PROJECT 2: PUBLISH AND SUBSCRIBE IPCS

Design Document

CS551: Operating Systems Design and Implementation

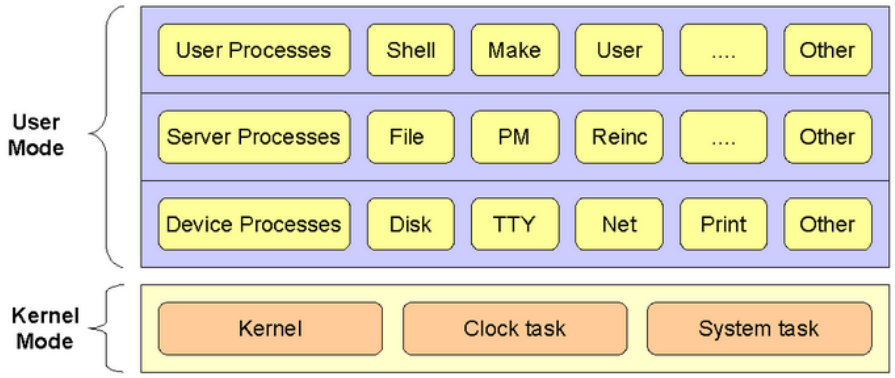
Lakshmi Karle(lkarle@hawk.iit.edu),

Vivek Pabani(vpabani@hawk.iit.edu),

Anirudh Sunkineni (asunkine@hawk.iit.edu),

Illinois Institute of Technology

1. Architecture and API



The MINIX 3 Microkernel Architecture

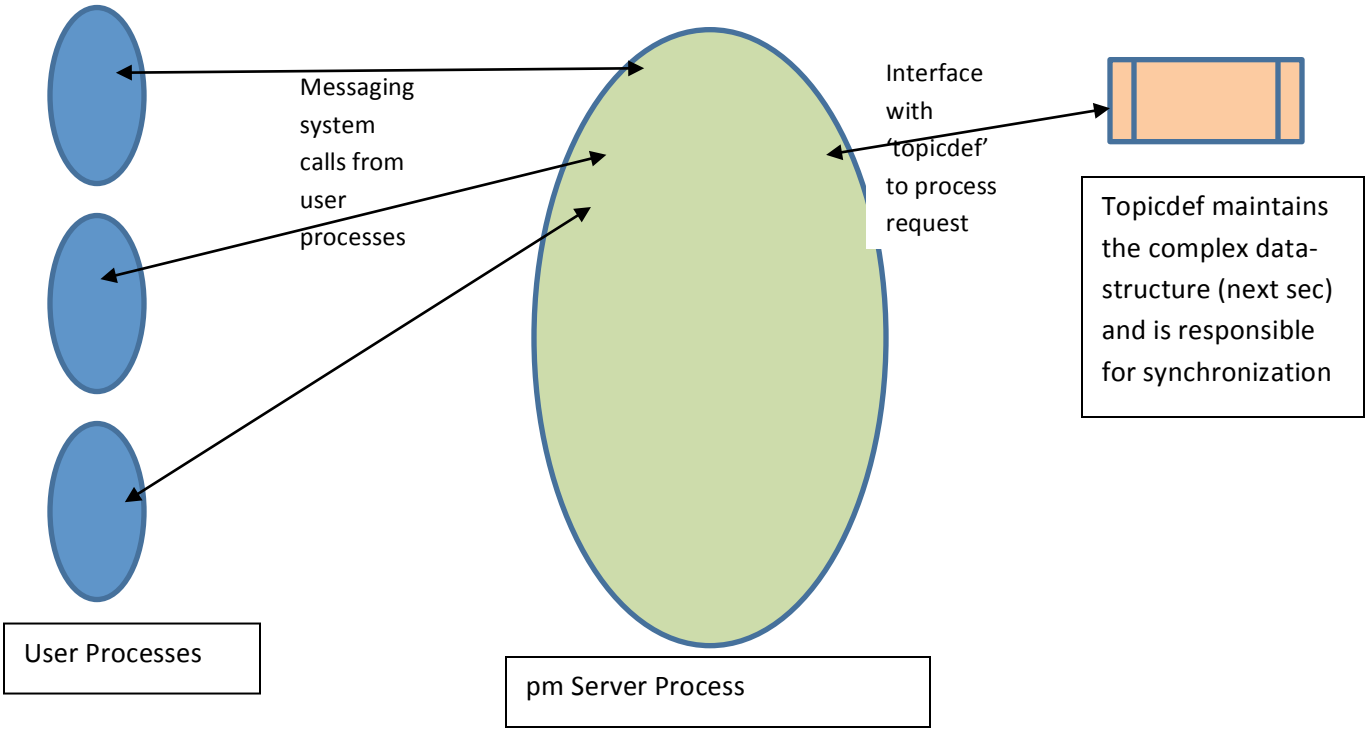
We have implemented the required API for project-2 in the pm server and created another layer of abstraction 'project.h' which works towards providing an even simpler interface for the user process. User processes shall use these API to communicate with other user processes.

Following is the API, the error codes listed as self-explanatory for the most part:

API	ERROR CODES	SUCCESS CODE
int sys_tlookup(char *topics)	No Errors	MESSAGING_SUCCESS; Topics are stored in *topics
int sys_tcreate(char *topic_name)	TOPIC_NAME_LEN_OVERFLOW, DUPLICATE_TOPIC, TOPICS_FULL	MESSAGING_SUCCESS
int sys_tpublisher(int pubId, char *topic_name)	INP_ERR_NULL_TOPIC_NAME, TOPIC_NOT_FOUND, DUPLICATE_PUBLISHER	MESSAGING_SUCCESS
int sys_tsubscriber(int subId, char *topic_name)	INP_ERR_NULL_TOPIC_NAME, TOPIC_NOT_FOUND, DUPLICATE_SUBSCRIBER	MESSAGING_SUCCESS
int sys_tpublish(int pubId, char *topic_name, char *mesg)	INP_ERR_NULL_TOPIC_NAME, MSG_LEN_OVERFLOW_ERROR, TOPIC_NOT_FOUND, BUFFER_OVERFLOW	MESSAGING_SUCCESS
int sys_tretrieve(char *topic_name, char *mesg, int subId)	INP_ERR_NULL_TOPIC_NAME, TOPIC_NOT_FOUND, NOT_SUBSCRIBER_OF_TOPIC, BUFFER_UNDERFLOW	MESSAGING_SUCCESS; Topic name is stored in *topic_name

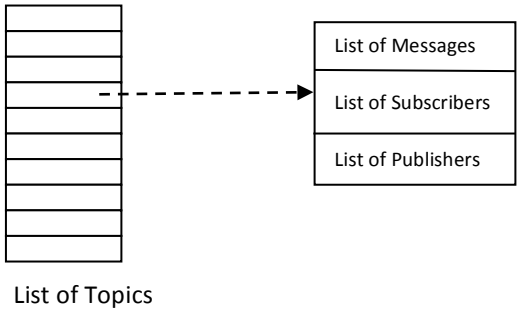
2. MessageServer

2.1 Architecture



- To support publish-subscribe messaging in Minix 3.2.1, we have added the supporting system calls into the pm server process. In addition we have also defined a project.h, which acts as an additional layer of abstraction to simplify the API even further.
- When the pm server starts, it initializes topicdef, which in turn initializes all the underlying data-structures.
- When a user process invokes a messaging system call, the pm process processes it and runs the appropriate function code as specified in table.c
- Depending upon the system call invoked, the function in turn interfaces with topicdef to add a topic, publisher, subscriber, retrieve messages or lookup topics
- The return codes are returned to the caller and in the case of lookup topics and retrieve, the result it copied into the buffer passed in.

2.2 Topics and Messages Data structure



- Topicdef maintains a complex data structure that is represented by the above picture.
- The topic data structure is the backbone of the messaging system calls
- A maximum of 10 topics can be created
- Each topic contains list of 5 messages, list of publishers and subscribers for the topic.
- When a process wants to join as a publisher, the publisher ID is added to the publisher list for that topic and similarly for the join as subscriber request, the subscriber ID is added to the subscriber list. This is done in response to sys_tpublisher() and int sys_tsubscriber() APIs
- Things get more interesting for publish and retrieve requests, when a publisher publishes a new messages, the message is added to the message list (of fixed size 5) for that particular topic. For each message, a list of subscribers who have already read the message is maintained.
- When a retrieve request comes in, the topic is first found and then we search through the message list to see if there are any messages that the requester hasn't received yet. This is tracked through the subscriber list that the message points to. When a message has been delivered to the subscriber, the subscriber is removed from the message's subscriber list. This allows us to perform the following:
    - Make sure that the message is received by all the subscribers [If the subscriber list isn't empty then the message isn't deleted]
    - Make sure that the message is received at most once by any subscriber [When the message is delivered to the subscriber then the subscriber is removed from the delivery list for that message and it isn't delivered on further requests]

2.3 Synchronization and deadlock prevention

BLOCKING CONDITIONS:

API	Blocking Conditions
int sys_tlookup(char *topics)	NONE
int sys_tcreate(char *topic_name)	NONE
int sys_tpublisher(int pubId, char *topic_name)	NONE
int sys_tsubscriber(int subId, char *topic_name)	NONE
int sys_tpublish(int pubId, char *topic_name, char *mesg)	NONE – If message list if full an error code is returned
int sys_tretrieve(char *topic_name, char *mesg, int subId)	NONE – If message list is full an error code is returned

How synchronization is achieved:

- Mutual exclusion to access the topic data structure is achieved by the way system calls are implemented. The pm server whose system call framework we are using already manages the serialization of calls since a server process is a monitor

Why deadlock cannot occur:

- As mentioned in the above table, there are no blocking calls. When a user tries to publish message and there are already max number of messages in the buffer (5 in our case) then system call returns an error. It is up-to the user to handle BUFFER_OVERFLOW error
- Also, when a user tries to retrieve a message and there are no messages to be read, BUFFER_UNDERFLOW error is returned. Again, it is up-to the user to handle this error.
- Since we don't need any synchronization and there are no blocking calls, deadlock cannot occur. Rather than block, we just return a retry-able error code and the user process can decide how to handle it. If the user-process decides to do a busy wait, this can cause a live-lock scenario however.

2.4 Test-cases and results

We have implemented a test.c program, which is an interactive user program that can be used to test the system calls that we implemented as part of the project. The following are the testcases that we have tested and that can be executed using test.c :

	Test-case description	Expected result	Test Result	
1	Create a topic "TOPIC-1"	MESSAGING_SUCCESS	PASS	
2	Create duplicate TOPIC, "TOPIC-1"	DUPLICATE_TOPIC	PASS	
3	CREATE 10 TOPICS with unique topic names	MESSAGING_SUCCESS for all 10 system calls	PASS	
4	Become publisher of "TOPIC_1"	MESSAGING_SUCCESS	PASS	
5	Become publisher of topic 'TOPIC_DOES_NOT_EXIST"	TOPIC_NOT_FOUND	PASS	
6	Become publisher of topic ""- Zero length string	INP_ERR_NULL_TOPIC_NAME	PASS	
7	Try to become publisher of topic "TOPIC_1" again	DUPLICATE_PUBLISHER	PASS	
8	Become subscriber of "TOPIC_1"	MESSAGING_SUCCESS	PASS	
9	Become subscriber of topic 'TOPIC_DOES_NOT_EXIST"	TOPIC_NOT_FOUND	PASS	
10	Become subscriber of topic ""- Zero length string	INP_ERR_NULL_TOPIC_NAME	PASS	

11	Try to become subscriber of topic “TOPIC_1” again	DUPLICATE_SUBSCRIBER	PASS	
12	Publish to an existing topic “TOPIC_1”	MESSAGING SUCCESS	PASS	
13	Publish to an existing topic “TOPIC_1” 5 times	MESSAGING SUCCESS	PASS	
14	Publish to an existing topic “TOPIC_1” 6^{th} time	BUFFER_OVERFLOW	PASS	
15	Publish to an non-existing topic “TOPIC_DOES_NOT_EXIST”	TOPIC_NOT_FOUND	PASS	
16	Publish to topic by specifying NULL topic name	INP_ERR_NULL_TOPIC_NAME	PASS	
17	Retrieve message from an existing topic “TOPIC_1” when there are no messages to be read	BUFFER_UNDERFLOW	PASS	
18	Retrieve message from an existing topic “TOPIC_1” when there are no messages to be read	MESSAGING SUCCESS	PASS	
19	Retrieve message from non-existing topic “TOPIC_DOES_NOT_EXIST”	TOPIC_NOT_FOUND	PASS	
20	Retrieve message by specifying NULL topic name	INP_ERR_NULL_TOPIC_NAME	PASS	
21	Retreive message from topic for which you are not subscriber	INVALID_SUBSCRIBER	PASS	
	Add 5 messages to topic Message 1(“Message1”) subscriber – 1,2,3 Message 2(“Message2”) subscriber – 2,3,4 Message 3(“Message3”) subscriber – 3, 4, 5 Following system calls should be executed in the same order			
22	Subscriber-5 retreives message	Message returned – “Message3” MESSAGING_SUCCESS	PASS	
23	Subscriber-4 retreives message	Message returned – “Message2” MESSAGING_SUCCESS	PASS	
24	Subscriber-2 retreives message	Message returned – “Message1” MESSAGING_SUCCESS	PASS	
25	Subscriber-1 retreives message	Message returned – “Message1” MESSAGING_SUCCESS	PASS	
26	Subscriber-2 retreives message	Message returned – “Message2” MESSAGING_SUCCESS	PASS	
27	Subscriber-3 retreives message	Message returned – “Message1” MESSAGING_SUCCESS	PASS	
28	Subscriber-4 retreives message	Message returned – “Message3” MESSAGING_SUCCESS	PASS	

29	Subscriber-5 retreives message	BUFFER_UNDERFLOW	PASS	
30	List topics	Returns list of topics available MESSAGING_SUCCESS	PASS	

2.5 Future Enhancements

- Messaging API to get snapshot of messages present in based on following criteria.
    - Topic
    - Topic and Subscriber
    - Topic and Publisher

  This will help user process in troubleshooting

- Process acting as a subscriber can exit abnormally. Since the message of topic for which subscriber is subscribed to is not deleted until all subscribers deleted, it is possible that some messages live forever. To avoid this a timestamp can be associated with each message and message should be automatically deleted after a configured time
- Deletion of topic and update of topic name by topic owner might be useful
- Unregister as subscriber and Unregister as publisher will also be useful
- The set of APIs can be made more secure by introducing key-based authentication mechanism when someone tries to become publisher or subscriber.