

Semafory

Systemy Operacyjne 2

Piotr Zierhoffer

10 grudnia 2011

IPC — Inter Process Communication

- kolejki komunikatów,
- pamięć współdzielona
- semafony
- polecenia bash: `ipcs`, `ipcrm`

Semafor

- struktura służąca do synchronizacji współbieżnego dostępu do zasobów
 - sekcja krytyczna
- identyfikowana kluczem
- atomowe operacje
- tworzone w zestawach

Edsger Wybe Dijkstra

Semafor binarny

- liczba binarna $S \in \{0, 1\}$
- “podniesiony” lub “opuszczony”

Semafor uogólniony

- wiele stanów
- liczba $S \in \{0, 1, \dots, \infty\}$

Operacje:

- opuszczenie, release, down, wait — p (prolaag)
- podniesienie, acquire, up, signal — v (verhoog)

Mordechai Ben-Ari

Definicja

Semafor jest pewną całkowitą liczbą nieujemną

Opuszczenie semafora jest równoważne wykonaniu instrukcji:

- jeśli $S > 0$, to $S = S - 1$
- w przeciwnym razie wstrzymaj działanie procesu próbującego opuścić semafor

Podniesienie semafora:

- jeśli są procesy wstrzymane przy próbie opuszczenia semafora S , to wznów jeden z nich,
- w przeciwnym wypadku $S = S + 1$

Mechanizmy IPC pozwalają na tworzenie zestawów semaforów. Możliwe jest wykonywanie operacji grupowych na semaforach w zestawie, z zasadą “wszystko albo nic”.

Każdy semafor opisany jest strukturą:

```
struct sem {
    ushort semval; //wartość semafora
    pid_t sempid; //PID procesu, który wykonał ostatnią
        //operację na semaforze
    ushort semncnt; //liczba procesów oczekujących na
        //podniesienie semafora
    ushort semzcnt; //liczba procesów oczekujących na
        //wartość 0 semafora
}
```

- Funkcja:
`int semget(key_t key, int nsems, int semflg)`
- Parametry:
 - `key` — klucz identyfikujący zestaw semaforów (“nazwa”)
 - `nsems` — ilość semaforów w zestawie (numerowane od 0)
 - `semflg` — flagi
- Wartość zwracana:
 - -1 w wypadku błędu
 - identyfikator zestawu powiązanego z kluczem `key`

Uwaga!

Semafor musi zostać zainicjalizowany!

Opis parametrów:

- key — klucz identyfikujący zestaw (“nazwa”)
 - IPC_PRIVATE — stwórz nowy zestaw o dowolnym identyfikatorze, ignoruj semflg oprócz praw dostępu
 - zwyczajowo wartość heksadecymalna
- nsems — interpretowane tylko z flagą IPC_CREAT (ale! patrz man semget -> EINVAL)
- semflg — flagi, sumowane bitowo
 - prawa dostępu — X nieużywane
 - IPC_CREAT — tworzenie kolejki
 - IPC_EXCL — “jeżeli kolejka istnieje i użyto IPC_CREAT, zwróć błąd”

- Funkcja:

```
int semctl(int semid, int semnum, int cmd)
int semctl(int semid, int semnum, int cmd, union
semun ctrl_arg)
```

- Parametry:

- `semid` — identyfikator zestawu zwrócony przez `semget`
- `semnum` — identyfikator semafora (liczone od 0)
- `cmd` — operacja IPC, na pojedynczym semaforze lub na grupie semaforów
- `ctrl_arg` — parametry operacji

- Wartość zwracana:

- -1 w wypadku błędu
- wartość nieujemna zależna od `cmd` lub 0

Tradycyjne operacje IPC:

- `IPC_STAT` — zwraca wartości struktury `semid_ds` dla zestawu o identyfikatorze `semid`, umieszczając ją w strukturze podanej jako 4. argument
- `IPC_SET` — modyfikuje wartości struktury `semid_ds`
- `IPC_RMID` — usuwa zestaw semaforów

Operacje na semaforze `semnum`:

- `GETVAL` — zwraca wartość semafora (`semval`)
- `SETVAL` — ustawia wartość semafora
- `GETPID` — zwraca wartość `sempid`
- `GETNCNT` — zwraca wartość `semncnt`
- `GETZCNT` — zwraca wartość `semzcnt`

Operacje na wszystkich semaforach z zestawu:

- GETALL – umieszcza wartości wszystkich semaforów w tablicy podanej jako 4. argument
- SETALL — ustawia wartości wszystkich semaforów na podstawie tablicy podanej jako 4. argument

Argument `ctrl_arg`:

```
union semun {  
    int val; //wartość dla SETVAL  
    struct semid_ds *buf; //bufor dla IPC_STAT i IPC_SET  
    unsigned short *array; //tablica dla GETALL i SETALL  
}
```

```
//stworzenie semaforów
int semid = semget(0xABC, 6, IPC_CREAT | 0600);

short tab[5] = {2,1,1,1,0};

//ustawienie 5 semaforów
semctl(semid, 0, SETALL, tab);

//ustawienie ostatniego semafora
semctl(semid, 5, SETVAL, 4);
```

- Funkcja:
`int semop(int semid, struct sembuf *sops, unsigned nsops)`
- Parametry:
 - `semid` — identyfikator zestawu semaforów
 - `sops` — **tablica struktur** definiujących operacje na semaforach
 - `nsops` — liczba elementów tablicy `sops`
- Uwagi:
 - operacje wykonywane są **atomowo!**
 - albo wykonują się wszystkie, albo żadna
 - wykonują się w “tym samym momencie”
- Wartość zwracana:
 - 0 lub -1 w wypadku błędu

Struktura komendy:

```
struct sembuf{  
    //numer semafora  
    unsigned short sem_num;  
  
    //operacja  
    short sem_op;  
  
    //flagi  
    short sem_flg;  
}
```

Operacja `sem_op`:

- `sem_op > 0` — operacja v, podnieś semafor o `sem_op`
- `sem_op = 0` — oczekiwanie, aż semafor będzie miał wartość 0
- `sem_op < 0` — operacja p, opuść semafor o `sem_op`

Flagi `sem_flg`:

- `IPC_NOWAIT` — nie oczekuj na wykonanie operacji, tylko zwróć błąd
- `SEM_UNDO` — wycofaj efekty operacji, jeżeli proces się zakończy
 - może nie zadziałać zgodnie z oczekiwaniami — patrz manual

Uwaga!

Operacje `sem_op = 0` i `sem_op < 0` mogą spowodować blokowanie całego zestawu operacji. System Linux, przeglądając zestawy zablokowanych operacji, najpierw odblokowuje te “najłatwiejsze”, a więc może dojść do zagłodzenia!
Operacja `semop` **nie ma gwarancji żywotności**.

```
struct sembuf operacja;  
  
void p(int semid, int semnum, int ile) {  
    operacja.sem_num = semnum;  
    operacja.sem_op = -ile;  
    operacja.sem_flg = 0;  
  
    if(semop(semid, &operacja, 1) == -1) {  
        perror("Błąd opuszczania semafora");  
        exit(1);  
    }  
}
```

```
struct sembuf operacja;

void v(int semid, int semnum, int ile) {
    operacja.sem_num = semnum;
    operacja.sem_op = ile;
    operacja.sem_flg = 0;

    if(semop(semid, &operacja, 1) == -1) {
        perror("Błąd podnoszenia semafora");
        exit(1);
    }
}
```


Zestaw wielu operacji

```
struct sembuf zestaw[5];
for(int i = 0; i < 5; i++)
{
    zestaw[i].sem_num = i;
    zestaw[i].sem_op = -1;
    zestaw[i].sem_flg = 0;
}

if(semop(semid, zestaw, 5) == -1) {
    perror("Błąd wykonania zestawu operacji");
    exit(1);
}
```

Zadanie 1

Zmodyfikować zadanie 1 z pamięci współdzielonej z wykorzystaniem semaforów.

Zadanie 2

Zbadać zachowanie funkcji semop przy wielu operacjach na jednym semaforze:

- inicjalizacja semafora na 1
- opuszczenie semafora o 2
- podniesienie semafora o 1

Zadanie 3

Zaimplementować rozwiązanie problemu producenta i konsumenta z wykorzystaniem dwóch procesów oraz z magazynem o wielkości 10.