Lecture3 Syscall

ARM简介

ARM：手机终端、苹果M1

AArch64指令集架构，指令长度相同（RISC, 32-bit）

31个64位通用寄存器，1个PC寄存器，4个栈寄存器，3个异常链接寄存器，3个程序状态寄存器

> Q：为什么有四个栈寄存器，而不是三个?
>
> A：L1->L2->L3->L4，上一层出问题由下一层处理

ISA

- RISC

1. 固定长度指令格式
2. 更多的通用寄存器
3. Load/Store结构
4. 简化寻址方式

> CISC内部CPU指令可能会将CISC指令转换为RISC，多了一层功耗

基地址加偏移量模式

基地址加偏移量模式

- **引用M[r_b, Offset]处的数据**
- **基地址寄存器r_b：64位通用寄存器**
- **偏移量Offset可以是下列选项之一**
  1. 立即数 **#imm**
  2. 64位通用寄存器 **r_i**
  3. 修改过的寄存器 **r_m, op**，在这里**op**可以是
     - 移位运算: **lsl #3**
     - 位扩展: **sxtw**

```
ldr w8, [x0, #8]
ldr w9, [x1, x0]
str w9, [x0, x0, lsl #2]
str w9, [x0, w0, sxtw]
```

ARM寻址模式小结

- **寻址模式是表示内存地址的表达式**
  - 基地址模式（索引寻址）
    - [r_b]
  - 基地址加偏移量模式（偏移量寻址）
    - [r_b, offset]
  - 前索引寻址（寻址操作前更新基地址）
    - [r_b, offset]!　　r_b += Offset; **寻址**M[r_b]
  - 后索引寻址（寻址操作后更新基地址）
    - [r_b], offset　　**寻址**M[r_b]; r_b += Offset

函数调用

函数调用指令（caller调用callee）

- **指令**
  - **bl** ***label*** （直接调用，调用函数）
  - **blr** ***Rn*** （间接调用，调用函数指针）

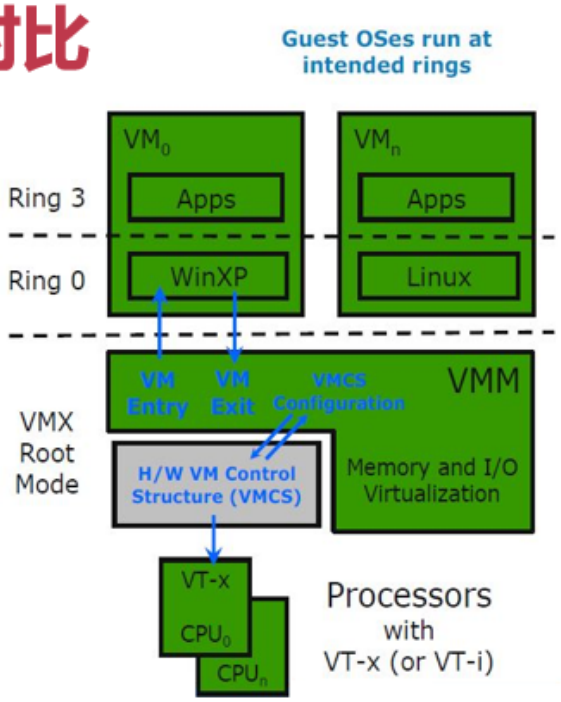
- **功能**
  - 将**返回地址**存储在**链接寄存器LR (x30寄存器的别名)**
    - LR: Link Register
  - 跳转到被调用者的**入口地址**

特权级

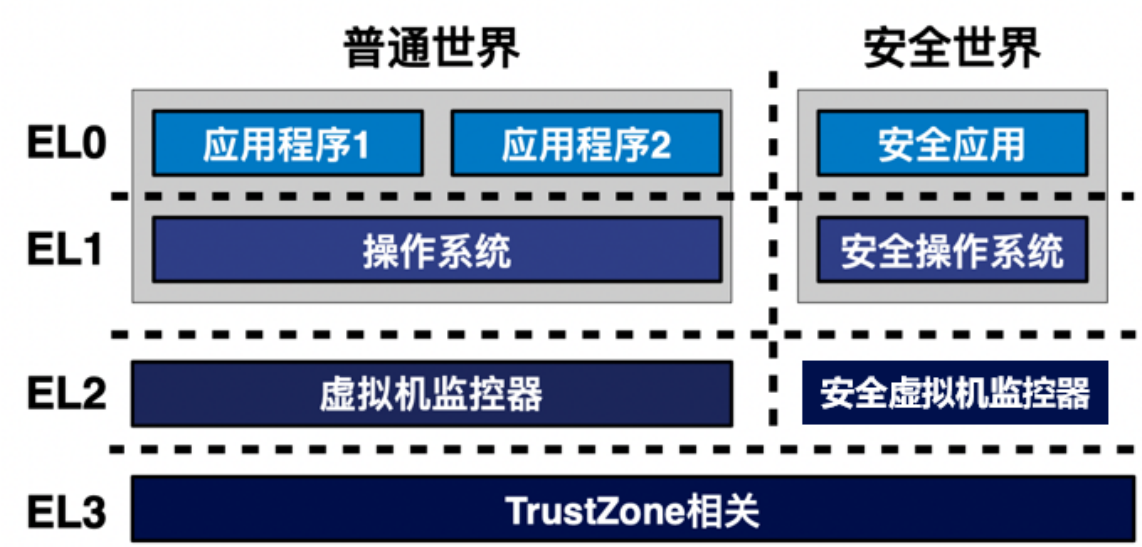
> x86和ARM的区别

特权级：与X86-64对比

- **Non-root :**
  - Ring 3： Guest app
  - Ring 0： Guest OS
- **Root:**
  - Ring 3： App
  - Ring 0： Hypervisor



特权级/ARM (Exception Level)



ARM输入/输出

- MMIO：服用ldr和str指令

  映射到物理内存的特殊地址段

MMIO与PIO

- **MMIO (Memory-mapped IO)**
  - 将设备映射到连续的物理内存中，使用相同的指令
  - 如，Raspi3映射到0x3F200000
  - 行为与内存不完全一样，读写会有副作用（回忆volatile）
- **PIO (Port IO)**
  - IO设备具有独立的地址空间
  - 使用特殊的指令（如x86中的in/out指令）

中断与异常处理

CPU执行逻辑

- 以PC为地址从内存中读取一条指令并执行，PC+=4， goto 1
- 执行过程中可能发生生两种情况

1. 指令执行出现错误，除0/缺页（同步异常）
2. 被外部设备I/O打断（异步异常）

- 以上两种情况均会导致CPU陷入内核态->在异常向量表中寻找handler处理->回到被打断的地方继续运行

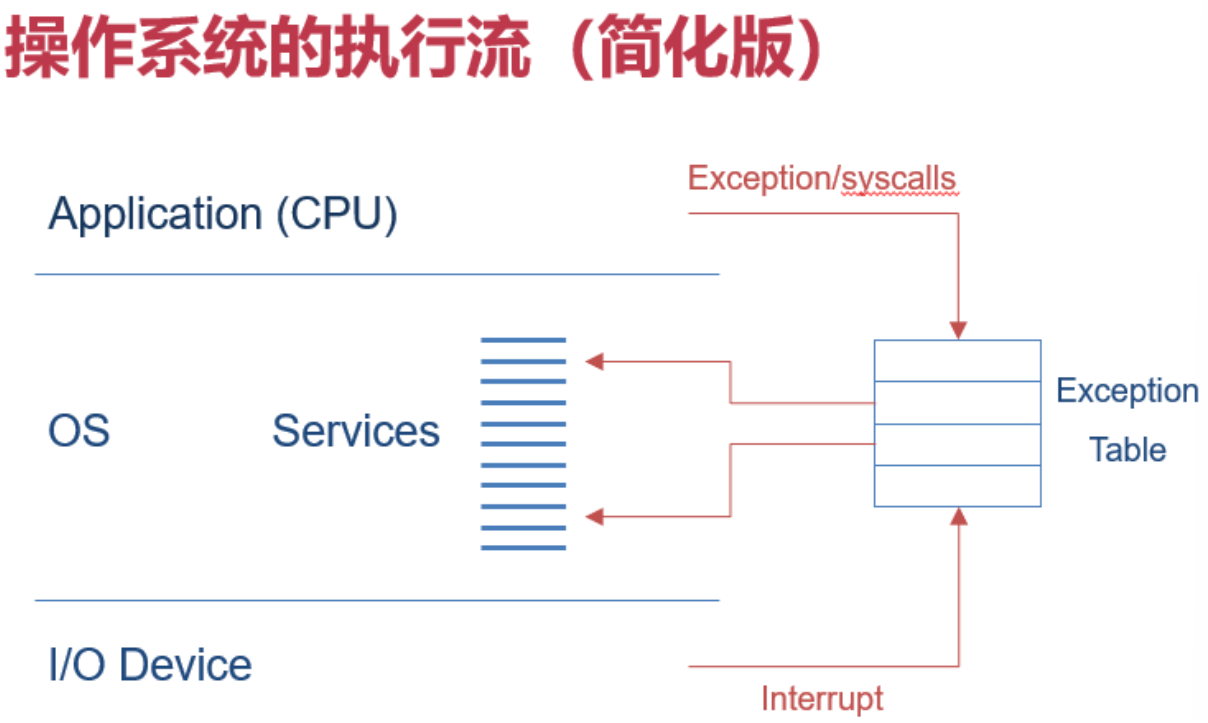
操作系统的执行流

- 设置异常向量表

> CPU上电后立刻执行，系统初始化的主要工作，在启动中断与启动第一个应用之前

- 实现对不同异常的处理函数

> 包括同步异常和异步异常

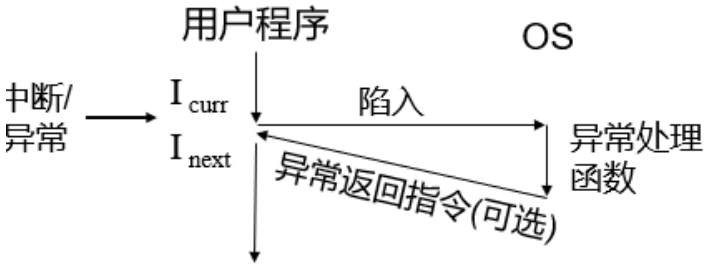


异常向量表

> 操作系统预先在一张表中准备好不同类型的处理函数

异常向量表

- **操作系统预先在一张表中准备好不同类型异常的处理函数**
  - 基地址存储在VBAR_EL1寄存器中
  - 处理器在异常发生时自动跳转到对应位置
  - ARM一共16项，其中4项最为常用



32

异常处理函数

- 异常处理函数运行在内核态

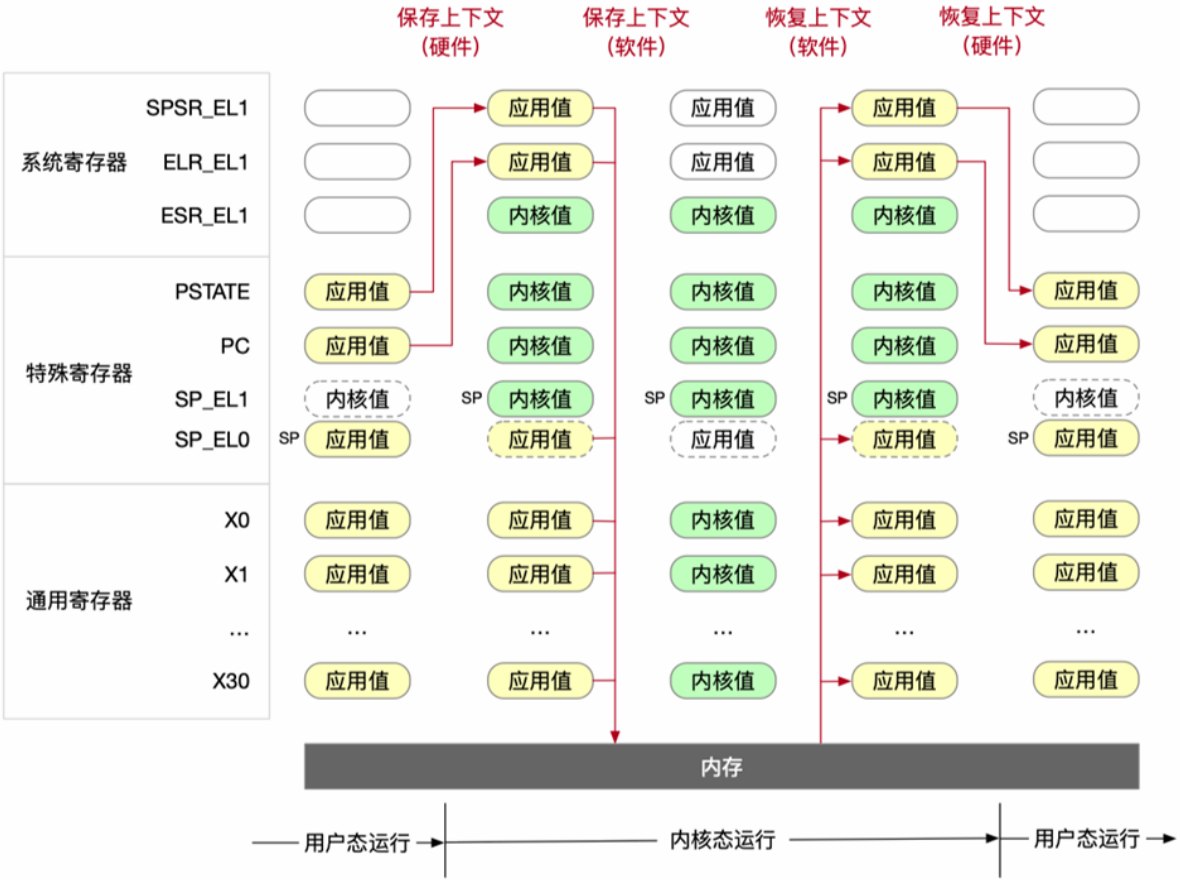
  > 可以不受限制地访问所有资源

- 处理器将异常类型存储在指定寄存器中（**ESR_EL1**）

  > 表明发生的是哪一种异常
  >
  > 异常处理函数根据异常类型执行不同逻辑

- 当异常处理函数完成异常处理后，将通过下属操作之一转移控制权：
  1. 返回异常指令
  2. 返回异常指令下一条指令
  3. 结束当前进程

内核态与用户态的切换

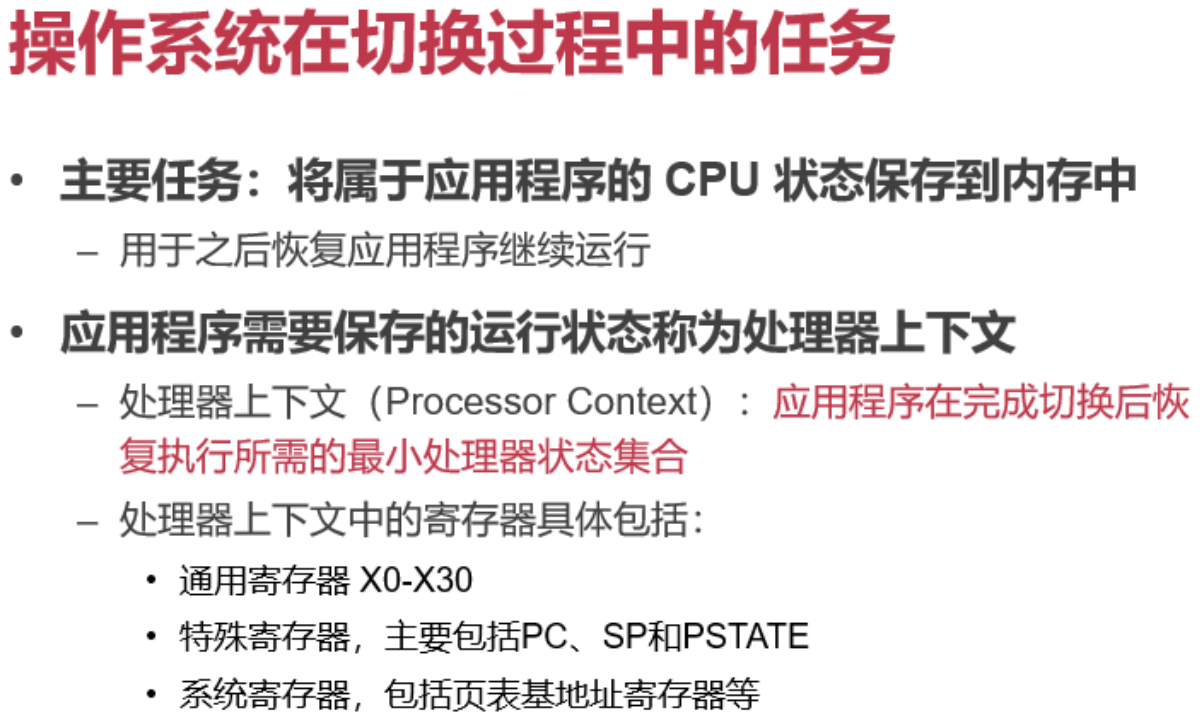


CPU在切换过程中的任务

处理器在切换过程中的任务

1. **将发生异常事件的指令地址保存在ELR_EL1中**
2. **将异常事件的原因保存在ESR_EL1**
   - 例如，是执行svc指令导致的，还是访存缺页导致的
3. **将处理器的当前状态（即PSTATE）保存在SPSR_EL1**
4. **将引发缺页异常的内存地址保存在FAR_EL1**
5. **栈寄存器不再使用SP_EL0（用户态栈寄存器），开始使用SP_EL1**
   - 内核态栈寄存器，需要由操作系统提前设置
6. **修改PSTATE寄存器中的特权级标志位，设置为内核态**
7. **找到异常处理函数的入口地址，并将该地址写入PC，开始运行操作系统**
   - 根据VBAR_EL1寄存器中保存的异常向量表基地址，以及发生异常事件的类型确定

操作系统在切换过程中的任务



系统调用(Syscall)

svc指令：从用户态进入内核态

> 下面是CPU的行为

1. 将处理器状态PSTATE保存到寄存器SPSR_EL1中
2. 将svc后第一条指令地址保存到ELR_EL1
3. 将栈指针寄存器由SP_EL0切换为SP_EL1
4. 将PSTATE中的特权级别切换到内核态(EL1)
5. 根据异常向量表中的配置，执行对应异常向量条目所配置的代码

eret指令：从内核态返回用户态

> 下面是CPU的行为

1. 将SPSR_EL1中存储的处理器状态重设到处理器状态PSTATE中
2. 将处理器正在使用的栈指针寄存器由SP_EL1切换到SP_EL0
3. 将ELR_EL1寄存器中所保存的返回地址重设到PC中，执行应用程序的代码

VDSO(Virtual Dynamic Shared Object)

> 1. 系统调用的时延不可忽略
> 2. 如何降低系统调用的时延

> > 大部分时延：U->K

与应用共享内存页，kernel可读可写，user只读（有点类似于producer-consumer模式）

Flex-Sc(Flexible System Call Scheduling with Exception-Less System Calls)

> 时间大部分用来做状态的切换（保存恢复状态，权限切换）
>
> 是否可能在不做切换的情况下进行syscall

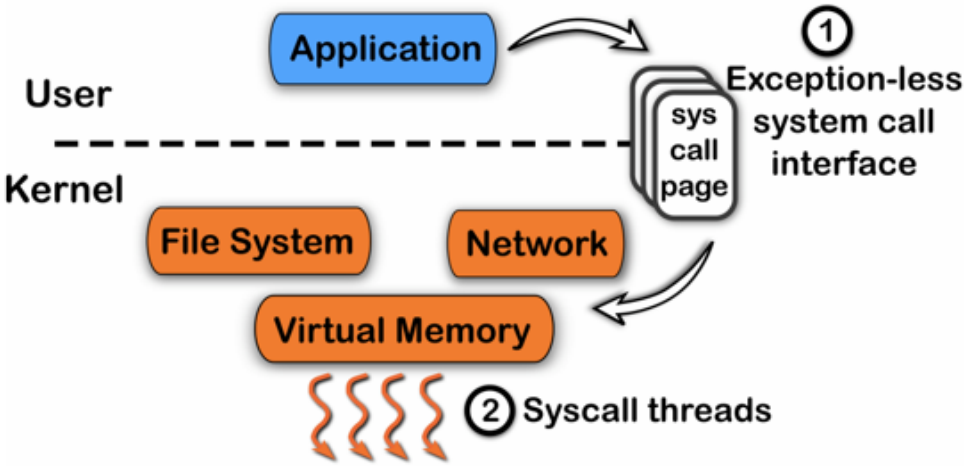
- 引入新的syscall机制，由user & kernel共享

  > 相比于VDSO，最大的改变是user也可以写

- Exception-less syscall

  > 将系统调用的调用和执行解耦，分布到不同的CPU核

System Call的另一种方法



> 把同步调用变成异步调用，对时延敏感型的应用来说不合适