Készítette: Csontos Balázs

Hibajelentés és javaslat: csontos.balazs@mik.uni-pannon.hu

A dokumentum verziója: 3

ZÁRTHELYI 1. ALGORITMUSOK

I. Holtpont

I/1. Biztonságos sorozat keresés (amely alapja, hogy a folyamat csak maximális erőforrás megléte esetén fut csak le)

Adott egy rendszer, amelyben 1 erőforrás osztály van, 10 erőforrással. A rendszerben 3 folyamat (P1, P2, P3) található, amelyek az alábbi foglalásokkal F=(3, 2, 2) és maximális igényekkel M=(9, 4, 7) rendelkeznek. Ha úgy véli, hogy a rendszer biztonságos állapotban van, akkor adja meg a folyamatoknak egy biztonságos sorrendjét, különben jelölje az összeset „-„ jellel:

Biztonságos sorozat:

	→		→	

1. lépés: Elkészítem a kiindulási táblázatot a megadott adatok alapján. Szükségünk van egy Foglal, egy Max, és egy Még oszlopra. Mivel a rendszerben 3 folyamat található meg (P1, P2, P3), így 3 sorra. A foglalásokat F=(3, 2, 2) a Foglal oszlopba, a maximális igényeket M=(9, 4, 7) a Max oszlopba írom be. A Még oszlopot kitöltöm a Max és a Foglal oszlopok értékeinek különbsége alapján. Utolsó lépésként összeadom a foglalt erőforrások számát, majd megnézem, hogy hány szabad maradt a 10 erőforrásból. Ezt az értéket a Szabad sorba írom be.

	Foglal	Max	Még
P1	3	9	6
P2	2	4	2
P3	2	7	5
Szabad: 3			

2. lépés: A szabad erőforrások száma alapján meghatározom, hogy melyik folyamat erőforrás igényét tudom maximálisan kielégíteni. A választásom a P2 folyamatra esett mivel neki még 2 szabad erőforrásra van szüksége. Ezt követően a foglalt erőforrások számát (Foglal) a maximum értékre, a kérhető erőforrások számát (Még) pedig 0 értékre állítom. Végül összeadom a foglalt erőforrásokat, és kivonom a rendszerben lévő erőforrásokat számából: 10-9=1. Az szabad erőforrások száma 1 lesz.

	Foglal	Max	Még
P1	3	9	6
P2	4	4	0
P3	2	7	5
Szabad: 1			

3. lépés: Lefut a P2 folyamat és visszaadja az általa foglalt erőforrásokat, így a szabad erdőforrások száma 5 lesz.

	Foglal	Max	Még
P1	3	9	6
P2	-	-	-
P3	2	7	5
Szabad: 5			

4. lépés: A szabad erőforrások száma alapján újra meghatározom, hogy melyik folyamat erőforrás igényét tudom maximálisan kielégíteni. A választásom a P3 folyamatra esett mivel neki még 5 szabad erőforrásra van szüksége. Ezt követően a foglalt erőforrások számát (Foglal) a maximum értékre, a kérhető erőforrások számát (Még) pedig 0 értékre állítom. Végül összeadom a foglalt erőforrásokat, és kivonom a rendszerben lévő erőforrásokat számából: 10-10=0. Az szabad erőforrások száma 0 lesz.

	Foglal	Max	Még
P1	3	9	6
P2	-	-	-
P3	7	7	0
Szabad: 0			

5. lépés: Lefut a P3 folyamat és visszaadja az általa foglalt erőforrásokat, így a szabad erdőforrások száma 7 lesz.

	Foglal	Max	Még
P1	3	9	6
P2	-	-	-
P3	-	-	-
Szabad: 7			

6. lépés: A szabad erőforrások száma alapján újra meghatározom, hogy melyik folyamat erőforrás igényét tudom maximálisan kielégíteni. A választásom a P1 folyamatra esett mivel neki még 6 szabad erőforrásra van szüksége. Ezt követően a foglalt erőforrások számát (Foglal) a maximum értékre, a kérhető erőforrások számát (Még) pedig 0 értékre állítom. Végül összeadom a foglalt erőforrásokat, és kivonom a rendszerben lévő erőforrásokat számából: 10-9=1. Az szabad erőforrások száma 1 lesz.

	Foglal	Max	Még
P1	9	9	0
P2	-	-	-
P3	-	-	-
Szabad: 1			

7. lépés: Lefut a P1 folyamat és visszaadja az általa foglalt erőforrásokat, így a szabad erdőforrások száma 10 lesz.

	Foglal	Max	Még
P1	-	-	-
P2	-	-	-
P3	-	-	-
Szabad: 10			

8. lépés: Mivel mindegyik folyamat erőforrásigényét teljesíteni lehetett, így meg lehet határozni egy biztonságos sorozatot:

P2	→	P3	→	P1

I/2. Coffman algoritmus

A Coffman algoritmus segítségével állapítsa meg, hogy van-e holtpont, ha a rendszerben négy folyamatunk van (P1, P2, P3 és P4), amelyek három erőforrásosztályból már foglalnak néhány erőforrást, de továbbiakat is igényelnek a következő táblázat szerint:

	Foglal			Kér			Szabad		
	A	B	C	A	B	C	A	B	C
P1	1	4	0	0	0	5	5	1	0
P2	5	1	3	2	0	2			
P3	1	0	3	0	2	0			
P4	2	4	1	4	0	0			

Ha úgy véli, hogy a rendszerben nincs holtpont, akkor adja meg a folyamatoknak egy biztonságos sorozatát, különben jelölje az összeset „-„ jellel:

Biztonságos sorozat:

	→		→		→	

1. lépés: Olyan folyamatot keresek, amely erőforrásigényét teljesíteni tudom a szabad erőforrások (A:5, B:1, C:0) függvényében. A választásom a P4 folyamatra esett, mivel az ő erőforrásigénye (A:4, B:0, C:0) teljesíthető.

2. lépés: Kivonom a szabad erőforrások számából a P4 folyamat erőforrás igényét. Majd ezt követően a P4 folyamat által foglalt erőforrásokat hozzáadom a szabad erőforrásokhoz.

510 – 400 = 110, 110 + 241 = 351

Az új szabad erőforrások száma a következő: A:3, B:5, C:1

3. lépés: A P4 által kért erőforrásokat áthelyezem a Foglal oszlopba. Mivel a P4 nem kér több erőforrást a lefutáshoz, így a P4-hez tartozó Kér oszlop mezői kihúzhatók.

	Foglal			Kér			Szabad		
	A	B	C	A	B	C	A	B	C
P1	1	4	0	0	0	5	3	5	1
P2	5	1	3	2	0	2			

P3	1	0	3	0	2	0			
P4	4	0	0	-	-	-			

4. lépés: Újra megvizsgálom, hogy tudok-e találni olyan folyamatot, amely erőforrásigényét teljesíteni tudom a szabad erőforrások (A:3, B:5, C:1) függvényében. A választásom a P3 folyamatra esett, mivel az ő erőforrásigénye (A:0, B:2, C:0) teljesíthető.

5. lépés: Kivonom a szabad erőforrások számából a P3 folyamat erőforrás igényét. Majd ezt követően a P3 folyamat által foglalt erőforrásokat hozzáadom a szabad erőforrásokhoz.

351 – 020 = 331, 331 + 103 = 434

Az új szabad erőforrások száma a következő: A:4, B:3, C:4

6. lépés: A P3 által kért erőforrásokat áthelyezem a Foglal oszlopba. Mivel a P3 nem kér több erőforrást a lefutáshoz, így a P3-hez tartozó Kér oszlop mezői kihúzhatók.

	Foglal			Kér			Szabad		
	A	B	C	A	B	C	A	B	C
P1	1	4	0	0	0	5	4	3	4
P2	5	1	3	2	0	2			
P3	0	2	0	-	-	-			
P4	4	0	0	-	-	-			

7. lépés: Újra megvizsgálom, hogy tudok-e találni olyan folyamatot, amely erőforrásigényét teljesíteni tudom a szabad erőforrások (A:4, B:3, C:4) függvényében. A választásom a P2 folyamatra esett, mivel az ő erőforrásigénye (A:2, B:0, C:2) teljesíthető.

8. lépés: Kivonom a szabad erőforrások számából a P2 folyamat erőforrás igényét. Majd ezt követően a P2 folyamat által foglalt erőforrásokat hozzáadom a szabad erőforrásokhoz.

434 – 202 = 232, 232 + 513 = 745

Az új szabad erőforrások száma a következő: A:7, B:4, C:5

9. lépés: A P2 által kért erőforrásokat áthelyezem a Foglal oszlopba. Mivel a P2 nem kér több erőforrást a lefutáshoz, így a P2-hez tartozó Kér oszlop mezői kihúzhatók.

	Foglal			Kér			Szabad		
	A	B	C	A	B	C	A	B	C
P1	1	4	0	0	0	5	7	4	5
P2	2	0	2	-	-	-			
P3	0	2	0	-	-	-			
P4	4	0	0	-	-	-			

10. lépés: Újra megvizsgálom, hogy tudok-e találni olyan folyamatot, amely erőforrásigényét teljesíteni tudom a szabad erőforrások (A:7, B:4, C:5) függvényében. A választásom a P1 folyamatra esett, mivel az ő erőforrásigénye (A:0, B:0, C:5) teljesíthető.

11. lépés: Kivonom a szabad erőforrások számából a P1 folyamat erőforrás igényét. Majd ezt követően a P1 folyamat által foglalt erőforrásokat hozzáadom a szabad erőforrásokhoz.

745 – 005 = 740, 740 + 140 = 880

Az új szabad erőforrások száma a következő: A:8, B:8, C:0

12. lépés: A P1 által kért erőforrásokat áthelyezem a Foglal oszlopba. Mivel a P1 nem kér több erőforrást a lefutáshoz, így a P1-hez tartozó Kér oszlop mezői kihúzhatók.

	Foglal			Kér			Szabad		
	A	B	C	A	B	C	A	B	C
P1	0	0	5	-	-	-	8	8	0
P2	2	0	2	-	-	-			
P3	0	2	0	-	-	-			
P4	4	0	0	-	-	-			

13. lépés: Mivel mindegyik folyamat erőforrásigényét teljesíteni lehetett, így meg lehet határozni egy biztonságos sorozatot:

P4	→	P3	→	P2	→	P1

II. Ütemezés

II/1. Legrégebben várakozó (First Come First Served, FCFS)

Adott öt folyamat P=(1, 2, 3, 4, 5), amelyeknek futásideje C=(1, 2, 6, 5, 3). A folyamatok T=(2, 4, 5, 0, 8) időpontban kerülnek futásra kész állapotba. A rendszerben legrégebben várakozó (FCFS) CPU ütemezés működik.

Adja meg a folyamatok ütemezését:

	→		→		→		→	

1. lépés: Elkészítem a kiindulási táblázatot a megadott adatok alapján. A folyamatokat az 1. oszlopba, a folyamatok futásra állapotainak idejét (T) a 2. oszlopba, még a futási időt (C) a harmadik oszlopba írom be.

	Érkezik	CPU-idő
P1	2	1
P2	3	2
P3	4	6
P4	0	5
P5	8	3

2. lépés: Az algoritmus alapja, hogy az a folyamat lesz leghamarabb kiszolgálva, amelyik leghamarabb érkezett meg futásra kész állapotba. Érkezési sorrendben a folyamatok ütemezése a következő:

P4	→	P1	→	P2	→	P3	→	P5

II/2. Körbeforgó (Round Robin, RR)

Adott négy folyamat P=(1, 2, 3, 4), amelyeknek futásideje C=(3, 6, 2, 5). A folyamatok T=(1, 3, 0, 5) időpontban kerülnek futásra kész állapotba. A rendszerben körforgó (RR) CPU ütemezés működik, az időszelet hossza 3 ms.

Adja meg a folyamatok ütemezését:

	→		→		→		→		→		→		→	

(Ha valamelyik helyre már nincs mit ütemezni, akkor jelölje "-" jellel!)

1. lépés: Elkészítem a kiindulási táblázatot a megadott adatok alapján. A folyamatokat az 1. oszlopba, a folyamatok futásra állapotainak idejét (T) a 2. oszlopba, még a futási időt (C) a harmadik oszlopba írom be.

	Érkezik	CPU-idő	Indul	Vége	Vár
P1	1	3			
P2	3	6			
P3	0	2			
P4	5	5			

2. lépés: Megkeresem a legkorábban érkező folyamatot (P3), majd az érkezési idejét, beírom az Indul oszlop megfelelő helyére. Mivel a P3 folyamat CPU-ideje (2) kisebb, mint az időszelet hossza (3), így az indulási időhöz hozzáadom a CPU-időt, majd ezt az értéket a Vége oszlop megfelelő sorába írom be. Végül, mivel a folyamatnak nem kellett várakoznia, így a Vár oszlop megfelelő sorába a 0 értéket írom be.

	Érkezik	CPU-idő	Indul	Vége	Vár
P1	1	3			
P2	3	6			
P3	0	2	0	2	0
P4	5	5			

3. lépés: Megkeresem a következő legkorábban érkező folyamatot (P1), majd az indulási idejének megadom az előző folyamat (P3) végzési idejét. Mivel a P1 folyamat CPU-ideje (3) megegyezik az időszelet hosszával (3), így az indulási időhöz hozzáadom a CPU-időt, majd ezt az értéket (5) a Vége oszlop megfelelő sorába írom be. Végül, mivel a P1 folyamat az 1. időszeletben érkezett, de csak a 2. időszeletben indult, így 1 időszeletet kellett várnia. Ezt az értéket a Vár oszlop megfelelő helyére írom be.

	Érkezik	CPU-idő	Indul	Vége	Vár
P1	1	3	2	5	1
P2	3	6			
P3	0	2	0	2	0
P4	5	5			

4. lépés: Megkeresem a következő legkorábban érkező folyamatot (P2), majd az indulási idejének megadom az előző folyamat (P1) végzési idejét. Mivel a P2 folyamat CPU-ideje (6) nagyobb, mint az időszelet hossza (3), így az indulási időhöz (5) hozzáadom az időszelet hosszát (3), majd ezt az értéket (8) írom be a végzési időnek. Ezt követően létrehozok egy új P2 folyamatot, amelynek érkezési ideje, az előző végzési ideje (8), a CPU-ideje pedig az eredeti CPU-idő és az időszelet különbsége (6-3=3). Végül, mivel a P2 folyamat a 3. időszeletben érkezett, de csak az 5. időszeletben indult el, így 2 időszeletet kellett várnia. Ezt az értéket a Vár oszlop megfelelő helyére írom be.

	Érkezik	CPU-idő	Indul	Vége	Vár
P1	1	3	2	5	1
P2	3	6	5	8	2
P3	0	2	0	2	0
P4	5	5			
P2	8	3			

5. lépés: Megkeresem a következő legkorábban érkező folyamatot (P4), majd az indulási idejének megadom az előző folyamat (P2) végzési idejét. Mivel a P4 folyamat CPU-ideje (5) nagyobb, mint az időszelet hossza (3), így az indulási időhöz (8) hozzáadom az időszelet hosszát (3), majd ezt az értéket (11) írom be a végzési időnek. Ezt követően létrehozok egy új P4 folyamatot, amelynek érkezési ideje, az előző végzési ideje (11), a CPU-ideje pedig az eredeti CPU-idő és az időszelet különbsége (5-3=2). Végül, mivel a P4 folyamat az 5. időszeletben érkezett, de csak a 8. időszeletben indult el, így 3 időszeletet kellett várnia. Ezt az értéket a Vár oszlop megfelelő helyére írom be.

	Érkezik	CPU-idő	Indul	Vége	Vár
P1	1	3	2	5	1
P2	3	6	5	8	2
P3	0	2	0	2	0
P4	5	5	8	11	3
P2	8	3			
P4	11	2			

6. lépés: Megkeresem a következő legkorábban érkező folyamatot (P2), majd az indulási idejének megadom az előző folyamat (P4) végzési idejét. Mivel a P2 folyamat CPU-ideje (3) megegyezik az időszelet hosszával (3), így az indulási időhöz hozzáadom a CPU-időt, majd ezt az értéket (14) a Vége oszlop megfelelő sorába írom be. Végül, mivel a P2 folyamat az 8. időszeletben érkezett, de csak a 12. időszeletben indult, így 4 időszeletet kellett várnia. Ezt az értéket a Vár oszlop megfelelő helyére írom be.

	Érkezik	CPU-idő	Indul	Vége	Vár
P1	1	3	2	5	1
P2	3	6	5	8	2
P3	0	2	0	2	0
P4	5	5	8	11	3
P2	8	3	11	14	3
P4	11	2			

7. lépés: Megkeresem a következő legkorábban érkező folyamatot (P4), majd az indulási idejének megadom az előző folyamat (P2) végzési idejét. Mivel a P4 folyamat CPU-ideje (2) kisebb, mint az időszelet hossza (3), így az indulási időhöz hozzáadom a CPU-időt, majd ezt az értéket (16) a Vége oszlop megfelelő sorába írom be. Végül, mivel a P4 folyamat a 12. időszeletben érkezett, de csak a 15. időszeletben indult, így 3 időszeletet kellett várnia. Ezt az értéket a Vár oszlop megfelelő helyére írom be.

	Érkezik	CPU-idő	Indul	Vége	Vár
P1	1	3	2	5	1
P2	3	6	5	8	2
P3	0	2	0	2	0
P4	5	5	8	11	3
P2	8	3	11	14	3
P4	11	2	14	16	3

7. lépés: Indulási idők növekvő sorendje alapján meghatározható a folyamatok ütemezése:

P3	→	P1	→	P2	→	P4	→	P2	→	P4	→	-	→	-

II/3. Prioritásos

Adott öt folyamat P=(1, 2, 3, 4, 5), amelyeknek futásideje C=(3, 6, 2, 1, 3), prioritása pedig Prioritas=(5, 4, 2, 3, 1). A folyamatok T=0 időpontban kerülnek futásra kész állapotba.

1. lépés: Elkészítem a kiindulási táblázatot a megadott adatok alapján. A folyamatokat az 1. oszlopba, a folyamatok futásra állapotainak idejét (T) a 2. oszlopba, még a futási időt (C) a harmadik oszlopba írom be.

	CPU-idő	Prioritás
P1	1	5
P2	2	4
P3	6	2
P4	5	3
P5	3	1

2. lépés: Mivel minden folyamat a 0. időegységben kerül futásra kész állapotba, így az algoritmusnál a prioritás értéke a mérvadó. Minél kisebb a prioritás annál hamarabb kerül beütemezésre a folyamat. Prioritási sorrendben a folyamatok ütemezése a következő:

P5	→	P3	→	P4	→	P2	→	P1

II/4. Legrövidebb hátralévő idejű (Shortest Remaining Time First, SRTF)

Adott négy folyamat P=(1, 2, 3, 4), amelyeknek futásideje C=(4, 5, 7, 1). A folyamatok T=(5, 2, 4, 0) időpontban kerülnek futásra kész állapotba. A rendszerben a legrövidebb hátralévő idejű (Shortest Remaining Time First - SRTF) CPU ütemezés működik.

Adja meg a folyamatok ütemezését:

	→		→		→		→		→		→		→	

(Megjegyzés: ha egy döntés után ugyanaz a folyamat folytatódik, akkor azt újból fel kell venni! Ha valamelyik hely már nincs mit ütemezni, akkor jelölje "-" jellel!)

1. lépés: Elkészítem a kiindulási táblázatot a megadott adatok alapján. A folyamatokat az 1. oszlopba, a folyamatok futásra állapotainak idejét (T) a 2. oszlopba, még a futási időt (C) a harmadik oszlopba írom be.

	Érkezik	CPU-idő	Indul	Vége	Vár
P1	5	4			
P2	2	5			
P3	4	7			
P4	0	1			

2. lépés: Megkeresem a legkorábban érkező folyamatot (P4), majd az érkezési idejét, beírom az Indul oszlop megfelelő helyére. Mivel futás közben nem érkezik új folyamat, így a végzési idő egyenlő lesz a P4 folyamat CPU-idejével (1).

	Érkezik	CPU-idő	Indul	Vége	Vár
P1	5	4			
P2	2	5			
P3	4	7			
P4	0	1	0	1	0

3. lépés: Megkeresem a legkorábban érkező folyamatot (P2), majd az érkezési idejét (2), beírom az Indul oszlop megfelelő helyére, mivel a P4 folyamat már hamarabb befejeződött. Mivel a CPU-idő 5, így egészen a 7. időszeletig tartana a folyamat. Azonban a 4. időszeletben futásra kész állapotba került a P3 folyamat is. Ekkor meg kell vizsgálni, hogy melyik folyamatnak kisebb a szükséges CPU-ideje a 4. időszeletben. Mivel a P2 folyamatnak 3, a P3 folyamatnak pedig 7 a CPU-ideje, így a P2 folyamat folytatódhat tovább. (Ezt jelezni kell a megoldásban úgy, hogy újra leírom a P2 folyamatot.) Az 5. időszeletben azonban futásra kész lesz a P1 folyamat is. Ekkor meg kell vizsgálni, hogy melyik folyamatnak kisebb a szükséges CPU-ideje az 5. időszeletben. Mivel a P2 folyamatnak 2, a P1 folyamatnak pedig 4 a CPU-ideje, így a P2 folyamat folytatódhat tovább. (Ezt jelezni kell a megoldásban úgy, hogy újra leírom a P2 folyamatot.) Ezt követően már nem lesz új folyamat, amely futásra kész áll, így a P2 végzési ideje az érkezési és CPU-idő összege lesz (2+5=7). Végül mivel nem kellett várakoznia, így a 0 értéket kell beírni a Vár oszlop megfelelő oszlopába.

	Érkezik	CPU-idő	Indul	Vége	Vár
P1	5	4			
P2	2	5	2	7	0
P3	4	7			
P4	0	1	0	1	0

3. lépés: Mivel a 7. időlépésben vagyok, így meg kell néznem, hogy vannak-e olyan folyamatok, amelyek futásra készen állnak. Ilyen a P1 és a P3 is. Mivel a P1-nek a kisebb a CPU ideje, így őt fogom következőnek beütemezni. P1 indulási ideje meg fog egyezni az előző folyamat (P2) végzési idejével (7). P1 végzési ideje pedig az indulási idő és a CPU-idő összege lesz (7+4=11). Várakozás idő szempontjából az 5. időszeletben kellett volna indulnia, azonban csak a 7. időszeletben tudott, így a várakozási idő 2 lesz.

	Érkezik	CPU-idő	Indul	Vége	Vár
P1	5	4	7	11	2
P2	2	5	2	7	0
P3	4	7			
P4	0	1	0	1	0

4. lépés: Végül az utolsó folyamatot (P3) is beütemezem. P3 indulási ideje meg fog egyezni az előző folyamat (P1) végzési idejével (11). P3 végzési ideje pedig az indulási idő és a CPU-idő összege lesz (11+7=18). Várakozás idő szempontjából az 4. időszeletben kellett volna indulnia, azonban csak a 11. időszeletben tudott, így a várakozási idő 7 lesz.

	Érkezik	CPU-idő	Indul	Vége	Vár
P1	5	4	7	11	2
P2	2	5	2	7	0
P3	4	7	11	18	7
P4	0	1	0	1	0

5. lépés Folyamatok ütemezésének meghatározása az indulási idők és a folyamatok megállásának és újraindulásának jelölésével.

P4	→	P2	→	P2	→	P2	→	P1	→	P3	→	-	→	-

Megjegyzés az algoritmus használatához: Ha egy döntés miatt egy folyamat nem folytatódhat tovább, akkor azt újra fel kell venni a folyamatok közé úgy, hogy a végzési idő lesz az új érkezési idő, a megmaradt CPU-idő pedig az új CPU-idő lesz.

ZÁRTHELYI 2. ALGORITMUSOK

I. Tárkezelés

I/1. Legjobban megfelelő (Best Fit, BF)

Egy rendszerbe korábban már a P1, P2 és P3 folyamatok betöltésre kerültek, amely eredményeként az aktuális memória kiosztása az alábbi:

Cím: 0000 1000 2000 2200 2400 3000 5000 [Byte]

|-------|-------|--------|-------|-------|-------|-------|

Proc: P1 - P3 - P2 - OS

Ha egy 900 Byte méretű P4 és egy 200 Byte méretű P5 folyamat érkezik, akkor a rendszer a best fit algoritmus használatával a P4-t a _____ címre, a P5-t pedig a _____ címre tölti be.

Az algoritmus alapja, hogy olyan memóriaterületet válasszunk ki egy folyamat számára, amely a legközelebb áll a folyamat méretéhez, de nem kisebb annál. Az ábrán látható, hogy a P1, P2, P3 folyamatnak korábban már sikerült helyet találnunk, így maradék üres helyekre kell beosztanunk a P4 (900 byte) és P5 (200 byte) folyamatot.

Az üres helyek a következők:

- 1000 és 2000 között
- 2200 és 2400 között
- 3000 és 5000 között

Ezek alapján látható, hogy P4-et az 1000-es címre, míg a P5-öt a 2200-as címre kell elhelyeznünk.

I/2. Első megfelelő (First Fit, FF)

Egy rendszerbe korábban már a P1, P2 és P3 folyamatok betöltésre kerültek, amely eredményeként az aktuális memória kiosztása az alábbi:

Cím: 0000 1000 2000 2200 2400 3000 5000 [Byte]

|-------|-------|--------|-------|-------|-------|-------|

Proc: P1 - P3 - P2 - OS

Ha egy 900 Byte méretű P4 és egy 200 Byte méretű P5 folyamat érkezik, akkor a rendszer a first fit algoritmus használatával a P4-et a _____ címre, a P5-t pedig a _____ címre tölti be.

Az algoritmus alapja, hogy a memória elejéről kiindulva az első megfelelő méretű memóriaterületet válasszuk ki az adott folyamat számára. Az ábrán látható, hogy a P1, P2, P3 folyamatnak korábban már sikerült helyet találnunk, így maradék üres helyekre kell beosztanunk a P4 (900 byte) és P5 (byte) folyamatot.

Az üres helyek a következők:

- 1000 és 2000 között
- 2200 és 2400 között
- 3000 és 5000 között

Ezek alapján látható, hogy P4-et az 1000-es címre, míg a P5-öt a 2200-as címre elhelyeznünk.

I/3. Következő megfelelő (Next Fit, NF)

Egy rendszerbe korábban már a P1, P2 és P3 folyamatok betöltésre kerültek, amely eredményeként az aktuális memória kiosztása az alábbi:

Cím: 0000 1000 2000 2200 2400 3000 5000 [Byte]

|-------|-------|--------|-------|-------|-------|-------|

Proc: P1 - P3 - P2 - OS

Ha egy 900 Byte méretű P4 és egy 200 Byte méretű P5 folyamat érkezik, akkor a rendszer a next fit algoritmus használatával a P4-et a _____ címre, a P5-öt pedig a _____ címre tölti be.

Az algoritmus alapja, hogy az első olyan memóriaterületet kell kiválasztanunk az adott folyamat számára, amely elegendő méretű, és a legutóbbi folyamat beillesztése után következik az aktuális pozícióban a memóriában. Ez azt jelenti, hogy az NF algoritmus nem a memóriakezdőponttól, hanem az utolsó beillesztett folyamat után folytatja a keresést. Ha nem talál, akkor kezdi előről a keresést.

Első lépésben a 900 byte méretű P4 folyamatnak kell memóriát foglalnunk. Az utolsó folyamat a P3, amely a 2200-ig foglalja a memóriát, ez a kiindulási alap. 2200 és 2400 között nem fér be. 2400 és 3000 között a memóriát a P2 folyamat foglalja. 3000 és 5000 között van szabad hely, így a P4-et a 3000-es címre helyezem. Következik a P5 folyamat 200 byte mérettel. Mivel a P4 folyamat után már az OS következik, így előről kell kezdem a keresést. 0000 és 1000 között a P1 folyamat szerepel, azonban 1000 és 2000 közé be tudom illeszteni a P5 folyamatot.

Kitekintés: Mikor növekedhet a tördelődés?

- Nem növekszik a külső tördelődés: Ha az új folyamatot közvetlenül az utolsó folyamat után lehet elhelyezni, és nem marad szabad terület az operációs rendszer és az utolsó folyamat között.
- Növekszik a külső tördelődés: Ha az új folyamatot nem lehet közvetlenül az utolsó folyamat után elhelyezni, és az új folyamat beillesztése új szabad területet eredményez az operációs rendszer és az utolsó folyamat között.
- Növekedhet a külső tördelődés: Ha az új folyamat beillesztésekor nem növekszik a külső tördelődés, de az új folyamat beillesztése után más folyamatok elmozdulhatnak a memóriában, ami újabb szabad területeket eredményez az operációs rendszer és az utolsó folyamat között, ezzel növelve a külső tördelődést.

I/4. Legrosszabban illeszkedő (Worst Fit, WF)

Egy rendszerbe korábban már a P1, P2 és P3 folyamatok betöltésre kerültek, amely eredményeként az aktuális memória kiosztása az alábbi:

Cím: 0000 1000 2000 2200 2400 3000 5000 [Byte]

|-------|-------|--------|-------|-------|-------|-------|

Proc: P1 - P3 - P2 - OS

Ha egy 900 Byte méretű P4 és egy 200 Byte méretű P5 folyamat érkezik, akkor a rendszer a worst fit algoritmus használatával a P4-et a _____ címre, a P5-öt pedig a _____ címre tölti be.

Az algoritmus alapja, hogy a legnagyobb elegendő méretű memóriaterületet válasszuk ki a folyamat számára. Az ábrán látható, hogy a P1, P2, P3 folyamatnak korábban már sikerült helyet találnunk, így maradék üres helyekre kell beosztanunk a P4 (900 byte) és P5 (200 byte) folyamatot.

Az üres helyek a következők:

- 1000 és 2000 között
- 2200 és 2400 között

- 3000 és 5000 között

Ezek alapján látható, hogy P4-et a 3000-es címre, míg a P5-öt az 1000-es címre kell elhelyeznünk.

II. Virtuális tárkezelés

II/1. Legrégebben használt lap (Least Recently Used, LRU)

A memóriában 1 folyamat fut, amelynek a virtuális memóriája 5 lapból áll. A végrehajtás során a lapokra az alábbi sorrendben hivatkozik: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. A folyamatok lapkerete 4.

Ha a rendszerben a LRU (legrégebben használt lap) lapcsere stratégiát alkalmazzuk, akkor a folyamat _____ db laphibával fog lefutni.

Az algoritmus alapja, hogy lapcserekor azt a lapot vesszük ki a memóriából, amelyre a folyamatok leghosszabb ideje nem hivatkoztak.

1. lépés: Kezdeti táblázat felírása a meglévő adatok alapján.

Laphivatkozások	1	2	3	4	1	2	5	1	2	3	4	5
Lapok a tárban	1	1	1	1	1	1	1	1	1	1	1	5
		2	2	2	2	2	2	2	2	2	2	2
			3	3	3	3	5	5	5	5	4	4
				4	4	4	4	4	4	3	3	3
Laphibák	x	x	x	x			x			x	x	x

2. lépés: Táblázat kitöltése. Lapcsere az LRU algoritmus alapján:

- Az 1-es bent van a tárban? Nincs. Beírom, és jelzem a laphibát.
- A 2-es bent van a tárban? Nincs. Beírom, és jelzem a laphibát.
- A 3-as bent van a tárban? Nincs. Beírom, és jelzem a laphibát.
- A 4-es bent van a tárban? Nincs. Beírom, és jelzem a laphibát.
- Az 1-es bent van a tárban? Igen. Nincs laphiba.
- A 2-es bent van a tárban? Igen. Nincs laphiba.
- Az 5-ös bent van a tárban? Nincs. Cserélem a legrégebb óta hivatkozott lapra (3), majd jelzem a laphibát.
- Az 1-es bent van a tárban? Igen. Nincs laphiba.
- A 2-es bent van a tárban? Igen. Nincs laphiba.
- A 3-as bent van a tárban? Nincs. Cserélem a legrégebb óta hivatkozott lapra (4), majd jelzem a laphibát.
- A 4-es bent van a tárban? Nincs. Cserélem a legrégebb óta hivatkozott lapra (5), majd jelzem a laphibát.
- Az 5-ös bent van a tárban? Nincs. Cserélem a legrégebb óta hivatkozott lapra (1), majd jelzem a laphibát.

3. lépés: Összeszámolom a laphibák számát, amely összesen 8 db.

II/2. Második esély (Secound Chance, SC)

A memóriában 1 folyamat fut, amelynek a virtuális memóriája 5 lapból áll. A végrehajtás során a lapokra az alábbi sorrendben hivatkozik: 1, 2, 3, 4, 3, 1, 5, 1, 2, 3, 1, 5. A folyamatok lapkerete 3.

Ha a rendszerben a Second Chance (második esély) lapcsere stratégiát alkalmazzuk, akkor a folyamat...

a. 5 db laphibával fog lefutni.
b. 6 db laphibával fog lefutni.
c. 7 db laphibával fog lefutni.
d. 8 db laphibával fog lefutni.
e. 9 db laphibával fog lefutni.
f. 10 db laphibával fog lefutni.

Az algoritmus alapja, hogy lapcserekor az a lap kerül ki a memóriából, amelyik legrégebb óta bent van, ha még nem hivatkoztak rá. A hivatkozásokat jelölő bittel jelöljük. Ha jelölőbittel ellátott lap a legrégebb óta bent lévő lap, akkor adunk neki egy második esélyt, azonban a jelölő bitjét 0-ra állítjuk.

1. lépés: Kezdeti táblázat felírása a meglévő adatok alapján.

Laphivatkozások	1	2	3	4	3	1	5	1	2	3	1	5
Lapok a tárban	1_0	1_0	1_0	4_0	4_0	4_0	5_0	5_0	5_0	3_0	3_0	3_0
		2_0	2_0	2_0	2_0	1_0	1_0	1_1	1_1	1_0	1_1	1_0
			3_0	3_0	3_1	3_1	3_0	3_0	2_0	2_0	2_0	5_0
Laphibák	x	x	x	x		x	x		x	x		x

2. lépés: Táblázat kitöltése. Lapcsere az SC algoritmus alapján:

- Az 1-es bent van a tárban? Nincs. Hivatkoztunk már? Nem. Beírom a hivatkozást (1), majd beállítom a jelölőbitet (0), hogy nincs rá hivatkozás. Végül jelzem a laphibát.
- A 2-es bent van a tárban? Nincs. Hivatkoztunk már? Nem. Beírom a hivatkozást (2), majd beállítom a jelölőbitet (0), hogy nincs rá hivatkozás. Végül jelzem a laphibát.
- A 3-es bent van a tárban? Nincs. Hivatkoztunk már? Nem. Beírom a hivatkozást (3), majd beállítom a jelölőbitet (0), hogy nincs rá hivatkozás. Végül jelzem a laphibát.
- A 4-es bent van a tárban? Nincs. Mivel nincs több lapkeret cserélnem kell. Először megnézem, hogy melyik lap van legrégebb óta a tárban. Az 1-es. Mivel nincs hivatkozva (jelölőbit = 0) így, kicserélem rá a 4-est.
- A 3-es bent van a tárban? Igen. Mivel még nem hivatkoztunk rá, de most igen, így a jelölőbitet 1-esre állítom. Nincs laphiba.
- Az 1-es bent van a tárban? Nincs, ezért cserélnem kell. Először megnézem, hogy melyik lap van legrégebb óta a tárban. A 2-es. Mivel nincs hivatkozva (jelölőbit = 0) így, kicserélem rá az 1-est.
- Az 5-ös bent van a tárban? Nincs, ezért cserélnem kell. Először megnézem, hogy melyik lap van legrégebb óta a tárban. A 3-as. Mivel hivatkozva van (jelölőbit = 1) így, kap egy újabb esélyt. A 3-as marad azonban a jelölőbitjét 0-ra állítom. Megnézem, hogy mi a következő legrégebben bent lévő lap. A 4-es. Mivel nincs hivatkozva (jelölőbit = 0) így, kicserélem rá az 5-öst.
- Az 1-es bent van a tárban? Igen. Mivel még nem hivatkoztunk rá, de most igen, így a jelölőbitet 1-esre állítom. Nincs laphiba.
- A 2-es bent van a tárban? Nincs, ezért cserélnem kell. Először megnézem, hogy melyik lap van legrégebb óta a tárban. A 3-as. Mivel nincs hivatkozva (jelölőbit = 0) így, kicserélem rá az 2-est.

- A 3-ös bent van a tárban? Nincs, ezért cserélnem kell. Először megnézem, hogy melyik lap van legrégebb óta a tárban. Az 1-es. Mivel hivatkozva van (jelölőbit = 1) így, kap egy újabb esélyt. Az 1-es marad azonban a jelölőbitjét 0-ra állítom. Megnézem, hogy mi a következő legrégebben bent lévő lap. Az 5-ös. Mivel nincs hivatkozva (jelölőbit = 0) így, kicserélem rá az 3-ast.
- Az 1-es bent van a tárban? Igen. Mivel még nem hivatkoztunk rá, de most igen, így a jelölőbitet 1-esre állítom. Nincs laphiba.
- Az 5-ös bent van a tárban? Nincs, ezért cserélnem kell. Először megnézem, hogy melyik lap van legrégebb óta a tárban. Az 1-es. Mivel hivatkozva van (jelölőbit = 1) így, kap egy újabb esélyt. Az 1-as marad azonban a jelölőbitjét 0-ra állítom. Megnézem, hogy mi a következő legrégebben bent lévő lap. A 2-es. Mivel nincs hivatkozva (jelölőbit = 0) így, kicserélem rá az 5-öst.

3. lépés: Összeszámolom a laphibák számát, amely összesen 9 db.

Megjegyzés: Előfordulhat, hogy az összes lapnak van már jelölőbitje, azonban valamelyik le kel le kell cserélnünk egy új lapra. Ilyenkor mindegyik jelölőbitjét 0-ra állítjuk, majd a legrégebben a tárban lévő lapot cseréljük le az új lapra.

III. Háttértár

III/1. N-lépéses pásztázó (N-Step-SCAN, N-SCAN)

Egy merevlemezen a cilinderek száma 0-99, a rendszerbe 8 kérés érkezik R= (1, 2, 3, 4, 5, 6, 7, 8), melyek beérkezési időpontja T= (0, 0, 0, 0, 0, 0, 0, 0), és az adatokat tartalmazó cilinderek száma:

C= (33, 27, 8, 47, 82,11, 91, 68). Adja meg a kiszolgálás sorrendjét, ha a háttértár a N-lépéses pásztázó (N-SCAN) algoritmust használja, a fej az egyik cilinderről a másikra 1 ms alatt mozog, és a 0. időpillanatban a 28. cilinder fölött növekvő irányba mozog és N = 3.

A kiszolgálás sorrendje:

	→		→		→		→		→		→		→	

Az algoritmus alapja, hogy az aktuális cilinderből kiindulva a megadott irányban N számú kérést teljesítünk majd irányt váltunk, ahol újabb N számú kérést teljesítünk. Ha az egyik irányban elfogyott az összes teljesíthető kérés, akkor a másik irány összes kérését teljesítjük.

Az útvonal következő:

- 28 -> 33 (R1)
- 33 -> 47 (R4)
- 47 -> 68 (R8)
- 68 -> 27 (R2)
- 27 -> 11 (R6)
- 11 -> 8 (R3)
- 8 -> 82 (R5)
- 82 -> 91 (R7)

A kiszolgálás sorrendje:

R1	→	R4	→	R8	→	R2	→	R6	→	R3	→	R5	→	R7

III/2. Legrövidebb fejmozgási idő (Shortest seek time first, SSTF)

Egy merevlemezen a cilinderek száma 0-99, a rendszerbe 8 kérés érkezik R= (1, 2, 3, 4, 5, 6, 7, 8), melyek beérkezési időpontja T= (0, 0, 0, 0, 0, 0, 0, 0), és az adatokat tartalmazó cilinderek száma: C= (33, 27, 8, 47, 82,11, 91, 68). Adja meg a kiszolgálás sorrendjét, ha a háttértár a legrövidebb fejmozgási idő (SSTF) algoritmus használja, a fej az egyik cilinderről a másikra 1 ms alatt mozog, és a 0. időpillanatban a 48. cilinder fölött csökkenő irányba mozog.

A kiszolgálás sorrendje:

	→		→		→		→		→		→		→	

Az algoritmus alapja, hogy a fej aktuális cilinderből kiindulva arra a cilinderre kell ugranunk, amelyet a legkisebb fejmozgással érhetünk el a mozgási iránynak megfelelően. Ha a fej mozgási irányában már nincsen közelebb cilinder, akkor a fejet irányt vált.

Az útvonal következő:

- 48 -> 47 (R4)
- 47 -> 33 (R1)
- 33 -> 27 (R2)
- 27 -> 11 (R6)
- 11 -> 8 (R3)
- 8 -> 68 (R8)
- 68 -> 82 (R5)
- 82 -> 91 (R7)

A kiszolgálás sorrendje:

R4	→	R1	→	R2	→	R6	→	R3	→	R8	→	R5	→	R7