Project 5
Discussion
Threads and Processes - Design Options

class ProcessControlBlock
    ...
    myThread: ptr to Thread

class Thread
    ...
    myProcess: ptr to ProcessControlBlock
Threads and Processes - Design Options

class ProcessControlBlock
  ...
  myThread: Thread
class ProcessControlBlock
    ...
    myThread: Thread

pcb = ...
... = pcb.myThread.stackTop
class ProcessControlBlock
    ...
    myThread: Thread

pcb = ...
... = pcb.myThread.stackTop

var th: Thread
    th = pcb.myThread
    ... = th.stackTop
## Threads and Processes - Design Options

```plaintext

class ProcessControlBlock
    ...
    myThread: Thread

pcb = ...
... = pcb.myThread.stackTop

var th: Thread
    th = pcb.myThread
    ... = th.stackTop

var th: ptr to Thread
    th = & pcb.myThread
    ... = th.stackTop
```

<table>
<thead>
<tr>
<th>PCB</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid</td>
<td></td>
</tr>
<tr>
<td>parentsPid</td>
<td></td>
</tr>
<tr>
<td>status</td>
<td></td>
</tr>
<tr>
<td>myThread</td>
<td></td>
</tr>
<tr>
<td>regs</td>
<td></td>
</tr>
<tr>
<td>stackTop</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td></td>
</tr>
<tr>
<td>status</td>
<td></td>
</tr>
<tr>
<td>systemStack</td>
<td></td>
</tr>
<tr>
<td>.isUserThread</td>
<td></td>
</tr>
<tr>
<td>userRegs</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td></td>
</tr>
<tr>
<td>exitStatus</td>
<td></td>
</tr>
<tr>
<td>addrSpace</td>
<td></td>
</tr>
<tr>
<td>fileDescriptor</td>
<td></td>
</tr>
</tbody>
</table>
class ProcessControlBlock
    
    myThread: Thread
    
    pcb = ...
    ...

var th: Thread
    th = pcb.myThread
    ...

var th: ptr to Thread
    th = & pcb.myThread
    ...

But

th.stack = ...!

is dangerous!
Another Option:

- Get rid of class Thread altogether
- Include all Thread fields in PCB
Threads and Processes - Design Options

Multiple Threads per Process

- PCB
  - threadList
    - Thread
      - next
        - Thread
          - next
            - Thread
              - next
                - Thread
                  - next
                    - null

Threads and Processes - Design Options

Multiple Threads per Process

```
PCB
xxx
xxx
xxx
threadList
xxx

Thread
next
myPCB
xxx
xxx

Thread
next
myPCB
xxx
xxx

Thread
next
myPCB
xxx
xxx

Thread
next
null
xxx
xxx
```
Class Thread

fields

regs: array [13] of int    -- Space for r2..r14
stackTop: ptr to void     -- Current system stack top ptr
name: ptr to array of char
status: int                -- JUST_CREATED, READY,
                            -- RUNNING, BLOCKED, UNUSED
initialFunction: ptr to function (int)
initialArgument: int
systemStack: array [SYSTEM_STACK_SIZE] of int
isUserThread: bool
userRegs: array [15] of int  -- Space for r1..r15
myProcess: ptr to ProcessControlBlock
Class Thread

methods

**Init** (n: ptr to array of char)
**Fork** (fun: ptr to function (int), arg: int)
**Yield** ()
**Sleep** ()
**CheckOverflow** ()
**Print** ()
Class ProcessControlBlock

fields

  pid: int  -- The process ID
  parentsPid: int  -- The pid of the parent of this process
  status: int  -- ACTIVE, ZOMBIE, or FREE
  myThread: ptr to Thread  -- Each process has one thread
  exitStatus: int  -- The value passed to Sys_Exit
  addrSpace: AddrSpace  -- The logical address space
  fileDescriptor: array [MAX_FILES_PER_PROCESS] of ptr to OpenFile
Class ProcessControlBlock

methods
    Init ()
    Print ()
Class ThreadManager

fields

threadTable: array [MAX_NUMBER_OF_PROCESSES] of Thread

tfreeList: List [Thread]

threadManagerLock: Mutex

-- These synchronization objects

aThreadBecameFree: Condition

-- apply to the "freeList"
Class ThreadManager

methods

    Init ()
    Print ()
    GetANewThread () returns ptr to Thread
    FreeThread (th: ptr to Thread)
Class ProcessManager

fields

processTable: array [MAX_NUMBER_OF_PROCESSES] of ProcessControlBlock

processManagerLock: Mutex

-- These synchronization objects

aProcessBecameFree: Condition

-- apply to the "freeList"

freeList: List [ProcessControlBlock]

aProcessDied: Condition

-- Signalled for new ZOMBIEs

nextPid: int
Class ProcessManager

methods

- **Init** ()
- **Print** ()
- **GetANewProcess** () returns ptr to ProcessControlBlock
- **FreeProcess** (p: ptr to ProcessControlBlock)
- **TurnIntoZombie** (p: ptr to ProcessControlBlock)
- **WaitForZombie** (proc: ptr to ProcessControlBlock) returns int
The “Stub” File System

Model of disk
  Sequence of sectors

The File System
  Sector zero contains the directory
  Only one directory (a “flat” file system)

Files:
  Name
  Starting sector
  Size (number of bytes)
  All sectors for a file are contiguous
The “Stub” File System

directory sector

0 1 2 3 4 5 6 7

MyFile  file1  UserProgram  SwapFile
The “diskUtil” Utility

Another BLITZ tool

Initialize the file system, create empty directory...
diskUtil  -i

List the directory of the BLITZ “DISK”...
diskUtil  -l

Copy a file from Unix to the BLITZ “DISK”...
diskUtil  -a  UnixFileName  BlitzFileName

Get help info...
diskUtil  -h
File-Related Classes

Class “FileControlBlock” (FCB)
Contains info kernel needs to read/write to a file
Where on disk
A buffer area to use
Length of file
Must have only one FCB per file

Class “FileManager”
A monitor
Where all the methods are
Class FileControlBlock

fields

  fcbID: int
  numberOfUsers: int
      -- count of OpenFiles pointing here
  startingSectorOfFile: int  -- or -1 if FCB not in use
  sizeOfFileInBytes: int
  bufferPtr: ptr to void      -- ptr to a page frame
  relativeSectorInBuffer: int -- or -1 if none
  bufferIsDirty: bool        -- Set to true when buffer is modified
Class FileControlBlock

methods

Init ()
Print ()
Class OpenFile

Contains:
  • Ptr to the FCB for this file
  • The “current position”
  • Processes point to “OpenFile”’s, not “FCB”’s
The Unix Open-File Model
File Descriptors

PCB

pid
parentsPid
status
fileDescriptor

0
1
2
3
...
MAX-1

null
null
null
null

OpenFile
pos=17

OpenFile
pos=52

OpenFile
pos=350

FCB

FCB

FCB
Class OpenFile

fields

  **currentPos**: int       -- 0 = first byte of file
  **fcb**: ptr to FileControlBlock  -- null = not open
  **numberOfUsers**: int  -- count of Processes pointing here
Class OpenFile

methods

Print ()

ReadBytes (targetAddr, numBytes: int) returns bool
    -- returns true if all okay

ReadInt () returns int

LoadExecutable (addrSpace: ptr to AddrSpace) returns int
    -- returns -1 if problems
Class FileManager

fields

fileManagerLock: Mutex

fcbTable: array [MAX_NUM_FILE_CONTROL_BLKS] of FileControlBlock

anFCBBecameFree: Condition

fcbFreeList: List [FileControlBlock]

openFileTable: array [MAX_NUM_OPEN_FILES] of OpenFile

anOpenFileBecameFree: Condition

openFileFreeList: List [OpenFile]

directoryFrame: ptr to void
Class FileManager

methods

  Init ()
  Print ()
  FindFCB (filename: String) returns
    ptr to FileControlBlock -- null if errors
  Open (filename: String) returns ptr to OpenFile
    -- null if errors
  Close (open: ptr to OpenFile)
  Flush (open: ptr to OpenFile)
  SynchRead (open: ptr to OpenFile,
    targetAddr, bytePos, numBytes: int) returns bool
  SynchWrite (open: ptr to OpenFile,
    targetAddr, bytePos, numBytes: int) returns bool
Class DiskDriver

fields

DISK_STATUS_WORD_ADDRESS: ptr to int
DISK_COMMAND_WORD_ADDRESS: ptr to int
DISK_MEMORY_ADDRESS_REGISTER: ptr to int
DISK_SECTOR_NUMBER_REGISTER: ptr to int
DISK_SECTOR_COUNT_REGISTER: ptr to int
semToSignalOnCompletion: ptr to Semaphore
semUsedInSynchMethods: Semaphore
diskBusy: Mutex
currentInterruptStatus = DISABLED
-- print ("DiskInterruptHandler invoked!\n")
if diskDriver.semToSignalOnCompletion
    diskDriver.semToSignalOnCompletion.Up()
endIf
Class DiskDriver

methods
  Init ()
  SynchReadSector
      (sectorAddr, numberOfSectors, memoryAddr: int)
  StartReadSector
      (sectorAddr, numberOfSectors, memoryAddr: int,
       whoCares: ptr to Semaphore)
  SynchWriteSector
      (sectorAddr, numberOfSectors, memoryAddr: int)
  StartWriteSector
      (sectorAddr, numberOfSectors, memoryAddr: int,
       whoCares: ptr to Semaphore)
Building a User-level Program

UserSystem.h
UserSystem.c
UserRuntime.s
MyProgram.h
MyProgram.c

%asm  UserRuntime.s
%kpl  UserSystem -unsafe
%asm  UserSystem.s
%kpl  MyProgram -unsafe
%asm  MyProgram.s
%lddd UserRuntime.o UserSystem.o MyProgram.o
-o MyProgram
Put Executable on the BLITZ Disk

% diskUtil –i
% diskUtil -a temp1 MyFileA
% diskUtil -a temp2 MyFileB
% diskUtil -a MyProgram MyProgram
% diskUtil –l

<table>
<thead>
<tr>
<th>StartingSector</th>
<th>SizeInSectors</th>
<th>SizeInBytes</th>
<th>FileName</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>8192</td>
<td>&lt; directory &gt;</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8192</td>
<td>MyFileA</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>17000</td>
<td>MyFileB</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>60264</td>
<td>MyProgram</td>
</tr>
</tbody>
</table>

The disk is emulated with a Unix file. The filename (“DISK”) is implicit.
The First User-Level Program

MyProgram.h

header MyProgram
uses UserSystem
functions
main()
endHeader

MyProgram.c

code MyProgram
function main()
print("My user-level program is running!
")
Sys_Shutdown()
endFunction
endCode
System Calls (Wrapper functions)

function `Sys.Exit` (returnStatus: int)
function `Sys.Shutdown` ()
function `Sys.Yield` ()
function `Sys.Fork` () returns int
function `Sys.Join` (processID: int) returns int
function `Sys.Exec` (filename: String) returns int
function `Sys.Create` (filename: String) returns int
function `Sys.Open` (filename: String) returns int
function `Sys.Read` (fileDesc: int, buffer: ptr to char, sizeInBytes: int) returns int
function `Sys.Write` (fileDesc: int, buffer: ptr to char, sizeInBytes: int) returns int
function `Sys.Seek` (fileDesc: int, newCurrentPosition: int) returns int
function `Sys.Close` (fileDesc: int)
Wrapper Function

```haskell
function Sys_Read (fileDesc: int,
    buffer: ptr to char,
    sizeInBytes: int) returns int
return DoSyscall (SYSCALL_READ,
    fileDesc,
    buffer asInteger,
    sizeInBytes,
    0)
endFunction
```
DoSyscall

external **DoSyscall** (funCode, arg1, arg2, arg3, arg4: int) returns int

DoSyscall:

```plaintext
load [r15+8],r1  ! Move arg1 into r1
load [r15+12],r2 ! Move arg2 into r2
load [r15+16],r3 ! Move arg3 into r3
load [r15+20],r4 ! Move arg4 into r4
load [r15+4],r5  ! Move funcCode into r5
syscall r5         ! Do the syscall
store r1,[r15+4]  ! Move result from r1 onto stack
ret               ! Return
```
Two copies (User, Kernel) must match!

def

enum
    SYSCALL_EXIT = 1,
    SYSCALL_SHUTDOWN,
    SYSCALL_YIELD,
    SYSCALL_FORK,
    SYSCALL_JOIN,
    SYSCALL_EXEC,
    SYSCALL_CREATE,
    SYSCALL_OPEN,
    SYSCALL_READ,
    SYSCALL_WRITE,
    SYSCALL_SEEK,
    SYSCALL_CLOSE
Virtual Address Space

- A Page for “environment” data
- Pages for the text segment
- Pages for the data segment
- Pages for the BSS segment
- Pages for the user’s stack

Typical:
- 0 environment pages
- 2 text pages
- 1 data page
- 0 BSS pages
- 1 stack page

4 pages \(\rightarrow\) 32Kbyte Virtual Address Space