OPERACIJSKI SUSTAVI

SKRIPTA

Autor: Božidar Kovačić

Rijeka, 2008.

SADRŽAJ

1. UVOD.....1

1.1. POVIJEST OPERACIJSKIH SUSTAVA.....2
1.2. VRSTE OPERACIJSKIH SUSTAVA.....5
1.3. OSNOVNI KONCEPTI OPERACIJSKIH SUSTAVA.....6

2. STRUKTURE OPERACIJSKIH SUSTAVA 8

3. PROCESI.....11

2.1. UVJETI NATJECANJA PROCESA (RACE CONDITIONS).....16
2.2. MUTUAL EXCLUSION ZA ZAPOSLENIM ČEKANJEM (BUSY WAIT).....19
2.3. MUTUAL EXCLUSION SA SLEEP AND WAKE UP TEHNIKOM 24
2.4. UPRAVLJANJE PROCESOROM 33
2.5. ZASTOJI 42

4. UPRAVLJANJE MEMORIJOM.....53

5. UPRAVLJANJE ULAZNO-IZLAZNIM JEDINICAMA 76

5.1 HARDVERSKA PODRŠKA RADU U-I UREĐAJA.....76
5.2. SOFVERSKA PODRŠKA RADU U-I UREĐAJA.....80

6. UPRAVLJANJE DATOTEČNIM SUSTAVOM.....84

6.1. DATOTEKA.....84
6.2. DIREKTORIJ.....88
6.3. ALOKACIJA VANJSKE MEMORIJE 90

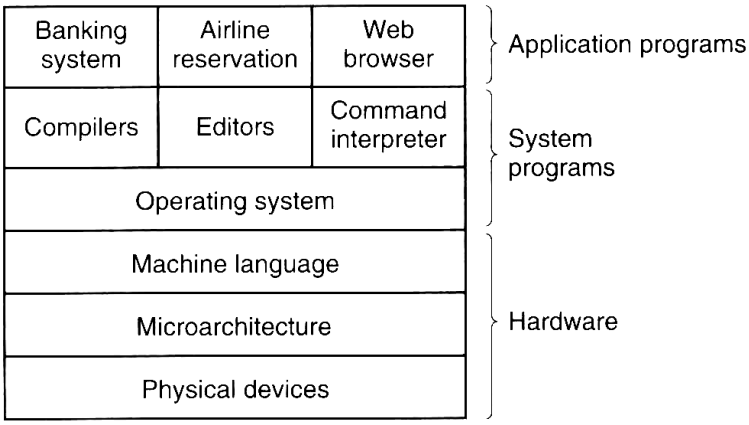
7. SIGURNOST I ZAŠTITA.....101

7.1. SIGURNOST 101
7.2. ZAŠTITA.....105

8. LITERATURA: 108

1. UVOD

Računalni sustav čini jedan ili više glavnih procesora, radna memorija, diskovi, pisači, tipkovnica, monitor, mrežne kartice i druge vrste ulazno-izlaznih uređaja. Pisanje učinkovitih programa za takav sustav je vrlo složen i težak posao. Zbog toga se računala nadograđuju softverskim slojem koji zovemo operacijski sustav (OS). Zadatak OS-a je povezivanje svih dijelova računalnog sustava i pružanje korisničkim programima jednostavno sučelje za rad sa hardverom. Mjesto operativnog sustava u hijerarhiji računalnog sustava prikazano je slikom 1.



Slika 1. Položaj OS-a u računalnom sustavu

Hardver čini fizički sloj računala, mikroarhitektura računala i strojni jezik (*machine language*). Mikroarhitekturu računala čine međusobno povezani fizički uređaji sa ciljem izvođenje određene funkcije (npr. skup registara unutar centralne procesorske jedinice CPU-a). Rad sa takvim fizičkim uređajima ostvaren je mikroprogramima. Skup uređaja i instrukcija koje koristi programski jezik asembler tvori *ISA – Instruction set arhitecture* ili strojni jezik (*machine language*). Na hardver računalnog sustava se nadograđuje OS kao softverski sloj koji programerima skriva složenost hardvera i pruža jednostavniji set instrukcija za rad sa računalom. Na višim razinama sistemskog softvera nalaze se sistemski programi (editori, kompajleri, tumači naredbi – *shell* itd.). Iznad sistemskog sloja nalaze se aplikativni programi koje koriste krajnji korisnici sustava. Pri radu računalo izvodi aplikativne i sistemske programe. Pri radu sa aplikativnim programima računalo je u

korisničkom načinu izvođenja (*user mode*), a kada izvodi sistemske programe u kernel načinu rada (*kernel mod* ili *supervisor mod*).

OS ima dvije osnovne namjene:

- Proširenje osnovnih mogućnosti hardvera pružanjem ugodnijeg sučelja za rad korisnicima.
- Upravljanje resursima.

Proširenje osnovnih mogućnosti računala (*extended machine*) rezultira virtualnom slikom hardvera računala prema korisnicima sa namjenom olakšanja poslova programiranja i uporabe računala.

Upravljanje resursima ogleda se u usklađivanju različitosti između dijelova računalnog sustava, rezerviranju i otpuštanju pojedinih komponenti omogućujući izvođenje zahtjeva korisničkih programa prema hardveru računala.

1.1. POVIJEST OPERACIJSKIH SUSTAVA

Razvoj operacijskih sustava uvjetovan je razvojem računala pa svaka generacijama razvoja računala postavlja nove zahtjeve prema sistemskom softveru određujući razvoj operacijskih sustava. Prema generacijama razvoja računala definiramo operacijske sustave na sljedeći način.

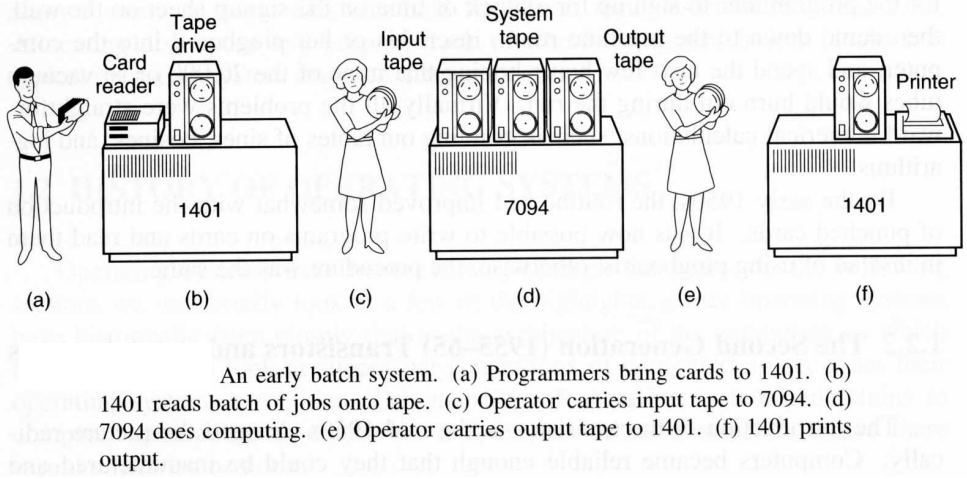
- 1. GENERACIJA (1945-55) ELEKTRONSKE CIJEVI I PREKIDAČI

  Osobitost ove generacije je izostanak potrebe za sistemskim softverom zbog neučinkovitosti računala .

- 2. GENERACIJA (1955-65) TRANZISTORI I SERIJSKA OBRADA

  Dolazi do pojave snažnijih računala sa mogućnošću specijalizirane obrade u znanstvene svrhe sa dobro podržanim matematičkim operacijama (IBM 7094) i komercijalne obrade sa podržanim radom sa ulazno-izlaznim uređajima (IBM 1401). Obrada se izvodila serijski, program za programom (*batch* obrada). Računalo IBM 1401 korišteno je za prijenos programa sa čitača bušenih kartica na magnetnu traku. Magnetna traka je zatim prenesena na računalo IBM 7094 gdje se izvodila konkretna obrada i zapis

rezultata na izlaznu traku. Zadatak OS-a, odnosno sistemskog softvera je učitavanje programa sa ulazne trake, zatim učitavanje programa za izvođenje učitanog programa korisnika sa sistemske trake, izvođenje programa korisnika i na kraju zapis rezultata na izlaznu traku. Postupak se ponavljao sve dok nisu izvedeni svi programi sa ulazne trake. Na kraju se izlazna traka prenijela na računalo IBM 1401 kako bi se ispisali izlazni rezultati na pisač. Opisan postupak prikazan je na slici 2.



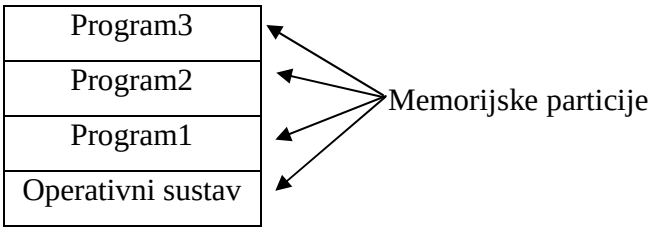
An early batch system. (a) Programmers bring cards to 1401. (b) 1401 reads batch of jobs onto tape. (c) Operator carries input tape to 7094. (d) 7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output.

Slika 2. Upotreba OS u izvođenu serijske (*batch*) obrade programa.

- 3. GENERACIJA (1965-1980) INTEGRIRANI KRUGOVI I MULTIPROGRAMIRANJE

Dolazi do smanjenje razlika između računala i razvoja OS za širu upotrebu računala OS/360. Također se uvode novi koncepti: *multiprogramiranje*, *spooling* i *timesharing*.

Multiprogramiranjem se ostvaruje izvođenje više programa na jednom računalu što se ostvaruje dijeljenjem memorije na dijelove (particije) i učitavanjem programa u jednu particiju (slika 3.).



Slika 3. Implementacija multiprogramiranja

Spooling je tehnika koja se zasniva na učitavanju programa u oslobođenu particiju odmah nakon završetka izvođenja ranije učitanog programa..

Timesharing je tehnika zasnovana na dodjeli vremena izvođenje procesora proporcionalno aktivnosti korisnika sa računalnim sustavom. Aktivniji korisnici dobivaju više procesorskog vremena i obratno.

Daljnje poboljšanje nastojalo se postići razvojem operativnog sustava namijenjenog širokom spektru korisnika. Projekt MULTICS (**MULT**iplexed **I**nformation and **C**omputin **S**ystem) razvijan je u suradnji sa vodećim informatičkim tvrtkama toga vremena, ali nije u potpunosti ispunio očekivanja jer je zadovoljenje zahtjeva jedne grupe korisnika uzrokovalo degradaciju kvaliteta usluga za druge korisnike. Istraživača tvrtke Bell Labs, Ken Thompson je razvio verziju MULTICS-a za osobno (preteču današnjih PC-a) računalo tipa PDP-1, što je označilo početak razvoja operativnog sustava UNIX.. UNIX je napisan u programskom jeziku C i prenosiv je na različite hardverske konfiguracije. Programski kod UNIX-a u počecima je bio dostupan omogućujući nadogradnju sustava prema vlastitim potrebama. To je rezultiralo velikim brojem nekompatibilnih verzija UNIX-a pa je definiran standard POSIX sa skupom osnovnih funkcija koje moraju biti podržane. Kasnije verzije UNIX-a su komercijalizirane što je programski kod učinilo nedostupnim. Za potrebe edukacije studenata stvoren je po uzoru na UNIX, operativni sustav MINIX sa jednostavnijom strukturom i mogućnošću nadogradnje i testiranja sustava. Namjena MINIX-a je edukacija studenata pa njegovi tvorci nisu pokazali interes za nadogradnjom MINIX-a. Student Linus Torvards izradio je napredniju verziju MINIX-a i nastavio njen razvoj što je rezultiralo operativnim sustavom Linux. Linux je OS razvijen od velikog broja programera širom svijeta, uvjetujući malu komercijalnu cijenu sustava i relativno skromne hardverske zahtjeve u odnosu na OS-e sličnih osobina.

- 4. GENERACIJA (1980-sadašnjost) OSOBNA RAČUNALA

Razvoj osobnih računala uvjetovao je i razvoj OS-a za osobna računala te prilagodbu postojećih OS-a. Pojavljuju se prvi OS-i za osobna računala CP/M (**Co**ntrol **P**rogram for **M**icrocomputers), DOS (**D**isk **O**perating **S**ystem), zatim MS-DOS, WINDOWS 3.1, WINDOWS 3.11, WINDOWS 95, WINDOWS 98, WINDOWS Millenium. Osobitost WINDOWS sustava je povezanost sa operativnim sustavom MS-DOS. Paralelno sa razvojem navedenih WINDOWS OS-a razvijaju se i WINDOWS NT (**N**ew **T**echnology) operativni sustavi namijenjeni profesionalnoj upotrebi. WINDOWS NT su 32-bitni OS neovisan o OS MS-DOS-u. Značajnije verzije WINDOWS NT OS-a su WINDOWS NT 4.0, WINDOWS

2000, te WINDOWS XP. Za Apple računala razvijen je poseban operacijski sustav – Macintosh operativni sustav.

Povezivanjem računala javila se potreba za nadogradnjom OS osobnih računala za mrežnu komponentu. Pouzdani mrežnim OS-i su UNIX, LINUX i WINDOWS NT. Najnoviji trend je razvoj OS-a za distribuiranu obradu. Mrežni operativni sustavi su proširenje OS-a osobnih računala za mogućnost kontrole pristupa podacima drugog računala i prijenos podataka između računala, dok distribuirani operativni sustavi omogućuju izvođenje obrade na više računala što znatno povećava kompleksnost OS-a. Razvoj OS-a i dalje će biti uvjetovan razvojem računalne tehnologije (hardvera i tehnoloških inovacija u prijenosu podataka) i prilagođavat će se zahtjevima krajnjih korisnika.

1.2. VRSTE OPERACIJSKIH SUSTAVA

Razvoj operacijskih sustava uvjetovao je različitosti u strukturi i namjeni OS-a. Prema namjeni operacijske sustave dijelimo na:

- Mainframe operacijske sustave – operacijski sustavi za centralizirana računala koje opslužuje veliki broj terminalima spojenih korisnika. Mainframe OS-i bili su dominantni prije pojave osobnih računala PC-a. Primjeri su OS/360 i OS/390.
- Server operacijski sustavi – operacijski sustavi koji velikom broju klijent računala pružaju usluge korištenje hardverskih i softverskih resursa server računala. Primjeri su WINDOWS NT, UNIX i LINUX.
- Multiprocesorski operacijski sustavi – podržavaju rad dva ili više centralno procesorskih jedinica CPU-a. U pravilu su to modificirani server operacijski sustavi sa posebnim osobinama za komunikaciju i spajanje.
- Operativni sustavi za osobna računala – omogućuju prikladno radno okruženje za jednog korisnika. Primjeri su WINDOWS 98, WINDOWS Millenium, WINDOWS 2000, LINUX, Macintosh OS itd.
- Real-time operacijski sustavi – sustavi koji moraju generirati rezultat obrade (odgovor, response) na ulazne podatke u realnom vremenu (u pravilu za manje od 1s). Ukoliko je kašnjenje reakcije pogubno za sigurnost sustava govorimo o hard-time sustavu, a ako se kašnjenje može tolerirati govorimo o soft-time sustavu.

- Embeded operacijski sustavi – operacijski sustavi za male konzole (elektronički adresari, memorijske pločice itd.) sa smanjenim opsegom funkcija.
- Smart card operacijski sustavi – operacijski sustavi namijenjeni korištenju različitih elektroničkih kartica (bankovne kartice, telefonske kartice, parkirne kartice itd.)

1.3. OSNOVNI KONCEPTI OPERACIJSKIH SUSTAVA

Osnovni koncepti operacijskih sustava podrazumijevaju hardverske komponente računalnog sustava, njihovu komunikaciju, te osnovne elemente sistemskog softvera kojima se izvodi upravljanje radom hardverskih komponenti i generira sučelje prema aplikativnom softveru. Hardverske komponente uključuju procesor, memoriju, ulazno-izlazne jedinice i sabirnice. Posebno izučavanje strukture i načina rada ovih komponenti nije predmet izučavanja kolegija Operacijski sustavi. Osnovni elementi sistemskog softvera čine procesi, zastoji, upravljanje memorijom, rad sa ulazno-izlaznim uređajima, datotečni sustav, sigurnost, shell i sistemski poziv. Svaki pojam detaljno je razrađen, a u nastavku su opisane osnovne značajke za svaki pojam.

Proces je osnovni element operacijskog sustava. Procesi su aktivnosti koje nastaju pri izvođenju programa korisnika. Svaki proces je program za sebe sa programskim kodom, podacima za obradu i stack-om. Procesi međusobno komuniciraju i sinhroniziraju svoje aktivnosti. To je omogućeno zaustavljanjem izvođenja procesa i nastavljanjem izvođenja nakon određenog vremenskog intervala. Da bi se proces mogao nastaviti izvoditi nužno je pamtiti njegovo prijašnje stanje što je ostvareno pohranom podataka o procesu u tabelu procesa.

Zastoji su nepoželjne situacije koje dovode do blokiranja izvođenja dva ili više procesa, a uzrokovane su slijedom rezerviranja resursa potrebnih za izvođenje. Zastoji se u pravilu detektiraju i otklanjaju, sprječavaju ili izbjegavaju.

Upravljanje memorijom je od velike važnosti za rad operacijskog sustava jer proces kao osnovni element izvođenja OS-a mora biti pohranjen u memoriji. Kako radna memorija ima ograničenja u pogledu trajnog pamćenja podataka, nužna je upotreba vanjske memorije što

upućuje na povezanost jezgre OS-a (*kernel*) sa upravljačem memorije i datotečnim sustavom.

Datotečni sustav je nužan za ažuriranje i pristup podacima na vanjskoj memoriji omogućujući procesima i izvođenim programima trajnu pohranu podataka.

Sigurnost su aktivnosti poduzimane u cilju sprječavanja nedozvoljenog pristupa podacima korisnika i kontroli korištenja resursa za vrijeme izvođenja procesa.

Shell ili tumač naredbi (*command interpreter*) omogućuje prihvat naredbi korisnika i njihov prijenos na niže razine operativnog sustava. Pomoću shell-a korisnici komuniciraju sa ostatkom računalnog sustava.

Nakon primitka naredbe korisnika shell prepoznaje naredbu i pokreće aktivnosti za izvršenje naredbe. Skup aktivnosti pokrenutih ka izvođenju naredbe korisnika objedinjen je u sistemski poziv putem kojeg shell prosljeđuje zahtjev ostatku operacijskog sustava za izvođenjem tražene aktivnosti korisnika.

2. STRUKTURE OPERACIJSKIH SUSTAVA

Struktura operacijskog sustava pruža uvid u unutarnju građu OS-a omogućujući lakše razumijevanje izvođenja OS-a. Najčešće korištene strukture OS-a su:

- monolitna struktura
- struktura slojeva
- struktura virtualne mašine (stroja)
- struktura exokernela
- klijent-server struktura.

Monolitna struktura

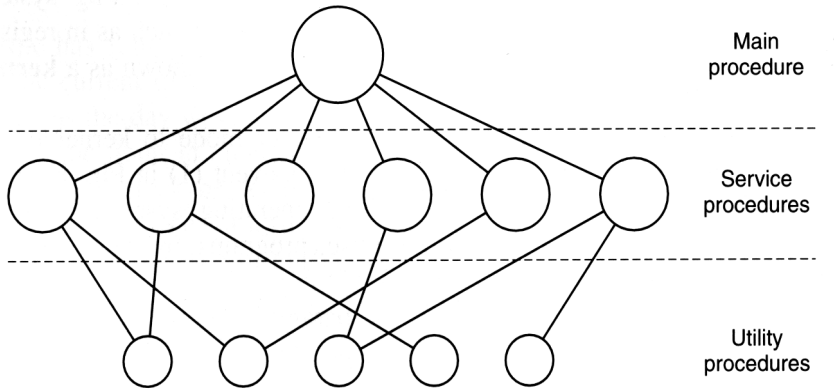
Operacijski sustav sastoji se od skupa procedura koje se mogu međusobno pozivati bez ograničenja. Nakon izrade procedura, slijedi kompajliranje i povezivanje u *exe* verziju koja čini *kernel*. Pri svakoj promjeni nužno je kompajlirati mijenjanu proceduru i izraditi novu *exe* verziju. Sve procedure su međusobno ravnopravne, ali između procedura postoji određena organizacija koja sadrži tri vrste procedura:

- glavne procedure
- servisne procedure
- korisničke procedure.

Zadatak glavne procedure je prihvat sistemskih poziva iz aplikacijskih programa i prosljeđivanje zadataka servisnim procedurama. Servisne procedure imaju zadatak izvođenja određene vrste usluge, a aktiviraju se na zahtjev glavne procedure. Korisničke procedure omogućuju izvođenje učestalih radnji, a pozivaju se iz servisnih procedura. Slika 4. prikazuje organizacije procedura monolitne strukture OS-a.

Struktura slojeva

Operacijski sustav izrađen je od zasebnih cjelina koje se nadograđuju jedna na drugu tvoreći slojeve. Niži slojevi orijentirani su hardveru, a viši slojevi ka aplikativnim programima. Svaki sloj stvara osnovu za primjenu višeg sloja. Primjer operacijskog sustava THE prezentiran je slikom 5.



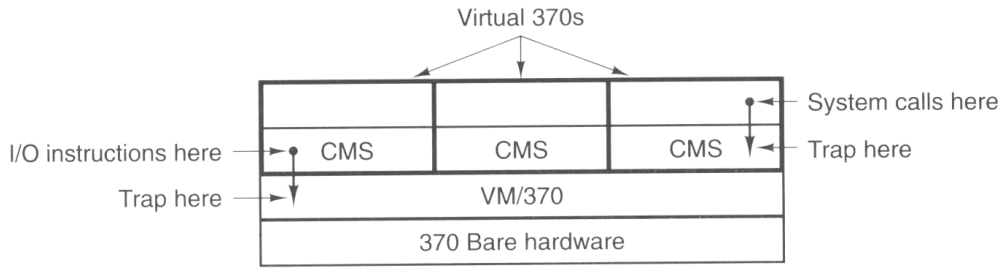
Slika 4. Organizacija procedura monolitne strukture OS-a

Sloj	Funkcija
5	Operater
4	Korisnički programi
3	Upravljanje ulazno-izlaznim uređajima
2	Komunikacija operater-proces
1	Upravljanje memorijom
0	Alokacija procesa i multiprogramiranje

Slika 5. Struktura THE operacijskog sustava

Struktura virtualne mašine

Operacijski sustav omogućuje svakom korisniku zaseban pogled na računalo, odnosno virtualnu sliku računala. Svaki korisnik ima mogućnost korištenja dijela resursa računala istovjetnu samostalnom radu na računalu. Korisnik može proizvoljno instalirati aplikativne programe ili operacijski sustav. Stvarni operacijski sustav izvodi prevođenje naredbi korisnika iz njemu predočene virtualne slike računala na konkretni hardver računala. Struktura virtualne mašine korištene su za mainframe računala sa velikim brojem korisnika kako bi im se omogućila autonomnost u radu sa operacijskim sustavom. Slika 6. prikazuje strukturu operacijskog sustava VM/370.



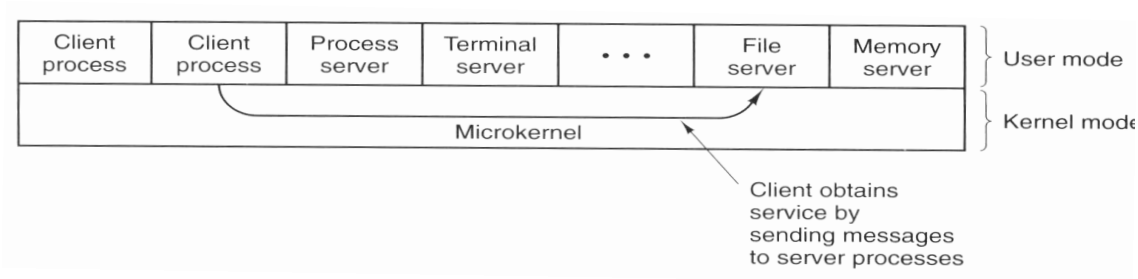
Slika 6. Struktura operacijskog sustava VM/370

Struktura exokernela

Operacijski sustav zasniva se na sličnom konceptu kao kod strukture virtualne mašine uz jednu bitnu razliku: struktura virtualne mašine korisnicima daje identičnu sliku hardvera na kojem rade, dok struktura exokernela korisnicima prezentira virtualnu sliku stroja koja se razlikuje od hardvera računala koje korisnici koriste. To je omogućeno povezivanjem virtualnih resursa koje koriste korisnici sa raspoloživim hardverskim resursima računala.

Klijent-server procedura

Klijent-server struktura zasniva se na minimaliziranom kernelu koji izvodi sistemske pozive prema ostalim dijelovima operacijskog sustava lociranim na višim nivoima. Kernel je u tom slučaju posrednik između korisničkih (aplikativnih) programa i dijelova operacijskog sustava. Time se smanjuje veličina kernela i pojednostavljuje njegovu izradu, dok je izmjena i dorada pojedinih dijelova OS-a izvediva promjenom modula karakterističnog za određene funkcije operacijskog sustava (upravljanje memorijom, datotečni sustav, mreža itd.). Slika 7. prikazuje klijent-server strukturu operacijskog sustava.



Slika 7. Klijent server struktura operacijskog sustava

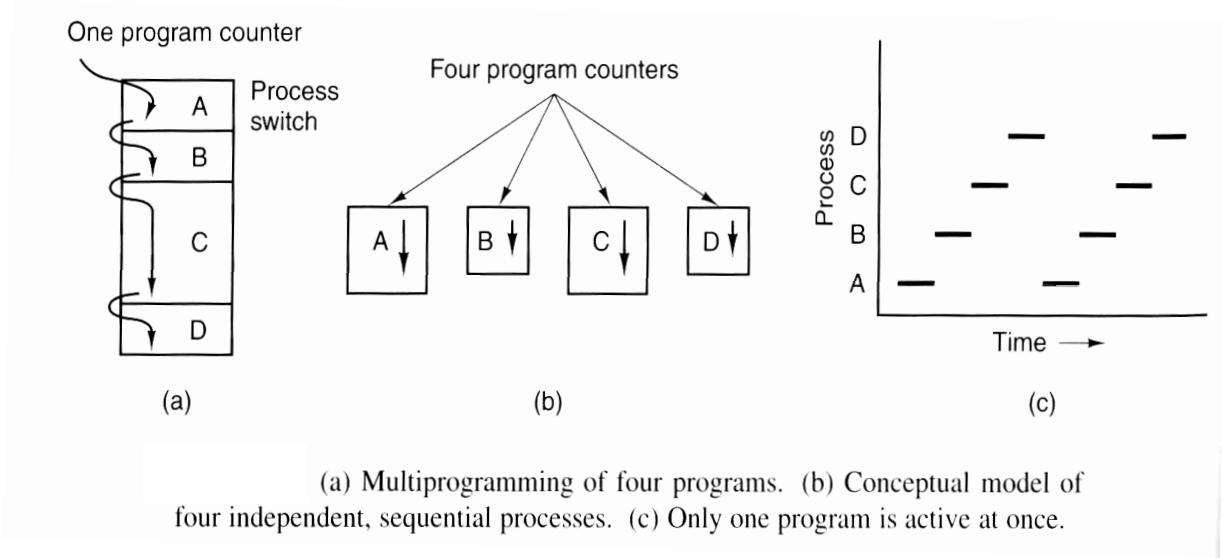
3. PROCESI

Proces je osnovni koncept operacijskog sustava. Svi aplikativni programi i sistemski programi izvode se organiziranim slijedom sekvencijalnih procesa. Pri izvođenju programa generira se niz organiziranih zasebnih aktivnosti koje zovemo procesi. Procese definiramo aktivnostima koje nastaju pri izvođenju programa ili jednostavno programom u izvođenju. Svaki proces smješten je u memoriju i sadrži programski kod, podatke za obradu i stack. Bolje iskorištenje računalnog sustava ostvaruje se paralelnim izvođenjem više programa ili multiprogramiranjem. Paralelno izvođenje više programa znači i paralelno izvođenje procesa koji nastaju pri izvođenju programa. CPU u jednom trenutku može izvoditi samo jednu programsku instrukciju jednog procesa, pa nije moguće izvoditi dva ili više procesa u jednom trenutku, odnosno istovremeno. Multiprogramiranje se ostvaruje kratkotrajnim izvođenjem procesa svakog programa i cikličkim ponavljanjem slijeda izvođenja procesa sve dok se program ne završi. Promatrano u dužem vremenskom intervalu stječe se dojam paralelne obrade više programa, ali u jednom trenutku uvijek se izvodi smo jedan proces. Dakle, ne postoji doslovno paralelno izvođenje programa već pseudoparalelno izvođenje programa ili pseudoparalelizam. Stvarna paralelna obrada programa izvediva je na računalnim sustavima sa više CPU-a. Slika 8. prikazuje multiprogramiranje 4 programa. Svaki program ima svoj zaseban programski brojač, a u nekom trenutku *t* izvodi se samo jedan program.

Izvođenje procesa može se ilustrirati sa primjerom kuhara koji priprema kolače. Paralelu sa računalnim sustavom prikazuje tabela 1.

Tabela 1. Paralela između kuhinje i računalnog sustava

Kuhinja	**Računalni sustav**
kuhar	CPU – glavni procesor
kuhinja (pećnica, mikser, posude itd.)	hardver
recept	program
brašno, jaja, šećer	podaci za obradu
aktivnosti kuhara na pripravljanju kolača	procesi



(a) Multiprogramming of four programs. (b) Conceptual model of four independent, sequential processes. (c) Only one program is active at once.

Slika 8. Multiprogramiranje sa 4 programa.

Kuhar priprema kolač izvođenjem različitih aktivnosti sa sastojcima za izradu kolača prema uputama iz recepta primjenjujući uređaje za obradu sastojaka (mikser, posude, pećnica). Izvođenje tih aktivnosti analogno je procesima u računalnom sustavu. Pretpostavimo da zazvoni telefon tijekom jedne od aktivnosti izrade kolača. Kuhar se javlja na telefon, ali prethodno registrira stanje aktivnosti koja se izvodila u trenutku telefonskog poziva. Nakon obavljenog telefonskog poziva kuhar će nastaviti pripravak kolača nastavljajući prekinutu aktivnost od mjesta prekida uzrokovanog telefonskim pozivom. Ovisno o prirodi telefonskog razgovora moguće je da kuhar prekine izradu kolača i uradi neke aktivnosti povezane sa nekim drugim poslom. Slične situacije događaju se u računalnom sustavu. Pri izvođenju nekog procesa moguća je pojava stanja u računalnom sustavu koja nisu uzrokovana izvođenjem tekućeg procesa, ali moraju biti brzo razriješene (na primjer komunikacija putem mrežne veze). Nužno je prekinuti izvođenje trenutnog procesa i pokrenuti novi proces za rješavanje nastale situacije. Prije pokretanja novog procesa mora se memorirati stanje izvođenje tekućeg procesa kako bi se kasnije moglo nastaviti njegovo izvođenje. Za pohranu podataka o stanju procesa koji omogućuju prekid izvođenja procesa i nastavak izvođenja procesa od trenutka prekida koristi se **tabela procesa**. Tabela procesa ima 3 glavna dijela koji se odnose na izvođenje procesa,

upravljanje adresnim prostorom u radnoj memoriji i upravljanje adresnim prostorom vanjske memorije. Tabela 2. prikazuje osnovne podatke pohranjene u tabeli procesa.

Tabela 2. Prikaz podataka tabele procesa

Process management	**Memory management**	**File management**
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Procesi mogu nastati u slijedećim slučajevima:

1. pri inicijalizaciji sustava
2. pozivom sistemskog poziva iz aplikativnog programa
3. korisničkim zahtjevom za stvaranje novog procesa
4. pokretanjem serijske (*batch)* obrade.

Pri generiranju novog procesa provjerava se tabela procesa i radna memorija kako bi se utvrdilo ima li prazno mjesto u tabeli stranica za upis novog procesa i ima li u radnoj memoriji dovoljno slobodnog adresnog prostora za smještaj procesa. Nakon toga aktivira se procedura za generiranje novog procesa. Operacijski sustav bazirani na UNIX platformama koriste sistemski poziv *fork* za stvaranje novog procesa, dok WINDOWS operacijski sustavi koriste WIN32 funkcijski poziv *CreateProcess*. Pri stvaranju procesa može se definirati hijerarhijski odnos između procesa roditelj-dijete (*parent-child*). Za UNIX operacijske sustave proces dijete je kopija procesa roditelja, a razlikuju se po vrijednosti identifikatora procesa. Za WINDOWS sustave proces roditelj i proces dijete su potpuno različiti, odnosno proces dijete nije kopija procesa roditelja.

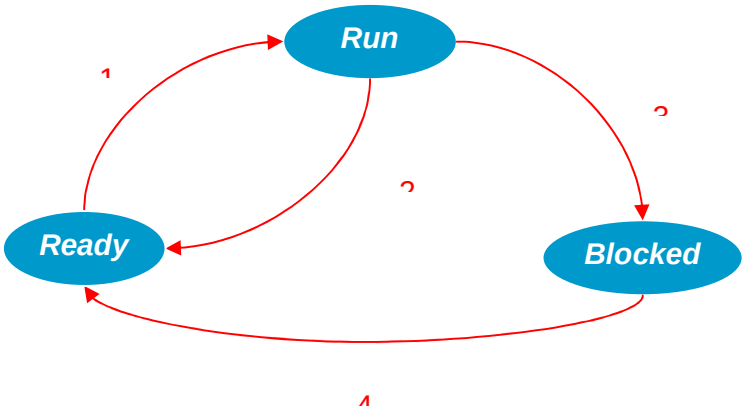
. Proces nestaje u sljedećim situacijama:

1. normalni završetak izvođenja
2. završetak izvođenja uzrokovan greškom (ovisno od procesa)
3. završetak uzrokovan fatalnom greškom (neovisno od procesa)
4. uništen nekim drugim procesom (npr. *kill* sistemskim pozivom; neovisno od procesa).

Stanja procesa

Implementacija multiprogramiranja uvjetuje kratkotrajno izvođenje procesa i dodjelu procesora drugim procesima. Za tako kratko vrijeme izvođenja procesora procesi u pravilu ne mogu završiti svoju obradu pa se pohranjuje njihovo stanje u trenutku gubitka procesora i nastavlja izvođenje od trenutka posljednjeg prekida pri novoj dodjeli procesora. Procesi pri tom mijenjaju stanja u kojima se nalaze. Slika 9. prikazuje stanja procesa:

1. ready ili stanje spremnosti izvođenja procesa na procesoru
2. run ili stanje izvođenja procesa na procesoru
3. blocked ili stanje blokiranosti procesa i čekanja na ispunjenje uvjeta za daljnje izvođenje (unos podataka sa tipkovnice, završetak učitavanja podataka sa hard diska i slično).

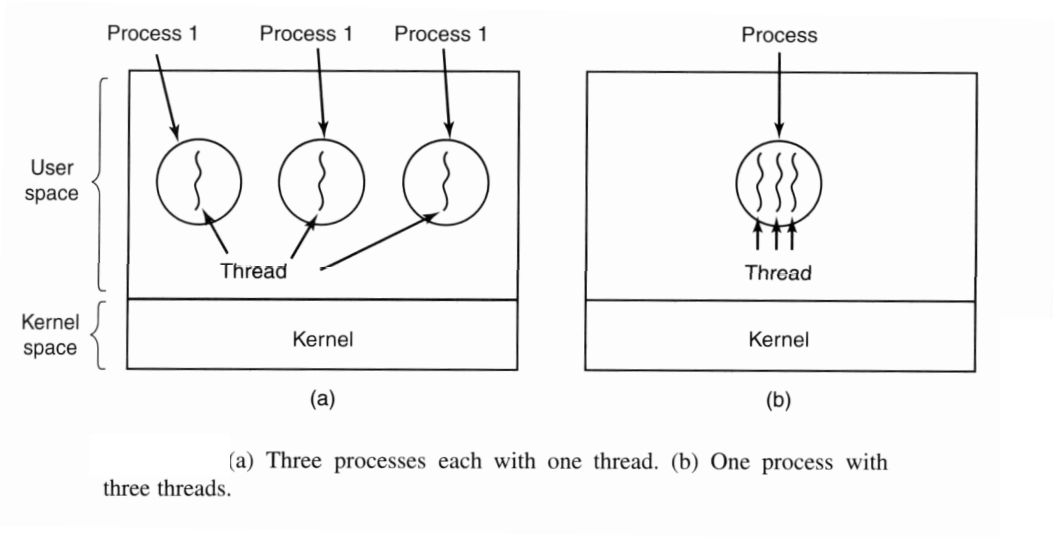


Slika 9. Stanja procesa

Završetkom postupka stvaranja novog procesa proces se nalazi u stanju *ready* i čeka dodjelu procesora. Kada dobije procesor prelazi u *run* stanje (strelica 1). Nakon gubitka procesora proces prelazi u *ready* stanje (strelica 2). Ako tijekom izvođenja nastanu uvjeti koji onemogućuju daljnje izvođenje procesa proces prelazi u stanje *blocked* (strelica 3). Nakon ispunjenja uvjeta za daljnje izvođenje procesi prelaze u *ready* stanje (strelica 4).

Procesne niti – threads

Pri izvođenu procesa unutar procesa izvodi se najmanje jedna procesna nit. Procesna nit se definira dijelom procesa ili procesom unutar procesa. Procesi mogu imati i više od jedne procesne niti koje dijele adresni prostor procesa, otvorene datoteke procesa i dodijeljene resurse. Izvođenje procesnih niti ekvivalentno je multiprogramiranju koje se izvodi unutar jednog procesa. Prednosti korištenja niti očituju se u jednostavnijem zaustavljanju i pokretanju procesnih niti u odnosu na procese jer se podaci o procesnim nitima ažuriraju unutar adresnog prostora procesa pa nije potrebno pohranjivati podatke u tabelu procesa što je obvezno u slučaju pokretanja novog procesa. Također je jednostavnije ostvariti komunikaciju između procesnih niti nego između procesa. Slika 10. prikazuje procese sa jednom niti i jedan proces sa više niti.



Slika 10. Prikaz procesa sa procesnim nitima

Procesne niti najčešće se upotrebljavaju u slijedećim situacijama:

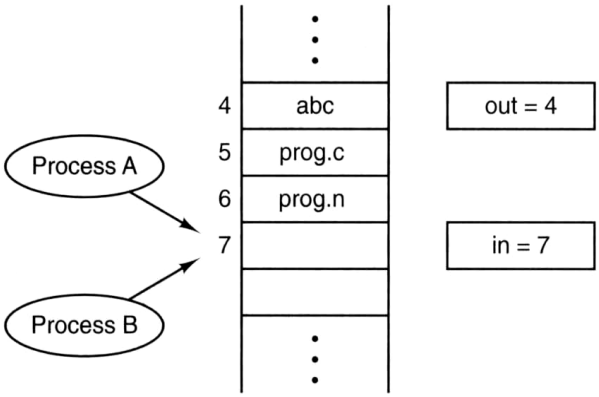
1. izbjegavanje komunikacije između više različitih procesa pri izvođenju aktivnosti koje čine jednu cjelinu
2. generiranje i terminiranje treads-a je znatno brže od stvaranja i uništavanja procesa
3. primjena treads-a daje bolje performanse izvođenja procesa
4. veća prednost pri sustavima sa više CPU-a.

Implementacija procesnih niti uvjetuje veću složenost operacijskog sustava te je potrebno odvagnuti prednosti i nedostatke upotrebe procesnih niti pri pokretanju određenih obrada.

2.1. UVJETI NATJECANJA PROCESA (RACE CONDITIONS)

Primjena multiprogramiranja uvjetuje kratkotrajno izvođenje procesa što u određenim situacijama može uzrokovati greške u radu procesa. Greške su moguće kod resursa kojima može pristupati više procesa tijekom izvođenja. Resurse dijelimo na nedjeljive i djeljive. Nedjeljive resurse može koristiti samo jedan proces (primjer je pisač), a djeljivim resursima može pristupati više procesa za vrijeme trajanja izvođenja jednog procesa (primjer je hard disk). Nastanak greške pri izvođenju ilustrirat ćemo primjerom ispisa dokumenata na pisaču. Dokumenti za ispis pohranjuju se u zaseban direktorij (*spooler directory*) u kome čekaju da ih proces *printer deamon* preusmjeri na ispis na pisaču. Proces *printer deamon* je stalno aktivan, odnosno čeka na slanje dokumenta za ispis na pisaču, te preusmjerava proces na ispis na pisaču. Za rad *spooler* direktorija bitne su dvije vrijednosti: varijabla *in* koja određuje slobodno mjesto za upis dokumenta u direktorij i varijabla *out* koja određuje dokument za ispis na pisaču. Slika 11. prikazuje moguću situaciju sa dva procesa A i B, te vrijednostima varijabli *in* = 7 i *out* = 4. Dva slučaja su moguća sa dva različita rezultata obrade. Prvi slučaj rezultira korektnom obradom:

- proces A pristupa *spooler* direktoriju kako bi utvrdio slobodno mjesto za upis novog dokumenta (*in* = 7),
- proces A izvodi upis dokumenta i povećava vrijednost varijable in za jedan više (*in* = 8),
- proces B dobije procesor utvrđuje mjesto 8 u *spooler* direktoriju slobodno.
- proces B upisuje svoj dokument te postavlja varijablu in na 8



Two processes want to access shared memory at the same time.

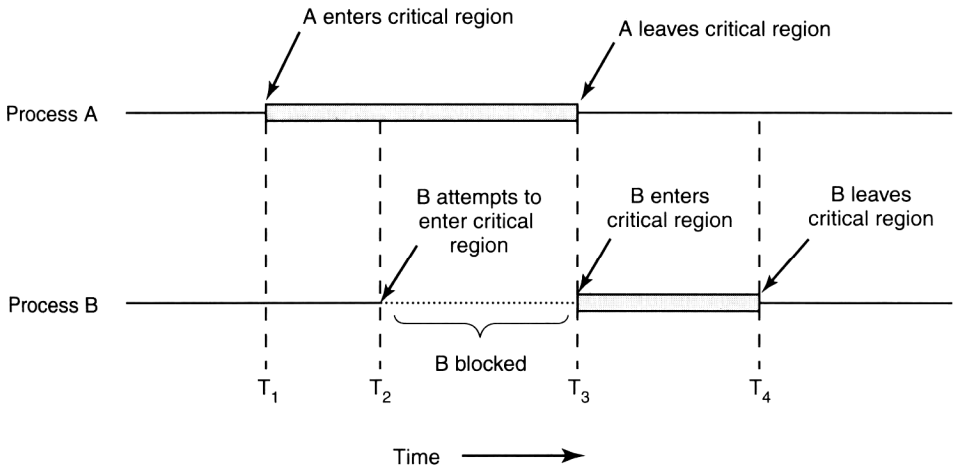
Slika 11. Upis dokumenata u spooler direktorij

Sada analizirajmo situacija koja dovodi do greške u izvođenju procesa:

- proces A pristupa spooler direktoriju i registrira mjesto 7 u spooler direktoriju slobodno za upis dokumenta,
- u tom trenutku proces A gubi procesor koji se dodjeljuje procesu B koji također utvrđuje mjesto 7 kao slobodno (proces A nije se izvodio dovoljno dugo da promijeni vrijednost varijable *in* na 8),
- proces B upisuje svoj dokument na mjesto 7 i povećava vrijednost varijable *in* na 8
- proces A dobiva procesor i nastavlja izvođenje od mjesta prekida, odnosno još uvijek je za proces A mjesto 7 slobodno (provjera slobodnog mjesta izvedena je u prethodnoj dodjeli procesora) pa nastavlja upis dokumenta u *spooler* direktorij na mjesto 7,
- proces A postavlja vrijednost varijable in na 8.

Krajnji rezultat izvođenja procesu A omogućuje ispis dokumenta na pisaču, dok dokument procesa B neće biti ispisan. Oba procesa su korektno izvela svoje obrade ali je nastala greška. Uzrok tome je dodjela procesora između procesa uvjetovana multiprogramiranjem, odnosno pseudoparalelizmom. Vrijeme dodjele procesora svakom procesu i trenutak gubitka procesora ne može se unaprijed sa sigurnošću odrediti što znači da je gubitak procesora moguć u bilo kom trenutku izvođenja procesora (trenutak gubitka procesora uvjetovan je brojem procesa, stanjima procesa i radom algoritma za dodjelu procesora

procesima te stanjem računalnog sustava odnosno generiranjem prekida). Kako je gubitak procesora moguć je u bilo kom trenutku izvođenja procesora moguće su gore opisane situacije koje uzrokuju greške i treba osigurati njihovo sprječavanje. Situacije koje dovode do natjecanja dva ili više procesa za rad sa jednim djeljivim resursom zovu se **race condition**, a faza izvođenja procesa u kojoj se pristupa djeljivom resursu zove se kritična sekcija (**critical region**). Rješenje ranije opisanog problema sa pogrešnim ispisom dokumenta na pisaču je u sprječavanju pristupa *spooler* direktoriju za vrijeme izvođenja kritične sekcije jednog procesa. To znači da za vrijeme provjere varijable *in*, upisa dokumenta u *spooler* direktorij i mijenjanja varijable *in* niti jedan drugi proces koji želi upisati dokument u *spooler* direktorij ne smije pristupiti *spooler* direktoriju. Na taj način proces A će sigurno upisati dokument u slobodno mjesto i neće doći do pogreške u tumačenju trenutno slobodnog mjesta u *spooler* direktoriju. Tehnike koje to omogućuju zovu se **mutual exclusion**. Slika 12 prikazuje način rada mutual exclusion-a. Vidljivo je da proces B prelazi u blokirano stanje sve dok proces A ne završi svoju kritičnu sekciju.



Mutual exclusion using critical regions.

Slika 12. Mutual exclusion na primjeru *spooler* direktorija

Mutual exclusion je dobro rješeno ako zadovoljava sljedeće uvjete:

1. dva ili više procesa ne smiju biti u kritičnoj sekciji
2. brzina CPU nema nikakvog utjecaja na nužnost sprječavanja pojave dva ili više procesa u kritičnim sekcijama na istom resursu
3. proces koji izvodi nekritičnu sekciju ne smije blokirati ulazak procesa u kritičnu sekciju

4. niti jedan proces ne smije čekati beskonačno (dugi interval vremena) dugo na ulazak u kritičnu sekciju.

Tehnike mutual exclusiona mogu biti:

1. Mutual exclusion sa zaposlenim čekanjem (busy wait)
2. Mutual exclusion sa sleep and wake up tehnikom

2.2. MUTUAL EXCLUSION ZA ZAPOSLENIM ČEKANJEM (BUSY WAIT)

Zadatak mutual exclusiona je onemogućavanje ulaska dva ili više procesa u kritičnu sekciju. Tehnika zaposlenog čekanja onemogućuje ulazak dva ili više procesa u kritičnu sekciju na sljedeći način:

- proces A šalje upit za ulazak u kritičnu sekciju i dobiva potvrdan odgovor na poslani upit,
- proces A je u kritičnoj sekciji i prije završetka kritične sekcije izgubi procesor,
- proces B dobiva procesor i pokušava ući u kritičnu sekciju i šalje upit za ulazak u kritičnu sekciju,
- proces B dobiva negativan odgovor na poslani upit, nakon čega proces B ponovo šalje upit za ulazak u kritičnu sekciju,
- proces B ponovo dobiva negativan odgovor, nakon čega ponovo šalje upit,
- postupak se ponavlja sve dok proces B ne izgubi procesor,
- proces A će nakon određenog vremena dobiti procesor i završiti kritičnu sekciju,
- proces B će kada dobije procesor poslati upit za ulazak u kritičnu sekciju koji će biti prihvaćen i proces B će ući u kritičnu sekciju.

Proces B je cijelo vrijeme korištenje procesora dobivao negativan odgovor na upit za ulazak u kritičnu sekciju zato jer proces A nije završio svoju kritičnu sekciju, a to može ostvariti pri sljedećoj dodjeli procesora, čemu mora prethoditi istek dodijeljenog procesorskog vremena procesa B. Dakle, samo kada proces B izgubi procesor moguć je završetak kritične sekcije procesa A, a time i kasniji ulazak procesa B u kritičnu sekciju. Kako je cijelo vrijeme rada procesa B potrošeno na postavljanje upita za ulazak u kritičnu

sekciju i dobivanje negativnog odgovora pri čemu proces nije napredovao kažemo da proces izvodi zaposleno čekanje (busy wait).

Načini implementacije tehnike zaposlenog čekanja su:

1. onemogućavanje prekida
2. upotreba varijabli
3. striktna alternacija
4. Petersonovo rješenje
5. TSL instrukcija.

Onemogućavanje prekida

CPU u svom radu može biti zaustavljen samo pojavom električnog signala (prekid, *interrupt*) generiranim radom ulazno-izlaznog uređaja, greškom u izvođenju programa ili satnim prekidom. Ukoliko nema tih signala CPU kontinuirano izvodi instrukcije programa sve do kraja programa. Primjena onemogućavanja prekida zasniva se na sprječavanju pojave prekida za vrijeme izvođenja kritične sekcije procesa čime se onemogućuje zaustavljanje procesa pri izvođenju kritične sekcije. Proces koji započne kritičnu sekciju ne može izgubiti procesor sve dok ne završi kritičnu sekciju. Nedostatak onemogućavanja prekida je gubitak bitnih signala kojima se operacijskom sustavu dojavljuju bitne informacije o stanju računalnog sustava za vrijeme izvođenja kritične sekcije.

Upotreba varijabli

Za kontrolu ulaska u kritične sekcije koristi se varijabla. Kada je ulazak dozvoljen vrijednost varijable je 0, a kada je neki proces u kritičnoj sekciji varijabla ima vrijednost 1. Početna vrijednost varijable je 0. Ako proces želi ući u kritičnu sekciju provjeriti će varijablu, ući u kritičnu sekciju i promijeniti varijablu na 1. Kada završi kritičnu sekciju vratiti će varijablu na 0. Ukoliko proces izgubi proces prije nego završi kritičnu sekciju, a drugi proces po dobitku procesora želi ući u kritičnu sekciju dobiti će negativan odgovor jer je vrijednost varijable 1, pa proces izvodi zaposleno čekanje (ponavljanje upita i dobivanje

negativnog odgovora). Proces će moći ući u kritičnu sekciju kada se procesor dodijeli procesu koji je u kritičnoj sekciji, nakon čega taj proces završava kritičnu sekciju i vraća varijablu na 0 omogućujući drugim procesima ulazak u kritičnu sekciju. Ovo rješenje ipak može dovesti do pojave dva procesa u kritičnoj sekciji: ako prvi proces provjeri varijablu i prije nego promijeni varijablu sa 0 na 1 izgubi procesor, a drugi proces po dobitku procesora provjeri varijablu i uđe u kritičnu sekciju te izgubi procesor prije završetka kritične sekcije, otvorena je mogućnost ulaska dva procesa u kritičnu sekciju. Oba procesa će biti u kritičnoj sekciji ako prvi proces dobije procesor i nastavi izvođenje od trenutka provjere varijable (prvi proces još uvijek pamti vrijednost varijable 0) i uđe u kritičnu sekciju. Vjerojatnost pojave ovog slučaja je mnogo manja nego ranije objašnjenog slučaja ulaska dva procesa u kritičnu sekciju pri upotrebi *spooler* direktorija.

Striktna alternacija

Koristi se za slučaj kada se dva procesa trebaju izmjenjivati u kritičnim sekcijama na nekom resursu. Programski kod za striktnu alternaciju prikazan je na slici 13.

```
while (TRUE) {
  while (turn != 0)  /* wait */ ;
  critical_region();
  turn = 1;
  noncritical_region();
}
```

(a)

```
while (TRUE) {
  while (turn != 1) /* wait */ ;
  critical_region();
  turn = 0;
  noncritical_region();
}
```

(b)

Slika 13. Programski kod za striktnu alternaciju

Varijabla *turn* omogućuje alternaciju procesa pri ulasku u kritičnu sekciju. Pretpostavimo da je *turn* = 0, proces A izvodi kod pod a), a proces B kod pod b). Pretpostavimo da se proces A dobije procesor. Kako uvjet u *while* petlji nije zadovoljen proces ulazi u kritičnu sekciju. Ako izgubi procesor i proces B dobije procesor slijedi zaposleno čekanje procesa B jer je uvjet u *while* petlji zadovoljen pa se cijelo vrijeme dodjele procesora izvodi *while* petlja. Kada proces A dobije procesor završi kritičnu sekciju i promijeni varijablu turn na 1 čime omogućuje ulazak procesa B u kritičnu sekciju pri novoj dodjeli procesora. Ako

proces A želi ući u kritičnu sekciju dok se u njoj nalazi proces B vrijedi ista logika kao za slučaj kada je proces B pokušavao ući u kritičnu sekciju dok se proces A nalazio u njoj. Nedostatak striktne alternacije je onemogućavanje ulaska procesa u kritičnu sekciju zbog izvođenje nekritične sekcije nekog drugog procesa. Ako jedan proces ima znatno veću nekritičnu sekciju u odnosu na drugi proces moguće je da proces promijeni varijablu turn i započne nekritičnu sekciju, no zbog dužine nekritične sekcije vjerojatan je gubitak procesora prije završetka nekritične sekcije, pa drugi proces može ući i završiti kritičnu sekciju, zatim izvesti nekritičnu sekciju i pokušati ući ponovo u kritičnu sekciju. Zbog trenutne vrijednost varijable turn to će biti onemogućeno, ali ne zato što je prvi proces u kritičnoj sekciji, već zato što nije završio nekritičnu sekciju.

Petersonovo rješenje

Petersonovo rješenje otklanja nedostatak striktne alternacije, odnosno izbjegava blokiranje ulaska procesa u kritičnu sekciju zbog izvođenja nekritične sekcije nekog drugog procesa. Na slici 14. prikazan je programski kod za Petersonovo rješenje.

```
#define FALSE  0
#define TRUE   1
#define N      2                        /* number of processes */

int turn;                               /* whose turn is it? */
int interested[N];                      /* all values initially 0 (FALSE) */

void enter_region(int process);         /* process is 0 or 1 */
{
     int other;                         /* number of the other process */

     other = 1 – process;               /* the opposite of process */
     interested[process] = TRUE;        /* show that you are interested */
     turn = process;                    /* set flag */
     while (turn == process && interested[other] == TRUE) /* null statement */ ;
}

void leave_region(int process)          /* process: who is leaving */
{
     interested[process] = FALSE;       /* indicate departure from critical region */
}
```

Slika 14. Programski kod za Petersonovo rješenje

TSL isntrukcija

TSL (test and set lock) instrukcija je hardversko rješenje. Zasniva se na rješenju koje je slično upotrebi varijabli, ali se hardverskom implementacijom izbjegava mogućnost pojave dva ili više procesa u kritičnim sekcijama. TSL instrukcija kopira sadržaj varijable u registar i postavlja vrijednost varijable *lock* na 1. Slika 15. prikazuje primjenu TSL instrukcije pri kontroli ulaska procesa u kritičnu sekciju.

```
enter_region:
        tsl register,lock       | copy lock to register and set lock to 1
        cmp register,#0         | was lock zero?
        jne enter_region        | if it was non zero, lock was set, so loop
        ret                     | return to caller; critical region entered

leave_region:
        move lock,#0            | store a 0 in lock
        ret                     | return to caller
```

Slika 15. Primjena TSL instrukcije

Vrijednost varijable *lock* inicijalno je postavljena na 0. Proces pri ulasku u kritičnu sekciju izvodi proceduru *enter_region*, a po završetku kritične sekcije *leave_region*. Proces A koji ulazi u kritičnu sekciju kopira vrijednost varijable *lock* u registar i postavlja vrijednost varijable *lock* na 1 (TSL instrukcija). Kako je registar 0 dozvoljen je ulazak u kritičnu sekciju. Ako proces A izgubi proces prije izlaska iz kritične sekcije i neki drugi proces B pokuša ući u kritičnu sekciju izvoditi će zaposleno čekanje. Uzrok izvođenje zaposlenog čekanja je vrijednost registra 1 kopirane iz varijable lock (TSL instrukcija) i ponovno izvođenje provjere varijable *lock*. Po isteku dodjele procesora procesu B, proces A može dobiti procesor i završiti kritičnu sekciju te izvođenjem procedure *leave_region* postaviti *lock* na 0 dozvoljavajući procesu B ulazak u kritičnu sekciju pri sljedećoj dodjeli procesora. TSL instrukcija je dobro rješenje ali mora imati hardversku podršku za izvođenje TSL instrukcije što ga čini ovisnim o hardveru računala.

Tehnika zaposlenog čekanje je jednostavniji način implementacije mutual exclusiona, ali uzrokuje nekorisno trošenje vremena rada procesora pa ga treba izbjegavati.

2.3. MUTUAL EXCLUSION SA SLEEP AND WAKE UP TEHNIKOM

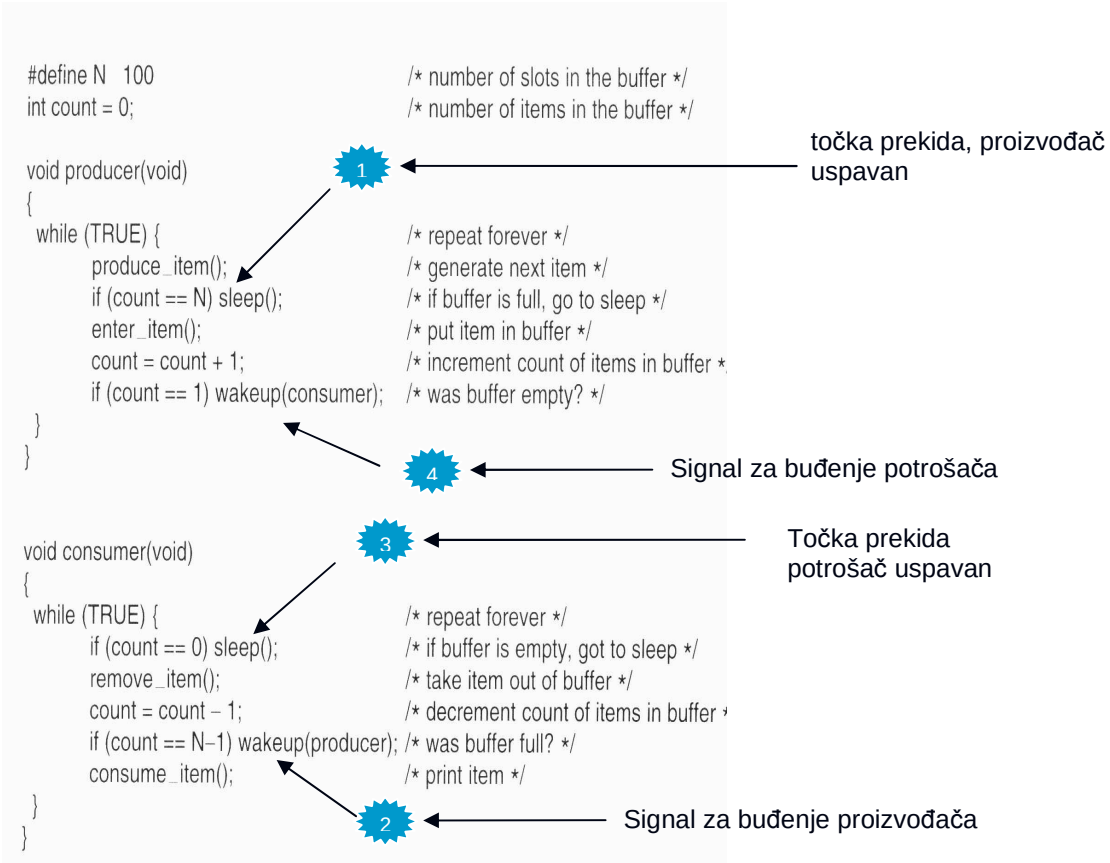
Mutual exclusion sa sleep and wake up tehnikom razlikuje se od tehnike zaposlenog čekanja upravo po onemogućavanju pojave zaposlenog čekanja. Kada proces nema uvjete za ulazak u kritičnu sekciju zbog nedovršene kritične sekcije nekog drugog procesa, proces se prevodi u sleep stanje, što odgovara blokiranom stanju, a kada se steknu uvjeti za ulazak u kritičnu sekciju izvodi se wake up, odnosno prevođenje procesa iz blokiranog stanja u ready stanje. Na taj način proces koji ne može ući u kritičnu sekciju nije ni kandidat za dodjelu procesora pa ne može ni izvoditi zaposleno čekanje. Tehnika sleep and wake up je bolje, ali i implementacijski složenije rješenje u odnosu na tehnike zaposlenog čekanja. Tehnike sleep and wake up mogu se implementirati primjenom:

1. semafora
2. monitora
3. poruka.

Primjenu tehnike sleep and wake up prikazat ćemo na problemu proizvođač-potrošač. Problem proizvođač-potrošač (P-P u nastavku) koristi međumemoriju za rad dva procesa, pri čemu proces proizvođač upisuje vrijednosti u međumemoriju, a proces proizvođač čita vrijednosti iz međumemorije. Međumemorija je ograničenog kapaciteta pa vrijede slijedeća ograničenja:

1. proces proizvođač može upisati samo onoliko vrijednosti koliko je slobodnih mjesta u međumemoriji, odnosno ne smije upisivati u 'punu' međumemoriju (u tom slučaju bi se prepisala nepročitana vrijednost)
2. proces potrošač može čitati samo onoliko vrijednosti koliko je trenutno upisano u međumemoriju, odnosno ne smije čitati iz 'prazne' međumemorije (u tom slučaju bi se jedna vrijednost dva puta pročitala)
3. za vrijeme upisa vrijednosti u međumemoriju ne smije se izvoditi čitanje vrijednosti iz međumemorije i obratno.

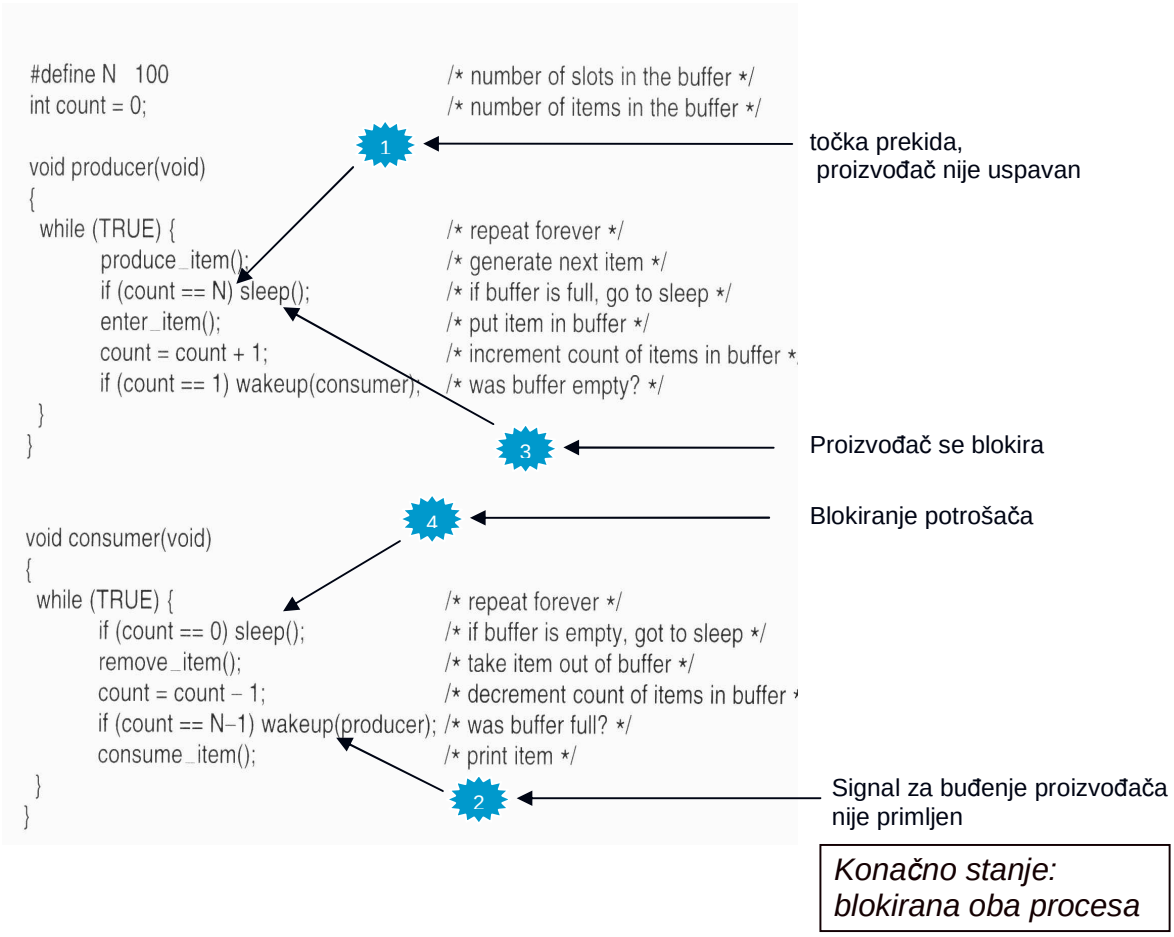
Prikaz rješenja problema P-P dat je na slici 16.



Slika 16.Rješenje problema proizvođač-potrošač

Varijabla *count* koristi se za brojanje upisanih vrijednosti u međumemoriju i na početku ima vrijednost 0. Ako bi se samo izvodio proces proizvođač za svaki prolaz while petljom upisala bi se jedna vrijednost u međumoriju, i povećala vrijednost varijable *count* za 1. Nakon upisnih 100 vrijednosti zbog vrijednosti varijable *count* = 100 proces proizvođač bi se preveo u sleep stanje (blokiranje procesa), što je označeno brojem 1 na slici 16. Proces potrošač prolaskom while petljom čita jednu vrijednost i umanjuje varijablu *count* za 1. Nakon čitanja 1 vrijednosti *count* će poprimiti vrijednost 99 što će ispuniti uvjet za pokretanjem wake up procedure koja će prevesti blokirani proces proizvođač iz blokiranog u ready stanje (na slici 16. to je označeno brojem 2) što će omogućiti upis vrijednosti u međumemoriju pri sljedećoj dodjeli procesora.. Pretpostavimo da je međumemoriju upisano 100 vrijednosti. Ukoliko bi se konstantno izvodio proces potrošač nakon čitanja 100-te vrijednosti varijabla *count* bi imala vrijednost 0 i uzrokovala bi blokiranje procesa potrošača pri pokušaju novog čitanja iz međumorije (slika 16. broj 3). Pokretanjem procesa

proizvođača upisati će se nova vrijednost u međumoriju, povećati *count* na 1 i poslati poziv wake up blokiranom procesu potrošaču koji će se prevesti u ready stanje (slika 16. broj 4.) Na taj način osiguravaju se ograničenja pri izvođenju rada sa međumorijom. Rješenje problema P-P je korektno i osigurava ispunjenje postavljenih ograničenja, ali primjena pseudopararelizma može dovesti do pojave blokiranja oba procesa iako u međumemoriji ima upisanih vrijednosti za čitanje ili slobodnih mjesta za upisivanje. Na slici 17. prikazan je slijed dodjele procesora koji dovodi do blokiranja oba procesa.



Slika 17. Prikaz blokiranje procesa pri radu sa međumemorijom

Pretpostavimo da je međumemorija puna, odnosno vrijednost varijable *count* je 100. Pri pokušaju upisa nove vrijednosti proizvođač ima ispunjen uvjet za prelazak u sleep stanje, ali prije nego pređe u sleep stanje izgubi procesor (slika 17. broj 1). Nakon toga potrošač dobije procesor, pročita jednu vrijednost, smanji varijablu *count* i šalje signal za buđenje proizvođača (slika 17. broj 2). Kako je proizvođač izgubio procesor prije nego li je prešao u

sleep stanje, signal potrošača za buđenje proizvođača ne može se iskoristiti. Potrošač sada može nastaviti sa čitanjem novih vrijednosti i smanjivanjem vrijednosti varijable *count*. Kada se procesor ponovo dodijeli proizvođaču on će nastaviti sa izvođenjem od točke prekida i otići u sleep (blokirano) stanje (slika 17. broj 3). Potrošač će dobiti procesor i nastaviti čitanje vrijednosti iz međumemorije, ali kako signal za buđenje šalje proizvođaču samo nakon čitanje prve vrijednosti iz pune međumemorije (count = N – 1), potrošač više neće slati signal za buđenje proizvođača. Na kraju će se potrošač blokirati nakon što pročita sve vrijednosti iz međumemorije. Krajnji rezultat je prazna memorija i blokirana oba procesa. Istom analogijom nastaje blokiranja oba procesa za slučaj kada iz prazne memorije potrošač pokušava čitati vrijednost. Pojava blokiranja oba procesa ovisna je o dodjeli procesora između procesa, a kako se dodjela procesora ne može unaprijed odrediti mora se osigurati sprječavanje pojave blokiranja oba procesa.

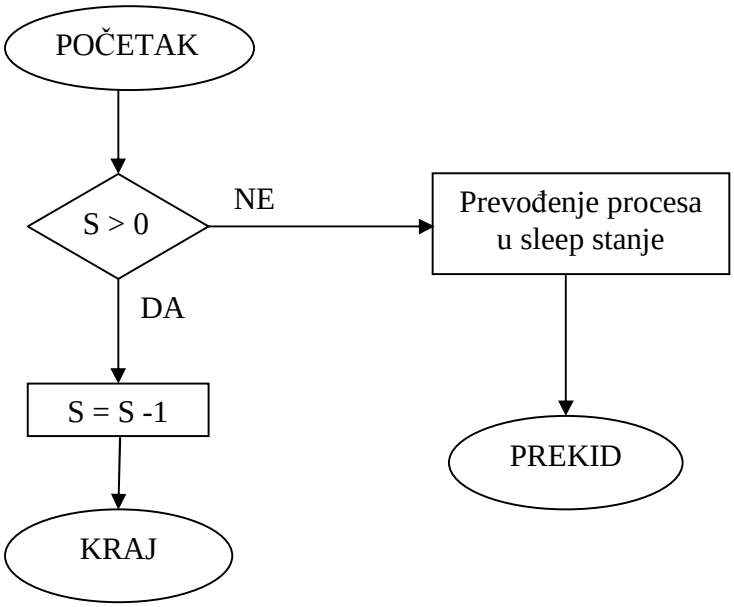
Semafori

Semafori su nenegativne cjelobrojne vrijednosti korištene za kontrolu dozvole pristupa resursima. Pri radu sa semaforima koriste se dvije atomične operacije DOWN i UP. Atomičnost operacija znači neprekidnost izvođenja aktivnosti tih operacija. Neprekidnost izvođenja ostvaruje se onemogućavanjem prekida za vrijeme izvođenja operacija DOWN i UP. Onemogućavanje prekida može dovesti do gubitka bitnih informacija o stanju sustava, ali kako izvođenje operacija DOWN i UP kratko traje ta mogućnost gubitka informacija je malo vjerojatna. Proces koji ulazi u kritičnu sekciju izvodi operaciju DOWN, a kada završi kritičnu sekciju izvodi operaciju UP.

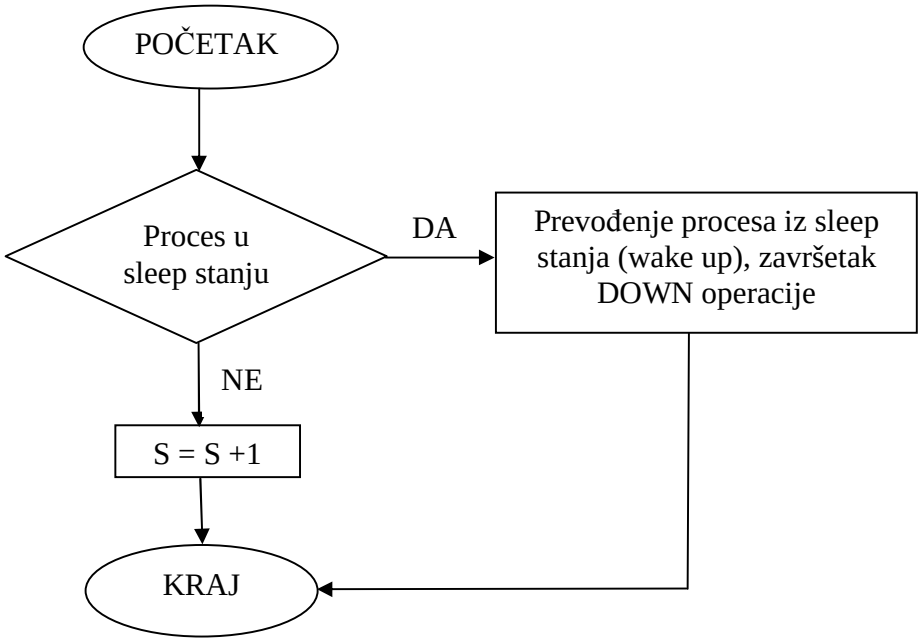
DOWN operacija provjerava stanje semafora i u slučaju pozitivne vrijednosti semafora, smanjuje semafor za 1, dozvoljava nastavak izvođenja procesa i njegov ulazak u kritičnu sekciju. Ako je semafor 0, DOWN operacija proces prevodi u sleep stanje i nije završena. Opisani skup aktivnosti izvodi se bez prekida. Slika 18. prikazuje rad operacije DOWN.

UP operacija izvodi se po završetku kritične sekcije. Izvodi se provjera čekanja procesa na dodjelu određenog resursa, odnosno provjeru prevedenosti nekog procesa u sleep stanje zbog vrijednosti semafora 0 pri izvođenju DOWN operacije. Ako postoji takav proces izvodi se wake up aktivnost na tom procesu (prevođenje u ready stanje) i vrijednost

semafora se ne mijenja. Ukoliko je više takvih procesa primjenom algoritma selektira se proces za izvođenje wake up aktivnosti. Ako nema procesa koji čekaju na dodjelu slijedi povećanje semafora za 1. Skup opisanih aktivnosti izvodi se kao jedna cjelina i ne može se prekidati u izvođenju. Slika 19. prezentira rad UP operacije.



Slika 18. rad DOWN operacije



Slika 19. Rad operacije UP

Za rješavanje problema P-P koriste se tri semafora:

- empty → brojač praznih mjesta u međumemoriji
- full → brojač upisnih vrijednosti u memoriji
- mutex → kontrola rada sa međumemorijom.

Rješenje problema P-P uz upotrebu semafora prikazano je slikom 20.

```
#define N 100                                   /* number of slots in the buffer */
typedef int semaphore;                          /* semaphores are a special kind of int */
semaphore mutex = 1;                            /* controls access to critical region */
semaphore empty = N;                            /* counts empty buffer slots */
semaphore full = 0;                             /* counts full buffer slots */

void producer(void)
{
 int item;

 while (TRUE) {                                 /* TRUE is the constant 1 */
        produce_item(&item);                    /* generate something to put in buffer */
        down(&empty);                           /* decrement empty count */
        down(&mutex);                           /* enter critical region */
        enter_item(item);                       /* put new item in buffer */
        up(&mutex);                             /* leave critical region */
        up(&full);                              /* increment count of full slots */
 }
}


void consumer(void)
{
 int item;

 while (TRUE) {                                 /* infinite loop */
        down(&full);                            /* decrement full count */
        down(&mutex);                           /* enter critical region */
        remove_item(&item);                     /* take item from buffer */
        up(&mutex);                             /* leave critical region */
        up(&empty);                             /* increment count of empty slots */
        consume_item(item);                     /* do something with the item */
 }
}
```

Slika 20. Problem P-P uz upotrebu semafora

Objašnjenje primjene semafora na problemu P-P opisat ćemo na slučaju dodjele procesora koji dovodi do blokiranja oba procesa. Međumemorija je puna (*empty* = 0, *full* = 100 i *mutex* = 1) i pokreće se proizvođač. On izvodi operaciju DOWN na semaforu *empty*. Kako je semafor *empty* 0 proizvođač se prevodi u sleep stanje, odnosno izvodi se blokiranje proizvođača. Kada procesor dobije potrošač on izvodi DOWN operaciju na semaforu *full*, postavlja semafor *full* na 99, izvodi DOWN operaciju na semaforu *mutex*, zatim čita jednu vrijednost iz međumemorije, izvodi operaciju UP na semaforu *mutex* i izvodi operaciju UP na semaforu *empty*. Kako je proizvođač u sleep stanju izvodi se wake up na proizvođaču,

odnosno prevođenje proizvođača iz blokiranog stajna u ready stanje, a vrijednost semafora *empty* ostaje ista (empty = 0). U slučaju da se procesor odmah nakon toga dodijeli proizvođaču slijedilo bi izvođenje DOWN operacije na semaforu *mutex*, upis vrijednosti u međumemoriju, izvođenje operacije UP na semaforu *mutex* i izvođenje operacije UP na semaforu *full*. Kako nema procesa u sleep stanju na semaforu full, UP operacija bi povećala vrijednost semafora *full* na 100. Dakle nakon čitanja jedne vrijednosti i upisa nove vrijednosti semafori bi imali vrijednosti: *empty* = 0, *full* = 100, *mutex* = 1. Kod klasičnog problema P-P dolazilo je do gubitka procesora u trenutku kada je proizvođač trebao preći u sleep stanje. Primjenom semafora to je izbjegnuto jer se prevođenje u sleep stanje izvodi kao dio DOWN operacije koja je atomična i ne dozvoljava prekid u izvođenju. Dakle, atomičnost operacija DOWN i UP osigurava izbjegavanje pojave oba procesa u blokiranom stanju. Ista analogija izvediva je za slučaj čitanja potrošača iz prazne memorije (*empty* = 100, *full* = 0, *mutex* = 1). Semafor *mutex* ima zadatak sprječavanje istovremenog pristupa međumemoriji za proces proizvođač i potrošač, a njegova programska izvedba prikazana je na slici 21. uz primjenu TSL instrukcije.

```
mutex_lock:
    TSL REGISTER,MUTEX          | copy mutex to register and set mutex to 1
    CMP REGISTER,#0             | was mutex zero?
    JZE ok                      | if it was zero, mutex was unlocked, so return
    CALL thread_yield           | mutex is busy; schedule another thread
    JMP mutex_lock              | try again later
ok: RET | return to caller; critical region entered


mutex_unlock:
    MOVE MUTEX,#0               | store a 0 in mutex
    RET | return to caller
```

Slika 21. Programski kod semafora mutex

Monitor

Monitori u posebne procedure koje dozvoljavaju pozivanje samo iz jednog procesa, odnosno sprječavaju pristup ostalim procesima sve dok proces koji trenutno koristi monitor ne završi rad sa njim. Monitori su softversko rješenje problema P-P prikazano slikom 22. Rješenje je vrlo slično klasičnom rješenju problema P-P koje dovodi do blokiranja oba procesa, ali primjena monitora onemogućuje tu pojavu. Pretpostavimo da je međumemorija

puna i da proizvođač pokušava upisati novu vrijednost pri čemu poziva monitor (*ProducerConsumer.inset(item)*) te prije prevođenja u sleep stanje izgubi procesor. Procesor se dodijeli potrošaču koji kod klasičnog problema P-P šalje signal za buđenje proizvođača, no kako se koriste semafori potrošaču nije dozvoljen pristup monitoru (*ProducerConsumer.remove()*). To znači da potrošač neće moći poslati signal za buđenje sve dok proizvođač ne završi rad sa monitorom, a to će se realizirati pri sljedećoj dodjeli procesora. Primjenom monitora izbjegnut je gubitak signala za buđenje proizvođača pa time i mogućnost blokiranja oba procesa. Nedostatak monitora je ograničenost primjene na relativno mali broj programskih jezika.

```
monitor ProducerConsumer
        condition full, empty;
        integer count;
        procedure insert(item: integer);
        begin
                if count = N then wait(full);
                insert_item(item);
                count := count + 1;
                if count = 1 then signal(empty)
        end;
        function remove: integer;
        begin
                if count = 0 then wait(empty);
                remove = remove_item;
                count := count - 1;
                if count = N - 1 then signal(full)
        end;
        count := 0;
end monitor;

procedure producer;
begin
        while true do
        begin
                item = produce_item;
                ProducerConsumer.insert(item)
        end
end;
procedure consumer;
begin
        while true do
        begin
                item = ProducerConsumer.remove;
                consume_item(item)
        end
end;
```

Slika 22. Primjena monitora u rješavanju problema P-P

Poruke

Komunikacija porukama može se iskoristiti u rješavanju problema P-P. Pri slanju poruka između dva procesa koriste se sljedeće poruke:

- send(destination,&message) → slanje poruke procesu sa identifikatorom *destination*
- receive(source,&message) → primanje poruke od procesa sa identifikatorom *source*

Nakon slanja poruke nekom procesu, proces nakon primitka poruke šalje poruku potvrde primitka poruke **acknowledgement**. Proces koji je poslao prvu poruku po primitku poruke *acknowledgement* saznaje da je poruka ispravno poslana. Ako prvi proces ne primi poruku *acknowledgement* poslati će ponovo početnu poruku. Prvi proces poruku acknowledgement neće primiti ako proces primalac nije primio poruku procesa pošiljaoca ili je pošiljalac poslao poruku ali ona nije stigla do primaoca. U prvom slučaju proces primalac primiti će poruku i poslati poruku *acknowledgement*, a u drugom slučaju primalac će primiti istu poruku dva puta, ali će prepoznati da je poslala ista poruka, te će je ignorirati i poslati pošiljaocu poruku *acknowledgement*. Pri slanju poruka može se koristiti sinkroni ili asinkroni način slanja poruka. Pri sinkronom načinu oba procesa moraju biti pripravna istovremeno na komunikaciju, dok asinkroni način ne uvjetuje spremnost oba procesa za komunikaciju. Za asinkroni način obično se koriste mailbox-ovi ili randezvouz način slanja i primanja poruka gdje je slanje nove poruke uvjetovano primitkom prijašnje poruke. Na slici 23. prikazan je problem P-P uz primjenu poruka.

Najprije potrošač šalje N poruka o slobodnim mjestima ('prazne' poruke) proizvođaču. Proizvođač pri upisu svake nove vrijednosti uzme jednu 'praznu' poruku i pošalje potrošaču jednu poruku o upisanim vrijednostima ('puna' poruka). Za punu memoriju proizvođač je potrošio sve 'prazne' poruke, a za praznu memoriju potrošač je iskoristio sve 'pune' poruke pa ne mogu nastaviti rad sa međumemerijom. Rad je moguć ako se pročita vrijednost iz pune međumemorije i pošale 'prazna' poruka ili upiše vrijednost u praznu međumemoriju i pošalje 'prazna' poruka.

```
#define N 100                                   /* number of slots in the buffer */

void producer(void)
{
  int item;
  message m;                                    /* message buffer */

  while (TRUE) {
        produce_item(&item);                    /* generate something to put in buffer */
        receive(consumer, &m);                  /* wait for an empty to arrive */
        build_message(&m, item);                /* construct a message to send */
        send(consumer, &m);                     /* send item to consumer */
  }
}

void consumer(void)
{
  int item, i;
  message m;

  for (i = 0; i < N; i++) send(producer, &m); /* send N empties */
  while (TRUE) {
        receive(producer, &m);                  /* get message containing item */
        extract_item(&m, &item);                /* extract item from message */
        send(producer, &m);                     /* send back empty reply */
        consume_item(item);                     /* do something with the item */
  }
}
```

Slika 22. Problem P-P uz primjenu poruka

2.4. UPRAVLJANJE PROCESOROM

Upravljač procesorom (*scheduler*) je dio operacijskog sustava sa zadaćom određivanje slijeda dodjele procesora procesima u ready stanju. Scheduler redoslijed dodjele procesora ostvaruje primjenom algoritma za dodjelu prioriteta. Frekvencija 50 ili 60 herca (interval od ≈ 20 ms) koristi se za aktiviranje vremenskog prekida nakon čega slijedi zaustavljanje izvođenje trenutnog procesa i aktiviranje shedulera kako bi odredio da li je potrebno nastaviti sa izvođenjem trenutno izvođenog procesa ili je potrebno pokrenuti neki novi proces. Dodjela procesora nekom drugom procesu može biti i ranije u slučaju pojave prekida i pokretanja servisne rutine za obradu prekida. Pri radu shedulera bitni su sljedeći kriteriji:

- Kriteriji uspješnosti algoritama:
  - Općenito:
    - Fairness – pravednost u dodjeli procesora
    - Policy enforcment – pridržavanje dogovorenih načela dodjele procesora
    - Balance – uravnoteženost aktivnosti svih dijelova sustava.
  - Batch
    - Throughput – maksimalno povećanje poslova po satu
    - Turnaround time – smanjivanje vremena između izvođenja i termimiranja
    - CPU utilization – težiti što većoj aktivnosti CPU-a.
  - Inteaktivni sustavi
    - Response time – vrijeme odziva sustava
    - Proportionality – postizanje očekivanje krajnjih korisnika.
  - Real time sustavi
    - Meeting deadlines – izbjegavanje gubitka podataka
    - Predictability – izbjegavanje degradacije kvalitete u multimedijalnim sustavima.

Algoritme za upravljanje procesorom dijelimo na:

1. Algoritme za batch sustave
2. Algoritme za interaktivne sustave
3. Algoritme za sustave u realnom vremenu
4. Algoritmi za procesne niti

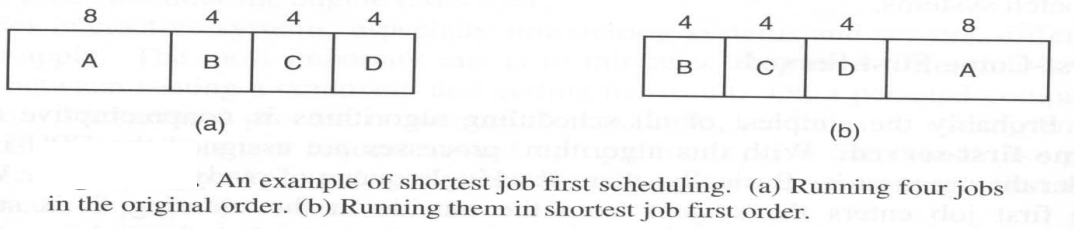
Algoritmi za batch sustave

Algoritmi za batch sustave određuju redoslijed izvođenja procesa za obrade sa malo komunikacije sa korisnicima tijekom izvođenja, pa se karakteriziraju dužim vremenima dodjele procesora procesima radi izbjegavanja gubitka vremena ažuriranja stanja procesa pri čestim dodjelama procesora. Algoritmi su:

- FIFO
- Shortest Job First
- Shortest remaining time next
- Three-Level scheduling

FIFO algoritam dodjeljuje procesor procesima po slijedu pojave procesa u sustavu. Proces koji je stigao prvi u sustav, prvi i dobiva procesor. Algoritam je jednostavan za implementaciju ali nije optimalan.

Shortest Job First je algoritam koji veći prioritet daje procesima čije je trajanje izvođenja kraće. Algoritam dodjeljuje procesor procesima poredanim po trajanju obrade od najkraće do najduže. Slika 23. prikazuje primjer za 4 procesa različitog trajanja na koje je primijenjen algoritam.



An example of shortest job first scheduling. (a) Running four jobs in the original order. (b) Running them in shortest job first order.

Slika 23. Primjer primjene algoritma Shortest Job First

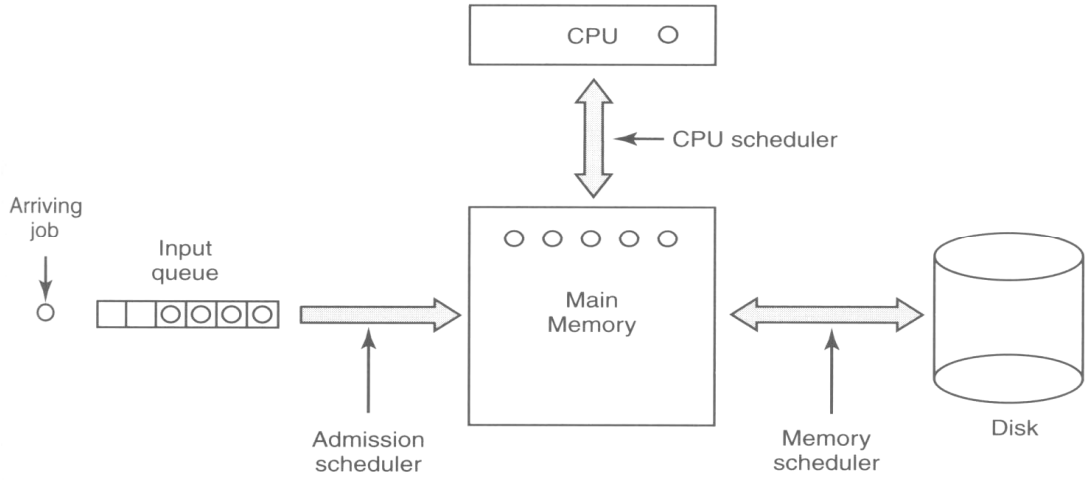
Formula u nastavku određuje prosječno vrijeme za izvođenje obrade:

$$T_{sr} = \frac{4 \cdot a + 3 \cdot b + \cdot c + d}{4} = \frac{4 \cdot 4 + 3 \cdot 4 + 2 \cdot 4 + 8}{4}$$

Vidljivo je da vrijeme trajanja izvođenja prvog procesa označeno slovom *a* 4 puta više utječe na rezultat u odnosu na vrijeme posljednje izvođenog procesa označenog slovom *d*, pa je logično da se izvode najprije procesi sa najkraćim trajanjem izvođenja.

Shortest remaining time next je modifikacija Shortest Job First algoritma jer se najprije izvodi proces čije je preostalo vrijeme izvođenja najkraće.

Three-Level Scheduling algoritam se zasniva na određivanju prioriteta na 3 nivoa: za procese u ready stanju, za procese koji se spremaju na disk i učitavaju sa diska i za procese koji se uvode u sustav. Slika 24. prikazuje 3 nivoa određivanja prioriteta. Pri uvođenju procesa u sustav određuje se prioritet među procesima, a time i redoslijed uvođenja procesa u ready stanje. Veći broj procesa može se izvoditi ako se procesi iz radne memorije kratko vrijeme pohranjuju na disku, a zatim izmjenjuju sa procesima u radnoj memoriji. Algoritam određuje redoslijed izmjene procesa između radne memorije i diska. Pri tome se ima u vidu koliko je dugo proces u radnoj memoriji, koliko je proces izvođen na procesoru, koliko je proces velik i kakva je važnost procesa. Na kraju se određuje prioritet procesa u ready stanju kako bi se odredio redoslijed dodjele procesora.



Slika 24. tri nivoa određivanja prioriteta

Algoritmi za interaktivne sustave

Algoritmi upravljanja procesorom za interaktivne sustave prilagođeni su za rada sa procesima koji imaju visok stupanj interakcija sa korisnicima. Upravljanje u Interaktivnim sustavima izvodi se sljedećim algoritmima:

- Round-Robin
- Priority Algorithm
- Multiple Queues
- Shortest process next
- Guaranted Scheduling
- Lottery Schedulig
- Fair-Share Scheduling

Round Robin algoritam

Algoritam dodjeljuje svakom procesu jednak interval vremena rada procesora koji se zove kvantum (*quantum*). Nakon dobivanja kvantuma za sve procese pokreće se novi ciklus dodjele kvantuma. Trajanje kvantuma ne smije biti kratko (vrijeme izvođenja pohrane stanja procesa iznosi 1 ms) niti predugo (na primjer kvantum od 500 ms određuje početak izvođenja 10-tog procesa za 4,5 s) pa se uzima kompromisna vrijednost od 100 ms. Proces koji dobije kvantum postavlja se na kraj liste za dodjelu procesora, proces se dodjeljuje sljedećem procesu u listi. Slika 25. prikazuje dodjelu kvantuma.

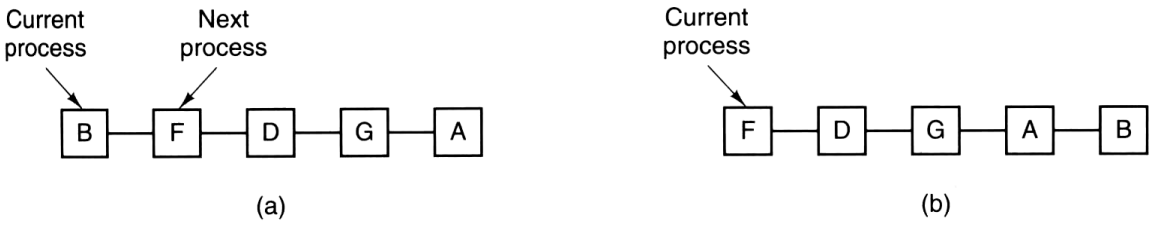


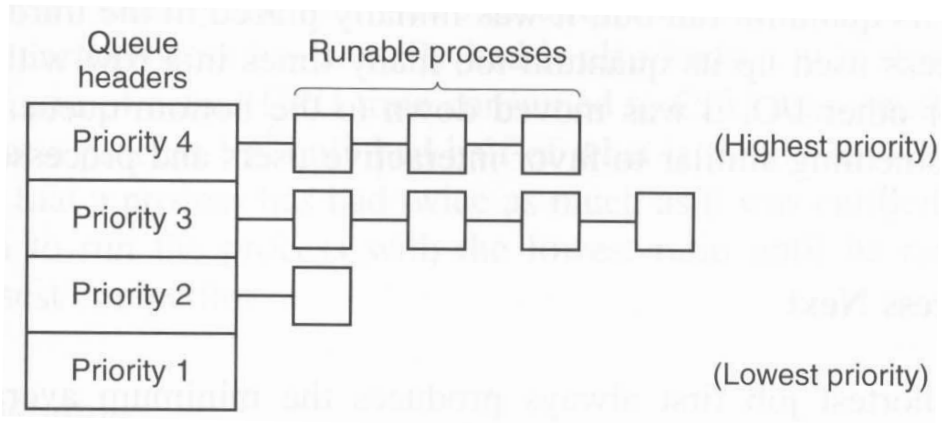
Figure 2-41. Round-robin scheduling. (a) The list of runnable processes. (b) The list of runnable processes after *B* uses up its quantum.

Slika 25. Round-Robin algoritam

Priority algoritam (algoritam vanjske dodjele prioriteta)

Algoritam se zasniva na podjeli korisnika u grupe i određivanju različitog prioriteta izvođenja za pojedine grupe. To znači da će procesi korisnika grupe višeg prioriteta uvijek imati prednost u dodjeli procesora u odnosu na procese korisnika grupe nižeg prioriteta (npr. najveći prioritet profesora, zatim studentske službe, studenata, dekanata fakulteta itd.).

Priority algoritam može se kombinirati sa drugim algoritmima. Primjer na slici 26. prezentira rad priority algoritma i Round-Robin algoritma.



Slika 26. Priority algoritam u kombinaciji sa Round-Robin algoritmom

Multiple Queues algoritam

Algoritam za svoj rad koristi dodjelu kvantuma vremena procesima, ali za razliku od Round-Robin algoritma kvantum vremena se udvostručuje za svaku novu dodjelu procesora (prvi put se dodjeli 1 kvantum, pa 2, 4, 8, 16 itd.). Na taj način se povećava prioritet ovisno o vremenu boravka procesa u sustavu.

Shortest process next

Algoritam prioritete određuje prema trajanju izvođenja procesa u interaktivnom načinu rada. Interaktivan način rada uvjetovan je dinamikom zadavanja naredbi korisnika. Zato se procjenjuje vrijeme potrebno za izvođenje nekog procesa mjerenjem prethodnih vremena između naredbi korisnika. Na taj način određuje se vrijeme potrebno za izvođenje procesa i dodjela prioriteta. Postupak procjenjivanja trajanja izvođenja procesa prema načinu izvođenja procesa u prošlosti zove se **agging**.

Guaranted Scheduling algoritam

Algoritam svakom procesu dodjeljuje jednak dio vremena rada procesora. To se postiže praćenjem broja procesa u sustavu i komparacijom proteklog vremena izvođenja procesora za svaki proces. Kako se broj procesa u sustavu dinamički mijenja uvijek je potrebno određivati prosječno vrijeme rada procesora po procesu definirano omjerom ukupnog vremena izvođenja procesora (T_o) i broja procesa (N): T_o/N. Ako je proces koji se trenutno izvodi dobio veći dio prosječnog vremena izvođenja procesora od izračunatog omjera T_o/N izvodi se kraće vremena, a ako je dobio manji interval vremena izvođenja procesora izvodi se duže vremena. Na taj način postiže se ujednačenost korištenja procesora za sve procese.

Nedostatak algoritma ogleda se u potrebi za stalnim izračunavanjem prosječnog vremena izvođenja procesora i praćenja vremena izvođenja procesora za svaki proces.

Lottery Schedulig algoritam

Algoritam se bazira na dodjeli brojeva procesima. Svaki proces dobiva određeni broj brojeva, a zatim se slučajnim metodom odabire broj iz skupa dodijeljenih brojeva. Proces koji ima taj broj dobiva procesor. Broj dodijeljenih brojeva proporcionalan je prioritetu procesora. Postoji mogućnost posudbe brojeva drugim procesima, ako proces prelazi u blokirano stanje.

Fair-Share Scheduling algoritam

Različiti broj korisnika na računalnom sustavu pokreće različit broj procesa čiji prioritet nije ovisan o vlasniku pokrenutog programa koji je generirao proces. Fire-share sheduling algoritam provjerava broj pokrenutih procesa i osigurava jednaku zastupljenost procesa svih korisnika. Svaki korisnik neovisno o broju pokrenutih procesa dobiva približno isto vrijeme rada procesora.

Algoritmi za real-time sustave

Real-time sustavi ili sustavi u realnom vremenu generiraju odgovor na postavljeni zahtjev u realnom vremenu, obično manjem od 1 s. Razlikujemo hard real-time sustave gdje je kašnjenje odgovora fatalno za sustav i soft real-time sustave gdje kašnjenje uzrokuje degradaciju parametara rada sustava ali ne i blokiranje rada cijelog sustava. Algoritmi za real-time sustava osiguravaju raspodjelu procesora između procesa koja omogućuje generiranje odziva sustava u realnom vremenu. Odziv u real-time sustavima može biti

periodičan (u točnim vremenskim razmacima) i neperiodičan. Za sustav koji zadovoljava sljedeći uvjet kaže se da je upravljiv, odnosno osigurava rad u realnom vremenu:

C_i – vrijeme rada CPU-a pri izvođenja događaja *i*

P_i – period između pojave događaja *i*

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

m - broj događaja

Algoritmi za procesne niti

Procesi tijekom izvođenja pokreću jednu ili više procesnih niti. Za procesne niti mogu biti određeni prioriteti pri izvođenju primjenom određenih algoritama (obično su to Round-Robin i priority algoritam). Procesnim nitima se može upravljati na nivou korisničkih procesa ili nivou kernela. Osnovna razlika je mogućnost izvođenja procesnih niti samo jednog procesa za korisnički nivo, dok na kernel nivou mogu biti izvođene procesne niti više procesa za vrijeme dodjele procesora jednom procesu. Izvođenje procesnih niti više procesa na kernel nivou uvjetuje pohranu stanja procesnih niti pri pokretanju procesne niti drugog procesa, dok kod korisničkog nivoa to nije potrebno. Upravljanje izvođenjem procesnih niti, odnosno određivanje prioriteta izvođenja procesnih niti optimizira izvođenje procesa u cjelini.

2.5. ZASTOJI

Zastoji su nepoželjne pojave uzrokovane raspodjelom resursa između procesa uvjetovane primjenom pseudoparalelizma. Za nastanak zastoja bitni su djeljivi resursi (*preemptable*) kojima može pristupati više resursa, za razliku od nedjeljivih (*nonpreemtable*) resursa koje može koristiti samo jedan proces. Procesi se prekidaju u izvođenju, pohranjuju im se stanja i kasnije nastavljaju izvoditi od trenutka prekida. Najčešće procesi ne uspiju rezervirati sve resurse potrebne za izvođenje u kratkom vremenu dodjele procesora pa zadržavaju resurse do sljedeće dodjele procesora. Time je omogućeno nastajanje situacija u kojima procesi potražuju resurse drugih procesa, a istovremeno zadržavaju resurse koje ti drugi procesi trebaju za obradu. Procesi jednom rezervirane resurse ne otpuštaju pa nastaju stanja koja blokiraju daljnje izvođenje procesa, a zovemo ih zastojima.

Skup procesa je u zastoju ako svaki proces iz skupa čeka na događaj koji može uzrokovati samo proces iz tog skupa procesa.

Zastoj nastaje ako su ispunjeni sljedeći uvjeti:

- mutual exclusion – sprječavanje izvođenja kritičnih sekcija za dva ili više procesa istovremeno
- hold and wait – pravo procesa da zadrži rezerviran resurs
- no preemption – onemogućavanje oduzimanja resursa procesu
- Circular wait – postojanje kružne veze između dva ili više procesa, odnosno međusobno povezivanje resursa.

Potraživanje resursa između procesa prikazuje se grafom alokacije resursa. Slika 27. prikazuje zastoj primjenu grafa alokacije resursa i zastoj dva procesa.

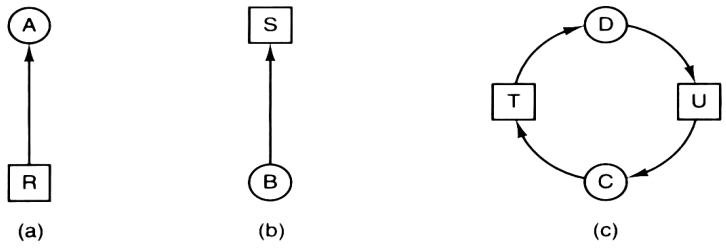
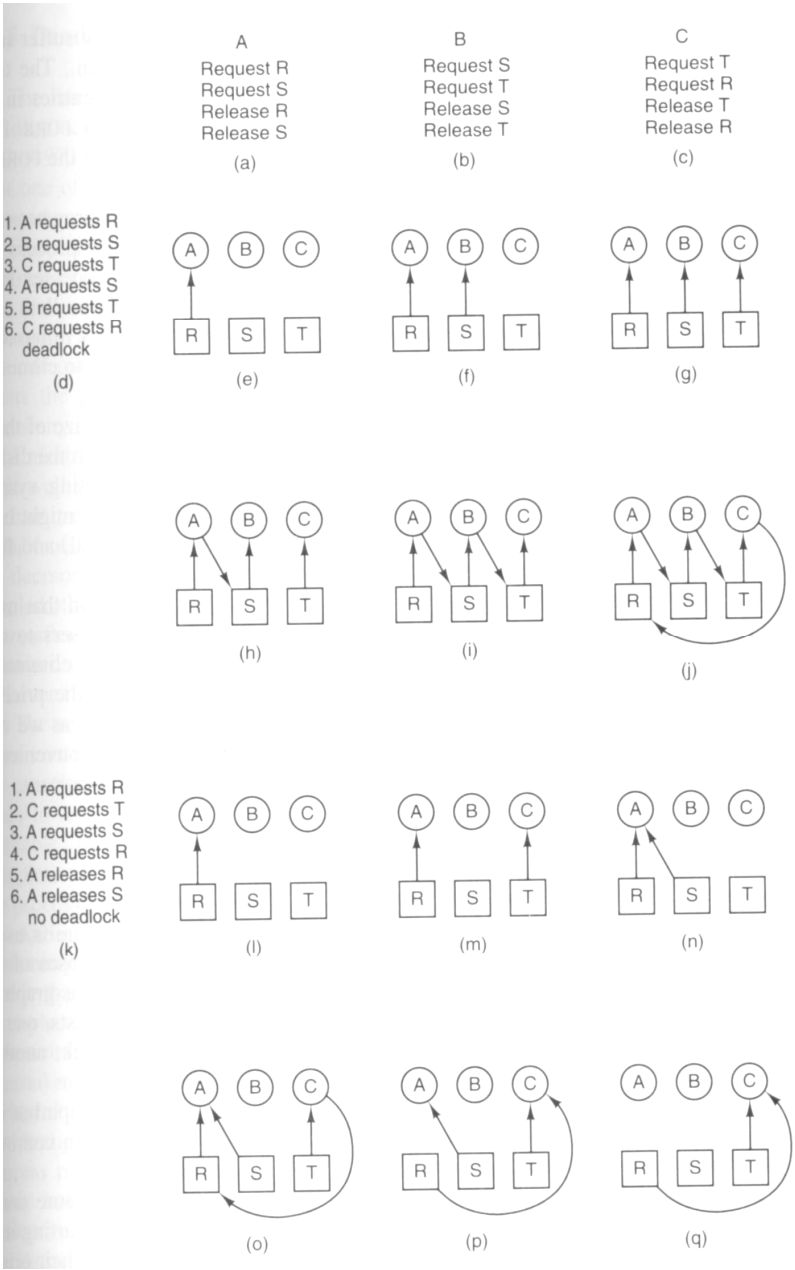


Figure 3-7. Resource allocation graphs. (a) Holding a resource. (b) Requesting a resource. (c) Deadlock.

Slika 27. Nastanak zastoja

Krug označava proces, a kvadrat resurs. Strelica usmjerena iz kvadrata u krug označava dodjelu resursa za proces, dok u obrnutom slučaju proces potražuje resurs. Zastoj nastaje zato jer proces D zadržava resurs T, a istovremeno potražuje resurs U dodijeljen procesu C koji istovremeno potražuje resurs T. Slika 28. prikazuje nastanak zastoja i otklanjanje zastoja oduzimanjem resursa procesu B. Nastanak zastoja uvjetovan je slijedom dodjele resursa (slika 28) procesima i u slučaju drugačije dodjele resursa zastoj ne bi nastao.



Slika 28. Nastanak i otklanjanje zastoja

Načini rješavanja zastojnih situacija su:

- Ostrich algoritam
- Detekcija i oporavak
- Izbjegavanje zastoja
- Sprječavanje zastoja.

Ostrich algoritam

Koristi se za operacijske sustave sa vrlo malim brojem zastoja u radu. Obrada zastoja u takvim sustavima ne postoji jer je ekonomski neopravdano implementirati obradu zastoja u operacijski sustav za mali broj zastoja. Operacijski sustavi sa malim brojem zastoja mogu doći u zastoj u određenim slučajevima, ali je pojava takvih situacija vrlo malo vjerojatna. Nastanak zastoja otklanja se ponovnim pokretanjem svih procesa i očekivanjem nastanka drugačije dodjele resursa između procesa.

Detekcija zastoja i oporavak

Detekcija zastoja ostvaruje se praćenjem stanja u grafu alokacije resursa. Pojava kružnih veza u grafu određuje nastanak zastoja. Zastoji se otklanjaju oduzimanjem resursa jednom procesu iz kružne veze i ponovnim restartanjem ostalih procesa. Ako su procesi i dalje u zastoju, resursi se oduzimaju sljedećem procesu sve dok se ostali procesi iz kružne veze ne pokrenu. Detekciju zastoja moguće je izvoditi za:

- jedan resurse određenog tipa
- više resursa određenog tipa.

Detekciju zastoja za jedan resurs određenog tipa realizira se primjenom algoritma za otkrivanje kružne veze. Algoritam je objašnjen na primjeru alokacije resursa prikazanom na slici 29. Procesi D, E i G su u kružnoj vezi pa su u zastoju. Otkrivanje kružne veze je vizualno vrlo jednostavno, no na operativnog sustava potrebno je koristiti formalni algoritam. Formalni algoritam koristi listu L za čvorove alokacijskog grafa resursa. Za svaki čvor N izvodi se sljedećih 5 koraka:

1. postavljenje prazne liste L za čvor
2. dodavanje trenutnog čvora na kraj liste L i provjera ponavljanje čvora u listi što određuje postojanje kružne veze i prekid izvođenje algoritma
3. za trenutno provjereni čvor odrediti postojanje neispitanih izlaznih strelica; ako ih ima slijedi korak 4, a ako nema slijedi korak 5
4. odabir izlazne strelice slučajnim načinom, zatim prijelaz na sljedeći čvor i ponavljanje koraka 3
5. dostignut je krajnji čvor, pa se briše posljednji čvor i vraćamo na prethodni čvor, ako je to početni čvor u listi nema kružnih veza i algoritam završava izvođenje.

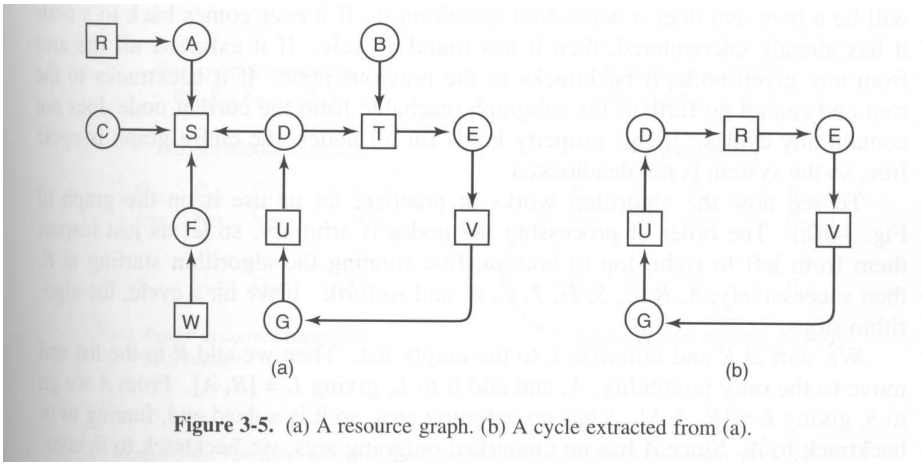


Figure 3-5. (a) A resource graph. (b) A cycle extracted from (a).

Slika 29. Alokacijski graf resursa sa kružnom vezom

Objasnimo rad algoritma na primjeru. Redoslijed analize čvorova je nebitan, pa počnimo od lijevo na desno i odozgo nadolje. Prvo analiziramo čvor R, dodavanjem prazne liste L za taj čvor i upisom oznake čvora L = [R]. Iz R postoji izlazna strelica ka A pa A dodajemo u listu L = [R,A]. Iz A strelica odlazi u S pa slijedi L = [R,A,S]. Kako iz S nema izlaznih strelica S brišemo iz liste i vraćamo se u A kako bismo provjerili postojanje neke druge izlazne strelice. Jedina izlazna strelica iz A je već provjerena pa se i A briše iz liste pa u listi ostaje samo R, a kako smo krenuli iz R algoritam za čvor R se završava. Ispitivanje čvora A bilo bi još kraće jer bi se u listu dodao čvor S, L = [A,S], no kako iz S nema izlaznih strelica, S se briše i vraćamo se u A pa algoritam terminira izvođenje. Ispitajmo sada čvor B. Nakon ispitivanja i upisa podataka o čvorovima dolazimo do čvora D koji ima dvije izlazne strelice. Vrijednost liste za čvor B je L = [B,T,E,V,G,U,D] i slučajno se odabir

jedna izlazna strelica. Neka je to strelica prema S pa se S dodaje u listu, ali zatim odmah i briše jer S nema izlaznih strelica pa se vraćamo u čvor D. Strelica prema T upisuje T u listu i provjerom liste L = [B,T,E,V,G,U,D,T] uočava se ponavljanje čvora što određuje nastanak kružne veze.

Detekcija zastoja za više resursa istog tipa koristi se matrica za provjeru pojave kružne veze u grafu. Potrebno je koristiti dva vektora i dvije matrice. Vektor E se koriste za pohranu podataka o broju postojećih resursa određenog tipa, a vektor A za podatke o broju raspoloživih resursa određenog tipa. Matrica C sadrži podatke o dodijeljenom broju resursa procesima za svaki tip resursa, dok matrica R evidentira broj resursa svakog tipa koji potražuje proces. Slika 30. prikazuje vektore E i A, te matrice C i R.

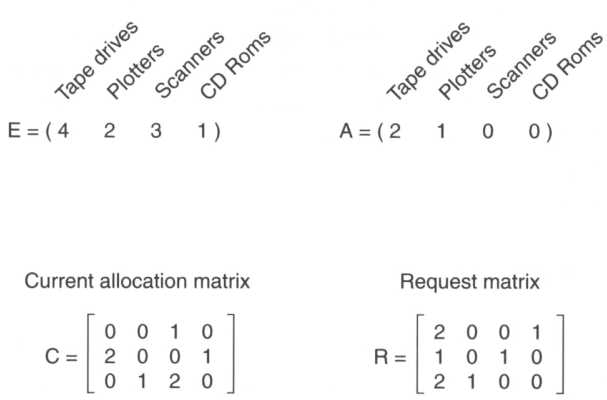


Figure 3-7. An example for the deadlock detection algorithm.

Slika 30. Prikaz vektora E i A i matrica C i R

Vrijedi formula $\sum_{i=1}^{n} C_{ij} + A_j = E_j$, odnosno broj dodijeljenih i potraživanih resursa jednak je ukupnom broju resursa. Algoritam postojanje zastoja bazira na usporedbi vektora A i vektora izvedenih iz matrice R (jedan red matrice R). Izvođenje procesa je moguće ako je vektor A veći ili jednak izvedenom vektoru iz R (svaki element iz A je jednak ili veći od pripadajućeg elementa iz R). Algoritam primjenjuje slijedeće korake:

1. provjera postojanja procesa P_i, za koji je i-ti redak iz R manji od vektora A
2. za takav proces dodaje se i-ti redak iz vektora C u vektor A, označava se proces i ponavlja korak 1.
3. ako takav proces ne postoji algoritam se terminira.

Slika 31 prikazuje mogući slučaj alokacije resursa za 3 procesa. Prvi proces se ne može izvoditi jer je 1 red matrice R veći od vektora A (CD Rom nije dostupan), drugi proces se također ne može izvoditi jer je skener nedostupan, ali treći proces može bit pokrenut jer je 3 redak iz matrice R jednak vektoru A. Matrica A će se uvećati za vrijednost 3 retka matrice C pa će imati vrijednost A = (2220) što će omogućiti izvođenje 2 procesa koji će nakon završetka izvođenja matrici A promijeniti vrijednost na A = (4221). Ukoliko bi proces 3 na početku potraživao CD Rom (3 red u matrici R bi imao vrijednost 2101) proces se ne bi mogao izvoditi i nastao bi zastoj. Pokretanje algoritma je vremenski zahtjevno i izvodi se ili nakon određenog vremenskog intervala ili kada se uoči smanjenje iskoristivosti CPU (zastoj uzrokuje blokiranje procesa i smanjenje broja procesa u ready stanju što uvjetuje smanjenje iskoristivosti CPU-a).

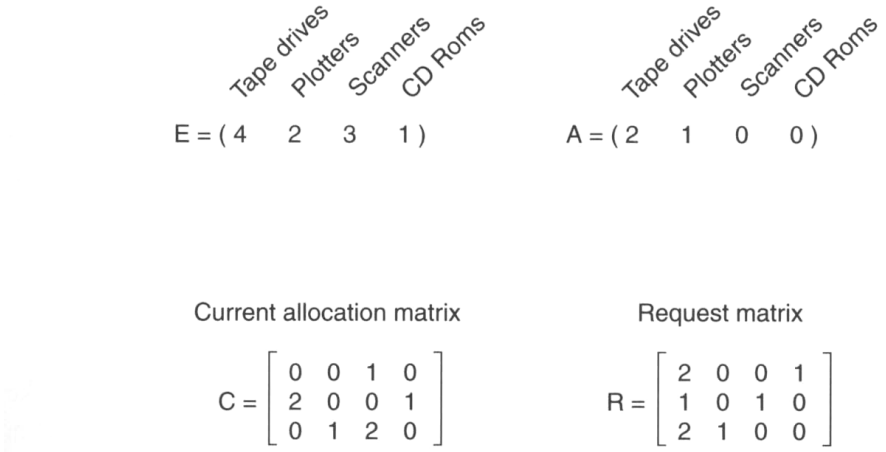


Figure 3-7. An example for the deadlock detection algorithm.

Slika 31. Primjer izvođenja algoritma za otkrivanje zastoja

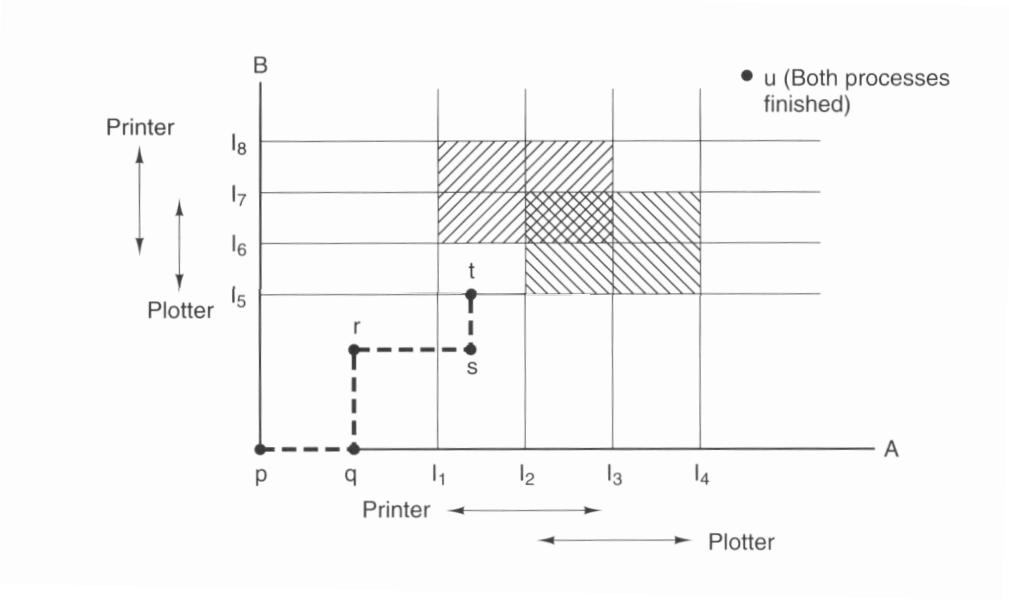
Oporavak se izvodi:

- otpuštanjem resursa – resurs se oduzima procesu i dodjeljuje nekom drugom procesu
- vraćanjem procesa u raniju fazu izvođenja (rollback) – proces u izvođenju ima pohranjene kontrolne točke (checkpoint), te se proces vraća na kontrolnu točku u kojoj nije rezervirao problematičan resurs uključen u kružnu vezu

- uništavanjem procesa – terminiranjem procesa iz kružne veze oslobađaju se rezervirani resursi i omogućuje nastavak izvođenja ostalih procesa; dozvoljeno je i terminiranje procesa koji nisu u kružnoj vezi ako time utječemo na rješavanje zastoja.

Izbjegavanje zastoja

Izbjegavanje zastoja zasniva se na provjeri uvjeta koji bi mogli dovesti do pojave zastoja i nedozvoljavanju nastavka izvođenja procesa u takvim situacijama. Slika 32. prikazuje trajektorije resursa procesa. Osjenčani dijelovi u grafikonu određuju vremenski interval u kojem proces potražuje resurs. Ne smije se dozvoliti da trajektorija resursa procesa (isprekidana linija) ne uđe u dvostruko osjenčani dio grafa.



Slika 32. Trajektorija rezerviranja resursa

Mogućnost nastanka zastoja provjerava se postojanjem sigurnih stanja sustava. Sigurna stanja dozvoljavaju izvođenje procesa i dodjelu svih resursa koje procesi mogu koristiti u izvođenju. Nesigurna stanja mogu dovesti do zastoja ako procesi zatraže maksimalno dozvoljeni broj resursa, no to ne znači da je pojava zastoja sigurna jer procesi mogu zatražiti i manji broj resursa od maksimalno dozvoljenog. Određivanjem nesigurnog stanja

izbjegava se i teoretska mogućnost nastanka zastoja. Slika 33 prikazuje sigurno stanje jer je broj slobodnih resursa dovoljan za zadovoljenje maksimalnih potreba procesa za resursima.

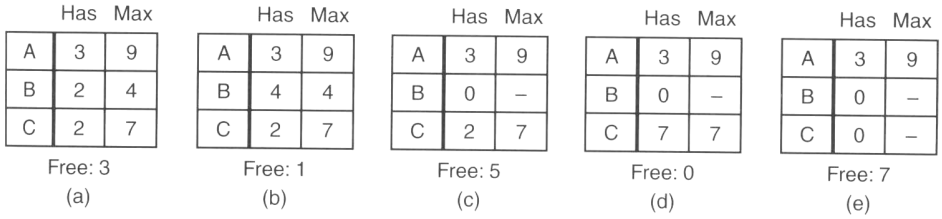


Figure 3-9. Demonstration that the state in (a) is safe.

Slika 33. Sigurno stanje

Slika 34 prikazuje nastanak nesigurnog stanja jer nije moguće zadovoljiti maksimalne zahtjeve procesa za resurse.

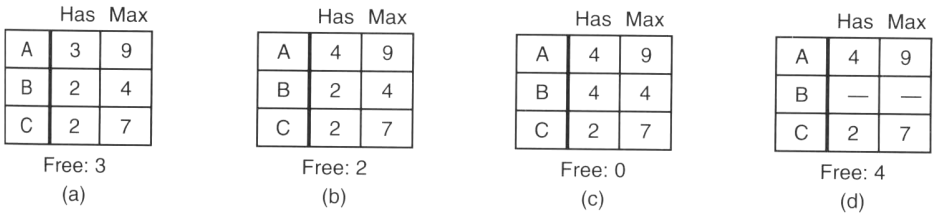


Figure 3-10. Demonstration that the state in (b) is not safe.

Slika 34. Nesigurno stanje

Kontrola sigurnih i nesigurnih stanja izvodi se primjenom Bankerovog algoritma za iste resurs svakog tipa (*single resource*) i Bankerovog algoritma za više resursa svakog tipa (*multiple resource*).

Bankerov algoritam za single resurse dodjeljuje resurse istog tipa procesima provjeravajući da li je stanje nakon dodjele sigurno. Ako je stanje nesigurno ne dozvoljava se daljnje izvođenje procesa. Slika 35 prikazuje rad Bankerovog algoritma za sigurna stanja. Algoritma provjerava mogu li slobodno resursi zadovoljiti maksimalni broj zahtijevanih resursa za barem jedan proces. Ukoliko je to moguće stanje je sigurno i ne može se dozvoliti nastavak izvođenja procesa.

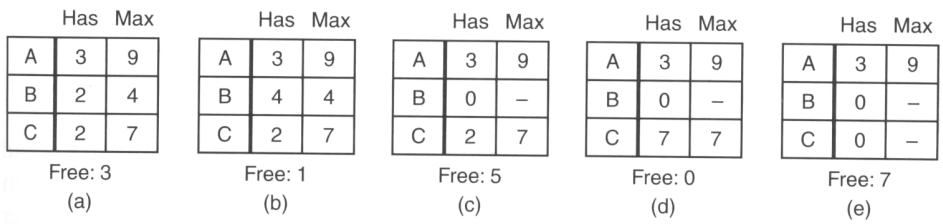


Figure 3-9. Demonstration that the state in (a) is safe.

Slika 35. Bankerov algoritam za single resurse

Bankerov algoritam za multiple resurse provjerava postojanje sigurnih stanja za veći broj resursa istog tipa koji se dodjeljuju procesima. Algoritam za rad koristi dvije matrice: matrica C za prikaz podataka o trenutno dodijeljenim resursima i matrica R za podatke o potrebnim resursima za izvođenje procesa. Algoritam koristi i tri vektora: vektor E određuje ukupni broj resursa za svaki tip resursa, vektor P trenutno pridružene resurse i vektor A trenutno raspoložive resurse. Rad algoritma izvodi se primjenom slijedećih koraka:

1. traži se proces P_i koji u matrici R ima vrijednosti koje su jednake ili manje od vektora A
2. proces se označava kao završen i dodaju se vektoru A vrijednosti iz reda matrice C koji pripada označenom procesu.
3. postupak se ponavlja sve dok nisu ispitani svi procesi; ako se ispitaju svi procesi stanje je sigurno, a ako za neki proces korak 1 ne može biti izveden stanje je nesigurno.

Slika 36 prikazuje rad Bankerovog algoritma za multiple resurse. Na slici je prezentirano sigurno stanje.

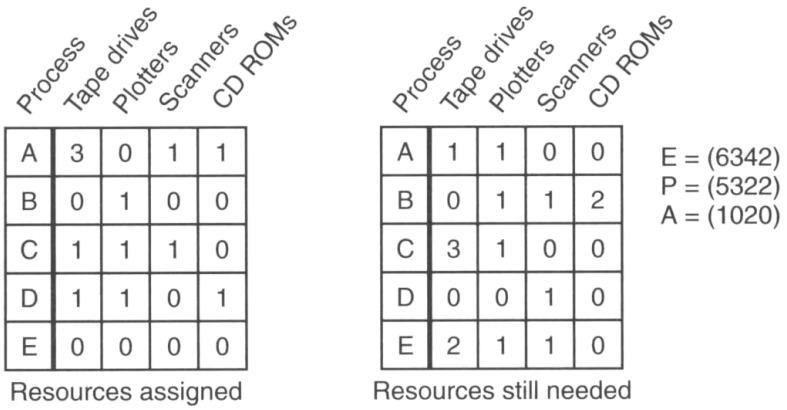


Figure 3-12. The banker's algorithm with multiple resources.

Slike 36. Bankerov algoritam za multiple resurse

Sprječavanje zastoja

Sprječavanje zastoja postiže se onemogućavanjem nastanka jednog od uvjeta za pojavu zastoja. Najbolje rješenje se postiže onemogućavanjem kružne veze. Mutual exclusion može biti onemogućen ali samo uz primjenu posebnog procesa *deamon* koji kontrolira pristup i rad sa resursom, odnosno potrebno je koristiti *spooling* u svim slučajevima gdje postoji potreba za mutual exclusion. Mogućnost zadržavanje resursa također nije jednostavno isključiti jer bi procesi morali odjednom rezervirati sve resurse potrebne za izvođenje, a ponekad procesi unaprijed ne mogu znati koji resursi su potrebni za izvođenje. Oduzimanje resursa nije praktično jer oduzimanje određenih resursa uzrokuje nemogućnost nastavka izvođenja započetih obrada (oduzimanje printera u trenutku ispisa nekog dokumenta). Preostaje sprječavanje nastanka kružne veze. Označavanjem resursa sa brojevima i primjenom pravila za dodjelu resursa može se spriječiti kružna veza. Pravilo ne dozvoljava dodjelu resursa ako je broj traženog resursa manji od broja resursa koji je procesu dodijeljen. U tom slučaju proces mora otpustiti dodijeljene resurse i pokušati ponovo rezervirati resurse. Time se izbjegava zastoj (slika 37).

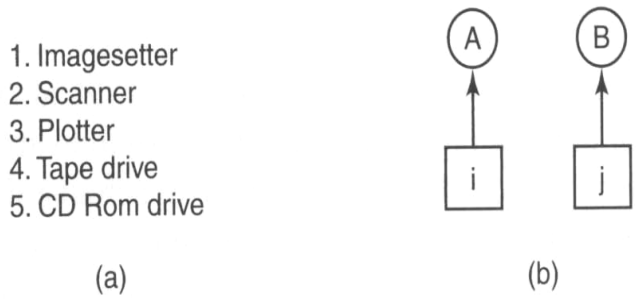


Figure 3-13. (a) Numerically ordered resources. (b) A resource graph.

Slika 37. Primjena numeracije resursa

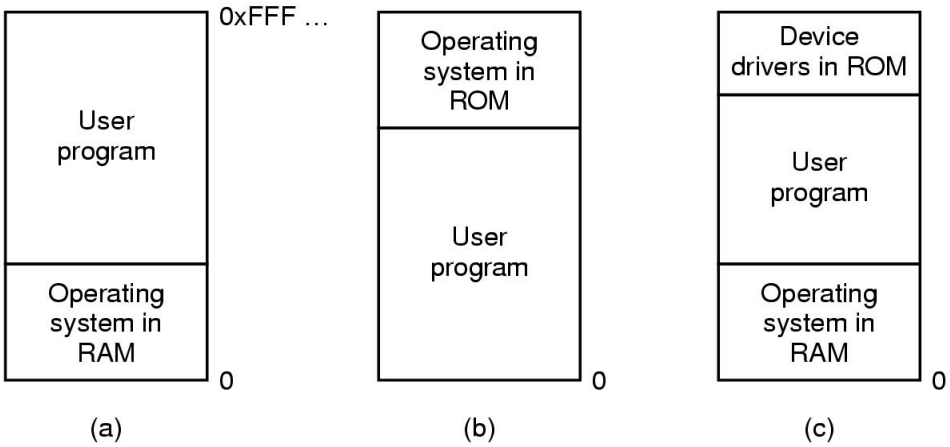
4. UPRAVLJANJE MEMORIJOM

Upravljanje memorijom je ključna funkcija operativnog sustava nužna za izvođenje procesa. Integriranost upravljanja procesima, upravljanja memorijom i upravljanja datotečnim sustavom omogućuje učitavanje procesa sa vanjske memorije u radnu memoriju, pokretanje procesa, pohrana rezultata obrade i spremanje podataka na vanjsku memoriju. Zadatak upravljača memorije je smještaj procesa u memoriju, pristup adresnom prostoru procesa, ažuriranje slobodnog prostora vanjske memorije i usklađivanje različitosti brzine rada memorija (cache memorija, registri, RAM memorija).
Upravljanje memorijom razvija se usporedno sa tehnološkim karakteristikama memorije i zahtjevima prema korištenju memorije. Razlikujemo sljedeće tehnike upravljanja memorijom:

1. upravljanje memorijom u monoprogramiranju
2. upravljanje memorijom u multiprogramiranu
3. swapping
4. virtualna memorija.

Upravljanje memorijom za monoprogamiranje

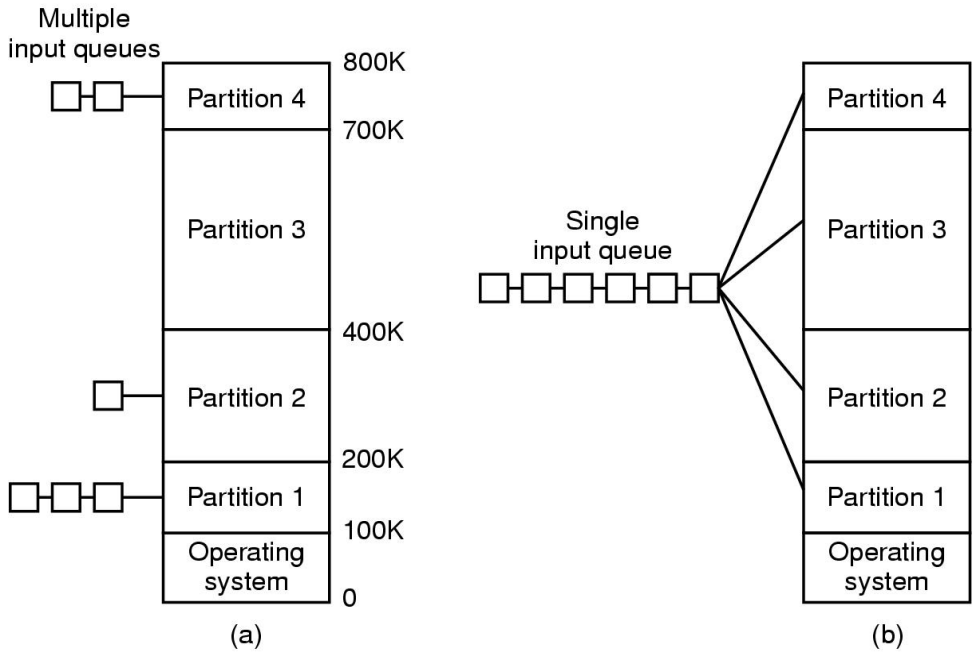
Za batch obradu upravljač memorijom učitava program u memoriju, a nakon izvođenja programa oslobađa se rezervirana memorija te učitava novi proces. Operativni sustav može biti učitan u radnu memoriju, djelomično smješten u radnu memoriju i ROM memoriju ili samo u ROM memoriju (slika 38. pod a, b i c). Danas prevladava učitavanje operativnog sustava djelomično u ROM memoriju (BIOS) i djelomično u radnu memoriju.



Slika 38. Učitavanje operativnog sustava u batch obradi

Upravljanje memorijom za multiprogramiranje

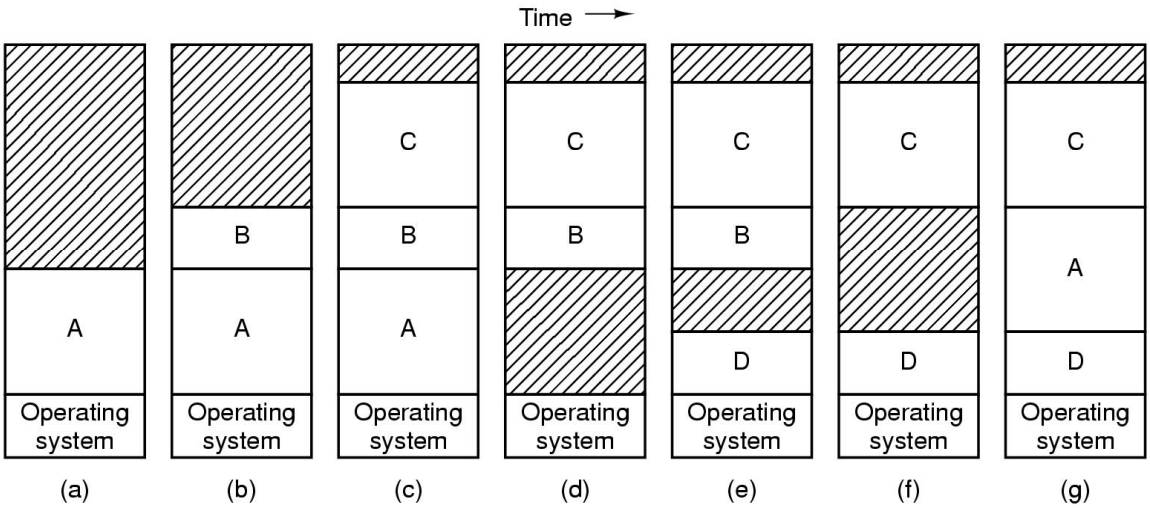
Izvođenje više programa omogućeno je podjelom radne memorije na particije i učitavanjem procesa u jednu od particija. Particije su različite veličine i procesi se učitavaju ovisno o njihovoj veličini (proces se učitava u prvu veću particiju). Slika 39. a) prikazuje podjelu memorije na particiju i nastanak redova čekanja na učitavanje na pojedinim particijama i istovremeno nepopunjenost jedne particije. Generiranjem jednog reda za učitavanje svih procesa izbjegava se čekanje na učitavanje procesa na nekim particijama dok su druge particije prazne jer se proces iz reda učitava u prvu slobodnu particiju neovisno o veličini particije (slika 39. b)). Primjena modifikacije izbjegava čekanje na dodjelu particija ali ne osigurava racionalno učitavanje procesa. Učitavanje procesa u particije uvjetuje rješavanje problema realokacije procesa i zaštite adresnog prostora procesa. Realokacija procesa izvodi se uvećavanjem adrese procesa za početnu adresu particije u kojoj je proces učitan (100 K procesa učitan u particiju na 400 K znači 100 + 400 = 500K). Zaštita adresnog procesa ostvaruje se primjenom dvaju registara *base* i *limit*. Pri izvođenju procesa u registar *base* učitava se početna adresa particije procesa, a u *limit* veličina particije. Svaka adresa izvan vrijednosti *base* i *base+limit* nije korektna i sustav je ne prihvaća.



Slika 39. Upravljanje memorijom u multiprogramiranju

Swapping

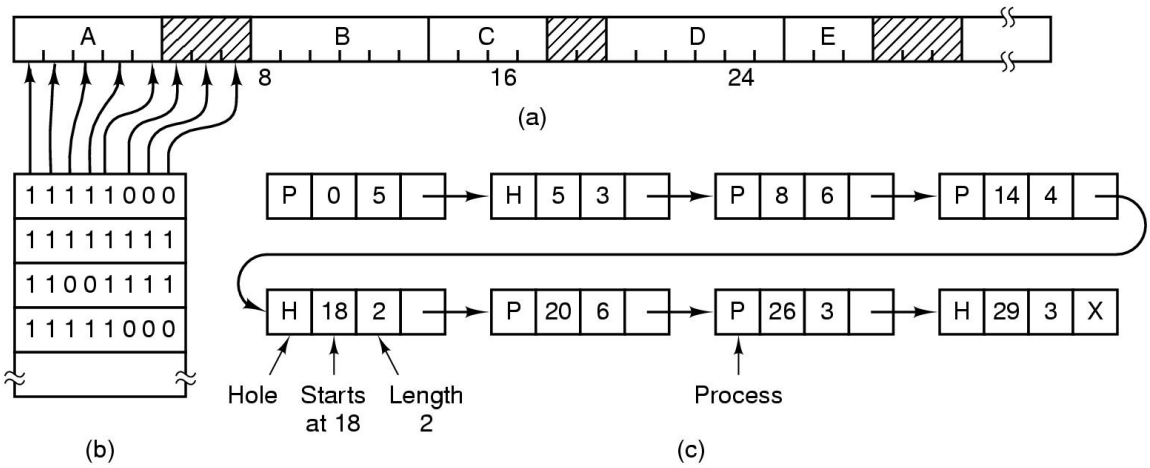
Swapping se zasniva na izmjeni procesa između radne i vanjske memorije. Procesi se učitavaju jedan do drugoga ostavljajući određeni prostor za širenje procesa (povećanje podataka raste prema višim, a stack-a prema nižim adresama i ne smije se dozvoliti njihovo preklapanje pa se proces proširuje na više adrese kako bi se izbjeglo njihovo preklapanje). Nakon oslobađanja adresnog prostora procesa u nastalu prazninu učitava se novi u pravilu manji proces. Nastaje manja praznina u koju se učitava neki manji proces, no u jednom trenutku više nema dovoljno malih procesa koji se mogu učitati u nastalu prazninu. Praznina se više ne može koristiti i zovemo je fragmentom, a cijelu pojavu eksternom fragmentacijom. Realokacijom memorije izvodi se pomicanjem adresa procesa jedan prema drugom te je moguće spojiti sve zasebne fragmente u jednu prazninu koja se može iskoristiti za smještaj novog procesa, no postupak realokacije memorije uvjetuje aktivnost računalnog sustava pa nije racionalno često izvođenje postupka. Slika 40. prikazuje nastanak eksterne fragmentacije i realokaciju memoriju.



Slika 40. Nastanak eksterne fragmentacije

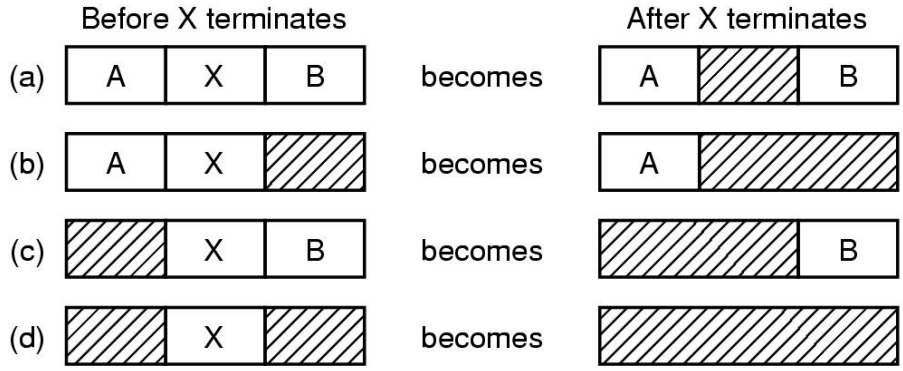
Ažuriranje slobodnog prostora memorije

Ažuriranje se izvodi primjenom bit mapa ili povezanih listi. Memorija se dijeli na dijelove iste veličine i ažurira se stanje svakog dijela memorije. Bit mape imaju onoliko bitova koliko je dijelova memorije. Vrijednost bita 1 označava zauzetost, a vrijednost 0 nezauzetost. Za smještaj procesa u memoriju potrebno je pronaći slijed 0 (nula) dovoljan za smještaj procesa. Implementacija bit mapa je jednostavna ali je njihovo pretraživanje dugotrajno, pa se češće koriste povezane liste. Povezane liste sadrže zapise sa podatkom o praznini ili procesu, početnom dijelu memorije u koji je proces ili praznina smješten i broju dijelova memorije. Kako je za smještaj praznina potrebna samo lista praznina često se povezane liste podijele na liste procesa i listu praznina. Primjena bit mapa i povezanih lista prezentirana je na slici 41.



Slika 41. Primjena bit mapa i povezanih lista.

Na slici 42. prikazan je postupak povezivanja praznina nakon završetka izvođenja procesa. Svaka od prikazanih situacija uvjetuje različit način ažuriranja lista procesa i praznina.



Slika 42. Povezivanje procesa i praznina nakon završetka izvođenja procesa

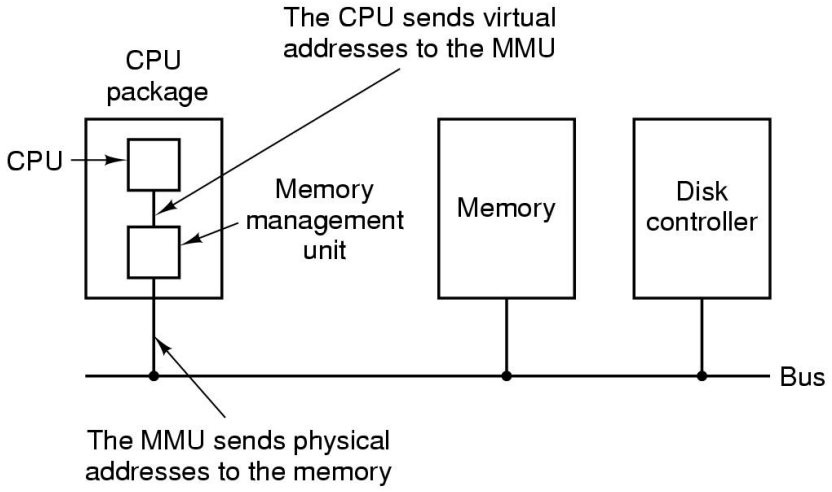
Pri učitavanju procesa mogu se koristiti sljedeće strategije:

1. prvo popunjavnje (*first fit*) – analiziraju se memorijske praznine počevši od nulte adrese i proces se smješta u prvu veću prazninu; jednostavna implementacija ali smještaj procesa nije optimalan
2. sljedeće popunjavanje (*next fit*) – isto kao first fit, ali se pri smještaju novog procesa ne polazi od nulte adrese već od posljednje korištenje praznine za smještaj procesa; rezultati su bolji od *first fit-a*.

3. najbolje popunjavanje (*best fit*) – unaprijed se analiziraju sve praznine i proces se smješta u prazninu sa najmanje neiskorištenog prostora nakon učitavanja procesa; primjenjuje se kada se očekuje povećanje procesa u budućem vremenu.
4. najlošije popunjavanje (*worst fit*) – analiziraju se sve praznine i proces se smješta u prazninu sa najviše neiskorištenog prostora nakon smještaja procesa; primjenjuje se kada se očekuje smanjenje veličine procesa u budućem vremenu.
5. brzo popunjavanje (best fit) – praznine se ažuriraju u liste praznina ovisno o veličini praznina i proces se smješta u slobodnu prazninu odgovarajuće veličine; smještanje je brzo ali je ažuriranje listi praznina vremenski zahtjevno.

Virtualna memorija

Virtualna memorija omogućuje izvođenje procesa čiji adresni prostor prelazi veličinu radne memorije. To se postiže djelomičnim učitavanjem procesa u virtualnu memoriju (rezervirani memorijski prostor na vanjskoj memoriji, odnosno hard disku) i djelomičnim učitavanjem u radnu memoriju. Kada je potrebni pristupiti adresnom prostoru procesa koji nije učitan u radnu memoriju, proces se prevodi u blokirano stanje te se u radnu memoriju učitava traženi adresni prostor procesa iz virtualne memorije kako bi se moglo nastaviti izvođenje procesa. Rad sa virtualnom memorijom moguć je primjenom memorijske upravljačke jedinice (*MMU*) integrirane u CPU (slika 43.)



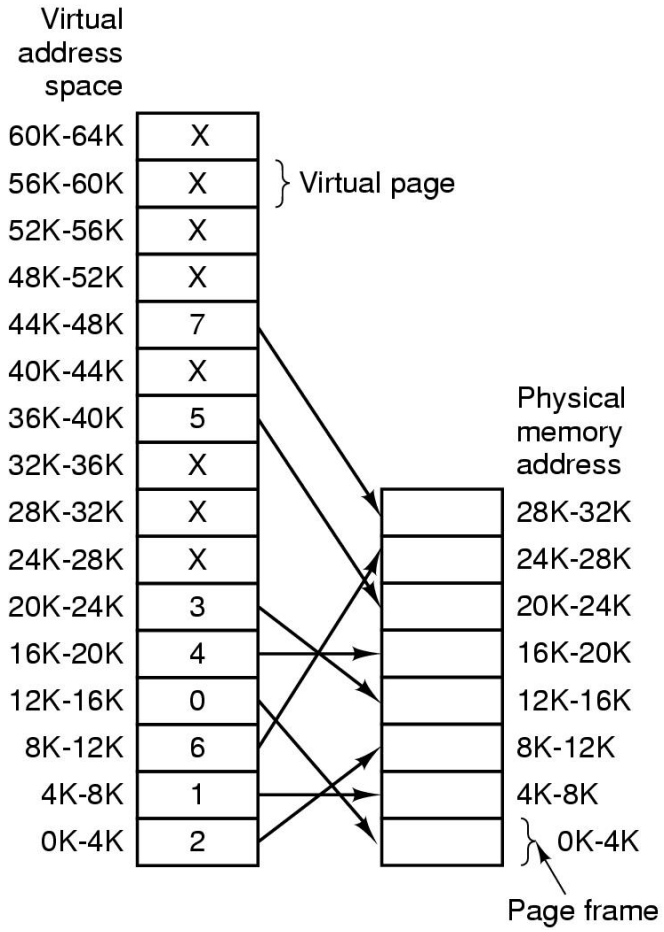
Slika 43. Prikaz MMU-a u radu sa virtualnom memorijom

Tehnike rada sa virtualnom memorijom su:

- straničenje
- segmentcija.

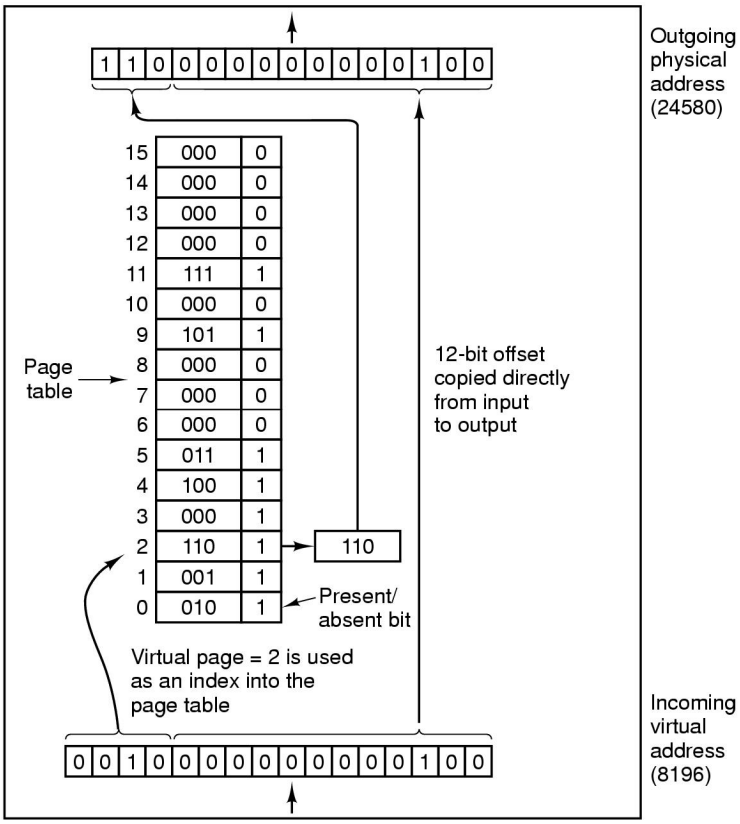
Straničenje

Straničenje se zasniva na podjeli virtualne i radne memorije na dijelove iste veličine. Dijelovi u virtualnoj memoriji su stranice virtualne memorije (*pages*), a dijelovi u radnoj memoriji su stranice radne memorije (*page frames*). Slika 44. prikazuje primjer učitavanja stranica iz virtualne u radnu memoriju za veličine virtualne memorije 64 KB i radne memorije 32 KB te veličinu stranice od 4 KB. Virtualna memorija ima 16 stranica, dok radna memorija ima 8 stranica.



Slika 44. Prikaz učitavanja stranica

Podaci o povezivanje stranica virtualne i radne memorije pohranjuju se u tabelu stranica. Tabela stranica ima broj zapisa jednak broju stranica u virtualnoj memoriji. Svaki zapis ima podatak o učitanosti stranice u radnu memoriju (*present/apsent bit*), adresu stranice u radnoj memoriji i određene bitove (bit modifikacije, referenciranja itd). Ako je stranica učitana u radnu memoriju, vrijednost bita prisutnosti je 1 i zapis u tabeli stranica sadrži adresu stranice radne memorije u koju je učitana stranica iz radne memorije. Ukoliko stranica nije učitana generira se zahtjev za učitavanjem stranice (*page fault*), nakon čega se pokušava pronaći slobodna stranica u radnoj memoriji u koju se učitava tražena stranica i ažurira tabela stranica. Ako nema slobodnih stranica nužno je primjenom odgovarajućeg algoritma odabrati jednu stranicu za brisanje i na njeno mjesto učitati traženu stranicu. Ukoliko odabrana stranica za brisanje nije mijenjala sadržaj nakon učitavanja u radnu memoriju nova stranica se može odmah učitati u radnu memoriju na mjesto stranice za brisanje, no ako je odabrana stranica mijenjana nužno je prije njene zamjene pohraniti stranicu na vanjsku memoriju. Nakon zamjene nužno je ažuriranje tabele stranica. Pri određivanju adrese u radnoj memoriji nužno je korištenje tabele stranica, pa se memoriji pristupa dva puta: prvi put u tabelu stranica, a zatim na konkretnu memorijsku adresu procesa. Veličina procesa u pravilu nije višekratnik veličine stranica pa je posljednja stranica djelomično nepopunjena. Kako se stranica ne može dijeliti, nepopunjeni prostor posljednje stranice procesa ne može se iskoristiti pa nastaje fragment. Pojavu fragmenata tijekom straničenja zovemo internom fragmentacijom i zanemariva je u odnosu na eksternu fragmentaciju. Slika 45. prikazuje postupak određivanja adrese u radnoj memoriji. Adresa u virtualnoj memoriji za primjer na slici 44. sadrži 16 bita, pri čemu prva 4 bita određuju zapis u tabeli stranica, a ostalih 12 bitova određuje adresu unutar stranice – *offset* (4 KB se adresira sa 14 bitova). Kako virtualna memorija ima 16 stranica potrebna su 4 bita za njeno adresiranje. Na slici prvi dio adrese određuje zapis sa oznakom 2 koji ima bit prisutnosti 1 pa je stranica učitana u radnu stranicu sa oznakom 6 (110 potrebna su tri bita jer radna memorija ima 8 stranica). Oznaka 110 dodaje se na početak adrese u radnoj memoriji nakon čega slijedi *offset*.



Slika 45. Određivanje adrese u radnoj memoriji

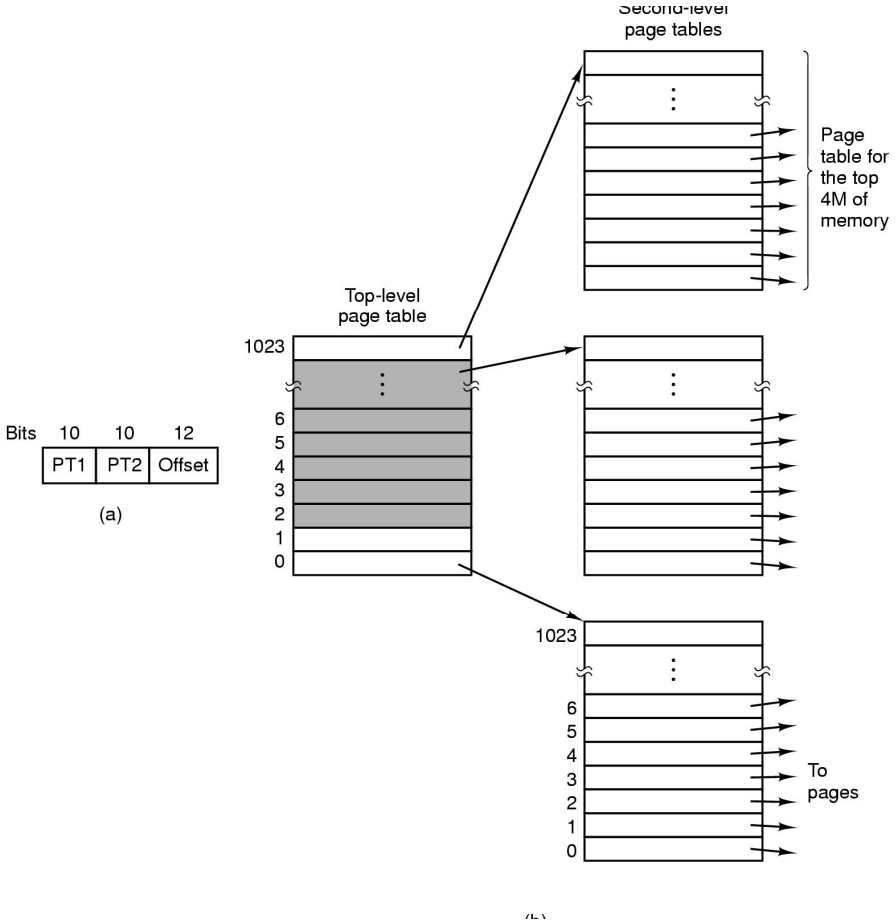
Tehnika straničenja ima sljedeće modifikacije:

- straničenje sa više nivoa
- korištenje registara
- inverzno straničenje.

Straničenje sa više nivoa primjenjuje se u slučaju velikog broja zapisa u tabeli stranica jer je pretraživanje stranica sporo. Modifikacijom se tabela stranica dijeli na manje dijelove i uvodi se glavna stranica koja postupak određivanja adrese preusmjerava na neku od tabela stranica. Slika 46. prikazuje primjenu straničenja sa više nivoa. Primjenom modifikacije pretraživanje traje kraće ali je potrebno jednom više pristupiti memoriji.

Korištenjem registara izbjegava se pristup memoriji radi čitanja tabele stranica. Koriste se registri sa strukturom podataka identičnoj zapisima iz tabele stranica. Pri pokretanju nekog procesa učitavaju se zapisi tog procesa iz tabele stranica u registre. Pri određivanju adrese

koriste se zapisi u registru pa nije potrebno pristupiti tabeli stranica u memoriji. Nedostatak registara je relativno mali kapacitet (obično do 64 zapisa) pa je za procese koji imaju više zapisa u tabeli stranica nužno pristupiti tabeli stranica ako traženi zapis nije upisan u registre. Slika 47. prikazuje zapise registara.

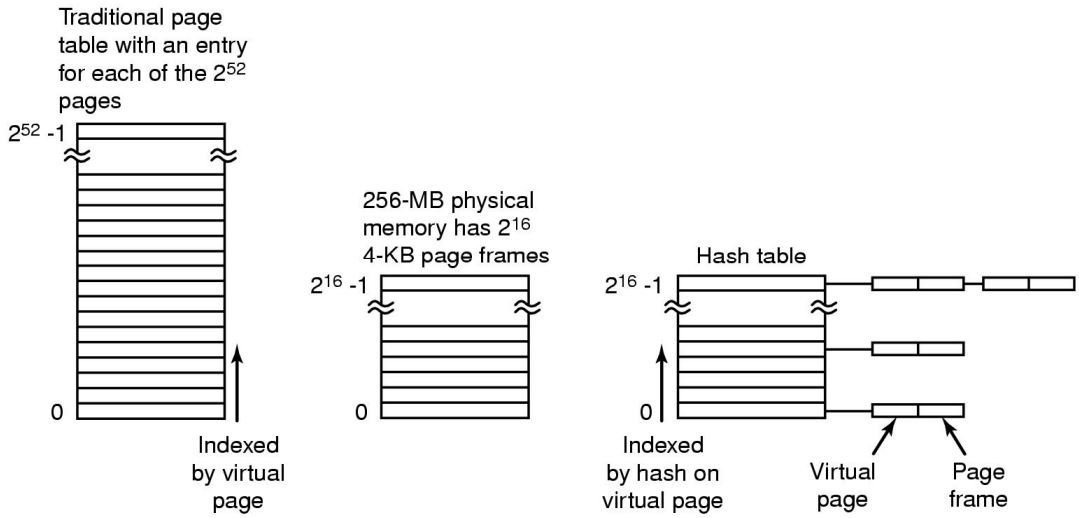


Slika 46. Primjena straničenja sa više nivoa

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

Slika 47. Sadržaj registra

Inverzno straničenje podrazumijeva invertiranu sliku tabela stranica. Tabela stranica sadrži podatke o stranicama u radnoj memoriji (8 zapisa za slučaj na slici 44.) pa je stranica sa manje zapisa. Problem nastaje pri pretraživanju stranice jer se moraju pretražiti svi zapisi u tabeli stranica da se utvrdi da stranica virtualne memorije nije u radnoj memoriji. U ovom slučaju bit prisutnosti označava da li je stranica u radnoj memoriji zauzeta ili slobodna. Slika 48. prikazuje primjenu inverznog straničenja.



Slika 48. Primjena inverznog straničenja

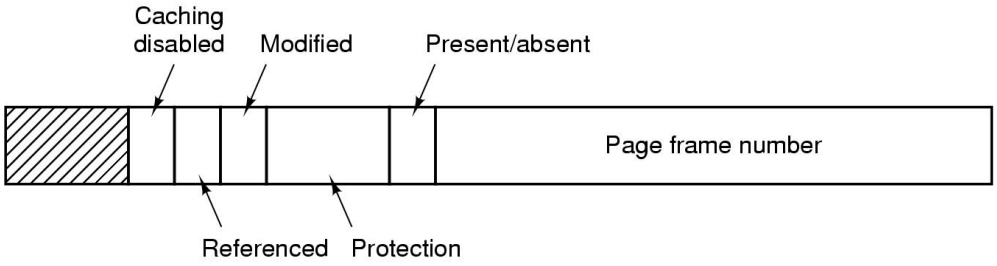
Algoritmi za zamjenu stranica

Pri odabiru stranica za brisanje koriste se sljedeći algoritmi:

- Optimalni
- Not Recently Used – NRU
- FIFO
- Second Chance
- Satni
- Last Recently Used LRU
- Not Frequently used NFU
- Agging
- Radni skup stranica algoritam
- WSClock.

Optimalni algoritam određuje stranicu koja se više neće koristiti, no kako se ne može sigurno odrediti kada stranica više neće biti korištena jer je nemoguće odrediti buduće ponašanje procesa, algoritam ima samo teoretski značaj.

Not Recently Used – NRU algoritam briše stranicu koja nije korištena u kratkom vremenskom intervalu. Za rad algoritma koristi se bit referenciranja iz tabele stranica (slika 49).



Slika 49. Zapis tabele stranica

Nakon 20 ms svim zapisima u tabeli stranica se bit referenciranja postavlja na 0, nakon čega svaka korištena stranica ima promijenjen bit referenciranja. Pri odabiru stranice odabire se stranica ovisno o vrijednostima bitova iz tabele stranica, odnosno stranica se svrstava u sljedeće klase:

Klasa 0: nije referencirana, nije modificirana

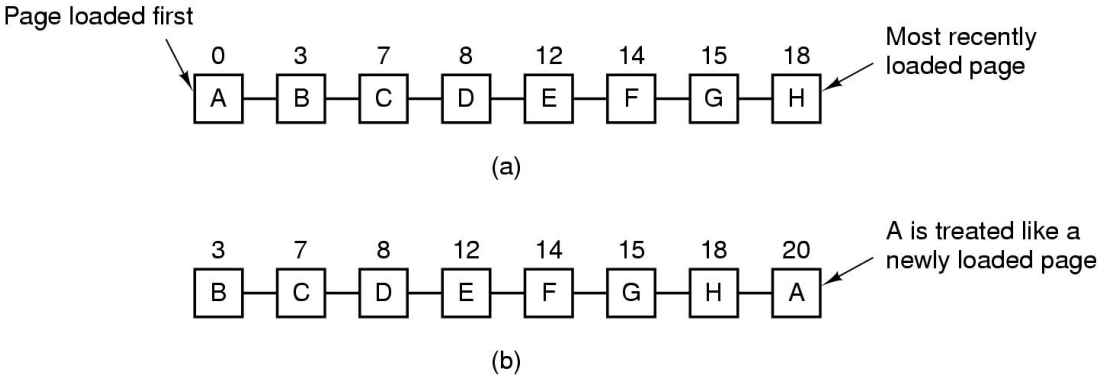
Klasa 1: nije referencirana, modificirana

Klasa 2: referencirana, nije modificirana

Klasa 3: referencirana, modificirana.

Najprije se bira stranica klase 0, ako nema takvih stranica bira se stranica klase 1 itd. Postavljanjem bit referenciranja na 0 svakih 20 ms osigurava se odabir stranice koja nije korištena u kratkom intervalu vremena.

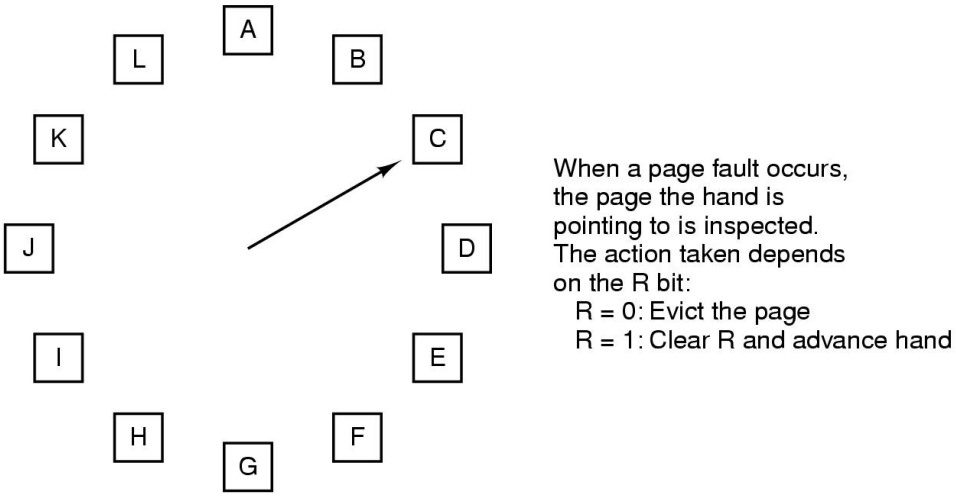
FIFO algoritam odabire stranice za brisanje prema redoslijedu učitavanja u memoriju. Slika 50. prikazuje rad FIFO algoritma.



Slika 50. FIFO algoritam

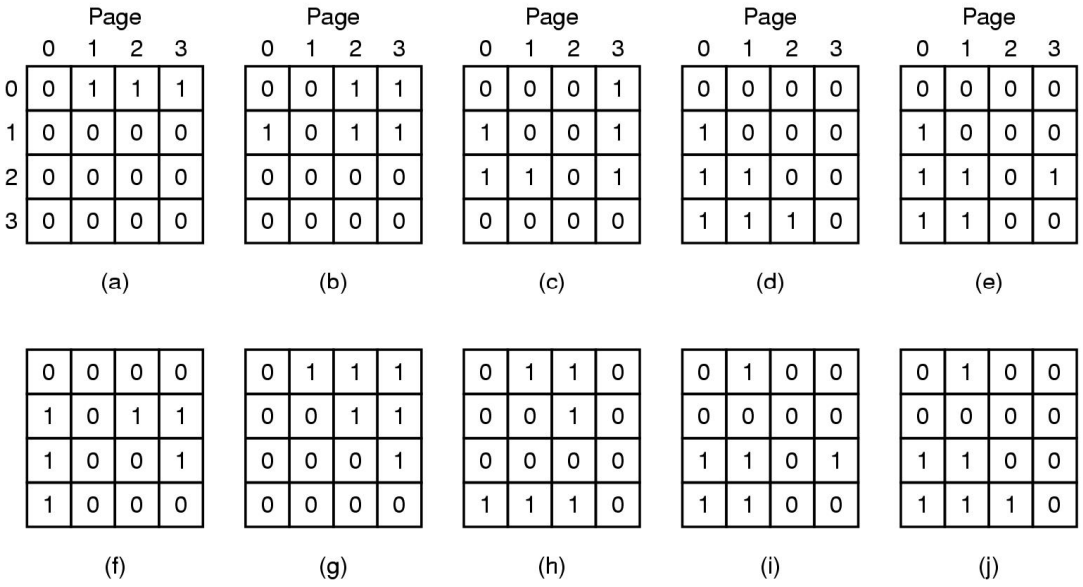
Second chance algoritam je modifikacija FIFO algoritma. Stranici koja po redoslijedu učitavanja treba biti obrisana provjerava se bit referenciranja. Ako je bit referenciranja 1 stranica se ne briše nego se postavlja na kraj liste uz promjenu bita referenciranja na 0. Ako bi sve stranice imale bit referenciranja 1 obrisala bi se prva provjeravana stranica jer bi nakon promijene bit referenciranja na 0 prva stranica sa bitom referenciranja 0 bila prva provjeravana stranica.

Satni mehanizam radi na isti način kao i second chance ali ne koristi listu već pokazivač na proces koji se pomiče po principu kazaljke na satu (slika 51.)



Slika 51. Satni mehanizam

Last Recently Used LRU Algoritam selektira stranicu koja se nije koristila najduže vremena. Kontrola korištenja vremena upotrebe stranice može biti ostvarena posebnim poljem u tabeli stranica koje pamti vrijednost brojača instrukcija u trenutku korištenja stranice. Kako se brojač povećava za svaku izvedenu instrukciju stranica koja ima najmanju vrijednost brojača u tabeli stranica odabire se za brisanje. Problem pri pohrani brojača u tabeli zapisa je dovoljno veliko polje za pohranu zapisa i kontrola pri poništavanu vrijednosti brojača (u jednom trenutku brojač počinje od 0). Zato se češće koristi primjena matrica veličine n x n (n je broj stranica). Na početku sve vrijednosti u matrici imaju vrijednost 0 a pri korištenju neke strance u redak matrice korištene stranice upisuju se 1, a za stupac stranice upisuju se 0. Kada se selektira stranica redak sa najmanjim brojem odgovara stranici koja nije korištena najdulje vremena. Slika 52. prikazuje rad algoritma za 4 stranice i redoslijed korištenja stranica: 0 1 2 3 2 1 0 3 2 3. Na kraju se briše stranica sa oznakom 1.

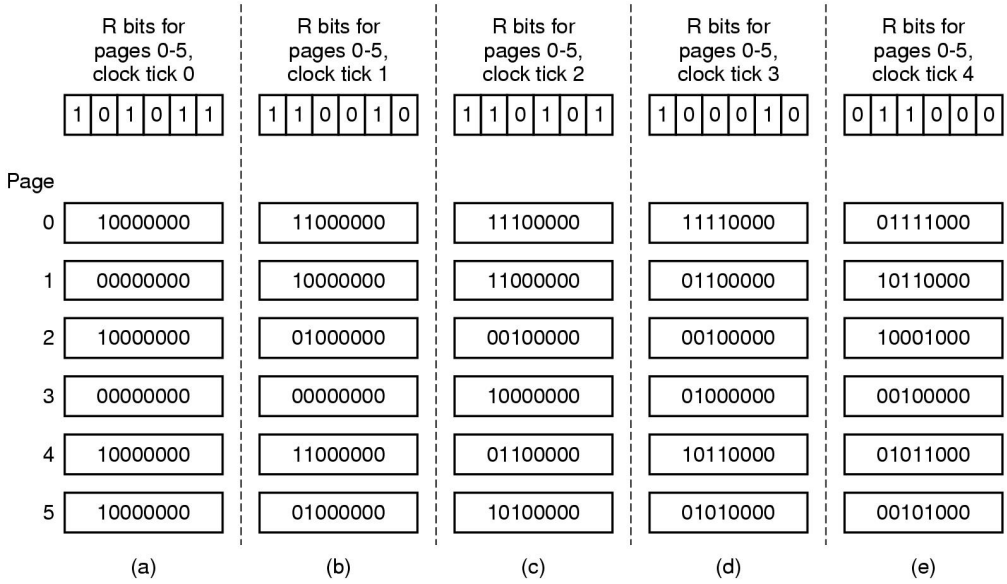


Slika 52. Rad LRU algoritma

Not Frequently used NFU algoritam. Implementacija LRU algoritma ovisna je o hardverskoj podržanosti implementacije rada matrica za praćenje korištenja stranica. Rješenje može biti korištenje softverske metode za određivanje najduže nekorištene

stranice, odnosno upotrebe NFU algoritma. U tabeli stranica dodaje se posebno polje za pohranu vrijednosti koja se dobiva na sljedeći način: nakon svakog satnog prekida dodaje se polju vrijednost bit referenciranja 0 ili 1, pa se pri brisanju odabire stranica sa najmanjom vrijednosti jer je ona u prosjeku najmanji broj puta referencirana. Ipak za slučaj intezivnog korištenja stranice u nekom ranijem vremenu biti će obrisana stranica koja se češće koristi u zadnjem intervalu vremena ali je manje referencirana od ranije korištene stranice. Zato se pribjegava korištenju algoritma *agging*.

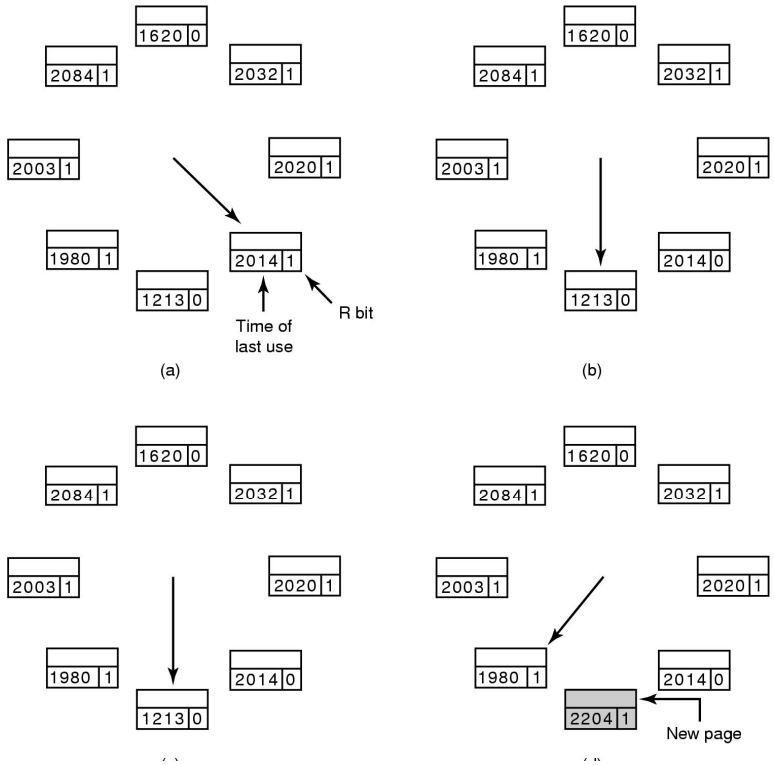
Agging algoritam. Svaka stranica ima polje koje se koristi za pohranu podatka o referenciranosti stranice. Nakon svakog satnog prekida provjerava se bit referenciranja ali se on ne zbraja sa vrijednošću polja, već se dodaje na početak niza bitova u polju, a postojeći bitovi se prije toga pomiču za jedno mjesto u desno. Pri brisanju selektira se stranica koja ima najmanju vrijednost polja. Slika 53. prezentira postupak agging-a.



Slika 53. Postupak agging-a

Radni skup stranica. Algoritam radi po principu zamjene stranice koja nije u radnom skupu stranica. Primjenom postupka određivanja radnog skupa stranica određuje se za svaki proces optimalan broj stranica, a zatim se odabire stranica za proces koji ima više stranica od radnog skupa stranica. Nedostatak algoritma je kontinuirano izračunavanje radnog skupa stranica što je vremenski zahtjevno.

WSClock algoritam radi na istovjetan način kao i satni algoritam ali prije brisanja stranice koja ima bit referenciranja 0 provjerava da nije stranica procesa u radnom skupu stranica, ako je stranica se zadržava u memoriji i selektira se stranica koja nije u radnom skupu stranica. Slika 54. prezentira rad WSClock algoritma.



Slika 54. Rad WSClock algoritma

Tablica 3. prikazuje usporedbu algoritama.

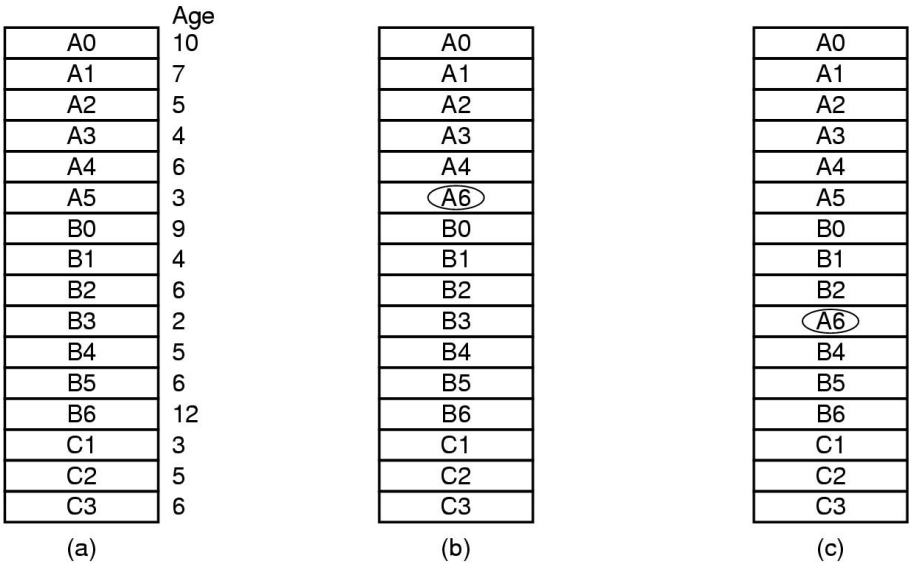
Tablica 3. Usporedba algoritama za zamjenu stranica

Algoritam	**Svojstvo**
Optimalni	Ne može se implementirati
Not Recently Used – NRU	Vrlo krut
FIFO	Mogućnost gubitka važnih stranica
Second Chance	Poboljšanje FIFO algoritma

Algoritam	Svojstvo
Satni	Realističan
Last Recently Used LRU	Odličan, ali težak za implementaciju
Not Frequently used NFU	Korektna aproksimacija LRU algoritma
Agging	Odličan algoritam koji izvrsno aproksimira LRU algoritam
Radni skup stranica algoritam	Skup za implementaciju
WSClock	Dobar i učinkovit algoritam

Lokalna i globalna zamjena stranica

Lokalnoj zamjena stranica pri odabiru stranice za zamjenu odabire jednu od stranica procesa koji je generirao zahtjev za učitavanjem nove stranice (*page fault*). Globalna zamjena stranica odabire stranicu bilo kojeg procesa, odnosno ne isključivo stranice procesa koji je generirao *page fault*. Slika 55. prikazuje lokalnu i globalnu zamjenu stranica.



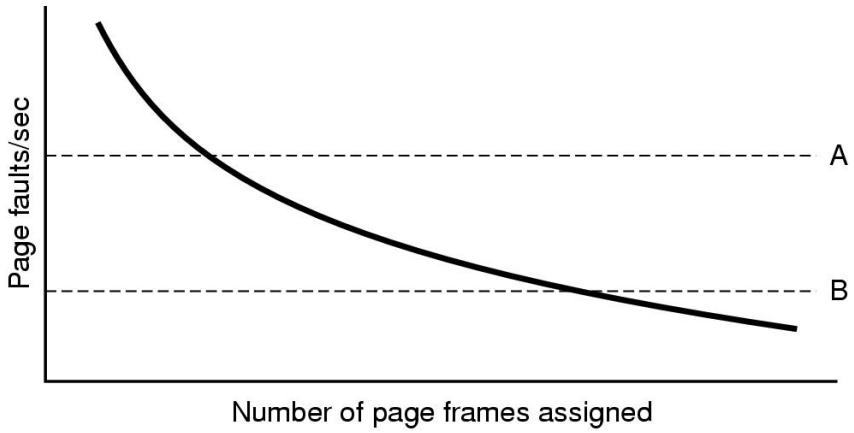
Slika 55. Lokalna i globalna zamjena stranica: a) početno stanje
b) lokalna zamjena stranica c) globalna zamjena stranica

Veličina stranica

Određivanje veličine stranica ovisno je o veličini tabele stranica i iskoristivosti memorijskog prostora. Ako je stranica malena broj zapisa u tabeli stranica je velik i pretraživanje je sporo, za velike stranice broj zapisa je malen ali je interna fragmentacija velika pa se odabir kompromisno rješenje za veličinu stranica od 1, 4 ili 8 KB.

Zagušenje (thrashing)

Povećanje iskoristivosti procesora može se postići povećanjem broja procesa u ready stanju što se postiže povećanjem ukupnog broja procesa u sustavu. Povećanje ukupnog broja procesa može biti ostvareno smanjenjem broja učitanih stranica procesa ili smanjenjem veličine stranice. Uglavnom se pribjegava smanjenju broja učitanih stranica u memoriju za svaki proces. Smanjenjem broja učitani stranica povećava se ukupni broj procesa u sustavu pa tako i broj procesa u ready stanju što povećava iskoristivost procesora. Nastavljanjem smanjenja učitanih stranica procesa u jednom trenutku dolazi do naglog opadanja iskoristivosti procesora. Razlog opadanja iskoristivosti procesora je smanjen broj učitanih stranica procesa u memoriji i povećan broj generiranih zahtjeva za učitavanjem novih stranica (*page fault*). Za vrijeme izvođenja *page faulta* proces je u blokiranom stanju, a zbog povećanog broja zahtjeva komunikacija između radne memorije i virtualne memorije (hard disk) je povećana. Broj stranica po procesu u trenutku maksimalne iskoristivosti procesora zove se radni skup stranica. Kontrola nastanka 'zasićenja' ostvaruje se praćenjem broja generiranih zahtjeva za učitavanjem novih stranica (slika 56). Ako je broj zahtjeva premalen procesu se oduzima nekoliko učitanih stranica, a ako je broj zahtjeva prevelik proces dobiva nekoliko stranica.

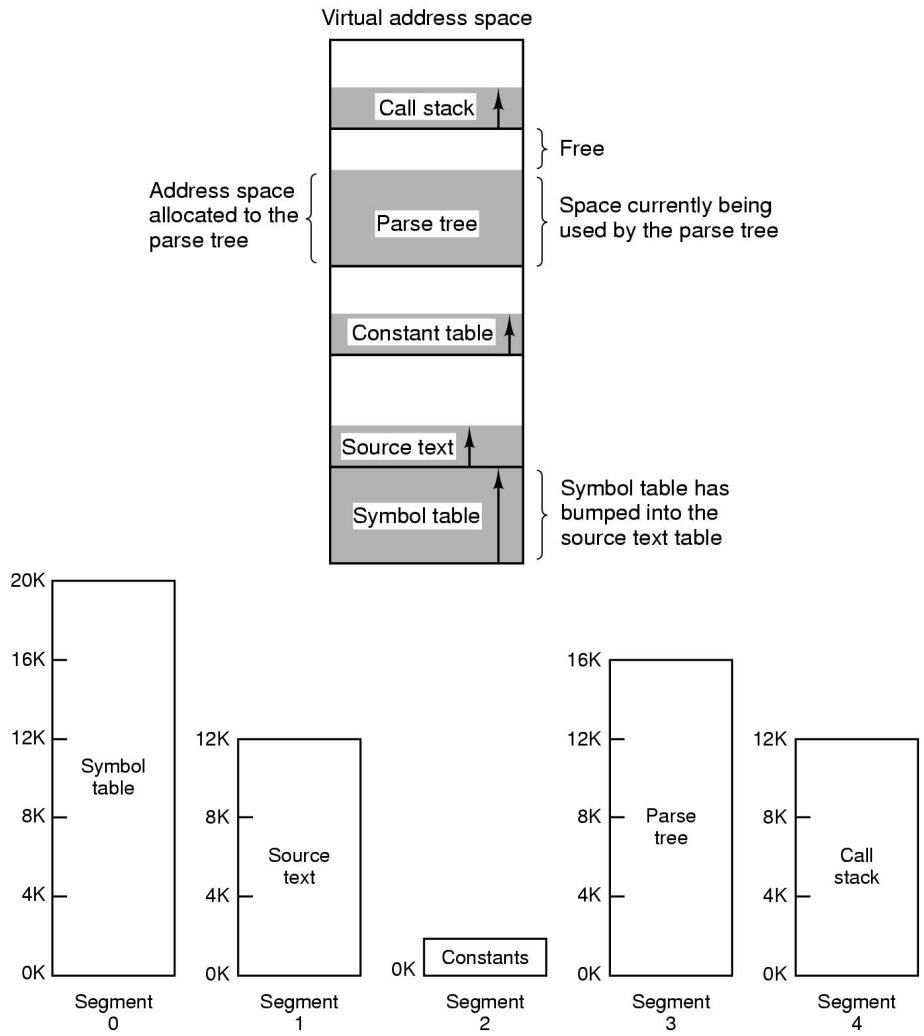


Slika 56. Kontrola broja zahtjeva za učitavanjem nove stranice

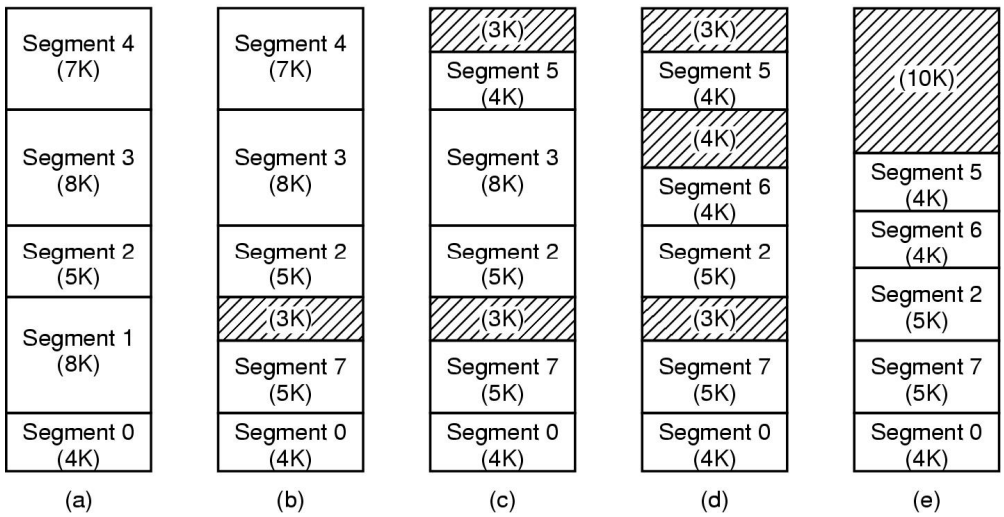
Segmentacija

Tehnika segmentacije zasniva se na logičkoj podjeli procesa na dijelove koji čine funkcionalne cjeline (zaglavlje programa, glavni modul, funkcije, procedure itd), učitavanju dijelova procesa koje zovemo segmenti u virtualnu memoriju i unosu određenog broja modula u radnu memoriju. Ukoliko segment potreban za daljnje izvođenje nije u radnoj memoriji potrebno ga je učitati u prazninu u radnoj memoriji, a ako praznina nije slobodna nužno je osloboditi dio radne memorije brisanjem jednog ili više segmenata. Slika 57. prikazuje podjelu procesa na segmente.

Za razliku od straničenja koje koristi jedan linearni adresni prostor segmentacija koristi linearni adresni prostor za svaki segment, odnosno svaki segment polazi od adrese 0. Svaki proces dobiva zasebnu tabelu segmenata sa zapisima o svim modulima procesa. Zapis tabele segmenata zvan deskriptor sadrži podatke o učitanosti segmenta u radnu memoriju, početnu adresu segmenta, veličinu segmenta, podatke o sigurnosti itd. Pri brisanju segmenata nastaje praznina koja se popunjava sa segmentom jednakim ili manjim od praznine. Najčešće je segment manji od praznine pa nastaju manje praznine koje se popunjavaju novim segmentima, a ako takvih segmenata nema nastaju fragmenti. Segmentacija stara eksternu fragmentaciju (slika 58.). Eksterna fragmentacija se može izbjeći memorijskim pomicanjem segmenata jedan do drugog, ali takav postupak okupira u potpunosti računalne resurse pa se uglavnom izbjegava.

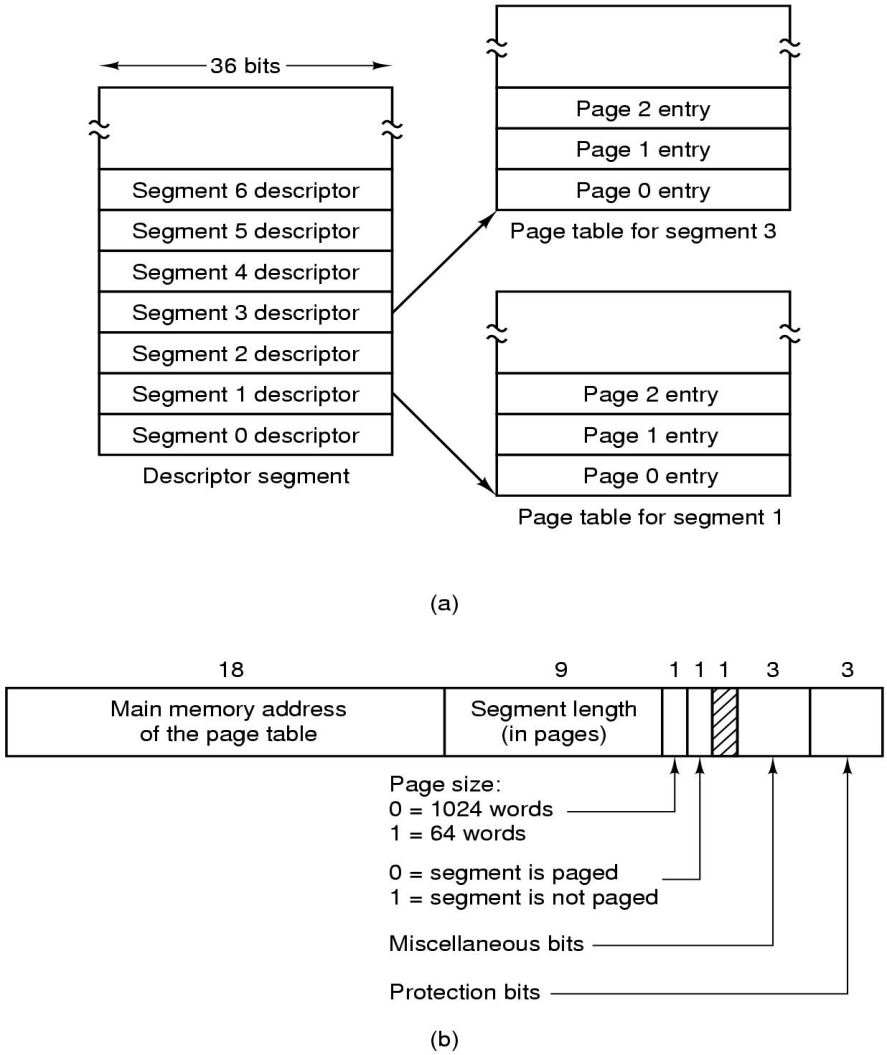


Slika 57. Podjela procesa na segmente



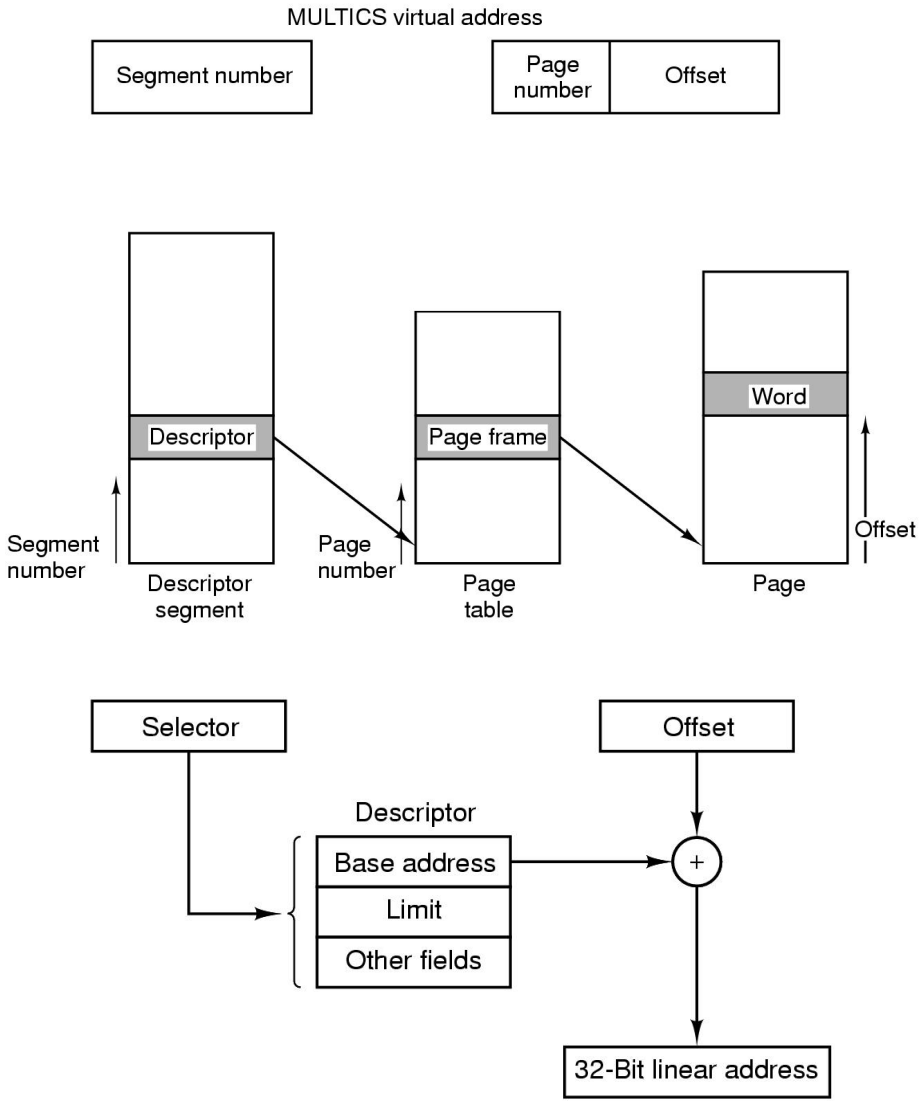
Slika 58. Pojava eksterne fragmentacije kod segmentacije

Smještaj segmenata u radnu memoriju omogućuje izvođenje relativno malog broja procesa. Za pokretanje većeg broja procesa nužno je korištenje segmentacije sa straničenjem. Svaki segment se dijeli na stranice, a isto se uradi i sa dijelom radne memorije rezerviranim za smještaj segmenta (sada je rezervirani dio u radnoj memoriji manji od segmenta). Svaki segment dobiva tabelu stranica te se u radnu memoriju učitava samo određeni broj stranica. Zapis u deskriptoru sada sadrži podatak o adresi tabele stranica, zapisu u tabeli stranica i offset-u unutar stranice. Primjenom segmentacije sa straničenjem omogućen je smještaj većeg broja procesa u radnu memoriju uz složeniju implementaciju. Slika 59. prezentira postupak izračunavanja adrese u radnoj memoriji.



Slika 59. Primjena segmentacije sa straničenjem

Iz deskriptora se saznaje početna adresa memorijskog prostora u radnoj memoriji korištenog za smještaj segmenta, adresa tabele stranica i oznaka zapisa u tabeli stranica. Nakon toga se pristupa u tabelu stranica kako bi se odredila stranica u radnoj memoriji u koju je učitana tražena stranica segment, a zatim se na početnu adresu segmenta u memoriji izračunava adresa početka stranice u radnoj memoriji te se na nju dodaje offset (slika 60.) .



Slika 60. Izračunavanje adrese pri straničenju

Dijeljenjem segmenata procesa moguće je dodijeliti posebna svojstva logičkim cjelinama procesa što straničenje nije omogućavalo. Segmentacija omogućuje zaštitu segmenata,

odnosno kontrolu pristupa segmentu pa je implementirana sigurnost. Segmentacija omogućuje i dijeljenje segmenata između više procesa pa jedan segment može biti korišten za više procesa (npr. kod kompajlera se učita jedanput i koristi za više procesa kompajliranja dokumenata). Generalno mogu se napraviti usporedbe između straničenja i segmentacije (tabela 4).

Tabela 4. Usporedba straničenja i segmentacije

Svojstvo	Straničenje	Segmentacija
Potreba upoznavanja programera o korištenoj tehnici	Ne	Da
Broj lineranih adresnih prostora	1	Više
Može li adresni prostor procesa premašiti veličinu radne memorije	Da	Da
Korištenje zaštite	Ne	Da
Jednostavnost povećanja tabela stranica/segmenata	Ne	Da
Mogućnost dijeljenja	Ne	Da
Namjena tehnike	Velik adresni prostor uz relativno malu veličinu radne memorije	Podjela procesa u logičke cjeline uz podržanu sigurnost i dijeljenje

5. UPRAVLJANJE ULAZNO-IZLAZNIM JEDINICAMA

Upravljanje ulazno-izlaznim jedinicama podrazumijeva usklađivanje različitosti između ulazno-izlaznih jedinica i omogućavanje izvođenja sistemskih poziva sa uređajima. Rad sa U-I jedinicama realizira se primjenom hardverskih i softverskih komponenti. Hardverske komponente ključne za rad U-I jedinica su komponente za implementaciju obradu prekida. U-I jedinice generiranjem prekida dojavljuju ostatku računalnog sustava o završetku određene aktivnosti. Druga važna komponenta je kontroler koji prenosi naredbe prema U-I jedinici, a također sadrži buffer za pohranu podataka koji se razmjenjuju sa U-I jedinicama. Korištenje buffer-a motivirano je oslobađanjem CPU-a pri stalnom prijenosu podataka sa U-I jedinice u glavnu memoriju i mogućnošću brze detekcije u slučaju pogrešnog prijenosa podataka. Softverska komponenta bitna za rad U-I jedinica je drajver koji upravlja radom U-I jedinice putem kontrolera.

5.1 HARDVERSKA PODRŠKA RADU U-I UREĐAJA

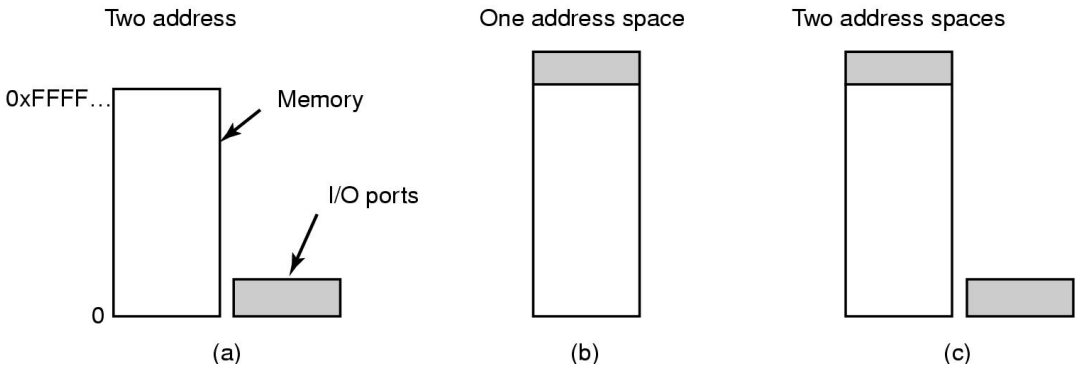
U-I jedinice dijelimo na blok i karakter uređaje. Blok uređaji podatke obrađuju u definiranom formatu i imaju mogućnost pristupa pohranjenim podacima (primjer uređaja je hard disk). Karakter uređaj obrađuje podatke nizom karaktera i ne mogu kasnije pristupati pohranjenim podacima. Slika 61. prikazuje razlike u brzini rada pojedinih U-I uređaja. Komunikacija s U-I jedinicama može se ostvariti primjenom porta ili memorijskim mapiranjem U-I jedinica (slika 62). Korištenjem U-I mapiranja izbjegavaju se sistemski pozivi

Ubrzanje komunikacije sa U-I uređajima ostvaruje se korištenjem brzih sabirnica namijenjenih komunikaciji sa U-I uređajima (slika 63). Slika 64. prikazuje sabirnice korištene za komunikaciju sa U-I jedinicama. Daljnje poboljšanje postiže se primjenom DMA prijenosa. Klasičnim postupkom CPU je kontinuirano aktivan pri prijenosu sadržaja buffer-a kontrolera u radnu memoriju. Pri DMA prijenosu CPU kontroleru proslijedi

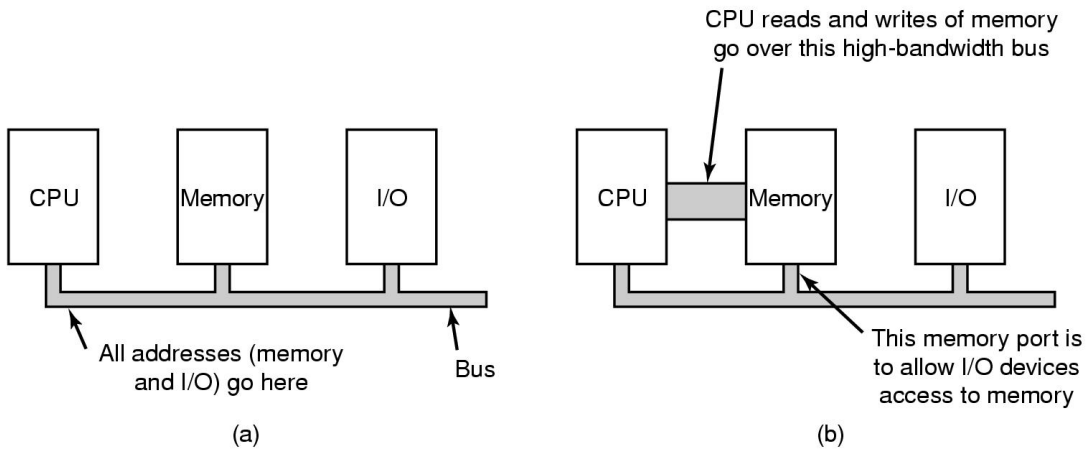
podatke o početnoj adresi u radnoj memoriji za smještaj podatka iz buffer-a te broj bajtova za prijenos. Nakon tog CPU može izvoditi druge poslove dok će kontroler obaviti prijenos podatka iz buffer-a u radnu memoriju. Proslijeđeni podaci pohranjuju se u registrima kontrolera, te se za prijenos svakog bajta podataka povećava adresa u memoriji i smanjuje broj bajtova za prijenos za 1. Postupak se ponavlja sve dok se brojač bajtova za prijenos ne smanji na 0 (slika 65.).

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet	12.5 MB/sec
ISA bus	16.7 MB/sec
EIDE (ATA-2) disk	16.7 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane	20 GB/sec

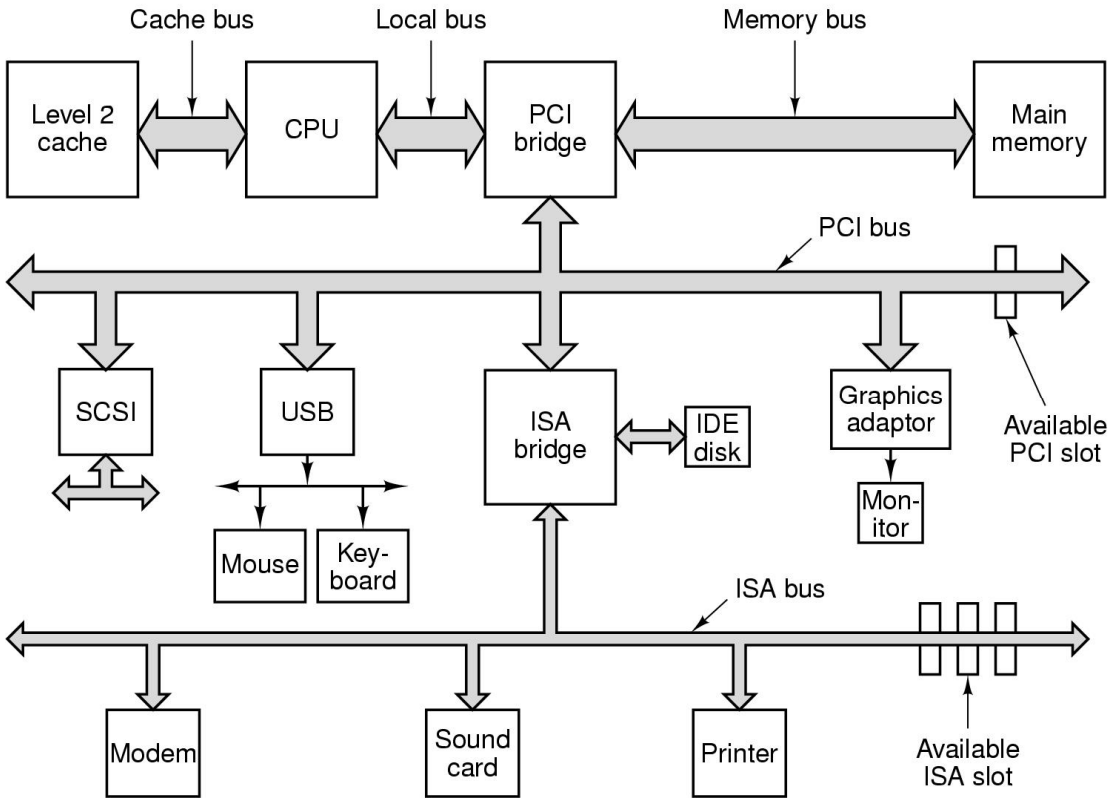
Slika 61. Usporedba brzina rada pojedinih uređaja



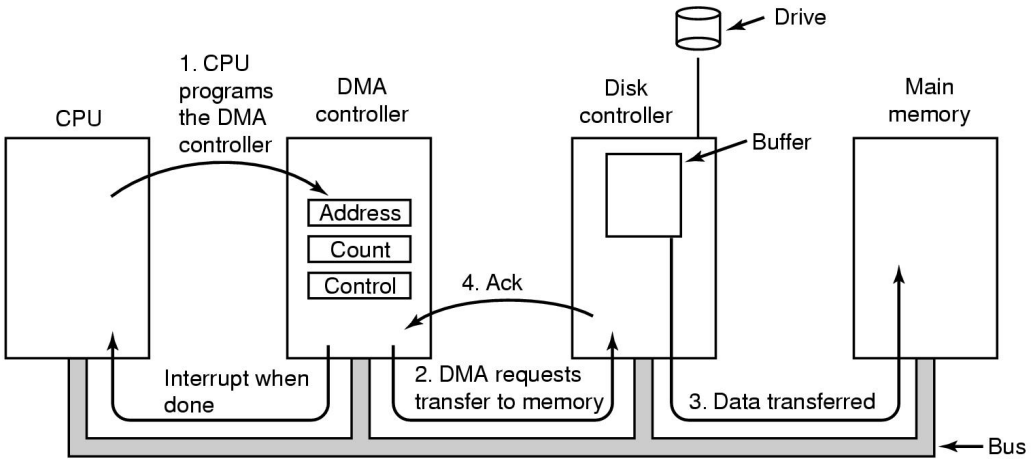
Slika 62. Primjena memorijskog mapiranja u radu U-I jedinica



Slika 63. Prikaz sustava sa jednom sabirnicom a) i više sabirnica b)

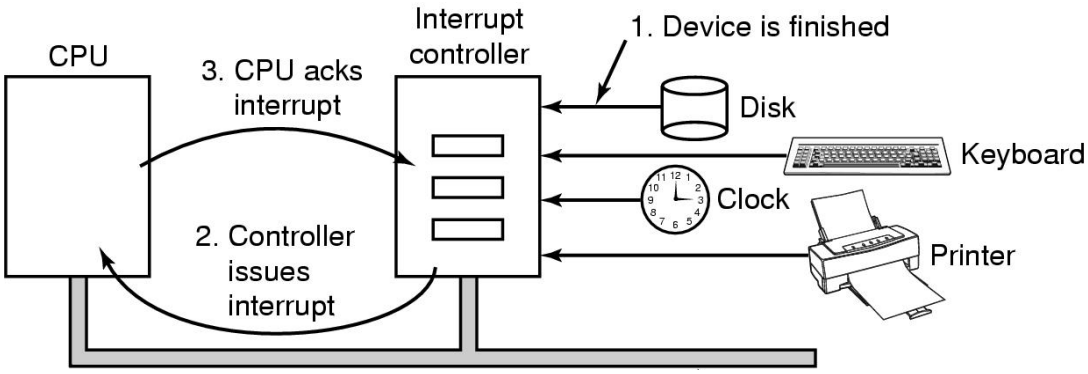


Slika 64. Prikaz sabirnica računalnog sustava



Slika 64. DMA prijenos podataka

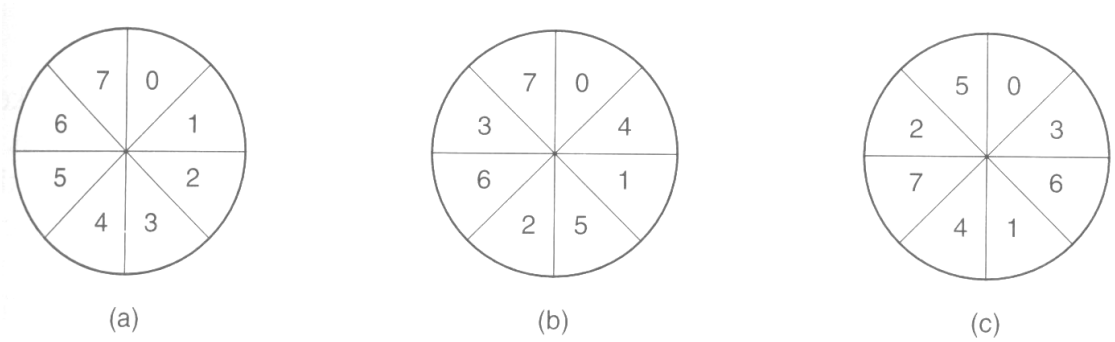
Slika 65. prikazuje implementaciju obrade prekida. Čip za obradu prekida prihvaća prekid od U-I jedinice, prepoznaje uređaj koji je poslao prekid i šalje signal CPU-u o pojavi prekida. CPU prekida izvođenje tekućeg procesa, pohranjuje podatke o stanju izvođenja procesa u tabelu stranica te prosljeđuje čipu za obradu prekida signal o spremnosti za pokretanje rutine za obradu prekida. Adrese rutina za obradu prekida na U-I jedinicama mogu biti pohranjene u čipu za obradu prekida, no češći je slučaj upotrebe početnih adresa u memoriji koje zovemo vektorima prekida. Čip na osnovi podataka iz vektora prekida pokreće rutinu za obradu prekida na CPU, nakon čega se nastavlja izvođenje nekog drugog procesa koji ne mora biti prekinuti proces u trenutku pojave prekida.



Slika 65. Obrada prekida

Bitno je napomenuti korištenje *interleaving*-a pri upotrebi blok uređaja. Ukoliko su logički blokovi razmješteni jedan do drugog pri čitanju dva susjedna bloka nužno je izvoditi rotaciju diska za jedan krug. Razlog tome je inercija ploče diska koji se ne može zaustaviti

u trenutku kada magnetna glava pročita blok već se magnetna glava pomakne preko početka susjednog bloka pa je nužna rotacija za cijeli krug kako bi se magnetna glava postavila na početak susjednog bloka. Postupkom *interleaving*-a susjedni logički blokovi nisu fizički spojeni jedan do drugog nego je između njih umetnut drugi blok pa se izbjegava potreba rotacije diska za puni krug pri čitanju dva susjedna bloka (inercija diska uzrokovat će pomak glave za čitanje iznad susjednog fizičkog bloka koji nije susjedni logički blok, pa je dovoljan mali pomak diska za postavljanje glave za čitanje iznad sljedećeg logičkog bloka). Ukoliko se između dva logička bloka umeću dva fizička bloka govorimo o dvostrukom *interleaving*-u. Slika 66. prikazuje primjenu *interleaving*-a.



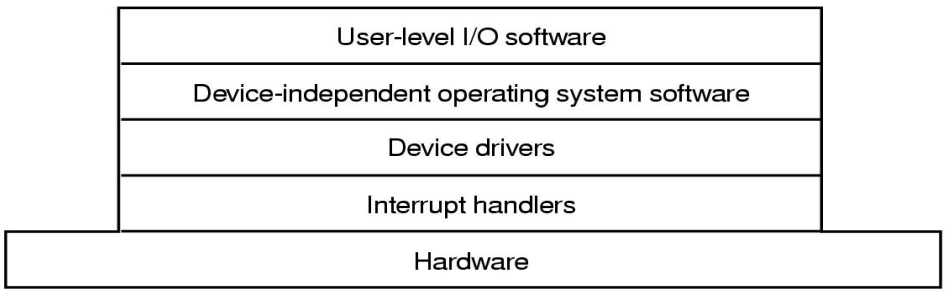
Slika 66. Primjena interleaving-a; a) bez interleaving-a
b) jednostruki interleaving c) dvostruki interleaving

5.2. SOFVERSKA PODRŠKA RADU U-I UREĐAJA

Softverska podrška radu U-I uređaja uključuje:

- Obrada prekida
- Drajver U-I uređaja
- Sloj neovisan o vrsti U-I uređaja
- Korisnički sloj.

Slika 67. prikazuje slojeve softvera za podršku U-I uređaja.



Slika 67. Slojevi softverske podrške radu U-I uređaja

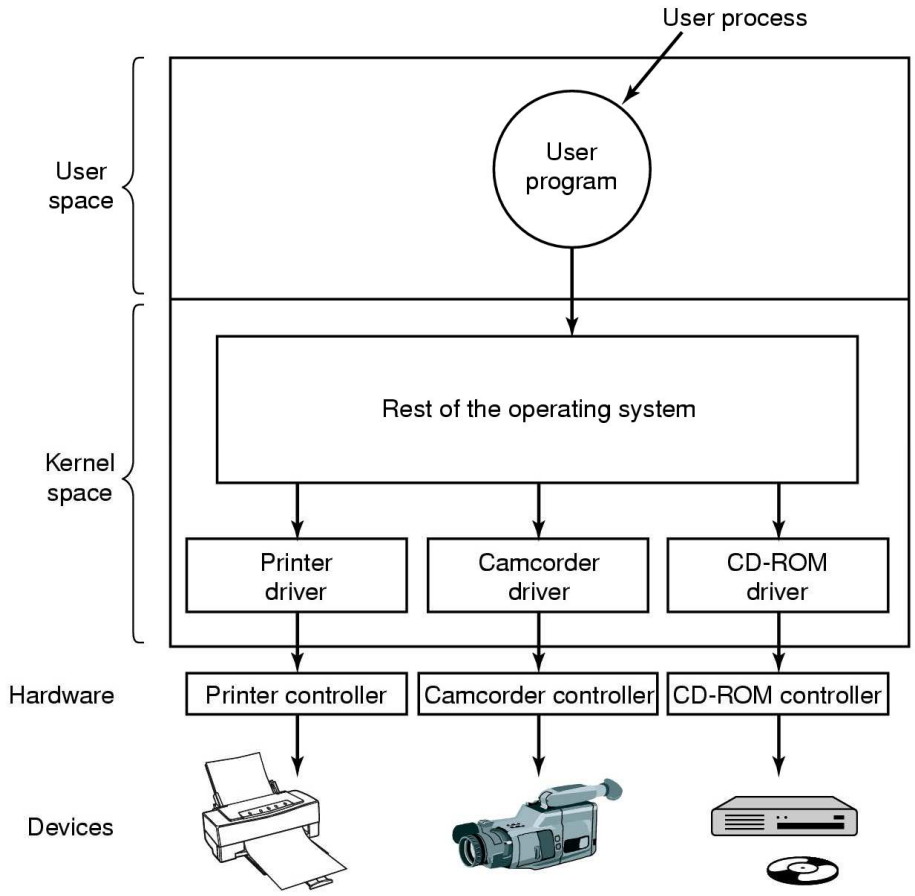
Obrada prekida je važna jer putem prekida U-I uređaj dojavljuje o završetku operacije koju izvodi. Zato se obrada prekida ugrađuje u kernel kao sastavni dio sloja kernela najbližeg hardveru računala.

Drajveri U-I uređaja koriste se za upravljanje radom U-I uređaja. Svaki U-I uređaj ima softverski program koji upravlja njegovim radom. Drajver poznaje strukturu U-I uređaja i putem kontrolera upravlja uređajem. Slika 68. prikazuje ulogu drajvera u upravljanju U-I uređaja.

Sloj neovisan o vrsti U-I uređaja ne razlikuje U-I uređaje već prema njima koristi jednako sučelje, a upravljanje njima izvodi primjenom drajvera. Njegove funkcije su:

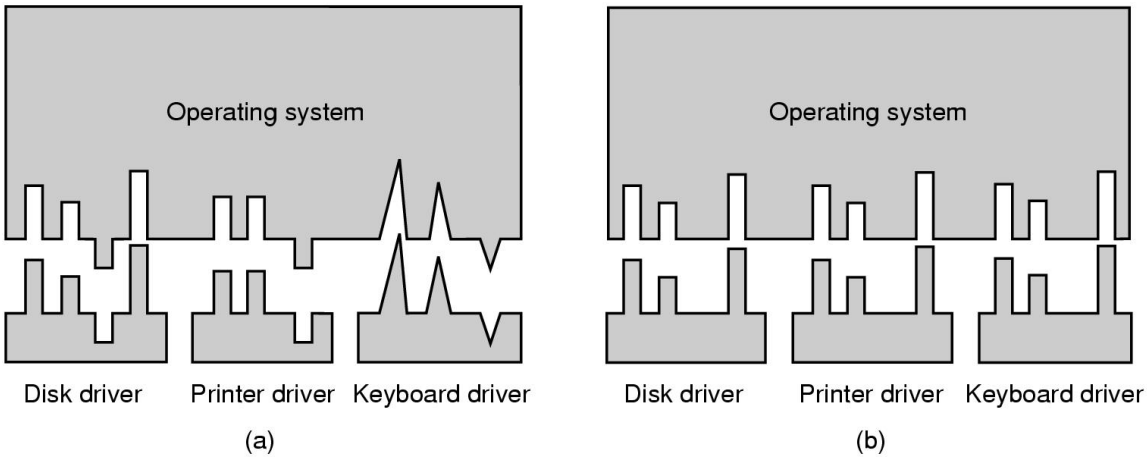
- određivanje imena U-I uređaja
- *buffering* – smanjenje potrebe za prijenosom korištenih podataka sa U-I uređaja
- dojava grešaka – u slučaju da kontroler ne uspije nakon nekoliko ponovljenih zahtijeva otkloniti grešku u prijenosu potrebno je korisnika informirati o nastalom problemu
- alokacija i otpuštanje U-I uređaja
- usklađivanje veličine bloka neovisno o U-I uređaju – U-I uređaji mogu imati različitu veličinu bloka pa je potrebno uskladiti razlike između uređaja.

Korisnički sloj podrazumjeva formatiranje poruka korisnika poslanih od sloja neovisnog o vrsti U-I uređaja te uzvođenje procesa koji prate rad pojedinih U-I uređaja (deamon procesi).



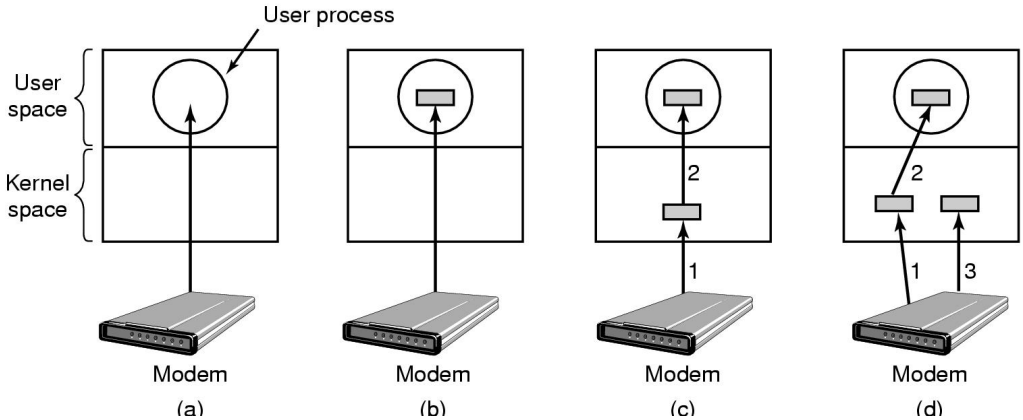
Slika 68. Primjena drajvera

Slika 69. prikazuje razliku između različitog i jedinstvenog sučelja prema U-I uređajima.



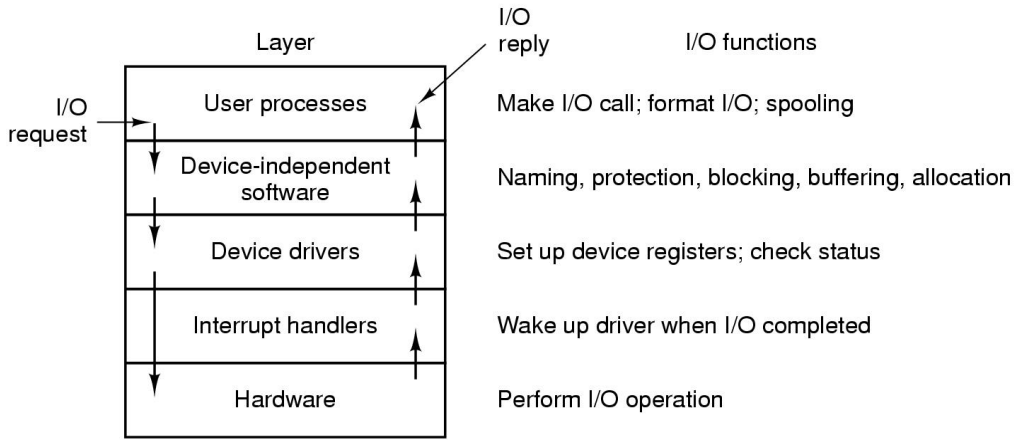
Slika 69. Primjena različitih sučelja: a) različito sučelje b) jedinstveno sučelje

Slika 70. prikazuje primjenu buffering-a. Pod a) prikazan je slučaj bez upotrebe buffer-a, pod b) koristi se buffer u korisničkom prostoru što uzrokuju problem ukoliko u trenutku brisanja buffer-a stigne mrežnom skup podataka za pohranu u buffer-u, pod c) se koristi buffer u korisničkom i kernel prostoru što rješava problem u slučaju b), ali ostaje problem prihvata podataka mrežnom komunikacijom ako iz buffer-a u kernelu nisu podaci prenešeni u buffer u korisničkom prostoru, a pod d) se koristi buffer u korisničkom prostoru i dva buffer-a u kernel prostoru što razrješava sve prethodno objašnjane situacije.



Slika 70 Primjena buffering-a

Slika 71. prikazuje slijed aktivnosti na izvođenju zahtijeva korisničkog procesa prema U-I uređaju. Generira se sistemski poziv prema sloju softvera neovisnom o vrsti uređaja. Sloj određuje vrstu U-I uređaja za izvođenje zahtijeva te prosljeđuje zahtjev drajveru. Drajver putem kontrolera šalje naredbu U-I uređaju. U-I uređaj po završetku obrade generira prekid koji se obradi te se o tome informira drajver. Drajver informira sloj neovisan o vrsti U-I uređaja i na kraju korisnički proces dobiva rezultat na poslani zahtjev.



Slika 71. Izvođenje sistemskog poziva za rad sa U-I uređajima

6. UPRAVLJANJE DATOTEČNIM SUSTAVOM

Malen kapacitet radne memorije i nemogućnost trajnog pamćenja podataka uvjetuje korištenje medija za trajno pamćenje podataka odnosno vanjske memorije. Dio operativnog sustava namijenjen izvođenju operacija manipulacije podacima na vanjskoj memoriji zove se upravljač datotečnog sustava. Njegove osnovne funkcije su: alokacija slobodnog i zauzetog prostora vanjske memorije, smještaj datoteka na vanjsku memoriju, sigurnost i opravak u slučaju nastanka grešaka, pohrana velike količine podataka, očuvanje podataka u slučaju terminiranja procesa i pristup podacima za više procesa istovremeno.
Ključni pojmovi datotečnog sustava su:

- datoteka
- direktorij
- alokacija vanjske memorije

6.1. DATOTEKA

Za rad sa datotekama bitno je odrediti:

- Pravila dodjele imena datoteka
- Atribute
- Strukturu
- Vrste
- Pristup
- Operacije.

Imena datoteka razlikuju se u maksimalno dozvoljenom broju znakova, dozvoljenom skupu znakova za imena datoteka, razlikovanju malih i velikih slova itd. Imena datoteka se sastoje od dva dijela: naziva koji određuje korisnik i ekstenzije dodane aplikativnim programom za obradu datoteke. Slika 72. prikazuje primjere nekih ekstenzija.
Atributi određuju svojstva datoteka. Po stvaranju datoteke određeni su atributi prema početno definiranim (default) vrijednostima, koje se mogu pregledavati i mijenjati. Slika 73. prikazuje neke od atributa datoteka.

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

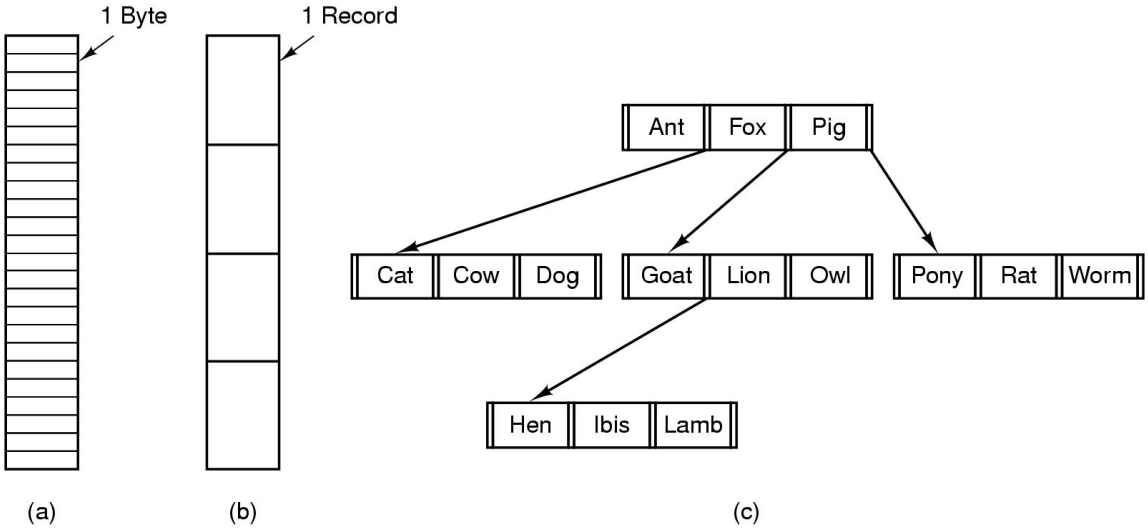
Slika 72. Imena datoteka

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Slika 73. Atributi datoteka

Struktura: datoteka može imati strukturu bajta, odnosno niza nula i jedinica, zatim strukturu zapisa (record) i strukturu stabla. Struktura zapisa bila je dominantna u vrijeme

korištenja tekstualnog moda u radu sa pisačima pa je zapis sadržavao broj znakova adekvatan broju znakova u ispisu jednog retka teksta na pisaču (80 ili 132 znaka). Struktura stabla omogućuje brzo pretraživanje ali ima kompleksnu strukturu pa se ne koristi često. Danas je struktura bajta dominantna. Slika 74. prikazuje strukture datoteka.



Slika 74. Struktura datoteka: a) bajt b) rekord c) stablo

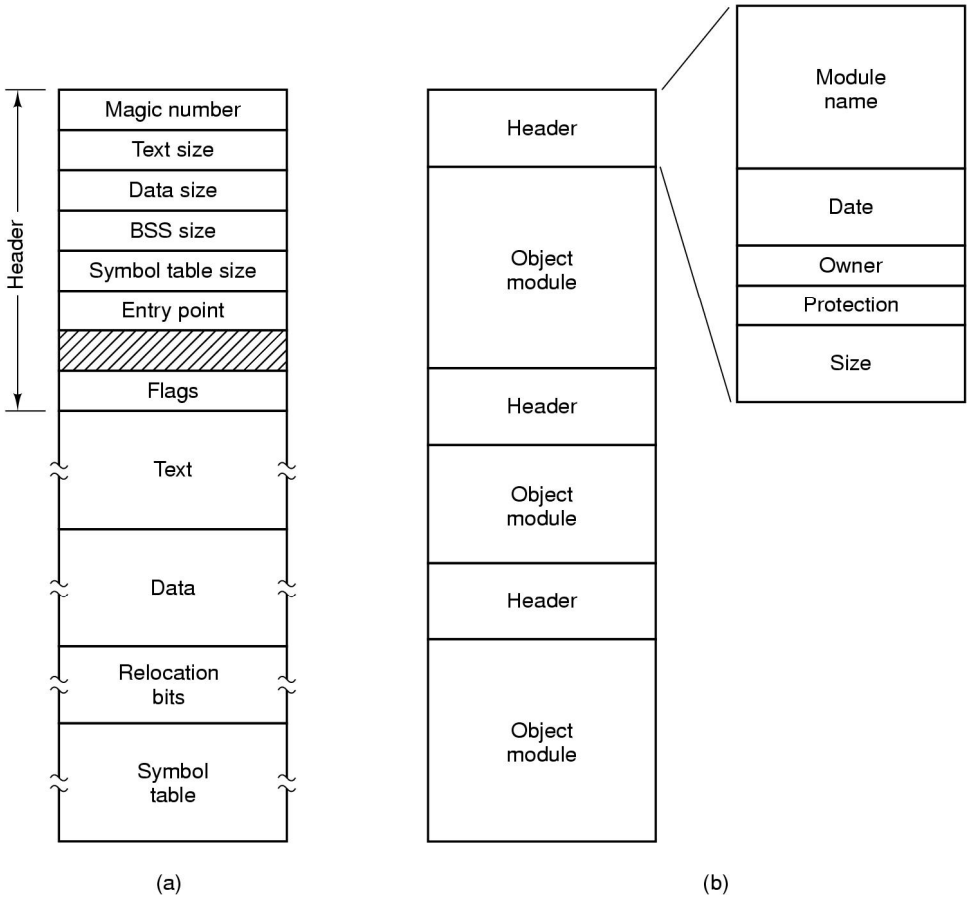
Struktura bajta znači niz nula i jedinica, ali u organizacijskom pogledu datoteka ima zaglavlje koje sadrži podatke o adresiranju ostatka datoteke. Slika 75. prikazuje strukturu izvršne datoteke i arhive.

Vrsta: datoteke mogu biti:

- regularne ili standardne za obradu podatka korisnika
- direktorij za pohranu podataka o logičkoj organizaciji datotečnog sustava
- blok specijalne za rad blok U-I uređaja
- karakter specijalne za rad karakter U-I uređaja.

Pristup datotekama može biti:

- sekvencijalan – za pristup podacima nužno je pristupiti svim podacima prije traženog
- random – za pristup podacima nije potrebno pristupiti svim podacima prije traženog jer je kretanje datotekom moguće u oba smjera (prema kraju i prema početku).



Slika 75. Struktura izvršne datoteke i arhive

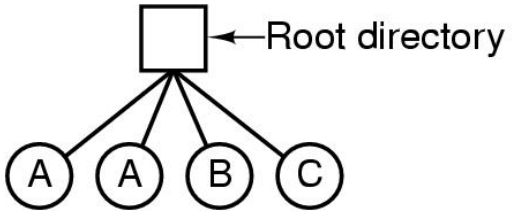
Operacije na datotekama su:

- Stvaranje: uvođenje datoteke u sustav, dodjela memorijskog prostora i default vrijednosti atributa
- Brisanje: uklanjanje datoteke iz sustava i oslobađanje dodijeljenog adresnog prostora
- Otvaranje: priprema datoteke za rad, čitanje njenih atributa i prikupljanje podataka o blokovima datoteke na vanjskoj memoriji
- Zatvaranje: brisanje datoteke iz liste otvorenih datoteka
- Čitanje: postavljanje pointera na određeno mjesto u datoteci i čitanje niza bajtova
- Pisanje: upis niza bajtova podataka od određenog mjesta u datoteci, ako je preostali dio datoteke premalen za upis podataka, datoteka se uvećava kako bi se pohranili svi podaci

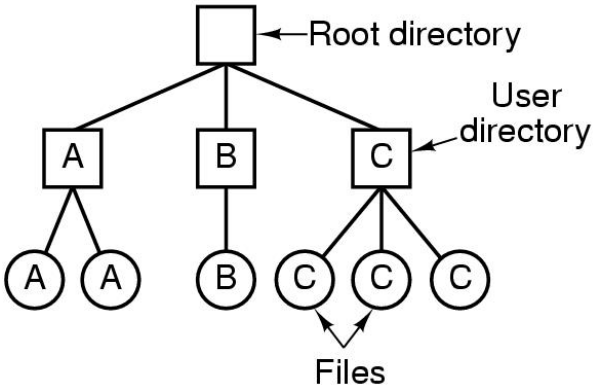
- Dodavanje: podaci se dodaju na kraj datoteke povećavajući datoteku
- Pretraživanje: postavljanje pokazivača (pointera) na skup znakova definiran prije pokretanja pretraživanja
- Čitanje atributa: dobivanje podataka o trenutnim vrijednostima atributa
- Postavljanje atributa: izmjena postojećih vrijednosti atributa
- Preimenovanje: promjena imena datoteka.

6.2. DIREKTORIJ

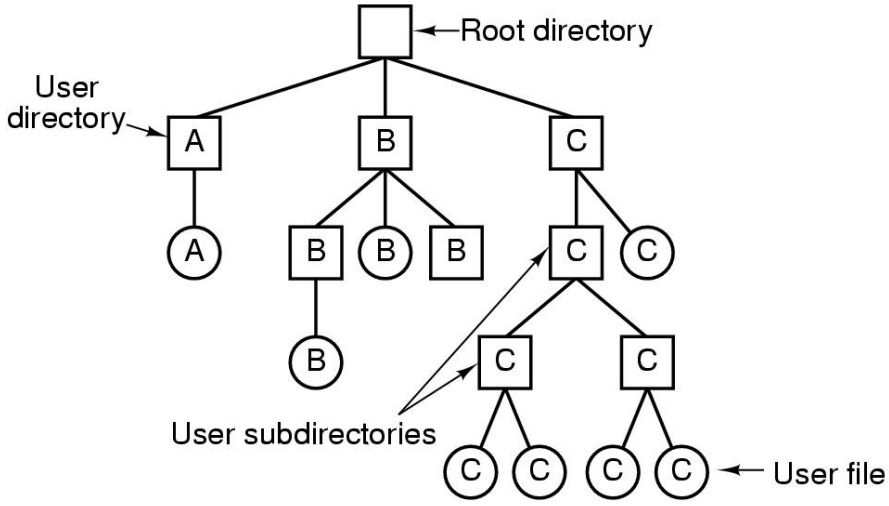
Direktorij (mapa, folder, kazalo) je posebna datoteka namijenjena pohrani podataka o strukturi datotečnog sustava. Od drugih datoteka razlikuje se po vrijednostima "**.**" i "**..**" koje upućuju na tekući direktoriji i prvi viši direktorij u odnosu na tekući. Svaki datotečni sustav ima glavni (root) direktorij. Na početku je korišten jedan direktorij (slika 76.) u kome nije bilo moguće ponavljati imena datoteka, zatim je korišten jedan direktorij za svakog korisnika (slika 77.) sa ograničenjem nemogućnosti ponavljanja imena datoteka za jednog korisnika. Danas je dominantna struktura stabla (slika 78.) sa ograničenjem nemogućnosti ponavljanja imena datoteke u jednom direktoriju.



Slika 76. Sustav sa jednim direktorijem

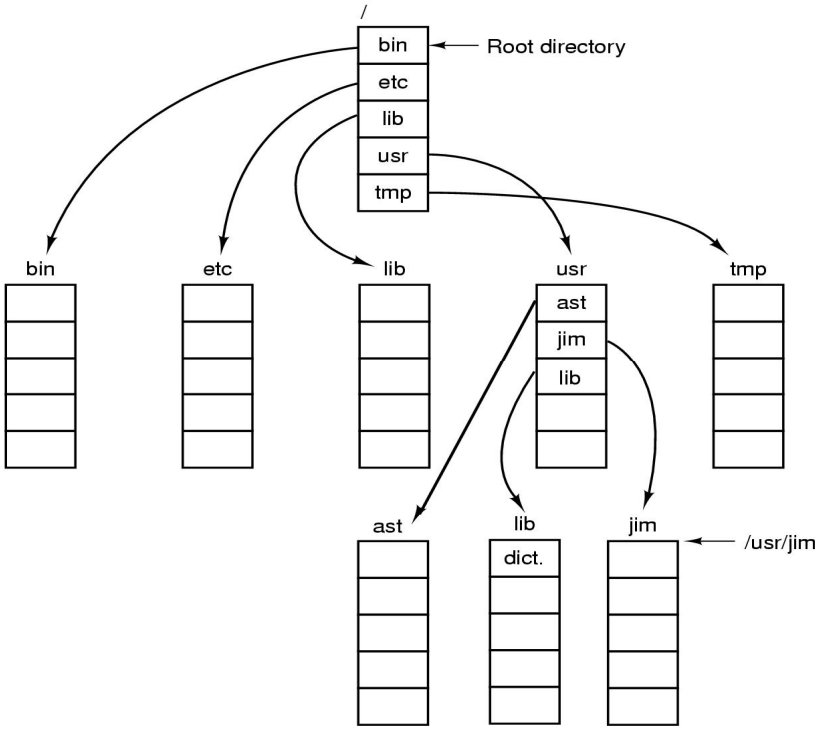


Slika 77. Sustav sa jednim direktorijem za svakog korisnika



Slika 78. Sustav sa stablom direktorijem

Svaki direktorij može sadržavati proizvoljan broj podmapa. Premještanje ili kopiranje datoteka između direktorija znači upis imena datoteka u odredišni direktorij, te brisanje imena datoteka iz polaznog direktorija (ako se izvodi premještanje). Pri brisanju datoteke koristi se sigurno brisanje premještanjem imena datoteka u specijalnu datoteku (Recycle bin) kako bi se mogla ponovo koristiti. Slika 79. prikazuje raspored direktorija za UNIX operativni sustav.



Slika 79. Raspored direktorija za UNIX operativni sustav

Na direktoriju mogu se izvoditi sljedeće operacije:

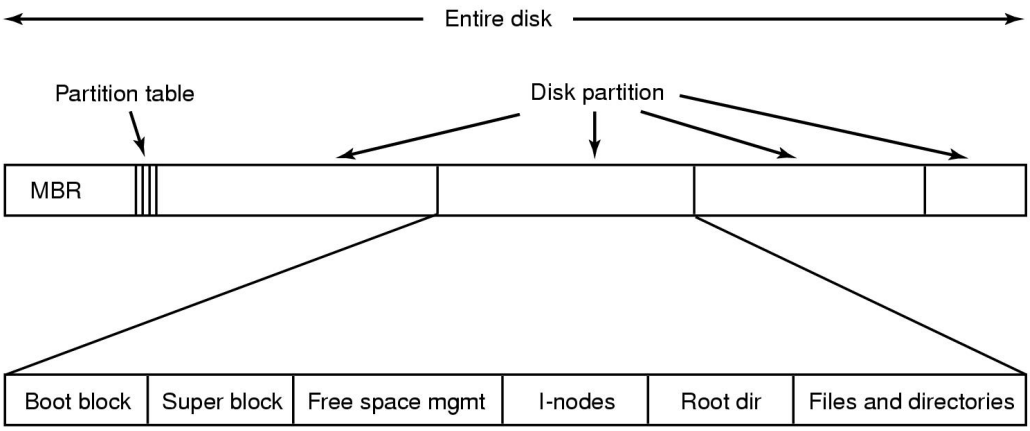
- Stvaranje
- Brisanje
- Otvaranje
- Zatvaranje
- Čitanje
- Promjena imena
- Povezivanje datoteka: postupak kojim dva ili više korisnika dijele adresni prostor jedne datoteke što se postiže usmjeravanjem zapisa istih ili različitih imena datoteka u direktorijima korisnika na blokove jedne datoteke
- Prekid povezivanja datoteke: postupak prekida dijeljenja jedne datoteke za više korisnika brisanjem zapisa o djeljivoj datoteci iz direktorija jednog od korisnika.

6.3. ALOKACIJA VANJSKE MEMORIJE

Alokacijom vanjske memorije povezuje se blokovi datoteke sa datotekom. Razlikujmo:

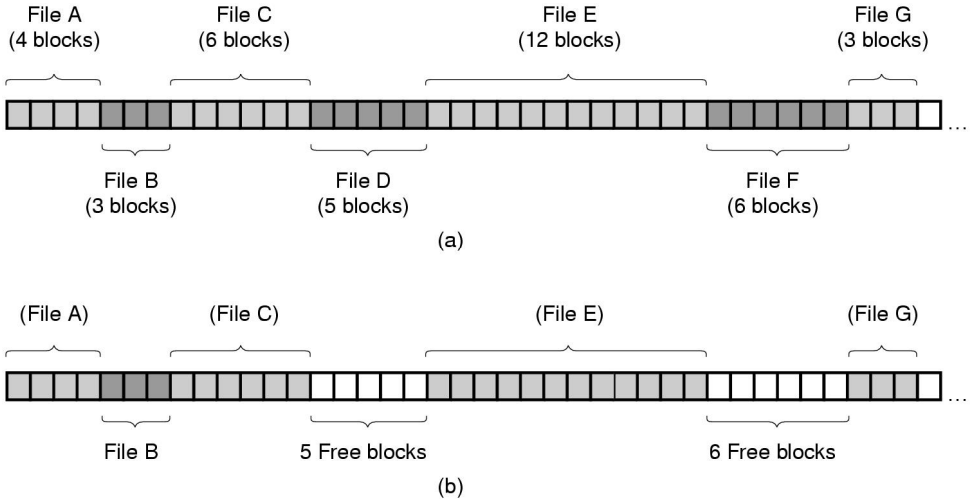
- Kontinuiranu alokaciju
- Alokaciju upotrebom povezanih listi
- Alokacija upotrebom povezanih listi uz korištenje indeksa
- Alokaciju upotrebom i-noda.

Slika 80. prikazuje moguću strukturu sustava vanjske memorije. Prezentiran je glavni zapis disks (master boot record - MBT) sa informacijama o particijama vanjske memorije, particije, startni zapisi particije (boot record), super blok sa podacima o strukturi datotečnog sustava particije, rezerviran prostor za ažuriranje slobodnog prostora particije, prostor za alokaciju vanjske memorije (u ovom slučaju i-node), root direktorij te dio za pohranu podataka.



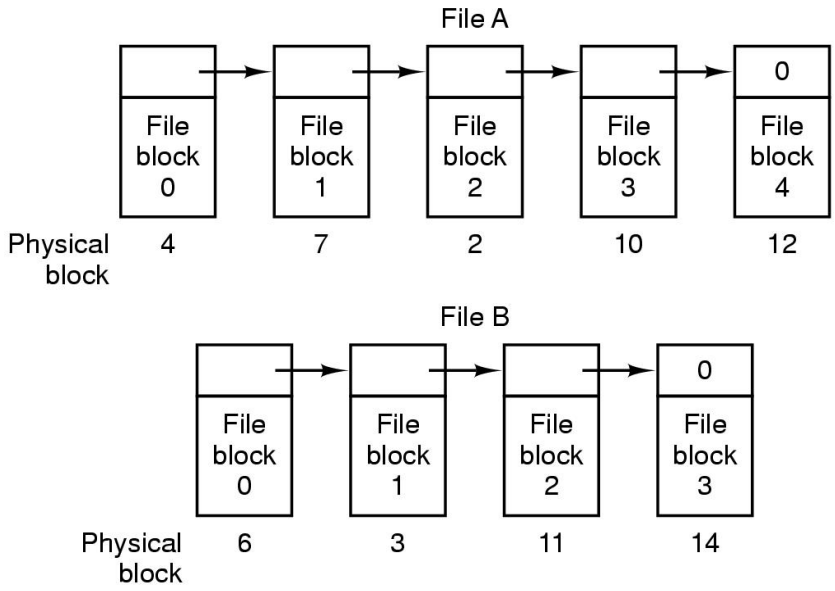
Slika 80. Moguća struktura vanjske memorije

Kontinuirana alokacija znači pohranu datoteke rezerviranjem kontinuiranog niza blokova za pohranu datoteke. Takav način pohrane datoteke omogućuje brzo pretraživanje i učitavanje datoteka (smanjen broj pomaka magnetnih glava za čitanje), ali pri brisanju datoteke dolazi do pojave eksterne fragmentacije na vanjskoj memoriji (slika 81.).



Slika 81. Kontinuirana alokacija

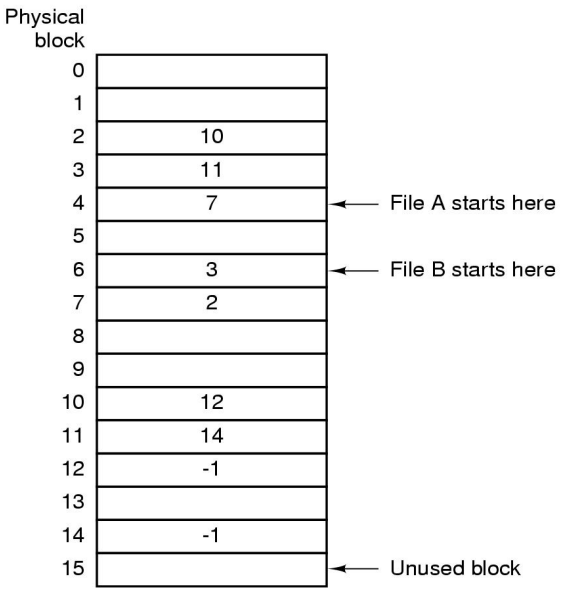
Povezane liste raspoređuju blokove datoteke ovisno o rasporedu slobodnih blokova na vanjskoj memoriji pri čemu svaki blok sadrži podataka o sljedećem bloku. Eksterna fragmentacija je izbjegnuta, ali je za pristup n-tom bloku potrebno pristupiti svim blokovima prije n-tog što uvjetuje veliku aktivnost magnetne glave za čitanje i sporost u radu (slika 82).



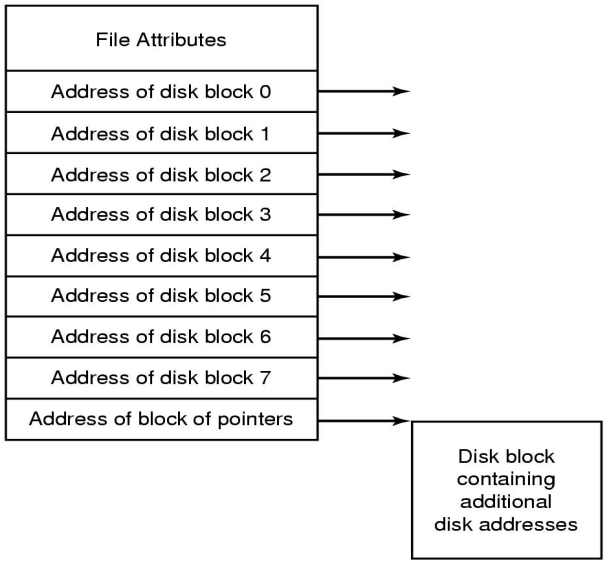
Slika 82. Povezane liste

Povezane liste uz upotrebu indeksa koriste isti princip kao i povezane liste, ali se podaci o sljedećim blokovima nalaze u posebnoj datoteci (file alocation table - FAT) koja ima onoliko zapisa koliko je blokova na vanjskoj memoriji. Za pristup n-tom bloku potrebno je u tabeli pristupiti zapisima svih blokova prije n-tog. Povezane liste uz upotrebu indeksa isto kao i kontinuirana alokacija i povezane liste blokovima pristupaju sekvencijalno. Za pristup n-tom bloku potrebno je imati podatak o početnom bloku. Slika 83. prikazuje FAT tabelu.
I-node je rješenje koje za pohranu podataka o blokovima datoteka koriste posebne tablice (i-node) koje pohranjuju podatke o određenom broju blokova datoteke (obično 64). Ako to nije dovoljno (datoteka ima više od 64 bloka) posljednji zapis upućuje na zaseban blok koji se koristi za pohranu podataka o ostalim blokovima datoteke. Pristup podacima je direktan pa nije potrebno pristupati podacima svih prethodnih blokova već se odmah može pristupiti traženom bloku. Slika 84. prikazuje upotrebu i-noda.

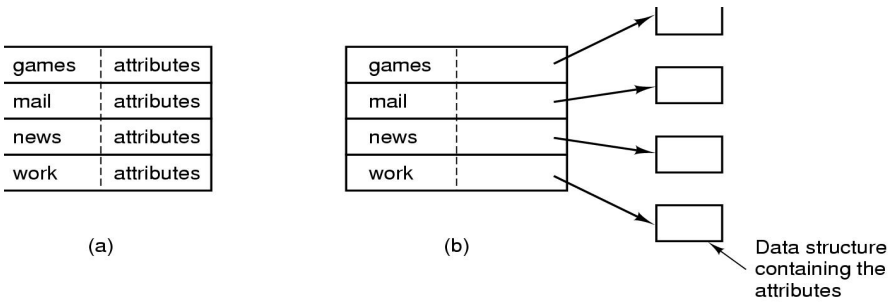
Implementacija direktorija ovisi o vrsti alokacije vanjske memorije. Zapis u direktoriju sadrži podatke o datoteci (naziv, atribute, veličina vremena nastanka i posljednje modifikacije itd.) te podatke za pristup blokovima datoteke. Atributi datoteke mogu biti smješteni u zapisu direktorija ili izvan njega. Ako su atributi pohranjeni u direktoriju olakšana je manipulacija atributima, ali je broj atributa ograničen (slika 85.).



Slika 83. FAT tabela

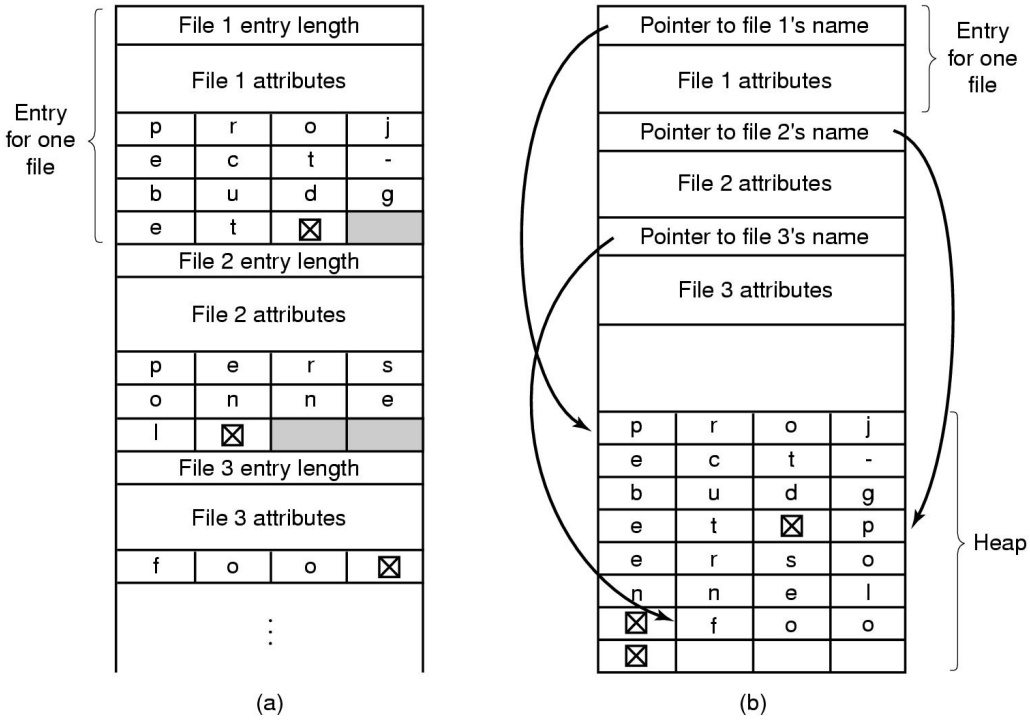


Slika 84. I-node



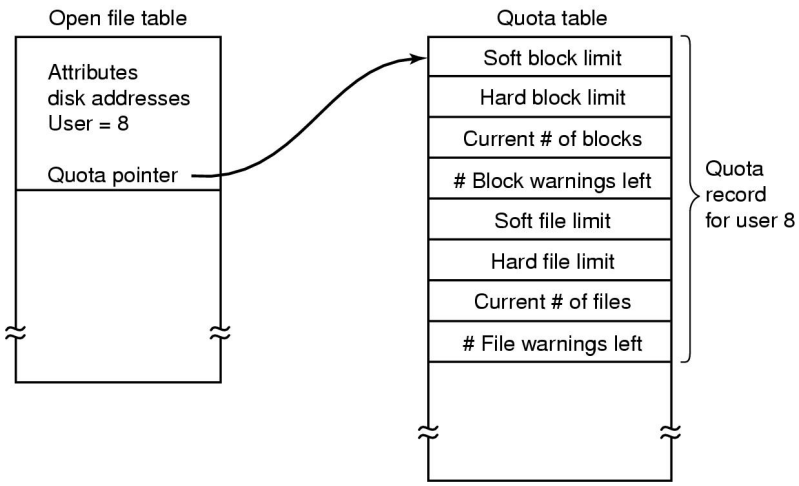
Slika 85. Pohrana atributa: a) u direktoriju b) izvan direktorija

Pohrana dugih imena datoteka postiže se upisom podataka o datoteci na početak zapisa u direktoriju, a kraj zapisa se rezervira za ime datoteke koje može biti proizvoljne dužine. Slika 86. prikazuje način upisa dugih imena datoteka. Za princip zapisa *in-line* imena se dodaju na kraj zapisa, a kod *heap* principa sva imena datoteka su dodana na kraj direktorija te se pointerima usmjeruje na njih iz zapisa datoteke u direktoriju.



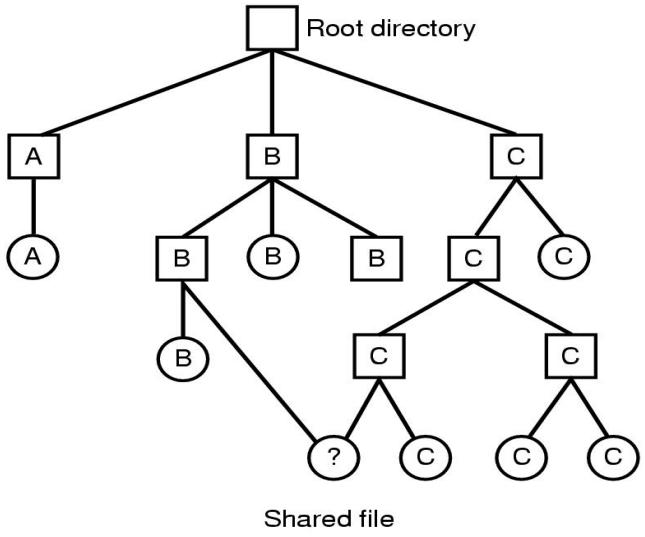
Slika 86. Pohrana dugih imena: a) in-line b) heap

Za grupe korisnika moguće je definirati ograničenja (*quota*) u pogledu korištenja vanjske memorije. Koristi se posebna tabela sa popisom svih grupa korisnika sustava i njihovim ograničenjima (slika 87.). Pri korištenju prostora vanjske memorije kontroliraju se dopuštene vrijednosti zauzetosti vanjske memorije i korisnik upozorava o pokušaju prekoračenja dozvoljenih vrijednosti.

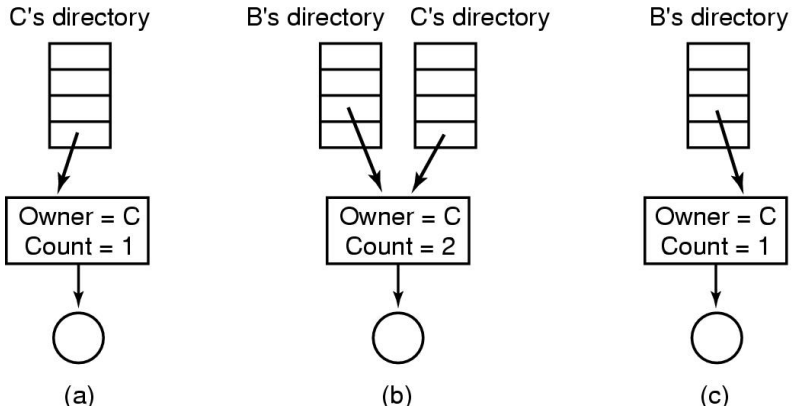


Slika 87. Primjena kvote za korisnike

Dijeljenje datoteka između dva korisnika prikazano je slikom 88. Vidljivo je da zapisi iz dva direktorija upućuju na jednu datoteku. Slika 89. prezentira situacije pri brisanju datoteke: kada vlasnik datoteka obriše datoteku provjerom brojača count registrirat će se korisnik koji dijeli datoteku i datoteka će biti brisana samo iz direktorija vlasnika datoteke (takvu vezu zovemo hard link), blokovi datoteke obrisati će se tek kada korisnik koji dijeli datoteku prekine vezu sa datotekom nakon čega se datoteka briše iz direktorija korisnika, a blokovi datoteke označavaju slobodnima. Ukoliko se koristi soft link u slučaju da vlasnik obriše datoteku korisnik koji je dijelio datoteku njoj više ne može pristupiti jer se zapis o dijeljenoj datoteci referencirao na zapis datoteke u direktoriju vlaniska datoteke.

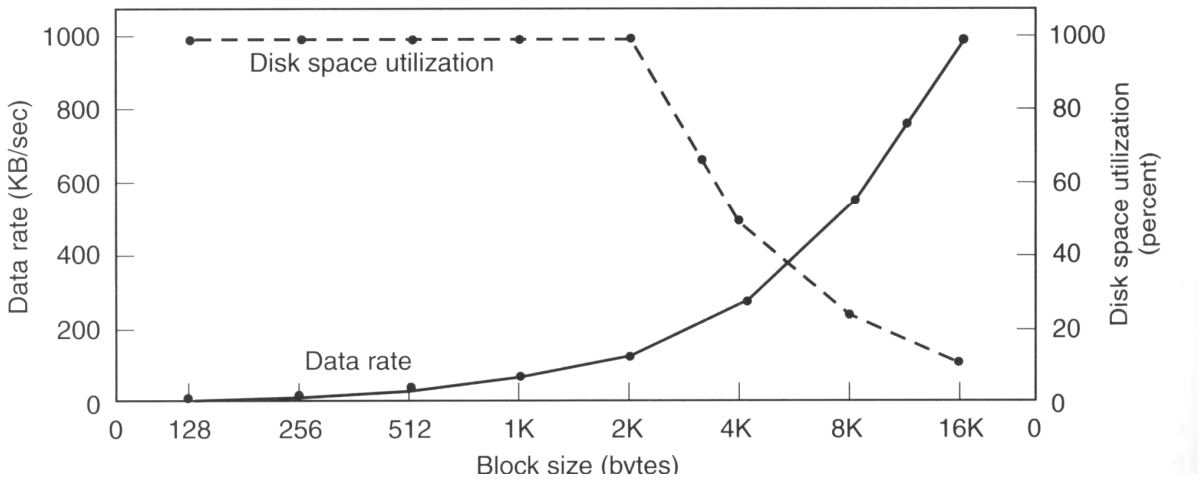


Slika 88. Dijeljenje datoteka



Slika 89. Postupak brisanja datoteke (za hard link)

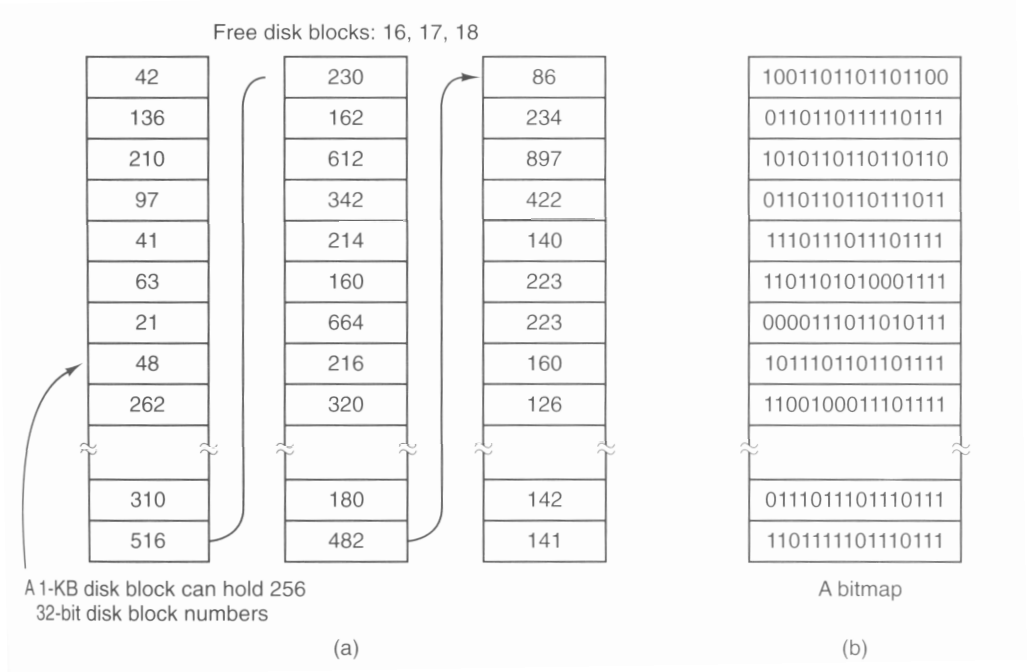
Veličina bloka je bitan element koji određuje rad sa vanjskom memorijom. Za male veličine blokova pojava interne fragmentacije (nepopunjenosti posljednjeg bloka datoteke) je zanemariva, ali je brzina prijenosa malena jer je potreban velik broj pristupa blokovima diska za učitavanje podataka zbog njihove male veličine. Vrijeme pomaka magnetne ruke diska i vrijeme rotacije diska je mnogo veće u odnosu na vrijeme čitanja diska pa je prijenos podataka spor. Za velike veličine blokova situacija je obrnuta: brzina čitanja je velika jer se pristupa malom broju blokova, ali se adresni prostor vanjske memorije slabo koristi jer je izražena interna fragmentacija. Slika 90. prikazuje krivulje brzine prijenosa podataka i iskorištenja adresnog prostora vanjske memorije. Zaključak je da je kompromisno rješenje za veličinu bloka 512 B, 1 ili 2 KB.



Slika 90. Krivulje brzine prijenosa podataka i iskorištenja vanjske memorije

Alokacija slobodnog i zauzetog prostora vanjske memorije ostvaruje se primjenom bit mapa ili posebnih blokova koji pohranjuju podatke o slobodnim blokovima (slika 91.). Ako je vanjska memorija većinom nepopunjena racionalnije rješenje su bit mape, a ako je memorija većinom popunjena prikladnija je upotreba posebnih blokova za pohranu podataka o slobodnim blokovima.

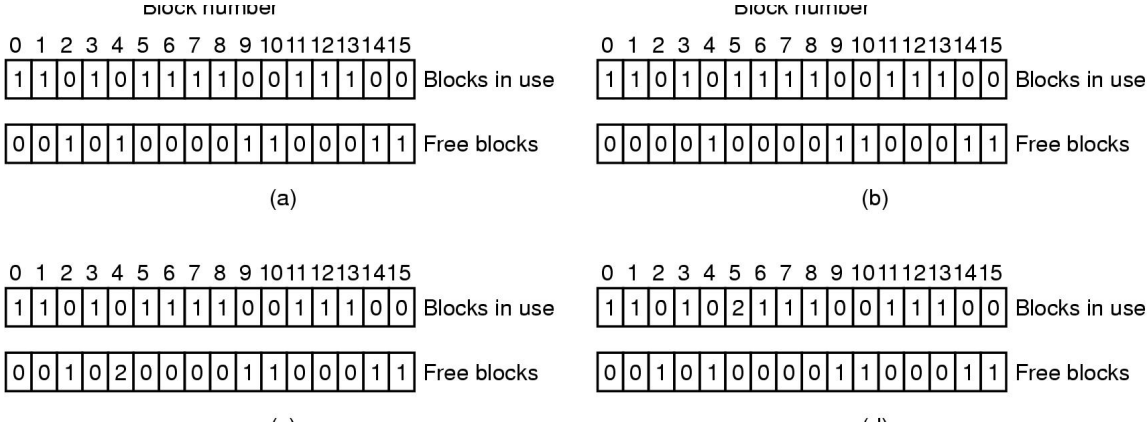
Konzistentnost vanjske memorije znači usklađenost podataka o zauzetim i slobodnim blokovima datoteke sa stvarnim stanjem vanjske memorije. Pri ažuriranju datotečnog sustava može doći do nekorektnog izvođenja procesa i grešaka u konzistentnosti sustava. Zato se provodi postupak provjere konzistentnosti datotečnog sustava u slučaju nekorektnog izvođenja pojedinih procesa. Pri provjeri konzistentnosti koriste se dvije liste: liste slobodnih i liste zauzetih blokova. Za svaki blok u jednoj listi mora biti vrijednost 1, a u drugoj 0 (blok slobodan: lista praznina ima vrijednost 1, a lista blokova 0). Svako odstupanje od navedenog mora se korigirati. Slika 92. prikazuje moguće situacije u listama slobodnih i zauzetih blokova. Pod a) je korektno stanje, slučaj b) prezentira pojavu dvaju 0 (blok sa oznakom 2) koja se rješava promjenom vrijednosti u listi slobodnih vrijednosti sa 0



Slika 91. Ažuriranje slobodnog prostora na vanjskoj memoriji

a) posebni blokovi b) bit mape

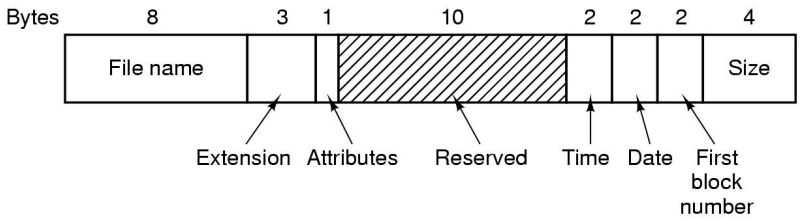
na 1. Pod c) imamo vrijednost u listi slobodnih blokova veću od 1 (blok sa oznakom 4), a rješava se postavljanjem vrijednosti u listi slobodnih blokova na 1, te u slučaju d) imamo vrijednost veću od 1 u listi zauzetih blokova (za blok sa oznakom 5) koja se rješava kopiranjem bloka u slobodni blok i postavljanjem vrijednosti u listi zauzetih blokova za blok 5 i dodijeljeni slobodni blok na 1.



Slika 92. Provjera konzistentnosti datotečnog sustava

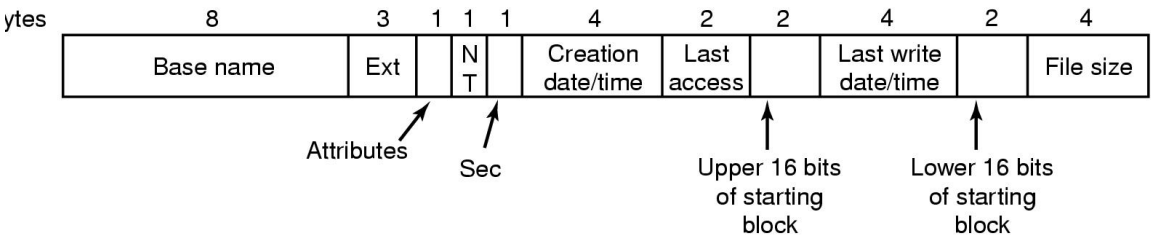
Sigurnost datotečnog sustava ostvaruje se izradom sigurnosnih kopija (backup) koji ovisno o važnosti podataka mogu biti godišnji, mjesečni, tjedni ili dnevni. Prema opsegu podataka mogu biti potpuni ili inkrementirani za pohranu promijenjenih podataka od posljednjeg backup-a (kontrolira se vrijeme modifikacije datoteka). Za sustave sa jako izraženim stupnjem sigurnosti (sustav upravljanja letjelicom) koriste se dodatni rezervni uređaj za pohranu podataka na koji se pohranjuju podaci identično kao na originalni uređaj pa u slučaju kvara glavnog uređaja rezervni preuzima njegovu funkciju.

Slika 93. prikazuje strukturu direktorija za MS-DOS operativni sustav.



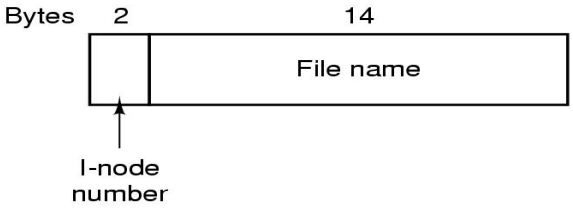
Slika 93. Direktorij u MS-DOS-u

Slika 94. prezentira strukturu direktorija Windows 98 operativnog sustava.

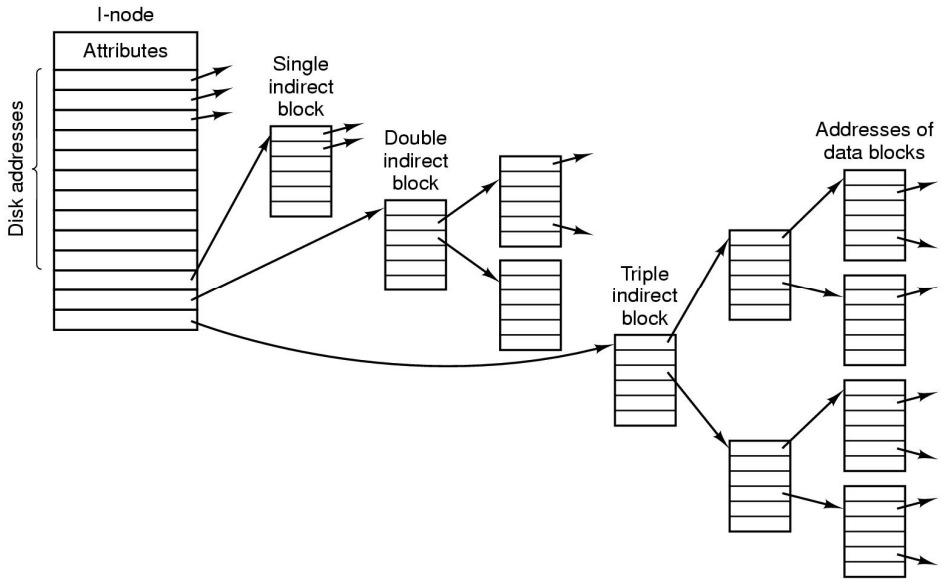


Slika 94. Direktorij u Windows 98 operativnom sustavu

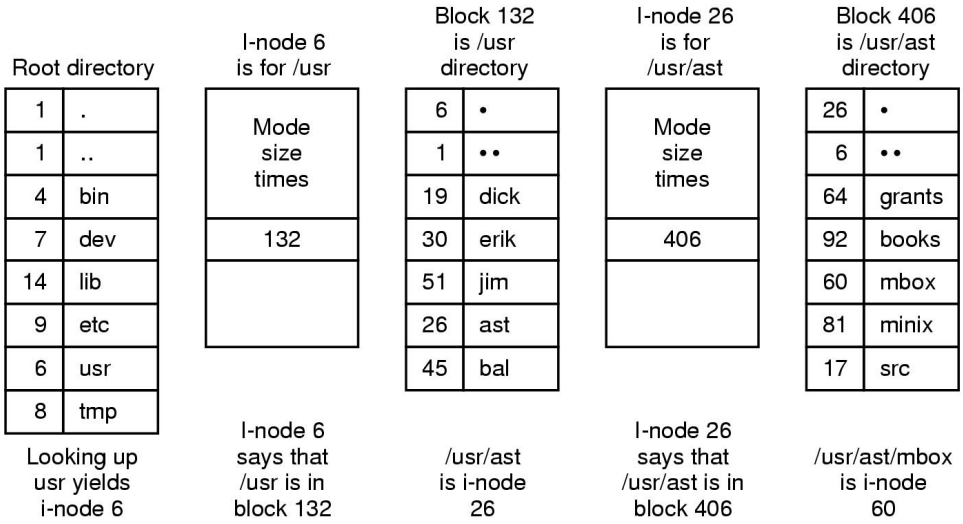
Slika 95. prezentira strukturu i-noda (UNIX operativni sustav). Slika 96. prezentira pohranu podataka blokova za velike datoteke uz upotrebu jednostrukog, dvostrukog i višestrukog bloka za pohranu podataka, a slika 97. prikazuje postupak pretraživanja direktorija za UNIX operativni sustav.



Slika 95. Direktorij u UNIX-u



Slika 96. UNIX i-node



Slika 97. Pristup datoteci /usr/ast/mbox

7. SIGURNOST I ZAŠTITA

Sigurnost operativnog sustava podrazumjeva mjere kontrole i sprječavanja nedozvoljenog pristupa podacima korinika. Zaštita u operativnom sustavu usmjerene je ka kontroli korištenja resursa pri izvođenju procesa.

7.1. SIGURNOST

Prijetnje usmjerene ka podacima korisnika usmjerene su ka:

- Tajnost podataka – spječavanje pristupa podacima korisnika
- Integritet podataka – sprječavanje izmjene podataka korisnika
- Dostupnost usluga sustava – sprječavanje uskraćivanja usluga sustava krajnjim korisnicima

Potencijalni neovlašteni korisnici podataka mogu biti:

- Tehnički nedovoljno educirani korisnici – korisnici koji iz radoznalosti otvaraju datoteke i mailove drugih korisnika
- Studenti, sistem programeri, operateri – dokazuju se probijanjem zaštita računalnih sustava
- Programeri u bankama i ostali korisnici sa namjerom korištenja bankovnih i drugih računa – motivirani su osobnom dobiti
- Vojna i industrijska špijunaža – otkrivanje podataka konkurencije ima ključnu važnost za uspješnost razvoja civilnih i vojnih projekata.

Moguća mjesta upada u sustav:

- Korištenje podataka datoteka koje se više ne koriste – podaci ostaju u memoriji neko vrijeme i čitanjem memorijskih adresa moguće je doći do podataka
- Korištenje nedozvoljenih sistemskih poziva ili dozvoljenih sistemskih poziva sa nedozvoljenim parametrima
- Lažni proces logiranja – proces simulira postupak logiranja te upisuje podatke korisnika koji se logira u posebnu datoteku

- Prekidanje postupka logiranja – pokušava se 'zbuniti' sustav kako bi se proces logiranja prihvatio kao uspješan
- Suradnja sa administratorom sustava – prijateljski dobivena dozvola pristupa sustavu uz izbjegavanje standardne procedure dobivanja pristupa sustavu
- Neovlašteni pristup podacima o korisnicima sustava u uredu firme.

Sigurnost ne može spriječiti gubitak podataka u sljedećim slučajevima

- Atmosferski utjecaji (vatra, grom, zemljotresi itd.)
- Hardverske i softverske greške
- Ljudske greške.

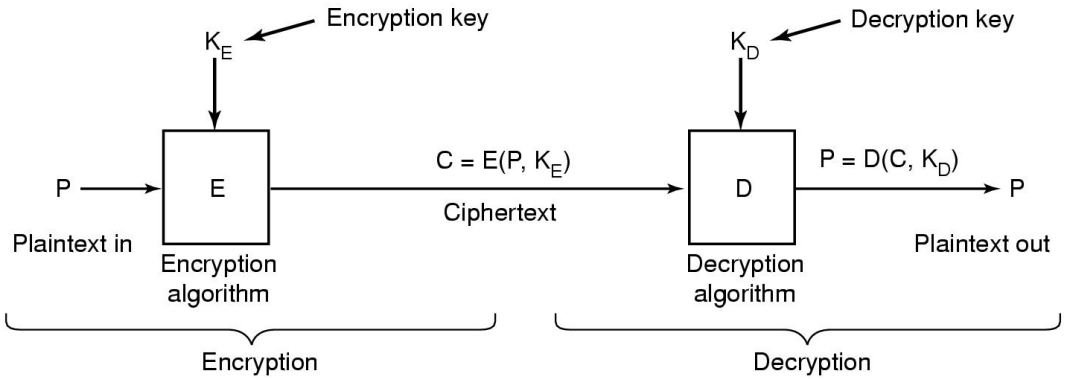
Za sigurnost nužno je korištenje kriptografije. Kriptografijom prevodimo tekst iz plaintext oblika (originalni dokument) u ciphertext oblik (kodirani dokument) na način koji autoriziranim osobama omogućuje povratak teksta u plaintext oblik.

P – plaintext; K_E – ključ za kodiranje (encription)

C – ciphertext; E – algoritam za enkripciju

$C = E(P,K_E)$ – funckija kodiranja

Slika 98. prikazuje postupak kodiranja i dekodiranja dokumenta.



Slika 98. Kodiranje i dekodiranje dokumenta

Kodiranje se koristi za:

- Kriptiranje upotrebom tajnog kjuča
- Kodiranje javnim ključem
- Digitalni potpisi.

Kriptiranje upotrebom tajnog ključa podrazumijeva poznavanje ključa za kriptiranje za osobu koja šalje podatke i osobu koja prima podatke.

Primjer ključa pošiljaoca je:

plaintekst A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

ciphertekst Q W E R T Y U I O P A S D F G H J K L Z X C V B N M

to znači da se riječ ATTACK prevodi u QZZQEA.

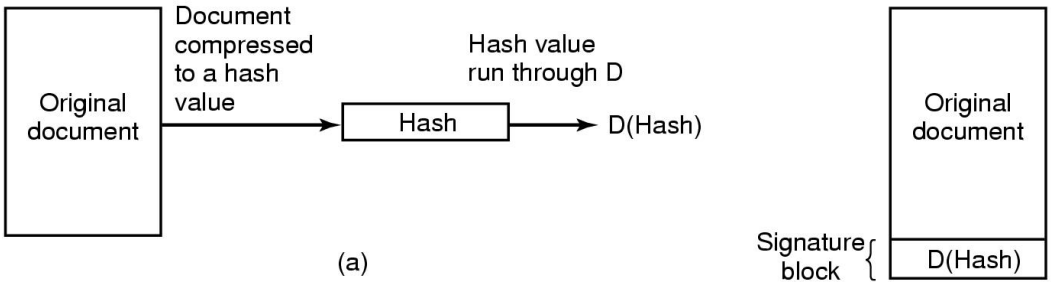
Ključ primaoca je:

chipertekst *Q W E R T Y U I O P A S D F G H J K L Z X C V B N M*

plaintekst A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Kodiranje javnim ključem omogućuje slanje dokumenata primaocu bez poznavanja privatnog ključa primaoca. Javni ključevi korisnika su javno dostupni i koriste se za slanje podataka korisnicima, ali samo korisnik koji ima privatni ključ može otvoriti dokument.

Digitalni potpisi koriste se za provjeru autentičnosti podataka poslanih u digitalnom obliku. Pri slanju dokumenata koristi se algoritam za kodiranje (MD5 ili SHA) kako bi se dobila hash verzija dokumenta, zatim se korištenjem privatnog ključa dobiva signiture blok D(hash). Primatelj korištenjem MD5 ili SHA algoritma otvara dokument i ako je sa njim suglasan korištenjem javnog ključa pošiljaoca provjerava ispravnost ključa pošiljaoca (slika 99.).



Slika 99. Slanje digitalnih dokumenata

Provjera autentičnosti korisnika izvodi se primjenom:

- passworda
- biometrije
- mjera kontrole.

Primjena passworda označava korištenje para login-password za svakog korisnika. Podaci o paru login-password pohranjeni su u posebnu datoteku i pri provjeri korisnika pristupa se pretraživanju datoteke sa podacima o loginu i passwordu kako bi se pronašao zapis koji odgovara upisnim podacima korisnika. Login je u datoteku upisan jednako njegovoj stvarnoj vrijednosti, a password je kodiran i ne odgovara stvarnom passwordu korisnika čime se sprječava doznavanje passworda za korisnike sustava. Pri pokušaju otkrivanja lozinke koristi se ping za detektiranje aktivnih računala, a zatim telnet za provjeru pristupa sustavu. Kombiniranjem različitih kombinacija znakova za login i password nastoji se otkriti login i password koji dozvoljava pristup računalu. Nakon toga pokušavaju se dobiti administratorske ovlasti i podesiti sustav prema osobnim potrebama.

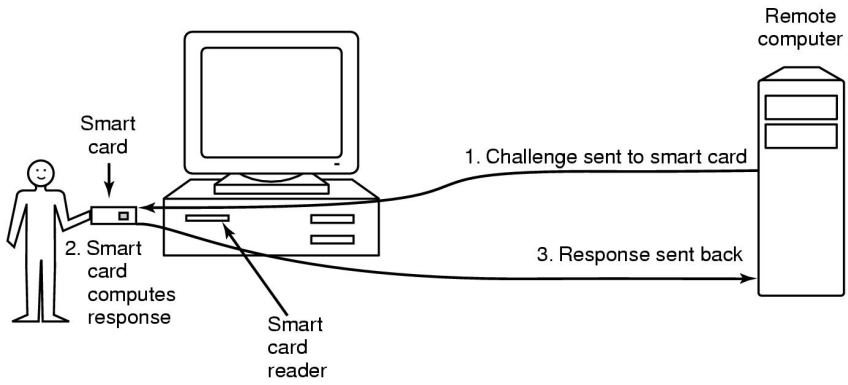
Mjere za poboljšanje passworda mogu biti:

- Minimalno korištenje 7 znakova
- Razlikovanje malih i velikih slova
- Korištenje najmanje jedne znamenke ili specijalnog znaka
- Izbjegavati imena osoba ili riječi iz riječnika.

Vrste paswworda

- Password za jednokratnu upotrebu – za svaku novu upotrebu koristi se novi password
- Kombinacije sistemskog i korisničkog passworda
- Vremensko ograničenje valjanosti passworda
- Korištenje dodatnih pitanja
- Korištenje fizičkih objekata
    - Magnetne kartice sa pohranjenim vrijednostima (1 KB)
    - Pametne magnetne kartice (4 MHz 8-bit CPU, 16 KB ROM-a, 4 KB EPROM-a 512 B RAM-a, 9600 bps komunikacijski kanal).

Slika 100. prikazuje upotrebu magnetnih kartica u postupku prijave korisnika sustava.



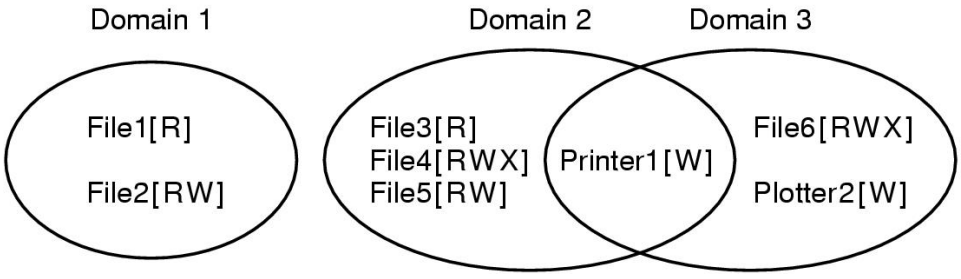
Slika 100. Upotreba magnetnih kartica

Biometrijom se provjeravaju svojstva korisnika sustava koja se specifična za svaku osobu (zjenica oka, boja glasa, otisak dlana i slično). U pravilu se biometrija provodi u kombinaciji sa passwordom.

Mjere sigurnosti mogu se odnositi na valjanost passworda u određenom vremenu dana ili određenom danu u tjednu, zatim pri uspostavi veze korisnika može se veza prekinuti i pozvati korisnika kako bi se otkrila lokacija korisnika koji pristupa sustavu itd.

7.2. ZAŠTITA

Zaštitom se ograničavaju dostupni resursi procesima tijekom njihovog izvođenja. Pri izvođenju procesa određuju se domene koje sadrže listu resursa i način njihove upotrebe. Tijekom izvođenja proces je u jednoj domeni, ali može i prelaziti iz domene u domenu. Slika 101. prikazuje primjer domena.



Slika 101. Prikaz domena.

Implementacija domena ostvarena je matricama koje za retke imaju oznake domena, a za stupce oznake resursa (slika 102.). U presjeku retka i stupca upisani su podaci koji definiraju način upotrebe resursa u domeni (R – čitanje, W – upisivanje, X – izvođenje).

	Object							
	File1	File2	File3	File4	File5	File6	Printer1	Plotter2
Domain 1	Read	Read Write						
2			Read	Read Write Execute	Read Write		Write	
3						Read Write Execute	Write	Write

Slika 102. Matrica za implementaciju domena

Prijelaz između domena ostvaruje se dodavanjem naziva domena u posljednje stupce matrice i unosom oznake u presjek reda domene iz koje se prelazi u stupac domene u koju se prelazi (slika 203.).

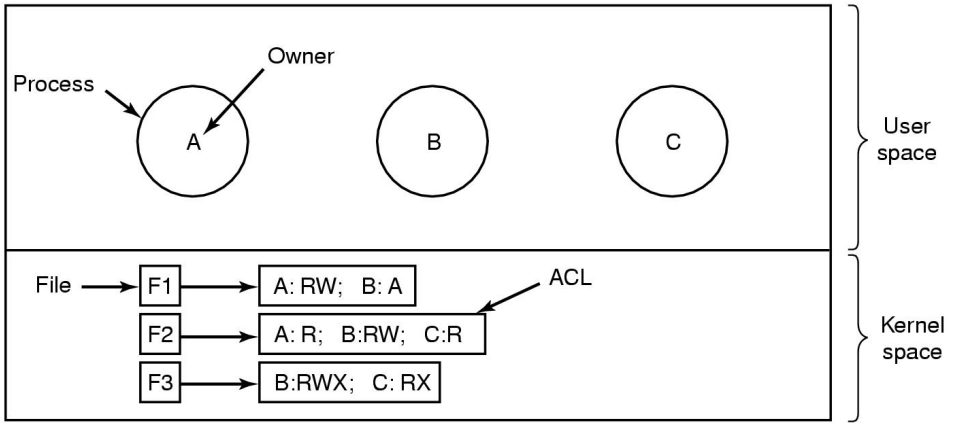
	Object										
	File1	File2	File3	File4	File5	File6	Printer1	Plotter2	Domain1	Domain2	Domain3
main 1	Read	Read Write								Enter	
2			Read	Read Write Execute	Read Write		Write				
3						Read Write Execute	Write	Write			

Slika 103. Prijelaz između domena

Matrice su nepraktične zbog potrebe za rezerviranjem velikog adresnog prostora u memoriji, pri čemu su uglavnom nepopunjene. Zato se koriste sljedeće modifikacije:

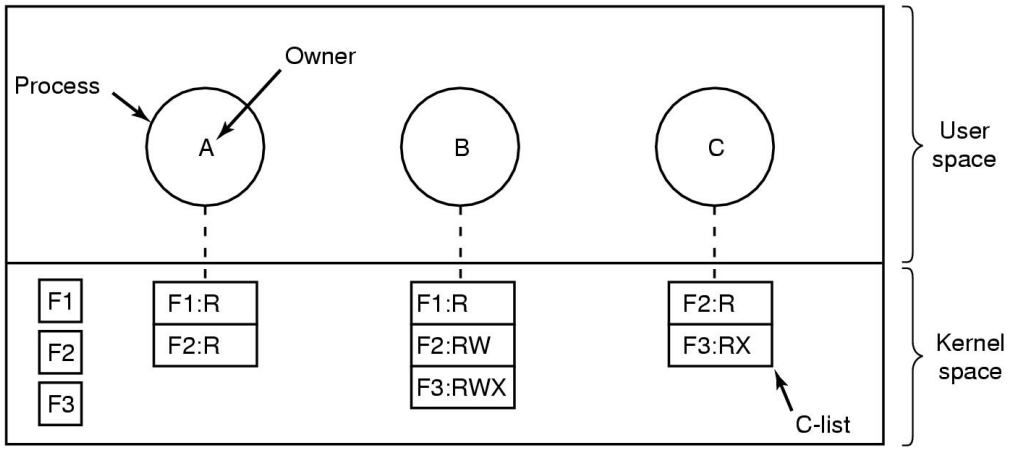
- Pristupne kontrolne liste (Access contol list)
- Liste sposobnosti (Capabilities list)

Pristupne kontrolne liste sadrže popis grupa korisnika sa pravima korištenja za svaki resurs posebno. Dakle svaki resurs ima podatke o korisnicima koji ga smiju koristiti i načinu upotrebe resursa (slika 104.). Pristupne kontrolne liste prikazuje reducirane stupce matrice.



Slika 104. Pristupne kontrolne liste

Liste sposobnosti sadrže podatke o resursima koji se koriste u domeni i načinu njihove upotreba i prikazuju reducirane retke matrice (slika 105.).



Slika 105. Liste sposobnosti

8. LITERATURA:

1. Andrew S. Tanenbaum: Modern Operating Systems, Prentice Hall, 2001.
2. Andrew S. Tanenbaum: Operating Systems, Design and Implementation, Prentice Hall, 1997.
3. Silberschatz, Galwin: Operating system concepts, Addison Wesley, 1994