Mechanizmy komunikacji międzyprocesowej IPC *(Inter Process Communication)*

- Semafory
- Pamięć dzielona
- Komunikaty

ipcs [-q | -m | -s]
ipcrm [-q | -m | -s] id
ipcrm [shm | msg | sem] id
ipcrm -a

Pamięć dzielona

- Dostęp do tej samej logicznej pamięci niespokrewnionym procesom
- Zakres adresów tworzony przez IPC w przestrzeni adresowej jednego procesu
- Inne procesy mogą dołączyć segment pamięci dzielonej do swojej przestrzeni adresowej (jak malloc)
- Brak mechanizmów synchronizacji (zadanie programisty)
- Wydajny dostęp do dużych obszarów pamięci

Obsługa pamięci dzielonej

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

#include <sys/ipc.h>

#include <sys/types.h>

void *shmat(int shm_id, const void *shm_addr, int shmflg);

int shmctl(int shm_id, int cmd, struct shmid_ds*buf);

int shmdt(const void *shm_addr);

int shmget(key_t key, size_t size, int shmflg);
```

Tworzenie pamięci dzielonej

int **shmget**(key_t key, size_t size, int shmflg);

- tworzy segment współdzielonej pamięci o rozmiarze size
- identyfikowany przez klucz key
- lub znajduje segment o takim kluczu, jeśli istnieje
- zwraca
  - identyfikator, który służy do odwoływania się do segmentu w pozostałych funkcjach operujących na pamięci dzielonej
  - -1 - błąd
- shmflg umożliwia przekazanie praw dostępu do pamięci oraz pewnych dodatkowych flag definiujących sposób jej tworzenia (np. IPC_CREAT, IPC_EXCL), połączonych z prawami dostępu operatorem sumy bitowej (np. IPC_CREAT|0660). /x-ignorowane/

dołączenie/odłączenie segmentu pamięci dzielonej do przestrzeni adresowej procesu

void ***shmat**(int shm_id, const void *shm_addr, int shmflg);

- przydziela segmentowi dzielonej pamięci, identyfikowanemu przez shm_id (shmget), adres w przestrzeni adresowej procesu i zwraca ten adres (shm_addr) /NULL/
- shmflg – znaczniki bitowe:
  - SHM_RND
  - SHM_RDONLY

int **shmdt**(const void *shm_addr);

- odłącza segment pamięci dzielonej, umieszczony pod adresem shm_addr.

operacje kontrolne na segmencie pamięci dzielonej

int **shmctl**(int shm_id, int cmd, struct shmid_ds *buf);

- umożliwia wykonywanie operacji kontrolnych (cmd) na segmencie pamięci dzielonej

cmd:

IPC_STAT – ustawia dane w shmid_ds

IPC_SET - ustawia wartości zw. z pam. dziel. na dane ze struktury shmid_ds

IPC_RMID - usuwa segment pamięci dzielonej

struktura **shmid_ds** – tryby i zezwolenia

SEMAFORY

- uogólnienie klasycznych semaforów Dijkstry
- zestaw operacji na semaforach jest obszerny (zestawy semaforów)
- semafory należące do danego zestawu - ponumerowane
- możliwe jest wykonywanie jednoczesnych operacji na semaforach należących do tego samego zestawu.

pliki nagłówkowe:

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

Podstawowe funkcje do obsługi semaforów

- **semget** - utworzenie zestawu semaforów (uzyskanie dostępu do istniejącego zestawu semaforów)
- **semop** - operacje na semaforach
- **semctl** - operacje administracyjne na zestawie semaforów

Utworzenie (uzyskanie dostępu do) zestawu semaforów

Klucz → identyfikator semafora (id)

semget - klucz; semop,semctl - id

funkcja systemowa semget:

int **semget** (key_t klucz, int lsem, int flagi)

- pobiera klucz i przekazuje identyfikator zestawu semaforów (-1 w przypadku błędu)
- klucz pełni role 'nazwy' zestawu semaforów. Procesy, które chcą korzystać z tego samego zestawu semaforów musza go utworzyć/otworzyć podając ten sam klucz
- lsem określa liczbę semaforów w danym zestawie. Otwierając istniejący zestaw semaforów można podać dowolna liczbę nie większa od faktycznej liczby semaforów w danym zestawie (w szczególności zero).

- parametr flagi służy do określenia: (jak open)
  - praw dostępu do zestawu semaforów
  - sposobu otwarcia zestawu semaforów:

    IPC_CREAT - polecenie utworzenia zestawu semaforów

    IPC_CREAT | IPC_EXCL - utworzenie nieistniejącego zestawu

    (porażka jeśli zestaw o danym kluczu już istnieje)

Operacje na zestawie semaforów

int **semop** (int ident, struct sembuf *operacje, unsigned loper)

- ident - identyfikator zestawu semaforów (przekazany przez semget)
- loper - liczba operacji do jednoczesnego wykonania na semaforach należących do danego zestawu a równocześnie liczba elementów tablicy 'operacje', definiujących operacje na poszczególnych semaforach zestawu

Struktura 'sembuf' (zdefiniowana w pliku 'sys/sem.h') jest następująca:

```
struct sembuf {
  short int sem_num;          /* semaphore number */
  short int sem_op;           /* semaphore operation */
  short int sem_flg;          /* operation flag */
};
```

Opis jednej operacji:
* numer semafora w zestawie (**sem_num**)
* rodzaj operacji (**sem_op**), będący liczbą całkowitą, mający następujące znaczenie:

- wartość dodatnia: zwiększenie wartości semafora (ewentualne uwolnienie czekających procesów),(V)
- wartość ujemna: zadanie zmniejszenia wartości semafora,(P)
- zero: sprawdzenie i ewentualne oczekiwanie aż wartością semafora będzie zero;

* opcje (**sem_flg**), na przykład:

- IPC_NOWAIT - operacja nieblokująca, tzn. jeśli operacja nie może być wykonana natychmiast, to semop natychmiast się kończy (z wartością -1), a zmiennej 'errno' zostaje nadana wartość EAGAIN,
- SEM_UNDO - jeśli operacja zmienia wartość semafora, zmiana ta zostanie zniwelowana w momencie zakończenia wykonywania procesu

Wartością funkcji semop jest zero, jeśli jej wykonanie zakończyło się sukcesem, a -1 w przypadku błędu.

Operacje administracyjne na zestawie semaforów

Do wykonywania dodatkowych operacji na zestawie semaforów (pojedynczych semaforach zestawu) służy funkcja systemowa **semctl**

int **semctl** (int ident, int semnum, int polec, union semun arg);

Znaczenie parametrow:

* ident - identyfikator zestawu semaforów (przekazany przez semget)
* semnum - numer semafora, na którym ma być wykonana operacja
* arg - parametry polecenia lub wartości przekazywane przez **semctl**

(unia 'semun' jest zdefiniowana w pliku 'semun.h')

Operacje administracyjne na zestawie semaforów c.d.

* polecenie -

GETVAL - pobranie wartość semafora

SETVAL - nadanie wartości semaforowi (inicjacja, ale nie tylko)

GETPID - przekazanie identyfikatora procesu, który jako ostatni wykonał 'semop' na semaforze 'semnum'

GETNCNT - przekazanie liczby procesów czekających na zwiększenie wartości semafora 'semnum'

GETZCNT - przekazanie liczby procesów czekających na uzyskanie przez semafor 'semnum' zera

Operacje administracyjne na zestawie semaforów c.d.

GETALL - pobranie wartości wszystkich semaforów w zestawie

SETALL - nadanie wartości wszystkim semaforom w zestawie

IPC_STAT - pobranie informacji o zestawie semaforów (m.in. identyfikator procesu, który ostatni wykona operacje na wskazanym semaforze, prawa dostępu, liczba semaforów, czas ostatniej operacji/zmiany)

IPC_SET - zmiana niektórych atrybutów zestawu semaforów (identyfikator właściciela/grupy, prawa dostępu)

IPC_RMID - usuniecie zestawu semaforów (obudzenie wszystkich czekających procesów - zmienna 'errno' ustawiana na EIDRM).

Operacje administracyjne na zestawie semaforów c.d.

Wartością funkcji jest przekazywana wartość (polecenia GETVAL, GETPID, GETNCNT, GETZCNT) lub -1 w przypadku błędu, a wówczas zmienna 'errno‘ określa rodzaj błędu. Podstawowe kody błędów:

EACCES - proces nie ma prawa wykonać podanej operacji

EINVAL - niepoprawna wartość (rodzaj polecenia, identyfikator zestawu).

KOLEJKI KOMUNIKATÓW

- Analogia komunikatów do nazwanych potoków FIFO
- Umożliwiają przesyłanie bloków danych pomiędzy procesami
- Limity –
  - rozmiar pojedynczych bloków danych **MSGMAX**(4096),
  - całkowity rozmiar **MSGMNB**(16384)

- Wszystkie kolejki - pamiętane w jądrze systemu; mają przypisane identyfikatory
- Procesy mogą czytać i zapisywać komunikaty do różnych kolejek
- komunikat ma następującą strukturę:
  - **typ komunikatu (liczba całkowita)**
  - **dane**
- W strukturze komunikatu nie ma „adresata” komunikatu. Komunikujące się ze sobą procesy powinny korzystać z tych samych kolejek oraz mieć „uzgodnione” znaczenie poszczególnych typów komunikatów.

- Proces może umieścić komunikat w kolejce niezależnie od tego czy istnieje inny proces oczekujący na ten komunikat.
- Każdy komunikat jest przechowywany w kolejce aż do momentu kiedy jakiś proces go odczyta (odczyt komunikatu powoduje usunięcie go z kolejki) lub do momentu usunięcia kolejki z systemu.
- Komunikaty są przechowywane w kolejce również gdy proces, który je wysłał zakończył się.

OGLĄDANIE / KASOWANIE KOLEJEK KOMUNIKATÓW

- wyświetlenie listy kolejek komunikatów aktualnie istniejących w systemie : **`ipcs -q`**
  - pełny opis kolejek - opcja **`-a`** – wyświetla m.in. :
    - `ID` – unikalny identyfikator kolejki
    - `KEY` – klucz na podstawie którego utworzono kolejkę
    - `MODE` – prawa dostępu

- usunięcie kolejki komunikatu:

`ipcrm msg ID`

  - gdzie `ID` jest identyfikatorem usuwanej kolejki komunikatów

TWORZENIE KOLEJKI KOMUNIKATÓW

```
#include <sys/msg.h>
int msgget(key_t key, int msgflg);
```

Tworzy kolejkę komunikatów i umożliwia dostęp do niej

Zwraca

- identyfikator kolejki komunikatów, tworzony na podstawie podanego klucza
- `-1 – błąd`

```
key – (nazwa kolejki) klucz  np. wygenerowany
    przez ftok
    IPC_PIRVATE – kolejka dostępna jedynie dla
    bieżącego procesu
msgflg  - znaczniki zezwoleń (jak semget)
```

IPC_CREAT ; IPC_CREAT | IPC_EXCL

generacja klucza

```
#include <sys/types.h>
#include <sys/ipc.h>
key_t  ftok(const char *path, int id);
```

Zwraca:

- wygenerowany na podstawie drugiego parametru klucz
- - 1 - błąd

Reprezentacja kolejki komunikatów

```
struct msqid_ds {
struct ipc_perm msg_perm; /* struktura praw dostępu */
struct msg *msg_first; /* wskaźnik do pierwszego
  komunikatu w kolejce */
struct msg *msg_last; /* wskaźnik do ostatniego
  komunikatu w kolejce */
```

…liczba bajtów w kolejce,
liczba komunikatów w kolejce,
max. liczba bajtów w kolejce,
PID procesu, który ostatnio wysłał komunikat,
PID procesu, który ostatnio odczytał komunikat,
czasy…

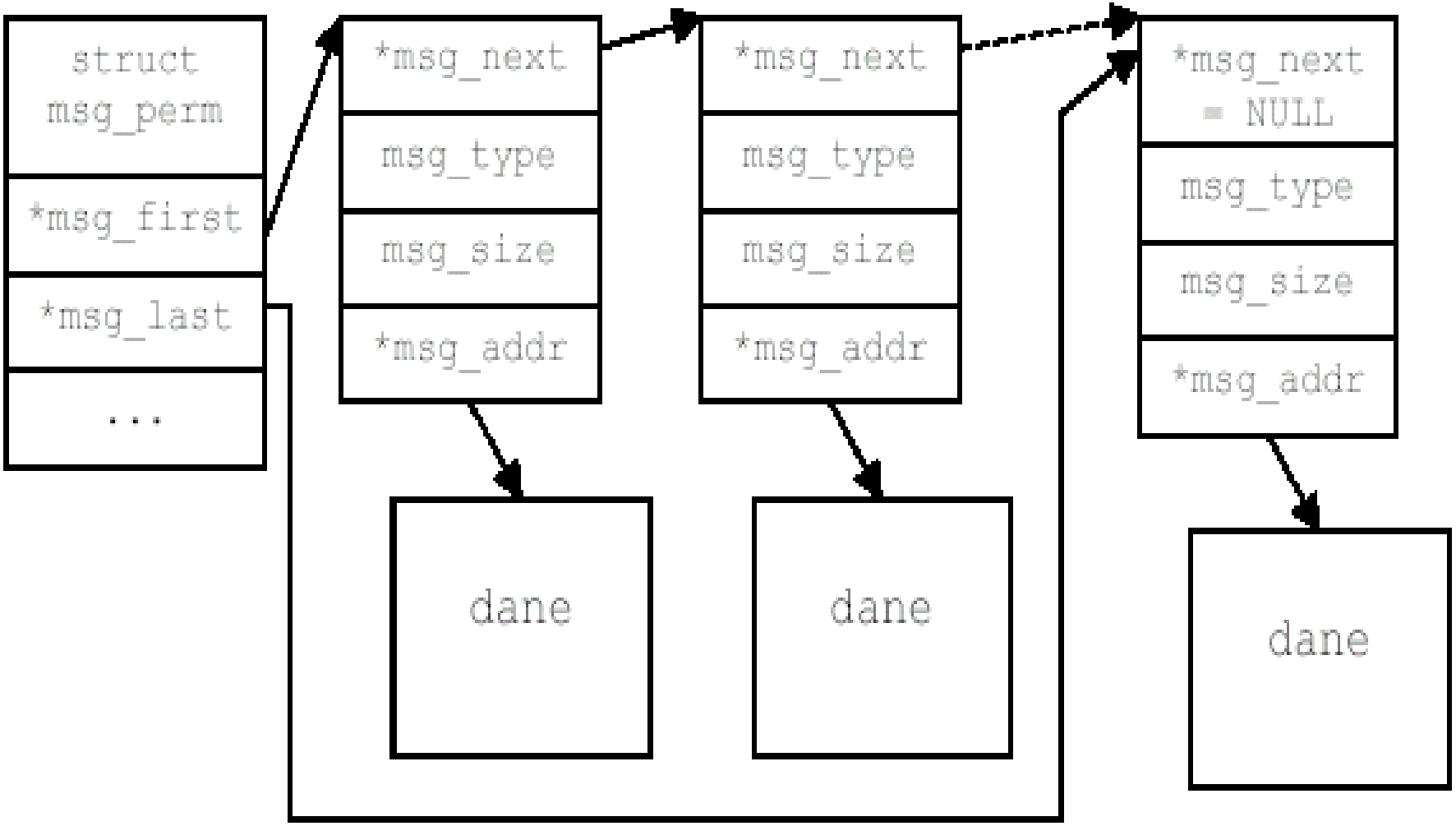
}

Reprezentacja komunikatów

Powiązane w listę struktury msg

```
struct msg {
struct msg *msg_next;        /* wsk. do nast. kom.*/
long msg_type;               /*typ komunikatu*/
size_t msg_size;             /*rozmiar treści kom.*/
void *msg_addr;              /*wsk. do bloku zawier. treść kom*/
};
```

STRUKTURA KOLEJKI KOMUNIKATÓW W JĄDRZE



WYSŁANIE KOMUNIKATU

```
#include <sys/msg.h>
int msgsnd(int msqid, const void *msg_ptr, size_t msgsz, int
   msgflg);
```

dodanie komunikatu do kolejki – zwraca 0 lub -1
argumenty:

- msqid - identyfikator kolejki komunikatów, do której wysyłany jest komunikat
- msg_ptr - adres, pod którym znajduje się struktura (reprezentująca cały komunikat)
- mgsz – rozmiar w bajtach (typ nie zalicza się do rozmiaru)
- msgflg – reakcja na przepełnienie kolejki komunikatów:
  - IPC_NOWAIT – przy braku możliwości wysłania komunikatu fcja zwraca -1
  - IPC_WAIT - -”- -„- -„- proces się zawiesza w oczekiwaniu na zwolnienie miejsca

struktura komunikatu z pliku m sys/msg.h (może być prawie dowoln
uzgodniona)

```
struct msgbuf {
    long mtype;  /* message type */
    char mtext[1];    /* message text */
  };
```

ODBIERANIE KOMUNIKATU

```
#include <sys/msg.h>
ssize_t msgrcv(int msqid, void *msg_ptr, size_t msgsz,
   long int msgtyp, int msgflg);
```

Funkcja `msgrcv` pobiera komunikat z kolejki. Zwraca:

- liczbę bajtów umieszczonych w buforze (msg_ptr)
- -1 - błąd

Argumenty:

- msgid - identyfikator kolejki komunikatów
- `*msg_ptr` – adres, pod którym znajduje się struktura reprezentująca cały komunikat do której ma zostać przekopiowany komunikat (Format wskazywanej struktury powinien być zgodny z formatem przesyłanych komunikatów.)
- msgsz - rozmiar porcji danych zapisywanych w strukturze wskazywanej przez `*msgp ( z wyłączeniem typu`
- msgtyp – mechanizm pierwszeństwa:
  - 0 - zostanie odebrany pierwszy komunikat z kolejki
  - >0 –zostanie odebrany pierwszy dostępny komunikat tego samego typu jak msgtyp
  - <0 - -"- -"- -"- typu ≤ | msgtyp |
- msgflg – jak w msgsnd

OPERACJE KONTROLNE

```
#include <sys/msg.h>
int msgctl(int msqid, int cmd, struct msqid_ds
   *buf);
```

Zwraca: 0 lub -1

Argumenty:

- `msqid` – identyfikator kolejki komunikatów
- cmd – operacja, która ma być wykonana
- *buf - wskazuje na zmienną typu struktura `msqid_ds`.

Funkcja `msgctl` umożliwia realizację operacji kontrolnych na kolejkach komunikatów **cmd**=

- IPC_STAT - pobiera informacje o kolejce komunikatów i wpisuje je do struktury wskazywanej przez argument `*buf`
- IPC_SET - ustawia informacje o kolejce komunikatów (z *buf)
- IPC_RMID – usuwa kolejkę komunikatów

Przykładowa aplikacja klient-serwer