CS 551 Project 3 Report

Group 20

Ryan Attard, Sean Wallace,Velin Sedlarski,Adrian Birylo

November 18, 2013

1 API

1.1 int IGInit()

This function initializes the framework by resetting all variables to that of their default values (namely 0). The information is stored in static global variables, so any process which calls this will reset all of the data for each process. In general, this function need not be called as the default values will be fine for the entirety of execution. This function always returns 0.

1.2 int IGLookup()

This function copies the global storage array for all interest groups back to the calling process. This is done using the `sys_datacopy` system call. This function always returns 0.

1.3 int IGPublisher()

This function registers a process as a publisher within a given interest group. If the group is public then it will be registered. If the group is privet and the process is allowed to register it then it will be registered else it will not be. It takes as argument the process ID of the requesting process and the ID of the interest group it wishes to join. In the event it specifies an invalid interest group, the function returns -1. If the process is already a publisher, then it returns -2. Otherwise, it returns 0 and registers the process as a publisher.

1.4 int IGSubscriber()

This function registers a process as a subscriber within a given interest group. If the group is public then it will be registered. If the group is privet and the process is allowed to register it then it will be registered else it will not be. It takes as argument the process ID of the requesting process and the ID of the interest group it wishes to join. In the event it specifies an invalid interest group, the function returns -1. If the process is already a subscriber, then it returns -2. Otherwise, it returns 0 and registers the process as a publisher.

1.5 int IGAssignGroupLeader()

This function is only called by the super-user to assign who can be the owner of a group. If the interest group does not exist or the process to be assigned does not exist an error of -1 is returned. Otherwise, 0 is returned and the process is assigned as the group leader.

1.6 int IGRemoveGroupLeader()

This function is only called by the super-user to assign who can be the owner of a group. Zero is returned and the process is removed. If the process doesn't belong to the group or the group does not belong -1 is returned and nothing is changed.

1.7 int IGCreateSecureGroup()

This function is called by the group leader to create a secure group. Only allowed publisher and subscribers are allowed to register with this group.

1.8 int IGCreatePublicGroup()

This function is called by the group leader to create a public group. Any publisher and subscribers are allowed to register with this group.

1.9 int IGRemoveGroup()

This function is called by the group leader to remove a group that was created. The function return 0 is success and -1 if the group does not exist and nothing is changed.

1.10 int IGPublish()

This function publishes a message to a specified interest group. It takes as argument the sending PID, the destination process id, and a message represented as a character array. As always, this information is taken from the message passed in from the system call. If the specified interest group is nonexistent, it returns -1. If there are already too many messages waiting, it returns -2. If the requesting process is not a publisher, it returns -3. Otherwise, the message is published to the interest group and returns 0.

1.11 int IGRetrieve()

This is the most complicated function of the group. It takes as argument the requesting PID, the interest group of choice, and a message buffer to be populated by the function. If the specified interest group is invalid, the function return -1. If the requesting PID is not a subscriber, it returns -4. If there are no messages to be retrieved, it returns -3. If all of the messages for the given interest group have already been read by the given process, it returns -2.

If none of these are the case, then the function first checks to see which message to return to the client. It does this by starting in the first index of the messages array present in the interest group structure. Checking one by one, it continues until it either finds a message that has not yet been picked up by this process or until it has exhausted all options.

In the event it finds a message, this message is marked as read by this process and then the function checks to see if all other processes which are marked as subscribers for this interest group have picked up this message. If they have not, the message is simply returned to the client. If they have, the message is removed from the interest group entirely.

2 Design

The design of the IPC system call is such that it was added to the Process Management (PM) service. Below is a block diagram of how the IPC is implemented and what is in the user space and what is a service and in the kernel space.

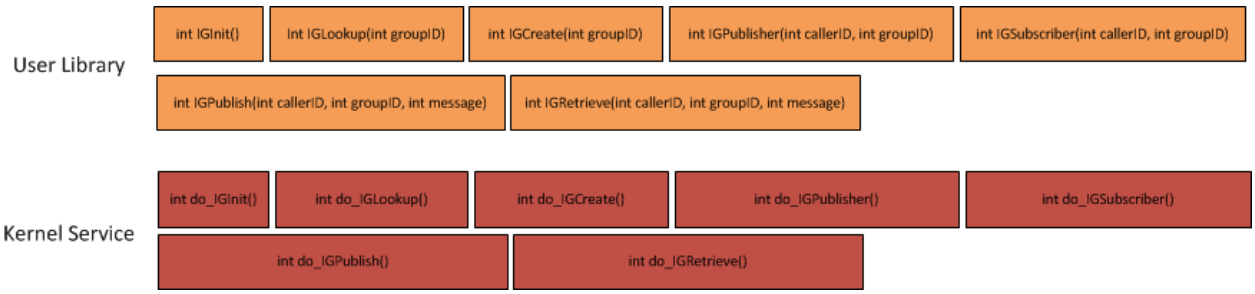

Figure 1: Flow Chart

3 Discussion

In general deadlock can not occur since there are no blocking calls but a poorly written program can go into deadlock. Since there is no blocking calls a process is never waiting to get or put data in the IPC. Since the

calls are not blocking they can fail and it is the responsibility of the programmer to make sure to recover from a failure gracefully and retry.

4 Test Cases

There are a number of test cases which test all aspects of the system:

1. testOverfillIG - Generates n interest groups where n is equal to the max supported. It then tries to generate one more. This should of course fail. If it does, the function returns success. If it fails at any point along the way, a failure notice is generated.

2. testAlreadyPublisher - Creates an interest group, adds a publisher, then tries to add the same publisher once again. If it does not, the function returns success. If it fails at any point along the way, a failure notice is generated.

3. testNotPublisher - Creates an interest group and tries to publish a message without having registered as a publisher first. If the message does not post then it returns success. If it fails at any point along the way, a failure notice is generated.

4. testPublisherInvalidInterestGroup - Creates a publisher and tries to publish to an invalid group. If it does not, the function returns success. If it fails at any point along the way, a failure notice is generated.

5. testAlreadySubscriber - Identical to testAlreadyPublisher except for a subscriber.

6. testNotSubscriber - Identical to testNotPublisher except for a subscriber.

7. testSubscribeInvalidInterestGroup - Identical to testPublisherInvalidInterestGroup except for a subscriber.

8. testRetriveNoMessages - Creates an interest group and a registers a subscriber. It then tries to retrieve when there are no messages that have been posted. If no messages are returned, the function returns success. False otherwise.

9. testAlreadyRetrivedAllMessages - Creates an interest group and registers a subscriber and a publisher and publishes a message. This message is subsequently read by the subscriber a first time then tries to read it a second time. Because there is only one message, no message should be returned on the second attempt. If none is, it returns success, false otherwise.

10. testOverfillMessages - Creates an interest group and a publisher. The publisher then proceeds to publish 5 messages and then tries to publish one more. Since the maximum is 5, the first 5 should succeed and the last should fail. If this is the case, it returns success, false otherwise.

11. testMessageDelete - Creates an interest group, a publisher, and two subscribers. The publisher pushes two messages to the interest group and the subscribers then read these messages. After the messages are read, they are checked to be deleted from the interest group. If so, the function returns success, false otherwise.

12. testMessageOrder - Creates an interest group, a publisher, and two subscribers. The publisher pushes three messages to the interest group and the first subscriber checks all three. The tests checks to make sure the messages received were in order and were not removed (as there is still another subscriber that has not checked these messages yet). The second subscriber then checks all three messages and then the test function checks to make sure that all the messages were removed from the interest group.

13. testSubscribeToSecureGroupAllowed - Creates a secure group, a group leader and a process that can register as a subscriber to the secure group. The subscriber tries to register as a subscriber and is allowed to register.

14. testPublichToSecureGroupAllowed - Identical to testSubscribeToSecureGroupAllowed except for a publisher.

15. testSubscribeToSecureGroupNotAllowed - Creates a secure group. Then try to register a subscribe to the secure group when the process wasn't allowed to belong to the group. The subscribe should not be allowed to register with the group.

16. testPublishToSecureGroupNotAllowed - Identical to testSubscribeToSecureGroupNotAllowed except for a publisher.

5 Running the Code

This is made very easy by a Makefile. The image contains everything that is necessary to run the project. Simply change directories into the "Project3" directory, make, reboot, and then run `./test` inside the Project3 folder.