Secure Operating System Design and Implementation
Ethos Tour

Jon A. Solworth

Dept. of Computer Science
University of Illinois at Chicago

February 1, 2011

Part I

Ethos Tour Overview

Overview

- This set of slides is a tour through the Ethos source code
- Ethos obtains significant advantage by implementing on top of Xen
- Much simpler I/O device interface
- Availability of a paired OS for performing OS services such as file system.

Architectural Components

- An operating system can be targeted to different architectures
- Ethos is designed for 64-bit architectures
- Which means its designed for architectures with considerable resources, especially memory
- Ethos current implementation is 32-bit
- Ethos will probably have two ultimate architectures, x86 and ARM.

Architectural Components

- An OS is implemented in two layers
- The base abstractions using machine dependent layer
- On top of that a far larger machine independent layer is built
- To port to a different architecture, need only implement a machine dependent layer
- This assumes that the layering is properly done
- And that takes a while, so we'll concentrate on a single architecture for now

Part II

Architectural dependent components

Architectural dependent components

The architectural dependant components include:

- Types are used to express size requirements of different components of the OS
- The compiler generates machine language for the architecture (only non-privileged instructions)
- The machine dependent OS components include:
  - The privileged instructions
  - Architectural dependent data structures
  - Bits of glue logic (such as for system calls)
- Since Ethos is built on top of Xen, it does not have any privileged instructions

Architectural dependent source files

arch/x86 contains the code for both 32-bit and 64-bit x86 architectures

- The 32-bit code works in non-PAE mode
- The PAE mode and 64-bit code are residuals from Mini-OS
- `debug.c`: performs a register dump on crash
- `archPageTable.c` multilevel page table operations (exports a flat page table with R/W permissions)
- `minios-x86_32.lds`: 32-bit load script for the OS
- `sched.c` scheduling and process code.
- `setup.c` various bits of code for Ethos start up
- `time.c` time using Xen's time facilities
- `trap.c` setting up traps and interrupts using Xen events
- `x86_32.S` assembly glue code for traps, etc.

Architectural dependent include files

include/ethos/x86 contains the architecture dependent files

- `archMemory.h` exports a flat page table info to architecture independent portion of OS
- `archPageTable.h` is the internal hierarchical page table
- `arch_debug.h` interface to dump registers
- `arch_elf.h` interface to elf object loader for user space
- `arch_sched.h` interface to scheduling and process primitives
- `bits.h` low-level locking primitives
- `cpu.h` low-level CPU definitions, such as interrupt numbers, etc.
- `ldsyms.h` symbols set by the linker script
- `synchro.h` various types of memory barriers defined
- `x86_32/hypercall-x86_32.h` hypercalls for the x86 32-bit architecture

Part III

Architecture-independent components

Architecture-independent components

Primary directories

- **console** Xen console devices. This is the code that processes Xen error messages
- **net** Xen pseudo ethernet driver code.
- **file** Filesystem implementation
- **mm** Memory management
- **process** process support, including process scheduling
- **rpc** remote procedure call support for communicating with the shadowdaemon
- **syscall** system call interface
- **userspace** contains the shadow daemon

Part IV

Xen Components

Xen components

Ethos is not intended to be run stand alone, and hence is dependent on Xen.

- Parts of the Xen-interfacing components in the architecturally dependent components
- Others are in the `Xen` directory
- The include files for Xen are in `include/xen`
- The Xen include files are currently from Xen version 3.2, and of course follow Xen coding style

Xen

The `xen` directory includes:

- `xenPageTable.c` includes procedures to:
  - Switch the current page table being used (this is part of a context switch)
  - To pin and unpin pages. Pinned pages are maintained in the Xen hypervisor to minimize checking of page reference.
- `xenSchedule.c` schedule the Ethos OS vs. other OSs running under Xen.
- `xenGrant.c` grant pages are pages which are shared between multiple OSs. Xen provides mechanism by which pages can be shared and sharing changing. This is used for device drivers (in particular, ethernet pseudo devices).
- `xen_hypervisor.c` Hypervisor commands which should be moved to other files.
- `xenbus.c` provides Xenbus/Xenstore interface to store and retrieve name-value pairs.

Part V

Include file overview

Include directories

include/ethos	include files which are used only within the Ethos kernel
include/userspace	this are include files which are used both within the Ethos kernel and by Dom0 and/or Ethos user space
include/xen	include files from Xen

Part VI

Initialize

Initialize components

Initialize starts up the kernel

- `kernelInitialize.c` initializes the Ethos kernel
- `initial_store.c` is an initial file system kept in RAM. We will be getting rid of this component, as the file system stabilizes.

KernelInitialize

Steps to initialize (boot start up code):

1. `arch_init` Architectural specific initialization
2. `trap_init` Initialize interrupt vector
3. `xen_event_init` initialize Xen events
4. `__sti` enable interrupts
5. `consoleInit` initialize the console
6. `setup_xen_features`
7. `memoryInit` initialize virtual memory
8. `xen_grant_init` initialize grant memory for pseudo-devices
9. `init_time` initialize wall clock time computation

KernelInitialize (cont'd)

10. `eventInit` initialize Ethos events
11. `xenbus_init`
12. `netInit` initialize the network interface
13. `pairedConnection` set up connection to Dom0 shadowdaemon
14. `fileSystemInit` initialize file system
15. `initstor_init` initialize the initstor
16. `debug_init` read in debugging info for the Ethos kernel
17. `scheduleInit` creates the `init` process, after which Ethos is up and running.

Part VII

Shadow daemon

Shadow daemon

`userspace/dom0/shadowdaemon` contains the userspace code for the shadowdaemon in Dom0

- The shadow daemon is a Dom0 process
- It performs services for the Ethos kernel
- Most notably, file system operations
- This includes read, write, search, and create operations
- It'll also be useful for networking
- Eventually, we'll eliminate the shadowdaemon,
- But as an implementation strategy, it enables us to scaffold and build the syscall interface faster.

Part VIII

Remote Procedure Call

Remote Procedure Call (RPC)

- Ethos deals with typed data
  - In the file system,
  - for inter-process communication, and
  - across the network.
- The most flexible way of implementing this is RPC
- RPC right now is used between the Ethos kernel and Dom0 shadowdaemon
- That means that the RPC code needs to be compiled in both environments

RPC components

- `netInterface.c` provides the interface to network device. This is the "logical" interface, the "physical" interface is either Xen's net front or UNIX sockets.
- `tunnel.c` provides a logical conduit between two network interfaces
- `connection.c` is a logical connection. Each connection is associated with a tunnel, a tunnel can support an arbitrary number of connections.
- `procedure.c` stores and looks up the RPC specification
- `rpcInterface.c` describes how to marshall and unmarshall RPC types
- `rpcType.c` describes the primitive types and vectors of the primitive types.
- `packet.c` connections are implemented with a sequence of packets

Procedure-dependent components

- RPC allows an interface to be defined consisting of a set of procedure
- `ethosRpc.c` describes the remote procedures. It consists of two types:
  - `procedureDispatcher` invokes a procedure. It has a case statement with an entry for each procedure.
  - `rpcInit` describes the remote procedures in a table format.
- It is possible to automatically generate ethosRpc.c from a short description, but to do that, we would need an IDL compiler. Unfortunately, we haven't gotten around to writing one yet.

Part IX

Devices

Devices

- Ethos devices are pseudo devices provided by Xen
- There are three principle pseudo devices in Xen

  console the OS console supports I/O

  network Xen can export a number of ethernet network devices to which ethernet packets can be sent/received

  block block devices, which can be read and written, are the abstraction for disk drives
- Since Ethos's file system is implemented on Dom0, we don't currently use the block pseudo-device

Console

In `console/console.c`

- Provides `printk`, the kernel printf equivalent

in `console/xencons_ring.c`

- Contains the interface to a 2 kilobyte Xen-based console buffer
- Not invoked directly, but only through console.c

Net

There are three externally facing functions in `net/net.c`

1. `netInit` initialize network interfaces
2. `netSend` send a packet to the network
3. `netIncoming` process incoming packets

`net/xenNetInterface.c` are the Xen interface. There needs to be more code moved from net.c to xenNetInterface.c.

Part X

Memory management

Memory Management

- `kmap.c` these are reserved mapping in the kernel so that Ethos doesn't run out of memory at inopportune times.
- `memoryInit.c` initializes the memory subsystem
- `pageAllocator.c` allocate pages using a buddy allocator. At kernel initialization, all unallocated pages are put into the buddy allocator.
- `pageFault.c` handles page faults
- `pageTable.c` exports a flat page table abstraction to rest of OS

Memory Management

- `pmem.c` this manages memory for a single process.
- `slob.c` sub-page allocator
- `vaddr.c` checks whether virtual addresses are mapped into the virtual address space.
- `vma.c` manages process regions
- `vmalloc.c` a second buddy allocator

Part XI

File system

File system

- `directory.c` directory operations to open, create, and delete directory entries.
- `file.c` file operations to read and write
- `fileSystem.c` create initial in memory data structures to access file system.
- `mode.c` permission string to/from integer values for access type
- `resourceClose.c` close a resource descriptor
- `terminal.c` terminal read and write

Part XII

Process

Process

- `process.c` manages processes, including creating, deleting, and blocking processes.
- `fileDescriptor.c` is the per-process file descriptors
- `schedule.c` is scheduling for processes, which happens when a process blocks or is coming out of a (completed) system call.
- `elf.c` loads elf file into a process on exec or creation of init process.

Part XIII

Syscalls

Syscalls

- System calls are the invocation of OS facilities from user space
- Ethos syscalls pass the values in the registers
  - Syscall number
  - Pointer to input parameter structure
  - Pointer to output parameter structure
- System calls must check that parameters are valid
- Including that pointers are to allocated memory in user space
- In addition, Ethos returns values whose size is not known when the call is made.
- Note that `syscall/syscall.c` is the kernel side of system calls, the user side is in sub-directories `userspace/ethos/` called `libsyscall`

Syscall macros

getArgs(_var) gets the argument structure and puts the result in _var. This retrieves the fixed sized arguments

getMemStruct(_memStruct) gets a variable size MemStruct, which is a pair of size and pointer. The memStruct is one of the arguments retrieved by getArgs. Allocates space in the kernel for the variable sized structure.

putReturn(_var) put the fixed sized components of the value to be returned to user space.

putMemStruct(_memStruct) puts the variable sized components of return value back to user space.

These macros branch to `done` if the macros fail. The variable size putMemStruct is mostly used in `retire`.

Part XIV

Abstraction

Abstraction

- Ethos has a number of abstractions used to build higher level OS mechanisms.
- These include
  - `handle.c` Handles are 64-bit values which map to objects and are guaranteed to never repeat.
  - `event.c` Events which provide support for asynchronous I/O. Note that these are Ethos Events, not Xen Events.
  - `eventTree.c` EventTrees which provide blocking on and/or trees of events
  - `timer.c` timers events which represent time
  - `resource.c` resources such as file descriptors, directory descriptors, and their management on significant events.