These slides are distributed under the Creative Commons Attribution 3.0 License

- You are free:
  - to share—to copy, distribute and transmit the work
  - to remix—to adapt the work
- under the following conditions:
  - Attribution: You must attribute the work (but not in any way that suggests that the author endorses you or your use of the work) as follows:
    "Courtesy of Gernot Heiser, UNSW Sydney"

The complete license text can be found at http://creativecommons.org/licenses/by/3.0/legalcode
### seL4 System Calls

- Notionally, seL4 has 6 syscalls:
  - `Task()` invokes scheduler
  - it only syscall which doesn't require a cap
  - `Send()`, `Recv()` and 3 variants/combinations thereof
  - `Signal()` is actually not a separate syscall but same as `Send()`
  - This is why I earlier said “approximately 3 syscalls”
- All other kernel operations are invoked by “messaging”
  - Invoking `Call()` on an object cap
  - Logically sending a message to the kernel
  - Each object has a set of kernel protocols
  - operations encoded in message tag
  - parameters passed in message words
  - Mostly hidden behind “syscall” wrappers

### seL4 Memory-Management Principles

- Memory (and caps referring to it) is typed
  - Untyped memory:
    - unused, free to `Retype` into something else
  - Frames:
    - can be) mapped to address spaces, no kernel semantics
  - Rest: TCBs, address spaces, CNodes, EPs
    - used for specific kernel data structures
- After startup, kernel `novk` allocates memory!
  - All remaining memory made Untyped, handed to initial address space
  - Space for kernel objects must be explicitly provided to kernel
  - Ensures strong resource isolation
  - Extremely powerful tool for shooting oneself in the foot!
  - We hide much of this behind the `cspace` and ut allocation libraries

### Cspace Operations

```c
seL4_CPtr cspace_alloc_slot(cspace_t *c);
seL4_CPtr cspace_alloc(cspace_t *c);
seL4_CPtr cspace_create_one_level(cspace_t *bootstrap, cspace_t *target, cspace_alloc_t cspace_alloc);
seL4_CPtr cspace_create_two_level(cspace_t *bootstrap, cspace_t *target, cspace_alloc_t cspace_alloc);
seL4_CPtr cspace_free_slot(cspace_t *c, seL4_CPtr slot);
seL4_CPtr cspace_destroy(cspace_t *c);
seL4_CPtr cspace_create(cspace_t *c);
seL4_CPtr cspace_destroy_reply_cap(cspace_t *c, seL4_Word badge);
seL4_CPtr cspace_untyped(ut_cap_t ut, seL4_CPtr cptr);
seL4_CPtr cspace_untyped_retype(cspace_t *cspace, seL4_CPtr ut, seL4_CPtr cptr, seL4_CapRights_t rights);
seL4_CPtr cspace_revoke(cspace_t *dest, seL4_CPtr dest_cptr, cspace_t *src, seL4_CPtr src_cptr, seL4_CapRights_t rights);
seL4_CPtr cspace_mutate(cspace_t *dest, seL4_CPtr dest_cptr, cspace_t *src, seL4_CPtr src_cptr);
seL4_CPtr cspace_move(cspace_t *dest, seL4_CPtr dest_cptr, cspace_t *src, seL4_CPtr src_cptr);
seL4_CPtr cspace_copy(cspace_t *dest, seL4_CPtr dest_cptr, cspace_t *src, seL4_CPtr src_cptr);
seL4_CPtr cspace_mint(cspace_t *dest, seL4_CPtr dest_cptr, cspace_t *src, seL4_CPtr src_cptr);
seL4_CPtr cspace_delete(cspace_t *c);
seL4_CPtr cspace_get(cspace_t *c, seL4_CPtr cptr);
seL4_CPtr cspace_set(cspace_t *c, seL4_CPtr cptr);
```

### seL4 Memory Management Approach

- Resources fully delegated, allows autonomous operation
- Strong isolation, No shared kernel resources

### Capability Derivation

- Copy, Mint, Mutate, Revoke are invoked on CNodes
  - Mint takes a cap for destination
  - CNode cap must provide appropriate rights
  - Copy takes a cap for destination
  - Allows copying of caps between Cspaces
  - Alternative to granting via IPC (if you have privilege to access Cspace!

### seL4 Memory-Management Principles

- Memory (and caps referring to it) is typed
  - Untyped memory:
    - unused, free to `Retype` into something else
  - Frames:
    - `can be) mapped to address spaces, no kernel semantics`
  - Rest: TCBs, address spaces, CNodes, EPs
    - used for specific kernel data structures
- After startup, kernel `novk` allocates memory!
  - All remaining memory made Untyped, handed to initial address space
  - Space for kernel objects must be explicitly provided to kernel
  - Ensures strong resource isolation
  - Extremely powerful tool for shooting oneself in the foot!
  - We hide much of this behind the `cspace` and ut allocation libraries

### Cspace Operations

```c
seL4_CPtr cspace_alloc_slot(cspace_t *c);
seL4_CPtr cspace_alloc(cspace_t *c);
seL4_CPtr cspace_create_one_level(cspace_t *bootstrap, cspace_t *target, cspace_alloc_t cspace_alloc);
seL4_CPtr cspace_create_two_level(cspace_t *bootstrap, cspace_t *target, cspace_alloc_t cspace_alloc);
seL4_CPtr cspace_free_slot(cspace_t *c, seL4_CPtr slot);
seL4_CPtr cspace_destroy(cspace_t *c);
seL4_CPtr cspace_create(cspace_t *c);
seL4_CPtr cspace_destroy_reply_cap(cspace_t *c, seL4_Word badge);
seL4_CPtr cspace_untyped(ut_cap_t ut, seL4_CPtr cptr);
seL4_CPtr cspace_untyped_retype(cspace_t *cspace, seL4_CPtr ut, seL4_CPtr cptr, seL4_CapRights_t rights);
seL4_CPtr cspace_revoke(cspace_t *dest, seL4_CPtr dest_cptr, cspace_t *src, seL4_CPtr src_cptr, seL4_CapRights_t rights);
seL4_CPtr cspace_mutate(cspace_t *dest, seL4_CPtr dest_cptr, cspace_t *src, seL4_CPtr src_cptr);
seL4_CPtr cspace_move(cspace_t *dest, seL4_CPtr dest_cptr, cspace_t *src, seL4_CPtr src_cptr);
seL4_CPtr cspace_copy(cspace_t *dest, seL4_CPtr dest_cptr, cspace_t *src, seL4_CPtr src_cptr);
seL4_CPtr cspace_mint(cspace_t *dest, seL4_CPtr dest_cptr, cspace_t *src, seL4_CPtr src_cptr);
seL4_CPtr cspace_delete(cspace_t *c);
seL4_CPtr cspace_get(cspace_t *c, seL4_CPtr cptr);
seL4_CPtr cspace_set(cspace_t *c, seL4_CPtr cptr);
```
Memory Management Mechanics: Retype

- The UT table handles allocation for you
- Objects are allocated by `Retype()` of Untyped memory
- The kernel will not allow you to overlap objects
- You can modify as required

---

Memory-Management Caveats

- The UT table handles allocation for you
- A simple list-based allocator, you need to understand how it works:
  - Freeing an object of size $n$: you can allocate new objects $<= n$
  - Freeing 2 objects of size $n$ does not mean that you can allocate an object of size $2n$

---

Multiple Frame Mappings: Shared Memory

- Each mapping requires its own frame cap even for the same frame

---

Memory Management Caveats

- Objects are allocated by `Retype()` of Untyped memory
- The kernel will not allow you to overlap objects
- `ut_alloc` and `ut_free()` manage user-level’s view of Untyped allocation:
  - Major pain if kernel and user’s view diverge
  - TIP: Keep objects address and CPtr together.

---

Memory Management Caveats

- Be careful with allocations!
- Don’t try to allocate all of physical memory as Frames, you need more memory for TCBs, endpoints etc.
- We provide a frametable that integrates with `ut_alloc` to manage the 4KiB Untyped size.
- You can modify as required

---

Address Space Operations

- Virtual address, phys_address, address_space and frame_cap
- You need to keep track of (frame_cap, PD_cap, v_adr, p_adr)

---

 sel4 CPtr new_frame_cap = cspace_alloc_slot(&cspace, frame, seL4_AllRights, seL4_Default_VMAttributes);
 sel4 AllRights, seL4_Default_VMAttributes);
Threads

- Threads are represented by TCB objects
- They have a number of attributes (recorded in TCB):
  - VSpace: a virtual address space
    - page global directory (PGD) reference
    - multiple threads can belong to the same VSpace
  - CSpace: capability storage
    - CNode reference (CSpace root) plus a few other bits
    - Fault endpoint
      - Kernel sends message to this EP if the thread throws an exception
  - IPC buffer (backing storage for virtual registers)
  - stack pointer (SP), instruction pointer (IP), user-level registers
  - Scheduling priority and maximum controlled priority (MCP)
  - Time slice length (presently fixed)
- These must be explicitly managed
  - ... we provide an example you can modify

Creating a thread

- Obtain a TCB object
- Set attributes:
  - Configure()
    - associate with VSpace, CSpace, fault EP, prio, define IPC buffer
- Set scheduling parameters
  - priority (maybe MCP)
  - Set SP, IP (and optionally other registers): WriteRegisters()
  - this results in a completely initialised thread
  - will be able to run if resume_target is set in call, else still inactive
- Activated (made schedulable): Resume()
### Exception Handling

- A thread can trigger different kinds of exceptions:
  - invalid syscall
    - may require instruction emulation or result from virtualization
  - capability fault
    - cap lookup failed or operation is invalid on cap
  - page fault
    - attempt to access unmapped memory
    - may have to grow stack, grow heap, load dynamic library, ...
  - architecture-defined exception
    - divide by zero, unaligned access, ...
- Results in kernel sending message to fault endpoint
  - exception protocol defines state info that is sent in message
- Replying to this message restarts the thread
  - endless loop if you don’t remove the cause for the fault first!

### Interrupt Handling

- Interrupt handler has Receive cap on that Notification
- Handler performs appropriate action.
- Handler waits on Notification
- Kernel ACKs IRQ

### Interrupt Management

- seL4 models IRQs as messages sent to a Notification
- Single IRQControl object
- single IRQControl cap provided by kernel to initial VSpace
- only purpose is to create IRQHandler caps
- Per-IRQ-source IRQHandler object
  - interrupt association and dissociation
  - interrupt acknowledgment
  - edge-triggered flag

### Device Drivers

- In seL4 (and all other L4 kernels) drivers are usermode processes
- Drivers do three things:
  - Handle interrupts (already explained)
  - Communicate with rest of OS (IPC + shared memory)
  - Access device registers
- Device register access
  - Devices are memory-mapped on ARM
  - Have to find frame cap from bootinfo structure
  - Map the appropriate page in the driver’s VSpace

```c
device_vaddr = sos_map_device(&cspace, 0xA0000000, BIT(seL4_PageBits));
...
*(void *) device_vaddr = ...
```

### Project Platform: ODROID-C2

- Armlogic S905 SoC
- ODROID-C2 Board
- 2 GB Memory
- Ethernet connector
- Serial connector

- Magic device register access

- ODROID-C2 Board
- Cortex-A53 Core
- Cortex-A53 Core
- Ethernet
- Other...

- ODROID-C2 Board
- Cortex-A53 Core
- Cortex-A53 Core
- Timer
- Serial

- M0: serial over LAN for user-level apps
- M5: Network File System (NFS)
in the Real World (Courtesy Boeing, DARPA)