WHAT IS IGUANA?

- Remember, L4 is a “strict” microkernel:
  - does not provide any services
  - does not provide policies (or only very few)
  - provides mechanisms
- L4 aspires to be generic kernel, suitable for all kinds of systems
- Almost any system requires a set of core services:
  - process management
  - memory management
  - security management
    - based on some system-wide policies
- Iguna provides these (or at least more tools for providing them)
- Iguna is designed for use in embedded systems
**WHAT DOES IGUANA PROVIDE?**

- Convenient way of using L4 primitives
  - OO-style method invocations instead of explicit IPC calls
  - IDL compiler for automatic generation of stubs
- Protection framework for access rights management
  - capability based, flexible
  - able to model most standard security models
- Virtual memory management
  - allocation, deallocation, sharing, ...
  - single-address-space view, supporting FASS on ARM
- Protection-domain (process) management
- Thread management

**OUTLINE**

- Introduction
- Iguana concepts, abstractions and mechanisms
- Iguana API
- Kenge

**IGUANA : BASIC APPROACH**

- Basic idea: single address space (SAS)
  - eases sharing of data
    - minimises copying
    - no problems with pointers
- Per-process protection domains
  - enforce security policy
    - any access is subject to access control
    - do not interfere with sharing
- SAS layout supports fast-address-space switching on ARM
  - avoids AS overlaps for non-shared date without use of PID relocation
  - advantage: 1MB domain granularity instead of 32MB for PID relocation
  - less internal fragmentation

**IGUANA CONCEPTS**

- Memory section
  - unit of VM allocation and protection
  - can be an encapsulated object with methods and data
- Thread
  - execution abstraction, as in L4
- Server
  - thread associated with memory section
  - invoked through methods with well-defined interfaces
- Protection domain
  - defines access and resource rights of a thread
  - corresponds to a process in traditional OS
Iguana Concepts

- **Session**
  - client-server (or peer-to-peer) communication channel
  - amortises authentication cost over many invocations

- **Capability**
  - represents access rights
  - basis of protection

- **Resource token**
  - represents resource usage right
  - basis of resource management

- **External Space**
  - address space extern to Iguana's SAS
  - for legacy support and large processes

Iguana Philosophy

- Small and lightweight
  - geared towards embedded systems
  - allow optimal utilisation of hardware

- Strong yet unintrusive protection
  - hide protection machinery from most apps
  - able to emulate most standard protection models

- Support for resource management
  - in principle, although it isn’t implemented yet!

- Legacy support
  - designed to run Linux server

Outline

- Introduction
- Iguana concepts, abstractions and mechanisms
- Iguana API
  - Note: Under development, details still changing
- Kenge
**OBJECTS**

- Six kinds of objects
  1. memory sections
  2. threads
  3. protection domains (PDs)
  4. sessions
  5. resource tokens (restoks)
     - not yet implemented, not covered here
  6. external spaces
     - not full Iguana objects
     - serve as proxies for non-Iguana objects
- Access controlled by capabilities

**OBJECTS: COMMONALITIES**

- Objects have a unique name — object ID (OID)
  - OIDs are addresses in Iguana’s SAS
  - only for memory sections does this address correspond to actual memory
- Objects have methods that can be invoked
  - one method that exists for all objects: destroy
  - each kind of object has a set of pre-defined methods
- Objects are created by invoking constructor on a PD:
  - `kind_cap = pd->new_kind(args);`
- Methods are grouped into interfaces
  - interfaces also have unique IDs (IIDs) that are OID + interface number
  - interfaces have capabilities
  - grant rights to invoke an interface’s methods
  - all pre-defined methods belong to separate interfaces
    - i.e., access is individually protected

**CAPABILITIES**

- A capability is a token that confers some access right(s)
- Two kinds of capabilities in Iguana:
  - master capability
    - created when an object is created
    - confers rights on all methods of object
    - allows creation of further capabilities
  - invocation capability
    - created when an interface is created
    - confers right to invoke methods of a single interface
- Capabilities are only active if stored in PD’s capability lists
  - details later

**CAPABILITIES**

- Memory sections represent virtual memory
  - allocation of a certain amount of virtual memory:
    - `mem_cap = pd->new_mem(size);`
- Memory sections are the only objects that support user-defined methods
  - others have pre-defined (standard) methods only
- Used to provide encapsulated services:
  - service = memory (data) + server (thread) + methods
**MEMORY SECTIONS...**

- To create a service:
  - register a server thread on memory section
    ```
    base->new_server(thread_id);
    ```
  - `base` is the base address (OID) of the memory section
  - register interfaces (user-defined methods)
    ```
    base = iid->new_cap();
    ```
  - `iid` refers to number of new interface
  - Registering interfaces supports user-defined methods
    - remember: each interface can have one or more methods
    - interface number only interpreted by server
    - similarly, the method number is an opcode delivered to the interface
  - IIDs and method numbers allocated by system implementor
    - part of the service’s interface protocol

**MEMORY SECTIONS: PSEUDO METHODS**

- Read (R), write (W), execute (X) are logically considered methods
  - subjects them to same protection mechanisms as other methods
  - no actual methods exist corresponding to those operations
- Further pseudo-method is clist (C)
  - needed for manipulating protection domains
  - more details later

**THREADS**

- Iguana threads are essentially L4 threads:
  - threads within same PD operated on by plain L4 syscalls
    - correspond to local L4 threads (i.e., same L4 AS)
    - ExchangeRegisters, IPC
  - direct IPC to non-local threads is not allowed
    - use method invocations (corresponding to server thread)
    - presently not enforced by Iguana
    - requires enhancements to L4 (forthcoming API) to do efficiently
      - will provide attribute to ensure enforcement (at a cost)
- Certain operations require privileges
  - e.g. thread creation and deletion done by privileged L4 ThreadControl() call
- Done by Iguana on invocation of appropriate methods

**THREAD OPERATIONS**

- Thread creation:
  ```
  thread_cap = pd->new_thread(l4_tid);
  ```
  - returns two kinds of thread IDs
    - Iguana thread ID (tid), part of the `thread_cap`
      - used for protection and other Iguana-specific purposes
    - L4 thread ID (l4_tid)
      - used for L4 syscalls
  - New thread created inactive
    - can be activated by:
      - L4 syscall ExchangeRegisters() (local threads only)
      - Iguana method tid->start(ip, sp)
**Thread Operations...**

- Obtain L4 thread ID
  
  \[ l4tid = tid->l4_tid(); \]

- Obtain own thread ID
  
  \[ tid = myself(); \]

- Obtain protection domain of thread
  
  \[ pd = tid->domain(); \]

- Obtain and modify scheduling parameters
  
  \[ tid->schedule_info(&info); \]

---

**Sessions**

- Sessions reduce authentication overheads of repeated calls

- Prior to invoking methods on a service, must establish session
  
  \[ session = pd->new_session(server); \]

  - Establishes session between target PD and server

  - `server` is a PD ID

    \[ \textbf{Note:} \text{ This is likely to change} \]

    - Iguana informs the server by invoking its notification method
      
      \[ server->session_created(pd); \]

    - Iguana notifies remaining partners if the session is destroyed
      
      \[ pd_or_server->session_destroyed(session); \]

---

**Iguana Capabilities**

- Iguana capabilities are user-level objects

  - `password capabilities`, consisting of OID and password

    | object ID | password |
    |-----------|----------|

  - Length of password is configurable (normally \( \geq 64 \) bits)

- Iguana has a list of all valid capabilities

  - When validating an operation, matches user's capability against list

- Capabilities are never explicitly presented to Iguana, instead

  - Client stores caps in PD's `capability list` (Clist) data structures

  - Client presents object ID to system on method invocation

  - System traverses client's Clists for matching capabilities

- Most applications don't need to know about capabilities

  - Protection system is unintrusive

  - Can emulate wide range of protection models

---

**Protection Domains**

- Protection domain is defined as a set of capabilities

  - Iguana PDs represented by a two-level data structure

    - PD associated with an array of Clists
      
      - Clist is an array of capabilities
      
      - Clist is (part of) a memory section

      \[ \text{subject to memory protection like any memory section} \]

    - PD may or may not contain its Clists

      - PD may or may not be able to modify itself

      - Can freeze access rights of a domain

      - Also control over adding and removing Clists
PROTECTION DOMAINS

- Two-level scheme for capability storage provides flexibility
  - can give users full control over their access rights
    -> purely discretionary access control, no system policies
  - can force all Clists to be kept by a single server (or set)
    -> allows server to implement almost arbitrary security policies
    -> essentially a segregated capability scheme
  - hybrid schemes are possible

EXTERNAL SPACES

- External spaces are "raw" L4 address spaces
  -> not part of Iguana SAS

- Provided to deal with restrictions of Iguana model
  -> 32-bit address space may not be large enough to share between all protection domains
  -> legacy support (e.g. strict fork() semantics) may require separate address spaces

- External spaces come at a cost
  - unable to make full use of fast address-space switching on ARM
  - not well integrated with Iguana world
    -> no fine-grained access control provided by Iguana capabilities
    -> not allowed to communicate with any PD other than creator
    -> not even with Iguana — cannot invoke methods
    -> this will be enforced as soon as L4 redirectors are implemented

PROTECTION DOMAINS

- Presently, access control is disabled
  -> implementation incomplete
    -> will be completed in the near future (code is mostly there)

- Present L4 mechanisms are deficient
  - L4 provides redirectors for information flow control
    -> presently not implemented
    -> to be done later this year
  - Redirectors are theoretically sufficient, practically inefficient
    -> would require all inter-PD communication to go via Iguana server
    -> doubling of number of IPC operations
  - L4 API revision in progress for resolving these issues
    -> Iguana ready to take advantage of this
    -> until then will have a security/performance tradeoff

EXTERNAL SPACES — OPERATIONS

- Creation requires explicit specification of KIP and UTCB address
  es = pd->new_es (kip, utcb_area);

- Thread creation also requires arguments similar to L4
  l4tid = es->new_thread(pager, scheduler, starter, utcb);
**HARDWARE ACCESS**

- Device drivers need to access raw hardware features
- Iguana provides a (static) hardware object for this
  - physical memory access:
    ```c
    hardware->back_mem(addr, p_addr, caching);
    ```
    - maps the memory section (addr) to the specified physical address with specified caching attributes
  - interrupt association:
    ```c
    hardware->register_interrupt(tid, irq);
    ```
    - registers the specified thread as the handler of the specified interrupt

**RESOURCE TOKENS**

- Iguana’s resource management mechanism
- Note: presently this only exists conceptually
  - details of the model still need to be worked out
  - however, model is based on our experience with a similar model in Mungi
- Basic idea: all resources have a price that must be paid by the user
- Model provides great flexibility for defining charging details

**OUTLINE**

- Introduction
- Iguana concepts, abstractions and mechanisms
- Iguana API
- Kenge

**KENGE**

- Kenge is a set of support libraries for building operating systems
  - mostly OS independent
    - ... but geared towards L4
  - implemented in C
- Kenge is **not**:  
  - an L4 server (or servers)
  - an OS personality
  - a part of Iguana
    - although Iguana’s implementation uses Kenge
KENGE COMPONENTS

**libc** a C library
- C99 compliant
- mostly OS independent, but can be specialised for particular OS
- I/O, memory allocation, CRT, ...

**libdriver** device driver library
- provides an API against which drivers can be developed
- host OS must provide wrappers implementing the required functionality
- provides a set of drivers (presently SA1100 UART only)
- more on drivers later...

**elf** library for parsing ELF files

**l4e** convenience functions around L4
- parsing bootinfo
- parsing memory descriptors

---

KENGE COMPONENTS

**l4** L4 system call library
- from L4Ka::Pistachio distribution
- more appropriate place for distribution

Generic data structures:
- **bit_fl**: free list based on a bit array
- **range_fl**: free list based on linked list of ranges
- **circular_buffer**:
- **hash**:
  - **ll**: linked list

---

DEVICE DRIVER FRAMEWORK

- Generic library to write device drivers to
- Write once, run everywhere
  - drivers portable across processor architectures
    - e.g., IDE disk, NICs
  - drivers portable across operating systems
    - Iguana user-level
    - Linux user-level and in-kernel

---

DEVICE DRIVER FRAMEWORK

- Handles driver’s interaction with environment transparently
  - interrupt model: interrupt invokes function in driver
- Handles allocation of device-specific memory
  - provision of PCI-consistent memory
  - pinning
  - virtual $\rightarrow$ physical address translation
DEVICE DRIVER FRAMEWORK

- Interaction of driver with environment
  - driver to export a certain API
  - dependent on device class:
    ➔ stream device
    ➔ network device
    ➔ block device
    ➔ frame buffer