Mechanizmy z grupy IPC

Podobnie jak łącza, **IPC** (**Inter Process Communication**) jest grupą mechanizmów komunikacji i synchronizacji procesów działających w ramach tego samego systemu operacyjnego. W skład pakiety wchodzą:

- kolejki komunikatów — umożliwiają przekazywanie określonych porcji danych,
- pamięć współdzielona — umożliwiają współdzielenie kilku procesom tego samego fragmentu wirtualnej przestrzeni adresowej,
- semafory — umożliwiają synchronizacje procesów w dostępie do współdzielonych zasobów (np. do pamięci współdzielonej).

Kolejki komunikatów

Komunikat jest pakietem informacji przesyłanym pomiędzy różnymi procesami. Każdy komunikat ma określony typ i długość. Nadawca może wysyłać komunikaty, nawet wówczas gdy żaden z potencjalnych odbiorców nie jest gotów do ich odbioru. Komunikaty są w takich przypadkach buforowane w kolejce oczekiwania na odebranie. Odbiorca może oczekiwać na pierwszy przybyły komunikat lub na pierwszy komunikat określonego typu. Komunikaty w kolejce są przechowywane nawet po zakończeniu procesu nadawcy, tak długo, aż nie zostaną odebrane lub kolejka nie zostanie zlikwidowana.

```
#include <sys / types.h>
#include <sys / ipc.h>
#include <sys / msg.h>
```

Funkcja MSGGET

```
PROTOTYPE: int msgget( key_t key, int  msgflg );
  RETURNS: success : identyfikator kolejki komunikatów
             error: -1
               errno =EACCES (kolejka skojarzona z wartością
                              key, istnieje, ale proces
                              wołający funkcję nie ma
                              wystarczających praw dostępu
                              do kolejki)
                      EEXIST (kolejka skojarzona z wartością
                              key istnieje i msgflg zawiera
                              flagi IPC_CREAT i IPC_EXCL)
                      EIDRM  (kolejka została przeznaczona
                              do usunięcia)
                      ENOENT (kolejka skojarzona z
                              wartością key nie istnieje,
```

zaś msgflg nie zawiera flagi **IPC_CREAT**)

ENOMEM (brak pamięci na utworzenie nowej kolejki)

ENOSPC (próba przekroczenia systemowego ograniczenia na ilość istniejących w systemie kolejek komunikatów)

PARAMETRY:

1. key- liczba, która identyfikuje kolejkę
2. msgflg– jest sumą bitową stałej IPC_CREAT i dziewięciu bitów określających prawa dostępu do kolejki.

UWAGI:

IPC_PRIVATE nie jest flagą tylko szczególną wartością key_t**.** Jeśli wartość ta zostanie użyta jako parametr key, to system uwzględni jedynie bity uprawnień parametru msgflg i zawsze będzie próbować utworzyć nową kolejkę.
Istnienie flag IPC_CREAT i IPC_EXCL w parametrze msgflg znaczy tyle samo, w przypadku kolejki komunikatów, co istnienie flag O_CREAT i O_EXCL w argumencie mode wywołania open , tzn. funkcja msgget nie wykona się prawidłowo jeśli msgflg będzie zawierać flagi IPC_CREAT i IPC_EXCL , zaś kolejka komunikatów skojarzona z kluczem key już będzie istnieć.

Funkcja MSGSND

PROTOTYPE: int msgsnd(int msgid, struct msgbuf *msgp, int msgs, int msgflg);

RETURNS:success : 0
error: -1

errno =**EACCES**(kolejka skojarzona z wartością key, istnieje, ale proces wołający funkcję nie ma wystarczających praw dostępu do kolejki)

EAGAIN(kolejka jest pełna, a flaga IPC_NOWAIT była ustawiona)

EFAULT(niepoprawny wskaźnik msgp)

EIDRM(kolejka została przeznaczona do usunięcia)

EINTR(otrzymano sygnał podczas oczekiwania na operację zapisu)

EINVAL(niepoprawny identyfikator kolejki, lub ujemny typ wiadomości, lub nieprawidłowy rozmiar wiadomości)

PARAMETRY:

1. `msgid` - identyfikator kolejki
2. `msgp` – wskaźnik do obszaru pamięci zawierającego treść komunikatu
3. `msgs` - rozmiar właściwej treści komunikatu
4. `msgflg` – flagi specyfikujące zachowanie się funkcji w warunkach nietypowych
   Wartość ta może być ustawiona na 0 lub **`IPC_NOWAIT`** (jeśli kolejka komunikatów jest pełna wtedy wiadomość nie jest zapisywana do kolejki, a sterowanie wraca do procesu. Gdyby flaga nie była ustawiona, proces jest wstrzymywany tak długo, aż zapis wiadomości nie będzie możliwy)

UWAGI:

Jeśli w kolejce komunikatów nie ma miejsca proces jest blokowany.
Ogólna struktura komunikatu wygląda następująco:

```
struct msgbuf{
     long mtype;        // typ komunikatu (>0)
     char mtext[1];     // treść komunikatu
}
```

1. Parametr `msgs` jest rozmiarem pola `mtext`,
2. Treść komunikatu może być dowolną strukturą
3. Pole `mtype` określa typ komunikatu, dzięki czemu możliwe jest przy odbiorze wybieranie z kolejki komunikatów określonego rodzaju. Typ komunikatu musi być wartością większą od 0.

Funkcja MSGRCV

```
PROTOTYPE: int msgrcv ( int msgid, struct msgbuf  *msgp, int
                        msgs, long msgtyp, int msgflg )
RETURNS:success : rozmiar odebranego komunikatu
           error: -1
            errno = analogicznie jak w funkcji msgsnd
```

PARAMETRY:

1. `msgid` - identyfikator kolejki
2. `msgp` – wskaźnik do obszaru pamięci zawierającego treść komunikatu
3. `msgs` - rozmiar właściwej treści komunikatu
4. `msgtyp` - typ komunikatu jaki ma być odebrany
   `msgtyp >0` wybierany jest komunikat którego typ jest dokładnie taki jak `msgtyp`
   `msgtyp <0` wybierany jest komunikat który ma najmniejszą wartość typu mniejszą lub równą bezwzględnej wartości z `msgtype`

msgtyp =0 typ komunikatu nie jest brany pod uwagę przy wyborze

5. msgflg – flagi specyfikujące zachowanie się funkcji w warunkach nietypowych
   Wartość ta może być ustawiona na 0 lub
   MSG_NOERROR, to komunikat przekraczający rozmiar bufora jest ucinany przy odbiorze, w przeciwnym razie odbierane są tylko komunikaty, których treść jest mniejsza od msgs
   IPC_NOWAIT, jeśli w kolejce nie ma komunikatów, i ustawiona jest w ten sposób flaga, to zwracana jest wartość –1, w przeciwnym wypadku proces jest blokowany, aż do czasu pojawienia się komunikatu

UWAGI:
Odebranie komunikatu oznacza pobranie go z kolejki i raz odebrany komunikat nie może zostać odebrany ponownie.

Funkcja MSGCTL

PROTOTYPE: int msgctl(int msgid, int cmd, struct msgid_ds. *buf);
RETURNS:success : 0
error: -1
errno = **EACCES** (nie ma praw do odczytu oraz cmd jest ustawiona na IPC_STAT)
EFAULT (adres wskazywany przez buf jest nieprawidłowy)
EIDRM (kolejka została usunięta)
EINVAL (msgqid nieprawidłowe, lub msgsz mniejsze od 0)
EPERM (komendy IPC_SET lub IPC_RMID zostały wydane podczas gdy process nie ma praw dostępu do zapisu)

PARAMETRY:
1. msgid - identyfikator kolejki
2. cmd – stała specyfikująca rodzaj operacji

cmd = IPC_STAT pozwala uzyskać informację o stanie kolejki komunikatów
cmd = IPC_SET pozwala zmienić związane z kolejką ograniczenia

cmd = IPC_RMID pozwala usunąć kolejkę z systemu
(`msgctl(qid, IPC_RMID,0)`

3. `buf` - wskaźnik na zmienną strukturalną przez którą przekazywane są parametry operacji

UWAGI:
Funkcja służy do zarządzania kolejką (np. usuwania kolejki, zmiany praw dostępu itp.)

Pamięć współdzielona

Pamięć współdzielona jest specjalnie utworzonym segmentem wirtualnej przestrzeni adresowej, do którego dostęp może mieć wiele procesów.
W zależności od praw dostępu procesy mogą odczytywać i/lub zapisywać wartości w pamięci współdzielonej, przy czym w przypadku współbieżnie działających procesów konieczne jest najczęściej synchronizowanie dostępu np. za pomocą semaforów .
Istnieje struktura opisująca obiekty pamięci współdzielonej o nazwie **shmid_ds.**

```
#include <sys/ipc.h>
#include <sys/shm.h>
```

Funkcja SHMGET

```
PROTOTYPE: int shmget ( key_t key, size_t size, int
                        shmflags );
RETURNS:success : identyfikator segmentu pamięci
                  współdzielonej
         error: -1
errno = EACCES (brak praw dostępu)
        ENOENT (segment pamięci nie istnieje)
        EIDRM  (segment pamięci został usunięty)
        EINVAL (nieprawidłowy rozmiar segmentu pamięci)
        ENOMEM (nie ma wystarczająco dużo miejsca by
                stworzyć segment pamięci współdzielonej)
        EEXIST (segment pamięci współdzielonej istnieje)
```

PARAMETRY:

1. `key` - wartość klucza, który identyfikuje segment pamięci współdzielonej (`ftok`, `IPC_PRIVATE`)
2. `size` – wielkość segmentu pamięci współdzielonej (w bajtach)
3. `shmflags` – prawa dostępu do pamięci współdzielonej (`IPC_CREAT`, `IPC_EXCL`)

UWAGI:
Funkcja tworzy segment pamięci współdzielonej

Funkcja SHMCTL

```
PROTOTYPE: int shmctl ( int shmid, int cmd, struct shmid_ds
                        *buf );
RETURNS:success : 0
         error: -1
```

errno = **EACCES** (nie ma praw do odczytu oraz cmd jest ustawiona na IPC_STAT)
EFAULT (adres wskazywany przez buf jest nieprawidłowy)
EIDRM (kolejka została usunięta)
EINVAL (msgqid nieprawidłowe, lub msgsz mniejsze od 0)
EPERM (komendy IPC_SET lub IPC_RMID zostały wydane podczas gdy process nie ma praw dostępu do zapisu)

PARAMETRY:

1. shmid - identyfikator pamięci współdzielonej
2. cmd – stała specyfikująca rodzaj operacji

cmd = IPC_STAT pozwala uzyskać informację o stanie pamięci współdzielonej
cmd = IPC_SET pozwala zmienić związane z kolejką ograniczenia
cmd = IPC_RMID pozwala usunąć pamięć współdzieloną z systemu (shmctl(shmid, IPC_RMID,0)

3. buf - wskaźnik na zmienną strukturalną przez którą przekazywane są parametry operacji

UWAGI:

Funkcja odpowiada funkcji msgctl. Przy próbie usunięcia segmentu odwzorowanego na przestrzeń adresową procesu system odpowiada komunikatem o błędzie.

Funkcja SHMAT

PROTOTYPE: char* shmat (int shmid, char* shmaddr, int shmflg);
RETURNS:success : adres, pod którym proces będzie „widział" przyłączony obszar pamięci.
error: -1

errno: **EACCES** (brak praw do odczytu)
EINVAL (msgqid nieprawidłowe, lub msgsz mniejsze od 0)

PARAMETRY:

1. shmid - identyfikator pamięci współdzielonej zwracany przez funkcję shmget

2. `shmaddr` – adres dla tworzonego segmentu pamięci współdzielonej lub wartość NULL, która powoduje, że segment dołączany jest w miejscu wybranym przez system (nie trzeba znać rozmieszczenia programu w pamięci)
3. `shmflg`
   `SHM_RDONLY` - segment jest przyłączany tylko dla odczytu
   `SHM_RND` - gdy jest ustawiony powoduje że przy wywołaniu funkcji adres `shmaddr` jest zaokrąglany w dół do granicy strony w pamięci, w przeciwnym razie pobierana jest wartość podana jako argument wejściowy.

`UWAGI:`
Zanim proces zacznie operować na pamięci współdzielonej musi przyłączyć jej segment. Zapisanie danych do pamięci współdzielonej odbywa się przez wykorzystanie adresu zwracanego przez funkcji `shmat`. W segmencie pamięci można umieścić każdy typ danych
oprócz wskaźników - przy dołączeniu do segmentu dwóch procesów z innymi adresami, wskaźnik wskazywać może różne obszary dla każdego z procesów.

Uwagi:
- ponieważ pamięć jest alokowana przy wywołaniu funkcji **shmat()** nie trzeba używać funkcji **malloc()** przy umieszczaniu danych w segmencie.

Funkcja SHMDT

```
PROTOTYPE: char* shmdt ( char* shmaddr );
RETURNS:success : 0
        error: -1
           errno = EINVAL (nieprawidłowy adres)
```

`PARAMETRY:`
1. `shmaddr` – adres stworzonego segmentu pamięci współdzielonej

`UWAGI:`
Odłączenie segmentu pamięci współdzielonej. Odłączenie to powinno nastąpić po zakończeniu pracy z danym segmentem. Po wywołaniu funkcji shmdt licznik dołączeń do segmentu jest zmniejszany o 1.

- Po skończeniu korzystania z segmentu powinien on być usunięty

Przykład usunięcia segmentu pamięci:

```
struct shmid_ds shm_desc;
shmctl(shm_id, IPC_RMID, shm_desc)
```

Semafory

- wykorzystanie semaforów zapobiega niedozwolonemu wykonaniu operacji na określonych danych jednocześnie przez większą liczbę procesów
- przez odpowiednie wykorzystywanie semaforów można zapobiec sytuacji w której wystąpi **zakleszczenie** (ang. ***deadlock*****)** lub **zagłodzenie** (ang. **starvation**)
- **Semafor binarny**
  - będący tylko w stanie podniesienia lub opuszczenia
  - reprezentowany przez liczbę binarną $S \in \{0, 1\}$
- **Semafor uogólniony**
  - możliwe wiele stanów
  - reprezentowany przez liczbę binarną $S \in \{0, 1, ..., \infty\}$
- Pojęcie semafora pierwszy raz zdefiniowane przez holendra **Edgara Dijkstrę** – [skróty: **v** dla operacji podniesienia semafora i **p** dla opuszczenia semafora]
- Praktyczna definicja semafora uogólnionego wg. Ben-Ariego:

Semafor jest pewną całkowitą liczbą nieujemną S.

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**

- Obszar programu składający się z instrukcji które może wykonywać tylko określona liczba procesów = **sekcja krytyczna**
- Operacje wykonywane na semaforze są **atomowe**.
- Semafor można traktować jako licznik, który jest zmniejszany („zamykany”) o 1 gdy jest zajmowany i zwiększany o 1 gdy jest zwalniany („podnoszony”)
- Semafor trzeba **zainicjować** wywołując operację podniesienia !!!
- Struktura opisująca obiekt będący semaforem: `semid_ds`
- W systemie Unix/Linux występują tzw. **zestawy semaforów**. Każdy semafor z tego zestawu posiada strukturę z nim związaną:

```
Struct sem{
   ushort semval  // wartość semafora
   pid_t sempid   // identyfikator procesu ostatnio wykonującego operację na semaforze
   ushort semncnt // liczba procesów, które czekają aż wartość semafora będzie zwiększona
   ushort semzcnt // liczba procesów które czekają aż semafor osiągnie wartość 0
}
```

Deklaracje funkcji znajdują się w plikach nagłówkowych :

```
#include <sys/ipc.h>
#include <sys/sem.h>
```

Funkcja SEMGET

`int semget (key_t key, int nsems, int semflags)`

- tworzenie semafora

- funkcja odpowiada funkcjom **`msgget()`**, **`semget()`**
- *key* – wartość klucza (**`ftok(), IPC_PRIVATE`**)
- *nsems* – liczba semaforów w zestawie (ponumerowanych od 0) //można utworzyć zestaw składający się z pojedynczego semafora
- *semflags* – podaje dodatkowe opcje dla tworzonego zestawu semaforów (**`IPC_CREAT, IPC_EXCL`**)
- funkcja zwraca identyfikator zestawu semaforów lub –1 w przypadku błędu

Funkcja SEMCTL

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

- funkcje kontrolujące pojedynczy semafor lub ich zestaw

- *semid* – identyfikator zestawu semaforów
- *semnum* – identyfikuje semafor
- *cmd* – podaje funkcję, może należeć do jednej z trzech grup :

  tradycyjne operacje IPC
  - **`IPC_STAT`** – zwraca wartości struktury *semid_ds* dla semafora (lub zestawu) o identyfikatorze *semid*, informacja jest umieszczana w strukturze wskazywanej przez 4 argument
  - **`IPC_SET`** – modyfikuje wartości struktury *semid_ds*
  - **`IPC_RMID`** – usuwa zestaw semaforów o identyfikatorze *semid* z systemu

  operacje na pojedynczym semaforze (dotyczą semafora określonego przez semnum)
  - **`GETVAL`** – zwraca wartość semafora (*semval*)
  - **`SETVAL`** – ustawia wartość semafora w strukturze
  - **`GETPID`** – zwraca wartość *sempid*
  - **`GETNCNT`** – zwraca *semcnt*
  - **`GETZCNT`** – zwraca *semzcnt*

  operacje na wszystkich semaforach:
  - **`GETALL`** – umieszcza wszystkie wartości semaforów w tablicy podanej jako 4 argument
  - **`SETALL`** – ustawia wszystkie wartości zgodnie z zawartością tablicy podanej jako 4 argument

- ostatni (4) argument jest unią.

```
union semun (
      int val;
      struct semid_ds *buf;
      unsigned short *array
)
```

.

Zastosowanie:

- ustawienie początkowej wartości semafora
- zbadanie zmiany właściciela semafora, praw dostępu do niego, czasu ostatniej zmiany , ilości procesów oczekujących na semafor i identyfikatora procesu ostatnio zmieniającego wartość semafora, itd.

Funkcja SEMOP

`int semop (int semid, stuct sembuf *sops, unsigned nsops)`

-zajmowanie i zwalnianie semafora

- zwraca 0 przy poprawnym wykonaniu, -1 w przypadku niepowodzenia
- *semid* - identyfikator zestawu semaforów otrzymany z funkcji **`semget()`**
- *sops* – tablica struktur `sembuf`

```
struct sembuf{
   short sem_num     //index semafora w zestawie ( numeracja od 0 )
   short sem_op      //operacja
   short sem_flg     //opcje operacji
}
```

- *nsops* – liczba struktur sembuf w tablicy
- jeśli jedna z operacji nie może być wykonana, żadna nie będzie wykonana. (wszystko albo nic)
- *sem_flg*:
  - IPC-NOWAIT – jeśli operacja na semaforze nie może być wykonana to przy ustawionej fladze następuje powrót do procesu, nie jest on blokowany
  - SEM_UNDO – pozwala na operację wycofania, gdy proces się zakończy. Dla każdego procesu utrzymywana jest wartość semadj, do tej wartości jest dodawane sem_op
- *sem_op* może przyjmować wartości:
  - >0 – zwiększenie wartości semafora – zwolnienie zasobu
  - <0 – zmniejszenie semafora – próba zajęcia lub otrzymanie zasobu
  - =0 – sprawdzenie, czy wartość semafora wynosi 0