Communication between modules

To inter-communicate modules, it will be used the IPC *(Inter-Process Communication)* defined in System V, and also found in Linux .

The interface to be used for communication between processes will be “message queues”, using it for both “command” messages and “data” messages. To differentiate the control messages from data or other messages, the *mtype* variable will be used. This variable is defined in the basic message structure (*msgbuf*). When *mtype* = MSG_CONTROL (value 1), the message is a command message, and when *mtype* = MSG_DATA (value 2), the message contains data.

To share status data or changing values, shared memory (IPC System V) will be used. When shared memory will be accessed for both read and write, semaphores will be used to define and manage a critical code area.

To create a common style and interface for use the IPC system calls, a set of functions are defined. No direct access to IPC system calls should be made from any module or procedure. This layer allows to change the interface calls to be used without made a lot of changes in the code of all modules.

Common IPC procedures

Message queue interface procedures

Use lc_msg_q.h

Create Message Queue

```
int iCreateQmsg(key_t Qkey)
```

Creates the message queue corresponding to a key. The function asigns the queue, deletes it and assigns it again, assuring the created queue is empty.

Qkey Is the Queue key.

Returns: queue identifier (int) or -1 on error.

Open message queue

```
int iOpenQmsg( key_t lQKey);
```

Open the message queue corresponding to a key. The function assigns the queue, if queue don't exist an error is returned.

Qkey Is the Queue key.

Returns: queue identifier (int) or -1 on error.

Delete message queue

```
int iDelQmsg(int id_c);
```

Deletes the message queue corresponding to a identifier.

id_c is the queue identifier returned by iOpenQmsg or iCreateQmsg

Returns o if queue deleted, -1 on error.

Send data to queue

```
int iSend2Queue(int id_c, const void *buf, int iMsgSize);
```

Sends the data contained in structure pointed by *buf. Buffer should have the msgbuf structure.

id_c	is the queue identifier returned by iOpenQmsg or iCreateQmsg
buf	Pointer to a msgbuf structures
iMsgSize	length of data area of buffer pointed by buf (sizeof(buf)-sizeof(long)).

Returns: 0 if message is sent, -1 otherwhise
Note: Upon sucessfully sending the message the memory allocated in buf is freed.

Send data to queue without freeing buffer

```
int iSend2QnoFree(int id_c, const void *buf, int iMsgSize);
```

Sends the data contained in structure pointed by *buf, without freeing the memory. Buffer should have the msgbuf structure.

id_c is the queue identifier returned by iOpenQmsg or iCreateQmsg
buf Pointer to a msgbuf structures
iMsgSize length of data area of buffer pointed by buf (sizeof(buf)-sizeof(long)).

Returns: 0 if message is sent, -1 otherwhise

Receiving message from queue

```
struct msgbuf* iRecQueue(int id_c);
```

Receive the first message from a queue.

id_c is the queue identifier returned by iOpenQmsg or iCreateQmsg

Returns: a pointer to a msgbuf structure containing the message. The memory for the structure is allocated by the function.
Note: This functions suspends the current process until a message is available and is received.

Receiving a specified type message from queue

```
struct msgbuf* iRecQueueType(int id_c, long lTypeMsg);
```

Receive the first message of a specified type from a queue.

id_c is the queue identifier returned by iOpenQmsg or iCreateQmsg
lTypeMsg is the type of message to be received.

Returns: a pointer to a msgbuf structure containing the message. The memory for the structure is allocated by the function.
Note: This functions suspends the current process until a message is available and is received.

Receiving a specified type message from queue without suspending the process

```
struct msgbuf* iRecQueueNwait(int id_c, long lTypeMsg);
```

Receive the first message of a specified type from a queue. If no message is in queue returns without suspend the process and with return value = -1 and errno = ENOMSG.

id_c	is the queue identifier returned by iOpenQmsg or iCreateQmsg
lTypeMsg	is the type of message to be received.

Returns: a pointer to a msgbuf structure containing the message. The memory for the structure is allocated by the function.

Shared Memory interface procedures

Use lc_shmem.h

Create and allocate a Shared Memory area

```
void* pCreateShMem(key_t SHmKey, int iLong);
```

Creates a shared memory area and attach it.

SHmKey	Key that identifies the shared memory area to be assigned.
lLong	Length in bytes of shared memory area to be allocated

Returns: a pointer to the allocated shared memory of NULL on error.

Assigns and allocates a previously created shared Memory area

```
void* pAssignShMem(key_t SHmKey, int iLong);
```

Allocates a previously created shared memory area and attach it.

SHmKey	Key that identifies the shared memory area to be assigned.
lLong	Length in bytes of shared memory area to be allocated

Returns: a pointer to the allocated shared memory of NULL on error.

Detach shared memory area

```
int iDetachShMem(void *punt);
```

Detach a previously attached shared memory area, making it unavailable for this process. This procedure do not destroys the shared memory area.

Punt	Pointer to attached shared memory area

Returns: 0 if memory is successfully detached, -1 otherwise

Destroy shared memory area

```
int iDelShMem(key_t SHmKey);
```

Destroys a shared memory area identified by its key.

SHmKey	Key that identifies the shared memory area to be assigned.

Returns: 0 if memory is successfully destroyed, -1 otherwise

Semaphore procedures

Use: lc_sem.h

Create Semaphore

```
int iCreateSem(key_t SemKey);
```

Creates and attach a semaphore for process synchronization. Semaphore status after creation is unlocked (value 1).

SemKey Key for semaphore creation

Returns: semaphore id (sem_id) if sussesfully, or -1 on errr

Attach semaphore

```
int iAssignSem(key_t SemKey);
```

attach a previously creates semaphore.

SemKey Key for semaphore creation

Returns: semaphore id (sem_id) if sussesfully, or -1 on errr

Deletes semaphore

```
int iDelSem(int sem_id);
```

Delettes an attached semaphore.

sem_id semaphore id returned by iAssignSem or iCreatteSem.

Returns: 0 on successfully deleted semaphore, -1 on error

Unlock semaphore

```
int iUnlockSem(int sem_id);
```

Unlocks semaphore, allowing the first waiting process enter in critic area.

sem_id semaphore id returned by iAssignSem or iCreatteSem.

Returns: 0 on successfully unlocked semaphore, -1 on error

Lock semaphore

```
int iLockSem(int sem_id);
```

Locks semaphore, avoiding any other process enter in critic area. If semaphore is locked by other process, calling process is suspended until the locking process unlocks the semaphore.

sem_id semaphore id returned by iAssignSem or iCreatteSem.

Returns: 0 on successfully locked semaphore, -1 on error
Note: Process will wait until semaphore is unlocked, if previosly was locked by another process.