# PUBLISHER-SUBSCRIBER IPC DESIGN DOCUMENT

## MANUAL PAGES

This section describes the man pages of all the system call that we have implemented.

### TOPIC CREATE:

This will allow a process to create an interest group.

***SYNOPSIS*:**

StatusCodes add_topic(char topic_name[]);

***DESCRIPTION:***

Function takes 3 parameters: topic name

This function Returns StatusCode which has an int datatype

**Success**: OPERATION_SUCCESS

**Failure**: TOPIC_NAME_LENGTH_EXCEEDED, TOPICS_FULL, DUPLICATE_TOPIC

***CONDITIONS:***

1) Topic – Maximum number of topic that can be active is 10 and Topic name – Maximum length of topic is 100 characters.

***BLOCKING CONDITIONS:***

NONE

***ALSO SEE:***

TopicPublisher , TopicSubscriber, Publish, Retrieve, Topic Lookup

### TOPICPUBLISHER:

This will allow a process to declare itself a publisher of a specific interest group.

***SYNOPSIS:***

StatusCodes add_publisher_to_topic(char topic_name[], int pubId);

***DESCRIPTION:***

Function takes 3 parameters: topic name, Publisher ID

This function Returns StatusCode which has an int datatype

**Success**: OPERATION_SUCCESS

**Failure**: TOPIC_NOT_FOUND, DUPLICATE_PUBLISHER

***CONDITIONS:***

2) Topic – Maximum number of topic that can be active is 10 and Topic name – Maximum length of topic is 100 characters.
3) Publisher ID – Maximum 10 Publishers can be active and Publishers ID should be a number.

***BLOCKING CONDITIONS:***

NONE

***ALSO SEE*:**

TopicCreate , TopicSubscriber, Publish, Retrieve, Topic Lookup

## TOPICSUBSCRIBER:

This will allow a process to declare itself a subscriber to an interest group

***SYNOPSIS:***

StatusCodes add_subscriber_to_topic(char topic_name[], int subId);

***DESCRIPTION:***

Function takes 3 parameters: topic name, Subscriber ID

This function Returns StatusCode which has an int datatype

**Success**: OPERATION_SUCCESS

**Failure**: TOPIC_NOT_FOUND, DUPLICATE_SUBSCRIBER

***CONDITIONS:***

4) Topic – Maximum number of topic that can be active is 10 and Topic name – Maximum length of topic is 100 characters.
5) Subscriber ID – Maximum 10 Subscribers can be active and Subscriber ID should be a number.

***BLOCKING CONDITIONS*:**

NONE

***ALSO SEE:***

TopicPublisher, TopicCreate, Publish, Retrieve, Topic Lookup

## PUBLISH:

This will allow a publisher to send a message to an interest group

***SYNOPSIS:***

StatusCodes publish_messsage(char topic_name[], char msg[], int pubId);

***DESCRIPTION:***

Function takes 3 parameters: topic name, message, Publisher ID

This function Returns StatusCode which has an int datatype

**Success**: OPERATION_SUCCESS

**Failure**: TOPIC_NOT_FOUND, TOPIC_NAME_LENGTH_EXCEEDED, MSG_BOX_FULL

***CONDITIONS:***

1) Topic – Maximum number of topic that can be active is 10 and Topic name – Maximum length of topic is 100 characters.
2) Message - Maximum message length is 100.
3) Publisher ID – Maximum 10 Publishers can be active and Publishers ID should be a number.

***BLOCKING CONDITIONS:***

If message list if full an error code is returned

***ALSO SEE:***

TopicCreate, TopicPublisher , TopicSubscriber,, Retrieve, Topic Lookup

## RETRIEVE

This will allow a subscriber to retrieve one message from an interest group

***SYNOPSIS:***

StatusCodes retrieve_message(char topic_name[], char **msg, int subsId);

***DESCRIPTION:***

Function takes 3 parameters: topic name, message, subscriber ID

This function Returns StatusCode which has an int datatype

**Success**: OPERATION_SUCCESS

**Failure**: TOPIC_NOT_SUBSCRIBED, TOPIC_NOT_FOUND, MSG_BOX_EMPTY

***CONDITIONS:***

1) Topic – Maximum number of topic that can be active is 10 and Topic name – Maximum length of topic is 100 characters.
2) Message - Maximum message length is 100.
3) Subscriber ID – Maximum 10 Subscribers can be active and Subscriber ID should be a number.

***BLOCKING CONDITIONS:***

If message list if full an error code is returned

***ALSO SEE:***

TopicCreate, TopicPublisher, TopicSubscriber, Publish, Topic Lookup

## TOPIC LOOKUP:

This will allow a process to discover what interest groups are there

***SYNOPSIS:***

StatusCodes lookup_topics()

***DESCRIPTION:***

This function Returns StatusCode which has an int datatype

**Success**: OPERATION_SUCCESS

**Failure**: NONE

***CONDITIONS:***

4) Topic – Maximum number of topic that can be active is 10.

***BLOCKING CONDITIONS:***

NONE

***ALSO SEE:***

TopicCreate, TopicPublisher , TopicSubscriber, Publish, Retrieve

## SYSTEM CALL DESIGN

Below are the files that we have modified/added to design our system calls. All the files are ordered as per the control flow.

Here we added supporting files for system calls into the pm server process. We also added user_ipc.h which acts as an interface/abstraction layer to simplify system call APIs.

- /usr/src/servers/pm/Main.c – This file contains the initialization of topic and the required header for initialization.
- /usr/src/servers/pm/MakeFile – We added topic.c in this file to compile and install in kernel.
- /root/user_ipc.c – This is the file which contains the logic for user actions on selection of menu. It contains the user information codes and contains user action functions.
- /usr/include/user_ipc.h – This file contains the function definition for system calls that are used by user functions.
- /usr/src/include/minix/callnr.h – This contains the system call number as a list of macro definitions.
- /usr/src/servers/pm/table.c – This includes the PM server table entry of system calls.
- /usr/src/servers/pm/proto.h – This file contains all the prototype for our system calls to invoke from user_ipc.h
- /usr/src/servers/pm/misc.c – This contains all the implementation for the system call that was prototyped in proto.h
- /usr/src/servers/pm/topic.h – This contains all the user StatusCodes, DataStructures and function prototype of the topics.
- /usr/src/servers/pm/topic.c – This contains all the implementation of the function that are defined in topic.h. This is the file which contains the main logic of handling topic related actions.

## EXCEPTION HANDLING

| Scenario | Error code | Message |
|---|---|---|
| When maximum count(10) for creating a topic is reached. | TOPICS_FULL | Topic list is Full. |
| When there are no more topics to read. | TOPIC_EMPTY | Topic list is empty. |
| When topic name length is exceeded. | TOPIC_NAME_LENGTH_EXCEEDED | Topic Name Length Exceeded. |
| When topic entered is not available. | TOPIC_NOT_FOUND | Topic Not Available. |
| When user tries to retrieve a message from a topic not subscribed. | TOPIC_NOT_SUBSCRIBED | Topic Not Subscribed |

| | | |
|---|---|---|
| When user tries to create a topic with existing topic name. | DUPLICATE_TOPIC | Duplicate Topic |
| When existing publisher for a topic tries to enroll as a publisher for the topic again. | DUPLICATE_PUBLISHER | Duplicate Publisher |
| When existing subscriber for a topic tries to subscribe for a topic again. | DUPLICATE_SUBSCRIBER | Duplicate Subscriber |
| When message length(100) inside a topic exceeds. | MSG_LENGTH_EXCEEDED | Message Length Exceeded |
| When maximum count for creating messages for a topic is reached. | MSG_BOX_FULL | Message Box Full |
| When there are no more messages for a topic is reached. | MSG_BOX_EMPTY | Message Box Empty |
| When user tries to publish a message to a topic for which he is not enrolled. | PUBLISHER_NOT_REGISTERED | Publisher Not Registered |

## WHY RETRIEVE IS MADE NON-BLOCKING?

If any part of the code is accessing or editing a common global variable, then it is very important to maintain the consistency during it access. Common sharing of resources is one of the strongest coupling and this is seen more commonly when one module tries to write to a globally shared variable and another module tries to read the same global variable. Whenever read/write relationship is seen it is always better to acquire the lock to avoid in-consistency. As we have implemented everything in pm server process Mutual exclusion to access the topic data structure is achieved by the way that we have implemented the system calls and that pm framework itself already manages the serialization and synchronization of calls since it uses monitor.

However, retrieve call is implemented in a non-blocking fashion i.e. when a user program make a retrieve system call a there are no Topics/Messages to send to user requesting the message then, the user is not blocked from doing other operation rather, we will show a user with custom message that says that there is no message to retrieve "Message Box Empty". The main reason behind this decision is that by blocking the user, user time is wasted and in complex program user can spend his value time in a productive way. Also, by giving the proper user information, user understands that there is no message waiting for him/her now. User can check for a new message by periodically check with a certain interval of time. So this design suffices that user has more options and there will be no restrictions from kernel side.

## DETECTION AND RECOVERY FROM DEADLOCK

We have implemented in such a way that the deadlock will not occur. Though, theoretically deadlock is not possible, we can't ignore the possibility of the occurrence of the deadlock when multiple threads are accessing same data. As mentioned in manual pages of the system call, all system calls are non-blocking.

When a user tries to publish a message and there are already reached the maximum size of List (5 as mentioned in the requirement) we show an custom message for user as "Message Box Full". By this publisher will understand that the subscriber has not yet read the message and he needs to keep checking in loop in certain interval of time so that is there any empty slots to publish the message.

Also, when a user tries to retrieve a message and there are no messages to read from the List then, we show a custom message "Message Box Empty" so, that user can understand that there is no message waiting for him for that topic. So, he can loop on to check if there is any message in the queue in certain interval of time.

Since we don't need any synchronization and there are no blocking calls (As said above we have implemented in pm server) deadlock can't occur. Rather than blocking, we inform user to retry.

## FLOW DIAGRAM

# Flow diagram of Publisher-Subscriber IPC