1. Sto je OS i koji su mu glavni ciljevi?

- OS je upravljacki program koji sluzi kao sucelje izmedju korisnika i racunarskog sklopovlja. Glavni ciljevi su mu sto jednostavnija uporaba racunarskog sklopovlja i sto veca efikasnost prilikom koristenja racunarskih resursa.

2. Koji su zadatci OS-a?

- Zadatci OS-a su:
  - Upravljanje procesima
  - Upravljanje memorijom
  - Upravljanje datotekama (datotecni sustav)
  - Upravljanje U/I uredjajima
  - Otklanjanje pogresaka
  - Rad sa mrezom
  - Zastita koristenja softvera
  - Tumacenje korisnikovih naredbi

3. Sto je proces, cime je odredjen, i kako OS vodi evidenciju o procesu?

- Proces je program u izvedbi, odnosno aktivni program. On zivi u radnoj memoriji, za razliku od pasivnog programa koji zivi na tvrdom disku i ne izvodi se. Proces se opisuje putem deskriptora procesa ili PCB-a (process control block). To je struktura u kojoj se nalaze informacije u vezi procesa poput registara, PC-a, raznih pokazivaca, te informacija poput id-a, imena procesa, korisnika, itd.

4. Navedi i objasni stanja u kojima se proces nalazi tijekom izvodjenja.

- Postoji 5 stanja:
  - Nov - program se ucitava, stvara se PCB, te proces prelazi u stanje Pripravan
  - Pripravan - proces je u redu cekanja za racunarske resurse (procesor)
  - Aktivan - proces je aktivan i koristi procesor
  - Ceka - procesor ceka na neku informaciju ili dogadjaj, najcesce od U/I uredjaja. Nakon primitka informacije, proces ulazi u stanje Pripravan
  - Zavrsen - proces je zavrsio izvedbu, brise se iz memorije i brise se PCB

5. Kako OS organizira procese kako bi ih prebacivao iz stanja u stanje?

- OS procese drzi u redovima cekanja implementiranih putem vezanih listi. Lista aktivnih procesa najcesce ima jedan proces, no moze imati vise procesa u slucaju vise-procesorskog racunalnog sustava. Postoje i liste “pripravan” i “ceka”, koji sadrze procese koji su u stanju Pripravan ili Ceka. Te liste se sastoje od 2 pokazivaca: jedan na pocetak liste a drugi na kraj. Svaki PCB u listi takodjer sadrzi pokazivac koji pokazuje na sljedeci PCB u redu. Kada se proces treba prebaciti iz jednog stanja u drugi, OS uradi context switch.

6. Objasni context switch.

- Context switch je operacija kojom OS zamjenjuje trenutni aktivni proces sa nekim drugim. Nacin na koji se to radi je da se prvo “spasi” trenutno stanje procesora (registri, PC, itd.) u PCB aktivnog procesa. Zatim, taj aktivni proces se stavlja na kraj liste Pripravan (ili Ceka), pritom pazeci da se azurira pokazivac na kraj liste kao i pokazivac predzadnjeg PCB-a da pokazuje na taj novi proces. Zatim, uzima se prvi proces iz liste Pripravan i stavlja se kao aktivni proces, ponovno vodeci brigu o pokazivacima liste i PCB-ova. Zatim, ucitaju se vrijednosti registara aktivnog PCB-a u procesor, te se izvedba nastavlja.

7. Objasni i navedi strategije dodjele procesora.

- Aktivnom procesu se moze putem raznih algoritama/pravila odrediti koliko dugo da se izvrsava prije nego sto se dadne prednost nekom drugom procesu. Te

strategije su:
    - FIFO (First In, First Out) - uvijek se uzima prvi proces iz liste Pripravan i izvodi se, nevazno o resursma koje zahtjeva ili vremena izvodjenja
    - SJF (Short Jobs First) - daje se prednost kratkim poslovima (procesima)
    - Round-Robin FIFO - isto kao FIFO ali postoji ogranicenje vremena izvodjenja
    - Ogranicavanje vremena - aktivni proces se mijenja nakon nekog odredjenog vremena

8. Na primjeru proizvodjac-potrosac objasni kriticni odsjecak.

- Proizvodjac-potrosac je racunarski sustav koji se cesto javlja i sastoji se od 2 dijela: proizvodjac koji proizvodi podatke i ubacuje ih u neku listu, i potrosac koji uzima podatke iz te liste i preradjuje ih. Ovi procesi su neovisni jedno o drugom, no imaju dijeljenu memoriju (listu) o kojoj se treba voditi racuna. U takvom sustavu kriticni odsjecak moze biti dio programa koji azurira pokazivace na pocetak/kraj liste bufera. Posto se operacija “povecaj za jedan” zapravo izvodi u 3 koraka (dohvati, povecaj, spremi), moze se dogoditi da program bude prekinut u sred izvodjenja te operacije. Kada drugi dio sustava ponovno bude citao zajednicku memoriju, program ce biti u nevazecem stanju. Ta operacija citanja+povecavanja+spremanja je kriticni odsjecak. Da bi rijesili ovaj problem moramo osigurati da program ne bude prekinut prilikom izvodjenja kriticnog odsjecka, i da ne moze bilo koji drugi proces uci u kriticni odsjecak. To se moze izvesti putem atomskih operacija poput test&set, ili struktura zasnovanih na atomskih operacijama poput semafora i kljuceva.

9. Kako se rjesava kriticni odsjecak putem test&set?

- Test&set je atomska operacija kojom se 3 operacije (procitaj, modificiraj/postavi, spremi) izvode u jednom taktu. Atomska operacija znaci da je nedjeljiva. To je korisno u viseprocesnim sustavima gdje imamo kriticne odsjecke. Putem te metode, mozemo zatraziti od OS-a (i nadalje sklopovlja) da dobijemo pristup nekoj zajednickog varijabli i da ju postavimo na neku vrijednost, vodeci racuna da se ta operacija ne smije izvesti dok ju neki drugi proces izvodi. To se najcesce izvodi putem while petlje: ulazimo u kriticni odsjecak putem while (text&set(flag)) { nop; }, a izlazimo tako sto postavimo zastavicu na nulu. U ovakvom pristupu se javlja problem aktivnog cekanja.

10. Objasni semafor.

- Semafor je metoda sinkronizacije procesa putem koje mozemo zastiti kriticni odsjecak procesa od drugih procesa, bez koristenja aktivnog cekanja. Semafori su implementirani na razini OS-a/sklopovlja, i njihova struktura se sastoji od broja (int) i pokazivaca na PCB. Komplementarne metode za semafor su wait() i signal(). Putem wait() metode proces ceka dok vrijednost semafora ne bude veca od minimalne (-1) prije nego sto izvodi neku operaciju, i cekanje se izvodi tako sto se proces stavi u listu cekanja semafora, i na ovaj nacin se izbjegne aktivno cekanje. Metoda signal() povecava vrijednost semafora za jedan, tako davajuci priliku drugim procesima da udju u svoj kriticni odsjecak. Postoje binarni i opci semafori.

11. Kako pomocu semafora rijesiti problem proizvodjac-potrosac?

- To se moze rijesiti koristeci 2 semafora - jedan postavljen na max. duljinu bufera (npr. 5) nazvan Spun i drugi postavljen na nulu nazvan Sprazan. Proizvodjac ce pozvati wait() na semafor Spun, tako “rezervirajuci” sam sebe kao proizvodjaca. Kada Spun dodje do negativne vrijednosti, onda ne mozemo vise proizvodjaca registrirati jer bi onda tako proizveli vise podataka nego sto moze stati u bufer. Na kraju, proizvodjac pozove signal() na semafor Sprazan, tako dajuci priliku potrosacu da izvodi svoj posao. Potrosac prvo pozove wait() na semafor Sprazan (ceka da se bar jedan podatak pojavi u buferu), te onda rezervira taj podatak sam za sebe, pa drugi procesi potrosaci moraju cekati za nove podatke. Nakon sto

zavrsi, pozove signal() na Spun, tako davajuci priliku proizvodjacima da proizvedu jos jedan podatak.

12. Objasni deadlock.

- Deadlock je naziv za stanje viseprocesnog sustava u kojem 2 procesa cekaju na jedno drugog. To se dogodi kada jedan proces zakljuca zajednicku varijablu A i drugi proces zakljuca varijablu B, zatim prvi proces pokusa zakljucati varijablu B, no ne moze jer i drugi proces pokusava zakljucati varijablu A (zauzet je), no ni taj drugi proces ne moze zakljucati varijablu A jer prvi proces vec ceka. Time ce ta 2 procesa vjecno cekati jedno na drugog.

13. Kako se memorija dodjeljuje po stranicama, objasnite paging.

- Prilikom izvedbe programa, adrese u tom programu su napisane u tzv. logickom adresnom prostoru. Prava memorija se naziva fizicki adresni prostor. Kada se proces pokrene, OS zajedno sa sklopovljem mora prevesti sve logicke adrese u fizicke, i to se radi stranicu po stranicu. Stranica je segment fiksne duljine logicke memorije, a svaka stranica se mapira u okvir fizicke memorije. Okvir je takodjer naziv za fiksni segment fizicke memorije. Posto je i tvrdi disk organiziran u okvirima iste velicine kao i okviri memorije, prilikom pokretanja procesa se mogu svi podatci vazni za taj proces direktno ucitati u memoriju, odnosno stranicu po stranicu u logickom adresnom prostoru. Na taj nacin ostvarimo tzv. virtualnu memoriju, pomocu kojom smanjujemo fragmentaciju memorije i pojednostavljujemo pristup memoriji programima. To se ostvaruje pomocu tablice stranica koja sadrzi logicke i fizicke adrese i moze prevesti iz jedne u drugu.

14. Za sto sluzi TLB?

- TLB je sklop koji sluzi za optimizaciju pristupa tablici stranica prilikom prevodjenja logicke u fizicku memoriju. Posto je tablica stranica poprilicno velika, ona se mora pohraniti u opcoj memoriji (ne u posebnom brzom sklopovlju), no ovo usporava pristup memoriji jer smo efektivno udvostrucili broj pristupa (jedan za dohvacanje tablica stranica, drugi za pristup fizickoj memoriji). Uvodi se TLB, koji sluzi kao sekundarna memorija tj. kes za tablicu stranica. OS prvo gleda u TLB za neku logicku adresu, i posto je TLB implementiran putem brzog sklopovlja (jer je manji pa je prihvatljivo), to ce ubrzati dohvacanje vrijednosti iz memorije. Ukoliko trazena adresa nije u TLB-u, onda se mora ici duzim putem do glavne memorije, no takodjer azuriramo TLB da mozemo toj adresi pristupiti ubuduce.

15. Sto je virtualna memorija?

- Virtualna memorija je naziv za memorijski prostor koji proces vidi prilikom izvedbe. Taj prostor izgleda kao beskonacan adresni prostor procesu (maksimalna adresa ovisi o sirini adresne linije), i omogucuje pojednostavljeni pristup memoriji te i dinamicko ucitavanje programa sa diska. To se ostvaruje dodavanjem tzv. “bita pristupa” na tablicu stranica u memoriji, i taj bit oznacava je li ta adresa ucitana sa diska ili ne. Na ovaj nacin mozemo ucitati program koji je veci od radne memorije tj. koji ne bi inace mogao stati u radnu memoriju. Kada proces zavrsi sa jednim dijelom, moze ga izbaciti iz radne memorije (postaviti bitove pristupa na 0 te osloboditi memoriju drugim procesima), zatim ucitati novi dio programa sa diska.

16. Koje su strategije izmjene stranica?

- U slucaju kada je tablica stranica puna, a procesu treba memorija, onda se treba neka adresa iz tablice izbaciti. Ovo se moze izvesti putem raznih algoritama: FIFO, LRU, ili nesto drugo.

17. Objasnite programsko obavljanje U/I operacija (ne IRQ).

- Programsko obavljanje U/I operacija je nacin slanja i primanja informacija sa U/I uredjaja ili sustava tako sto proces konstantno provjerava je li trazeni podatak spreman. Opcenito, OS upravlja U/I uredjajima pa moze svakog od njih prikazati na jedinstven nacin svim programima, npr. kao posebne datoteke na disku ili kao posebni dio radne memorije. Program zatim salje zahtjev OS-u da zeli pristupiti U/I uredjaju, te zatim ceka. U/I uredjaj posalje trazenu informaciju te postavi status operacije na “gotovo”, dok proces cijelo vrijeme provjerava je li podatak spreman. Kada jest, onda proces nastavlja obradu. Nedostatak ovog pristupa je aktivno cekanje sto trosi resurse.

18. Objasnite obavljanje U/I operacija pomocu IRQ.

- Obavljanje U/I operacije putem IRQ (interrupt request) je nacin da saljemo i primamo informacije sa U/I uredjaja putem sustava sklopovskih prekida (hardware interrupts) koje OS podrzava. Proces putem pokazivaca odabere uredjaj, te operaciju koja se treba izvrsiti (npr. READ). Proces dobije statusni odgovor od U/I uredjaja, te se PCB procesa stavlja u red Ceka dok se ceka podatak. Kada je podatak spreman, proces se odblokira/probudi i vraca u red pripravnih programa.

19. Sto je deskriptor datoteka?

- Deskriptor datoteka je struktura kojom se opisuje skup podataka lociranih na disku. Taj skup podataka se zove datoteka, i deskriptor datoteka takodjer sadrzi dodatne informacije o datoteci (putanja, uredjaj na kojem se nalazi, vlasnik, zastavice pristupa, vremena, itd.). Sam pristup podatcima unutar datoteke se moze izvesti putem raznih nacina, ovisno na koji je nacin datoteka rasporedjena u memoriji.

20. Koji su nacini zapisa datoteke na disk?

- Postoji:
  - Kontinuirana alokacija - podatci su pohranjeni u neprekinutom dijelu memorije diska, a deskriptoru datoteke su dovoljna 2 podatka: velicina datoteke u bajtovima i adresa prvog bajta datoteke.
  - Vezana lista - podatci nisu pohranjeni kontinuirano, nego je datoteka podijeljena u vise dijelova, svaki od kojih moze stati u neku od supljina fragmentirane memorije na disku. Svaki taj dio sadrzi pokazivac na sljedeci dio, sto nalikuje vezanoj listi. Ovaj nacin bolje iskoristava prostor, no sporiji je jer za pristup bilo kojem podatku (osim prvog i/ili zadnjeg) moramo proci kroz svaki podatak prije njega u listi.
  - Indeksna alokacija - zajedno sa datotekom je pohranjena i tablica indeksa i adresa na disku. Posto su datoteke zapisane u segmentima na disku (blokovima), ta tablica se sastoji od indeksa bloka i adrese tog bloka na disku. Na ovaj nacin mozemo bolje iskoristiti fragmentiranu memoriju bez usporavanja prilikom nasumicnog pristupa, no velicina zauzetog prostora na disku je veca jer se tablica blokova takodjer treba pohraniti i azurirati.