Justin Shuck

CS333 – Fall 2014

Proj 6 – Kernel.c

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

----------------------------- Handle\_Sys\_Exit ---------------------------------

function Handle\_Sys\_Exit (returnStatus: int)

-- NOT IMPLEMENTED

ProcessFinish(returnStatus)

endFunction

----------------------------- Handle\_Sys\_Join ---------------------------------

function Handle\_Sys\_Join (processID: int) returns int

-- Identify the child process, make sure that the PID that is passed

-- in is the PID of a valid process (and that it is really a child of this process)

-- If it is not than we return '-1' to the caller. Then we can call Wait ForZombie

-- and return whatever it returns

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

--print("processID = ")

--printInt(processID)

--print("\n")

var

i: int

child: ptr to ProcessControlBlock

parent: ptr to ProcessControlBlock

childsExitStatus: int

parent = currentThread.myProcess

-- Identify the child Process.

-- Run validation to ensure that we are grabbing the correct process

for i = 0 to MAX\_NUMBER\_OF\_PROCESSES-1

child = &processManager.processTable[i]

if child.pid == processID && child.parentsPid == parent.pid && child.status != FREE

-- Wait for it to terminiate and get its exidCode when it returns

childsExitStatus = processManager.WaitForZombie(child)

return childsExitStatus

endIf

endFor

return -1

endFunction

----------------------------- Handle\_Sys\_Fork ---------------------------------

function Handle\_Sys\_Fork () returns int

-- Allocate and set up new Thread and ProcessControlBlock objects

-- Make a copy of the address space

-- Invoke Thread.Fork to start up the new processs thread

-- return the childs pid

var

newPCB: ptr to ProcessControlBlock

oldPCB: ptr to ProcessControlBlock

newThread: ptr to Thread

ignore: int

i: int

oldUserPC: int

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

-- Enable Interrupts

ignore = SetInterruptsTo(DISABLED)

-- Get new thread and PCB and initialize them

newPCB = processManager.GetANewProcess()

oldPCB = currentThread.myProcess

newThread = threadManager.GetANewThread()

-- Initialize PCB

newPCB.parentsPid = oldPCB.pid

-- Initialize thread

newThread.name = currentThread.name

newThread.status = newPCB.status

newThread.myProcess = newPCB

newPCB.myThread = newThread

-- Grab the values in the user register and store a copy

-- in the new Thread

SaveUserRegs(&newThread.userRegs[0])

-- Re-enable inturrupts

ignore = SetInterruptsTo(ENABLED)

--TODO: Must share OpenFiles with parent

-- We then need to reset the system stack top and

--ensure that no other threads will touch our user/new stack.

newThread.stackTop = &(newThread.systemStack[SYSTEM\_STACK\_SIZE-1])

-- Next we need to allocate the new frames for this address space

frameManager.GetNewFrames(& newPCB.addrSpace, oldPCB.addrSpace.numberOfPages)

-- Copy all the pages!

for i = 0 to oldPCB.addrSpace.numberOfPages-1

if oldPCB.addrSpace.IsWritable(i)

newPCB.addrSpace.SetWritable(i)

else

newPCB.addrSpace.ClearWritable(i)

endIf

MemoryCopy( newPCB.addrSpace.ExtractFrameAddr(i),

oldPCB.addrSpace.ExtractFrameAddr(i),

PAGE\_SIZE)

endFor

-- Get the User PC (That is buried in the system stack of the current Process)

-- This value should point to the instruction following the syscall

oldUserPC = GetOldUserPCFromSystemStack()

--Fork a new thread and have it 'resume execution in user-land'

newThread.Fork(ResumeChildAfterFork, oldUserPC)

return newPCB.pid

endFunction

----------------------------- Handle\_Sys\_Yield ---------------------------------

function Handle\_Sys\_Yield ()

-- NOT IMPLEMENTED

-- Not really a need for a Yield syscall in any OS that has preemptive scheduling,

-- but it can be used to make sure that the other processes are really running.

currentThread.Yield()

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

endFunction

----------------------------- ResumeChildAfterFork ------------------------

function ResumeChildAfterFork(initPC: int)

-- This new thread should:

-- \* Initilize the user registers

-- \* Initilize the user and system stacks

-- \* Figure out whfere in the user's address space to reurn to

-- \* invoke BecomeUserThread and jump into the user-level processID

var

ignore: int

initSystemStackTop: int

initUserStackTop: int

-- Begin by disabling interrupts

ignore = SetInterruptsTo(DISABLED)

-- set the page table registers to point to the process's page

-- table and set the user registers

currentThread.myProcess.addrSpace.SetToThisPageTable()

RestoreUserRegs(&currentThread.userRegs[0])

-- Any future interrupts will save the user regs to the thread

currentThread.isUserThread = true

-- Reset system stake top and invoke 'BecomeUserThread'

initSystemStackTop = (& currentThread.systemStack[SYSTEM\_STACK\_SIZE-1]) asInteger

initUserStackTop = currentThread.userRegs[14]

BecomeUserThread(initUserStackTop, initPC, initSystemStackTop)

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.

--

var

proc: ptr to ProcessControlBlock

ignore: int

-- Save exitStatus

currentThread.myProcess.exitStatus = exitStatus

-- Disable Interrupts

ignore = SetInterruptsTo(DISABLED)

-- Disconnect the PCB from the Thread

proc = currentThread.myProcess

currentThread.myProcess = null

proc.myThread = null

currentThread.isUserThread = false

-- Close any open files (FOR NEXT PROJECT)

--Re-enable interrupts

ignore = SetInterruptsTo(ENABLED)

-- Return all frames to the Free Pool and turn process into ZOMBIE

frameManager.ReturnAllFrames( &proc.addrSpace)

processManager.TurnIntoZombie(proc)

--Terminate thread (Parent will deal with the Zombie)

ThreadFinish()

endFunction

---------- ProcessManager . TurnIntoZombie ----------

method TurnIntoZombie (p: ptr to ProcessControlBlock)

-- Passed a pointer to a process to turn it into a zombie (dead but not gone), so that

-- its exitStatus may be retrieved if needed by its parent

-- Steps:

-- 1. Lock the process manager (since we will be messing with other PCBs)

-- 2. Identify the processes who are zombies. These children are now no longer

-- needed so for each zombie child, change its status to 'FREE' and add it back to

-- the PCB free list. Signal 'aProcessBecameFree' since other threads may be waiting for

-- a free PCB.

-- 3. Identify p's parents (The parent may be terminiated, so they may not have one)

-- 4. If p's parent is 'ACTIVE' then the method must turn p into a zombie. Execute a broadcast on

-- the aProcessDied condition, because the parent of p may be waiting for p to exit.

-- 5. Otherwise (our parent is a zombie or non-existent) we do not need to turn p into a zombie, so just change p's

-- status to 'FREE', add it to the PCB free list, and signal the aProcessBecameFree condtion variable

-- 6. Unlock the process manager

var

i: int

child: ptr to ProcessControlBlock

parent:ptr to ProcessControlBlock

-- 1.Lock the process Manager

processManagerLock.Lock()

-- 2. Identify zombies and Free them

for i=0 to MAX\_NUMBER\_OF\_PROCESSES-1

child= &processTable[i]

if child.parentsPid == p.pid && child.status == ZOMBIE

child.status = FREE

freeList.AddToEnd(child)

aProcessBecameFree.Signal(& processManagerLock)

endIf

endFor

-- 3. Identify p's parents

parent = null

for i=0 to MAX\_NUMBER\_OF\_PROCESSES-1

if processTable[i].pid == p.parentsPid

parent = &processTable[i]

endIf

endFor

-- 4. If p's parents Active (turn to zombie)

-- 5. Otherwise our parents non-existent/not Active

if parent && parent.status == ACTIVE

p.status = ZOMBIE

aProcessDied.Broadcast(&processManagerLock)

else

p.status = FREE

freeList.AddToEnd(p)

aProcessBecameFree.Signal(&processManagerLock)

endIf

--6. Unlock process manager

processManagerLock.Unlock()

endMethod

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

method WaitForZombie (proc: ptr to ProcessControlBlock) returns int

-- The method waits for a process to turn into a zombie.

-- The exit status is saved and ads the PCB back to the freelist

-- The exit status is returned.

var

exitStatusToReturn: int

-- 1. Lock the Process Manager

processManagerLock.Lock()

-- 2. Wait until the status of proc is ZOMBIE

while proc.status != ZOMBIE

aProcessDied.Wait(& processManagerLock)

endWhile

-- 3. Get procs exit status

exitStatusToReturn = proc.exitStatus

-- 4. Change Proc's status to FREE and Add the PCB back to the list. S

-- signal the 'aProcessBecameFree' variable

proc.status = FREE

freeList.AddToEnd(proc)

aProcessBecameFree.Signal(& processManagerLock)

-- 5. Unlock the process manager

processManagerLock.Unlock()

-- 6. Return exitStatus

return exitStatusToReturn

endMethod