JEGYZŐKÖNYV

Operációs rendszerek BSC

2022. tavasz féléves feladat

Készítette:

Kopecskó Zsolt Bsc

Mérnökinformatikus

IVO46O

Miskolc, 2022

1. Írjon egy olyan C programot, amely létrehozza, olvassa, írja és törli az osztott memóriát!
   A műveletet a parancssoron keresztül adja meg.
   Amennyiben egy művelet kiadásakor a közös memória nem létezik, a program automatikusan hozza azt létre!

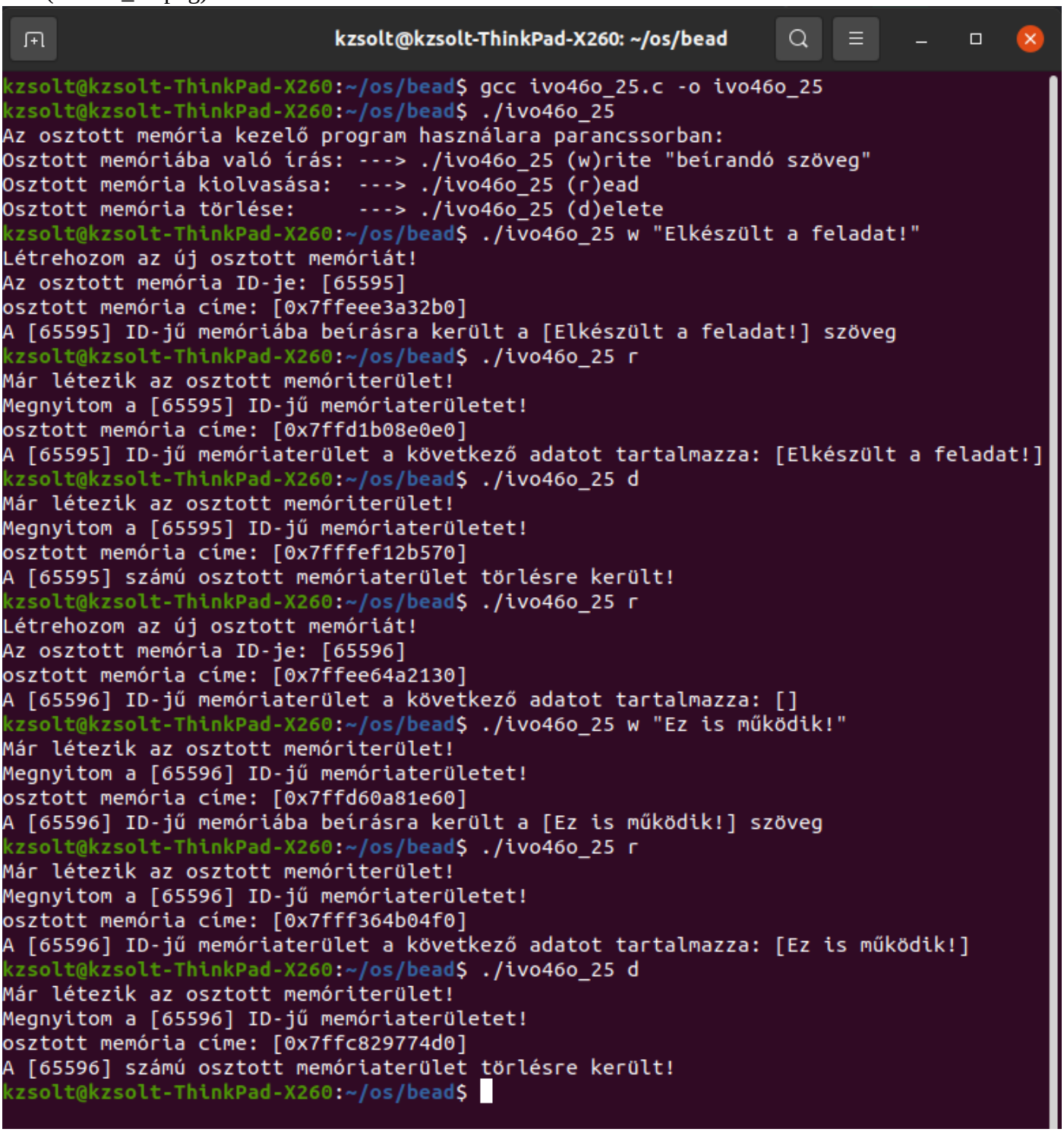
   (ivo46o_25.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <ctype.h>

void howtouse(void); /*Súgó a program működéséhez!*/
void shmwrite(int shmid, char *shmsegpt, char *text); /*Osztott memóriaterületre való írás*/
void shmread(int shmid, char *shmsegpt); /*Osztott memóriaterületről való olvasás*/
void shmdelete(int shmid, char *shmsegpt); /*Osztott memóriaterület törlése*/

int main (int argc, char* argv[]){ /*paraméterek a parancssorban*/
    key_t key; /*jogosultságot meghatározó kulcs*/
    int shmid; /*osztott memóriaterület ID-je*/
    int execshmid; /*létező osztott memória terület címe*/
    int shmsegsize; /*osztott memória mérete*/
    shmsegsize = 100; /*osztott memória mérete bájtban megadva*/
    char *shmsegpt; /*osztott memória címe*/

    if(argc == 1){ /*ha csak egy argumentum kerül megadásra a parancssorban, akkor lefut a súgó függvény*/
            howtouse(); /*Súgó a program működéséhez*/
    }
    /*Kulcs generálás*/
    key = ftok(".", 'A'); /*key generáló függvény egy kulcsot ad vissza az elérési út és az azonosító alapján. A függvény ugyanazt a kulcsot adja vissza minden olyan elérési úthoz, amely ugyanarra a fájlra mutat, ha ugyanazzal az azonosító értékkel hívják meg.*/
    /*oszotott memóriaterület létrehozása*/
    shmid = shmget(key, shmsegsize, IPC_CREAT|IPC_EXCL|0666);
    /*A shmget függvény visszatérési értéke a key kulcshoz rendelt memóriazóna azonosítója. Hiba esetén -1.*/
    if(shmid == -1){ /*ha -1 a visszatérési érték, akkor már van a kulcshoz rendelt osztott memóriaterület.*/
            execshmid = shmget(key, 0, 0); /*a létező osztott memóriaterület ID-jének lekérdezése.*/
            printf("Már létezik az osztott memóriterület!\nMegnyitom a [%d] ID-jű memóriaterületet!\n", execshmid);
            if ((shmid = shmget(key, shmsegsize, 0)) == -1){
     perror("shmget hiba");
     exit(1);
    }
```

```
    }else if(shmid != -1){ /*Ha nem -1 a visszatérési érték, akkor sikeresen létrejött az osztott memóriaterület.*/
                printf("Létrehozom az új osztott memóriát!\n");
                printf("Az osztott memória ID-je: [%d]\n", shmid);
    }
    /*Memórirész hozzárendelés*/
    /*A *shmat(int shmid, void *addr, int flg) függvény feladata egy folyamat címterületéhez hozzárendelni egy osztott memóriazónát. A hozzárendelés után a folyamat írhat, olvashat erről a memóriaterületről. A függvény visszatérési értéke egy pointer a közös memóriazónára, ha a művelet sikeres volt. Ellenkező esetben -1. Az addr egy pointer típusú változó a közös memóriarész hozzáférési címe a hívó folyamat adatszegmensében. Az addr=0, a memóriarész a rendszer által kiválasztott első szabad címe.*/
    shmsegpt = shmat(shmid, 0, 0);
    printf("osztott memória címe: [%p]\n", &shmsegpt);

    /*műveletek szerinti bontás*/
    switch (tolower(argv[1][0])){ /*tolower fv. segítségével a bevitt nagybetűt kisbetűre változtatjuk*/
            case 'w':
                shmwrite(shmid, shmsegpt, argv[2]); //beíró fv. meghívása
                break;
            case 'r':
                shmread(shmid, shmsegpt); //kiolvasó fv. meghívása
                break;
            case 'd':
                shmdelete(shmid, shmsegpt); //memória törlő fv. meghívása
                break;
    }
return 0;
}

void howtouse(void){
    printf("Az osztott memória kezelő program használara parancssorban:\n");
    printf("Osztott memóriába való írás: ---> ./ivo46o_25 (w)rite \"beírandó szöveg\"\n");
    printf("Osztott memória kiolvasása:  ---> ./ivo46o_25 (r)ead\n");
    printf("Osztott memória törlése:     ---> ./ivo46o_25 (d)elete\n");
    exit(1);
}

void shmwrite(int shmid, char *shmsegpt, char *text){
    if(text==NULL){
            printf("A [%d] ID-jű memóriaterületre nem lett beállítva beírandó szöveg!\n", shmid);
            strcpy(shmsegpt, "");
            exit(1);
    }else
    printf("A [%d] ID-jű memóriába beírásra került a [%s] szöveg\n", shmid, text);
    strcpy(shmsegpt, text);
}

void shmread(int shmid, char *shmsegpt){
    printf("A [%d] ID-jű memóriaterület a következő adatot tartalmazza: [%s]\n", shmid, shmsegpt);
}
```

```
void shmdelete(int shmid, char *shmsegpt){
    /*A shmctl(int shmid, int cmd, struct shmid_ds *buf) függvény az osztott memóriarész információinak
    lekérdezésére, módosítására és törlésére használható.
    A függvény visszatérési értéke 0, ha a művelet sikeres volt, ellenkező esetben -1.
    A cmd argumentumban megadott IPC_RMID az osztott memóriarész törlésére szolgál.
    A végleges törlés akkor kerül sor, amikor az utolsó folyamat is, amelyik ezt a területet használja,
    megszakítja a kapcsolatot ezzel a memóriarésszel.*/
    shmctl(shmid, IPC_RMID, 0);
    printf("A [%d] számú osztott memóriaterület törlésre került!\n", shmid);
}
```

(ivo46o_25.png)



2. Adott egy számítógépes rendszer, melyben a
   - szabad memória területek: 23KB, 64KB, 10KB, 80KB, 12KB, 50KB, 40KB melynek
   - foglalási igénye: 65KB, 21KB, 48KB, 13KB, 62KB

   Határozza meg változó méretű partíció esetén a következő algoritmusok felhasználásával: first fit, best fit a foglalási igényeknek megfelelő helyfoglalást – táblázatos formában! Magyarázza a kapott eredményeket és hogyan lehet az eredményen javítani!

First Fit algoritmus: *(FirstFit_BestFit.xlsx)*

FF		Szabad memóriaterületek													
monopr		R1(23)		R2(64)		R3(10)		R4(80)		R5(12)		R6(50)		R7(40)	
proc	KB	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt
P1	65							**P1(65)**	**15**						
P2	21	**P2(21)**	**2**												
P3	45			**P3(45)**	**19**										
P4	13											**P4(13)**	**37**		
P5	62	nem		nem		nem		nem		nem		nem		nem	

A fenti táblázat esetén a first fit algoritmus szerinti helyfoglalások láthatók monoprocess esetén, amikor is csak egy folyamat engedélyezett egy memóriaterületen.

A megvalósítás az alábbi lépésekkel történt:

- P1 process helyfoglalása 65KB. Az algoritmus R1 memóriaterülettől kezdve megvizsgálja lokációkat, hogy valamelyik terület elég nagy méretű-e ahhoz, hogy a folyamat beleférjen.
- P1 process számára az első alkalmas méretű területtel az R4(80) memóriaterület rendelkezik. Itt a folyamat lefoglal 65KB területet és marad 15KB szabad terület.
- Az algoritmus veszi a következő P2 process-t, melynek a helyfoglalása 21KB. Az algoritmus megvizsgálja, hogy a szabad memóriaterületek közül melyik rendelkezik a folyamat számára alkalmas területtel.
- P2 process számár az R1(23) az első alkalmas memóriaterület, így a folyamat itt lefoglal 21KB területet és marad 2KB szabad hely.
- Az algoritmus veszi a következő P3 process-t, melynek a helyfoglalása 45KB. Az algoritmus megvizsgálja, hogy a szabad memóriaterületek közül melyik rendelkezik a folyamat számára alkalmas területtel.
- P3 process számár az R2(64) az első alkalmas memóriaterület, így a folyamat itt lefoglal 45KB területet és marad 19KB szabad hely.
- Az algoritmus veszi a következő P4 process-t, melynek a helyfoglalása 13KB. Az algoritmus megvizsgálja, hogy a szabad memóriaterületek közül melyik rendelkezik a folyamat számára alkalmas területtel.
- P4 process számár az R6(50) az első alkalmas memóriaterület, így a folyamat itt lefoglal 13KB területet és marad 37KB szabad hely.
- Az algoritmus veszi a következő P5 process-t, melynek a helyfoglalása 62KB. Az algoritmus megvizsgálja, hogy a szabad memóriaterületek közül melyik rendelkezik a folyamat számára alkalmas területtel.
- Látható, hogy egyik szabad memóriaterület R3(10), R5(12), R7(40) közül egyik sem alkalmas a folyamat kiszolgálására, így a folyamat kiosztatlan marad.

Memóriagazdálkodási szempontból javít az eredményen, ha engedélyezve van a multiprocess lehetősége, vagyis egy memóriaterületet több folyamat is használhat.

FF multipr		Szabad memóriaterületek													
		R1(23)		R2(64)		R3(10)		R4(80)		R5(12)		R6(50)		R7(40)	
proc	KB	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt
P1	65							**P1(65)**	**15**						
P2	21	**P2(21)**	**2**												
P3	45			**P3(45)**	**19**										
P4	13			**P4(13)**	**6**										
P5	62	nem		nem		nem		nem		nem		nem		nem	

A megvalósítás az alábbi lépésekkel történt:

- P1 process helyfoglalása 65KB. Az algoritmus R1 memóriaterülettől kezdve megvizsgálja lokációkat, hogy valamelyik terület elég nagy méretű-e ahhoz, hogy a folyamat beleférjen.
- P1 process számára az első alkalmas méretű területtel az R4(80) memóriaterület rendelkezik. Itt a folyamat lefoglal 65KB területet és marad 15KB szabad terület.
- Az algoritmus veszi a következő P2 process-t, melynek a helyfoglalása 21KB. Az algoritmus megvizsgálja, hogy a szabad memóriaterületek közül melyik rendelkezik a folyamat számára alkalmas területtel.
- P2 process számár az R1(23) az első alkalmas memóriaterület, így a folyamat itt lefoglal 21KB területet és marad 2KB szabad hely.
- Az algoritmus veszi a következő P3 process-t, melynek a helyfoglalása 45KB. Az algoritmus megvizsgálja, hogy a szabad memóriaterületek közül melyik rendelkezik a folyamat számára alkalmas területtel.
- P3 process számár az R2(64) az első alkalmas memóriaterület, így a folyamat itt lefoglal 45KB területet és marad 19KB szabad hely.
- Az algoritmus veszi a következő P4 process-t, melynek a helyfoglalása 13KB. Az algoritmus megvizsgálja, hogy a szabad memóriaterületek közül melyik rendelkezik a folyamat számára alkalmas területtel.
- P4 process számár az még van elegendő hely a P3(45) által használt R2(19) memóriaterületen, így a P4 folyamat itt lefoglal 13KB-ot és marad 6KB további szabad hely.
- Az algoritmus veszi a következő P5 process-t, melynek a helyfoglalása 62KB. Az algoritmus megvizsgálja, hogy a szabad memóriaterületek közül melyik rendelkezik a folyamat számára alkalmas területtel.
- Látható, hogy bár a monoprocess-hez képest eggyel több memóriaterület maradt szabad R3(10), R5(12), R6(50), R7(40), de egyik sem rendelkezik elegendő területtel a P5 folyamat kiszolgálására, így kiosztatlan marad.

Best Fit algoritmus: *(FirstFit_BestFit.xlsx)*

BF monopr		Szabad memóriaterületek													
		R1(23)		R2(64)		R3(10)		R4(80)		R5(12)		R6(50)		R7(40)	
proc	KB	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt
P1	65							**P1(65)**	**15**						
P2	21	**P2(21)**	**2**												
P3	45											**P3(45)**	**5**		
P4	13													**P4(13)**	**27**
P5	62			**P5(62)**	**2**										

A fenti táblázat esetén a Best Fit algoritmus szerinti helyfoglalások láthatók monoprocess esetén, amikor is csak egy folyamat engedélyezett egy memóriaterületen.

A megvalósítás az alábbi lépésekkel történt:

- Az algoritmus veszi az első P1 process-t, melynek a helyfoglalási igénye 65KB és R1-től kezdve sorban megvizsgálja, hogy melyik memóriaterült felhasználásával marad a legkevesebb szabad terület a folyamat helyfoglalását követően.
- P1 process esetén az R4(80) memóriaterület adja a legkevesebb szabad területet, így a folyamat befoglal 65KB-ot és marad 15KB további szabad terület.
- Az algoritmus veszi a soron következő P2 folyamatot, melynek helyfoglalási igénye 21KB és megvizsgálja a szabad memoriterületeket, hogy a process helyfoglalása után melyik adja a legkevesebb fennmaradó szabad területet.
- P2 process esetén az R1(23) memóriaterület adja a legkevesebb szabad területet, így a folyamat befoglal 21KB-ot és marad 2KB további szabad terület.
- Az algoritmus veszi a soron következő P3 folyamatot, melynek helyfoglalási igénye 45KB és megvizsgálja a szabad memoriterületeket, hogy a process helyfoglalása után melyik adja a legkevesebb fennmaradó szabad területet.
- P3 process esetén az R6(50) memóriaterület adja a legkevesebb szabad területet, így a folyamat befoglal 21KB-ot és marad 2KB további szabad terület.
- Az algoritmus veszi a soron következő P4 folyamatot, melynek helyfoglalási igénye 13KB és megvizsgálja a szabad memoriterületeket, hogy a process helyfoglalása után melyik adja a legkevesebb fennmaradó szabad területet.
- P4 process esetén az R7(40) memóriaterület adja a legkevesebb szabad területet, így a folyamat befoglal 13KB-ot és marad 27KB további szabad terület.
- Az algoritmus veszi a soron következő P5 folyamatot, melynek helyfoglalási igénye 62KB és megvizsgálja a szabad memoriterületeket, hogy a process helyfoglalása után melyik adja a legkevesebb fennmaradó szabad területet.
- P5 process esetén az R2(64) memóriaterület adja a legkevesebb szabad területet, így a folyamat befoglal 62KB-ot és marad 2KB további szabad terület.
- R3(10) és R5(12) memóriaterületek használatlanok maradnak.

Memóriagazdálkodási szempontból a best fit esetén is javít az eredményen, ha engedélyezve van a multiprocess lehetősége, vagyis egy memóriaterületet több folyamat is használhat.

BF multipr		Szabad memóriaterületek													
		R1(23)		R2(64)		R3(10)		R4(80)		R5(12)		R6(50)		R7(40)	
proc	KB	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt	Foglalt	Maradt
P1	65							**P1(65)**	**15**						
P2	21	**P2(21)**	**2**												
P3	45											**P3(45)**	**5**		
P4	13							**P4(13)**	**2**						
P5	62			**P5(62)**	**2**										

A megvalósítás az alábbi lépésekkel történt:

- Az algoritmus veszi az első P1 process-t, melynek a helyfoglalási igénye 65KB és R1-től kezdve sorban megvizsgálja, hogy melyik memóriaterült felhasználásával marad a legkevesebb szabad terület a folyamat helyfoglalását követően.
- P1 process esetén az R4(80) memóriaterület adja a legkevesebb szabad területet, így a folyamat befoglal 65KB-ot és marad 15KB további szabad terület.
- Az algoritmus veszi a soron következő P2 folyamatot, melynek helyfoglalási igénye 21KB és megvizsgálja a szabad memoriterületeket, hogy a process helyfoglalása után melyik adja a legkevesebb fennmaradó szabad területet.
- P2 process esetén az R1(23) memóriaterület adja a legkevesebb szabad területet, így a folyamat befoglal 21KB-ot és marad 2KB további szabad terület.
- Az algoritmus veszi a soron következő P3 folyamatot, melynek helyfoglalási igénye 45KB és megvizsgálja a szabad memoriterületeket, hogy a process helyfoglalása után melyik adja a legkevesebb fennmaradó szabad területet.
- P3 process esetén az R6(50) memóriaterület adja a legkevesebb szabad területet, így a folyamat befoglal 21KB-ot és marad 2KB további szabad terület.
- Az algoritmus veszi a soron következő P4 folyamatot, melynek helyfoglalási igénye 13KB és megvizsgálja a szabad memoriterületeket, hogy a process helyfoglalása után melyik adja a legkevesebb fennmaradó szabad területet.
- P4 process esetén az R4-es memóriaterületen maradt még a folyamat számára elegendő terület, miután a P1 process már befoglalta a számára szükséges 65KB területet. A P4 process így az R4(15) memóriaterületre helyezkedik el, ahol lefoglal 13KB-ot és marad 2KB szabad terület.
- Az algoritmus veszi a soron következő P5 folyamatot, melynek helyfoglalási igénye 62KB és megvizsgálja a szabad memoriterületeket, hogy a process helyfoglalása után melyik adja a legkevesebb fennmaradó szabad területet.
- P5 process esetén az R2(64) memóriaterület adja a legkevesebb szabad területet, így a folyamat befoglal 62KB-ot és marad 2KB további szabad terület.
- A multiprocess hatására a monoprocess-hez képest az R3(10) és R5(12) memóriaterületek mellett az R7(40) terület is használatlan marad.

Értékelés:

A fenti példákból megállapítható, hogy erőforráskezelés szempontjából javulást lehet elérni a multiprocess alkalmazásával.

Látható az is, hogy a két algoritmus közül ugyan a first fit a leggyorsabb, de nem képes allokálni az összes folyamatot, míg a best fit sikeresen allokál minden folyamatot valamely memóriaterületre, de a minden egyes folyamat esetén a legmegfelelőbb terület kiválasztása időigényesebb és jóval komplexebb számítási igényű.

Módszerek az eredmények javítására:

Az algoritmusok hatékonyságának növelése érdekében érdemes szemétgyűjtést (garbage collection) alkalmazni. A szemétgyűjtés, ha van kielégítettlen folyamat, akkor a memória allokáció futás idejű átrendezését hajtja végre, új szabad memóriaterületek kialakítása érdekében. A szemétgyűjtésnek a hátránya, hogy erőforrásigényes.