ISPITNA PITANJA – SPPuRV

1. Pojam datoteke: predstava podataka i programa?
   Datoteka (fajl): skup slogova.
   Slog: osnovna jedinica obrađivane informacije. Slog predstavlja listu objekata (polja) informacionog karaktera.
   Polje: skup alfanumeričkih znakova.

2. Šta je asembler?
   Asembler je program koji prevodi polazni ili izvorni program napisan na asemblerskom jeziku u izvršivi mašinski program.

3. Šta je zadatak prvog a šta drugog prolaza asemblera?
   Prvi prolaz: definicija vrednosti simbola, koji se pojavljuju u programu. Dodeljuje memorijsku lokaciju svakoj instrukciji i svim literalima u programu.
   Drugi prolaz: prevođenje na mašinske instrukcije i određivanje vrednosti operanada izračunavanjem izraza koji predstavljaju vrednost operanada u instrukciji.

4. Tabela simbola?
   Definicija simbola predstavlja proces pridruživanja simbola sa brojem, koji predstavlja njegovu vrednost. Tabela simbola služi za upisivanje simbola i njima pridruženih vrednosti.

5. Tabela literala?
   Literali su specijalna vrsta simbola koji se odnose na konstante. Tabela literala sadrži adresu i binarni kod konstante koji odgovara datom literalu.

6. Tabela koda operacije?
   Ugrađena je u asembler. Ima po jednu vrstu za svaki simbolički kod operacije.
   Definiše razliku između mašinskih i pseudo operacija.

7. Koja su četiri osnovna modula asemblera?
   Modul za prvi prolaz, modul za drugi prolaz, modul za izračunavanje izraza i modul za konverziju konstanti.

8. Tok informacija u prvom i drugom prolazu?
   Ulaz u prvi prolaz je polazni program na asemblerskom jeziku, a izlazi su tabela simbola, tabela literala, kopija polaznog programa i ostale informacije i one predstavljaju ulaz u drugi prolaz. Izlaz iz drugog prolaza je izveštaj o prevođenju i datoteka prevedenog programa.

9. Šta je makroasembler?
   Makroasembler prevodi polazni program napisan na makroasemblerskom jeziku u izvršivi mašinski program.

10. Koja su potrebna proširenja asemblera radi obrade makroinstrukcija?
    Prvo: da prepozna makroinstrukciju.
    Drugo: da proširi makroinstrukciju.
    Da bi to uradio on najpre treba da pronađe i sačuva makrodefiniciju, koja odgovara makroinstrukciji. Makro definicija započinje sa MACRO, a završava se sa ENDM pseudo instrukcijom.

11. Prepoznavanje makroinstrukcija?
    Najprostiji način je dodati vrstu u tabeli koda operacije. Ova vrsta sadrži naziv makroinstrukcije i ukazivač na odgovarajuću makro definiciju. Definicije makroa se čuvaju u TABELI MAKRO DEFINICIJA.

12. Šta je lista naziva parametara?
    Ona uspostavlja vezu između fiktivnih i stvarnih parametara makro instrukcije.

13. Proširivanje makro instrukcija?
    Obavlja se u fazi prevođenja. Makro instrukcija se zamenjuje telom odgovarajuće makro definicije. Formalni parametar u telu makro definicije se zamejuje odgovarajućom vrednošću stvarnog parametra iz polja operanda makroinstrukcije koja se proširuje.
    Dalje sledi prvi prolaz asemblera.

14. Šta je kompajler, procedura i algoritam?
    Kompajler: prevodioc sa višeg programskog jezika na mašinski jezik.
    Procedura: skup operacija koje se izvršavaju u konačnom vremenu i dovode do jednoznačnog rezultata.
    Algoritam: skup operacija i skup pravila o njihovoj primeni nad polaznim podacima u cilju dobijanja rezultata.

15. Formalni sistem?
    Neinterpretativan matematički sistem, koji čine azbuka, aksiome i skup pravila zaključivanja. Formalni sistem je META JEZIK za definisanje programskih jezika. Simboli jezika objekta nazivaju se terminalnim simbolima, a simboli meta-jezika su neterminalni simboli.
    Formalni sistem definiše oblik odnosno sintaksu programskog jezika.

16. Šta je azbuka?
    AZBUKA T je konacan skup terminalnih simbola. Npr. T={a,b,c}

17. Šta je rečnik?
    Formula u azbuci (rečenica ili niz znakova) se dobija pripajanjem simbola: ab, ac...
    Skup svih konačnih reči (nizova) azbuke T* je REČNIK formiran u T.

18. Šta je jezik?
    JEZIK L: podskup rečnika, definisan pomoću jednog ili više znakova smene (smena zadatu reč zamenjuje jednom od reči u koju se ona transformiše).

19. Koje su dve vrste specificiranja jezika?
    Specifikacija jezika:
    1. Generativna: ukoliko je neku rečenicu moguće generisati uvek kada je to potrebno.
    2. Analitička: algoritam određuje da li je zadata reč važeća ili ne važeća u jeziku.

20. Formalna gramatika?
    Skup četiri elementa G (N, T, Σ, P) gde je:
    (1) N - skup neterminalnih simbola
    (2) T - skup terminalnih simbola
    (3) Σ - Pocetni simboli; Σ je element N
    (4) P - skup SMENA α -> β, gde su α i β
    elementi (NUT)* i α nije nula
    (5) N i T su disjunktni

21. Rečenična forma?
    Bilo koja reč koja može da se razvije od polaznog simbola Σ.

22. Rečenica?
    Rečenica je rečenična forma koja sadrži samo terminalne simbole.

23. Jezik definisan gramatikom?
    Jezik L koji je definisan gramatikom G:
    L(G) = {W je element T* takav da se indirektno razvija iz Σ}.

24. Kako se definiše BNF notacija (osnovni simboli)?
    Bakus-Naurova forma je notacija za izražavanje produkcije.

Svaka produkcija definiše sintaksnu klasu.
Elementi ovog jezika su:

::=	predstavlja strelicu smene
< >	ograđuje ime sintaksne klase
|	simbol "ILI", omogućava dodelu više smena.

25. Semantički opis?
Semantički opis daje smisao rečenicama koje se formiraju programskim jezikom.

26. Koje su tri komponente kompajlera?
Preprocesor, kompajler (prevodioc), interpreter.

27. Osnovne faze kompajlera?
Postoji 5 osnovnih faza u postupku preslikavanja osnovnog u jezik objekta:
1. **Leksička** analiza
2. **Sintaksna** analiza (parsing) – prepoznavanje strukture programa
3. **Optimizacija** koda
4. **Generisanje** koda (prevođenje strukture u mašinski jezik)
5. **Asembliranje**

28. Rezultat leksičke analize?
Leksičku analizu čine tri osnovna zadatka:
1. rasčlanjivanje polaznog programa (niz ASCII znakova) na niz osnovnih simbola (tokena) višeg programskog jezika,
2. formiranje tabele literala i tabele identifikatora
3. formiranje tabele uniformnih simbola.

29. Šta se popunjava u tabelama u fazi leksičke analize?
Popunjava se tabela literala, tako što se za svaki literal formira vrsta u tabeli u koju se unosi literal, osnova, skala, preciznost i druga inf. . U vrstu tabele identifikatora se unosi naziv identifikatora, dok se u tabelu uniformnih simbola unosi tip simbola i indeks unutar tabela.

30. Sintaksna analiza?
Ona prepoznaje rečenice (sintaksne konstrukcije) i interpretira značenje tih konstrukcija. Sintaksna analiza ukazuje i na uočene greške i omogućava kompajleru da nastavi pretraživanje drugih grešaka.

31. Šta je rezultat semantičke analize?
Prelazni oblik polaznog programa. Tipovi prelaznih oblika su:
1. Stablo sintaksne analize
2. Matrični prelazni oblik

32. Šta je zadatak optimizacije?
Radi nad prelaznim oblikom programa. Postoji mašinski nezavisna i mašinski zavisna optimizacija. Primer mašinski nezavisnih: eliminacija zajedničkih podizraza, prethodno izračunavanje vrednosti, uočavanje invarijantnog računanja unutar ciklusa...
Primer mašinski zavisnih: uklanjanje viška LOAD i STORE, JUMP na JUMP, itd.

33. Pridruživanje memorije?
Dodela memorije svim promenljivama, dodela memorije svim privremenim lokacijama koje su potrebne za međurezultate, dodela memorije literalima.
Pored dodele memorije obezbeđuje i inicijalizaciju odgovarajućih lokacija.

---

34. Šta je punjač?

Posebna komponenta programske podrške čiji je zadatak da puni programe na različite adrese u operativnoj memoriji.

35. Relokacija?

Relokacija programa: razmeštaj programa, počev od prve slobodne adrese.
Punjač relocira program dodavanjem konstante na svaku relokatibilnu adresu u programu.
Program na izlazu prevodioca je relakatibilan program, a izvršivi mašinski program je apsolutni.

36. Funkcije punjača?

1. Dodela memorijskog prostora programima (alkolacija);
2. Određivanje vrednosti simbolickih referenci izmedu relokatibilnih programa (povezivanje)
3. Podešavanje svih adresno osetljivih lokacija na odgovarajući dodeljeni prostor (relokacija);
4. Fizicki prenos mašinskih instrukcija i podataka u memoriju (punjenje).

37. Tipovi punjača?

- Prevedi i kreni.
- Apsolutni.
- BSS.
- Relokatibilni sa direktnim povezivanjem.
- Povezivač i punjač (apsolutne memorijske slike i editor povezivanja).
- Punjač prekrivača.
- Dinamički povezivač.

38. Prevedi i kreni?

Punjač u okviru programa prevodioca, asemblera ili kompajlera.
Prednost – vrlo su prosti.
Nedostaci: (1) nekorisno trošenje memorijskog prostora i procesorskog vremena, i (2) teško je održavati veći broj različitih segmenata.

39. Apsolutni punjač?

Asembler formira u celini prevedeni program. To označava da su adrese prvih instrukcija u programu i proceduri, definisane u vreme asembliranja.
Apsolutni punjač jednostavno unosi sa spoljne memorije program u definisane memorijske lokacije.

40. Koji su nedostaci apsolutnog punjača?

- Negira suštinu asemblera
- Nemoguće je rešiti problem biblioteka ili programa koji se sastoji od više nezavisnih modula.
- Ove nedostatke eliminiše relokatibilan punjač.

41. Šta je segment?

Fundamentalni koncept relokatibilnih punjača vezan je za pojam segmenta.
Segment je skup kontinualnih reči poređanih jedna za drugom.
Segment je **najmanja jedinica** za smeštanje programa ili podataka koju prevodioc i punjač prepoznaju.

42. Koja su dva tipa povezivanja?

Lokalno i međusegmentno.

43. Kako se klasifikuju simboli u proceduri?

Klasifikuju se kao lokalni i globalni.

44. BSS punjač i njegovi nedostaci?

Omogućava punjenje više segmenata programa nad samo jednim zajedničkim segmentom podataka.

Izlaz iz asemblera je prevedeni program i informacija o svim drugim programima na koje se on poziva. Navodi se i informacija o tome, koje reference treba menjati (relokaciona informacija).
Nedostaci:
Vektor prelaza zauzima prostor u memoriji, a koristi se samo za povezivanje.
Obrađuju segmente procedura, ali ne olakšavaju pristup segmentima deljivih podataka.

45. RELOKATIBILNI PUNJAČ sa direktnim povezivanjem?
Univerzalni relokatibilni punjač.
Omogućuje punjenje višestrukih segmenata procedure i višestrukih segmenata podataka.

46. Koja tri tipa tablica izdaje program prevodilac?
1) Rečnik eksternih simbola (ESD) sadrži informaciju o svim ulaznim simbolima i svim izlaznim simbolima.
2) Tabela teksta (TXT) sadrži prevedenu verziju relokatibilnog mašinskog programa.
3) Relokacioni katalog povezivanja (RLD) sadrži informaciju o svim adresno osetljivim lokacijama unutar programa.

47. Koja je uloga punjača u prvoj a koja u drugoj fazi punjenja?
Glavna uloga punjača u PRVOJ FAZI je dodela i pridruživanje, svakom segmentu programa i biblioteke, memorije i formiranje tabele simbola u koju se unose globalni simboli i apsolutne adrese.
Glavna funkcija DRUGE FAZE je punjenje stvarnog programa i obavljanje relokacione modifikacije neke adresne komponente koja traži tu modifikaciju.

48. Povezivač i punjač?
Povezivac u stvari obavlja funkciju dodele, relokacije i spajanja, a punjač modula funkciju punjenja. Postoje dve klase povezivača:
Najprostiji tip povezivača razvija modul punjenja sličan izvršivom kodu kod apsolutnog punjača. Naziva se modulom memorijske slike.

49. Punjač pokrivača?
Obično se pojedini programi i potprogrami traže u različito vreme.
Ukoliko se eksplicitno zna koji programi i potprogrami pozivaju druge, moguće je realizovati strukturu POKRIVAČA (overlay) kojima se identifikuju međusobno isključivi potprogrami.

50. Dinamički povezivač?
Najopštiji tip punjača.
To je mehanizam kod koga se punjenje i povezivanje eksternih referenci vrši u vreme izvršavanja.
U slučaju poziva na spoljnu adresu ili globalnu promenljivu, poziva se punjač, koji tek tada puni segment koji sadrži eksternu referencu.

51. Faze u prednjem delu kompajlera?
Preprocesor, leksička naliza, sintaksna analiza, prevođenje, prevođenje u kanonički oblik, izbor instrukcija i mašinski zavisna optimizacija.

52. Navesti i ukratko objasniti faze prednjeg dela kompajlera (jedna rečenica po fazi).
Faze prednjeg dela kompajlera su:
1. Preprocesor – iz izvornog programa pravi preprocesiran program, prelaz sa spoljnjeg oblika jezika na osnovni jezik.
2. Leksička analiza – raščlanjivanje znakova izvornog koda, s ciljem da se proizvede izlaz kao niz simbola koji se nazivaju leksemi.
3. Sintaksna analiza – provera redosleda terminalnih simbola (leksema) u izvornom kodu programa i formiranje stabla sintaksne analize.
4. Prevođenje – prevodi stablo sintaksne analize u stablo međukoda, koji nije vezan ni za određeni programski jezik, niti za određenu ciljnu platformu.

5. Prevođenje u kanonički oblik – kanonizacija međukoda (svođenje na osnovni oblik), kanonizacija dovodi do kanoničkog stabla.
6. Izbor instrukcija – izbor instrukcija je problem popločavanja stabla međukoda minimalnim skupom šablona stabla.
7. Mašinski zavisna optimizacija – preuređenje koda (uklanjanje viška LOAD i STORE, JUMP na JUMP, itd.)

53. Stablo sintaksne analize?

Predstavlja sve iskaze i izraze polaznog programa.
Čvorovi su instance sintaksnih klasa, koje se još nazivaju i klasama apstraktne sintakse.
Postoje dve glavne grupe klasa apstraktne sintakse, prva predstavlja iskaze, a druga izraze.

54. Šta je kanoničko stablo?

Kanonizacija dovodi do kanoničkog stabla koje: ne sadrži SEQ (niz od dva iskaza) i ESEQ (niz od jednog iskaza i jednog izraza) čvorove, predak CALL (poziv funkcije) čvora EXP (izračunat izraz) ili MOVE (prebaci rezultat izraza u zadatu lokaciju) čvor.

55. Kako se definiše osnovni programski blok?

Osnovni programski blok se definiše kao niz susednih iskaza koji:
započinje labelom LABEL, a završava se iskazom skoka JUMP ili CJUMP, i unutar bloka ne postoji LABEL, JUMP ili CJUMP.

56. Faze zadnjeg dela kompajlera?

Analiza toka upravljanja, analiza toka podataka, dodela resursa, prilagođenje koda protočnoj strukturi, generisanje koda, povezivanje.

57. Prelivanje (spill) promenljivih?

Problem nastaje kada broj privremenih promeljivih prevaziđe broj raspoloživih registara, a njegovo rešenje je smeštanje prekobrojnih promenljivih u sledeću klasu resursa.

58. Analiza životnog veka promenljive?

Životni vek promenljive započinje prvom definicijom promenljive (upis početne vrednosti u promenljivu), a završava njenom zadnjom upotrebom (poslednje očitavanje sadržaja promenljive).
Prvi korak u ovoj analizi je konstruisanje grafa toka upravljanja.

59. Napisati ime života?

Šta je ovo čoveče?

60. Jednačina životnog veka?

Životni vek promenljive započinje prvom definicijom promenljive (upis početne vrednosti u promenljivu), a završava njenom zadnjom upotrebom (poslednje očitavanje sadržaja promenljive).
Jednačine životnog veka:
in[n] = use[n] U (out[n] – def[n])
out[n] = $U_{succ[n]}$ in[s]

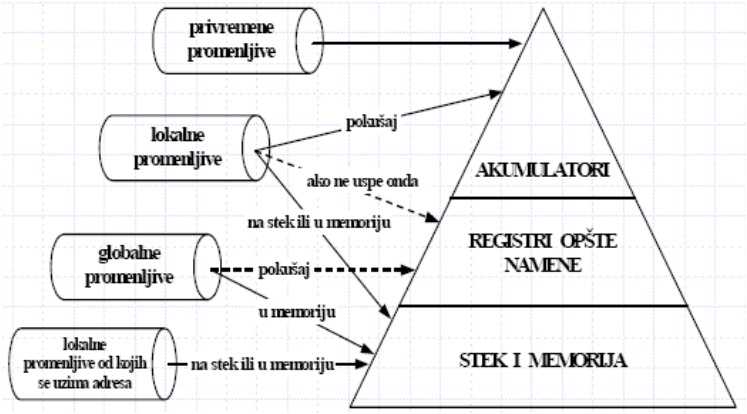
61. Kako se dodeljuju resursi (grafovi)?

Faza dodele resursa se u literaturi naziva i fazom dodele registara, pri čemu se pod registrima podrazumeva skup registara istog tipa.
Dodela registara promenljivama iz grafa smetnji se obavlja tzv. bojenjem grafa smetnji.

62. Bojenje grafa smetnji?

Svakom registru odredišnog procesora odgovara jedna boja, tako da se kod procesora sa K registara, graf smetnji boji sa K boja, što se naziva K-bojenje.

63. Piramida resursa?



64. Faza pripreme koda za protočnu obradu?

Da bi se generisani kod pripremio za protočnu obradu, u kojoj su faze obrade susednih instrukcija međusobno preklopljene, između susednih instrukcija se mora umetnuti potreban broj NOP instrukcija.

65. Zadatak povezivača?

Zadatak povezivača je da objektne datoteke nastale prevođenjem pojedinačnih modula izvornog koda poveže u jedan izvršivi program, koji se puni u PROM, ili u jedan prekrivač, koji se puni u DRAM, a zatim smešta u programski adresni prostor.

66. Faze povezivanja programa?

učitavanje ulaznih i komandnih datoteka,
prvi prolaz povezivanja,
optimizacije (neobavezna faza),
drugi prolaz povezivanja i
pisanje izlaznih datoteka i datoteka sa informacijama za otkrivanje grešaka.

67. Šta je kompaktor?

Kompaktor je uslužni program za mašinski zavisnu optimizaciju. Ulaz u kompaktor je memorijska struktura, koju formira povezivač nakon svog prvog prolaza. Izlaz je kompaktovana memorijska predstava povezanog programa, nad kojim nastavlja rad drugi prolaz povezivača.

68. Funkcija kompaktora?

Osnovni zadatak kopaktora je da minimizira broj mašinskih instrukcija, tako što iterativno menja kod i uklanja nepotrebne mašinske instrukcije (pre svega NOP).

69. Osnovni zahtev prilikom kompaktovanja?

Očuvati semantiku programa!

70. Programski segment?

Linearni programski segment ili osnovni blok programa je niz instrukcija koji započinje labelom a završava se instrukcijom grananja.

71. Kompaktor, faza analize?

U fazi analize, kompaktor na osnovu polaznog asemblerskog koda, koji treba kompaktovati, i ugrađenog modela procesora, formira model polaznog programa.

72. Virtuelni programski segment?

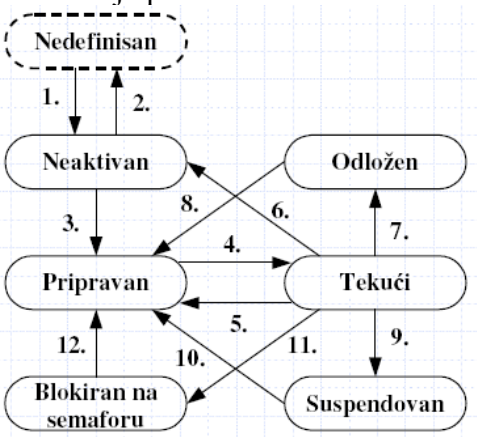
Virtuelni programski segment je nelinearni segment (sadrži i instrukcije skoka) koji obuhvata posmatrani linearni segment.

73. Tehnike kompaktovanja koda?

Organizovane su u dva nivoa: tehnike nižeg i tehnike višeg nivoa.
Tehnike nižeg nivoa: zameni, izbaci, zameni-prethodni, zameni-naredni, preimenuj, mutiraj.
Tehnike višeg nivoa: traži-unapred, traži-unazad, kompaktuj-sa-prethodnim i kompaktuj-sa-narednim.

74. Signalizacija između procesa?
- zabrana smene tekućeg procesa
- upotreba semafora
- poštansko sanduče

75. Graf stanja procesa?



76. Koja su dva načina aktiviranja raspoređivača?
- aktiviranje sistemskim pozivom
- vremenski periodično aktiviranje

77. Modeli smene konteksta?
- Model sa istiskivanjem procesa
- Model bez istiskivanja procesa
- Kombinovani model

78. Definicije Operativnog Sistema?
I def. OS: skup programskih proširenja nad primitivnim elementima fizičke arhitekture koja dovode do virtuelnih mašina.
II def. OS: operativni sistem je upravljački program za dodelu resursa procesima prisutnim u računarskom sistemu.

79. Funkcije OS?
- formiranje (stvaranje) i odstranjivanje procesa
- upravljanje tokom realizacije procesa
- delovanje u uslovima izuzetnih događaja
- dodela resursa fizičke arhitekture procesima
- obezbeđenje pristupa programskim resursima (npr. datotekama, editoru, kompajlerima, asemblerima, bibliotekama i programskim sistemima)
- obezbeđenje zaštitnog mehanizma, upravljanje pristupom i bezbednošću informacija
- obezbeđenje komunikacije između procesa i sinhronizacije

80. Koji su glavni resursi operativnog sistema?
Procesor, memorija, ulazno/izlazni podsistem, informacije (podaci i programi).

81. Klasifikacija operativnih sistema prema vrsti primene?
1) opštenamenski
2) sa radom u realnom vremenu

3) orijentisani na transakcionu obradu

82. Operativni sistemi opšte namene, dva osnovna režima rada?
operativni sistemi za serijsku obradu grupa programskih paketa
multiprogramski operativni sistemi

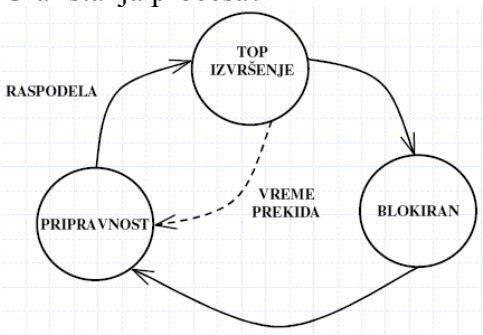
83. Multiprogramski OS?
Računarski sistem radi u multiprogramskom režimu, ukoliko je u sistemu prisutno više korisničkih programa istovremeno.

84. Koja su tri osnovna tipa multiprogramskog režima?
multiprogramiranje bez vremenskih prekida
multiprogramiranje sa vremenskim prekidom
multiprogramiranje sa vremenskim prekidom i programskim prioritetima na bazi podele vremena (TIME SHARING)

85. Operativni sistemi za rad u realnom vremenu?
Ulazni događaji se moraju uneti i opslužiti unutar unapred definisanog intervala vremena.
Vremenski kritični procesi imaju najviši prioritet.
OS mora posedovati časovnik realnog vremena velike preciznosti.
Sistemi moraju omogućiti komunikaciju preko deljivih segmenata operativne memorije.
Da bi se minimiziralo vreme pristupa datotekama, one moraju biti smeštene na susednim zonama na spoljnoj memoriji.
Planeri OS u realnom vremenu mogu reagovati na prekide ili na vremenske cikluse.

86. Klasifikacija OS prema strukturi računarskog sistema?
Multiprocesorski operativni sistemi
Operativni sistemi u računarskim mrežama

87. Definicije procesa?
Def. 1: Sekvencijalni proces je niz aktivnost koje izvršava procesor pri izvršavanju programa sa njemu pridruženim podacima.
Def. 2: Proces je aktiviranje izvršive jedinice programa.
Def. 3: Proces je program u fazi izvršenja, odnosno proces je asinhrona aktivnost.

88. Graf stanja procesa?


89. Operacije nad procesima?
formiranje (stvaranje) procesa
rasformiranje (uništavanje) procesa
suspenzija procesa
ponovno uključenje (aktiviranje) procesa
promena prioriteta procesa
blokiranje procesa i
dodela procesora (raspoređivanje) procesu.

90. Šta sadrži kontrolni blok procesa?
    - jednoznačnu identifikaciju procesa
    - tekuće stanje procesa
    - prioritet procesa
    - ukazivač na memoriju dodeljenu procesu
    - ukazivače dodeljenih resursa
    - zonu za sklanjanje stanja procesa u momentu prekida

91. Asinhroni procesi?
    Procesi su konkurentni ili paralelni ukoliko se izvršavaju u isto vreme i ako su realizovani potpuno nezavisno jedan u odnosu na drugog.

92. Semaforske primitive?
    Semafor S je nenegativan ceo broj. Dve operacije: V(S) i P(S).

93. Napisati 3 rešenja kritične sekcije?
    Dekerov algoritam, TEST & SET instrukcija, Semafori.

94. Planer poslova?
    Prati stanje svih poslova, označavanjem koji posao traži opsluživanje kao i stanje svih koji su opsluživani (pripravni, obrađivani ili blokirani).
    Izabira politiku ulaska poslova u sistem, a na bazi karakteristika tipa, prioriteta traženih resursa ili ravnoteže opterećenosti sistema.
    Dodeljuje potrebne resurse planiranom poslu.
    Oslobađa zauzete resurse po obavljenom poslu.

95. Šta sadrži kontrolni blok posla?
    Identifikator posla, tekuće stanje, prioritet, vremenska ocena i ostalo.

96. Koji su ciljevi planiranja?
    - procesi moraju biti tretirani ravnomerno
    - maksimizirati propusnost sistema
    - maksimizirati broj interaktivnih korisnika
    - poslovi moraju da se izvršavaju u tačno određeno vreme
    - minimiziranje praznog hoda resursa

97. Kakvi mogu biti prioriteti procesa?
    Mogu biti **statički** i **dinamički**.

98. Algoritmi planiranja procesa?
    - prvi u listu prvi iz liste
    - redom u krug (ROUND ROBIN - RR)
    - planiranje davanjem prednosti kraćim poslovima
    - planiranje na bazi najkraćeg preostalog vremena
    - planiranje na bazi najvećeg odnosa vremena odziva
    - višenivovski redovi čekanja sa povratnom spregom

99. Kakva može biti organizacija planera poslova?
    Distribuiran i Centralizovan planer.

100. Međusobno blokiranje procesa (DEADLOCK)?

Za proces u multiprogramskom sistemu se kaže da je u stanju međusobnog blokiranja ukoliko čeka na neki poseban događaj koji se ne može desiti.

101. Tehnike za borbu protiv međusobnog blokiranja?

Sprečavanje, izbegavanje, detekcija, oporavak.

102. Sprečavanje međusobnog blokiranja?

- Svaki proces može da traži sve njemu potrebne resurse odjednom i ne može da nastavi izvršenje sve dotle dok mu svi oni ne budu dodeljeni.
- Ukoliko proces koji drži neke resurse odustane od zahteva, mora osloboditi svoje polazne resurse i ako je potrebno zatražiti ih sve ponovo i sa dodatnim resursima (izbegavanje nesmenjivosti resursa).
- Uslovljavanje linearnog traženja tipa resursa za sve procese, tj. ako proces ima dodeljene resurse datog tipa može tražiti samo one resurse tog tipa koji slede po redosledu.

103. Neophodni uslovi za međusobno blokiranje?

a) proces zahteva isključivu kontrolu nad resursima koje traži (uslov međusobne isključivosti)
b) proces drži njemu već dodeljene resurse za vreme čekanja na dodatne resurse (uslov čekanja)
c) resurs ne može da bude uzet procesu koji ga drži sve dok je resurs potreban radi završetka procesa (uslov nesmenjivosti)
d) postoji kružni lanac procesa u kome svaki proces drži jedan ili više resursa koji se traže od sledećih procesa u lancu (uslov kružnog čekanja)

104. Izbegavanje blokiranja?

Bankarov algoritam.

105. Redukovani graf dodele resursa?

Koristi se za određivanje da li postoji situacija međusobnog blokiranja i jedna je od tehnika koja je korisna u detekciji međusobnog blokiranja.

---

106. RMA?

Jedna od tehnika analize sistemskog opterećenja je tzv. analiza monotone stope (RMA – Rate Monotonic Analysis) izvršenja zadataka.

107. Sinhronizacija između zadataka u PERC-u?

Je tzv. atomski iskaz (engl. atomic statement).
Telo atomskog iskaza je neprekidivo.

108. Dovoljan uslov rasporedivosti?

Dovoljan uslov je da svi zadaci završe svoj posao da završetka njihove periode:
$\mathbf{C_1/T_1 + ... + C_n/T_n \leq U(n)}$
$\mathbf{U(n)= n(2^{1/n} - 1)}$

Ovaj test je veoma važan jer govori o tome da li je zadati skup zadataka rasporediv ili ne. Skup zadataka je rasporediv ako svaki zadatak završava svoj posao do završetka njegove periode.

109. Šta su pretpostavke za uslov rasporedivosti?

Problematične pretpostavke osnovne RMS teorije su:

1. Smena zadatak je trenutna
2. Vreme rada OS nije uračunato
3. Nije dozvoljena međusobna interakcija zadataka
4. Zadaci postaju pripravni za izvršenje tačno na početku svoje periode
5. Krajnji rok je jednak periodi izvršenja
6. Zadaci sa kraćom periodom dobijaju veći prioritet a da se ne razmatra kritičnost zadataka

7. Izvršenje zadataka je uvek konzistentno sa RM prioritetima

110. Koja je četvrta pretpostavka za uslov rasporedivosti? :)

111. Koja je druga tehnika za raspoređivanje?

Prvo zadaci sa najranijim krajnjim rokom (engl. earliest deadline scheduling algorithm).
On je superioran, pošto je granica raspoređivanja uvek jednaka jedinici:
$C_1/T_1 + ... + C_n/T_n \leq U(n) = 1$

112. Koja je tehnika da se ukalupi aperiodičan u periodičan impuls (valjda zadatak)?

Tehnika prozivanja (polling), aperiodičan server i sporadičan server.

113. Aperiodičan server?

Njemu se dodeljuje određen vremenski budžet i period ponovnog punjenja budžeta.
Aperiodičan server rukuje zahtevima sve dotle dok se dodeljeni budžet ne potroši.
Kad se budžet isprazni, zahtevi upućeni aperiodičnom serveru se opslužuju u pozadini, sve dotle dok se budžet aperiodičnog servera ponovo ne napuni.

114. Sporadičan server?

Vremenski budžet dedeljen sporadičnom serveru se ponovo puni tek kad se potpuno istroši.
Npr. serveru sa budžetom od 10 ms i periodom ponovnog punjenja budžeta od 100 ms će ponovo biti napunjen budžet od 10 ms, 100 ms nakon što je tekući budžet potpuno potrošen.
Algoritam sporadičnog servera eliminiše pojavu odloženog izvršenja (engl. deferred execution).

115. Kako se sprečava beskonačna inverzija prioriteta?

U cilju sprečavanja pojave neograničene inverzije prioriteta, u sistemima koji rade u realnom vremenu, sa čvrstim vremenskim ograničenjima, se za potrebe međusobne sinhronizacije zadataka koriste binarni semafori sa plafonom prioriteta.

116. Osnovni protokol nasleđivanja prioriteta?

Delimično rešenje ovih problema je osnovni protokol nasleđivanja prioriteta (engl. basic priority inheritance protocol, PIP).
On rešava problem neograničene inverzije prioriteta, ali ne rešava problem međusobnog blokiranja.

117. Protokol sa plafonom prioriteta?

Potpuno rešenje oba problema je tzv. protokol sa plafonom prioriteta (engl. ceiling priority protocol, CPP).
Dve važne osobine:

- onemogućava međusobno blokiranje procesa
- onemogućava neograničenu invreziju prioriteta, naime, najviše jedan proces nižeg prioriteta može blokirati proces višeg prioriteta za vreme dok proces nižeg prioriteta ne napusti svoju kritičnu sekciju.

118. Prošireni test rasporedivosti?

Ovaj test uračunava moguća blokiranja zadataka na semaforima koji kontrolišu deljene resurse.
Za potrebe ovog testa se definiše B_i kao najduže ukupno vreme blokiranja zadatka ti unutar njegove periode.

$$C_1/T_1 + B_1/T_1 \leq 1(2^{1/1}-1)$$
$$C_1/T_1 + C_2/T_2 + B_2/T_2 \leq 2(2^{1/2}-1)$$
...
$$C_1/T_1 + C_2/T_2 + ... + C_k/T_k + B_k/T_k \leq k(2^{1/k}-1)$$
...
$$C_1/T_1 + C_2/T_2 + ... + C_n/T_n \leq n(2^{1/n}-1)$$

119. Problem promene režima?

Promena režima izvršenja se može zamisliti kao brisanje nekih zadataka i dodavanje drugi zadatka, ili kao promena parametara određenih zadataka.

120. Zahtevi za promenu režima?

1) Kompatibilnost: moraju se sačuvati dve važne osobine protokola plafona prioriteta: (1) nemoguće je međusobno blokiranje zadataka i (2) zadatak nižeg nivoa prioriteta može blokirati najviše jednom zadatak višeg nivoa prioriteta.
2) Podrška održavanju: mora dozvoliti dodavanje novih zadataka
3) Performansa: bar kao kod monitora

Rešenje: Protokol za promenu režima (engl. Mode Change Protocol, MCP):
On se sastoji iz tri dela koji odgovaraju originalnim zahtevima:
- aspekt kompatibilnosti
- aspekt održavanja
- aspekt performanse

121. Koji uslovi su ispunjeni (MCP)?

aspekt kompatibilnosti
aspekt održavanja
aspekt performanse

122. Aspekti performansi?

Protokol promene režima za RM raspoređivanje treba da se obavlja bar istom brzinom kao promene radnog režima kod cikličnog egzikjutiva (monitorska petlja).

123. Iskorišćenje veće od U(n)?

Rešenje je analiza vremena odziva zadatka ti.

$$a_0 = \sum_{j \in H+\{i\}} C_j \qquad a_{n+1} = C_i + \sum_{j \in H} \left\lceil \frac{a_n}{T_j} \right\rceil C_j$$

H je skup zadataka čiji prioritet je veći od prioriteta ti.
$a_{n+1} = a_n$ je kriterijum za završetak rekurzivnog razvoja. Vreme odziva zadatka je jednako a_{n+1}.

124. Kašnjenje zbog smene zadatka?

$$a_0 = B_i + \sum_{j \in H+\{i\}} (C_j + 2S)$$

$$a_{n+1} = B_i + C_i + 2S + \sum_{j \in H} \left\lceil \frac{a_n}{T_j} \right\rceil (C_j + 2S)$$

2S – vreme potrebno za dve smene zadataka.

125. Mogu biti zadaci

126. Kanonizacija?

Zadatak ima kanonički oblik ukoliko se sastoji od podzadataka čiji prioritet se ne smanjuje – algoritam kanonizacije:
Postavi Pim(i)' = Pim(i) gde je m(i) broj podzadataka zadatka τi
Ako je (Pij' < Pij-1) onda postavi Pij-1' = Pij-1 u protivnom postavi Pij-1' = Pij'
Ponavljaj ovaj postupak pomerajući se od zadnjeg prema prvom podzadatku.

127. Klasifikacija zadataka?

Prilikom analize zadatka τi, ostali zadaci se razvrstavaju na osnovu odnosa prioriteta njihovih podzadataka i zadatka τi. Ključan kriterijum je odnos prioriteta prvog podzadatka i minimalnog prioriteta svih podzadataka zadatka τi, $Pmin_i$.

128. Tri osnovna koraka u korišćenju RAPID RMA?
- Formiranje modela zadataka
- Formiranje modela resursa
- Analiza rasporedivosti sistema

129. Korišćenje RMA tokom životnog ciklusa?
RMA se može koristiti u svim fazama razvoja programske podrške.
Tipične faze razvoja programske podrške u realnom vremenu su:
- faza analize zahteva (engl. software requirements analysis),
- faza preliminarnog projektovanja (engl. preliminary design),
- faza detaljnog projektovanja (engl. detailed design), i
- faza realizacije (engl. implementation).

---

130. U/I operacije?
Čitanje, pisanje i kontrolna operacija.

131. Prekidi i U/I procesi?
U/I radnje se realizuju kroz saradnju tri asinhrona procesa:
a) Korisnički proces koji traži U/I uslugu
b) Rukovalac prekidima
c) Rukovalac U/I uređajem

132. Nezavisnost programa od ulazno-izlaznih jedinica?
Da bi korisnički programi postigli veći stepen adaptivnosti i prenosivosti, moraju biti pisani tako da ne zavise od neke specifične U/I jedinice.
Radi toga je neophodno uvesti pojam SIMBOLIČKOG NAZIVA ULAZNO-IZLAZNE JEDINICE I STANDARDNE SPREGE ULAZNO-IZLAZNE JEDINICE.
To je dovelo do pojma VIRTUELNE U/I JEDINICE.

133. Virtuelna UIJ?
Da bi se neki proces izvršio on koristi stvarnu U/I jedinicu, međutim, slično kao kod memorije, on se obraća virtuelnoj U/I jedinici, umesto stvarnoj.
Svaka virtuelna UIJ ima simbolički naziv, koji koriste procedure u procesu da bi ih nazvale.
Pri tome, mora postojati i mehanizam prevođenja, kojim se pozivanje virtuelne UIJ prevodi u pozivanje stvarne UIJ.

134. Kako komuniciraju procesi van realnog vremena i oni u realnom vremenu?
Linux procesi i RTLinux zadaci u realnom vremenu, komuniciraju preko FIFO bafera, koji ne zahtevaju zaključavanje (engl. lock-free FIFO baffers).

135. Koji su drugi razlozi zašto Linux nije RTOS?
Pre svega zbog činjenice da su prekidi zabranjeni u toku izvršenja jezgra OS, zbog čega se odgovarajući proces ne može istisnuti.
U druge probleme spadaju:
raspoređivanje procesa radi deljenja vremena (engl. time-sharing)
nepredvidivost vremenskih odnosa u sistemima sa virtuelnom memorijom,
nedostatak fizičkih vremenskih brojača (engl. timer) visoke rezolucije (npr. sa korakom manjim od 1ms).

136. Koja su 3 načina rukovanja memorijom?
1) Partitivna dodela memorijskog prostora
2) Relokatibilna partitivna dodela memorijskog prostora
3) Stranična dodela memorijskog prostora

137. Čemu služe prstenovi zaštite?

138. Kako se rešavaju dugačke kritične sekcije?