System V

System V is an platform specific API for linux/unix and introduces 3 Inter-Process Communication (IPC) facilities to share data among processes, each object of these facilities takes the form of one of the following; A:

- Message Queue,
- Shared Memory Segment, or
- Semaphore Set

Each IPC *object* has a unique IPC identifier associated with it that is used to:

- Create,
- Control, and
- Remove it

This unique key is unique w.r.t the type of the object; i.e. we can have a message queue with key id 555 and a shared memory segment with the same key id 555.

Remember: You can always use the "**man**" command to know more information about the functions and commands listed below

Linux commands used:

Command	Description
ipcs	When you write a program using these facilities, you need to monitor its consumption of the previous types of objects, the command is used for this purpose, and you need to make sure that your program doesn't leave any dangling objects behind. To handle this inside the code use the removal control commands (system calls) assigned to each type
ipcrm	Sometimes you need to remove IPC objects manually, using this command

Any IPC object needs to be created or retrieved, the programmer controls and manipulates it, and when it's no longer needed, it should be removed. The following table summarizes the system calls that can be used with each IPC object, and the header files needed for such type

IPC object type	System calls	Brief usage	Header Files (#include)
Message Queue	msgget msgctl msgsnd msgrcv	**create or access** **control** send message receive message	**<sys/types.h>** **<sys/ipc.h>** <sys/msg.h>

Semaphore Set	semget semctl semop	**create or access** **control** execute operation	**<sys/types.h>** **<sys/ipc.h>** <sys/sem.h>
Shared Memory Segment	shmget shmctl shmat shmdt	**create or access** **control** Attach memory to process Detach memory from process	**<sys/types.h>** **<sys/ipc.h>** <sys/shm.h>

General Used Function

Fn	Params List	Return	Description

ftok	**const char** **pathname*, **int** *proj_id*	Key_t key	Generate an IPC unique key
			Example
			char *path = "/tmp";
			int id = 'S';
			key_t key = ftok(path, id);

Message Queue system calls

Message queue is like the mail box (async communication method)
Mailbox has a capacity (number, size of the content)

First we need to create a container for the message :)

Message Buffer

Each message should be saved in a *msgbuf* structure. This particular data structure can be thought of as a ***template*** for message data. The most basic message buffer is defined as follows:

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

There are two members in the *msgbuf* structure:
mtype: The message type, represented as a positive number.
mtext: The message data itself

The ability to assign a given message a *type*, essentially gives you the capability to *multiplex* messages on a single queue. For instance, client processes could be assigned a magic number, which could be used as the message type for messages sent from a server process. The server itself could use some other number, which clients could use to send messages to it. In another scenario, an application could mark error messages as having a message type of 1, request messages could be type 2, etc. The possibilities are endless. This structure can get redefined by the application programmer. Consider this redefinition:

```
struct my_msgbuf {
        long    mtype;          /* Message type */
        long    request_id;     /* Request identifier */
        struct  client info;    /* Client information structure */
};
```

Function	Params List	Return	Description
msgget	key_t key, int flags	int: Object identifier on success, or -1 on error	To create/retrieve a queue object If the key doesn't exist or Key = IPC_PRIVATE it creates a new one Flags: **IPC_CREAT** Create the object if it doesn't already exist in the kernel, otherwise returns the id of the exisiting one. **IPC_EXCL** When used with IPC_CREAT, fail if the object already exists. **<octal_Number>** e.g. 0666 which is the object permissions Example: int id =msgget(8, IPC_CREAT | 0660)) 8 is a queue key
msgsnd	Failure returns -1, otherwise returns 0	int msqid, struct msgbuf *msgp, int msgsz, int msgflg	Flags: **0** (ignore) **IPC_NOWAIT** (Queue is full return) **MSG_NOERROR** (Size of the msg greater than msgsz? truncate, if this flag is not specified, call fails and the message remains in the queue)

msgrcv	Failure -1 returns the, success returns the number of bytes actually copied into the mtext array.	int msqid, struct msgbuf *msgp, int msgsz, long mtype, int msgflg	mtype: 0 (oldest), otherwise receive from a specific type Flags: **IPC_NOWAIT**(No message in the queue return) **MSG_EXCEPT** (the first message in the queue of type not equal to mtype will be read.) **MSG_NOERROR** (Size of the msg greater than msgsz? truncate, if this flag is not specified, call fails and the message remains in the queue)
msgctl		int msqid, int cmd, struct msqid_ds *buf	The control operation changes by changie the command Possible values of *cmd* are: ***IPC_STAT*** : Retrieves the message queue data structure, and stores it in the address pointed to by buf . ***IPC_SET*****:** Write the values of the msqid_ds structure pointed to by buff to the message queue data structure. ***IPC_RMID*****:** Removes the queue from the kernel.

*****Practice**
Use the source file *ipc_msg.c* to test the usage of messages.

References
The linux documentation project