Justin Shuck
CS333 – Fall 2014
Proj 5 – Kernel.c

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

  -- Justin Shuck
  -- CS333 Proj 5
  -- Due: 11/4/2014

             XXXXXXXXXXXXXXXXXXXXXXXX     SKIPPED CODE     #XXXXXXXXXXXXXXXXXXXXXXXX

-- ############   NEW code   ############
-----------------------------  InitFirstProcess  --------------------------

function InitFirstProcess()
  var
    ptrThread: ptr to Thread

  ptrThread = threadManager.GetANewThread()
  ptrThread.Init("UserProgramThread")
  ptrThread.Fork(StartUserProcess,0)

endFunction
-- ############   NEW code   ############
-- ############   NEW code   ############
-----------------------------  StartUserProcess  --------------------------

function StartUserProcess(arg : int)
    -- We need to allocate a new PCB and connect it with the current thread.
    -- We then initialize the thread field in the PCB and the myProcess
    -- field in the current thread. We then open the executable file (hard code).
    -- We then create the Logicaladdress space and read the executable into it.
    -- We need to remember to close the executable file we opened earlier.
    -- Then we need to compute the inital value for the user-level stack.
    -- Finially we jump into the user-level program.

    var
      ptrOpenFile: ptr to OpenFile
      ptrToPCB: ptr to ProcessControlBlock
      ptrInitSystemStackTop: ptr to int
      initPC: int
      initUserStackTop: int
      previousStatus: int

    --Allocate a new PCB and connect it with the current thread
    ptrToPCB = processManager.GetANewProcess()
    ptrToPCB.myThread = currentThread
    currentThread.myProcess = ptrToPCB

    -- Open the executable (hard coded)
    ptrOpenFile = fileManager.Open("TestProgram1")
    if ptrOpenFile == null
        FatalError("ERROR: Cannot open 'TestProgram1'.")
      endIf

    -- create the LogicalAddress space using 'LoadExecutable'
    -- And make sure to close the executable (otherwise a syste
    -- recourse will become permanently locked up)
    initPC = ptrOpenFile.LoadExecutable(& ptrToPCB.addrSpace)
    fileManager.Close(ptrOpenFile)

    -- Compute the initial value(# of pages * Page size) and then jump into the
    -- user-level program
    initUserStackTop = (ptrToPCB.addrSpace.numberOfPages * PAGE_SIZE)
    ptrInitSystemStackTop = &currentThread.systemStack[SYSTEM_STACK_SIZE-1]
    previousStatus = SetInterruptsTo(DISABLED)
    ptrToPCB.addrSpace.SetToThisPageTable()
    currentThread.isUserThread = true
    BecomeUserThread(initUserStackTop, initPC, ptrInitSystemStackTop asInteger)
endFunction
-- ############    NEW code   ############


             XXXXXXXXXXXXXXXXXXXXXXXX     SKIPPED CODE     #XXXXXXXXXXXXXXXXXXXXXXXX


  -- ############   NEW code   ############
-----------------------------  DiskInterruptHandler  --------------------------

  function DiskInterruptHandler ()
    --
    -- This routine is called when a disk interrupt occurs.  It will
    -- signal the "semToSignalOnCompletion" Semaphore and return to
    -- the interrupted thread.
    --
    -- This is an interrupt handler.  As such, interrupts will be DISABLED
    -- for the duration of its execution.
    --
    -- Uncomment this code later...
    -- FatalError ("DISK INTERRUPTS NOT EXPECTED IN PROJECT 4")

      currentInterruptStatus = DISABLED
      -- print ("DiskInterruptHandler invoked!\n")
      if diskDriver.semToSignalOnCompletion
        diskDriver.semToSignalOnCompletion.Up()
      endIf

    endFunction
    -- ############   NEW code   ############


             XXXXXXXXXXXXXXXXXXXXXXXX     SKIPPED CODE     #XXXXXXXXXXXXXXXXXXXXXXXX


-----------------------------  Handle_Sys_Exit  ---------------------------------
  -- ############  NEW code   ############
  function Handle_Sys_Exit (returnStatus: int)
      -- NOT IMPLEMENTED
      print("Handle_sys_Exit invoked!\n")
      print("returnStatus = ")
      printInt(returnStatus)
      print("\n")
    endFunction
  -- ############   NEW code   ############

-----------------------------  Handle_Sys_Shutdown  ---------------------------------

  function Handle_Sys_Shutdown ()
      -- Mock out a system shutdown by calling a FatalError
      FatalError("Syscall 'Shutdown' was invoked by a user thread")
    endFunction

-----------------------------  Handle_Sys_Yield  ---------------------------------
  -- ############   NEW code   ############
  function Handle_Sys_Yield ()
      -- NOT IMPLEMENTED
      print("Handle_Sys_Yield invoked! \n")
    endFunction
  -- ############   NEW code   ############
```

```
-----------------------------  Handle_Sys_Fork  ---------------------------------
  -- ############   NEW code   ############
  function Handle_Sys_Fork () returns int
      -- NOT IMPLEMENTED
      print("Handle_Sys_Fork invoked! \n")
      return 1000
    endFunction
  -- ############   NEW code   ############
-----------------------------  Handle_Sys_Join  ---------------------------------
  -- ############   NEW code   ############
  function Handle_Sys_Join (processID: int) returns int
      -- NOT IMPLEMENTED
      print("Handle_Sys_Join invoked!\n")
      print("processID = ")
      printInt(processID)
      print("\n")
      return 2000
    endFunction
  -- ############   NEW code   ############
-----------------------------  Handle_Sys_Exec  ---------------------------------
  -- ############   NEW code   ############
  function Handle_Sys_Exec (filename: ptr to array of char) returns int
      -- This function will read a new executable program from disk and copy it into
      -- the address space of the process which invoked the Exec. This begins execution of the
new program.
      -- The implementation is similar to InitFirstProcess and StartUserProcess with some
differences.
      -- We have to work with 2 virtual address spaces. Since LoadExecutable may fail, thus our
kernel must be able
      -- to return to the process that was invoked with Exec with an error code.
      -- This implementation will use a local variable of AddrSpace, and then coppy it into the
PrcoessControlBlock.
      -- The frames of the previous address space must be freed first!
      -- We then need to copy the characters into an array variable (use MAX_STRING_SIZE)

      var
        ptrOpenFile2: ptr to OpenFile
        newAddrSpace: AddrSpace = new AddrSpace
        stringStorage: array[MAX_STRING_SIZE] of char
        ptrToPCB: ptr to ProcessControlBlock
        initPC: int
        numOfBytes: int
        initUserStackTop: int
        ptrInitSystemStackTop: ptr to int
        previousStatus: int

    -- init newAddrSpace
    newAddrSpace.Init()

    -- Point to the currentThreads process
    ptrToPCB = currentThread.myProcess

    -- Get the filename into system space
    numOfBytes = ptrToPCB.addrSpace.GetStringFromVirtual(&stringStorage, filename asInteger,
MAX_STRING_SIZE)
    if numOfBytes < 0
        return -1
      endIf

    -- Open the executable
    ptrOpenFile2 = fileManager.Open(&stringStorage)
    if ptrOpenFile2 == null
        return -1
      endIf
    -- create the LogicalAddress space using 'LoadExecutable'
    -- And make sure to close the executable (otherwise a syste
    -- recourse will become permanently locked up)
    -- Check to see if there was an error loading a program into
    -- memory
    initPC = ptrOpenFile2.LoadExecutable(& newAddrSpace)
    if initPC < 0
        return -1
      endIf

    -- Compute the initial value(# of pages * Page size) and then jump into the
    -- user-level program
    ptrToPCB.addrSpace = newAddrSpace
    fileManager.Close(ptrOpenFile2)
    frameManager.ReturnAllFrames(& currentThread.myProcess.addrSpace)
    initUserStackTop = (newAddrSpace.numberOfPages * PAGE_SIZE)
    ptrInitSystemStackTop = & currentThread.systemStack[SYSTEM_STACK_SIZE-1]
    previousStatus = SetInterruptsTo(DISABLED)
    currentThread.isUserThread = true
    BecomeUserThread(initUserStackTop, initPC, ptrInitSystemStackTop asInteger)
      return 3000
    endFunction
  -- ############   NEW code   ############
-----------------------------  Handle_Sys_Create  ---------------------------------
  -- ############   NEW code   ############
  function Handle_Sys_Create (filename: ptr to array of char) returns int
    var
      stringStorage: array[MAX_STRING_SIZE] of char
      numOfBytes: int

      numOfBytes = currentThread.myProcess.addrSpace.GetStringFromVirtual(&stringStorage,
filename asInteger, MAX_STRING_SIZE)

      --Check to see if theres an error when getting string from Virtual
      if numOfBytes < 0
          FatalError("ERROR: Error has occured in Handle_Sys_Create")
        endIf
      print("Handle_Sys_Create invoked!\n")
      printHexVar("virt addr of filename = ", filename asInteger)
      print("filename = ")
      printString(&stringStorage)
      print("\n")
      return 4000
    endFunction
  -- ############   NEW code   ############
-----------------------------  Handle_Sys_Open  ---------------------------------
  -- ############   NEW code   ############
  function Handle_Sys_Open (filename: ptr to array of char) returns int
      -- NOT IMPLEMENTED
      var
        stringStorage: array[MAX_STRING_SIZE] of char
        numOfBytes: int
      numOfBytes = currentThread.myProcess.addrSpace.GetStringFromVirtual(&stringStorage,
filename asInteger, MAX_STRING_SIZE)

      --Check to see if theres an error when getting string from Virtual
      if numOfBytes < 0
          FatalError("ERROR: Error has occured in Handle_Sys_Open")
        endIf
      print("Handle_Sys_Open called invoked! \n")
      printHexVar("virt addr of filename = ", filename asInteger)
      print("filename = ")
      printString(&stringStorage)
      print("\n")
      return 5000
    endFunction
  -- ############   NEW code   ############
-----------------------------  Handle_Sys_Read  ---------------------------------
  -- ############   NEW code   ############
  function Handle_Sys_Read (fileDesc: int, buffer: ptr to char, sizeInBytes: int) returns int
      -- NOT IMPLEMENTED
      print("Handle_Sys_Read invoked! \n fileDesc = ")
      printInt(fileDesc)
      print("\nvirt addr of buffer = ")
      printHex(buffer asInteger)
      print("\nsizeInBytes = ")
      printInt(sizeInBytes)
      print("\n")

      return 6000
    endFunction
  -- ############   NEW code   ############
-----------------------------  Handle_Sys_Write  ---------------------------------
  -- ############   NEW code   ############
  function Handle_Sys_Write (fileDesc: int, buffer: ptr to char, sizeInBytes: int) returns int
      -- NOT IMPLEMENTED
      print("Handle_Sys_Write invoked!\n")
      print("fileDesc = ")
      printInt(fileDesc)
      print("\nvirt addr of buffer = ")
      printHex(buffer asInteger)
      print("\nsizeInBytes = ")
      printInt(sizeInBytes)
      print("\n")
      return 7000
    endFunction
  -- ############   NEW code   ############
-----------------------------  Handle_Sys_Seek  ---------------------------------
  -- ############   NEW code   ############
  function Handle_Sys_Seek (fileDesc: int, newCurrentPos: int) returns int
      -- NOT IMPLEMENTED
      print("Handle_Sys_Seek invoked!\n")
      print("fileDesc = ")
      printInt(fileDesc)
      print("\nnewCurrentPos = ")
      printInt(newCurrentPos)
      print("\n")
      return 8000
    endFunction
  -- ############   NEW code   ############
-----------------------------  Handle_Sys_Close  ---------------------------------
  -- ############   NEW code   ############
  function Handle_Sys_Close (fileDesc: int)
      print("Handle_Sys_Close invoked!\n")
      print("fileDes = ")
      printInt(fileDesc)
      print(".\n")
    endFunction
  -- ############   NEW code   ############
-----------------------------  printString  ---------------------------------

  -- ############   NEW code   ############
function printString( arg: String)
    -- Helper function to print a char array string
    print(arg)
  endFunction
  -- ############   NEW code   ############
             XXXXXXXXXXXXXXXXXXXXXXXX     SKIPPED CODE     #XXXXXXXXXXXXXXXXXXXXXXXX

endCode
```