

Operativni sistemi

- odgovori na pitanja iz knjige -

Mina Milošević
mi17081@alas.matf.bg.ac.rs

2018/2019

Sadržaj

1. Uvod u operativne sisteme.....	3
2. Upravljanje procesima.....	10
3. Konkurentnost i sinhronizacija procesa.....	18
4. Zaglavljivanje.....	28
5. Upravljanje memorijom.....	36
6. Virtuelna memorija.....	43

1. Uvod u operativne sisteme

1.1 Na kojim principima su zasnovani savremeni racunarski sistemi?

Savremeni racunari se sastoje od 1 ili vise procesora, memorije, veceg broja U\I uredjaja, komunikacione opreme itd. Da bi ovaj hardver imao upotrebnu vrednost, potreban je softver koji ce ga uciniti upotrebljivim za korisnike razlicitih profila.

1.2 Po cemu se razlikuju racunarski sistemi danas od ranijih generacija?

Najveca razlika danasnijih sistema u odnosu na prve operativne sisteme je u upravljanju mrezama. Racunarske mreze nisu postojale u danasnjem obliku u trenutku zacetka prvih generacija OS-a tako da rad sa njima nije bio jedan od prioriteta.

1.3 Sta je hardver racunarskog sistema?

Hardver (tehnicki sistem racunara) cine uredjaji racunarskog sistema, najpre procesor i radna memorije koji predstavljaju srce tehnickog sistema racunara, a zatim i U\I uredjaji.

1.4 Sta cini softver racunarskog sistema?

Softver (programska sistem racunara) cine aplikativni i sistemski softver.

1.5 Sta je aplikativni softver?

Aplikativni softver je najvisi sloj u hijerarhiji i cine ga programi koji sluza za obavljanje specificnih zadataka na racunaru.

1.6 Navesti primere aplikativnih softvera?

Primeri: programi za obradu teksta, programi za reprodukciju multimedijalnih sadrzaja, programi za matematicka izracunavanja itd.

1.7 Sta je sistemski softver?

Sistemski softver predstavlja skup programa koji bi trebalo da pruze sto udobniji interfejs za koriscenje hardvera i da omoguce izvrsavanje aplikativnih programa (predstavlja sponu izmedju hardvera i aplikativnog softvera).

1.8 Koji su osnovni zadaci operativnih sistema?

Sustina operativnih sistema je da obezbede okruzenje u kojem ce korisnici imati mogucnost da sto jednostavnije pokrecu i izvrsavaju programe, a da se hardver koristi sto efikasnije. Takodje, treba da zastiti hardver od direktnog pristupa korisnika tj. od korisnickih programa.

1.9 Cime bi operativni sistemi trebalo da upravljaju?

Osnivni zadaci OS-a su da omoguce sto efikasniju realizaciju sledecih aktivnosti:

- upravljanje procesima (programima u izvrsavanju) – kreiranje, izvrsavanje, dodeljivanje resursa procesima, sinhronizacija itd.
- upravljanje memorijom – rasporedjivanje procesa u okviru radne memorije
- upravljanje U\I uredjajima – kontrola i transfer podataka izmedju uredjaja i sistema
- upravljanje podacima – cuvanje podataka, vodjenje evidencije o njima, manipulacija itd.
- upravljanje mrezama – umrezavanje i komunikacija izmedju racunara

1.10 Objasniti odnos sistemskog softvera i operativnog sistema?

Operativni sistem je deo sistemskog softvera koji je odgovoran za upravljanje računarskim resursima koji treba da obezbedi sto bolje uslove za koriscenje računara. Predstavlja sloj softvera koji je naslonjen na hardver i trebalo bi da ga ucini upotrebljivim, da pri tome prikrije razlike izmedju razlicitih hardverskih komponenti istog tipa i da korisniku omoguci sto vecu udobnost za rad. Izvrsava se sve vreme, prati i nadgleda funkcionisanje sistema.

1.11 Sta su sistemski pozivi?

Sistemski pozivi su skup funkcija koji predstavlja interfejs ka operativnom sistemu. Usluge koje operativni sistem moze da pruzi aplikativnim programima ostvaruju se uz pomoc sistemskih poziva. Programi uz pomoc sistemskih poziva komuniciraju sa jezgrom i pomocu njega dobijaju mogucnost da izvrse osetljive operacije u sistemu.

1.12 U kojim rezimima mogu da rade procesori racunarskih sistema?

Mogu da rade u bar dva razlicita rezima rada: korisnickom (user mod) i sistemskom (supervisor, kernel mod).

1.13 Objasniti potrebu za promenom rezima u toku izvrsavanja programa?

Kada aplikativni programi izvrse sistemski poziv, parametri sistemskog poziva se postave na predvidjene lokacije u memoriji. Zatim se menja rezim rada u sistemski u kojem su, za razliku od korisnickog rezima, dozvoljene sve operacije koje procesor moze da uradi.

1.14 Sta je jezgro operativnog sistema?

Jezgro (kernel) je deo operativnog sistema u koji su smestene najvaznije funkcije koje obezbedjuju osnovne servise operativnog sistema. Odgovorno je za funkcionisanje sistema i ima zadatku da upravlja hardverskim i softverskim resursima na najnizem nivou.

1.15 U kojem trenutku se jezgro ucitava u radnu memoriju?

Prvi se ucitava u radnu memoriju, pri pokretanju računarskog sistema i ostaje u njoj do zavrsetka rada tj. iskljucivanja sistema.

1.16 Na koji nacin se izvrsavaju sistemski pozivi?

Sistemski pozivi koriste jezgro da bi omogucili razlicite servise operativnog sistema. Svi programi funkcionisu na nivou iznad jezgra u korisnickom rezimu rada. Sistemske aktivnosti, koje se pokrecu sistemskim pozivima, obavljamaju se na nivou jezgra tj. u sistemskom rezimu rada. Jezgro se obicno ucitava u poseban, zasticeni deo memorije.

1.17 Sta se postize izvrsavanjem sto veceg broja aktrivnosti u korisnickom rezimu (umesto u sistemskom)?

Na taj nacin se povecava stabilnost sistema. Eventualne greske pri izvrsavanju programa ili problemi uglavnom ne mogu da ugroze funkcionisanje sistema.

1.18 Koja je uloga korisnickog okruzenja?

Korisnicko okruzenje ima zadatku da olaksa koriscenje ostalih delova operativnog sistema, a i celokupnog računarskog sistema.

1.19 Na koji nacin se mogu podeliti korisnicka okruzenja?

Mogu se podeliti na tekstualna i graficka tj. na linijska i ekranska. Linijska podrazumevaju konzole, terminale, komandne linije itd. Koje omogucavaju da se OS-om upravlja kucanjem linija teksta. Ekranska pruzaju mogucnost da se OS-om upravlja koriscenjem celog ekran-a.

1.20 Koji je najvazniji deo linijskog korisnickog okruzenja?

Najvazniji deo je komandni interpreter. Njegova uloga je da nardbe i podatke koje korisnik unese u tekstualnom obliku prepozna i nalozi operativnom sistemu izvrsavanje odgovarajucih operacija.

1.21 Sta su drajveri?

Drajveri (upravljacki programi) – nadogradjuju se na kontrolere i omogucavaju komunikaciju tj. upravljanje U\I uredjajima.

1.22 Koje uslove bi drajveri trebalo da zadovolje kada su u pitanju razliciti tipovi iste vrste uredjaja?

Programiraju se tako da za razlicite tipove iste vrste uredjaja definisu jedinstven skup dozvoljenih instrukcija. Razliciti uredjaji mogu zahtevati posebne nardbe ili parametre za izvrsavanje, a zadatak drajvera je da apstrahuju ove razlike i naprave uniformni interfejs.

1.23 Kakvi pristupi postoje pri projektovanju operativnih sistema kada je skup funkcija koje ce se nalaziti u jezgru u pitanju?

U odnosu na tip jezgra postoje:

1. monolitni sistemi
2. slojевити sistemi
3. sistemi zasnovani na mikrojezgru
4. hibridni sistemi
5. sistemi zasnovani na egzojezgru

1.24 Objasniti pojам monolitnog jezgra?

Arhitektura operativnog sistema ima monolitno jezgro ako se u jezgru nalaze svi servisi operativnog sistema zajedno sa drajverima integrисани u jedan program. Svi delovi se pokrecu u istom trenutku, izvrsavaju u sistemskom rezimu i u istom delu memorije. Jezgro se u memoriju ucitava u celosti kao jedan izvrsni program. Funkcije jezgra mogu jedna drugu pozivati bez ogranicenja. Predstavljaju jedinu apstrakciju nad hardverom tako da se aplikativnim programima na raspolaganje stavlaju sistemski pozivi kroz koje su implementirane usluge OS-a. Mana je losa otpornost na greske.

Primeri: MS-DOS, BSD, AIX, Windows 98, GNU/Linux

1.25 Sta je mikrojezgro?

Mikrojezgro – minimalno jezgro u kojem se nalaze samo najosnovnije funkcije. Deo funkcija koje bi u prethodnim slucajevima bile deo jezgra se izmestaju u korisnicki prostor i to u zasebne prostore u memoriji tako da sa bezbednjeg nivoa pristupaju jezgru. Usluge koje obavljaju slicne zadatke u okviru OS-a se grupisu u procese koji se nazivaju serverski procesi (serveri). Drajveri se cesto ne nalaze u mikrojezgru, vec pripadaju odgovarajucim serverima.

1.26 Koje osobine odlikuju arhitekturu sistema zasnovanih na mikrojezgru?

Arhitekturu mikrojezgra odlikuje veci stepen sigurnosti u odnosu na monolitne sisteme, ali su oni obicno sporiji. Promene memorijskog prostora dovode do kasnjenja i manje propusnosti u poređenju sa sistemima sa monolitnim jezgrom. Cesto prosledjivanje poruka izmedju mikrojezgra i servera takodje dovodi do kasnjenja.

Primeri: Mach, Minix, QNX, L4

1.27 Sta je hibridna arhitektura jezgra?

Hibridna jezgra predstavljaju kompromis izmedju monolitne i arhitekture koja se zasniva na mikrojezgru. Bitne i funkcije koje se cesto izvrsavaju se spustaju u jezgro kako bi se povecala brzina i efikasnost, ali se dobar deo funkcija zadrzava u nivoima iznad jezgra.

Primeri: Apple Mac OS X, Microsoft Windows NT 3.1, NT 3.5, NT 3.51, NT 4.0, 2000, XP, Vista, 7

1.28 Koji su glavni zadaci prilikom projektovanja hibridnih jezgara?

Najvažnije je precizno odrediti koje funkcije treba spustiti u jezgro, a koje ostaviti van.

1.29 Koje su odlike egzojezgra?

Koncept egzojezgra predstavlja kompromis izmedju dva suprotna pristupa. Ideja je da jezgro obezbedi osnovne resurse i da aplikacijama prepusti rad sa njima. Ovo se postize prenestanjem apstrakcije iznad hardvera u posebne biblioteke koje obezbedjuju minimalne apstrakcije uredjaja. Moze da doprinese znatnom ubrzavanju i poboljsanju performansi. Ali dodatna fleksibilnost za korisnicke aplikacije moze da dovede do smanjenja konzistentnosti i neurednosti koda.

Primeri: XOK, ExOS

1.30 Na kojoj tehnologiji su bili zasnovani racunari prve generacije?

Bili su zasnovani na elektronskim (vakuumskim) cevima. Ulazne tehnologije su se zasnavale na busenim karticama i magnetnim trakama.

1.31 Koja je bila uloga operatera kod prvih racunarskih sistema?

Operateri su bili zaduzeni za upravljanje racunarskim sistemima. To je bila osoba koja opsluzuje racunarski sistem i njegov zadatak je bio da pripremi sve sto je potrebno da se zadatak moze obaviti. Najveci deo vremena se trosio na poslove operatera i U/I operacije.

1.32 Koji sistem se smatra prvim operativnim sistemom?

Sistem GM-NAA I/O koji se isprogramiran za racunar IBM 704, 1956. godine.

1.33 Koji je novi koncept donela generacija racunara zasnovana na tranzistorima?

Novi koncept je paketna obrada podataka (Batch processing).

1.34 Koje su glavne odlike paketne obrade podataka?

Ovaj nacin izvrsavanja programa podrazumeva da se oni nadovezuju jedan na drugi tj. da se blokovi kartica korisnickih programa redjaju jedan za drugim. Operativni sistem specijalizovanog racunara je vodio racuna o punjenju i praznjenju memorije. Ovakav pristup je omogucavao da se u trenutku izvrsavanja jednog programa ucitava sledeci koji je na redu.

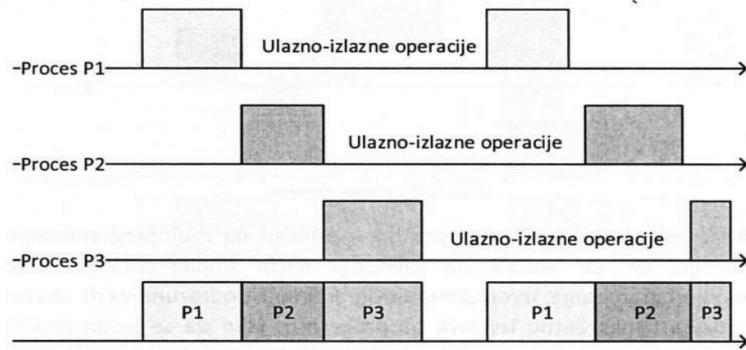
1.35 Koje su posledice uvodjenja integrisanih kola kada su komponenti racunara u pitanju?

Dolazi do velikog nesklada kada su brzine u pitanju, najpre izmedju procesora i periferijskih uredjaja. Vremenske jedinice kojom su se merile brzine rada procesora bila je ns, diskova ms, a stampaca s. Procesor je bio nedovoljno iskoriscen jer je cesto morao da ceka na sporije komponente.

1.36 Objasniti koncept multiprogramiranja?

Osnovna ideja multiprogramiranja je bila da se u radnu memoriju smesti vise programa (procesa) kako bi se poboljsala iskoriscenost procesora. Memorija bi se podelila na delove u koje bi se ucitavali programi.

1.37 Ilustrovati multiprogramiranje primerom.



1.38 Kako izgleda životni vek procesa u sistemima koji podrzavaju multiprogramiranje?

Programi koji imaju potrebu da rade na procesoru bi se smenjivali na njemu tako da on uvek bude zaposlen, a dok se jedan program izvrsava na procesoru drugi imaju mogucnost da izvrsavaju U\I operacije.

1.39 Koji su osnovni ciljevi multiprogramiranja?

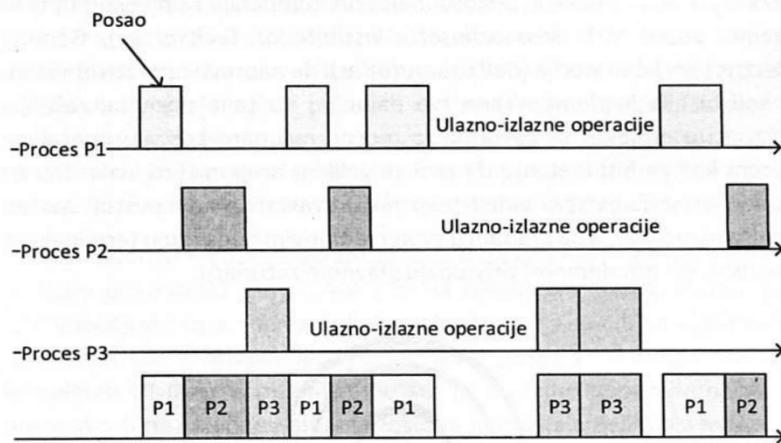
Glavni cilj je maksimalno povecanje iskoriscenosti sistema, ali je pozeljno da vreme izvrsavanja programa bude sto manje.

1.40 Koja je osnovna ideja kad je deljenje vremena u pitanju?

Deljenje vremena je koncept koji se zasniva na deljenju racunara izmedju vise korisnika. Procesor se deli „vremenski“ tako sto bi svaki korisnik dobio odredjeno vreme u kojem bi imao procesor na raspolaganju. Po isteku deljenog vremena procesor bi dobio sledeci korisnik.

1.41 Objasniti pojam multitaskinga?

Multitasking se odnosi na efikasniji nacin implementacije ideje multiprogramiranja.



1.42 U cemu je glavna razlika izmedju multitaskinga i multiprogramiranja?

Za razliku od multiprogramiranja koje podrazumeva da se vise procesa konkurentno izvrsava na procesoru, tako da se jedan proces izvrsava sve dok na red ne dodju U\I operacije; multitasking podrazumeva da je jedinica izvrsavanja na procesoru posao (task) koji ne mora nuzno da obuhvata izvrsavanje procesa na procesoru izmedju dve U\I operacije.

1.43 Sta se podrazumeva pod multiprocesiranjem?

Multiprocesiranje se obично odnosi na postojanje više procesora u računarskom sistemu koji hardverski omogućavaju istovremeno izvršavanje više poslova procesa.

1.44 Koji se operativni sistemi smatraju najznačajnijim u istoriji?

Multics, UNIX, Linux, DOS i Windows, Apple OS, Android

1.45 Sta je glavni doprinos operativnog sistema Multics kada je razvoj operativnih sistema u pitanju?

Zeeli su da kreiraju operativni sistem koji će biti u stanju da radi sa velikim brojem terminala. Budućnost računarstva su videli tako da u svakom gradu postoji mogući centralni računar, a da građani u svojim domovima poseduju terminale uz pomoć kojih (modemom) pristupaju glavnom računaru.

1.46 Koji su glavni razlozi za neuspeh Multics-a?

Trosio je puno procesorskog vremena na donesenje odluka, a malo je ostavljao korisnicima.

1.47 Kako je nastao operativni sistem UNIX?

Programeri iz Bel laboratorija, Ken Thompson i Denis Rici sa svojim kolegama, su odlucili da napisu mali OS za mali računar PDP-7. Ovaj sistem je bio uprscena varijanta Multics-a i iz njega se razvio UNIX.

1.48 Ko je i iz kojih razloga osmislio programski jezik C?

Denis Rici je osmislio programski jezik C kako bi na njemu bio isprogramiran deo operativnog sistema. Kritični delovi OS-a su bili isprogramirani na asembleru, a ostatak u C-u.

1.49 Koji su glavni razlozi za veliku popularnost UNIX-a?

Kompletan kod se ustupio univerzitetima i drugim kompanijama kako bi se dalje razvijao. Najpoznatije verzije UNIX-a su implementirane na Univerzitetu Berkli (BSD). U ovim verzijama UNIX-a prvi put je implementirana podrška za umrezavanje.

1.50 Koje su najpoznatije verzije UNIX-a?

UNIX, BSD, Unix System, SunOS, Unix System V Release, IRIX, OSF, Solaris, BSD/OS, Novell UnixWare, FreeBSD, NetBSD, OpenBSD, SCO UnixWare, True64 UNIX,

1.51 Iz kojih razloga je znacajan rad Ricarda M. Stolmana?

On je 1983. godine pokrenuo inicijativu koja je trebalo da dovede do stvaranja slobodnog OS-a na osnovama UNIX-a. Ovaj projekat je nazvan GNU.

1.52 Sta se podrazumeva pod pojmom Linux?

Pod pojmom Linux se podrazumeva jezgro operativnog sistema.

1.53 Ko je i kada kreirao Linux?

Linux je kreirao Linus Torvalds 1991. godine.

1.54 Ko je i u koje svrhe napravio operativni sistem Minix?

Razvio ga je Endru Tanenbaum za potrebe svojih kurseva.

1.55 Prodiskutovati odnos GNU-a i Linux-a.

Linux je naziv jezgra, a GNU naziv alata. Njihovim spajanjem je kompletiran operativni sistem GNU/Linux.

1.56 Koje su glavne prednosti GNU/Linux-a u odnosu na ostale operativne sisteme?

Sve verzije GNU/Linux-a odlikuje stabilnost, brzina i visok stepen bezbednosti. Jezgro je softver otvorenog koda sa licencom GNU GPL, što znači da su korisnici slobodni da preuzimaju izvorni kod i prave izmene.

1.57 Koje su najpoznatije distribucije GNU/Linux operativnog sistema?

Danas postoji veliki broj razlicitih distribucija GNU/Linux sistema a neke od najpoznatijih su: Ubuntu, Slackware, Debian, Fedora, SuSE i druge.

1.58 Prodiskutovati razloge za svojevremenu veliku popularnost operativnog sistema DOS.

Dobar marketing i pravi trenutak za pojavljivanje operativnih sistema kompanije Microsoft bio je presudan za veliku popularnost.

1.59 Na cemu su bili zasnovani WINDOWS operativni sistemi?

Windows je bio graficka nadogradnja na DOS. Bio je zasnovan na racunarskim "prozorima".

1.60 Sta je glavna odlika operativnih sistema kompanije Apple?

Najpre su mogli da se koriste i razvijali su se samo za Apple-ove uređaje.

1.61 Na kakvim racunarima su prvi operativni sistemi kompanije Apple mogli da se koriste?

Prve verzije sistema su bile namenjene iskljucivo Macintosh racunarima.

1.62 Koje novine je doneo iOS?

iOS je mobilna verzija sistema OS X. Novina koju je ovaj sistem doneo je korisnicki interfejs koji se zasniva na multitac pristupu ugradnjom kapacitivnih ekrana u iPhone uređaje. Usavrseno je skrolovanje i omoguceno uvelicavanje prstima.

1.63 Kakvim uređajima je prvenstveno namenjen operativni sistem iOS?

Prvenstveno je namenjen pametnim telefonima iPhone, a zatim i za iPodTouch, iPad, iPad mini itd.

1.64 Na kojoj osnovi je zasnovan operativni sistem Android?

Zasnovan je na jezgru operativnog sistema Linux.

1.65 Kakvoj vrsti uređaja je namenjen Android?

Prvenstveno je namenjen za mobilne uređaje osetljive na dodir kao sto su pametni telefoni i tabletovi.

1.66 Koji su glavni razlozi za veliku popularnost Android-a?

Android je projekat otvorenog koda što znači da proizvodjacima omogućava da ga menjaju i prilagodjavaju potrebama svojih uređaja. Privukao je pažnju programera koji su dobili slobodu da razvijaju različite aplikacije.

2. Upravljanje procesima

2.1 Objasniti razliku izmedju programa i procesa.

Program je pasivan i opisuje sekvencu instrukcija koje treba izvršiti, dok je proces aktivan i izvršava se korak po korak po algoritmu koji je implementiran u programu.

2.2 Koji su memorijski zahtevi procesa?

Instrukcije se nalaze u memoriji zajedno sa ostalim podacima.

2.3 Koje su najbitnije informacije o toku izvršavanja procesa? Gde se cuvaju?

Bitne informacije su izvrsni kod, podaci, sadrže ih i registri – programski brojac koji vodi računa o tome dokle se stiglo sa izvršavanjem procesa; podaci o otvorenim fajlovima; informacije o dozvolama i vlasniku procesa itd.

2.4 Od cega zavise duzine vremenskih intervala koje procesi dobijaju za izvršavanje na procesoru?

Vremenski intervali zavise od operativnog sistema i ne moraju biti jednaki.

2.5 Koje segmente obicno procesi imaju naraspolažanju kada se ucitaju u memoriju?

Procesi raspolažu sa 4 dela memorije (segmenti): stek (lokalne promenljive i parametri funkcija), hip (prostor koji se dinamicki alocira), segment podataka (globalne promenljive) i kod segment (instrukcije koje proces treba da izvrsti).

2.6 Sta je izvorni kod programa?

Izvorni kod programa predstavlja niz instrukcija koje treba izvršiti i cuva se kao skup znakova (tekst).

2.7 Sta je izvrsni program i kako on nastaje?

Izvrsni program nastaje prevodenjem napisanog izvornog programa na masinski jezik. To je fajl na nekoj od sekundarnih memorija.

2.8 Koja stanja procesa obicno podrzavaju operativni sistemi?

Vecina sistema podrzava sledeća osnovna stanja:

- novi – proces je upravo kreiran i prelazi u spremno stanje
- spremjan – proces ceka da OS donese odluku da mu bude dodeljen procesor
- izvršavanje – proces se izvršava na procesoru
- cekanje – za dalji rad procesa je potreban neki zauzeti resurs tako da on ceka da se stvore uslovi da bi mogao da nastavi sa radom
- završen – proces je završio sa izvršavanjem i trebalo bi ga izbaciti iz sistema

2.9 Pod kojim uslovima procesi menjaju stanje?

Iz trenutnog stanja proces prelazi u neko drugo u sledecim situacijama:

- spremjan → izvršavanje – kada se procesor oslobodi i OS odabere njega iz liste spremnih procesa i dodeli mu procesor
- izvršavanje → cekanje – kada su procesu za dalje izvršavanje potrebni zauzeti resursi
- izvršavanje → spremjan – kada istekne vreme koje je procesu unapred određeno prilikom dodeljivanja procesora ili ako OS donese odluku da prekine proces
- cekanje → spremjan – kada proces dodje do potrebnih resursa i spremjan je za dalji rad

2.10 Sta se dogadja sa suspendovanim procesima?

Suspendovan proces oslobadja resurse koje je zauzimao pre suspenzije i prestaje da konkurise za druge resurse koji su mu potrebni za izvrsavanje, ali i dalje ostaje proces. Cesto se prebacuju na disk do prestanka suspenzije, cime se oslobadja i deo radne memorije.

2.11 Iz kog razloga se najcesce suspenduju spremni procesi?

Do prelaska procesa iz stanja spreman u stanje suspendovan i spreman najcesce dolazi zbog preopterenosti sistema zbog velikog broja spremnih procesa. Do suspendovanja moze doci i da bi se izbeglo zaglavljivanje sistema ili ako korisnik zeli privremeno da zaustavi izvrsavanje procesa kako bi proverio medjurezultate.

2.12 Zbog cega se najcesce suspenduju procesi koji cekaju?

Najcesce se vrsti da bi se oslobodili resursi kojima proces raspolaže i time sprecilo zaglavljivanje ili ubrzao rad sistema.

2.13 Cemu sluzi kontrolni blok procesa?

Kontrolni blok procesa je dinamicka struktura podataka koja cuva informacije o svakom pokrenutom procesu i sadrzi najznacajnije podatke koji omogucavaju identifikaciju i upravljanje procesima.

2.14 Kako se mogu podeliti informacije u procesima koje obicno cuva kontrolni blok procesa?

Kontrolni blok procesa sadrzi sledece informacije o procesima:

1. jedinstveni identifikator (PID) – broj koji proces dobija u trenutku nastanka
2. stanje procesa – informacija o trenutnom stanju u kojem se nalazi proces
3. programski brojac – informacija o sledecoj instrukciji koju proces treba da izvrsi
4. sadrzaj registara procesora – vrednosti koje se nalaze u registrima
5. prioritet procesa – informacija o vaznosti procesa u odnosu na ostale procese u sistemu
6. adresa memorije gde se nalazi proces – pokazivaci na adrese gde se nalaze segmenti procesa
7. adrese zauzetih resursa – informacija na kojim lokacijama treba OS da trazi potebne podatke

2.15 Koje mehanizme za manipulaciju kontrolnim blokovima procesa bi trebalo da obezbede operativni sistemi?

Kreiranje kontrolnog bloka za novi proces, unistavanje kontrolnog bloka procesa koji se izvrsio, menjanje stanja procesa, menjanje prioriteta procesa, izbor procesa za izvrsavanje.

2.16 Objasniti kako funkcionise prebacivanje konteksta.

Prebacivanje konteksta je postupak kojim se proces koji se trenutno izvrsava na procesoru prekida, pamte njegovi parametri, a zatim se umesto njega pokrece neki drugi proces i pri tome se ucitavaju parametri drugog procesa.

2.17 Koji modul operativnog sistema je zaduzen za prebacivanje konteksta?

Modul koji je odgovoran da izvrsava prebacivanje konteksta se cesto naziva dispecer.

2.18 Koje su najcesce operacije koje se izvode sa kontrolnim blokovima procesa?

Punjene registra procesa, prebacivanje u korisnicki rezim rada i skok na odgovarajucu lokaciju u korisnickom programu kako bi se proces nastavio.

2.19 Objasniti koncept niti.

Niti (threads) predstavljaju osnovne jedinice za izvrsavanje u okviru procesa. Niti su delovi jednog procesa i izvrsavaju se koriscenjem resursa koji su njemu pridruzeni.

2.20 Sta je od podataka zajednicko za sve niti jednog procesa, a sta nije?

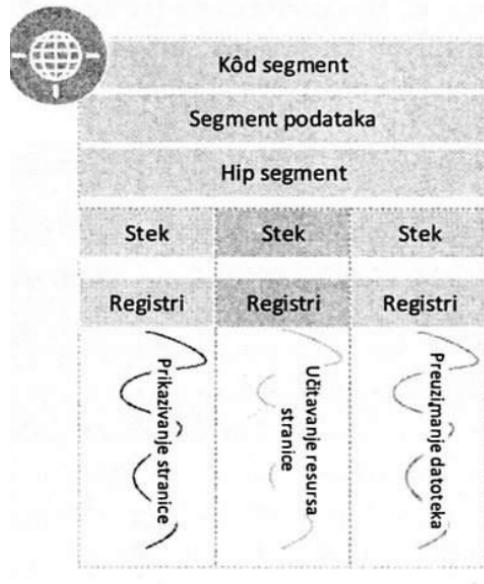
Niti imaju i sopstvene resurse. Svaka nit ima svoje registre, programski brojac i stek, a razlikuje ih i jedinstveni identifikator (TID). Kod segment, segment podataka, podaci o otvorenim fajlovima itd. su zajednicki za sve niti jednog procesa.

2.21 Koje su najvažnije prednosti koje donosi koriscenje niti?

Smanjuje se zauzeti prostor i omogucava se da vise niti obavlja razlicite zadatke u okviru jednog procesa. Pruza se mogucnost da se bolje iskoriste prednosti koje donosi viseprocesorska arhitektura.

2.22 Navesti primer upotrebe niti.

Upotreba niti se moze ilustrovati na primeru veb pregledaca. Oni se mogu implementirati tako da koriste bar tri niti. Jedna nit sluzi za prikazivanje hiperteksta u okviru prozora, druga ucitava podatke sa nekog servera, dok treca sluzi za preuzimanje podataka preko mreze. Ove niti dele podatke kroz zajednicke resurse i istovremeno se mogu izvrsavati na razlicitim jezgrima istog procesora.



2.23 Na kojim nivoima se moze pruziti podrška za rad sa nitima?

Korisnickim i sistemskim.

2.24 Na koji nacin se omogucava korespondencija korisnickih i sistemskih niti?

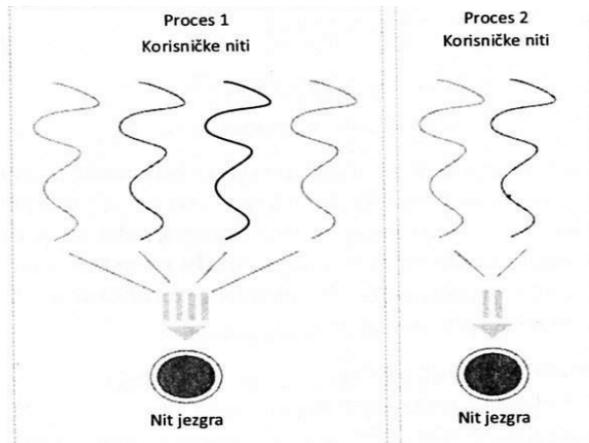
Omogucava se preslikavanjem (mapiranjem) korisnickih niti u niti jezgra. Najcesce podrzana preslikavanja su: preslikavanje vise u jednu, jedna u jednu ili preslikavanje vise u vise.

2.25 Koje su glavne odlike preslikavanja vise u jednu?

Podrazumeva da se vise korisnickih niti (sve koje pripadaju jednom procesu) preslikaju u jednu nit jezgra. Broj niti jezgra je jednak broju procesa koji postoje u sistemu. Jezgro manipulise iskljucivo sa procesima, tako da sve niti jednog procesa izvrsava kroz jednu nit jezgra. Nitima se upravlja iz korisnickog rezima bez uticaja jezgra operativnog sistema. Jezgro OS-a i dalje upravlja procesima i donosi odluke koja ce se od niti jezgra izvrsavati na procesoru.

Glavni nedostatak je u tome sto, u situacijama kada se neka od korisnickih niti blokira, blokira se i odgovarajuca nit jezgra, tako da dolazi do blokiranja celog sistema tj. svih niti koje ga cine.

Ovim pristupom se ne mogu iskoristiti prednosti viseprocesorske arhitekture.



2.26 Koja je najveća mana modela preslikavanja jedna u jednu?

Najveća mana je ogranicavanje broja niti jezgra a samim time i broja niti u korisnickom delu. Zbog toga sto se nezanemarljivo malo vremena i prostora trosi pri stvaranju i odrzavanju niti jezgra, mnogi operativni sistemi imaju ogranicenje dozvoljenog broja ovakvih niti.

2.27 Od cega zavisi broj niti jezgra koje ce biti kreiranje prilikom preslikavanja vise u vise?

Zavisi od konkretnog procesa, ali cesto i od broja procesora u sistemu.

2.28 Od cega se sastoji izvrsavanje procesa u sistemima koji podrzavaju multiprogramiranje i slicne koncepte?

Izvrsavanje procesa se obicno sastoji od naizmenicnog koriscenja procesora i cekanja na U\I operacije. Da procesor ne bi bio besposlen tokom U\I operacija procesa koji se izvrsava na njemu, nekom drugom procesu se dozvoljava da koristi procesor.

2.29 Koju ulogu imaju planeri u operativnim sistemima?

Uloga im je da od svih procesa koji bi trebalo da se izvrse, biraju podskup koji ce se odmah ucitati u memoriju i izvrsavati. Zatim odredjuju redosled kojim procesi dobijaju procesor i ostale uredjaje i koliko vremena mogu da koriste dobijene resurse.

2.30 Sta je red spremnih procesa i sta se u njemu nalazi?

Red spremnih procesa sadrzi procese koji su izabrani po nekom kriterijumu koji je odredjen na nivou operativnog sistema. Prosesi u ovom redu su spremni za izvrsavanje i smestenu su u glavnu memoriju.

2.31 Od cega zavisi izbor procesa za red spremnih procesa?

Izbor procesa zavisi od njihovih potreba, odnosno od procene da li ce u toku svog izvrsavanja vise koristiti procesor ili ce vise vremena provoditi u radu sa U\I uredjajima.

2.32 O kojim redovima procesa operativni sistemi vode racuna?

Vode racuna o redovima spremnih procesa i o redovima procesa koji cekaju na neke od uredjaja.

2.33 Koji se planeri najcesce implementiraju u okviru operativnih sistema?

Dugorocni i kratkorocni planeri, kao i srednjorocni planeri.

2.34 Koja je osnovna funkcija dugorocnog planera?

Funkcija dugorocnih planera je da od skupa svih procesa koji bi trebalo da se izvrse izaberu one koji će da se aktivno uključe u sistem i pocnu za izvrsavanjem (trebalo bi da naprave dobar odabir procesa za red spremnih procesa).

2.35 Sta je glavni zadatak kratkorocnih planera?

Kratkorocni planeri imaju zadatak da donose odluke o tome koji će se od spremnih procesa izvrsavati i koliko dugo će dobiti procesor.

2.36 Koja je uloga srednjorocnog planera?

Uloga srednjorocnog planera je da poboljsa kombinaciju izabranih poslova ili da osloboди memoriju u slučaju zagusenja.

2.37 Kakva se resenja za rasporedjivanje procesa mogu implementirati u viseprocesorskim sistemima?

Postoje dva pristupa: simetricno i asimetricno multiprocesiranje.

Simetricno multiprocesiranje podrazumeva da su procesori ravnopravni i da se procesi rasporedjuju na bilo koji od njih u zavisnosti od primjenjenog algoritma za rasporedjivanje.

Asimetricno multiprocesiranje se zasniva na ideji da neki procesori mogu biti zaduzeni za odredjene funkcije.

2.38 Kakvi pristupi postoje kada je resenje za rasporedjivanje procesa u viseprocesorskim sistemima sa zajednickim redom cekanja u pitanju?

Mogu se implementirati resenja koja podrazumevaju jedan zajednicki red cekanja procesa za sve procesore ili razlicite redove cekanja za svaki od procesora.

2.39 Sta se podrazumeva pod balansiranjem opterecenja?

Balansiranje opterecenja je postupak kojim se tezi da se sto ravnomernije podele poslovi izmedju procesora. Najcesca dva nacina koja se primenjuju su: prenosenje i preuzimanje migracije.

2.40 Iz kojih razloga se uvodi afinitet procesa prema procesoru?

Iz razloga sto se rilikom prebacivanja procesa sa jednog na drugi procesor, mora isprazniti memorija prvog procesora a sadrzaj premestiti na drugi sto predstavlja izgubljeno vreme.

2.41 Kakav moze biti afinitet procesa prema procesoru?

Može biti slab ili jak. Slab afinitet podrazumeva da operativni sistem omogucava ali i ne garantuje da će proces nastaviti izvrsavanje na istom procesoru. Jak afinitet podrazumeva da sistem dopusta da se procesu precizno odredi podskup procesora na kojima može da se izvrsava.

2.42 Sta se podrazumeva pod rasporedjivanjem procesa?

Pod rasporedjivanjem procesa se podrazumeva donosenje odluka o toku izvrsavanja (promeni stanja) procesa koji se nalaze u memoriji. Rasporedjivanje bi trebalo da bude sto blize optimalnom, jer od njega zavisi efikasnost sistema.

2.43 Koji su kriterijumi koji se posmatraju prilikom ocenjivanja algoritama rasporedjivanja?

Kvalitet algoritama se obicno ocenjuje na osnovu sledecih kriterijuma:

- iskoriscenost procesora – procenat koliko je procesor bio zaposlen u određenom periodu
- propusna moc – broj procesa koji se mogu zavrsiti u jedinici vremena
- vreme obrade – vreme koje protekne od momenta pokretanja procesa do njegovog zavrsetka

- vreme cekanja – ukupno vreme koje proces provede cekajući u redu spremnih procesa
- vreme odziva – vreme koje protekne od prijavljivanja procesa do trenutka kada se proizvede prvi izlaz programa
- kolicnik duzine vremena obrade i trajanja izvršavanja procesa

2.44 Koje su glavne odlike FCFS algoritma za rasporedjivanje procesa?

FCFS (First Come, First Served) je zasnovan na ideji da se procesor dodeljuje procesima po redu kako su ga zatrazili tj. prijavili se za njegovo koriscenje. Jednostavno se implementira koriscenjem FIFO strukture. Nema verziju koja bi dozvoljavala prekidanje. Vreme cekanja je cesto veoma veliko. Ovaj pristup može dovesti do situacije kod koje dosta procesa ceka da se izvrsti jedan vremenski zahtevan proces.

2.45 Koja je osnovna ideja SPF algoritma za rasporedjivanje procesa?

SPF (Shortest-Process-First) se zasniva na ideji da se favorizuju procesi čiji je sledeći zahtev za procesorom manji.

2.46 U cemu je razlika izmedju verzija SPF algoritma sa i bez prekidanja?

Razlika je u tretmanu novih procesa koji imaju sledeći zahtev za procesorskim vremenom u odnosu na proces koji se izvršava na procesoru. Ako se dozvoli prekidanje procesa, kada se pojavi novi proces cija je sledeća operacija na procesoru manje zahtevna od one koja se trenutno izvršava, vrši se prebacivanje konteksta procesa koji se izvršava a novom procesu se dozvoljava da na procesoru izvrsti operaciju.

2.47 Koji je najveći problem kada je realizacija SPF algoritma u pitanju?

Najveći problem predstavlja odredjivanje duzine sledećeg zahteva za procesorom, odnosno procena duzine trajanja sledeće aktivnosti procesa na procesoru.

2.48 Na koji nacin se obicno vrši procena duzine sledeće aktivnosti procesa na procesoru?

Obično se za predviđanje duzine sledeće aktivnosti koristi eksponencijalna sredina prethodnih aktivnosti i prethodnih procena potrebnog vremena. Koristi se pretpostavka da će sledeća aktivnost biti slične duzine kao prethodne.

2.49 Prodiskutovati algoritam sa prioritetima za rasporedjivanje procesa.

Podrazumeva da se svakom procesu pridruži velicina koja predstavlja njegov prioritet i da se kao sledeći za izvršavanje bira onaj koji ima najviši prioritet.

2.50 Koje su mane algoritma sa prioritetima?

Najveća mana ovog algoritma je potencijalno izgladnjivanje procesa. Može se dogoditi da procesi nizeg prioriteta dugo cekaju na procesor jer u sistem stalno pristizu procesi viseg prioriteta.

2.51 Na koji nacin se mogu definisati prioriteti kod algoritma sa prioritetima?

Prioriteti se mogu definisati interno ili eksterno. Interno definisani prioriteti koriste neku merljivu velicinu za izracunavanje prioriteta procesa. Eksterni prioriteti se namecu na osnovu kriterijuma koji su spoljasnji u odnosu na operativni sistem, odnosno uglavnom su političke prirode.

2.52 Kako se moze resiti problem “izgladnjivanja” procesa kod algoritma sa prioritetima?

Može se resiti povećanjem prioriteta procesa sa porastom njegovog vremena provedenom u redu za cekanje.

2.53 Koja je osnovna ideja kruznoga algoritma za rasporedjivanje procesa?

Zasnovan je na ideji za opsluzivanje racunara slicno kao kada je deljenje vremena u pitanju. Svaki proces dobija procesor na unapred zadati vremenski interval (kvantum vremena).

2.54 Kako se ponasa kruzni algoritam u zavisnosti od duzine kvantuma?

Ako ima n spremnih procesa, a vremenski kvantum je duzine q, onda svaki proces dobija najmanje $1/n$ procesorskog vremena, najvise po q u jednom prolazu. Nijedan proces ne ceka vise od $(n-1)q$ vremena.

Ako je kvantum vremena veci od svih mogucih zahteva za procesorom, algoritam se ponasa kao FCFS algoritam. Ako je kvantum veoma mali, kruzni algoritam se naziva deljenje procesora i izgleda da svaki od n procesa u redu ima sopstveni procesor.

2.55 O cemu treba voditi racuna prilikom odredjivanja duzina kvantuma vremena kod kruznog algoritma?

Pozeljno je da se kvantum vremena izabere tako da bude znatno duzi od vremena potrebnog za prebacivanje konteksta. Takodje i vreme obrade zavisi od kvantuma vremena, odnosno ono je krace ako vecina poslova svoju aktivnost na procesoru zavrsava za jedan kvantum vremena.

2.56 Prodiskutovati karakteristike pristupa zasnovanog na redovima u vise nivoa za rasporedjivanje procesa.

Redovi u vise nivoa predstavljaju pristup koji je zasnovan na ideji da se red spremnih procesa podeli u vise redova sa razlicitim prioritetima. Svaki red moze imati sopstveni algoritam planiranja, a izmedju samih redova postoji jasno definisana razlika u prioritetu.

2.57 Koja je glavna ideja kada su redovi u vise nivoa sa povratnom vezom za rasporedjivanje procesa u pitanju?

Kod redova u vise nivoa sa povratnom vezom dozvoljeno je kretanje procesa izmedju redova. Ideja je da se u toku izvrsavanja izdvoje procesi sa razlicitim trajanjem aktivnosti na procesoru tako da se procesi koji koriste previse procesorskog vremena premeste u redove sa nizim prioritetom. Svaki proces u visim redovima dobija odredjeni kvantum vremena i kada ga iskoristi, premesta se u red na nizem nivou.

2.58 Navesti primer algoritma zasnovanog na redovima u vise nivoa sa povratnom spregom.

Ako postoje tri reda sa sledecim algoritmima rasporedjivanja: Q_0 – kruzni algoritam sa kvantumom od 10 vremenskih jedinica; Q_1 - kruzni algoritam sa kvantumom od 20 vremenskih jedinica; Q_2 – FCFS algoritam. Kada se pojavi novi proces, on se smesta u prvi red Q_0 . Tu dobija pravo da koristi 10 vremenskih jedinica. Ako za to vreme proces ne zavrsi sa izvrsavanjem, premesta se u red Q_1 . U Q_1 proces dobija novih 20 vremenskih jedinica, a ako ni tada ne zavrsi, prebacuje se u red Q_2 gde ce ostati do kraja izvrsavanja.

2.59 Koje su glavne odlike sistema za rad u realnom vremenu?

Spadaju u operativne sisteme sa specijalnom namenom. Obicno se projektuju za odredjenu primenu i koriste u situacijama kada postoje vremenska ogranicenja za izvrsavanje unapred definisanih poslova. Od sustinskog je znacaja da odredjeni poslovi budu obavljeni na vreme.

2.60 Koje vrste sistema za rad u realnom vremenu postoje?

Postoje cvrsti i labavi sistemi.

Cvrsti sistemi za rad u realnom vremenu podrazumevaju da u sistemu postoji bar jedan proces koji ima cvrsti krajnji rok za izvrsavanje koji se mora ispostovati.

Labavi sistemi za rad u realnom vremenu podrazumevaju da je postovanje zadatog roka za izvrsavanje procesa pozeljno, ali nije od presudnog znacaja za funkcionisanje sistema.

2.61 Kako se procesima u sistemima za rad u realnom vremenu mogu dodeliti prioriteti?

Prioriteti se mogu dodeliti staticki ili dinamicki.

Staticko odredjivanje prioriteta se obavlja u fazi implementiranja sistema i ovako dodeljeni prioriteti se ne mogu menjati u toku rada sistema.

Dinamicko odredjivanje prioriteta podrazumeva da se oni mogu menjati u toku rada sistema.

2.62 Koje su odlike RMS algoritma za rasporedjivanje procesa u sistemima za rad u realnom vremenu?

RMS (Rate Monotonic Scheduling) algoritam za rasporedjivanje procesa se primenjuje kod sistema za rad u realnom vremenu sa periodicnim procesima. Prioriteti se određuju na osnovu perioda procesa tako da proces sa najkracom periodom dobija najviši prioritet. Dozvoljava prekidanje.

2.63 Koje su odlike EDFS algoritma za rasporedjivanje procesa u sistemima za rad u realnom vremenu?

EDFS (Earliest-Deadline-First Scheduling) je dinamicki algoritam koji najviši prioritet dodeljuje procesu koji je najbliži svom krajnjem roku za izvrsavanje. Garantuje održivost sistema pod uslovom da je opterećenje sistema manje od maksimalnog.

3. Konkurentnost i sinhronizacija procesa

3.1 Koje su negativne posledice konkurentnog izvršavanja procesa?

Pri cestim promenama procesa/niti koji koriste procesor nekontrolisano pristupanje zajednickim podacima moze dovesti do nezeljenog krajnjeg rezultata. Deljeni podaci su veoma osetljivo mesto. Prekidanje procesa koji im je pristupao i propustanje drugog moze dovesti do nekonzistentnosti podataka i neocekivanih gresaka.

3.2 Navesti primere trke za resurse.

Primer 3.2. Izvršavanje dva procesa

Proces 1:

Smestiti promenljivu X u registar R ;
Uvećati registar R za 1;
Prekid procesa;

Proces 2:

Smestiti promenljivu X u registar R ;
Uvećati registar R za 1;
Premestiti sadržaj регистра R u promenljivu X ;
Prekid procesa;

Proces 1:

Premestiti sadržaj регистра R u promenljivu X ;

Primer 3.4. Proizvođač – potrošač

Glavni program:

```
//Prva slobodna pozicija u magacinu na koju se dodaje novi proizvod  
unutra = 0;  
//Pozicija najstarijeg proizvedenog proizvoda u magacinu  
van = 0;  
//Broj proizvoda u magacinu  
brojač = 0;  
//Procesi proizvođač i proces potrošač se izvršavaju paralelno  
PARALLEL WHILE (true)  
    //Procesi mogu biti pozvani u proizvoljnom trenutku  
    proizvođač();  
    potrošač();  
END PARALLEL WHILE
```

proizvođač()

WHILE (true)

```
    Novi proizvod je proizведен;  
    //Magacin je pun ako je brojač proizvoda u magacinu jednak broju  
    //mesta u magacinu i treba sačekati  
    WHILE (brojač == veličina_magacina)  
        //Aktivno čekanje dok se ne osloboди bar jedno mesto u magacinu  
ENDWHILE
```

```
    //Novi proizvod se dodaje na prvu sledeću slobodnu poziciju
```

```
    magacin[unutra] = novi_proizvod;
```

```
    //Pomera se indeks na sledeću poziciju
```

```
    unutra = (unutra + 1) mod veličina_magacina;
```

```
    //Uvećava se brojač koji govori koliko je proizvoda u magacinu
```

```
    brojač++;
```

ENDWHILE

potrošač()

```
    //Ako je brojač koji broji koliko je proizvoda u magacinu jednak
```

```
    //0 to znači da je magacin prazan
```

```
    WHILE(brojač == 0)
```

```
        //Aktivno čekanje da se bar jedan proizvod proizvede
```

ENDWHILE

```
    //Uzima se najstarije proizvedeni proizvod
```

```
    prodaja = magacin[van];
```

```
    //Povećava se brojač koji pokazuje na proizvod koji je
```

```
    //najduže u magacinu
```

```
    van = (van + 1) mod veličina_magacina;
```

```
    //Ažurira se broj proizvoda koji se nalazi u magacinu tako da
```

```
    //odgovara stvarnom broju tj. umanjuje se za jedan
```

```
    brojač--;
```

3.3 Kako se u racunarstvu definise pojam kriticne sekcije?

Kriticna sekcija je deo programa u kojem se pristupa zajednickim podacima.

3.4 Sta bi trebalo da obuhvati kriticna sekcija?

Kriticna sekcija treba da obuhvati deljene podatke.

3.5 Koje uslove bi trebalo da zadovolji dobro resenje za zastitu kriticne sekcije?

Uslovi:

- uzajamna iskljucivost – dva procesa ne mogu u isto vreme da budu u kriticnoj sekciji
- uslov progrusa – proces koji nije u kriticnoj sekciji i ne zeli da udje u nju ne treba da ometa druge procese da udju u nju
- uslov konacnog cekanja – trebalo bi da postoji granica koliko jedan proces moze da ceka na ulazak u kriticnu sekciju

3.6 Koji su glavni nedostaci resenja problema zastite kriticne sekcije koja podrazumevaju iskljucivanje prekida?

Sustina implementacije multiprogramiranja je zasnovana na sistemu prekida, pa bi njegovo cesto iskljucivanje dovelo do smanjenja efikasnosti sistema. Takodje, cesto nije jasno definisano sta kriticna sekcija treba da obuhvati.

3.7 Prodiskutovati resenja za zastitu kriticne sekcije koja se zasnivaju na aktivnom cekanju.

Aktivno cekanje – cekanje u ciklusu (petlji) dok se ne stvore uslovi za ulazak procesa u kriticnu sekciju. Ideja je da se aktivnim cekanjem u ciklusu saceka da promenljiva *moze* postane 1, odnosno da ulaz u kriticnu sekciju bude slobodan.

U prvom algoritmu najvazniji uslov uzajamne iskljucivosti nije zadovoljen. Moze se dogoditi da jedan proces izvrsi proveru da li je *moze* jednako 1 i posto utvrdi da jeste, kreće da je postavi na 0. Ako taj proces bude prekinut pre nego sto se *moze* postavi na 0, drugi proces moze videti da je ulaz u kriticnu sekciju slobodan i u nju uci.

Algoritam 3.1. Zaštita kritične sekcije (1)

<p>Proces 1:</p> <pre>WHILE (može == 0) //Aktivno čekanje ENDWHILE može = 0; //Kritična sekcija može = 1;</pre>	<p>Proces 2:</p> <pre>WHILE (može == 0) //Aktivno čekanje ENDWHILE može = 0; //Kritična sekcija može = 1;</pre>
---	---

Kao bolje resenje uvode se promenljive *zeli_1* i *zeli_2* umesto promenljive *moze*. Svaki proces najavi da zeli u kriticnu sekciju i aktivno ceka dok i drugi to zeli. Kada zavrsti sa radom u kriticnoj sekciji, svoju promenljivu postavlja na 0 cime drugi proces oslobadja od aktivnog cekanja. Moze se dogoditi da se procesi zaglave.

Algoritam 3.2. Zaštita kritične sekcije (2)

<p>Proces 1:</p> <pre>želi_1 = 1; WHILE(želi_2 == 1) //Aktivno čekanje ENDWHILE //Kritična sekcija želi_1 = 0;</pre>	<p>Proces 2:</p> <pre>želi_2 = 1 WHILE(želi_1 == 1) //Aktivno čekanje ENDWHILE //Kritična sekcija želi_2 = 0;</pre>
--	---

3.8 Napisati i analizirati algoritam striktne alternacije.

Striktna alternacija je jedno od resenja za zastitu kriticne sekcije. Zasniva se na koriscenju promenljive *na_redu* kojom se odredjuje koji od dva procesa ima prednost. Na pocetku se prednost daje prvom procesu (*na_redu = 1*).

Procesi aktivno cekaju da dodju na red. Kada napusti kriticnu sekciju, proces promenljivoj *na_redu* postavlja vrednost broja drugog procesa koji tako dobija sansu da pristupi kriticnoj sekciji.

Ispunjava samo uslov uzajamne iskljucivosti. Ako jedan proces ima prednost a ne zeli da udje u kriticnu sekciju, on ce blokirati drugi proces i time onemoguciti progres, a posto blokiranje moze da bude na neodredjeno vreme, ni uslov konacnog cekanja nije ispunjen.

Algoritam 3.3. Striktna alternacija

Proces 1:	Proces 2:
WHILE (<i>na_redu == 2</i>) //Aktivno čekanje ENDWHILE //Kriticna sekcija <i>na_redu = 2;</i>	WHILE (<i>na_redu == 1</i>) //Aktivno čekanje ENDWHILE //Kriticna sekcija <i>na_redu = 1;</i>

3.9 U kojim situacijama je pogodno koristiti algoritam striktne alternacije?

Striktna alternacija je pogodna za problem sinhronizacije kada procesi treba naizmenicno da pristupaju deljenim resursima.

3.10 Napisati i prodiskutovati Dekerov algoritam za zastitu kriticne sekcije.

Dekerov algoritam predstavlja prvo kompletno resenje za uzajamno iskljucivanje u slucaju kada dva procesa konkuruju za deljene podatke. Za resenje su potrebne tri promenljive: *zeli_proces_1* - prvi proces najavljuje da zeli u kriticnu sekciju; *zeli_proces_2* - drugi proces najavljuje da zeli u kriticnu sekciju; *na_redu* - koji je proces na redu da udje u kriticnu sekciju (1 ili 2).

Na pocetku se jednom procesu daje prednost. Neka je *na_redu = 1*. Kada prvi proces zeli da pristupi kriticnoj sekciji on to najavi: *zeli_proces_1 = 1*. Zatim se proverava da li postoji konkurenca.

Ako je *zeli_proces_2 = 0*, prvi proces je slobodan da udje u kriticnu sekciju. Po zavrsetku rada u kriticnoj sekciji proces označava da je zavrsio sa njenim koriscenjem, *zeli_proces_1 = 0*.

Ako je *zeli_proces_2 = 1*, onda se proverava ko ima prednost i oba procesa ulaze u spoljasnju petlju u kojoj ce ostati dok jedan od procesa ne postavi svoju zelju na 0. Ako je drugi proces na redu, prvi menja vrednost svoje promenljive na 0 cime drugi proces napusta spoljasnju petlju i ulazi u kriticnu sekciju. Prvi tada aktivno ceka u unutrasnjoj petlji dok drugi proces ne zavrsi rad u kriticnoj sekciji i postavi *na_redu* na 1. Time prvom procesu omogucava izlaz iz unutrasnje petlje, a postavljanjem *zeli_proces_2* na 0 i iz spoljasnje petlje. Prvi proces moze tada da udje u kriticnu sekciju.

Algoritam 3.4. Dekerov algoritam

Proces 1:	Proces 2:
//Proces najavljuje da želi u kriticnu sekciju <i>zeli_proces_1 = 1;</i> //Ako to želi i drugi... WHILE (<i>zeli_proces_2 == 1</i>) //Proverava se da li drugi ima prednost IF (<i>na_redu == 2</i>) //Menja se želja <i>zeli_proces_1 = 0;</i> //Proverava se da li drugi ima prednost WHILE (<i>na_redu == 2</i>) //Aktivno čekanje ENDWHILE <i>zeli_proces_1 = 1;</i> ENDIF ENDWHILE //Kriticna sekcija <i>na_redu = 1;</i> <i>zeli_proces_2 = 0;</i>	<i>zeli_proces_2 = 1;</i> WHILE (<i>zeli_proces_1 == 1</i>) IF (<i>na_redu == 1</i>) <i>zeli_proces_2 = 0;</i> WHILE (<i>na_redu == 1</i>) ENDWHILE <i>zeli_proces_2 = 1;</i> ENDIF ENDWHILE //Kriticna sekcija <i>na_redu = 1;</i> <i>zeli_proces_2 = 0;</i>

3.11 Koja je najveca mana Dekerovog algoritma?

Moze se dogoditi da jedan proces prodje dva puta zaredom cime je narusen princip pravednosti. Moze se dogoditi da prvi proces izadje iz kriticne sekcije i da najavi da hoce ponovo u nju, kao i da drugi proces izadje iz aktivnog cekanja ali da ne uspe da stigne da postavi *zeli_proces_2* na 1. U tom slučaju, prvi proces ponovo ulazi u kriticnu sekciju.

Ovaj algoritam je namenjen iskljucivo za situacije kada dva procesa konkurisu za iste podatke.

3.12 Napisati i analizirati Pitersonov algoritam za zastitu kriticne sekcije.

Osnovna ideja ovog algoritma je u tome da kada proces najavi da zeli da udje u kriticnu sekciju, ustupa prednost drugom procesu. Ova prednost vazi samo ako drugi proces zeli da udje u kriticnu sekciju.

Da bi usao u kriticnu sekciju, proces treba da to i najavi, *zeli_proces* = 1. Zatim on daje prednost drugom procesu i ceka dok drugi proces zeli da udje u kriticnu sekciju i ima prednost. Ovim je obezbedjeno da procesi naizmenicno ulaze u kriticnu sekciju, ako to zele, bez obzira na brzinu.

Algoritam 3.5. Pitersonov algoritam	
Proces 1: <i>zeli_proces_1</i> = 1; <i>na_redu</i> = 2; WHILE (<i>zeli_proces_2</i> == 1 AND <i>na_redu</i> == 2) //Aktivno čekanje ENDWHILE //Kritična sekcija <i>zeli_proces_1</i> = 0;	Proces 2: <i>zeli_proces_2</i> = 1; <i>na_redu</i> = 1; WHILE (<i>zeli_proces_1</i> == 1 AND <i>na_redu</i> == 1) //Aktivno čekanje ENDWHILE //Kritična sekcija <i>zeli_proces_2</i> = 0;

3.13 Koji su glavni nedostaci Pitersonovog algoritma?

Mana algoritma je to sto je primenljiv samo za dva procesa.

3.14 Prodiskutovati karakteristike Lamportovog (pekarskog) algoritma za zastitu kriticne sekcije.

Lamportov algoritam je uopstenje Pitersonovog algoritma za n procesa. Za jednostavniji prikaz algoritma potrebno je uvesti sledeće oznake:

- $(A, B) < (C, D)$ ako $A < C$ ili $A = C$ i $B < D$.
- $\max(A_1, A_2, \dots, A_n) = K$, pri cemu je K takvo da je $K \geq A_i$, $i = 1, 2, \dots, n$.

Algoritam 3.6. Lamportov (pekarski) algoritam

Proces i:
//Zaštita uzimanja broja
uzima[i] = 1;
broj[i] = $\max(\text{broj}[0], \dots, \text{broj}[n - 1]) + 1$;
uzima[i] = 0;
//Glavna petlja koja je graničnik kritične sekcije
FOR ($j = 0; j < n; j++$)
 WHILE (*uzima[j]* == 1)
 //Aktivno čekanje da j -ti proces dobije broj
ENDWHILE
 WHILE (*broj[j] != 0 AND (broj[j], j) < (broj[i], i)*)
 //Aktivno čekanje da proces koji ima prednost završi
ENDWHILE
ENDFOR
//Kritična sekcija
broj[i] = 0;

Potrebno je kao globalne promenljive deklarisati celobrojni niz *broj[n]* i niz logickih promenljivih *uzima[n]* i pri tome oba niza inicijalizovati na nulu.

Uz pomoc *uzima[i]* stiti se dobijanje broja za i -ti proces (*broj[i]*) koji dobija vrednost koja je za jedan veca od trenutno najveceg broja koju imaju ostali procesi. Zatim se redom obilaze svi procesi j koji imaju bolji broj od njega. Proces i treba da saceka da j -ti proces zavrssi sa uzimanjem broja ukoliko je u toj fazi, a zatim i da, ako j -ti proces ima prednost, saceka da on zavrssi sa pristupom kriticnoj sekciji i postavi *broj[j] = 0*.

3.15 Koje su najcesce instrukcije koje se hardverski implementiraju kako bi pomogle pri zastiti kriticne sekcije?

Najcesce se koriste sledece tri instrukcije:

- TAS (Test and Set) – operise sa dve promenljive $A = TAS(B)$ i funkcione tako sto se vrednost koja se nalazi u B prebacuje u A, a u B se stavlja 1.
- FAA (Fetch and Add) – operise sa dve promenljive. Sintaksa je $FAA(A, B)$ i pri njenom koriscenju se vrednost B prebacuje u A, a vrednost $B+A$ se smesta u B (uzima se staro A).
- SWAP (zamena) – operise sa dve promenljive, $SWAP(A, B)$ i rezultat je atomicna zamena vrednosti promenljivih A i B. Zamena se sastoji od bar tri operacije.

Obicno postoji jedna promenljiva kojom se stiti ulaz u kriticnu sekciju, a procesi aktivno cekaju da se ulaz oslobodi. Kada se to desi, onda atomicnost ovih instrukcija omogucava da tacno jedan proces udje u kriticnu sekciju.

3.16 Napisati algoritam za zastitu kriticne sekcije koji se zasniva na koriscenju TAS instrukcije.

Deklarise se globalna promenljiva *zauzeto* i postavlja se na nulu, dok svaki proces ima lokalnu promenljivu *ne_moze*. Promenljiva *ne_moze* se na pocetku izvrsavanja procesa postavlja na 1, a onda se u petlji stalno proverava da li je *zauzeto* postavljeno na 0 (da li je ulaz u kriticnu sekciju slobodan). Kada se oslobodi ulaz, *zauzeto* = 0, atomicnom operacijom ce tacno jedan proces biti obavesten da moze da udje u kriticnu sekciju (*ne_moze* = 0) i kriticna sekcija ce se zatvoriti za ostale procese jer TAS operacija postavlja *zauzeto* na 1.

Algoritam 3.7. Zaštita kritične sekcije (TAS)

Proces i:

```
ne_može = 1;  
WHILE (ne_može == 1)  
    ne_može = TAS(zauzeto);  
ENDWHILE  
//Kritična sekcija  
zauzeto = 0;
```

3.17 Koje su prednosti i mane algoritma za zastitu kriticne sekcije zasnovane na koriscenju TAS instrukcije?

Mana je cinjenica da je resenje zasnovano na hardverskoj podrisci, odnosno potrebno je da procesor ima ugradjenu tu instrukciju. Moguce je izgladnjivanje procesa, ali se retko dogadja.

Prednost je sto se bez bilo kakve modifikacije moze primeniti na proizvoljan broj procesa, ali ne postoji granica koliko ce neki proces da ceka.

3.18 Na koji nacin se standardni algoritam za zastitu kriticne sekcije koriscenjem TAS instrukcije moze modifikovati?

Potrebno je deklarisati globalni niz *ceka[n]* ciji se clanovi inicializuju na nulu i promenljivu celobrojnog tipa *zauzeto* koja se postavlja na nulu.

Modifikacije se sastoje u tome da po zavrsetku rada u kriticnoj sekciji, proces koji je zavrsio pronadje sledeci od procesa koji cekaju i da ga pozove da udje u kriticnu sekciju tako sto ce mu promenljivu *ceka[j]* postaviti na 0 i time ga osloboediti iz aktivnog cekanja. Ako takav proces ne postoji, promenljiva *zauzeto* se postavlja na 0 cime se ulaz u kriticnu sekciju oslobadja.

Algoritam 3.8. Zaštita kritične sekcije (TAS) – modifikacija

```
Proces i:  
čeka[i] = 1;  
WHILE (čeka[i] == 1 AND TAS(zauzeto) == 1)  
    //Aktivno čekanje dok je kritična sekcija zauzeta  
ENDWHILE  
čeka[i] = 0;  
//Kritična sekcija  
  
//Traži se sledeći proces koji čeka  
j = (i + 1) mod n;  
//Treba ići dok se ne pronađe proces koji čeka ili ne obrne krug i dođe  
//do i-tog procesa  
WHILE (čeka[j] == 0 AND j != i)  
    j = (j + 1) mod n;  
ENDWHILE  
IF (i == j)  
    //Ako je obrnut ceo krug onda niko ne čeka na ulazak u kritičnu sekciju  
    zauzeto = 0;  
ELSE  
    //Inače se j-tom procesu prekida aktivno čekanje i on ulazi u  
    //kritičnu sekciju  
    čeka[j] = 0;  
ENDIF
```

3.19 Napisati algoritam za zastitu kriticne sekcije koriscenja SWAP instrukcije.

Brava je globalna promenljiva koja se postavlja na 0 sto označava da je prolaz u kriticnu sekciju slobodan. Procesi koji se nadmecu za pristup kriticnoj sekciji imaju promenljivu *kljuc* cija vrednost 1 govori da *kljuc* nije u bravi, odnosno da čekaju da udje u kriticnu sekciju. Procesi u petlji pokušavaju da kroz SWAP operaciju ubace *kljuc* u bravu (da ugrabe trenutak kada je vrednost promenljive *brava* jednak 0 pa da zamenom te vrednosti sa 1, koja se nalazi u *kljuc*, dobiju uslov za izlazak iz ciklusa i ulazak u kriticnu sekciju). Istovremeno *brava* dobija prethodnu vrednost *kljuc* koja je bila 1. Na ovaj nacin se ulazak u kriticnu sekciju blokira sve dok proces koji je u njoj ne završi i postavi *brava* na 0 cime se otvara mogućnost da jedan od procesa koji čekaju udje u kriticnu sekciju.

Algoritam 3.9. Zaštita kritične sekcije (SWAP)

```
//Brava je globalna promenljiva inicijalizovana na 0
Proces i:
    ključ = 1;
    WHILE (ključ != 0)
        SWAP (brava, ključ);
    ENDWHILE
//Kriticna sekcija
brava = 0;
```

3.20 Koje su mane hardverskih resenja za zastitu kriticne sekcije?

Zasnivaju se na aktivnom čekanju, odnosno tome da se u petlji proverava neki uslov i na taj nacin čeka da se ulaz u kriticnu sekciju oslobodi cime se trosi procesorsko vreme.

3.21 Na koji nacin funkcionišu metode za zastitu kriticne sekcije koje se ne zasnivaju na aktivnom čekanju?

Ove metode se obično zasnivaju na tehnikama koje podrazumevaju zaustavljanje procesa, bez aktivnog čekanja i njihovo pokretanje u odgovarajućem trenutku. Ideja je da se proces koji ne može da udje u kriticnu sekciju na neki nacin blokira pa da se po sticanju uslova probudi/pozove i odblokira.

3.22 Koje su najpoznatije metode za zastitu kriticne sekcije koje se ne zasnivaju na aktivnom čekanju?

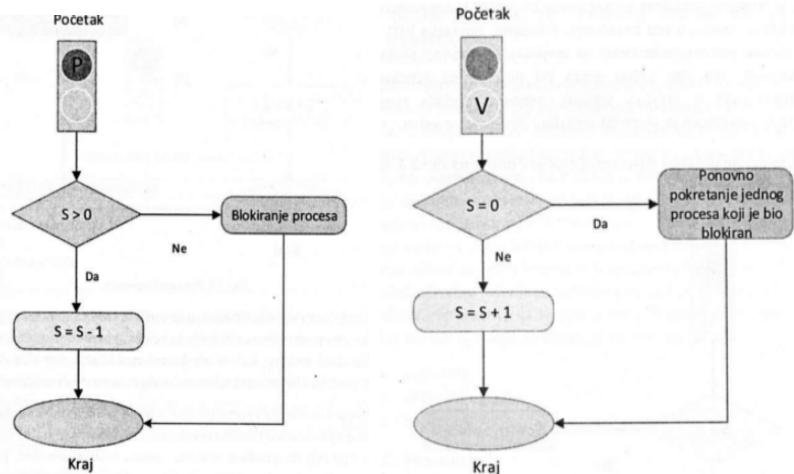
Najpoznatije metode se zasnivaju na koriscenju: semafora, kriticnih regiona ili monitora.

3.23 Koji je osnovni princip na kojem funkcionišu semafori?

Semafor je apstraktan tip podataka, odnosno struktura koja može da blokira proces na neko vreme i da ga propusti kada se steknu određeni uslovi. Nad semaforom se, pored standardnih operacija kreiranja i oslobođanja, mogu izvesti samo još dve operacije: P i V. Sintaksa poziva ovih operacija je P(S) i V(S), pri cemu je S semafor.

Operacijom P(S) se testira da li je vrednost promenljive S pozitivna i ako jeste, proces, koji je izvršio tu operaciju, se propusta dalje u kriticnu sekciju, a vrednost semafora se umanjuje za jedan. U suprotnom se proces blokira i stavljaju u red za čekanje.

Operacija V(S): ako postoje procesi koji čekaju blokirani na semaforu S, propusta tacno jedan od procesa da nastavi sa izvršavanjem, odnosno prekida njegovo blokiranje. U suprotnom se vrednost semafora uvećava za jedan.



Slika 3.3. Dijagram P operacije

Slika 3.4. Dijagram V operacije

3.24 Kakvog tipa je promenljiva u okviru semafora?

Promenljiva u okviru semafora je celobrojnog tipa.

3.25 Koje vrednosti moze imati promenljiva u okviru semafora?

Moze imati proizvoljne nenegativne celobrojne vrednosti (brojacki), ili samo vrednosti 0 ili 1 (binarni).

3.26 Na kom nivou se implementiraju semafori?

Semafori se obcni realizuju kao servis operativnog sistema.

3.27 Koji su glavni nedostaci semafora?

Pri njihovom koriscenju treba biti oprezan jer se malom greskom mogu napraviti veliki problemi.

3.28 U cemu je razlika izmedju brojackih i binarnih semafora?

Brojacki dozvoljavaju nenegativne celobrojne vrednosti koje određuju broj procesa koji se mogu propustiti kroz semafor, a binarni imaju samo vrednosti 0 ili 1.

3.29 Na koji nacin se binarni semafori mogu iskoristiti za zastitu kriticne sekcije?

Proces koji zeli da predje u kriticnu sekciju testira semafor, pa ako je njegova vrednost 1, on prolazi u kriticnu sekciju, a semafor postavlja na 0. Ako je vrednost 0, neki drugi proces je u kriticnoj sekciji i treba da saceka. Po zavrsetku rada u kriticnoj sekciji, proces izvrsava operaciju V(S) i time, ako neki od procesa cekaju blokirani na semaforu, dozvoljava onome koji je na redu da udje u kriticnu sekciju ili samo postavlja promenljivu S na 1.

3.30 Prodiskutovati implementaciju brojackih semafora koriscenjem binarnih.

Za implementaciju su potrebna dva binarna semafora S_1 i S_2 i promenljiva C koja je celobrojnog tipa. Prilikom inicijalizacije algoritma, vrednost semafora S_1 se postavlja na 1, semafora S_2 na 0, dok promenljiva C dobija vrednost n . Semafor S_1 se koristi da obezbedi ekskluzivan pristup instrukcijama u okviru aritmetickih operacija nad promenljivom C brojackog semafora, odnosno da obezbedi atomicnost ovih operacija. Semafor S_2 se koristi za cekanje procesa. Promenljiva C cuva vrednost brojackog semafora, odnosno određuje koliko procesa semafor moze da propusti.

Implementacija operacije P: Na pocetku se testira binarni semafor S_1 i ako je zatvoren, proces dobija ekskluzivno pravo da pristupi brojackom semaforu i izvrsi operaciju P. Proces prvo umanjuje vrednost promenljive C za 1. Ako je n procesa vec proslo kroz semafor, $C < 0$, onda ovaj proces ne moze da prodje pa se oslobadja

semafor S_1 i blokira se operacijom $P(S_2)$ na semaforu S_2 . Inace, $C > 0$, operacijom $V(S_1)$ se drugim procesima daje sansa da se odblokiraju.

Implementacija operacije V: Pomocu $P(S_1)$ se obezbedjuje ekskluzivan pristup brojackom semaforu.

Promenljiva C se uvecava za jedan i ako je $C < 0$, to znaci da neki proces ceka blokiran na semaforu S_2 . Tada se operacijom $V(S_2)$ propusta tacno jedan proces koji je blokiran. Inace se sa $V(S_1)$ oslobadja semafor S_1 koji stiti ekskluzivni pristup operacijama brojackog semafora.

Algoritam 3.12. Implementacija brojačkog semafora korišćenjem binarnih	
<p>P(C)</p> <p>//Ekskluzivni pristup</p> <p>P(S_1);</p> <p>//Umanjenje brojačke</p> <p>//promenljive</p> <p>$C--;$</p> <p>//Ako je C negativno semafor</p> <p>//nije otvoren</p> <p>IF ($C < 0$)</p> <p>//Blokira se proces</p> <p>V(S_1);</p> <p>P(S_2);</p> <p>ENDIF</p> <p>V(S_1);</p>	<p>V(C)</p> <p>//Ekskluzivni pristup</p> <p>P(S_1);</p> <p>//Uvećanje brojačke promenljive</p> <p>$C++;$</p> <p>//Ako C nije pozitivno postoje</p> <p>//procesi koji čekaju</p> <p>IF ($C \leq 0$)</p> <p>//Oslobađanje jednog procesa</p> <p>//koji čeka</p> <p>V(S_2);</p> <p>ELSE</p> <p>//Inače se oslobađa semafor</p> <p>// koji štiti ekskluzivni pristup</p> <p>V(S_1);</p> <p>ENDIF</p>

3.31 Sta su kriticni regioni i kako se implementiraju?

Kriticni regioni predstavljaju implementaciju zastite pristupa kriticnoj sekciji na visem programskom jeziku. Potrebno je definisati koju promenljivu deli vise procesa i oznakom *region* obezbediti ekskluzivan pristup toj promenljivoj u okviru niza naredbi koje se nalaze u okviru tog regiona.

3.32 U cemu je prednost kriticnih regiona u odnosu na semafore?

Kriticni regioni garantuju da, dok se izvrsavaju naredbe u okviru njih, nijedan drugi proces ne moze da im pristupa ukoliko su pridruzeni istoj deljenoj promenljivoj.

3.33 Na koji nacin se resava problem kriticne sekcije kriticnim regionima?

Deljenoj promenljivoj se moze pristupati samo u okviru nekog od regiona gde je ona zasticena.

3.34 Kako se resava problem sinhronizacije procesa koriscenjem kriticnih regiona?

Problem sinhronizacije se resava uslovnim kriticnim regionima i dopustanjem da se privremeno prekine rad u kriticnoj sekciji i proveri neki uslov koriscenjem naredbe AWAIT.

3.35 Sta su uslovni kriticni regioni?

Kriticni regioni se aktiviraju samo u slucaju kada je odredjeni logicki uslov tacan. Ako je uslov ispunjen, proces ima pravo da pristupi regionu (ako neki drugi proces vec ne pristupa deljenoj promenljivoj) i u njemu ima ekskluzivan pristup nad deljenom promenljivom. Ako logicki uslov nije ispunjen, ulazak procesa u kriticni region se odlaze dok uslov ne bude zadovoljen i u kriticnom regionu ne bude drugih procesa.

3.36 Koji su nedostaci kada je koncept kriticnih regiona u pitanju?

Imaju samo jednu proveru uslova na pocetku kriticnog regiona.

3.37 Objasniti koncept monitora.

Monitori predstavljaju najvisi nivo apstrakcije kada je zastita kriticne sekcije bez aktivnog cekanja u pitanju. Oni su konstrukcije programskih jezika u okviru kojih su implementirani mehanizmi za zastitu kriticne sekcije, ali i za sinhronizaciju. U okviru monitora u jednom trenutku moze da bude aktivan samo jedan proces cime se obezbedjuje zastita bilo kojeg dela programa.

Za potrebe sinhronizacije implementirane su specijalne uslovne promenljive. Nad njima su definisane dve operacije: wait i signal, cija sintaksa moze biti x.wait() - operacija kojom se proces blokira i x.signal() - operacija kojom se proces moze odblokirati signalom iz drugog procesa.

3.38 Ko ima prava pristupa uslovnim promenljivama monitora?

Prava pristupa uslovnim promenljivama imaju iskljucivo procedure iz monitora.

3.39 Koje su prednosti monitora u odnosu na ostale koncepte za zastitu kriticne sekcije i sinhronizaciju procesa?

Monitori pruzaju vise mogucnosti od semafora zbog nivoa na kome se implementiraju i koriste se na komplikovаниji nacin. Prednost monitora u odnosu na kriticne regije je u mehanizmima (wait i signal) koji olaksavaju sinhronizaciju.

3.40 Na kojem nivou se implementiraju monitori?

Na nivou programskih jezika.

3.41 Sta se dogadja sa poslatim signalima koje ni jedan proces ne ceka?

Ako nijedan proces ne ceka signal (nije blokiran), poslati signal ce se izgubiti.

3.42 Analizirati situaciju u monitoru koja nastaje kada jedan proces posalje signal drugome. Koji od njih treba da nastavi sa radom u monitoru?

Ako neki proces ceka signal, poslati signal ce probuditi taj proces. Jedan poslati signal moze probuditi tacno jedan proces koji na njega ceka. Probudjeni proces nastavlja sa radom u monitoru.

3.43 Na koji nacin se obicno organizuju redovi cekanja na uslovnu promenljivu monitora?

Redovi cekanja na uslovnu promenljivu se obicno organizuju po FCFS (First Come First Served) algoritmu, odnosno prednost ce imati onaj proces koji je ranije blokiran na uslovnoj promenljivoj.

3.44 Analizirati resenja problema filozofa koji veceraju koja se dobijaju koriscenjem semafora i monitora.

Semafor:

Za resavanje ovog problema potrebno je deklarisati ni semafora *S[5]*, promenljivu *mutex* za zastitu ekskluzivnog pristupa i niz *stanje[5]* koji ce za svakog filozofa cuvati informaciju o stanju u kojem se nalazi.

Monitor:

Resenje koriscenjem monitora je zasnovano na istoj ideji kao kod semafora, ali je realizacija dosta olaksana zbog mehanizama koje poseduju monitori.

Primer 3.9. Filozofi koji večeraju (koriscenjem semafora)

```
Glavni program:  
//Paralelno izvršavanje procesa  
PARALLEL WHILE (true)  
    filozof(0);  
    filozof(1);  
    filozof(2);  
    filozof(3);  
    filozof(4);  
END PARALLEL WHILE  
  
//Procesi koji simuliraju život filozofa  
filozof(i):  
    WHILE (true)  
        razmišlja();  
        uzmi_štapiće(i);  
        jede();  
        vrati_štapiće(i);  
    //Pokušaj uzimanja štapića  
    uzmi_štapiće(i):  
        P(mutex);  
        stanje[i] = GLADAN;  
        test(i);  
        V(mutex);  
        P(S[i]);  
    vrati_štapiće(i):  
        P(mutex);  
        //Pošto je večerao filozof može da razmišlja  
        stanje[i] = RAZMIŠLJA;  
        //Testiraju se susedni filozofi kako bi dobili priliku da dođu do pribora  
        test(levi);  
        test(desni);  
        V(mutex);  
    test(i):  
        IF (stanje[i] == GLADAN AND stanje[levi] != JEDE AND  
            stanje[desni] != JEDE)  
            stanje[i] = JEDE;  
            V(S[i]));  
        ENDIF
```

Primer 3.21. Filozofi koji večeraju (koriscenjem monitora)

```
filozof[5]: monitor;  
stanje[5]: {RAZMIŠLJA, GLADAN, JEDE};  
  
uzmi_štapiće(i):  
    stanje[i] = GLADAN;  
    test(i);  
    IF (stanje[i] != JEDE)  
        filozof[i].wait();  
    ENDIF  
  
vrati_štapiće(i):  
    stanje[i] = RAZMIŠLJA;  
    //Testira levog i desnog od sebe  
    test(levi);  
    test(desni);  
  
test(i):  
    IF (stanje[levi] != JEDE AND stanje[i] = GLADAN AND  
        stanje[desni] != JEDE)  
        stanje[i] = JEDE;  
        filozof[i].signal();  
    ENDIF
```

4. Zaglavljivanje

4.1 Sta se podrazumeva pod pojmom zaglavljivanje?

Zaglavljivanje (deadlock) je situacija u kojoj dva ili vise medjusobno zavisnih procesa blokirani cekaju na resurse koje nikada nece dobiti. Ovakvo zaglavljivanje je trajno i ima za posledicu zaglavljivanje celog sistema.

4.2 Navesti primere zaglavljivanja.

Ukoliko cetiri kolone automobila istovremeno sa cetiri strane udju u raskrsnicu, pri tome postujuci pravilo desne strane, blokirace jedni druge tako da ce i ostali ucesnici u saobracaju biti zaglavljeni. Ako bi svaki od pet filozofa, koji bi trebalo da veceraju, istovremeno uzeo po stanicu, doslo bi do zaglavljivanja. Nijedan od njih nece spustiti stanicu, a pri tome ce svaki cekati da to uradi filozof pored njega.

4.3 Sta je zivo blokiranje (livelock)? Ilustrovati primerima.

Zivo blokiraje je situacija u kojoj procesi nisu blokirani ali nemaju napretka u izvrsavanju.

Primeri:

Scena u kojoj u uskom prolazu treba da se mimojdju dve osobe pri cemu se obe istovremeno pomeraju u levu pa u desnu stranu pokusavajuci da propuste jedna drugu. Pri tome, osobe ne napreduju, ali ni ne stoje u mestu.

Situacija u kojoj se automobili blokiraju na raskrsnici, ali se vracaju unazad kako bi ponovo pokusali da prodju raskrsnicu. Posto ovo rade istovremeno, ponovo se blokiraju i ponavljaju postupak.

4.4 U kakvim situacijama se moze reci da je doslo do izgladnjivanja procesa?

Izgladnjivanje (starvation) se odnosi na situacije koje podrazumevaju da u sistemu generalno postoji napredak, ali da neki procesi dosta dugo ne napreduju. Obicno nije trajna situacija u sistemu.

4.5 Navesti primere izgladnjivanja procesa.

Ako su automobili koji se krecu u kolonama na putu sa prvenstvom prolaza, ostali nece imati mogucnost da udju u raskrsnicu. Kada svi automobili iz kolona, koje se nalaze na putu sa prvenstvom prolaza prodju raskrsnicu, stvorice se uslovi da i ostali automobili prodju raskrsnicu i nastave svojim putem.

4.6 U kojem stanju se nalazi proces pre nego sto se zaglavi?

Proces se pre zaglavljivanja nalazi u stanju *Blokiran*.

4.7 Koliki je najmanji broj procesa koji mogu biti zaglavljeni?

Jedan proces.

4.8 Na kojoj vrsti resursa obicno dolazi do zaglavljivanja?

Do zaglavljivanja dolazi i na hardverskim (disk, memorija, procesor...) i na softverskim resursima (semafor, fajlovi, sloganovi u bazama podataka...).

4.9 Koji uslovi treba da budu ispunjeni u sistemu kako bi doslo do zaglavljivanja?

Kofmanovi uslovi:

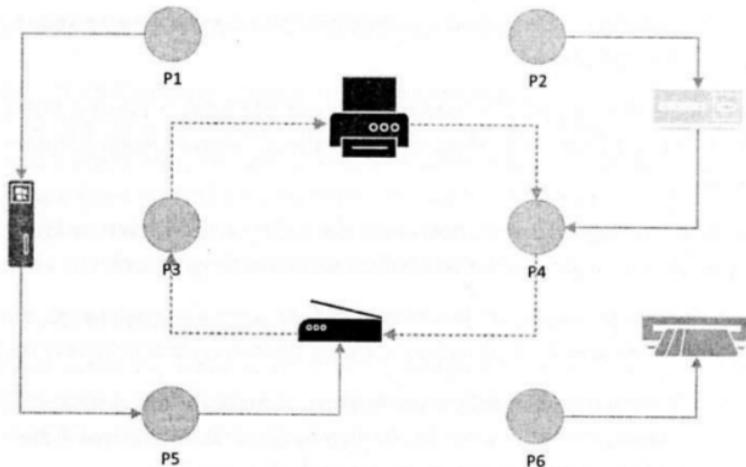
1. Uzajamno iskljucivanje – u jednom trenutku tacno jedan proces moze da koristi resurs
2. Cekanje i drzanje
3. Nemogucnost prekidanja
4. Kruzno cekanje

4.10 Sta se podrazumeva pod cekanjem i drzanjem resursa?

Dok proces drzi resurse sa kojima radi, moze da zahteva noce koji su mu potrebni za dalje izvrsavanje i da ceka one koji trenutno nisu raspolozivi.

4.11 Objasniti pojma kruznog cekanja. Ilustrovati primerima.

Podrazumeva da u sistemu moze da postoji lanac procesa koji cekaju jedni na druge tj. na resurse koji su im dodeljeni, cineći pri tome krug. Dakle, svaki proces drzi resurse koji su mu dodeljeni i ceka na druge koje drzi sledeći proces u nizu i tako do poslednjeg koji zahteva resurse koje drzi prvi proces.



4.12 Sta podrazumeva nemogucnost prekidanja?

Podrazumeva da operativni sistem nema pravo da oduzme resurse koje je dodelio procesu i kasnije mu ih vrati, kako bi te resurse, u medjuvremenu, dodelio drugim procesima. Dakle, resurse moze osloboditi samo proces koji ih je dobio na koriscenje.

4.13 Da li istovremena ispunjenost svih Kohmanovih uslova obavezno dovodi do zaglavljivanja?

Da. Do zaglavljivanja nece doci ako se bilo koji od ovih uslova ukloni.

4.14 Kakvi pristupi postoje kada je u pitanju tretman zaglavljivanja u odnosu na mere koje se primenjuju?

1. Sprecavanje – pristup koji podrazumeva sistemske mere koje se implementiraju u operativni sistem i kojima se iskljucuje mogucnost zaglavljivanja
2. Izbegavanje – dinamicke mere koje se preduzimaju kako bi se sistem vodio kroz stanja koja obezbedjuju da ne dodje do zaglavljivanja.
3. Detekcija i oporavljanje
4. Nepreduzimanje bilo cega

4.15 Kada je nepreduzimanje bilo cega, kako bi se problem zaglavljivanja resio, opravdana mera?

Nepreduzimanje bilo cega je mera koja je prihvatljiva na sistemima na kojima do zaglavljivanja retko dolazi ili u situacijama kada se ne radi o veoma vaznim uslovima. Kada do zaglavljivanja dodje, najcesce se problem resava ponovnim pokretanjem procesa ili citavog sistema.

4.16 U cemu je razlika izmedju mera za sprecavanje i mera za izbegavanje zaglavljivanja?

Izbegavanje je liberalnija mera, ali se i ovim pristupom garantuje da do zaglavljivanja nece doci.

4.17 Objasniti ideju detekcije i oporavka kada je zaglavljivanje u pitanju.

To je pristup koji dozvoljava da do zaglavljivanja dodje, ali ima mehanizme za otkrivanje da li je do zaglavljivanja doslo, tretman procesa koji su zaglavljeni i eventualno spasavanje dela rezultata koji su dobijeni pre zaglavljivanja u slucajevima kada je to moguce.

4.18 U kojem trenutku se primenjuju mere za sprecavanje zaglavljivanja?

Primenjuju se u startu definisanjem pravila za izvrsavanje procesa i eliminisu bar jedan od uslova koji dovode do zaglavljivanja.

4.19 Koji su glavni nedostaci mera za sprecavanje zaglavljivanja?

Ogranicenja su velika i ne ovom merom se ne moze ukloniti prvi Kofmanov uslov (uzajamno iskljucivanje).

4.20 Na koji nacin se moze eliminisati uslov cekanja i drzanja?

Glavna ideja za eliminisanje uslova cekanje i drzanje je uvodjenje ogranicenja koje onemogucava da proces drzi neke resurse dok trazi nove. Obicno se koriste dva pristupa za resavanje ovi problema.

4.21 Koji su nedostaci pristupa za sprecavanje zaglavljivanja koji podrazumeva da procesi pre pocetka izvrsavanja zahtevaju dodelu svih potrebnih resursa?

Losa strana ovog pristupa je to sto za vreme svog izvrsavanja proces drzi sve potrebne resurse i time onemogucava druge procese da ih koriste u trenucima kada su njemu nepotrebni.

4.22 Da li se svaki od Kohmanovih uslova moze eliminisati primenom neke metode kako ne bi doslo do zaglavljivanja?

Da.

4.23 Na koji nacin se moze eliminisati nemogucnost prekidanja?

Mere kojima se uklanja uslov nemogucnosti prekidanja se zasnivaju na ideji da se, u slucaju kada se potrebe procesa ne mogu zadovoljiti, otpustaju svi resursi koje je drzao. Kada proces drzi neke resurse a zahteva druge koji mu se ne mogu odmah dodeliti, on otpusta sve resurse koje drzi i ustupa ih drugim procesima na koriscenje. Pri tome, oduzeti resursi se dodaju na listu onih koje proces ceka.

4.24 U cemu je razlika izmedju mera koje se primenjuju za eliminisanje uslova nemogucnosti prekidanja i uslova cekanja i drzanja?

Kod prevencije cekanja i drzanja se uvodi ogranicenje koje onemogucava da proces drzi resurse dok trazi nove, dok se kod eliminisanja uslova nemogucnosti prekidanja oduzimaju resursi procesa samo ako se ne mogu zadovoljiti potrebe procesa.

4.25 Kako se moze eliminisati kruzno cekanje?

Obicno se koriste resenja oja podrazumevaju oznacavanje tj. enumeraciju tipova (klasa) resursa. Ideja je da se svakom resursu dodeli jedan broj, a da proces koji drzi odredjene resurse moze da zahteva samo one resurse kojima je dodeljen veci broj od brojeva njegovih resursa. Proces resurse nizeg ranga moze zahtevati kada otpusti sve resurse viseg ranga.

4.26 Koji su glavni problemi kada je enumeracija procesa u pitanju?

Javljam se teskoce oko numerisanja tj. dileme kako je najbolje napraviti enumeraciju. Takodje, jedan od problema je i potreba za ponovnom enumeracijom svaki put kada se u sistemu pojavi novi resurs.

4.27 Kada se primenjuju mere za izbegavanje zaglavljivanja?

Mere za izbegavanje zaglavljivanja se preduzimaju u određenim momentima u zavisnosti od stanja u sistemu.

4.28 Koja je osnovna ideja Bankarevog algoritma?

Ideja je da se dinamicki ispituje pokusaj dodele resursa kako bi se obezbedilo da nikada ne dodje do kruznog cekanja.

4.29 Koje informacije o procesima bi trebalo da budu poznate pre pocetka izvrsavanja Bankarevog algoritma?

Pre pocetka izvrsavanja, Bankarev algoritam podrazumeva da je poznat maksimalni broj resursa svakog tipa koji ce procesu biti potreban.

4.30 Na koji nacin se definise bezbedna sekvenca?

Smatra se da je stanje sistema *bezbedno* ako sistem moze u odredjenom redosledu da dodeljuje resurse kako bi svaki proces mogao da zavrssi sa radom. Redosled kojim se vrsi ovakva dodela resursa procesima se naziva *bezbedna sekvenca*.

Def: Sekvenca procesa $\langle P_1, P_2, \dots, P_n \rangle$ je *bezbedna* za tekuce stanje dodele resursa akko se potrebe svakog procesa P_i mogu zadovoljiti slobodnim resursima i resursima koje trenutno drze procesi P_j ($j < i$).

4.31 Konstruisati primere bezbednih i nebezbednih sekvenci.

Neka sistem ima ukupno 10 instanci odredjenog resursa i neka su u tabeli date informacije o zahtevima i trenutnom rasporedu dodeljenih instanci tog resursa procesima.

	<i>Maksimalne potrebe</i>	<i>Trenutno stanje</i>
P_1	7	3
P_2	5	2
P_3	9	2

Sekvenca $\langle P_2, P_1, P_3 \rangle$ je bezbedna. Sa druge strance, sekvenca $\langle P_1, P_2, P_3 \rangle$ je nebezbedna.

4.32 Kada se moze reci da je sistem u bezbednom stanju?

Sistem je u *bezbednom stanju* akko postoji bezbedna sekvenca.

4.33 Da li nebezbedno stanje sistema, u nekom trenutku, obavezno povlaci zaglavljivanje?

Ne. To znaci da postoji mogucnost da do zaglavljivanja dodje ali nije obavezno.

4.34 Koji su ulazni podaci potrebni za izvrsavanje Bankarevog algoritma?

Potrebni ulazni podaci: broj procesa - n , broj resursa - m , broj raspolozivih resursa odredjenog tipa - $raspolozivo[m]$, maksimalni zahtevi svakog procesa - $max[n][m]$, broj resursa svakog tipa koji je trenutno dodeljen svakom procesu - $dodeljeno[n][m]$, preostale potrebe svakog procesa za resursima - $potrebno[n][m]$ ($potrebno[i,j] = max[i, j] - dodeljeno[i,j]$).

Potrebno je definisati i nejednakost dva niza iste dimenzije: $A \leq B$, ako $\forall i \in \{1, 2, \dots, n\}, A[i] \leq B[i]$

4.35 Opisati Bankarev algoritam.

Algoritam 4.1. Provera stanja

```
//Inicijalizacija  
radni = raspoloživo;  
gotov[i] = 0, za i = 0, 1, ..., n - 1;  
sledeći_proces_u_sekvenci:  
//Provera da li postoji proces koji može biti zadovoljen raspoloživim  
//resursima  
IF (ne postoji i takvo da je gotov[i] == 0 i potrebno[i] ≤ radni)  
    idи na kraj;  
ENDIF  
//Pronalazi se proces koji može biti zadovoljen raspoloživim resursima  
Odaberi prvo i takvo da je gotov[i] == 0 i potrebno[i] ≤ radni;  
//Radnom skupu se dodaju resursi procesa koji ima mogućnost da se izvrši  
radni = radni + dodeljeno[i];  
gotov[i] = 1;  
idi na sledeći_proces_u_sekvenci;  
kraj:  
//Ako su svi procesi imali mogućnost sa se izvrše – stanje je bezbedno  
IF (gotov[i] == 1 za svako i)  
    bezbedno_stanje = true;  
ELSE  
    bezbedno_stanje = false;  
ENDIF
```

Niz *radni* se inicijalizuje tako da bude jednak nizu raspolozivih resursa u prvom trenutku izvrsavanja. Niz *gotov*, koji nosi informaciju o tome da li *i*-ti proces ima uslova da se izvrsi sa raspolozivim resursima, se inicijalizuje na nulu. U koraku *sledeći_proces_u_sekvenci* se trazi proces koji nije imao uslove da se izvrsi (*gotov[i] == 0*), a u trenutnom radnom nizu postoje raspolozivi resursi koji su mu potrebni da bi se zavrsio. Ako takav proces postoji, u sledecem koraku se niz *radni* prosiruje sa resursima koje je *i*-ti proces drzao jer se prepostavlja da on moze da zavrsi sa radom i oslobodi dodeljene i svoje resurse. Vrednost *gotov[i]* postaje 1 i ponovo se, sa vecim radnim nizom, trazi proces koji moze biti zadovoljen. Ako u koraku *kraj* ne postoji proces koji zadovoljava postavljene uslove, to znaci da je algoritam zavrsio sa radom i da treba proveriti da li je sistem u bezbednom stanju ili ne. Ako se prolazom kroz niz *gotov* utvrdi da su svi procesi imali uslove da dobiju potrebne resurse, iskoriste ih i vrate, onda je sistem u bezbednom stanju. Kada sistem dobije zahtev za resursima od nekog procesa, on pokrece Bankarev algoritam kako bi doneo odluku da li ve dodeliti resurse procesu, odloziti ili odbiti zahtev.

Algoritam 4.2. Dodela resursa

```
//Provera da li su zahtevi u skladu sa najavljenim pre početka izvršavanja  
IF (potrebno[i] < zahtev[i])  
    Došlo je do greške jer proces prevazilazi svoje prethodno definisane  
    maksimalne zahteve;  
ENDIF  
IF (raspoloživo < zahtev[i])  
    Resursi nisu raspoloživi pa će Pi morati da sačeka;  
ENDIF  
//Sistem pokušava da dodeli tražene resurse procesu Pi modifikujući  
//stanje na sledeći način:  
raspoloživo = raspoloživo - zahtev[i];  
dodeljeno[i] = dodeljeno[i] + zahtev[i];  
potrebno[i] = potrebno[i] - zahtev[i];  
IF (provera stanja sistema je bezbedna)  
    transakcija je završena, a proces Pi je dobio svoje resurse;  
ELSE  
    Pi će morati da sačeka zahtevane resurse iz zahtev[i] a sistem se vraća na  
    staro stanje dodele resursa;  
ENDIF
```

4.36 Na kojim principima funkcionisu mere za detekciju i otklanjanje zaglavljivanja?

Dozvoljava da do zaglavljivanja dodje, ali ima mehanizme za otkrivanje da li je do njega doslo, tretman zaglavljenih procesa i eventualno spasavanje dela rezultata.

4.37 Sta se podrazumeva pod detekcijom zaglavanja?

Detekcija zaglavljivanja je koncept koji ne garantuje da do zaglavljivanja nece doći, vec dopusta da do njega dodje, a onda se primenjuju mere da se takva situacija otkloni.

4.38 Kakvi pristupi postoje kada je ucestalost provere da li je do zaglavljivanja doslo u pitanju?

Postoje aktivan – koriscenjem algoritma za detekciju se proverava da li je doslo do zaglavljivanja i pasivan pristup – provera se vrši u situacijama kada se registruje da u sistemu nema aktivnosti ili da nesto nije u redu.

4.39 Od cega zavisi ucestalost provere da li je u sistemu doslo do zaglavljivanja?

Ucestalost provere zavisi od frekvencije zaglavljivanja i može varirati na razlicitim sistemima.

4.40 Na koji nacin se moze modelovati stanje u sistemu kada su u pitanju procesi i resursi?

Stanje sistema se može modelovati koriscenjem grafova. Cvorovi grafova su procesi (krugovi) i resursi (pravougaonici). Resursi sa vise instanci se predstavljaju sa vise tacaka u pravougaoniku. Grane grafova su oznake da li neki proces drži neki resurs (strelica od resursa ka procesu) i da li neki proces zahteva resurs (strelica od procesa ka resursu).

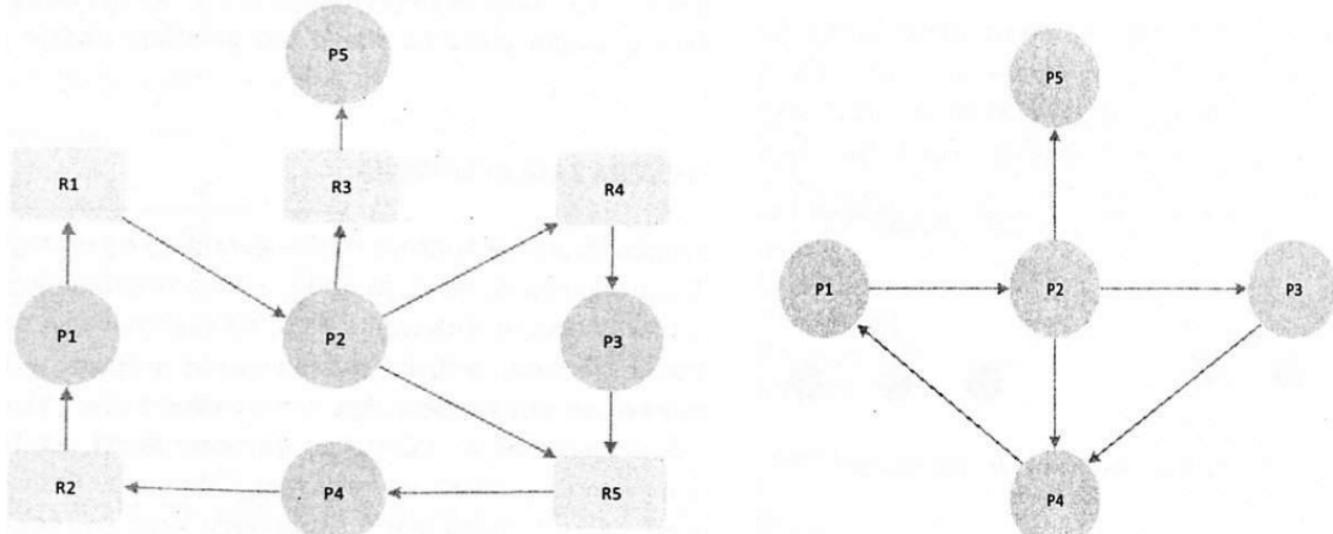
4.41 Na sta se svodi detekcija zaglavljivanja u sistemima u kojima svi resursi imaju po jednu instancu?

Svodi se na pronađenje ciklusa u grafu. Graf koji se posmatra nastaje kada se iz grafa koji opisuje stanje sistema uklone resursi, a grane povezu procese koji čekaju jedni na druge.

4.42 Da li ciklus u grafu obavezno povlaci da u sistemu postoji zaglavljivanje?

Da, ako su u pitanju svi resursi sa po jednom instancom.

4.43 Konstruisati primere sistema koji imaju cikluse u grafu koji modeluje stanje sistema.



4.44 Koja je slozenost algoritma za detekciju zaglavljivanja u sistemima u kojima svi resursi imaju po jednu instancu?

Slozenost ovog algoritma je $O(n)$, pri cemu je n broj grana grafa.

4.45 Koja je osnovna ideja algoritma za otkrivanje zaglavljivanja u sistemima sa vise instanci istih resursa?

Algoritam za otkrivanje zaglavljivanja se zasniva na traženju procesa koji mogu da budu zadovoljeni sa raspolozivim resursima. Algoritam je slican algoritmu za proveru da li je stanje sistema bezbedno.

4.46 Koje se metode primenjuju za otklanjanje zaglavljivanja?

Suspendovanje blokiranih procesa tj. prekidanje jednog ili vise zaglavljenih procesa ili oduzimanje resursa zaglavljenim procesima.

4.47 Koji su kriterijumi za odabir procesa koji ce biti prekinut kada je algoritam za otklanjanje zaglavljivanja prekidanjem izabralih procesa u pitanju?

- Prioritet procesa – trebalo bi prvo prekinuti procese nizeg prioriteta
- Koliko se dugo proces izvrsava i koliko ima do zavrsetka – ideja je da se prekidaju procesi koji su najkasnije poceli da se izvrsavaju, a da se prekidanje onih koji su pri kraju odlaze do poslednjeg trenutka
- Koje resurse proces koristi i koji su mu potrebni da zavrssi sa radom – tezi se da se prekidaju procesi koji su veliki potreosaci resursa ili procesi koji drže resurse koji su potrebni mnogim drugim procesima

4.48 Na cemu se zasniva pristup za otklanjanje blokiranja oduzimanjem resursa?

Zasniva se na ideju da se odredjenim procesima oduzmu resursi kako bi se uz pomoc tih resursa otklonilo zaglavljivanje.

4.49 O cemu treba voditi racuna prilikom oduzimanja resursa procesima?

Treba voditi racuna o tome da se takvi procesi ostave u stanju koje je pogodno da kasnije mogu nastaviti sa radom.

4.50 Da li za otklanjanje zaglavljivanja oduzimanjem resursa procesima, resurse iskljucivo treba oduzimati zaglavljenim procesima?

Ne. Ponekad oduzimanje resursa procesu koji nije zaglavljen može dovesti do oslobođenja resursa koji mogu omogućiti da se sistem ne zaglavi.

5. Upravljanje memorijom

5.1 Sta je memorija?

Memorija predstavlja jedan od osnovnih resursa racunarskog sistema. Svaka procesorska instrukcija se mora naci u memoriji, a za izvrsavanje vecine instrukcija potrebni su podaci iz memorije koji se moraju donesti u procesor ili iz procesora prebaciti u memoriju.

Memorija je bilo koji fizicki uredjaj koji moze privremeno ili trajno da cuva podatke.

5.2 Kako se na osnovu nacina na koji se cuvaju podaci mogu podeliti memorije?

Mogu se podeliti na memorije koje privremeno cuvaju podatke (dok je racunar ukljucen) i memorije koje imaju mogucnost trajnog cuvanja podataka.

5.3 Kako se na osnovu brzine pristupa mogu podeliti memorije?

Mogu se podeliti na cetiri grupe: registri, kes, primarna i sekundarna memorija.

5.4 Koje su aktuelne karakteristike memorije?

Danasnji racunari imaju nekoliko hiljada puta vise memorije od najmocnijih racunara iz 60ih godina.

5.5 Sta su registri?

Registri su memorejske celije ugradjene u procesor i u njima se nalaze podaci koje procesor trenutno obradjuje. Registri su memorija koja privremeno cuva podatke.

5.6 Koja je osnovna uloga kes memorije?

Kes memorija je veoma brza memorija koja takodje cuva podatke privremeno i predstavlja bafer izmedju primarne memorije i procesora. Obicno sadrzi delove podataka za koje se predpostavlja da ce ih procesor cesto koristiti. Nalazi se u samom procesoru, a delovi se mogu naci i van njega.

5.7 Sta se nalazi u primarnoj (glavnoj) memoriji?

Glavna memorija sadrzi instrukcije i podatke sa kojima procesor trenutno operise (radna memorija).

5.8 Koje su glavne odlike RAM memorije?

RAM (Random Access Memory) je posebna vrsta glavne memorije. Procesor moze direktno da pristupi bilo kom delu RAM-a, pri cemu vreme pristupa ne zavisi od lokacije i sadrzaja podataka. RAM je memorija koja privremeno cuva podatke i bez koje racunar ne moze da funkcioniše.

5.9 Sta je ROM memorija?

ROM (Read Only Memory) je vrsta glavne memorije. ROM cuva podatke kada je racunar iskljucen i moguce je samo citanje podataka iz ove memorije.

5.10 Sta se nalazi u ROM memorije?

Sadrzi kriticne programe koji sluze za pokretanje operativnog sistema.

5.11 Koja je uloga sekundarne (spoljasnje) memorije?

Sekundarna memorija ne gubi sadrzaj nakon prestanka rada racunara, pa se zbog toga koristi za trajno cuvanje podataka. Da bi procesor pristupio podacima iz sekundarne memorije, najpre je potrebno da se oni ucitaju u radnu memoriju. Nije neophodna za rad racunara.

5.12 Navesti primere sekundarnih memorija i njihove karakteristike.

Primeri sekundarne memorije su hard disk, CD-ROM, DVD itd. Vreme pristupa iz ove memorije zavisi od njihove lokacije.

5.13 Od cega se sastoji memorija?

Memorija se sastoji od niza memorijskih celija koje mogu da sacuvaju najmanju kolicinu podataka – bit. Bitovi se organizuju u bajtove – grupe od po 8 bitova.

5.14 Sta je memorijska rec?

Memorijska rec odredjuje kolicinu podataka koje procesor moze da obradi u jednom trenutku. Njena velicina je odredjena arhitekturom konkretnog racunara.

5.15 Sta su simbolische, a sta fizicke adrese?

U trenutku pisanja programa nije poznato u kom delu memorije ce program biti smesten. Zato se koriste simbolische adrese, koje odgovaraju imenima promenljivih u programskim jezicima. Kada se program ucita u memoriju on mora da radi sa realnim fizickim adresama.

5.16 Kada se moze obaviti povezivanje logickih i fizickih adresa?

Proces prevodjenja simbolickih u fizicke adrese se naziva povezivanje adresa. Moze se obaviti u razlicitim trenucima. Ako je poznato u kom delu memorije ce se ucitati, povezivanje moze da se obavi u tom trenutku. Inace, kompilator generise relativne adrese u odnosu na pocetak dela memorije koja se dodeljuje procesu.

5.17 Objasniti pojma relativne adrese.

Kompilator generise relativne adrese u odnosu na pocetak dela memorije koja se dodeljuje procesu. Ove adrese se jos nazivaju i relokabilne adrese. Kompilator prevodi izvorni kod u objektni modul, a program se moze sastojati iz vise objektnih modula. Linker povezuje sve objektne module u izvrsni program. Prilikom ucitavanja izvrsnog programa, punilac (loader) preslikava relativne adrese u fizicke.

5.18 U cemu se sastoji interakcija procesora i memorije tokom izvrsavanja procesa?

Tokom izvrsavanja programa, interakcija sa memorijom se odvija kroz niz citanja ili pisanja u lokacije koje se nalaze na odredjenoj adresi. Procesor uzima instrukcije i podatke iz memorije ili smesta podatke u memoriju. Procesor tada manipulise logicke adresama koje sam generise. Skup svih logickih adresa naziva se logicki ili virtualni adresni prostor. Svakoj logickoj adresi odgovara fizicka adresa.

5.19 Sta je fizicki adresni prostor?

Fizicki adresni prostor je skup svih fizickih adresa koje odgovaraju adresama logicnog adresnog prostora.

5.20 U kojim situacijama se logicki i fizicki adresni prostori podudaraju?

Podudaraju se kada se povezivanje adresa obavlja za vreme prevodjenja i punjenja programa.

5.21 Koja je uloga jedinice za upravljanje memorijom (MMU uredjaja)?

Uloga MMU (Memory-Management Unit) uredjaja je da preslikava logicke u fizicke adrese. Moze se predstaviti kao uredjaj koji poseduje poseban register koji se naziva bazni register. MMU dodaje vrednost baznog registra na logicku adresu i tako generise fizicku adresu.

5.22 Koji su glavni uslovi koje bi operativni sistemi trebalo da ispune prilikom upravljanja memorijom?

1. Svaki proces mora imati dovoljno memorije za izvrsavanje i ne sme pristupati memoriji nekog drugog procesa kao sto drugi proces ne sme pristupati njegovoj memoriji
2. Razlicite vrste memorija unutar racunarskog sistema moraju biti koriscene tako da se svaki proces izvrsava najefikasnije moguce

5.23 Objasniti koncept monoprogramiranja.

Monoprogramiranje je najjednostavniji pristup za upravljanje memorijom koji podrazumeva da se u memoriji izvrsava samo jedan korisnicki proces. Jedan deo memorije se odvaja za operativni sistem, dok ostatak memorije koristi proces koji se izvrsava. Postoje razlicite organizacije memorije:

- operativni sistem se ucitava na nize adrese, a procesor koristi vise adrese memorije
- operativni sistem se nalazi na vrhu memorije, a nize adrese koristi procesor
- deo operativnog sistema se ucitava na nize adrese memorije, drajveri se ucitavaju na adrese koje se nalaze na vrhu memorije, a ostatak sluzi za korisnicki program

Ucitavanje operativnog sistema u memoriju se izvrsava prilikom ukljucivanja sistema. Tada se aktivira punilac (bootstrap loader) koji ima zadatak da prenese operativni sistem iz sekundarne memorije u radnu memoriju. Po ucitavanju operativnog sistema, proces se puni na adrese odmah posle operativnog sistema ili na drugi kraj memorije.

5.24 Na koji nacin se obezbedjuje zastita koda i podataka kod monoprogramiranja?

Ovakvu vrstu zastite obicno pruza hardver, a najjednostavniji pristup podrazumeva da se svaka adresa koju generise korisnicki program poredi sa sadrzajem zastitnog registra. Ako je ona manja od sadrzaja zastitnog registra, generise se prekid (pogresna adresa) i operativni sistem preduzima odgovarajuce akcije. Obicno je reakcija na takvu vrstu greske da se prekine korisnicki program uz izdavanje odgovarajuce poruke o problemu.

5.25 Na cemu se zasniva tehnika prebacivanja za upravljanje memorijom?

Prebacivanje je jednostavna tehnika upravljanja memorijom (i procesima) koju operativni sistem moze koristiti u cilju povecanja iskoriscenosti procesora. Ideja je da se blokirani proces privremeno premesti u sekundarnu memoriju (disk), a da se u memoriju ucita drugi proces. Na ovaj nacin se formira red provremeno blokiranih procesa. Procesi iz reda blokiranih, koji su u mogucnosti da nastave sa radom, formiraju red spremnih procesa. Nakon prebacivanja blokiranog procesa u sekundarnu memoriju operativni sistem odlucuje da li ce ucitati novi program u memoriju ili ce aktivirati neki iz reda spremnih procesa.

5.26 Na koji nacin se moze poboljsati efikasnost tehnike prebacivanja?

Nepredniji pristup podrazumeva da se efikasnost sistema poboljsa preklapanjem prebacivanja i izvrsavanja procesa. Ideja je da se preklopi prebacivanje jednog procesa sa izvrsavanjem drugog. Potrebno je rezervisati dva memorijska regiona (bafera) koji mogu prihvchati dve memorijske slike procesa.

5.27 Koje su glavne odlike tehnike particonisanja memorije?

Memorija se deli na n delova (particija) koje predstavljaju neprekidne delove memorije. U svaku particiju se moze smestiti po jedan proces. Stepen multiprogramiranja je n . Ako su sve particije zauzete, a neki program treba da zapocne izvrsavanje, onda se formira red spremnih procesa. Kada se neka od zauzetih particija osloboodi, ucitava se neki od spremnih procesa.

5.28 Od cega zavisi stepen multiprogramiranja u sistemima koji podrzavaju particionisanje memorije?

Zavisi od broja delova na koje se memorija deli.

5.29 Na koji nacin se realizuje zastita particija od nezeljenih pristupa?

Zastita particije od pristupa drugih procesa se realizuje hardverski, pomocu dva registra za svaku particiju. U zavisnosti od implementacije, ti registri mogu sadrzati fizicke adrese pocetka i kraja particije ali i adresu pocetka i velicinu particije. Tako je svakom procesu ogranicen pristup samo na adrese koje odgovaraju njegovoj particiji. Ako proces pokusa da oristupi nekoj nedozvoljenoj adresi, operativni sistem preduzima odredjene akcije.

5.30 Kakvi pristupi postoje kada je particionisanje memorije u pitanju?

Postoje staticki i dinamicki pristup.

5.31 Koje su odlike statickih particija?

Staticki pristup podrazumeva da se velicina particije ne menja u toku rada. Radna memorija se staticki deli na fiksni broj particija, gde svaka particija moze sadrzati proces.

5.32 Od cega zavisi broj statickih particija u memoriji?

Broj particija zavisi od razlicitih faktora: prosecna velicina procesa, velicina radne memorije, brzina procesora itd.

5.33 Kakve strategije postoje kada je u pitanju izbor particije u koju ce odredjeni proces biti smesten?

Prva strategija podrazumeva da svaka particija ima svoj red poslova u koji se smestaju procesi ciji memorijski zahtevi odgovaraju velicini particije.

Druga strategija je da se svi poslovi smestaju u jedan red, a da planer donosi odluku koji je sledeci proces koji bi trebalo da se izvrsi. Tada se ceka da se particija odgovarajuce velicine osloboodi.

5.34 Koje su odlike dinamickog pristupa kod particionisanja memorije?

Dinamicki pristup kod particionisanja memorije je slican kao i staticki, osim toga sto velicine particija nisu fiksirane. Sistemi koji podrzavaju dinamicke particije moraju voditi racuna o slobodnoj i zauzetoj memoriji. Kada se pokrene proces koji je prvi predvidjen za izvrsavanje, slobodna particija se deli na dva dela. Vremenom neki procesi zavrsavaju svoje izvrsavanje ostavljajuci pri tome prazne particije koje su koristili.

5.35 Na koji nacin se obicno vodi evidencija o slobodnim i zauzetim delovima memorije kada je dinamicki pristup particionisanja memorije u pitanju?

Evidencija se vodi koriscenjem bit-mapa ili povezanih listi.

Bit-mape podrazumevaju da se memorija podeli na jednake delove i da se svakom od tih delova pridruzi po jedan bit. Ovaj bit ima vrednost 1 ako je odgovarajuci deo memorije zauzet, a inace 0. Za punjenje procesa u memoriju potrebno je pronaci dovoljno veliki niz nula.

Povezana lista se sastoji od cvorova, pri cemu svaki cvor sadrzi sledece podatke: da li je odgovarajuci deo na koji se cvor odnosi smesten proces ili je u pitanju slobodna memorija; pocetak odgovarajuceg dela memorije; duzina tog dela memorije; pokazivac na sledeci cvor koji cuva informacije o narednom delu memorije.

5.36 Sta je fragmentacija u sistemima sa particijama?

Fragmentacija nastaje kada u racunarskom sistemu postoje delovi memorije koji su slobodni, ali ih sistem ne moze iskoristiti.

5.37 Kakvi tipovi fragmentacije postoje?

Postoje interna i eksterna fragmentacija. Interna se odnosi na prostor koji ostaje neiskoriscen kada se proces smesti u deo memorije veci od njegovih memorijskih potreba. Eksterna fragmentacija podrazumeva da u sistemu postoje delovi memorije koji su slobodni ali nisu dovoljno veliki da se u njih smesti neki proces.

5.38 Koji tipovi fragmentacije se javljaju kod statickih, a koji kod dinamickih particija?

Interna fragmentacija se skoro sigurno javlja kod statickog particionisanja. Kod dinamickih particija moze nastati eksterna fragmentacija. Eksterna fragmentacija se moze javiti i kod statickih particija kada ni jedna slobodna particija nije dovoljno velika za izvrsavanje procesa.

5.39 Sta je kompakcija?

Kompakcija je jedan od nacina da se resi problem eksterne fragmentacije. Podrazumeva da operativni sistem zaustavi procese i da se izvrsi realokacija procesa tako da se oni sabiju na pocetak ili kraj memorije. Pri tome slobodna memorija se grupise na drugi kraj memorije i formira jedinstvenu particiju.

5.40 Koji nacini za dodelu raspolozive memorije procesu postoje?

Postoje: prvo poklapanje; sledece poklapanje; slucajno rasporedjivanje; najbolje poklapanje i najgore poklapanje.

5.41 Prodiskutovati pristup prvo poklapanje za dodeljivanje memorije i njegovu modifikaciju – sledece poklapanje.

Prvo poklapanje podrazumeva da se proces smesti u prvu slobodnu zonu koja je dovoljno velika da on moze da stane u nju. Pretraga kreće od pocetka memorije.

Sledece poklapanje podrazumeva da pretraga za slobodnom memorijom ne kreće svaki put od pocetka, vec da se nastavi od lokacije gde je prethodna pretraga stala.

5.42 Analizirati slucajno (random) rasporedjivanje.

Slucajno poklapanje podrazumeva da se od svih slobodnih i dovoljno velikih zona za smestanje procesa u memoriji slucajno izabere ona u koju ce se smestiti proces.

5.43 Prodiskutovati pristup najbolje poklapanje za smestanje procesa u memoriju.

Najbolje poklapanje je pristup zasnovan na ideju da se proces smesti u slobodni deo memorije koji je najmanji od svih slobodnih delova koji su veci od potreba procesa. Potrebno je proci kroz celu memoriju, izracunati velicinu svih slobodnih regiona i pronaci najmanji koji je veci od memorijskih zahteva koje ima proces.

5.44 Analizirati pristup najgore poklapanje kada je smestanje procesa u memoriju u pitanju.

Najgore poklapanje je pristup koji se zasniva na ideji da se proces smesti u deo memorije u koji se najgore uklapa tj. u region sa najvecim kapacitetom.

5.45 Sta je glavna novina koju je stranicenje donelo kada je upravljanje memorijom u pitanju?

Stranicenje dozvoljava da memorija dodeljena procesu ne bude alocirana u jednom komadu tj. iskljucivo na adresama koje se nalaze jedna za drugom. Omogucava se da proces dobije memoriju bez obzira gde se ona nalazi ukoliko je slobodna.

5.46 Od cega se sastoji logicka adresa kada je stranicenje u pitanju?

Za vreme izvrsavanja programa procesor radi sa logickim adresama. Logicka adresa se sastoji iz dva dela: broja stranice – p i pozicije u okviru stranice – d .

5.47 Sta je tabela stranica?

Tabela stranica je posebna struktura kojom se logicke adrese prevode u fizicke.

5.48 Na koji nacin se logicka adresa prevodi u fizicku kod stranicenja?

Pomocu tabela stranica. Operativni sistem generise po jednu tabelu stranica za svaki proces koji se izvrsava. Tabela stranica svakog procesa sadrzi podatke o tome u kojim okvirima u memoriji su smestene njegove stranice. Fizicka adresa se dobija na osnovu ovih podataka dodavanjem vrednosti pozicije u stranici (d) na adresu okvira u kojem se nalazi odgovarajuca stranica.

5.49 Navesti prednosti koje donosi velicina stranice koja je jednaka stepenu broja 2 kada je stranicenje u pitanju.

Velicina stranice koja je jednaka nekom stepenu broja 2 omogucava da relativne i logicke adrese budu jednake. Ako je velicina stranice 2^n , tada nizih n bitova relativne adrese predstavljaju poziciju u okviru stranice, dok visi bitovi relativne adrese odgovaraju rednom broju stranice. Velicina stranice koja je jednaka stepenu broja 2 omogucava i da se preslikavanje logickih u fizicke adrese izvede na jednostavan nacin.

5.50 Na koji nacin se pokrece novi proces na sistemima koji podrzavaju stranicenje?

Kada proces zeli da krene sa izvrsavanjem, planer izracunava njegovu velicinu izrazenu u stranicama i proverava raspolozive memorejske okvire. Za izvrsavanje procesa koji zahteva n stranica potrebno je da n stranicnih okvira bude slobodno.

5.51 Kakva je situacija sa fragmentacijom kada je stranicenje u pitanju?

Kod stranicenja nema eksterne fragmentacije jer svaki slobodni okvir moze da bude dodeljen. Skoro sigurno postoji interna fragmentacija jer obicno poslednji okvir dodeljen procesu ne bude potpuno popunjen.

5.52 Kakva je hardverska podrska potrebna da bi se realizovalo stranicenje?

U najjednostavnijem slucaju tabela stranica se cuva u skupu registara. Koriscenje registara za cuvanje tabele stranica je pogodno samo u situacijama kada je tabela relativno male velicine. Kada je tabela stranica veca, ona se cuva u memoriji, a na njenu lokaciju u memoriji pokazuje *bazni register tabele stranica – PTBR*.

5.53 Sta je asocijativna memorija?

Asocijativna memorija je brza memorija koja se koristi za implementaciju efikasne tabele stranica. On je u stvari kes memorija koja je posebno dizajnirana za potrebe stranicenja.

5.54 Analizirati prednosti koje donosi koriscenje asocijativne memorije za stranicenje.

Asocijativna memorija je veceg kapaciteta, pa se deo tabele stranica moze smestiti u nju. Cuva podatke u parovima (*kljuc, vrednost*), a pretraga se vrsti odjednom nad svim kljucevima.

5.55 Sta se podrazumeva pod pojmom nivo pogotka (hit ratio) kada je stranicenje u pitanju?

Nivo pogotka je procenat uspesnog pronalaska zeljenih podataka u asocijativnoj memoriji.

5.56 Koja je osnovna ideja segmentacije?

Osnovna ideja segmentacije je da svakom segmentu dodeli poseban memorijski prostor. Svaki segment je atomican tj. ili ce se ceo segment naci u memoriji ili nece uopste biti ucitan u memoriju. Segmenti se mogu naci bilo gde u memoriji, ali jedan segment mora biti u neprekidnom memorijskom bloku.

5.57 Od cega se sastoji logicka adresa kada je segmentacija u pitanju?

Logicke adrese kod segmentacije se sastoje iz dva dela: broja segmenata s i pozicije u segmentu d .

5.58 Prodiskutovati prednosti i mane segmentacije.

Prednost segmentacije se ogleda u zastiti memorije. Verovatno je da ce se sve stavke segmenta koristiti na isti nacin. Ocekivano je da neki segmenti sadrze instrukcije, a neki podatke. Segmenti sa instrukcijama se mogu označiti tako da se mogu samo citati, cime se stite od nezeljenih promena i pristupa.

Segmentacija pruza mogucnost deljenja koda i podataka izmedju razlicitih procesa.

Segmenti su najcesce razlicite velicine. Potrebno je naci i dodeliti memoriju za sve segmente korisnickog programa. Dodela memorije predstavlja problem dinamicke dodele memorije.

5.59 Prodiskutovati pristupe koji kombinuju koncepte segmentacije i stranicenja.

Segmentacija sa stranicenjem predstavlja pristup koji podrazumeva podelu segmenata na stranice cime se omogucava da segmenti ne moraju da se smestaju u neprekidne memorijske blokove. Na ovaj nacin veliki segmenti se lakse smestaju u memoriju i smanjuje se eksterna fragmentacija.

6. Virtuelna memorija

6.1 Objasniti osnovnu ideju koncepta virtuelne memorije.

Virtuelna memorija predstavlja nacin upravljanja memorijom koji omogucava da se procesu na raspolaganje stavi memorija koja je drugacije velicine od one koja stvarno (fizicki) postoji u sistemu. Pri tome operativni sistem je zaduzen da omoguci preslikavanje virtuelne memorije u fizicku memoriju.

Virtuelna memorija je memorijski model koji razdvaja memorijski prostor koji je dostupan programeru, odnosno procesu za izvrsavanje, od fizickog memorijskog prostora koji ima radna memorija.

6.2 Kakve su bile prve upotrebe koncepta virtuelne memorije?

Prve upotrebe koncepta virtuelne memorije podrazumevale su da se, za tadasnje prilike, velika radna memorija podeli na manje delove koje bi onda programeri koristili za smestanje svojih procesa (sistemi za deljenje vremena).

6.3 Za sta se obicno vezuje pojam virtuelne memorije na savremenim racunarskim sistemima?

Pojam virtuelne memorije se najcesce vezuje za tehniku koja omogucava da se izvrsava proces koji nije u potpunosti u radnoj memoriji. Positivna posledica ovog koncepta je veci stepen multiprogramiranja. Korisnicki procesi mogu biti veci od fizicke memorije.

6.4 Koje su najpoznatije implementacije virtuelne memorije?

Najpoznatije implementacije su: prekrivaci, dinamicko punjenje i stranicenje na zahtev.

6.5 Objasniti koncept prekrivaca.

Prekrivaci (overlays) predstavljaju jedan od prvih pristupa za upravljanje memorijom, zasnovanih na ideji da se omoguci izvrsavanje programa koji nisu kompletno ucitani u memoriju. Osnovni zadatak je da se identifikuju moduli programa koji su relativno nezavisni – moduli se nazivaju prekrivaci. Oni obicno sadrze delove programa koji se koriste u razlicitim vremenskim trenucima, delove koji se retko koriste, ali i one za koje postoji sansa da ne budu potrebni tokom izvrsavanja procesa.

Prekrivaci se odvajaju od glavnog dela programa i smestaju se u sekundarnu memoriju. Prolikom alociranja memorije za glavni program, rezervise se jedan slobodan deo koji je dovoljno velik za prihvatanje najveceg prekrivaca. Kada je potreban deo programa iz nekog prekrivaca, taj prekrivac se ucitava u rezervisani deo memorijskog prostora.

6.6 Sta je glavni nedostatak prekrivaca?

Najveci nedostatak ovog pristupa je to sto operaciju deljenja programa na delove precizno i dovoljno dobro moze da uradi jedino programer, odnosno ona se ne moze automatizovati. Veoma je tesko predvideti tok izvrsavanja programa.

6.7 Objasniti koncept dinamickog punjenja.

Dinamicko punjenje je zasnovano na slicnim principima na kojima se zasnivaju i prekrivaci. Osnovna ideja je da se funkcije i procedure ne smestaju u memoriju sve do trenutka dok ne budu potrebne, odnosno pozvane. Funkcije i procedure se nalaze na sekundarnoj memoriji, a glavni program se smesta u memoriju i izvrsava. Kada se pozove neka od funkcija ili procedura, sistem prvo proverava da li je ona vec u memoriji. Ako nije, vrsi se prebacivanje. Punilac se poziva da bi obavio ovo punjenje memorije, ali i azurirao odgovarajuce tabele kako bi odgovarale novom stanju u sistemu. Nakon ucitavanja, nova funkcija pocinje da se izvrsava.

6.8 U cemu je razlika izmedju prekrivaca i dinamickog punjenja?

Prednost dinamickog punjenja je to sto se funkcije koje nisu potrebne za izvrsavanje procesa nikada ne pune u memoriju.

6.9 U kojim situacijama do izrazaja dolaze prednosti koje donosi dinamicko punjenje?

Prednosti dinamickog punjenja dolaze do izrazaja u programima koji imaju dosta koda koji se odnosi na slucejeve koji se retko ili skoro nikada ne dogadjaju.

6.10 Na kojem konceptu se zasniva stranicenje na zahtev?

Zasniva se na konceptu stranicenja – deli programe na stranice. Prilikom pokretanja programa u radni memoriju se smestaju samo one stranice koje su neophodne u prvom trenutku.

6.11 Sta je potrebno za implementaciju stranicenja na zahtev?

Stranicenje na zahtev se implementira uz pomoc tabele koja sadrzi informacije o stranicama.

6.12 U cemu je glavna razlika izmedju stranicenja i stranicenja na zahtev?

Za razliku od stranicenja, stranicenje na zahtev dozvoljava da neke od stranica ne budu u radnoj memoriji tokom izvrsavanja procesa, vec da se po potrebi ucitavaju. Za potrebe odlozenog ucitavanja koristi se poseban modul koji se naziva lenji prebacivac ili pejdzer.

6.13 Sta je bit validnosti i cemu sluzi?

Tabele stranica kod stranicenja na zahtev obicno imaju i dodatnu kolonu u kojoj se cuva bit validnosti. Za stranicu se kaze da je “validna” ako se nalazi u radnoj memoriji i tada je odgovarajuci bit validnosti postavljen na 1. Ako stranica nije u memoriji, ona je “nevalidna” i odgovarajuci bit validnosti ima vrednost 0. Kada proces pokusa da pristupi nekoj stranici, proverava se ovaj bit.

6.14 Sta se podrazumeva pod pojmom promasaja stranice?

Kada stranica nije vec u radnoj memoriji a proces pokusa da joj pristupi, dolazi do promasaja stranice i prekida. Do promasaja nece doci ako proces nikada ne pokusa da pristupi stranicama koje nisu u radnoj memoriji. Na prekid se reaguje i omogucava se nastavak izvrsavanja procesa donosenjem potrebne stranice u memoriju.

6.15 Koji se koraci preduzimaju kada dodje do promasaja stranice?

Kada dodje do promasaja, prvo se vrsi provera da li trazena stranica uopste pripada procesu koji ju je trazio. Ako to nije slucaj, dolazi do prekida programa usled greske pri adresiranju. Ako stranica pripada procesu, onda je potrebno ucitati je u radnu memoriju. Ucitavanje se izvodi u tri koraka:

1. pronalazi se slobodan memorijski okvir ako postoji ili se oslobadja jedan od zauzetih
2. ucitava se stranica iz sekundarne memorije
3. modifikuje se tabela stranica tako da bit validnosti govori da je stranica u radnoj memoriji

Donosenjem potrebne stranice stvaraju se uslovi za nastavak izvrsavanja prekinutog procesa na mestu gde je doslo do promasaja stranice.

6.16 Sta se podrazumeva pod pojmom cisto stranicenje?

Jedan od specijalnih slucajeva stranicenja na zahtev je cisto stranicenje. Ovakav pristup podrazumeva da proces kreće u izvrsavanje bez i jedne ucitane stranice. Promasaji ce se redjati u pocetku sve dok se ne iskristalise radni skup stranica koje su potrebne za izvrsavanje.

6.17 Opisati algoritam za zamenu stranice.

Algoritam se moze predstaviti na sledeci nacin:

1. Naci lokaciju zeljene stranice na disku.
2. Naci slobodan okvir.
 - Ako postoji slobodan okvir, koristi njega.
 - Ako ne postoji, upotrebi algoritam za izbor okvira zrtve
 - Upisati zrtvu na disk i saglasno tome izmeniti tabelu stranica.
3. Ucitati zeljenu stranicu u novooslobodjeni okvir i azurirati tabelu stranica.
4. Nastaviti sa izvrsavanjem procesa.

6.18 Koje su mane slucajnog izbacivanja stranice iz memorije?

Najveci problem kod ovog algoritma je cinjenica da slucajno izbacivanje stranice cesto moze da dovede do izbacivanja nekih veoma koriscenih stranica, cime se doprinosi velikom broju promasaja stranica. Algoritam nije zanemarljive vremenske slozenosti jer implementacija nije trivijalna, a ne daje ni dobre rezultate.

6.19 Koji je najveci nedostatak Beladijevog pristupa za izbor stranice za izbacivanje?

Najveci problem Beladijevog optimalnog algoritma je taj sto ga je skoro nemoguce implementirati. Kada nastane promasaj stranice, operativni sistem nije u mogucnosti da zna koja stranica ce biti potrebna u sledecem koraku.

6.20 Sta je potrebno za implementaciju FIFO algoritma za izbacivanje stranica?

Potreban je jedan memorijski registar po stranici u koji bi se upisivao trenutak kada je stranica usla u memoriju, a onda bi se ta vremena sortirala. Cesce se za implementaciju koristi povezana lista u koju se smestaju redni brojevi stranica. Prilikom ucitavanja stranice njen redni broj se smesta na kraj liste, a stranica za izbacivanje se nalazi na pocetku liste.

6.21 Sta je FIFO anomalija?

FIFO anomalija je pojava da kada se poveca broj okvira za neki proces, moze se dogoditi da se poveca i broj promasaja stranica.

6.22 Konstruisati primer FIFO anomalije.

Velicina	Vreme												
	1	1	1	1	1	1	5	5	5	5	4	4	
	2	2	2	2	2	2	2	1	1	1	1	5	
		3	3	3	3	3	3	3	2	2	2	2	
			4	4	4	4	4	4	4	3	3	3	
	F	F	F	F			F	F	F	F	F	F	

Navedena sekvenca zahtevanih stranica ispoljava ponasanje FIFO anomalije kada se broj mogucih okvira poveca na cetiri. U tom slucaju FIFO algoritam dovodi do 10 promasaja stranica.

6.23 Objasniti algoritam druge sanse.

FIFO algoritam se na jednostavan nacin moze modifikovati tako da se izbegne izbacivanje veoma bitnih stranica koje se prve ucitavaju. Algoritam druge sanse to omogucava. Modifikacija podrazumeva postojanje jednog bita po stranici koji se naziva bit referisanosti (R). Bit referisanosti se postavlja na 1 kada se stranica ucitava u memoriju i kada se pristupa stranici. Potraga za stranicom za izbacivanje kreće od prve ucitane stranice. Ako je njen bit referisanosti 1, onda se on postavlja na 0, a vreme ucitavanja stranice se resetuje na tekuce vreme, tako da se stranica tretira kao poslednje ucitana (daje joj se druga sansa). Ako se ta stranica cesto koristi, njen bit referisanosti se ubrzo ponovo postati 1. Kada program naidje na stranicu sa bitom referisanosti 0, to znaci da ta stranica nije bila referisana od trenutka kada joj je data druga sansa, pa se izbacuje iz memorije.

6.24 Sta je najveci problem kada je algoritam druge sanse u pitanju?

Problem se javlja u slucaju kada su svi bitovi referisanosti postavljeni na 1. Tada ce algoritam posetiti svaku stranicu i postaviti njen bit na 0. Na kraju, pretraga za stranicom srtvom ce se vratiti na pocetak i tada ce prva posecena stranica biti izbacena.

6.25 Objasniti kako funkcione algoritam sata.

Algoritam sata je modifikacija algoritma druge sanse. Prilikom primene algoritma sata, svi memorijski okviri jednog procesa se posmatraju kao povezana kruzna lista.

6.26 Sta je potrebno za implementaciju algoritma sata?

Za njegovu implementaciju se koristi jedan pokazivac, koji u pocetku pokazuje na prvi memorijski okvir. I ovde se svakoj stranici pridruzuje bit referisanosti cija vrednost ima isto znamenje kao kod algoritma druge sanse.

6.27 Na kojoj ideji se zasniva LRU algoritam?

LRU (Least Recently Used) algoritam se zasniva na ideji da se iz memorije izbaci stranica koja je koriscena najdalje u prolosti u odnosu na ostale stranice.

6.28 Prodiskutovati slozenost LRU algoritma.

LRU algoritam je veoma zahtevan za implementaciju.

6.29 Sta je potrebno za implementaciju LRU algoritma?

Implementacija zahteva register koji bi cuvao informacije o vremenu koriscenja stranice. Ta vremena je potrebno sortirati pri svakom izbacivanju stranice.

6.30 Koja je najveca mana LRU algoritma?

Najveca mana se odnosi na cinjenicu da se nikada ne moze predvideti koja ce sledeca stranica procesu biti potrebna.

6.31 Sta je NRU algoritam?

NRU (Not Recently Used) algoritam predstavlja aproksimaciju LRU algoritma. Predlaze izbacivanje stranice koja nije skoro koriscena, pri cemu ta stranica ne mora nuzno biti najduze nekoriscena.

6.32 Koja je glavna prednost NRU algoritma u odnosu na LRU algoritam?

Implementacija NRU algoritma je jednostavnija u odnosu na LRU algoritam.

6.33 Na koji nacin se NRU algoritam moze modifikovati tako da daje bolje rezultate?

Modifikacija podrazumeva da se uvede jos jedan bit koji se naziva bit modifikacije (M). Ovaj bit je jednak nuli kada se stranica ucita, a postavlja se na 1 ako se sadrzaj stranice menja. Sa ovako definisanim bitom modifikacije i bitom referisanosti moguce su sledece njihove vrednosti (RM):

1. stranica koja nije referisana ni modifikovana – 00
2. stranica koja nije referisana a modifikovana je – 01
3. stranica je referisana a nije modifikovana – 10
4. stranica koja je referisana i modifikovana – 11

Ovim redom su stranice oznacene kao pozeljne za izbacivanje. Kombinacija 01 je moguca jer kada istekne vremenska konstanta t , resetuju se samo bitovi referisanosti, a bitovi modifikacije ostaju nepromenjeni.

6.34 Na kojoj ideji se zasniva LFU algoritam?

Osnovna ideja LFU (Least Frequently Used) algoritma je da se iz memorije izbaci najmanje cesto koriscena stranica, odnosno stranice koja od svih stranica u memoriji ima najmanju frekvenciju koriscenja.

6.35 Sta je potrebno za implementaciju LFU algoritma?

Implementacija ovog algoritma je hardverski zahtevnija od prethodnih jer treba voditi racuna o frekvenciji pojavljivanja, odnosno stalno azurirati i sortirati podatke.

6.36 Koja je najveca mana LFU algoritma?

Velika mana ovog pristupa lezi u losem tretmanu stranica koje su tek usle u sistem, ali su veoma potrebne i dosta ce se koristiti u buducnosti. Frekvencija koriscenja takvih stranica je mala, pa ce one biti kandidati za izbacivanje.

Drugi problem se javlja u slucajevima kada se pojedine stranice koriste cesto u odredjenom vremenskom intervalu, a kasnije nece biti potrebne.

6.37 Prodiskutovati MFU algoritam za izbacivanje stranice.

MFU (Most Frequently Used) algoritam podrazumeva da se iz memorije izbacuju stranice sa najvecom frekvencijom koriscenja. Zasniva se na ideji da ce u buducnosti biti potrebnije stranice koje su tek usle u sistem i imaju manju frekvenciju koriscenja.