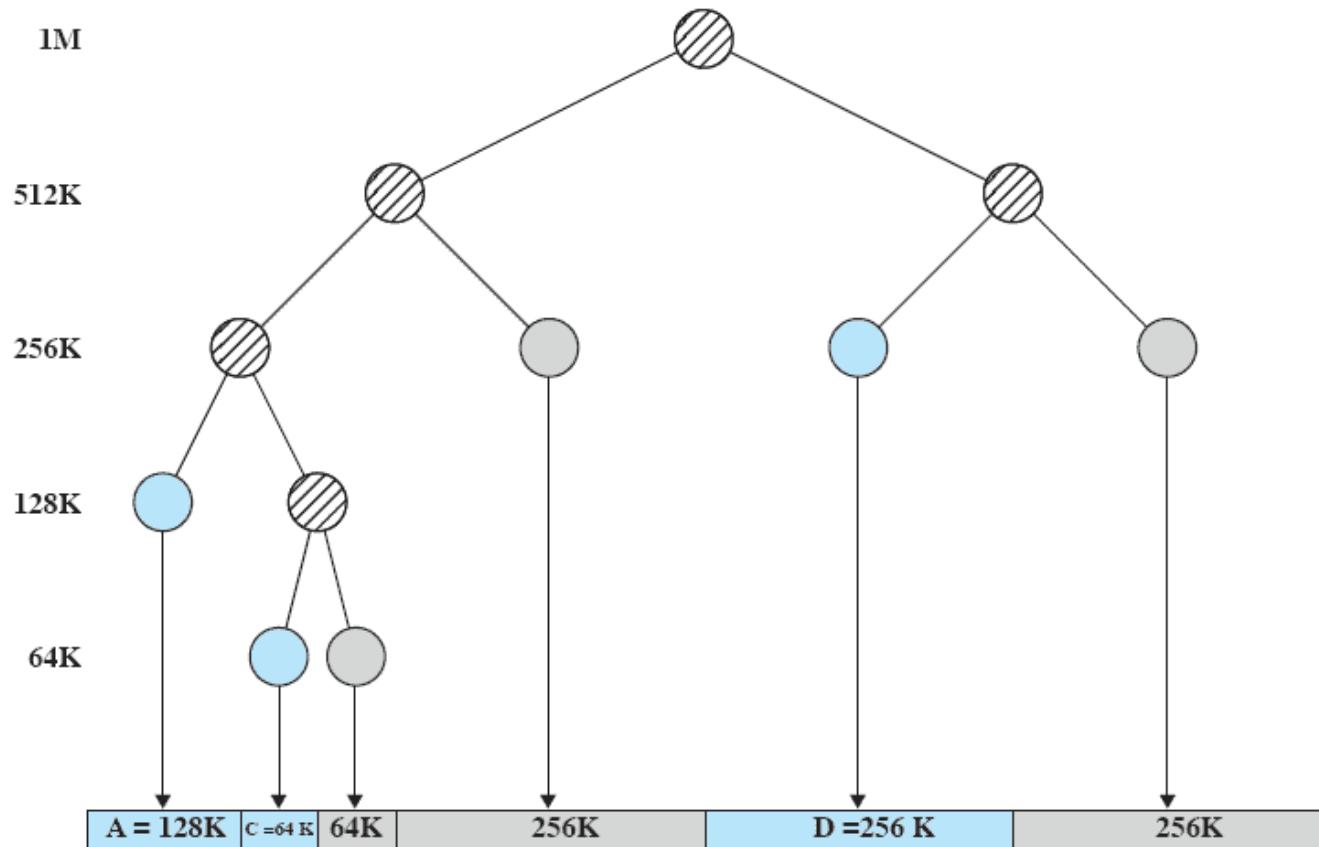
Release E



Release D



Reprezentacija sistema partnera – struktura stabla



Upravljanje memorijom
Operativni sistemi

Relokacija

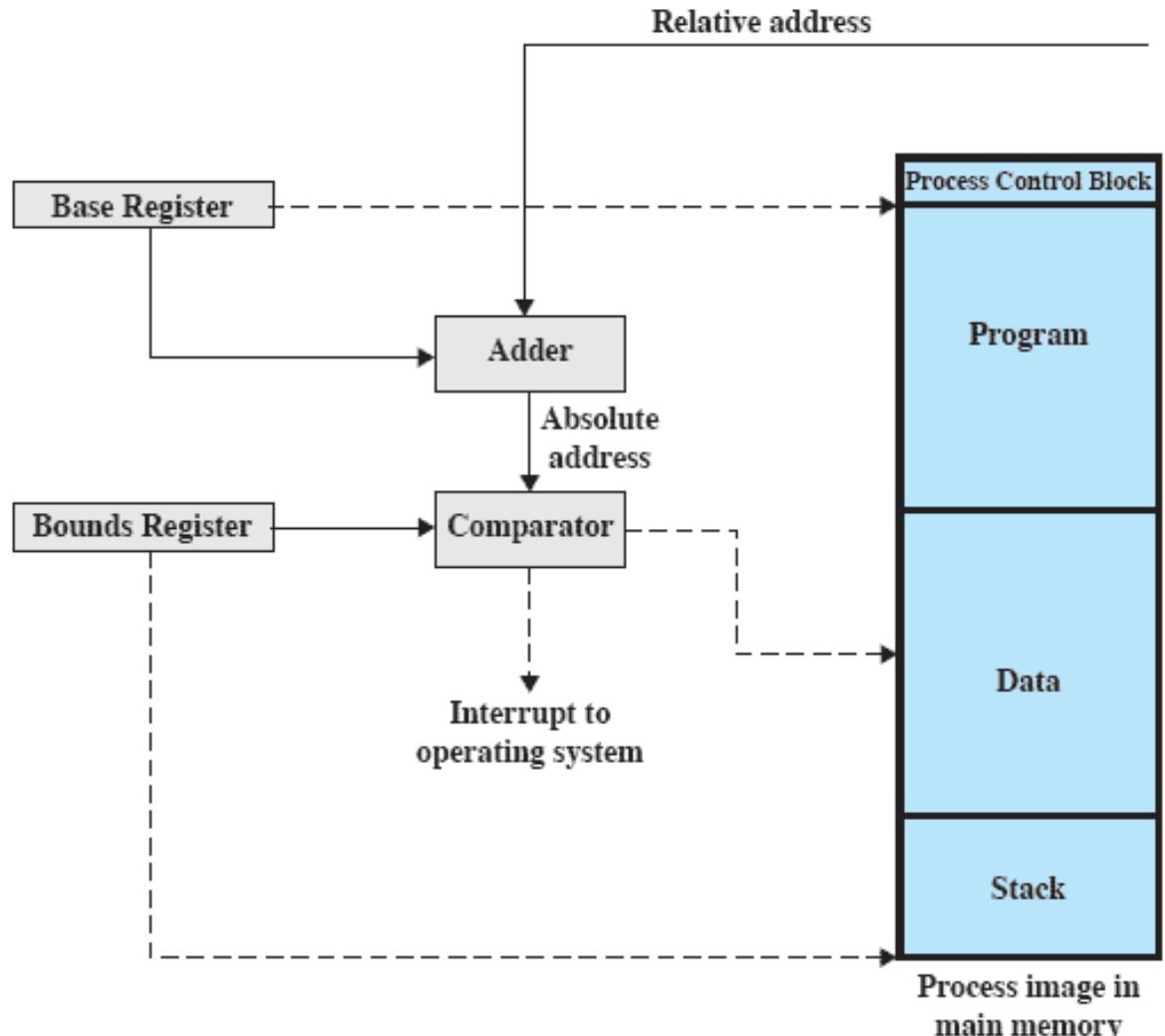
- ➊ Relokacija – smeštanje i pomeranje procesa u memoriji
- ➋ Kada je program (*main* funkcija, korisničke funkcije, bibliotečke funkcije, itd.) linkovan u link editoru, formira se izvršni kôd (modul) na disku sa logičkim adresama.

Relokacija se može obaviti:

- ▢ **U vreme punjenja**, modifikovanjem svih adresa u skladu sa memorijskom adresom punjenja (IBM OS/360)
 - ▢ **U vreme izvršenja**, korišćenjem **base** (relocation) i **bounds** registara
- ➌ Memorijske adrese
 - ▢ **Logička adresa** – referenca na memorijsku lokaciju nezavisna od stvarnog smeštanja procesa u memoriju
 - ▢ **Relativna adresa** – poseban primer logičke adrese koja se izražava relativno u odnosu na neku poznatu adresu, obično vrednost u CPU registru
 - ▢ **Fizička adresa** – adresa lokacije u glavnoj memoriji

Relokacija

- Hardverska podrška za relokaciju





Nekontinualna dodela memorije

- ➊ Nekontinualna dodela memorije – logički adresni prostor procesa ne mora biti smešten kontinualno u glavnoj memoriji
- ➋ Metode nekontinualne dodele:
 - Straničenje (*Paging*)
 - Segmentacija (*Segmentation*)
 - Segmentacija sa straničenjem (*Segmentation with paging*)



Straničenje

- ❖ Adresni prostor procesa je podeljen na delove određene veličine - **Stranice** (*Page*) – Logičke stranice
- ❖ Fizička memorija je (logički) podeljena na delove iste veličine – **Stranični okviri** (*Page Frame*) – Fizičke stranice
- ❖ Veličina stranica i straničnih okvira je od 512B – 64KB (tipično 4KB)
- ❖ Veličina stranice savremenih računara može biti promenljiva:
 - ❖ MIPS (4KB-16MB), UltraSparc (8KB – 4MB), Pentium (4KB – 4MB), PowerPC (4KB), DEC Alpha (8KB), Itanium (4KB – 256MB)
- ❖ Transformisanje iz logičke (virtuelne) u fizičku adresu obavlja se korišćenjem **Tabele stranica** (*Page Table*)

Stranice i okviri

- Da bi se startovao proces veličine **n** stranica neophodno je postojanje **n** slobodnih straničnih okvira u memoriji – nema eksterne fragmentacije
- Interna fragmentacija – u okviru poslednje stranice postoji interni fragment, jer proces (skoro) nikad ne zahteva ceo broj stranica
 - Zahtev 32128 B -> 8 stranica od 4KB = 32768
 - Interni fragment 640 B u poslednjoj stranici

Frame number	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	D.0
5	D.1
6	D.2
7	C.0
8	C.1
9	C.2
10	C.3
11	D.3
12	D.4
13	
14	

Upra

Operativni sistemi

Straničenje – tabele stranica

- ◆ Tabele stranica za procese sa prethodne slike u poslednjem vremenskom trenutku

0	0
1	1
2	2
3	3

Process A
page table

0	—
1	—
2	—

Process B
page table

0	7
1	8
2	9
3	10

Process C
page table

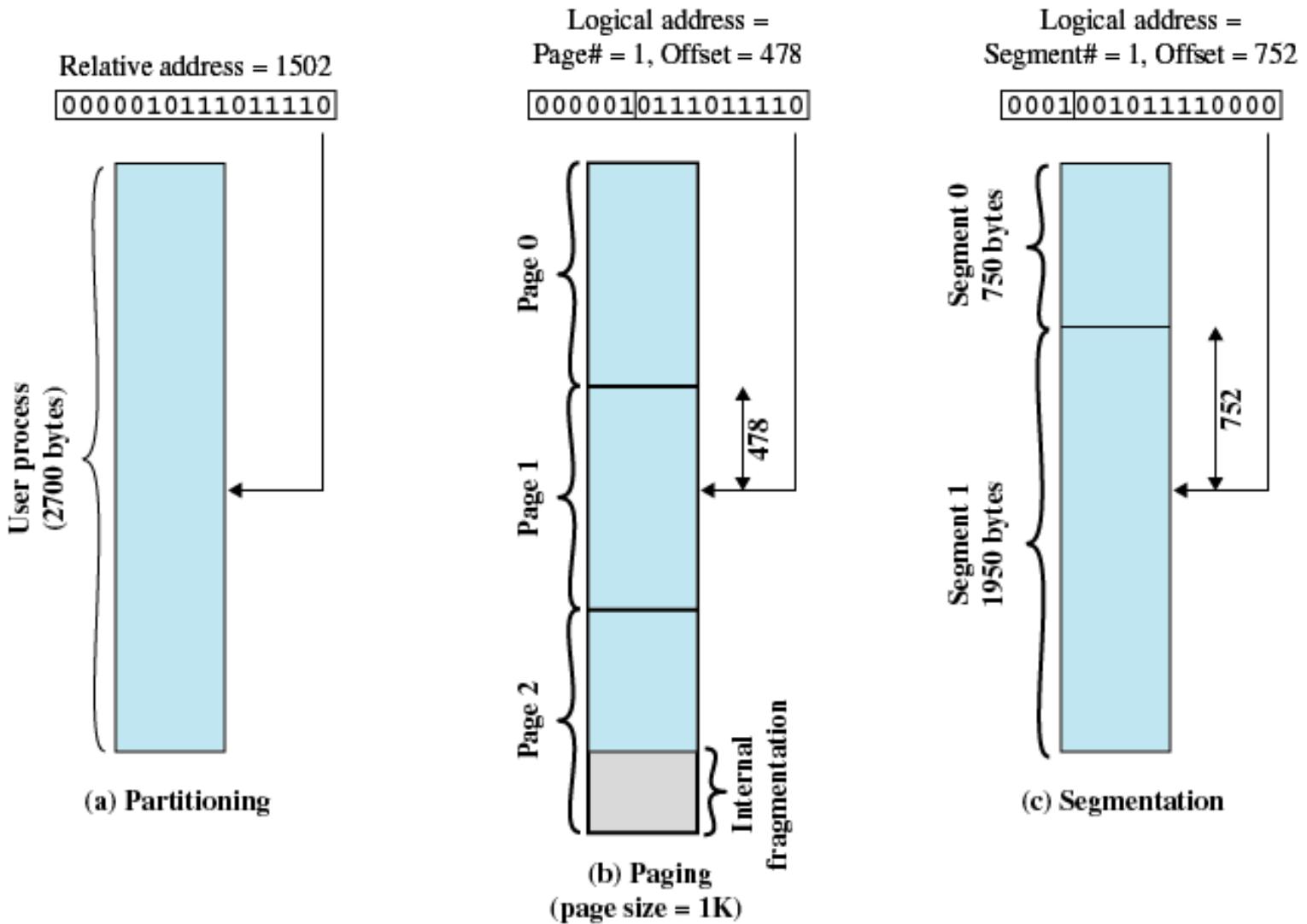
0	4
1	5
2	6
3	11
4	12

Process D
page table

13
14

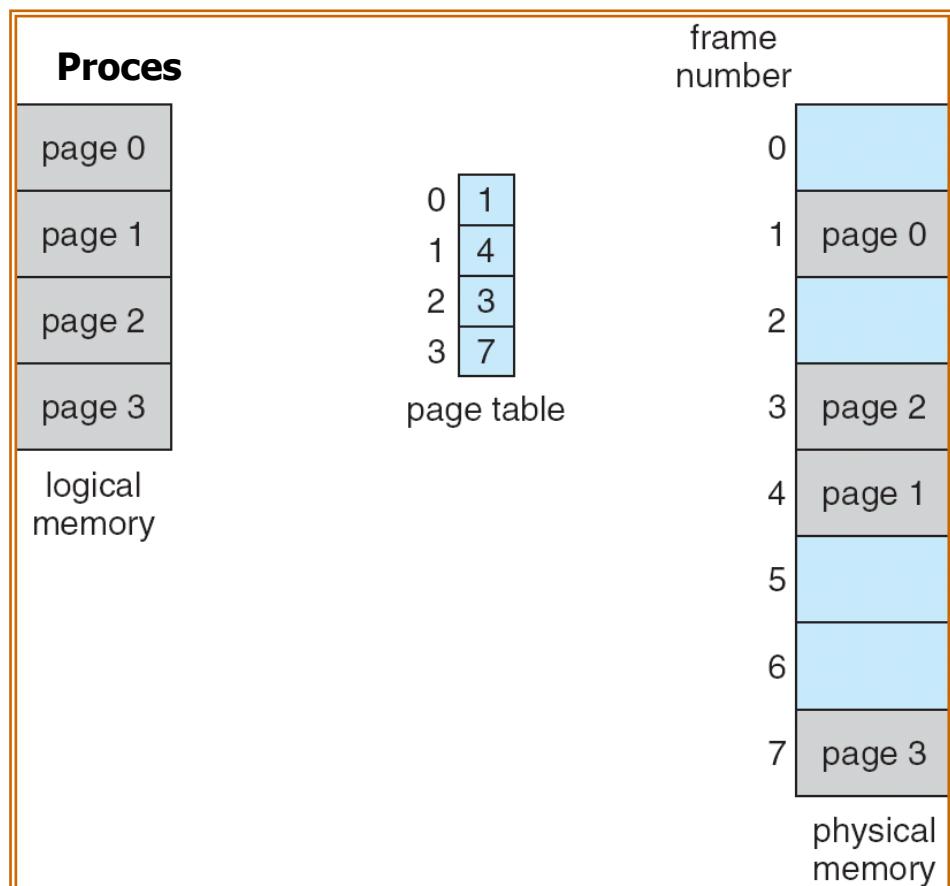
Free frame
list

Logičke adrese (primer za adresu 1502)



Straničenje – primer

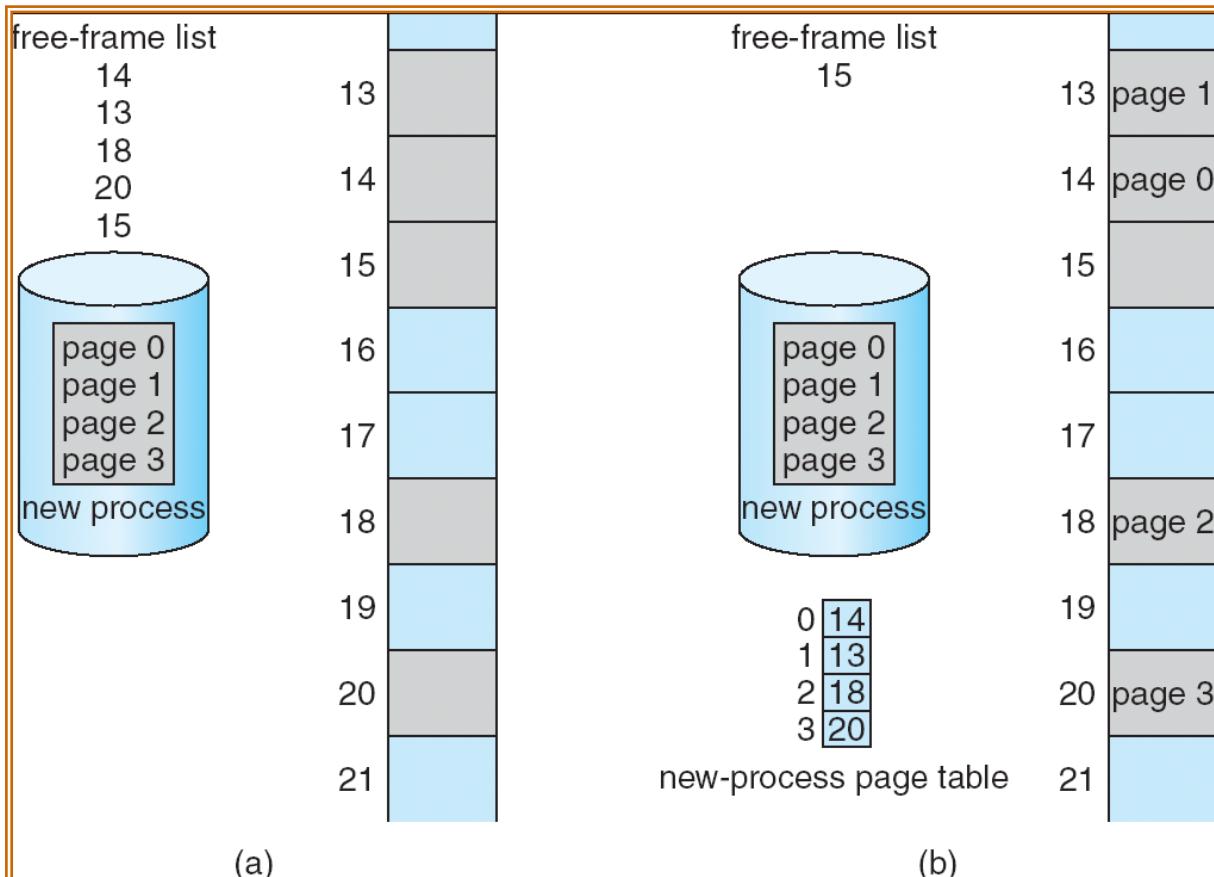
- Adresni prostor procesa sadrži 4 stranice
- Fizička memorija sadrži 8 straničnih okvira



Silberschatz, 2011

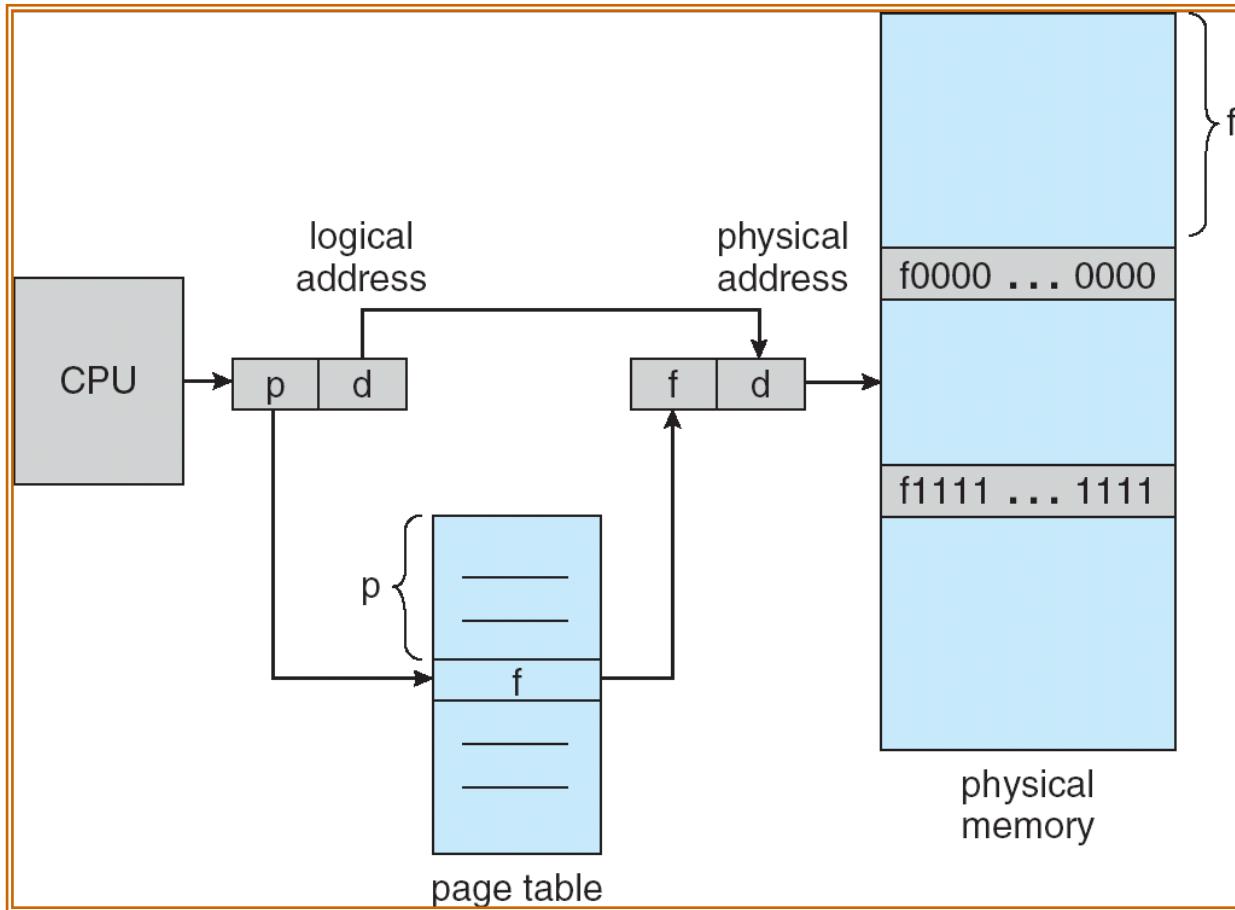
Dodela memorije straničenjem

- Primer



Silberschatz, 2011

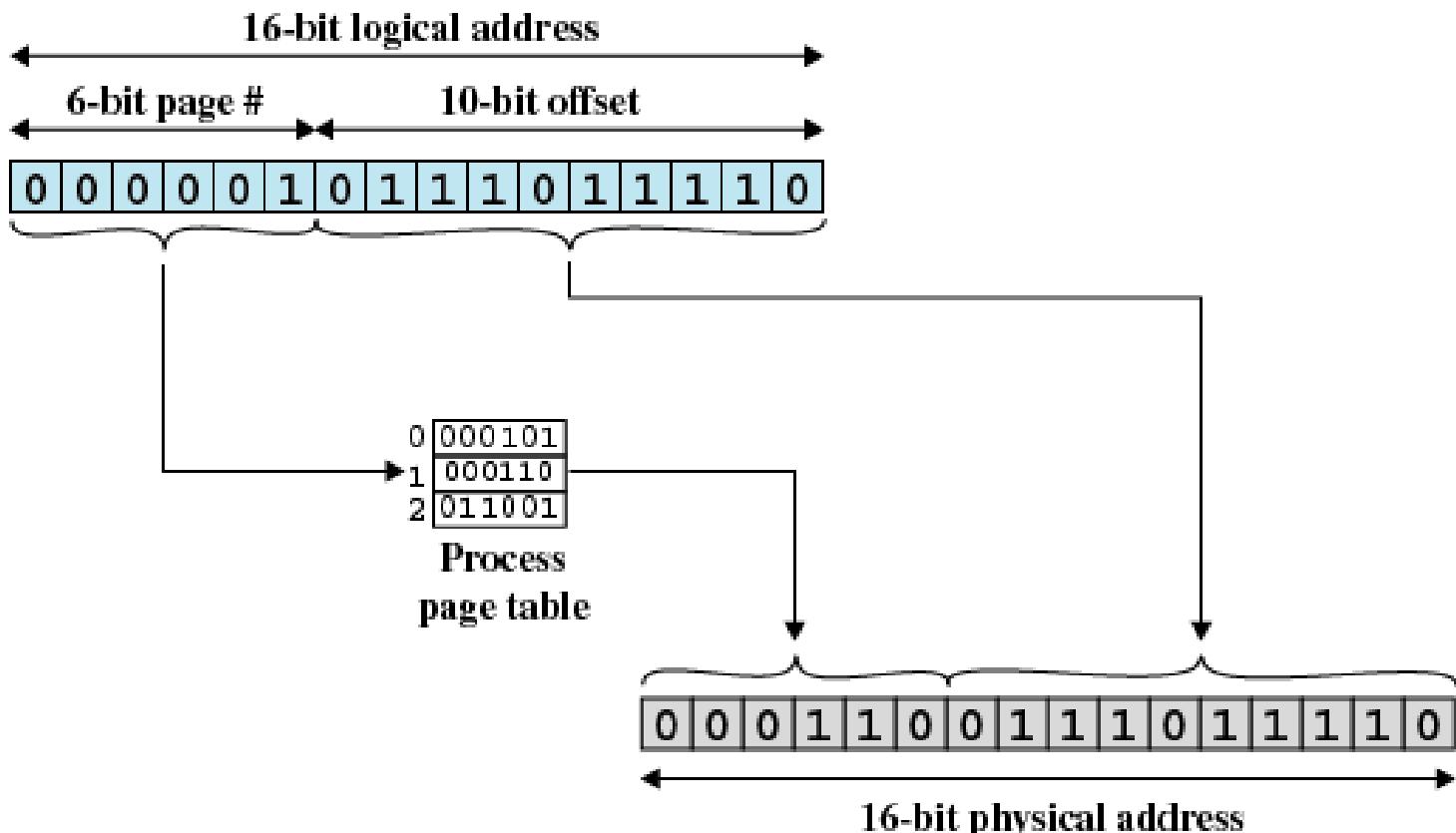
Prevodenje logičke u fizičku adresu – tabela stranica



Silberschatz, 2011

Prevodenje logičke u fizičku adresu – primer

- Stranice su veličine 1KB



Segmentacija

- ❖ Šema za upravljanje memorijom koja odgovara korisnikovom pogledu na memoriju u kome je program kolekcija segmenata
- ❖ Segment je logička jedinica poput: glavnog programa (*main*), procedura, funkcija, metoda, objekata, lokalnih i globalnih promenljivih, *common* blokova, magacina, tablica simbola, polja, itd.
- ❖ Svaki segment predstavlja poseban adresni prostor i sadrži linearnu sekvencu adresa, od 0 do određenog maksimuma.
- ❖ Prednosti segmentacije:
 - Segmenti mogu da rastu i smanjuju se nezavisno jedan od drugog, i da budu smešteni u nekontinualne memorijske particije
 - Svaki segment ima poseban tip zaštite u skladu sa tipom segmenta
 - Segmentacija omogućava deljenje segmenata sa procedurama i podacima između različitih procesa (deljene biblioteke – *shared library*)
- ❖ Nedostatak:
 - Eksterna fragmentacija

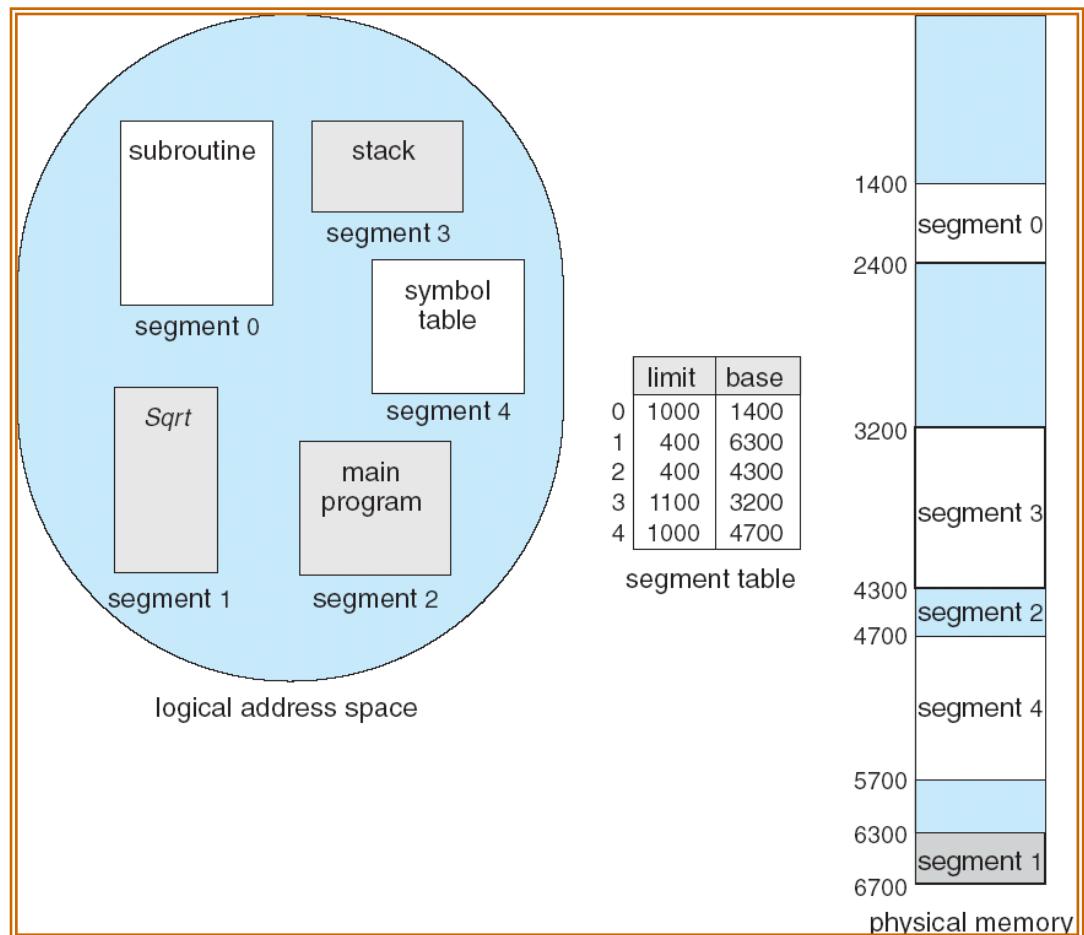


Implementacija segmentacije

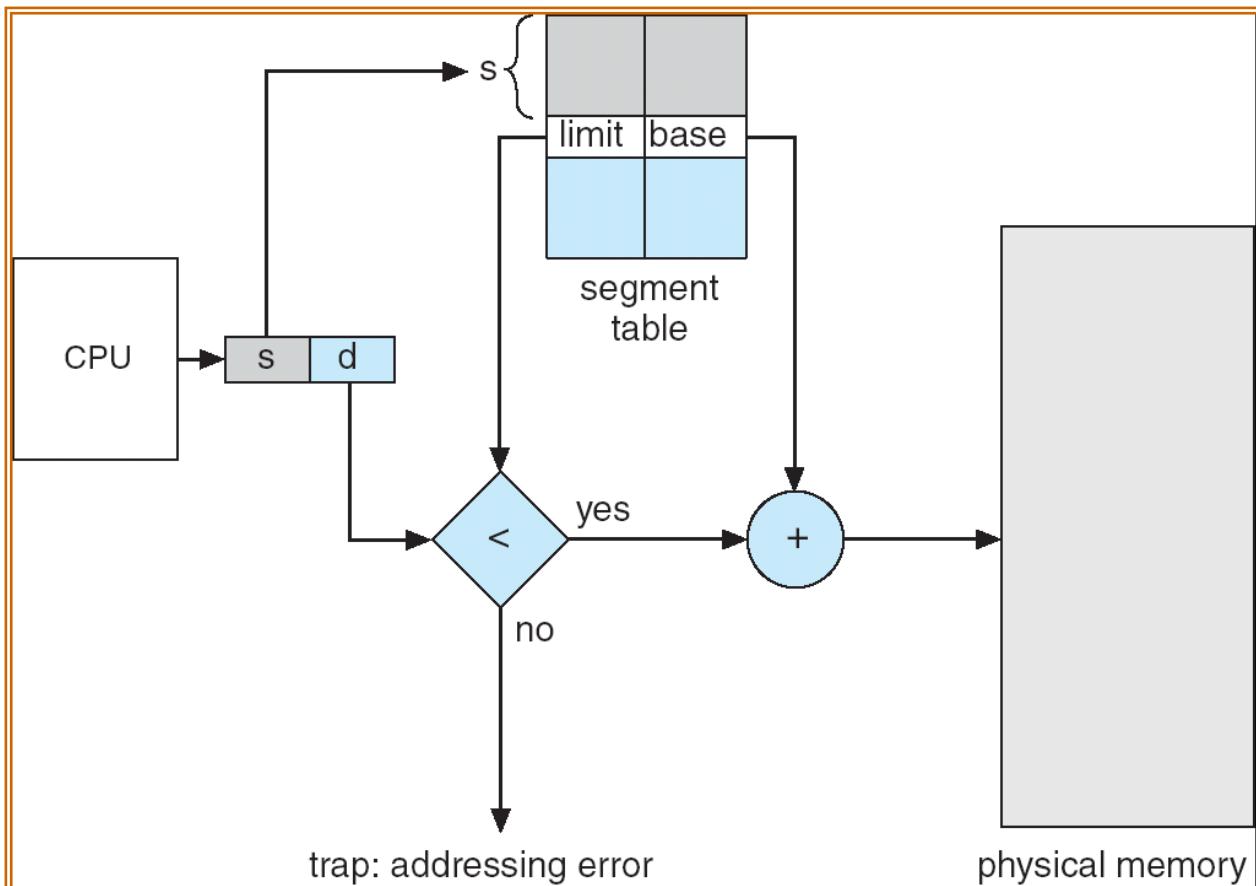
- ◆ Logička adresa se sastoji od dva dela
<broj_segmenta, offset>
- ◆ Tabela segmenata – transformiše logičku u fizičku adresu
- ◆ Ulaz u tabelu sadrži:
 - Baza (*Base*) – sadrži startnu fizičku adresu segmenta u memoriji
 - Limit (*Length*) – specificira veličinu segmenta
 - Zaštita – specificira prava pristupa sadržaju segmenta
 - Validnost – da li je segment u memoriji ili na disku
- ◆ Bazni registar **tabele segmenata** – sadrži adresu početka tabele segmenata u memoriji
- ◆ Granični registar **tabele segmenata-** sadrži broj segmenata programa ili adresu završetka tabele segmenata

Segmentacija - primer

- Smeštanje u memoriju procesa sa pet segmenata

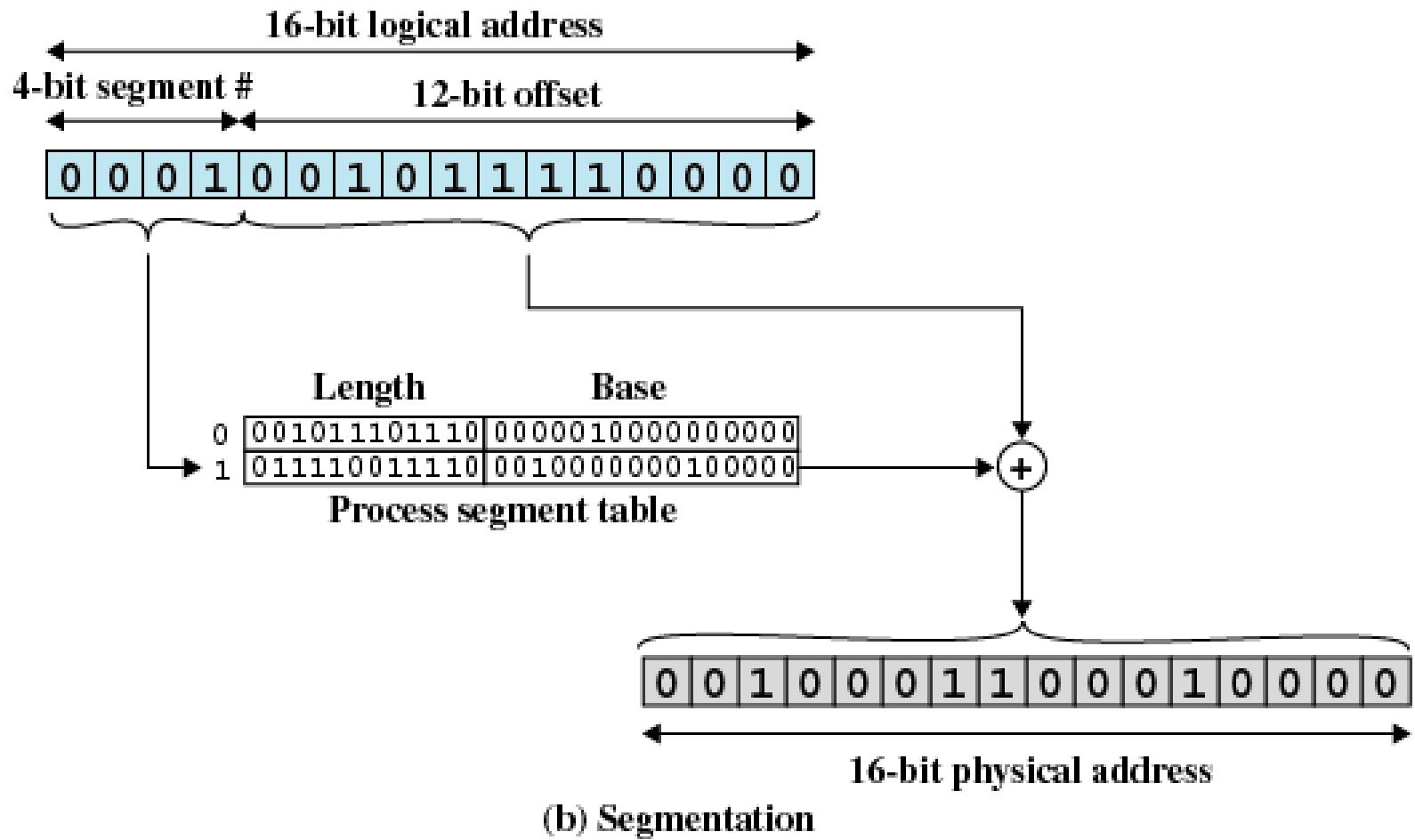


Prevodenje logičke u fizičku adresu korišćenjem tabele segmenata



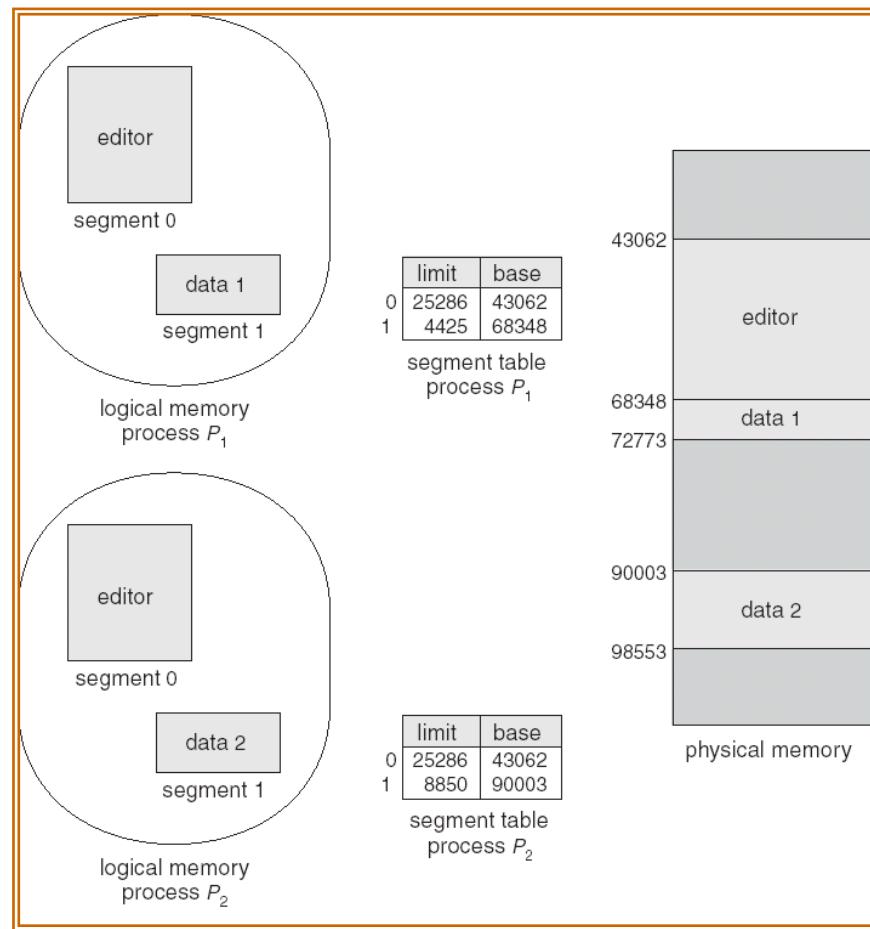
Silberschatz, 2011

Prevođenje logičke u fizičku adresu - primer



Deljenje segmenata

- Više procesa koristi tekst editor, tako da dele segment sa kôdom editora, dok svaki proces ima sopstveni segment sa podacima



Silberschatz, 2011



Domaći zadatak

- ❖ Poglavlje **7 Upravljanje memorijom**
 - ❖ 7.8 Ključni pojmovi, kontrolna pitanja i problemi
- ❖ Paging & segmentation animations
 - ❖ <https://apps.uttyler.edu/Rainwater/COSC3355/Animations>



Operativni sistemi

- Virtuelna memorija -

Prof. dr Dragan Stojanović

Katedra za računarstvo
Univerzitet u Nišu, Elektronski fakultet



Literatura

- ➊ *Operating Systems: Internals and Design Principles*, edition, W. Stallings, Pearson Education Inc., 7th – 2012, (5th -2005, 6th - 2008, 8th – 2014 , 9th – 2017)
 - <http://williamstallings.com/OperatingSystems/>
 - <http://williamstallings.com/OperatingSystems/OS9e-Student/>
- ➋ Poglavlje 8: Virtuelna memorija



Virtuelna memorija

- ❖ Nije neophodno da sve stranice ili segmenti procesa budu u glavnoj memoriji u toku izvršavanja
- ❖ Logički adresni prostor procesa može biti veći od fizičkog adresnog prostora (raspoložive memorije)
- ❖ Samo neophodni delovi procesa su u memoriji (rezidentan skup), a ostatak je na disku
- ❖ Kada proces u svom izvršavanju referencira logičku adresu u okviru stranice (segmenta) koja nije u glavnoj memoriji, generiše se prekid koji ukazuje na grešku u pristupu memoriji i tražena stranica se smešta u memoriju
 - ▣ Straničenje na zahtev (*Demand paging*)
 - ▣ Segmentacija na zahtev (*Demand segmentation*)
- ❖ Omogućava deljenje adresnog prostora od strane više procesa

Tabela stranica

- Virtuelna (logička) adresa sadrži:
 - Broj stranice (bitovi višeg reda)
 - Ofset unutar stranice (bitovi nižeg reda)
- Broj stranice se koristi kao indeks u tabelu stranica u cilju nalaženja odgovarajućeg broja straničnog okvira u određenoj stavci (ulazu, *Page Table Entry*) u tabeli stranica
- Broj straničnog okvira se pridružuje kao viši deo bitovima ofseta i formira fizičku adresu

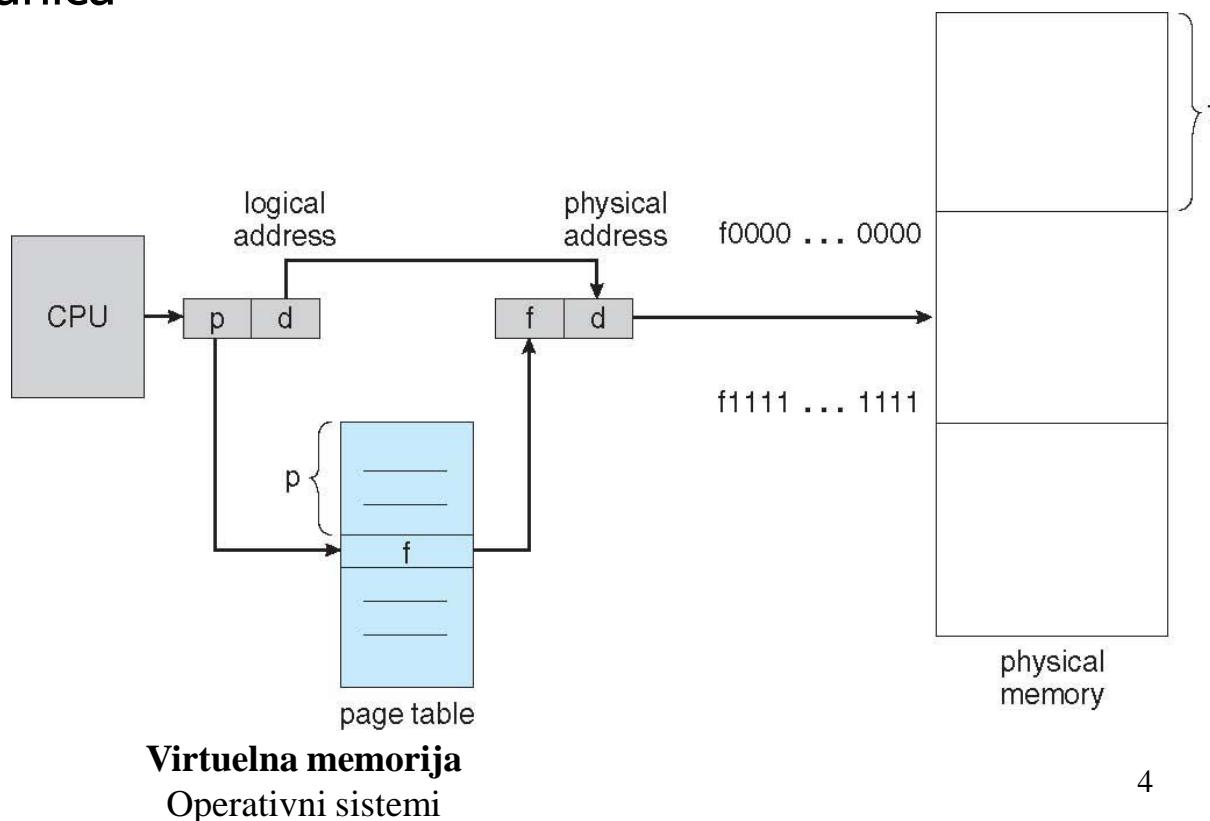
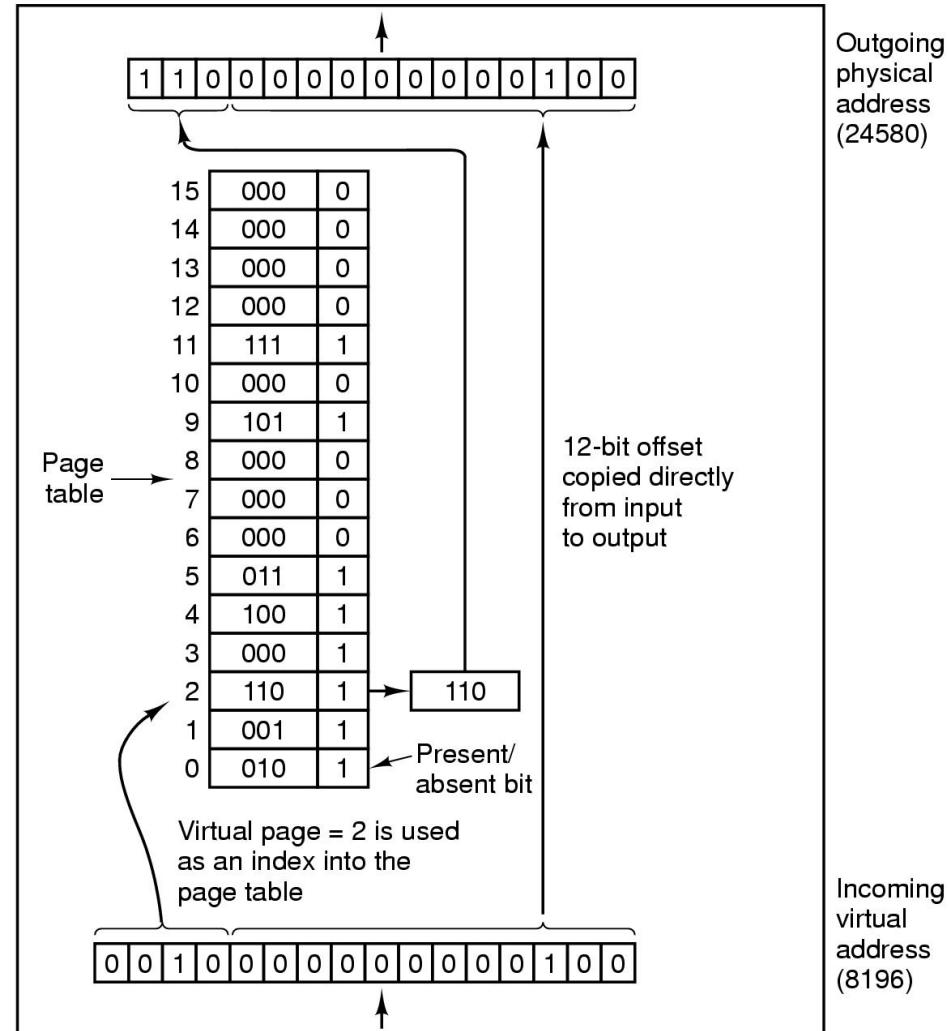


Tabela stranica - primer

- U 16 bitnoj virtuelnoj adresi pri stranicama veličine 4KB,
 - viša 4 bita specificiraju jednu od 16 virtuelnih stranica
 - nižih 12 bitova specificiraju offset (pomeraj) u okviru stranice (0-4095)
- Bit u tabeli stranica specificira da li je stranica prisutna u memoriji u odgovarajućem straničnom okviru (*Valid-Invalid, Present/Absent* bit)

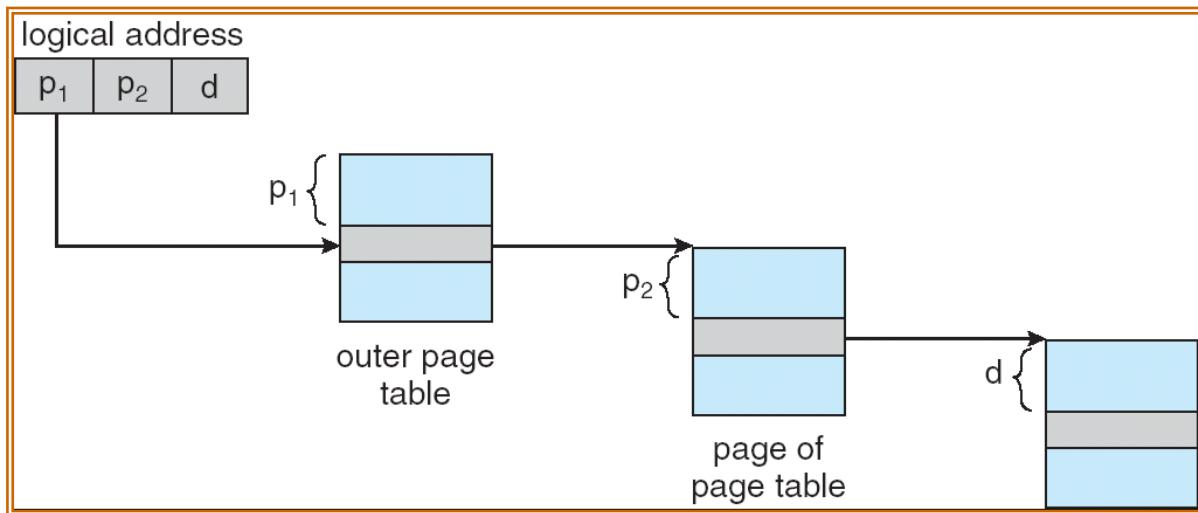


Implementacija tabele stranica

- ➊ Tabela stranica **je smeštena u glavnoj memoriji**
 - ▣ Bazni registar tabele stranica (*Page-table base register* - PTBR) sadrži adresu početka tabele stranica
 - ▣ Registar veličine tabele stranica (*Page-table length register* - PRLR) sadrži veličinu tabele stranica
- ➋ **Problemi:**
 - ▣ Veličina tabele stranica - Za virtuelne adrese od 32 bita i veličinu stranice od 4KB, broj stranica je 1 M, tako da je neophodno isto toliko stavki (ulaza) u tabeli stranica
 - ▣ Svaki pristup instrukcijama i podacima procesa zahteva dva pristupa memoriji, jedan za pristup tabeli stranica, a drugi pristup instrukciji/podacima
- ➌ Implementacija tabele stranica
 - ▣ Kontinualne tabele stranica u
 - Brzim hardverskim registarima
 - Kontinualnom području u memoriji
 - ▣ Tabele stranica u više nivoa
 - ▣ Invertovane (obrnute) tabele stranica

Tabele stranica u više nivoa

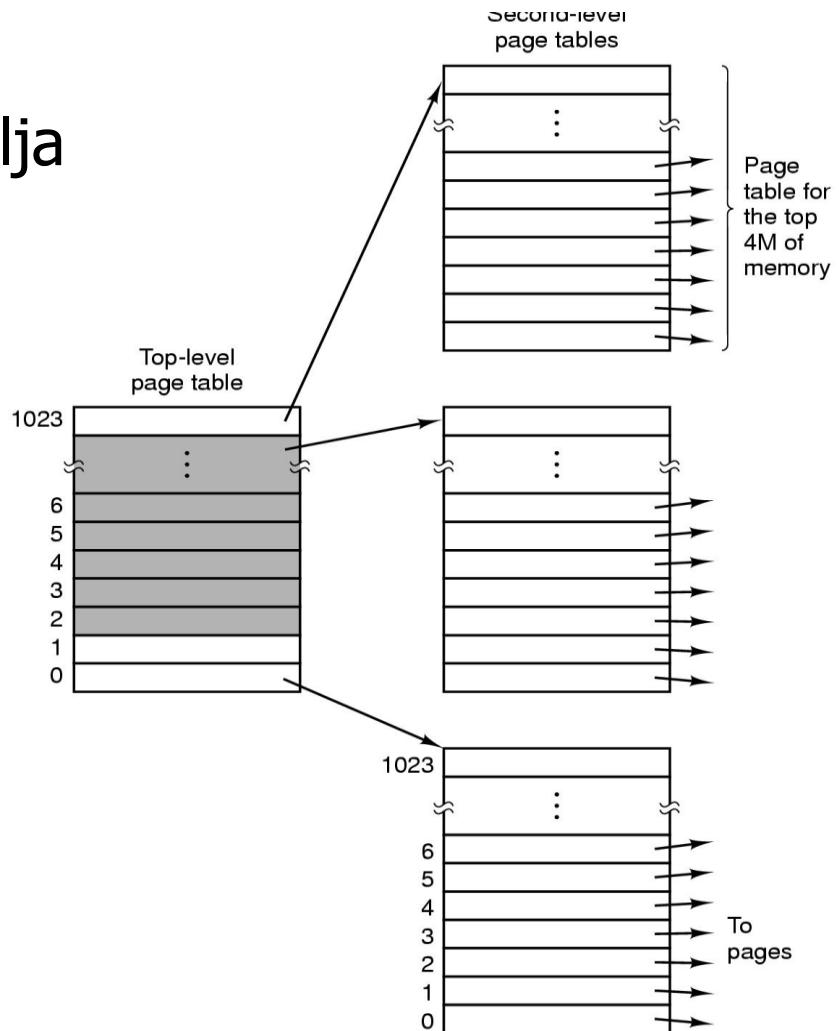
- Tabela stranica se predstavlja u obliku stranica (straniči se)
- Virtuelna adresa se deli na tri polja
 - PT1 – indeks u tabelu stranica prvog nivoa
 - PT2 – indeks u tabelu stranica drugog nivoa
 - Offset – memorijska adresa u okviru stranice
- Nema potrebe čuvati sve tabele stranica u memoriji već samo one potrebne
- Može biti tri, četiri ili više nivoa tabela stranica



Virtuelna memorija
Operativni sistemi

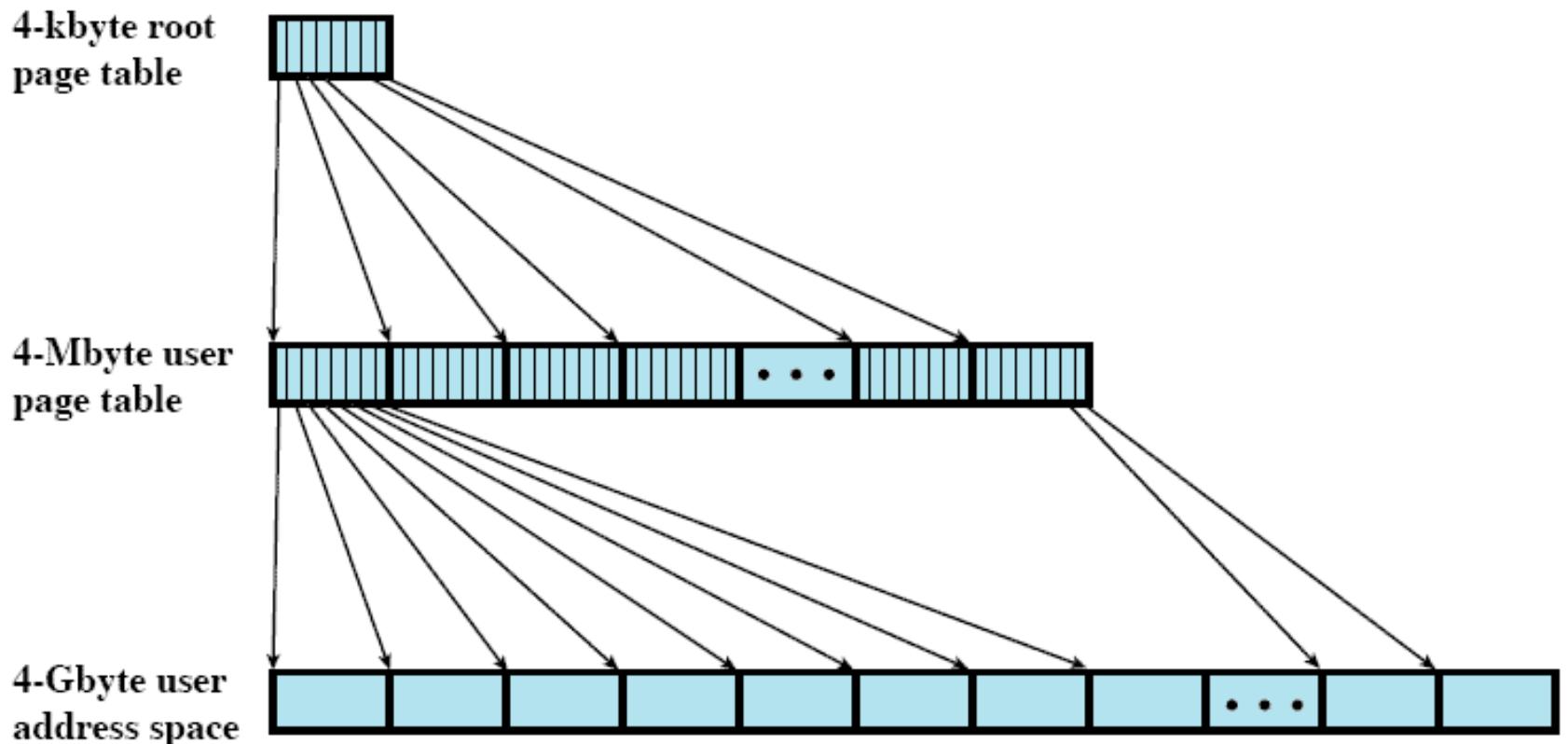
Tabele stranica u dva nivoa

- Stranice veličine 4KB
- 32-bitna adresa se deli na tri polja
 - PT1 – 10 bita
 - PT2 – 10 bita
 - Offset – 12 bitova
- Sve tabele stranica su veličine jedne stranice (4KB), a svaka stavka u tabeli 32 bita
- Za proces adresnog prostora 4MB za kod, 4MB za podatke i 4MB za stek ukupno je potrebno 4 tabele stranica



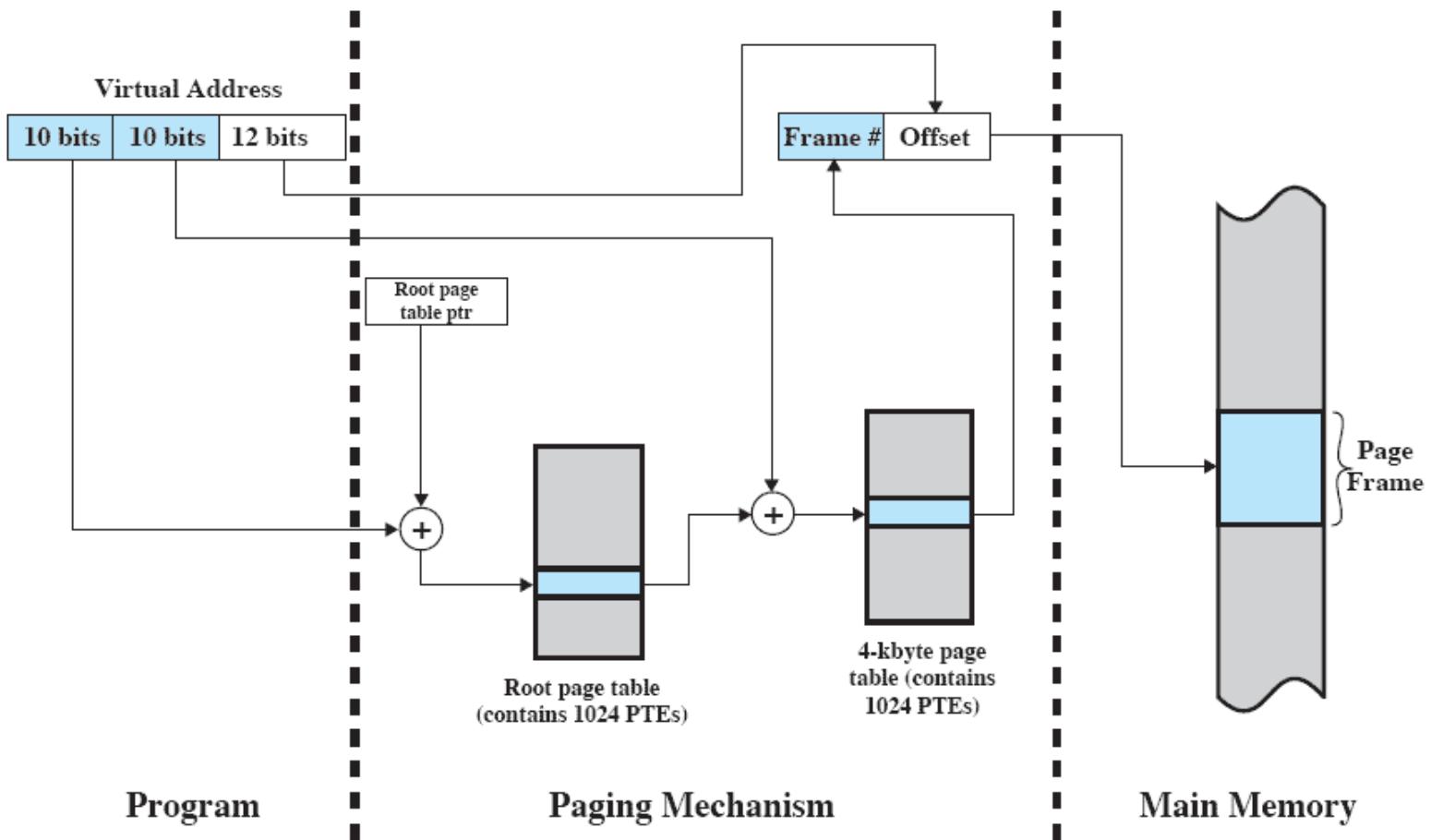
Tabele stranica u dva nivoa (2)

- Hijerarhijska šema tabela stranica u dva nivoa



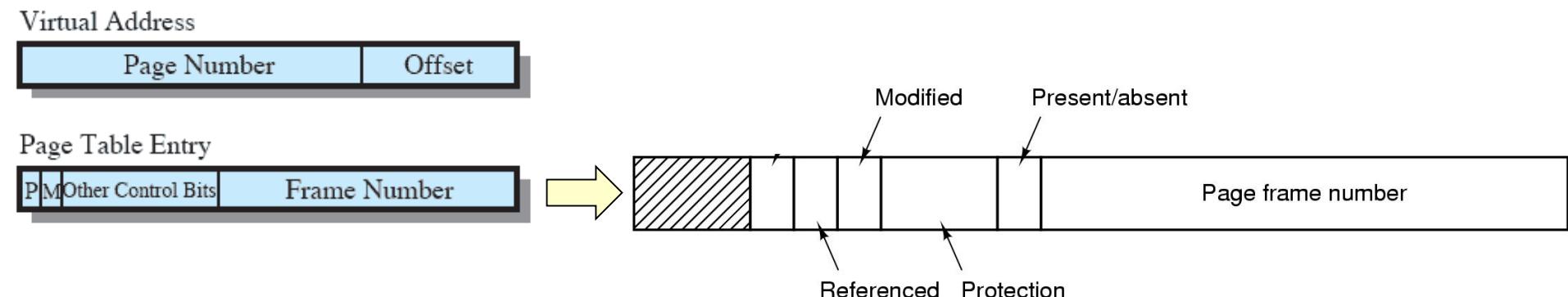
Tabele stranica u dva nivoa (3)

Prevođenje virtuelne adrese



Struktura stavke u tabeli stranica

- Uobičajena veličina stavke (ulaza) iznosi 32 bita
 - Page Frame Number – broj fizičke stranice
 - Valid (*Present*) bit – ako je bit postavljen na 1, stranica je u memoriji; pristup stranici sa bitom vrednosti 0 generiše grešku zbog stranice (*page fault*)
 - Zaštita – prava pristupa sadržaju stranice (čitanje, upis, izvršavanje)
 - Bit modifikacije (*dirty*) – da li je stranica modifikovana u memoriji
 - Bit referenciranja (korišćenja) – postavlja se kad god je stranica referencirana; koristi se kod zamene stranica



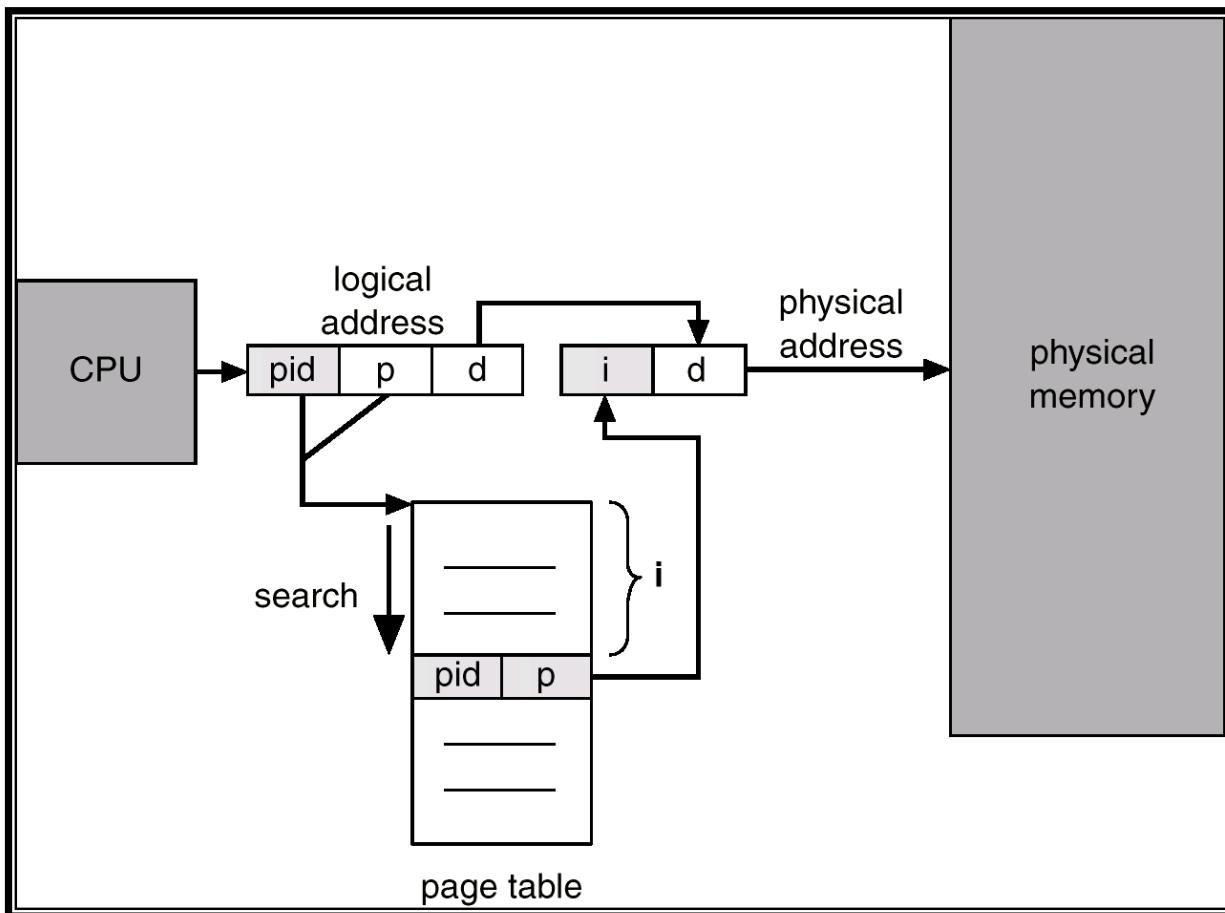


Invertovane tabele stranica (1)

- ◆ U tabeli stranica postoji **jedna stavka za svaku fizičku stranicu** (stranični okvir) u memoriji
- ◆ Svaka stavka u tabeli stranica **sadrži virtualnu adresu stranice** koja je smeštena u datu fizičku stranicu, kao i podatke o procesu kojem ta stranica pripada
- ◆ **Smanjuje memorijski prostor** neophodan za smeštanje tabele stranica, ali **povećava vreme** neophodno za pretraživanje tabele po određenom broju virtualne stranice
- ◆ Koristi se heš (hash) tabela, hešovana po virtualnoj adresi za ubrzanje pretraživanja po prethodno opisanom principu
- ◆ Da bi se izbegao dodatno referenciranje memorije uz svaki memorijski pristup koristi se TLB
- ◆ Implementacija na PowerPC, UltraSPARC i IA-64

Invertovane tabele stranica (2)

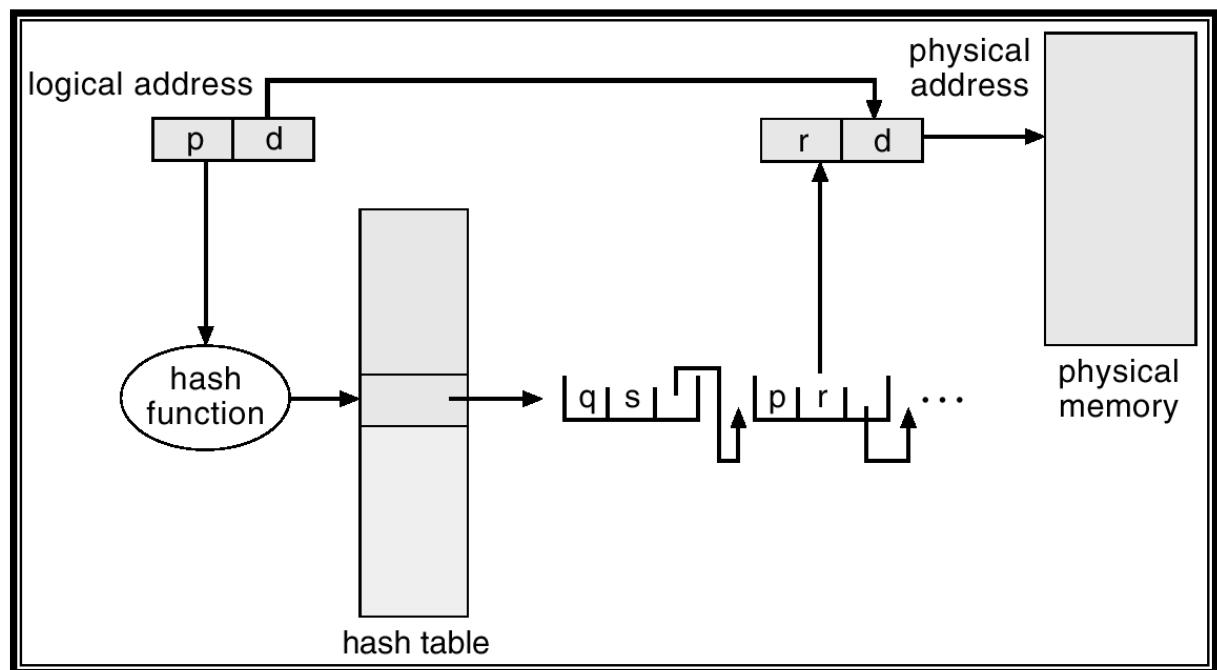
- Transformacija virtuelne (logičke) adrese u fizičku korišćenjem invertovane tabele stranica



Virtuelna memorija
Operativni sistemi

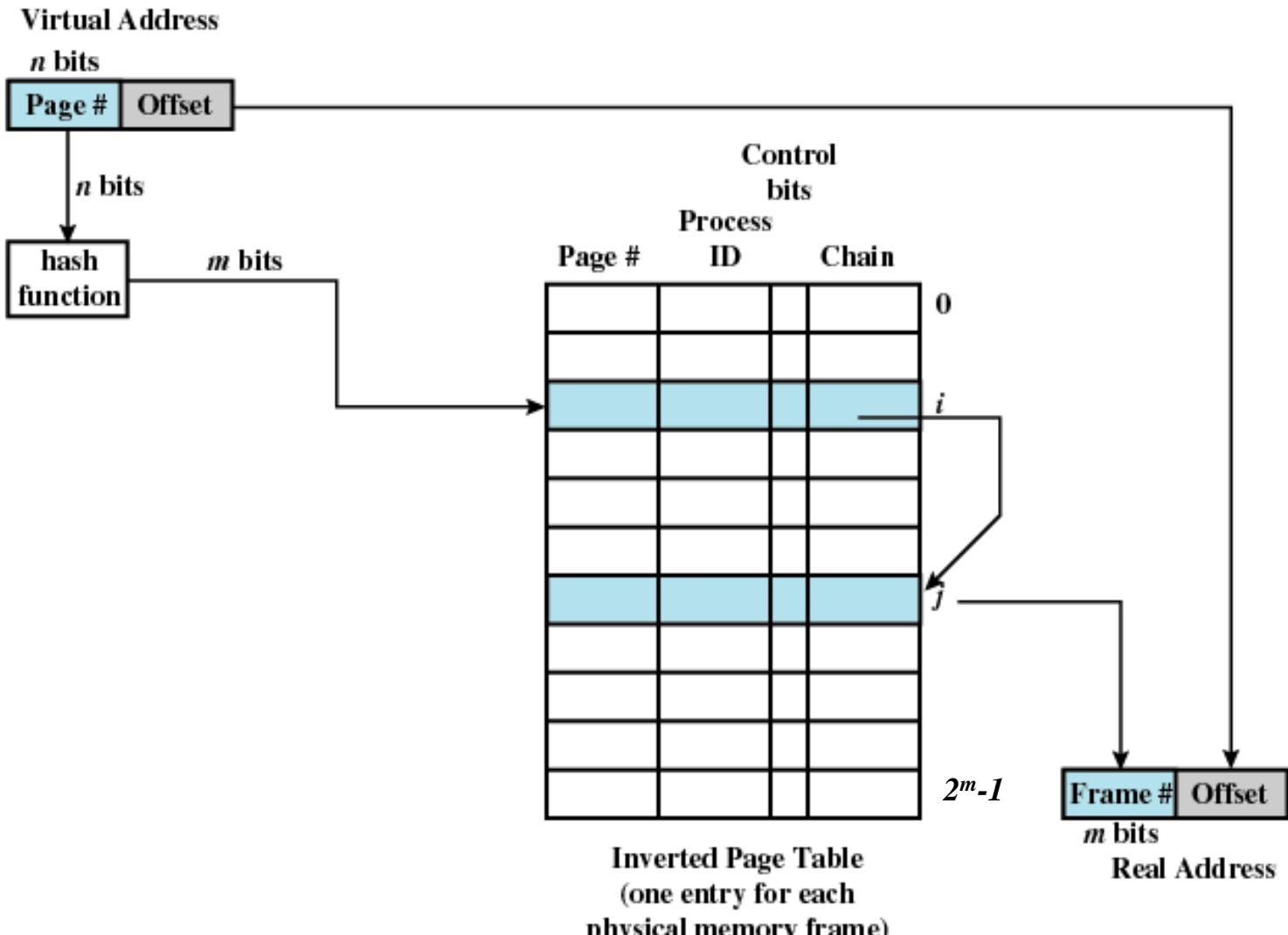
Invertovane tabele stranica sa heš funkcijom

- Broj virtuelne stranice se preslikava u heš vrednost korišćenjem heš funkcije. Heš vrednost predstavlja indeks (pokazivač) na invertovanu tabelu stranica čija svaka stavka sadrži lančanu listu elementa formata (virtuelne stranica, stranični okvir) koji su heš funkcijom preslikani u istu heš vrednost
- Na osnovu broja virtuelne stranice pretražuje se lista i određuje broj straničnog okvira



Virtuelna memorija
Operativni sistemi

Struktura invertovane tabele stranica



TLB – *Translation Lookaside Buffer*

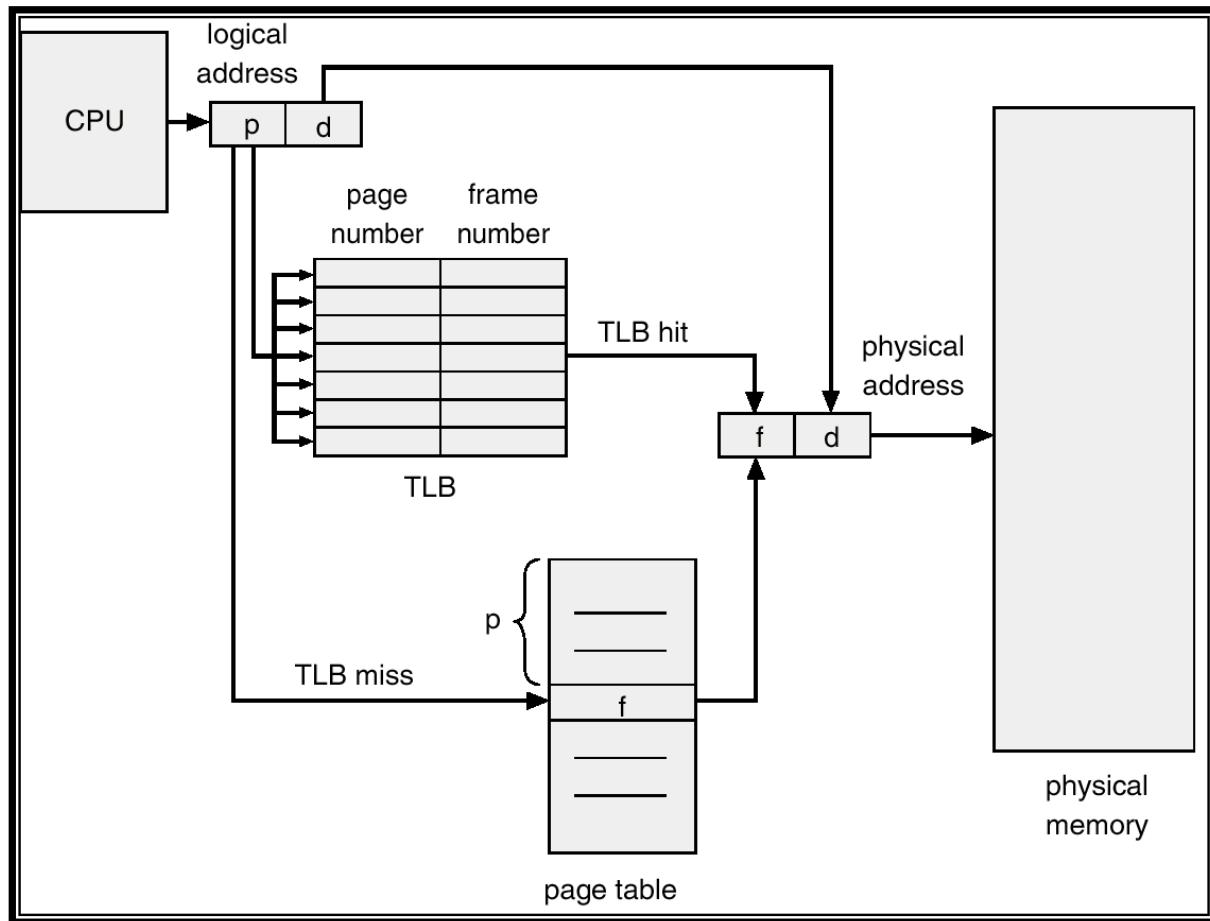
- Hardverska komponenta u okviru MMU za transformisanje (prevodenje) virtuelnih u fizičke adrese bez pristupa tabeli stranica
- Asocijativna memorija –paralelno pretraživanje svih stavki u TLB-u po datom ključu (broju virtuelne stranice)
- Ima od 64 – 1024 stavki formata kao na sledećoj slici

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Virtuelna memorija
Operativni sistemi

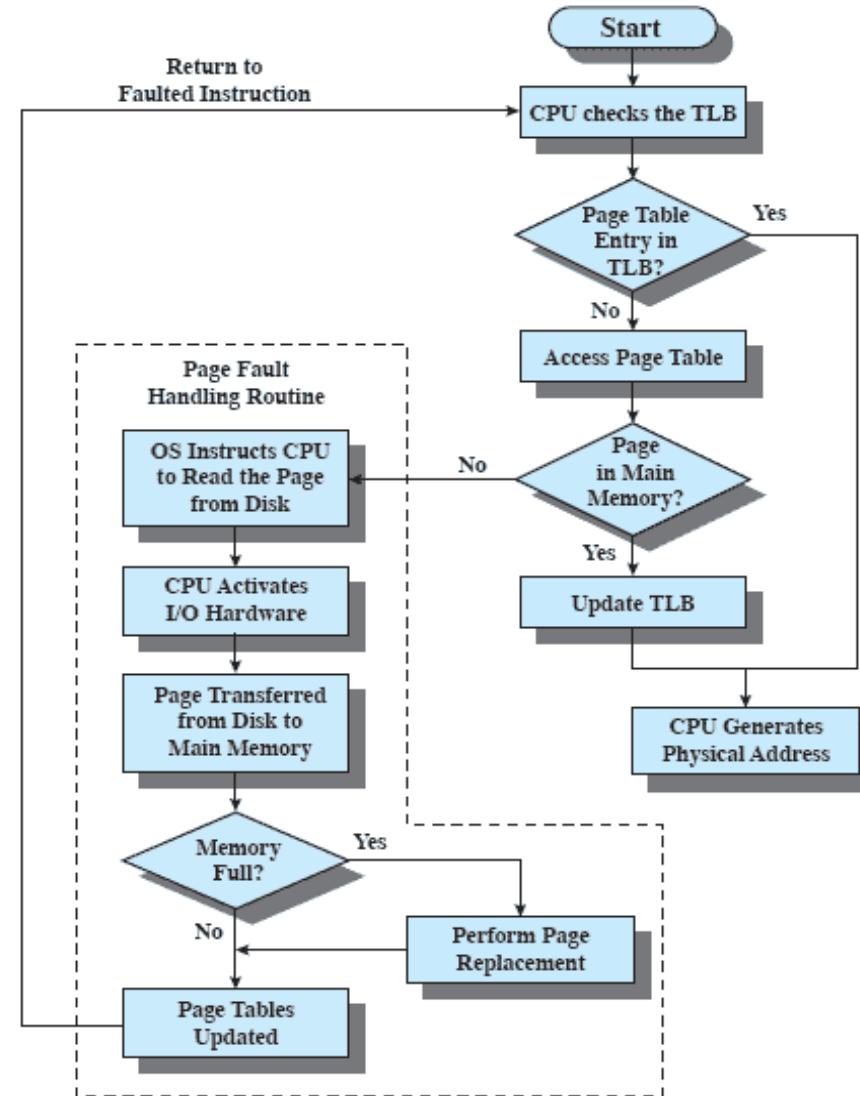
Prevođenje adresa korišćenjem TLB

- Upravljanje TLB može biti hardversko od strane MMU, ili softversko od strane OS (RISC – SPARC, MIPS, Alpha, HP PA, itd.)



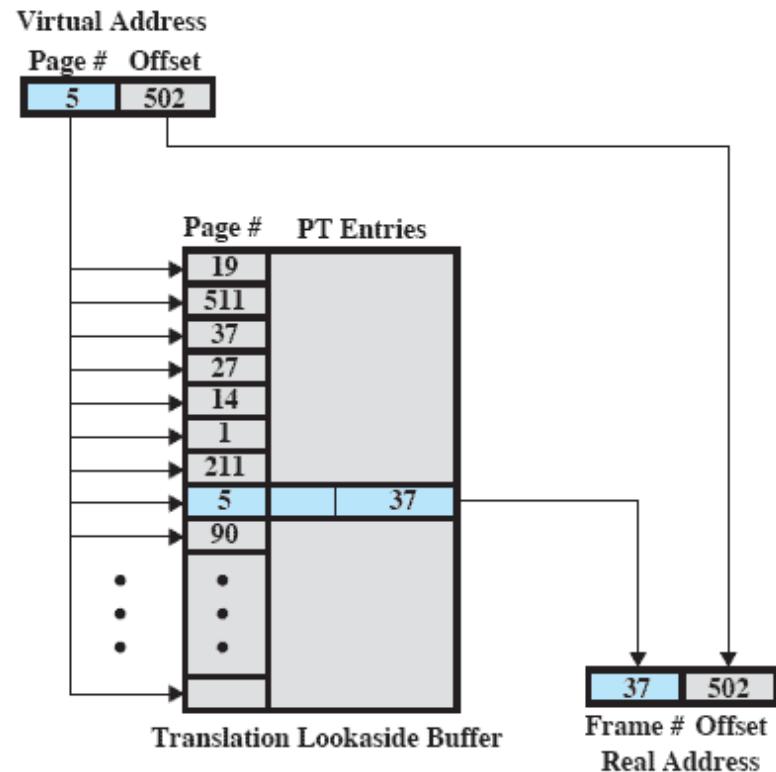
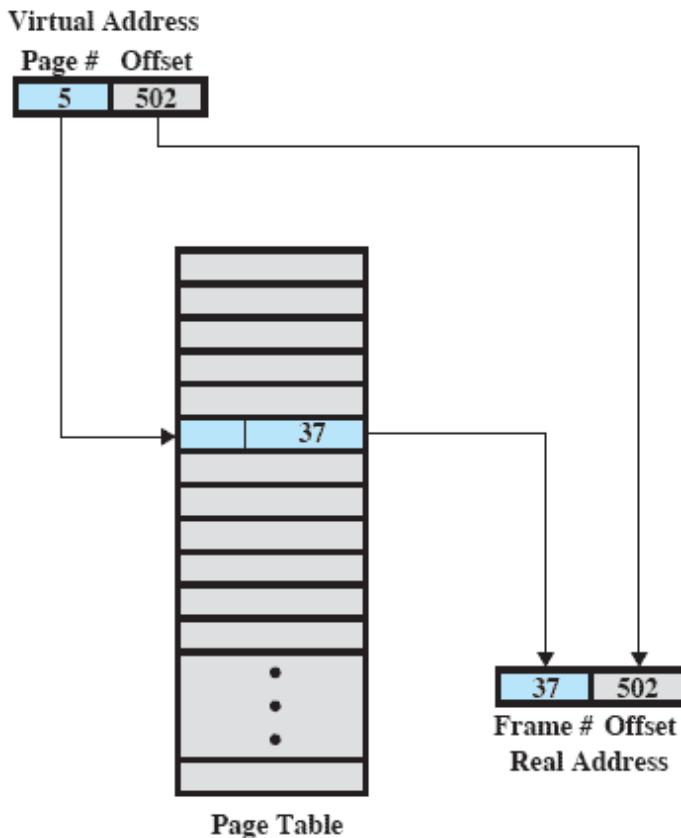
Algoritam prevodenja adresa

- Algoritam prevodenja adresa korišćenjem TLB i tabele stranica

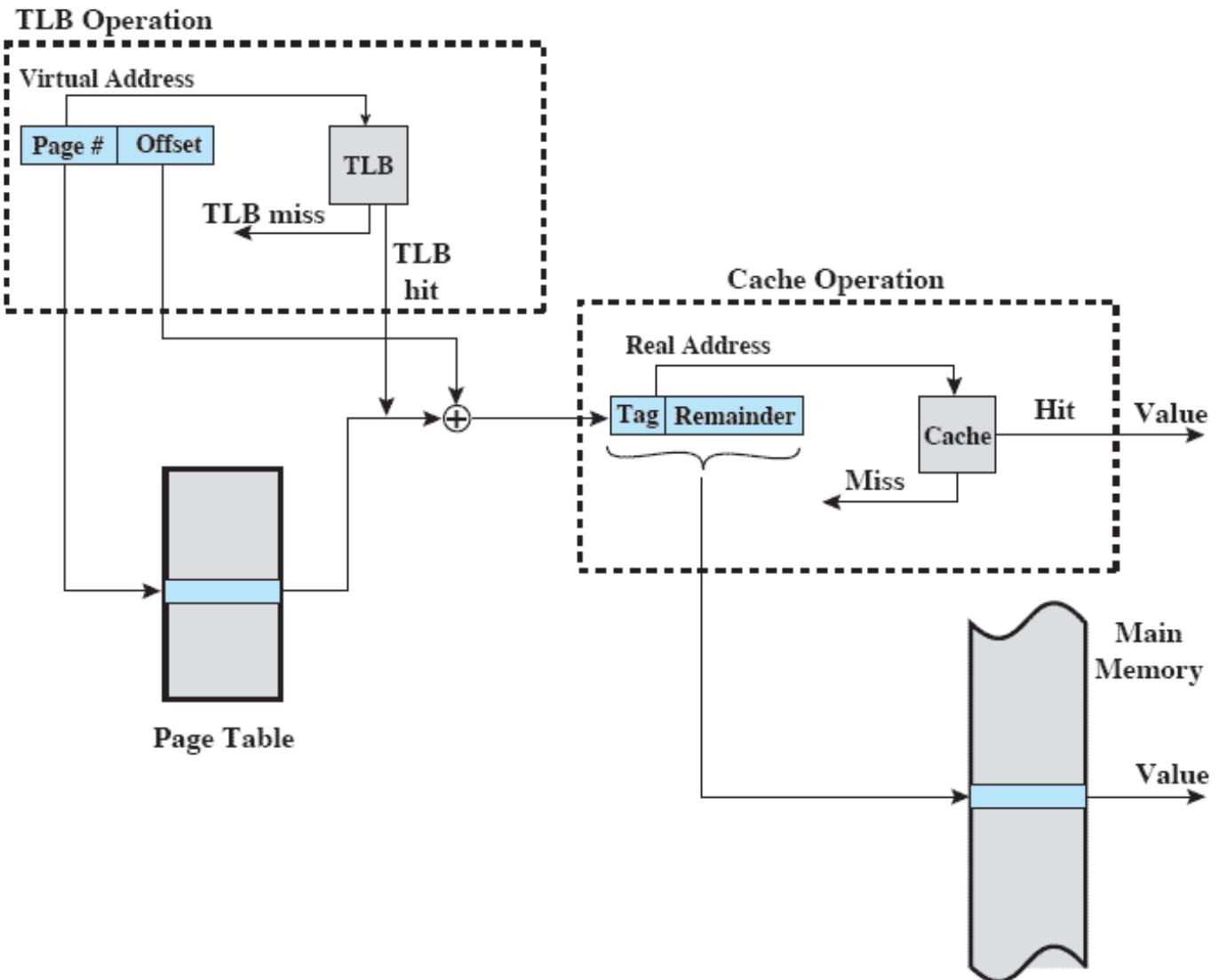


Direktno i asocijativno traženje

- Razlika između direktnog i asocijativnog traženja (lookup) stavke u tabeli stranica



TLB i keš memorija



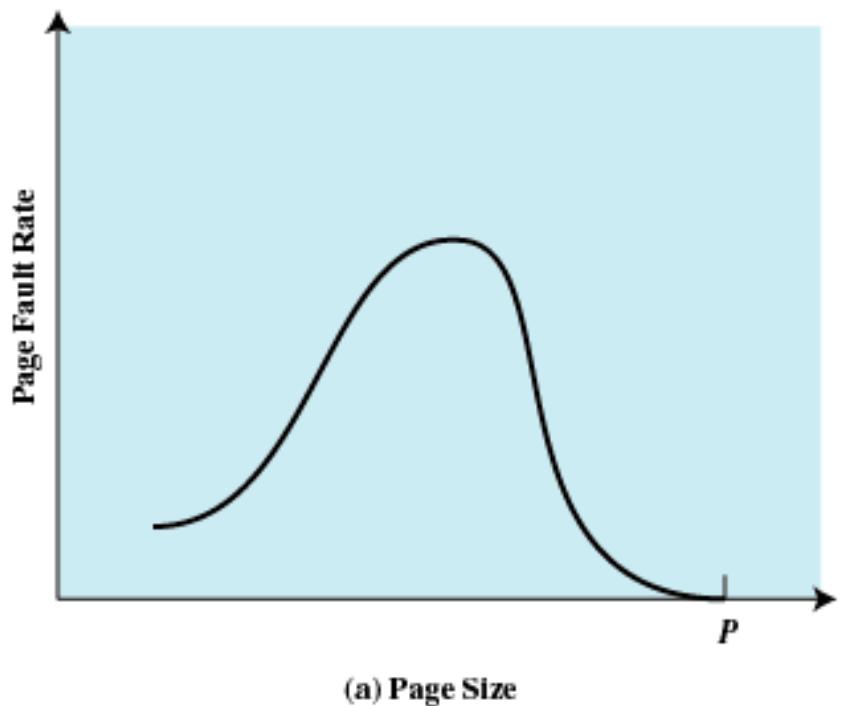
Virtuelna memorija
Operativni sistemi



Veličina stranice

- ◆ Manja veličina stranice
 - ▣ Manja količina interne fragmentacije unutar poslednje stranice
 - ▣ Više stranica je neophodno za svaki proces, što znači veće tabele stranica
 - ▣ Veće tabele stranica znači da i tabele stranica **moraju da se straniče**, tj. da se formiraju u virtuelnoj memoriji, tako da za jednu memorijsku referencu može da postoji dvostruka greška stranica, jedna zbog tabele stranica, druga zbog same stranice
- ◆ Veće stranice
 - ▣ Sekundarna memorija je tako projektovana da efikasno prenosi velike blokove podataka tako da je prednost u većim stranicama radi efikasnijeg blokovskog prenosa
 - ◆ Sa manjom veličinom stranice u memoriji može da bude veliki broj stranica procesa i relativno je mali broj grešaka stranice (*page fault*)
 - ◆ Sa povećanjem veličine stranice gubi se princip lokalnosti i učestalost grešaka stranica raste, međutim na kraju učestalost opada sve do 0 kada je veličina stranice jednaka veličini procesa (ceo proces je u memoriji)

Veličina stranice (2)



P = size of entire process

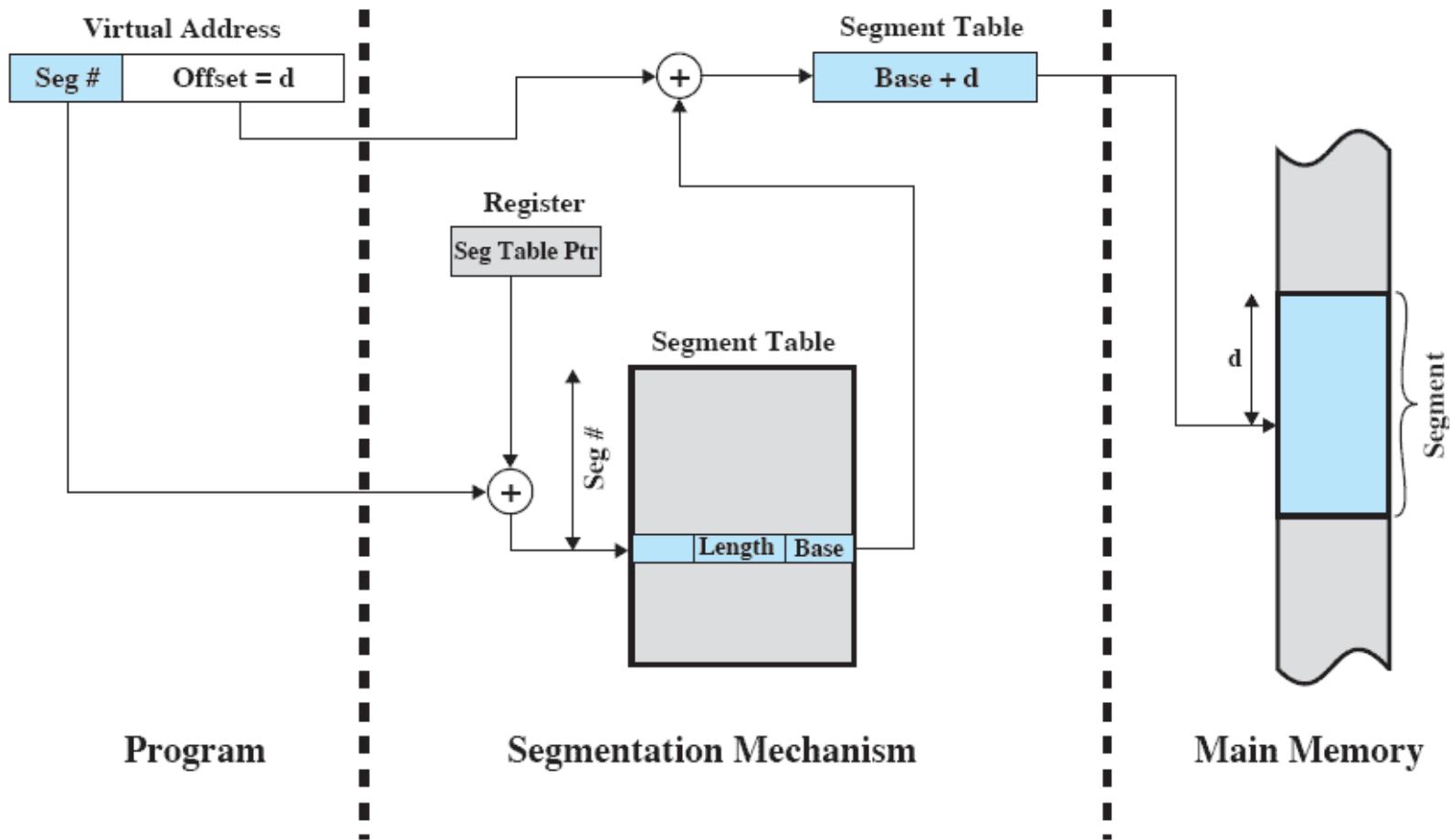
W = working set size

N = total number of pages in process

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1,024 36-bit words
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 Kbytes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
Intel Itanium	4 Kbytes to 256 Mbytes
Intel core i7	4 Kbytes to 1 Gbyte

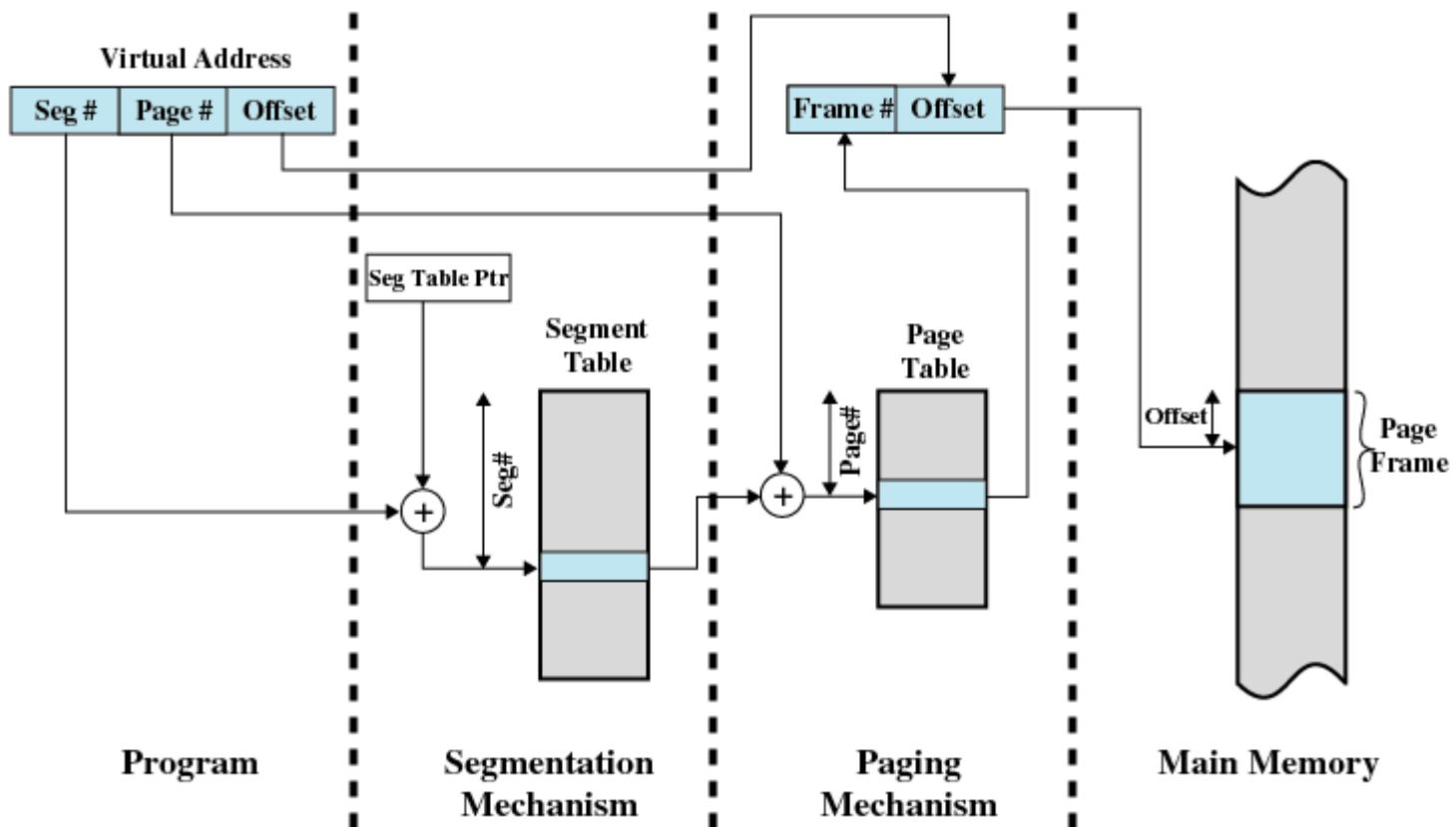
Segmentacija

- Prevođenje adrese u sistemu segmentacije



Segmentacija sa straničenjem

- Program je logički podeljen na segmente, a svaki segment na stranice
- Uključuje dobre karakteristike obe šeme upravljanja memorijom



Segmentacija sa straničenjem (2)

- Virtuelna adresa, stavka tabele segmenata i stavka tabele stranica kod segmentcije sa straničenjem

Virtual Address



Segment Table Entry



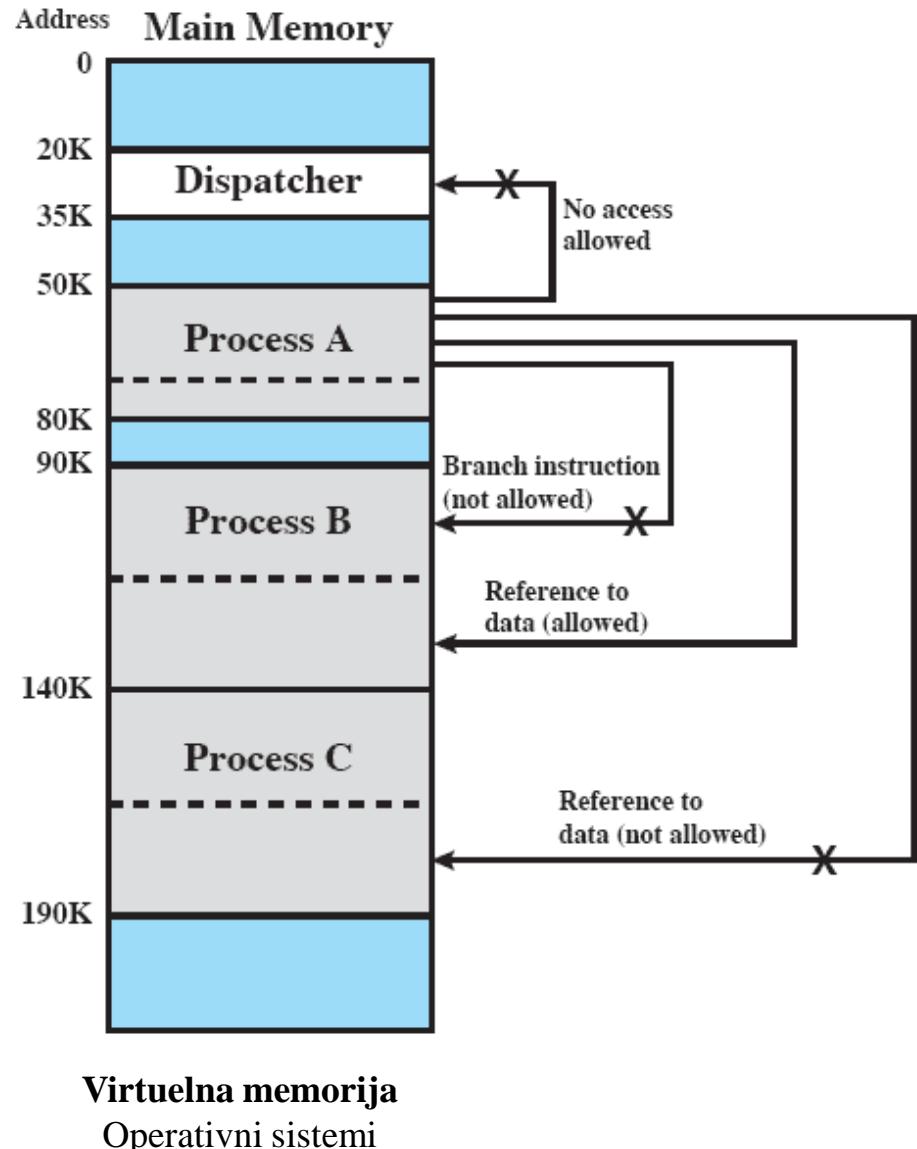
Page Table Entry



P= present bit
M = Modified bit

Zaštita i deljenje memorije

◆ Mehanizmi zaštite segmenata





Softver operativnog sistema za upravljanje memorijom

● Sistem za upravljanje memorijom OS određen je sledećim:

- Da li koristi tehnike virtuelne memorije
- Da li koristi straničenje ili segmentaciju
- Koje algoritme koristi za različite aspekte upravljanja memorijom

● Algoritmi (strategije) za upravljanje memorijom

- Politika učitavanja (donošenja) stranice
- Politika smeštanja stranice
- Politika zamene
- Upravljanje rezidentnim skupom
- Politika čišćenja
- Upravljanje učitavanjem



Politika učitavanja (donošenja)

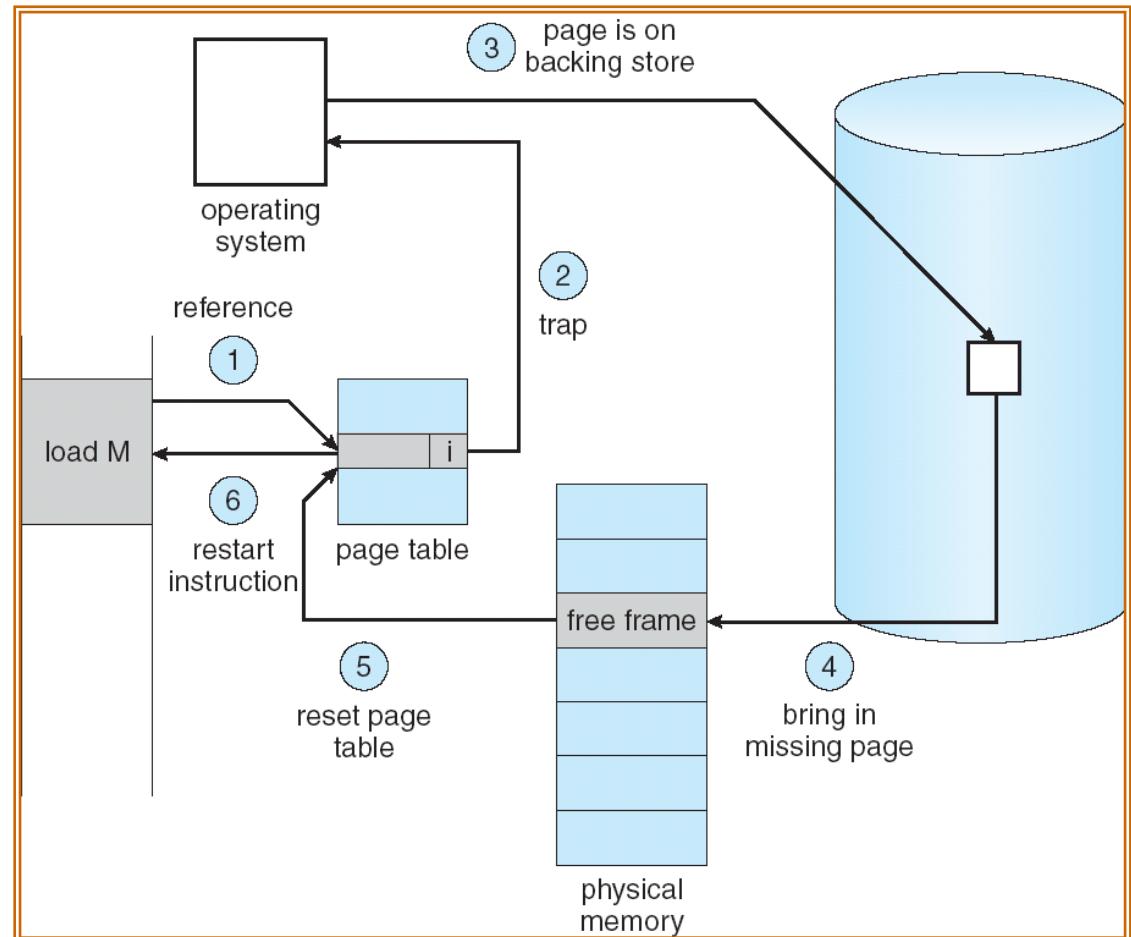
- ➊ Određuje kada stranica treba da bude učitana u glavnu memoriju
- ➋ Straničenje na zahtev (*demand paging*) učitava stranice u memoriju samo kada se napravi referenca na memorijsku lokaciju (adresu) unutar te stranice
 - ▣ Uzorkuje veliki broj grešaka stranice kada se proces startuje
- ➌ Prestraničenje (*Prepaging*) učitava u memoriju više stranica nego što je trenutno potrebno, ali se očekuje da će u bliskoj budućnosti biti
 - Efikasnije je učitati stranice koje su kontinualno smeštene na disku

Straničenje na zahtev

- ❖ Stranice se smeštaju (*load*) u glavnu memoriju **tek kada se refrencira** memorijska lokacija u okviru stranice
 - ❖ Na početku izvršavanja procesa generiše se veliki broj grešaka stranice (*page fault*)
- ❖ Kada se javi greška stranice, operativni sistem treba da stranicu učita sa diska, smesti u slobodni stranični okvir i ažurira tabelu stranica
- ❖ Ukoliko nema slobodnog straničnog okvira u memoriji, operativni sistem mora da odabere jednu stranicu koju će da obriše iz memorije i na njeno mesto smesti stranicu koja je generisala grešku - **Politika i algoritmi zamene stranica** (*Page replacement*) - generalni algoritmi primenljivi na keš memorije ili keš Web servera
- ❖ Ukoliko je stranica koja se briše modifikovana u memoriji, ona mora biti sačuvana na disku, ukoliko nije menjana (na primer, stranica sa kôdom programa), njena kopija na disku je ažurna i ona se samo prepisuje novom stranicom

Obrada greške stranica

1. Instrukcija referencira memorijsku lokaciju
2. Valid = 0 -> Trap u OS
3. Nalaženje odgovarajuće stranice na disku
4. Smeštanje stranice sa diska u memoriju (po potrebi oslobođiti stranični okvir u memoriji)
5. Ažuriranje tabele stranica
6. Restartovanje instrukcije



Efektivno vreme pristupa memoriji

- ◆ p – verovatnoća nastanka greške stranica ($0 \leq p \leq 1$)
- ◆ Vreme za čuvanje stranja prekinutog procesa, obradu prekida i restartovanje prekinutog procesa je reda nekoliko 100 mikrosekundi do 1ms
- ◆ Vreme za zamenu stranica = rotation latency + seek + transfer
 $= 8 \text{ ms} + 15 \text{ ms} + 1 \text{ ms} = 24\text{ms}$
- ◆ Vreme pristupa memoriji je 10ns – 200ns zavisno da li je sadržaj u kešu ili ne; srednje vreme 100ns
- ◆ **Efektivno vreme pristupa** = $(1 - p) * 100 \text{ ns} + p * (25 \text{ ms} + 100 \text{ ns})$
 $= 100 \text{ ns} + p * 25\,000\,000 \text{ ns}$
- ◆ Ukoliko želimo da **Efektivno vreme pristupa (EFP)** bude do 10% lošije u odnosu na vreme pristupa bez grešaka stranica, mora se postići pojava najviše jedne greške stranica na $2.5 * 10^6$ pristupa memoriji

$$\text{EFP} = 110 \text{ ns} \quad \longrightarrow \quad p = 1 / 2.5 * 10^6$$



Politika smeštanja

- ➊ Određuje gde u stvarnoj memoriji proces treba da bude smešten (stranice procesa)
- ➋ U sistemu sa čistom segmentacijom politika smeštanja je važna zbog problema fragmentacije (najbolje poklapanje, prvo poklapanje, itd.)
- ➌ Za sistem koji koristi čisto straničenje ili segmentaciju sa straničenjem smeštanje je obično nevažno, jer je prevođenje virtuelnih adresa u fizičke podjednako efikasno za bilo koju kombinaciju stranica-okvir



Politika zamene

- ❖ Kada su svi stranični okvir u glavnoj memoriji zauzeti **politika zamene** određuje koju stranicu iz glavne memorije treba zameniti kad treba da se učita nova stranica
- ❖ Trebalo bi zameniti onu stranicu koja će sa najmanjom verovatnoćom biti referencirana u budućnosti
- ❖ Politike zamene pokušavaju da predvide buduće referenciranje stranica na osnovu referenciranja u prošlosti
- ❖ Zaključavanje straničnih okvira – stranice u zaključanim okvirima ne mogu da se zamene; okvir se zaključava na osnovu bita zaključavanja pridruženog svakom okviru
 - ▣ Kernel OS
 - ▣ Glavne upravljačke strukture
 - ▣ U/I baferi



Algoritmi zamene stranica

- ❖ Cilj je postizanje minimuma grešaka stranica
- ❖ Evaluacija algoritama na osnovu niza memorijskih referenci i izračunavanja broja grešaka stranica na tom nizu referenci
- ❖ Algoritmi:
 - ❖ Optimalni algoritam zamene stranica
 - ❖ Prva unutra, prva napolje (*FIFO – First-In, First-Out*)
 - ❖ Najmanje skoro korišćena stranica (*LRU – Least Recently Used*)
 - ❖ Algoritam časovnika (*Clock*)

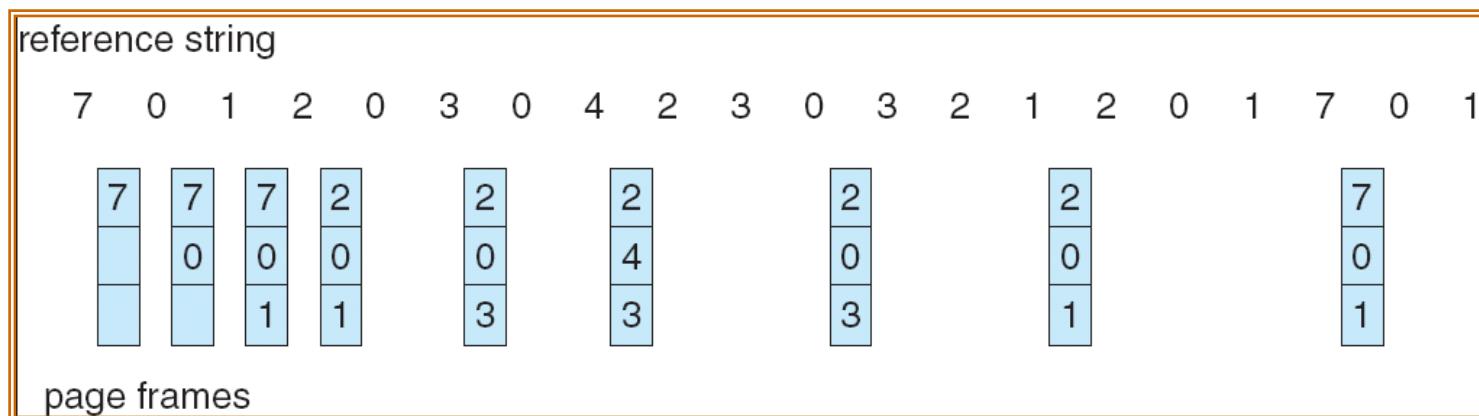


Optimalni algoritam zamene stranica

- ❖ Najbolji algoritam, ali ga je nemoguće implementirati
- ❖ U trenutku nastanka greške stranica u memoriji su smeštene stranice različitih procesa; svaka od njih će (moguće) biti referencirana u budućnosti posle određenog broja instrukcija
- ❖ Za zamenu se bira ona stranica koja će najkasnije biti referencirana, tako da se greška stranica što je moguće kasnije odloži
- ❖ Implementacija je nemoguća, jer u trenutku greške stranica OS ne zna kada će koja stranica u memoriji biti referencirana
- ❖ Koristi se za evaluaciju algoritama zamene stranica; implementira se na osnovu prvog izvršenja programa i registrovanja referenci na stranice tokom izvršavanja

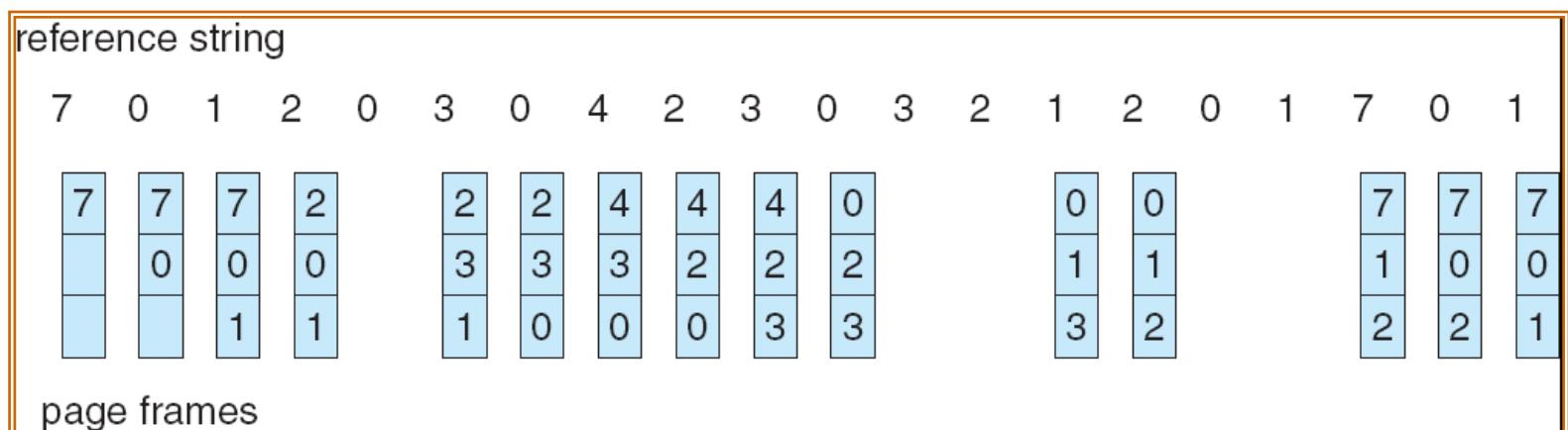
Optimalni algoritam - primer

- ➊ Evaluacija algoritma zamene stranica se izvodi njegovim izvršavanjem i primenom na određenoj nizu memorijskih referenci (*reference string*) i određivanjem broja grešaka stranica na tom nizu
 - ▢ Niz referenci: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
 - ▢ Memorija sadrži tri stranična okvira
- ➋ Generiše 9 grešaka stranica



FIFO algoritam

- ❖ Sistem za upravljanje memorijom održava listu svih stranica trenutno u memoriji, pri čemu je prva stranica u listi najranije učitana u memoriju ("najstarija"), a poslednja najskorije učitana ("najmlađa")
- ❖ Prilikom greške stranica, algoritam uklanja prvu stranicu u listi i zamenjuje je novom stranicom koju smešta u memoriju i dodaje na kraj liste
- ❖ FIFO algoritam se retko koristi u osnovnom obliku
- ❖ Primer
 - ❖ Algoritam generiše 15 grešaka stranica





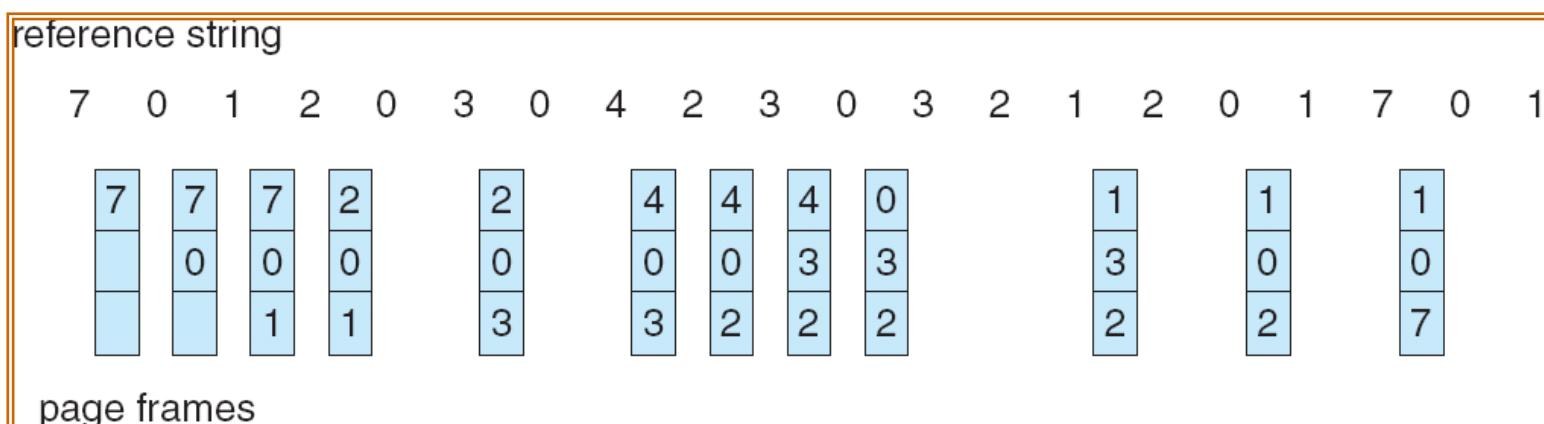
Algoritam zamene najmanje skoro korišćene stranice (LRU)

- Zasniva se na pretpostavci da stranice koje su poslednjih nekoliko instrukcija bile referencirane, biće referencirane i u narednim instrukcijama
- Prilikom greške stranica, zamenjuje se stranica koja najduže vreme nije referencirana.
- Težak za implementaciju - Uz svaku stranicu u memoriji registruje se vreme poslednje reference te stranice što bi uzrokovalo velike režijske troškove za evidentiranje i pretraživanje vremena referenciranja



LRU algoritam - primer

- Za test niz referenci generiše 12 grešaka stranica

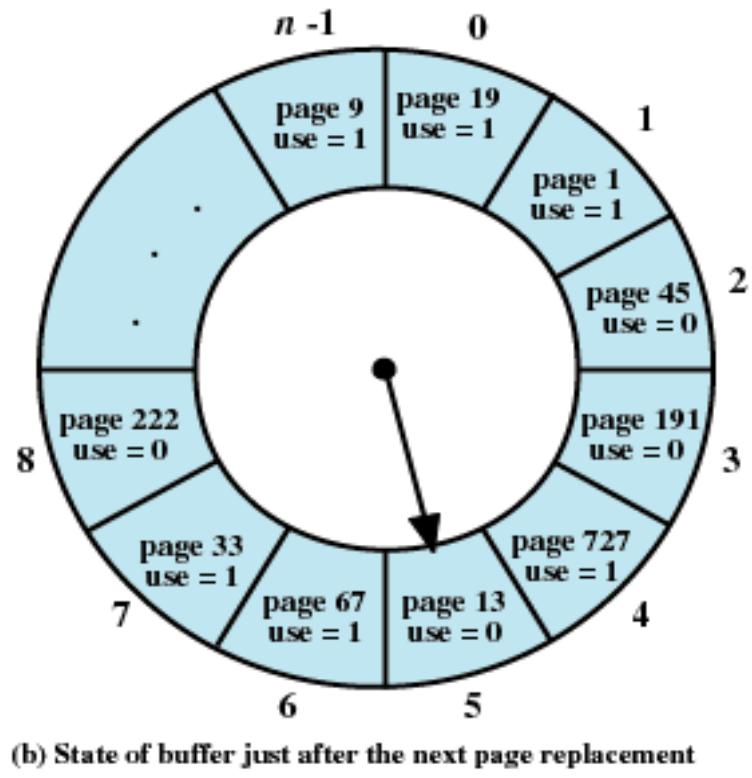
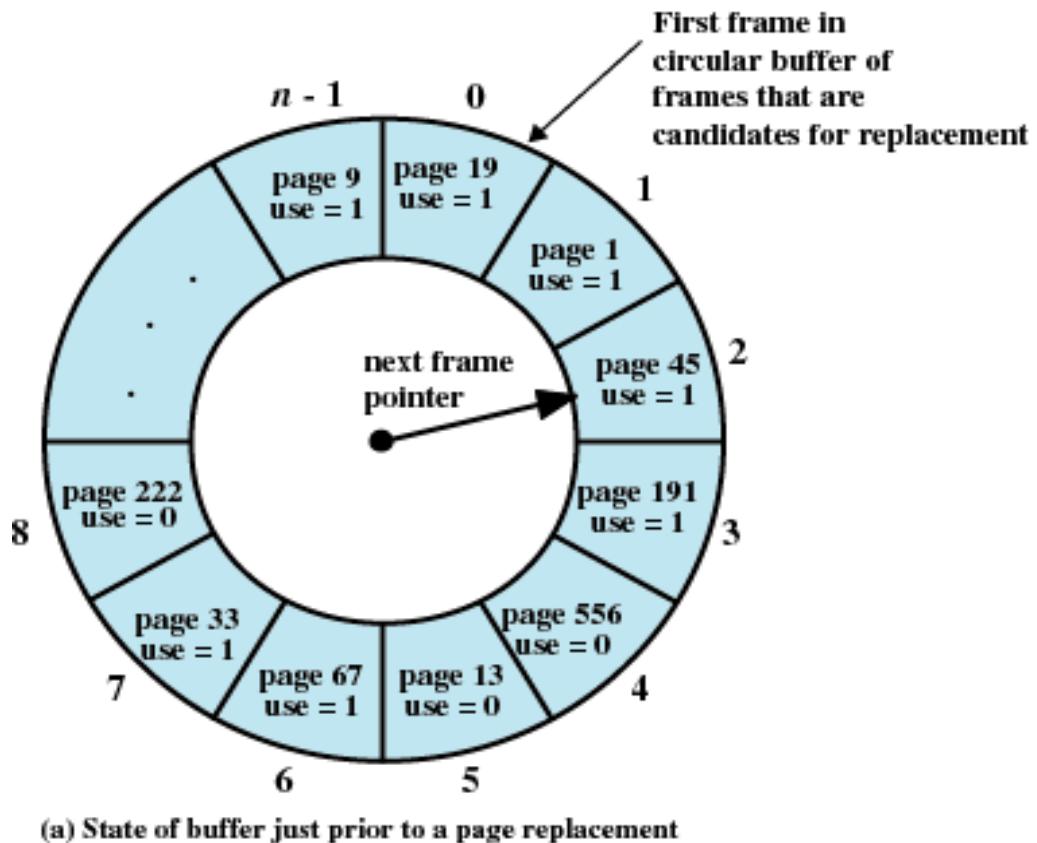


Algoritam časovnika (*Clock*)

- ❖ Jednostavna modifikacija FIFO algoritma koja pored "starosti" stranice uzima u obzir i vrednost bita upotrebe (*use*) – **u** bita, koji se postavlja se na 1 kada se stranica učita u memoriju i kad god je referencirana
- ❖ Algoritam časovnika posmatra skup svih okvira koji su kandidati za zamenu (ako se radi o okvirima koje zauzima taj proces – **lokalni opseg**, a svih okvira u glavnoj memoriji – **globalni opseg**) kao kružni bafer kome je pridružen pokazivač
 - Ako je $u = 0$, stranica je i stara i ne skoro referencirana pa se zamenjuje novom stranicom
 - Ako je $u = 1$, bit se postavlja na 0, pokazivač prelazi na sledeći okvir u kružnom baferu i ispitivanje stranica se nastavlja
- ❖ Ukoliko su **u** bitovi svih stranica postavljeni na 1, algoritam postaje FIFO

Algoritam časovnika

- Kružni bafer od n okvira, zahteva se učitavanje stranice 727

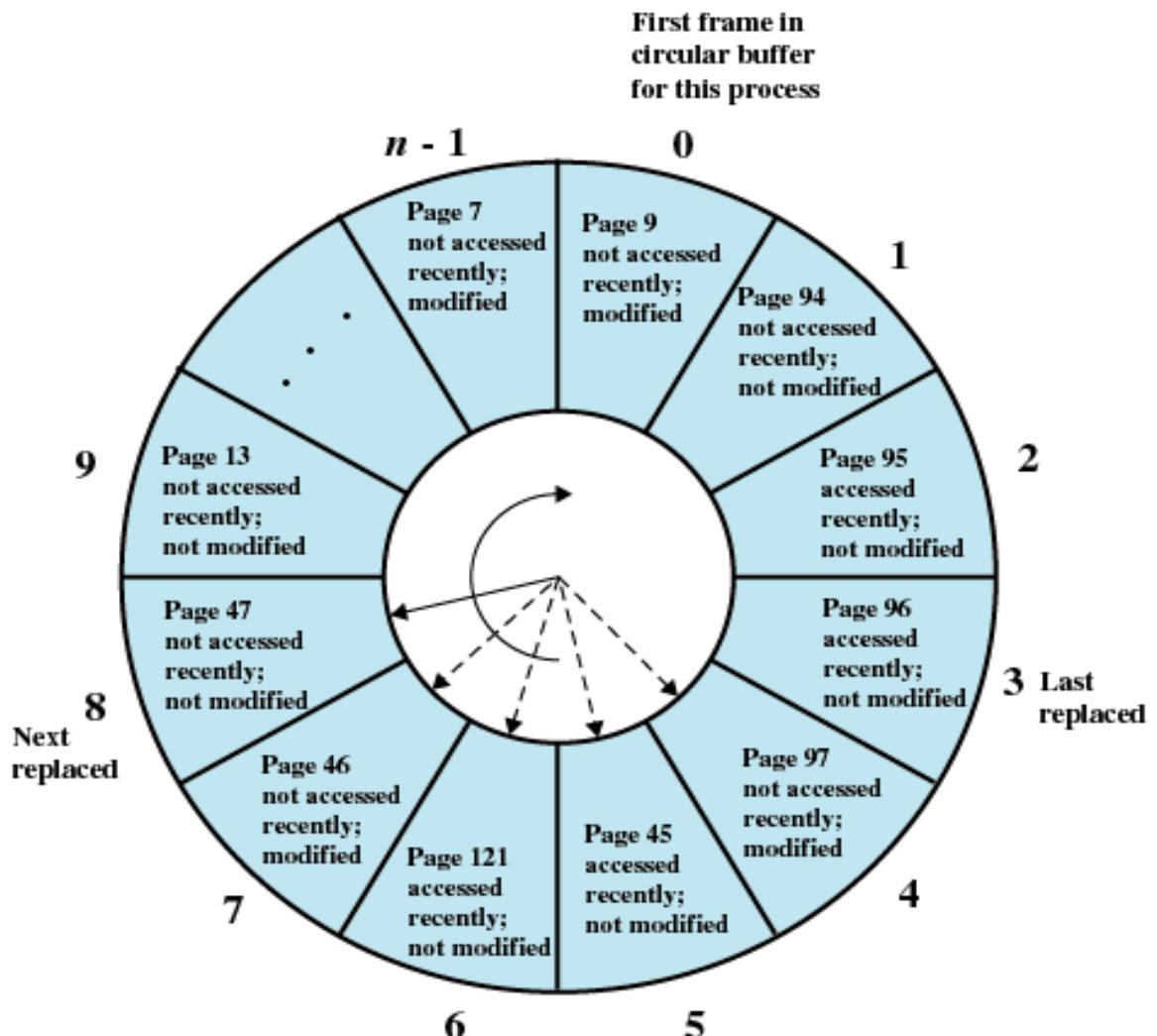


Modifikacija algoritma časovnika

- ⊕ Koristi se još jedan bit pridružen svakoj stranici
 - ▣ Bit promene (m bit) – postavlja se na 1 kada je stranica modifikovana
- ⊕ Ovaj bit je potreban da kada se sadržaj stranice promeni ona ne bude zamenjena dok se ne upiše na sekundarnu memoriju, a pošto je to vremenski “skupa” operacija, onda je bolje zameniti stranicu koja nije modifikovana
- ⊕ Četiri klase stranica
 - ▣ Stranica nije referencirana niti modifikovana - ($u=0, m=0$)
 - ▣ Stranica nije referencirana, ali je modifikovana - ($u=0, m=1$)
 - ▣ Stranica je referencirana, ali nije modifikovana - ($u=1, m=0$)
 - ▣ Stranica je referencirana i modifikovana - ($u=1, m=1$)
- ⊕ Algoritam
 1. Algoritam obilazi bafer okvira počev od pozicije pokazivača, pri čemu ne menja bit upotrebe stranicama kod kojih je $u=1$. Bira za zamenu prvi okvir sa vrednostima $u=0$ i $m=0$ za stranicu u njemu
 2. Ako ne uspe u prvom prolazu, ponovo obilazi bafer okvira i traži okvir sa vrednostima $u=0$ i $m=1$ i bira za zamenu prvi takav okvir na koji najde. U toku skeniranja postavlja bit upotrebe na 0 u svakom okviru koji obide
 3. Ako ne uspe korak 2, pokazivač je vraćen na prvobitni položaj i svi okviru u skupu će imati bit upotrebe 0, pa ponavlja korak 1

Modifikacija algoritma časovnika (2)

Primer



Algoritmi zamene stranica – primer 2

Page address
stream

2 3 2 1 5 2 4 5 3 2 5 2

OPT

2	2	2	2	2	2	4	4	4	2	2	2
	3	3	3	3	3	3	3	3	3	3	3
			1	5	5	5	5	5	5	5	5

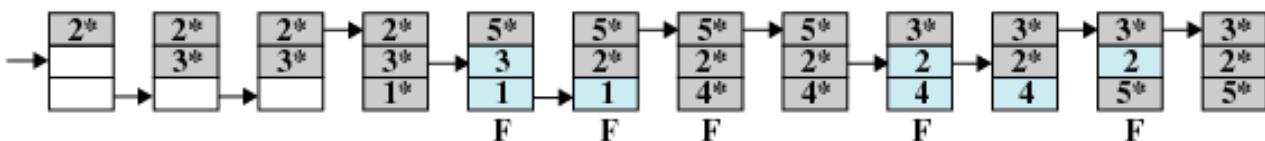
LRU

2	2	2	2	2	2	2	2	3	3	3	3
	3	3	3	3	3	3	3	5	5	5	5
			1	1	1	4	4	4	2	2	2

FIFO

2	2	2	2	5	5	5	5	3	3	3	3
	3	3	3	3	3	2	2	2	5	5	5
			1	1	1	4	4	4	4	4	2

CLOCK



F = page fault occurring after the frame allocation is initially filled

Baferovanje stranica

- ❖ Stranica koja je izabrana za zamenu se ne izbacuje iz memorije (briše) već se stavlja u jednu od dve liste:
 - ❖ Listu slobodnih stranica ukoliko stranica nije menjana
 - ❖ Listu modifikovanih stranica ukoliko jeste
- ❖ Sadržaj stranice ostaje u memoriji, a uklanja se samo stavka za tu stranicu u tabeli stranica (ili postavlja bit prisutnosti P na 0)
 - ❖ Grupišu se modifikovane stranice, pa se odjednom upisuju na disk čime se redukuje broj U/I operacija i poboljšavaju performanse
- ❖ Operativni sistem u svakom trenutku održava određeni broj slobodnih okvira – kada nova stranica treba da se učita (zbog greške stranice) bira se prvi okvir/stranica iz liste slobodnih stranica i novom stranicom prebriše prethodni sadržaj
- ❖ Stranica koja treba da se zameni ostaje u memoriji, pa ako je proces opet referencira ona se vraća u rezidentni skup tog procesa



Upravljanje rezidentnim skupom

- **Veličina rezidentnog skupa** – koliko stranica procesa učitati u memoriju
 - Fiksno dodeljivanje
 - Dodeljuje procesu fiksan broj okvira u glavnoj memoriji unutar kojih se izvršava
 - Promenljivo dodeljivanje
 - Dozvoljava da se broj okvira dodeljenih procesu, a samim tim i njegov rezidentni skup menja tokom vremena
- **Opseg zamene** – čiju stranicu zameniti kada dođe do greške stranice
 - Lokalna zamena
 - Bira se stranica za zamenu samo između stranica procesa koji je napravio grešku stranice
 - Globalna zamena
 - Razmatra sve nezaključane stranice u memoriji kao kandidate za zamenu



Upravljanje rezidentnim skupom (2)

- Moguće strategije upravljanja rezidentnim skupom
 - Fiksno dodeljivanje, lokalni opseg
 - Promenljivo dodeljivanje, lokalni opseg
 - Promenljivo dodeljivanje, globalni opseg
 - Implementirano od strane mnogih operativnih sistema

Strategija radnog skupa (*Working Set*)

- ❖ Tokom određene faze u izvršenju, proces referencira relativno mali deo svojih stranica – lokalnost referenci
- ❖ Radni skup - skup stranica koje proces trenutno koristi
- ❖ **Radni skup $W(t, \Delta)$** u virtuelnom trenutku t čine stranice referencirane u poslednjih Δ jedinica virtuelnog vremena (memorijskih referenci)
 - Virtuelno vreme se meri u memorijskim referencama
 - Δ – okvir (prozor) virtuelnog vremena u kome se proces posmatra
- ❖ Strategija
 - Nadgledati radni skup svakog procesa
 - Periodično uklanjati iz rezidentnog skupa procesa one stranice koje nisu u radnom skupu (koristi se LRU)
 - Ukoliko je celokupni radni skup u memoriji, proces se izvršava bez greški stranica, sve dok ne pređe u sledeću fazu izvršenja
 - Prilikom greške stranica treba naći stranicu koja nije u radnom skupu procesa i zameniti je novom stranicom u memoriji

Radni skup procesa

- Radni skup procesa u zavisnosti od veličine "prozora"

Sequence of
Page
References

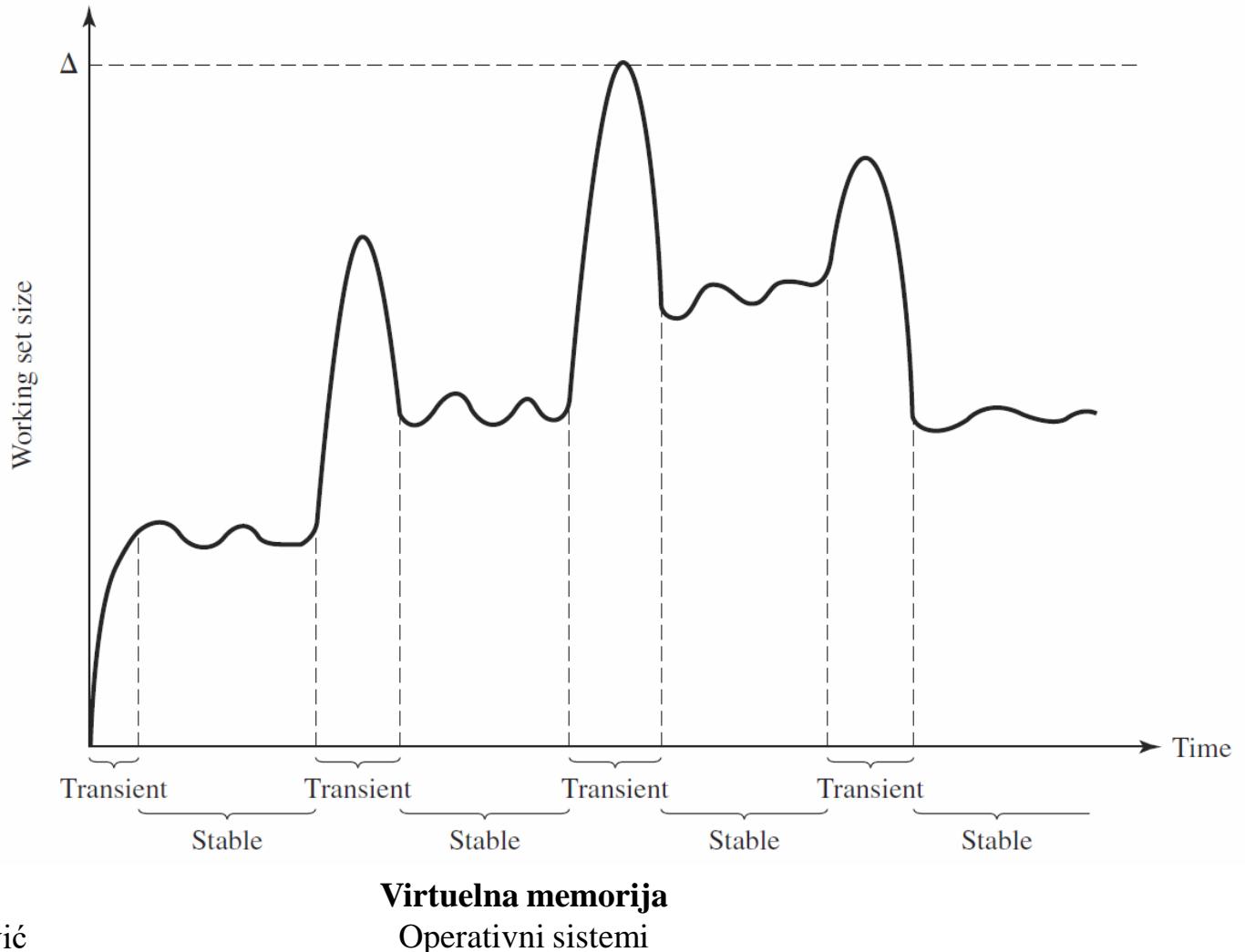
24
15
18
23
24
17
18
24
18
17
17
15
24
17
24
18

Window Size, Δ

2	3	4	5
24	24	24	24
24 15	24 15	24 15	24 15
15 18	24 15 18	24 15 18	24 15 18
18 23	15 18 23	24 15 18 23	24 15 18 23
23 24	18 23 24	•	•
24 17	23 24 17	18 23 24 17	15 18 23 24 17
17 18	24 17 18	•	18 23 24 17
18 24	•	24 17 18	•
•	18 24	•	24 17 18
18 17	24 18 17	•	•
17	18 17	•	•
17 15	17 15	18 17 15	24 18 17 15
15 24	17 15 24	17 15 24	•
24 17	•	•	17 15 24
•	24 17	•	•
24 18	17 24 18	17 24 18	15 17 24 18

Veličina radnog skupa

- Graf promene veličine radnog skupa tokom izvršenja procesa



Politika čišćenja

- ❖ Određuje kada promenjena stranica treba da bude upisana na sekundarnu memoriju
- ❖ Dve tehnike
 - ❖ Čišćenje na zahtev – stranica se upisuje na sekundarnu memoriju samo kada se izabere za zamenu
 - ❖ Predčišćenje – promenjene stranice se upisuju pre nego što su njihovi stranični okviri potrebni
- ❖ Najbolje je koristiti **baferovanje stranica** – zamenjene stranice se smeštaju na dve liste: one koje su promenjene i one koje nisu. Stranice na listi promenjenih mogu periodično da se upisuju na sekundarnu memoriju u paketima i pomeraju na listu nepromenjenih. Stranica na listi nepromenjenih se ili ponovo uspostavlja kada se referencira, ili se gubi kada se njen okvir dodeli novoj stranici



Upravljanje učitavanjem

- ❖ Određuje broj procesa koji će biti rezidentni u glavnoj memoriji (čije su stranice smeštene u glavnu memoriju) – nivo multiprogramiranja
- ❖ Ako je suviše malo procesa rezidentno u bilo kom trenutku, često će biti situacija da su svi blokirani i CPU nezaposlen, i trošiće se mnogo vremena na razmenu sadržaja između glavne i sekundarne memorije (*swapping*)
- ❖ Ako je suviše procesa rezidentno u memoriji, njihove stranice koje su učitane u memoriju uglavnom neće predstavljati radni skup, tako da će dolaziti do velike učestanosti grešaka stranica (*trashing*)
- ❖ Ako nivo multiprogramiranja treba da se smanji, jedan ili više procesa treba da se *suspenduje* (njegove stranice swap-uju na sekundarnu memoriju)



Upravljanje memorijom u realnim operativnim sistemima



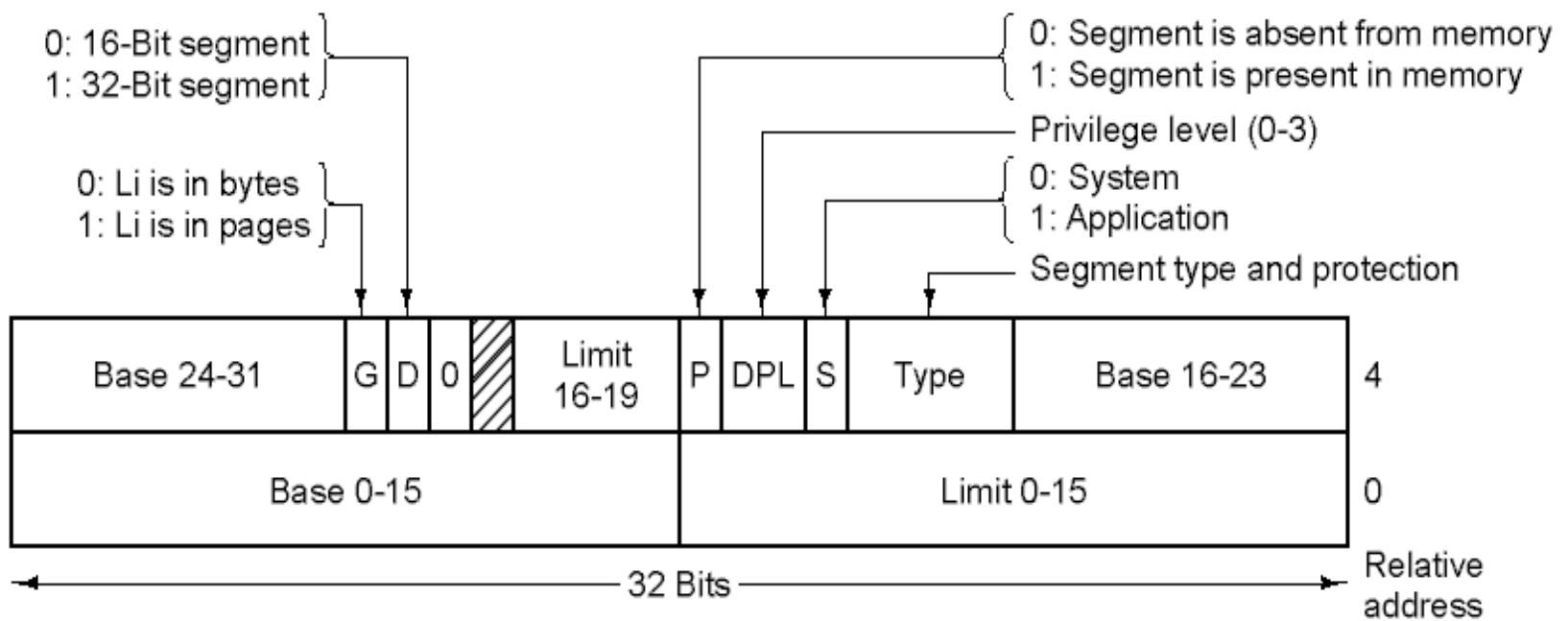
Intel Pentium

- ❖ Segmentacija sa straničenjem (stranice veličine 4KB)
- ❖ Pentium obezbeđuje programima virtuelni adresni prostor od 16K nezavisnih segmenata, svaki sa 1G 32-bitnih reči
- ❖ Osnovu virtuelne memorije Pentiuma čine dve tabele
 - LDT (Local Descriptor Table) – svaki program ima sopstvenu tabelu i ona opisuje segmente programa: kod, podatke, magacin, itd.
 - GDT (Global Descriptor Table) – jedinstvena tabela koju dele svi programi – opisuje sistemske segmente uključujući i OS
- ❖ Da bi se pristupilo segmentu, selektor segmenta mora biti smešten u jedan od 6 segmentnih registara procesora
- ❖ Svaki selektor je 16 bitova
 - Indeks u LDT/GDT (13 bitova) – svaka tabela sadrži maksimum 8K deskriptora segmenata
 - Prava pristupa i zaštita (2 bita)
 - Da li se radi o lokalnom (LDT) ili globalnom (GDT) segmentu (1 bit)

Pentium – Deskriptor segmenta

- Deskriptor segmenta je veličine 8 B:

- Bazna adresa segmenta (Base) (32 b)
- Veličina segmenta (Limit) (20 b) u bajtovima ili stranicama (1 b)
- Tip i zaštita segmenta, sistemski/korisnički segment (1 b), nivo privilegija (2 b), segment u memoriji (1 b), itd.



Stavka tabele stranica – Intel procesori

- Stavka tabele stranica - **Page Table Entry (PTE)** –
 - MS Windows na Intel x86 i AMD x64 arhitekturama

63	62	52	51		12	11	9	8	7	6	5	4	3	2	1	0
N X	AVL	Physical page number			AVL	G	P A T	D	A	P C D	P W T	U /S	R /W	P		

NX No eXecute

AVL AVaiLable to the OS

G Global page

PAT Page Attribute Table

D Dirty (modified)

A Accessed (referenced)

PCD Page Cache Disable

PWT Page Write-Through

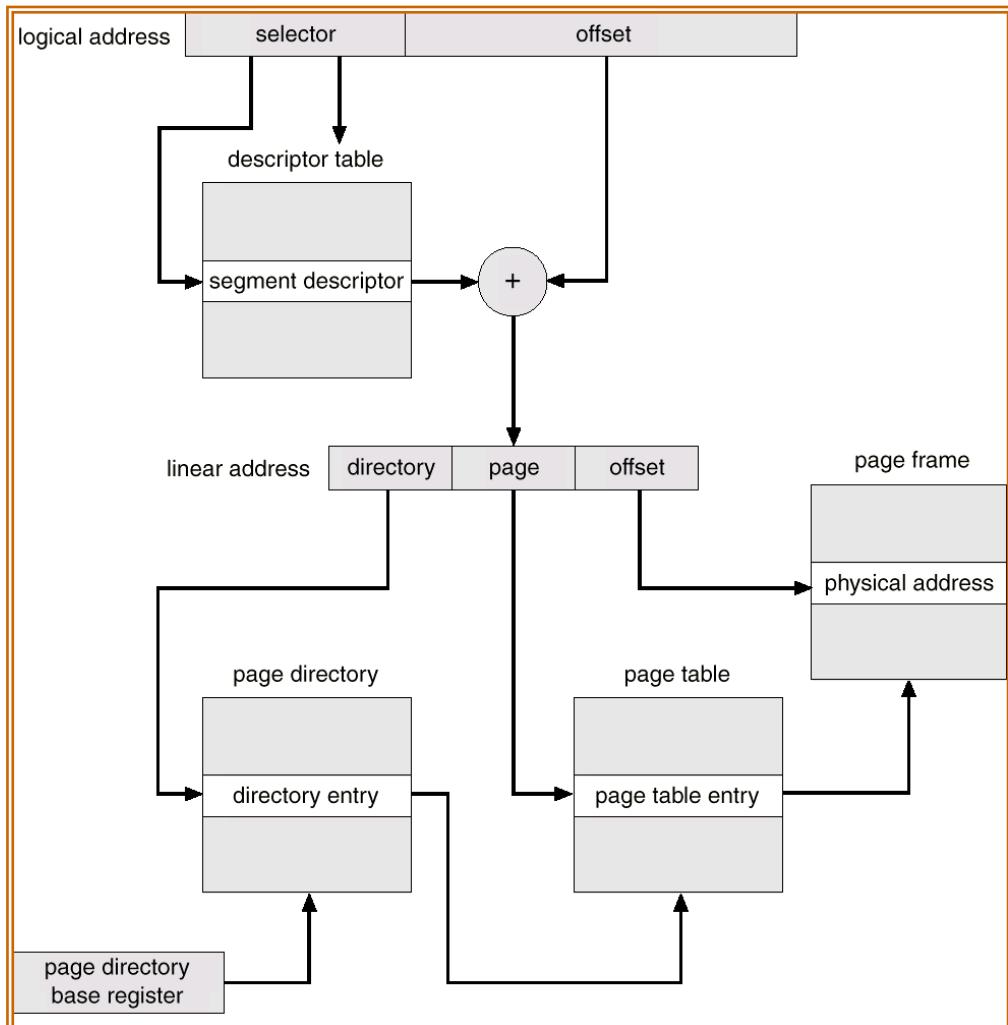
U/S User/Supervisor

R/W Read/Write access

P Present (valid)

Pentium – Transformisanje adresa (1)

- Virtuelna (logička) adresa
 - Selektor segmenta
 - Ofset u okviru segmenta
- Na osnovu vrednosti selektora pristupa se tabeli deskriptora segmenta
- Na 32-bitnu baznu adresu segmenta dodaje se offset i formira linearna adresa
- Linearna adresa (32b) se deli
 - Indeks u direktorijum stranica (10 bitova)
 - Indeks u tabelu stranica (10b)
 - Ofset u okviru stranice (12b)

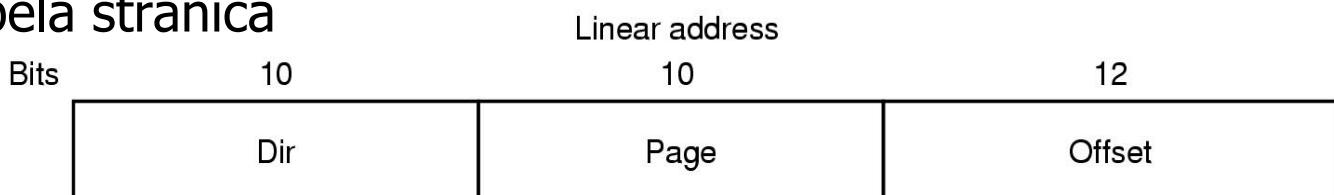


Virtuelna memorija
Operativni sistemi

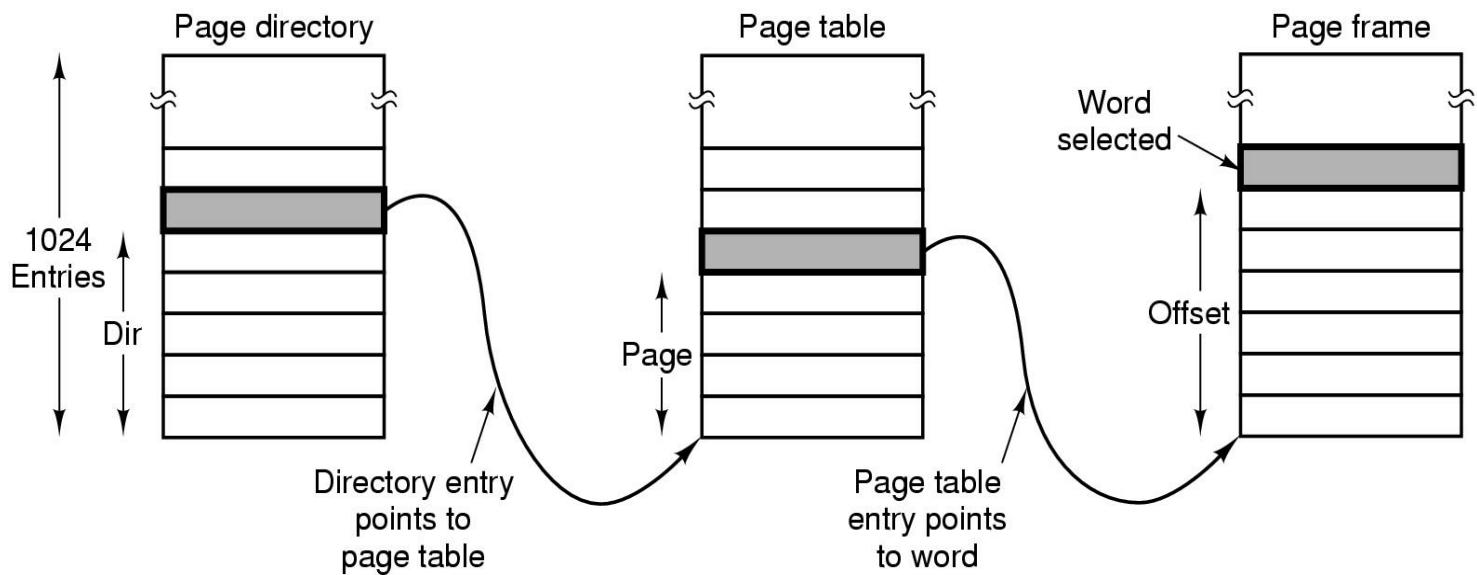
Pentium – Transformisanje adresa (2)

● Tabela stranica u dva nivoa

- Direktorijum stranica
- Tabela stranica



(a)





Pentium - Upravlja memorijom

- Koristi TLB za smeštanje podataka o naskorije korišćenim stranicama u cilju ubrzanja transformacije virtuelne u fizičku adresu (Promašajima se upravlja po algoritmu LRU)
- Pentium MMU arhitektura omogućava da OS implementira
 - Čisto straničenje (Koristi se jedinstveni selektor segmenta, čiji deskriptor ima baznu adresu 0, a veličine segmenta je postavljena na maksimum – 4GB)
 - Čistu segmentaciju
 - Segmentaciju sa straničenjem



Unix & Solaris upravljanje memorijom

◆ Straničenje – koriste se sledeće strukture podataka

- Tabela stranica (*Page Table*): Jedna tabela stranica za svaki proces, sa jednom stavkom (ulazom) za svaku stranicu.
- Deskriptor bloka diska (*Disk Block Descriptor*): Za svaku stranicu postoji stavka koji opisuje disk kopiju te stranice.
- Tabela podataka okvira stranica (*Page Frame Data Table*): Opisuje svaki stranični okvir glavne memorije, a indeks joj je broj okvira. Koristi se kod algoritama zamene stranica.
- Tabela swap-ovanja (*Swap Use Table*): Postoji jedna tabela swap-ovanja za svaki uređaj za razmenu (swap), sa jednom stavkom u tabelu za svaku stranicu na uređaju



Unix & Solaris upravljanje memorijom

Page Frame Number	Age	Copy on write	Modify	Referenced	Valid	Protect
-------------------	-----	---------------	--------	------------	-------	---------

Page Table Entry

Swap Device Number	Device Block Number	Type of Storage
--------------------	---------------------	-----------------

Disk Block Descriptor

Page State	Reference Count	Logical Device	Block Number	Pf Data Pointer
------------	-----------------	----------------	--------------	-----------------

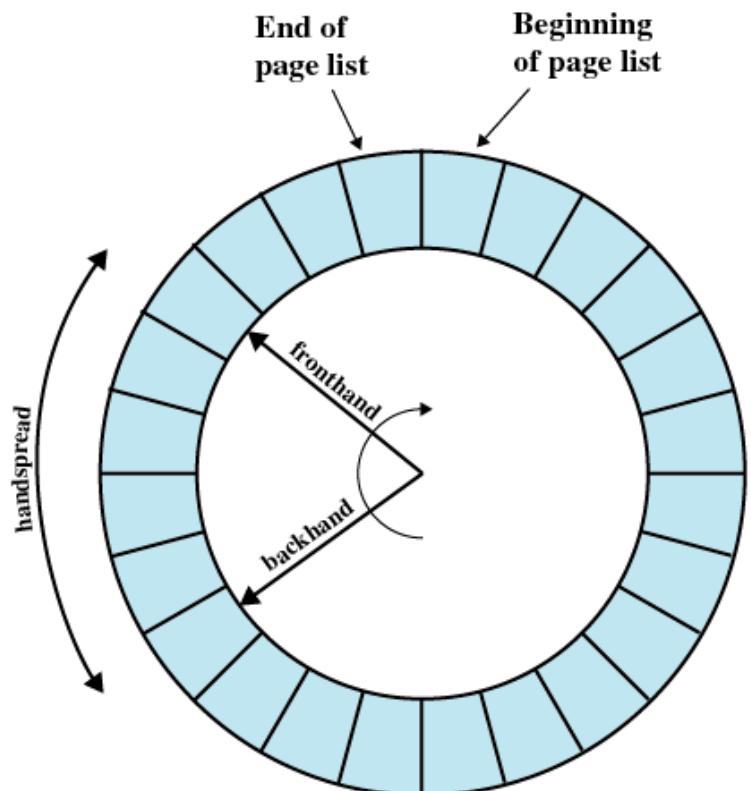
Page Frame Data Table Entry

Reference Count	Page/Storage Unit Number
-----------------	--------------------------

Swap Use Table Entry

Unix & Solaris zamena stranica

- ❖ Tabela podataka okvira stranica koristi se za zamenu
- ❖ Svi raspoloživi okviri se povezuju zajedno u listu slobodnih okvira
- ❖ Kada broj raspoloživih okvira padne ispod izvesnog praga, oslobađaju se dodatni okviri
- ❖ Algoritam **časovnika sa dve kazaljke**
 - ❖ Prednja kazaljka (pokazivač) prolazi preko liste slobodnih stranica i postavlja bit referenciranja (upotrebe) na 0
 - ❖ Nešto kasnije druga kazaljka prelazi preko iste liste i proverava bit reference, ako je jednak 1, stranica je skoro referencirana i nije kandidat za zamenu, a ako je jednak 0 ta stranica se smešta na listu slobodnih stranica/okvira

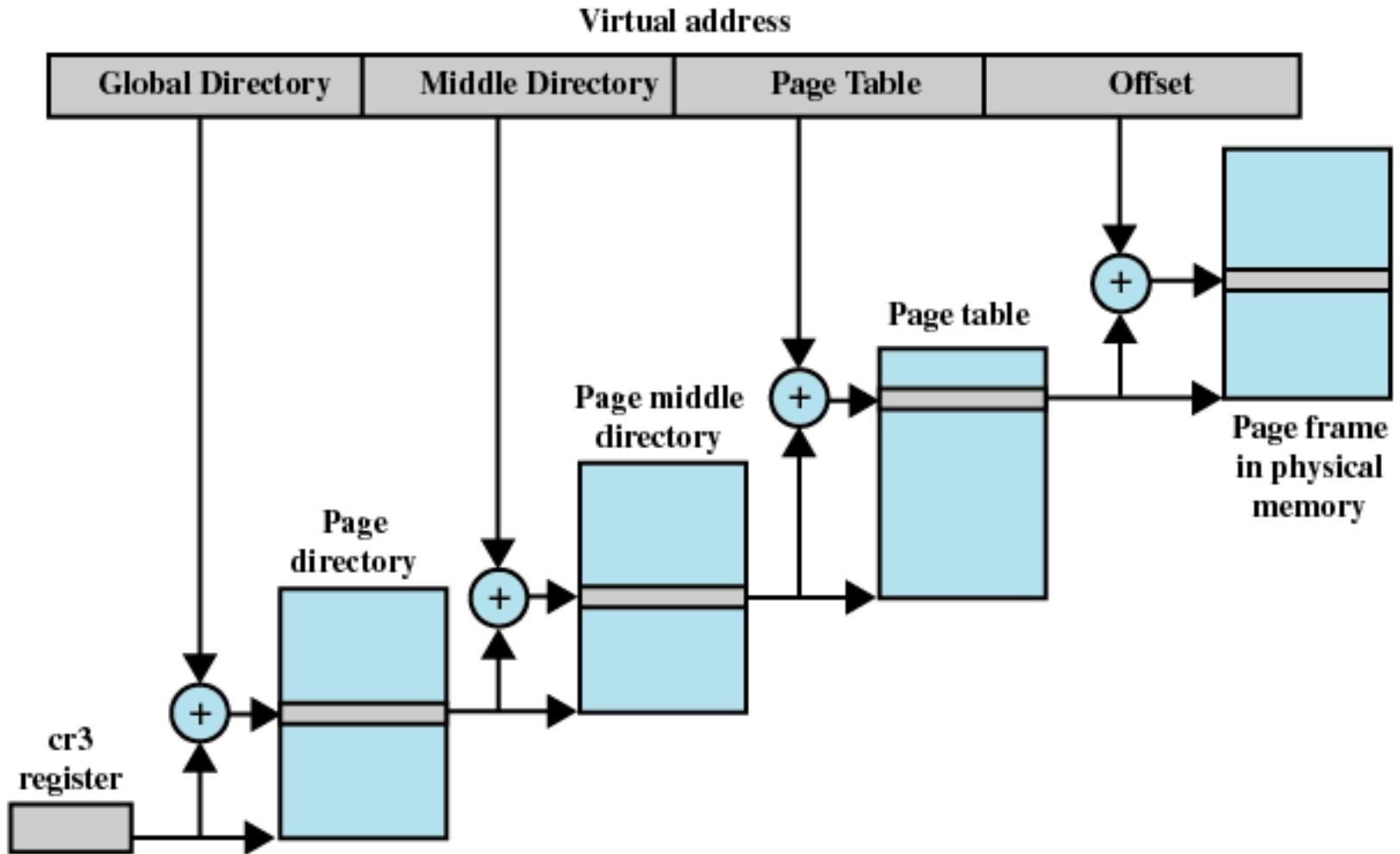




Linux upravljanje memorijom

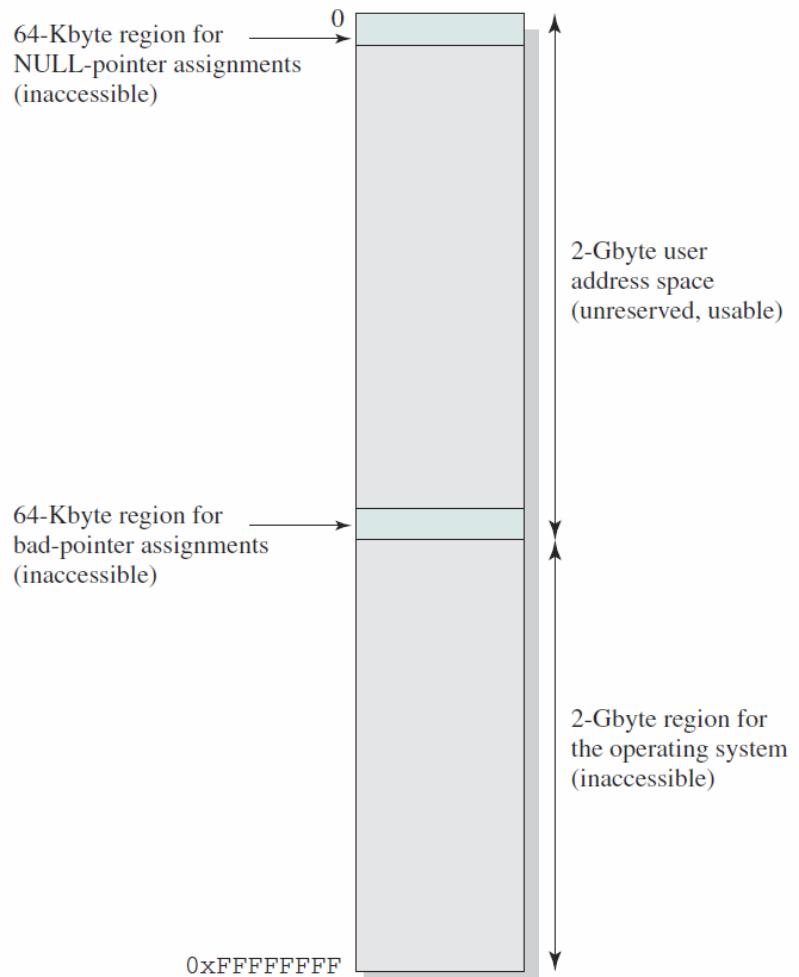
- Straničenje sa tabelama stranica u tri nivoa:
 - direktorijum stranica, srednji direktorijum stranica i tabela stranica
- Dodeljivanje stranica – održava blokove susednih stranica koji su smešteni u susedne stranične okvire (1, 2, 4, 8, 16 ili 32 okvira) – koristi se partnerski sistem (*Buddy system*)
- Algoritam za zamenu po principu časovnika
 - Bit upotrebe je zamenjen 8-bitnim poljem za starost: svaki put kad se pristupi stranici polje za starost se uvećava
 - Linux periodično prolazi kroz globalni skup stranica i smanjuje polje za starost
 - Što je veća vrednost polja za starost stranica je češće referencirana u bliskoj prošlosti; stranica sa starošću “0” je kandidat za zamenu

Linux upravljanje memorijom



Windows upravljanje memorijom

- ❖ Procesi rade sa 32 bitnim adresnim prostorom.
- ❖ Svaki proces po kreiranju na raspolaganju ima 4GB (2^{32}) virtuelnog adresnog prostora.
- ❖ Niža 2GB su na raspolaganju procesu za njegov kod i podatke dok se viša 2GB mapiraju za potrebe jezgra.
- ❖ Niža 2GB su privatna za proces i njima ne može pristupiti nijedan drugi proces. Viša 2GB (osim jednog malog dela) su zajednička za sve procese, ali njima ne može direktno da pristupi nijedan proces.



Windows upravljanje memorijom (2)

- ❖ Virtual Memory Manager (VMM) – modul koji održava memory map tabelu, upravlja straničenjem i vrši prevodenje virtuelnih u fizičke adrese.
- ❖ Stranice su veličine 4KB (Pentium procesori) odnosno 8 ili 16KB (Itanium procesori), a moguća je veličina do 64KB. Ekskluzivno kernel ima pravo da koristi i veće stranice (4MB) kako bi smanjio veličinu memory map tabele. Stranice se učitavaju tek kada se javi zahtev za njima (*demand paging*)
- ❖ Koristi se straničenje na zahtev u kombinaciji sa tehnikom *clustering*. Kada proces referencira neku stranicu, postoji velika verovatnoća (zbog sekvensijalne prirode programa i podataka) da će u bliskoj budućnosti referencirati susednu stranicu. Kada se desi *page fault* VMM radi učitavanje (*load*) referencirane stranice i nekoliko susednih stranica.
- ❖ Za zamenu stranica koristi se algoritam radnog skupa (working set), dok se radnim skupom upravlja tehnikom ***promenljivo dodeljivanje, lokalni opseg***.



Domaći zadatak

- ❖ Poglavlje **8 Virtuelna memorija**
 - ❖ 8.8 Ključni pojmovi, kontrolna pitanja i problemi
- ❖ Paging & segmentation animations
 - ❖ <https://apps.uttyler.edu/Rainwater/COSC3355/Animations>



Operativni sistemi

- Upravljanje U/I i planiranje
diska -

Prof. dr Dragan Stojanović

Katedra za računarstvo
Univerzitet u Nišu, Elektronski fakultet



Literatura

- ➊ *Operating Systems: Internals and Design Principles*, edition, W. Stallings, Pearson Education Inc., 7th – 2012, (5th -2005, 6th - 2008, 8th – 2014 , 9th – 2017)
 - <http://williamstallings.com/OperatingSystems/>
 - <http://williamstallings.com/OperatingSystems/OS9e-Student/>

- ➋ **Poglavlje 11:** Upravljanje U/I i planiranje diska



Upravljanje U/I

- ❖ Upravljanje U/I predstavlja kompleksan i “najzamršeniji” deo dizajna i implementacije operativnog sistema
- ❖ Veoma je teško realizovati generalno, konzistentno rešenje za upravljanje U/I pošto postoji velika raznolikost U/I uređaja, kao i primena tih uređaja



Kategorije U/I uređaja

● Čitljivi za čoveka

- koriste se za komunikaciju sa korisnikom
- štampači
- video terminali
 - ekran
 - tastatura
 - miš

● Čitljivi za mašinu

- koriste se za komunikaciju sa elektronskom/računarskom opremom
- jedinice diska i trake
- senzori
- kontroleri
- aktuatori

● Komunikacioni

- koriste se za komunikaciju sa udaljenim uređajima
- mrežni adapteri
- modemi

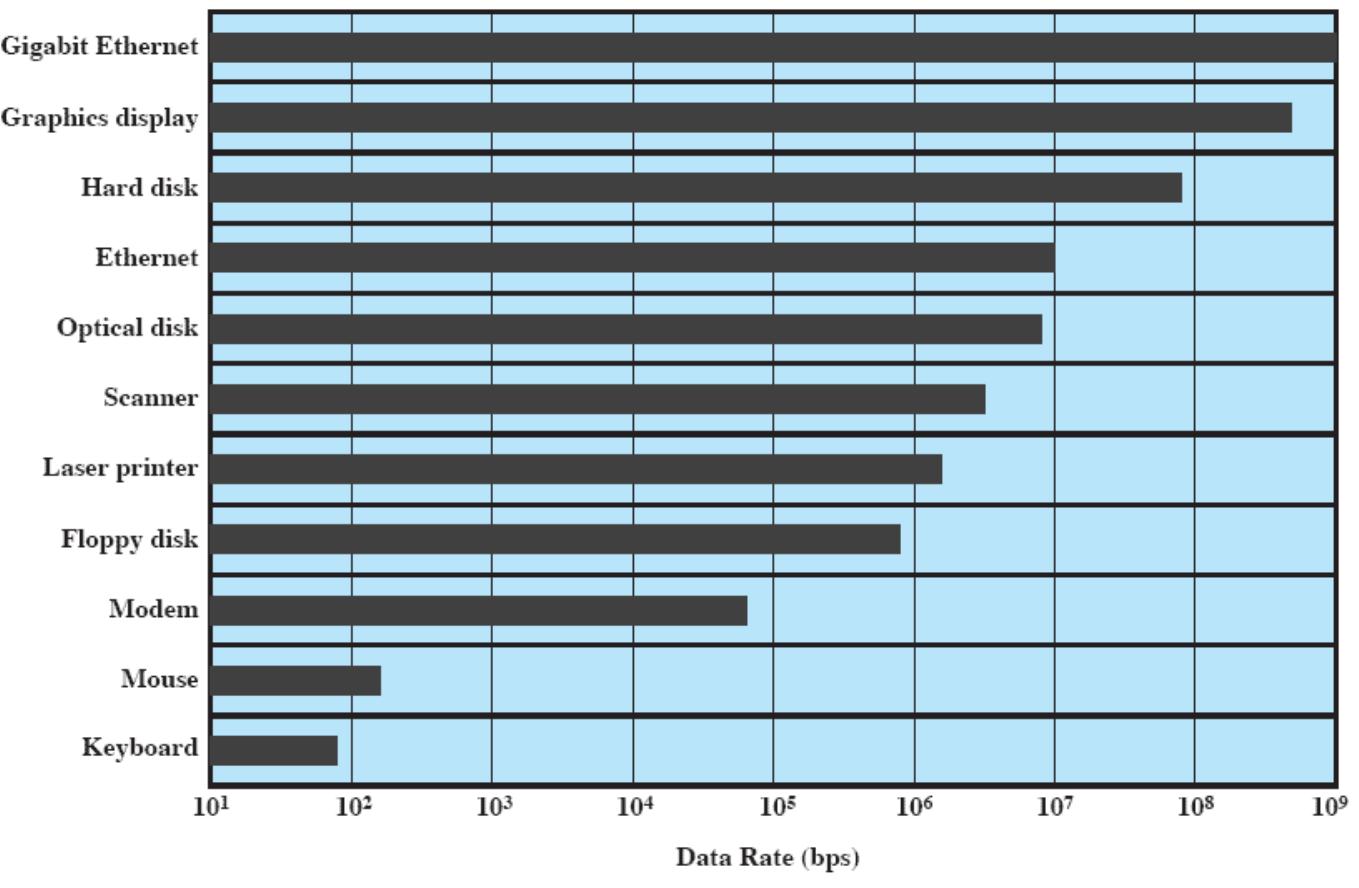


Razlike među U/I uređajima

- ❖ Postoje velike razlike između klasa uređaja, čak i značajne razlike u okviru iste klase uređaja
 - ❖ Brzina prenosa podataka
 - ❖ Primena uređaja
 - ❖ Složenost upravljanja
 - ❖ Jedinica prenosa podataka – niz znakova (bajtova) ili blokovi
 - ❖ Reprezentacija podataka
 - ❖ Uslovi greške
- ❖ Ove različitosti otežavaju postizanje uniformnog i konzistentnog pristupa u upravljanju U/I uređajima u okviru operativnog sistema

Brzina prenosa podataka

- Tipična brzina prenosa podataka U/I uređaja



Upravljanje U/I i planiranje diska
Operativni sistemi



Jedinica prenosa podataka

- ◆ **Prema jedinici prenosa podataka** U/I uređaji se mogu podeliti u dve kategorije:
 - ▣ blok uređaji
 - ▣ znakovni uređaji
- ◆ **Blok uređaji:** prenose podatke u blokovima fiksne veličine koji su adresibilni, npr. disk ili DVD
 - ▣ Tipična veličina bloka 512 bajta do 32 KB
 - ▣ Može se čitati i zapisivati svaki blok nezavisno od ostalih
 - ▣ Komande `read`, `write`, `seek`
- ◆ **Znakovni uređaji:** prihvataju i isporučuju tokove znakova (bajtova) koji nisu adresibilni
 - ▣ Npr. printeri, mrežni interfejsi, miš, tastatura, serijski portovi
 - ▣ Komande `get`, `put`
- ◆ Ima uređaja koji se ne mogu svrstati u ove dve kategorije
 - ▣ Na primer *clock* (tajmer)



Tehnike za izvođenje U/I

⊕ Programirani U/I

- Procesor u korist procesa izdaje U/I komandu U/I modulu
- U/I modul izvršava U/I operaciju i u U/I statusni registar postavlja fleg da je operacija završena
- Proces koji je izdao U/I komandu je u radnom čekanju dok se U/I operacija ne završi i za to vreme proverava statusni registar

⊕ U/I vođen prekidima

- Procesor u korist procesa izdaje U/I komandu U/I modulu
- Procesor nastavlja da izvršava naredne instrukcije
- U/I modul izvršava U/I komandu i šalje prekid kada je U/I komanda obrađena

⊕ U/I korišćenjem DMA

- DMA modul upravlja razmenom podataka između glavne memorije i U/I modula
- Procesor šalje zahtev za prenos bloka podataka DMA modulu
- Procesor se prekida tek pošto se čitav blok prenese



Odnos među U/I tehnikama

- U mnogim računarskim sistemima DMA je dominantan oblik prenosa podataka koji mora biti podržan od strane operativnog sistema

	NE KORISTI PREKID	KORISTI PREKID
Prenos podataka između U/I uređaja i memorije ide preko procesora	Programirani U/I	Prekidima vođen U/I
Direktni prenos podataka između U/I uređaja i memorije		Direktni pristup memoriji (DMA)



Evolucija funkcije U/I

1. Procesor direktno upravlja U/I uređajima
 - Ovo se javlja kod jednostavnih uređaja kontrolisanih mikroprocesorima
2. Dodaje se kontroler ili U/I modul
 - Procesor koristi programirani U/I bez prekida. Na ovaj način procesor je odvojen od specifičnih detalja interfejsa periferijskih uređaja
3. Ista konfiguracija kao i pod 2, ali se koriste prekidi.
 - Procesor ne troši vreme čekajući na završetak U/I operacije čime se povećava efikasnost
4. U/I modulu je omogućena direktna kontrola memorije preko DMA
 - Moguće je prebacivati blok podataka iz memorije ili u memoriju bez uključenja procesora, osim na početku i kraju prenosa



Evolucija funkcije U/I (2)

5. U/I modul je proširen tako da predstavlja poseban procesor sa specijalnim skupom instrukcija prilagođenim U/I - **U/I kanal**
 - Centralni procesor (CPU) upućuje U/I procesor da izvrši U/I program koji se nalazi u glavnoj memoriji. U/I procesor pribavlja i izvršava ove instrukcije bez intervencije CPU. Ovim se omogućava da CPU specificira sekvencu U/I aktivnosti i da bude prekinut tek kada se celokupna sekvenca izvrši.
6. U/I modul poseduje sopstvenu lokalnu memoriju i u stvari je samostalni računar - **U/I procesor**
 - Sa ovom arhitekturom omogućava se upravljanje velikog skupa U/I uređaja uz minimalno angažovanje procesora

Direct Memory Access (DMA)

- Procesor prosleđuje U/I operaciju DMA modulu
- DMA modul vrši prenos podata direktno u ili iz memorije
- Kada završi prenos DMA modul šalje signal prekida procesoru

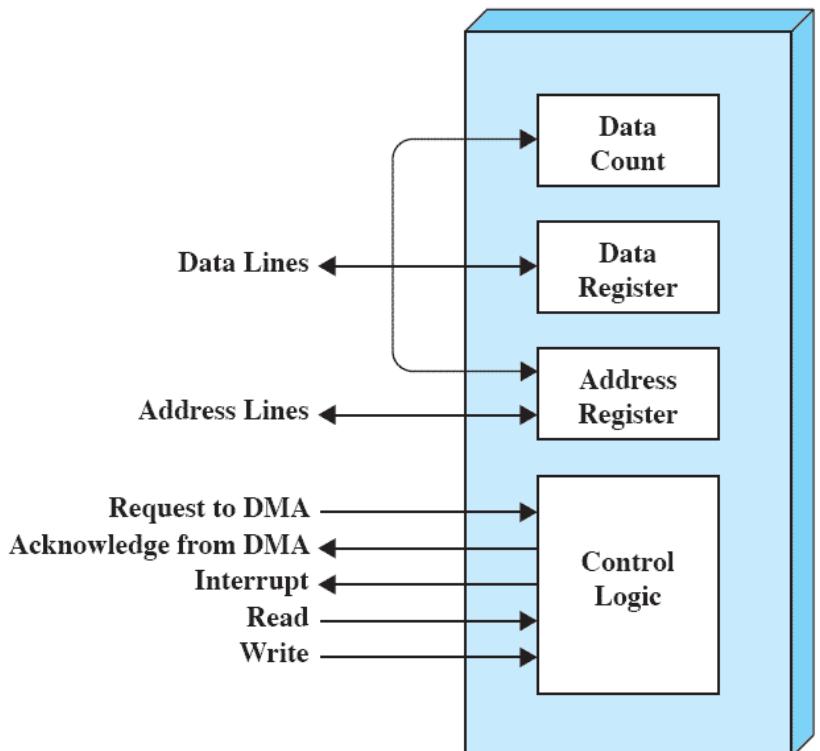
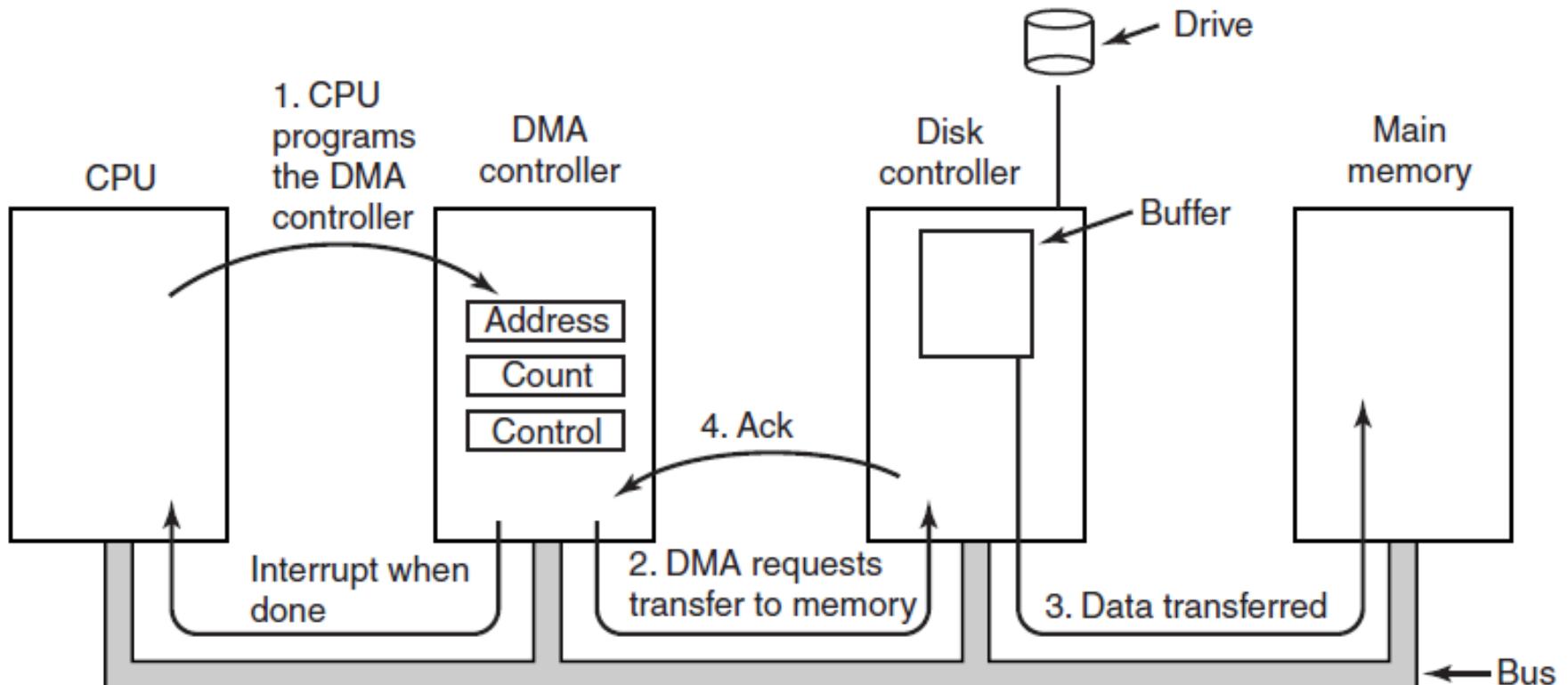


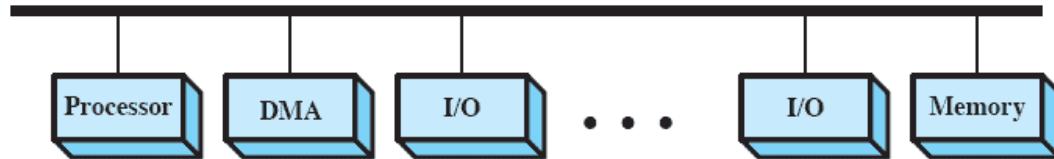
Figure 11.2 Typical DMA Block Diagram

Direct Memory Access (DMA)

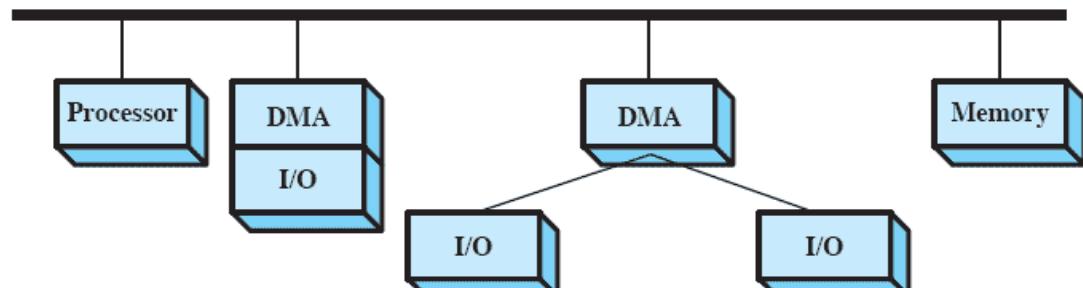


Tanenbaum, 2014

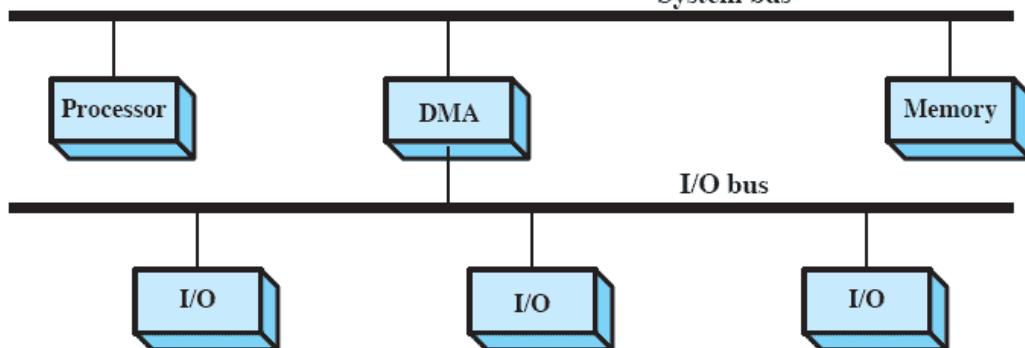
DMA konfiguracije



(a) Single-bus, detached DMA



(b) Single-bus, Integrated DMA-L/O



(c) I/O bus

Upravljanje U/I i planiranje diska
Operativni sistemi



Ciljevi U/I sistema - Efikasnost

- Većina U/I uređaja je daleko sporija od glavne memorije i procesora
- Korišćenje multiprogramiranja omogućava da dok neki procesi čekaju na U/I, drugi procesi se izvršavaju
- Međutim U/I je daleko sporiji od procesora, tako da bi se obezbedila zaposlenost procesora mora se primeniti *swap*-ovanje (što je takođe U/I operacija)
- CPU se sve više oslobađa zadataka u vezi U/I –a i to prenosi na U/I modul (U/I kanal, U/I procesor), što poboljšava performanse,
 - Posebna pažnja je usmerena na U/I diska

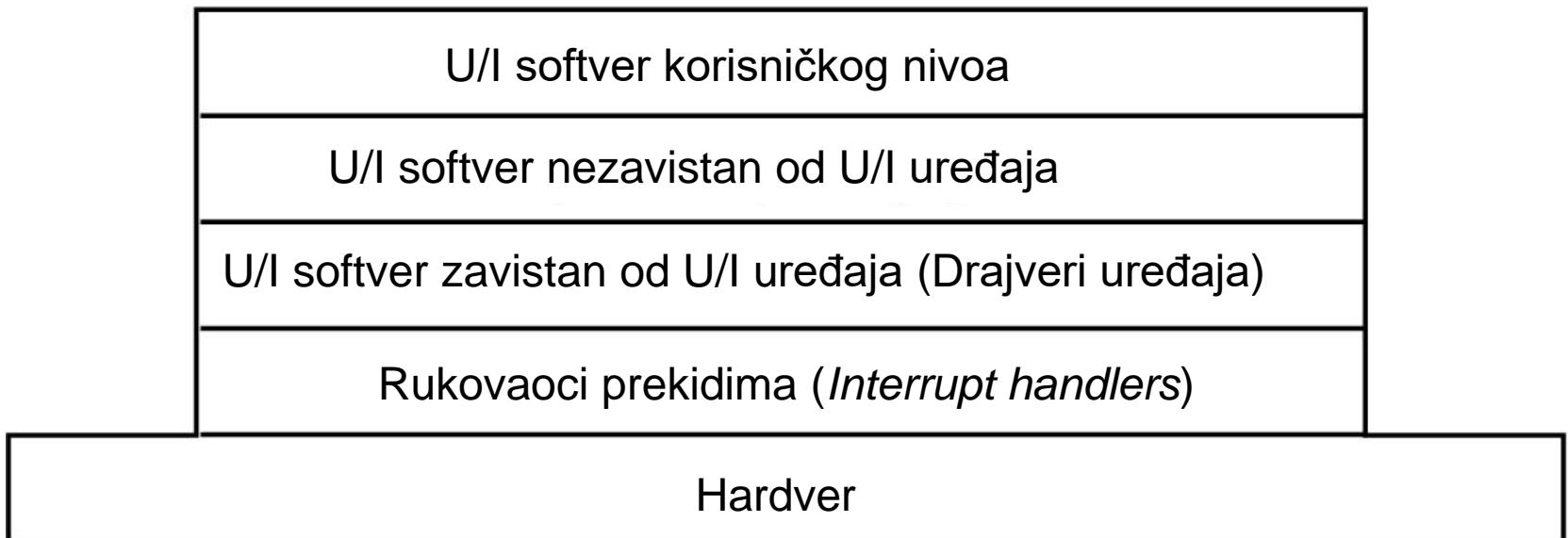


Ciljevi U/I sistema - Generalnost

- Sa U/I uređajima treba raditi na jednoobrazan (uniforman, generalan) način
 - Procesi treba da vide na uniforman način U/I uređaje
 - OS treba na uniforman način da upravlja U/I uređajima



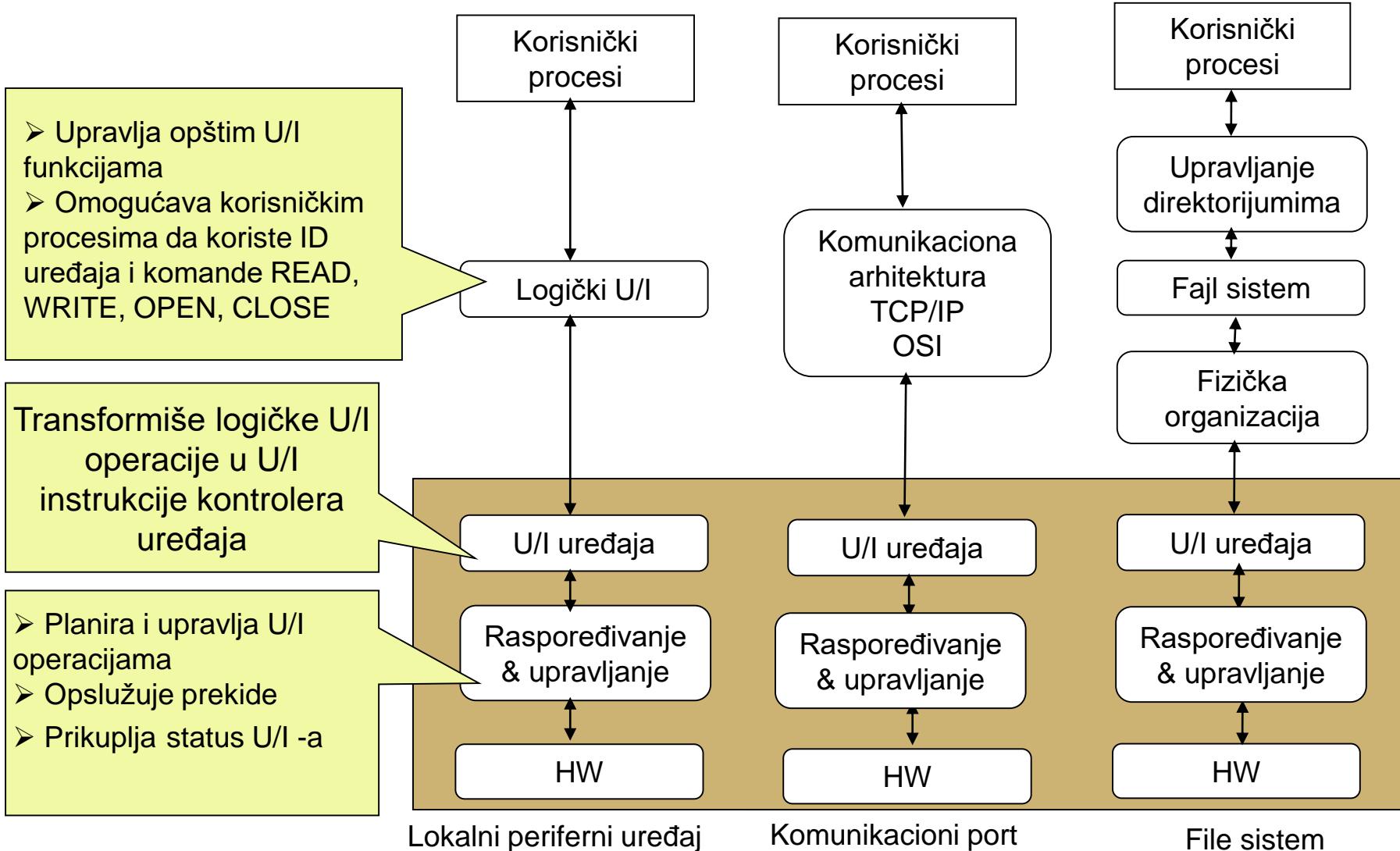
Slojevi U/I softvera



Tanenbaum, 2014

- ➊ Svaki sloj ima precizno definisanu funkciju i dobro definisan interfejs prema susednim slojevima
- ➋ Promene u jednom sloju se ne odražavaju na susedne slojeve

Modeli hijerarhijske organizacije U/I



Upravljanje U/I i planiranje diska Operativni sistemi



Lokalni periferijski uređaji

- Logički U/I:
 - Upravlja uređajem kao logičkim resursom
- U/I uređaja:
 - Transformiše logičke U/I operacije u sekvencu U/I instrukcija kontrolera uređaja
- Planiranje i upravljanje:
 - Izvršava uređenje i planiranje U/I operacija i upravljanje operacijama. Rukovanje izuzecima i očitavanje i prijavljivanje U/I statusa se obavljaju na ovom sloju.



Komunikacioni port

- Identično kao u prethodnom slučaju samo što je logički U/I modul zamjenjen komunikacionom arhitekturom
 - Ova arhitektura se sastoji od više slojeva
 - Primer, TCP/IP

File sistem

● Upravljanje direktorijumima

- Na ovom nivou simbolička imena datoteka se prevode u identifikatore kojima se referenciraju datoteke, i obezbeđuje podrška za korisničke operacije nad direktorijumima, poput *add*, *delete*, *reorganize*

● File sistem

- Obezbeđuje logičku strukturu datoteka, korisničke operacije, poput *open*, *close*, *read*, *write*, kao i prava pristupa

● Fizička organizacija

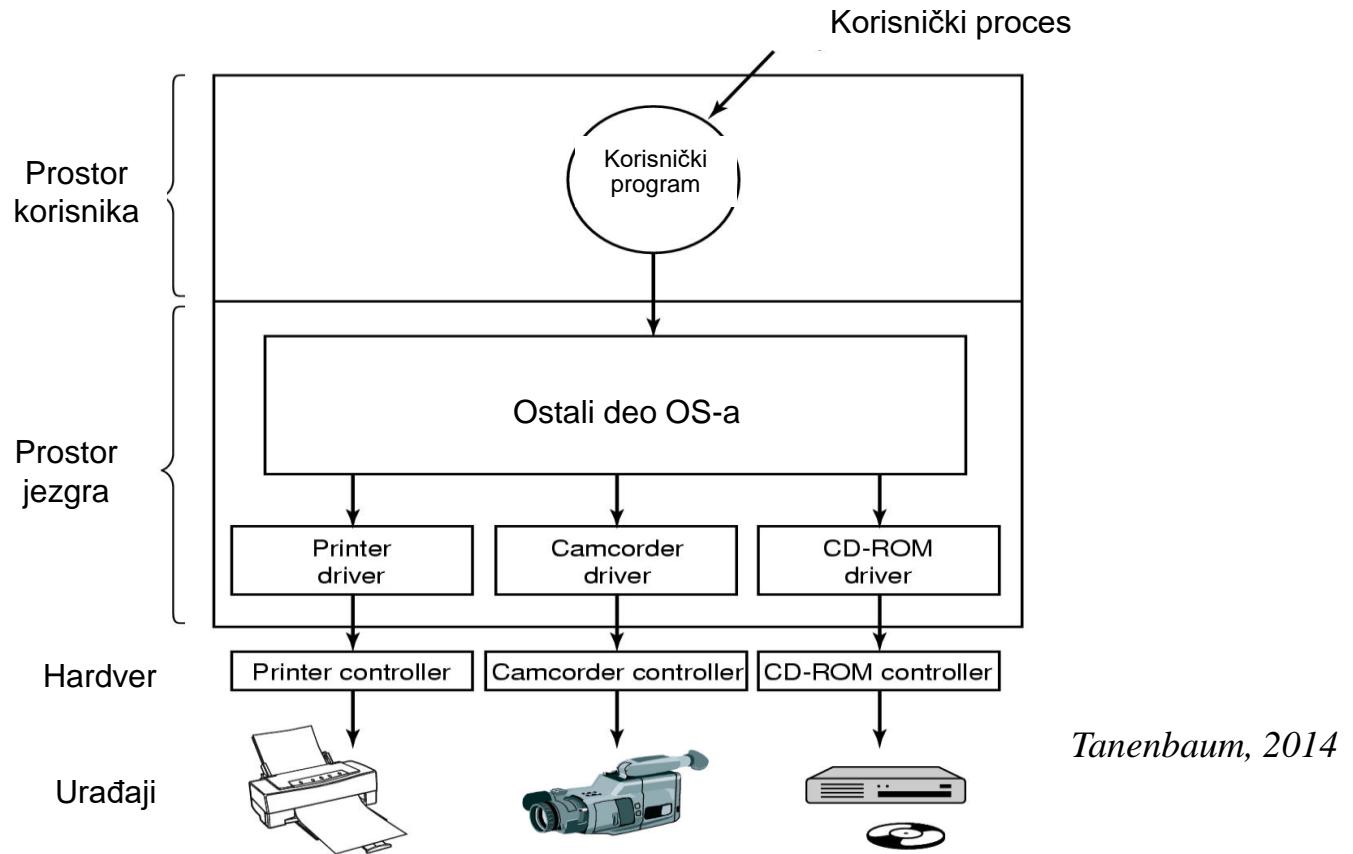
- Logičke reference na datoteke se prevode u fizičke adrese na sekundarnoj memoriji, vodeći računa o fizičkoj strukturi medijuma



Drajveri uređaja (1)

- ❖ Za svaki uređaj koji se priključuje na računar potreban je specifičan program koji će njime upravljati - to je **drajver uređaja**
- ❖ Najčešće ga isporučuju proizvođači uz sam U/I uređaj
- ❖ Drajver uređaja je prilagođen konkretnom OS-u (Unix, Linux, MS Windows)
- ❖ Mnoge specifičnosti samog U/I uređaja su ugrađene u drajver uređaja
- ❖ Svaki drajver uređaja normalno rukuje jednim tipom U/I uređaja ili jednom klasom sličnih U/I uređaja
- ❖ Drajver uređaja je deo jezgra OS-a, pa može pristupati registrima uređaja
- ❖ Drajver uređaja ima:
 - ▣ dobro definisan interfejs ka ostalim delovima OS-a
 - ▣ precizno definisaniu funkcionalnost

Drajveri uređaja (2)



- ❖ Komunikacija između drajvera i kontrolera uređaja ide preko sistemске magistrale



Drajveri uređaja (3)

Model aktivnosti drajvera

◆ **Stvarna kontrola U/I uređaja** obuhvata

- Određivanje sekvence komandi koje treba poslati U/I uređaju
- Upis komandi u registar uređaja
 - Nakon upisa svake komande drajver proverava da li kontroler uređaja prihvata tu komandu i da li je spreman da prihvati sledeću komandu
 - Neki kontrolери prihvataju komandu po komandu, dok neki imaju u memoriji bafer gde se može smestiti više komandi, odakle ih kontroler uzima na obradu bez podrške OS-a

◆ Nakon slanja svih komandi

- Drajver se blokira i čeka da ga prekid od U/I uređaja deblokira (ako obrada traje) ili
- Drajver se ne blokira (ako operacija ne traži kašnjenje)

◆ Nakon završetka operacije, drajver proverava da li je nastala greška

- Ako nije, prosleđuje podatke zahtevaocu U/I operacije
- Ako je nastala neka greška, vraća zahtevaocu kod greške
- Nakon toga bira sledeći zahtev za obradu, ako ih ima u redu zahteva, a ako ne on se blokira i čeka novi zahtev



U/I baferovanje

- ❖ Procesi moraju da čekaju dok se ne završi U/I i tek kada su podaci raspoloživi mogu da nastave sa izvršenjem
- ❖ Da bi se izbeglo uzajamno blokiranje neke stranice procesa moraju biti ("zaključane") u glavnoj memoriji tokom izvršavanja U/I
- ❖ Efikasnije je da se transfer ulaznih podataka obavi unapred u odnosu na zahtev, a da se transfer izlaznih podataka izvrši sa zakašnjenjem u odnosu na zahtev
- ❖ Bafer je oblast u memoriji gde se smeštaju podaci dok se prenose između U/I uređaja i memorijskog prostora procesa
 - ❖ Kada se ova U/I operacija izvodi može se garantovati da je zapisano stanje podataka u trenutku poziva U/I operacije



U/I baferovanje (2)

● Baferovanje orijentisano na blokove

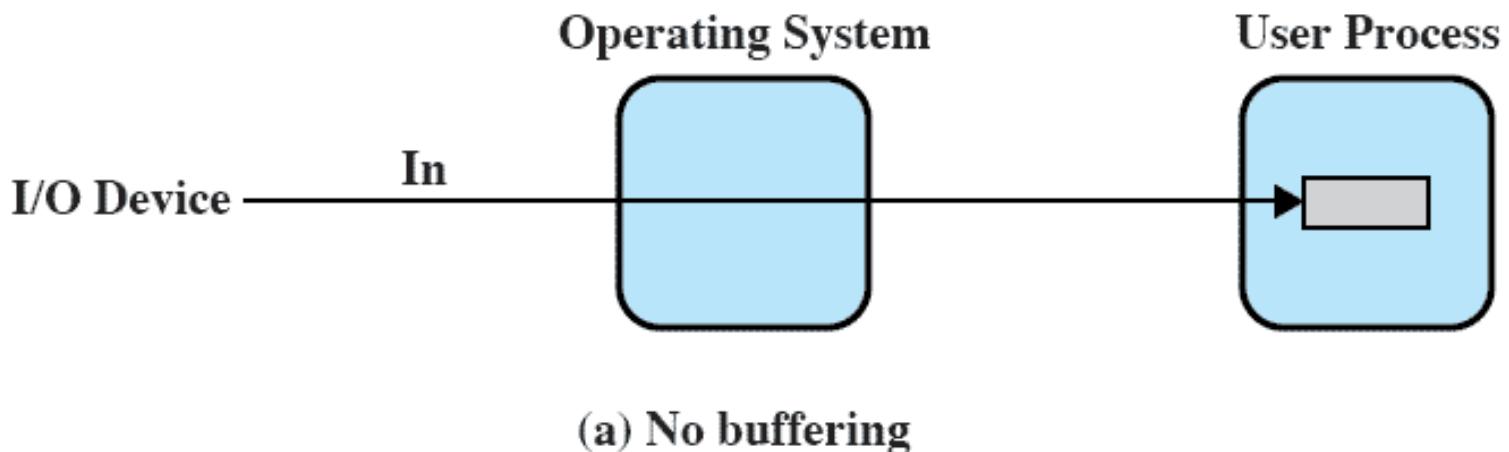
- Podaci su smešteni i prenose se u blokovima fiksne veličine
- Transfer se obavlja pojedinačno, jedan blok u jednom trenutku, na osnovu reference bloka
- Koristi se za sekundarne memorije, diskove, USB flash memorije, itd.

● Baferovanje orijentisano na tokove

- Podaci se prenose u tokovima (stream) bajtova
- Koristi se kod terminala, printer-a, komunikacionih portova, miša, tastature, itd

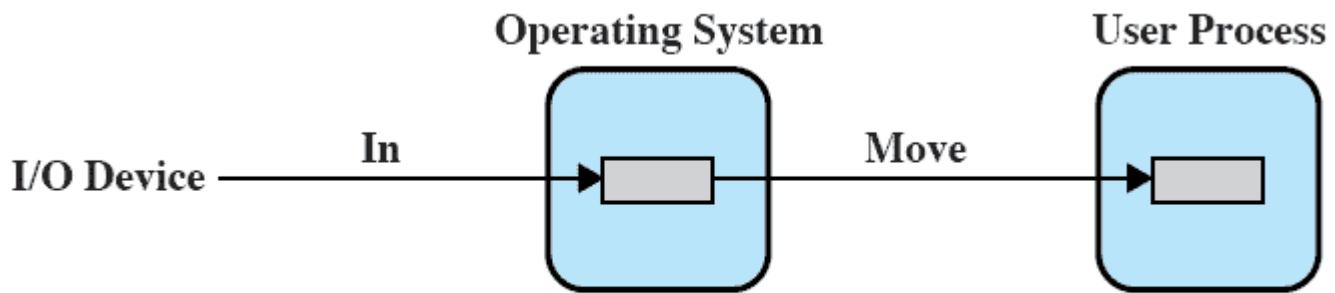
Bez U/I bafera

- Bez bafera OS direktno pristupa uređaju kada je neophodno



Jedan bafer

- Operativni sistem obezbeđuje jedan (pojedinačni) bafer u glavnoj memoriji za opsluživanje U/I zahteva

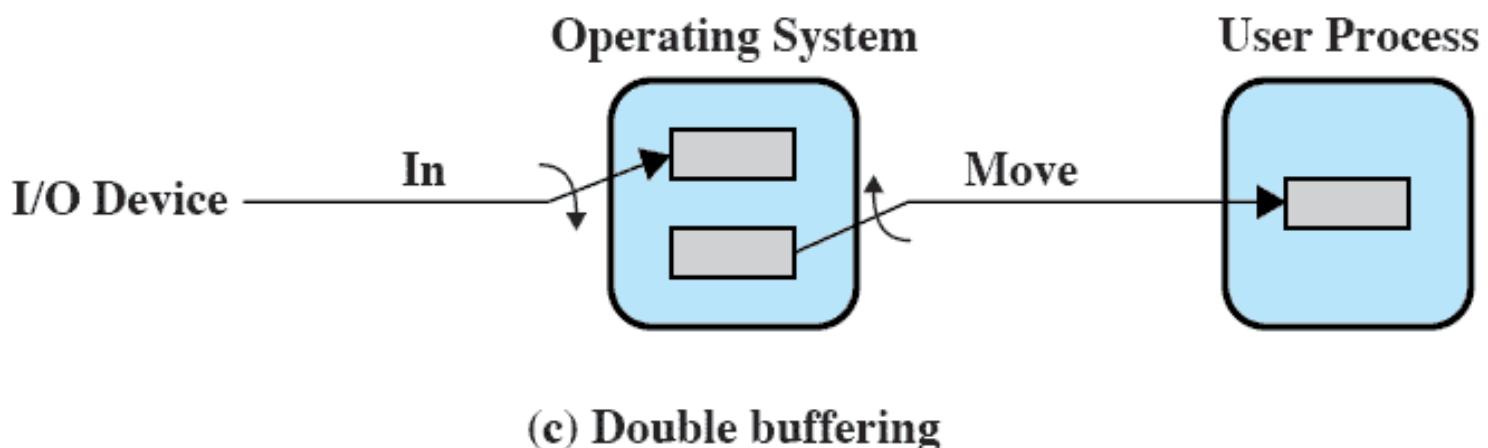


(b) Single buffering

- Blok orijentisani i tok orijentisani pojedinačni bafer

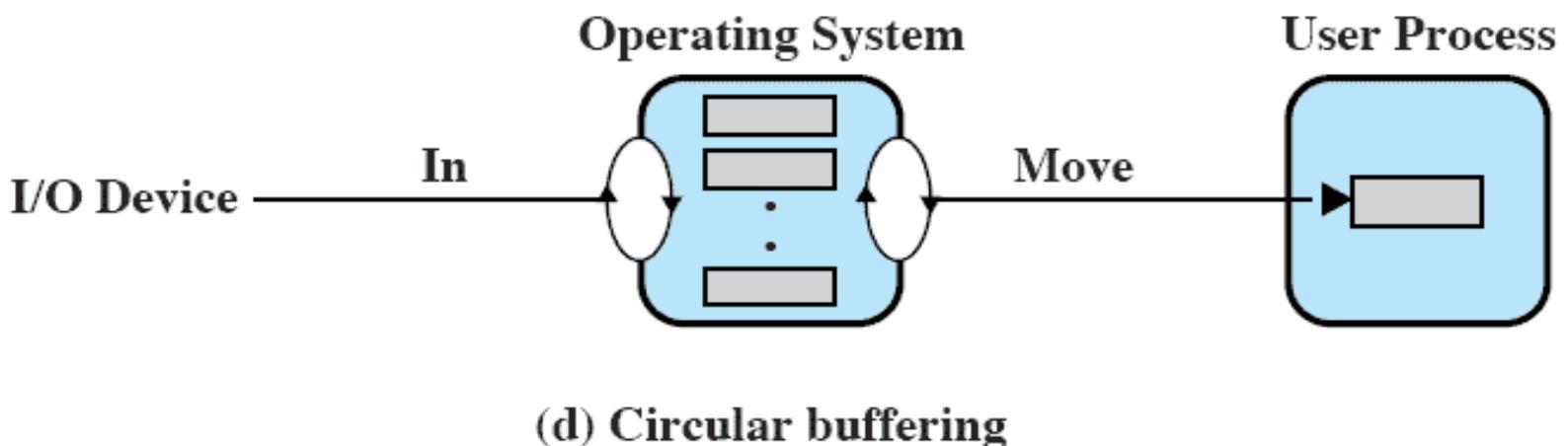
Dvostruki bafer

- Proces može prenosi podatke u ili iz jednog bafera dok operativni sistem prazni ili puni drugi bafer



Cirkularni bafer

- Svaki bafer je jedinica u okviru cirkularnog bafera
 - Proizvođač-potrošač preko ograničenog kružnog bafera
 - Koristi se kada U/I operacija mora da uskladi brzinu sa procesom



Parametri performansi diska

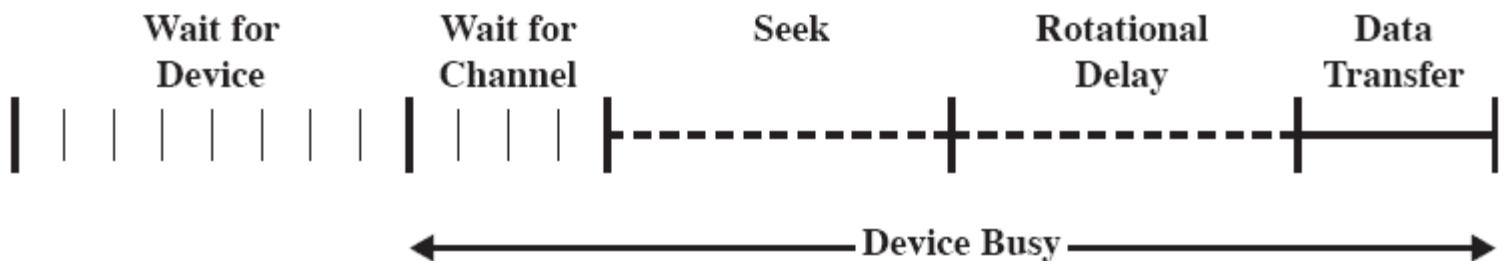
- ➊ Da bi se nešto pročitalo ili zapisalo na disk potrebno je glavu diska pozicionirati na željenu trasu i na početak željenog sektora
- ➋ Vreme potrebno za čitanje i upis bloka određuju 3 parametra
 1. **Vreme traženja (Seek time)**
 - Vreme potrebno za pozicioniranje glave diska na željenu trasu
 2. **Rotaciono kašnjenje (Rotational delay ili latency)**
 - Vreme potrebno da bi se rotiranjem diska traženi sektor našao ispod glave diska
 3. **Vreme prenosa (Transfer time)**
 - Prenos se odvija tako što se sektor pomera iznad glave diska

Vreme pristupa (access time) = Vreme traženja + Rotaciono kašnjenje + Vreme prenosa

- ➌ Vreme prenosa celog fajla je brže kada je ceo fajl zapisan na istom cilindru i u susednim sektorima
- ➍ Kontrolu grešaka radi kontroler diska

Vreme disk transfera

● Vreme transfera podataka sa diska



● $T_a = T_s + 1/2r + b/rN$

- r – rotaciona brzina diska u obrtajima u sekundi
- b - broj bajtova koji se prenose
- N – broj bajtova na stazi diska



Strategije raspoređivanja diska (1)

- ❖ Vreme traženja je razlog za razlike u performansama
- ❖ Za jedan disk postoji više U/I zahteva
- ❖ Ako se zahtevi biraju slučajno, to može dovesti do loših performansi diska
- ❖ Moguće strategije (politike) raspoređivanja
 - FIFO ili FCFS (First Come First Served)
 - Prioritet
 - LIFO
 - SSF (Shortest Seek First) ili SSTF (short service time first)
 - SCAN ili elevator
 - C-SCAN
 - N-step-SCAN
 - FSCAN



Strategije raspoređivanja diska (2)

● FIFO (prvi došao prvi uslužen)

- Zahtevi se obrađuju sekvenčno
- Fer za sve procese

● Prioritet (prvo zahtev najvišeg prioriteta)

- Cilj nije optimizacija korišćenja diska već ispunjenje nekih drugih ciljeva
- Kraći batch poslovi mogu imati prioritet
- Nudi dobro vreme odgovora kod interaktivnog režima

● LIFO (zadnji došao prvi uslužen)

- Dobar za sisteme za obradu transakcija
 - Uredaj se daje poslednjem korisniku tako da se glava malo pomera
- Moguće izgladnjivanje

Strategije raspoređivanja diska (3)

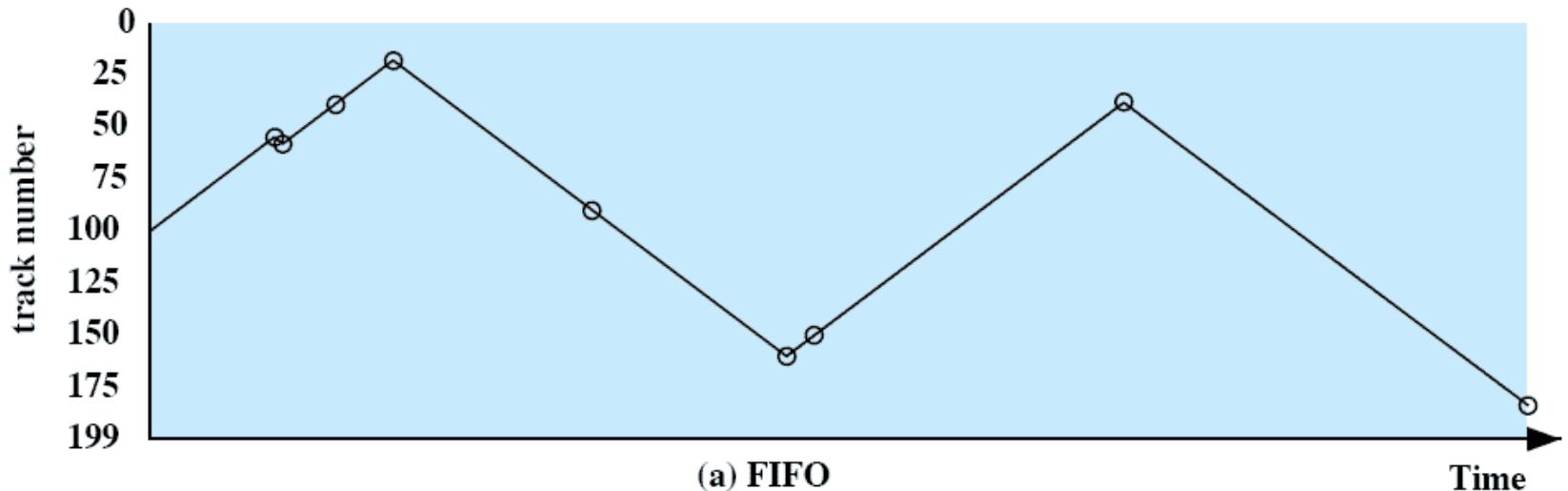
- ❖ SSF (Shortest Seek First) ili SSTF (Shortest Seek Time First)
 - ❖ Bira se U/I zahtev koji zahteva najmanje kretanje glave diska sa njegove trenutne pozicije
 - ❖ Uvek se bira minimalno vreme traženja
- ❖ SCAN ili elevator
 - ❖ Glava se pomera samo u jednom pravcu, zadovoljavajući zahteve sve dok ne dostigne poslednju trasu u tom pravcu
 - ❖ Tada se glava okreće i pomera se u obrnutom pravcu, nastavljujući da uslužuje zahteve na koje naiđe
- ❖ C-SCAN
 - ❖ Ograničava se skeniranje samo na jedan pravac
 - ❖ Kada se dostigne zadnja trasa u jednom pravcu ili kada više nema zahteva u tom pravcu, glava se vraća na drugi kraj diska i tada ponovo vrši skenirenje
- ❖ N-step-SCAN
 - ❖ Segmentira se red zahteva za diskom u podredove dužine N
 - ❖ Podredovi se obrađuju istovremeno korišćenjem SCAN
- ❖ FSCAN
 - ❖ Dva reda; jedan red je prazan za novi zahtev



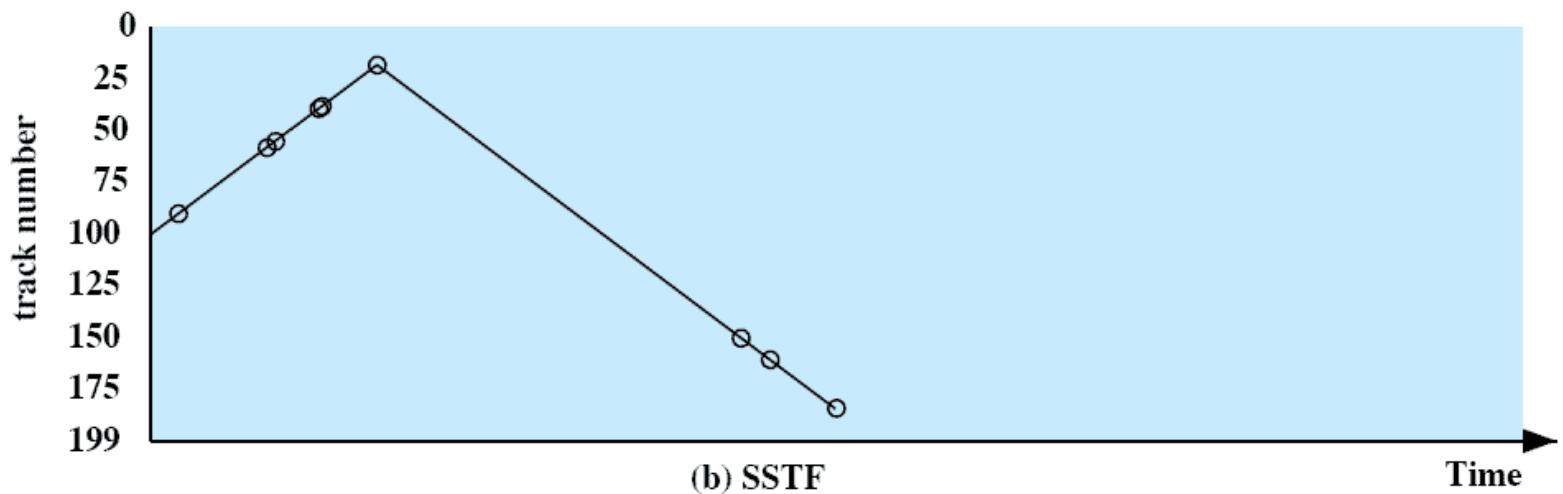
Algoritmi za planiranje diska

- ➊ Primer:
- ➋ Disk ima 200 staza
- ➌ Glava diska je inicijalno na stazi 100
- ➍ U redu zahteva su pristigli zahtevi za podacima na određenim stazama diska
 - 55, 58, 39, 18, 90, 160, 150, 38, 184

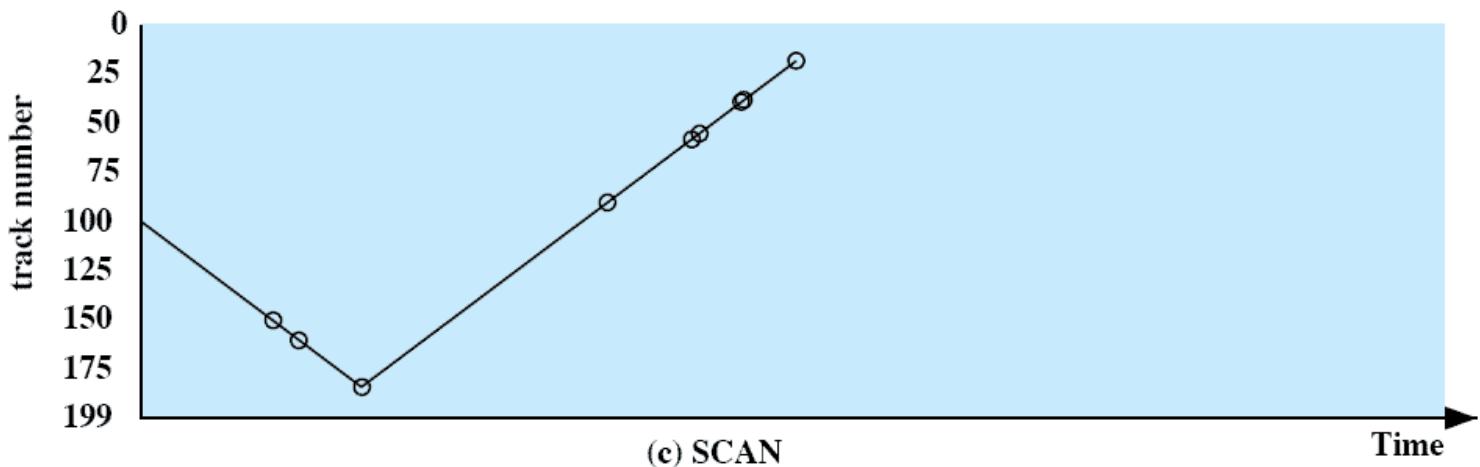
FIFO



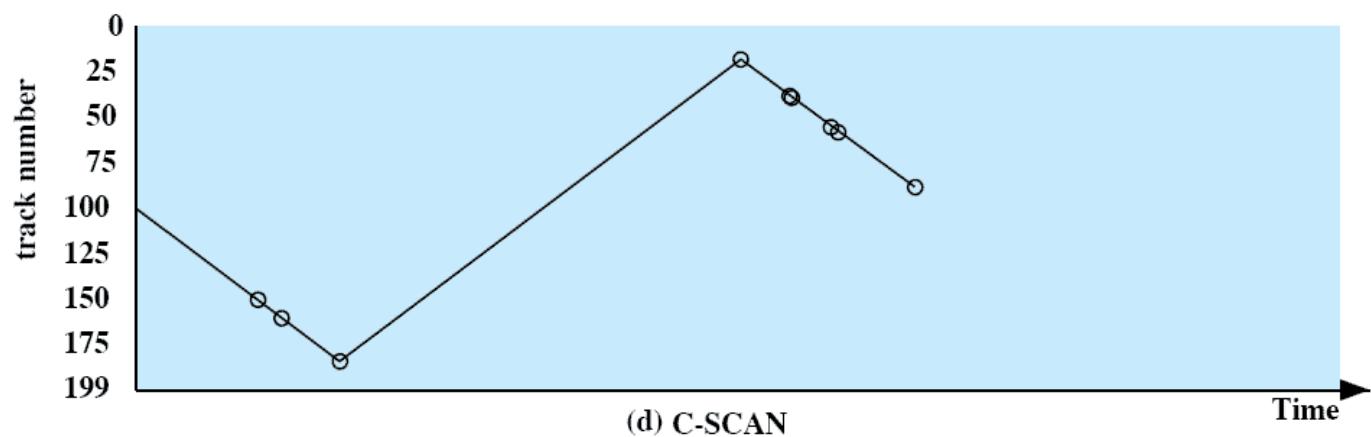
Shortest Service Time First



SCAN



C-SCAN



Uporedni pregled performansi

(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	Number of tracks traversed	Next track accessed	Number of tracks traversed	Next track accessed	Number of tracks traversed	Next track accessed	Number of tracks traversed
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	Average seek length	27.5	Average seek length	27.8	Average seek length	35.8

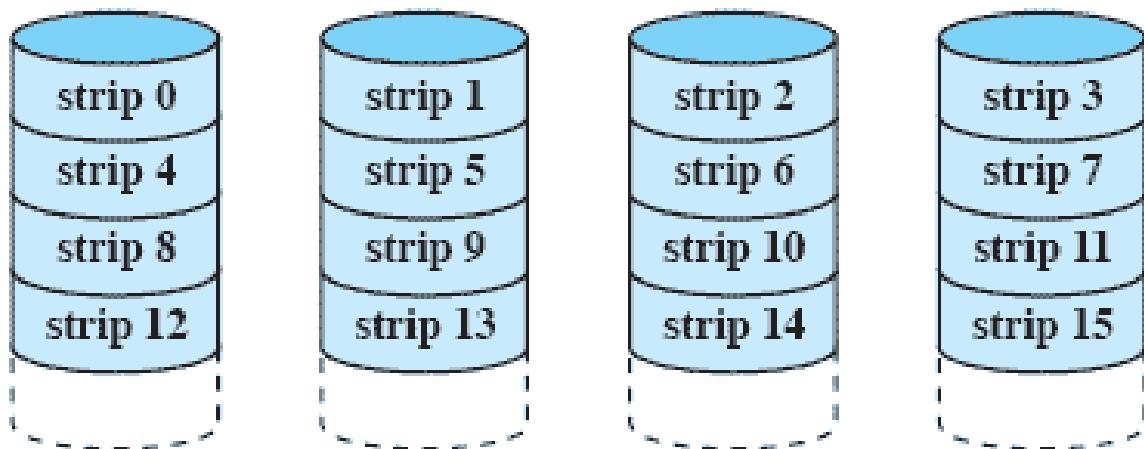


RAID

- ❖ *Redundant Array of Independent Disks*
- ❖ Skup fizičkih disk drajvova koje operativni sistem vidi kao jedan logički drajv
- ❖ Podaci su distribuirani po fizičkim drajvovima
- ❖ Redundantnost se koristi za smeštanje informacija o parnosti (kontrola i ispravka greške) čime se omogućuje oporavak nakon greške ili otkaza diska

RAID 0 – Podela u strip-ove

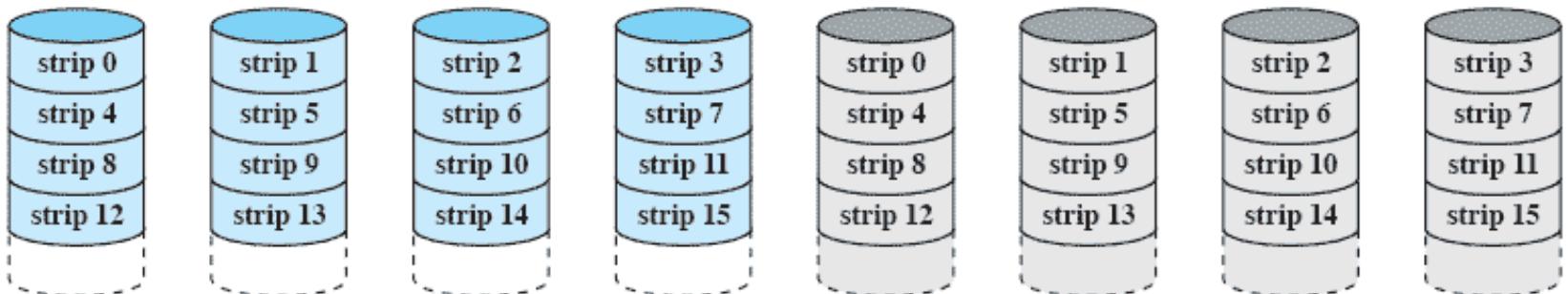
- Nije potpuni RAID – nema redundantnosti
- Otkaz diska ne može da se ispravi
- Veoma brz usled paralelnog čitanja/upisa



(a) RAID 0 (non-redundant)

RAID 1 - Mirroring

- Redundantnost kroz dupliciranje umesto kroz parnost
- Zahtev za čitanjem može biti obavljen paralelno
- Jednostavan oporavak usled otkaza diska

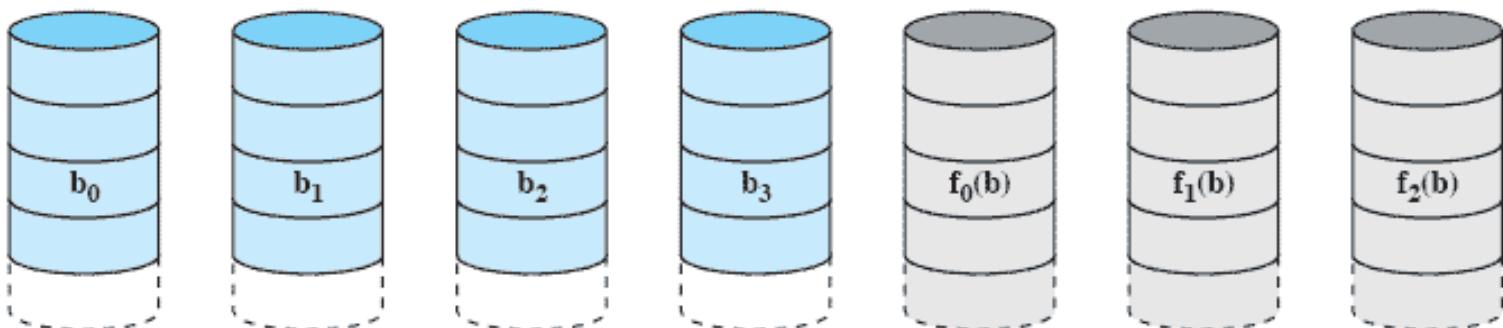


(b) RAID 1 (mirrored)

RAID 2

(Korišćenje Hamming koda)

- ◆ Sinhronizovano rotiranje diska
- ◆ Podela podatak (stripping) se koristi u ekstremno malim jedinicama (bajt, reč)
- ◆ Hamming kod se koristi za korigovanje grešaka na jednom bitu i detektovanje grešaka na dva bita

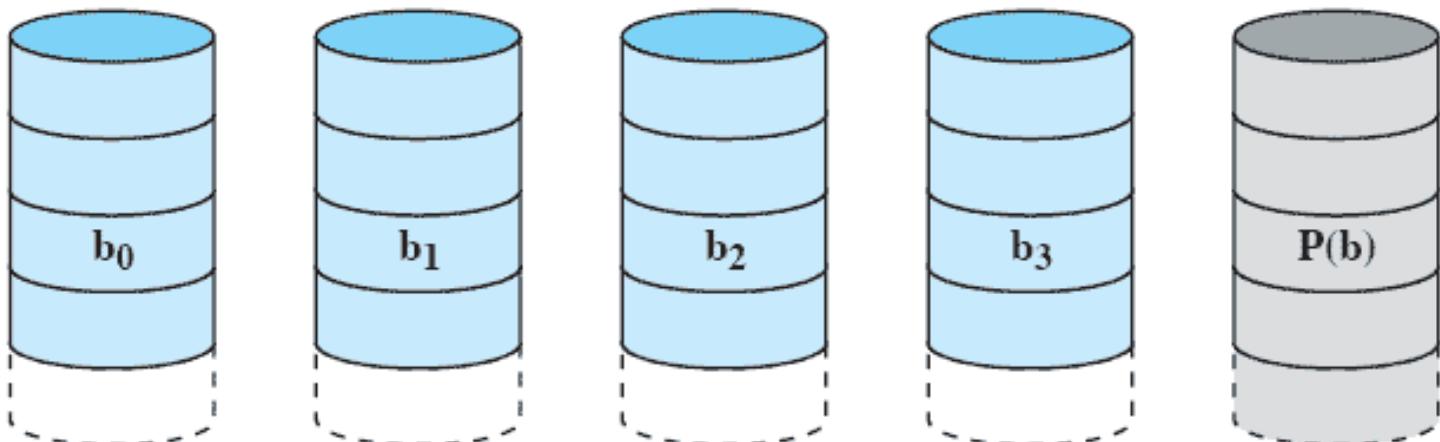


(c) RAID 2 (redundancy through Hamming code)

RAID 3

Bit parnosti

- Slično RAID-2 ali koristi bitove parnosti koji su smešteni na posebnom disku

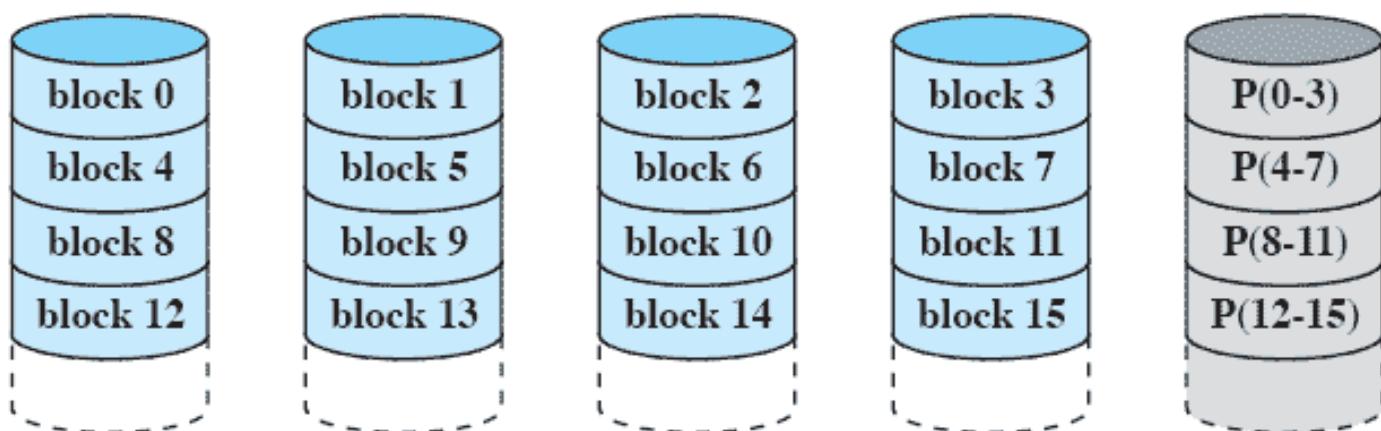


(d) RAID 3 (bit-interleaved parity)

RAID 4

Parnost na nivou bloka

- Parnost je izračunata nad odgovarajućim bitom po različitim blokovima i smeštena na poseban disk u odgovarajućem bloku

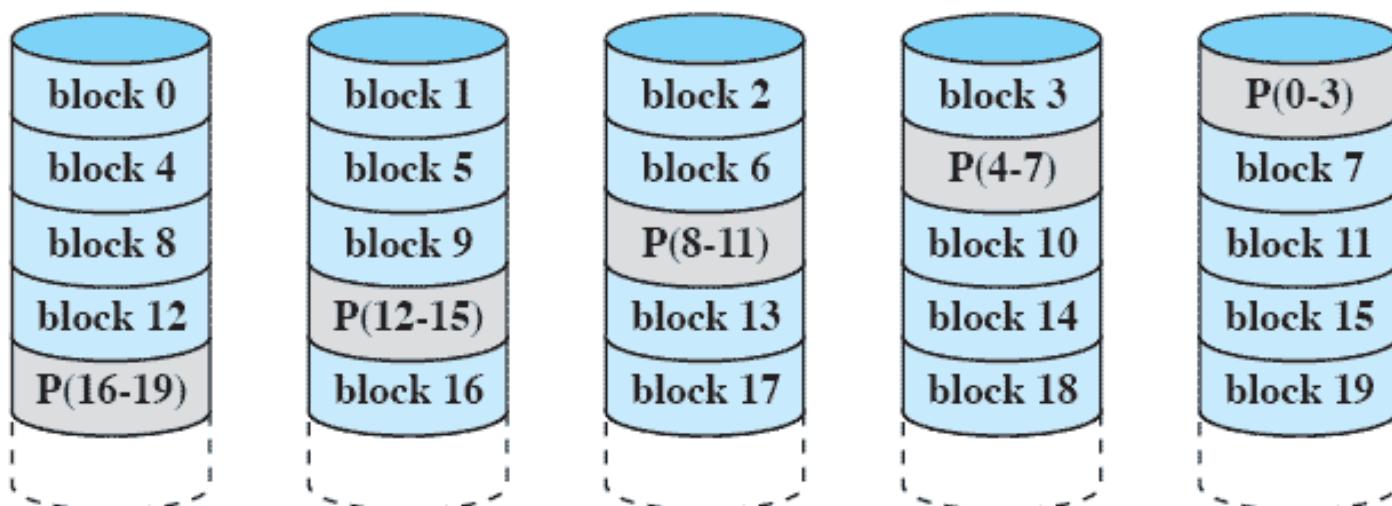


(e) RAID 4 (block-level parity)

RAID 5

Distribuirana parnost na nivou bloka

- Slično RAID-4 ali se bitovi parnosti distribuiraju po svim diskovima

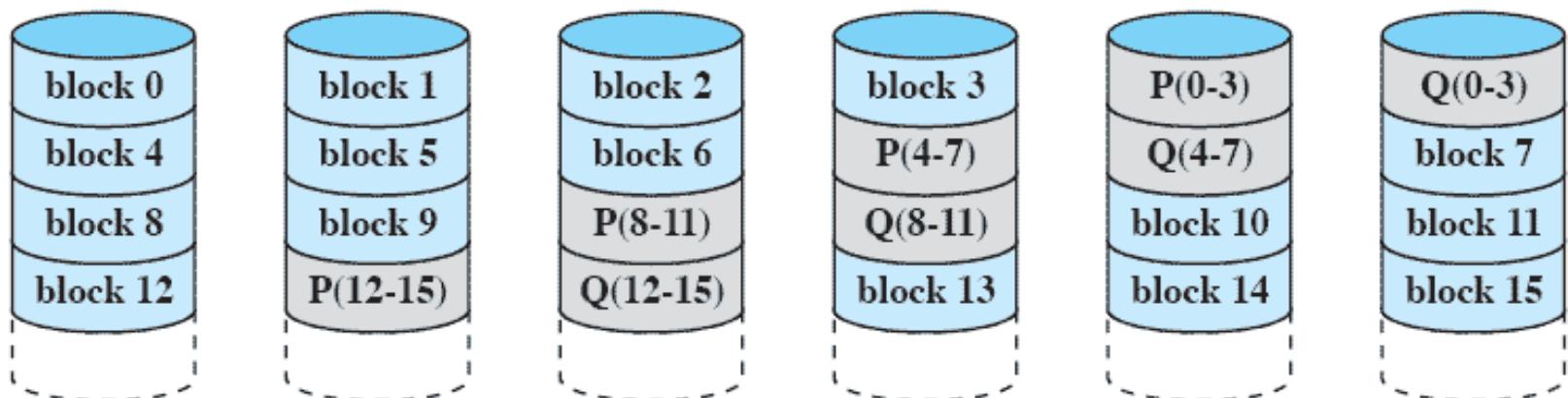


(f) RAID 5 (block-level distributed parity)

RAID 6

Dualna redundantnost

- Obavljuju se dva različita izračunavanja parnosti i smeštaju u posebne blokove na različitim diskovima
- Može se oporaviti od otkaza dva diska



(g) RAID 6 (dual redundancy)

Disk keš

- ➊ **Keš** je bafer u glavnoj memoriji gde se čuvaju kopije podataka sa diska (blokova diska)
 - Pristup kopijama podataka u kešu je mnogo efikasniji nego originalnim podacima
- ➋ Primer U/I diska:
 - Kad stigne U/I zahtev, OS prvo proverava da li su traženi podaci u kešu
 - Ako jesu, procesu se prosleđuju podaci iz memorije, bez pristupa disku
 - Ako nisu u kešu, OS ih učitava sa diska
 - Slično je i kod upisa na disk - U kešu se skupljaju podaci pre upisa na disk
- ⌋ Strategije zamene blokova u kešu
 - LFU - *Least Frequently Used*
 - LRU - *Least Recently Used*

UNIX SVR4 I/O

- ❖ Svakom U/I uređaju je pridružena specijalna datoteka
 - Obezbeđuje jasan i uniforman interfejs ka uređajima za korisnike i procese
- ❖ Za pristup uređaju izdaju se zahtevi za čitanje i upis (*read, write*) na specijalnu datoteku
- ❖ Dva tipa U/I: baferovan i nebaferovan

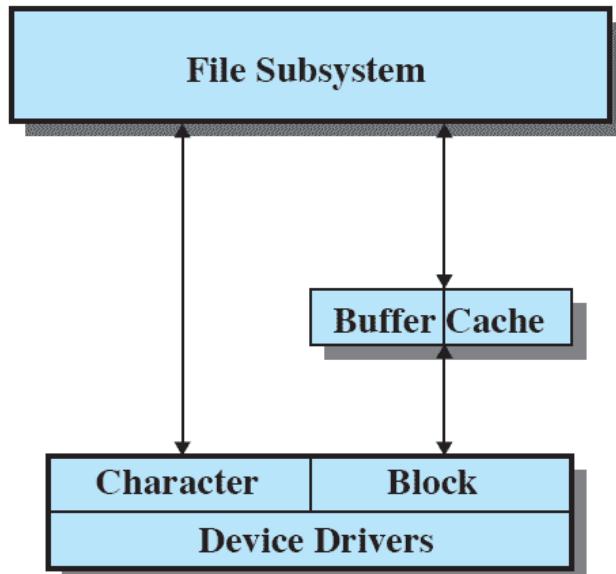


Figure 11.12 UNIX I/O Structure

Keširanje

- Keš bafer - održavaju se tri liste

- Lista slobodnih
- Lista uređaja
- U/I red drajvera

- Keš znakova

- Koristi se za znakovno orijentisane uređaje, terminale, štampače

- Nebaferovan U/I - DMA

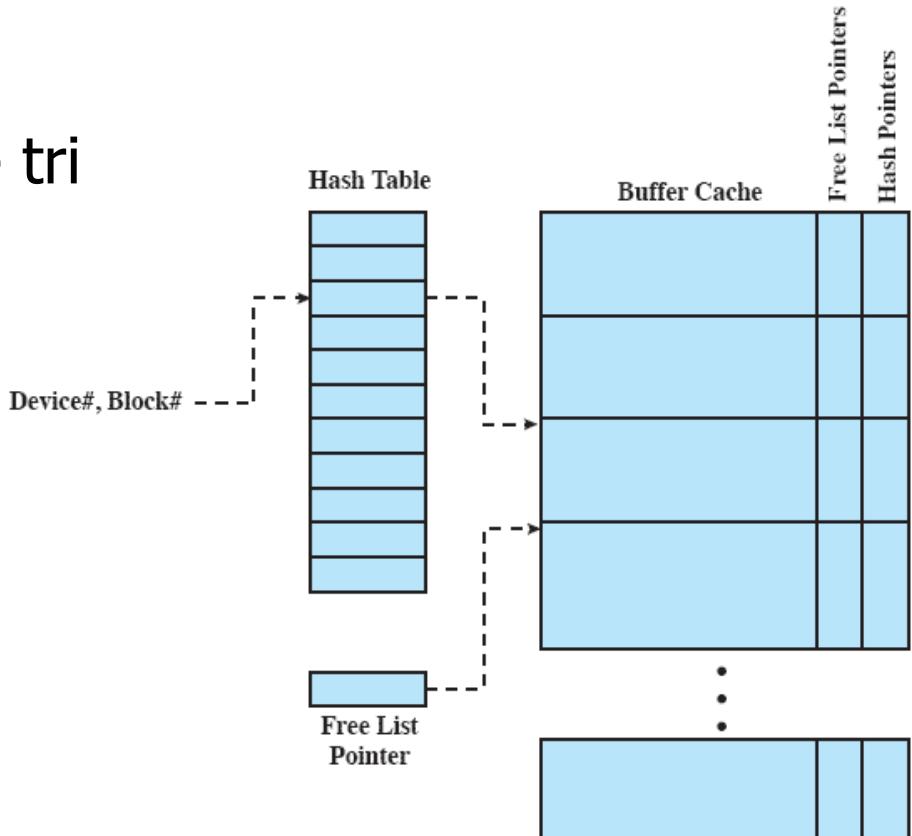


Figure 11.13 UNIX Buffer Cache Organization

Linix U/I

- ➊ Linux i UNIX imaju veoma slične mehanizme za upravljanje U/I
 - Linux kernel dodeljuje specijalnu datoteku svakom U/I drajveru uređaja
 - Razlikuju se blok, znakovni i mrežni uređaji
- ➋ Koristi Linux Elevator planer za planiranje diska – varijanta LOOK (SCAN) algoritma
 - U Linux 2.6 proširen sa dva dodatna algoritma: *Deadline* U/I planer (*Rasporedi vač po roku*) i *anticipatory* U/I planer (*Rasporedi vač sa predviđanjem*)
- ➌ Održava jedinstveni *page* keš za sav saobraćaj između diska i glavne memorije

Windows U/I

- ❖ Windows I/O manager - odgovaran za U/I u okviru Windows OS
- ❖ Obezbeđuje jedinstven interfejs za sve tipove drajvera uređaja
- ❖ Sastoji se od 4 modula
- ❖ Dva moda U/I operacija
 - ❖ Sinhroni
 - ❖ Asinhroni
- ❖ Podržani su RAID-0, 1, 5, 6

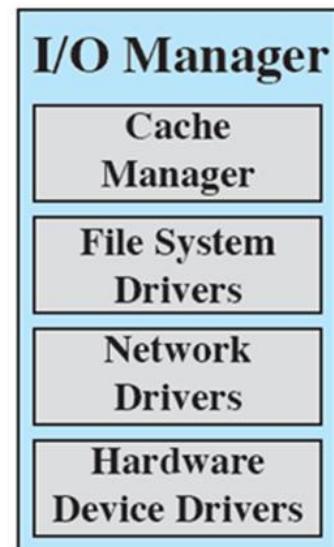


Figure 11.15 Windows I/O Manager



Domaći zadatak

◆ Poglavlje **11 Upravljanje U/I i planiranje diska**

- ▣ 11.13 Ključni pojmovi, kontrolna pitanja i problemi

◆ U/I animations

- ▣ <https://apps.uttyler.edu/Rainwater/COSC3355/Animations>

- ▣ Interrupt-Driven I/O Cycle
- ▣ The Life Cycle of an I/O Request
- ▣ Disk Scheduling Algorithms: FCFS, SSTF, SCAN



Operativni sistemi

- Upravljanje datotekama -

Prof. dr Dragan Stojanović

Katedra za računarstvo
Univerzitet u Nišu, Elektronski fakultet



Literatura

- ➊ *Operating Systems: Internals and Design Principles*, edition, W. Stallings, Pearson Education Inc., 7th – 2012, (5th -2005, 6th - 2008, 8th – 2014 , 9th – 2017)
 - ▣ <http://williamstallings.com/OperatingSystems/>
 - ▣ <http://williamstallings.com/OperatingSystems/OS9e-Student/>

- ➋ Poglavlje 12: Upravljanje datotekama



Sistem za upravljanje datotekama

- ❖ **Datoteka** (*file*) je imenovana kolekcija povezanih podataka zapisana na sekundarnoj memoriji (magnetni diskovi, magnetne trake, optički diskovi, USB flash, itd.)
- ❖ **Sistem za upravljanje datotekama** (**Datotečni sistem**, *File management sistem*, *File system*) je deo OS-a zadužen za upravljanje datotekama koji obezbeđuje pogodan interfejs za korisnike datoteka
- ❖ Detalji o organizaciji i implementaciji sistema za upravljanje datotekama nisu bitni za korisnika, ali jesu za projektante OS-a i sistem programere



Sistem za upravljanje datotekama

- ◆ *File system, File management sistem*
- ◆ Omogućuje korisnicima kreiranje datoteke kao kolekcije podataka, koja ima sledeće svojstva:
 - Dugoročno postojanje
 - Deljiva je između procesa
 - Poseduje odgovarajuću internu strukturu
- ◆ Obezbeđuje operacije nad datotekama
 - Kreiranje (*Create*)
 - Brisanje (*Delete*)
 - Otvaranje (*Open*)
 - Zatvaranje (*Close*)
 - Čitanje (*Read*)
 - Upis (*Write*)



Struktura datoteke

- ➊ Datoteka može biti struktuirana na dva različita načina:
 1. Sekvenca (niz, *stream*) **bajtova** (Unix, Unix-like OS i Windows)
 2. Sekvenca **slogova** (*records*) fiksne ili promenljive dužine
 - **Polje** (*field*) predstavlja elementarni podatak u okviru sloga, koji se karakteriše imenom, dužinom polja i tipom podatka (ime zaposlenog, datum zaposlenja, itd.). Može biti fiksne ili promenljive dužine
 - **Slog** je kolekcija polja koja se tretiraju kao celina od strane aplikacionog programa; ukoliko su polja u okviru sloga promenljive dužine, i slog je promenljive dužine
 - **Datoteka** predstavlja kolekciju povezanih slogova koja se referencira imenom
 - **Baza podataka** predstavlja kolekciju podataka u datotekama koji su eksplicitno povezani i koji se odnose na određeni domen (npr. podaci o fakultetu, profesorima, asistentima, studentima, predmetima, itd.)



Atributi datoteke

- ◆ Skup atributa se razlikuje od jednog do drugog OS
- ◆ Obično su to:
 - *Ime datoteke*: simboličko ime čitljivo za čoveka
 - *Identifikator (descriptor)*: jedinstvena oznaka datoteke, obično broj, kojom se identificuje fajl unutar fajl sistema; nije čitljiv za čoveka
 - *Tip*: neophodan kod OS-a koji podržavaju različite tipove
 - *Lokacija*: pokazivač na uređaj i lokaciju datoteke na tom uređaju
 - *Veličina*: tekuća veličina datoteke u bajtovima ili blokovima
 - Ponekad se kao atribut uključuje max dozvoljena veličina datoteke
 - *Zaštita*: informacija o kontroli pristupa datoteci kojom se određuje ko može čitati, upisivati, izvršavati, itd. datoteku
 - *Vreme, datum i identifikacija korisnika*: odnose se na kreiranje, zadnju modifikaciju i zadnje korišćenje; korisno za zaštitu, sigurnost i monitoring korišćenja
- ◆ Informacije o datotekama se čuvaju u
 - Direktorijumima,
 - Posebnoj strukturi podataka (*i-čvor, FCB*)

Upravljanje datotekama

Operativni sistemi



Sistemi za upravljanje datotekama

- ➊ Sistem za upravljanje datotekama je onaj deo operativnog sistema koji korisnicima i aplikacijama obezbeđuje usluge u pogledu upotrebe datoteka
- ➋ Programer ili korisnik nema potrebe da razvija softver posebne namene za svaku aplikaciju za upravljanje datotekama



Ciljevi sistema za upravljanje datotekama

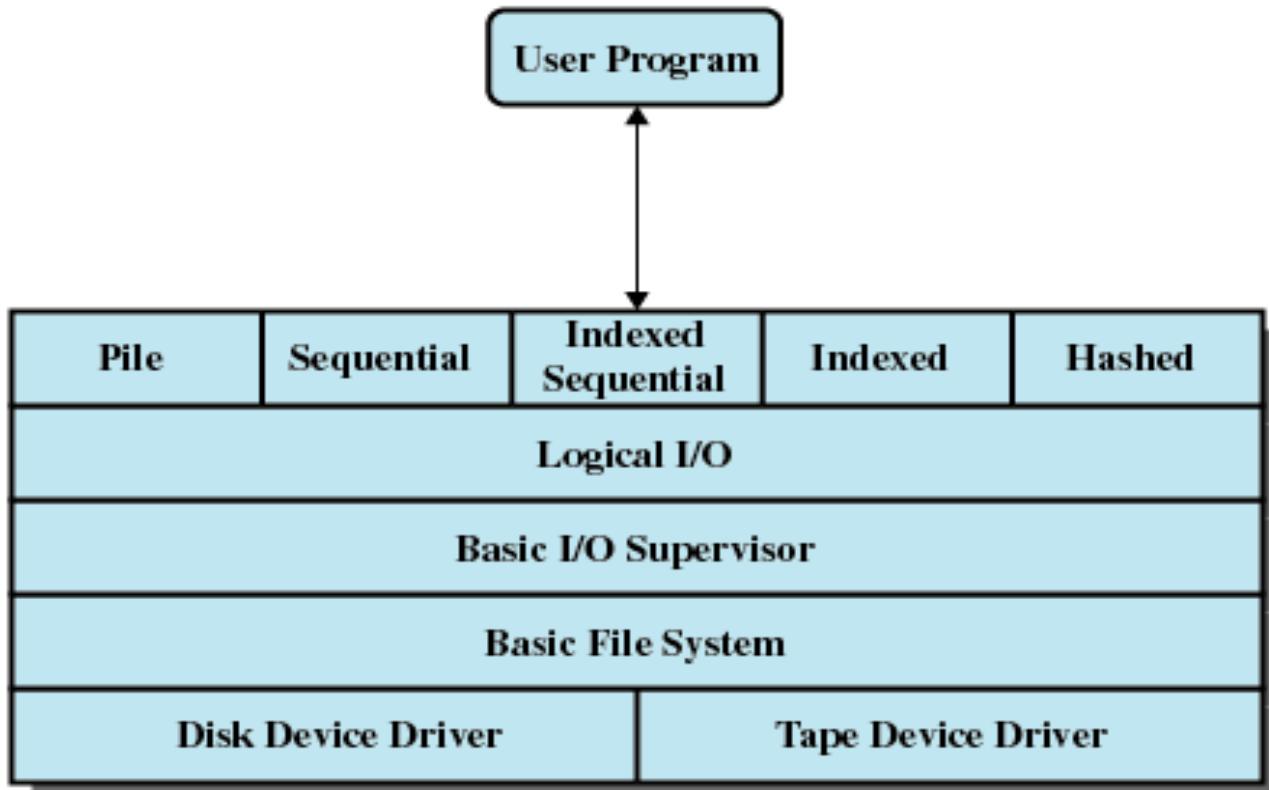
- ◆ Zadovoljiti potrebe i **zahteve korisnika za upravljanje podacima** (skladištenje podataka i operacije za rad sa datotekama)
- ◆ Garantovati da su podaci u datotekama **validni**
- ◆ Optimizovati **performanse** (propusna moć, vreme odziva)
- ◆ Obezbediti U/I podršku za **razne vrste** uređaja za skladištenje podataka
- ◆ Svesti na najmanju meru ili odstraniti mogućnost **gubitka** ili uništenja podataka
- ◆ Obezbediti standardizovani **skup U/I funkcija** kao interfejsa za korisničke procese
- ◆ Obezbediti U/I podršku za **više korisnika** (kod višekorisničkih sistema)



Zahtevi korisnika za upravljanje podacima

1. Svaki korisnik bi trebalo da može da kreira, briše, čita, upisuje i modifikuje datoteke
2. Svaki korisnik može da ima kontrolisani pristup datotekama drugih korisnika
3. Svaki korisnik može da kontroliše koje vrste pristupa korisničkim datotekama su dozvojene
4. Svakom korisniku bi trebalo omogućiti da restruktuiru korisničke datoteke u oblik koji odgovara konkretnom problemu
5. Svakom korisniku bi trebalo omogućiti da pomera podatke između datoteka
6. Svakom korisniku bi trebalo omogućiti da pravi rezervne kopije i da obnavlja korisničke datoteke u slučaju oštećenja
7. Svakom korisniku bi trebalo omogućiti da pristupa korisničkim datotekama koristeći simbolička imena

Arhitektura sistema datoteka (*File system-a*)





Upravljački programi

Device Drivers

- ❖ Najniži nivo
- ❖ Komunicira direktno sa periferijskim uređajem
- ❖ Odgovoran je za startovanje U/I operacija na uređaju
- ❖ Obrađuje kompletiranje U/I zahteva
- ❖ Deo je OS-a



Osnovni sistem datoteka

- ❖ Fizički U/I sloj
- ❖ Primarni interfejs sa okolinom izvan računarskog sistema
- ❖ Radi sa blokovima podataka koji se razmenjuju sa sistemima diska ili trake
- ❖ Bavi se smeštanjem blokova na uređaju sekundarne memorije i baferovanjem tih blokova u glavnoj memoriji
- ❖ Ne razume sadržaj podataka, niti strukturu datoteka koji su uključeni u U/I
- ❖ Deo je OS-a



Osnovni U/I supervizor

- Odgovoran je za svako iniciranje i završavanje operacije U/I datoteke
- Održava upravljačke strukture koje su odgovorne za
 - U/I uređaja
 - Planiranje diska
 - Status datoteka
- Bira uređaj na kome treba da se izvrši U/I datoteke zavisno od selektovane datoteke
- Bavi se raspoređivanjem diska i trake u cilju optimizovanja performansi
- Na ovom nivou se dodeljuju U/I baferi i alocira potrebna sekundarna memorija
- Deo je OS



Logički U/I

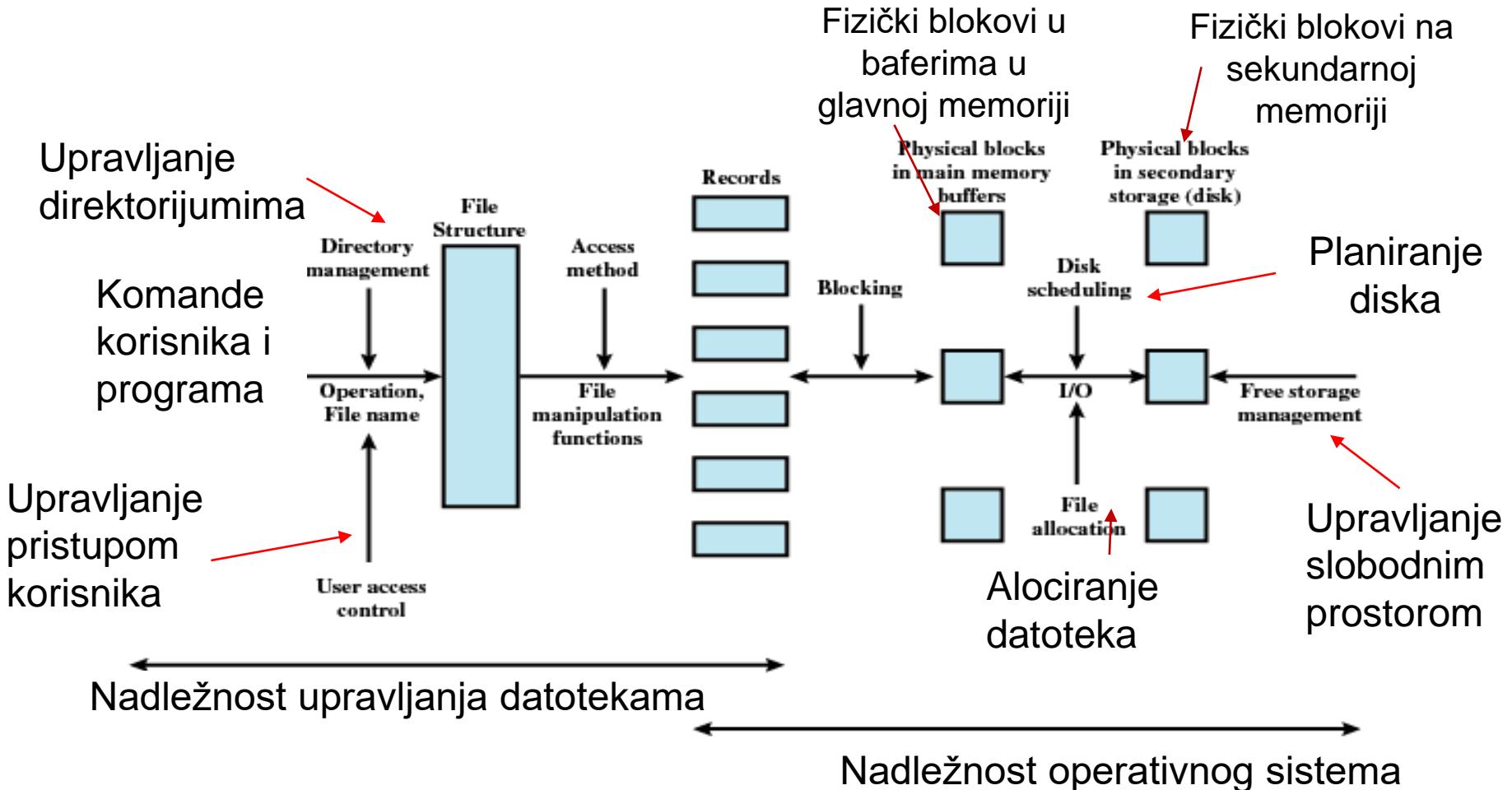
- ➊ Omogućava korisnicima i aplikacijama da pristupaju slogovima (*records*)
- ➋ Obezbeđuje U/I podršku opšte namene za slogove
- ➌ Održava osnovne podatke o datotekama



Metode pristupa

- ◆ Sloj *file system-a* najbliži korisnicima
- ◆ Obezbeđuje standardni interfejs između aplikacija i datotečnog sistema i uređaja na kojima su smeštene datoteke
- ◆ Uzima u obzir strukturu datoteka (serijsku, sekvencijalnu,...)
- ◆ Za različite strukture datoteka obezbeđuje različite metode pristupa i obrade podataka

Funkcije za upravljanje datotekama





Funkcije upravljanja datotekama

- Identificuje i locira datoteku nad kojom se zahteva operacija od strane korisnika ili aplikacionog programa
 - korišćenjem direktorijuma kojim se opisuju lokacije svih datoteka i njihovi atributi
- Ako je dozvoljeno deljenje datoteka opisuje pristupne privilegije
- Vrši grupisanje slogova u blokove i obrnuto za pristup datotekama radi čitanja i upisa
- Vrši dodelu slobodnih blokova datotekama
- Bavi se raspoređivanjem diska i alociranjem datoteke u cilju optimizovanja performansi
- Upravlja slobodnim prostorom na disku (blokovi)



Organizacija i pristup datotekama

◆ U izboru organizacije datoteke važni su sledeći kriterijumi

- Kratko vreme pristupa
- Lako ažuriranje
- Ekonomičnost smeštanja datoteke
- Jednostavno održavanje
- Pouzdanost

◆ Tipovi organizacija datoteke:

- Gomila (*pile*)
- Sekvencijalna datoteka
- Indeksirano-sekvencijalna (indeks-sekvencijalna) datoteka
- Indeksirana datoteka
- Direktna (heširana) datoteka



Tipovi organizacije datoteka i pristupa (1)

● Gomila

- Podaci su organizovani u redosledu u kom su stizali, pri čemu slogovi mogu imati različit redosled polja ili različita polja
- Pristup podacima (slogovima) zahteva iscrpno pretraživanje

● Sekvencijalna datoteka

- Slogovi su fiksnog formata
- Slogovi mogu imati polje koje predstavlja ključ koje ga jedinstveno identificuje
- Novi slogovi dodaju se na kraj datoteke, ili na određeno mesto ukoliko je datoteka sortirana po ključu
- Pogodan kada je medijum magnetna traka ili disk
- Loše performanse za interaktivni pristup koji uključuje upite i pretraživanja individualnih slogova



Tipovi organizacije datoteka i pristupa (2)

Indeksna sekvencijalna datoteka

- Organizuje slogove u sekvenci u skladu sa vrednošću ključa
- Za podršku *random* pristupu dodaje se indeksna datoteka pomoću koje se na osnovu ključa pristupa slogovima u blizini traženog sloga
- Indeksna datoteka je sekvencijalna i svaki slog sadrži dva polja: ključ (isti kao u glavnoj datoteci) i pokazivač na glavnu datoteku
- Novi slog se dodaje u tzv. *overflow* datoteku, i na njih ukazuje odgovarajući slog glavne datoteke koji mu neposredno prethodi na osnovu vrednosti ključa
- Moguće je indeksiranje u dva ili više nivoa



Tipovi organizacije datoteka i pristupa (3)

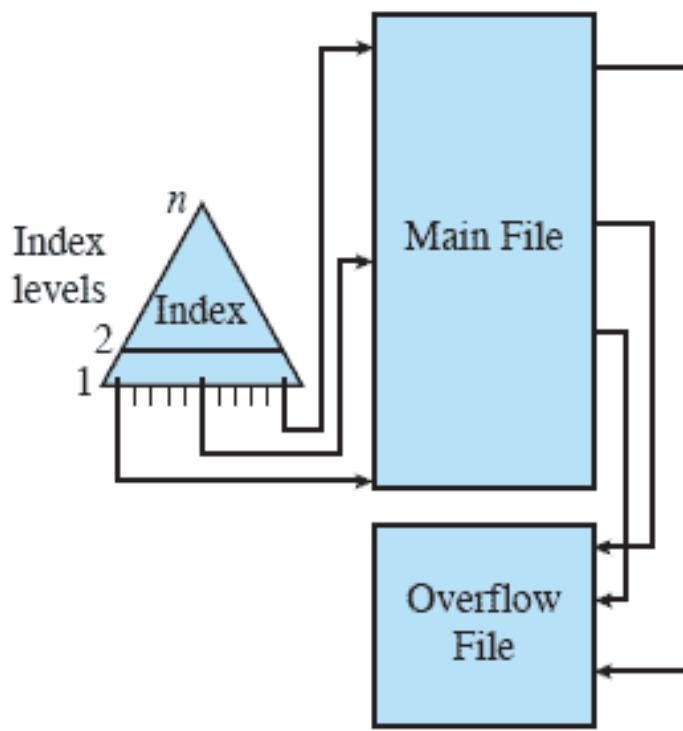
◆ Indeksna datoteka

- Višestruki indeksi za svako polje po kome se vrši pretraživanje
- Sekvencijalnost ne postoji, slogovima se pristupa isključivo putem indeksa
- Indeks može biti iscpnri koji sadrži jedan ulaz za svaki slog u datoteci, ili delimični koji ukazuje na grupu slogova u okviru kojih postoji slog sa traženom vrednošću polja
- Koriste se u aplikacijama u kojima je zahteva brz pristup podacima

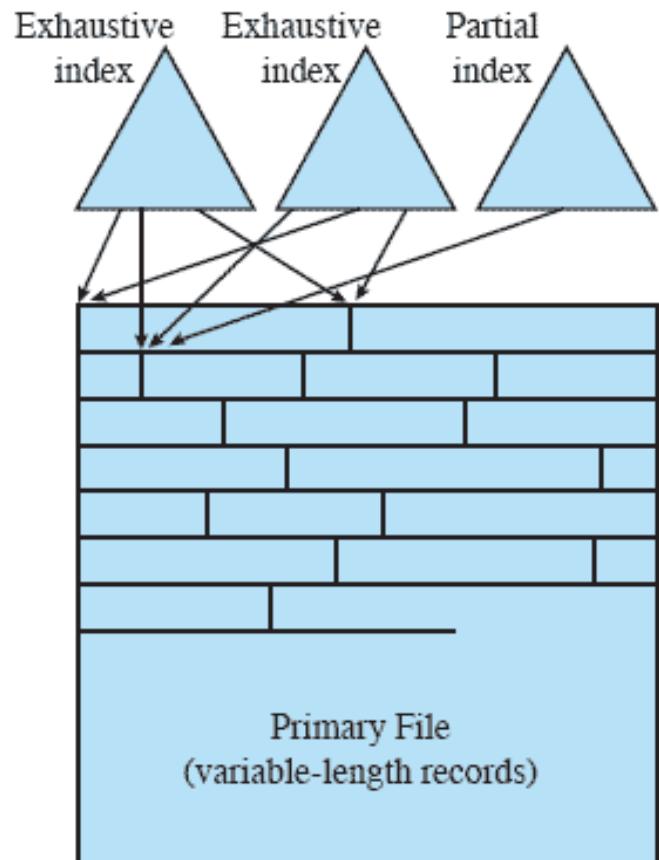
◆ Direktna (random, heširana) datoteka

- Zasnivaju se na karakteristikama diska kojima se omogućava direktni pristup bilo kom bloku sa poznatom adresom
- Bajtovi/slogovi se čitaju ili zapisuju u bilo kom poretku
- Svaki slog ima ključ, a direktan pristup je omogućen na osnovu heširanja po vrednosti ključa
- Bitno za sisteme baza podataka
- Pogodan za aplikacije koje zahtevaju brz pristup slogovima fiksne dužine i formata

Indeksno-sekvencijalna i indeksna datoteka



(c) Indexed Sequential File



(d) Indexed File

Performanse

◆ Ocene performansi 5 osnovnih organizacija datoteka

File Method	Space Attributes		Update Record Size		Retrieval		
	Variable	Fixed	Equal	Greater	Single record	Subset	Exhaustive
Pile	A	B	A	E	E	D	B
Sequential	F	A	D	F	F	D	A
Indexed sequential	F	B	B	D	B	D	B
Indexed	B	C	C	C	A	B	D
Hashed	F	B	B	F	B	F	E

A = Excellent, well suited to this purpose $\approx O(r)$
B = Good $\approx O(o \times r)$
C = Adequate $\approx O(r \log n)$
D = Requires some extra effort $\approx O(n)$
E = Possible with extreme effort $\approx O(r \times n)$
F = Not reasonable for this purpose $\approx O(n^{>1})$

where

r = size of the result

o = number of records that overflow

n = number of records in file



Operacije nad datotekama (1)

1. Kreiranje (*Create*)
 2. Brisanje (*Delete*)
 3. Otvaranje (*Open*)
 4. Zatvaranje (*Close*)
 5. Čitanje (*Read*)
 6. Upis (*Write*)
 7. Dodavanje (*Append*)
 7. Traženje (*Seek*)
 8. Uzimanje atributa (*Get attributes*)
 9. Postavljanje atributa (*Set Attributes*)
 10. Preimenovanje (*Rename*)
- Neke operacije može direktno pozivati krajnji korisnik - *komande*
 - Mogu se pozivati iz programa - *sistemski pozivi*
 - Postoje i drugi sistemske pozive za rad sa datotekama
 - Detalji u Praktikumu



Operacije nad datotekama (2)

● Kreiranje

- Datoteci se dodeljuje ime i neki atributi
- Datoteka je bez podataka
- Evidentira se u sistemske tablice

● Brisanje

- Oslobađa se prostor koji datoteka zauzima
- Uklanja se iz sistemskih tablica

● Otvaranje

- Pre upotrebe proces mora otvoriti datoteku
- Omogućava se sistemu da pristupa atributima datoteke
- Kreira se upravljački blok procesa (File Control Block - FCB)- struktura podataka u memoriji preko koje sistem brže pristupa datoteci
- OS ograničava broj otvorenih datoteka

● Zatvaranje

- Datoteka se zatvara kada više nije potrebna
- Oslobađa se prostor u internim tablicama



Operacije nad datotekama (3)

● Čitanje

- Čitaju se podaci iz datoteke, obično od tekuće pozicije
- Specificira se koliko bajtova/slogova treba pročitati i gde smestiti pročitane podatke

● Upis

- U datoteku se upisuju podaci, obično od tekuće pozicije
- Specificira se koliko bajtova/slogova treba zapisati i podaci koje treba zapisati

● Dodavanje

- Vrsta upisa, ali na kraj datoteke

● Traženje (*seek*)

- Kod random datoteka specificira se pozicija odakle će početi upis ili čitanje

● Pribavljanje atributa

- Čitaju se atributi datoteke

● Postavljanje atributa

- Postavljaju se atributi za koje je zadužen vlasnik
- Npr. zaštita

● Preimenovanje

- Menja se tekuće ime datoteke



Direktorijum (adresar, folder)

- ❖ **Direktorijum** je datoteka čiji je vlasnik OS i koji služi za organizovanje datoteka u logičke celine
- ❖ Sadrži informacije o datotekama
 - ❖ Atribute
 - ❖ Lokacija
 - ❖ Vlasinštvo
 - ❖ Ostali atributi
- ❖ Obezbeđuje preslikavanje između imena datoteke koje je poznato korisnicima i aplikacijama i same datoteke koja je smeštena na nekom periferijskom uređaju (disku, CD, DVD, USB Flash, ...)
- ❖ Direktorijumi mogu biti različito organizovani
 - ❖ U 1 nivou
 - ❖ U 2 nivoa
 - ❖ U obliku stabla
 - ❖ U obliku grafa



Informacije koje sadrži direktorijum

● Osnovne informacije

- **Ime datoteke** – jedinstveno ime u okviru direktorijuma, izabrano od strane kreatora (korisnika, programa)
- **Tip datoteke** – tekstualna, binarna, izvršna, itd.
- **Organizacija datoteke** – u sistemima koji podržavaju različite organizacije

● Adresne informacije

- **Volumen** – označava uređaj na kome je smeštena datoteka
- **Početna adresa** – početna adresa datoteke na sekundarnoj memoriji (npr. cilindar, staza, broj bloka na disku)
- **Veličina datoteke** – trenutna veličina datoteke u bajtovima ili blokovima
- **Maksimalna veličina datoteke**

● Informacija o kontroli pristupa

- **Vlasnik datoteke** – Korisnik koji ima pravo upravljanja datotekom; dodeljuje/oduzima prava ostalim korisnicima
- **Informacije o pristupu** – najjednostavnije korisničko ime i lozinka svakog autorizovanog korisnika datoteke
- **Dozvoljene akcije** – čitanje, upis, izvršavanje, menjanje, itd.

Upravljanje datotekama

Operativni sistemi



Informacije koje sadrži direktorijum (nastavak)

Informacije o korišćenju

- Datum kreiranja
- Identitet kreatora datoteke (obično, ali ne i obavezno tekući vlasnik)
- Datum poslednjeg čitanja datoteke
- Identitet poslednjeg čitaoca datoteke
- Datum poslednjeg modifikovanja datoteke (upis, brisanje, ažuriranje)
- Identitet korisnika koji je izvršio poslednju modifikaciju datoteke
- Datum poslednjeg backup-a datoteke
- Tekuće korišćenje – informacije o tekućim aktivnostima nad datotekom, procesima koji su otvorili tu datoteku, da li je datoteka modifikovana u memoriji, ali još ne i na disku, itd.



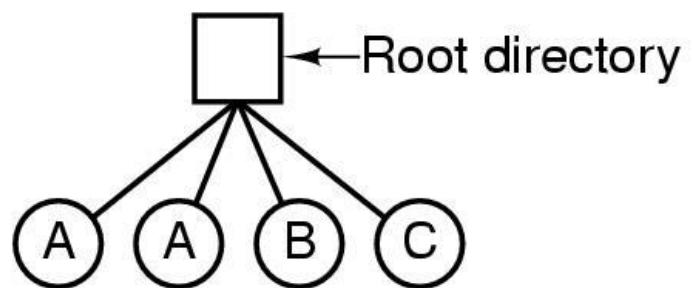
Operacije nad direktorijumima

Tipovi operacija nad direktorijumima

- Traženje
- Kreiranje datoteke
- Brisanje datoteke
- Listanje sadržaja direktorijuma
- Ažuriranje direktorijuma

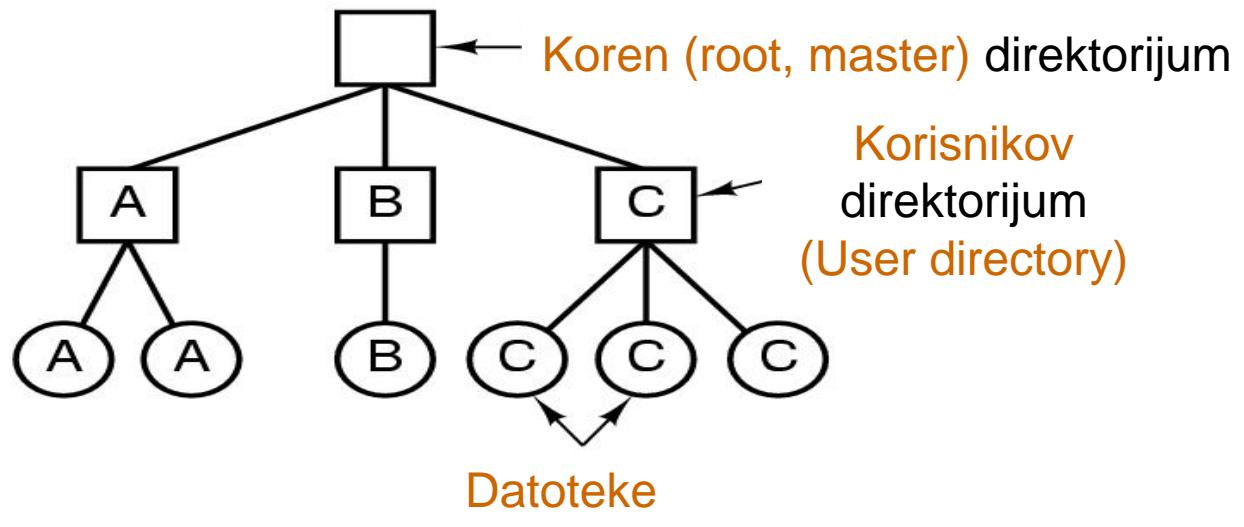
Direktorijum u 1 nivou

- Lista ulaza (*entry*), po jedan za svaku datoteku
- Sekvencijalna datoteka gde se ime datoteke koristi kao ključ
- Ne obezbeđuje pomoć u organizovanju datoteka
- Korisnik mora voditi računa da sve datoteke imaju različita imena



Direktorijum u 2 nivoa

- Po jedan direktorijum za svakog korisnika plus glavni (master) direktorijum
- Master direktorijum sadrži za svakog korisnika
 - adresu i informacije za kontrolu pristupa
- Svaki korisnički direktorijum je prosta lista datoteka tog korisnika
- Još uvek ne obezbeđuje pomoć u organizovanju datoteka



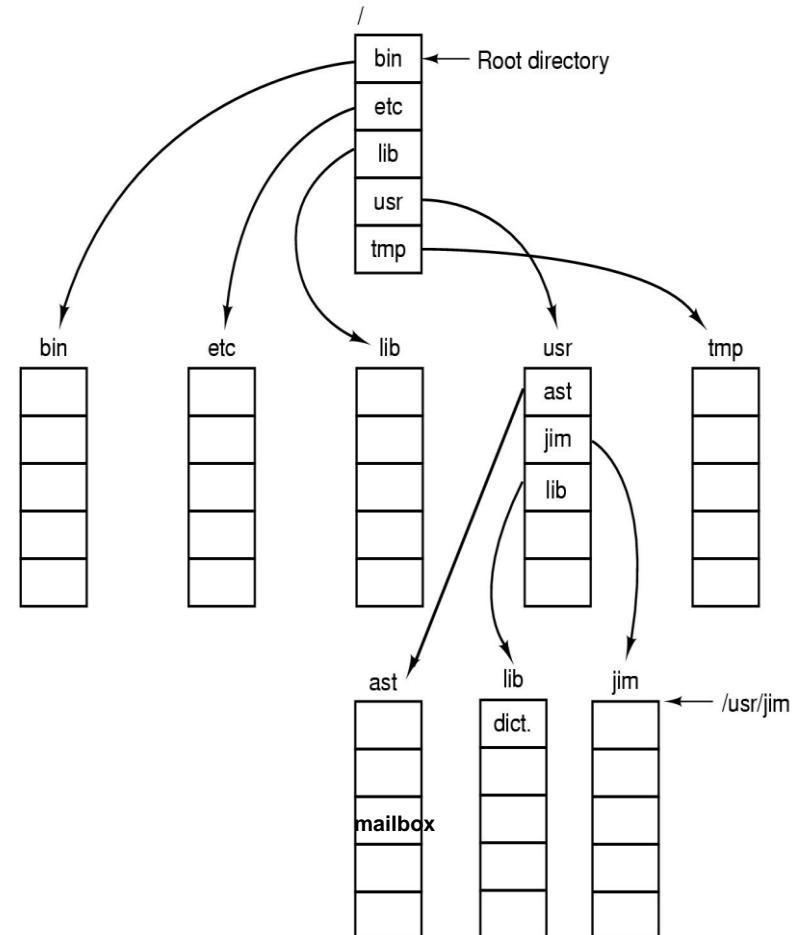


Direktorijumi strukture stabla ili hijerarhijski direktorijumi

- ❖ *Master direktorijum (koren)* sa korisničkim direktorijumima ispod njega
- ❖ Svaki korisnički direktorijum može imati kao ulaze poddirektorijume i datoteke
- ❖ Datoteke se mogu locirati praćenjem puta od korena direktorijuma, preko poddirektorijuma do datoteke
 - To je *ime puta (path name)* datoteke
- ❖ Može postojati više datoteka sa istim imenom sve dok oni imaju različita imena puta
- ❖ Tekući direktorijum je *radni direktorijum (working)*
- ❖ Datoteke se referenciraju relativno u odnosu na radni direktorijum
 - To je *relativno ime puta*

Ime puta

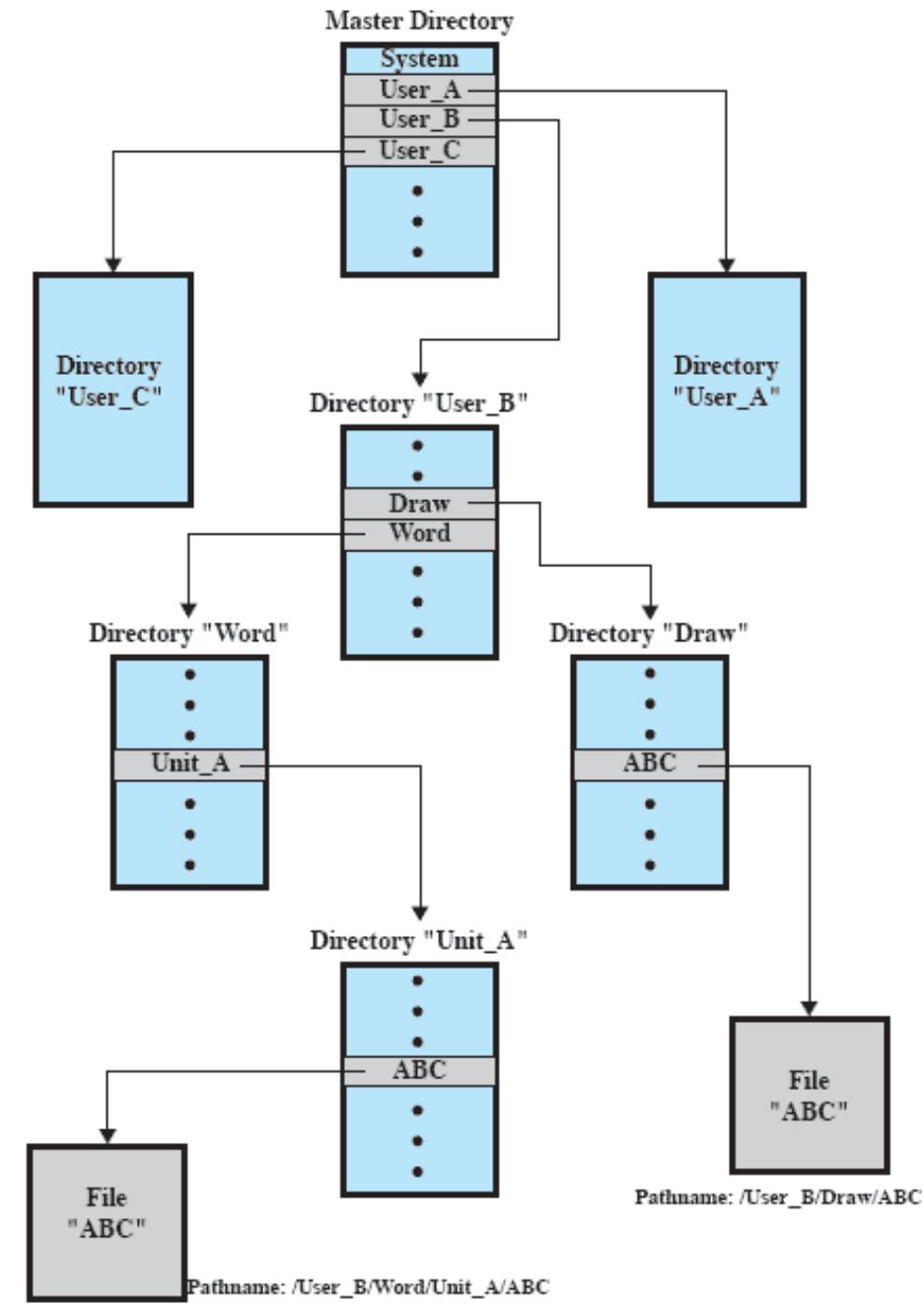
- Ime puta
 - absolutno
 - UNIX */usr/ast/mailbox*
 - Windows *\usr\ast\mailbox*
 - MULTICS *>usr>ast>mailbox*
 - relativno
 - UNIX *ast/mailbox*
 - Windows *ast\mailbox*
 - MULTICS *>usr>ast>mailbox*
 - *Apsolutno ime puta* uvek počinje od korena
 - *Relativno ime puta* počinje od *radnog (tekućeg)* direktorijuma
 - Svaki proces ima sopstveni radni direktorijum
 - U svakom direktorijumu postoje dva specijalna ulaza
 - . tekući direktorijum
 - .. njegov roditeljski direktorijum



Stablo direktorijuma u UNIX-u



Primer direktorijuma strukture stabla





Operacije direktorijuma (1)

1. Kreiranje (*Create*)
2. Brisanje (*Delete*)
3. Otvaranje direktorijuma (*OpenDir*)
4. Zatvaranje direktorijuma (*CloseDir*)
5. Čitanje direktorijuma (*ReadDir*)
6. Preimenovanje (*Rename*)
6. Povezivanje (*Link*)
7. Razvezivanje (*Unlink*)

- Korisnik može direktno pozivati ove funkcije – *komande*
- Operacije se mogu pozivati iz programa - *sistemski pozivi*
- Postoje i drugi sistemski pozivi za rad sa direktorijumima (*Praktikum*)



Operacije nad direktorijumima (2)

• Kreiranje (**Create**)

- Kreira direktorijum
- direktorijum ima samo . i ..
- Unosi se u sistemske tablice

• Brisanje (**Delete**)

- Briše direktorijum
- Može se brisati samo prazan direktorijum
- Oslobađa se prostor koji direktorijum drži
- Uklanja se iz sistemskih tablica

• Otvaranje (**Opendir**)

- Pre upotrebe proces mora otvoriti direktorijum
- Omogućava se sistemu da pristupa atributima direktorijuma
- OS ograničava broj otvorenih direktorijuma

• Zatvaranje (**Closedir**)

- direktorijum se zatvara kada više nije potreban
- Oslobađa se prostor u internim tablicama

• Čitanje (**Readdir**)

- Čita se jedan ulaz iz direktorijuma

• Preimenovanje (**Rename**)

- Menja se tekuće ime direktorijuma

• Povezivanje (**Link**)

- Omogućava da se isti datoteka javlja u više direktorijuma

• Razvezivanje (**Unlink**)

- Uklanja se link ulaz direktorijuma

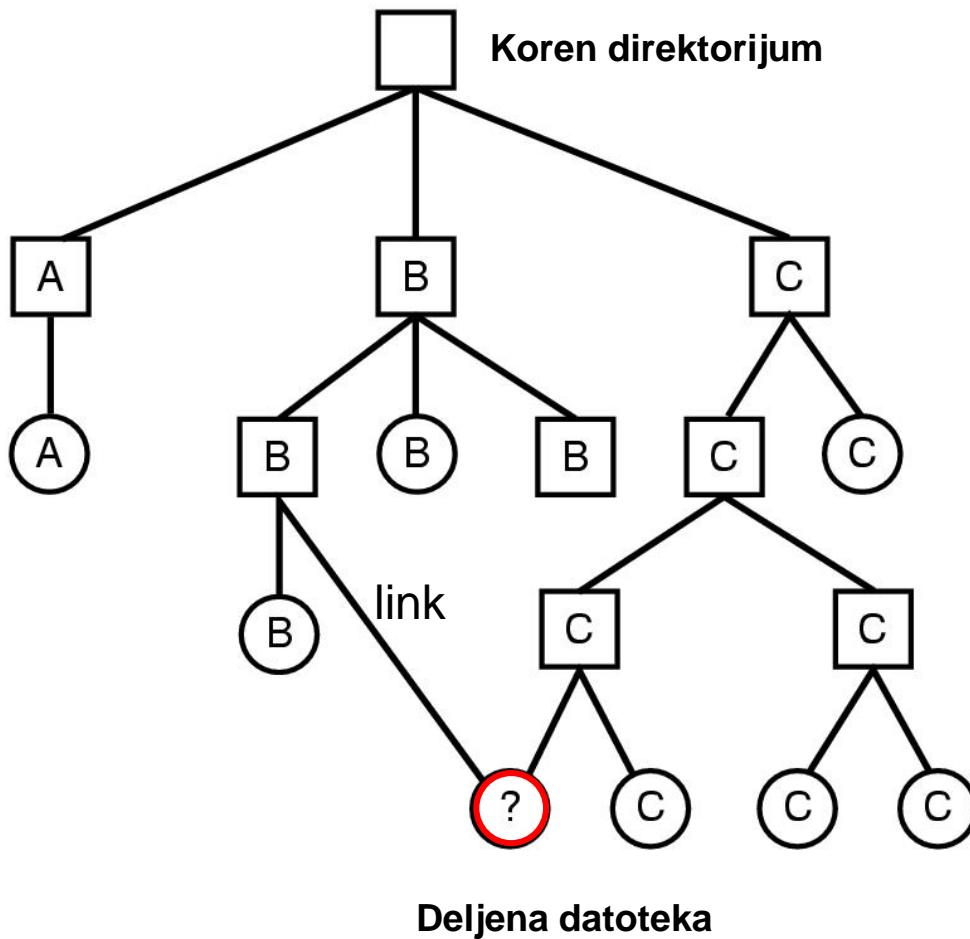


Deljenje datoteka

File Sharing

- U višekorisničkom sistemu, dozvoljeno je da se datoteke dele među korisnicima
 - Da se zajednički koriste
- Dva su nova problema
 - Prava pristupa (*Access rights*)
 - Upravljanje simultanim pristupom

Primer file system-a koji sadrži deljenu datoteku





Implementacija deljenih datoteka

- Deljena datoteka se može implementirati na više načina. Niže su prikazane dve metode:

Metoda 1 (hard link)

- Podaci o blokovima koji su dodeljeni datoteci se ne nalaze u direktorijumu, već u posebnoj strukturi uz datoteku, a u direktorijumu je pokazivač na tu strukturu
- Za deljenu datoteku se u direktorijumu korisnika kome je dozvoljen pristup formira klasičan ulaz za datoteku, a u originalnom ulazu se inkrementira *brojač linkova*
 - moguće kod sistema sa i-čvorom (npr. Unix)

Metoda 2 (simboličko linkovanje)

- U direktorijumu korisnika kome je dozvoljen pristup se kreira nova datoteka tipa LINK
 - Nova datoteka sadrži samo put do datoteke koja je linkovana
 - Kada se čita takva datoteka, OS prepoznaje da je tipa LINK, uzima put do datoteke i čita originalnu datoteku

Prava pristupa (1)

- ➊ Nije dozvoljeno znati da datoteka postoji (*None*)
 - Korisnik ne sme znati da datoteka postoji
 - Korisniku nije dopušteno da čita direktorijum koji sadrži datoteku
- ➋ Dozvoljeno je znati da datoteka postoji, ali je za pristup potrebna dozvola (*Knowledge*)
 - Korisnik jedino može utvrditi da datoteka postoji i ko je vlasnik
 - Mora od vlasnika tražiti dozvolu pristupa



Prava pristupa (2)

● Izvršenje (*Execution*)

- Korisnik može loadovati i izvršavati program, ali ga ne može kopirati

● Čitanje (*Reading*)

- Korisnik može čitati datoteku u bilo koje svrhe, uključujući kopiranje i izvršenje

● Dodavanje (*Appending*)

- Korisnik može dodavati podatke datoteci, obično na kraj datoteke, ali ne može modifikovati ili brisati sadržaj datoteke

Prava pristupa (3)

● Ažuriranje (*Updating*)

- Korisnik može modifikovati, brisati i dodavati podatke datoteci. Ovo uključuje kreiranje datoteke, ponovni upis i brisanje svih ili dela podataka

● Izmena zaštite (*Changing protection*)

- Korisnik može promeniti pristupna prava koja je dodelio drugim korisnicima

● Brisanje (*Deletion*)

- Korisnik može obrisati datoteku

Prava pristupa (4)

❖ Vlasnici

- ❖ Imaju sva prethodno navedena prava pristupa
- ❖ Mogu prenositi prava drugim korisnicima koristeći sledeće klase korisnika
 - Specifični korisnik (*specific user*)
 - Pojedini korisnici koji imaju svoj **userID**
 - Grupe korisnika (*user groups*)
 - Skup korisnika koji imaju zajednički identifikator **groupID**
 - Svi (*public*)

Istovremeni pristup

- ➊ Ako je dozvoljeno dodavanje ili ažuriranje datoteke većem broju korisnika tada OS mora obezbediti nesmetani simultani pristup
 - ▢ Prosto rešenje: dozvoliti korisniku da zaključa celu datoteku kada je ažurira
 - ▢ Finije rešenje: dozvoliti korisniku da zaključa slog koji ažurira
 - ▢ Svodi se na sinhronizacioni problem čitaoci-pisci
- ➋ Za deljene datoteke treba rešiti uzajamno isključivanje i samrtni zagrljaj



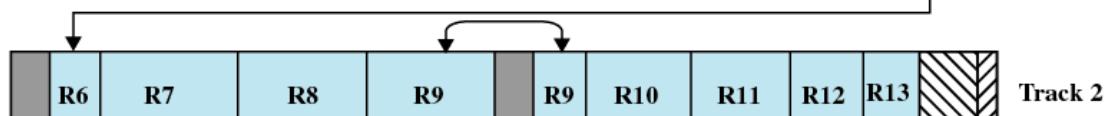
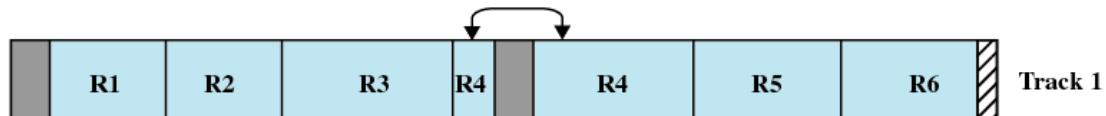
Blokiranje slogova

- ❖ Organizovanje slogova u blok radi U/I
- ❖ Tri metode blokiranja
 - ❖ Fiksno blokiranje – u blok je smešten celi broj slogova fiksne veličine
 - Interna fragmentacija
 - ❖ Blokiranje varijabilne dužine sa prebacivanjem – slogovi varijabilne dužine su smešteni u blok sa mogućim prebacivanjem ostatka poslednjeg sloga u bloku u sledeći blok
 - ❖ Blokiranje varijabilne dužine bez prebacivanja - slogovi su varijabilne dužine, ali nije moguće prebacivanje slogova između blokova

Blokiranje slogova - primer



Fixed Blocking



Variable Blocking: Spanned

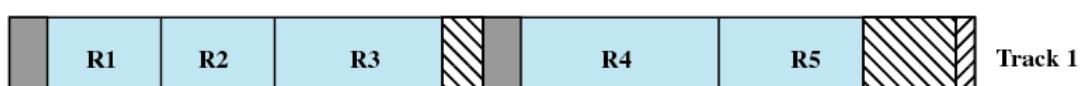
Data

Gaps due to hardware design

Waste due to block fit to track size

Waste due to record fit to block size

Waste due to block size constraint from fixed record size



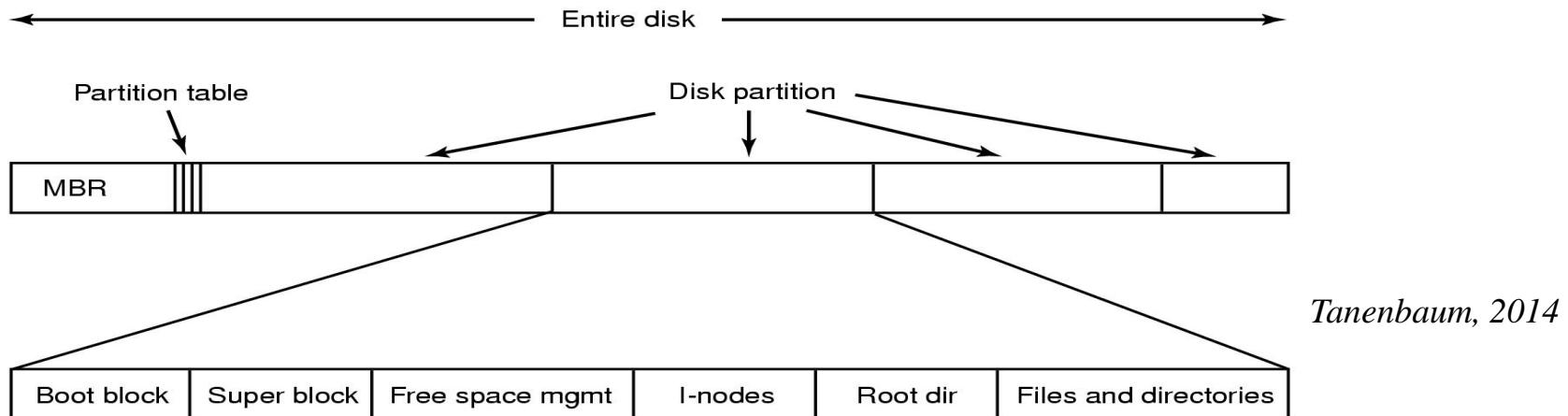
Variable Blocking: Unspanned



Strukture podataka file system-a

- File system na disku i u memoriji održava određene strukture podataka neophodne za podršku upravljanju datotekama
- **Na disku** se čuvaju informacije za inicijalno punjenje (bootovanje) OS-a, ukupan broj blokova, broj i lokacije slobodnih blokova, struktura direktorijuma, kao i same datoteke i direktorijumi
- **U memoriji** se čuvaju informacije neophodne za upravljanje file system-om i za poboljšanje performansi (keširanje)

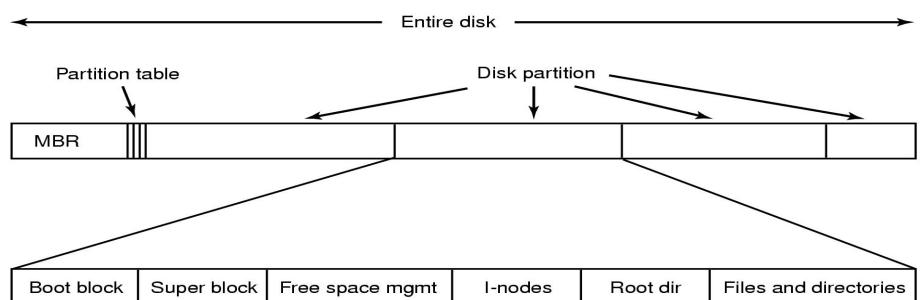
Struktura diska



- ❖ File system se memoriše na disku
- ❖ Na slici je prikazana moguća struktura (*layout*) diska
 - svaki OS ima sopstveni *layout*
- ❖ Mnogi diskovi su podeljeni na 1 ili više particija
- ❖ Sektor 0 diska se zove *MBR* (*Master Boot Record*)
 - koristi se za *boot-ovanje* sistema (inicijalno load-ovanje operativnog sistema)
- ❖ Na kraju MBR-a se nalazi *Tabela particija* (*Partition Table*)
 - čuva adresu početka i kraja svake particije
- ❖ Jedna particija je označena kao aktivna – sadrži OS

Struktura particije

- Boot block (Unix), Partition boot sector (NTFS)
 - Sadrži punilac (*bootstrep loader*)
- Super blok (Unix), Master File Table (NTFS)
 - Sadrži sve ključne parametre o file system-u
 - Broj blokova u particiji
 - Veličina bloka
 - Broj slobodnih blokova
 - Pokazivač na listu slobodnih blokova (bitmapa slobodnih blokova)
 - Slobodne FCB-e
 - Pokazivač na listu FCB
 - Učitava se u memoriju kada se računar boot-uje ili kada se prvi put pristupa file system-u
- Informacija o slobodnom prostoru za smeštanje podataka
 - Bit mapa ili lista pokazivača (adresa) na slobodne blokove
- i-čvorovi (*i-node*) (Unix), FCB
 - Po jedan za svaku datoteku
 - Sadrži atributе datoteke
- Koren direktorijum (root)
- Sačuvane datoteke i direktorijumi





Strukture podataka file system-a u glavnoj memoriji

● Tabela particija u memoriji

- Sadrži informacije o svim mount-ovanim particijama

● Struktura direktorijuma u memoriji

- Sadrži informacije o direktorijumima koji su skoro korišćeni
- Pokazivač na Tabelu particija gde je direktorijum mount-ovan

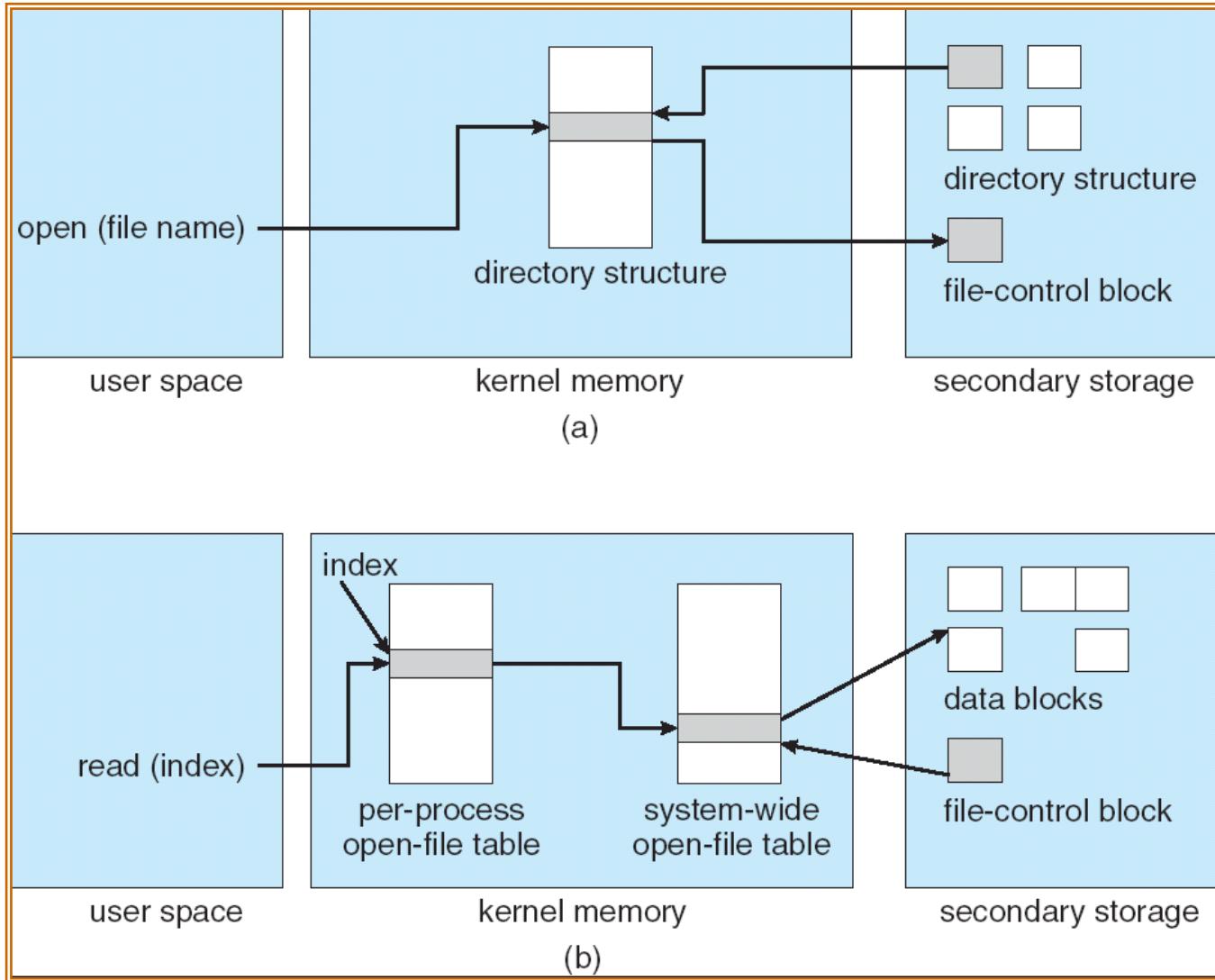
● Tabela otvorenih datoteka sistema (1 za ceo sistem)

- Sadrži kopije FCB-ova svih otvorenih datoteka, kao i druge sistemske informacije

● Tabela otvorenih datoteka procesa (po 1 za svaki proces)

- Sadrži pokazivač na odgovarajući ulaz Tabela otvorenih datoteka sistema, kao i druge informacije

Strukture podataka file system-a u glavnoj memoriji (2)



Silberschatz, 2012



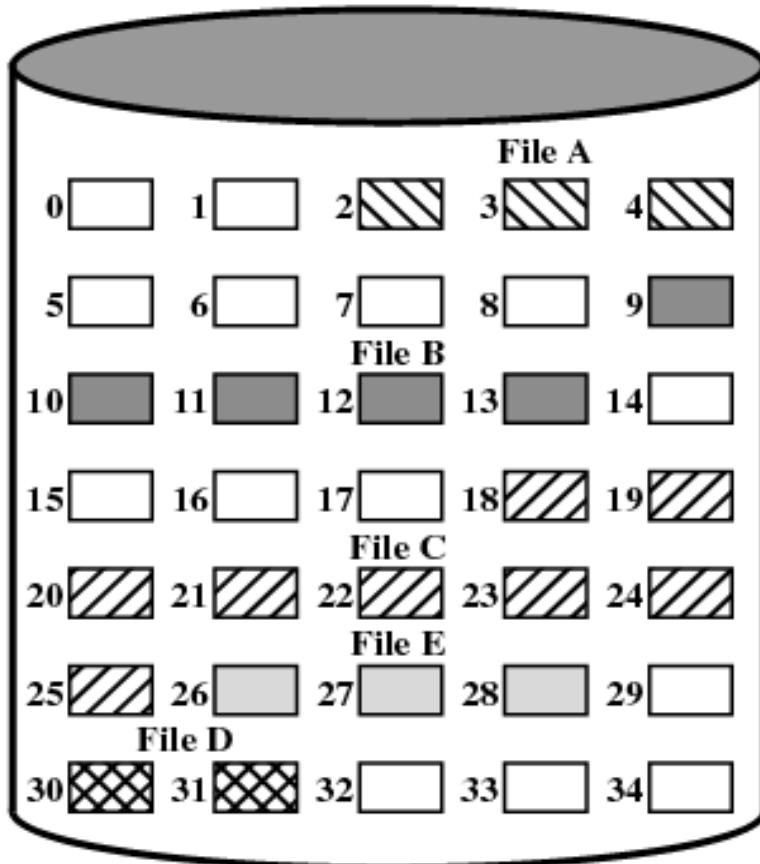
Upravljanje sekundarnom memorijom

- ➊ Datoteke se sastoje od kolekcije blokova i file system vodi evidenciju o datotekama na sekundarnoj memoriji
 - Alokacija (dodeljivanje) datoteke
 - Upravljanje slobodnim prostorom na sekundarnoj memoriji
- ➋ **Dodeljivanje datoteke** – dodela blokova datoteci na osnovu određene strategije i evidencija dodeljenih blokova
 - **Dodeljivanje unapred** zahteva da se maksimalna veličina datoteke specificira u vreme kreiranja datoteke, dok **dinamičko dodeljivanje** dodeljuje datoteci blokove po potrebi
 - Datoteci se može dodeliti jedinstvena **kontinualna grupa blokova**, **više grupa kontinualnih blokova** različite veličine, ili **pojedinačni blokovi**. Ukoliko se dodeljivanje datoteke obavlja u varijabilnim grupama kontinualnih blokova, primenjuju se strategije prvo, najbolje i sledeće poklapanje (*first, best, next fit*)
- ➋ Za dodeljivanje datoteke se koriste metode:
 - **Kontinualna dodela**
 - **Nekontinualna dodela**
 - Ulančavanje blokova
 - Indeksiranje blokova

Kontinualna dodela

- Najprostija šema dodele (npr. IBM VM/CMS OS)
- Svakom fajlu se dodeljuje potreban broj susednih blokova na disku
- Prednosti:
 - Jednostavna evidencija dodeljenog prostora
 - U adresaru se za svaku datoteku pamti adresa prvog dodeljenog bloka i broj dodeljenih blokova (dužina)
 - Odlične performanse čitanja
 - Vreme traženja je minimalno
 - Potrebno je samo postaviti glavu diska na prvi blok
 - Nema pomeranja glave kod čitanja sledećeg bloka
 - Ceo fajl se može pročitati jednom operacijom
 - Jednostavan sekvencijalni i direktni pristup
 - Sekvencijalni – iz adresdirektorijumčita adresa prvog bloka, a zatim se nakon čitanja prvog bloka, adresa uvećava za 1, itd.
 - Direktni pristup – blok i datoteke koji počinje od bloka b se nalazi na adresi $b+i$
- Nedostaci
 - Eksterna fragmentacija
 - Datoteke ne mogu da rastu
 - Povremeno je potrebno vršiti kompakciju
- Primena: CD-ROM

Primer kontinualne dodele (1)

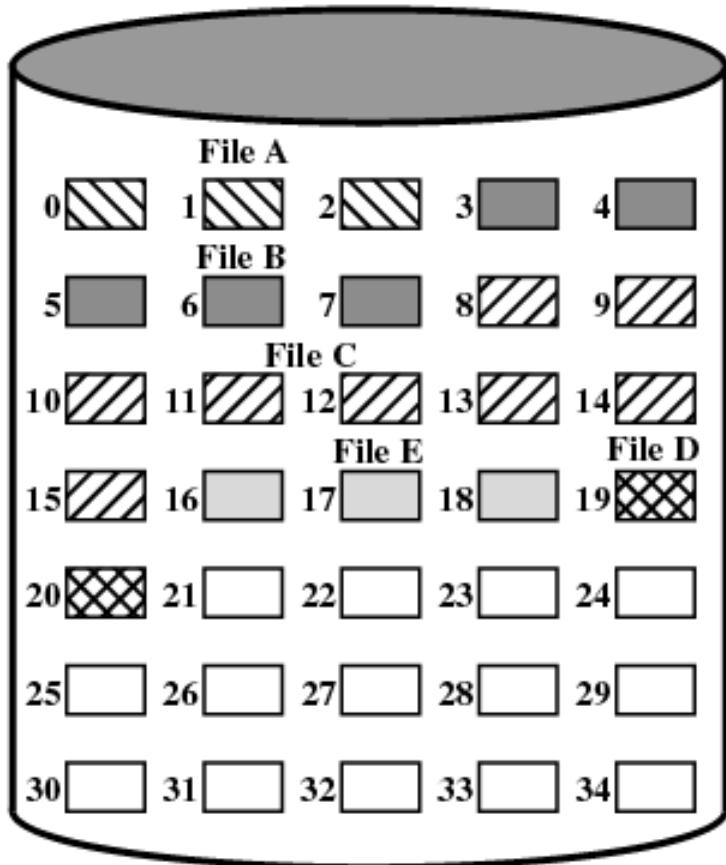


File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Primer kontinualne dodele (2)

- nakon kompakcije



File Allocation Table

File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

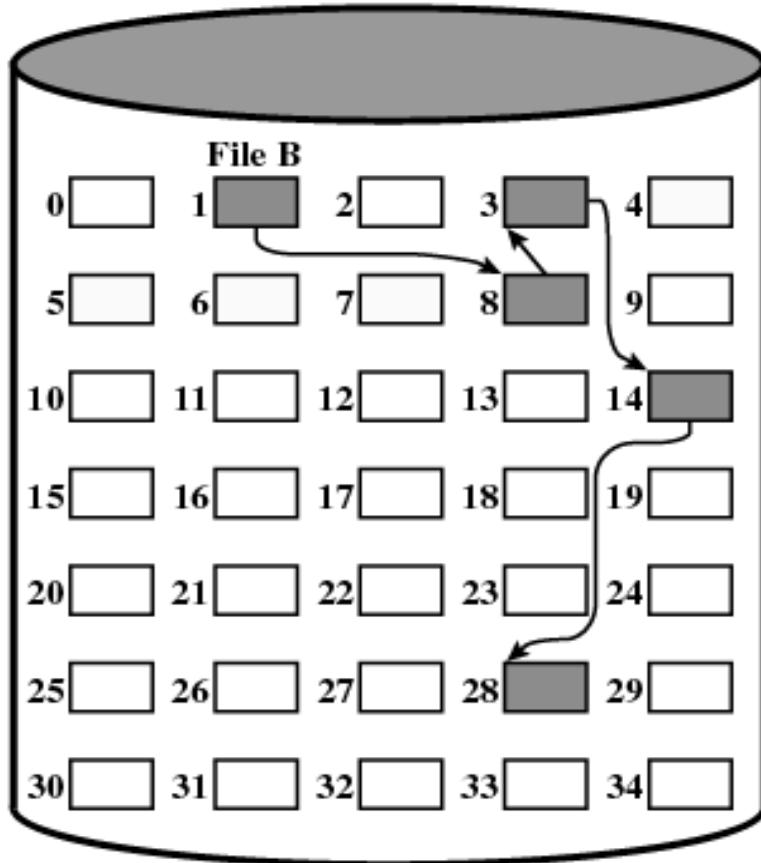
Dodela ulančavanjem (1)

- ❖ Blokovi dodeljeni datoteci su povezani u lančanu listu
- ❖ Prva reč svakog bloka se koristi za čuvanje adrese na sledeći blok datoteke
- ❖ Ostali deo bloka su podaci
- ❖ U direktorijumu se čuva samo adresa prvog bloka datoteke
- ❖ Svojstva:
 - Nema eksterne fragmentacije
 - Brz sekvencijalni pristup
 - Neefikasan random pristup: svodi se na sekvencijalni pristup, odnosno na obilazak lančane liste blokova startujući od prvog bloka

blok



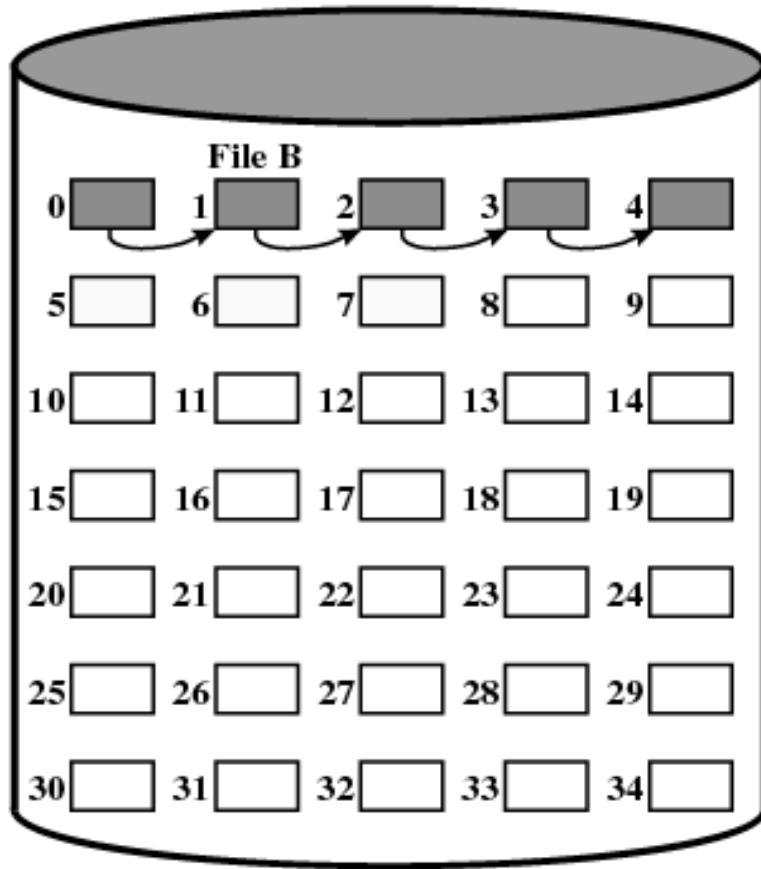
Primer dodele ulančavanjem (2)



File Name	Start Block	Length
...
File B	1	5
...

Primer dodele ulančavanjem (2)

- nakon integrisanja



File Allocation Table		
File Name	Start Block	Length
...
File B	0	5
...

Dodela ulančavanjem korišćenjem tabele u memoriji

Fizički blok

FAT

0	
1	
2	10
3	11
4	7
5	
6	3
7	2
8	
9	
10	12
11	14
12	-1
13	
14	-1
15	

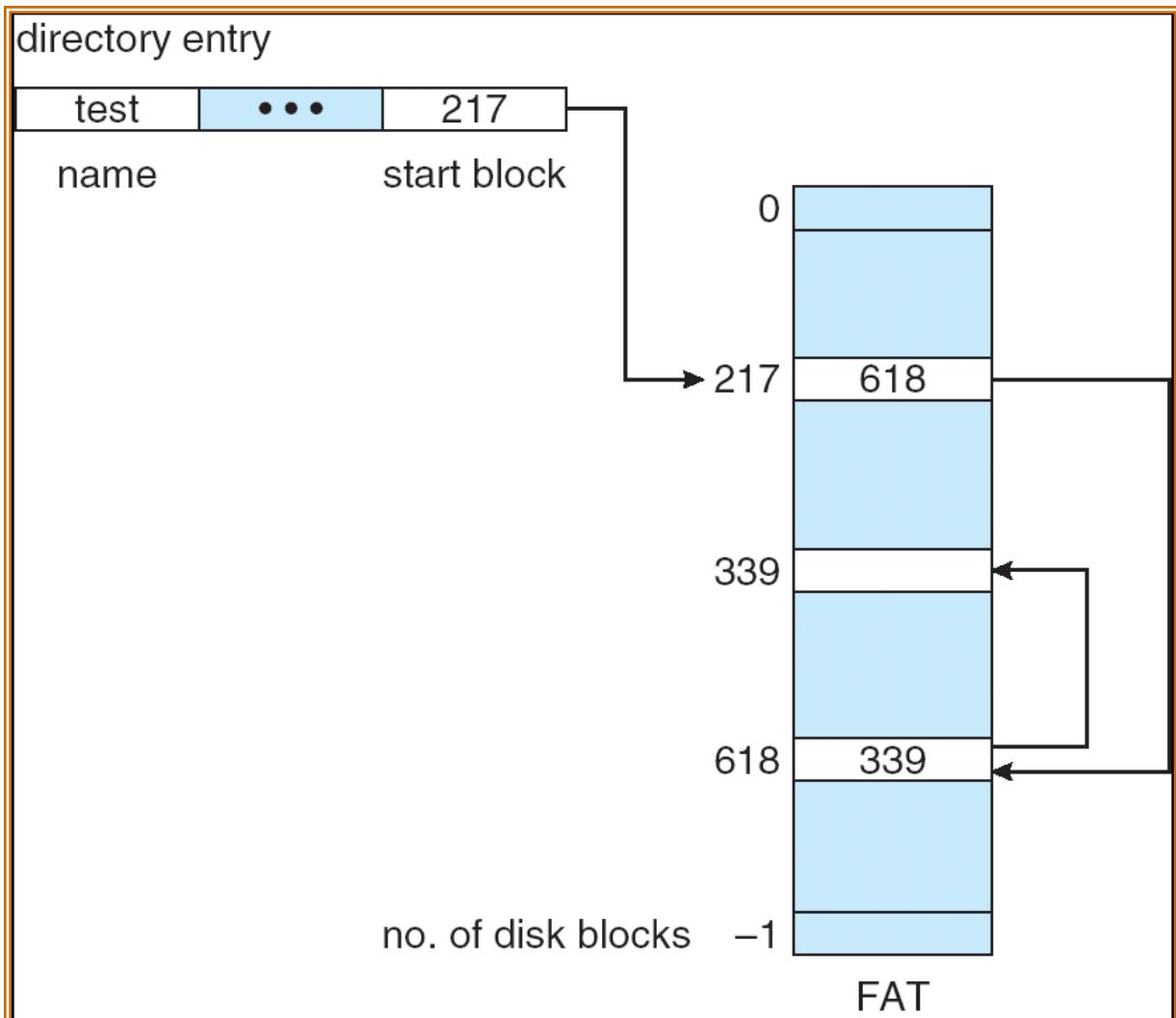
Fajl A počinje ovde

Fajl B počinje ovde

Blok koji se ne koristi

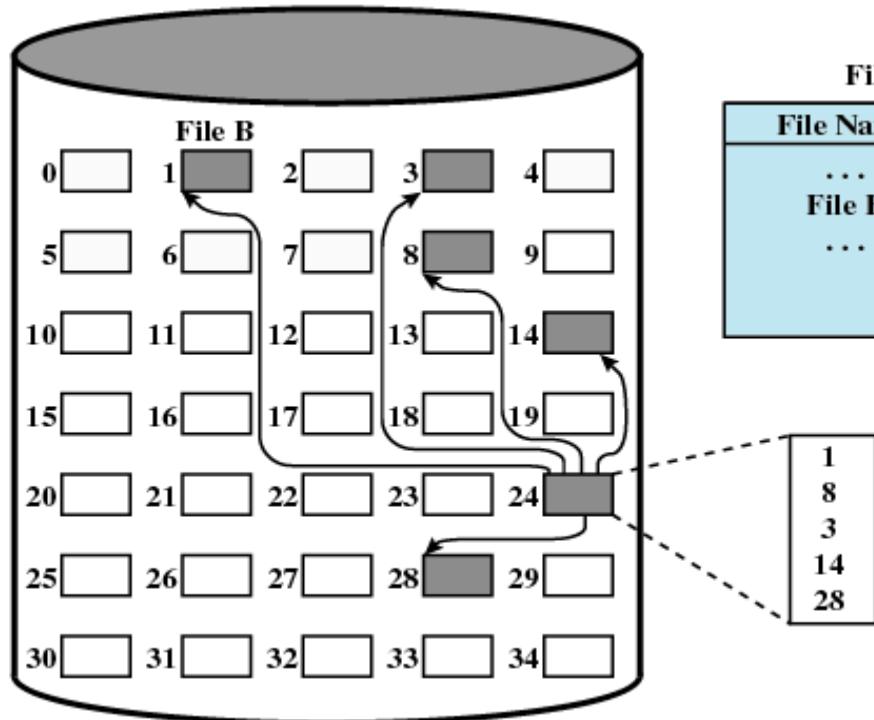
- Uvodi se **Tabela dodele datoteke (FAT - File Allocation Table)**
- U FAT-u se održava lančana lista dodele (primer: MS-DOS, Windows 95/98/Me, OS/2)
- U direktorijumu se čuva samo adresa prvog bloka datoteke
- Svojstva:
 - Ceo blok se koristi za podatke
 - Brži random pristup
- Nedostatak:
 - Cela tabela mora biti u memoriji
 - Za disk od 20GB sa veličinom bloka 1KB, FAT ima 20 miliona ulaza
 - Svaki ulaz minimalno 3B ili 4B
 - Za FAT neophodno 60 do 80 MB

FAT primer



Dodela indeksiranjem

- Svakoj datoteci se dodeljuje **indeksni blok** (*index block*)
- U *indeksnom bloku* su smeštene adrese blokova (grupa blokova) datoteke
- U okviru ulaza direktorijuma (*File Allocation Table*) registruje se adresa (broj) bloka koji predstavlja indeksni

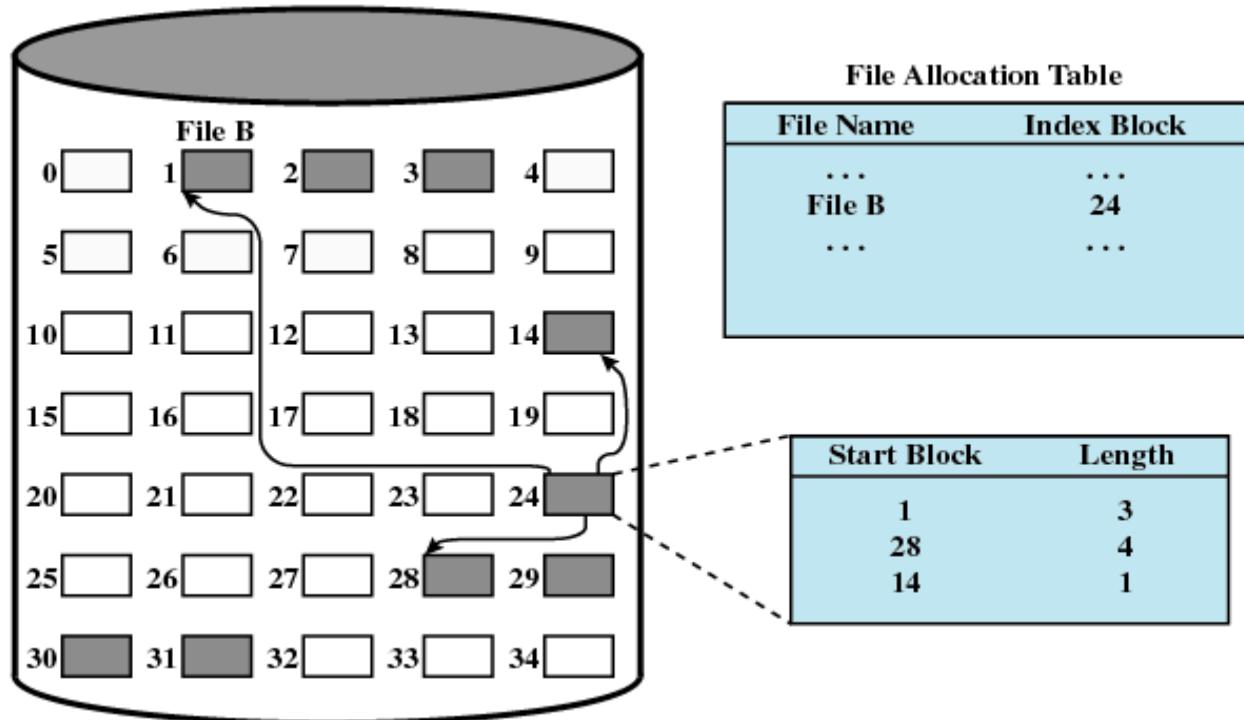


File Allocation Table	
File Name	Index Block
...	...
File B	24
...	...

Indeksni blok registruje sve blokove datoteke

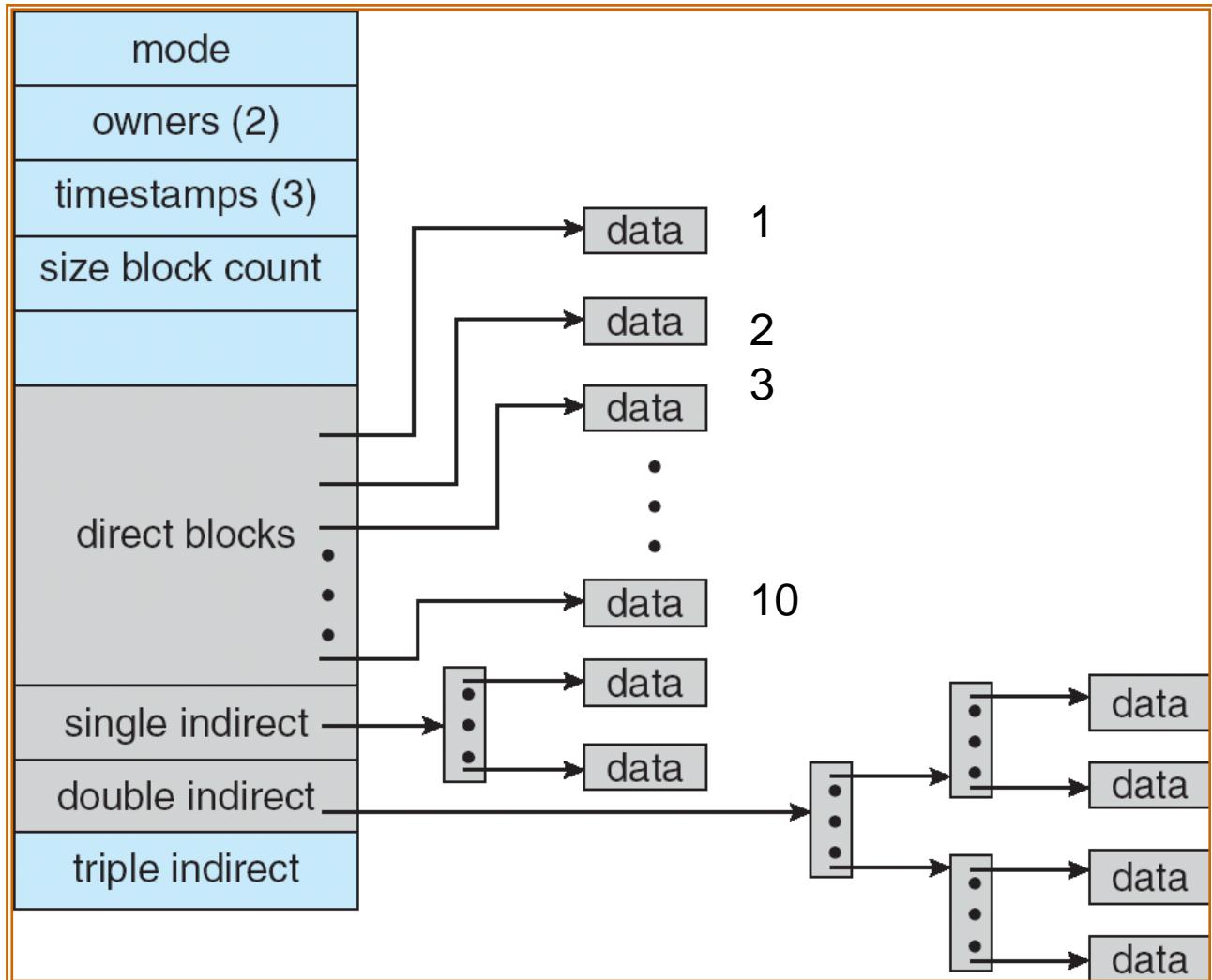
Varijanta dodele indeksiranjem

- Datoteci se dodeljuju blokovi u grupama blokova različite veličine, pa se u indeksnom bloku registruju te varijabilne grupe blokova datoteke



Dodela blokova u UNIX

i-čvor
(*i-node*)





Upravljanje slobodnim prostorom na disku

- ❖ Današnji OS koriste blok (klaster) kao jedinicu zauzimanja/oslobađanja
- ❖ Ostaje pitanje kolika je veličina bloka poželjna?
 - ❖ Umnožak veličine sektora
 - ❖ 2^n KB
- ❖ Primjenjene tehnike
 - ❖ Bit tabela (bit vektor)
 - ❖ Ulančavanje slobodnih blokova (delova sastavljenih od grupe sukcesivnih blokova)
 - ❖ Indeksiranje
 - ❖ Lista slobodnih blokova



Bit tabela (bit vektor)

- Bit vektor ili bit tabela
 - Svaki blok na disku je predstavljen 1 bitom
 - Traženje slobodnog prostora od n blokova se svodi na nalaženje sekvene od n nula u bit vektoru
 - Bit vektor se čuva u slobodnim blokovima na disku
 - U memoriju se prenosi blok po blok
 - Bit tabela zahteva ekstra prostor - primer:
 - veličina bloka = $512 \text{ B} = 2^9 \text{ B}$
 - veličina diska = $16 \text{ GB} = 2^{34} \text{ B}$
 - $n = 2^{34}/2^9 = 2^{25}$ bitova = 4 MB (veličina bit tabele)
 - Mnogi file system-i održavaju pomoćnu strukturu podataka koja na osnovu logičke podele bit vektora na podopsege, u toj strukturi registruje za svaki podopseg broj slobodnih blokova i maksimalni broj kontinualnih slobodnih blokova u podopsegu.



$$\text{bit}[i] = \begin{cases} 0 & \Rightarrow \text{blok}[i] \text{ slobodan} \\ 1 & \Rightarrow \text{blok}[i] \text{ zauzet} \end{cases}$$

Upravljanje datotekama

Operativni sistemi



Ulančavanje slobodnog prostora

● Lančana lista (slobodna lista)

- Ulančavanje svih slobodnih disk blokova ili grupa blokova
 - Pokazivač na prvi blok (grupu blokova) ove liste se drži na specijalnoj lokaciji na disku i kešira u memoriji
- Ovaj metod je pogodan za sve metode alokacije datoteka
- Nedostaci ove metode
 - Kontinualni prostor željene veličine se ne može lako naći,
 - Ukoliko se alociranje vrši u pojedinačnim blokovima neophodan je višestruki pristup disku da bi se alocirao odgovarajući broj blokova
 - Datoteci se dodeljuje prvi slobodan blok (grupa blokova) iz slobodne liste
 - Ukoliko je alociranje u grupama kontinualnih blokova, prostor diska bi bio fragmentiran, sa grupama blokova uglavnom veličine jednog bloka

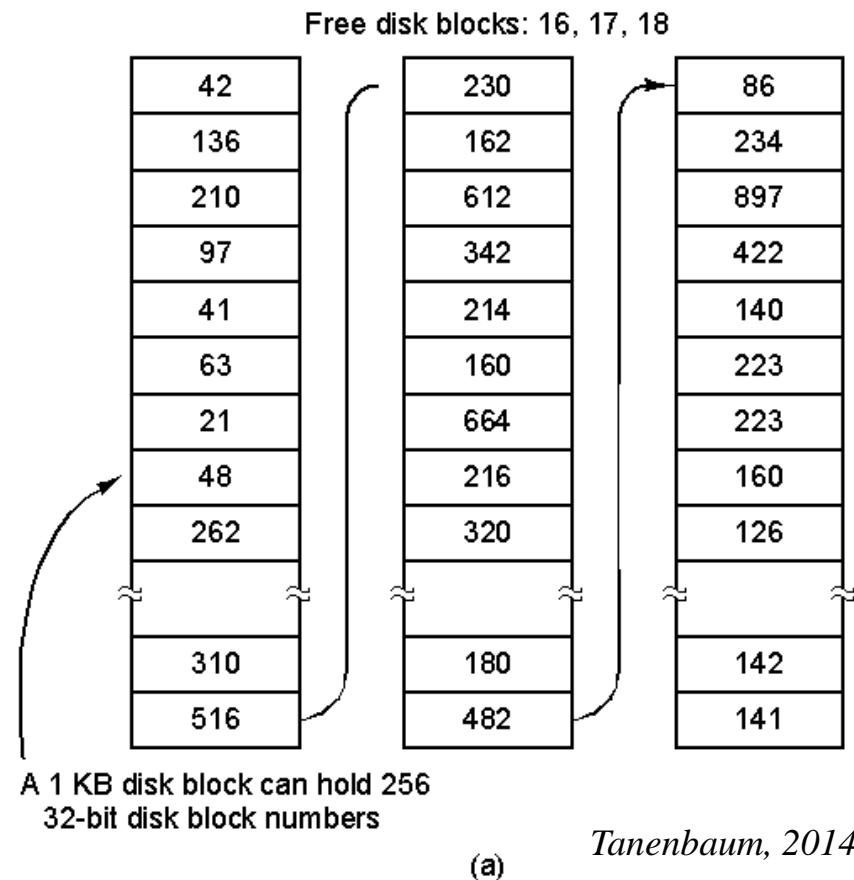
Indeksiranje i lista slobodnih blokova

Indeksiranje

- Slobodni prostor se tretira kao datoteka i slobodni blokovi (grupe blokova) se indeksiraju

Lista slobodnih blokova

- Adrese (brojevi) slobodnih blokova (veličine 3-4 B) se registruju u određenim blokovima na disku koji su ulančani
- Deo ove liste (jedan ili dva bloka) se drži u glavnoj memoriji i po potrebi snima na disk (kada se blok napuni) ili učitava sa diska (kada se slobodni blokovi dodele) po principu magacina ili FIFO reda



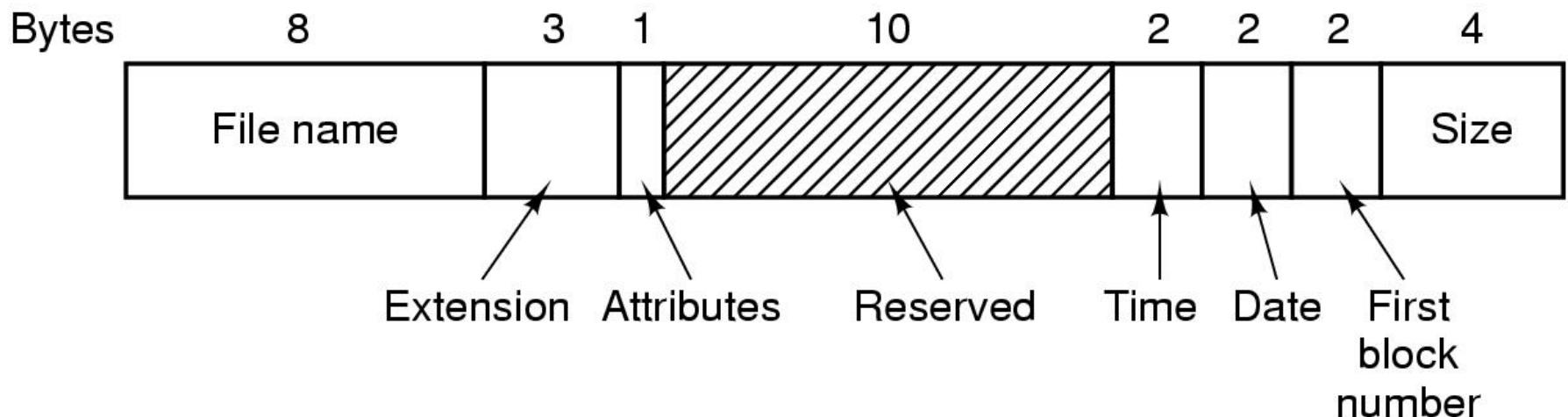


Primeri datotečnih sistema

- Windows (FAT i NTFS)
- Unix
- Linux

MS-DOS file system (1)

Stavka direktorijuma



- ❖ FAT file system dolazi u 3 verzije za MS-DOS
 - ❖ FAT-12: adresa bloka na disku 12 bitova
 - ❖ FAT-16: adresa bloka na disku 16 bitova
 - ❖ FAT-32: adresa dis bloka na disku ka 32 bitova
- ❖ Veličina bloka na disku je umnožak 512 B (može biti različita u svakoj particiji)

Tanenbaum, 2014

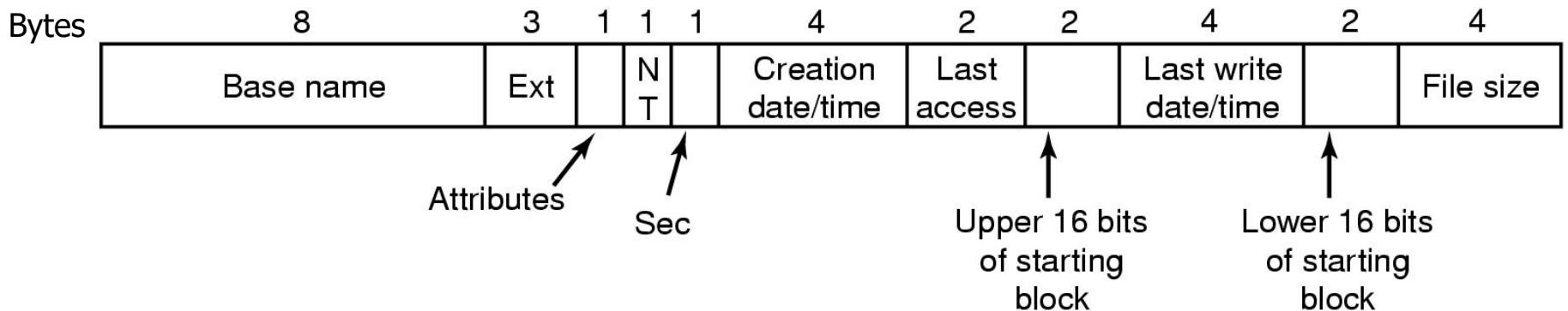


MS-DOS file system (2)

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

- ❖ Maksimalne veličine particija za različite veličine blokova
- ❖ Prazne stavke označavaju zabranjene kombinacije

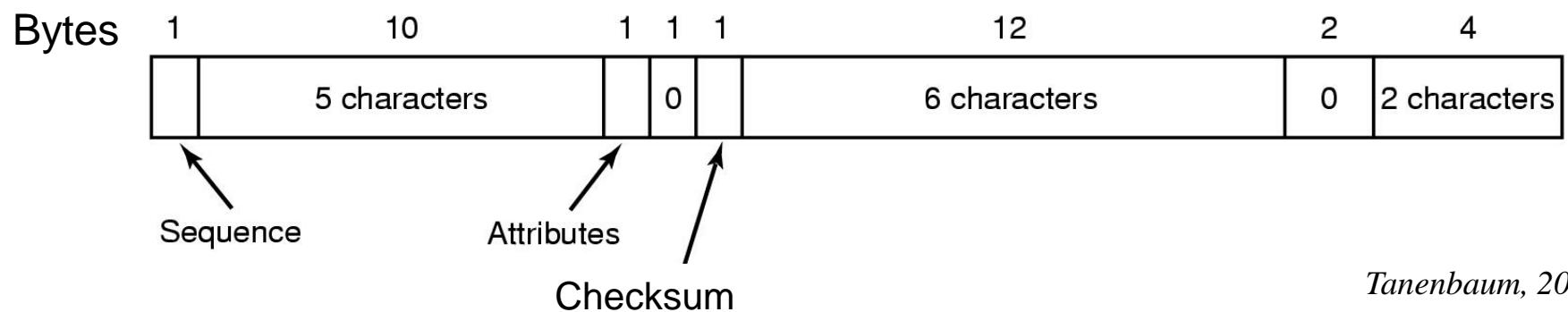
Windows 98 file system (1)



Tanenbaum, 2014

Proširen MS-DOS ulaz direktorijuma korišćen u Windows 98

Windows 98 file system (2)



Ulas direktorijuma za duga imena u Windows 98

Windows 98 file system (3)

68	d o g	A 0 C K						0	
3	o v e	A 0 C K	t h e l a					0	z y
2	w n f o	A 0 C K	x j u m p					0	s
1	T h e q	A 0 C K	u i c k b					0	r o
T H E Q U I ~ 1		A N T S	Creation time	Last acc	Upp	Last write		Low	Size
Bytes									

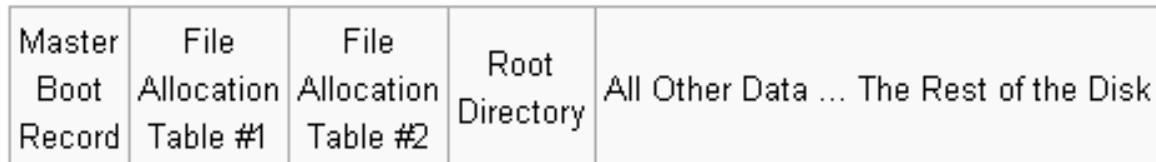
Tanenbaum, 2014

Primer kako se dugačko ime pamti u Windows 98



FAT

Format FAT diska

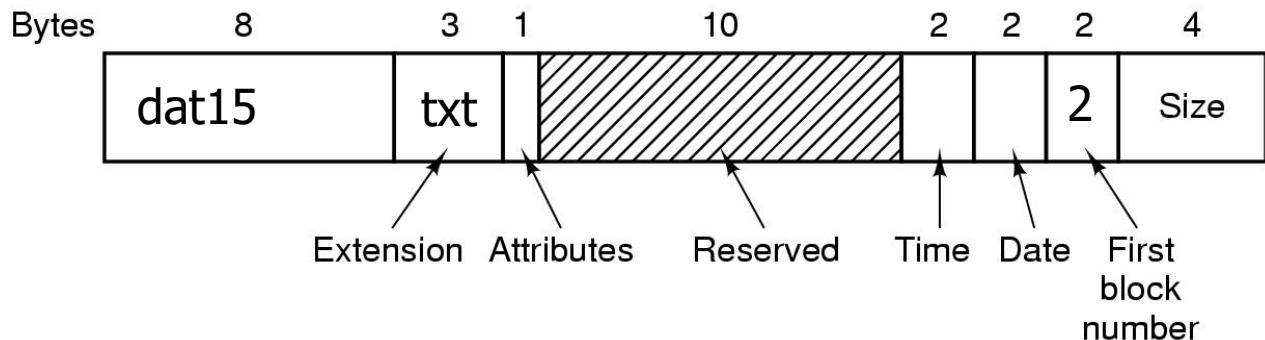


Sadržaj ulaza (*entry*) u FAT za različite verzije FAT

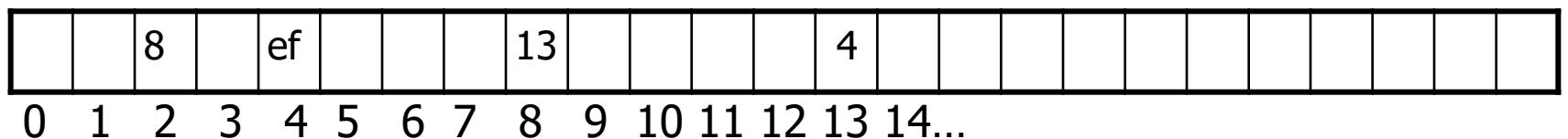
FAT12	FAT16	FAT32	Description
0x000	0x0000	0x?00000000	Free Cluster
0x001	0x0001	0x?00000001	Reserved Cluster
0x002 - 0xFEF	0x0002 - 0xFFFFEF	0x?00000002 - 0x?FFFFFFEF	Used cluster; value points to next cluster
0xFF0 - 0xFF6	0xFFFF0 - 0xFFFF6	0x?FFFFFF0 - 0x?FFFFFF6	Reserved values
0xFF7	0xFFFF7	0x?FFFFFF7	Bad cluster
0xFF8 - 0xFFF	0xFFFF8 - 0xFFFFF	0x?FFFFFF8 - 0x?FFFFFFF	Last cluster in file

FAT primer

Primer entry-a u direktorijumu FAT 16



FAT



- ef je kod završetka datoteke, na primer 0xFFFF u FAT 16



NTFS Windows file system

- ❖ NTFS je file system razvijen za Windows NT
- ❖ Koristi se od Windows 2000 u svim kasnijim verzijama (Windows Vista, Windows 7)
- ❖ Ključne karakteristike NTFS-a
 - ❖ Ima sposobnost oporavka nakon pada sistema i greški na disku
 - ❖ Sigurnost
 - ❖ Podrška za veoma velike diskove i datoteke
 - ❖ Višestruki tokovi podataka (sadržaj daoteke se tretira kao tok bajtova, pa je moguće definisati višestruke tokove podataka nad istom datotekom)
 - ❖ Sposobnost generalnog indeksiranja



NTFS struktura volumena i datoteke

● Sektor

- Najmanja fizička jedinica memorisanja na disku (skoro uvek 512B)

● Klaster (*cluster*)

- Jedan ili više susednih sektora (1, 2, 4, 8, 16, ...), maksimalno 2^{16} B = 64 KB.
- Maksimalna veličina datoteke 2^{32} klastera što iznosi maksimalno 2^{48} B)

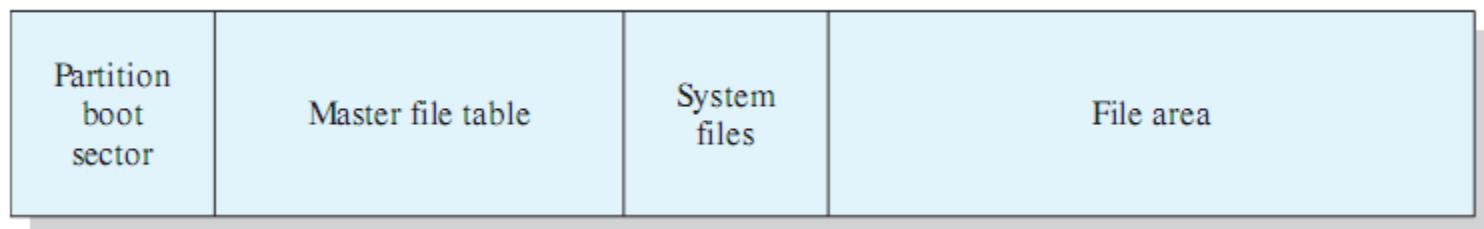
● Volumen

- Logički deo diska (particija diska)
- Maksimalna veličina 2^{64} B



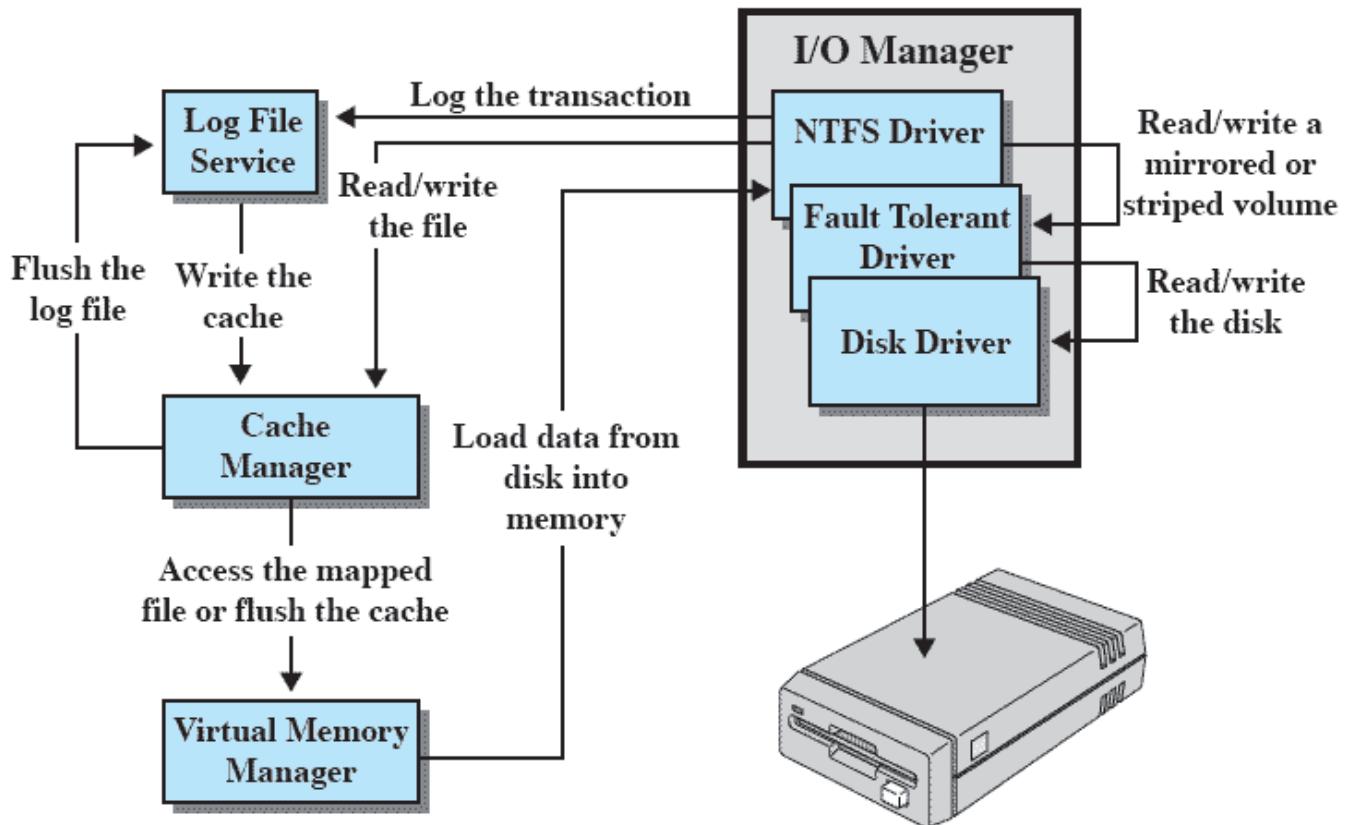
NTFS struktura volumena i datoteke

- Svaki element na volumenu predstavlja datoteku, i svaka datoteka se sastoji od kolekcije atributa.
 - Čak i podaci koji čine sadržaj datoteke se tretiraju kao atribut.
 - Layout NTFS volumena

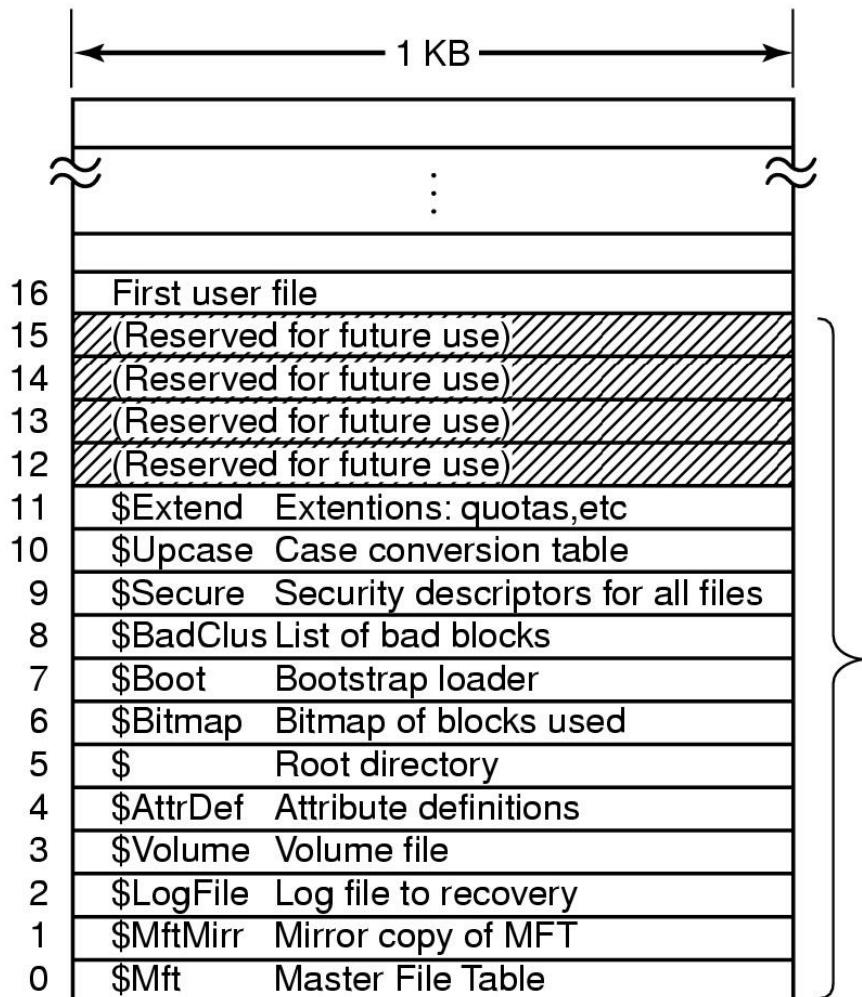


Komponente Windows NTFS-a

- Podrška za oporavljanje strukture fajl sistema



Master File Table - MFT (1)



- MFT je datoteka u vidu tabele slogova i može imati maksimalno 2^{48} slogova, pri čemu je svaki slog veličine 1024 B
- Svaki slog opisuje jedan fajl na volumenu, uključujući i MFT koja se tretira kao datoteka
- Prvih 16 slogova je namenjeno datotekama koje opisuju fajl sistem
- Svaki MFT slog se sastoji od sekvenca atributa opisanih parom (*header, vrednost*)
- Prvi slog opisuje MFT datoteku i gde su locirani klasteri MFT datoteke (adresa prvog klastera MFT datoteke se nalazi u partition boot sector-u)
- Datoteka sa root direktorijumom je opisana u slogu 5
- Slog 6 opisuje datoteku sa bitmapom kojom se registruju slobodni i zauzeti blokovi (klasteri)

Tanenbaum, 2008

Upravljanje datotekama

Operativni sistemi



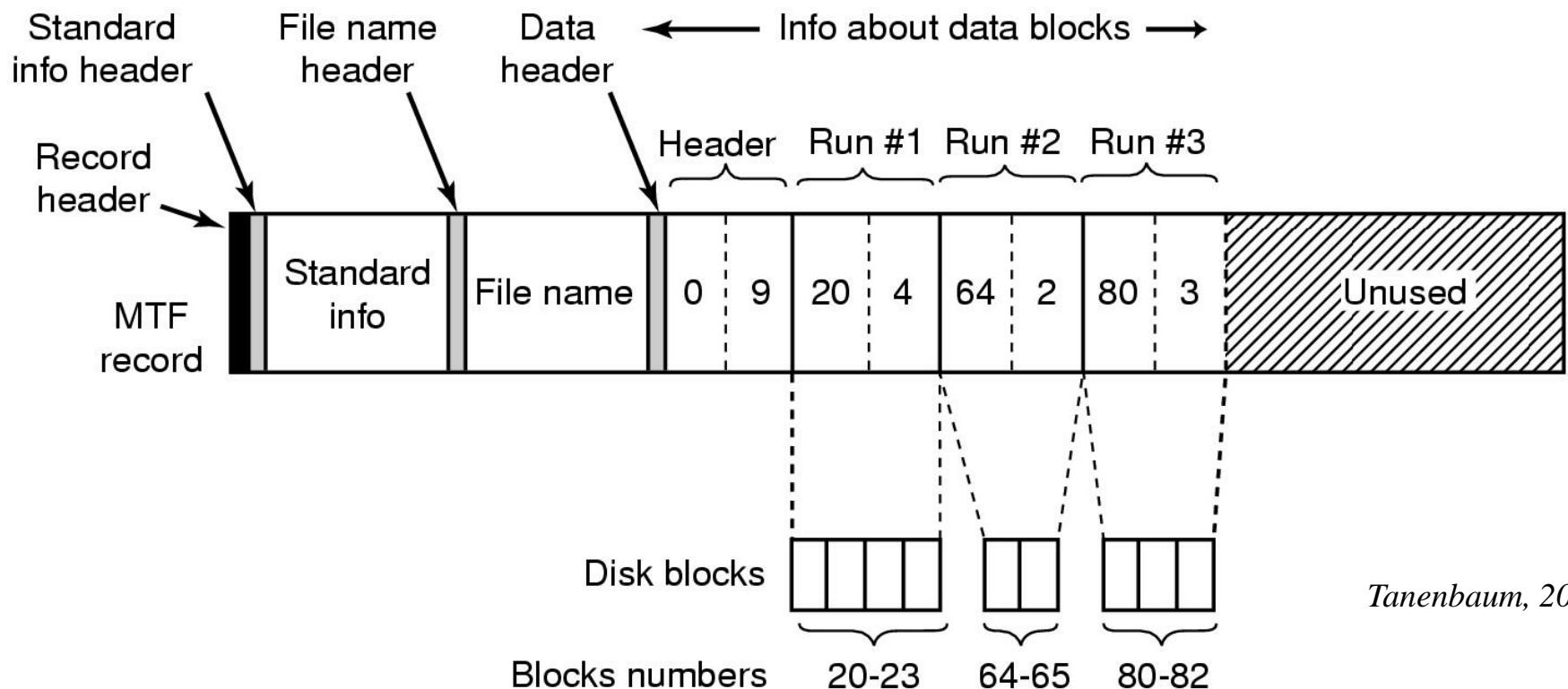
Atributi MFT sloga

(osenčani plavo su obavezni)

Attribute Type	Description
Standard information	Includes access attributes (read-only, read/write, etc.); time stamps, including when the file was created or last modified; and how many directories point to the file (link count).
Attribute list	A list of attributes that make up the file and the file reference of the MFT file record in which each attribute is located. Used when all attributes do not fit into a single MFT file record.
File name	A file or directory must have one or more names.
Security descriptor	Specifies who owns the file and who can access it.
Data	The contents of the file. A file has one default unnamed data attribute and may have one or more named data attributes.
Index root	Used to implement folders.
Index allocation	Used to implement folders.
Volume information	Includes volume-related information, such as the version and name of the volume.
Bitmap	Provides a map representing records in use on the MFT or folder.

MFT slog za datoteku (1)

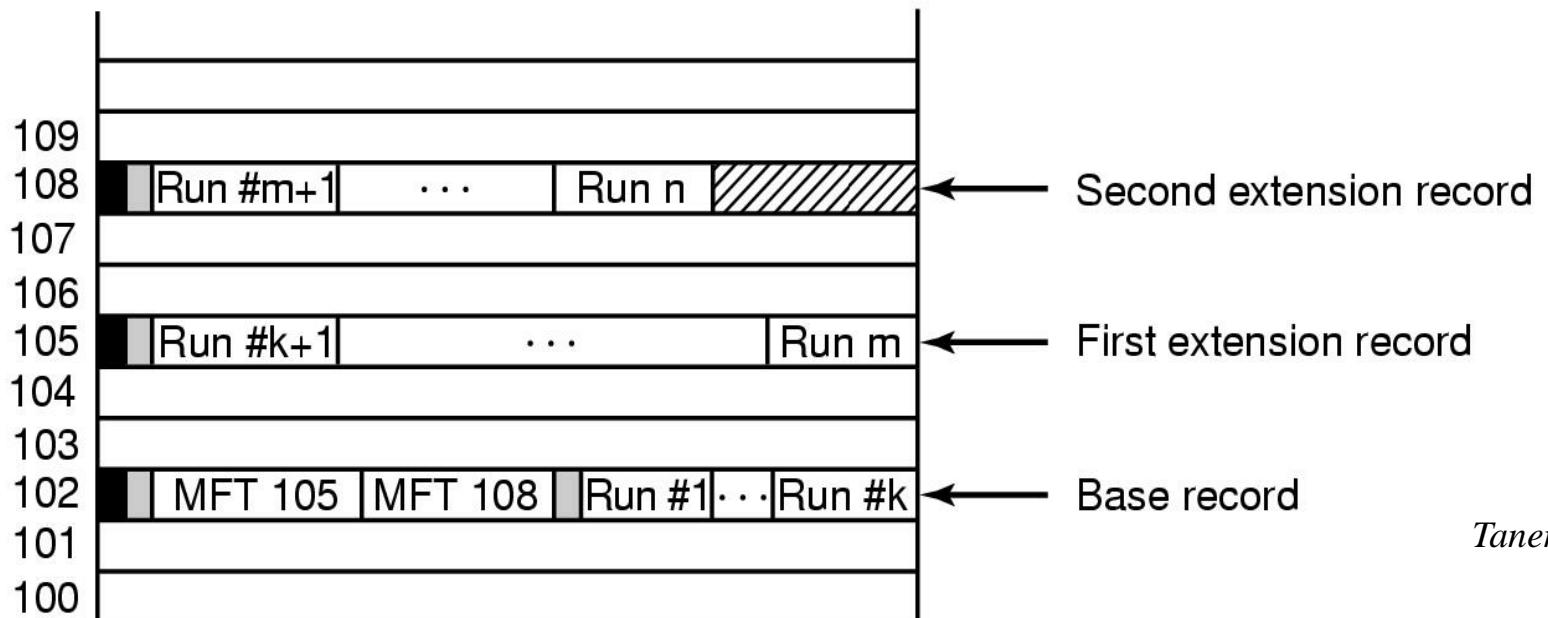
- MFT slog za datoteku od 9 blokova smeštenih na disku u tri grupe sukcesivnih blokova



Tanenbaum, 2014

MFT slog za datoteku (2)

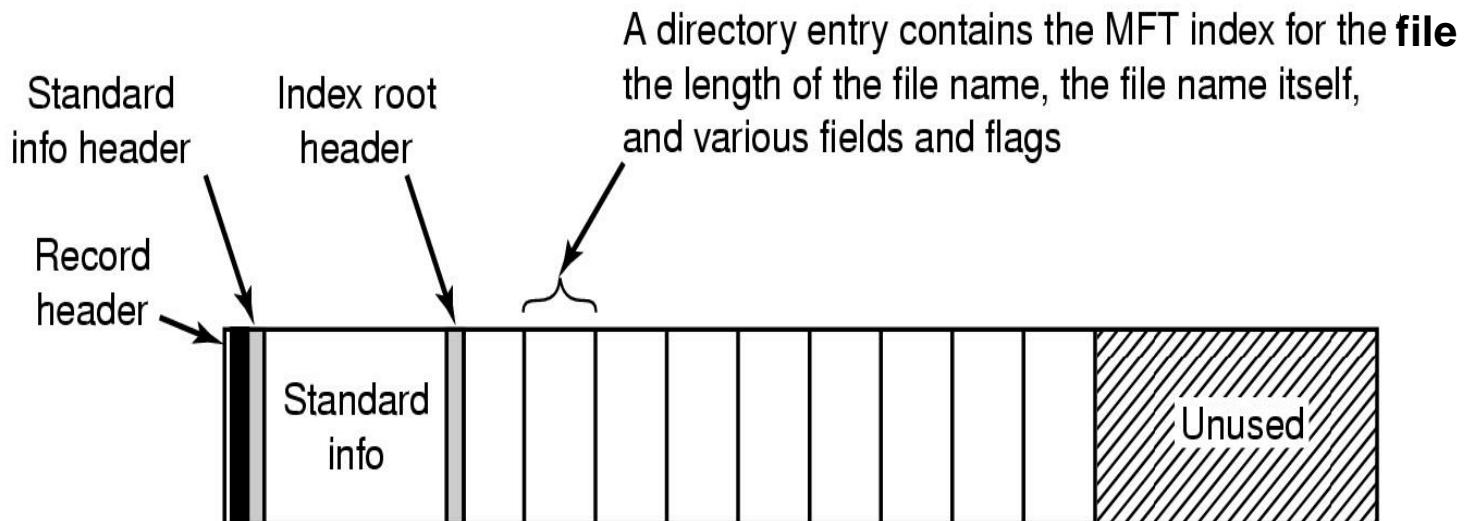
- Datoteka za koju je neophodno 3 MFT sloga za registrovanje blokova



Tanenbaum, 2014

MFT slog za folder

• MFT slog za mali folder

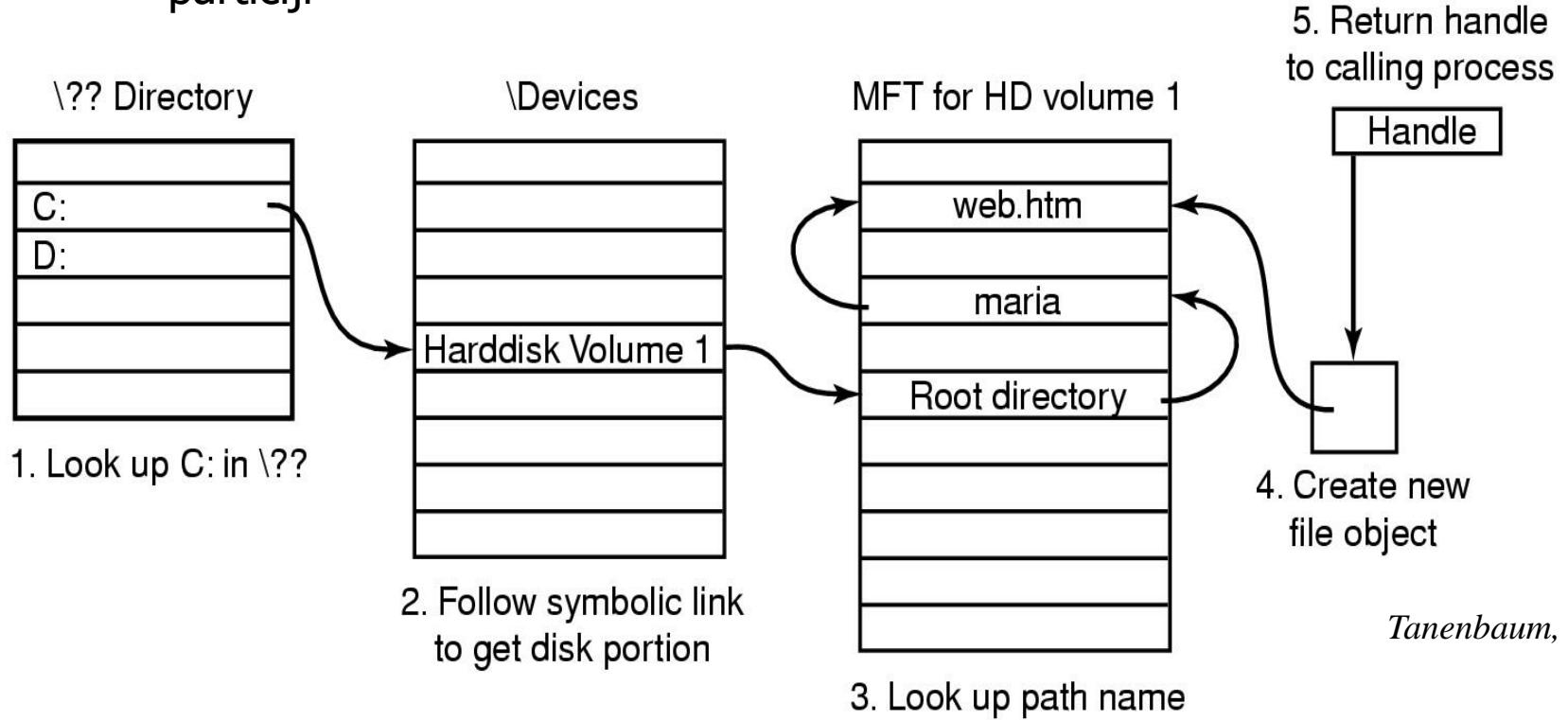


Tanenbaum, 2014

MFT - kreiranje datoteke

- Koraci u kreiranju datoteke C:\maria\web.htm

- Pretražuje se *Object manager* name space i direktorijum \?? da bi se našao C:, zatim na osnovu tok simboličkog linka pretražuje se direktorijum \Device i nalazi objekat \Device\HarddiskVolume1 koji odgovara prvoj partriciji prvog hard diska, a zatim pronalazi MFT za tu particiju



Tanenbaum, 2014



UNIX upravljanje datotekama

◆ Tipovi datoteka

- Regularni ili obični (*ordinary*)
- Direktorijum
- Specijalni
- Imenovani datavodi (*Named pipes*)
- Linkovi
- Simbolički linkovi

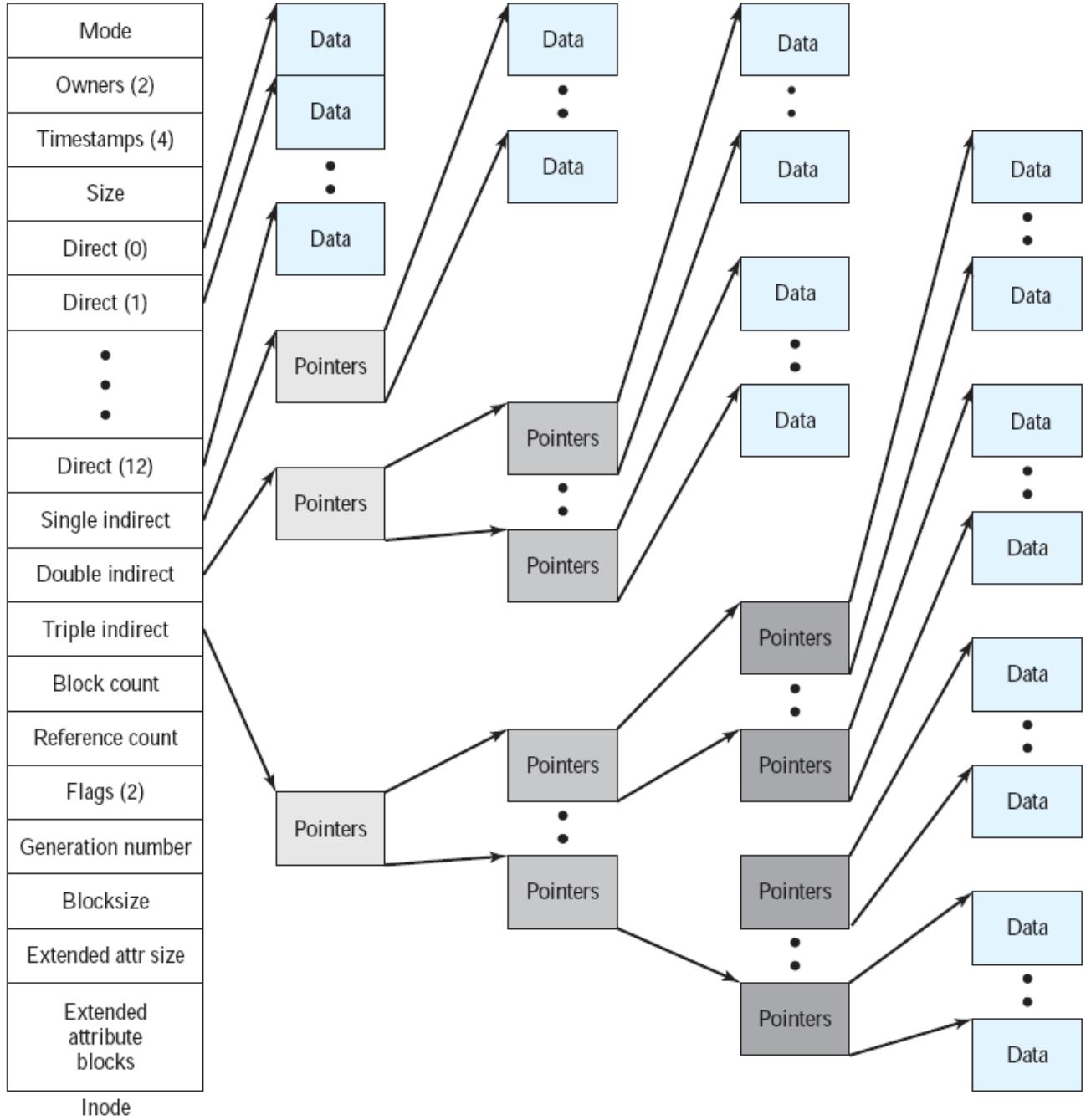


UNIX i-čvor (*i-node*)

- ◆ Sadrži osnovne atributе datoteke

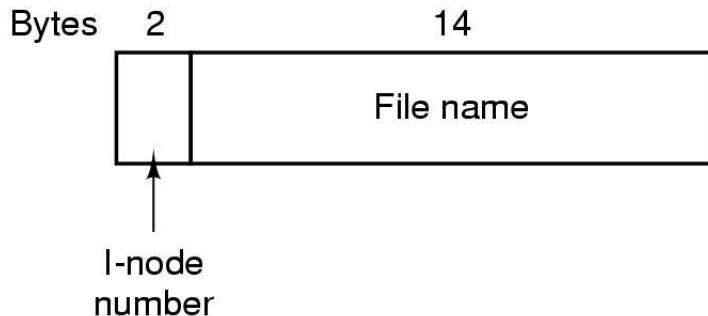
File Mode	16-bit flag that stores access and execution permissions associated with the file. 12-14 File type (regular, directory, character or block special, FIFO pipe 9-11 Execution flags 8 Owner read permission 7 Owner write permission 6 Owner execute permission 5 Group read permission 4 Group write permission 3 Group execute permission 2 Other read permission 1 Other write permission 0 Other execute permission
Link Count	Number of directory references to this inode
Owner ID	Individual owner of file
Group ID	Group owner associated with this file
File Size	Number of bytes in file
File Addresses	39 bytes of address information
Last Accessed	Time of last file access
Last Modified	Time of last file modification
Inode Modified	Time of last inode modification

FreeBSD UNIX i-node

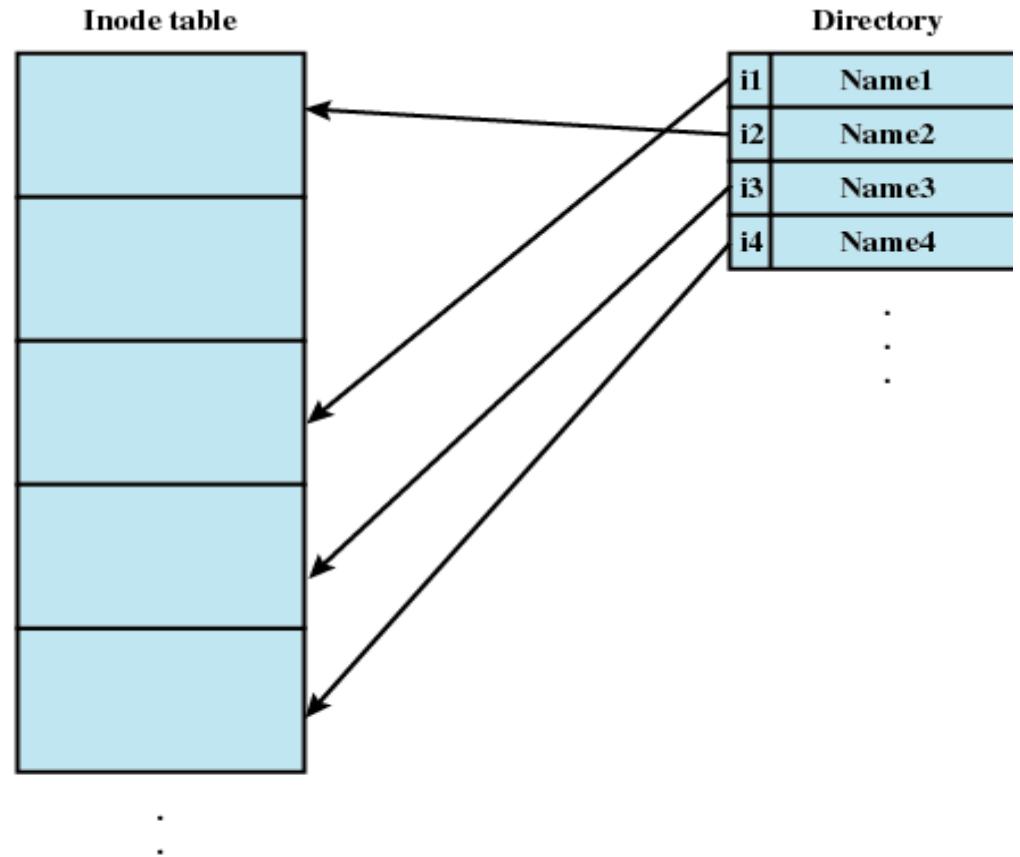


Управљање датотекама Оперативни системи

UNIX direktorijum i i-čvorovi



Ulaz direktorijuma





UNIX file system (3)

Koraci u traženju */usr/ast/mbox*

Root directory

1	.
1	..
4	bin
7	dev
14	lib
9	etc
6	usr
8	tmp

Looking up
usr yields
i-node 6

I-node 6
is for /usr

Mode	
size	
times	
132	

I-node 6
says that
/usr is in
block 132

Block 132
is /usr
directory

6	•
1	..
19	dick
30	erik
51	jim
26	ast
45	bal

/usr/ast
is i-node
26

I-node 26
is for
/usr/ast

Mode	
size	
times	
406	

I-node 26
says that
/usr/ast is in
block 406

Block 406
is /usr/ast
directory

26	•
6	..
64	grants
92	books
60	mbox
81	minix
17	src

/usr/ast/mbox
is i-node
60

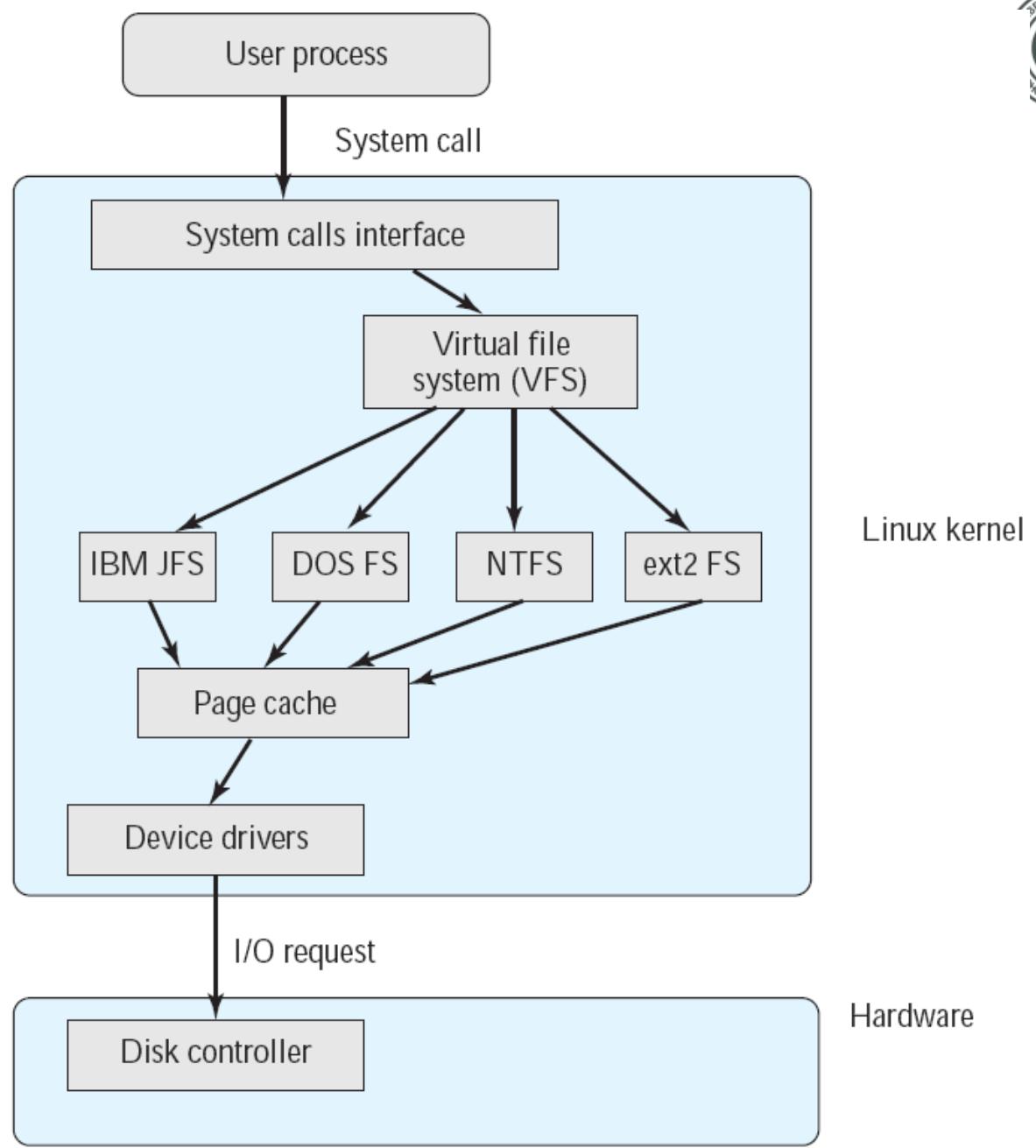
Tanenbaum, 2014



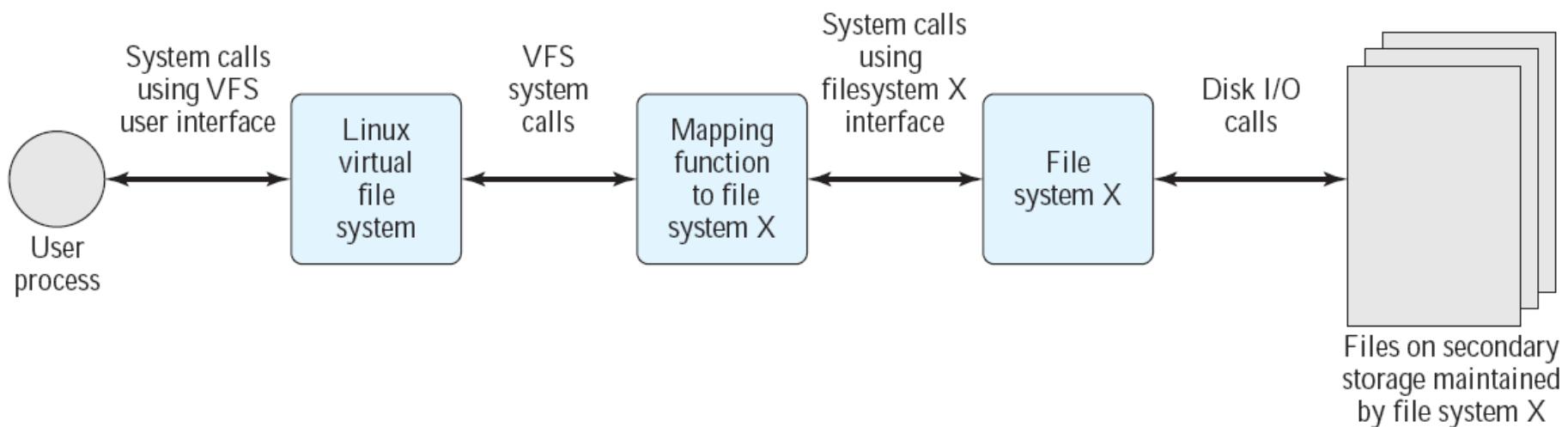
Linux virtuelni file system (VFS)

- ◆ Uniformni interfejs file system-a za korisničke procese
- ◆ Predstavlja opšte karakteristike i ponašanje svih podrazumevanih file system-a
- ◆ Polazi od pretpostavke da su datoteke objekti koji dele osnovna svojstva bez obzira na ciljni file system

Kontekst Linux VFS-a



Koncept Linux VFS-a





Primarni objekti u VFS-u

(implementirani kao C strukture)

❖ *Superblock object*

- ❖ Predstavlja specifičan mount-ovan file system

❖ *Inode object*

- ❖ Predstavlja specifičnu datoteku

❖ *Dentry object*

- ❖ Predstavlja specifičan ulaz direktorijuma (directory entry)

❖ *File object*

- ❖ Predstavlja otvorenu datoteku pridruženu procesu



Domaći zadatak

◆ Poglavlje **12 Upravljanje datotekama**

- ▣ 12.14 Ključni pojmovi, kontrolna pitanja i problemi

◆ U/I animations

- ▣ <https://apps.uttyler.edu/Rainwater/COSC3355/Animations>

- ▣ Interrupt-Driven I/O Cycle
- ▣ The Life Cycle of an I/O Request
- ▣ Disk Scheduling Algorithms: FCFS, SSTF, SCAN