**CSS 551: Operating System Design and Implementation**

**Project 3**

**Protection for the Mailbox IPC**

**Sunny Changediya (A20353568)**

**Parth Desai (A20351426)**

**Chang Liu (A20365835)**

**Chengxi Sun (A20366976)**

**Contents**

1. Secure & Protected System Call
2. Design & Use of System Call
3. Exception Handling
4. Test Cases
   1. **Secure System Call Manual:**

|  |  |  |  |
| --- | --- | --- | --- |
| System Call | Call\_nr | Function | Argument |
| MINIT | 79 | Initialize mailbox & owner List | NA |
| MSENDER | 108 | Register User to mailbox | (UserID, mailbox\_id) |
| USRSEND | 69 | User process send system call | (sendID,msg) |
| USRRECIVE | 70 | User process receive system call | (recvID,msg) |
| MSEARCH | 103 | Search ownerID, usedID, mailbox\_id, mailbox\_name, and map to specific ID | (userID,regnum,mailbox\_id)  (name,regnum,mailbox\_id) |
| REMUSER | 105 | To remove sender/recv from list | (userID, mailbox\_id) |
| MGARBAGE | 56 | To garbage mailbox messages | NA |
| SNAP | 44 | To display messages from mailbox | NA |
| MCREATE | 58 | Create a new Mailbox | (ownerID,name,type) |
| CHANGEP | 97 | Update permission of user | (userID,perm) |
| DENYACCESS | 110 | To block access to user | (userID,new perm) |

* 1. **Design of System Calls:**
* All mailbox system calls are non-blocking. As PM process handles most of the part and it is a monitor process so system call synchronization is achieved within PM server itself.
* Mutual exclusion to access mailbox is achieved through the way system calls are developed. User process is informed of failure to deliver or retrieve message from mailbox and potential loss of messages can be avoided.
  1. **Exception and Deadlock handling:**

1. **Deadlock Condition and handling:**

* Since all system calls are non-blocking, deadlock will not occur. When sender tries to send a message and there is not enough room for messages to be stored, the sender will be informed of message deposit failure or mailbox overflow error. So potential loss of messages can be handled.
* For receivers, when no messages are available they are informed with error mailbox\_underflow and suggested to try again. So now it depends on receiver to work further.
* In both cases, sender and receiver are informed of deadlock errors and terminated instead of keeping them waiting for each other. Due to this busy wait condition can be avoided and CPU time is saved on unnecessary running sender or receiver.

1. **Exception Occurrence and Handling:**

* The worst case is possible that mailbox contains messages for different receiver ID’s and maximum registered receiver ID’s are different than needed. We represent this case as Mailbox\_Halt.

Eg: Mailbox message is for receiver ID: 1, 2, 3, 4. And maximum registered receiver id: 5, 6, 7, 8, 9.

* To recover from mailbox halt, as no messages will be read by any receiver and deposited by sender, we have provided smart HALT\_RECOVER system call which will present admin with information related to receiver’s available and corresponding messages against those receivers. So is informed about pending messages for particular receiver. So user can invoke those receivers first to read their messages making room in mailbox for depositing messages. Further user also provided with option to de-register any receiver no longer needed.
* So potential loss of mailbox garbage or receiver list deletion is avoided with presenting simple information to user. This way we recover from halting.
* Another worst case exception can occur in which sender and receiver not able to send or retrieve messages. We have provided MGARBAGE call with admin privileges to clear sender\_list, receiver-list or complete mailbox to recover from exceptions.
  1. **Test Cases:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NO.** | **Test case description** | **Expected results** | **Test results** | **status** |
| 1 | Register sender1 | Sender registered | SUCCESS | PASS |
| 2 | Register reciver2 | Receiver registered | SUCCESS | PASS |
| 3 | Deposit a message: sender1🡪 receiver2, message:m1 | Message deposited | SUCCESS | PASS |
| 4 | Notify the mailbox | Deposited | SUCCESS | PASS |
| 5 | Receiver2 retrieve message from mailbox | Message received | SUCCESS | PASS |
| 6 | Garbage collect, using right password | SUCCESS | SUCCESS | PASS |
| 7 | Garbage collect, using wrong password | Not authorized to clean the mailbox | Not authorized to clean the mailbox | PASS |
| 8 | Retrieve message (sender1, reciever2,”message1”) twice. | 1st retrieve success, 2nd retrieve no result | 1st retrieve success, 2nd retrieve no result | PASS |
| 9 | 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 mailbox is empty | Mailbox underflow | Mailbox underflow | PASS |
| 17 | Deposit a message using sender8 (not registered) | Hint sender not registered | Hint sender not registered | PASS |
| 18 | Sender1 send message “m1” to receiver 4,5,6 | SUCCESS | SUCCESS | PASS |
| 19 | Insert 16 messages in the mailbox | SUCCESS | SUCCESS | PASS |
| 20 | Insert 17 messages in the mailbox | While inserting the 17th message, give error that mailbox is full | When inserting the 17th message, give error that mailbox 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 retrieve mail from mailbox when there is no mail 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 mailbox, and retrieve all | SUCCESS | SUCCESS | PASS |
| 29 | Snapshot the mailbox when it is empty | Display an empty mailbox | Display an empty mailbox | PASS |
| 30 | Sender1 send a mail to receiver2, sender2 send a mail to receiver1, and receiver11, receiver2 retrieve the mails | 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 |
| 33 | Mailbox is full with 16 messages all for the receiver1 but receiver1 is not registered and the registration list for receiver is full. | Should ask for the administrative calls to handle the halt situation | Asks for administrative calls with two options  1. Garbage the registration list for the receivers.  2. Garbage all the messages in the mailbox. | PASS |

* 1. **Future Enhancements:**

1. HALT\_RECOVER can be used and utilized in much better way so that system can take action appropriately.
2. MGARBAGE can be further developed to provide more useful functionalism.