Ftok i Semafory

Poznajemy sposoby tworzenia unikalnego klucza oraz zasady działania semaforów.

1.Wstęp teoretyczny

•Do tworzenia służy funkcja semget

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget (key_t key, int nsem, int semflag);
```

•zwraca identyfikator semafora

•nsem - ile semaforów w zbiorze, jeżeli nie tworzymy tylko otwieramy już dany zbiór semaforów to można dać tu 0. (w utworzonym zbiorze semaforów nie można zmienić ich liczby)

•semflag - jest kombinacją stałych symbolicznych określających prawa dostępu

Wartość liczbowa	Stała Symboliczna	Znaczenie
0400	SEM_R	czytanie przez właściciela
0200	SEM_A	zmienianie przez właściciela
0040	SEM_R>>3	czytanie przez grupę
0020	SEM_A>>3	zmienianie przez grupę
0004	SEM_R>>6	czytanie przez innych
0002	SEM_A>>6	zmienianie przez innych
1000	IPC_CREAT	
2000	IPC_EXCL	

•Do wykonywania operacji na semaforach służy funkcja

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semop (int semid, struct sembuf *opstr, unsigned int nops;
```

•funkcja zwraca 0 jeśli się powiedzie lub -1 w przypadku błędu

•opstr wskazuje na tablice następujących struktur:

```
struct sembuf
    {
    ushort sem_num; /* numer semafora */
    short semi_op; /* operacja na semaforze */
    short sem_flag; /* znacznik operacji */
    };
```

•każdy element tej tablicy określa operację na wartości jednego semafora ze zbioru semaforów

•sem_num - określa który semafor (licząc od 0)

•sem_op

•>0 wartość tą dodaj do bieżącego semafora (uwolnij zasoby) operacja V(s)

•=0 proces wywołujący funkcję semop chce czekać, aż wartością semafora stanie się 0

•<0 proces wywołujący czeka aż wartość semafora stanie się większa niż (lub taka sama jak) wartość bezwzględna tego pola. Następnie zostaną zsumowane, czyli przydział zasobów operacja P(s). Np. s=1+(-1). s=0 semafor opuszczony.

•sem_flag - ma kilka opcji np.

•SEM_UNDO - cofanie zmian wykonanych przez proces na tym semaforze jeżeli proces "padnie"

•IPC_NOWAIT - na sem_flag informuje system że nie chcemy czekać , aż operacja będzie ukończona.

•semid identyfikator semafora

•nops - liczba elementów w tablicy struktur sembuf na którą wskazuje opstr

•Mamy zapewnione że przekazana do funkcji semop tablica operacji będzie wykonana jako jedna operacja niepodzielna. Jądro wykona albo wszystkie operacje albo nie wykona żadnej

•Do sterujących semaforami służy

int semctl(int semid, struct semnum, int cmd, union semun arg);
gdzie:

•semid - identyfikator semafora

•semun - zbudowana następująco:

```
union semun {
                    int val; /* używane tylko dla SETVAL */
                    struct semid_ds. *buff; /* używane dla IPC_STAT oraz IPC_SET */
                    ushort *array; /*używane dla GETVAL oraz SETVAL */
                    }arg;
```

•cmd - polecenie:
•IPC_RMID - usunięcie semafora

•GETVAL - pobranie wartości semafora, funkcja zwróci jego wartość

•SETVAL - nadawanie wartości semaforowych val w unii semun;

•senum - określa którego semafora dotyczy

2.Funkcja ftok

```
#include <sys/types.h>
    #include <sys/ipc.h>
    key_t ftok(char *pathname, char proj);
```

•Na podstawie nazwy ścieżki i numeru projektu (proj jest liczbą 8b), tworzy nam (prawie) unikalny klucz. Prawie! bo jest to liczba 32 bitowa a jest tworzona na podstawie i-węzła (32b) nr projektu(8b) i tzw. małego numeru urządzenia systemu plików (8b). Istnieje więc bardzo mała szansa, że dla różnych ścieżek będą te same numery. W praktyce jednak jest to niemal niemożliwe.
•Wpisać, skompilować i uruchomić poniższy program:

```
#include <sys/types.h>
#include <sys/ipc.h>
main()
{
key_t klucz;
klucz = ftok("/tmp/",3); //plik musi być bo jak nie to ftok zwróci -1
printf("ftok 0x%X\n",klucz);
}
```

•zmienić nazwę programu i uruchomić, jakie dostaliśmy identyfikatory?
•zmienić nr projektu lub/i ścieżkę, porównać otrzymane wyniki

3.Semafory

•Przepisać i uruchomić poniższy program:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#define PERMS 0666
static struct sembuf op_lock[1] = {
    0, -1, 0
};
static struct sembuf op_unlock[1] = {
    0, 1, 0
};
void blokuj(int semid)
{
if (semop(semid, &op_lock[0], 1)<0)
        perror("blad lokowania semafora");
}
void odblokuj(int semid)
{
if (semop(semid, &op_unlock[0], 1) < 0)
    perror("blad odlokowania semafora");
}
main()
{
```

```
int semid = -1; // identyfikator semafora
int co;
int jeszcze;
if  ((semid = semget(ftok("/tmp",0), 1, IPC_CREAT | PERMS)) < 0)
    perror("blad tworzenia semafora");
printf("Podaj Polecenie\n 1 - podnies semafor\n 2 - opusc semafor\n 0 - wyjscie\n");
for (jeszcze = 1;jeszcze;)
    {
    scanf("%d",&co);
    printf("wybrano %d\n",co);
    switch(co)
        {
        case 2:
            {
            printf("przed blokuj\n");
            blokuj(semid);
            printf("po blokuj\n");
            break;
            }
        case 1:
            {
            printf("przed odblokuj\n");
            odblokuj(semid);
            printf("po odblokuj\n");
            break;
            }
        case 0:
            {
            jeszcze = 0;
            break;
            }
        default:
            {
            printf("nie rozpoznana komenda %d\n",co);
            }
        }
    }
}
```

•Co się dzieje gdy wykonamy operacje V (podniesienie) za pierwszym razem?
•Co się stanie gdy wykonamy operacje V za drugim razem?
•Ile razy możemy teraz wykonać operację P (opuszczenie)?
•Otworzyć drugą konsolę i uruchomić nasz program, podnieść semafor, co się stało na pierwszym terminalu?
•Opuścić semafor w pierwszym terminalu, co się stało?
•Otworzyć trzeci terminal i podnieść semafor, czy na obu terminalach doszło do odblokowania?
•Na trzecim terminalu jeszcze raz podnieść semafor
•Gdy wszystkie terminale odblokowane wyjść przez 0.

•Komendą ipcs sprawdzić czy semafor został w systemie.

•Komendą ipcrm -s usunąć semafor z systemu

•zmodyfikować tablicę operacji tak by operacja opuszczenia semafora była nie blokująca (IPC_NOWAIT)
•Uruchomić program i spróbować opuścić semafor podnieść go po czym znowu opuścić.
•Wyjść i usunąć semafor, można usunąć IPC_NOWAIT
•Uruchomić program podnieść kilka razy semafor i wyjść (nawet przez ^C)
•Nie usuwać semafora, uruchomić program jeszcze raz tym razem próbując opuścić semafor. Czy stan semafora został zapamiętany?
•Wprowadzić modyfikacje do tablic operacji polegającą na dodaniu opcji SEM_UNDO
•Usunąć semafor, skompilować i uruchomić program.
•Powtórnie podnieść kilka razy semafor i wyjść
•uruchomić i spróbować opuścić semafor. Czy udało nam się zapamiętać ten stan?

W powyższym przykładzie nowo zainicjowany semafor był zawsze opuszczony. Możemy zrealizować semafor na odwrót (powinien czekać aż będzie 0 i operacja P będzie zwiększać wartość semafora natomiast operacja V znmiejszać).

```
static struct sembuf op_lock[2] = {
```

```
        0,0,0, /* czekaj, aż semafor nr 0 stanie się zerem */
        0,1,0 /* następnie zwiększ ten semafor  1*/
};
static struct sembuf op_unlock[1] = {
        0, -1, IPC_NOWAIT /* zmniejsz semafor nr 0 o 1 bez czekania bo to zwolnienie
zasobów */
};
```

•Spróbować wykonać operacje blokowania i odblokowywania za pomocą nowych operacji
•Napisać "programik" usuwający semafor.