Programowanie Współbieżne

Komunikacja między procesowa IPC

IPC

W systemie V są 3 rodzaje komunikacji między procesowej.

- kolejki komunikatów
- semafory
- pamięć wspólna

IPC

	Kolejka komunikatów	Semafor	Pamięć wspólna
plik nagłówkowy	<sys/msg.h>	<sys/sem.h>	<sys/shm.h>
funkcja systemowa tworzenia lub otwierania	msgget	semget	shmget
funkcja systemowa operacji sterujących	msgctl	semctl	shmctl
funkcje systemowe przesyłania	msgsnd msgrcv	semop	shmat shmdt

IPC

Jądro trzyma informacje na temat każdego kanału komunikacji międzyprocesowej w strukturze:

```
#include <sys/ipc.h>
struct ipc_perm
{
ushort uid; /* id użytkownika dla właściciela */
ushort gid; /* id grupy dla właściciela */
ushort cuid; /* id użytkownika dla twórcy */
ushort cgid; /* id grupy dla twórcy */
ushort mode; /* tryb dostępu */
ushort seq; /* numer koleiny */
key_t key; /* klucz */
}
```

IPC

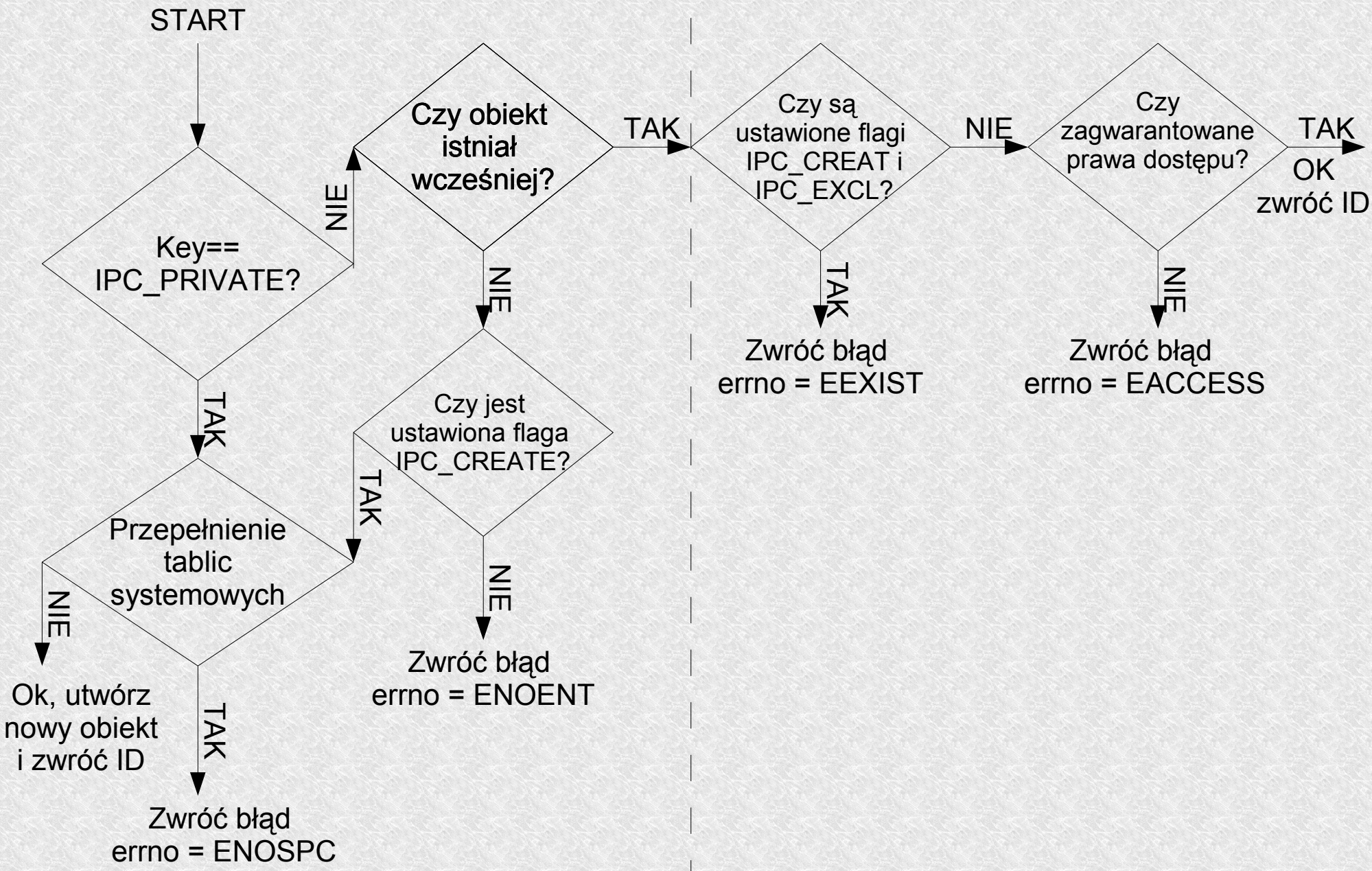
- Do pobierania wartości tej struktury służą funkcje **`msgctl`**, **`semctl`** oraz **`shmctl`**

- Wszystkie trzy funkcje do tworzenia kanału komunikacji IPC **`msgget`**, **`semget`** oraz **`shmget`** pobierają klucz key_t a zwracają identyfikator kanału.

IPC

Wszystkie też mają argument flag, którego

- 9 najmniej znaczących bitów określa tryb dostępu do kanału
- pozostałe określają czy ma być utworzony kanał czy nie itp.
  - jeżeli argument key ma wartość **IPC_PRIVATE** to będzie utworzony prywatny kanał komunikacji międzyprocesowej.
  - Nie istnieje taka kombinacja ścieżki i identyfikatora by ftok wygenerowało **IPC_PRIVATE**
  - **IPC_CREAT** w argumencie flag spowoduje utworzenie nowego elementu w tablicy systemowej w jądrze jeżeli jest brak, jeżeli już jest to będzie przekazany on jako wartość zwrotna.
  - jeżeli flag ma **IPC_CREAT + IPC_EXCL** uda się utworzenie kanału tylko wtedy gdy go nie było jeżeli był to funkcja zwróci błąd.
  - **IPC_EXCL** w tym przypadku jest tylko do tworzenia przy korzystaniu już nie obowiązuje i inni mogą używać zgodnie z prawami dostępu.
  - **IPC_EXCL** bez **IPC_CREAT** nie ma znaczenia


START
Key== IPC_PRIVATE?
NIE
TAK
Czy obiekt istniał wcześniej?
TAK
Czy są ustawione flagi IPC_CREAT i IPC_EXCL?
NIE
Czy zagwarantowane prawa dostępu?
TAK
OK zwróć ID
TAK
Zwróć błąd errno = EEXIST
NIE
Zwróć błąd errno = EACCESS
NIE
Czy jest ustawiona flaga IPC_CREATE?
TAK
NIE
Zwróć błąd errno = ENOENT
Przepełnienie tablic systemowych
NIE
Ok, utwórz nowy obiekt i zwróć ID
TAK
Zwróć błąd errno = ENOSPC
Tworzymy nowy obiekt
Korzystamy z istniejącego obiektu

IPC

- **podczas tworzenia** 9 najmłodszych bitów argumentu flag inicjuje pole *ipc_perm*
- cuid, cgid, uid i gid przejmują obowiązujące identyfikatory użytkownika i grupy dla procesu wywołującego funkcję.
- identyfikatory twórcy kanału nigdy się nie zmieniają (cuid i cgid)
- identyfikatory właściciela mogą ulec zmianie przez odpowiednie zawołanie funkcji *msgctl*, *semctl* lub *shmctl*
- *ctl służą także do zmiany praw dostępu
- każda operacja na kanałach IPC (zapis, odczyt) powoduje sprawdzanie podobne do sprawdzania praw dostępu w systemie plików

IPC

- sprawdzanie odbywa się na podstawie pola ***ipc_perm***
  - Proces nadzorcy ma zawsze przyznane prawa dostępu
  - jeżeli *uid* lub *cuid* oraz odpowiedni bit prawa dostępu się zgadza, to proces ma przyznane prawo dostępu.
  - jeżeli *gid* lub *cgid* się zgadzają oraz odpowiedni bit prawa dostępu, to proces ma przyznane to prawo dostępu
  - jeżeli powyższe się nie powiodą do odpowiedni bit prawa dostępu dla innych musi być ustawiony

Kolejki Komunikatów IPC

- wszystkie komunikaty są pamiętane w jądrze systemu i mają przypisany identyfikator kolejki komunikatów (*message queue identyfier*)
- msqid – identyfikuje konkretną kolejkę komunikatów
- procesy czytają i piszą do dowolnych kolejek komunikatów
- nie wymaga się by jeden proces czekał już na pojawienie się komunikatu zanim drugi zacznie pisać
- proces może umieścić komunikat w kolejce i zakończyć swoje działanie.
- każdy komunikat w kolejce ma następujące atrybuty
  - **typ** – liczba całkowita typu *long*
  - **długość** – porcji danych, może być 0
  - **dane** – jeżeli długość jest większa niż 0

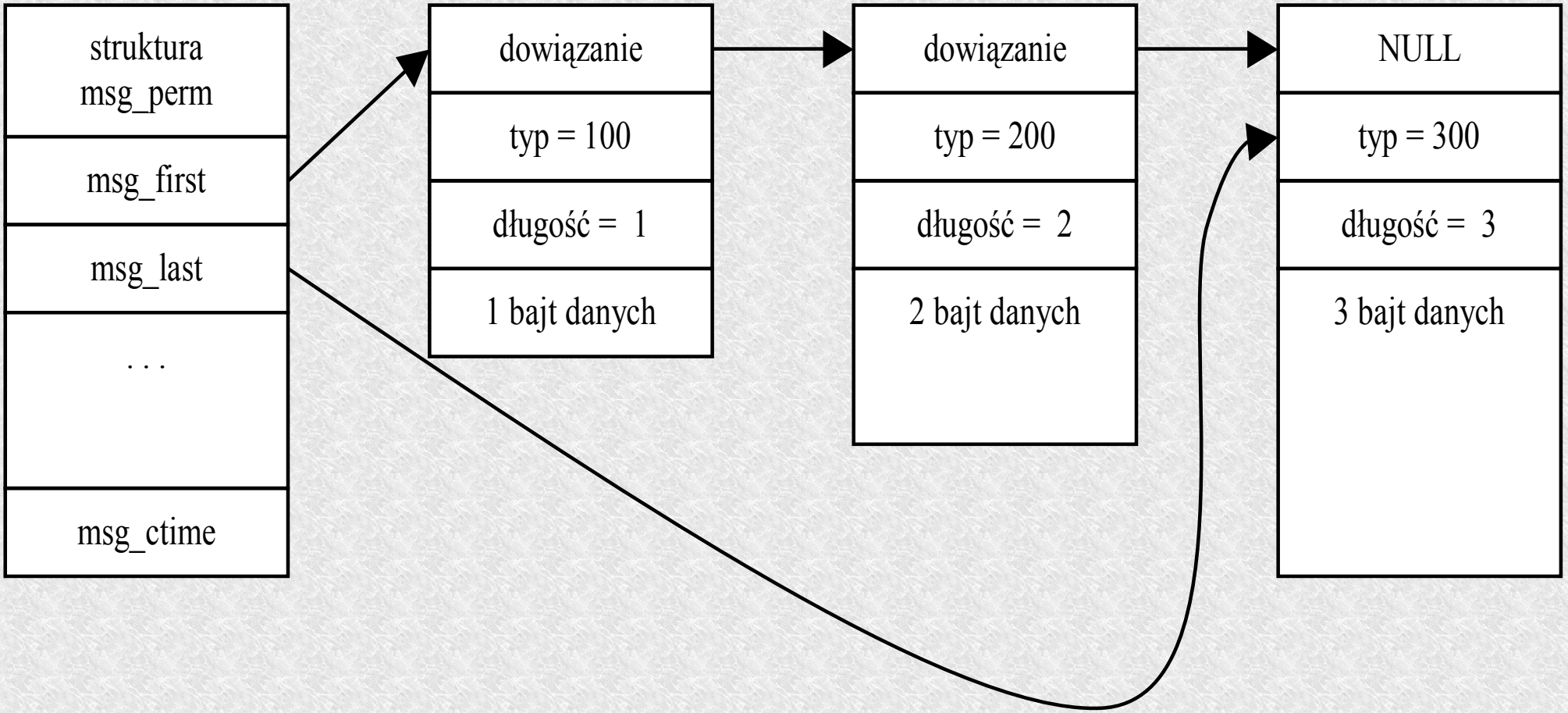
Kolejki Komunikatów IPC

Dla kolejki komunikatów jądro utrzymuje taką strukturę

```
#include <sys/types.h>
#include <sys/ipc.h>
struct msqid_ds.
{
struct ipc_perms msg_perms; /* struktura praw dostępu dla
operacji */
struct msg *msg_first; /* wskaźnik do 1 komunikatu w kolejce */
struct msg *msg_last; /* wskaźnik do ostatniego komunikatu w
kolejce */
ushort msg cbytes; /*bieżąca liczba bajtów w kolejce */
ushort msg_gnum; /* bieżąca liczba komunikatów w kolejce */
ushort msg_gbytes; /* maksymalna liczba bajtów dla kolejki */
ushort msg_lspid; /* identyfikator procesu który ostatnio pisał
coś do kolejki */
ushort msg_lrpid; /* identyfikator procesu, który ostatnio
wywołał msgrcv */
time_t msg_strime; /* czas ostatniego wywołania funkcji msgsnd */
time_t msg_rtime; /* czas ostatniego wywołania funkcji msgrcv */
time_t msg_ctime /* czas ostatniego wywołania funkcji msgctl
która zmieniła wartość powyższych pól */
};
```

Kolejki Komunikatów IPC

Przypuśćmy, że w kolejce mamy trzy komunikaty o długości 1,2,3 bajty typach 100,200,300 i w tej tez kolejności nadeszły to:



Kolejki Komunikatów IPC

Do tworzenia służy *msgget*

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

   int msgget (key_t key, int msgflag);
```

funkcja zwróci id kolejki lub –1 gdy się nie powiedzie

Kolejki Komunikatów IPC

msgflag określa prawa dostępu:

Wartość liczbowa	stała symboliczna	znaczenie
0400	MSG_R	czytanie dla właściciela
0200	MSG_W	pisanie dla właściciela
0040	MSG_R >> 3	czytanie dla grupy
0020	MSG_W >> 3	pisanie dla grupy
0004	MSG_R >> 6	czytanie dla innych
0002	MSG_W >> 6	pisanie dla innych
1000	IPC_CREAT	
2000	IPC_EXCL	
4000	IPC_NOWAIT	

Kolejki Komunikatów IPC

Do umieszczenia komunikatu w kolejce służy:

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

int msgsnd(int msqid, struct msgbuf *ptr, int
length, int flag);
```

- lenhgth określa długość komunikatu w bajtach jest to liczba bajtów danych zdefiniowanych przez użytkownika, czyli te po polu long, może być też 0.
- flag może przyjąć stałą symboliczną IPC_NOWAIT lub 0.
- jeżeli będzie IPC_NOWAIT to gdy kolejka jest pełna lub jest zbyt wiele komunikatów w całym systemie to funkcja zwróci natychmiast –1 a errno = EAGAIN.
- gdy *msgsnd* zakończymy sukcesem dostaniemy zwrotne 0.

Kolejki Komunikatów IPC

- ptr jest wskaźnikiem do struktury o następującym wzorcu

```
struct msgbuf
{
long mtype; /* typ komunikatu */
char *mtext; /* dane komunikatu */
}
```

  - mtext jest mylący bo mogą tam być różne dane
  - przez wzorzec rozumiemy że ptr musi wskazywać na liczbę całkowitą typu long która zawiera typ komunikatu i poprzedza sam komunikat (o ile długość > 0)
  - jądro nie interpretuje treści komunikatu
  - można zdefiniować własną strukturę np.

```
typedef struct my_msgbuf {
   long mtype;
   short mshort; /*dowolność */
   char mchar[8]; /* dowolność */
} Message;
```

Kolejki Komunikatów IPC

Do odbioru służy

```
int msgrcv(int msqid, struct msgbuf *ptr, int length, long
msgtype, int flag);
```

- ptr wskazuje gdzie ma być zapisany komunikat
- length – rozmiar danych ptr'a
- msgtype – określa pożądany przez odbiorcę komunikat
  - 0 oznacza pierwszy który wszedł do kolejki (zasada FIFO)
  - >0 pierwszy komunikat którego typ jest taki sam
  - <0 pierwszy komunikat z najmniejszych typów które nie są większe niż wartość bezwzględna msgtype.

Kolejki Komunikatów IPC

Dla przykładowej kolejki z typami 100L, 200L i 300L mamy

Argument msgtype	Typ przekazanego komunikatu
0L	100L
100L	100L
200L	200L
300L	300L
-100L	100L
-200L	100L
-300L	100L

Kolejki Komunikatów IPC

- flag – określa co należy zrobić gdy w kolejce brak komunikatów
  - gdy ustawiony IPC_NOWAIT wtedy natychmiast wracamy, funkcja zwraca –1 i errno = ENOMSG
  - jeżeli nie ma IPC_NOWAIT to proces czeka aż nastąpi jedno z 3 zdarzeń
    - będzie można otrzymać komunikat żądanego typu
    - kolejka komunikatów zostanie usunięta z systemu.
    - proces przechwyci odpowiedni sygnał
  - jeżeli *flag* ma MSG_NOERROR to jeśli otrzymamy więcej danych niż wskazaliśmy przez length to nadmiar będzie pominięty a funkcja nie wykaże błędu

Kolejki Komunikatów IPC

Do sterowania kolejką komunikatów służy

```
int msgctl(int msqid, int cmd, struct msqid_ds.
*buff);
```

gdy cmd jest:

IPC_RMID – 0 usunięcie kolejki
IPC_SET – 1 ustawia na podstawie buff
IPC_STAT – 2 pobiera pobiera do buff

Kolejki Komunikatów IPC

Klient – serwer.

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

extern int errno;
#define MKEY1 1234L
#define MKEY2 2345L
#define PERMS 0666
```

Kolejki Komunikatów IPC

```
#include „msgq.h”

main() //dla serwera
{
int readid, writeid;
if ((readid = msgget(MKEY1,PERMS|IPC_CEAT))<0)
   perror(„serwer: nie mogę otworzyć kolejki 1”);
if ((writeid = msgget(MKEY2, PERMS | IPC_CREAT)) <0)
   perror(„serwer: nie mogę otworzyć kolejki 2”);
serwer(readid, writeid);
exit(0);
}
```

Kolejki Komunikatów IPC

```
#include „msgq.h”

main()//dla klienta
{
int readid, writeid;
if ((writeid = msgget(MKEY1,0))<0)
   perror(„klient: nie mogę otworzyć kolejki 1”);
if ((readid = msgget(MKEY2,0)) < 0)
   perror(„klient: nie mogę otworzyć kolejki 2”);
client(readid, writeid);
/* usuwamy kolejki */
if (msgctl(readid, IPC_RMID,(struct msqid_ds*)0) <0)
   perror(„klient: nie mogę usunąć kolejki 1”);
if (msgctl(writeid,IPC_RMID, NULL) < 0)
   perror(„klient: nie mogę usunąć kolejki 2”);
exit(0);
}
```

Kolejki Komunikatów IPC

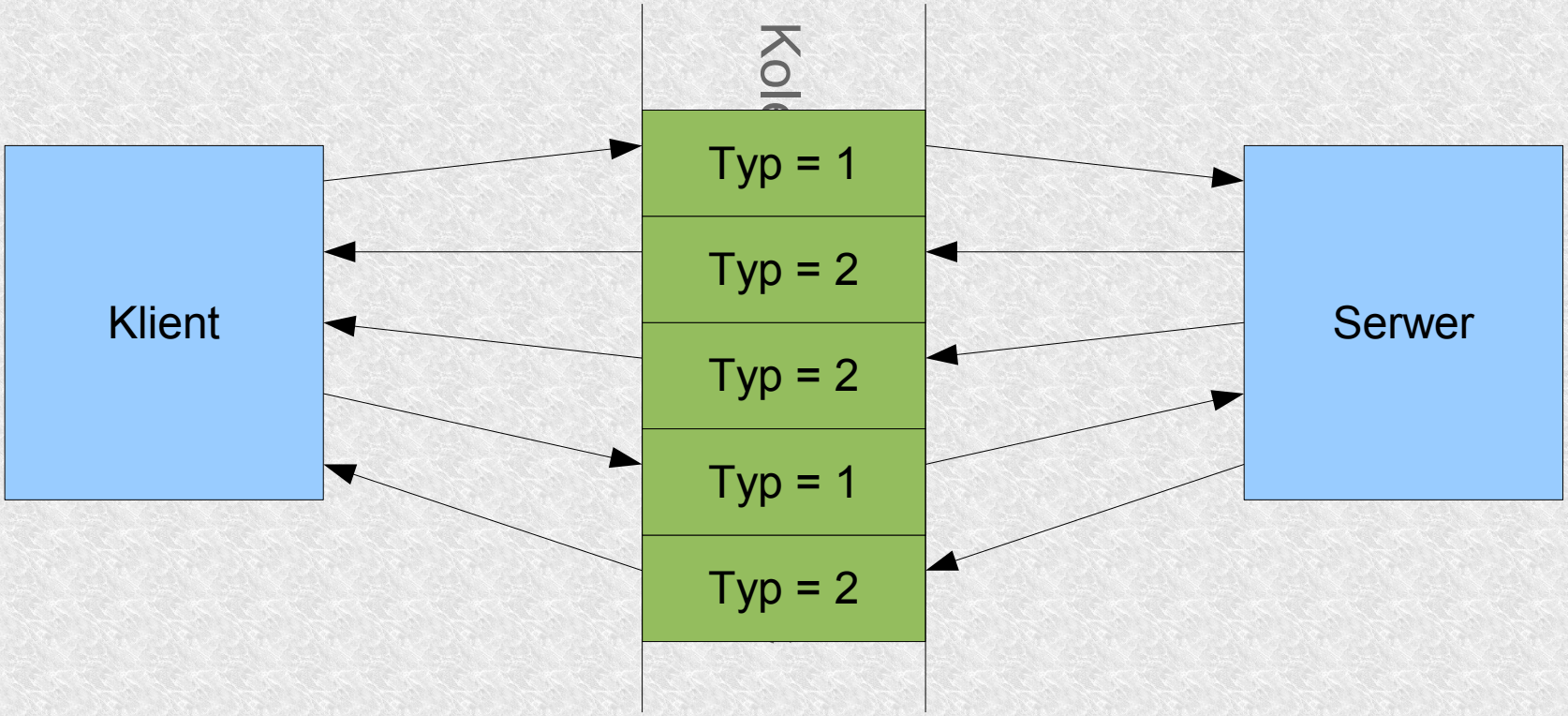
```
#include „mesgq.h”
void mesg_send(int id, Mesg *mesgptr)
{
if (msgsnd(id, (char*)&(mesgptr->mesg_type),
   mesgptr->mesg_len,0) !=0)
      perror(„błąd wysyłania komunikatu”);
}

int mesg_recv(int id, Mesg *mesgptr)
{int n;
n = msgrcv(id, (char*) &mesgptr->mesg_type),
   MAXMESGDATA, mesgptr->mesg_type, 0);
if ((mesgptr->mesg_len = n) < 0)
   perror(„blad odbioru”);
return(n); /*jeśli koniec pliku to n=0*/
}
```

Kolejki Komunikatów IPC

Stosując typy komunikatów można osiągnąć komunikacje dwustronną

- 2 typ oznacza komunikaty od serwera do klienta
- 1 typ oznacza komunikaty od klienta do serwera



Kolejki Komunikatów IPC

Stosując typy komunikatów można osiągnąć komunikacje dwustronną dla serwera i wielu klientów.

- Klient wysyłając komunikat ustawia typ = 1 a w treści zamieszcza swój pid
- Serwer odbiera wszystkie komunikaty gdzie typ = 1
- Odczytuje z treści pid nadawcy
- Odpowiada wkładając do kolejki komunikat gdzie typ = pid
- Klient odczytuje wszystkie komunikaty o typie takim jak jego pid.

Kolejki Komunikatów IPC