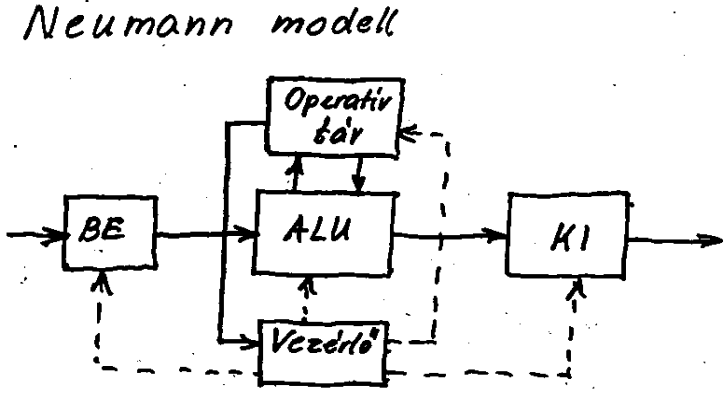
1.,
a., Adott egy két utas set asszociatív cache. Hogyan változhat egy ilyen cache találati aránya egy vele megyegyező méretű direkt leképezésű cache-hez képest? Indokolja a választ! Használhatja-e a vezérlő direkt leképezés esetén az LRU blokkcsere stratégiát? Indokolja a választ!
A két utas set asszociatív cacheben a direkt cachehez képest azonos „modulóval” (talán még emlékeztek, hogy a cache-t blokkokra osztottuk és az adott blokk kezdőcímét osztottuk a cache méretével és az osztás maradéka adta meg, hogy hányadik blokk helyére került be a cahcbe a kiválasztott blokk. Más néven ez volt a CBA.) két blokk is szerepelhet, ergó két helyen is benn lehet az adat, a találati arány mindenképpen nő. (Gáz lenne, ha férfi lenne.) Triviálisan direkt cachenél semmilyen blokkcserélési stratégiát nem alkalmazhatunk, hiszen ott az adatok helye egyértelmű, ettől direkt a leképezés.
b., Rajzolja fel a digitális számítógép Neumann-féle modelljének blokkvázlatát, sorolja fel a modell működését meghatározó alapelveket.



Könyv eleje, illetve első diák valamelyike.
A modell működését az alábbi alapelvek határozzák meg:
- Belső programtárolás
- Adat és program nem megkülönböztetett, a memóriarekesz tartalmát csak az utasítás-számláló értéke mondja me
- szekvenciális utasításvégrehajtás

c., Magyarázza el, hogyan lehet a négycímes számítógépeknél alkalmazott megoldásból 3, illetve két címes megoldást létrehozni.
A 4 címes utasításban az alábbi mezők szerepelnek:

műveleti kód	első op. címe	második op. címe	eredmény címe	köv. utasítás címe

Ebből 3 címes utasítást úgy tudunk csinálni, hogyha rögzítjük a következő utasítás címét, például olyan formában, hogy az mindig az utasítást követő cím. (köv ut. címe=utasítás címe+utasítás hossza)

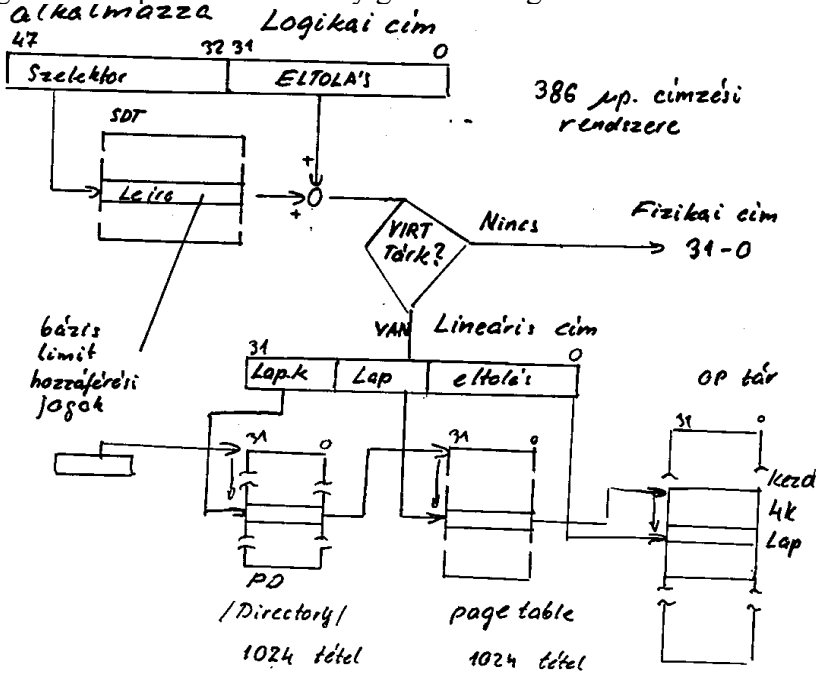
műveleti kód	első op. címe	második op. címe	eredmény címe

Ebből a két címes utasítás úgy keletkezik, hogy rögzítjük az eredmény helyét, mondjuk az első operandus címe legyen az.

műveleti kód	első op. címe	második op. címe

2.,

a., Rajzolja le a 386 mikroprocesszornál, bekapcsolt virtuális tárkezelés (lapszervezés) esetén, logikai cím-fizikai cím transzformációjának blokkvázlatát! A szegmentálás milyen hátrányát küszöböli ki a szegmentált lapszervezés? Hány globális szegmensleíró tábla lehet?



A szegmentálás alapvető hátránya, hogy egybefüggőnek kell lennie a szegmenshez rendelt címtartománynak, ebből következően a memóriában rendelkezésre kell álljon egy ilyen nagyságú címtartomány szabadon. Előfordulhat azonban, hogy bár rendelkezésre áll az adott méret a memóriában, de nem egybefüggően. Ezt a memória töredezésének hívjuk. Ezen a lapszervezés úgy segít, hogy már csak a logikai címtartománynak kell összefüggőnek lenni, a lapleíró táblán keresztül már tetszőleges módon bepakolhatjuk az adatokat a tárba. Biztos, hogy nem lesznek lyukak a memóriában, ellenben a szegmens most már mindenképpen lefoglalja a memóriából a méretéhez szükséges lapszámot, vagyis a keletkező lyukak most a szegmensek végén lesznek. Ez előnyösebb, mert így viszont a rendelkezésre álló memória nagyobb hányadát tudjuk kihasználni.
Ezen felül kell a kétlépcsős leképezés, mert különben túl nagy memóriaterületre lenne szükség a cucchoz.

b., Sorolja fel az i386 mikroprocesszor szegmentálásnál alkalmazott privilégium szintjeit, és a hozzájuk kapcsolódó elérési szabályokat!

386-ban 4 privilégium szint van, melyeket a sorszámukkal jellemzünk. 0 a legfontosabb, 3 a legkevésbé fontos. Ha nem akarunk privilégiumokat definiálni, akkor célszerűen állítsuk mindet 0-ra. (Ugyanis csak ezen a szinten engedélyezett minden utasítás.) Egyszerű esetben az operációs rendszer a 0-s és minden más a 3-as privilégium szintet használja. Tipikus esetben a Kernel, vagyis az OS magjának privilégiuma a 0-s, az operációs rendszer privilégiuma az 1-es, az ún. „3.rd party product” (más cégek termékei) a 2-es és a felhasználói programok a 3-mas szintet kapják.
A privilégiumokra érvényes az, hogy másik privilégiumszint szegmenseit, kódjait, szóval más besorolás alá eső adatot nem érhetünk el közvetlenül, vagyis csak a saját szintünkön mozoghatunk kvázi büntetlenül. Bizonyos utasítások végrehajtását is korlátozzák az egyes privilégiumok. Ez megnehezíti azonban például a perifériák elérhetőségét, tekintve, hogy az I/O tkp. a kernel feladata, mivel I/O utasítást triviálisan csak a 0. privilégiumszint adhat ki. Így szükséges bevezetni olyan speciális elérési pontokat melyeken keresztül a más privilégiumszintre eső adatokat elérhetjük. (Ez a tulajdonképpeni rendszerhívás.)

c., Multitaszkos rendszernél mi a különbség a fizikai és virtuális processzor között? Hogyan teremthető kapcsolat a kettő között?

A fizikai és a virtuális processzor között az a különbség, hogy a virtuális processzort folyamatokra értelmezzük és az adott folyamat számára az egyes virtuális processzorok úgy látszanak, mintha

nekik egy teljes processzoruk lenne. A valóságban ezeknek egyetlen processzora van, de mivel az egyes taszkok közül egy időpillanatban csak egy fut, így az egyes processzek időben osztoznak a fizikai processzoron, vagy processzorokon. A taskok cseréje a környezetváltás, melynek során az egyes taskok állapotát lementjük és a fizikai processzort átadjuk egy másik tasknak. Ez a fizikai processzor tulajdonképpeni hozzárendelése a virtuális processzorokhoz.
3.,
a., Multibus II. rendszernél az A, B, C egység az alábbi arbitrációs azonosító kódot adja a buszra.

	ARB5					***ARB0***
A	1	0	0	0	1	0
B	1	0	1	1	0	0
C	1	1	1	1	0	1

Mutassa be, hogyan dől el, ki kapja meg elsőnek a buszvezérlés jogot! Mi a további sorrend, ha a másodikként kiszolgálásra kerülő master buszvezérlése alatt az elsőként kiszolgált masternél újabb buszvezérlési igény jelentkezik? Indokolja a választ!

A sorrendet sokat kitalálták, vagy megcsinálták, vagy mi a szösz, azért én is vázolom.

	ARB5					***ARB0***
A	1	0	0	0	1	0
B	1	0	1	1	0	0
C	1	1	1	1	0	1
	1	0	0	0	0	0

	ARB5					***ARB0***
A	1	0	0	0	1	1
B	1	0	1	1	1	1
C	1	1	1	1	1	1
	1	0	0	0	1	1

	ARB5					***ARB0***
A	1	0	0	0	1	0
B	1	0	1	1	1	1
C	1	1	1	1	1	1
	1	0	0	0	1	0

Kiemeltem az 1-eseket, hogy jobban átlássuk.
Szóval első lépésben vesszük a cumóink bitenkénti és kapcsolatát. Az eredményt leírjuk, majd ugrunk és elkezdjük a második táblázatot kitölteni. A második táblázatba addig írjuk soronként az eredeti kódot, ameddig az megegyezik az előző művelet eredményével. Ahol eltérés van, oda egyest írunk és az utána következő biteket is egyesnek vesszük. Az így nyert átmeneti (pszeudo) kódot ismételten logikailag összeséseljük és a 3. táblázatba azoknak a soroknak visszaírjuk az eredeti kódját, melyeknek pszeudo kódja megegyezik az eredménnyel. A többi sort meghagyjuk mezei pszeudo kód alakjában. Az így képzett ismételt logikai és pontosan egy sort választ ki.

b., Hány db. megszakításkezelő és hány db. megszakításkérő egység lehet egy VME rendszerben? Indokolja a választ
Megszakításkezelő arhitektúrálisan 7, mert 7 IT vezeték van és megszakítás-kérő már lehet akár 256 is, mert ugye a megszakítás buszvektoros és ilyenkor egy 8 bites kódot rak ki a kérő a vonalra, ami pontosan 256 különböző esetet jelent.
c., Sorolja fel a buszmegszerzési stratégiákat! Melyiket milyen masterekhez célszerű alkalmazni?
Buszmegszerzési stratégiák az alábbiak:
közös kérés, közös válasz (Daisy Chain)
független kérés független válasz (Round Robin)
egyenlő esély (Fairness)
Egyenlő esélyt akkor célszerű alkalmazni, ha a rendszer lassú perifériákat köt össze és a perifériák üzenetváltásos rendszerben kommunikálnak. A felfűzést akkor jó használni, ha szeretnénk prioritási sorrendet felállítani az egyes materek között, ugyanakkor nem szeretnénk túlzottan bonyolult arhitektúrát alkalmazni a kérések és válaszok kezelésére. Tessék vitatkozni. :-D

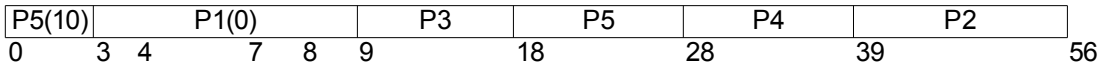
4.,
a.,Adja meg a bináris szemafor definíciós programját (pszeudo kód)!
Várakozás:
wait/sleep:
P(s):
while s<=0 do {};
s:=s-1;
wakeup/signal:
V(s):
s:=s+1
Bináris, ha s=1; kezdetben.

b., Mutassa be a bináris szemafor egy multiprogramozott rendszerben használható megvalósítását (pszeudo kód). Felhasználhatja az operációs rendszer folyamatkezelő műveleteit (sleep, wakeup) valamint listakezelő műveleteket is használhat.
Erre lusta vagyok.

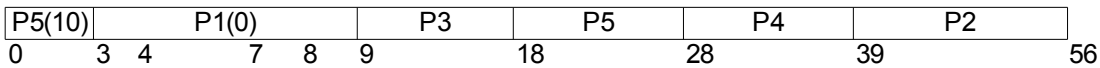
5., Rajzolja fel a lefutás Gantt-diagrammját, és határozza meg az átlagos átfutási idő értékét a táblázatban megadott terhelés esetén a következő CPU-ütemezési algoritmusok alkalmazása mellett:
- SJF (legrövidebb löketidejű)
- SRTF (legrövidebb maradékidejű)
- preemptív statikus prioritásos
- időszeleteléses (Round Robin), prioritás nélkül, időszelet: 6 egység

folyamat	*érkezési idő*	*következő cpu löket hossza*	*prioritás*
P1	3	6	2
P2	8	17	5
P3	4	9	1
P4	7	11	4
P5	0	13	3

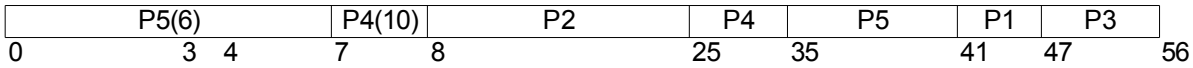
Nagyobb prioritási szám a fontosabb folyamatot jelöli.
SJF:



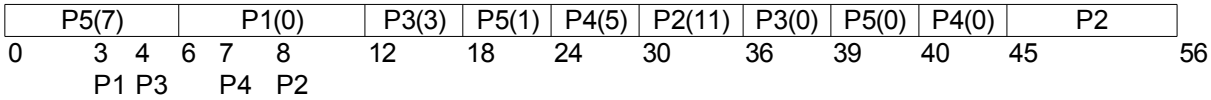
SRTF



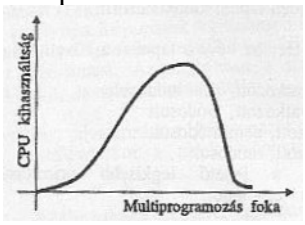
PS:



RR:



6.,
a., Rajzolja fel a multiprogramozás foka függvényében a CPU-kihasználás görbéjét egy virtuális tárkezelést alkalmazó multiprogramozott operációs rendszerben. Magyarázza meg a görbe alakját.



Scanelt opre könyv 72. oldala.
A görbe alakjára a magyarázat kb. ez lenne az én mezei logikámmal:
Kezdetben vala a multiprogramozatlan rendszer. Ebben a rendszerben annyi volt a hatásfok amennyi. Aztán jött a szoftver és elkezdte aztat használni és mivel a processzor hardveresen támogatott egy csomó mindent, így bizonyos dolgokat már hipp, meg hopp meg tudott csinálni, mert a hardverhez nem kellett neki szoftverutasítások hadát lehozni a memóriából. És mivel ilyen jó volt neki a processzor lekezdett további programokat behozni. Ennek több vége is lehet. Például az, hogy a proci annyi processzt behoz, hogy nem lesz elég lapocska és a processzek csúnya gonosz bitsorozatokként követnek el laprablásokat egymástól. Másik hiba lehet, hogy nincs elég lapja a folyamatnak a tárban és emiatt gyakran hív be a háttértárról adatot.
b., Milyen lényeges tulajdonságai vannak a memóriának, mint erőforrásnak, a holtpontkezelés szempontjából.
A memória holtpontkezelés szemszögéből többszörös erőforrás, ami menthető.