Pamięć współdzielona, semafory

Systemy Operacyjne 2

Piotr Zierhoffer

10 grudnia 2011

Mechanizmy IPC

Pamięć współdzielona

- segment wirtualnej przestrzeni adresowej
- identyfikowany kluczem
- o określonej długości i prawach dostępu
- współdzielony przez wiele procesów
- zawartość usuwana przy usunięciu segmentu
- często wymaga synchronizacji semaforami

Semafor

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

Stworzenie segmentu pamięci współdzielonej

- Funkcja:
  `int shmget(key_t key, size_t size, int shmflg)`
- Parametry:
  - `key` — klucz identyfikujący segment ("nazwa")
  - `size` — rozmiar segmentu w bajtach
  - `shmflg` — flagi
- Wartość zwracana:
  - -1 w wypadku błędu
  - identyfikator segmentu powiązanego z kluczem `key`

Stworzenie segmentu, cd.

Opis parametrów:

- `key` — klucz identyfikujący segment ("nazwa")
  - `IPC_PRIVATE` — stwórz nowy segment o dowolnym identyfikatorze, ignoruj `shmflg` oprócz praw dostępu
  - zwyczajowo wartość heksadecymalna
- size — rozmiar segmentu pamięci, zaokrąglany w górę do wielokrotności rozmiaru strony (`getconf PAGE_SIZE`)
- `shmflg` — flagi, sumowane bitowo
  - prawa dostępu — w niektórych implementacjach nieużywane
  - `IPC_CREAT` — tworzenie kolejki
  - `IPC_EXCL` — "jeżeli kolejka istnieje i użyto `IPC_CREAT`, zwróć błąd"

Przyłączenie segmentu

- Funkcja:
  `void* shmat(int shmid, char *shmaddr, int shmflg)`
- Parametry:
  - `shmid` — identyfikator segmentu zwrócony przez `shmget`
  - `shmaddr` — adres, pod którym segment ma być widoczny w procesie
  - `shmflg` — flagi
- Wartość zwracana:
  - -1 w wypadku błędu
  - adres, pod którym widoczny jest przyłączony segment pamięci

Przyłączenie segmentu, cd.

- `shmflg` — flagi specyfikujące zachowanie funkcji
  - `SHM_RDONLY` — przyłącz pamięć tylko do odczytu
  - `SHM_REMAP`, `SHM_RND` — dotyczą montowania pod konkretny adres
- `shmaddr` — adres, pod którym segment ma być widoczny
  - `NULL` — zalecana wartość, system sam określa adres (różne adresy względne w różnych procesach!)
  - dowolny adres wyrównany do granicy strony pamięci

Odłączenie segmentu

- Funkcja:
  `int shmdt(const void* shmaddr)`
- Parametry:
  - `shmaddr` — adres początku segmentu pamięci, zwrócony przez `shmat`
- Uwagi:
  - odłączenie segmentu we wszystkich procesach nie powoduje jego usunięcia
  - segment nie zostanie usunięty aż wszystkie procesy go odłączą oraz zostanie oznaczony do usunięcia
- Wartość zwracana:
  - 0 lub -1 w wypadku błędu

Zarządzanie segmentami

- Funkcja:
  `int shmctl(int shmid, int cmd, struct shmid_ds *buf)`
- Parametry:
  - `msqid` — identyfikator segmentu pamięci zwrócony przez `shmget`
  - `cmd` — operacja do wykonania
    - IPC_RMID — zaznaczenie segmentu do usunięcia
    - usunięcie po odłączeniu wszystkich procesów
    - IPC_SET — zmiana parametrów segmentu: właściciela, grupy i praw na podstawie `buf`
    - IPC_STAT — pobranie parametrów segmentu do `buf`
  - `buf` — struktura na parametry operacji
- Wartość zwracana:
  - 0 lub -1 w wypadku błędu

Wywołania systemowe

Wpływ wywołań systemowych na segmenty pamięci współdzielonej:

fork() w wyniku wywołania fork() proces potomny dziedziczy dołączone segmenty pamięci wspólnej.

exec() po wykonaniu exec() wszystkie odwzorowane segmenty są odłączane (nie są usuwane).

exit() po wykonaniu exit() wszystkie dołączone segmenty pamięci wspólnej są odłączane (nie są usuwane).

Uwaga!

Jeden segment może być podłączony wiele razy w ramach jednego procesu!

Wykorzystanie pamięci współdzielonej

- Do pamięci współdzielonej odwołujemy się przez wskaźniki.
- Zapisać można dowolne dane z wyjątkiem wskaźników — są one zazwyczaj sensowne tylko w obrębie jednego procesu!
- Pamięci współdzielonej nie trzeba dodatkowo alokować.

```
int *base_addr;

int shmid = shmget(0xABC, MAX_SIZE * sizeof(int), IPC_CREAT);
if (shmid == -1) return -1;

addr = (int *) shmat(shmid, NULL, 0);

for(i = 0; i < MAX_SIZE, ++i)
     base_addr[i] = (int)i*i;


shmdt(addr);
```

Semafory

Edsger Wybe Dijkstra

Semafor binarny

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

Semafor uogólniony

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

Operacje:

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

Definicja operacji

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 $S = S + 1$

Zestaw semaforów

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
}
```

Tworzenie semaforów

- 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!

Tworzenie semaforów, cd.

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"

Zarządzanie semaforami

- 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

Zarządzanie semaforami, parametr cmd

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

Zarządzanie semaforami, cd.

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
}
```

Inicjalizacja semaforów

```
//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
semtcl(semid, 5, SETVAL, 4);
```

Operacje na semaforach

- 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

Operacje na semaforach, cd.

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

Operacje na semaforach, cd.

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.**

Opuszczenie semafora

```
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);
     }
}
```

Podniesienie semafora

```
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);
}
```