Assignment 1 – Message Queues

PART – 1

Summary of POSIX Message Queues:

The main functions in the POSIX message queue API are the following:

1 The mq_open() function creates a new message queue or opens an existing queue, returning a message queue descriptor for use in later calls.

2 The mq_send() function writes a message to a queue.

3 The mq_receive() function reads a message from a queue.

4 The mq_close() function closes a message queue that the process previously opened.

5 The mq_unlink() function removes a message queue name and marks thequeue for deletion when all processes have closed it.

mq_open:

- *syntax: mqd_t mq_open(const char *name, int oflag, /* mode_t mode, struct mq_attr *attr */);* name argument identifies the message queue.
- The oflag argument is a bit mask that controls various aspects of the operation of mq_open() Flag Description:

O_CREAT - Create queue if it doesn't already exist
O_EXCL - With O_CREAT, create queue exclusively
O_RDONLY - Open for reading only
O_WRONLY - Open for writing only
O_RDWR - Open for reading and writing
O_NONBLOCK - Open in nonblocking mode

mq_send:

- *Syntax:* **int mq_send(mqd_t** *mqdes*, **const char** ***msg_ptr*, **size_t** *msg_len*, **unsigned** *sg_prio*);

- mq_send() adds the message pointed to by *msg_ptr* to the message queue referred to by the descriptor *mqdes*

- The *msg_len* argument specifies the length of the message pointed to by *msg_ptr*; this length must be less than or equal to the queue's*mq_msgsize* attribute. Zero-length messages are allowed

- The *msg_prio* argument is a nonnegative integer that specifies the priority of this message. Messages are placed on the queue in decreasing order of priority, with newer messages of the same priority being placed after older messages with the same priority.

mq_receive:

- *syntax: ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned *msg_prio);*
- mq_receive() removes the oldest message with the highest priority from the message queue referred to by the descriptor *mqdes*, and places it in the buffer pointed to by*msg_ptr*.
- The *msg_len* argument specifies the size of the buffer pointed to by *msg_ptr*
- If *prio* is not NULL, then the buffer to which it points is used to return the priority associated with the received message

mq_close:

- *syntax: int mq_close(mqd_t mqdes);*
- mq_close() closes the message queue descriptor *mqdes*.
- If the calling process has attached a notification request to this message queue via*mqdes*, then this request is removed, and another process can now attach a notification request.

mq_unlink:

- *syntax: int mq_unlink(const char *name);*
- mq_unlink() removes the specified message queue *name*. The message queue name is removed immediately. The queue itself is destroyed once any other processes that have the queue open close their descriptors referring to the queue.

PART – 2

(The source code implementing all the functions is located at pintos/newprogs/assignment1.c)

Data Structures:

- We used doubly linked lists defined in the lib/kernel/list.h file and all the operations were done on them were using the functions in this file only.
- A message structure was defined keeping record of
  - Message in it (using a string of predefined maximum length)
  - Priority (unsigned)
  - Struct list_elem (defined in the list.h file)
- A doubly linked message list can now be thus formed with the message structure(above) as a node, and this list forms a queue and is contained in queue data structure as below.
- A queue structure is defined keeping a record of:
  - Message list of message structure as defined above.
  - The qid (the unique message queue id) of the queue
  - Name of the queue (using a string of predefined length)
- A queue_id integer defines the total no of message queues present.

Design and interactions:

- mq_open:
  - As new message queues are created they get descriptor ids in increasing order(making use of the queue_id defined above)
  - If a queue already exists but is closed then we open it again and assign it a descriptor id. (closed queues have a negative message descriptor id)
  - Otherwise an empty message queue is initialized and its corresponding descriptor id returned
- mq_send:
  - Traverses through all the queues and finds a queue with the descriptor id passed in the arguments
  - Traverses through the message queue and finds an appropriate position for the message to be entered
  - The message is entered such that the resulting queue is in decreasing order of priorities and among messages of same priorities older messages appear first
- mq_receive:
  - Traverses through all the queues and finds a queue with the descriptor id passed in the arguments

    - Removes the first message of the queue found and stores the message in the appropriate argument passed.
    - Saves the pointer in the appropriate buffer passed in the argumens if it is not NULL
  - <u>mq_close:</u>
    - Traverses through all the queues and finds a queue with the descriptor id passed in the arguments
    - Closes the message queue found by making its descriptor id negative and hence inaccessible by mq_open and other threads not already using the message queue.
  - <u>mq_unlink:</u>
    - Traverses through all the queues and finds a queue with the descriptor id passed in the arguments
    - Removes the message queue found from the list (double linked) of message queues.

<u>THANK YOU.</u>

Team Members,

SAKAAR KHURANA – 10627

ADITI KRISHN – 10039

NIKHIL AGGARWAL - 10446