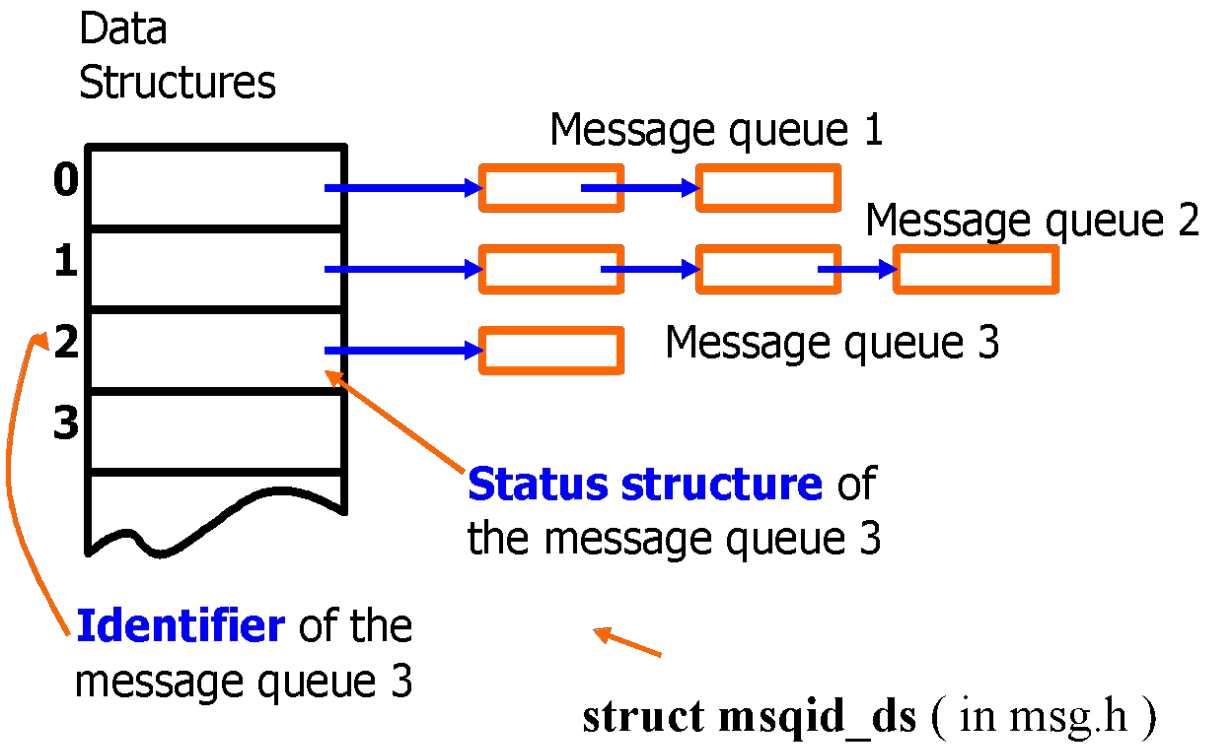
Interprocess Communication
IPC
Message Queues

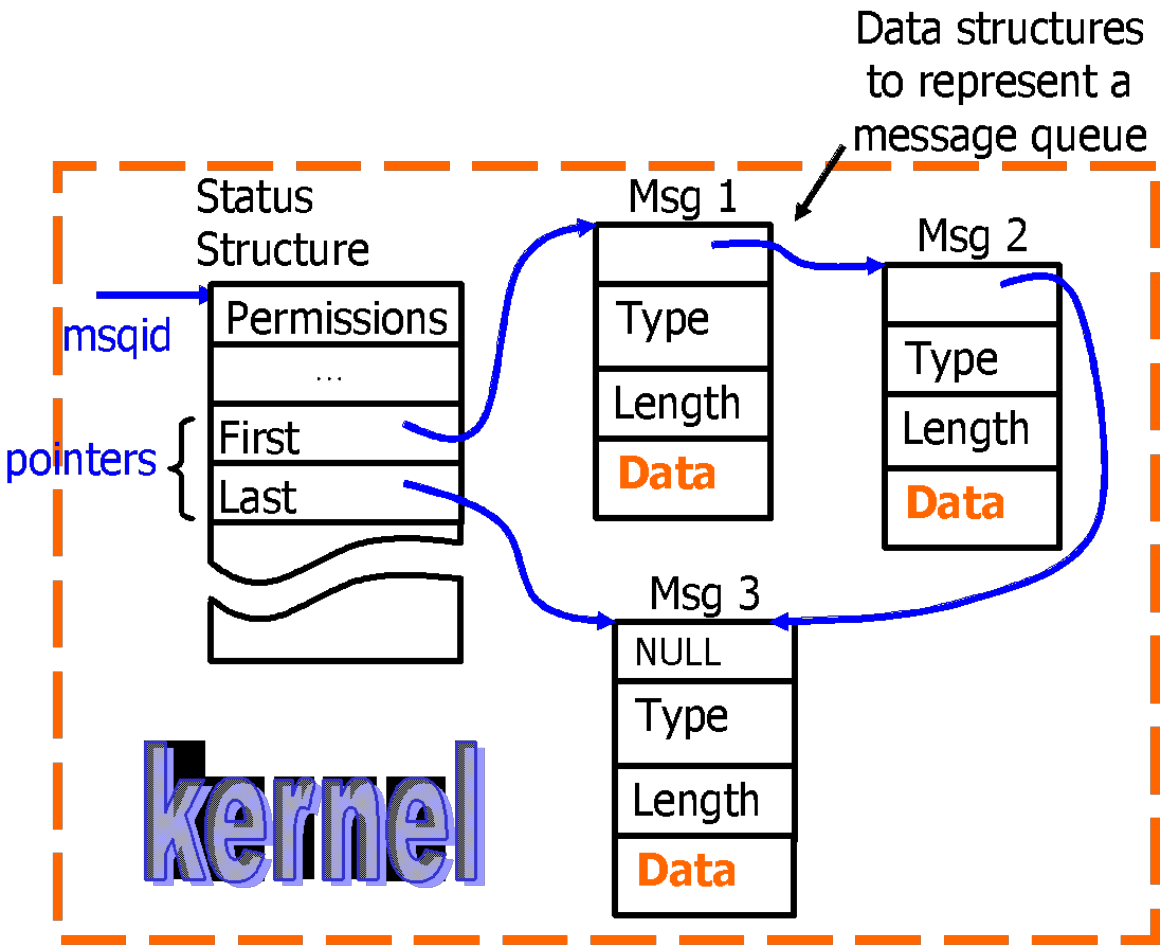
Message Queues

- An ordered list of messages held in the kernel
- Is identified by a numeric key
- It has ownership and access modes
- It exists quite independently from any particular user process
- Functionality is mid-way between pipes (FIFO) and shared memory (random)

Message Queues Housekeeping



Detailed Message Queue

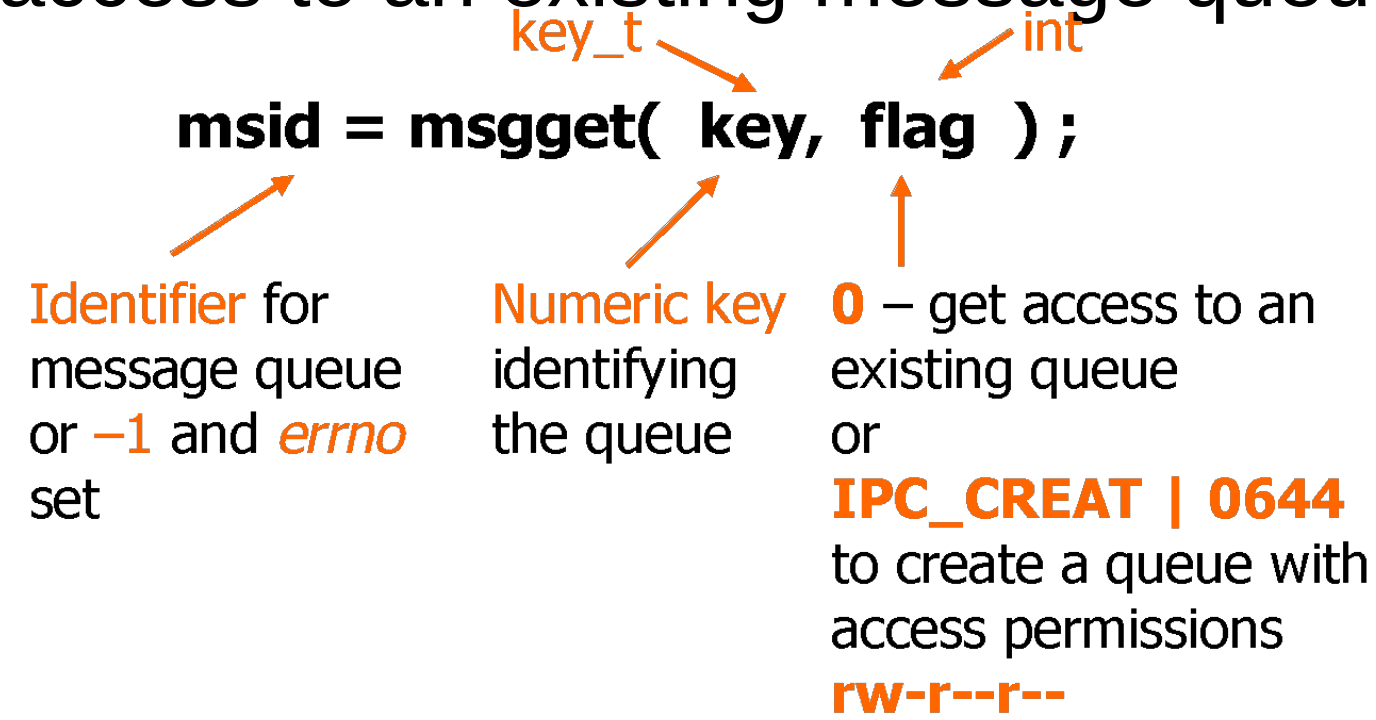


msqid_ds (in msg.h)

```
stuct msqid_ds
{
stuct ipc_perm msg_perm;   /* opertion permission struct   */
struct  msg  *msg_first; /* ptr to first message on q  */
struct  msg  *msg_last;  /* ptr to last message on  q  */
ushort       msg_cbytes;     /* current # bytes on q   */
ushort       msg_qnum;   /* current # of messages on q */
ushort       msg_qbytes  /* max # of bytes allowed on q   */
ushort       msg_lspid;  /* pid of last msgsnd     */
ushort       msg_lrpid;      /* pid of last msgrcv     */
time_t       msg_stime;  /* time of last msgsnd        */
time_t       msg_rtime;      /* time of last msgrcv        */
time_t       msg_ctime;  /* time of last msgct1        */
}
```

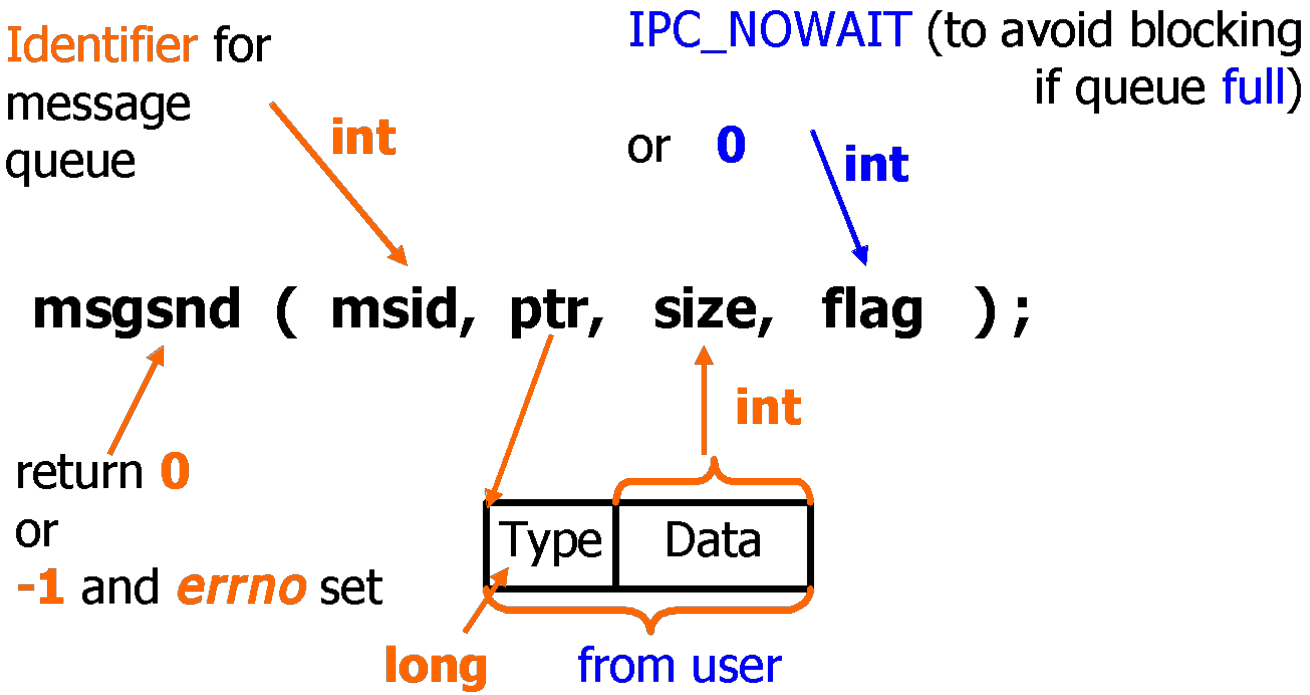
Related System Calls

1. Creating a message queue or getting access to an existing message queue:



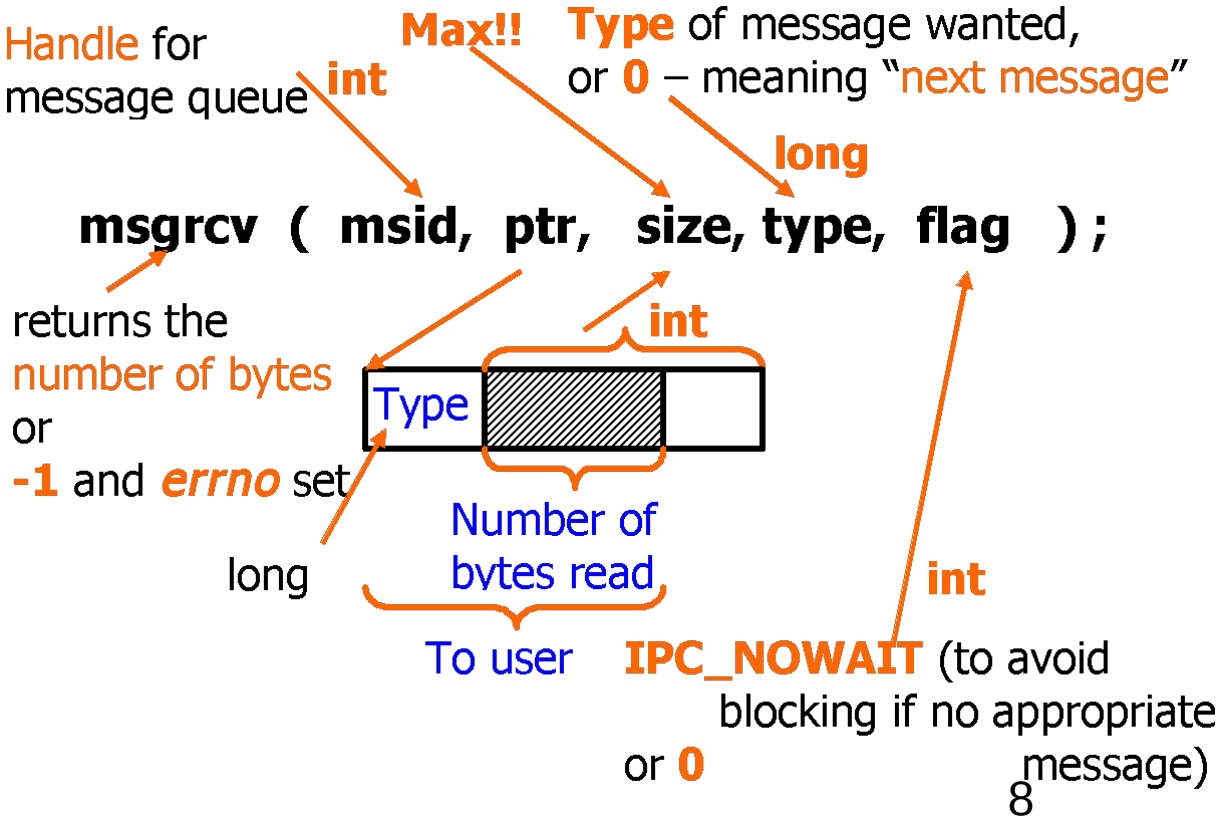
Related System Calls

1. Sending a message to a queue:



Related System Calls

1. Reading from a message queue (and removing the message)



Receiving a message

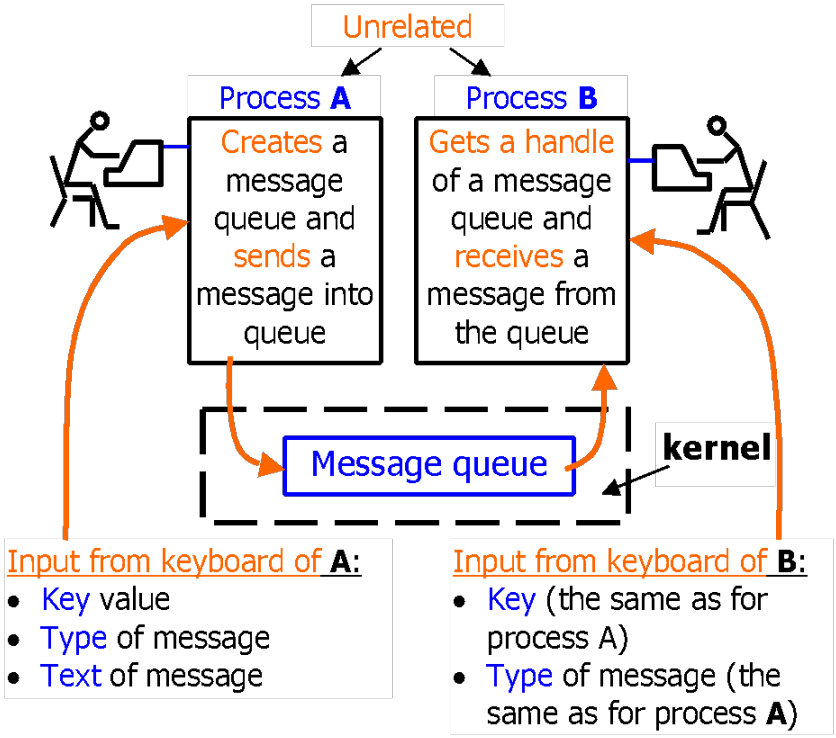
- User has some control over the order of how messages are retrieved
  - Type: 0 → Next msg in Q
  - Type: +ve → gets the 1st msg of that type
  - Type: -ve → gets the 1st msg of the lowest type which is less than or equal to the absolute value of type

1. Deallocate or change permissions for the message queue:

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

- Cmd:
- IPC_RMID remove the message queue msqid and
  destroy the corresponding msqid_ds
- IPC_SET set members of the msqid_ds data
  structure from buf
- IPC_STAT copy members of the msqid_ds data
  structure into buf

Example



It is **desirable** to start process **A** first.

Process **A** and **B** may be started from different terminals (but on the same UNIX).

Example: (Sender)

```
#include < … >

struct text_message {
   long   mtype  ;
   char   mtext[100] ;
}  ;

main(int argc, char *argv[]) /*usage :a.out <key><type><text>*/
{
   int  msid, v  ;
   struct  text_message  mess ;
   /* Creating a message queue  */
   msid = msgget((key_t) atoi( argv[1] ), IPC_CREAT | 0666  );
   if ( msid == -1)
   {
       …   ;
       exit(1) ;
   }
```

```
/* Preparing a message  */
   mess.mtype = atoi( argv[2] ) ;
   strcpy( mess.mtext, argv[3] ) ;
   /* write a message onto queue  */
   v = msgsnd( msid, &mess, strlen( argv[3] ) + 1, 0 ) ;
   if ( v < 0 )
   {
      error message …;
      exit(0);
   }
}
```

- **The result:** Process has created a message queue, put one message onto the queue, and finished.

Example (Receiver)

```
#include < … >
struct text_message {
   long   mtype  ;
   char   mtext[100];
}  ;

main(int argc, char *argv[])  /* usage :a.out <key>  <type>   */
{
   int  msid, v  ;
   struct  text_message mess ;
   /* Get a message handle  */
   msid = msgget( (key_t) atoi( argv[1] ), 0 )  ;
   if ( msid == -1)    {   …   ;   exit(1) ;   }
   /* Read a message of the given type */
   v = msgrcv( msid, &mess, 100, atoi( argv[2] ), IPC_NOWAIT ) ;
```

```
if ( v < 0 )
   {  error message and exit
   }
else
   printf("%d  %s\n", mess.mtype, mess.mtext ) ;
/* Remove the message queue from UNIX  */
   msgctl( msid, IPC_RMID, 0 ) ; exit(0) ;
}
```

- **The result**: The process **got** a message queue identifier, **read** the message from the queue, and **removed** the message queue.