Operációs rendszerek BSc

11.gyak.

2021. 04. 27.

Készítette: Nagy Róbert

Programtervező Informatikus
Neptunkód:JMDRGG

Miskolc, 2021

1.feladat:
First fit: A rendelkezésre álló szabad területek közül a legelső elegendő méretűt foglaljuk le.

first fit

	Memória terület - szabad terület					
Foglalási igény	30	35	15	25	75	45
39					36 (75 - 39)	
40						5 (45 - 40)
33		2 (35 - 33)				
20				5 (25 - 20)		
21	9 (30 - 21)					

Next fit: Az Első megfelelő stratégiával szemben itt a keresést nem a tár elejéről kezdjük, hanem az után a terület után, amit legutoljára foglaltunk.

next fit

	Memória terület - szabad terület					
Foglalási igény	30	35	15	25	75	45
39					36 (75 - 39)	
40						5 (45 - 40)
33		2 (35 - 33)				
20				5 (25 - 20)		
21					15 (36 - 21)	

Best fit: A legkisebbet foglaljuk le azon szabad területek közül, amelyek legalább akkorák, mint a lefoglalandó terület.

best fit

	Memória terület - szabad terület					
Foglalási igény	30	35	15	25	75	45
39						6 (45 - 39)
40					35 (75 - 40)	
33		2 (35 - 33)				
20				5 (25 - 20)		
21	9 (30 - 21)					

Worst fit: Az elérhető legnagyobb szabad területet allokáljuk. A spekuláció az, hogy a maradék terület még talán elegendő lesz egy újabb foglalás számára.

worst fit

	Memória terület - szabad terület					
Foglalási igény	30	35	15	25	75	45
39					36 (75 - 39)	
40						5 (45 - 40)
33					3 (36 - 33)	
20		15 (35 - 20)				
21	9 (30 - 21)					

2.feladat: semset.c
Létrehoz n darab szemafort, majd inicializálja 0 értékekkel.
A semget hívással lehet létrehozni a szemafort, hasonlóan a korábbi ipc mechanizmusokhoz itt is kell egy kulcs, illetve jogosultságok.
A semctl hívással tudjuk a szemafort kontrollálni, ebben a SETALL flag beállítja az arg union semun típusú array nevű tömbjében szereplő értékekre. A semun union tartalmazza a számot, amit az adott szemafor tartalmaz, descriptort, tömböt és egy információt tartalmazó buffert.

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdlib.h>
#define KEY 123456L

union semun {
    int val;                  /* Value for SETVAL */
    struct semid_ds *buf;     /* Buffer for IPC_STAT, IPC_S
    unsigned short *array;    /* Array for GETALL, SETALL *
    struct seminfo *__buf;    /* Buffer for IPC_INFO (Linux
};

void main() {
    union semun arg;

    int n = 5;
    int semID = semget(KEY, n, IPC_CREAT | 0666);

    if (semID == -1) {
        perror("Nem sikerult szemaforokat letrehozni");
        exit(-1);
    }

    arg.array = (short *)calloc(n, sizeof(int));


    if (semctl(semID, 0, SETALL, arg)) {
        perror("Nem sikerult beallitani az erteket\n");
        exit(-1);
    }
```

Semval.c
Itt lekérdezzük a tartalmat. Mivel itt nem kell létrehozni új szemafort, ezért itt argumentumnak csak a Key kell.

```c
int semID = semget(KEY, 0, 0);
int n = 5;
if (semID == -1) {
    perror("Nem sikerult szemaforokat lekerdezni\n");
    exit(-1);
}

union semun arg;

printf("Szemaforok tartalma: \n");
arg.array = (short *)calloc(n, sizeof(int));

semctl(semID, 0, GETALL, arg);

for (int i = 0; i < n; i++)
    printf("%d ", arg.array[i]);
```

Futtatás eredménye:

```
→ JMDRGG_0427 git:(main) X cd "/mnt/c/Users/
mval
Szemaforok tartalma:
0 0 0 0 0 %
→ JMDRGG_0427 git:(main) X
```

Semkill.c

Szemaforok törlése a semctl hívás IPC_RMID flag használatával

```c
void main() {
    int n = 5;
    int semID = semget(KEY, 0, 0);
    if (semID == -1) {
        perror("Nem sikerult szemaforokat lekerdezni\n");
        exit(-1);
    }

    for (int i = 0; i < n; i++)
        semctl(semID, i, IPC_RMID);

}
```

Semup.c

A semop függvény feladata egy szemaforköteg egy elemének növelése és csökkentése. Az előző programok létrehoztak egy 5 elemű szemafort. Ez a program a 4. szemafort fogja inkrementálni.

A sembuf strucktúrában van egy sem_num változó, ami a szemaforok számát fogja kapni. A sem_op inkrementálás(1) vagy dekrementálás(-1) lehet. A sem_flg pedig a hosszáférési jogokat tartalmazza.

```
    struct sembuf buffer;

    buffer.sem_num = 4;      //a 4.ik szemafort
    buffer.sem_op = 1;       //inkrementaljuk a szemaforokat
    buffer.sem_flg = 0666;   //jogok

    if (semop(semID, &buffer, 1)) {
        perror("Sikertelen\n");
        exit(-1);
    }
```

Futtatások eredménye:

```
→  JMDRGG_0427 git:(main) x code .
→  JMDRGG_0427 git:(main) x ./semset
→  JMDRGG_0427 git:(main) x ./semval
Szemaforok tartalma:
0 0 0 0 0 %
→  JMDRGG_0427 git:(main) x ./semup
→  JMDRGG_0427 git:(main) x ./semval
Szemaforok tartalma:
0 0 0 0 1 %
→  JMDRGG_0427 git:(main) x
```

2.a feladat: Megpróbáljuk lekérdezni a szemafort, ha ez nem lehetséges, mert nem létezik, akkor az errno változó az ENOENT értéket kapja értékül. Ha ez teljesül akkor bekérünk egy számot, különben 1 re állítjuk a szemafor értékét.

```c
void main() {
    union semun arg;

    int semID = semget(KEY, 0, 0);
    if (errno == ENOENT) {
        semID = semget(KEY, 1, IPC_CREAT | 0666);
        printf("Szam: ");
        scanf("%d" ,&(arg.val));

    } else {
        arg.val = 1;
    }


    semctl(semID, 0, SETVAL, arg);

    printf("A szemafor értéke (1) : %d\n", semctl(semID, 0, GETVAL));

}
```

Másik processz használja a szemafort és a kritikus szakaszában legyen egy 3 másodperces sleep hívás.

```c
void main() {
    int semID = semget(KEY, 0, 0);

    if (semID == -1) {
        perror("Nem sikerult megnyitni\n");
        exit(-1);
    }

    //belepesi szakasz
    printf("Kritikus szakasz\n");
    down(semID);
    sleep(3);
    printf("pid : %d\n", getpid());
    printf("%d \n", semctl(semID, 0, GETVAL));
    up(semID);
    printf("kritikus szakasz vege\n");
}
```

```
void up(int semId) {
    struct sembuf buffer;
    buffer.sem_num = 0;
    buffer.sem_op = 1;
    buffer.sem_flg = 0;

    semop(semId, &buffer, 1);
}

void down(int semId) {
    struct sembuf buffer;
    buffer.sem_num = 0;
    buffer.sem_op = -1;
    buffer.sem_flg = 0;

    semop(semId, &buffer, 1);
}
```

Harmadik program torolje a szemafort.

```
void main() {
    int semID = semget(KEY, 0, 0);

    if (semID == -1) {
        perror("Nem sikerult megnyitni\n");
        exit(-1);
    }

    if (semctl(semID, 0, IPC_RMID) == -1) {
        perror("Nem sikerult torolni\n");
        exit(-1);
    }

    printf("Torolve\n");


}
```

Futtatás eredménye:

```
→  JMDRGG_0427 git:(main) x ./gyak11_2
Szam: 31
A szemafor értéke (1) : 31
→  JMDRGG_0427 git:(main) x
```

```
→  JMDRGG_0427 git:(main) x ./gyak11_2_masik
Kritikus szakasz
pid : 5534
28
kritikus szakasz vege
→  JMDRGG_0427 git:(main) x
```

```
→  JMDRGG_0427 git:(main) x ./gyak11_2_masik
Kritikus szakasz
pid : 5535
28
kritikus szakasz vege
→  JMDRGG_0427 git:(main) x
```

```
→  JMDRGG_0427 git:(main) x ./gyak11_2_masik
Kritikus szakasz
pid : 5536
29
kritikus szakasz vege
→  JMDRGG_0427 git:(main) x
```

```
→  JMDRGG_0427 git:(main) x ./gyak11_2_masik
Kritikus szakasz
pid : 5550
30
kritikus szakasz vege
→  JMDRGG_0427 git:(main) x
```

```
  JMDRGG_0427 git:(main) x ./gyak11_2_harmadik
orolve
  JMDRGG_0427 git:(main) x
```