**Design Document**

*Xiaowei An ( A20346524 )*

*Ambalkar Sachin( A20343863 )*

*Ying ma (A20313747)*

1. **Introduction:**

We have implemented total 8 system calls. These system calls enable the user processes to communicate with each other. This system call implemented by taking reference from Java Messaging Service(JMS). Details for each system call described below.

1. **Architecture:**

|  |  |  |
| --- | --- | --- |
| **User level (main.c)** | Level 3 | **Exception handling service** |
| **Library Header File (usr/include/newSystemcall.h)** | Level 2 |
| **System Call (/usr/src/services/pm/newSystemcall.c)** | Level 1 |
| **Process service** | Level 0 |

We divide our project into 4 levels and another module called Exception handling service. The Level 2, Level 1, Level 0 can call the Exception handling service any time to deal with the error occurs. Level 2 module contains all the system call interface details. Level 1 contains actual implementations of system calls. In the Process service level, we process the data passed from system call level and render the answer to the user.

1. **System Call**

We have added total 8 system calls. We made changes in callnr.h, proto.h and table.c file. And we have added error\_code.h, newSystemcall.c and newSystemcall.h files.

1. **Topic\_create\_SC** : [**TopicCreate**]  
   Input : char[] (Topic Name)  
   Output : int (Return Value)

This system call will create new Topic.   
If topic is created successfully, then it will return SUCCESS (integer value 99)

**Exceptional Handling:**If topic is already presents, then it will return TOPIC\_ALREADY\_EXIST(-5).  
  
To add this system call, following changes are made:  
**callnr.h** : #define TOPIC\_CREATE 35  
**Table.c** **:** do\_createtopic   
**Proto.h** : int do\_createtopic( char[] );

Implementation of “**do\_createtopic**” present in **/usr/src/servers/pm/newSystemcall.c**

1. **Topic\_Display\_Topic** [**TopicLookup**]  
   Input : void  
   Output : int (Return Value)

This system call will display all the topics that are present.   
  
**Exception Handling:**If no topics are present then it will display “Topic data is empty” , error message.  
  
Following entries are added in system files:  
Callnr.h : TOPIC\_DISPLAY\_TOPIC  
Table.c : do\_display\_topic  
Proto.h : int do\_display\_topic(void)

1. **Topic\_publisher\_SC** [**TopicPublisher**]  
   Input : char[] (Topic Name)  
   Output : int (Return Value)

This system call allows a process to declare itself a publisher of a specific interest group.  
  
**Exception Handling:**It gives error if process tries to declare itself as a publisher of a invalid group or group which is not present.

Following entries are added in system files:  
Callnr.h : TOPIC\_PUBLISHER  
Table.c : do\_topic\_publisher  
Proto.h : int do\_topic\_publisher(char[],int)

1. **Topic\_subsriber\_SC**  [**TopicSubscriber**]  
   Input : char[] (Topic name)  
   Output : int (Return Value)

It allows a process to declare itself as a subscriber to an interest group.

**Exception Handling:**It gives error if process tries to declare itself as a subscriber of a invalid group or group which is not present.

Following entries are added in system files:  
Callnr.h : TOPIC\_SUBSCRIBER  
Table.c : do\_topic\_subscriber  
Proto.h : int do\_topic\_subscriber(char[],int);

1. **Topic\_publish\_msg\_SC**  [**Publish**]  
   Input : char[] (Topic Name)  
   Output : int (Return Value)

It allows a publisher to send a message to an interest group.   
 **Exceptional Handling:**It allows only one process to publish message at a time.  
It blocks all other processes, who tries to access critical section of publish process.

Following entries are added in system files:  
Callnr.h : TOPIC\_PUBLISH\_MSG  
Table.c : do\_topic\_publish\_msg  
Proto.h : int do\_topic\_publish\_msg(char[],int);

1. **Topic\_Receive\_msg\_SC**  [**Retrieve**]  
   Input : void   
   Output : int (Return Value)

It allows a subscriber to retrieve one message from an interest group at a time.

**Exceptional Handling:**It gives error message if no message is available in interest group buffer.

Following entries are added in system files:  
Callnr.h : TOPIC\_RECEIVE\_MSG  
Table.c : do\_receive\_msg  
Proto.h : int do\_receive\_msg(void)

1. **Topic\_Display\_publisher:**Input : void  
   Output : int

It displays all the processes details who registered as a publisher.

Following entries are added in system files:  
Callnr.h : TOPIC\_DISPLAY\_PUB  
Table.c : do\_display\_pub  
Proto.h : int do\_display\_pub(void);

1. **Topic\_Display\_subscriber**   
   Input : void  
   Output : int

It displays all the processes details who registered as a subscriber.

Following entries are added in system files:  
Callnr.h : TOPIC\_DISPLAY\_SUB  
Table.c : do\_display\_sub  
Proto.h : int do\_display\_sub(void);

1. **Blocking Conditions:**
2. We have implemented **Peterson’s solution** to handle critical section. **Topic\_publish\_msg\_SC**, this system call allows only one process to access critical section at a time. And keeps all other processes in blocking state until it completes its task.
3. If any process tries to publish message to any topic group whose buffer size is full, then we discard this publish operation and sends “Buffer is full” error message. We find this suitable solution rather than blocking the publish processes until the subscriber should read the message.
4. Our Retrieve system call is non-blocking. We prefer non-blocking implementation because of following reasons.

i. We don’t have any idea about how much time will require for publisher to publish message. So we can’t keep process in busy state.

ii. Non-blocking implementation gives quick response for any process request. And it does not require to manage a lots of resource management operations such as queue.

1. **Conclusion:**

In this project, we have implemented a system call and we know some system level operation. This project has given us a thorough understanding of the system call.