InterProcess Communication - IPC

- With System V, AT&T introduced three new forms of IPC facilities:
  - **message queues,**
  - **semaphores,**
  - **and shared memory.**

  While the POSIX committee has not yet completed its standardization of these facilities, most implementations do support these. In addition, Berkeley (BSD) uses sockets as its primary form of IPC, rather than the System V elements. Linux has the ability to use both forms of IPC (BSD and System V).

IPC identifier

- Each IPC *object* has a unique IPC identifier associated with it.
- The uniqueness of an identifier is relevant to the *type* of object in question.
- Often, the ftok() function is used to generate key values for both the client and the server.

```
key_t ftok (char *pathname,char proj);
RETURNS:     new IPC key value if successful
         -1 if unsuccessful, errno set to return of stat() call
```

Message Queues

- Message queues can be best described as an internal linked list within the kernel's addressing space.
- Messages can be sent to the queue in order and retrieved from the queue in several ways.
- data structures that reside within the confines of the kernel itself:
  - Message buffer
  - Kernel `msg` structure
  - Kernel `msqid_ds` structure
  - Kernel `ipc_perm` structure

Message buffer

- can be thought of as a *template* for message data

- While it is up to the programmer to define structures of this type, **it is imperative that you understand that there is actually a structure** of type `msgbuf`.

- It is declared in **`linux/msg.h`** as follows:

```
/* message buffer for msgsnd and msgrcv calls */
struct msgbuf {
long mtype;  /* type of message */
char mtext[1];      /* message text */
};
```

  mtype - This *must* be a positive number!
  mtext - The message data itself.

- In Linux the maximum size is defined in `linux/msg.h`:

```
#define MSGMAX 4056 /* <= 4056 bytes*/
```

Kernel `msg` structure

- The kernel stores each message in the queue within the framework of the `msg` structure (`linux/msg.h`)

```
/* one msg structure for each message */
struct msg {
struct msg *msg_next; /* next message on queue */
long msg_type;
char *msg_spot; /* message text address */
short msg_ts; /* message text size */
};
```

`msg_next` - This is a pointer to the next message in the queue. They are stored as a singly linked list within kernel addressing space.
`msg_type` - This is the message type, as assigned in structure `msgbuf`.
`msg_spot` - A pointer to the beginning of the message body.
`msg_ts` - The length of the message text, or body.

Kernel `msqid_ds` structure

- Internal data structure which is maintained by the kernel.
- Kernel creates, stores, and maintains an instance of this structure for every message queue created on the system.
- `linux/msg.h`

```
struct msqid_ds {
    struct ipc_perm msg_perm;
    struct msg *msg_first;  /* first message on queue */
    struct msg *msg_last;   /* last message in queue */
    time_t msg_stime;       /* last msgsnd time */
    time_t msg_rtime;       /* last msgrcv time */
    time_t msg_ctime;       /* last change time */
    struct wait_queue *wwait;
    struct wait_queue *rwait;
    ushort msg_cbytes;
    ushort msg_qnum; /*Number of messages currently in the queue.*/
    ushort msg_qbytes;      /* max number of bytes on queue */
    ushort msg_lspid;       /* pid of last msgsnd */
    ushort msg_lrpid;       /* last receive pid */
};
```

Kernel `ipc_perm` structure

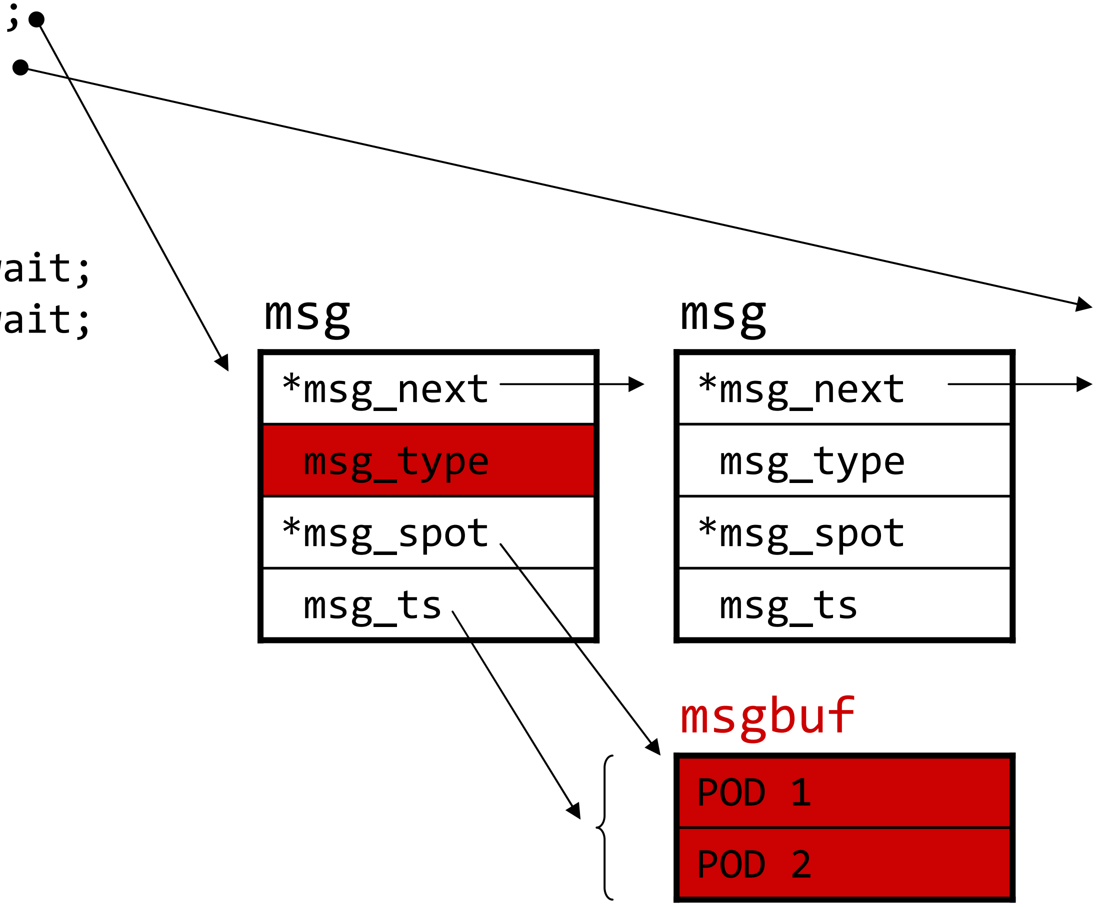
- The kernel stores permission information for IPC objects in a structure of type `ipc_perm`
- `linux/msg.h`

```
struct ipc_perm
{
  key_t  key;
  ushort uid;   /* owner euid and egid */
  ushort gid;
  ushort cuid;  /* creator euid and egid */
  ushort cgid;
  ushort mode;  /* access modes see mode flags below */
  ushort seq;   /* slot usage sequence number */
};
```

- `seq` - *slot usage sequence* number. Each time an IPC object is closed via a system call (destroyed), this value gets incremented by the maximum number of IPC objects that can reside in a system.

Kernel strukture

```
struct msqid_ds {
struct ipc_perm msg_perm;
struct msg *msg_first;
struct msg *msg_last;
time_t msg_stime;
time_t msg_rtime;
time_t msg_ctime;
struct wait_queue *wwait;
struct wait_queue *rwait;
ushort msg_cbytes;
ushort msg_qnum;
ushort msg_qbytes;
ushort msg_lspid;
ushort msg_lrpid;
};
```



SYSTEM CALL: msgget()

- The kernel stores permission information for IPC objects in a structure of type ipc_perm

```
PROTOTYPE: int msgget ( key_t key, int msgflg );
RETURNS: message queue identifier on success
  -1 on error: errno = EACCESS (permission denied)
                       EEXIST (Queue exists, cannot create)
                       EIDRM (Queue is marked for deletion)
                       ENOENT (Queue does not exist)
                       ENOMEM (Not enough memory to create queue)
                       ENOSPC (Maximum queue limit exceeded)
```

- Primer:

```
int open_queue( key_t keyval )
{
   int     qid;
   if((qid = msgget( keyval, IPC_CREAT | 0660 )) == -1)
   { return(-1); }
   return(qid);
}
```

SYSTEM CALL: msgsnd()

- deliver a message to a queue

```
PROTOTYPE:
int msgsnd (int msqid,struct msgbuf *msgp,int msgsz,int msgflg);
RETURNS: 0 on success
        -1 on error: errno = EAGAIN
          (queue is full, and IPC_NOWAIT was asserted)
         EACCES (permission denied, no write permission)
         EFAULT (msgp address isn't accessable - invalid)
         EIDRM  (The message queue has been removed)
         EINTR  (Received a signal while waiting to write)
         EINVAL (Invalid message queue identifier, nonpositive
                message type, or invalid message size)
         ENOMEM (Not enough memory to copy message buffer)
```

- Primer:

```
int send_message( int qid, struct mymsgbuf *qbuf )
{       int result, length;
        length = sizeof(struct mymsgbuf) - sizeof(long);
        if((result = msgsnd( qid, qbuf, length, 0)) == -1)
          {return(-1);}
        return(result);}
```

SYSTEM CALL: `msgrcv()`

- read a message from a queue

```
PROTOTYPE: int msgrcv ( int msqid, struct msgbuf *msgp, int msgsz,
long mtype, int msgflg );
RETURNS: Number of bytes copied into message buffer
 -1 on error:
errno = E2BIG  (Message length is greater than msgsz, no
   MSG_NOERROR)
        EACCES (No read permission)
        EFAULT (Address pointed to by msgp is invalid)
        EIDRM  (Queue was removed during retrieval)
        EINTR  (Interrupted by arriving signal)
        EINVAL (msgqid invalid, or msgsz less than 0)
        ENOMSG (IPC_NOWAIT asserted, and no message exists
                              in the queue to satisfy the request)
```

- Primer:

```
int read_message( int qid, long type, struct mymsgbuf *qbuf )
{         int     result, length;
   length = sizeof(struct mymsgbuf) - sizeof(long);
        if((result = msgrcv( qid, qbuf, length, type,  0)) == -1)
           {  return(-1); }
        return(result);}
```

SYSTEM CALL: `msgctl()`

```
PROTOTYPE: int msgctl ( int msgqid, int cmd, struct msqid_ds *buf );
RETURNS: 0 on success
-1 on error: errno = EACCES (No read permission and cmd is IPC_STAT)
    EFAULT (Address pointed to by buf is invalid with IPC_SET and
            IPC_STAT commands)
    EIDRM  (Queue was removed during retrieval)
    EINVAL (msgqid invalid, or msgsz less than 0)
    EPERM  (IPC_SET or IPC_RMID command was issued, but
   calling process does not have write (alter access to the queue)
```

`IPC_STAT` Retrieves the msqid_ds structure for a queue, and stores it in the address of the buf argument.
`IPC_SET` Sets the value of the ipc_perm member of the msqid_ds structure for a queue. Takes the values from the buf argument.
`IPC_RMID` Removes the queue from the kernel.

SYSTEM CALL: `msgctl()`

Primer 1.

```
int get_queue_ds( int qid, struct msgqid_ds *qbuf )
{ if( msgctl( qid, IPC_STAT, qbuf) == -1) { return(-1); }
   return(0); }
```

Primer 2.

```
int remove_queue( int qid )
{ if( msgctl( qid, IPC_RMID, 0) == -1) { return(-1); }
  return(0);}
```

Primer 3.

```
int change_queue_mode( int qid, char *mode )
{ struct msqid_ds tmpbuf; /* Retrieve a current copy of the internal
          data structure */
get_queue_ds( qid, &tmpbuf); /* Change the permissions using an old
            trick */
sscanf(mode, "%ho", &tmpbuf.msg_perm.mode); /* Update the internal
            data structure */
if( msgctl( qid, IPC_SET, &tmpbuf) == -1) { return(-1); }
return(0);
}
```

SYSTEM CALL: `msgctl()`

Primer 1.

```
int get_queue_ds( int qid, struct msgqid_ds *qbuf )
{ if( msgctl( qid, IPC_STAT, qbuf) == -1) { return(-1); }
   return(0); }
```

Primer 2.

```
int remove_queue( int qid )
{ if( msgctl( qid, IPC_RMID, 0) == -1) { return(-1); }
  return(0);}
```

Primer 3.

```
int change_queue_mode( int qid, char *mode )
{ struct msqid_ds tmpbuf; /* Retrieve a current copy of the internal
          data structure */
get_queue_ds( qid, &tmpbuf); /* Change the permissions using an old
           trick */
sscanf(mode, "%ho", &tmpbuf.msg_perm.mode); /* Update the internal
           data structure */
if( msgctl( qid, IPC_SET, &tmpbuf) == -1) { return(-1); }
return(0);
}
```

signal()

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <signal.h>
int main(void)
{ void sigint_handler(int sig); /* prototype */
  char s[200];
  if (signal(SIGINT, sigint_handler) == SIG_ERR) {
     perror("signal");
     exit(1);}
  printf("Enter a string:\n");
  if (gets(s) == NULL) perror("gets");
   else
   printf("You entered: \"%s\"\n", s);
   return 0;
}
void sigint_handler(int sig) {
   printf("Not this time!\n");
}
```