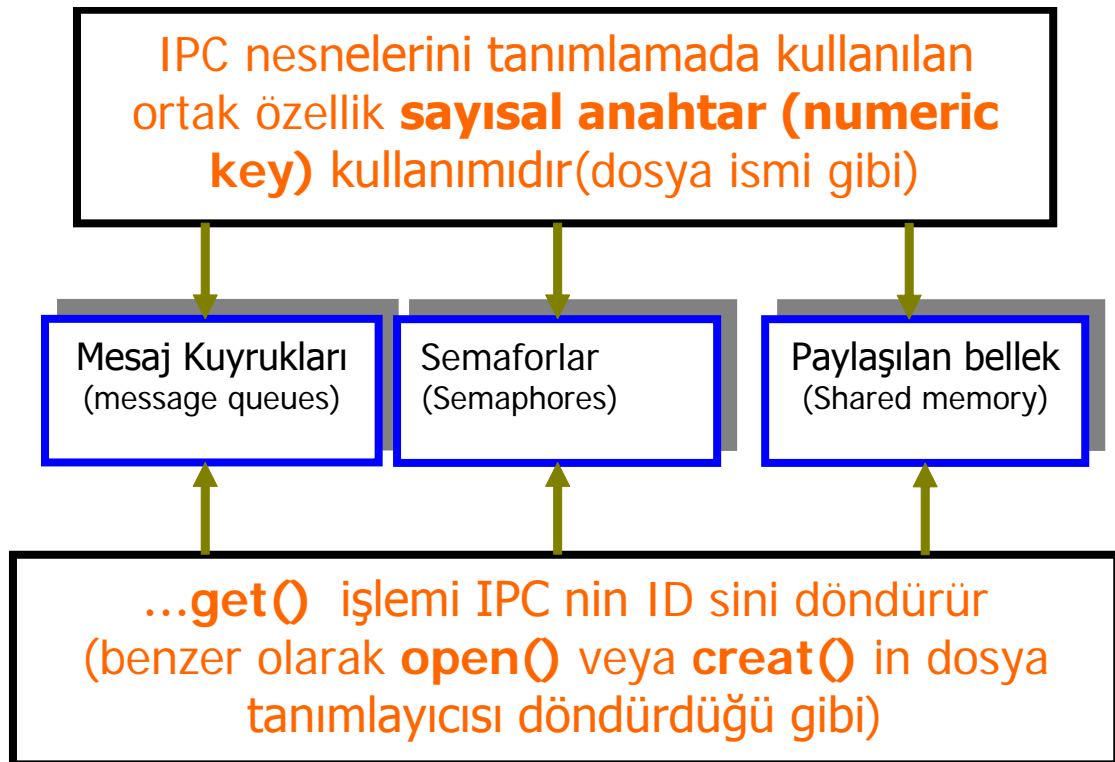


Bir grup ilişkili IPC mekanizması



UNIX de yaratılan her nesne(object) karmaşık veri yapısı (complex data structure), veya tanımlayıcı(descriptor) olarak temsil edilmiştir.

Bu veri yapısının bir parçasında izin alanıdır (permission field).

mesaj kuyrukları, semaforlar ve paylaşılan bellek mekanizmalarının sistem çağrıları

| Mesaj kuyruğu | Semafor | Paylaşılan bellek | İşlevi |
|------------------|---------|-------------------|--|
| msgget | semget | shmget | IPC açmak veya kullanım hakkı elde etmek |
| msgsnd msgrcv | semop | shmat shmdt | Temel IPC işlemleri |
| msgctl | semctl | shmctl | Kontrol işlemleri |

Bu gurup IPC mekanizmaları hakkında komut satırından bilgi sorgulama

%ipcs -b /* bir kuyrukta bulunan maksimum byte sayısını, paylaşılan belleğin maksimum büyüklüğünü, bir sette bulunan maksimum semafor sayısını verir */

IPC status from helium ...

T ID KEY Mode Owner Group QBYTES

Message Queues:

q 50 0x00000125 -Rrw-rw---- Ahmet other 4096

Shared memory facility not in system

T ID KEY Mode Owner Group NSEMS

Semaphores:

s 0 0x00012341 --ra-ra-ra root root 2

s 1 0x00012342 --ra-ra-ra root root 1

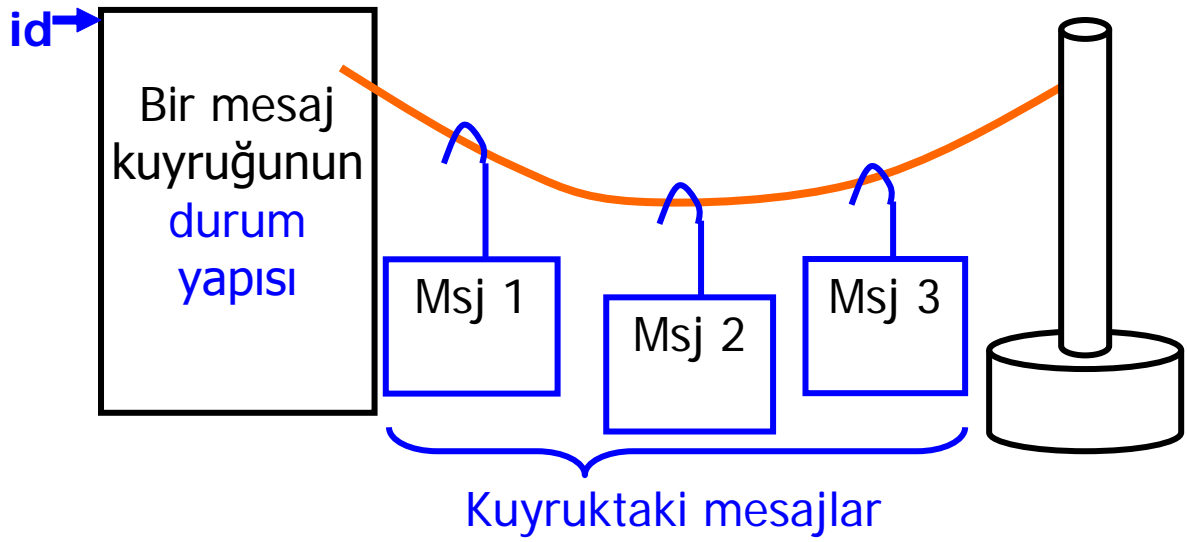
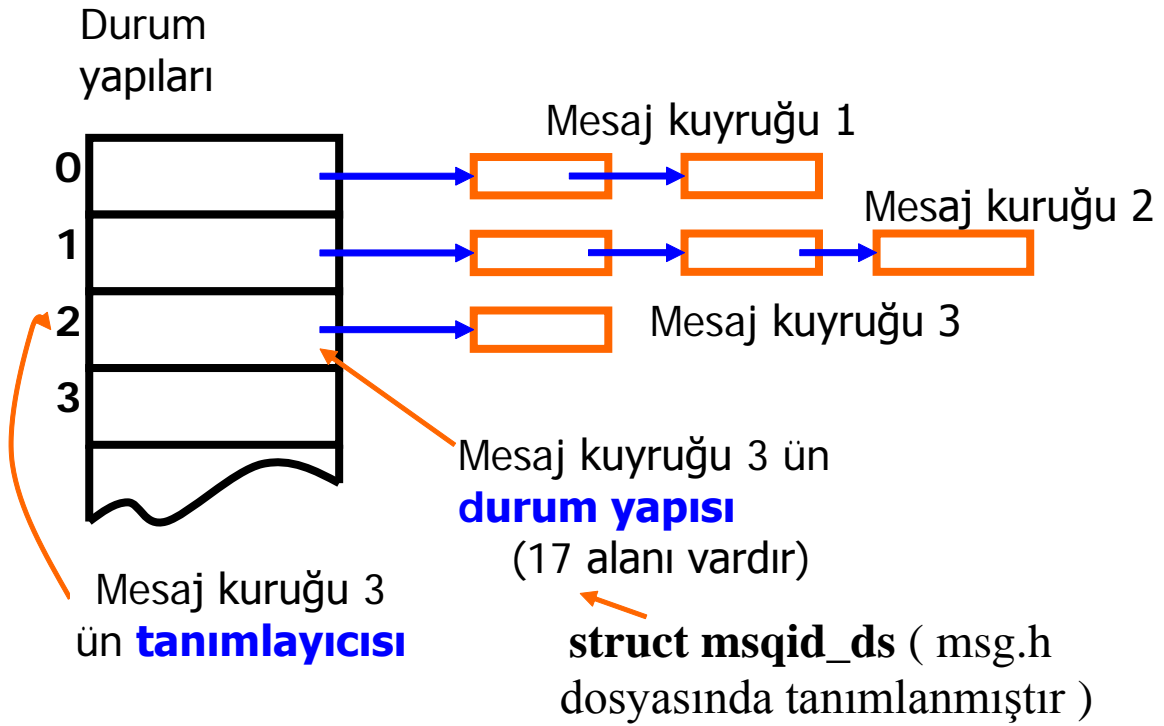
Bir mesaj kuyruğunu silmek için:

%ipcrm -q 50

veya

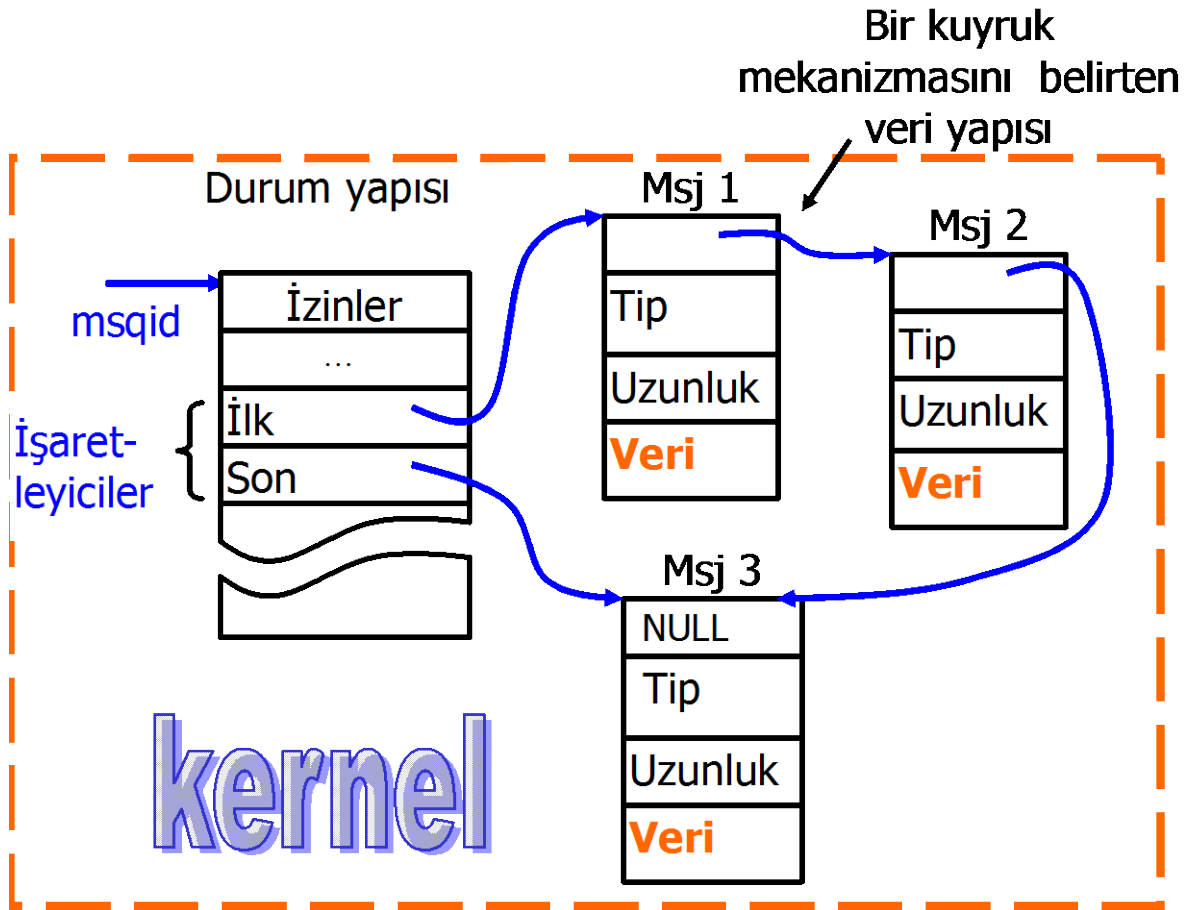
%ipcrm -Q 0x00000125

Mesaj kuyrukları için sistem veri yapıları

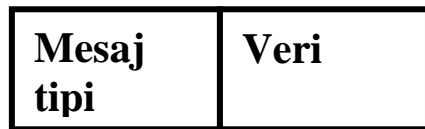


Mesaj Kuyrukları

Bir kuruk mekanizması (çekirdekte bulunan) bir mesaj listesidir.



Proses tarafından çalıştırılan programda mesaj kuyruğuna eklemek için hazırlanan mesajın veri yapısı



long integer

text(char)

Mesajın yapısı:

```
struct msgbuf
{
    long mtype;
    char mtext[ ? ]
};
```

Mesaj kuyrukları için sistem çağrıları

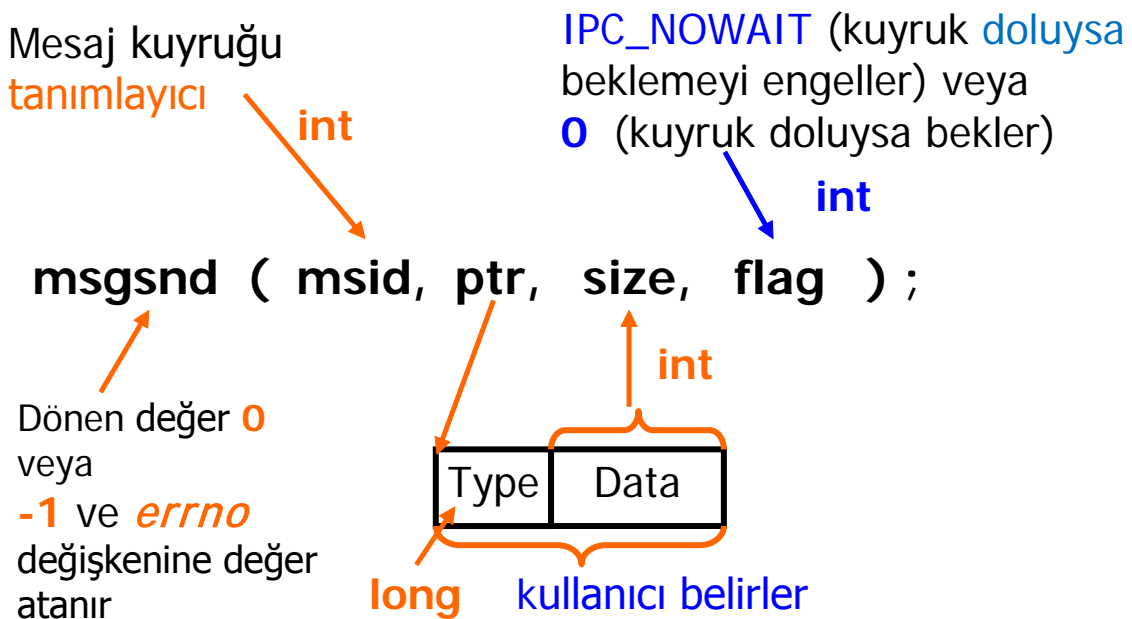
1. Mesaj kuyruğu yaratmak veya olan bir mesaj kuyruğunu kullanmaya hak kazanmak:



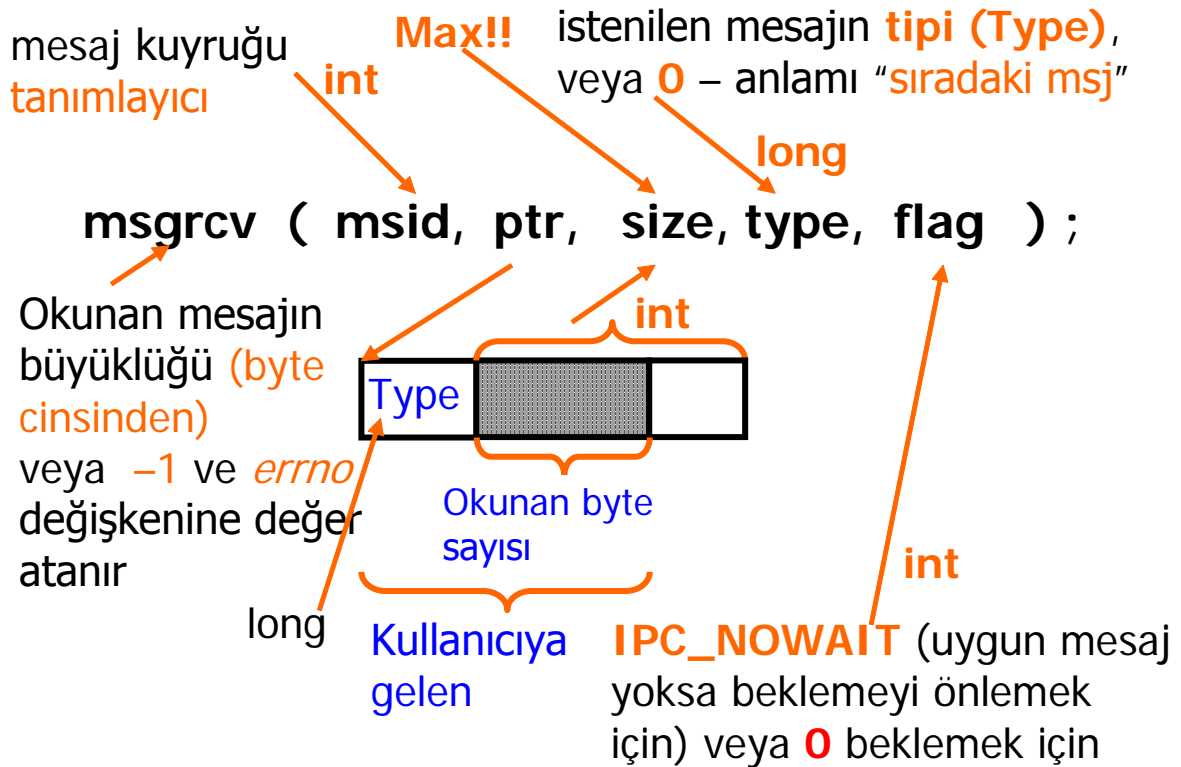
Not: [key\(anahtar\)](#), mesaj kuyruğunu sistemde tanımlar. Anahtara verilen değer sistem tarafından tek bir mesaj kuyruğu tanımlayıcısı üretmek için kullanılır.

[flag\(etiket\)](#) mesaj kuyruğunun giriş izinlerini belirlemek için kullanılır.

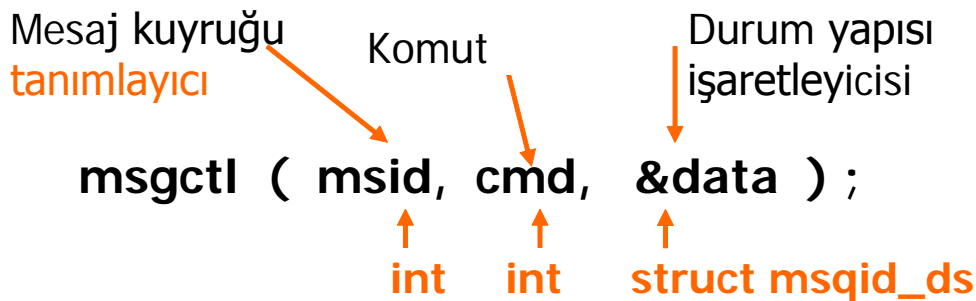
2. Kuyruğa mesaj göndermek



3. Mesaj kuyruğundan okuma (mesajın kuyruktan çıkarılması)



4. Mesaj kuyruklarında kontrol işlemleri



Komutlar:

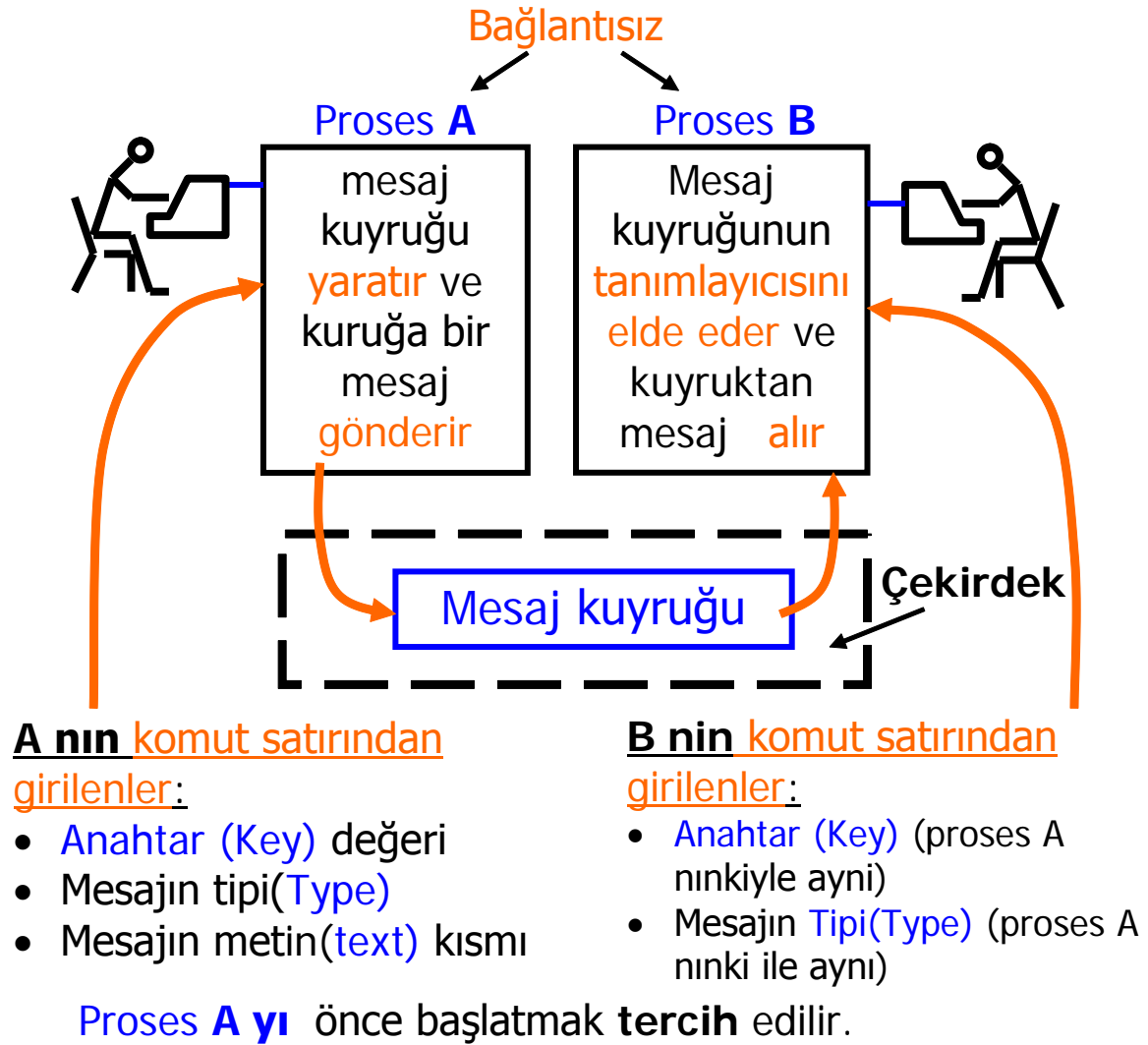
| | |
|-----------------|--|
| IPC_STAT | mesaj kuyruğu tanımlayıcısını oku |
| IPC_SET | mesaj kuyruğu yapısının bazı alanlarını değiştir |
| IPC_RMID | mesaj kuyruğunu sil |

vb...

Kullanılan kütüphane dosyaları: (bütün mesaj kuyruğu sistem çağrılarını için)

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

Mesaj kuyruklarının kullanımına örnek



Proses **A** ve **B** farklı terminallerden başlatılabilir (fakat aynı UNIX sisteminde olmalıdırlar).

Yaratıcı/gönderen, proses A'nın programı

```
#include < ... >

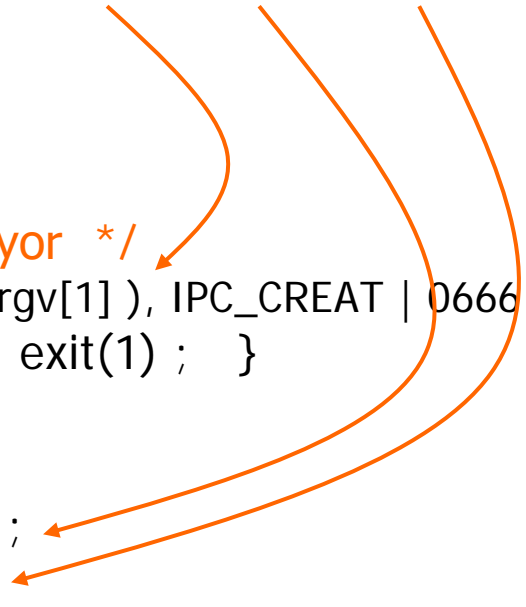
.....
struct msgbuf {
    long  type ;
    char  txt[100] ;
} ;

main(argc, argv) /*Kullanımı :<key><type><text>*/
...
{ int mid, v ;
  struct msgbuf msg ;

  /* mesaj kuyruğu yaratılıyor */
  mid = msgget((key_t) atoi( argv[1] ), IPC_CREAT | 0666 );
  if ( mid == -1 ) { ... ; exit(1) ; }

  /* Mesaj hazırlanıyor */
  msg.type = atoi( argv[2] ) ;
  strcpy( msg.txt, argv[3] ) ;

  /* kuyruğa mesaj yazılıyor */
  v = msgsnd( mid, &msg, strlen( argv[3] ) + 1, 0 ) ;
  if ( v < 0 ) { hata mesajı ...; exit(0); }
}
```



Sonuç: proses mesaj kuyruğu yaratır,
kuyruğa mesaj koyar,
ve sonlanır.

Not: msgbuf in yapısı <sys/msg.h> dosyasında bulunur

Alıcı, proses B nin programı:

```
#include < ... >
```

```
struct msgbuf {  
    long   type ;  
    char   txt[100] ;  
} ;
```

```
main(argc, argv)      /* Kullanımı : <key> <type> */
```

```
...
```

```
{ int mid, v ;  
  struct msgbuf msg ;   int m=50; /* büyüklük */
```

```
    /* Mesaj kuyruğu tanımlayıcısını al */
```

```
mid = msgget( (key_t) atoi( argv[1] ), 0 ) ;  
if ( mid == -1 ) { ... ; exit(1) ; }
```

```
    /* Verilen tipteki mesajı oku */
```

```
v = msgrcv( mid, &msg, m, atoi(argv[2] ), IPC_NOWAIT ) ;  
if ( v < 0 ) { hata mesajı yazdır ve çık }  
else  
    printf("%d %s\n", msg.type, msg.txt ) ;
```

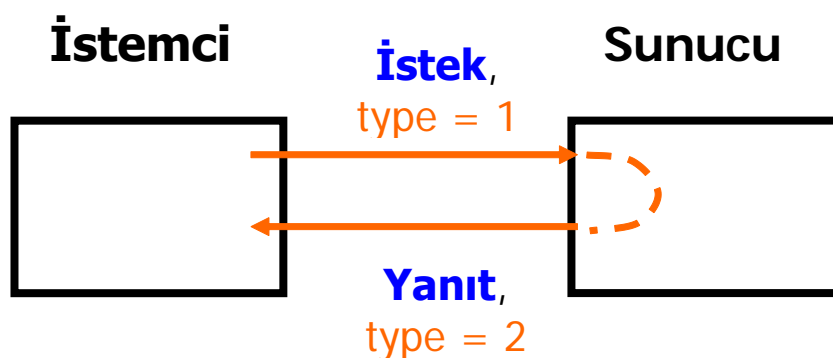
```
    /* UNIX den mesaj kuyruğunu sil */
```

```
msgctl( mid, IPC_RMID, 0 ) ; exit(0) ;  
}
```

Sonuç: Proses mesaj kuyruğu tanımlayıcısını **alır**,
kuyruktan mesaj **okur** ve kuyruğu **siler**.

Mesaj kuyrukları ile **istemci-sunucu** sistemi (client-server system)

| Sunucu(Server) | İstemci(Client) |
|---|---|
| <pre> key=123 ; main() { struct msgbuf sbuf, rbuf; int mid ; mid=msgget(key, IPC_CREAT 0666); msggrcv(mid,&rbuf,128,0,0); sbuf.type=2 ; strcpy(sbuf.txt,"Sunucudan gelen"); msgsnd(mid,&sbuf, strlen(sbuf.txt)+1,0); } </pre> | <pre> key=123 ; main() { struct Msg sbuf, rbuf; int mid ; mid=msgget(key,0); sbuf.type=1 ; strcpy(sbuf.txt,"İstemciden gelen"); msgsnd(mid,&sbuf,strlen(sbuf.txt)+1,0); msggrcv(mid,&rbuf,128,2,0); printf(...,rbuf.txt); msgctl(mid,IPC_RMID,0); } </pre> |



| Sunucu işlemi ← önce başlar | İstemci işlemi |
|--|---|
| <pre> #include ... struct Msg{ long mtype, char txt[128];}; main() { struct Msg sbuf, rbuf; int mid ; key_t key = 123 ; if ((mid=msgget(key, IPC_CREAT 0666))<0) { error ... ; exit(1) ; } /* receive a message */ if (msgrcv(mid,&rbuf, 128, 0,0) < 0) (any type) → (wait) { error ..., exit(1) ; } /* send a message */ sbuf.type=2 ; sbuf.txt[]<-"Sunucudan gelen"; if (msgsnd(mid,&sbuf, strlen(sbuf.txt)+1,0)<0) { error ..., exit(1) ; } exit(0) ; } </pre> | <pre> #include ... struct Msg{ long mtype, char txt[128];}; main() { struct Msg sbuf, rbuf; int mid ; key_t key = 123 ; if ((mid=msgget(key,0))<0) { error ... ; exit(1) ; } /* send message */ sbuf.type=1 ; /*type=1*/ sbuf.txt[]<-"İstemciden gelen"; if (msgsnd(mid,&sbuf, (wait) strlen(sbuf.txt)+1,0)<0) { error ..., exit(1) ; } /* receive a message */ if (msgrcv(mid,&rbuf,128, (type 2) → (wait) 2,0)<0) { error ..., exit(1) ; } /* print received message */ printf("%s\n",rbuf.txt); exit(0); } </pre> |

Curry, D.A., UNIX Systems Programming for SVR4, O'Reilly & Associates, 1996, pp. 137-140