seL4: Capabilities

廖东海

ctrlz.donghai@gmail.com

What’s a capability?

- *A capability is a unique, unforgeable token that gives the possessor permission to access an entity or object in system.*
- *One way to think of a capability is as a pointer with access rights.*
- Capability主要分为三种：
  - 内核对象：TCB、CNode、VSpace、Endpoint等。
  - 抽象资源（IRQControl）。
  - Untyped
- 在seL4内核初始化时，指向内核控制资源的所有的capability全部被授权给特殊的任务 ***root task***。
- 用户代码向修改某个资源的状态，必须使用kernel API在指定的capability上请求操作

What's the Root Task

- 操作系统在boot时初始化的一个用户态任务而不是内核任务，通过Boot Info将所有的权限传递给Root Task.
- 因为拥有整个系统的所有权限，所以是可信组件的一部分。
- 类似于Unix中的init任务。
- seL4中没有真正意义上的内核任务，内核被抽象为一个权限的监督层和进程交互的控制层。

```
typedef struct seL4_BootInfo {
    seL4_Word          extraLen;        /* length of any additional bootinfo information */
    seL4_NodeId        nodeID;          /* ID [0..numNodes-1] of the seL4 node (0 if uniprocessor) */
    seL4_Word          numNodes;        /* number of seL4 nodes (1 if uniprocessor) */
    seL4_Word          numIOPTLevels;   /* number of IOMMU PT levels (0 if no IOMMU support) */
    seL4_IPCBuffer    *ipcBuffer;       /* pointer to initial thread's IPC buffer */
    seL4_SlotRegion    empty;           /* empty slots (null caps) */
    seL4_SlotRegion    sharedFrames;    /* shared-frame caps (shared between seL4 nodes) */
    seL4_SlotRegion    userImageFrames; /* userland-image frame caps */
    seL4_SlotRegion    userImagePaging; /* userland-image paging structure caps */
    seL4_SlotRegion    ioSpaceCaps;     /* IOSpace caps for ARM SMMU */
    seL4_SlotRegion    extraBIPages;    /* caps for any pages used to back the additional bootinfo information */
    seL4_Word          initThreadCNodeSizeBits; /* initial thread's root CNode size (2^n slots) */
    seL4_Domain        initThreadDomain; /* Initial thread's domain ID */
#ifdef CONFIG_KERNEL_MCS
    seL4_SlotRegion    schedcontrol; /* Caps to sched_control for each node */
#endif
    seL4_SlotRegion    untyped;         /* untyped-object caps (untyped caps) */
    seL4_UntypedDesc   untypedList[CONFIG_MAX_NUM_BOOTINFO_UNTYPED_CAPS]; /* information about each untyped */
    /* the untypedList should be the last entry in this struct, in order
     * to make this struct easier to represent in other languages */
} seL4_BootInfo;
```

Example

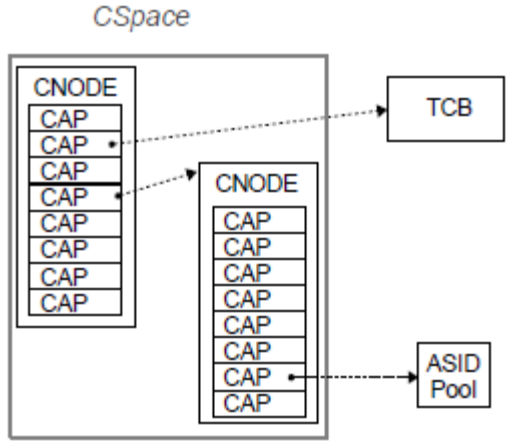
- Root Task拥有的TCB（任务控制块）内核对象在内核Boot的时候已经被创建好了。
- 无法通过虚拟地址直接访问，只能通过预定义的capability进行访问和控制。
- 现在考虑这样一个场景，seL4预定义默认分配给Root Task的函数栈大小为4K字节，这是由这样一个需求，需要在栈上分配一个4M字节的内存变量。
- 通过TCB capability来更改当前任务的栈顶指针。

```
seL4_UserContext registers;
seL4_Word num_registers = sizeof(seL4_UserContext)/sizeof(seL4_Word);

/* Read the registers of the TCB that the capability in seL4_CapInitThreadTCB grants access to. */
seL4_Error error = seL4_TCB_ReadRegisters(seL4_CapInitThreadTCB, 0, 0, num_registers, &registers);
assert(error == seL4_NoError);

/* set new register values */
registers.sp = new_sp; // the new stack pointer, derived by prior code.

/* Write new values */
error = seL4_TCB_WriteRegisters(seL4_CapInitThreadTCB, 0, 0, num_registers, &registers);
assert(error == seL4_NoError);
```

CSpace, CNode & CSlot

- A CSpace (capability-space) is the full range of capabilities accessible to a **thread**, which may be formed of one or more CNodes.
- A CNode (capability-node) is an object full of capabilities: you can think of a CNode as an array of capabilities.
- The positions in the array we call CSlots.
- 这里只讨论一个CSpace中只有一个CNode的情况。
- Root Task的CSpace被内核在boot阶段设置，主要情况如下：

```
/* caps with fixed slot positions in the root CNode */
enum {
    seL4_CapNull                =  0, /* null cap */
    seL4_CapInitThreadTCB       =  1, /* initial thread's TCB cap */
    seL4_CapInitThreadCNode     =  2, /* initial thread's root CNode cap */
    seL4_CapInitThreadVSpace    =  3, /* initial thread's VSpace cap */
    seL4_CapIRQControl          =  4, /* global IRQ controller cap */
    seL4_CapASIDControl         =  5, /* global ASID controller cap */
    seL4_CapInitThreadASIDPool  =  6, /* initial thread's ASID pool cap */
    seL4_CapIOPortControl       =  7, /* global IO port control cap (null cap if not supported) */
    seL4_CapIOSpace             =  8, /* global IO space cap (null cap if no IOMMU support) */
    seL4_CapBootInfoFrame       =  9, /* bootinfo frame cap */
    seL4_CapInitThreadIPCBuffer = 10, /* initial thread's IPC buffer frame cap */
    seL4_CapDomain              = 11, /* global domain controller cap */
    seL4_CapSMMUSIDControl      = 12,  /*global SMMU SID controller cap, null cap if not supported*/
    seL4_CapSMMUCBControl       = 13,  /*global SMMU CB controller cap, null cap if not supported*/
#ifdef CONFIG_KERNEL_MCS
    seL4_CapInitThreadSC        = 14, /* initial thread's scheduling context cap */
    seL4_NumInitialCaps         = 15
#else
    seL4_NumInitialCaps         = 14
#endif /* !CONFIG_KERNEL_MCS */
};
```



CSpace Addressing

- Invocation
  - 每个线程有在 TCB 中装载了一个特殊的 CNode 作为它 CSpace 的 root。这个 root 可以为空（代表这个线程没有被赋予任何 capability ）
  - 在 Invocation 方式中，我们通过隐式地调用线程的 CSpace root 来寻址 CSlot 。例如：我们使用对 seL4_CapInitThreadTCB CSlot 的调用来读取和写入由该特定 CSlot 中的功能表示的 TCB 的寄存器。

```
seL4_TCB_WriteRegisters(seL4_CapInitThreadTCB, 0, 0, num_registers, &registers);
```

- Direct CSpace address
  - 与 Invocation 默认在 CSpace root 中查找不同，你可以指定你要在哪个 CNode 中查找。这种操作主要用于构建和操作 CSpace 的形状（可能是另一个线程的 CSpace ）
  - Direct Addressing 一般需要以下几个参数：
    - _server/root 需要操作的 capability 所在的 CNode。
    - index 需要操作的 Slot 在 CNode 中的序号。
    - depth 在定位到 Slot 之前遍历 CNode 的距离。
  - 下面的例子中直接定位了 root task 的 TCB ，然后在 CSpace root 的第 0 个 Slot 中复制它

CSpace Addressing

- Invocation
  - 每个线程有在 TCB 中装载了一个特殊的 CNode 作为它 CSpace 的 root。这个 root 可以为空（代表这个线程没有被赋予任何 capability ）
  - 在 Invocation 方式中，我们通过隐式地调用线程的 CSpace root 来寻址 CSlot 。例如：我们

```
seL4_Error error = seL4_CNode_Copy(
    seL4_CapInitThreadCNode, 0, seL4_WordBits, // destination root, slot, and depth
    seL4_CapInitThreadCNode, seL4_CapInitThreadTCB, seL4_WordBits, // source root, slot, and depth
    seL4_AllRights);
assert(error == seL4_NoError);
```

- Direct CSpace address
  - 与 Invocation 默认在 CSpace root 中查找不同，你可以指定你要在哪个 CNode 中查找。这种操作主要用于构建和操作 CSpace 的形状（可能是另一个线程的 CSpace ）
  - Direct Addressing 一般需要以下几个参数：
    - _server/root 需要操作的 capability 所在的 CNode。
    - index 需要操作的 Slot 在 CNode 中的序号。
    - depth 在定位到 Slot 之前遍历 CNode 的距离。
  - 下面的例子中直接定位了 root task 的 TCB ，然后在 CSpace root 的第 0 个 Slot 中复制它

Example

- Copy
  - seL4_CNode_Copy
- Delete
  - seL4_CNode_Revoke(seL4_CNode _service, seL4_Word index, seL4_Uint8 depth)
  - seL4_CNode_Delete(seL4_CNode _service, seL4_Word index, seL4_Uint8 depth)
- Move.
  - seL4_CNode_Move

问答