**CSS 551 - Operating System Design and Implementation - Project 2**

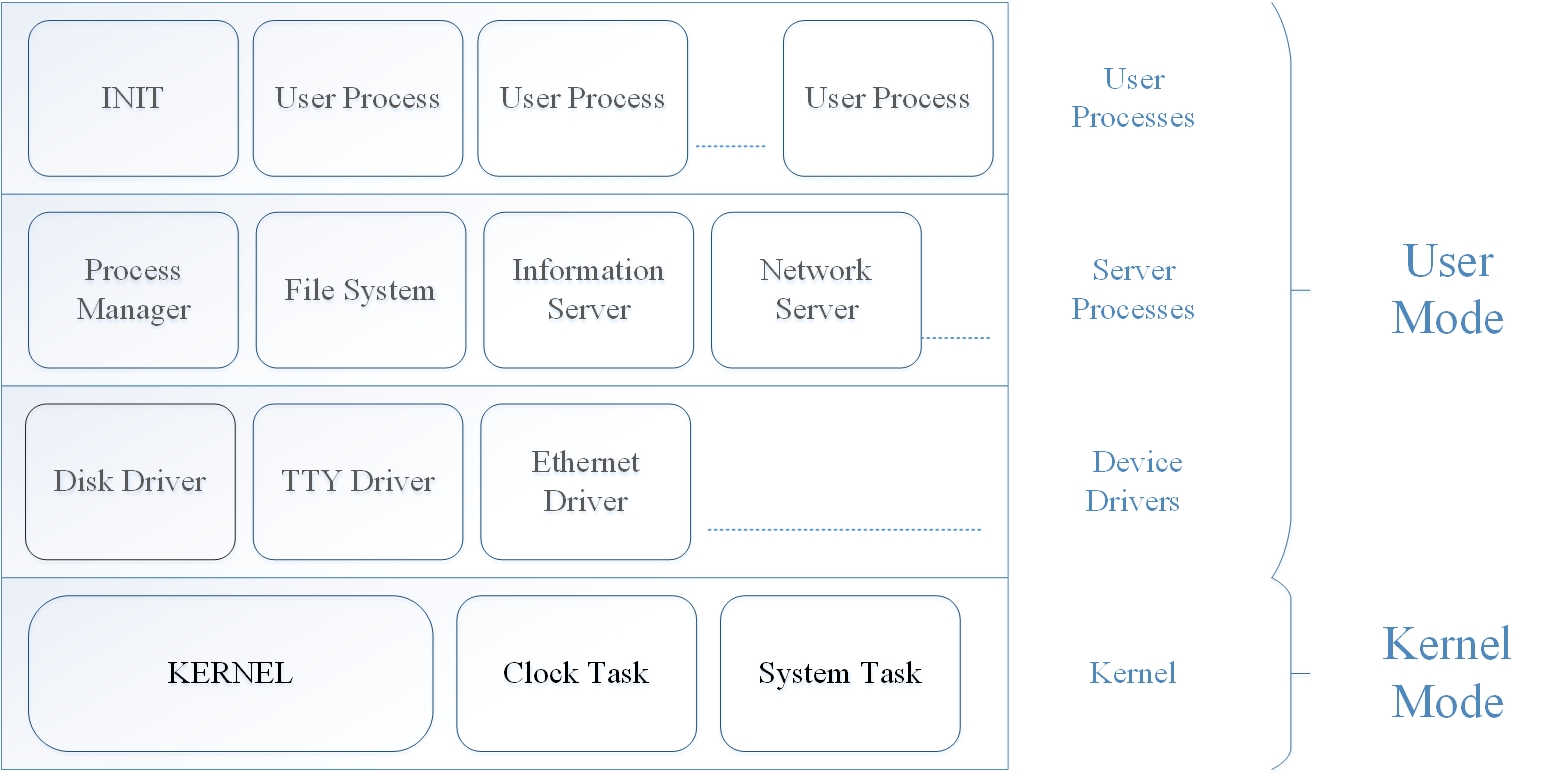
**IPC: Message Passing System Calls**

**Prepared by:**

**KIRUBHA SHANKAR MURUGESAN (A20345391)**

**Contents**

1. MINIX Architecture and our API.
2. Implementation
   1. Architecture
   2. Design for system calls.
   3. Exception Handling.
   4. Reasons for making system calls non-blocking.
   5. Deadlock Handling.
   6. Message ordering in design.
3. Test Cases.
4. **MINIX Architecture and our API.**



The above image shows the architecture of the MINIX3 Microkernel.

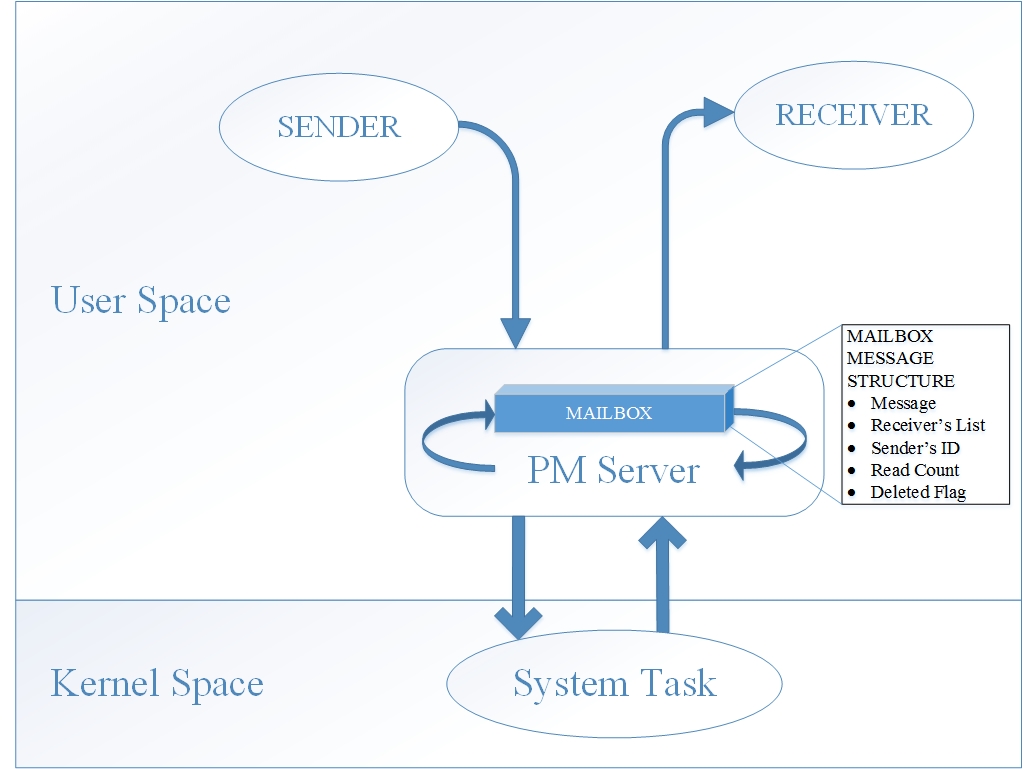
In this project we have created a simple interface which is used to communicate user process. Following table discusses the functions we have used for our design.

|  |  |  |  |
| --- | --- | --- | --- |
| API | function | Error code | Success code |
| **int** **mq\_open**(char que\_name[]) | This function calls to create a queue with the given name or returns the qid if the queue already exists. | No error code | 1 |
| **int** **sender\_search**(**int** sid, **int** regnum) | This function calls search\_sender() system call. | -1,1 | 0 |
| **int** **reciver\_search**(**int** rid, **int** regnum) | This function calls search\_reciver () system call. | -1 | 0 |
| **int** **mq\_send**(char q\_name[],**int** sendID, **char** recvID[], **char** msg[]) | This function calls mq\_send() system call. | QUEUE\_OVERFLOW | 1 |
| **int** **mq\_recive**(char q\_name[],**int** recvID, **char** \*msg) | This function calls mq\_receive() system call. | 0 | A positive integer |
| **int get\_attr(int val)** | This function calls mq\_get system call. | No error code | 0 |
| **int** **mq\_close**(**int** val) | This function calls mq\_close() system call. | No error code | 0 |
| **int** **set\_attr**(int val) | This function calls mq\_set() call, | No error code | 0 |
| **int mq\_reqnotify()** | This function calls to the mq\_reqnotify |  |  |

1. **IMPLEMENTATION**

Project creates a message passing feature for user created process using buffer of size 10. It is designed as follows.

* 1. **Architecture.**



* On executing the program, it will first ask the user to create a new message queue.
* Minix is micro kernel architecture based system so all the process takes place at the servers.
* When a user process want to send a message to the message queue it will first register its ID in sender’s list by calling a search\_sender and search\_reciver system call and if its ID is registered, it will add the message to the queue in pm server by calling the system call mq\_send().
* The PM server handles the queue and defines the system calls. We can clear the queue using mq\_close().
* The receivers receive the messages from the queue on the PM server by calling the system call mq\_reciver() and passing the receivers ID for which the message is to be received.
  1. **Design for System calls.**

|  |  |  |
| --- | --- | --- |
| function | description | Return value |
| int mq\_open(char q\_name]) | Call function que\_list() in messfun.c ,which in turn checks the name if it is already there then the q\_id is returned and if it is not there it calls the queue\_creat() function which, initialize the queue, set space for queue, sender\_list, and recv\_list. It also initialize mess\_cnt, no\_of\_sender, no\_of\_recv to zero | 1 (initialize successfully) |
| int search\_sender(void) | Pass the parameters: senderID, regnum to function add\_sender\_list in messfun.c, which registers a sender to the list | 0 (successful),  -1(reach sender array limit),  1 (sender already exists). |
| int search\_reciver(void) | Pass parameters: receiverID, regnum to function add\_recv\_list in messfun.c, which registers a receiver to the list | 0 (successful),  -1 (reach sender array limit),  1 (receiver already exists) |
| int mq\_send(void) | Add a message to the message queue, pass five paramaters (queue\_name,message,senderID,destination receivers array,number of receivers) to function message\_send in messfun.c, which insert the message into the queue. | 8 (QUEUE\_OVERFLOW) (when there are already 16 messages in the queue), 1(successful) |
| int mq\_receive() | Retrieve a message from the queue, use a while loop to call function message\_receive in messfun.c, which will retrieve one message at a time, and then put the message together and return it to its caller | 9 (MESSAGE\_UNDERFLOW),  0 (successful) |
| int mq\_set() | Sets the message length according to the user input | 0 (successful) |
| int mq\_close() | Clean all the message in the queue, and set mess\_cnt back to 0 | 0 (successful) |
| Int mq\_get() | Prints the current queue length or the feature based on the user input | 0 (successful) |

* 1. **Exception Handling.**

We will discuss the various exception handling code we have implemented throughout the process creation.

**1.** Register a sender.

* If the sender has already registered, the system will give a duplicate register prompt.
* If the sender array has already reached the maximum size, then the system will give a prompt that queue has reached its maximum size.

**2.**  Register a receiver.

* If the receiver has already registered, the system will give a duplicate register prompt.
* If the receiver array has already reached the maximum size, then the system will give a prompt that queue has reached its maximum size.

**3.**    Send a message

* If the sender hasn’t registered to the sender array, then the system will give a prompt that sender is not registered, and it will automatically register for the user.
* If the number of receivers of the send is greater than 3, then it will accept the first three receivers and ignore the rest.
* If the content in the message is larger than the user set length or default length, then the system will give a prompt that message length has reached its maximum size.
* If the queue size has reached its maximum size (i.e. 10), the system will give an exception that the queue has overflowed.

**4.**   Receive a message

* If the receiver hasn’t registered in the receiver array, then the system will give an error that receiver is not registered.
* If the queue is empty, the system will give a prompt that message underflow has occurred.
* If there’s no message for the current receiver, then the system will give a prompt that there’s no message for the current receiver.
  1. **Reasons for making system calls non-blocking.**

The detection and recovery of deadlock is very tedious task. The deadlock condition occurs when the all the processes tries to send or all the processes tries to receive at the same time and the system call is blocking. So in case the deadlock is encountered the system can either detect and recover from it or avoid it. For this it has to implement the algorithms which to handles the deadlock condition such as safe state or banker’. This creates an overhead on the system. But in our case to avoid the deadlock handling overhead we use non-blocking system-calls so that there is no overhead of deadlock handling mechanisms on the system.

* 1. **Deadlock Handling.**

The system calls implemented in this project are of non-blocking nature and so there is no deadlock at any point. The sender sends the message to queue and on the other end the receiver register retrieves it. There are chances that buffer is full and the sender is unable to send messages to queue since there is no space in queue. For the same reason the receiver register shall not be able to retrieve message and as message is reached queue message. As message will be read its deleted and for this reason there shall be no deadlock and we do not need deadlock detection mechanism in our design.

* 1. **Message ordering in design.**

The data structure used to store messages is a single dimensional array. This structure has the following information: a) Message b) Receiver’s List c) Sender’s Process ID d) Receiver Process ID e) Number of reads done f) Deleted flags.   
The receiver begins the retrieving of processes from index zero till end of sequence in loops. This makes message reception in same sequence as loop transverses.

1. **Test Cases.**

Following are the test cases that we have tested on the program.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NO. | Test case description | Expected results | Test results | status |
| 1 | Register sender1 | SUCCESS | SUCCESS | PASS |
| 2 | Register reciver2 | SUCCESS | SUCCESS | PASS |
| 3 | Send a message: sender1, receiver2, message:m1 | SUCCESS | SUCCESS | PASS |
| 4 | Sender created automatically if not there | SUCCESS | SUCCESS | PASS |
| 5 | Receiver 2 retrieve message from queue | SUCCESS | SUCCESS | PASS |
| 6 | Clearing after the message is read | SUCCESS | SUCCESS | PASS |
| 7 | Retrieve message (sender1, reciever2,”message1”) twice. | 1st retrieve success, 2nd retrieve no result | 1st retrieve success, 2nd retrieve no result | PASS |
| 8 | Receiver3 retrieve a message(not registered) | Error occur on registering the receiver | Error occur on registering the receiver | PASS |
| 10 | Sender1 send five messages to receiver2(“m1”, “m2”, “m3”, “m4”, “m5”) and receiver2 receives all the message | Get five results in one receive (and in sequence) | Get five results in one receive (and in sequence) | PASS |
| 11 | Sender1, Sender2, Sender3, Sender4 send “m1”, ”m2”, ”m3”, ”m4” respectively to receiver5 and receiver5 receives message. | Get five results in one receive | Get five results in one receive | PASS |
| 12 | register sender1, sender2, sender3, sender4, sender5, | Successfully added | Successfully added | PASS |
| 13 | register sender1, sender2, sender3, sender4, sender5,sender6 | Sender1-5 ok, sender6 sender array full cannot added | Sender1-5 ok, sender6 sender array full cannot added | PASS |
| 14 | Register receiver1, receiver2, receiver3, receiver4, receiver5, | Successfully added | Successfully added | PASS |
| 15 | Register receiver1, receiver2, receiver3, receiver4, receiver5,receiver6 | Receiver1-5 ok, sender6 sender array full cannot added | Receiver1-5 ok, sender6 sender array full cannot added | PASS |
| 16 | Retrieve message when queue is empty | Message underflow | Message underflow | PASS |
| 18 | Sender1 send message “m1” to receiver 4,5,6 | SUCCESS | SUCCESS | PASS |
| 19 | Insert 16 messages in the queue | SUCCESS | SUCCESS | PASS |
| 20 | Insert 17 messages in the queue | While inserting the 17th message, give error that queue is full | When inserting the 17th message, give error that queue is full | PASS |
| 21 | S1🡪 r4,r5,r6 m1  S1🡪 r4,r5,r6 m2  S1🡪 r4,r5,r6 m3  And r4,r5,r6 receive message | Each get three messages in single call and in the same sequence | Each get three messages in single call and in the same sequence | PASS |
| 22 | S1🡪 r4,r5,r6 m1  S1🡪 r4,r5,r6 m2  S1🡪 r4,r5,r6 m3  And r4, r4, r5, r6 receive message | Each get three messages in single call and in the same sequence without redundancy | Each get three messages in single call and in the same sequence without redundancy | PASS |
| 23 | Exit the program | Exit the program | Exit the program | PASS |
| 24 | Register sender1, and register sender1 again | 1st sender registers successfully, 2nd  time gives error that the sender already existed | 1st sender registers successfully, 2nd  time gives error that the sender already existed | PASS |
| 25 | Register reciever1, and register receiver1 again | 1st receiver registers successfully, 2nd  time gives error that the sender already existed | 1st receiver registers successfully, 2nd  time gives error that the sender already existed | PASS |
| 26 | Receiver1 message from queue when there is no message in it. | Underflow in the terminal and prompt that user process has no message to receive in other terminal. | Underflow in the terminal and prompt that user process has no message to receive in other terminal. | PASS |
| 27 | Sender1 send message to receiver4, receiver5, receiver6, receiver7, | Give hint that there can only be three receivers at maximum for each message | Give hint that there can only be three receivers at maximum for each message | PASS |
| 28 | Insert 16 messages in the queue, and retrieve all | SUCCESS | SUCCESS | PASS |
| 30 | Sender1 send a message to receiver2, sender2 send a message to receiver1, and receiver11, receiver2 retrieve the message | No deadlock occur, retrieve successfully | No deadlock occur, retrieve successfully | PASS |
| 31 | Register sender1, sender2, sender3, sender4, sender5,then sender1 again | First 5 successful, but the last register give hint that sender1 already registered | First 5 successful, but the last register give hint that sender1 already registered | PASS |
| 32 | Sender1 send a message that has length of 200 | Max length of message has reached | Max length of message has reached | PASS |