**CIS 657: Principles of Operating Systems**

**Lab4 Devices – Group 6**

- Bharath Darapu

- Avinash Peddi

In this lab project we are assigned to finish the following files:  
1. Device.java

2. IORB.java

3. DiskInterruptHandler.java

In this module the Device class is used to implement scheduling of I/O requests, DiskInterruptHandler implements the interrupt handler for I/O devices, and IORB implements the input/output request block.

**Approach used:**

The approach used is pretty simple and was as described in the book (OSP2 approach by Michael Kifer).

Let me explain the implementation in the order of occurrence in the program.

**Class IORB**

The first thing this class does is calling super() with the same set of arguments namely, ThreadCB thread, PageTableEntry page, int blockNumber, int deviceID, int ioType, OpenFile openFile.

*super(thread, page, blockNumber, deviceID, ioType, openFile);*

**Class Device**

This class implements the I/O scheduler and performs other functions, such as starting I/O operations on devices.

**public static void init():** used to initialize static variables

*//nothing in code*  
**public Device(int id, int numberOfBlocks)**: call super(id,numberOfBlocks) and then initialize the device object and variable iorbQueue.

*super(id, numberOfBlocks);*

*this.iorbQueue = new GenericList();*

**public int do enqueueIORB(IORB iorb):** Executed on a device object and puts iorb on the waiting queue of that device.

1. We need to lock the page associated with the iorb using the lock() method of class PageTableEntry.

*if (iorb.getPage().lock(iorb) == FAILURE) return FAILURE;*

1. We need to increment the IORB count of the open file handle associated with iorb.   
   *iorb.getOpenFile().incrementIORBCount();*
2. We must set the iorb’s cylinder, using the method setCylinder(), to the cylinder that contains the disk block mentioned in the IORB  
   *iorb.setCylinder(iorb.getBlockNumber());*
3. We check that the thread that requested the I/O is still alive, If the thread has died, the method do enqueueIORB() should return FAILURE. If the thread is alive and the device is idle, we can start the I/O operation immediately using the method startIO() on the device object and passing it the iorb as a parameter. If the device is busy, then put the iorb on the device queue and exit by returning SUCCESS.

*if (iorb.getThread().getStatus() == ThreadKill) return FAILURE;*

*//if device is busy*

*if (this.isBusy()) {*

*//put the iorb on the device queue*

*this.iorbQueue.append(iorb);*

*return SUCCESS;*

*} else {*

*//if device is idle and thread is alive then we start the I/O operation immediately*

*this.startIO(iorb);*

*return SUCCESS;*

*}*

**public IORB do dequeueIORB()**This method selects an IORB from the device queue according to some scheduling strategy, deletes it from the queue, and returns the selected IORB. If the queue is empty, null is returned.

*if (this.iorbQueue.isEmpty()) return null;*

*else return (IORB) this.iorbQueue.removeHead();*

**public void do cancelPendingIO(ThreadCB thread)** For each IORB associated with thread found in the queue, we must unlock the buffer page used by that IORB. we must decrement the IORB count of the open file handle associated with the IORB. Finally, we should try to close the open file handle associated with the IORB. One of the places where closePending flag should be checked is the do cancelPendingIO() method.

*for (int i = this.iorbQueue.length(); i >= 0; ++i) {*

*//get the IORB from the queue*

*IORB iorb = (IORB) this.iorbQueue.getAt(i);*

*if (iorb.getThread().equals(thread)) {*

*//unlock the page, can also be done in the device input handler but since we are removing it from the queue here service it here*

*iorb.getPage().unlock();*

*//decrement the IORB counter*

*iorb.getOpenFile().decrementIORBCount();*

*//close the open file handle associated with the IORB*

*if (iorb.getOpenFile().closePending && iorb.getOpenFile().getIORBCount() == 0) iorb.getOpenFile().close();*

*//finally remove it from the queue*

*this.iorbQueue.remove(iorb);*

*}*

*}*

**Class DiskInterruptHandler:**

**public void do handleInterrupt()**1. Obtain information about the interrupt from the interrupt vector, class InterruptVector.

*IORB iorb = (IORB) InterruptVector.getEvent();*

2. The IORB count of the open file handle associated with the IORB must be decremented using decrementIORBCount()

*iorb.getOpenFile().decrementIORBCount();*

3. If the open file has the closePending flag set and the IORB count is 0, the file might need to be closed. The IORB count of a file handle can be obtained via the method getIORBCount().

*if (iorb.getOpenFile().closePending && iorb.getOpenFile().getIORBCount() == 0)*

*iorb.getOpenFile().close();*  
4. The page associated with the IORB must be unlocked, because the I/O operation (due to which the page was locked) is over.

*iorb.getPage().unlock();*

5. If the I/O operation is not a page swap-in or swap-out, then, unless the thread that created the IORB is dead, we need to set the frame associated with the IORB’s page as referenced using the method setReferenced() of FrameTableEntry. In addition if it was a read operation (I/O type FileRead) then the frame must be set dirty (using the method setDirty() of FrameTableEntry).

*iorb.getPage().getFrame().setReferenced(true);*

*//if iotype is write*

*if (iorb.getIOType() == MemoryWrite)*

*//then we set the frame as clean*

*iorb.getPage().getFrame().setDirty(true);*

6. If the I/O was directed to the swap device and the task that owns the thread and the IORB is alive, we should mark the frame as clean (setDirty(false)).

*if (iorb.getThread().getStatus() != ThreadKill)*

*//then set the frame as clean*

*iorb.getPage().getFrame().setDirty(false);*

7. If the task that owns the IORB is dead (status TaskTerm) and the frame associated with the IORB was reserved by that task (verifies using getReserved()), we must unreserve the frame using setUnreserved().

*if (iorb.getThread().getTask().getStatus() != TaskLive)*

*//then set the frame of the page as unreseerved, for the thread's task*

*iorb.getPage().getFrame().setUnreserved(iorb.getThread().getTask());*

8. The threads waiting on the IORB must be waken up by a call to notifyThreads()

*iorb.notifyThreads();*

9. The device must be set to idle using the method setBusy() with the appropriate flag.

*Device dev = Device.get(iorb.getDeviceID());*

*dev.setBusy(false);*

10. The device must be told to service a new I/O request. This IORB is picked up using the method dequeueIORB(). If it returns a non-null object, the device should be restarted with that IORB using the method startIO().

*iorb = dev.dequeueIORB();*

*if (iorb != null) {*

*dev.startIO(iorb);*

*}*

11. Finally, a new thread must be dispatched using the method dispatch() of ThreadCB.  
 *ThreadCB.dispatch();*

After all the implementation we navigated to the project path and

* Compiled the .java file using ‘javac –classpath .;OSP.jar \*.java’ and
* Later used ‘java -classpath .;Demo.jar osp.OSP’ command to run the simulation.

OSP.log file which was generated during the execution is also attached.