**Završni ispit**

**Napomena: Dokument otvoriti u Web layout formatu radi adekvatnog formatiranja teksta. Donja desna strana Worda.**

**Januar 2017**

1. **Koristi se segmentacija sa staničenjem, jedna tabela stranica po segmentu:**

**Fizička memorija – 234B,**

**Maksimalni broj segmenata - 64k = 216**

**Maksimalna veličina segmenata - 220 stranica**

**Stranice veličine - 212 B**

Čim imaš segmentaciju sa straničenjem zamisli je kao tablicu stranica u 2 nivoa.

1. **Koliko bitova ima logička adresa (virtuelna)**

Virtuelna adresa: 16b + 20b + 12b = 48b

Zašto? Zato što je format virtuelne adrese: Broj segmenta / Broj stranice / Ofset unutar stranice (Virtuelna memorija / 24)

1. **Koliko je format logičke (virtuelne) adrese? Obrazložiti.**

Bolje je pitanje Kakav je format:

16b 20b 12b

|  |  |  |
| --- | --- | --- |
| Segment | Broj stranica | Veličina stranica |

1. **Kolika je veličina straničnog okvira?**Veličina straničnog okvira je jednaka veličini stranice => 212 = 4KB
2. **Koliko ima ulaza u tabelu segmenata, a koliko u tabelu stranica?**

216 ulaza u tabelu segmenata, jer ima 64k segmenata, a to je 26 x 210

A u tabelu stranica 220 jer je max veličina segmenta 220 stranica.

Jer pazi: ti imaš prvo tabelu segmenata, sa 216 ulaza. Svaki od tih ulaza ima po 20b. Odatle ćeš da dobiješ ulaz u tabelu stranica. Imaš 220 mogućnosti za ulaze. Znači ti iz svakog segmenta imaš po 220 mogućnsti. Na tako očitanu vrednost iz tabele segmenata konkateniraš ofset iz virtuelne adrese i dobio si fizičku adresu.

1. **Koliko bitova u fizičkoj adresi specificira stranični okvir?**

Ukupna fizička memorija je adresirana sa 34b, od toga je 12b otpalo na adresiranje unutar stranice, odakle sledi da je 22b zaduženo za adresiranje stranice, odnosno straničnog okvira.

1. **Šta se menja u tabeli stranica pri obradi greške stranice?**

Pri obradi greške se referencira sekundarna memorija kako bi se pribavila odgovarajuća stranica, koja se zatim smešta u fizičku (RAM) memoriju, a njena fizička adresa se upisuje u Tabelu stranica ukoliko ima mesta; ako mesta nema onda se jednim od algoritama zamene bira stranica na čije mesto će biti upisana novopribavljena stranica.

**Vreme učitavanja stranice sa diska je 24ms, prekidanje i ponovno restartovanje procesa je 5ms, vreme izvršenja algoritma za zamenu stranice je 1ms, vreme pristupa memoriji 50ns. Odrediti efektivno vreme, ako se koristi straničenje u 2 nivoa i na svakih 104 memorijskih referenci nastaje 1 greška stranica. Objasniti.**

p = 1/104 = 10-4 verovatnoća nastajanja greške

EFP = (1 - p)\*Pogodak + p\*Promašaj = (1 – 10 -4)\*(50ns + 50ns + 50ns) + 10-4\*(50ns + 50ns +24ms + 5ms + 1ms + 50ns)

Pri pretraživanju tabele stranica postoje 2 moguća scenarija:

Prvi, da nije nastala greška (1 - p) I (zato puta) tada se pristupa prvom nivou Tablice straničenja, pa drugom, pa memoriji, otud 3\*50ns. Objasnili smo deo: (1 - p)\*Pogodak

ILI, zato ide plus, nastala je greška I (otud puta) pristupamo dvama (hehe) nivoima gde na jednom uočavamo grešku stranice, opslužujemo tu grešku i nastavljamo pristup memoriji. Objasnili smo: p\*Promašaj.

1. **a) Kom procesu će Windows/Unix/Linux OS dati prednost pri planiranju procesa algoritma planiranja:**

**P1: prethodno deblokiran jer je sadržaj datoteke koji je zahtevao učitan u RAM**

**P2: prethodno istrošio vremenski kvant određujući sledeći potez šahovskog algoritma**

**Ukratko objasniti kako.**

Prednost će dobiti P1, jer je on UI orijentisan, za razliku od P2 koji je CPU orijentisan.

Unix – će dati prednost procesu P1 jer njegov prioritet raste pošto nije iskoristio svoj vremenski kvant, a procesu P2 opada prioritet jer jeste iskoristio svoj vremenski kvant. U zavisnosti od prioriteta procesi će se nalaziti u različitim redovima.

Linux – Ima 2 reda: aktivne i istekle redove (koji su prethodno iskoristili vremenski kvant)

Prednost daje nitima koje su U/I orijentisane u odnosu na CPU orijentisane. Sledi prednost dobija ponovo P1. Kapiram da sve što se odnosi na niti odnosi se i na procese. Barem je tako reko na početku časa.

Windows – analogno sve. Ako se iskorisi čitav vremenski kvant onda se smanjuje prioritet. Znači

P1 ima prednost.

Napomena: Mislim da nije trebalo za svaki OS ponaosob, ali sam to kasnije shvatio.

**b) Za sledeći skup procesa prikazati redosled izvršenja procesa i prosečno vreme zadržavanja / prolaska (turnaround time) ukoliko se koriste algoritmi:**

**1. RR (q = 2)**

**2. SRT**

**3. ML Feedback q = 2i**

Napomena: Moguće da sam nešto i omašio kad sam ubacivao kolone, ali logika je ta.

|  |  |  |
| --- | --- | --- |
| Proces | Vreme izvršenja | Vreme aktiviranja |
| A | 6 | 0 |
| B | 10 | 3 |
| C | 4 | 6 |
| D | 7 | 9 |
| E | 5 | 12 |

1. Objašnjenje: RR, q = 2, tj. Svakom procesu se uvek daju po 2 vremenske jedinice. Ukoliko procesu nisu potrebne 2 vremenska kvanta onda je on prekinut i naredni kreće sa izvršavanjem.

Vodi računa da kad dođe novi proces da nešto verovatno već čeka u redu. Ipak ti je ovo ’’cirkularni FCFS’’.

* Vremenski trenutak

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* |
| A | \* | \* | \* | \* |  |  | \* | \* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| B |  |  |  |  | \* | \* |  |  |  |  | \* | \* |  |  |  |  |  |  | \* | \* |  |  |  |  | \* | \* |  |  |  | \* | \* |  |
| C |  |  |  |  |  |  |  |  | \* | \* |  |  |  |  | \* | \* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | \* |
| D |  |  |  |  |  |  |  |  |  |  |  |  | \* | \* |  |  |  |  |  |  | \* | \* |  |  |  |  | \* | \* |  |  |  |  |
| E |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | \* | \* |  |  |  |  | \* | \* |  |  |  |  | \* |  |  |  |

Red: A / B A / A C B / C B D / B D C / D C E B / C E B D / B D E / D E B / E B D / B D E / D E B / E B D / B D

Vreme prolaska zadatka (turnaround time) je rezidentno vreme Tr, ili ukupno vreme koje ta stavka provodi u sistemu (vreme čekanja plus vreme usluge).

Kako najlakše da ga odrediš: Trenutak završetka – Trenutak dolaska. Primer: proces B je aktiviran još u trenutku 3, iako se izvršava od trenutka 4, tako da je najbezbedniji načn za određivane vremena prolaska pomoću ove formule, jer samo tablica ne daje odgovarajuće rezultate. Ako je poslednji iskorišćeni vremenski kvant n, onda je trenutak završetka n + 1.

Sledi,

Tr(A) = 8 – 0 = 8

Tr(B) = 31 – 3 = 28

Tr(C) = 32 – 6 = 24

Tr(D) = 28 – 9 = 19

Tr(E) = 29 – 12 = 17

Prosečno vreme je (8 + 28 + 24 + 19 + 17) / 5 = 96 / 5 = 19,2.

2) SRT – Shortest Remaining Time

|  |  |  |
| --- | --- | --- |
| Proces | Vreme izvršenja | Vreme aktiviranja |
| A | 6 | 0 |
| B | 10 | 3 |
| C | 4 | 6 |
| D | 7 | 9 |
| E | 5 | 12 |

Objašnjenje: Treba proceniti vreme izvršenja i izabrati najkraće, pri čemu se trenutno aktivni proces MOŽE PREKINUTI ako se pojavi proces sa manjim procenjenim vremenom izvršenja.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* |
| A | \* | \* | \* | \* | \* | \* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| B |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* |
| C |  |  |  |  |  |  | \* | \* | \* | \* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D |  |  |  |  |  |  |  |  |  |  | \* | \* | \* | \* | \* | \* | \* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| E |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | \* | \* | \* | \* | \* |  |  |  |  |  |  |  |  |  |  |

Tr(A) = 6 – 0 = 6

Tr(B) = 31 – 3 = 28

Tr(C) = 9 – 4 = 5

Tr(D) = 16 – 7 = 9

Tr(E) = 21 – 5 = 16

Prosečno vreme je (6 + 28 + 5 + 9 + 16) / 5 = 64 / 5 = 12,8.

3) Feedback q = 2i

|  |  |  |
| --- | --- | --- |
| Proces | Vreme izvršenja | Vreme aktiviranja |
| A | 6 | 0 |
| B | 10 | 3 |
| C | 4 | 6 |
| D | 7 | 9 |
| E | 5 | 12 |

Objašnjenje: Više redova spremnih procesa. {RQ0, RQ1,… RQn}. Proces može da se izvršava 2i vremenskih kvantova, odnosno 2 na broj reda u kome se nalazi proces. E sad dalje: Proces kad dođe on se smešta u red RQ0, ako se ne završi pre isteka vremenskog kvanta onda se prekida I smešta u red ispod tj. RQ1, sad može da se izvršava duplo duže odnosno 2 vremenska kvanta. Svi redovi su FCFS, sem poslednjeg koji je RR.

**Problem: Koliko iznosi n?**

Nema fiksnog broja. Ovde će najviše biti 4 reda.

CRTAJ REDOVE!

RQ0: ~~A~~ ~~B~~ ~~C~~ ~~D~~

RQ1: ~~A~~ ~~B~~ ~~C~~ ~~D~~

RQ2: ~~A~~ ~~B~~ ~~C~~ ~~D~~

RQ3: ~~B~~

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* | \* |  |
| A | \* | \* | \* |  |  |  |  |  |  |  |  |  |  |  |  | \* | \* | \* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| B |  |  |  | \* | \* | \* |  |  |  |  |  |  |  |  |  |  |  |  | \* | \* | \* | \* |  |  |  |  |  |  |  | \* | \* | \* |
| C |  |  |  |  |  |  | \* | \* | \* |  |  |  |  |  |  |  |  |  |  |  |  |  | \* |  |  |  |  |  |  |  |  |  |
| D |  |  |  |  |  |  |  |  |  | \* | \* | \* |  |  |  |  |  |  |  |  |  |  |  | \* | \* | \* | \* |  |  |  |  |  |
| E |  |  |  |  |  |  |  |  |  |  |  |  | \* | \* | \* |  |  |  |  |  |  |  |  |  |  |  |  | \* | \* |  |  |  |

1. **Prednost evidencije blokova podataka datoteke ulančavanjem pomoću FAT u odnosu na klasično ulančavanje blokova datoteke?**

Prednosti su što se sada ceo blok koristi samo za podatke, ne i za pokazivače i brži je random pristup jer tačno znamo koji blok ukazuje na taj konkretan random da tako kažem. Jer kod klasičnog ulančavanja treba da se učita blok da bi se uzela adresa n-tog bloka, a ovde imamo FAT u memoriji pa u memoriji nađemo adresu n-tog bloka što je, naravno, mnogo brže.

**UNIX file sistem ima blokove veličine 4KB, disk adrese od 4B. Koliko ja max veličina datoteke ako se u i-čvoru smešta 10 direktnih blokova, kao i adrese jednostrukog, dvostrukog i trostrukog indirektnog bloka?**

564. strana u knjizi

Veličina bloka = 4KB

Disk adresa = 4B

PROVERI DEO: Veličina bloka / disk adresa = broj adresa, odnosno 212 / 22 = 210 = 1024 adresaima u jednom bloku. Tačno je.

|  |  |  |
| --- | --- | --- |
| Nivo | Broj blokova | Broj bajtova |
| Direktan | 10 | 10 x 4KB = 40KB |
| Jednostruki indirektan | 1024 | 1024 x 4KB = 4MB |
| Dvostruki indirektan | 1024 x 1024 = 1M | 1M x 4KB = 4 GB |
| Trostruki indirektan | 1024 x 1M = 1G | 1G x 4KB = 4TB |

Maksimalna veličina datoteke u ovoj šemi je preko 4TB.

**Napisati pseudokod za čitanje N-tog bloka datoteke.**

U i-čvoru su ti smešteni blokovi datoteke, imaj to na umu sve vreme! Znači pristupaš bloku datoteke! U sistemu Unix V, veličina bloka je 1KB.

Ako je n manje od deset čita se i-node na mestu n. Ako je n između 10 i 1034 onda se pristupa i-nodu na mestu 11 pa se odatle traži adresa n – 10, ( n – deseta ) stavka u tom bloku. Jer 10 si već preskočio kad ovih direktnih, zato sad oduzimaš. Blokovi od 11. do 1034 su u indirektnom, blokovi od 1034 do 1024 x 1024 + 1024 + 10 su u ovim sledećim. Logično je, imaš deset direktnih blokova datoteke, pa jednostruke, pa dvostruke. (Upravljanje datotekama slajd 61/ Knjiga str. 564)

U pseudokodu treba napisati upravo to - koji blok se čita gde.

If (N < 10)

Return read(blok[N]);

Else if (N < 1024 + 10)

Jednostruki = read(blok[11]);

Return read(jednostruki[ N – 10 ])

Else if (N < 1024 x 1024 + 1024 + 10)

Dvostruki = read(blok[12])

Return read(dvostruki[N – 1024 - 10]);

1. **Koja komponenta (sloj) UI SW vrši prevođenje logičke adrese u zadate u obliku broja bloka u fizičku adresu cilindar/staza/sektor, a koja komponenta UI SW se izvršava kada kontroler diska prekidom označi kraj transfera?**

Slojevi UI uređaja - da, praktično drajver to radi. I sloj za planiranje i upravljanje – može tako, to je interrupt handler, onaj najniži sloj.

**Niz zahteva 97, 129, 110, 186, 147, 35, 10, 84, 120 za svaku od sledećih politika raspoređivanja: SSF, SCAN, F-SCAN (ako su posle 3-eg obrađnog zahteva stigli novi zahteva 130, 35, 80). Glava se nalazi na stazi 100, i kreće se prema nižim brojevima staza.**

Papir i olovku u ruke!

SSF: 97, 84, 110, 120, 129, 130, 147, 186, 80, 35, 10

Bira se U/I zahtev sa najmanjim kretanjem glave diska sa njegove trenutne pozicije (Shortest seek first)

*Šta ako 2x imamo isti broj? Mislim da je dovoljno 1 čitanje u tom slučaju* *akko u jednom trenutku postoje 2 zahteva za istom memorijskom referencom. A ako imam zahtev 35, opslužim ga, pa posle ponovo dođe 35, moraću ponovo da pročitam. #SamoKažem*

Tako je.

Objašnjenje kako sam uradio: Pratiš pravilo i kad nisi siguran koji je naredni tada uzmeš prvi veći i prvi manji od trenutnog broja i tražiš manju razliku po apsolutnoj vredosti. Primer – 7, 9, 4 -> |7-9|=2 i |7 - 4| = 3, 2 < 3 tako da je naredni broj 9. Posle 3. Izabranog broja računaš i dodatne brojeve.

SCAN: 97, 84, 35, 10, 80, 110, 120, 129, 130, 147, 186

Glava se pomera samo u 1 smeru zadovoljavajući zahteve sve dok ne dostigne poslednju trasu u tom smeru, zatim se okreće i nastavlja u suprotnom smeru.

F-SCAN: 97, 84, 35, 10, 186, 147, 129, 120, 110, 130, 80, 35

Dva reda, jedan red je prazan za novi zahtev. Redovi se obrađuju po 1 u trenutku, koristeći politiku SCAN. Znači, uradiš sve u prvom nizu SCAN logikom, pa preostale 3 cifre, takođe, SCAN logikom.

**April 2017**

1. **Za šta služi TLB i da li korišćenje TLB-a smanjuje broj grešaka stranica i kako?**

TLB – Translation Lookaside Buffer je HW komponenta u okviru MMU za transformisanje – prevođenje VIRTUELNIH u FIZIČKE adrese bez pristupa tabeli stranica. Ja mislim da ne dolazi do smanjenja grešaka stranica. Zašto: Greška stranice znači da smo pristupili Tabeli stranica i ustanovili da za stranicu koja je nama potrebna bit prisutnosti nije postavljen, tj. Ta stranica nije u glavnoj memoriji i moramo da je pribavimo. E sad, ako ja iskoristim TLB ja ću samo imati fizičku adresu stranice koja je nedavno prevedena, ali ako hoću stranice koje nisu u glavnoj memoriji džabe mi TLB, dolazi do promašaja. TLB samo ubrzava pretraživanje, jer se koristi asocijativno preslikavanje, odnosno moguće je pretražiti ceo TLB jednovremeno zahvaljujući posebnom HW, ali ne smanjujem broj grešaka tako. Barem ja tako mislim.

Tačno, ubrzava prevođenje, ali ne smanjuje broj grešaka.

**Nacrtati prevođenje virtuelne u fizičku adresu koristeći Tabelu stranica u 2 nivoa i TLB.**

*Slika. – poznajući njega, okej je.*

**Vreme pristupa glavnoj memoriji T. Pristup TLB-u T/5. Odrediti procenat pogodaka da pristup memoriji bude 1.6T.**

Te = (1-p)(Ttlb + T) + p(Ttlb + T + T + T)

Objašnjenje:

p – verovatnoća da je nastala greška.

T – vreme pristupa memoriji

Formula: Nije nastala greška treba da pristupimo jednom TLB-u i da pristupimo memoriji (T), ILI (otud plus) nastala je greška pa treba da pristupimo TLB-u pa smo videli da nije tamo pa pristupamo prvoj tabeli stranica, pa drugoj, pa memoriji (otud 3xT)

Nakon svođenja:

Te = Ttlb + T + p(T + T)

Te = T/5 + T + p \* 2T = 1.6T

1.6T = T(1/5 + 1 + 2p)

1.6 = 6/5 + 2p

8/5 = 6/5 + 2p

8 = 6 + 10p

P = 0.2 – 20% grešaka, odnosno 20% pogodaka (1-p)

**2.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Broj virtuelne**  **stranice** | **Stranični okvir** | **Vreme učitavanja** | **Vreme referenciranja** | **R bit**  **Referenciranje** | **M bit**  **Modifikacija** |
| **0** | **2** | **6** | **31** | **0** | **1** |
| **1** | **1** | **13** | **30** | **1** | **1** |
| **2** | **0** | **3** | **35** | **1** | **0** |
| **3** | **3** | **2** | **37** | **1** | **0** |
| **4** | **4** | **9** | **33** | **0** | **1** |

**U trenutku 40 nastaje greška za virtuelnu stranicu 5. Koja će stranica biti zamenjena od strane procesa za svaki od algoritama i zašto:**

1. **FIFO**

Prva stranica u listi je najranije UČITANA u memoriju, a poslednja je najskorije učitana! Znači, gledamo kolonu Vreme učitavanja. Prva učitana je v.s. broj 3 pa će ona biti zamenjena stranicom broj 5.

1. **LRU**

Gledamo sad stranicu koja je najdavnije REFERENCIRANA!

To je virtuelna stranica broj 1. Referencirana u trenutku 30, sve ostale su kasnije pa će ona biti zamenjena stranicom broj 5.

1. **Clock**

Gde se nalazi pokazivač? Ako je na stranici 3 jer je najkasniji trenutak 37 onda će zameniti stranicu 4 jer ona prva nije referencirana. Ako kreće ispočetka zameniće stranicu 0.

Svejedno kako se posmatra, ne treba to razmatrati. bitno je uzeti (0, 1), ako nema (0, 0). Formalno bi trebalo da se pokazivač nalazi na broju virtuelne stranice 2, jer je ona poslednja UČITANA! GLEDA SE ŠTA JE POSLEDNJE UČITANO!

1. **Optimalni**

S obzirom da ne znamo šta će biti potrebno u budućnosti mislim da će prosto zameniti prvu stranicu na koju naiđe, odnosno virtuelnu stranicu 0.

Tačan odgovor je ’’Ne zna se.’’

**3.Objasniti SJF I SRTF i reći zašto ih je teško implementirati na realnim OS?** *Navesti alternativni algoritam sa istim ciljem koji se lako može implementirati.*

Pretpostavljam da je SJF Shortest Job First, tj. Shortest Process Next. Neka je tako.

E ako je tako onda su SJF I SRTF isti. Bira se proces sa najkraćim očekivanim vremenom obrade. Jedina razlika je u tome što kod SJF nisu dozvoljeni prekidi, a kod SRTF jesu. Teško ih je implementirati zato što nije lako odrediti koliko vremena će biti potrebno procesu za izvršenje. ’’Ne može se predvideti budućnost’’.

*FCFS i RR su laki za implementaciiju, a svi od algoritama u ovoj grupi imaju isti cilj, tako da može bilo koji, ali ovi su mi najlakši za implementaciju, ne znam da li se misli na neki konkretan.*

**Drugi deo pitanja je sličan januaru 2017.**

**4. Koliki je minimalan broj disk blokova koji mora biti pročitan da bi se pročitao 215 bajt datoteke /os/ispit/april2017, ukoliko fajl sistem koristi blokove od 1KB, 10 direktnih blokova u i-nodu i 32b adrese blokova i brojeve i-noda. Direktorijum os je veličine 20KB, dok je direktorijum ispit veličine 7KB.**

(Upravljanje datotekama / slajd 91)

215B kad podelimo PO 210 je 32. blok datoteke, znači nije među direktnim nego među jednostrukim indirektnim.

U indirektnom imamo 32b adrese, znači 256 blokova u indirektnom, nama treba 32. blok. Knjiga, strana 564. kaže sledeće: U sistemu UnixV , dužina bloka je 1KB, a svaki može da sadrži ukupno 256 adresa blokova.

Direktorijum os je 20KB pa ima 20 blokova, a ispit 7 blokova. Ako je root direktorijum u glavnoj memoriji onda imamo da pročitamo jedan i-čvor os-a, to je jedan blok, a onda treba da pročitamo njegovih 20 blokova. Ali pošto je 20 blokova, onda je prvih 10 direktnih blokova, jedan indirektni i još 10. Kad vidimo tu sadržaj nađemo broj foldera datoteke ispit, to je plus 1. Svih sedam blokova je direktnih, tu vidimo datoteku april, pročitamo i-čvor i da bi pročitali i-čvor moramo da pročitamo jednostruki indirektni i 22. blok u njemu.

To treba sabrati: (1 + 10 + 1 + 10) + (1 + 7) +( 1 + 1+22)

1. **Šta je vektor prekida, čemu služi, gde je lociran? U kakvoj je relaciji sa drajverom uređaja i rutinom za obradu prekida (interrupt handler). Opisati njegovu povzanost i funkcionisanje na primenu U/I sistemskog poziva receive() za čitanje podataka sa mrežne kartice i koji je redosled događaja za obavljanje ove UI operacije i ko ih generiše. Šta se dešava sa procesom koji je pozvao receive?**

Vektor prekida je adresa interrupt handler rutine, ili tabelu u fiksnom delu memorije gde su za sve prekide stavljene adrese interrupt handler rutina. Zatim mi na osnovu adrese interrupt handler rutine skočimo na adresu nju samu.

**Oktobar 2017**

1. **a) Relokacija procesa i kako se obavlja u okvriru OS?**

Relokacija je smeštanje i pomeranje procesa u glavnoj memoriji. Lokacije koje proces referencira nisu fiksne, one će se pomerati svaki put kada se proces zameni, ili pomeri.

Slika HW podrška za relokaciju (Upravljanje memorijom / slajd 24.)

U osnovni registar se učita adresa u fizičkoj memoriji kada se proces učita, ili zameni. Sve vreme izvršenja programa radi se sa relativnim adresa pri čemu se koristi taj registar sa početkom i granični registar. (316. str. u knjizi)

**b) Navesti bar 2 razloga da se dozvoli da se dva, ili više procesa dele određenu memoriju.**

Ako više procesa izvršava isti program, pogodno je da svaki program pristupa istoj kopiji programa, a ne da ima sopstvenu kopiju.

Procesi koji sarađuju na istom zadatku treba da pristupaju zajedničkoj strukturi podataka.

**c) Interna i eksterna fragmentacija razlika?**

Interna fragmentacija se javlja kod particija fiksnih veličina, kada se program učita u memoriju najverovatnije neće ispuniti celu particiju, ali će se ta particija voditi kao zauzeta, taj neiskorišćeni, a navodno zauzeti deo, se naziva interna fragmentacija.

Eksterna fragmentacija se javlja kod dinamičkog deljenja particija. Tada se između particija nalazi slobodan prostor, ali nije ga moguće iskoristiti jer nije kontinualan.

1. **Dvodimenzionalno polje A[][] = new int [100][100] gde se A[0][0] nalazi na adresi 200 u straničenoj memoriji procesa, veličine stranice 200. Kod procesa se nalazi na adresama 0 – 199. Ukoliko imamo 3 stranična okvira, koliko se grešaka stranica javlja pri izvršenju sledećeg koda za inicijalizaciju matrice korišćenjem LRU algoritma:**
2. **for (int j = 0; j < 100; j++)**

**for (int i = 0; i < 100; i++)**

**A[i][j] = 0;**

1. **for (int i = 0; i < 100; i++)**

**for (int j = 0; j < 100; j++)**

**A[i][j] = 0;**

Mi imamo u 1. straničnom okviru kod procesa i to ne smemo da menjamo, znači ostaju nam 2 stranice gde možemo da upisujemo matricu.

Imamo 10.000 elemenata, stranica veličina 200, pa su ti elementi su u 50 stranica. Svake 2 vrste su u jednoj stranici pošto je u C-u po vrstama smeštanje.

a) Za j = 0 imamo 50 grešaka stranica, za j = 1 ponovo 50 grešaka stranica. Znači 50 x 100 ukupno 5000 grešaka stranica.

b) Smestimo prvu i drugu stranicu i nemamo greške stranica po petlji jot. Kad je i = 2 i 3 smeštamo u drugi stranični okvir i nemamo greške po jot. Kad je i = 4 i 5, 6 i 7 biće grešaka...Poenta: Imamo grešku stranica posle svake 2 vrste, što znači da imamo 50 grešaka stranica.

Napomena: Isuviše je komplikovano, verovatno neće biti.

1. **Kojim procesima / nitima savremeni OS daju prednost, onima koji su CPU, ili U/I orijentisani?**

Prednost uvek imaju U/I orijentisani procesi/niti, zato što će oni kratko zadržati procesor i moći će da čekaju na akciju korisnika. Prioritet je da korisnik ne ispašta. A da prioritet imaju CPU orijentisani procesi onda bi korisnik morao uvek mnogo da čeka.

**Prednosti i nedostaci malog kvanta izvršenja?**

Ne mogu da nađem tačno gde sam pročitao, ali naći ću pa tačno da prepišem, ali pamtim poentu: Kad je kraći vremenski kvant onda je bolje iskorišćen procesor i svi procesi će biti opsluženi barem na kratko. Ipak, nedostak je često menjanje procesa i to zahteva učešće OS.

**Različite dužine vremenskog kvanta u MultiLevel Feedback-u?**

Pa svaki od nivoa ima određeni prioritet. Što je dalji nivo, dalji od jedinice, to će mu se kasnije pristipiti jer se nivou n pristupa tek kad je nivo n – 1 prazan, ali zato mu se dodeljuje veći vremenski kvant da se na taj način kompenzuje kasnije pristupanje, a i pošto se nalazi u daljem nivou znači da mu je istekao vremenski kvant koji mu je prethodno dodeljen i pretpostavlja se da će mu biti potrebno više vremena da se izvrši.

1. **Proveri: Disk kapaciteta 32 bloka od 1024B. Svi blokovi slobodni (prazan je disk). Za slobodan prostor se koristi bit mapa. Fajl sistem dobija zahtev od procesa P8 za upis fajla STUDENTI dužine 9 blokova, a od procesa P2 za upis fajla OCENE dužine 3 bloka. Oba fajla su u direktorijumu FAKULTET. Gde će fajl sistem smestiti ove fajlove ako za dodelu prostora na disku koristi**
2. **Kontinualnu dodelu**
3. **FAT tabelu**
4. **Indeksiranje**

**Prikažite sadržaj direktorijuma i dodeljene blokove fajlovma STUDENTI i OCENE za sva 3 algoritma dodele prostora na disku. Takođe prikažite sadržaj bit mape pre i posle dodele prostora za fajlove STUDENTI i OCENE. Gde fajl sistem čuva bit mapu, a gde direktorijum?**

Bit mapa = Bit vektor

Bit vektor se čuva u slobodnim blokovima na disku. Bit mapa bi trebalo biti na disku, možda bi trebalo izabrati blok 0 da se tu smesti bit mapa.

**Gde se čuva direktorijum?** Disk.

Traženje slobodnog prostora od n blokova se svodi na nalaženje sekvence od n nula u bit vektoru.

1. Kontinualna dodela – fajl STUDENTI će biti smešten od bloka sa indeksom 0 zaključno sa indeksom 8, a fajl OCENE od 9 zaključno sa 11.

Tada će u bit mapi na prvih 12 mesta biti 1-ce, dok će ostaslo biti 0.

Direktorijum Fakultet:

|  |  |  |
| --- | --- | --- |
| Naziv datoteke | Blok početka | Dužina |
| Studenti | 0 | 9 |
| Ocene | 9 | 3 |

1. FAT tabela – ’’Izbor blokova je sada jednostavna stvar – svaki slobodan blok može da se doda lancu.’’ Možemo da izaberemo blokove proizvoljno jer je disk prazan. Najbolje bi bilo da idu sekvencijalno i pošto imamo bit mapu mislim da je to i nalogičnije. Direktorijum ponovo sadrži startni blok i dužnu. Tačno. Samo je sadržaj FAT-a takav da je FAT od 0 na 1, od 1 na 2 i tako dalje.
2. Indeksiranje – Anologno postupku pod b) s razlikom što će direktorijum sada izledati ovako pod pretpostavkom da smo zauzeli prvih 9 pa 3 bloka. Da se u bloku 9 nalazi indeksni blok za Studente i u bloku 13 ind.blok za Ocene.

Bit mapa na prvih 14 mesta ima 1-ce pa onda 0.

Direktorijum:

|  |  |
| --- | --- |
| Datoteka | Indeksni blok |
| Studenti | 9 |
| Ocene | 13 |

1. **Objasniti kako Virtual File System omogućuje da se u okviru OS pristupa različitim fajl sistemima?**

Virtuelni sistem datoteka predstavlja 1 jednoobrazni interfejs od korisničkih procesa ka sistemu datoteka. VFS definiše zajednički model datoteke koji je sposoban da predstavlja svaku zamislivu opštu osobinu ili ponašanje sistema diska. VFS podrazumeva da su datoteke objekti u masovnoj memoriji računara koji dele osnovne osobine bez obzira na ciljni sistem datoteka, ili HW procesora na koji se oslanja.

Opširnije: strana 565.

Preskoci neće da se padne. Možeš da pročitaš razjašnjeno je.

**Datoteka sa 100 blokova. FCB (ili indeksni blok) je već u memoriji. Izračunati broj UI operacija neophodnih za kontinualnu, linkovanu ili indeksiranu dodelu blokova datoteke. U slučaju kontinualne pretpostaviti da nema prostora za širenje datoteke na početku, ali ima na kraju datoteke. Sadržaj novog bloka je u memoriji.**

Blok je dodat a) na početku b) sredini c) kraju datoteke

1. Blok je obrisan sa početka e) sredine f) kraja datoteke

Kontinualna –

Dodavanje: ako je na kraju c) onda se samo doda nakon poslednjeg bloka i to ostaje sve kontinualno, sve je okej, jedna operacija. Ako se dodaje blok u sredini b), onda će polovina morati da se pomeri za po jedno mesto ka kraju dototeke, to je 50 operacija + 1 operacija da se doda odgovarajući blok u sredinu. A za početak će morati sve da se pomeri za po jedno mesto ka kraju i plus doda novi blok, tj. 101 operacija. Tačno.

Brisanje: S kraja nikom ništa – 1 operacija. Sredina 51, izbrišemo 50. blok i preostalih 50 vratimo za pojedno mesto. Početak 1 operacija, samo obrišemo prvi i ćao. Nemamo razloga ništa da pomeramo*.* Tačno.

Linkovana –

Dodavanje: Kad dodajemo na početak to samo treba u direktorijumu, opet to je 1 disk operacija, promenimo broj prvog bloka i da upišemo tu vrednost u blok koji upisujemo. Kao ulančavanje. Kad čitamo disk blok to je 1 disk operacija, kad menjamo to su : čitamo – menjamo – vraćamo.

Indeksiranje – Uzmemo indeksni blok i menjamo ga. Ako dodajemo na početak treba da pomerimo sve i da ga vratimo, to je 1 disk operacija.

**April 2016**

1. **Ceo zadatak: Straničenje sa tablicama u 1 nivou. Max. veličina adresnog prostora je 16MB. Tablica stranica procesa koji se izvršava je:**

**Stranica Stranični okvir**

**0 4**

**1 8**

**2 16**

**3 17**

**4 9**

**Veličina stranica je 1024B, max. Veličina fizičke memorije je 2MB.**

Broj straničnih okvira je 21 – 10 = 11, odnosno 211 straničnih okvira.

* Prvo mi je problem bio šta je adresni proctor, evo šta kaže internet:

An address space is a range of valid addresses in memory that are available for a program or process. That is, it is the memory that a program or process can access. The memory can be either physical or virtual and is used for executing instructions and storing data.

Knjiga: Adresni prostor je opseg adresa dostupnih računarskom programu.

**Koliko b je potrebno za svaki ulaz u tablicu stranica?**

Treba nam 11b za svaki stranični okvir + 3b za present/absent, bit referenciranja i bit modifikacije.

**Koliko b ima virtuelna adresa?**

24b jer je adresni prostor 16MB, a to je 224, ako se adresira svaki bajt. 11b je stranicni okvir, 10b je offset, 3b gorenavedena. Adresni prostor je memorijski prostor koji mi hocemo da adresiramo u tom procesu.

**U koju fizičku adresu se preslikava virtuelna adresa 1524?**

Tipična greška je prevođenje u binarni.

1524 podeliti sa 1024, to je 500. bajt stranice 1. Stranica broj 1, offset je 500. Gledamo sad tabelu, stranica 1 je slikana u stranični okvir broj 8. 8 x 1024 = 8192 + 500 = 8692 to je fizička adresa.

**Koja virtuelna adresa se preslikava u fizičku 10020?**

10020 podelimo sa 1024 i dobijemo da je to stranični okvir 9 i ofset 804. Stranični okvir 9 je virtuelna stranica 4. Virtuelna adresa je 4 x 1024 + 804 = 4900.

1. **Redosled referenciranja: 1 2 1 3 2 1 4 3 1 1 2 4 1 5 6 2 1**

**Prikazati koliko grešaka se javlja u svakom slučaju, ako se koriste tri raspoloživa stranična okvira, i sledeći algoritam:**

1. **LRU**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 |
|  | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 6 | 6 | 6 |
|  |  |  | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 2 | 2 | 2 | 5 | 5 | 5 | 1 |

F F F F F F F F F F F

Ukupno 11 grešaka kod LRU.

Objašnjenje: Gledaš na levo, šta je najdavnije referencirano i to menjaš.

1. **Clock**

Polazimo od toga da se ne menja pokazivač pri referenciranju stranice koja je unutar bilo kog od 3 straničnog okvira. Znači, ako je pokazivač na prvoj vrsti a traži se stranica u trećoj ta stranica će moći da bude preuzeta, a da pokazivač ostane gde jeste. Pogledaj prezentaciju ’Virtuelna memorija’, slajd 44. poslednju kolonu za Clock. Tu se to vidi.

Napomena: Obavezno crtaj strelicu gde ti se nalazi pokazivač!

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1\* | 1\* | 1\* | 1\* | 1 | 1\* | 1\* | 1\* | 1\* | 1\* | 1\* | 1\* | 1\* | 1 | 1 | 2\* | 2 |
|  | 2\* | 2 | 2 | 2\* | 2 | 4\* | 4\* | 4\* | 4 | 2\* | 2\* | 2\* | 5\* | 5\* | 5\* | 1\* |
|  |  |  | 3\* | 3\* | 3 | 3 | 3\* | 3\* | 3 | 3 | 4\* | 4\* | 4 | 6\* | 6\* | 6 |

F F F F F F F F

Osam grešaka kod Clock-a.

Objašnjenje: Imaš pokazivač koji se šeta pri traženju stranice. Vidi gorepomentu sliku. I nakon referenciranje i ucitavanja stranice pokazivač se pomera na narednu poziciju. Jako bitno!

1. **Optimalni**

**Redosled referenciranja: 1 2 1 3 2 1 4 3 1 1 2 4 1 5 6 2 1**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|  | 2 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 5 | 6 | 6 | 6 |
|  |  |  | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |

F F F F F F F

Sedam grešaka u optimalnom algoritmu.

Objašnjenje: Gledaš šta je najkasnije referencirano od 3 stranice koja imaš u straničnim okvirima. Gledaš na desno.

1. **Razlika između planiranja (preemptive) sa i bez prekidanja (non-preemptive). Ostatak pitanja je standardno januar 2017.**

Režim izbora određuje trenutke u vremenu u kojima se izvršava funkcija izbora. Postoje 2 opšte kategorije:

Bez prekidanja – Proces je u stanju izvršavanja sve dok se ne završi, ili se sam ne blokira da bi čekao na UI ili da bi zahtevao neku uslugu OS

Sa prekidanjem – Proces koji se izvršava može da se prekine i smesti u stanje spreman. Odluka o prekidanju se donosi kada dođe novi proces, ili kada se desi prekid koji stavlja blokirani proces u red spremnih ili periodično na osnovu prekida iz generatora takta.

1. **Razlika između drajvera i kontrolera uređaja i osnovne funkcije.**

Drajver predstavlja SW komponentu za upravljanje UI uređajem, dok je kontroler HW komponenta. Kontroler ima svoje registre, memoriju i CPU specjalne namene. U njegove registre drajver smešta komandu, ili sekvencu komandi u zavisnosti od postojanja bafera; kontroler te komande izvršava i ako je nastala greška vraća zahtevaocu kod greške.

**Sistem sa diskom od 8 sektora po stazi i 512B po sektoru. Disk se rotira brzinom od 3000rpm i ima prosečno vreme traženja (seek time) od 15ms. Posmatrati datotetku koja ima 8 blokova pri čemu 1 blok zauzima 1 sektor i izarčunati ukupno vreme neophodno za pristup celoj datoteci ako se koriste metode alokacije**

1. **Kontinualna**
2. **Indeksiranje**

8 sektora

1 sektor = 512B

r = 3000 rpm / 60= 500rps => 1/2r = 1/ 2 x 500 = 1 / 1000 = 10^(-3) = 1ms; Odavde zaključujemo da je rotaciono kašnjenje 1ms.

Ts = 15ms

1 sektor = 1 blok

Ta – vreme pristupa datoteci

1. Ta = Ts + 1/2r + b/rN = 15ms + 1ms + vreme čitanja 8 sektora

Vreme čitanja osam sektora je jednako čitanju 1 cele staze, odnosno: N / r x N = 1 / r = 2ms

Ta = 15 + 1 + 2 = 18ms

Napomena: Kod kontinualne dodele samo jednom računamo vreme pozicioniranja, preostale staze, ukoliko ih ima, se čitaju bez ikakvog vremena pozicionaranja jer je glava već tu gde treba.

1. Sad stvari počinju da se lome. :D Sad moramo da računamo da pomeramo glavu svaki put. Već smo odredili da je rotaciono kašnjenje 1ms, to ostaje isto. Kao i vreme traženja. Ako mi celu traku pročitamo za 2ms, onda jedan sektor pročitamo za 0.25ms. (2ms / 8)

Sledi, Ta = (15 + 1 + 0.25) x 8 = 130ms

1. **Prikazati strukture koje Fajl sistem koristi kod evidencije sobodnog prostora na disku.**

Koristi se bit vektor, ili lista pokazivača na slobodne blokove. Detaljnije – prezentacije.

**Fajl sistem X koristi:**

**1 blok = 8 reči**

**1 reč = 4B**

**Disk koji se koristi za smeštanje fajl sistema ima 32 bloka.**

**Inicijalni blok, blok 0, sadrži ulaz direktorijuma koji se sastoji od imena fajla i pointera na prvi i-čvor u bloku 1.**

**Struktura je data na slici.**

|  |  |
| --- | --- |
| **Reč** | **Vrednost** |
| **0** | **Pristupne dozvole** |
| **1** | **Veličina fajla u rečima** |
| **2** | **Direktni blok** |
| **3** | **Direktni blok** |
| **4** | **Direktni blok** |
| **5** | **Direktni blok** |
| **6** | **Jednostruki indirektni** |
| **7** | **Dvostruki indirektni** |

**Fajl sistem X ima trenutno samo fajl PET dužine 16 reči podataka: prvi direktni indeks ukazuje na blok 31, a drugi na blok 29. Blokovi diska 4, 7, 10, 15 su označeni kao neispravni. Slobodni blokovi se dodeljuju u redosledu 0, 1, 2,... 31. Operacija WRITE upisuje 1 blok. Ako fajlu PROBA treba dodati 100 reči koliko je WRITE operacija potrebno za ovo dodavanje? Prikazati stanje i-čvora, dodeljene i slobodne blokove pre i posle ove operacije dodavanja.**

Ako se dodaje 100 blokova onda je to 13 blokova novih moramo da upišemo. Druga stvar koja je bitna je da nam treba jednostruki i dvostruki indirektni blok. Fajl PET ima samo 2 direktna bloka. Mi kad dodamo 13 nama treba 2 ova... Bitno je ovde da je disk adresa 1 reč. Ako je ovo 1 blok disk adresa je 1 reč. U jednom indirektnom bloku imamo 8 disk adresa. Nama treba 13, znači 2 + 8 i treba nam dvostruki indirektni za još tri bloka, odnosno adrese. Znači samo treba staviti pravougaonik od 0 do 31 i markirati samo one koji su neispravni i alocirati. Treba nam 13 novih blokova i treba nam još 3 indirektna bloka. Jednostruki indirektni i 2 dvostruko indirektna. I samo se redom slažu brojevi i adrese.

**Septembar 2016**

1. **Straničenje u 2 nivoa. Fizička memorija 234B, stranice 216B, a ulaz u tablicu stranica 4B**

**Odrediti:**

1. **Koliko b ima logička (virtuelna) adresa?**

U straničenju u 2 nivoa, svaka tabela stranica i 1. i 2. nivoa je veličine stranice. Ako je stranica 216  tabela prvog nivoa ima 214, znači 14b treba da se indeksira ulaz i u 1. i u 2. tabelu stranica. Ovo smo dobili jer je vrsta u tablici stranica 4B.

Kao što smo na slajdovima imali primer za straničenje u 2 nivoa 10, 10, 12, ovde je 14, 14, 16.

Pa je ukupna virtuelna adresa 44b!

1. **Koliko b je ofset?**

16b je ofset, jer je veličina stranice 216

1. **Veličinu str. Okvira?**

Stranični okvir je takođe 216, iz istog razloga.

1. **Koliko ulaza u tablicu prvog i drugog nivoa ima?**

214

1. **Koliko b u fizičkoj adresi specificira stranični okvir?**

16b ponovo.

1. **Isto kao April 2017.**
2. **Januar 2017.**
3. **Objasniti ukratko:**

**FCFS -** Prvi došao, prvi uslužen. Zahtevi se obrađuju sekvencijalno. Fer za sve procese.

**SSF –** objašnjen

**SCAN –** objašnjen

**C-SCAN –** Ograničava kretanje na samo 1 pravac. Kada se dostigne zadnja trasa u jednom smeru, ili kada više nema zahteva u tom smeru, glava se okreće na drugi kraj diska i tada se skeniranje ponovo startuje.

**Ostatak pitanja je poznat. Uradi kad stigneš.**

1. **Windows 98 koristi lančanu listu blokova diska za svaku datoteku, koja se memoriše u FAT32, a UNIX koristi i-čvor. Pretpostavimo da oba OS-a koriste blokove diska veličine 4KB i adresu bloka od 4B.**

**Posmatrajmo datoteku veličine 2MB (tj. 512 blokova). Objasnite šta Windows98, a šta UNIX treba da uradi da bi pročitao poslednji bajt iz ove datoteke,a šta da bi dodao novi blok (tj. 4KB novog sadržaja) na početak datoteke?**

Za Unix je priča sa i-čvorom, ide redom dok ne dođe do praznog pokazivača i to je onda poslednji bajt. A Win98 da ide kroz FAT dok ne dođe do poslednjeg.

A za dodavanje Unix samo da doda na kraj, u FAT-u da se prelanča.

Tu brojke nisu toliko bitne, one su onako date.

**Oktobar 2016**

1. **Virtuelna adresa 47b, veličina stranice 16K = 214, svaki ulaz u stranicu 8B.**
2. **Koliko nivoa tablica stranica je potrebno da bi se mapirao v.a. prostor ako se zahteva da svaka tabela stranica bude smeštena u 1 stranicu?**

14b je za ofset. Svaki ulaz u tabelu stranica je 8B, što znači da jedna tabela stranica ima 211 ulaza/stavki. Jer je veličina tablice jednaka veličini stranice odnosno 214 , 214 / 23  = 211. Znači treba nam 11 bitova u adresi da indeksiramo jednu tabelu stranica ako ima 211 stavki. 11b za jednu tabelu stranica, 11b za drugu, 11b za direktorijum.

11b + 11 + 11 + 14 = 47

Znači 3 nivoa je odgovor.

1. **Objasniti detaljno i dati strukturu virtuelne adrese.**

Virtuelna adresa je u formatu <stranica, pomeraj>. Stranica se koristi pri pristupanju Tabeli stranica na osnovu koje se određuje fizička adresa odnosno stranični okvir kome se pristupa sa pomerajem navedenim u drugom delu adrese. Ovde se za stranicu koristi 33b, a za pomeraj 14b. Ipak, kako se koristi adresiranje u 3 nivoa, prva 33b biće podeljeni na 3 dela, po jedan za pristup svakom nivou.

1. **Veličina fizičke memorije je 1GB, koliko b je potrebno za svaki ulaz u tablicu stranica?**

1GB = 230

30 – 14 = 16b => 216 straničnih okvira je u toj fizičkoj memoriji, pod preptostavkom da i dalje važi da je veličina stranice 16KB.

Tako da je moj konačan odgovor da je potrebno 16 + ona 3 b, kako bi se adresirali svi stranični okviri.

1. **Analogno Januaru 2017.**
2. **Šta je prioritet sa prekidanjem?**

Kad se pojavi process višeg prioriteta da može prekinuti tekući proces.

1. **Zašto je korišćenje bit vektora za evidentiranje slobodnih blokova** *pouzdanije* **I boljih perfromansi u odnosu na listu slobodnih blokova?**

Prednosti bit vektora:

Lako je pronaći slobodan blok, ili susednu grupu takvih blokova.

Mala je koliko je to moguće. Veličina diksa / 8 x veličina bloka sistema datoteka.

Ako se jedan blok ošteti gubimo lanac kod liste slobodnih blokova, a kod bit vektora svaki bit živi svi jo život.

1. **Slično sam već uradio.**