# 18. Linux 系统对中断的处理

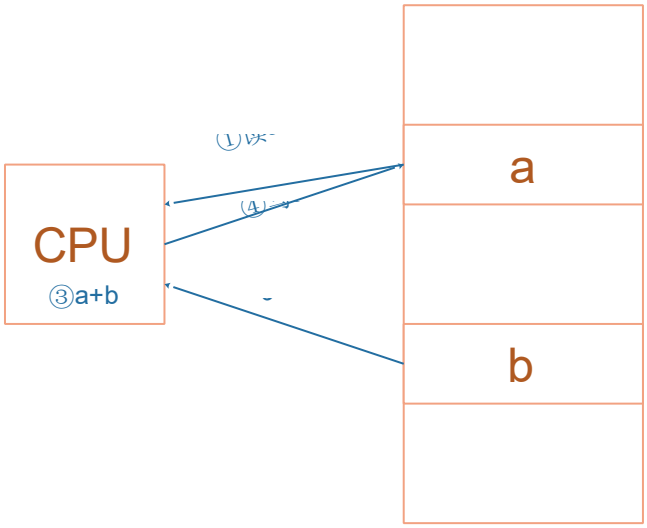
## 18.1 进程、线程、中断的核心：栈

中断中断，中断谁？
中断当前正在运行的进程、线程。
进程、线程是什么？内核如何切换进程、线程、中断？
要理解这些概念，必须理解栈的作用。

### 18.1.1 ARM 处理器程序运行的过程

ARM 芯片属于精简指令集计算机(RISC：Reduced Instruction Set Computing)，它所用的指令比较简单，有如下特点：
① 对内存只有读、写指令
② 对于数据的运算是在 CPU 内部实现
③ 使用 RISC 指令的 CPU 复杂度小一点，易于设计

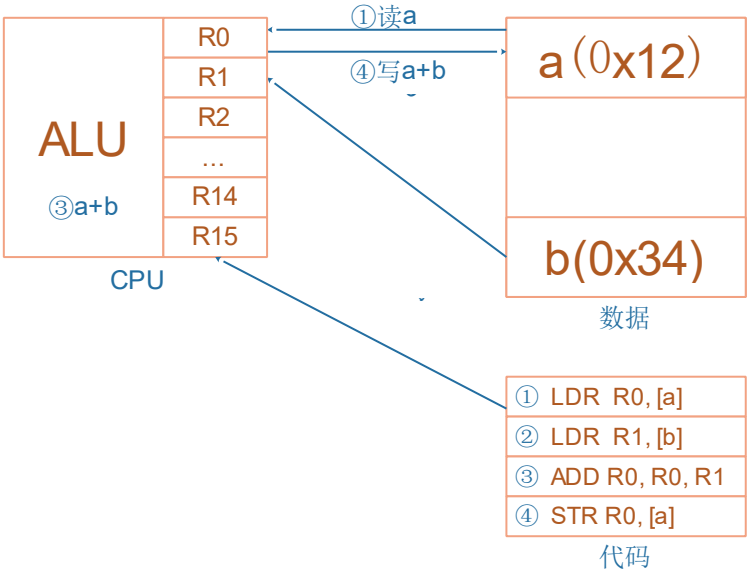
比如对于 a=a+b 这样的算式，需要经过下面 4 个步骤才可以实现：




细看这几个步骤，有些疑问：
① 读 a，那么 a 的值读出来后保存在 CPU 里面哪里？
② 读 b，那么 b 的值读出来后保存在 CPU 里面哪里？
③ a+b 的结果又保存在哪里？

我们需要深入 ARM 处理器的内部。简单概括如下，我们先忽略各种 CPU 模式(系统模式、用户模式等等)。
注意：如果想入理解 ARM 处理器架构，应该从裸机开始学习。我们即将写好近 30 个裸机程序的文档，估计还 3 月底发布。

**注意**：为了加快学习速度，建议先不看裸机。




CPU 运行时，先去取得指令，再执行指令：

① 把内存 a 的值读入 CPU 寄存器 R0
② 把内存 b 的值读入 CPU 寄存器 R1
③ 把 R0、R1 累加，存入 R0
④ 把 R0 的值写入内存 a

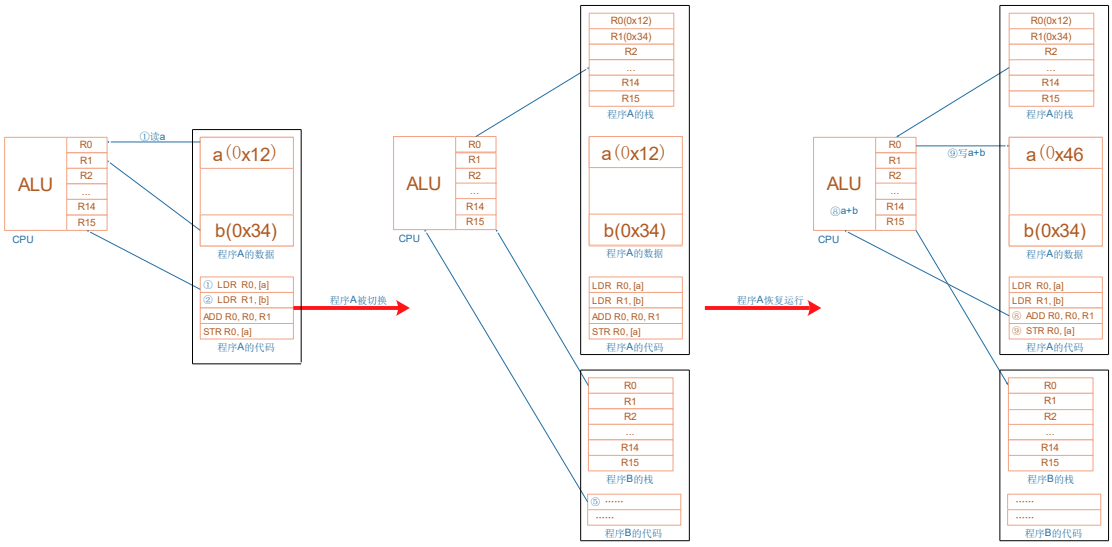
### 18.1.2 程序被中断时，怎么保存现场

从上图可知，CPU 内部的寄存器很重要，如果要暂停一个程序，中断一个程序，就需要把这些寄存器的值保存下来：这就称为保存现场。

保存在哪里？内存，这块内存就称之为栈。

程序要继续执行，就先从栈中恢复那些 CPU 内部寄存器的值。

这个场景并不局限于中断，下图可以概括程序 A、B 的切换过程，其他情况是类似的：




a. 函数调用：

在函数 A 里调用函数 B，实际就是中断函数 A 的执行。

那么需要把函数 A 调用 B 之前瞬间的 CPU 寄存器的值，保存到栈里；
再去执行函数 B；
函数 B 返回之后，就从栈中恢复函数 A 对应的 CPU 寄存器值，继续执行。

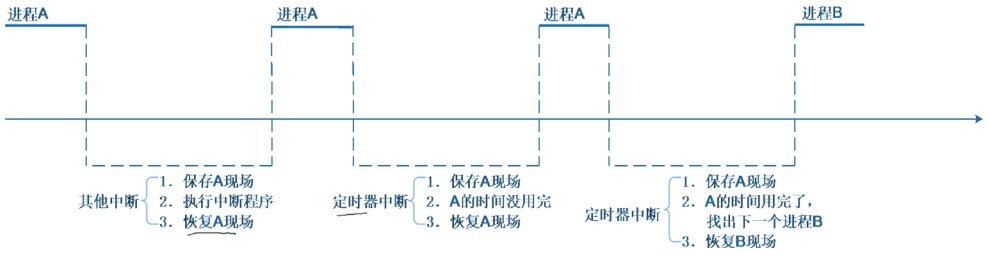
b. 中断处理

进程 A 正在执行，这时候发生了中断。
CPU 强制跳到中断异常向量地址去执行，
这时就需要保存进程 A 被中断瞬间的 CPU 寄存器值，
可以保存在进程 A 的内核态栈，也可以保存在进程 A 的内核结构体中。
中断处理完毕，要继续运行进程 A 之前，恢复这些值。

c. 进程切换

在所谓的多任务操作系统中，我们以为多个程序是同时运行的。
如果我们能感知微秒、纳秒级的事件，可以发现操作系统时让这些程序依次执行一小段时间，进程 A 的时间用完了，就切换到进程 B。
怎么切换？
切换过程是发生在内核态里的，跟中断的处理类似。
进程 A 的被切换瞬间的 CPU 寄存器值保存在某个地方；
恢复进程 B 之前保存的 CPU 寄存器值，这样就可以运行进程 B 了。

所以，在中断处理的过程中，伴存着进程的保存现场、恢复现场。
进程的调度也是使用栈来保存、恢复现场：




### 18.1.3 进程、线程的概念

假设我们写一个音乐播放器，在播放音乐的同时会根据按键选择下一首歌。把事情简化为 2 件事：发送音频数据、读取按键。那可以这样写程序：

```
int main(int argc, char **argv)
{
    int key;

    while (1)
    {
        key = read_key();

        if (key != -1)
        {
            switch (key)
            {
```

```
                    case NEXT:
                        select_next_music(); // 在 GUI 选中下一首歌
                        break;
                }
            }
            else
            {
                send_music();
            }
        }

        return 0;
}
```

这个程序只有一条主线，读按键、播放音乐都是顺序执行。

无论按键是否被按下，read_key 函数必须马上返回，否则会使得后续的 send_music 受到阻滞导致音乐播放不流畅。

读取按键、播放音乐能否分为两个程序进行？可以，但是开销太大：读按键的程序，要把按键通知播放音乐的程序，进程间通信的效率没那么高。

这时可以用多线程之编程，读取按键是一个线程，播放音乐是另一个线程，它们之间可以通过全局变量传递数据，示意代码如下：

```
int g_key;
void key_thread_fn()
{
    while (1)
    {
        g_key = read_key();
        if (g_key != -1)
        {
            switch (g_key)
            {
                case NEXT:
                    select_next_music(); // 在 GUI 选中下一首歌
                    break;
            }
        }
    }
}

void music_fn()
{
    while (1)
    {
        if (g_key == STOP)
            stop_music();
```

```
            else
            {
                send_music();
            }
        }

}

int main(int argc, char **argv)
{
        int key;

        create_thread(key_thread_fn);
        create_thread(music_fn);
        while (1)
        {
            sleep(10);
        }
        return 0;
}
```

这样，按键的读取及 GUI 显示、音乐的播放，可以分开来，不必混杂在一起。

按键线程可以使用阻塞方式读取按键，无按键时是休眠的，这可以节省 CPU 资源。

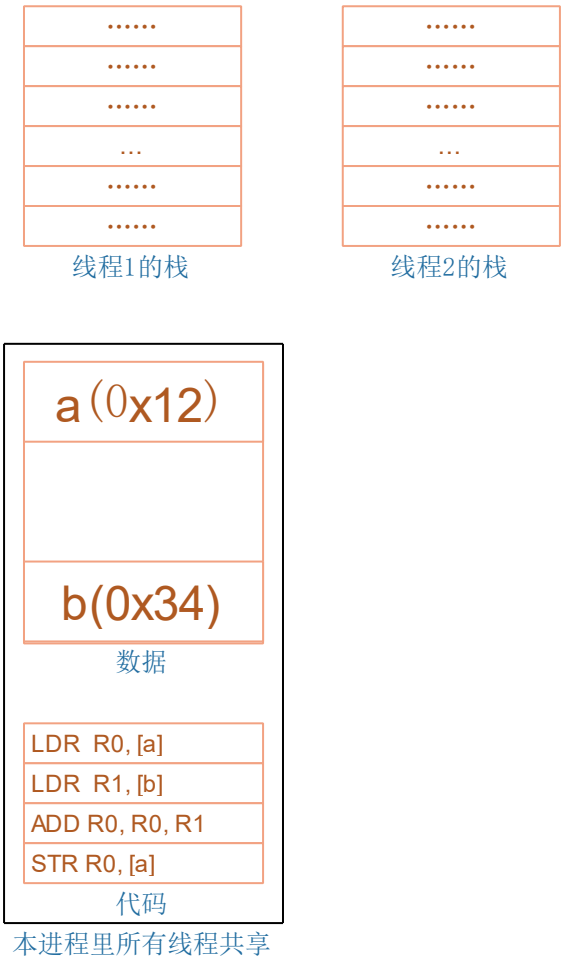
音乐线程专注于音乐的播放和控制，不用理会按键的具体读取工作。

并且这 2 个线程通过全局变量 g_key 传递数据，高效而简单。

在 Linux 中：资源分配的单位是进程，调度的单位是线程。

也就是说，在一个进程里，可能有多个线程，这些线程共用打开的文件句柄、全局变量等等。

而这些线程，之间是互相独立的，“同时运行”，也就是说：每一个线程，都有自己的栈。如下图示：




## 18.2 Linux 系统对中断处理的演进

## 18.3 重要数据结构

## 18.4 编程

### 18.2.1 Linux 的中断不能嵌套

参考文档:

a. 内核 Documentation\devicetree\bindings\Pinctrl\ 目录下:

```
Pinctrl-bindings.txt
fsl,imx-Pinctrl.txt、fsl,imx6ul-Pinctrl.txt
rockchip,Pinctrl.txt
ti,omap-Pinctrl.txt
```

b. 内核 Documentation\gpio 目录下:

```
Pinctrl-bindings.txt
fsl,imx-Pinctrl.txt、fsl,imx6ul-Pinctrl.txt
rockchip,Pinctrl.txt
```

c. 内核 Documentation\devicetree\bindings\gpio 目录下:

```
gpio.txt
```

2.6.30 引入 threaded irq

devm_request_any_context_irq
request_any_context_irq
request_threaded_irq
request_irq

参考资料

中断处理不能嵌套:
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=e58aa3d2d0cc

genirq: add threaded interrupt handler support
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=3aa551c9b4c40018f0e261a178e3d25478dc04a9

Linux RT(2) – 硬实时 Linux(RT-Preempt Patch)的中断线程化
https://www.veryarm.com/110619.html

Linux 中断管理 (1)Linux 中断管理机制
https://www.cnblogs.com/arnoldlu/p/8659981.html