# 系统整体架构

Unix V6++的内核由进程管理、内存管理、设备管理、文件系统、中断处理、系统调用以及硬件管理这几个模块构建而成。下面的系统整体架构图给出了各个模块在内核中的地位以及相互间关系(图4.1)。

硬 件

硬件抽象层

设备管理模块

进程管理模块

内存管理模块

文件系统模块

高速缓存

系统调用

系统调用

用户程序

**用户态**

**操作系统内核**

**硬件**

**Kernel类**

**Machine类**

#### 图 4.1 系统整体架构

## Machine类和Kernel类

在Unix V6++的设计中，有两个重要的Singleton类：Machine和Kernel，这两个类分别抽象了机器实例与操作系统实例。

设计这两个类是出于这样的考虑：操作系统中对底层硬件的操作与内核各个模块本身的数据结构、处理逻辑这两者应当加以区分。在原来的 Unix V6中，存在着大量的全局变量，这些全局变量被内核各个模块引用，例如proc[]数组被进程管理模块引用，bfreelist链表被高速缓存模块引用，中断向量表被PDP-11中断机制引用。同样的做法假如应用于Unix V6++中的话，就会出现GDT、IDT实例与proc[]数组同时出现在全局域中。显然，全局变量并不能表达它们之间的区别，并且容易导致各个模块数据结构层次不清晰。

于是在Unix V6++的设计中，我们引入了Machine类封装对机器硬件以及体系结构相关的数据与操作；引入Kernel类封装内核各个模块的数据与操作，换言之，Machine类进行对机器硬件的抽象，而Kernel类对内核整体架构进行抽象。

从上面的图4.1系统整体架构中可以直观地看出Machine类和Kernel类的界定范围：Machine类负责系统中硬件抽象层部分，系统中硬件控制芯片，汇编语言抽象类，I386体系结构的抽象属于硬件抽象层，它们与Machine类相关；而Kernel类负责进程管理、内存管理、设备管理、文件系统等内核模块，是对内核各个模块的数据和操作进行封装，处于系统的核心位置。

### Machine类

Machine类是一个Singleton(单体类)模式，在操作系统运行时，只有一个实例。从逻辑上来讲，一台机器只能有一个Machine实例是合理的。所有与机器硬件相关的内存数据结构都由Machine类进行初始化。当Machine的唯一实例被初始化时，相应的GDT，IDT，页目录表与页表都将被初始化。因为Machine在整个系统运行过程中只有一个实例，因此所有上述数据结构也都只有唯一的实例。

下图给出了Machine类与GDT、IDT、TSS以及页目录和页表的关系(图4.2)。Machine类成员变量中包含对GDT、IDT、页目录等实例的引用，在系统初始化过程中，对IDT、GDT对象的初始化设置都是经由Machine类所持有的对这些实例的引用来完成。



#### 图 4.2 Machine类

Machine类还提供一系列成员函数封装对系统中体系结构相关数据的初始化设置操作，如InitGDT()和LoadGDT()函数用于初始化和加载全局描述符表；InitIDT()和LoadIDT()函数初始化和加载中断描述符表；EnablePageProtection()函数用于开启分页机制。

在Machine类将初始化GDT、IDT和开启分页机制的操作细节使用函数封装之后，系统初始化过程中对这些数据结构的设置就可以通过调用Machine类中一组目的明确的函数接口来实现，如下面的代码所示(代码4.1)。

/\* 获取Machine单体对象的引用 \*/

Machine& machine = Machine::Instance();

/\* 初始化、加载GDT \*/

machine.InitGDT();

machine.LoadGDT();

/\* 初始化、加载IDT \*/

machine.InitIDT();

machine.LoadIDT();

#### 代码4.1 Machine类初始化GDT和IDT

这里对Machine类的作用及其成员函数的功能进行了介绍，但有关Machine类中各成员函数的细节则不在此具体展开，这里侧重于内核整体架构以及Machine类的作用、地位而非其实现细节。

### Kernel类

Kernel类的设计初衷与Machine类完全一致，且与Machine类遥相呼应。Kernel类封装了所有内核所使用到的数据与操作。在Unix V6++设计中，各个内核模块都以类的形式进行独立封装其内部的数据和操作，保持各个模块的独立性，类命名方式以“模块名+Manager后缀”标识，例如进程管理模块ProcessManager，高速缓存模块BufferManager，设备管理模块DeviceManager等。所有这些模块都经由Kernel类进行访问，Kernel类在整个内核生命周期中只有一个副本。

下图给出了Kernel类与内核各个模块管理类之间的关系(图4.3)。Kernel类的成员变量包含对内核中各个模块实例的引用，如成员变量m\_ProcessManager是对进程管理模块实例的引用，m\_DeviceManager是对设备管理模块的引用等。



#### 图4.3 Kernel类

可以看到，Kernel类是这些类的中心，需要经由Kernel类获得对各个内核模块对象的访问。在初始化时，由Kernel类通过自己持有的对各个模块实例的引用进行所有这些内核模块的初始化工作，对整个内核的初始化通过调用Kernel::Initialize()函数完成(代码4.2)。

void Kernel::Initialize()

{

InitMemory();

InitProcess();

InitBuffer();

InitFileSystem();

// add other Init\*() functions here

}

#### 代码4.2 Kernel::Initialize()函数

该函数是Kernel类的成员函数之一，它调用同属于Kernel类中的其它成员函数，如：InitMemory()、InitProcess()和InitBuffer()等进行内存管理、进程管理、高速缓存等各个模块的初始化。

###### GetUser()函数

Kernel类中最为特殊的是GetUser()函数。该函数将返回当前运行进程user结构的指针，在Unix V6中，u为全局变量，为与之保持一致，仍保留u结构的作用，只是考虑到封装，修改此变量为GetUser()成员函数，并在Kernel类中实现。

### 小结

Machine类与Kernel类两个Singleton类的主要考虑到这两部分的职责分工比较清晰，若仅使用一个类，该类既描述底层体系结构数据结构的操作又管理内核各个模块，这样设计的类会显得比较臃肿。而现在，直接使用类名就清晰的表明了各个函数主要的职能范围，使代码思路更为清晰。