Skripta OS

1.1 Operativni sistemi

Operativni sistem je deo sistemskog softvera koji je odgovoran za upravljanje racunarskim resursima, kao i što je 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. OS je najkomplikovaniji i najveći program.

1.2 Osnovni koncepti OS-a

Osnovni zadaci OS-a su da omoguće što efikasniju realizaciju sledećih aktivnosti:

- **Upravljanje procesima** - kreiranje, izvršavanje, dodela resursa sinhronizacija, redosled izvršavanja…
- **Upravljanje memorijom** - raspoređivanje procesa u radnoj memoriji.
- **Upravljanje I/O uredjajima** - kontrola, transfer podataka između uredjaj i ostatka sistema.
- **Upravljanje podacima** - čuvanje podataka, vođenje evidencije, manipulacija, itd.
- **Upravljanje mrežama** - umrežavanje i komunikacija među uređajima.

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

Uz pomoć **Sistemskih Poziva** OS može da pruži usluge aplikativnim programima. Uz njihovu pomoć programi komuniciraju sa jezgrom i pomoću njega dobijaju mogućnost da izvrše osetljive operacije u sistemu. Sistemski pozivi su skup funkcija koji predstavlja interfejs ka

operativnom sistemu. Dozvoljavaju se samo operacije koje ne mogu biti štetne po sistem. Sistemski pozivi koriste jezgro da bi omogućili razne servise OS-a.

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

Korisničko okruženje olakšava kroišćenje ostalih delova OS-a, a i celokupnog računarskog sistema. Korisnička okruženja možemo deliti na:

- **Linijska korisnička okruženja** - konzole, terminali, komandne linije… koje omogućavaju da se OS-om upravlja kucanjem tekstualnih komandič. Komandni interpreter je najvažniji deo i njegova uloga je da naredbe i podatke prepozna i naloži OS-u izvršavanje odgovarajućih operacija.
- **Ekranska korisnička okruženja** - omogućavaju upravljanje OS-om korišćenjem cele površine ekrana. Osnovni deo je radna površina na kojoj su aplikacije i podaci prestavljeni vizuelnim elementima.

Drajveri su 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 OS-a

Monolitni sistemi u jezgru sadrže sve servise OS-a zajedno sa drajverima, sve skupa povezani u jedan program. Svi delovi se pokreću u istom trenutku, izvršavaju u sistemskom režimu i u istom

delu memorije. Jezgro se u memoriju učitava u celosti kao jedan izvršni program. Veoma velika povezanost funkcija na niskom nivou obično doprinosi brzini i efikasnosti ovakvih jezgara. Jedna od najvećih mana je što je loša otpornost na greške. U slulaju da dođe do greške u nekom od podsistema, dolazi do problema koji mogu uticati na ceo sistem i najčešće je jedino rešenje njegovo ponovno podizanje.

Slojeviti sistemi podrazumevaju da je OS izgrađen od zasebnih slojeva koji se nadograđuju jedan na drugi. Slojevi se implementiraju tako da mogu da koriste isključivo usluge prvog sloja ispod sebe. Najveći problem ovakvih sistema je neefikasnost. Sistemski poziv prolazi kroz više slojeva, pri svakom prolazu se prosleđuju podaci, menjaju parametri…

Sistemi zasnovani na mikrojezgru podrazumevaju 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. Promene memorijskog prostora dovode do kašnjenja i manje propusnosti u poređenju sa sistemima sa monolitnim jezgrom. Mikrojezgro pruža veči stepen sigurnosti u odnosu na slojevite sisteme, ali brzina nije odlika nijednog od ta dva.

Hibridni sistemi su kompromis između monolitne i mikrojegro arhitekture. Funkcije koje su veoma bitne ili se ćesto izvršavaju se spuštaju u jezgro dok ostatak funkcioniše na nivou iznad. Najvažnije je precizno odabrati koje funkcije treba spustiti u jezgro, a koje ostaviti van.

Sistemi zasnovani na egzojezgru imaju deiju da jezgro obezbedi resurse i prepusti aplikacijama rad sa njima. Funckije se smeštaju u posebne biblioteke, kojima korisnik može da dodaje i svoje. Ovakav način rada može znatno da ubrza performanse međutim, fleksibilnost za korisničke aplikacije može da dovede do neurednosti koda.

1.4 Razvoj OS i Istorijat

Multiprogramiranje je osnovna ideja 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, u što manje vremena.

Deljenje vremena je 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. Mali vremenski intervali koje korisnik dobija stvarali su iluziju da korisnik sve vreme ima na raspolaganju ceo procesor koji radi samo za njega.

Multitasking je efikasniji način implementacije ideje multiprogramiranja. Podrazumeva da je jedinica izvršavanja posao koji ne mora nužno da obuhvata izvršavanje procesa na procesoru između dve I/O operacije.

Multiprocesiranje je postojanje više procesora u sistemu.

1.5 Značajni OS

Multics je zajednički pokušaj više univerziteta i laboratorija kao što su MIT, Bell laboratorija… Glavna mana je da je puno procesorskog vremena trošio na odluke, a malo je ostavljao korisnicima. Nikada nije ozbiljnije zaživeo na računarima.

UNIX familija je uprošćena verzija Multics-a. Napisali Ken Tompson i Denis Riči. Razvijen na C-u. Prepušten univerzitetima na dalji razvoj. Najpoznatije implementacije na Berkliju (BSD).

GNU je napisao Stolman 1983. počevši kao inicijativa da se napravi slobodan OS na osnovama UNIX-a, GNU = "GNU nije UNIX". **Linux** je započet 1991. od strane Linusa Torvaldsa. Linux je prvobitno bio monolitan ali spajanjem Linux jezgra sa GNU alatom kompletiran je OS **GNU/LINUX.**

Bil Gejts sa IBM-om sklapa ugovor o kupovini MS-DOSa. Vremenom se odvaja od DOSa i razvojem grafičkog interfejsa osvaja tržište. Najadekvatnije ime “**Windows**”, najbolje opsiuje računarske “prozore” koji su temelj Windowsa i dan danas.

Mac OS se pojavljuje 1984. Prva izdavanja 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 tehnologijil.

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

2.1 Procesi

Proces je 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, memorije, I/O uređaja, podataka… 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 programski brojač koji vodi računa o tome dokle se stiglo sa izvršavanjem procesa.

Zadatak OS-a je da obezvede efikasne mehanizme za:

- Kreiranje i brisanje procesa
- Upravljanje procesima
- Komunikaciju između procesa
- Sinhronizaciju procesa

Kada se **proces** učita **u memoriju**, raspolaže sa 4 memorijska segmenta:

- Stek segment
- Hip segment
- Data segment
- Kod segment

OS definiše skup **stanja** u kojima se procesori mogu naći:

- **Novi** - proces je upravo kreiran, OS je napravio dokumentaciju i prelazi u spremno stanje.
- **Spreman** - proces čeka da operativni sistem donese odluku da mu bude dodeljen procesor.
- **Izvršavanje** - proces se izvršava na procesoru.
- **Čekanje** - proces se izvršavao ali je za njegov dalji rad potreban neki resurs koji nije slobodan tako da on čeka da se stvore uslovi da bi mogao da nastavi sa radom.
- **Završen** - proces je završio sa izvršavanjem i trebalo bi ga izbaciti iz sistema.

Suspendovan proces podržavaju samo određeni OS. Procesi koji su u stanju Spreman ili Čekanje se mogu suspendovati. Suspendovan proces oslobađa resurse koje je zauzimao pre suspenzije i prestaje da konkuriše za druge resurse koji su mu potrebni za izvrašavanje, ali idalje ostaje proces. Do ovoga dolazi zbog preotperećenja sistema usled velikog broja spremnih procesa.

2.3 Kontrolni blok procesa

Kontrolni blok je struktura u kojoj se vodi precizna evidencija o procesima radi olakšavanja multiprogramiranja. Sadrži sledeće podatke:

- **PID** - broj koji proces dobija u trenutku pokretanja, PID-ovi su jedinstveni
- **Stanje procesa** - informacija o trenutnom stanju u kojem se proces nalazi.
- **Programski brojač** - čuva informaciju o sledećoj instrukciji koju proces treba da izvrši.
- **Sadržaj registara procesora** - vrednosti koje se nalaze u registrima kako bi proces posle gubitka prava da koristi procesor mogao da nastavi sa radom.
- **Prioritet procesa** - informacija o važnosti procesa
- **Adresa gde se nalazi proces** - pokazivači na adrese u memoriji gde se nalaze segmenti procesa.
- **Adrese zauzetih resursa** - informacije o tome na kojim lokacijama da se traže potrebni podaci.

OS i Kontrolni blok realiziju sledeće zadatke:

- Kreiranje KB za nov proces
- Uništavanje KB koji se završio
- Menjanje stanja procesa, kao i prioriteta
- Izbor procesa za izvršvanje.

Prebacivanje kontreksta je 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 postpuka se naziva **dispečer**.

2.4 Niti

Nit je 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, programski brojač i stek, kao i TID. Rad sa više niti podrazumeva mogućost OS-a da podrži konkurentno izvršavanje više niti.

Kontrolna nit je inicijalna nit svakog procesa. Ima zadatak da obavlja potrebne inicijalizacije i kreira ostale niti po potrebi.

Prednost niti je značajna ušteda memorijskog prostora i vremena, zauzimaju manje prostora od nezavisnog procesa. Pružaju mogućnost aplikacijama da nastave rad u situacijama kada bi nezavistan proces privremeno zaustavio ostale operacije.

Preslikavanje niti je 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 procesima, dok se nitima upravlja iz korisničkog režim. Koja nit se izvršava se odlučuje na korisničkom nivou, dok jezgro upravlja nitima na procesorksom nivou. Glavna mana je da ako blokira jedna nit, blokiran je ceo proces.
2. **Jedna u jednu** - 1:1 preslikavanje. Upravljenje nitima se prepušta jezgru. Niti se kreiraju sporije i manje ih je, alli zato jezgro pouzdanije radi sa njima. Ovime se obezbeđuje konkurentno izvršavanje niti, ali se ograničava broj niti jezgra samim tim i u korisničkom delu.
3. **Više u više** - hibrid prethodna dva pristup. Broj niti, kao i način raspodele, se određuje na korisničkom nivou, dok se upravljanje nitima prepušta jezgru. Najkompleksniji je ali i najkvalitetniji.

2.5 Redovi procesa

Redovi se formiraju 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 u RAM. Izbor procesa zavisi od toga da li će više vremena koristiti procesor ili provoditi u radu sa O/I uređajima.
3. **Redom Čekanja** - procesi u stanju čekanja.

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

2.6 Planeri

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

- **Dugoročni** - iz reda poslova biraju procese koji će aktivno da se uključe u sistem i počnu sa izvršavanjem. Sporiji su, jer su ređe potrebni.
- **Kratkoročni** - biraju koji će se od spremnih procesa izvršavati i koliko vremena imaju. Kod njih je brzina rada važnija od optimalnosti odluke.

Swapping je proces suspendovanja procesa i kasnijeg vraćanja u memoriju. Posebno se implementira srednjoročni planer.

2.7 Višeprocesorski sistemi

Višeprocesorski sistemi dozvolajvaju veću fleksibilnost kada je rad sa procesima i nitima u pitanju. Procesi se mogu izvršavati paralelno na različitim procesorima, što sistem čini efikasnijim u odnosu na JPS. Kada su u pitanju metode raspoređivanja procesa na različite procesore razlikujemo:

1. **Simetrično multiprocesiranje** - procesori su ravnopravni i procesi se raspoređuju na bilo koji. Svi procesori dele zajedničku memoriju i pristup I/O.
2. **Asimetrično multiprocesiranje** - neki procesori mogu biti zadužene za određene funkcije. Jedan procesor se proglasi glavnim i on upravlja jezgrom OS-a, dok se ostali uključuju na poziv glavnog.

Raspoređivanje procesa se radi preko dva pristupa:

1. **Zajednički red čekanja** - omogućava 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 procesi 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 nalaženje najmanje zauzetog procesora.
2. **Pull migracija** - u red slobodnijeg procesora se prebacuju procesi iz reda zauzetijih. Obično se bira proces visokog prioriteta ili sa malo vremena izvršavanja.

Proces razvija **Afinitet** prema procesoru na kom je počeo izvršavanje. Razlikujemo **slab** i **jak afinitet**. Slab afinitet omogućava ali ne garantuje da će se proces nastaviti sa izvršavanjem na istom procesoru. Analogno za Jak…

2.8-2.9 Algoritmi raspoređivanja procesa

Raspoređivanje procesa znači 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, min vreme obrade, min vreme odziva, min vreme čekanja.**

FCFS algoritam (First come First served) radi tako što 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 algoritam (Shortest process first) radi tako što se favorizuju procesi sa najkraćim vremenom izvršenja. Svakom procesu se pridružuje očekivano vreme za izvršavanje na procesoru dok procesor dobija procese sa najmanjom vrednošću. Ukoliko dva procesa imaju isto očekivano vreme za izvršavanje bira se preko FCFS algoritma. 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 može aproksimirati sa formulom: $T_{n+1} = \alpha T_n+(\alpha+1)t_n$.

Algoritam sa prioritetom radi tako što svakom procesu se pridruži prioritet na osnovu kog se bira proces koji će se izvršiti. Prioriteti se mogu definisati interno i eksterno. Mana je “izglađivanje” procesa, niži prioriteti će duže čekati.

Kružni algoritam radi tako što svaki proces unapred dobije vremeski 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 prebacivanje konteksta.

Redovi u više nivoa predstavljaju pristup zasnovan na ideji da se red spremnih procesa podeli u više redova sa različitim 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.

Redovi u više nivoa sa povratnom vezom predstavljaju modifikaciju prethodnog pristupa. Naime dozvoljeno je kretanje procesa između redova.

2.10 Rad u realnom vremenu

Preskočili smo na predavanjima. :P

3. Konkuretnost i sinhornizacija procesa

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

Primer trke: dva procesa uvećavaju zajedničku promenjivu X za 1. Ako drugi proces prekine prvi, prvi nije stigao da upiše vrednost izračunavanja u X, te krajnji rezultat neće biti jednak onom koji je potrebam.

3.1 Kritična sekcija

Kritična sekcija je deo programa u kom se pristupa zajedničkim podacima.

Rešenje u vidu iskučivanja 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 kritičnoj sekciji i ne želi da uđe u nju ne treba da ometa druge procese da uđu u nju.
- **Uslov konačnog čekanja** - trebalo bi da postoji razumna granica koliko jedan proces može da čeka na ulazak u kritičnu sekciju.

Algoritam za **Zaštitu kritične sekcije**	

P1:
WHILE(može == 0)
//aktivno čekanje
ENDWHILE
može = 0;
Kritična sekcija
može =1;
	P2:
	WHILE(može == 0)
	//aktivno čekanje
	ENDWHILE
	može = 0;
	Kritična sekcija
	može =1;

Pitersonov algoritam radi tako što proces “džentlmenski” ustupa prednost drugom procesu kada najavi da hoće u kritičnu sekciju. Ova prednost važi samo ako oba procesa žele da uđu u kritičnu sekciju. U suprotnom, proces odmah ulazi u kritičnu sekciju.

Pitersonov algoritam	

P1:
želi_1 = 1;
WHILE(želi_2 == 1)
//aktivno čekanje
ENDWHILE
Kritična sekcija
želi_1 = 0;
	P2:
	želi_2 = 1;
	WHILE(želi_1 == 1)
	//aktivno čekanje
	ENDWHILE
	Kritična sekcija
	želi_2 = 0;

Lamportov algoritam je uopštenje Pitersenovog za n procesa. Zasnovan je na ideji redosleda usluživanja mušterija u pekari. Radi za proizvoljan broj procesa i zadovoljava sve uslove.

Harderska rešenja su ideje da se naprave mašinske instrukcije koje su u stanku da urade dve atomske operacije. Najčešće se koriste:

- **TAS(TEST AND SET)** - operiše kao A=TAS(B), pri čemu je A=B, B=1.
- **FAA(FETCH AND ADD)** - operiše kao FAA(A,B), pri čemu je A=B, B=A+B.
- **SWAP** - OPERIŠE (A,B), A=B, B=A.

TAS ALGORITAM	**SWAP ALGORITAM**

Proces i:
ne_može = 1;
WHILE(ne_može == 1)
ne_može =
TAS(zauzeto);
ENDWHILE
Kritična sekcija
zauzeto = 0;
	Proces i:
	ključ = 1;
	WHILE(ključ != 0)
	SWAP(ključ,brava);
	ENDWHILE
	Kritična sekcija
	brava = 0;

• Rešenje podrazumeva da TAS postoji kao operacija. • Može se primeniti na proizvoljan broj procesa ali ne garantuje za čekanje.	• Brava se inicijalizuje na 0 kao globalna promenjiva.

3.3 Rešenja za zaštitu kritične sekcije bez aktivnog čekanja

Ove metode se zasnivaju na zaustavljanju procesa bez aktivnog čekanja i njihovo “buđenje” u odgovarajućem trenutku. Bolje su od pristupa sa aktivnim čekanjem, ali i komplikovanije za implementaciju.

Semafori su apstraktni tip podataka, tj. Struktura koja može da blokira proces na neko vreme i propusti ga kada se 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.

P(s)	**V(s)**

S.vrednost = S.vrednost - 1;
IF(S.vrednost < 0)
dodati trenutni proces u S.lista;
blokirati proces;
ENDIF
	S.vrednost = S.vrednost + 1;
	IF(S.vrednost <= 0)
	Izabrati proces iz S.lista;
	odblokirati taj proces;
	ENDIF

P(c) Binarni	**V(c) Binarni**

P(S1);
c--;
IF(c<0)
V(S1);
P(S2);
ENDIF
V(S1);
	P(S1);
	c++;
	IF(c<=0)
	V(S2);
	ELSE
	V(S1);
	ENDIF

Kritični regioni su implementacija zaštite pristupa kritičkoj sekciji na višem programskom jeziku. Potrebno je definisati koju promenjivu deli više procesa i oznakom region obezbediti ekskluzivni pristup toj promenjivoj u okviru niza naredbi koje se nalaze u okviru tog regiona. Slaba tačka im je sinhronizacija procesa. Napredniji koncept podrazumeva uslovne kritične regione, koji se aktiviraju na osnovu logičkih uslova.

Monitori su najviši nivo apstrakcije kada je zaštita kritične sekcije u pitanju. Predstavljaju konstrukcije slične klasama, koje mogu da sadrže procedure i promenjive. Glavna odlika je da u jednom trenutku može da bude aktivan samo jedan proces. Za potrebe sinhronizacije su uvedene specijalne uslovne promenjive. Nad ovakvim promenjivama definisane su **wait** i **signal**. Korišćenjem ovih promenjivih i operacija proces se može blokirati operacijom wait i kasnije odblokirati signalom signal iz drugog procesa. Operacije nisu ekvivalentne brojačkim semaforima jer čekanje procesa mora da nastupi pre slanja signala. Pristup kritičkoj sekciji se uz monitore vrlo

lako rešava. Implementiraju se na nivou programskog jezika i koriste se na komplikovaniji način. Prednost monitora u odnosu na kritične regione leži u mehanizmima wait i signal koji olakšavaju sinhronizaciju.

4. Zaglavljivanje

Zaglavljivanje je situacija u kojoj dva ili više međusobno vezanih procesa blokirani čekaju na resurse koje nikada neće dobiti. Trajno je i za posledicu često ima zaglavljivanje čitavog sistema. Ako sistem sadrži procese koji us zaglavljeni smatra se da je i on zaglavljen.

Živo blokiranje je situacija u kojoj pocesi nisu blokirani, ali nemaju napretka u izvršavanju.

Izgladnjivanje znači da u sistemu generalno postoji napredak, ali neki procesi dosta dugo ne napreduju.

Da bi do zaglavljivanja došlo neophodno je:

- **Uzajamno isključivanje** - podrazumeva da u jednom trenutku tačno jedan proces može da koristi resurs, dok ostali čekaju.
- **Čekanje i držanje** - dok drži resurse sa kojima radi, proces može da zahteva nove resurse i čeka trenutno nedostupne.
- **Nemogućnost prekidanja** - OS nema pravo da oduzme resurse pocesu kom ih je dodelio već to može da uradi samo sam proces.
- **Kružno čekanje** - može da postoji lanac procesa koji čekaju jedni na druge, čineći krug.

Do zaglavljivanja 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 zaglavljivanja.
2. **Izbegavanje** - dinamičke mere koje se preslikavaju kako bi se sistem vodio kroz stanja koja obezbeđuju da ne dođe do zaglavljivanja. Liberalnija od 1.

3. **Detekcija i oporavljanje** - do zaglavljivanja može doći, ali postoje mehanizmi za detektovanje zaglavljivana, tretman zaglavljivanja procesa i eventualno spašavanje dela rezultata, ukoliko je moguće.
4. **Nepreduzimanje bilo čega** - mera prihvatljiva na saistemima gde zaglavljivanje retko dolazi ili u situacijama nevažnih poslova. Najjednostavnije rešenje, podrazumeva ponovno pokretanje procesa ili celog sistema u nadi da će se zaglavljivanje ne pojavi opet.

4.1 Sprečavanje

Sprečavanja su statičke, sistemske mere koje obično nisu algoritamski zahtevne, a za cilj imaju da u startu definisanjem pravila eliminišu bar jedan od uslova za zaglavljivanje. Troškovi ovih algoritama su mali, ali su zatom ograničenja velika.

Prevencija čekanja i držanja je uvođenje ograničenja koje onemogućava da proces drži neke resurse dok traži nove. Obično se koriste dva pristupa za rešavanje ovakvih problema:

- Pre početka izvršavanja, proces zahteva da mu se dodele svi potrebni resursi i da u izvršavanje krene tek kada ih sve dobije. Loša strana je to što za vreme svog izvršavanja proces drži sve potrebne resurse, a i često na početku nije poznato koji sve resursi će biti potrebni procesu.
- Proces traži resurse u trenucima kada su mu potrebni, ali da pre svakog uzimanja novih resursa ima obavezu da one koje drži vrati OS-u kako bi OS doneo odluku da li će ih ponovo dodeliti njemu ili ih ustupiti nekom drugom procesu. Na ovaj način proces ne drži nijedan resurs u momentu kada traži nove resurse. Mana je moguće “izgladnjivanje” procesa koji ne mogu da dođu na red jer su u svakom trenutku neki od resursa nedostupni.

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

Za **Prevenciju kružnog čekanja** koriste se rešenja koja podrazumevaju enumeraciju. Ideja je da se svaki resurs dodeli broj, a da proces koji drži određene resurse može da zahteva samo one resurse kojima je dodeljen veći broj od brojeva njegovih resursa.

$$F(Ri) = k \Rightarrow F(Rj) > F(Ri)$$

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

4.2 Mere izbegavanja

Izbegavanje je dinamička mera koja se preduzima u određenim momentima u zavisnosti od stanja na sistemu. Troškovi algoritma su veći nego kod sprečavanja, ali se povećava efikasnost sistema.

Bankarev algoritam je predložen 1965. Dijkstra. Ime je dobio po principu davanja zajmova u bankama. Podrazumeva da je pre početja izvršavanja svakog od procesa poznat maksimalan broj resursa svakog tipa koji će mu biti potreban. Ideja je da ase 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 zaglavljvianja. Kada sistem dobije zahtev za resurse od procesa, on pokreće Bankarev algoritam kako bi doneo odluku da li će se dodeliti resursi procesu, odložiti ili odbiti zahtev. Ovaj algoritam je teorijski dobar, ali ima ozbiljna ograničenja u praksi.

4.3 Detekcija zaglavljivanja

Detekcija zaglavljivanja dopušta da dođe do zaglavljivanja, a onda primenjuje mere da se ta situacija otkloni. Otkrivanje da je do zaglavljivanja došlo može biti:

1. **Aktivno** - algoritam za detektovanje zaglavljivanja se pokreće i proverava da li je došlo do zaglavljivanja, u zavisnosti od frekvencije zaglavljivanja.
2. **Pasivno** - algoritam 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 zaglavljivanja se tada svodi na pronalaženje ciklusa u grafu. Ako resursi imaju više instanci, ciklus ne mora da znači da je došlo do zaglavljivanja.

Oporavak od zaglavljivanja je kada se utvrdi da je došlo do zaglavljivanja, sistem bi trebalo da interveniše i pomogne da se zaobiđe situacija. Algoritam za oporavak, koji se prvi nameće, podrazumeva da se uklone svi blokirani procesi, što je neefikasno rešenje jer se tako gube svi privremeni rezultati. Zahtevniji algoritam postepeno prekida servise 1 po 1 dok se ne dglavi. Pristup koji podrazumeva oduzimanje resursa se zasniva na ideji da se određenim procesima oduzmu resursi kako bi se uz pomoć tih resursa otklonilo zaglavljivanje. Treba ostaviti te resurse u pognodnom stanju kako bi kasnije mogli nastaviti sa radom.

5. Upravljanje memorijom

Memorija je osnovni resurs OS-a. Svaka procesorska instrukcija se mora naći u memoriji, a za izvršavanje većine instrukcija potrebni su podaci iz memorije. Pošto je memorija ograničen resurs, njeno efikasno korišćenje je jedan od osnovnih zadataka OS-a. Ovim problemom se obično bavi poseban modul OS-a koji se zove ***Menadžer memorije.*** Kad je reč o čuvanju podataka dele se na memorije koje privremeno čuvaju podatke i memorije koje imaju mogućnost trajnog čuvanja podataka. Na osnovu brzine pristupa memorija se deli u četiri grupe:

1) **Registri** - memorijske ćelije ugrađene u procesor i u njima se nalaze podaci koje procesor trenutno obrađuje. Registri privremeno čuvaju podatke.

2) **Keš memorija** - veoma vrza memorija koja čuva podatke koji se često koriste, a može predstavljati i bafer između primarne memorije i procesora. Obično se nalazi u samom procesoru, ali se delovi mogu naći i van njega.
3) **Primarna memorija** - sadrži instrukcije i podatke kojima procesor trenutno operiše, naziva se radnom memorijom. Po brzini je izmedju Keš i Sekundarne memorije. Postoje:
    a) **RAM** - vreme pristupa ne zavisi od lkacije i sadržaja podatka. Privremeno čuva podatke i bez koje računar ne može da funkcioniše.
    b) **ROM** - čuva podatke kada je računar isključen. Moguće je samo čitanje podataka iz ROM-a. Sadrži kritične programe za pokretanje operativnog sistema.
    c) **EEPROM** - takođe čuva podatke kada je računar isključen, ali za razliku od ROM-a podaci se mogu brisati i ponovo upisivati.
4) **Sekundarna memorija** - ne gubi sadržaj nakon prestanka rada računara pa se zbog toga koristi za trajno čuvanje podataka. Da bi procesor pristupio podacima, najpre je potrebno da se oni učitaju u radnu memoriju. Nije neophodna za rad ralunara i vreme pristupa zavisi od lokacije.

Memorija se sastoji od niza memorijskih ćelija koje mogu da sačuvaju najmanju količinu podataka - **bit**. Bitovi se organizuju u **bajtove** - grupe od po 8 bitova. Na većini računara bajt je najmanja jedinica.

Memorijska reč određuje količinu podataka koje procesor može da obradi u jednom trenutku. Njena veličina je određena arhitekturom konkretnom računara. Najzastupljeniji su računari sa 64-bitnom arhitekturom. Kada se govori o veličine memorijske reči, misli se i na adresibilni prostor računara.

Radna memorija se može predstaviti kao matrica bitova. Redni broj bajta zapisan u binarnom isstemu predstavlja adresu tog bajta. Na osnovu te adrese procesor može da pristupi bajtu. U trenutku pisanja

programa nije poznato u kom delu memorije će program biti smešten. Zato se koriste **simboličke adrese**, koje odgovaraju imenima promenjivih u programskim jezicima. Kada se program učita u memoriju on mora da radi sa realnim **fizičkim adresama**. Prevođenje simboličkih u fizičke adrese se naziva **povezivanje adresa** i može se obaviti u različitim trenucima. Kompilator generiše relativne adrese u odnosu na početak dela memorije koja se dodeljuje procesu. Ove adrese se nazivaju i **relokatibilne adrese**. Prilikom učitavanja izvršnog programa, loader preslikava relativne adrese u fizičke. Procesor uzima instrukcije i podatke iz memorije ili smešta podatke u memoriju. U navedenim akcijama procesor ne manipuliše fizičkim adresama, već logičkim koje sam generiše. Skup svih logičkih adresa naziva se logički ili **virtuelni adresni prostor**. Svakoj logičkoj adresi odgovara fizička adresa. Skup svih fizičkih adresa koje odgovaraju adresama logičkog adresnog prostora naziva se **fizički adresni prostor.** Ukoliko se povezivanje adresa obavlja za vreme prevođenja i punjenja programa, logički i fizički adresni prostori se podudaraju. To nije slučaj kada se povezivanje adresa obavlja za vreme izvršavanja.

MMU je 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 memorija unutar računarskog sistema moraju biti korišćene tako da se svaki proces izvršava najefikasnije moguće.

5.1 Monoprogramiranje

Monoprogramiranje znači da se u memoriji u jednom trenutku izvršava samo jedan korisnički proces. Kod ovog pristupa postoje različite organizacije memorije:

1. **OS na niže adrese** - procesor koristi više adrese.
2. **OS na vrhu memorije** - proces koristi niže adrese.
3. **OS na dnu** - drajveri na vrhu, korisniku ostatak.

Učitavanje OS-a se izvršava prilikom uključivanja sistema, kada loader 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 adrese 0, dok korisnik ima pravo na sve adrese posle zaštitnog registra. Vrednost zaštitne 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 omogućaju da se više procesa istovremena rade u memoriji, sa **pseudoparalelnim** izvršavanjem. Kada neki proces čeka I/O operaciju, drugi mogu da se izvršavaju mimo njega.

Prebacivanje je ideja da se blokiran proces privremeno prebaci u sekundarnu memoriju, a da se u memoriju ulita 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 sekundarne memorije. Teži se da vreme izvršavanja procesa bude veće od vremena prebacivanja kako bi iskorišćenost procesora bila što bolja. Napredniji pristup podrazumeva da se efikasnost sistema poboljša preklapanjem prebacivanja i izvršavanja procesa.

Particionisanje je ideja da se memorija podeli na n particija, koje predstavljaju neprekidne delove memorije. U svaku particiju se može

smestiti po jedan proces. Ako su sve particije zauzete, formira se red spremnih procesa, iz kog se izvlače procesi nakon oslobadjanja particija. Particije se najčešće štite uz pomoć dva regista za svaku particiju. Postoje dva pristupa particionisanju ***statički*** i ***dinamički.***

Statičke particije znači da se memorija statički deli na fiksni broj particija. Broj particija zavisi od raznih faktora, kao što su veličina procesa i radne memorije, brzine procesa… 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 poslova u koji se smeštaju procesi koji odgovara veličini particije.
2. Svi poslovi se smeštaju u jedan red, dok planer donosi odluku koji je sledeći proces.

Dinamičke particije znači da veličine particija nisu fiksirane. Pri pokretanju računaram slobodan deo memorije obrazuje jednu slobodnu particiju. Procesi se ulitavaju tako da zauzmu deo slobodne particije koji mu je potreban, dok ostatak ostaje slobodan. Tada svaki sledeći proces traži najveći slobodan deo. OS vodi evidenciju o slobodnim i zauzetim delovima memorije korišćenjem bit-mapa ili povezanih listi.

Fragmentacija je neželjena pojava kod particionisanja. Nađe se kada u sistemu postoje slobodni delovi memorije, ali nisu dovoljno veliki za korišćenje. Može biti:

1. **Interna** - odnosi se na prostor koji ostaje neiskorišćen kada se proces smesti u particiju većih memorijskij kapaciteta od potrebnog.
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 memorije tako da sabije sve procese.

Algoritmi za dodelu memorije:

- **First fit** - smešta proces u prvu slobodnu zonu dovoljno veliku za njega. Pretraga kreće od početka memorije. Često dovodi do fragmentacije.
- **Next fit** - modifikacija prvog, podrazumeva da će sledeća pretraga da počne na mestu gde je prethodna završila.
- **Random** - od svih slobodnih prostora slučajno se izabere gde će da se smesti proces.
- **Best fit** - proces smešta u najmanji slobodan deo potreban za proces.
- **Worst fit** - suprotan od best fit.

5.3 Straničenje

Straničenje dozvoljava da dodeljena memorija ne bude uzastopna. Ideja je da se radna memorija podeli na okvire fiksirane veličine, a logička podeli na stranice istih veličina kao i okviri. Kod ovakve organizacije 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 određovanje procesa odgovoran je **planer poslova**. Zaštita memorije se vrši uz pomoć zaštitnih bitova.

Tabela stranica se čuva u skupu registara. Korišćenje registara je pogodno samo u situacijama kada je tabela relativno male veličine. U situacijama kada je tabela stranica veća, ona se čuva u emoriji, a na njenu lokacij uu memoriji pokazuje *Bazni Registar Tabele Stranica.*

Asocijativna memorija se koristi za implementaciju efikasne tabele stranica. Ona je u stvari keš memorija koja je posebno dizajnirana za potrebe straničenja. Malo je sporija od registara ali većeg je kapaciteta. Podaci se čuvaju u parovima ključ i vrednost a pretraga se vrši nad svim ključevima.

5.4 Segmentacija

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

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

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

6. Virtuelna memorija

Virtuelna memorija je način upravljanja memorijom koji omogućava da se procesu na raspolaganje stavi memorija drugačije veličine od one koja fizički postoji u sistemu. OS je zadužen da omogući preslikavanje virtuelne memorije u fizičku. Najčešće se vezuje za omogućavanje izvršavanja procesa koji nije u potpunosti u radnoj memoriji.

Prekrivači su ideja da se omogući izvršavanje programa koji nisu kompletno učitani u memoriju. Osnovni zadatak je da se indentifikuju delovi programa koji su relativno nezavisni, koji se odvajaju od glavnog dela programa i smeštaju u sekundarnu memoriju. Prilikom alociranja memorije se rezerviše prostor za prekrivač najveće veličine, u kom se zatim smenjuju svi pokrivači u zavisnosti od potrebe. Najveći nedostatak ovakvog pristupa je to što operaciju deljenja programa na delove precizno i dovoljno dobro može da uradi jedino programer, odnosno ona se ne može automatizovati.

Dinamičko punjenje je ideja da se funkcije i procerue ne smeštaju u memoriju sve do trenutka dok ne budu potrebne, tj. Pozvane. Nalaze

se u sekundarnoj memoriji, i nakon poziva procesoriveru da li je već učitano ili ne. Prednost je što se funkcije koje se ne koriste nikad neće ni ulitati u memoriju. Jednostavno se implementira.

Straničenje na zahtev je najefikasniji i najpopularniji vid implementacije virtuelne memorije. Deli program na stranice. Pri pokretanju programa u radnu memoriju 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 sekundarne memorije u radnu. Veličina logičkog adresnog prostora nije ograničena fizičkom memorijem, lime se obezbeđuje velika virtuelna memorija. Implementira se uz pomoć tabele stranica, koja dodatno sadrži bit validnosti.

Page fault je pokušaj pristupa stranici koja nije u radnoj memoriji. Ako se stranica promaši, prvo se proverava da li stanica 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 sekundarne u radnu memoriju.
3. **Modifikacija bita validnosti** u tabeli stranica.

Čisto straničenje je proces koji kreće u učitavanje bez ijedne učitane strane. Promašaji će seređati u startu sve dok se ne formira radni skup stranica.

Zamena stranice je ideja 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.

6.4 Algoritmi za izbacivanje stranice

Slučajno izbacivanje radi tako što se slučajno bira stranica za izbacivanje. Jedno od najgorih rešenja. Često dovodi do izbacivanja nekih veoma korišćenih stranica, čime se doprinosi velikom broju promašaja stranica.

Beladijev optimalan algoritam je 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 algoritam radi tako što se iz memorije izbacuju stranice u redosledu u kom su učitane. Mana je što se prvo učitavaju stranice od važnosti, koje će se ovim pristupom prve izbaciti.

Algoritam druge šanse je modifikacija FIFA**,** podrazumeva postojanje jednog bita po stranici koji se naziva *bit referisanosti.* Bit referisanosti se postavi na 1, kada dođe na red za izbacivanje postavlja se na 0, resetuje se vreme izvršavanja i gura se na kraj liste. Problem nastsaje kada su svi bitovi referisanosti postavljeni na 1.

Algoritam sata je modifikacije druge šanse, u kom su sve stranice povezane u kružnu listu, kako se ne bi gubilo vreme za premeštanjem strana na kraj liste.

LRU radi tako što se iz memorije izbacuje stranica koja je korišćena najdalje u prošlosti u odnosu na ostale. Podrazumeva da su skoro korišćene stranice važne i da će tako ostati i u budućnosti. Najveća mana je što se ne može predvideti sledeća stranica.

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

LFU i MFU iliti ređe/češće korišćena stranica. Mane su:

- **LFU** - loš tretman stranica koje su tek ušle u sistem, ali su veoma potrebne ii će se dosta koristit u budućnosti. Druga mana se javlja u slučajevima kada se pojedine stranice koriste često u određenom vremenskom perioud, a kasnije neće biti potrebe.

- **MFU** - mnogo problema, jedan od njih nastaje kada program tokom celog svog izvršavanja često koristi neku stranicu, pa se tada ta stranica često izbacuje.