Justin Shuck
CS333 – Fall 2014
Proj 4 – Kernel.c

```
########################################################################
code Kernel

  -- Justin Shuck
  -- CS333 Proj 4
  -- Due: 10/28/2014

    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   SKIPPED CODE    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

-----------------------------  ThreadManager  ---------------------------------

  behavior ThreadManager

      ----------  ThreadManager . Init  ----------

      method Init ()
        --
        -- This method is called once at kernel startup time to initialize
        -- the one and only "ThreadManager" object.
        --

        -- ############   PART1: NEW code   ############
        var
          index: int

        print ("Initializing Thread Manager...\n")
        freeList = new List[Thread]
        threadTable = new array of Thread {MAX_NUMBER_OF_PROCESSES of new Thread}

        -- Allocating a fixed number of threads to re-use
        threadTable[0].Init("thread_0")
        threadTable[1].Init("thread_1")
        threadTable[2].Init("thread_2")
        threadTable[3].Init("thread_3")
        threadTable[4].Init("thread_4")
        threadTable[5].Init("thread_5")
        threadTable[6].Init("thread_6")
        threadTable[7].Init("thread_7")
        threadTable[8].Init("thread_8")
        threadTable[9].Init("thread_9")

        -- We need to set the status for each thread
        -- to UNUSED, then add it to the freeList
        for index = 0 to MAX_NUMBER_OF_PROCESSES-1
            threadTable[index].status = UNUSED
            freeList.AddToEnd( & threadTable[index])
          endFor

        -- Initialize the ThreadManager Lock and
        -- condition variables
        threadManagerLock = new Mutex
        threadManagerLock.Init()
        aThreadBecameFree = new Condition
        aThreadBecameFree.Init()
        leadThread = new Condition
        leadThread.Init()
        -- ############   PART1: NEW code   ############
        endMethod

      ----------  ThreadManager . Print  ----------
```

```
method Print ()
  --
  -- Print each thread.  Since we look at the freeList, this
  -- routine disables interrupts so the printout will be a
  -- consistent snapshot of things.
  --
  var i, oldStatus: int
    oldStatus = SetInterruptsTo (DISABLED)
    print ("Here is the thread table...\n")
    for i = 0 to MAX_NUMBER_OF_PROCESSES-1
      print ("  ")
      printInt (i)
      print (":")
      ThreadPrintShort (&threadTable[i])
    endFor
    print ("Here is the FREE list of Threads:\n   ")
    freeList.ApplyToEach (PrintObjectAddr)
    nl ()
    oldStatus = SetInterruptsTo (oldStatus)
  endMethod

----------  ThreadManager . GetANewThread  ----------

method GetANewThread () returns ptr to Thread
  --
  -- This method returns a new Thread; it will wait
  -- until one is available.
  --
  -- ############   PART1: NEW code   ############
  -- If the freeList is empty
  -- wait on condition of a thread becoming available
  var
    threadToReturn: ptr to Thread
  threadManagerLock.Lock()

  while freeList.IsEmpty()
      leadThread.Wait(&threadManagerLock)
    endWhile

  threadToReturn = freeList.Remove()
  threadToReturn.status = JUST_CREATED
  aThreadBecameFree.Signal(& threadManagerLock)
  threadManagerLock.Unlock()
  return threadToReturn
  -- ############   PART1: NEW code   ############

  endMethod

----------  ThreadManager . FreeThread  ----------

method FreeThread (th: ptr to Thread)
  --
  -- This method is passed a ptr to a Thread;  It moves it
  -- to the FREE list.

  -- ############   PART1: NEW code   ############
  --  - Add a Thread back to the freelist
  --  - Signal anyone waiting on the condition
  threadManagerLock.Lock()
  if th
      th.status = UNUSED
      freeList.AddToEnd(th)
      leadThread.Signal(& threadManagerLock)
  else
      FatalError("Trying to Free an Invalid Thread")
    endIf
```

```
        threadManagerLock.Unlock()
        -- ############   PART1: NEW code   ############
        endMethod

    endBehavior

-------------------------  ProcessControlBlock  ------------------------------

  behavior ProcessControlBlock

      ----------  ProcessControlBlock . Init  ----------
      --
      -- This method is called once for every PCB at startup time.
      --
      method Init ()
          pid = -1
          status = FREE
          addrSpace = new AddrSpace
          addrSpace.Init ()
-- Uncomment this code later...
/*
          fileDescriptor = new array of ptr to OpenFile
                      { MAX_FILES_PER_PROCESS of null }
*/
        endMethod

      ----------  ProcessControlBlock . Print  ----------

      method Print ()
        --
        -- Print this ProcessControlBlock using several lines.
        --
        -- var i: int
          self.PrintShort ()
          addrSpace.Print ()
          print ("    myThread = ")
          ThreadPrintShort (myThread)
-- Uncomment this code later...
/*
          print ("    File Descriptors:\n")
          for i = 0 to MAX_FILES_PER_PROCESS-1
            if fileDescriptor[i]
              fileDescriptor[i].Print ()
            endIf
          endFor
*/
          nl ()
        endMethod

      ----------  ProcessControlBlock . PrintShort  ----------

      method PrintShort ()
        --
        -- Print this ProcessControlBlock on one line.
        --
          print ("  ProcessControlBlock   (addr=")
          printHex (self asInteger)
          print (")   pid=")
          printInt (pid)
          print (", status=")
          if status == ACTIVE
            print ("ACTIVE")
          elseIf status == ZOMBIE
            print ("ZOMBIE")
          elseIf status == FREE
            print ("FREE")
          else
            FatalError ("Bad status in ProcessControlBlock")
          endIf
          print (", parentsPid=")
          printInt (parentsPid)
          print (", exitStatus=")
          printInt (exitStatus)
          nl ()
        endMethod

    endBehavior

-----------------------------  ProcessManager  ---------------------------------

  behavior ProcessManager

      ----------  ProcessManager . Init  ----------

      method Init ()
        --
        -- This method is called once at kernel startup time to initialize
        -- the one and only "processManager" object.
        --

        -- ############   PART2: NEW code   ############
        -- We need to initialize:
        --   - processTable array
        --   - the ProcessControlBlocks in that array
        --   - the processManagerLock
        --   - the aProcessBecameFree and aProcessDied
        --   - the freeList
        var index: int

        ---------- freeList -----------------
        freeList = new List[ProcessControlBlock]

        -------- processTable of ProcessControlBlock ------------
        processTable = new array of ProcessControlBlock {MAX_NUMBER_OF_PROCESSES of new
ProcessControlBlock}
        for index = 0 to MAX_NUMBER_OF_PROCESSES-1
            processTable[index].Init()
            freeList.AddToEnd(& processTable[index])
          endFor

        ----------- processManagerLock, aProcessBecameFree & aProcessDied --------------
        processManagerLock = new Mutex
        processManagerLock.Init()
        aProcessBecameFree = new Condition
        aProcessBecameFree.Init()
        aProcessDied = new Condition
        aProcessDied.Init()

        -- ############   PART2: NEW code   ############
        endMethod

      ----------  ProcessManager . Print  ----------

      method Print ()
        --
        -- Print all processes.  Since we look at the freeList, this
        -- routine disables interrupts so the printout will be a
        -- consistent snapshot of things.
        --
        var i, oldStatus: int
          oldStatus = SetInterruptsTo (DISABLED)
          print ("Here is the process table...\n")
          for i = 0 to MAX_NUMBER_OF_PROCESSES-1
            print ("  ")
            printInt (i)
            print (":")
            processTable[i].Print ()
          endFor
          print ("Here is the FREE list of ProcessControlBlocks:\n   ")
          freeList.ApplyToEach (PrintObjectAddr)
          nl ()
          oldStatus = SetInterruptsTo (oldStatus)
        endMethod

      ----------  ProcessManager . PrintShort  ----------

      method PrintShort ()
        --
        -- Print all processes.  Since we look at the freeList, this
        -- routine disables interrupts so the printout will be a
        -- consistent snapshot of things.
        --
        var i, oldStatus: int
          oldStatus = SetInterruptsTo (DISABLED)
          print ("Here is the process table...\n")
          for i = 0 to MAX_NUMBER_OF_PROCESSES-1
            print ("  ")
            printInt (i)
            processTable[i].PrintShort ()
          endFor
          print ("Here is the FREE list of ProcessControlBlocks:\n   ")
          freeList.ApplyToEach (PrintObjectAddr)
          nl ()
          oldStatus = SetInterruptsTo (oldStatus)
        endMethod

      ----------  ProcessManager . GetANewProcess  ----------

      method GetANewProcess () returns ptr to ProcessControlBlock
        --
        -- This method returns a new ProcessControlBlock; it will wait
        -- until one is available.
        --
        -- ############   PART2: NEW code   ############
        -- GetANewProcess is similar to GetANew Thread
        -- thus, I used that framework to create this method
        var
          processToReturn: ptr to ProcessControlBlock
        processManagerLock.Lock()

        while freeList.IsEmpty()
            aProcessBecameFree.Wait(&processManagerLock)
          endWhile

        processToReturn = freeList.Remove()
        processToReturn.status = ACTIVE

        processManagerLock.Unlock()
        return processToReturn

        -- ############   PART2: NEW code   ############

        endMethod

      ----------  ProcessManager . FreeProcess  ----------

      method FreeProcess (p: ptr to ProcessControlBlock)
        --
```

```
        -- This method is passed a ptr to a Process;  It moves it
        -- to the FREE list.
        --
        -- ############   PART2: NEW code   ############
        -- FreeProcess method needs to change the process status to FREE
        -- and add it to the free list
        processManagerLock.Lock()
        p.status = FREE
        freeList.AddToEnd(p)
        aProcessBecameFree.Signal(& processManagerLock)
        processManagerLock.Unlock()
        -- ############   PART2: NEW code   ############
        endMethod


    endBehavior

-----------------------------  PrintObjectAddr  ---------------------------------

  function PrintObjectAddr (p: ptr to Object)
    --
    -- Print the address of the given object.
    --
      printHex (p asInteger)
      printChar (' ')
    endFunction

-----------------------------  ProcessFinish  --------------------------

  function ProcessFinish (exitStatus: int)
    --
    -- This routine is called when a process is to be terminated.  It will
    -- free the resources held by this process and will terminate the
    -- current thread.
    --
      FatalError ("ProcessFinish is not implemented")
    endFunction

-----------------------------  FrameManager  ---------------------------------

  behavior FrameManager

      ----------  FrameManager . Init  ----------

      method Init ()
        --
        -- This method is called once at kernel startup time to initialize
        -- the one and only "frameManager" object.
        --
        var i: int
          print ("Initializing Frame Manager...\n")
          framesInUse = new BitMap
          framesInUse.Init (NUMBER_OF_PHYSICAL_PAGE_FRAMES)
          numberFreeFrames = NUMBER_OF_PHYSICAL_PAGE_FRAMES
          frameManagerLock = new Mutex
          frameManagerLock.Init ()
          newFramesAvailable = new Condition
          newFramesAvailable.Init ()
          -- Check that the area to be used for paging contains zeros.
          -- The BLITZ emulator will initialize physical memory to zero, so
          -- if by chance the size of the kernel has gotten so large that
          -- it runs into the area reserved for pages, we will detect it.
          -- Note: this test is not 100%, but is included nonetheless.
          for i = PHYSICAL_ADDRESS_OF_FIRST_PAGE_FRAME
                   to PHYSICAL_ADDRESS_OF_FIRST_PAGE_FRAME+300
                   by 4
      if 0 != *(i asPtrTo int)
       FatalError ("Kernel code size appears to have grown too large and is overflowing
into the frame region")
      endIf
     endFor
    endMethod

   ----------  FrameManager . Print  ----------

   method Print ()
    --
    -- Print which frames are allocated and how many are free.
    --
     frameManagerLock.Lock ()
     print ("FRAME MANAGER:\n")
     printIntVar ("  numberFreeFrames", numberFreeFrames)
     print ("  Here are the frames in use: \n    ")
     framesInUse.Print ()
     frameManagerLock.Unlock ()
    endMethod

   ----------  FrameManager . GetAFrame  ----------

   method GetAFrame () returns int
    --
    -- Allocate a single frame and return its physical address.  If no frames
    -- are currently available, wait until the request can be completed.
    --
     var f, frameAddr: int

     -- Acquire exclusive access to the frameManager data structure...
     frameManagerLock.Lock ()

     -- Wait until we have enough free frames to entirely satisfy the request...
     while numberFreeFrames < 1
      newFramesAvailable.Wait (&frameManagerLock)
     endWhile

     -- Find a free frame and allocate it...
     f = framesInUse.FindZeroAndSet ()
     numberFreeFrames = numberFreeFrames - 1

     -- Unlock...
     frameManagerLock.Unlock ()

     -- Compute and return the physical address of the frame...
     frameAddr = PHYSICAL_ADDRESS_OF_FIRST_PAGE_FRAME + (f * PAGE_SIZE)
     -- printHexVar ("GetAFrame returning frameAddr", frameAddr)
     return frameAddr
    endMethod

   ----------  FrameManager . GetNewFrames  ----------

   method GetNewFrames (aPageTable: ptr to AddrSpace, numFramesNeeded: int)
     -- ############   PART3: NEW code   ############
     -- This method aquires the frame manager lock and then
     -- waits on newFramesAvailable until there are enough frames to.
     -- After looping over the frames we adjust the number of free frames,
     -- set aPageTable.numberOfPages to the number of frames we just allocated
     var
      index, addr, frame: int

     -- Aquire frame manager lock
     frameManagerLock.Lock()

     --Waits on newFramesAvailable until there are enough frames
```

```
          while numberFreeFrames < numFramesNeeded
              newFramesAvailable.Wait(& frameManagerLock)
            endWhile

          -- Loop on the frames using the technique described in the hw assignment:
          -- Determine which frames are free (using BitMap), Figure out the address
          -- of the free framesand execute a setFrameAddr to  set to store the address of the
frame
          for index = 0 to numFramesNeeded-1
              frame = framesInUse.FindZeroAndSet()
              addr = PHYSICAL_ADDRESS_OF_FIRST_PAGE_FRAME + (frame * PAGE_SIZE)
              aPageTable.SetFrameAddr(index, addr)
            endFor

          -- Adjust the number of free frames
          numberFreeFrames = numberFreeFrames - numFramesNeeded

          -- Sets aPageTable.numberOfPages to the number of frames that were allocated
          aPageTable.numberOfPages = aPageTable.numberOfPages + numFramesNeeded

          frameManagerLock.Unlock()
          -- ############   PART3: NEW code   ############
        endMethod

      ----------  FrameManager . ReturnAllFrames  ----------

      method ReturnAllFrames (aPageTable: ptr to AddrSpace)
          -- ############   PART3: NEW code   ############
          -- Think about this as doing the opposite as 'GetNewFrames',
          -- We want to begin by aquiring the frame lock, get the number
          -- of frames to return, and then perform a loop over the frames to clear each bit
          -- (by getting the address from the page table and get the corresponding bitnumber).
After
          -- looping we want to do a broadcast, update the aPageTable.numberOfPages
          -- and release the lock
          var
            index, holdFrames, addr, bit: int

          -- Aquire the lock
          frameManagerLock.Lock()
          aPageTable.SetToThisPageTable()

          holdFrames = aPageTable.numberOfPages

          -- The loop that was described in the method call.
          -- Basically we want to get the address from the page table, get its
          -- bit number and clear each bit
          for index = 0 to holdFrames-1
              addr = aPageTable.ExtractFrameAddr(index)
              bit = (addr - PHYSICAL_ADDRESS_OF_FIRST_PAGE_FRAME) / PAGE_SIZE
              framesInUse.ClearBit(bit)
              numberFreeFrames = numberFreeFrames+1
            endFor

          -- Broadcast that the frames we allocated are available
          newFramesAvailable.Broadcast(& frameManagerLock)

          -- Update the aPageTable.numberOfPages
          aPageTable.numberOfPages = aPageTable.numberOfPages - holdFrames
          -- Release the lock
          frameManagerLock.Unlock()
          -- ############   PART3: NEW code   ############
        endMethod

    endBehavior
     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   SKIPPED CODE    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```