OPERATIVNI SISTEMI

1 - UVOD U OPERATIVNE SISTEME

1.1 - OPERATIVNI SISTEMI

* Operativni sistem - deo sistemskog softvera koji je odgovoran za upravljanje računarskim resursima, kao i što bolje uslove za korišćenje računara. Suština OS-a je da obezbedi okruženje u kom će korisnici imati mogućnost da što jednostavnije pokreću programe, pri čemu se hardver koristi što efikasnije. Spadaju u najkomplikovanije i najveće programe.

1.2 - OSNOVNI KONCEPTI OS-A

* Osnovni zadaci OS-a su da omogućе što efikasniju realizaciju sledećih aktivnosti:
  - upravljanje procesima - kreiranje, izvršavanje, dodela resursa, sinhronizacija, redosled izvršavanja...
  - upravljanje memorijom - raspoređivanje procesa u radnoj memoriji
  - upravljanje I/O uređajima - kontrola i transfer podataka između uređaja i ostatka sistema
  - upravljanje podacima - čuvanje podataka, vođenje evidencije, manipulacija itd.
  - upravljanje mrežama - umrežavanje i komunikacija među računarima

* Jezgro (kernel) - deo OS-a u koji su smeštene najvažnije funkcije koje obezbeđuju osnovne servise OS-a. Odgovoran je za funkcionisanje sistema i upravlja resursima na najnižem nivou. Prvi se učitava u radnu memoriju i ostaje u njoj do završetka rada. Najniži sloj u hijerarhiji sistemskog softvera. Zbog osetljivosti se učitava u odvojen deo memorije.

* Sistemski pozivi - usluge koje OS može da pruži aplikativnim programima se ostvaruju putem sistemskih poziva. Uz njihovu pomoć programi komuniciraju s jezgrom i pomoću njega dobijaju mogućnost da izvrše osetljive operacije u sistemu. Praktično, to je skup funkcija koji predstavlja interfejs ka OS. Dozvoljavaju samo operacije koje ne mogu biti štetne po sistem. Sistemski pozivi koriste jezgro da bi omogućili razne servise OS-a.

* Režimi rada - korisnički (user mode) i sistemski (kernel mode). U sistemskom režimu je moguće izvršiti sve f-je, dok je korisnički režim redukovan. Pristup I/O uređajima, zaštićenim delovima memorije itd. je dozvoljen samo sistemskom režimu rada. Aplikativni programi se izvršavaju u korisničkom režimu. Prelazak između ova dva režima se obezbeđuje kroz sistemske pozive.

* Korisničko okruženje - olakšava korišćenje ostalih delova OS-a, a i celokupnog rač. sistema. Korisnička okruženja možemo deliti na:

- linijska korisnička okruženja - konzole, terminali, komandne linije itd. koje omogućavaju da se OS-om upravlja kucanjem tekstualnih komandi. Komandni interpreter je najvažniji deo i njegova uloga je da naredbe i podatke prepozna i naloži OS-u izvršavanje odgovarajućih ~~procesa~~ operacija.
- ekranska korisnička okruženja - omogućavaju upravljanje OS-om korišćenjem cele površine ekrana. Osnovni deo je radna površina na kojoj su aplikacije i podaci predstavljeni vizuelnim elementima.

*Drajveri - upravljački programi koji se nadograđuju na kontrolere i omogućavaju komunikaciju i upravljanje I/O uređajima. Programiraju se tako da za različite tipove iste vrste uređaja definišu jedinstven skup dozvoljenih instrukcija.

1.3 - ARHITEKTURE OPERATIVNIH SISTEMA

*Monolitni sistemi - u jezgru se nalaze svi servisi OS-a zajedno sa drajverima, sve skupa povezani u jedan program. Svi delovi se pokreću u istom trenu, izvršavaju u sistemskom režimu i u istom delu memorije. Ovakva povezanost doprinosi brzini i efikasnosti jezgra. Najveća mana ovakvih sistema je loša otpornost na greške. Primeri: Windows 98, GNU/Linux.

*Slojeviti sistemi - OS izgrađen od zasebnih celina koje se nadograđuju jedna na druge. Slojevi se imple-

-mentiraju tako da mogu da koriste isključivo usluge prvog sloja ispod sebe. Najveći problem je neefikasnost. Primeri: THE, Multics.

* Sistemi zasnovani na mikrojezgru - podrazumeva minimalno jezgro sa najosnovnijim funkcijama. Deo funkcija iz jezgra ide u zasebne prostore, kako bi se greške što manje odražavale na jezgro. Usluge koje obavljaju slične zadatke se grupišu u servere. Mikrojezgro pruža veći stepen sigurnosti u odnosu na složenije sisteme, ali brzina nije odlika nijednog od ta dva. Primeri: Minix, L4.

* Hibridni sistemi - kompromis između monolitne i mikrojezgro arhitekture. Funkcije koje su veoma bitne ili se često izvršavaju se spuštaju u jezgro dok ostatak funkcioniše na nivou izvršavanja. Primer: Windows

* Sistemi zasnovani na egzojezgru - ideja je da jezgro obezbedi resurse i ~~obezbedi~~ prepusti aplikacijama rad sa njima. Funkcije se smeštaju u posebne biblioteke, kojima korisnik može da dodaje i svoje. Ovakav način rada može znatno da ubrza performan međutim, fleksibilnost za kor. aplikacije može da dovede do neurednosti koda. Primer: ExOS.

1.4 - RAZVOJ OS I ISTORIJAT

* Multiprogramiranje - osnovna ideja je da se u radnu memoriju smesti više programa kako bi se poboljšala iskorišćenost procesora. Memorija bi se podelila u particije u koje se učitavaju programi. Glavni cilj je maksimalno povećanje iskorišćenosti sistema, ~~ta~~ što manje vremena.

* Deljenje vremena - koncept koji se zasniva na deljenju računara između više korisnika. Procesor se deli „vremenski" tako što bi svaki korisnik dobio određeno vreme, po čijem isteku procesor prelazi na sledećeg.

* Multitasking - efikasniji način implementacije ideje multiprog. Podrazumeva da je jedinica izvršavanja posao koji ne mora nužno da obuhvata ~~ne~~ izvršavanje između dve I/O operacije

* Multiprocesiranje - postojanje više procesora u sistemu.

1.5 - ZNAČAJNI OS

* Multics - zajednički pokušaj MIT, Bell laboratorija. Puno procesorskog vremena je trošio na odluke, a malo je ostavljao korisnicima. Nikada nije ozbiljnije zaživeo na računarima.

* UNIX familija - uprošćena verzija Multicsa (Bell + Denis Rič + Ken Tom). Razvijen na C-u. Prepušten univerzitetima na dalji razvoj. Najpoznatije implementacije na Berkliju (BSD).

* GNU/Linux - Stolman 1983. na osnovama UNIX-a razvija GNU inicijativu (GNU's not UNIX). Linus Torvalds 1991. počinje razvoj Linux-a, po ugledu na Minix. Spajanjem GNU-a i Linux-a nastaje GNU/Linux OS.

* Microsoft familija - Bil Gejts (izof MS-a) sa IBM-om sklapa ugovor o kupovini MS-DOS-a. Vremenom se odvaja od DOS-a i razvojem grafičkog interfejsa osvaja tržište. Najadekvatnije ime - „Windows" - najbolje opisuje računarske „prozore" koji su temelj Windowsa i dan danas.

* Apple familija - Mac OS se pojavljuje 1984. Prva izdanja su mogla da pokreću i izvršavaju samo jedan program u datom trenutku. Posebno se izdvaja iOS, koji Apple razvija za telefone, poznat po svojoj multitač tehn.

* Android - najpopularniji OS za mobilne uređaje. Zasnovan na Linuxu. Njegov razvoj započinje istoimena kompanija koju kupuje Google 2005. Android je open-source, što proizvođačima omogućava da ga prilagođavaju svojim potrebama.

2 - UPRAVLJANJE PROCESIMA

2.1 - PROCESI

* Proces - program u izvršavanju. Izvorni kod programa se prevodi na mašinski jezik, učitava u radnu memoriju računara i izvršava na procesoru, čime program postaje aktivni proces. Da bi obavio zadatak, procesu su potrebni resursi poput procesora, mem, U/I, podataka itd. Zadatak OS-a je da obezbedi da se proces može efikasno izvršavati. Procesi mogu biti korisnički ili sistemski, a efikasnost se postiže konkurentnim izvršavanjem, što je osnova mnogih koncepta, poput multiprogramiranja. Bitne informacije o toku izvršavanja procesa sadrže i registri, a posebno sistemski brojač koji vodi računa o tome dokle se stiglo.

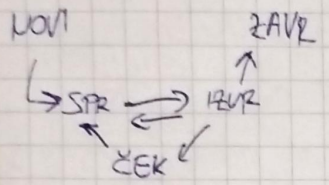
* OS i proces - zadatak OS-a je da obezbedi efikasne mehanizme za kreiranje i brisanje, upravljanje, komunikaciju i sinhronizaciju procesa.

* Proces u memoriji - po učitavanju u memoriju, procesi raspolažu sa 4 memorijska segmenta. To su:

  1. stek segment - lokalne promenljive i parametri f-ja;
  2. hip -||- - podaci tokom izvršavanja procesa, tj. ono što se dinamički alocira;
  3. data -||- - globalne promenljive;
  4. kôd -||- - instrukcije koje proces treba da izvrši.

2.2 - STANJA PROCESA

*Osnovna stanja - OS definiše skup stanja u kojima se proces može naći, u zavisnosti od situacije u sistemu. To su:



1. NOVI - proces je upravo kreiran, OS napravio potrebnu dokumentaciju i prebacio 2°;
2. SPREMAN - proces čeka da OS donese odluku da mu bude dodeljen procesor;
3. IZVRŠAVANJE - proces se izvršava na procesoru;
4. ČEKANJE - za dalji rad procesa potreban zauzet resurs, te se čeka na njegovo oslobađanje.
5. ZAVRŠEN - proces završio, potrebno izbaciti iz sistema.

*Suspendovan proces - određeni OS podržavaju i ovo stanje, kada procesi oslobađaju resurse koje su zauzimali pre susp., ali i dalje ostaju proces. Do ovog dolazi zbog preopt. sistema usled velikog broja spremnih proces.

2.3 - KONTROLNI BLOK PROCESA

*Kontrolni blok - struktura u kojoj se vodi precizna evidencija o procesima radi olakšavanja multiprog. Sadrži sledeće podatke:

- PID - ID procesa, dodeljen pri pokretanju;
- STANJE - informacija o stanju;
- PROG. BROJAČ - čuva info o sledećoj komandi;
- SADRŽAJ REGISTARA - podaci koji se nalaze u reg. kako bi posle gubitka prava brzo odmah da nastavi;

- PRIORITET - važnost procesa;
- ADRESA MEMORIJE;
- ADRESE ZAUZETIH RESURSA.

* OS i K.B. - realizuje sledeće zadatke:
  - kreiranje KB za nov proces;
  - uništavanje KB koji se izvršio;
  - menjanje stanja procesa, kao i prioriteta;
  - izbor procesa za izvršavanje.

* Prebacivanje konteksta - ~~proces~~ postupak kojim se proces koji se trenutno izvršava prekida, pamte parametri, a zatim se umesto njega pokreće neki drugi proces. Modul koji je zadužen za izvršavanje ovog postupka se naziva dispečer.

2.4 - NITI

* Nit - osnovna jedinica za izvršavanje u okviru procesa. To su delovi jednog procesa i osim resursa procesa, imaju i svoje resurse. Svaka nit poseduje svoje registre, PC i stek, kao i TID. Rad sa više niti (multithreading) podrazumeva mogućnost OS-a da podrži konkurentno izvršavanje više niti.

* Kontrolna nit - inicijalna nit svakog procesa. Ima zadatak da obavi potrebne inicijalizacije i kreira ostale niti po potrebi.

* Prednost niti - značajne uštede mem. prostora i vremena; zauzimaju manje prostora od nezavisnog procesa, pružaju mogućnost npr. da rastereti rad u situacijama kada bi nezavisan proces privremeno zaustavio ostale operacije.

*Preslikavanje niti - mapiranje korisničkih niti u niti jezgra.
Postoje:

1. VIŠE-U-JEDNU - više korisničkih niti koje pripadaju jednom procesu se mapiraju u jednu nit jezgra. Jezgro manipuliše isključivo sa procesima, dok nitima upravlja korisnički nivo. Koja nit se izvršava se odlučuje na korisničkom nivou, dok jezgro upravlja nitima na procesorskom nivou. Glavna mana je da ako blokira jedna nit, blokiran je i ceo proces.

2. JEDNA-U-JEDNU - 1:1 presl. Upravljanje nitima se prepušta jezgru. Niti se kreiraju sporije i manje ih je, ali zato jezgro paralelno radi sa njima. Ovime se obezbeđuje konkurentno izvršavanje niti, ali se ograničava i broj niti u jezgru, samim tim i korisničkom delu.

3. VIŠE-U-VIŠE - hibrid prethodna dva pristupa. Broj niti, kao i način raspodele, se određuje na korisničkom nivou, dok se upravljanje nitima prepušta jezgru. Najkompleksniji je, ali i najkvaliteetniji.

2.5 - REDOVI PROCESA

* Redovi - formiraju se usled lakšeg upravljanja procesima

1. RED POSLOVA - početni red, u koji se smeštaju pokrenuti procesi;

2. RED SPREMNIH PROCESA - sadrži procese izabrane po nekom kriterijumu, određenom na nivou OS-a. Procesi na ovom nivou su spremni i učitani su u RAM. Izbor procesa zavisi od odnosa procesor : U/I, radi što produktivnijeg sistema.

3. REDOVI ČEKANJA - procesi u stanju čekanja.

- Redovi se uglavnom implementiraju kao povezane liste KB-ova koje ne moraju biti organizovane po FIFO-u.

2.6 - PLANERI

* Planeri - vrše odabir procesa i njihovo raspoređivanje po redovima, kao i organizaciju čekanja. Obično postoje dve vrste planera:

1. DUGOROČNI - iz reda poslova biraju procese koji će aktivno da se uključe u sistem i počnu sa izvršavanjem. Retki su, jer su retko potrebni.

2. KRATKOROČNI - biraju koji će se od spremnih procesa izvršavati i koliko vremena imaju. Kod njih je brzina rada važnija od optimalnosti odluke.

* Swapping - proces suspendovanja procesa i kasnijeg vraćanja u memoriju. Posebno se implementira srednjeročni planer.

2.7 - VIŠEPROCESORSKI SISTEMI

VPS - višepr. sis.
JPS - jednopr. sis.

* VP sistemi - dozvoljavaju veću fleksibilnost kada je rad sa procesima i nitima u pitanju. Procesi se mogu izvršavati paralelno na razl. procesorima, što sistem čini efikasnijim u odnosu na JPS. Kada su u pitanju metode raspoređivanja procesa, razlikujemo:

1. SIMETRIČNO MULTIPROC. - procesori su ravnopravni i procesi se raspoređuju na bilo koji. Svi procesori dele zaj. memoriju i pristup U/I.

2. ASIMETRIČNO -||- - neki proc. mogu biti zadženi za određene f-je. Jedan procesor se proglasi glavnim i on upravlja jezgrom OS-a, dok se ostali uključuju na poziv glavnog.

* Raspoređivanje procesa - postoje dva pristupa:

1. ZAJEDNIČKI RED ČEKANJA - onemogućava ravnomerno opterećenje procesora. Jednostavno se implementira, ali treba voditi računa da se isti proces ne dodeli više procesora.

2. RAZLIČITI REDOVI - može se dogoditi da pros. čekaju, a da postoje slobodni procesori. Izbegavanje ovoga se postiže balansiranjem opterećenja.

* Balansiranje opterećenja - teži da se što ravnomernije podele poslovi između procesora. Najčešće se primenjuju dva načina:

1. PUSH MIGRACIJA - iz reda opterećenog procesora proces šaljemo u redove besposlenog ili manje zauzetog. Potrebno je implementirati razne algoritme za određivanje opt.

2. PULL MIGRACIJA - u red slobodnijeg proc. se prebacuju procesi iz reda zauzetijih. Obično se bira proces visokog prioriteta ili malo vremena izvršavanja.

* Afinitet - proces razvija afinitet prema procesoru na kom je počeo izvršavanje. Razlikujemo slab i jak afinitet.

2.8-9 - ALGORITMI RASPOREĐIVANJA PROCESA

* Raspoređivanje procesa - donošenje odluka o toku izvršavanja procesa koji se nalaze u memoriji. Dobro organizovano raspoređivanje može značajno ubrzati rad sistema. Optimizacija podrazumeva:
  - max iskorišćenost prostora;
  - max propusna moć (br. procesa istovremeno);
  - min vreme obrade;
  - min vreme odziva;
  - min vreme čekanja.

* FCFS algoritam - First Come, First Served. Procesor se dodeljuje procesima po redosledu kojim su ga tražili. Implementira se FIFO-m. Proces „staje" na kraj reda i čeka svoj red. Jednostavno se implementira, ali je vreme čekanja često veliko (efekat konvoja).

* SPF alg. - Shortest Process First. Favorizuju se procesi sa najkraćim vremenom izvršenja. Svakom procesu se pridružuje

očekivano vreme za izvršavanje na procesoru dok procesor dobija procese sa najmanjom vrednošću. FCFS najbolji. Može se implementirati sa i bez prekidanjem procesa. Optimalan je za srednje vreme čekanja, dok je najveći problem procena dužine trajanja procesa. Tada se koristi ekspon. sredina, sa formulom:

$$T_{n+1} = \alpha T_n + (\alpha+1) t_n$$

*Alg. sa prioritetom - svakom procesu se pridruži prioritet na osnovu kog se bira proces koji će se izvršiti. Prioriteti se mogu definisati interno i eksterno. Mana je „izgladnjivanje" procesa (nizak prioritet → dugo čekanje).

*Kružni alg. - svaki proces unapred dobije vreme ~~izvršenja~~ interval, nakon čijeg isteka procesor prelazi na sledeći proces. Prekinut proces ide na kraj reda spremnih. Efikasnost zavisi od kvantuma vremena. Poželjno je da kvantum bude duži od vremena potrebnog za preb. konteksta.

*Redovi u više nivoa - red spremnih se podeli u više redova sa razl. prioritetima. Svaki red može imati sopstven algoritam planiranja, između redova postoji jasno definisana razlika u prioritetu. Proces koji je prvi u redu čeka da se svi redovi iznad njega isprazne.

* —||— sa povratnom vezom - procesi mogu da šetaju između redova.

2.10 - RAD U REALNOM VREMENU

* Real-time OS - OS sa specijalnom namenom. Koriste se u situacijama kada postoje vremenska ograničenja za izvršenje unapred definisanih poslova. U zavisnosti od toga koliko moraju da poštuju vremenska ograničenja, dele se na:
    1. ČVRSTE - u sistemu postoji bar jedan proces koji ima čvrsti vremenski rok. Ako se rok ne ispoštuje, ceo sistem pada.
    2. LABAVE - poštovanje zadatog roka je poželjno, ali ne od presudnog značaja. Prekoračenje roka se može tolerisati do nekih granica.

* Procesi u RTOS - periodični i neperiodični.

* Prioritet u RTOS - može se dodeliti:
    1. STATIČKI - obavlja se u fazi implementiranja sistema i ne mogu se menjati u toku rada.
    2. DINAMIČKI - može se menjati u toku rada sistema.

* RMS alg. - primenjuje se kod periodičnih procesa. Pretpostavlja n procesa P_i, sa svojim periodima T_i i vremenom izvršavanja C_i. Prioriteti se dodeljuju na osnovu perioda procesa.

$$\sum_{i=0}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}}-1)$$

* EDFS alg. - najviši prioritet dodeljuje procesu koji je najbliži svom krajnjem roku za izvršavanje.

3 - KONKURENTNOST I SINHRONIZACIJA PROCESA

* Trka za resurse - situacija u kojoj krajnji rezultat zavisi od redosleda izvršavanja koraka različitih procesa koji manipulišu zajedničkim podacima.

NPR. X=13

P1: X → R 13
R++ 14
PREKID

P2: X → R 13
R++ 14
R → X 14
PREKID

P1: R → X 14

* Primer trke - dva procesa uvećavaju zajedničku promenljivu X za 1. Ako drugi proces prekine prvi, prvi nije stigao da upiše vrednost izračunavanja u X, te krajnji rezultat će biti jednak onom koji je potreban.

3.1 - KRITIČNA SEKCIJA

* Kritična sekcija - deo programa u kom se pristupa zajedničkim podacima.

* Rešenja - isključivanje prekida se vrlo retko primenjuje jer bi to dovelo do smanjenja efikasnosti sistema. Dobro rešenje treba da zadovolji:

  - UZAJAMNA ISKLJUČIVOST - dva procesa ne mogu u isto vreme biti u kritičnoj sekciji
  - USLOV PROGRESA - proces koji nije u krit. sek. i ne želi da uđe u nju ne treba da ometa druge procese koji da uđu u nju.
  - USLOV KONAČNOG ČEKANJA - trebalo bi da postoji razumna granica koliko jedan proces može da čeka na ulazak u krit. sek.

P1	P2

želi_1 = 1;
while(želi_2 == 1) {
if(na_redu == 2) {
želi_1 = 0;
while(na_redu == 2)
čeka;
želi_1 = 1;
}}
KRIT_SEK
na_redu = 2;
želi_1 = 0;
	želi_2 = 1;
	while(želi_1 == 1) {
	if(na_redu == 1) {
	želi_2 = 0;
	while(na_redu == 1)
	čeka;
	želi_2 = 1;
	}}
	KRIT. SEK.
	na_redu = 1;
	želi_2 = 0;

* Pitersonov alg. - proces „džentlmenski" ustupa prednost drugom procesu kada najavi da hoće u krit. sek. Ova prednost važi samo ako oba procesa žele da uđu u krit. sek. U suprotnom, proces odmah ulazi u krit. sek.

P1	P2

želi_1 = 1;
na_redu = 2;
while (želi_2 == 1 &&
na_redu == 2)
čeka;
KRIT SEK
želi_1 = 0;
	želi_2 = 1;
	na_redu = 1;
	while(želi_1 == 1 && na_redu == 1)
	čeka;
	KRIT. SEK.
	želi_2 = 0;

* Lamportov alg. - uopštenje Pitersonovog alg. za n procesa. Zasnovan je na ideji redosleda opsluživanja mušterija u pekari. Radi za proizvoljan broj procesa i zadovljava sve uslove.

* Hardverska rešenja - ideja je da se naprave mašinske instrukcije koje su u stanju da urade dve atomične operacije. Najčešće se koriste:

1. TAS (TEST AND SET) → operiše kao A=TAS(B), pri čemu A=B, B=1.

2. FAA (FETCH AND ADD) → operiše kao FAA(A,B), pri čemu A=B, B=A+B.

3. SWAP → operiše SWAP(A,B), A=B, B=A.

TAS ALGORITAM	SWAP ALGORITAM

ne_moze=1;
while(ne_moze==1)
ne_moze=TAS(zauzeto);
KRIT. SEK.
zauzeto=0;
	kljuc=1;
	while(kljuc!=0)
	SWAP(kljuc, brava);
	KRIT. SEK.
	brava=0;

-Rešenje podrazumeva da TAS postoji kao operacija -Može se primeniti na proizvoljan broj procesa, ali ne garantuje za čekanje.	-brava inic. na 0 kao globalna promenljiva

3.3 - BEZ AKTIVNOG ČEKANJA

* Ove metode se zasnivaju na zaustavljanju procesa bez aktivnog čekanja i njihovo „buđenje" u odg trenutku. Bolje su od pristupa 1Č, ali i kompilovanje za implementaciju.

* Semafori - apstraktni tip podataka, tj. struktura koja može da blokira proces na neko vreme i propusti ga kada se ~~[illegible]~~ steknu uslovi. Osim kreiranja i brisanja, postoje još dve posebne operacije: P i V. P(S) smanjuje vrednost semafora za 1, dok V(S) uvećava. Proces se propušta dalje ako je vrednost semafora nenegativna. Ovakvi semafori su brojački, dok postoje i binarni, kojima su dozvoljene samo 0 i 1.

PRIMENA SEMAFORA

```
P(S);
[KRIT SEK]
V(S);
```

* Struktura semafora - osim vrednosti semafora, uglavnom čuvaju i listu procesa na čekanju.

P(S)	V(S)

S.vr=S.vr-1;
if(S.vr<0){
dodati pr u S.lista,
blokirati pr;
}
	S.vr = S.vr+1;
	if(S.vr <=0){
	izbaciti pr. iz S.lista,
	odblokirati pr;
	}

P(S1);
c--;
if(c<0){
V(S1);
P(S2);
}
V(S1);
	P(S1);
	c++;
	if(c<=0){
	V(S2);
	}
	ELSE
	V(S1);

*Kritični regioni – implementacija zaštite pristupa KS na višem prog. jeziku. Potrebno je definisati prom. deli više procesa i oznakom region obezbediti ekskluzivni pristup toj promenljivoj u okviru niza naredbi u okviru regiona. KR daju garanciju da nijedan drugi proces ne može da pristupi deljenoj promenljivoj, kao i da toj promenljivoj smemo prist. samo u okviru regiona. Slaba tačka im je sinhronizacija procesa. Napredniji koncept podrazumeva uslovne kritične regione, koji se aktiviraju na osnovu logičkih uslova.

ATOMIČNO SAMO

```
v: deljeni int;
REGION
    V=V+1;
ENDREGION
```

*Monitori – najviši nivo apstrakcije kada je KS u pitanju. Predstavljaju konstrukcije slične klasama, koje mogu da sadrže procedure i promenljive. Glavna odlika je da u jednom trenutku može da bude aktivan samo jedan proces. Za potrebe sinhronizacije su uvedene specijalne uslovne promenljive. Nad ovakvim pr. definisane su wait i signal komande za blokiranje i odblokiranje procesa. Nisu ekvivalentne brojačkim semaforima jer čekanje procesa mora da nastupi pre slanja signala. Pristup KS se uz monitore vrlo lako rešava. Implementiraju se na nivou prog. jez. i koriste se na kompl. način. Pružaju više mogućnosti i od sem. i od KR.

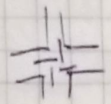
MONITOR

```
monitor: klasa
  S,P: uslovne prom.
  P1(){ };
  P2(){ };
  ⋮
  Pn(){ };
  init(){ };
```

4 - ZAGLAVLJIVANJE

* Zaglavljivanje - situacija u kojoj dva ili više međusobno vezanih procesa blokiran čekaju na resurse koje nikad neće dobiti. Trajno je i za posledicu često ima zagl. celog sistema. Ako sistem sadrži zagl. procese, smatra se i da je on sam zaglavljen.

AUTOMOBILI NA RASKRSNICI



* Živo blokiranje - situacija u kojoj procesi ~~nemaju~~ nisu blokirani, ali nemaju napretka u izvršavanju.

* Izgladnjivanje - u sistemu generalno postoji napredak, ali neki procesi dosta dugo ne napredjuju.

* Da bi do zagl. došlo, neophodno je:

NEOPHODNI USLOVI

1. UZAJAMNO ISKLJUČIVANJE - u jednom trenutku tačno jedan proces može da koristi resurs, drugi čeka
2. ČEKANJE I DRŽANJE - dok drži resurse sa kojima radi, proces može da zahteva nove resurse i čeka trenutno nedostupne
3. NEMOGUĆNOST PREKIDANJA - OS nema pravo da oduzme resurse procesu kom ih je dodelio, već to može da uradi samo sam proces.
4. KRUŽNO ČEKANJE - može da postoji lanac procesa koji čekaju jedni na druge, čineći pri tome krug.

*Do zagljuđ: neće doći ako bilo koji od datih uslova nije ispunjen.

*Pristupi: mogu se svrstati u četiri grupe:

1° SPREČAVANJE - sistemske mere se implementiraju u OS i isključuju mogućnost zagl.

2° IZBEGAVANJE - dinamičke mere koje se pred. kako bi se sistem vodio kroz stanja koja dob. da ne dođe do zagl. Liberalnija od 1°.

3° DETEKCIJA I OPORAVLJANJE - do zagl. može doći, ali postoje mehanizmi za detektovanje zagl, brisanje zagl. procesa i eventualno spašavanje dela resultata, ukoliko je moguće.

4° NEPREDUZIMANJE BILO ČEGA - mera primljena na sistemima gde zagl. retko dolazi ili u situacijama neužitnih poslova. Najjednostavnije rešenje, podrazumeva ponovno pokretanje procesa ili celog sistema u nadi da se zagl. ne pojavi opet.

X SEM 4°

4.1 - SPREČAVANJE

*Sprečavanje - statičke, sistemske mere koje obično nisu algoritamski zahtevne, a za cilj imaju da u startu definisanim pravila eliminišu bar jedan od uslova za zagl. Troškovi ovih alg. su mali, ali su zato ograničenja velika.

* Prevencija ček. i drž. - glavna ideja je uvođenje ograničenja koje onemogućava da proces drži resurse dok traži nove. Postoje dva pristupa

1° Pre izvršavanja, proces zahteva da mu se dodele svi resursi, da bi tek nakon toga krenuo u izvršavanje. Loša strana ovog pristupa je što proces sve vreme drži resurse, a i uvek često nije poznato koji su sve resursi potrebni

2° Proces traži resurse kada su mu potrebni, ali pre toga vraća sve resurse OS-u kako bi OS odredio da li će ih vratiti ili podeliti nekom drugom procesu.

* Eliminacija nemogućnosti - ideja je da se, u slučaju kada se potrebe procesa ne mogu ispuniti, otpuštaju svi resursi koje je proces držao, i ustupaju se drugima na korišćenje.

* Prevencija kruž. ček. - koriste se rešenja koja podrazumevaju enumeraciju. Ideja je da se svaki ~~broj~~ resurs označi, a da proces koji sadrži određene resurse može da zahteva samo one resurse kojima je dodeljen veći broj od brojeva njegovih resursa.

$$F(R_i) = k \Rightarrow F(R_j) > F(R_i)$$

Glavni problem je sama enumeracija resursa, kao i ponovno enumerisanje pri pojavi novog resursa.

* Prevencija ček. i drž. - glavna ideja je uvođenje ograničenja koje onemogućava da proces drži resurse dok traži nove. Postoje dva pristupa:

1. Pre izvršavanja, proces zahteva da mu se dodele svi resursi, da bi tek nakon dobitka krenuo u izvršavanje. Loša strana ovog pristupa je što proces sve vreme drži resurse, a i vrlo često nije poznato koji su sve resursi potrebni.
2. Proces traži resurse kada su mu potrebni, ali pre toga vraća sve resurse OS-u kako bi OS odredio da li će ih vratiti ili podeliti nekom drugom procesu.

* Eliminacija nemogućnosti - ideja je da se, u slučaju kada se potrebe procesa ne mogu ispuniti, otpuštaju svi resursi koje je proces držao, i ustupaju se drugima na korišćenje.

* Prevencija kruž. ček. - koriste se rešenja koja podrazumevaju enumeraciju. Ideja je da se svaki resurs označi, a da proces koji sadrži određene resurse može da zahteva samo one resurse kojima je dodeljen veći broj od brojeva njegovih resursa.

$$F(R_i) = k \Rightarrow F(R_j) > F(R_i)$$

Glavni problem je sama enumeracija resursa, kao i ponovno numerisanje pri pojavi novog resursa.

4.2 - IZBEGAVANJE

* Izbegavanje - dinamičke mere koje se preduzimaju u određenim momentima u zavisnosti od stanja na sistemu. Troškovi algoritma veći nego kod sprečavanja, ali se povećava efikasnost sistema.

* Bankarev algoritam - 1965. predložio Dijkstra. Ime dobio po principu davanja zajmova u bankama. Podrazumeva da je pre početka izvršavanja svakog od procesa poznat maksimalan broj resursa svakog tipa koji će mu biti potreban. Ideja je da se dinamički ispituje pokušaj dodele resursa kako nikad ne bi došlo do kružnog čekanja. Smatra se da je stanje sistema BEZBEDNO ako sistem može u određenom vremenskom (sekvenci) da dodeljuje resurse kako bi svaki proces mogao da završi sa radom. Ako stanje nije bezbedno postoji mogućnost da dođe do zag.

  Kada sistem dobije zahtev za resurse od procesa, on pokreće Bankarev alg. kako bi doneo odluku da li će dodeliti resurse proc, odložiti ili odbiti zahtev. Ovaj algoritam je teorijski dobar, ali ima ozbiljna ograničenja u praksi.

* Izbegavanje - dinamičke mere koje se preduzimaju u određenim momentima u zavisnosti od stanja na sistemu. Troškovi algoritma veći nego kod sprečavanja, ali se povećava efikasnost sistema.

* Bankarev algoritam - 1965. predložio Dijkstra. Ime dobio po principu davanja zajmova u bankama. Podrazumeva da je pre početka izvršavanja svakog od procesa poznat maksimalan broj resursa svakog tipa koji će mu biti potreban. Ideja je da se dinamički ispituje pokušaj dodele resursa kako nikad ne bi došlo do kružnog čekanja. Smatra se da je stanje sistema BEZBEDNO ako sistem može u određenom redosledu (sekvenci) da dodeljuje resurse kako bi svaki proces mogao da završi sa radom. Ako stanje nije bezbedno postoji mogućnost da dođe do zagl.

  Kada sistem dobije zahtev za resurse od procesa, on pokreće Bankarev alg. kako bi doneo odluku da li će dodeliti resurse proc, odložiti ili odbiti zahtev. Ovaj algoritam je teorijski dobar, ali ima ozbiljna ograničenja u praksi.

4.3 - DETEKCIJA ZAGLAVLJIVANJA

*Det. zag. - dopušta da dođe do zaglavljivanja, a onda primenjuje mere da se ta situacija otkloni. Otkrivanje da je do zagl. došlo može biti:

1° AKTIVNO - alg. za detektovanje zagl. se pokreće i proverava da li je došlo do zagl., u zavisnosti od frekvencije zagl.

2° PASIVNO - alg. se pokreće samo kada sistem nema aktivnosti ili nešto nije u redu.

*Stanje sistema se može skicirati kao graf, gde krugovi predstavljaju procese, a pravougaonici resurse. Detekcija zagl. se tada svodi na pronalaženje ciklusa u grafu. Ako resursi imaju više instanci, ciklus ne znači da je došlo do zaglavljivanja.

*Oporavak od zagl. - kada se utvrdi da je došlo do zagl, sistem bi trebalo da interveniše i pomogne da se zaobide situacija. Alg. za oporavak, koji se primenjuje, podrazumeva da se uklone svi blokirani procesi, što je neefikasno rešenje jer se tako gube svi priv. rezultati. Efikasniji alg. podrazumeva prekidanje servisa 1 po 1 dok se ne otkloni zaglavljivanje. Pristup koji podrazumeva oduzimanje resursa se koristi kako bi dati resursi pomogli u otklanjanju zagl.

PROCES SE PREKIDA NA OSNOVU PRIORITETA, VREMENA IZVRŠAVANJA I RESURSA KOJE KORISTI

5-UPRAVLJANJE MEMORIJOM

* Memorija - osnovni resurs OS-a. Svaka proc. instrukcija se mora naći u memoriji, a za izvršavanje istih su potrebni podaci iz memorije. Pošto je ogr. resurs, efikasno korišćenje memorije je jedan od osnovnih zadataka OS-a, što čini modul koji se zove MENADŽER MEMORIJE. U memoriju spada bilo koji fizički uređaj koji privremeno ili trajno čuva podatke. Deli se na osnovu brzine pristupa:

1° REGISTRI - memorijske ćelije ugrađene u procesor i u njima se nalaze podaci koje procesor trenutno obrađuje. Privremeno čuvaju podatke.

2° KEŠ MEMORIJA - veoma brza memorija koja čuva često korišćene podatke, a može da predstavlja i bafer između 1° i 3°. Obično se nalazi u samom procesoru, ali se delovi mogu naći i van njega.

3° PRIMARNA MEM - sadrži podatke i instrukcije sa kojima procesor trenutno operiše. Brzina između 2° i 4°. Postoje potrebne vrste ove mem:

a) RAM - procesor može da pristupi bilo kom delu ove memorije, pri čemu vreme pristupa ne zavisi od veličine i tipa podatka. Privremena.

b) ROM - čuva podatke i kada je rač isključen. Moguće je samo čitanje iz ROM-a i svaki računar ima neki ROM-a. EEPROM

4° SEKUNDARNA MEM. - služi za trajno skladištenje podataka.

* Nastavak mem. - sastoji se od niza mem. ćelija koje mogu da sačuvaju najmanju kol. podataka - bit. Bitovi se organizuju u bajtove - grupe od 8 bitova, što predstavlja najmanju adresibilnu jedinicu.

* Memorijska reč - određuje količinu podataka koje procesor može da obradi odjednom. Zavisi od arhitekture računara, i trenutno su najzastupljeniji računari od 64 bita, dok su u prošlosti postojali i sa 8, 16 i 32 bita.

* Radna memorija - može se predstaviti kao matrica bitova, odnosno shvatiti kao niz bajtova. Svaki program koji se izvršava mora se učitati u radnu memoriju. U trenutku pisanja programa se koriste SIMBOLIČKE ADRESE, dok proces radi sa FIZIČKIM ADRESAMA. Proces prevođenja simboličkih u fizičke se naziva POVEZIVANJE ADRESA. U najčešćem slučaju, prevodilac generiše RELATIVNE ADRESE u odnosu na početak dela mem. koja se dodeljuje procesu. Povezivač povezuje sve objektne module u izvršni program. Prilikom učitavanja u memoriju, punilac prevodi relativne adrese u fizičke. Ako se program može premeštati, tada se pov. adr. dešava tokom izvršavanja.

* MMU - hardverski uređaj koji preslikava logičke adrese u fizičke. Jednostavni MMU se predstavlja kao uređaj sa baznim registrom (fizička = bazni + logička).

*OS - prilikom upravljanja memorijom bi trebalo da ispuni sledeća dva uslova:

1° Svaki proces mora imati dovoljno memorije za izvršavanje i ne sme pristupati memoriji nekog drugog procesa, kao i obrnuto.

2° Različite vrste memorije unutar r.s. moraju biti korišćene tako da se svaki proces izvršava što efikasnije.

5.1 - MONOPROGRAMIRANJE

*Monoprogramiranje - u memoriji se u jednom trenutku izvršava samo jeden korisnički proces. Kod ovog pristupa postoje različite organizacije memorije:

1° OS na niže adrese, proces koristi više adr,
2° OS na vrhu mem, proces koristi niže adr,
3° OS na dno, drajveri na vrh, korisniku ostatak.

Učitavanje OS-a se izvršava prilikom uključivanja sistema, kada punilac prenosi OS iz sekundarne u radnu memoriju. Pri ovakvom načinu upravljanja potrebno je zaštititi OS od slučajnih ili namernih izmena od strane korisnika, što čini ZAŠTITNI REGISTAR. Adresni prostor počinje od 0, dok korisnik ima pravo na sve adrese posle zaš. reg. Vrednost zaš. adrese može da se pomera, ali je bitno da bude fiksirana tokom izvršavanja programa.

5.2 - MULTIPROGRAMIRANJE

* Da bi se povećala iskorišćenost procesora, koriste se napredniji pristupi zasnovani na multiprogramiranju. Ovakvi pristupi omogućavaju da se više procesa istovremeno nađe u memoriji, sa pseudoparalelnim izvršavanjem. Kada neki proces čeka U/I operaciju, drugi mogu da se izvršavaju mimo njega.

* Prebacivanje - ideja ove tehnike je da se blokirani proces privremeno prebaci u sekundarnu memoriju, a da se u memoriju učita drugi proces. Na ovaj način se formira red blokiranih procesa, odakle procesi prelaze u red spremnih kad se steknu uslovi za dalje izvršavanje. Efikasnost zavisi od brzine sek. mem. Teži se da vreme izvršavanja procesa bude veće od vremena prebacivanja kako bi iskorišćenost procesora bila što bolja. Napredniji pristup podrazumeva preklapanje prebacivanja i [illegible]

* Particionisanje - ideja je da se memorija podeli na n particija, koje predstavljaju neprekidne delove mem. U svaku particiju se može smestiti po jedan proces. Ako su sve particije zauzete, formira se red spremnih procesa, iz kog se izlazi proces nakon oslobađanja particija. Particije se najčešće štite uz pomoć dva registra za svaku particiju. Postoje dva pristupa particionisanju: statički i dinamički.

* Statičke particije - memorija se statički deli na fiksni broj particija. Broj particija zavisi od raz. faktora, kao što su veličina procesa i radne memorije, brzine procesa itd. Veličine particija su fiksne i najčešće nisu iste. Manje particije idu manjim procesima, veće većim. Postoje dve strategije za odabir particije:

1° svaka particija ima svoj red postava u koji se smeštaju procesi koji odg. veličini particije,

2° svi postavi se smeštaju u jeden red, dok planer donosi odluku koji je sledeći proces.

* Dinamičke particije - veličine particija nisu fiksirane. Pri pokretanju računara, slobodan deo memorije obrazuje jednu slobodnu particiju. Procesi se učitavaju tako da zauzmu deo slobodne particije koji mu je potreban, dok ostatak ostaje slobodan. Za svaki sledeći proces se traži najveći slobodan deo. OS evidenciju o slobodnim i zauzetim delovima memorije vodi korišćenjem bit-mapa ili povezanih listi.
(0-sl, 1-za)

* Fragmentacija - neželjena pojava kod particionisanja. Nastaje kada u sistemu postoje slobodni delovi memorije, ali nisu dovoljno veliki za korišćenje. Može biti:

STATIČKI → 1° INTERNA - odnosi se na prostor koji ostaje neiskorišćen kada se proces smesti u particiju većih memorijskih kapaciteta od potr.

DINAMIČKA → 2° EKSTERNA - u sistemu postoje slobodni delovi, ali nisu dovoljno veliki za korišćenje. Jedan od načina za rešavanje je KOMPAKCIJA - OS zaustavlja procese i izvršava realokaciju mem. tako da sabije sve procese.

*Algoritmi za dodelu memorije - više njih:

1° FIRST-FIT - smešta proces u prvu slobodnu zonu dovoljno veliku za njega. Pretraga kreće od početka memorije. Često dovodi do fragmentacije

2° NEXT-FIT - modifikacija 1°, podrazumeva da sledeća pretraga počinje na mestu gde je prethodna

3° RANDOM - od svih slobodnih prostora slučajno izabere gde će da smesti proces.

4° BEST-FIT - proces smešta u najmanji slobodan deo potreban za proces

5° WORST-FIT - suprotan 4°

5.3 - STRANIČENJE

*Straničenje - dovodjava da dodeljena memorija ne bude uzastopna. Ideja je da se radna memorija podeli na okvire fiksirane veličine, a logička podeli na stranice iste veličina kao i okviri. Kad ovakve org. logička adresa sadrži broj stranice, kao i poziciju u okviru iste. Logičke adrese se u fizičke prevode putem tabele stranica, koje OS pojedinačno generiše za svaki proces. Za određivanje procesa odgovoran je planer posla. Zaštita memorije se vrši uz pomoć zaštitnih bitova.

veličina → uglavnom 2^n

*Tabela stranica -
- 1° čuvanje u registrima - male tabele
- 2° -//- u mem - veće tabele, tada na njih pokazuje PTBR
- 3° asocijativna memorija - posebna keš mem. specijalizovana za straničenje. HIT RATIO - uspešno pronalaženje podatka

5.4 - SEGMENTACIJA

* Prevedeni korisnički program se može posmatrati kao skup segmenata. Osnovna ideja je da se svakom segmentu dodeli poseban mem. prostor, pri čemu je svaki segment atomičan. Mogu se naći bilo gde u memoriji, ali moraju biti u neprekidnom bloku.

* Tabela segmenata - dodeljuje se svakom procesu. Svaki red tabele sadrži redni broj segmenta, adresu početka i dužinu segmenta. STBR sadrži adresu lokacije tabele. Ta dužinu tabele vodi računa STLR.

* Karakteristike - prednost se ogleda u zaštiti memorije. Pruža mogućnost deljenja koda i podataka između više procesa. Dodeljivanje mem. je slično kao kod straničenja, dok veličina segmenta može biti različita, pa podseća na dinamičko particionisanje.

6 - VIRTUELNA MEMORIJA

* Virtelna memorija - način upravljanja memorijom koji omogućava da se procesu na raspolaganju stavi memorija drugačije vel. od one koja fizički postoji u sistemu. OS je zadužen da omogući preslikavanje virtuelne memorije u fizičku. Najčešće se koristi za omogućavanje izvršavanja procesa koji nije u potpunosti u radnoj memoriji.

* Prekrivači - ideja je da se omogući izvršavanje programa koji nisu kompletno učitani u memoriju. Osnovni zadatak je da se identifikuju delovi programa koji su relativno nezavisni (prekrivači), koji se odvajaju od glavnog dela programa i smeštaju u sekundarnu memoriju. Prilikom adresiranja mem. se rezerviše prostor za prekrivač najveće veličine, u kom se zatim smenjuju svi prekrivači u zavisnosti od potrebe. Najveća mana je što ovaj proces nije automatski već samo programeri mogu dovoljno dobro da ga urade.

* Dinamičko punjenje - ideja je da se funkcije i procedure ne smeštaju u memoriju sve do trenutka dok ne budu potrebne, tj. pozvane. Nalaze se u sekundarnoj mem, i nakon poziva proces proveri da li je već učitano ili ne. Predrost je što se f-je koje se ne koriste nikad neće ni učitati u memoriju. Jednostavno se implementira.

*Straničenje na zahtev - najefikasniji i najpopularniji vid implementacije vir.mem. Deli program na stranice. Pri pokretanju prog. u radnu mem. se učitavaju samo neophodne stranice. Kada se neka stranica zatraži, prvo se proverava da li je već učitana ili ne, i shodno tome se prebacuje iz sek.mem. u radnu. Veličina logičkog adresnog prostora nije ograničena fizičkom memorijom, čime se ~~omogućuje~~ obezbeđuje velika virtuelna memorija. Implementira se uz pomoć tabele stranica, koja dodatno sadrži bit validnosti (1 - validna, jeste u mem; 0 - nevalidna, nije u mem).
~~Ako~~

*Page fault - pokušaj pristupa stranici koja nije u radnoj mem. ~~Tak~~ Ako se stranica promaši, prvo se proverava da li stranica pripada procesu koji je tražio. Ako to nije slučaj, program se prekida. Ako jeste, stranica se učitava u 3 koraka:

1° oslobađanje/traženje memorijskog okvira
2° učitavanje stranice iz sek. u radnu mem.
3° modifikacija bita validnosti u tabeli stranica

*Čisto straničenje - proces kreće u učitavanje bez jedne učitane strane. Promašaji će se retati u startu sve dok se ne formira radni skup stranica.

*Zamena stranice - ideja je da se po nekom kriterijumu nađe okvir koji će se isprazniti da bi se u njega učitala nova stranica. Omogućava straničenje na zahtev i efikasnost.

~~GRAMICA~~ *Alg

6.9 - ALGORITMI ZA IZBACIVANJE STRANICE

*Slučajno izbacivanje - slučajno se bira stranica za izbacivanje. Jedno od najgorih rešenja. Može da dovede do izbacivanja nekih veoma korišćenih stranica, čime se doprinosi velikom broju promašaja stranica.

*Beladijev optimalni alg - najbolje rešenje. U trenutku promašaja, određen skup stranica je već u memoriji. Cilj je da se izbaci neka stranica koja bi bila potrebna najkasnije u budućnosti. Najveći problem je taj što ga je skoro nemoguće implementirati.

*FIFO alg - iz mem. se izbacuju stranice redosledom kojim su učitane. Mana je što se pri učitavanju stranice od raznosti, koje će se ovim pristupom prve izbaciti.

*Alg. druge šanse - modifikacija FIFA, podrazumeva postojanje bita referisanosti. Bit ref. se postavi na 1, kada dođe na red za izbacivanje postavlja se na 0, resetuje se vreme izv. i gura se na kraj liste. Problem nastaje kada su svi ref. bitovi postavljeni na 1.

~~[illegible]~~ *Alg

GR-ALGORITMI ZA IZBACIVANJE STRANICE

*Slučajno izbacivanje - slučajno se bira stranica za izbacivanje. Jedno od najgorih rešenja. Ovo može da dovede do izbacivanja nekih veoma korišćenih stranica, čime se doprinosi velikom broju promašaja stranice.

*Bolji je optimalni alg - najbolje rešenje. U trenutku promašaja, ažuriran skup stranica je već u memoriji. Ideja je da se izbaci neka stranica koja bi bila potrebna najkasnije u budućnosti. Najveći problem je taj što ga je skoro nemoguće implementirati.

*FIFO alg - iz mem. se izbacuju stranice redosledom kojim su učitane. Ideja je što se pre učitaju stranice od ranije, koje će se ovim postupkom pre izbaciti.

*Alg. druge šanse - modifikacija FIFA, podrazumeva postojanje bita referisanosti. Bit ref. se postavi na 1, kada dođe na red za izbacivanje postavlja se na 0, nastavlja se premeštanje i gura se na kraj liste. Problem nastaje kada su svi ref. bitovi postavljeni na 1.

~~STRANICE~~ *Alg.

GR-ALGORITMI ZA IZBACIVANJE STRANICE

*Slučajno izbacivanje - slučajno se bira stranica za izbacivanje. Jedno od najgorih rešenja. Ovo može da dovede do izbacivanja nekih veoma korišćenih stranica, čime se doprinosi velikom broju promašaja stranica.

*Beladijev optimalni alg - najbolje rešenje. U trenutku promašaja, određeni skup stranica je već u memoriji. Cilj je da se izbaci neka stranica koja bi bila potrebna najkasnije u budućnosti. Najveći problem je taj što ga je skoro nemoguće implementirati.

*FIFO alg - iz mem. se izbacuju stranice redosledom kojim su učitane. Mana je što se prvo učitane stranice od raznosti, koje će se ovim pristupom prve izbaciti.

*Alg. druge šanse - modifikacija FIFA, podrazumeva postojanje bita referisanosti. Bit ref. se postavi na 1, kada dođe na red za izbacivanje postavlja se na 0, resetuje se vreme izv. i gura se na kraj lista. Problem nastaje kada su svi ref. bitovi postavljeni na 1.

*Alg. sata - modifikacija druge šanse, u kom su sve stranice povezane u kružnu listu, kako se ne bi gubilo vreme sa premeštanjem str. na kraj liste.

*LRU - iz memorije se izbacuje stranica koja je korišćena najdalje u prošlosti u odnosu na ostale. Podrazumeva da su skoro korišćene str. važne i da će tako ostati i u budućnosti (analogno i za dalje). Najveća mana je što se ne može predvideti sledeća stranica.

*NRU - aproksimacija LRU. Predlaže samo stranice koje nisu skoro korišćene, ne uzima u obzir koliko dugo nisu korišćene. Implementacija jednostavnija od LRU-a, sa bitom ref. Modifikacija uvodi i bit modifikacije, kako bi alg. imao željene rezultate.

*LFU i MFU - najredje/najčešće korišćena stranica.

Mane:

LFU - ne vodi računa o tek ubačenim stranicama ili stranicama koje više nisu potrebne, a često su korišćene

MFU - problemi su razni