

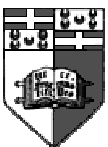


Systems V Release 4

1

IPC

- System V IPC offers semaphores, message queues and shared memory segments.
- All these structures use the same access protocol.
- Each IPC has associated with it a given **key**.
- This key is global to the whole system. Two processes using the same key will be given *access* to the same IPC structure.
- In supplying the key an **identifier** is returned, which is then used to *operate* on the IPC structure. The identifier is unique in the system too.
- Each time an IPC is created, it exists in the system until removed by a process or until the *ipcrm* command is invoked.

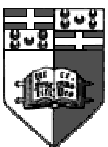




SVR4 IPC (cont)

- Keys and identifiers will run out in the system if no IPC structures are removed.
- A special key exists: `IPC_PRIVATE` which is guaranteed to be an unused key.
- The identifier or the key must somehow be passed between different processes using the same IPC structure.
- Keys or identifiers can be passed between processes using the filesystem, command line values, an agreed upon header, parent/child inheritance or using *ftok()*.
- *ftok()* converts an agreed upon pathname and project ID into an IPC key.
- The general format to obtain an identifier is: *Xget(key,flag)*.

```
#include <sys/types.h>
#include <sys/ipc.h>
key_t ftok(char *pathname, char projID);
```

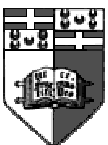




SVR4 IPC (cont)

- If a key is passed, the second process will have to 're-create' the IPC structure, if the identifier is passed, second process will access the IPC using this identifier.
- A new IPC structure will be created if the key is equal to `IPC_PRIVATE` or a new key is given and `IPC_CREAT` is specified in *flag* (inside *Xget()*).
- Using `IPC_EXCL` with `IPC_CREAT` in *flag* will produce an error with *errno* set to `EEXIST` if an IPC already exists with the same key.
- When creating an IPC, access permissions are given in *flag* of *Xget()* according to the following table:

| Permission | Message Queue | Semaphore | Shared Memory |
|-------------|---------------|-----------|---------------|
| user-read | MSG_R | SEM_R | SHM_R |
| user-write | MSG_W | SEM_A | SHM_W |
| group-read | MSG_R>>3 | SEM_R>>3 | SHM_R>>3 |
| group-write | MSG_W>>3 | SEM_A>>3 | SHM_W>>3 |
| other-read | MSG_R>>6 | SEM_R>>6 | MSG_R>>6 |
| other-write | MSG_W>>6 | SEM_A>>6 | MSG_W>>6 |

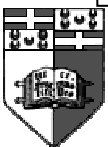




Message Queues

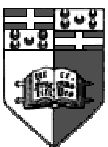
- This is a linked list of messages stored within the kernel.
- `msgget()` opens an existing queue or creates a new one and returns the IPC queue identifier.
- `msgctl()` is used to remove the structure created by the same *effective* user. Removal is immediate and anyone using the structure will get an error of EIDRM.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget(key_t key, int flag);
    returns message queue ID or -1 on error
int msgctl(int msgid, IPC_RMID, NULL);
    returns -1 on error
```



Unix Message Queues (cont)

- Some limits exist for message queues:
 - MSGMAX: largest message allowed in bytes
 - MSGMNB: max size of a queue in bytes.
 - MSGMNI: max number of queues, system wide
 - MSGTQL: max number of messages, system wide
- Each message consists of a positive *long* value specifying the message type followed by data *nbytes* long.
- One can use this structure to hold messages:
 - struct messages {
 long mtype;
 char data[nbytes];
}

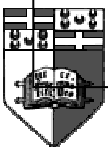




Sending Messages

- *msgsnd()* puts a message on the message queue.
- *ptr* points to a structure similar to *struct message*. *nbytes* gives the size of the message data not of the whole structure. *flag* can only be `IPC_NOWAIT`.
- This system call blocks until there is space on the queue, the queue is removed from the system (`EIDRM`) or a signal is caught (`EINTR`).
- If `IPC_NOWAIT` is specified in *flag*, *msgsnd()* never blocks but sets *errno* to `EAGAIN` on error.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgsnd(int msgid, const void *ptr,
           size_t nbytes, int flag);
           returns -1 on error
```

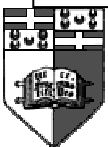




Receiving Messages

- *msgrcv()* receives a message from the queue.
- This blocks until there is a message on the queue of specified type, the queue is removed (EIDRM) or a signal is caught (EINTR).
- If `IPC_NOWAIT` is specified in *flag*, *msgrcv()* never blocks but sets *errno* to `ENOMSG` if no message of specified type exists.

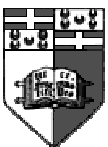
```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgrcv(int msgid, void *ptr, size_t nbytes,
           long type, int flag);
           returns -1 on error
```





Receiving Messages

- *type* sets how the queue behaves
 - $type=0$: First message on queue retrieved.
 - $type>0$: First message whose type value equals *type*.
 - $type<0$: First message whose type value is equal or smaller than the absolute value of *type*.
- *nbytes* is the maximum size of the desired data, not of the whole message.
- If message on queue is larger than *nbytes*, the message remains on queue and *msgrcv* returns with *errno* set to E2BIG.
- If MSG_NOERROR is set in *flag*, and the retrieved message is larger than *nbytes*, the message is removed from the queue but is truncated. Note that no error message is returned if message is truncated.





Exercises

- Create a message queue and place messages upon it and retrieve them from other functions, using the queue as a sort of temporary storage.
- Create the consumer and producer scenario where the IPC identifier is shared using the parent/child relationship.
- Make the above consumer non-blocking but which checks occasionally for messages to retrieve.
- Make the producer send signals to the non-blocking consumer to tell it that there is a message present on the queue.
- Modify the producer and consumer to make them share identifiers using the filesystem and again using command line options.

