Justin Shuck

CS333 – Fall 2014

Proj 8 – Kernel.c

######################################################################

**----------------------------- Handle\_Sys\_Open ---------------------------------**

function Handle\_Sys\_Open (filename: ptr to array of char) returns int

-- Gets the file name, does verification and sets the

-- file in an empty position in the fileDescriptor array.

-- Returns the index position in the fileDescriptor array

-- Implementation:

-- 1. Copy filename string from virtual space to a small buffer

-- 2. Make sure the legnth of the name doesnt exceed the max size

-- 3. Locate an empty slot in fileDescriptor (if none return -1)

-- 4. Allocate OpenFile obj (return -1 if this fails)

-- 5. set the entry to point at the open File

-- 6. return index of the fileDescriptor array

var

numOfBytes: int

stringStorage: array[MAX\_STRING\_SIZE] of char

i: int

pcb: ptr to ProcessControlBlock

open: ptr to OpenFile

holdI: int

-- 0. Init variables

pcb = currentThread.myProcess

-- 1. Copy filename into a small buffer

numOfBytes = pcb.addrSpace.GetStringFromVirtual(&stringStorage, filename asInteger, MAX\_STRING\_SIZE)

-- 2. make sure the lenth of the name doesnt exceed max (return -1)

if stringStorage arraySize > MAX\_STRING\_SIZE

return -1

endIf

-- 3a. locatean empty slot in fileDescriptor

-- 4a. Allocate OpenFile obj

open = null

holdI = -1

for i = 0 to MAX\_FILES\_PER\_PROCESS - 1

if pcb.fileDescriptor[i] == null

-- ########## NEW CODE ##########

-- Check for terminal, else do normal execution

if StrEqual(&stringStorage, "terminal")

pcb.fileDescriptor[i] = &fileManager.serialTerminalFile

else

pcb.fileDescriptor[i] = fileManager.Open(&stringStorage)

endIf

-- Check to see if there was an error.

-- Otherwise return index of where the file was stored at

if pcb.fileDescriptor[i] != null

return i

else

return -1

endIf

endIf

endFor

-- Catch all for any other errors

return -1

endFunction

**----------------------------- Handle\_Sys\_Read ---------------------------------**

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")

var

open: ptr to OpenFile

virtAddr: int

virtPage: int

offset: int

copiedSoFar: int

nextPosInFile: int

thisChunksize: int

sizeOfFile: int

hold: bool

destAddr: int

holdChar: char

i: int

in: int

in = SetInterruptsTo(ENABLED)

-- Begin by checking fileDesc

if fileDesc >= MAX\_NUMBER\_OF\_OPEN\_FILES || fileDesc < 0

return -1

endIf

-- Check to see if sizeInBytes is negative

if sizeInBytes < 0

return -1

endIf

--Get the OpenFile

open = currentThread.myProcess.fileDescriptor[fileDesc]

if open == null

return -1

endIf

virtAddr = buffer asInteger

virtPage = virtAddr / PAGE\_SIZE

offset = virtAddr % PAGE\_SIZE

copiedSoFar = 0

nextPosInFile = open.currentPos

-- ############ NEW CODE ##################

-- If we're dealing with a 'terminal'

if open.kind == TERMINAL

while true

-- Compute Size of Chunk

thisChunksize = PAGE\_SIZE - offset

if copiedSoFar + thisChunksize > sizeInBytes

thisChunksize = sizeInBytes - copiedSoFar

endIf

-- Check to see if We're done

if thisChunksize <= 0

break

endIf

-- check for various errors

if virtPage < 0 || virtPage > NUMBER\_OF\_PHYSICAL\_PAGE\_FRAMES || !currentThread.myProcess.addrSpace.IsValid(virtPage) || !currentThread.myProcess.addrSpace.IsWritable(virtPage)

return -1

endIf

--Set dirtyBit for this page

currentThread.myProcess.addrSpace.SetDirty(virtPage)

--set referencedBit for this page

currentThread.myProcess.addrSpace.SetReferenced(virtPage)

-- Get the destination address

destAddr = currentThread.myProcess.addrSpace.ExtractFrameAddr(virtPage) + offset

if destAddr == 0

return copiedSoFar

endIf

for i = 0 to thisChunksize - 1

holdChar = serialDriver.GetChar()

copiedSoFar = copiedSoFar + 1

-- Handle the special Characters and then return

if holdChar == '\n' || holdChar == '\r'

\*(destAddr asPtrTo char + i) = '\n'

return copiedSoFar

endIf

-- Handle EOF

if holdChar == 0x04

return copiedSoFar - 1

endIf

-- Put the character into the destination address

\*(destAddr asPtrTo char + i) = holdChar

endFor

-- Increment and repeat

nextPosInFile = nextPosInFile + thisChunksize

virtPage = virtPage + 1

offset = 0

endWhile

-- Incase we haven't returned already, return the count

return copiedSoFar

endIf

-- ############ NEW CODE ##################

-- Else we're dealing with a File, Handle as before

virtAddr = buffer asInteger

virtPage = virtAddr / PAGE\_SIZE

offset = virtAddr % PAGE\_SIZE

copiedSoFar = 0

nextPosInFile = open.currentPos

sizeOfFile = open.fcb.sizeOfFileInBytes

-- Each iteration will compute the size of the next chunk and process it

while true

--compute size of chunk

thisChunksize = PAGE\_SIZE - offset

if nextPosInFile + thisChunksize > sizeOfFile

thisChunksize = sizeOfFile - nextPosInFile

endIf

if copiedSoFar + thisChunksize > sizeInBytes

thisChunksize = sizeInBytes - copiedSoFar

endIf

-- Check to see if we're done

if thisChunksize <= 0

break

endIf

-- check for various errors

if virtPage < 0 || virtPage > MAX\_PAGES\_PER\_VIRT\_SPACE - 1 || !currentThread.myProcess.addrSpace.IsValid(virtPage) || !currentThread.myProcess.addrSpace.IsWritable(virtPage)

return -1

endIf

--Do the read:

--Set dirtyBit for this page

currentThread.myProcess.addrSpace.SetDirty(virtPage)

--set referencedBit for this page

currentThread.myProcess.addrSpace.SetReferenced(virtPage)

destAddr = currentThread.myProcess.addrSpace.ExtractFrameAddr(virtPage) + offset

if destAddr == 0

return copiedSoFar

endIf

-- Perform read into destAddr(with next postion in file and chunksize)

hold = fileManager.SynchRead(open, destAddr, nextPosInFile,thisChunksize)

-- Increment

nextPosInFile = nextPosInFile + thisChunksize

open.currentPos = nextPosInFile

copiedSoFar = copiedSoFar + thisChunksize

virtPage = virtPage + 1

offset = 0

-- Check to see if we're done

if copiedSoFar == sizeInBytes

break

endIf

endWhile

return copiedSoFar

endFunction

**----------------------------- Handle\_Sys\_Write ---------------------------------**

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")

var

open: ptr to OpenFile

virtAddr: int

virtPage: int

offset: int

copiedSoFar: int

nextPosInFile: int

thisChunksize: int

sizeOfFile: int

hold: bool

destAddr: int

i: int = 0

holdChar: char

in: int

in = SetInterruptsTo(ENABLED)

-- Begin by checking fileDesc

if fileDesc >= MAX\_NUMBER\_OF\_OPEN\_FILES || fileDesc < 0

return -1

endIf

-- Check to see if sizeInBytes is negative

if sizeInBytes < 0

return -1

endIf

--Get the OpenFile

open = currentThread.myProcess.fileDescriptor[fileDesc]

if open == null

return -1

endIf

virtAddr = buffer asInteger

virtPage = virtAddr / PAGE\_SIZE

offset = virtAddr % PAGE\_SIZE

copiedSoFar = 0

nextPosInFile = open.currentPos

-- ########### NEW CODE ############

-- Handle the case where the file is a 'terminal'

if open.kind == TERMINAL

while true

-- Get the chunk size

thisChunksize = PAGE\_SIZE - offset

if copiedSoFar + thisChunksize > sizeInBytes

thisChunksize = sizeInBytes - copiedSoFar

endIf

-- Check to see if we're done

if thisChunksize <= 0

break

endIf

-- check for various errors

if virtPage < 0 || virtPage > NUMBER\_OF\_PHYSICAL\_PAGE\_FRAMES || !currentThread.myProcess.addrSpace.IsValid(virtPage) || !currentThread.myProcess.addrSpace.IsWritable(virtPage)

return -1

endIf

-- Set referenced bit

currentThread.myProcess.addrSpace.SetReferenced(virtPage)

-- Calculate frame address

destAddr = currentThread.myProcess.addrSpace.ExtractFrameAddr(virtPage) + offset

if destAddr == 0

return copiedSoFar

endIf

for i = 0 to thisChunksize - 1

-- Acquire the character from the destination address

holdChar = \*(destAddr asPtrTo char + i)

-- Check to see if the character is EOF, if so return immediately

if holdChar == 0x04

return copiedSoFar

endIf

-- Replace \n with \r

if holdChar == '\n'

serialDriver.PutChar('\r')

endIf

-- Place the character on Put Buffer

serialDriver.PutChar(holdChar)

-- Increment counter

copiedSoFar = copiedSoFar + 1

endFor

-- Increment and repeat

nextPosInFile = nextPosInFile + thisChunksize

virtPage = virtPage + 1

offset = 0

endWhile

-- Incase we haven't returned the count already

return copiedSoFar

endIf

-- ############ NEW CODE ##################

--Handle the File case as before

open = currentThread.myProcess.fileDescriptor[fileDesc]

if open == null

return -1

endIf

virtAddr = buffer asInteger

virtPage = virtAddr / PAGE\_SIZE

offset = virtAddr % PAGE\_SIZE

copiedSoFar = 0

nextPosInFile = open.currentPos

sizeOfFile = open.fcb.sizeOfFileInBytes

-- Each iteration will compute the size of the next chunk and process it

while true

--compute size of chunk

thisChunksize = PAGE\_SIZE - offset

if nextPosInFile + thisChunksize > sizeOfFile

thisChunksize = sizeOfFile - nextPosInFile

endIf

if copiedSoFar + thisChunksize > sizeInBytes

thisChunksize = sizeInBytes - copiedSoFar

endIf

-- Check to see if we're done

if thisChunksize <= 0

break

endIf

-- check for various errors

if virtPage < 0 || virtPage > NUMBER\_OF\_PHYSICAL\_PAGE\_FRAMES || !currentThread.myProcess.addrSpace.IsValid(virtPage) || !currentThread.myProcess.addrSpace.IsWritable(virtPage)

return -1

endIf

--Do the write:

--set referencedBit for this page

currentThread.myProcess.addrSpace.SetReferenced(virtPage)

destAddr = currentThread.myProcess.addrSpace.ExtractFrameAddr(virtPage) + offset

if destAddr == 0

return copiedSoFar

endIf

-- Perform read into destAddr(with next postion in file and chunksize)

--fileManager.fileManagerLock.Unlock()

hold = fileManager.SynchWrite(open, destAddr, nextPosInFile,thisChunksize) --I

-- Increment

nextPosInFile = nextPosInFile + thisChunksize

open.currentPos = nextPosInFile

copiedSoFar = copiedSoFar + thisChunksize

virtPage = virtPage + 1

offset = 0

-- Check to see if we're done

if copiedSoFar == sizeInBytes

break

endIf

endWhile

return copiedSoFar

endFunction

**----------------------------- Handle\_Sys\_Seek ---------------------------------**

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")

-- Implementation:

-- 1. Lock the FileManager

-- 2. Check fileDesc and get a pointer to the Open File

-- 3. Make sure the file is open (null entry == not open)

-- 4. Deal with new curPos == -1

-- 5. Deal with new curPos < -1 (Zero is okay)

-- 6. Deal with new curPos > filesize

-- 7. Update currentPos

-- 8. return new curPos

var

pcb: ptr to ProcessControlBlock

open: ptr to OpenFile

-- 0. Initilize

pcb = currentThread.myProcess

-- ######### NEW CODE ##########

-- If we're trying to seek a Terminal File, return -1

if pcb.fileDescriptor[fileDesc].kind == TERMINAL

return -1

endIf

-- ######### NEW CODE ##########

-- 1. Lock the FileManager

fileManager.fileManagerLock.Lock()

-- 2. Check fileDesc and get a pointer to the open File

/\*for i = 0 to MAX\_FILES\_PER\_PROCESS - 1

if pcb.fileDescriptor[i] == null

open = pcb.fileDescriptor[i]

break

endIf

endFor\*/

if fileDesc > MAX\_FILES\_PER\_PROCESS || fileDesc < 0

fileManager.fileManagerLock.Unlock()

return -1

endIf

open = pcb.fileDescriptor[fileDesc]

if open == null

fileManager.fileManagerLock.Unlock()

return -1

endIf

-- 3. Make sure the file is open

if open.fcb == null

fileManager.fileManagerLock.Unlock()

return -1

endIf

-- 4. Deal with new Current Position being -1

if newCurrentPos == -1

newCurrentPos = open.fcb.sizeOfFileInBytes

endIf

--5. Deal with new current Position being < -1

--6. Deal with new current Position being > filesize

if newCurrentPos < -1 || newCurrentPos > open.fcb.sizeOfFileInBytes

fileManager.fileManagerLock.Unlock()

return -1

endIf

--7. update currentPos

open.currentPos = newCurrentPos

--8. return new curPos

fileManager.fileManagerLock.Unlock()

return newCurrentPos

endFunction

----------------------------- Handle\_Sys\_Close ---------------------------------

function Handle\_Sys\_Close (fileDesc: int)

-- Check the argument (is it a legal array index/ point to an open file)

--print("Handle\_Sys\_Close invoked!\n")

--print("fileDes = ")

--printInt(fileDesc)

--print(".\n")

var

open: ptr to OpenFile

-- ############ NEW CODE ##################

-- Check to see if we're trying to close the Terminal

if currentThread.myProcess.fileDescriptor[fileDesc].kind == TERMINAL

currentThread.myProcess.fileDescriptor[fileDesc] = null

return

endIf

-- ############ NEW CODE ##################

-- Check to see if the index passed in is valid.

-- Can't be greater than or equal to MAX OR less than 0

if fileDesc >= MAX\_NUMBER\_OF\_OPEN\_FILES || fileDesc < 0

return

endIf

open = currentThread.myProcess.fileDescriptor[fileDesc]

currentThread.myProcess.fileDescriptor[fileDesc] = null

--Make sure the file was really open. Return if can't find file

if open == null

return

endIf

fileManager.Close(open)

endFunction

---------------- serialHandlerFunction ---------------------

function serialHandlerFunction()

serialDriver.SerialHandler()

endFunction

----------------------------- SerialInterruptHandler --------------------------

function SerialInterruptHandler ()

--

-- This routine is called when a serial 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.

--

currentInterruptStatus = DISABLED

if serialHasBeenInitialized

serialDriver.serialNeedsAttention.Up()

endIf

endFunction

**------------------------ SerialDriver ----------------------------**

behavior SerialDriver

**--------------------- SerialDriver . Init() -------------------**

method Init()

-- Initialize method for Serial Driver

print( "Initializing Serial Driver...")

serial\_status\_word\_address = SERIAL\_STATUS\_WORD\_ADDRESS asPtrTo int

serial\_data\_word\_address = SERIAL\_DATA\_WORD\_ADDRESS asPtrTo int

serialLock = new Mutex

serialLock.Init()

-- Initialize 'Get' variables

getBuffer = new array of char { SERIAL\_GET\_BUFFER\_SIZE of '\0' }

getBufferSize = 0

getBufferNextIn = 0

getBufferNextOut = 0

getCharacterAvail = new Condition

getCharacterAvail.Init()

-- Initialize 'Put' variables

putBuffer = new array of char { SERIAL\_PUT\_BUFFER\_SIZE of '\0' }

putBufferSize = 0

putBufferNextIn = 0

putBufferNextOut = 0

putBufferSem = new Semaphore

putBufferSem.Init(SERIAL\_PUT\_BUFFER\_SIZE)

serialNeedsAttention = new Semaphore

serialNeedsAttention.Init(0)

serialHandlerThread = new Thread

serialHandlerThread.Init("serialHandlerThread")

serialHandlerThread.Fork(serialHandlerFunction, 0)

serialHasBeenInitialized = true

endMethod

**--------------------- SerialDriver . Put Char() ---------------**

method PutChar(value: char)

--Put a character onto the PutBuffer queue

-- If the buffer is full, this method will block.

-- Otherwise return immediately after buffering the character

-- This will not wait for the I/O to complete

-- If the buffer is full, then block

putBufferSem.Down()

-- Aquire SerialLock

serialLock.Lock()

-- Add character to the next "in spot"

putBuffer[putBufferNextIn] = value

--Adjust putBufferNextIn and BufferSize

putBufferNextIn = (putBufferNextIn + 1) % SERIAL\_PUT\_BUFFER\_SIZE

putBufferSize = putBufferSize + 1

-- Release SerialLock

serialLock.Unlock()

-- Signal 'serialNeedsAttention'

serialNeedsAttention.Up ()

endMethod

**--------------------- SerialDriver . GetChar() ----------------**

method GetChar() returns char

-- Get a character from the GetBuffer queue.

-- If the queue is empty, this will block and wait for the

-- user to type a character.

var

holdChar: char

-- Aquire SerialLock

serialLock.Lock()

-- if getBufferSize == 0, we must wait on getCharacterAvail

if getBufferSize == 0

getCharacterAvail.Wait(& serialLock)

endIf

-- Hold character before adjusting values

holdChar = getBuffer[getBufferNextOut]

-- Adjust getBufferNextOut and getBufferSize

getBufferNextOut = (getBufferNextOut + 1) % SERIAL\_GET\_BUFFER\_SIZE

getBufferSize = getBufferSize - 1

-- Release SerialLock & signal serialNeedsAttention

serialLock.Unlock()

--Return character

return holdChar

endMethod

**--------------------- SerialDriver . SerialHandler() ----------**

method SerialHandler()

-- Everytime a device interrupts the CPU, this will be awakened

-- everytime a character is put in the buffer this will be awakened

-- (1) If a new character has been recieved, the new character must be feteched

-- from the device and moved to the getBuffer

-- (2) If the serial transsmission channel is free and there are more characters

-- waiting in putBuffer to be printed, the outputting must be started

-- (3) If other threads wait on getBuffer (becoming non-empty) and putPuffer (becoming non-full)

var

inChar: char

outChar: char

-- Infinite loop

while true

-- Wait on serialNeedsAttention

serialNeedsAttention.Down()

-- HANDLE INPUT STREAM

-- Check available bit of the device status register

if (\*serial\_status\_word\_address & SERIAL\_CHARACTER\_AVAILABLE\_BIT) == 1

-- Aquire Lock

serialLock.Lock()

-- Handle overflow

if getBufferSize >= SERIAL\_GET\_BUFFER\_SIZE - 1

print("\nSerial input buffer overrun - character '")

printChar(inChar)

print ("' was ignored\n")

else

-- Get the character from serial device data register

inChar = \*(serial\_data\_word\_address+3) asPtrTo char

-- Add it to the next position in the get Buffer

getBuffer[getBufferNextIn] = inChar

-- Adjust variables

getBufferNextIn = (getBufferNextIn + 1) % SERIAL\_PUT\_BUFFER\_SIZE

getBufferSize = getBufferSize + 1

-- Signal to getCharacterAvail

getCharacterAvail.Signal(&serialLock)

endIf

-- Release Lock

serialLock.Unlock()

endIf

-- HANDLE OUTPUT STREAM

-- Check Output Ready bit of the device status register

if (\*serial\_status\_word\_address & SERIAL\_OUTPUT\_READY\_BIT) == 2

-- Aquire the Lock

serialLock.Lock()

-- Check Put Buffer Queue

if putBufferSize != 0

-- Get the character from the Buffer

outChar = putBuffer[putBufferNextOut]

-- Set the device Register with the outPut character

\*serial\_data\_word\_address = outChar

-- Make Adjustments to Put Buffer Values

putBufferNextOut = (putBufferNextOut + 1) % SERIAL\_PUT\_BUFFER\_SIZE

putBufferSize = putBufferSize - 1

-- Wake up any PutChar Threads waiting to add characters to a full buffer

putBufferSem.Up()

endIf

--Release Lock

serialLock.Unlock()

endIf

endWhile

endMethod

endBehavior