Message queues provide a reasonably easy and efficient way of passing data between two unrelated processes . It has the advantage over the named pipes that the message queue exists independently of both sending and receiving processes, which removes some of the difficulties that occur in synchronizing the opening and closing of named pipes.
Message queue provide a way of sending a block of data from one process to another. Additionally , each block of data is considered to have a type, and a receiving process may receive blocks of data having different type values independently. Disadvantage of message queues are that there's a maximum size limit imposed on each block of data and also a limit on the maximum total size of all blocks on all queues throughout the system.
The message queue function definition are as follows:

```
#include<sys.msg.h>
int msgctl (int msqid, int cmd, struct msqid_ds *buf );
int msgget( key_t key, int msgflg);
int msgrcv(int msqid, void *msg_ptr, size_t msg_sz, long int msgtype, int msgflg);
int msgsnd(int msqid, void *msg_ptr, size_t msg_sz, int msgflg);
```

As with semaphores and shared memory, the include files sys/types.h and sys/ipc.h are normally also required.

msgget

We create and acess a message queue using the msgget function:

```
int msgget( key_t key, int msgflg);
```

The program must provide a key value that names a particular message queue. The special value IPC_PRIVATE creates a private queue, which in theory is accessible only by the current process. The second parameter , msgflg consists of nine permission flags. A special bit defined by IPC_CREAT must be bitwise Ored with the permissions to create a new message queue. It's not an error to set the IPC_CREAT flag and give the key of an existing message queue. This flag is simply ignored if that queue already exists.
This function returns a positive number, the queue identifier on success or -1 on failure.

msgsnd

The msgsnd function allows us to add a message to a message queue:

```
int msgsnd(int msqid, void *msg_ptr, size_t msg_sz, int msgflg);
```

The structure of the message is constrained in two ways. First, it must be smaller than the system limit and second, it must starts with a long int, which will be used as a message type in the receive function.
The first parameter, msqid is the message queue identifier returned from a msgget function.
The second parameter, msg_ptr, is a pointer to the message to be sent, which must starts with long int type.
The third parameter, msg_sz, is the size of the message pointed to by msg_ptr.
The fourth parameter, msgflg, controls what happens if either the current message queue is full or the system wide limit on queued messages has been reached. If msgflg has the IPC_NOWAIT flag set, the function will return immediately without sending the message and the return value will be -1. If the msgflg has the IPC_NOWAIT flag clear, the sending process will be suspended, waiting for space to become available in the queue.
On success, the function will return 0, on failure -1. If the call is successful, a copy of the message data has been taken and placed on the message queue.

msgrcv

The msgrcv retrieves messages from a message queue:

```
int msgrcv(int msqid, void *msg_ptr, size_t msg_sz, long int msgtype, int msgflg);
```

The first parameter, msqid, is the message queue identifier returned from a msgget function.
The second parameter, msg_ptr, is a pointer to the message to be received, which must start with a long int type.

The third parameter, msg_sz, is the size of the message pointed to by msg_ptr.
The fourth parameter, msgtype, is a long int, which allows a simple form of reception priority to be implemented. If this field has the value 0, the first available message in the queue is retrieved. If it's greater than 0, the first message with the same message type is retrieved. If it's less than 0, the first message that has the same as or less than the absolute value of msgtype is retrieved. If you simply want to retrieve messages in the order in which they were sent, set msgtype to 0, if want to retrieve messages with a specific message type, set msgtype equal to that value and if want to receive messages with a type of n or smaller, set msgtype to -n.
The fifth parameter, msgflg, controls what happens when no message of the appropriate type is waiting to be received. If the IPC_NOWAIT flag is set, the call will return immediately with a return val of -1and if this flag is clear, the process will be suspended, waiting for an appropriate type of message to arrive.
On success, msgrcv returns the number of bytes placed in the receive buffer, the message is copied into the user allocated buffer pointed to by msg_ptr, and the data is deleted from the message queue. It returns -1 on error.

msgctl

The final message queue function is msgctl, which is used as a control function:

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

The msqid_ds structure has at least the follwing members:

```
struct msqid_ds {
        uid_t msg_perm.uid;
        uid_t msg_perm.gid;
        mode_t msg_perm.mode;
}
```

The first parameter, msqid, is the queue identifier returned from msgget.
The second parameter, command, is the action to take. It can take three values:

Command	***Description***
IPC_STAT	Sets the data in the msqid_ds structure to reflect the values associated with the message queue
IPC_SET	If the process has permission to do so, this sets the values associated with the message queue to those provided in the msqid_ds structure.
IPC_RMID	Deletes the message queue

0 is returned on success and -1 on failure. If a message queue is deleted while a process is waiting in a msgsnd or msgrcv function, the send or receive function will fail.

IPC Status Commands for Message Queue:~
For message queues the commands are ipcs -q, which shows any message queue is running or not and ipcrm -q <id> (or ipcrm msg <id>) which allows a message queue to be removed.