Operating Systems Design Project 2 Design Document

Task I

(30%, 125 LINES) IMPLEMENT THE FILE SYSTEM CALLS (CREATE, OPEN, READ, WRITE, CLOSE, AND UNLINK, DOCUMENTED IN SYSCALL.H). YOU WILL SEE THE CODE
FOR HALT IN USERPROCESS.JAVA; IT IS BEST FOR YOU TO PLACE YOUR NEW SYSTEM CALLS HERE TOO. NOTE THAT YOU ARE *NOT* IMPLEMENTING A FILE SYSTEM; RATHER, YOU ARE SIMPLY GIVING USER PROCESSES THE ABILITY TO ACCESS A FILE SYSTEM THAT WE HAVE IMPLEMENTED FOR YOU.

Implementation

First off we're going to have to implement unloadSections() in order to get all the rest to work. unloadSections() frees up the specified page out of the page table.

```
for each translation entry in the page table
      take each entry out of the page table
      if that entry is valid
            free that page
```

handleSysCall() takes five ints as parameters, here we're calling them syscall, a0, a1, a2, a3. What happens here is based on the inputs, this function figures out which further functions to direct the process to. The way that the required function is narrowed down is by comparing syscall to some predetermined variables. Now it's kind of hard to give pseudocode here without making it be actual code, but here we go:

```
switch (syscall)
case syscallHalt
      return handleHalt()
case syscallExit
      return handleExit(a0)
case syscallExec
      return handleExec(a0, a1, a2)
case syscallJoin
      return handleJoin(a0)
case syscallCreate
      return handleCreate(a0)
case syscallOpen
      return handleOpen(a0)
case syscallRead
      return handleRead(a0, a1, a2)
case syscallWrite
      return handleWrite(a0, a1, a2)
case syscallClose
      return handleClose(a0)
case syscallUnlink
      return handleUnlink(a0)
```

```
otherwise
      set an alarm saying there's an unknown syscall
```

handleCreate() takes an integer (name) as a parameter, and from there a file from the VM and puts it into the storage. After putting it into storage, the filename is inserted into the fileTable.

```
read the filename from virtual memory (looked up from name)
open the filename (put it into a variable named file)
if file isn't null
      for (2 to the length of fileTable)
            if fileTable at that spot is empty
                  insert file into fileTable at that spot
                  return i
```

handleOpen() takes an integer (name) as a parameter. With that it does the exact same thing as handleCreate() did. As far as the filesystem is concerned they are the exact same thing.

handleRead() takes three ints as parameters (fd, buffer, and s). It takes a file out of storage and moves it into the VM.

```
grab entry from fileTable at spot fd
if the file is null
      return -1 (we have a problem)
byte[] tempbuff = array of bytes[s]
int readSize = read a file(tempbuff, 0, s)
write into VM (buffer, tempbuff)
return readSize
```

handleWrite() takes three ints as parameters (fd, buffer, s). This one takes a file from VM and moves it into storage.

```
grab entry from fileTable at spot fd
if the file is null
      return -1 (we have a problem)
byte[] tempbuff = array of bytes[s]
read from VM(buffer, tempbuff)
return file.write(tempbuff, 0, s)
```

handleClose() Takes a file descriptor (int). Call close() on that location in your file table and set it to null.

```
grab entry from fileTable at spot fd
if the file is not null
      call close on fileTable[fd]
      set fileTable[fd] to null
return 0
```

handleUnlink(): Takes a file descriptor (int). From here send error notifiers if the file system can not successfully unlink the file.

```
try
      String name = read from VM(fd, STRINGSIZE)
      if(name is null or its length is 0)
```

```
            return -1 (we have a problem)
      make a new file system variable (fs) set to
Machine.stubFileSystem()
      if fs.remove(name)
            return 0 (it worked!)
      else
            return -1 (we have a problem)
catch(Exception e)
      return -1
```

Task II

(25%, 100 LINES) IMPLEMENT SUPPORT FOR MULTIPROGRAMMING. THE CODE WE HAVE GIVEN YOU IS RESTRICTED TO RUNNING ONE USER PROCESS AT A TIME; YOUR JOB IS TO MAKE IT WORK FOR MULTIPLE USER PROCESSES.

Implementation

readVirtualMemory() takes three ints (vaddr, offset, and length) and an array of bytes (data). For what needs to be added to the existing functions, we need to figure out where in the virtual memory the item is sitting, make sure that there's nothing wrong with its physical address, and copy it from the physical address.

```
int vpn = vaddr / pageSize
int voffset = vaddr % pageSize
make a translation entry (entry) from the pageTable (vpn)
set entry's used state to true
int physAddr = entry.ppn*pageSize + voffset

if (physAddr < 0 or physAddr >= memory.length or entry is invalid)
      return 0 (we have a problem)

int amount = Math.min(length, memory.length-physAddr)
System.arraycopy(memory, physAddr, data, offset, amount)

return amount
```

writeVirtualMemory() just like the read counterpart, it takes three ints (vaddr, offset, and length) and a byte array (data). For what needed to be added, most of it was the same as the read one, but instead of moving it *from* physical to, we're moving it *to* the physical memory.

```
int vpn = vaddr / pageSize
int vpoffset = vaddr % pageSize
make a translation entry (entry) from the pageTable (vpn)
set the entry's used state to true
calculate the physical address passed on the pageSize + voffset

if (physAddr < 0 or physAddr >= memory.length or entry is invalid)
      return 0 (we have a problem)

make sure the entry's dirty bit is set to true
```

```
make an int (amount) set to the min of length and memory.length-
physAddr
Systen.arraycopy(data, offset, memory, physaddr, amount)
```

loadSections() for what we needed to implement, within the for loop (within the other for-loop) we had to take the given address and follow it to the actual address.

```
make an int (vpn) set to section.getfirstVPN()+1
make a new translation entry (entry) set to pageTable[vpn]
entry.ppn = UserKernel.availablePages.removeFirst()
set entry.valid to true
entry.readOnly = section.isReadOnly()
section.loadPage(loop variable i, entry.ppn)
```

Task III

(30%, 125 LINES) IMPLEMENT THE SYSTEM CALLS (EXEC, JOIN, AND EXIT, ALSO DOCUMENTED IN SYSCALL.H).

Implementation

handleExec takes three parameters, all ints (name, argc, argv). Then it takes instructions out of VM and lines them up for execution in a child thread.

```
make an array of strings (args) set to newString[argc]
for i < argc
     byte[] pointer = new byte[ADDRESSSIZE]
     readVirtualMemory(argv + i*ADDRESSSIZE, pointer)
     args[i] = readVirtualMemoryString(Lib.bytesToInt(pointer, 0),
STRINGSIZE)
make a new userProcess(child)
add child to myChildren
make a string(childname) set to readVirtualMemoryString(name,
STRINGSIZE)
return child.execute (childname, args) ? child.myPID : -1
```

handleJoin() takes one int parameter (pid). It searches through all the children, and if that child's pID matches the one that needs to be joined, it is.

```
for every child in myChildren
     if the child's pID matches the one we're looking for
          mychildren.remove(child)
          return child.myExitStatus
return -1
```

handleExit takes one int parameter, (exit). It tries to shut down all the threads, halts the machine if it can, calls KThread.finish(), and returns the exit number.

```
set myExitStatus to exit
call unloadSections()
for every open file in the file table
     if the file is not null
          call file.close()
if myPID equals 0
```

```
      call machine.halt()
KThread.finish()
return exit
```

Task IV

(15%, 50 LINES+EXISTING PRIORITY SCHEDULER) IMPLEMENT A LOTTERY SCHEDULER (PLACE IT IN THREADS/LOTTERYSCHEDULER.JAVA). NOTE THAT THIS CLASS EXTENDS PRIORITYSCHEDULER, YOU SHOULD BE ABLE TO REUSE MOST OF THE FUNCTIONALITY OF THAT CLASS; THE LOTTERY SCHEDULER SHOULD NOT BE A LARGE AMOUNT OF ADDITIONAL CODE. THE ONLY MAJOR DIFFERENCE IS THE MECHANISM USED TO PICK A THREAD FROM A QUEUE: A LOTTERY IS HELD, INSTEAD OF JUST PICKING THE THREAD WITH THE MOST PRIORITY. YOUR LOTTERY SCHEDULER SHOULD IMPLEMENT PRIORITY DONATION. (NOTE THAT SINCE THIS IS A LOTTERY SCHEDULER, PRIORITY INVERSION CAN'T ACTUALLY LEAD TO STARVATION! HOWEVER, YOUR SCHEDULER MUST DO PRIORITY DONATION ANYWAY.)

Implementation

In implementing the lottery scheduler, the necessary methods added were setPriority, increasePriority(), decreasePriority(), and getEffectivePriority(). There is also the constructor LotteryScheduler() which doesn't do anything.

setPriority() takes two parameters, the first is a KThread (thread) and an int (priority).It then makes sure that interrupts are disabled, and sets the priority.

```
make sure that interrupts are disabled
make sure that the priority is greater than or equal to the
MINPRIORITY
set the thread state to priority
```

increasePriority() takes no parameters. First it disables interrupts, makes sure that the priority isn't the max priority (if it is it returns false) and sets the priority to one higher, and re-enables interrupts (then returns true).

```
disable interrupts
grab the current thread (thread)
grab thread's priority (priority)
if priority is the MAXPRIORITY
      return false
set thread's priority to priotrity + 1
re-enable interrupts
return true
```

decreasePriority() takes no parameters. It then basically does the exact same thing that increase did, but in the opposite direction.

```
disable interrupts
grab the current thread (thread)
grab thread's priority (priority)
if priority is the MINPRIORITY
      return false
set thread's priority to priotrity - 1
re-enable interrupts
```

```
return true
```

getEffectivePriority() takes one parameter, a ThreadState (mystate). It then grabs mystates's priority, goes through the waiting queue inside the donation queue and if the current thread's state is the same as mystate, we continue, otherwise adds the effective priority to mystate's, and ends up returning it.

```
grab the priority of mystate and put it into variable tickets
for every priority queue (myQueue) in mysteries' donation queue
     for every KThread (currentThread) in myQueue's waiting queue
          if currentThread's thread state is equal to mistake
               continue
          tickets += currentThread's effective priority
return tickets
```