seL4 API and Libraries

High level intro

Ihor Kuz, Stephen Sherratt, Kofi Doku Atuah

August 2016

www.csiro.au

seL4 API

seL4 API: Overview

- Key Concepts
  - Kernel Object
    - in-kernel datastruct, only directly accessible by kernel
  - Capability
    - reference to a kernel object
    - allows holder to invoke functions on the objects
      - i.e. ask kernel to do something with the object
      - holder: thread invoking the cap
- Low-level interface for key activities
  - create kernel objects (retype untyped)
  - create and manage caps in a CSpace
  - create and manage VSpace
  - create and manage threads
  - communicate between threads (IPC)

seL4 Capability

- Kernel-maintained
  - user-level cannot directly access or manipulate a capability
  - capability is stored in CSpace
  - pass CSpace address of cap in system calls
- Datatypes
  - `seL4_CPtr`
    - index into current thread's CSpace root (CNode)
    - this can be tricky….

Untyped and retyping

- Untyped Memory Object
  - region of (RAM) memory
  - must be retyped to another object to use it
    - results in a nested tree from a root untyped to other objects:
- Retyping
  - kernel uses part of untyped's memory region to store a new kernel object
    - can only create an object if you have a cap to a big enough untyped object
  - retype provides user with cap to the new object
- seL4_Untyped_Retype
  - `seL4_Untyped_Retype(seL4_Untyped` **`service`**`, int` **`type`**`, int` **`size_bits`**`, seL4_CNode root, int node_index, int node_depth, int node_offset, int` **`num_objects`**`)`

TCB (Thread Control Block)

- TCB Object:
  - kernel's representation of a thread.
  - contains:
    - Caps: CSpace, VSpace, IPC Buffer Frame
    - Other: IP (instruction pointer), SP (stack pointer), IPC Buffer, Priority

CSpace slots and addresses

- CNode Object
  - consists of slots in which capabilities are stored
  - can also store CNode caps in slots, creates hierarchical CSpace structure
  - CNode size (radix): 2^{radix} slots
- CSpace structure
  - Hierarchy of CNodes: 1-level, 2-level, 3-level, …
- CSpace Slot (CSlot) address
  - CPtr: 32-bit Word. resolved based on CSpace structure (see examples)
  - CPtr resolution relative to a **root cnode**
  - **Depth:** how many bits of CPtr to resolve
- Use of CSlot addresses in syscalls:
  - Just CPtr (implicit root = TCB's CNode cap & implicit 32-bit depth)
  - Explicit root (is a 32-bit depth CPtr with implicit root) & CPtr (called index) & explicit depth

CSpace example and addressing

CNode 0: 2^{16}	
0	
1	A
2	U: cnode 1
3	V: cnode 0
4	
…	…

Cnode 1: 2^{16}	
0	
1	W: cnode 2
2	B
3	X: cnode 0
4	
…	…

Cnode 2: 2^{16}	
0	
1	
2	C
3	
4	Y: cnode 3
…	…

Cnode 3: 2^{16}	
0	
1	
2	D
3	Z: cnode 4
4	
…	…

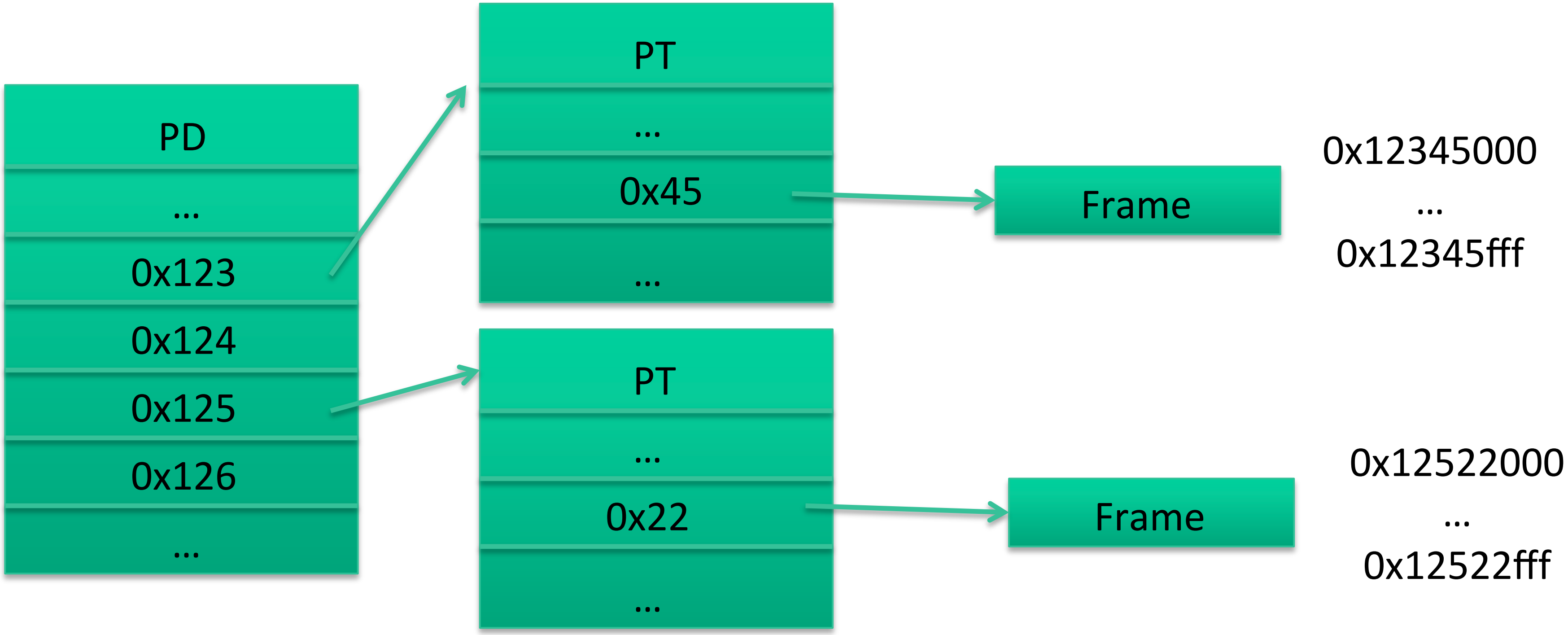
Cnode 4	
0	
1	E
2	
3	
4	
…	…

- B: CPtr: 0x 0002 0002
- W: CPtr: 0x 0002 0001
- X: CPtr: 0x 0002 0003
- A: CPtr: 0x 0001 ????
  - CPtr: 0x 0003 0001
  - Or: Root: X index 0x 0001 depth: 16
- U: CPtr: 0x 0002 ????
  - CPtr: 0x 0003 0002
  - Or: Root: X index: 0x 0002 depth: 16

- C: CPtr: 0x 0002 0001 ~~0002~~
  - Root: W index: 0x 0002 depth: 16
- Y: CPtr: 0x 0002 0001 ~~0004~~
  - Root: W index: 0x 0004 depth 16
- D: Cptr: 0x 0002 0001 ~~0004 0002~~
  - root: W index: 0x 0004 0002 depth: 32
- Z: Cptr: 0x 0002 0001 ~~0004 0003~~
  - root: W index: 0x 0004 0003 depth: 32
- E: root: W index: 0x 0004 0003 ~~0001~~

seL4 API: VSpace

- VSpace:
  - Objects: PageDir (PD), PageTable (PT), Frame
  - represents mapping: virtual address → physical address
    - i.e. abstraction of CPU page table
- Example (on ARM):



seL4 API: Vspace (contd.)

- VSpace-related objects are platform-specific
- ARM
  - PD: 16KiB, 4 byte slots (1MiB of address space/slot)
  - PT: 1KiB, 4 byte slots
  - Frame: 4KiB, 64KiB, 1MiB, 16MiB
- ARM HYP (uses LPAE long descriptors)
  - PD: 16KiB, 8 byte slots (2MiB of address space/slot)
  - PT: 4KiB, 8 byte slots
- x86
  - PD: 4KiB, 4 byte slots
  - PT: 4KiB, 4 byte slots
  - Frame: 4KiB, 4MiB

What’s a seL4 process?

- seL4 doesn’t have a concept of “process”
- So roll your own…
- Traditional process:
  - Each process has its own:
    - CSpace & VSpace & Threads
- Non-traditional process:
  - Different combinations of VSpace and CSpace sharing:
    - Shared CSpace, separate VSpaces
    - Separate CSpace, shared Vspace
    - Partially shared CSpace
    - Partially shared VSpace

Using seL4 API to start a process

- Retype untyped to TCB
  - Find big enough untyped
  - Retype to TCB.
    - Store TCB cap somewhere (local CSpace), remember where!
  - Remember new size of untyped
- Retype untyped to CNode (assuming single level CSpace)
- Retype untyped to PD
- Retype untyped to PT (repeat as necessary)
- Retype untype to Frame (repeat as necessary)
- Build VSpace
  - Map Frames into PTs, Map PTs into PDs
  - Find and load code and data into Vspace (new frame cap, map in own vspace, copy, …)
- Setup new CSpace if necessary
- Configure TCB: CSpace, Vspace, IPC buffer, IP, SP, etc.
- Start thread running through TCB

IPC in seL4 – Endpoints

- Endpoint Object
  - Formerly: synchronous endpoint object
  - enables synchronous (blocking) communication
  - communicating threads must hold caps to same endpoint
- Endpoint Caps
  - ***master cap*** (typically receiver): received when creating the endpoint object
  - ***derived caps*** (senders): minted from master (or other) caps
  - ***badge***: identifies specific sender cap
  - ***reply cap***: temporary cap allows receiver to reply to sender for two-way communication
- Message Registers
  - Data to be sent is stored in message registers (MR)
  - Stored in machine registers or in IPC buffer

IPC – Endpoints: API

- Sending and Receiving
  - `seL4_Send`: send message registers. Blocks if receiver not Recving.
  - `seL4_Recv`: wait for a send on endpoint. Blocks if no send pending.
  - `seL4_Call`: send and recv in one syscall. Also sends a reply cap.
  - `seL4_Reply`: send a message using reply cap.
  - `seL4_ReplyRecv`: send and recv in one syscall. Using reply cap to send.
- Message Registers
  - `seL4_GetMR`: retrieve a given message register from IPC buffer.
  - `seL4_SetMR`: set a given message register in IPC buffer.
  - `seL4_GetCap`: retrieve a cap sent in an IPC.
  - `seL4_SetCap`: prepare a cap to send in an IPC.

seL4 IPC – Notification

- Notification Object
  - Formerly: asynchronous endpoint object
  - allows one thread to send a notification to another
  - notification: asynchronous (non-blocking) message
- Notification Object Caps
  - master cap (receiver), derived caps (senders)
- API
  - `seL4_Signal`:
    - ORs sender badge
  - `seL4_Wait`:
    - Blocks if no new notification since last wait

seL4 System Startup
- Bootinfo and root task

seL4 system startup

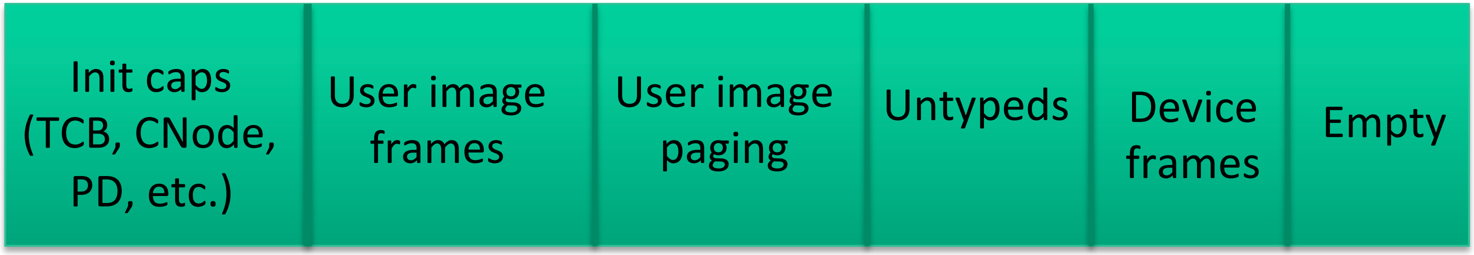
- Image
  - Kernel image
  - User-space image
    - Root task & Cpio file containing elf files
- Boot loader
  - Loads kernel into memory
  - Loads user-space image into memory
  - Starts kernel running
- Kernel startup
  - Kernel creates object (untypeds, frames for device memory)
  - Kernel creates root task objects
  - Loads and runs root task
- Root Task
  - Responsible for setting up the rest of the system

BootInfo: Start-up Information

- Kernel creates:
  - root task CSpace, root task VSpace, Root task TCB
  - frames for device memory
  - untyped caps for RAM memory
- All startup objects are available to root task
  - kernel places caps to these objects in root task CSpace
  - kernel needs to tell root task
    - what caps it has
    - what the objects are
- Bootinfo
  - info about all the initial objects and the caps to them

Root task CSpace and Bootinfo

CSpace



Init caps (TCB, CNode, PD, etc.)

User image frames

User image paging

Untypeds

Device frames

Empty

Bootinfo

Name	Data	Description
empty	Start slot, end slot	Empty CSpace slots
userImageFrames	Start slot, end slot	Slots with root task image's (code, data) frame caps
userImagePaging	Start slot, end slot	Slots with PD and PTs for root task VSpace
untyped	Start slot, end slot	Slots with untyped object caps (sorted by size)
untypedPaddrList	Array of addresses	Physical address for each untyped object
deviceRegions	Array of {paddr, size, start slot, end slot}	Information about all device memory

seL4 Libraries
- Making life a bit easier

Intro to seL4 Libraries

- Goal:
  - Make seL4 programming less “user-unfriendly”
  - Do a bunch of the hard things for you
- Interfaces vs Implementations
  - *Interface*
    - key datastructs
    - function definitions
    - generic code to facilitate use of interface
  - *Implementation*
    - adds implementation-specific parts to datastructs
    - implements interface functions

Key interfaces and libraries

- Key Interfaces
  - **simple**: access to initial caps
  - **vka**: virtual kernel allocator
  - **vspace**: VSpace management
- Key Libraries
  - **libseL4**: seL4 kernel API
  - **allocman** (vka): allocator manager
  - **sel4utils** (vspace, io operations): higher level concepts
- Other Libraries
  - **muslc**: C library
  - **platsupport**, **sel4platsupport**: device access
  - **utils**, **debug**, **benchmark**: other useful functionality

Library: Simple

- Easy way to access initial caps
  - Includes: untypeds, device memory, initial CSpace, initial VSpace
- Abstracts over spec of initial caps
  - root task: uses bootinfo
  - user-level task: can use bootinfo or some other format
- Key concepts
  - location of resources, caps to resources
  - acquiring resource without cap
- Implemented by
  - simple-default, simple-stable, simple-camkes

Allocators

- Allocating objects requires
  - Pool of untypeds to retype into the new objects
  - CSpace slots to put the cap to the new object into
  - Memory for bookkeeping structures
  - Book keeping:
    - What untypeds are available, what sizes are they, where are their caps?
    - How much of the untyped has been used?
    - What CSpace slots are available?
    - Which objects have been created, and which untypeds where used and in which Cspace slots are their caps stored?
- Allocator
  - Manages the untyped pools, the CSpace slots
  - Takes care of bookkeeping
  - Maybe even allows objects to be freed!

Library: VKA

- VKA: Virtual Kernel Allocator
- Interface for allocating kernel objects
  - abstracts away
    - creation of objects through retyping untypeds
    - managing CSpace and book keeping
- Key concepts
  - ***vka***: allocator
  - ***Objects***: represent kernel objects
  - ***CSpace slots***: slot in local cspace where caps can be found
  - ***cspace path***: fully qualified capability address
  - ***utspace***: pool of untyped memory, used to create objects
- Implemented by: allocman

Library: Allocman

- Allocator Manager
  - implements vka interface
  - framework combining independent CSpace and utspace allocators
  - solves difficult recursion problems in allocation: Black Magic!
- Key concepts
  - resources: allocator needs underlying resources (e.g. untypeds)
  - memory pool: needs an initial pool for internal allocations
- Interfaces implemented
  - `vka`
  - `allocman`: to add resources after initialisation

Library: vspace

- Interface for managing VSpaces
  - manage current VSpace
  - manage other VSpaces
    - note: create is not part of vspace API !
  - allocate frames and map them into a VSpace
- Key concepts
  - ***reservation***: portion of VSpace, that will not be given to others
  - ***mapping***: frame mapped into a VSpace at a virtual address
- Implemented by
  - sel4utils

Library: sel4utils

- Utility code to make life easier
  - create and manage threads and processes
  - create vspaces, implement vspace interface
  - load ELF code
- Key concepts
  - process: CSpace + VSpace + TCB
- Interfaces implemented
  - `vspace`
  - `sel4utils`: util functions provided by the library
    - Thread and process management
    - Logging and profiling
    - Architecture agnostic functions

What’s Next?

Practical Experience

- Programming for seL4
  - Building and running
  - Finding code and debugging
- More in-depth
  - sel4 API
  - Libraries
- Hands on exercises
  - Hello world
  - Starting a thread
  - Doing IPC
  - Starting a “process”