CSCE 311

Programming Assignment #5

Objective

To implement interprocess communication management in the OSP2 simulator. Ports management is described in chapter 8 of the OSP 2 manual. The java classes to be implemented in this project will deal with the creation and management of ports to support interprocess communion, see sections 3.4 and 3.5 of the Silberschatz textbook for related material.

Required to turn in: Follow all directions regarding hand-in procedures. **Points will be deducted if you do not follow directions precisely.**

1. You must submit an electronic copy of the *.java files that comprise your solution (or your best try) via dropbox.
2. You must document your code. Each java file should include your name and email address at the top of the file.
3. You must provide a one-page explanation of how you accomplished the assignment (or what you have currently and why you could not complete). You should describe your use, creation, and manipulation of data structures to accomplish the assignment.

Building and executing the simulation

Download the archive file containing the files for this project.
For unix or linux or Mac: If you are using a unix or linux machine then you will probably want to download the tar file: Ports.tar. Download the archive and extract the files using the command
tar –xvf Ports.tar

For a windows box: If you are using a windows box, then you will probably be happier with a zipped folder: **Ports.zip**. Download the archive and then extract the files by right-clicking on the file and selecting the "extract all…" option from the popup menu.

You should have extracted the following files:

```
Ports/Demo.jar
Ports/Makefile
Ports/Misc
Ports/OSP.jar
Ports/Message.java
Ports/PortCB.java
Ports/Misc/params.osp
Ports/Misc/wgui.rdl
```

As per the discussion in the OSP2 text, `Makefile` is for use in unix and linux systems. The demo file `Demo.jar` is a compiled executable. The only files you should have to modify are `Message.java` and `PortCB.java`. Modifying the other files will probably "break" OSP2. Compile the program using the appropriate command for your environment (unix/linux/windows).

```
(unix)        javac -g -classpath .:OSP.jar: -d . *.java
(windows)     javac -g -classpath .;OSP.jar; -d . *.java
```

This will create an executable called `OSP`. Run the simulator with the following command:

```
java -classpath .;OSP.jar osp.OSP
```

Interprocess Communication in OSP

There are two classes involved in interprocess communication that have methods that you are required to implement:

Message – (section 8.3 page 144) this is by far the easiest class. All you need to do is implement the constructor. For the purposes of this project, all you need do is call `super(length)`.

PortCB – (section 8.4 pages146-150) this is a little more involved than the `Message` class, but not much more. You must implement:

1. **`PortCB()`–** Simply call super(). See page 156 of OSP2 text.

2. **`init()`–**As in all OSP2 projects, the `init()` method is only called once, at the beginning of the simulation to allow you to initialize any data structures or static variables that you are using that require initialization.

3. **`do_create()`–** This method is responsible for creating, initializing, and associating a PortCB object with a Task. Based on the description on pages 146-147, you should follow these steps:
   a) create a new PortCB object using the constructor.
   b) Next assign the new PortCB object to the current task.
      i. Get the current Task.
      ii. Verify that the current task does not already have the maximum number of Ports allowed. The maximum number is defined by the global constant **`MaxPortsPerTask`**. If it already has the maximum number then return the **`null`** object.
      iii. Use the TaskCB method addPort() to assign the PortCB object to the current Task. If the addPort() returns FAILURE, then return the **`null`** object.
   c) Use the setTask() method to set the owner of the PortCB object.
   d) Use setStatus() to set the status of the PortCB object to `PortLive`.
   e) Finally, it is up to you to keep track of the state of the port's message buffer, i.e., how much of the buffer is used. Presumably, you need to create a variable and initialize it to keep track of this. The size of the message buffer is defined by the global constant **`PortBufferLength`**.

4. **`do_destroy()`–** In order to destroy are PortCB object, you must:
   a) Set the status of the PortCB object to `PortDestroyed`.
   b) Notify the threads that might be waiting for an event associated with the port using the event method `notifyThreads()`.
   c) Remove the port from the Task's table of active ports via the TaskCB method

`removePort()`.

d) Set the owner to the port to null using the PortCB method `setTask()`.

5. **do_send()–**
   a) Start by verifying that the message to be sent is well-formed. The Message object `msg` is considered not to be well-formed if
      i. the object is null or
      ii. the length of the message (msg.getLength) is larger than the port buffer length.
   b) If it is not well-formed then return FAILURE.
   c) Next, create a new system event using the constructor SystemEvent(name). The argument to the constructor is a string used to distinguish the event from other events in the system log (see page 99).
   d) Suspend the current thread on the system event created in the previous step. As suggested in the OSP2 manual, you should read section 4.3 (paragraph 3 of page 64 in particular) to understand that this does not block the thread but instead causes it to change from a user thread to a system thread.
   e) Okay, now comes the tricky part. You need to check if there is room in the Port Buffer to accept this message. Remember the discussion in the do_create() method about keeping track of how much of the buffer is free. If there isn't enough room, then you must suspend the thread on the port, i.e., the port object '`this`' is the argument to the thread `suspend()` method. This thread will be woken up when another thread frees up space in the buffer by doing a `receive()` operation. When this thread wakes up, it should again check if there is now enough space in the buffer. If not, then it needs to suspend itself again. So the pattern is to loop:
      i. checking for room in the buffer and
      ii. suspending until there is finally enough room.
   f) There are two further complications: the thread could have been killed while it was asleep and/or the port could have been destroyed. So in addition to checking if there is room in the buffer, you should also check that the port status is still `PortLive` and also that the thread status is not `ThreadKill`.
      i. If the thread status is `ThreadKill` then remove the thread from the port using the removeThread() method and return FAILURE.
      ii. If the port status is not `PortLive`, then wake up all threads suspended on this port using the event method `notifyThreads()` relative to the `SystemEvent` object you created in step C) and return FAILURE. For example, if you assigned the `SystemEvent` object to the variable myPortEvent, then the call would be `myPortEvent.notifyThreads()`.
   g) At this point, you've determined that there is enough room in the Port buffer for the message and both the port and the thread are still alive. You now insert the message into the port buffer using the `appendMessage()` method.
   h) If the port buffer was previously empty, then notify any possible waiting threads using the `notifyThreads()` method relative to the port object, *i.e.*,

`this.notifyThreads()`.

- **i)** Since you are keeping track of how much free space there is in the buffer, be sure to update this value after appending the message to the buffer.
- **j)** Finally, call `notifyThreads()` on the `SystemEvent` object you created in step C) and return SUCCESS.

6. `do_receive()`–

- **a)** Start by getting the current thread (use `PTBR` as in previous projects).
- **b)** Next, verify that the task that owns the thread also owns the port. (Both the ThreadCB class and PortCB class have a getTask() method.) If this is not the case, then the receive operation is not allowed and you should return `null`.
- **c)** Just as in the `do_send()` method (step C in the above write up), you need to create a `SystemEvent` object to suspend the Thread `T`. As explained in step `D` of the `do_send()` method, this does not block the thread but instead causes it to change from a user thread to a system thread.
- **d)** Suspend the current thread on the system event created in the previous step. As previously explained, this does not block the thread but instead causes it to change from a user thread to a system thread.
- **e)** Okay, now comes the tricky part. You need to check if there is a message in the Port Buffer (use the `isEmpty()` method). If there is no message then you must suspend the thread on the port. Use the port object '`this`' as the argument to the thread `suspend()` method. This thread will be woken up when another thread sends a message to the Port. When this thread wakes up, it should again check if there is now a message in the buffer. If not, then it needs to suspend itself again. So the pattern is to loop:
  - **i.** checking for a message in the buffer and
  - **ii.** suspending until there is finally a message.
- **f)** As in `do_send()`, there are two further complications: the thread could have been killed while it was asleep and/or the port could have been destroyed. So in addition to checking if there is a message in the buffer, you should also check that the port status is still `PortLive` and also that the thread status is not `ThreadKill`.
  - **i.** If the thread status is `ThreadKill` then remove the thread from the port using the `removeThread()` method, notify other threads using the event method `notifyThreads()` relative to the `SystemEvent` object you created in step C) and return `null`.
  - **ii.** If the port status is not `PortLive`, then wake up all threads suspended on this port using the event method `notifyThreads()` relative to the `SystemEvent` object you created in step C) and return `null`.
- **g)** Assuming that the Thread was not killed and that the Port is was not destroyed, then:
  - **i.** use the method `removeMessage()` to get the `Message` object. Be sure to assign this to a variable so that you can return it from this method.
  - **ii.** Update the length of the buffer. Recall that in step e) of do_create() that you initialized a variable you defined in order to keep track of buffer space

in the port. The size of the message that was received in this method is given by the `Message` method `getLength()`.

`iii.` wake up all threads suspended on this port using the event method `notifyThreads()` relative to the Port object, *i.e.*, `this.notifyThreads()`

`iv.` execute a `notifyThreads()` relative to the `SystemEvent` object you created in step C) and return the Message object that you got in step `i)` above.

Standard Tips

- This is assignment is more difficult than thread scheduling. It is advisable that you start as soon as possible.
- Seek help from me if you need it. You may discuss the project with others but you may not share code.
- Minimize the amount of code you write. It will not affect your grade (unless poorly commented), but will reduce the complexity of your solution and make it easier to debug.

How do I get an A on this assignment?
This is very simple to do. Implement the methods as described in this document and hand in everything specified in the "Required to turn in" section on time. It should also be well documented so the grader can understand your implementation decisions. Your solution should complete successfully with no warnings or aborts of the OSP2 simulator.

Turning in Your Assignment via dropbox
When you are satisfied that your implementation of the PORTS project works, use dropbox to submit your project. Use dropbox to submit the following files:

1. `Message.java`
2. `PortCB.java`
3. `One-page explanation of how you accomplished the assignment (or what you have currently and why you could not complete).`

Remember to turn your assignment in on time. **Late assignments will be penalized 10% per day.**