**CS551 Project 2 IPC System Call Implementation**

**Design Document**

**Team Members**

Balaji A R (A20347964)

Paulo Vitor Becker Ayrosa Monterio de Andrade (A20341532)

Sai Ravali Nannuru (A2035434)

**Flow Chart:**

Flow chart for each system call is placed in the flow-chart folder of the archive.

**User Manual:**

1. int mq\_close(char \*queue\_name)

**Description:**

This system call will check whether queue exist or not. If queue exist, procedure will free the data structures used for this queue. The array which store messages will be cleared. If queue does not exist return -1.

**Exception:**

1. If queue does not exist, program will generate error message saying queue does not exist.
2. In the process of queue clearing, if not able to clear. This system call returns with -1.
3. int mq\_reqnotify(char \*queue\_name, int receiver\_pid)

**Description**:

This system call first check queue exist or not. If queue does not exist return -1. If queue exist and will check receiver pid exist in the system and adds receiver pid to the receiver list for the current queue.

**Exception:**

1. If receiver pid does not exist in the system, generate error message to the user.
2. int mq\_send(char \*queue\_name, char \*message,int receiver\_pid)

**Description:**

The send() function shall attempt to send a message to the message queue, for the receiver id.

The maximum length of the *string* is 30 characters. A single message can be send to multiple number of receivers. Maximum number of receivers can the equal to the size of message queue. It internally uses kernel call sys\_datacopy() to copy the message from calling user process to the pm server’s message queue.

**Exception:**

1) If it is unable to send data to the message queue, i.e. the sys\_datacopy() fails, don’t keep waiting and return the control to the user process, with return value of -1. This makes the system call non-blocking.

2) If the message queue to store messages is full, it shall not wait for the mailbox to become empty. It will return back to the user process.

1. int mq\_receiver(char \*queue\_name, int receiver\_pid)

**Description:**

The mq\_receive system call first check queue exist or not. If queue exist and will check any message exist for the current receiver. If message found for the receiver, retrieve message and return to user process.

**Exceptions:**

1. If it is unable to retrieve data from the message queue, i.e. the sys\_datacopy() fails, don’t keep waiting and return the control to the user process, with return value of -1. This makes the system call non-blocking.
2. If message does not exist for the receiver, the function will generate error message saying message does not exist.
3. int mq\_setattr(char \*queue\_name, int no\_of\_messages, int sys\_call\_type)

**Description:**

System call will search for the message queue to change its properties. If no of messages is greater than the current size of the queue. It will create additional store to store new message and if it is less the size of the queue will get shrink and return.

**Exception:**

1. If memory allocation fails based on the new size the system call will return the control to the user.
2. int mq\_getattr(char \*queue\_name)

**Description:**

System call get queue details like current size of the message queue and type of system call currently supported by this queue

**Exception:**

1. If queue does not exist generate error message and return -1 for queue not found
2. If queue size not able to calculate then return control to calling process and return -1.
3. int mq\_open(char \* queue\_name)

**Description:**

This system call helps user to create a new queue. If queue already exist, the system call will return its queue descriptor. If queue does not exist, system call will create new queue with default size to store 15 messages and return queue descriptor to the user.

**Exceptions:**

Check maximum limit of number of queues created. If max limit is reached generate error message saying queue cannot be created to user.

**System Call Choice made:**

We have designed mq\_send, mq\_retreive system call to be non-blocking. So whenever the user tries to send a message to the receiver. The process will send message to queue and continue its operations. This approach we choose because if process gets blocked after sending the message, the process cannot to do any other work. That’s the main reason we designed as non-blocking.

In order for these calls to become blocking, one option to be held is to the user program to wrap the mq\_send() and mq\_receive() syscalls into a loop, waiting for an appropriate response from the called function and then proceed with normal operation.

This could be implemented also using a flag indicating the type of each of the system calls and implementing inside the functions a few if statements, in order to check for the type of setting and executing the looped (blocking) or non-looped (non-blocking) section of code.

However, in this approach there is a possibility of life lock to happen. This condition happens when the user executes the system call in an infinite loop. The user process can identify life lock and can restore from it by breaking the loop or closing the message queue or retrieve the message from the queue.

**Test Cases**

1. create a queue using mq\_open() system call. Based on return value, check queue details by calling mq\_getattr(). If queue is created system call should output the queue details else queue does not exist message should be returned.
2. Send one message to the queue and try to retrieve a message from the queue. The message should be displayed to the user
3. After receiving the deposited message, call receive once again to check any message exist to get proper message.
4. Run the receive\_notification user process and get its pid. Request the notification for the pid currently created. Send new message to the queue with that receiver pid. The system call will send send signal to the receive\_notificaiton function.
5. Call mq\_close() to close the queue. If queue exist close the message queue else generate error message.
6. We can create two process parallel which sends and receives the message in the queue to check deadlock does not exist.