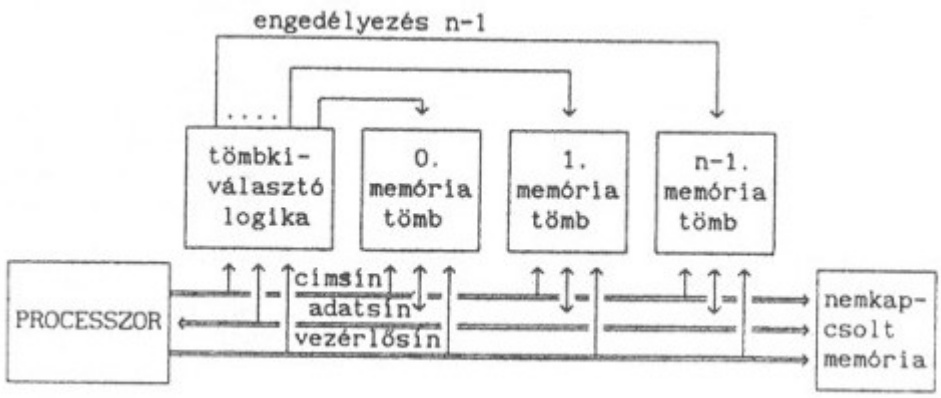
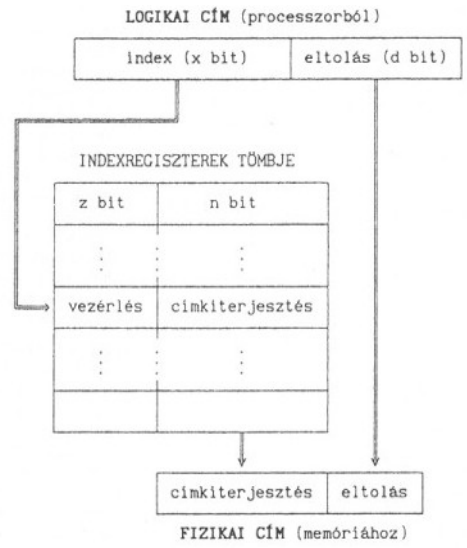
2004. 01. 08. Informatika IC1 -- Móczár Géza (1-3.), Kondorosi Károly (4-6.)

== 1 ==

a) Rajzolja fel a tömbkapcsolásos memóriakezelő blokkvázlatát! Írja le működését.
Blokkvázlat:



Működés:
A cucc kvázi egyszerűen működik. A célja az, hogy a processzor címtartományát kibővítse, amit úgy ér el, hogy van egy tömbkiválasztó logika, amit a processzor az adatsínjén keresztül tud például vezérelni. Pl. a tömbkiválasztót mint i/o perifériát megcímezve kap egy kódot, amiből tudni fogja, hogy melyik memóriatömböt kell neki aktuálisan engedélyezni, így az adott címtartományra párhuzamosan bekötött modulok között tud váltani. Hozzátartozik, hogy kell legyen olyan memóriarész, mely nem kapcsolt, ugyanis az alapprogramnak valahol el kell helyezkedni. Ez egy egyszerű és könnyen bővíthető memóriakiterjesztés.



b) ***Indexelt leképzés*** esetén a logikai cím ***16*** bites, melyből a legmagasabb helyértékű ***4*** bit az index. A fizikai memória mérete ***2 Mbyte***. Számítsa ki mekkora az indexregisztertömb mérete, ha vezérlésre 2 bitet alkalmaz!
Adott egy 2000H kezdőcímre címfolytonosan lefordított ***12 KB*** méretű program. A fizikai memóriában az ***512 kB..520 KB*** (080000H-081FFFH) és az utolsó ***8 KB*** tartományban (1FE000H-tól) van szabad memóriahely. Írja fel az indexregisztertömb ***programhoz tartozó*** regisztereinek sorszámát és a címrészének az értékeit!
A feladat első fele nagyon primitív. A számítás menete az oldalsó ábra alapján egyszerűen megérthető. A lényeg, hogy x=4, d=12 hiszen x+d=16 és x=4 adott. Továbbá 1Mbyte 20 címbitet igényel, akkor a 2 Mbyte 21-et. Ebből n=21-12=9. Tehát az indexregisztertömb 16 bejegyzést tartalmaz, melyek hossza z+n=11 bit.
A kérdés második fele már picit bonyibb, de annyival nem, mint amennyinek látszik. Ismerve, hogy az offset 12 bit, így gyorsan kiszámolhatjuk, hogy 10 bit 1 kB, 12 bit akkor 4 kB adatot címez. Ergó egy bejegyzés az indexregisztertömbben pontosan 4 kB-s. Ez azt jelenti, hogy a 12 kByte 3 részre oszlik fel és 3 bejegyzés kerül az indexregisztertömbbe. Látszik, hogy az első 8kB az 512kB utáni részre elfér, illetve a maradék 4-et a memória végére kell pakolnunk. Most bontsuk fel a 2000H címet:
2000H= 0010 0000 0000 0000 B
4 kB-vel arrébb:
3000H= 0011 0000 0000 0000 B
4 kB-vel arrébb:

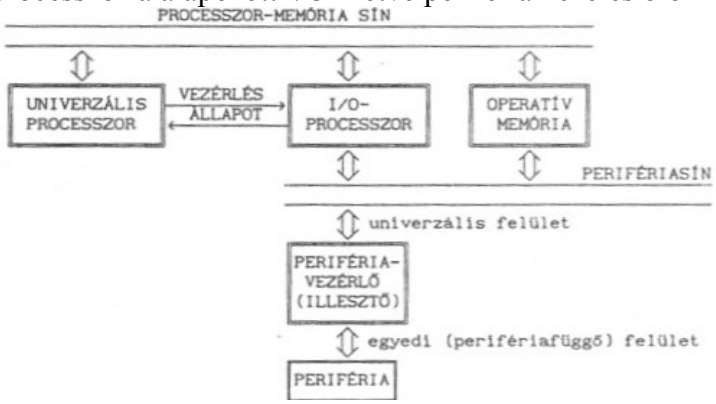
4000H= 0100 0000 0000 0000 B
Ugye mivel a felső 4 bit az indexregisztertömb egyes sorait választja ki, így látszik, hogy a programhoz tartozó rekeszek a 2-es, 3-mas és 4-es rekeszek lesznek, tehát azok sorait kell kitölteni. Azok sorai innentől pedig kvázi egyszerűen adódnak, mert ugye a második és a negyedik sor kezdőcímei a feladatkiírásban adottak, a harmadikat pedig mondjuk így számíthatjuk:
08 0000H + 1000 H = 08 1000H
Így tehát az indexregisztertömb az alábbi módon néz ki:

sorszám	*címkiterjesztés*
2.	08 0H
3.	08 1H
4.	1FEH

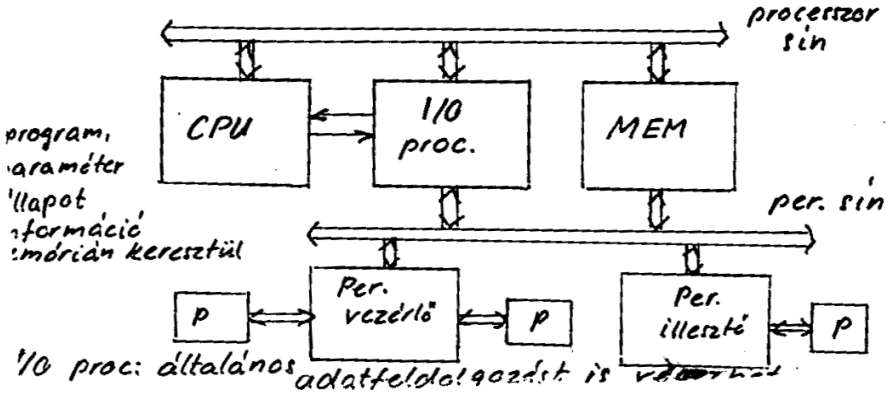
Megjegyzés: A 3 blokkot a memória megadott 4 db 4kB-s részén tetszőleges módon elhelyezhetjük a saját perverziónknak megfelelően. A három darab 0 a címkiterjesztésben azért nem szerepel, mert ugye az csak 12 bit így az alsó 12 bitet egyszerűen levágtuk.

c) Mi a különbség a ***write through cache*** írási stratégia és a lapszervezésű ***virtuális tárkezelés*** írási stratégiája között? Indokolja röviden a válaszát!
A write-through cache esetén az írás módosítja a cache-t is és az eredeti tartalmat is (az operatív memóriában). Virtuális tárkezelésnél íráskor a módosítást csak a memóriában végezzük el (beállítjuk a lap ún. dirty bitjét); és csak később -- amikor a memóriába ennek helyébe új lapot hozunk be -- írjuk ki a lemezre a módosítást.

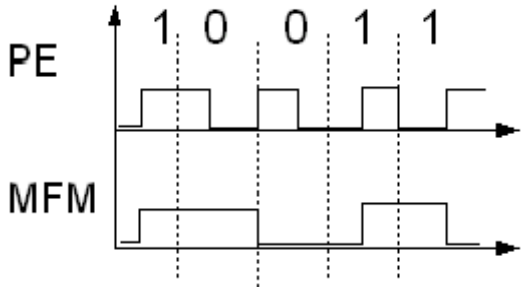
== 2 ==
a) ***Rajzolja*** fel az I/O processzorra alapozott I/O illetve periféria kezelés blokkvázlatát!



Tk. 44. oldal 2.15 ábra. Bár szerintem a dián a 7/3-as sokkal jobb és szemléletesebb.



b) ***Rajzolja fel PE és MFM*** kódolás esetén a mágnesezettség (vagy íróáram) jelalakját a következő bitsorozatra: ***10011***

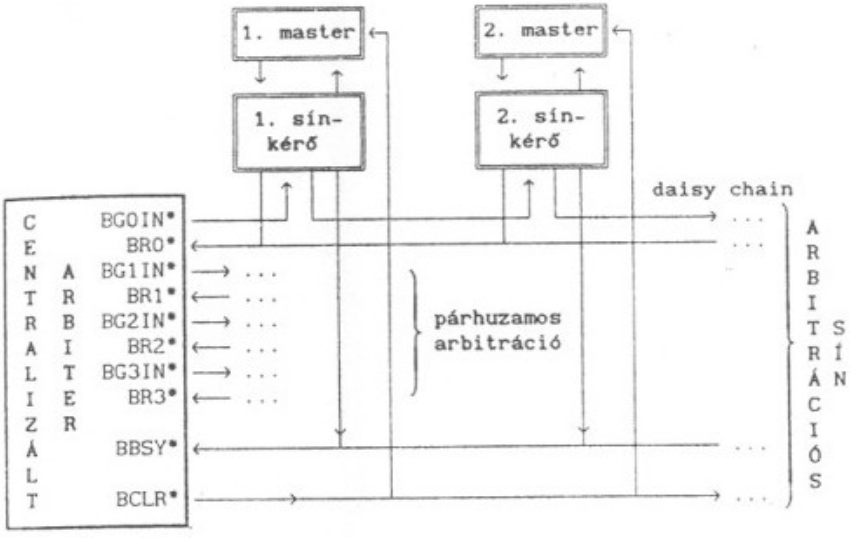


Az elv:
PE-nél, 1 esetén felfutó átmenet, 0 esetén lefutó.
MFM-nél, 1-es esetén átmenet két órajel között, 0 eseteb átmenet a közvetlenül előtte lévő órajelen (KIVÉVE az 1-es után következő első 0-át)

c) Floppy diszknél ***szoftszektoros*** szervezés esetén mi a ***mark*** (jelző) szerepe? Hogyan valósítják meg?
Mark jelöli a szektor kezdetét (egyébként nem lehetne azonosítani) szándékosan kódolási hibával van megvalósítva.

d) ***Multiprocesszoros*** szorosan csatolt rendszereknél ***milyen hardver*** támogatás szükséges a szemaforkezelésnél szükséges ***kölcsönös kizárás*** biztosításához? ***Írjon*** példákat a megoldásra!
A hardvernek támogatnia kell olyan utasítást, ami oszthatatlanul vizsgál és állít be egy memóriacellát (TEST-AND-SET) vagy olyan utasításra, amely biztosítja azt, hogy ameddig a következő utasítás le nem futott a processzor HOLD kérésnek nem tesz eleget (LOCK), illetve külön kérésre lehetővé kell tennie az adott memóriarekesz eléréséhez szükséges busz lefoglalását hosszabb időre (több utasításon keresztül.)

== 3 ==
a) ***Rajzolja fel*** a **VME** ***busz kombinált arbitrációs*** rendszerének ***blokkvázlatát***! Hány **arbitrációs modul** (arbitter) és hány **master modul** lehet egy ilyen rendszerben? **Indokolja a választ!**



137. oldal 4.7-es ábra.
Arbitrációs modul logikailag csak egyetlen egy lehet a rendszersín szempontjából. (A VME tipikusan centralizált arbitrálású, egy decentralizált arbitrálású rendszer esetén attól függ, hogy

soros, vagy párhuzamos a decentralizálás (Multibus I és II.) Ha MBI. (soros), akkor max 4 master lehet, mert volt egy olyan megkötés, hogy 1 órajelen belül végig kell terjedjen a láncon a jel. Arbiter ilyenkor nincs. Párhuzamos esetben a sínre több arbiter csatlakozik, melyeket az arbitrációs vonalak vezérelnek. Itt a maximális száma az arbitereknek szerintem ARB0-4 miatt 32 lehetne, de kártyahely csak 20 van, so 20. Masterek száma az arbiterekre kapcsolható masterek számától függ.) Master modulok számát csak a Daisy chain elve szabja meg, ami szerint a lánc tetszőlegesen hosszú lehet, de azért nyilván a rnedszer szempontjából a terjedési időt célszerű figyelembe venni.

b) ***Milyen hiba*** léphet fel ***teljesen kapcsolt aszinkron*** protokollt használó rendszerben egy ***nem létező címre*** történő íráskor illetve olvasáskor? ***Hogyan kezelik*** ezt a problémát?
Átfogalmaztam az eredeti választ picit:
A reteszelt átvitelben ugye várunk egy nyugtát a vevőtől. Azonban a kiküldött adat, ha fals címre ment, válasz soha nem érkezik, tehát a rendszer alapesetben „lefagy". Erre a megoldás az, hogy bevezetünk egy időzítőt, amely küldés után rögtön indul és ha az időzítés lejártáig nem jön válasz (nyugta), akkor hibajelzést adunk, illetve az adatátvitelt sikertelennek tekintjük.
c) ***Hány megszakítás kezelő*** és **kérő** lehet ***MULTIBUS II*** rendszerben? ***Indokolja*** a válaszát!
Nem tudom, még keresem a választ.
Megszakításkezelő 7, kérő 256. Indoklás a konzi anyagában...

== 4 ==
a) Milyen védelmi igények merültek fel az egyszerű monitor alkalmazásakor?
Gondolom, a monitor saját memóriaterületének a védelme, a perifériák védelme (nem kezdeményezhetnek a programok i/o-t) és a futó processzek memóriaterületeinek védelme egymástól.

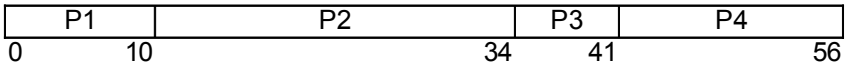
b) Mi az utasítás roll-back (visszapörgetés, újraindítás), és miért van rá szükség?
Amikor kivétel (exception) keletkezik egy utasítás végrehajtása során, az operációs rendszer (kernel) szeretne anélkül valami galádságot csinálni, hogy azt a felhasználó program észrevenné (pl. memória belapozása a merevlemezről vagy floating point coprocesszor leszimulálása szoftverből), és ekkor szükség van arra, hogy a félbemaradt utasítás megszakításakor olyan legyen, mintha a végrehajtása el sem kezdődött volna (rollback), majd a szükséges közbeavatkozás után újraindítható legyen.

c) Melyek a **CS**P (Communicating Sequential Processes) modell legfontosabb jellemzői?
A modell alakpja, hogy a programok elkülönülten futnak és közöttük bármiféle adatmozgás üzenetváltással megy végbe. Ez ügyen a Kondorossi weblapján lévő jegyzetben van bővebb infó.
== 5 ==
5) **A**dja meg a végrehajtás Gantt-diagramját, és számítsa ki az átlagos futási időket a táblázat szerinti folyamatok esetére az alábbi ütemezési algoritmusokkal:
-FCFS
-SRTF (legrövidebb maradék idejű)
-preemptív statikus prioritásos
-időszeleteléses (Round-Robin) prioritás nélkül (időszelet: 8 egység)!

folyamat	érkezés(a futásra kész sorba)	következő CPU löket	prioritás
P1	0	10	2
P2	3	24	1(legkevésbé fontos)
P3	7	7	5(legfontosabb)
P4	12	15	3

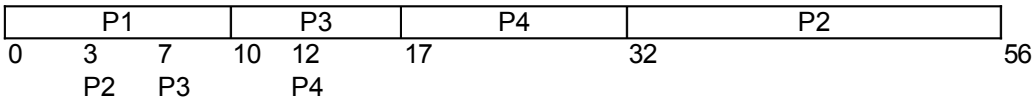
Átfutási idő átlaga=szumma(kezdet-érkezés)/darabszám

FCFS:



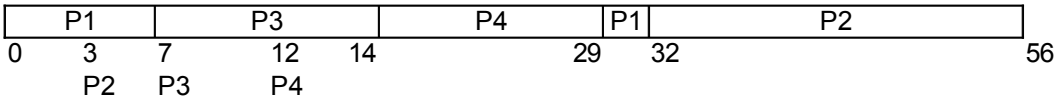
Az elv tiszta és egyszerű. Ahányadik a sorban, annyiadiknak szolgáljuk ki.

SRTF: (Shortest Remaining Time First)



Minden egyes belépő folyamat esetén megnézzük melyiknek van vissza a legkevesebb ideje a CPU löketéből és az folytatja tovább a futást akinek kevesebb van. Abban az esetben, ha két azonos maradék idejő processz van, akkor értelemszerűen az folytatódik, aki eddig is futott.

Preemptív prioritásos: (statikus prioritás)



Nincs öregedés. Ilyenkor mindig a prioritási szintek határozzák meg ki fut. Akinek a legmagasabb, az fog lefutni elıször.

Round Robin:

Itt egy forgó prioritási sor van és az újonnan érkező processzek mindig a sor legvégére állnak be. Ha egy processz lefut, akkor újraütemezzük a futást. Az újraütemezés annyit jelent, hogy a futó folyamat időegység előtti leállásakor egész egyszerűen indítjuk a következő folyamatot és nem várjuk ki az időszelet végét. A következő folyamat természetesen teljes 8 időegységet kap a saját szeletében.
A futásidők a különböző üzemezések esetén:

alg.	P1	P2	P3	P4	átlag
FCFS	10	31	34	44	29.75
SRTF	10	53	10	20	23.25
SPPE	32	53	7	17	27.25
RR	24	53	15	36	32

== 6 ==

a) **A**dja meg a holtpont definícióját!
Az egyszerre vagy időosztásban futó közös erőforrásokat felhasználó feladatok rendszere holtponton van, ha megadható a folyamatok egy olyan halmaza, amelynek minden eleme olyan erőforrás felszabadulására vár (csak a felszabadulás után folytatja a végrehajtást), amit egy ugyancsak ebbe a halmazba tartozó process birtokol. A halmazba tartozó processek soha sem fognak lefutni preemptio nélkül.

b) Mi az optimális lapcsere algoritmus virtuális tárkezelő rendszerekben, és miért nem használható a gyakorlatban?
A lapokhoz való jövőbeni hozzáférés relatív gyakoriságát felhasználva adható meg a lapcserék optimális ütemezése. Ez megvalósíthatatlan, hiszen a jövőt nem ismerjük és a program jövőbeni viselkedését pontosan megadni éppen a lefuttatásával lehet (ún. Turing-probléma, a Turing-gép

megállásával való analógiája miatt).

c) A multiprocesszoros operációs rendszerekben fizikai, vagy logikai perifériánként alakítanak ki várakozási sorokat? Miért?
Logikai perifériánként. Várakozási sorra akkor van szükség, ha több process együttes használata esetén hibák következnének be. A logikai periféria használata kizárólagos, de a fizikai perifériát (ezt az operációs rendszer kezeli) -- például több logikai periférián keresztül -- egyszerre több program is használhatja anélkül, hogy egymást zavarnák ezzel.

Jó készülést!

Sanya és KM