第2章 操作系统结构

孙承杰

E-mail： sunchengjie@hit.edu.cn

哈工大计算学部人工智能教研室

2023年秋季学期

第2章 操作系统结构

- **OS结构：由管理的硬件和提供的服务决定的OS接口、架构**
  - 单机、多核vs多机、集群（云）
  - 单用户、多用户vs海量用户（负载均衡）
  - **进程管理**：多进程管理vs海量任务管理
  - **进程间通信**：进程间通信IPCvsRPC,分布式消息框架
  - **CPU调度**：CPU调度vs集群/虚拟机/容器调度
  - **存储管理**：单机/网络文件系统vs分布式文件系统
  - **内存管理**：单机内存管理vs分布式内存管理和内存计算框架
  - **缓存技术**：磁盘与内存vs分布式文件系统与内存计算框架融合

基本内容

讨论操作系统结构，一般包括3个方面：

- **操作系统所提供的服务**
  - **2.1 操作系统服务**
- **操作系统为用户和程序员提供的接口**
  - **2.2 操作系统的用户界面**
  - **2.3 系统调用（重点）**
  - **2.4 系统调用类型**
  - **2.5 系统程序**
- **操作系统结构及其相互关系**
  - **2.6 操作系统结构**
  - **2.7 虚拟化技术**
  - **2.8 虚拟机与分区操作系统**
  - **2.9 Docker容器**

2.1 操作系统服务

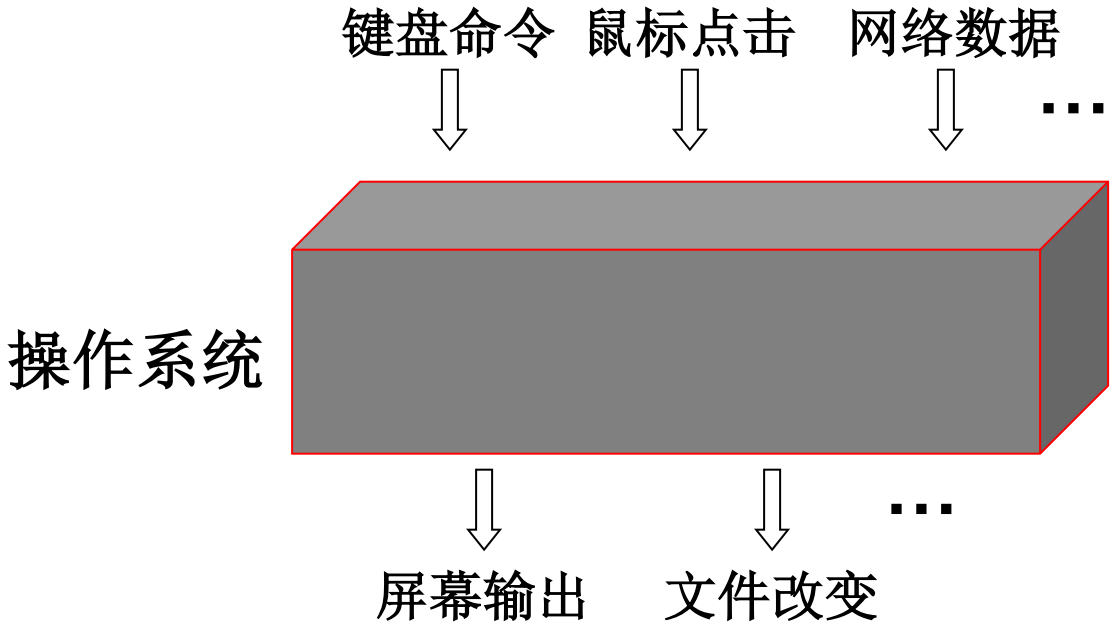
操作系统服务包括：

- 用户界面：CUI、GUI
- 程序执行：装入程序、运行程序
- I/O 操 作：底层的I/O操作（启动、读写、关闭等）
- 文件系统操作：创建/复制/移动/删除/检索文件/目录、列举文件信息等
- 通　　信：IPC（进程间通信）、RPC（远程过程调用）等
- 错误检测：硬件错误、算术溢出、非法地址访问等
- 资源分配：CPU、存储器、外部设备等
- 统　　计：用户用时记帐、资源利用率等
- 保护与安全：登录验证、进程越界访问、非法访问等

操作系统的目标需求决定了其服务类型和集合

2.2 操作系统的用户界面

- 系统接口 —— 用户通过它来使用操作系统
- 对大多数用户来说，操作系统是一个“黑盒子”



- 引导程序告诉了我们这个盒子是怎么放入内存的?
- 在“打开这个盒子”之前要“了解盒子的入口”

2.2 操作系统的用户界面

2.2.1 命令解释程序

- 命令解释程序是OS的特殊程序
- DOS中的COMMAND.COM
- UNIX、Linux中的Shell（B-Shell、C-Shell）
- 命令解释程序的主要作用是获取并执行用户给定的下一条命令
  - 内部命令、外部命令

面向专业人员

2.2 操作系统的用户界面

```
cst:/home/lizhijun# ./output "hello"
ECHO:hello
cst:/home/lizhijun# ./output "OS students"
ECHO:OS students
```

命令行是怎么回事?命令是什么? 命令输入后发生了什么?

```
#include <stdio.h>
int main(int argc, char * argv[ ])
{  printf( “ECHO:%s\n” , argv[1]); }
```

一段程序而已

也是一段程序: shell，即/bin/sh

```
gcc –o output output.c
```

```
./output “hello”
```

```
int main(int argc, char * argv[])
 {  char cmd[20];
    while(1)
       {    scanf( “%s” , cmd);
            if(!fork()) {exec(cmd);}
            else {wait();}
       } //while(1)
 }
```

- 命令得以工作的原因:

  OS提供了printf,fork, exec等函数(接口)

我们可以自己开发的命令行界面！！

2.2 操作系统的用户界面

2.2.2 图形用户界面

- DOS 中的Windows1.0-3.12
- Windows中的Desktop
- Linux中的X-Window
- Mac OS X Aqua

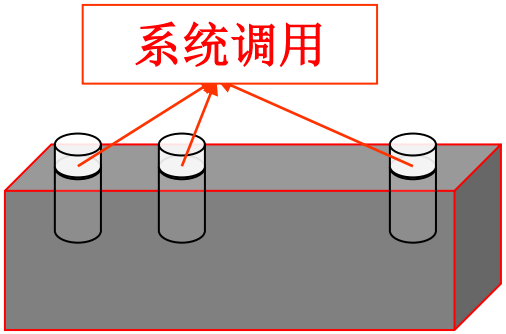
2.2 操作系统的用户界面

2.2.3还有其他形式的用户界面吗?

- 语音识别技术
- Google，百度语音识别系统和相应的API
- 语音识别接口将集成到操作系统中，一些语音命令将同传统的图形用户接口和命令行具有相同的能力。
- 当前的智能手机几乎都有语音交互能力，还有一些具备手势交互

2.3 系统调用

- 总结前述最重要的概念



  - 命令行：命令解释程序+shell+字符显示
  - GUI: 消息处理程序+消息框架+图形显示
  - 应用程序：将上述部分组成一个整体…

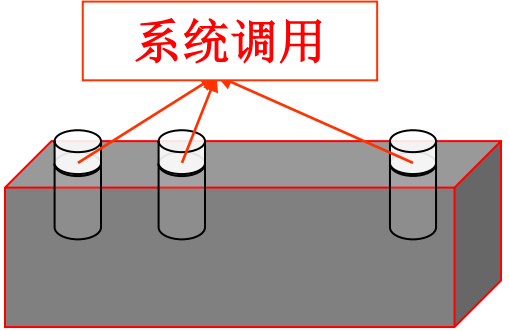
- 用户使用计算机总结：用操作系统提供的接口编写程序；这些程序解决具体的问题

  - 因此：用户通过OS接口使用计算机；OS接口影响计算机的使用方式

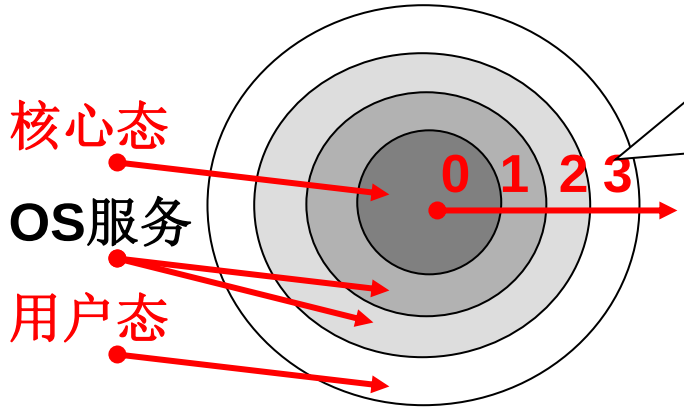
这么重要，得起个名字： 接口表现为函数调用，又由OS提供，叫系统调用

  - 系统调用是学习操作系统的首要任务…

怎么实现系统调用?
（参考“Linux内核完全注释v3.0”第4.5.1.3节）

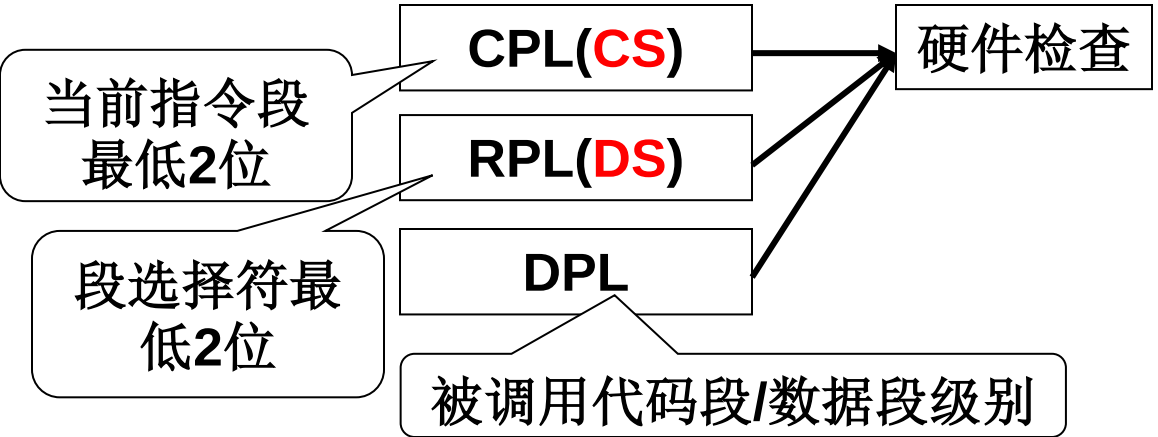


- 提供系统调用能力，但将内核程序和用户程序隔离比较安全!
  - 区分内核态和用户态：一种处理器“硬件设计”



当前程序执行在什么态(哪层环)?
由于CS:IP是当前指令，所以用CS的最低两位来表示:
0是内核态，3是用户态

处理器保护环



- 内核态可以访问任何数据，用户态不能访问内核数据
- 对于指令跳转也一样实现了隔离…

2.3 系统调用

硬件提供了“主动进入内核的方法”

对于Intel x86，那就是中断指令int（trap陷阱指令）

- DOS为INT 21H，Windows2K为 2EH，Linux为0x80H）
- int指令将使CS中的CPL改成0，“进入内核”
- 这是用户程序发起的调用内核代码的唯一方式
- 若用户程序想调内核代码: 须由用户态进入内核态，写一段包含int指令的代码
- 系统调用的实现:

由谁做? 开发语言库函数!

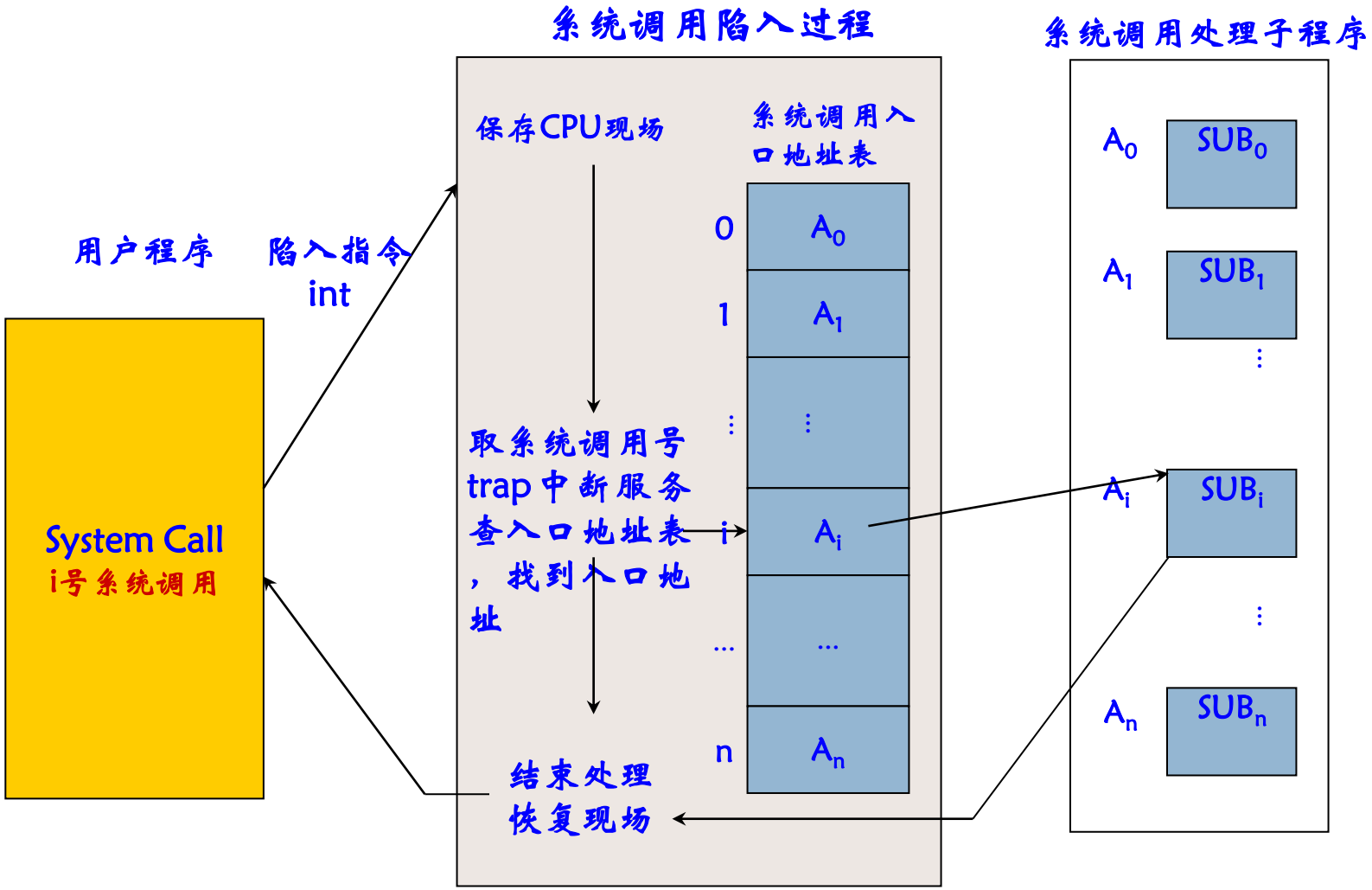
(1) 用户态程序中写上一段包含int指令的代码

(2) OS包含中断处理代码，获取欲调用程序的编号

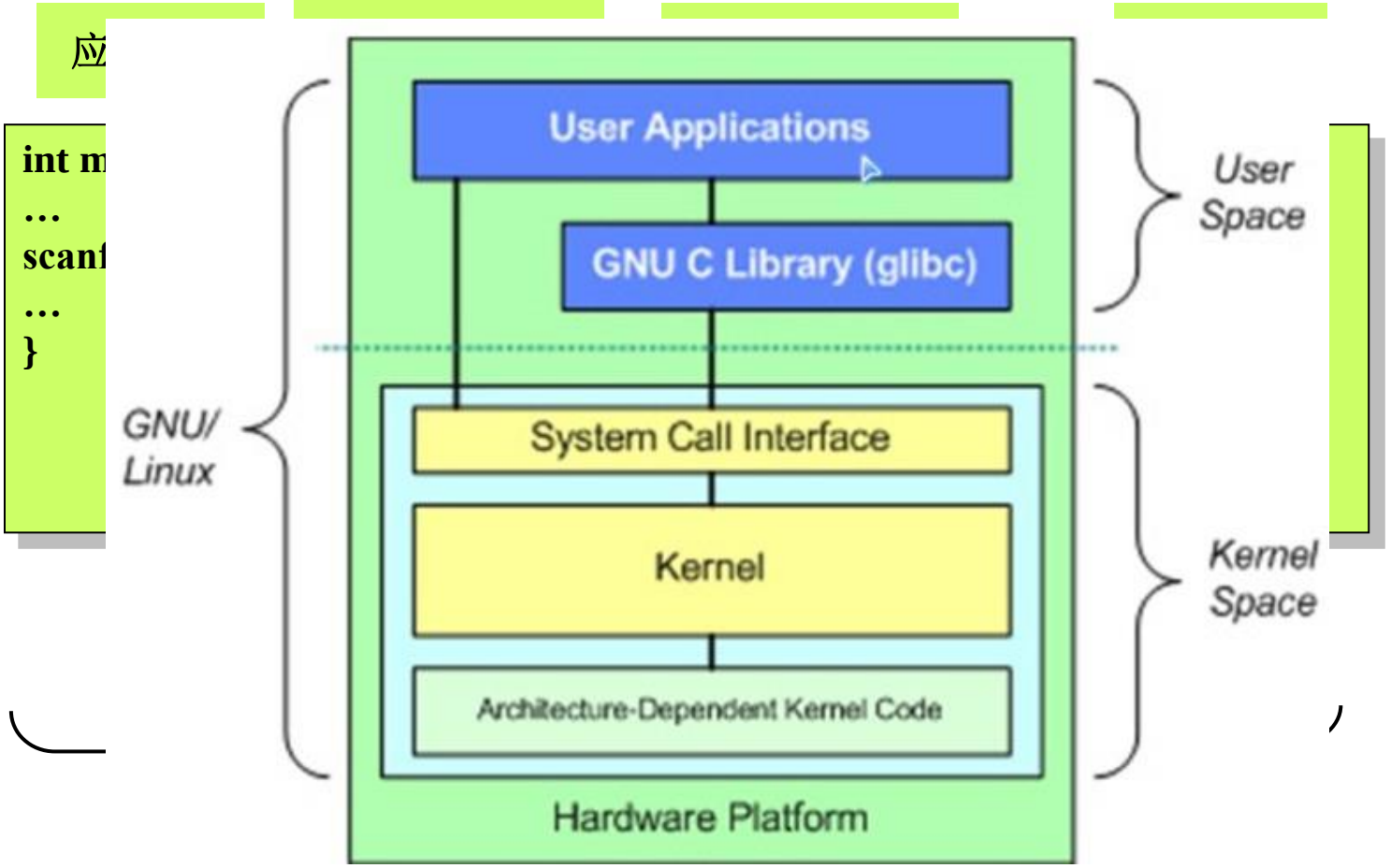
(3) OS根据编号转去执行相应的代码

系统调用编号

2.3 系统调用
系统调用的处理过程



2.3 系统调用
系统调用展开执行示例



2.3 系统调用

系统调用参数传递：规则谁来制定？谁来实现规则？ 答案：编译器

- 系统调用号： 用寄存器eax内容指定
- 最多直接传递6个参数：用寄存器ebx，ecx，edx…
- 有几种传递参数的方式：

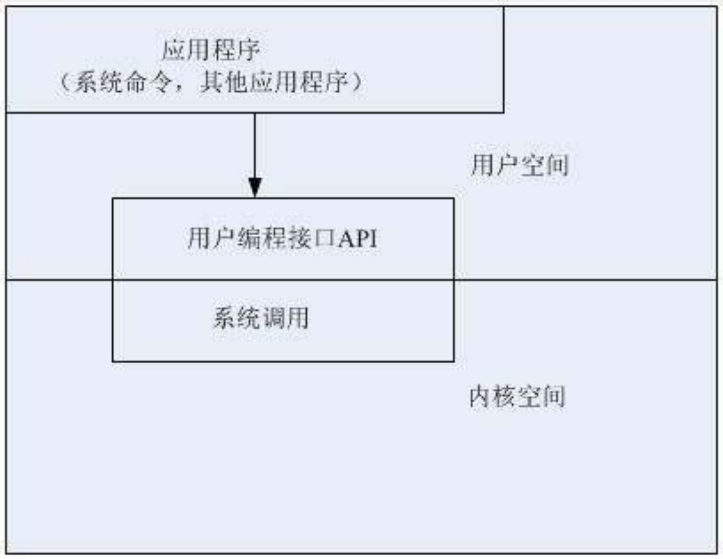
(1) 直接用寄存器。x86_32位，参数则必须按顺序放到寄存器 ebx，ecx，edx，esi，edi ，ebp中。 intel体系的系统调用限制最多六个参数。

(2) 指定内存块。 x86_32位，当系统调用参数大于6个时，全部参数可以依次放在一块连续的内存区域里，同时寄存器 ebx 中保存指向该内存区域的指针

(3) 也可以考虑用系统栈，任意多个参数。用的较少，实现比较复杂。

2.3 系统调用

API与SystemCall的区别与联系?



- API：Application Program Interface
- 很多API是对SystemCall的封装
- Win32 API 2000多个
- POSIX API 100多个
- Java API 10000多个
- Linux2.X版本有200多个系统调用
- SystemCall更底层，使用需知道更多细节
- API是用SystemCall封装而成，使用方便，程序可移植性好

2.4 系统调用类型

五大类：进程控制、文件管理、设备管理、信息维护、通信

- **进程控制：** 创建、装入、执行、终止、等待、唤醒、内存分配与释放… …
- **文件管理：** 创建、删除、打开、关闭、读、写、重定位、属性获取及设置… …
- **设备管理：** 请求、释放、读、写、重定位、属性获得设置、连接与断开
- **信息维护：** 读取/设置系统数据、读取/设置时间及日期、读取/设置进程/文件/设备等属性
- **通　　信：** 创建/删除通信连接、收发消息、连接/断开远端设备

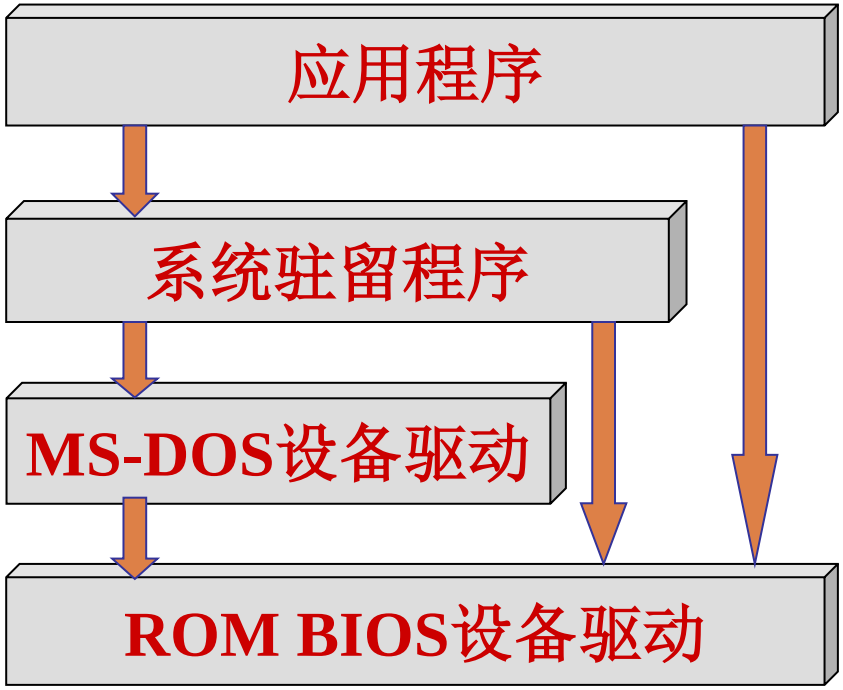
2.5 系统程序

系统程序：介于操作系统与用户应用程序之间的程序，主要提供一个方便的环境，利于开发程序和执行程序

- **文件管理：**创建、删除、复制、命名、备份、格式化等，如资源管理器等
- **系统维护：**监测、设置、性能分析等，如：安装软件，TaskManager（任务管理器），RegEdit（注册表），PS，LS，Kill等
- **程序开发支持：**各类语言编辑/编译器，IDE开发环境，数据库系统
- **通　　信：**远程登录，网上邻居，网页浏览器等

2.6 操作系统结构

1. 简单结构：整个操作系统近乎是个单一的整体，不注重模块的划分和接口与功能层次。典型例子：DOS

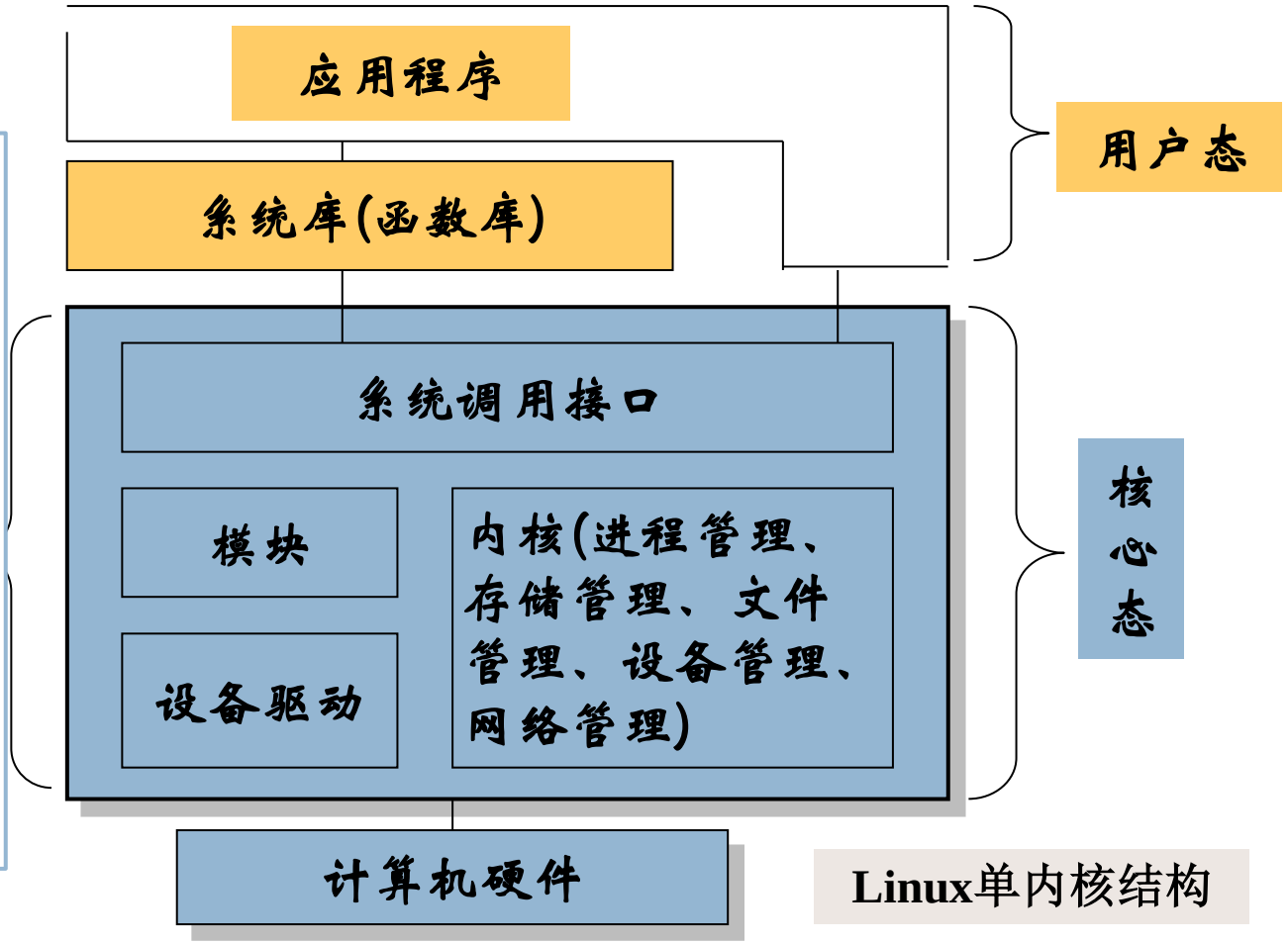


MS-DOS层次结构

2.6 操作系统结构

2. 模块化：面向对象技术生成模块化的内核。使用动态加载模块，现代UNIX, Solaris, Linux, MacOS X

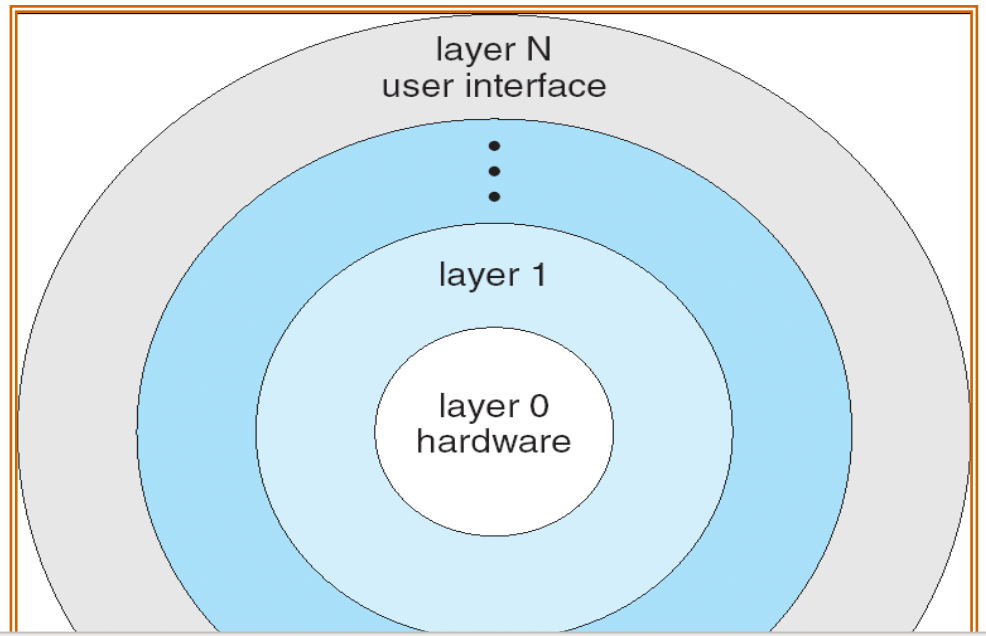
单内核(Monolithic kernel)，有时称之为宏内核Macro kernel：单内核是个很大的进程。他的内部又能够被分为若干模块（或是层次或其他形式）。但是在运行的时候，他是个单独的二进制大映象。其模块间的通讯是通过直接调用其他模块中的函数实现的，而不是消息传递。



Linux单内核结构

2.6 操作系统结构

3. 分层结构：整个操作系统分为若干层，至底向上层层封装



优点是构造和调试的简单化。缺点：分层法主要困难是对层的详细定义，这是因为一层只能使用其下的较低层。分层结构效率低。

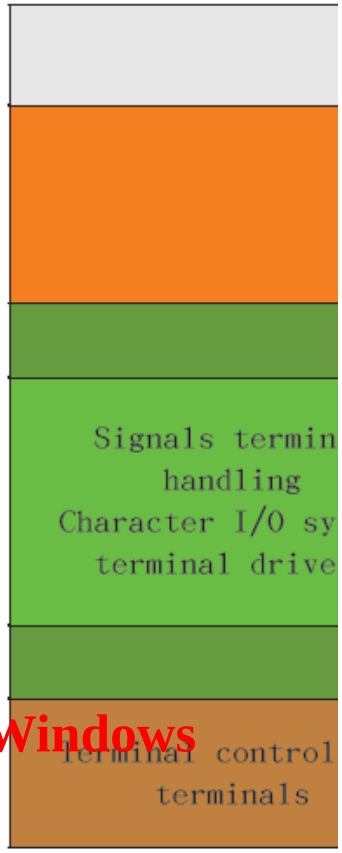
第i层只能调用0..i-1层提供的函数或调用；

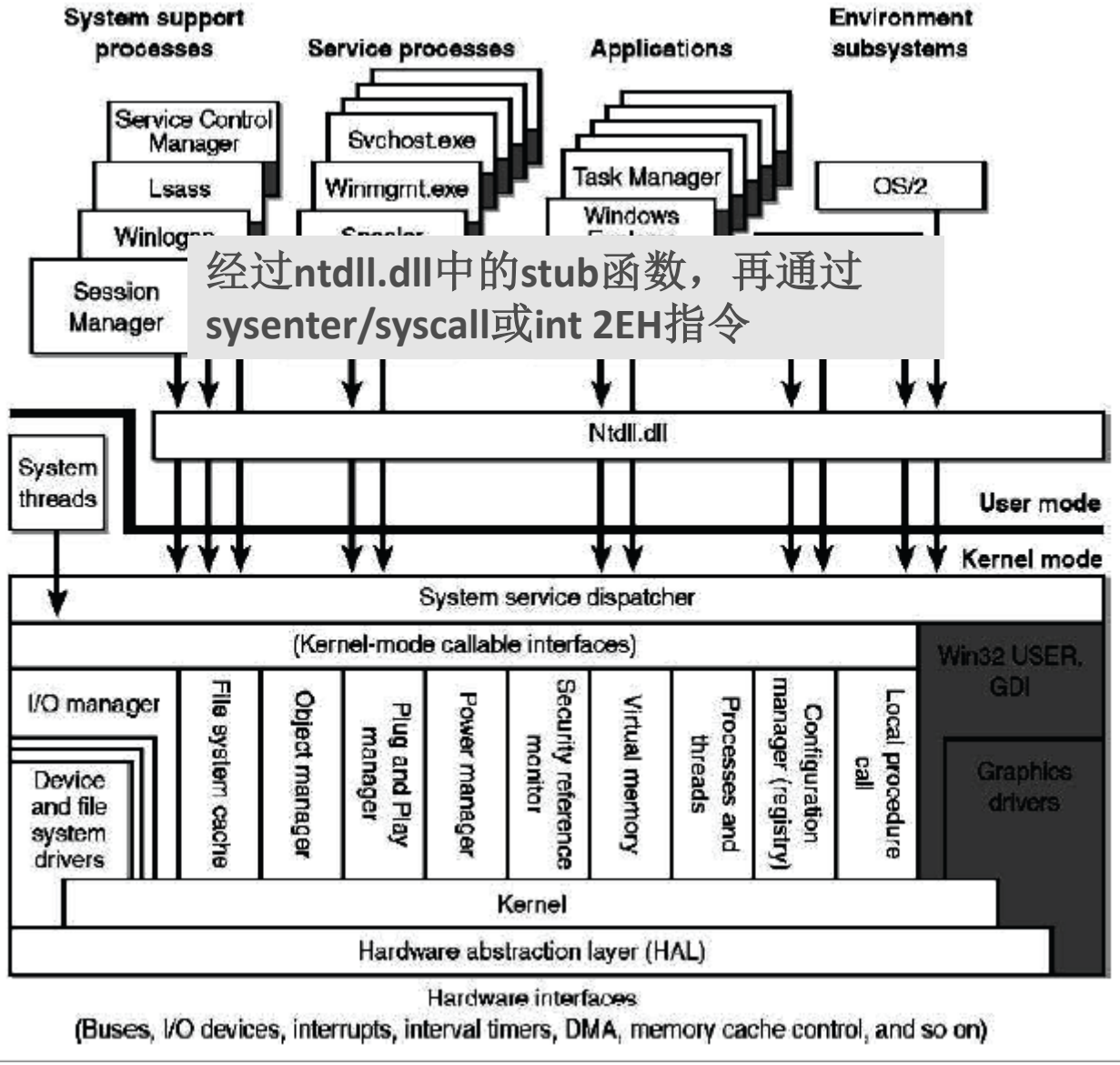
更严格的分层：第i层只能调用i-1层提供的函数或调用

分层操作系统结构模型

2.6 操作系统结构

3. 分层结构

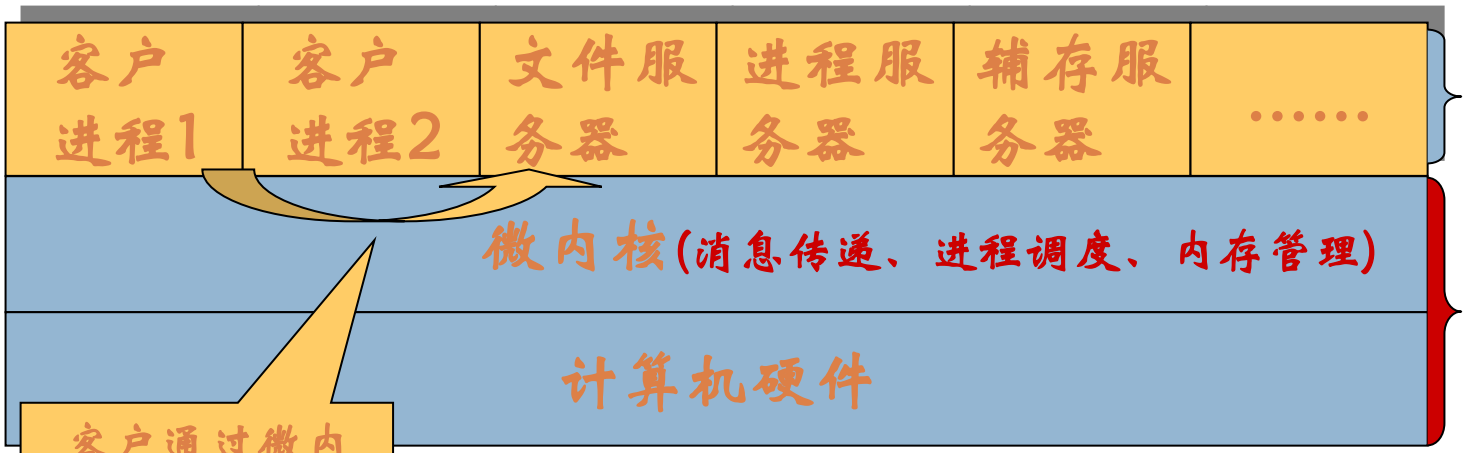




2.6 操作系统结构

4 微内核结构：最基本的功能作为内核，包括最小的进程管理、内存管理和通信功能，其他功能作为系统程序或用户程序出现。采用CS模式（client-server），微内核经常扮演消息传递的功能。很多嵌入式操作系统都是这种结构。

单内核的支持者声称微内核的消息传递开销引起了效率的损失。
微内核的支持者则认为增加的内核设计的**灵活性和可维护性（可裁剪性）**能够弥补开销损失。



微内核结构模型（QNX, Mach,Vxworks等）

2.6 操作系统结构

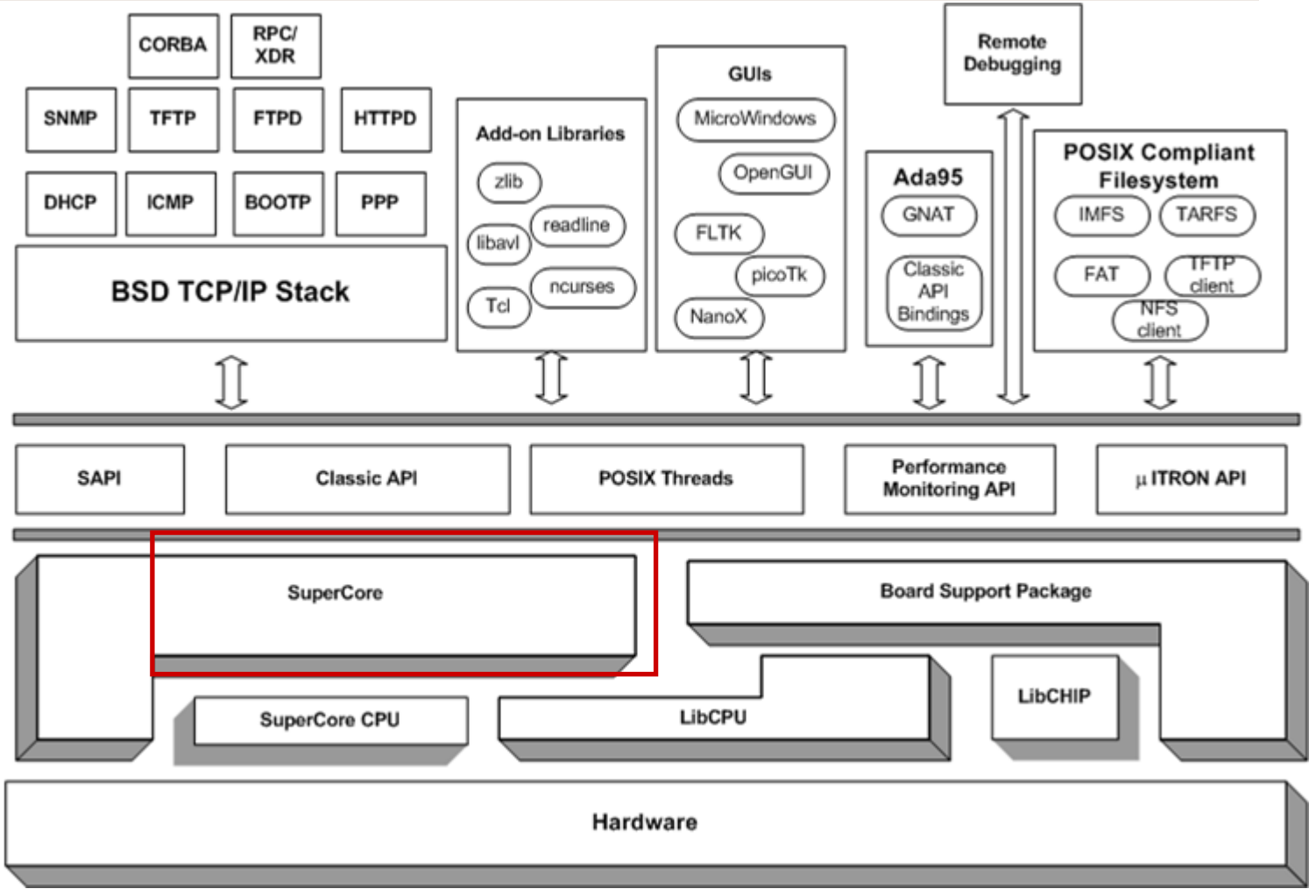
4. 微内核结构：微内核嵌入式操作系统几乎都支持定制裁剪。嵌入式操作系统多数不区分用户态核心态（一个可执行映像）。

- **RTMES: 一种微内核抢占式实时操作系统**
  - 现在： Real Time Executive for Multiprocessor Systems
  - 最早：Real Time Executive for Missile Systems
  - 后来：Real Time Executive for Military Systems
- 4.0.0核心代码约9万行，目前最新版本为5.1
- 维护网络：https://www.rtems.org/

- 1988年为导弹控制系统开发
- 1994年开放源代码
- 2006年水星探测仪，火星探测仪
- 2007年火/木星间小行星探测
- 2008欧洲宇航局Herschel/Plank计划
- 2009 REXUS 5火箭
- 2009 RUAG 空间站管理控制单元
- 2010 THEMIS 卫星
- 2010 RTEMS RFS文件系统

2.6 操作系统结构

4. 微内核结构：RTEMS是使用面向对象思想设计的开源OS



2.6 操作系统结构

4. 微内核结构

- 神舟嵌入式实时操作系统（神舟OS）
  - 针对航天嵌入式关键控制系统的高可靠、高性能、自主可控的嵌入式实时操作系统
  - 航天型号实现小型化、智能化、集成化的关键支撑软件，是保障我国新一代航天装备自主发展的核心基础软件。
  - 内核精简高效，最小可配置8KB
  - 强实时支持，中断响应及任务切换时间微秒级

2.6 操作系统结构

4. 微内核

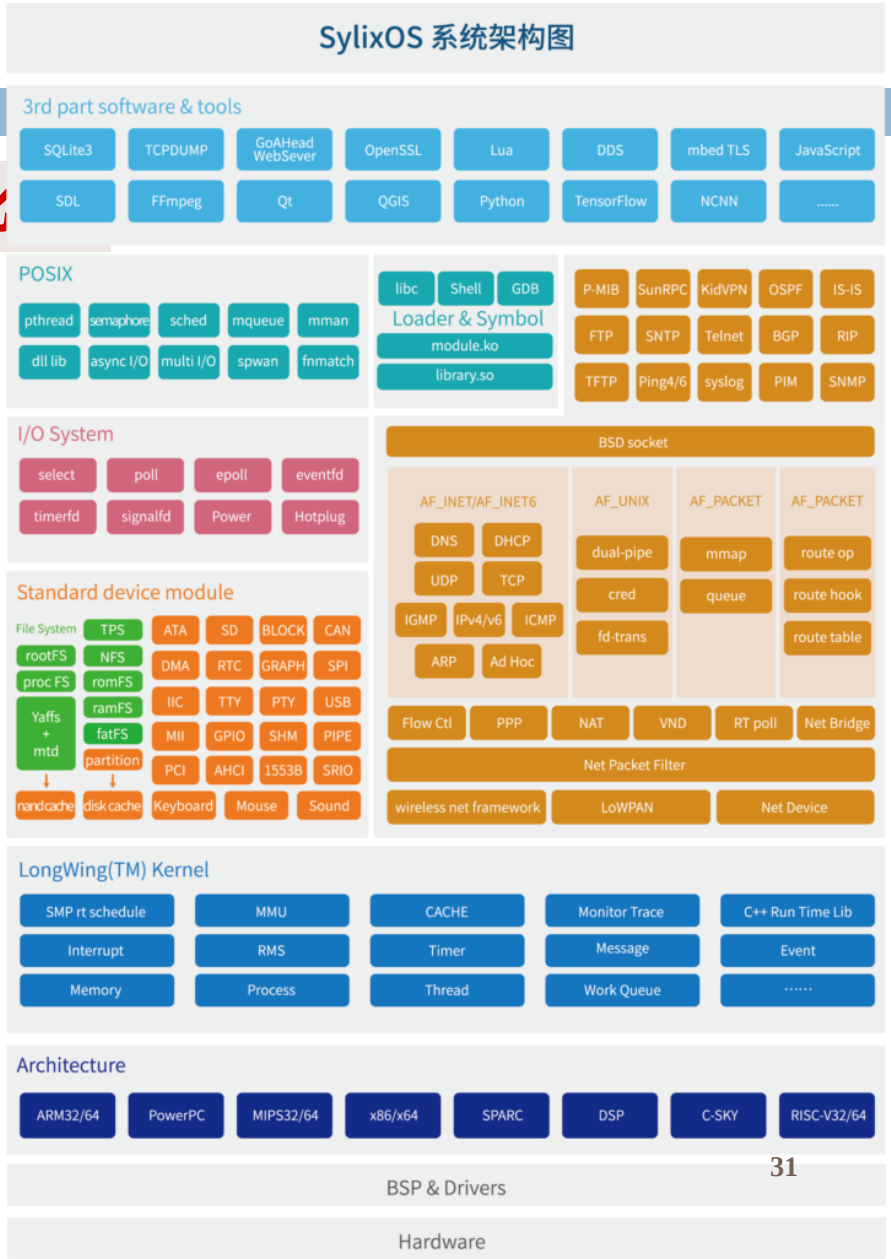
嵌入式操作系统的比较	VxWorks	QNX/Neutrino	uclinux	RTEMS
内存管理	虚拟内存支持是可选项。通常用于不带 MMU 的系统，直接访问物理内存，无内存保护机制（6.0 版本以前）	需要处理器带 MMU，虚拟内存是必须项。内核和系统服务模块受内存保护	不支持虚拟内存，直接访问物理内存。无内存保护机制。	不支持虚拟内存，直接访问物理内存。无内存保护机制
系统 API	非常丰富，超过 1800 个专有 API	丰富	丰富	丰富，自带 API，并且外带支持 pSOS + 和 ITRON 的 API
网络应用	自带 TCP/IP 协议栈，标准 BSD 套接字，RPC 等，并带 FTP，SNMP 等可选，有丰富的网络应用支持	自带完整 BSD 兼容的 TCP/IP 协议栈，还有缩小化的 TCP/IP 协议栈，并带有丰富的网络应用支持，如 ppp，NFS 等。	自带 TCP/IP 协议栈，标准 BSD 套接字，和 Linux 一样，带有极为丰富的网络应用支持	自带 TCP/IP 协议栈，标准 BSD 套接字，并带 FTP，Web-server，NFS，SNMP 等服务
文件系统支持	丰富，自带常见文件系统支持，比如 MSDOS，RawDisk，TrueFFS，ISO 9660，tapeFS 等	丰富，自带常见文件系统，比如 MSDOS，Windows SMB，ISO 9660，Flash 文件系统等	非常丰富，自带常见文件系统支持，如 ROMFS，EXT2，RAMFS，NFS，JFFS，YAFFS 等	较丰富，自带文件系统支持，比如 FAT，IMFS 等

2.6 操作系统结构

4. 微内核结构：几种优秀操作系统比

国产翼辉——硬实时嵌入式OS

- SylixOS 内核自主化率达到 100% （依据工信部评估报告）， 拥有完全自主可控的技术能力，满足国产化需求
- SylixOS **支持对称多处理器（SMP）平台，并且具有实时进程及动态加载机制**，基于优先级的抢占式任务调度
- 支持中断嵌套
- 支持同优先级任务调度
- 支持实时进程
- 互斥量支持优先级继承，防止优先级翻转
- 任务调度时间与负载无关，时间复杂度为 O(1)



https://www.acoinfo.com/product/108/?category=42&subCategory=262&curCategory=108

2.6 操作系统结构

4. 微内核结构：几种优秀操作系统比较

Huawei LiteOS是华为面向IoT领域，构建的轻量级物联网操作系统，可广泛应用于智能家居、个人穿戴、车联网、城市公共服务、制造业等领域。

Huawei LiteOS开源项目目前支持ARM64、ARM Cortex-A、ARM Cortex-M0，Cortex-M3，Cortex-M4，Cortex-M

优势

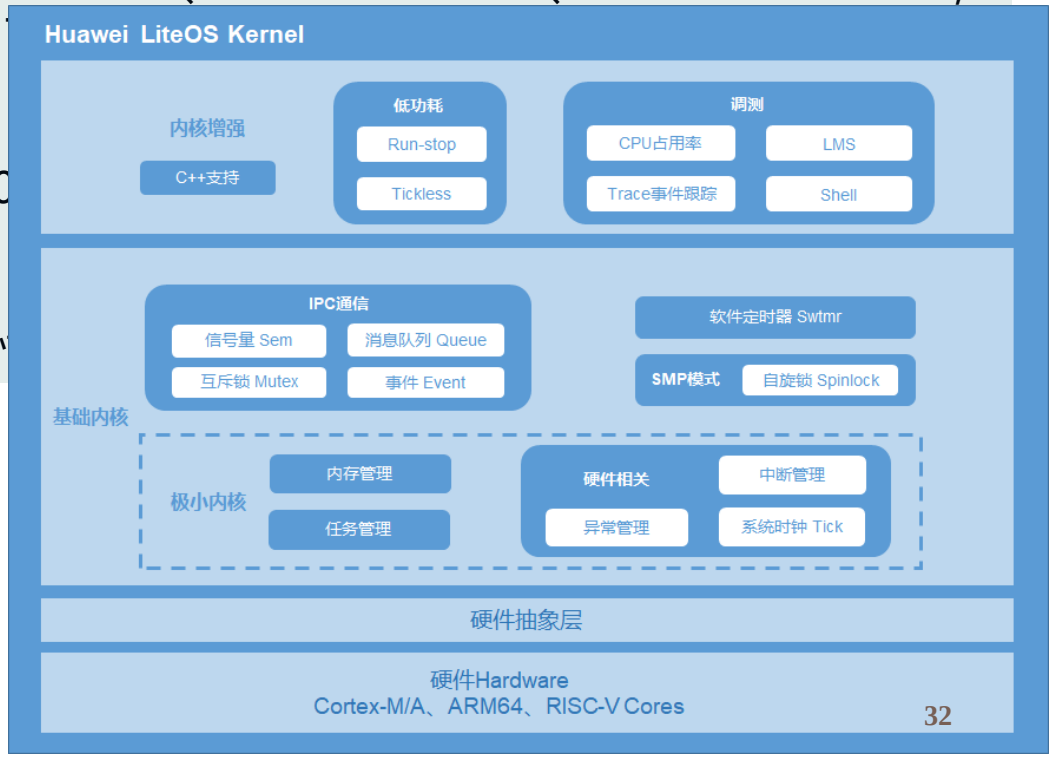
高实时性，高稳定性。

超小内核，基础内核可以裁剪至不到10

低功耗，配套芯片整体功耗低至uA级。

支持功能静态裁剪。

Huawei LiteOS遵循BSD-3开源许可协

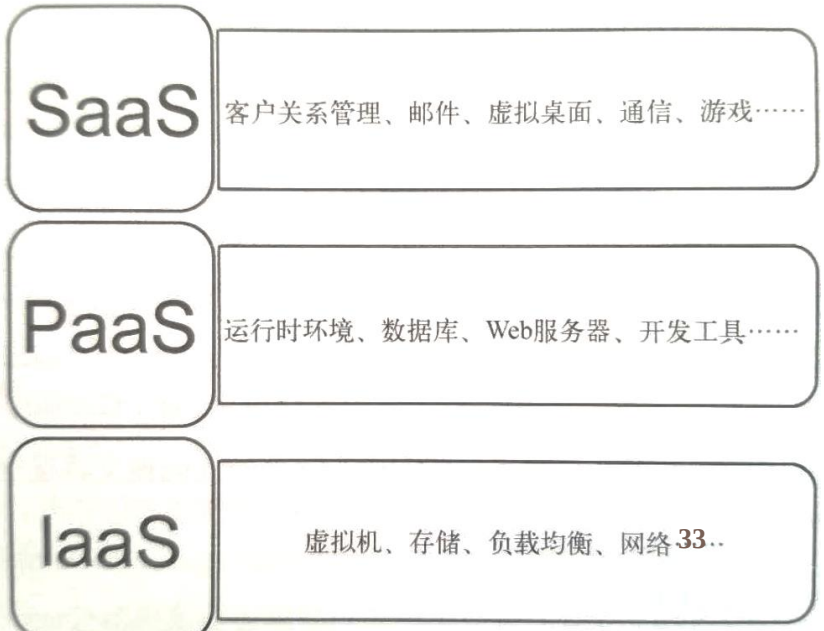


2.6 操作系统结构

5. 分布式特征：

- **分布式资源管理特征：复杂异构的硬件体系统一管理，任务运行时环境差异化需求，硬件资源与服务的动态可伸缩配置**
- **虚拟机与容器：vmware，docker，kubernets（容器管理）**
- 云平台框架：Openstack
- 分布式文件：hadoop等
- 微服务框架：Spring boot, cloud spring, dubbo, zookeeper
- 内存计算：spark，redis
- 消息管理：Apache Kafka
- 结构化与非结构化数据库：
  mysql，mongdb
- 。。。

云计算操作系统层→



2.6 操作系统结构

5. 分布式特征：阿里飞天、AliOS、百度DureOS、ROS

- **飞天（阿里云自主研发计算机操作系统）**
- 飞天（Apsara）是由阿里云自主研发、服务全球的超大规模通用计算操作系统。
- 它可以将遍布全球的百万级服务器连成一台超级计算机，以在线公共服务的方式为社会提供计算能力。
- 飞天的革命性在于将云计算的三个方向整合起来：提供足够强大的计算能力，提供通用的计算能力，提供普惠的计算能力。

2.6 操作系统结构

5. 分布式特征：阿里飞天

飞天操作系统的核心竞争力和核心能力是什么?

. 自主可控：对云计算底层技术体系的把控力，（自主）研发，自己解决核心问题。
. 调度能力：10K（单集群1万台服务器）的任务分布式部署和监控。
. 数据能力：EB（10亿GB）级的大数据存储和分析能力。
. 安全能力：为中国35%的网站提供防御。
. 大规模实践：经受双11、12306春运购票等极限并发场景挑战。
. 开放的生态：兼容大多数生态软件和硬件，比如Cloud foundry、Docker、 Hadoop。

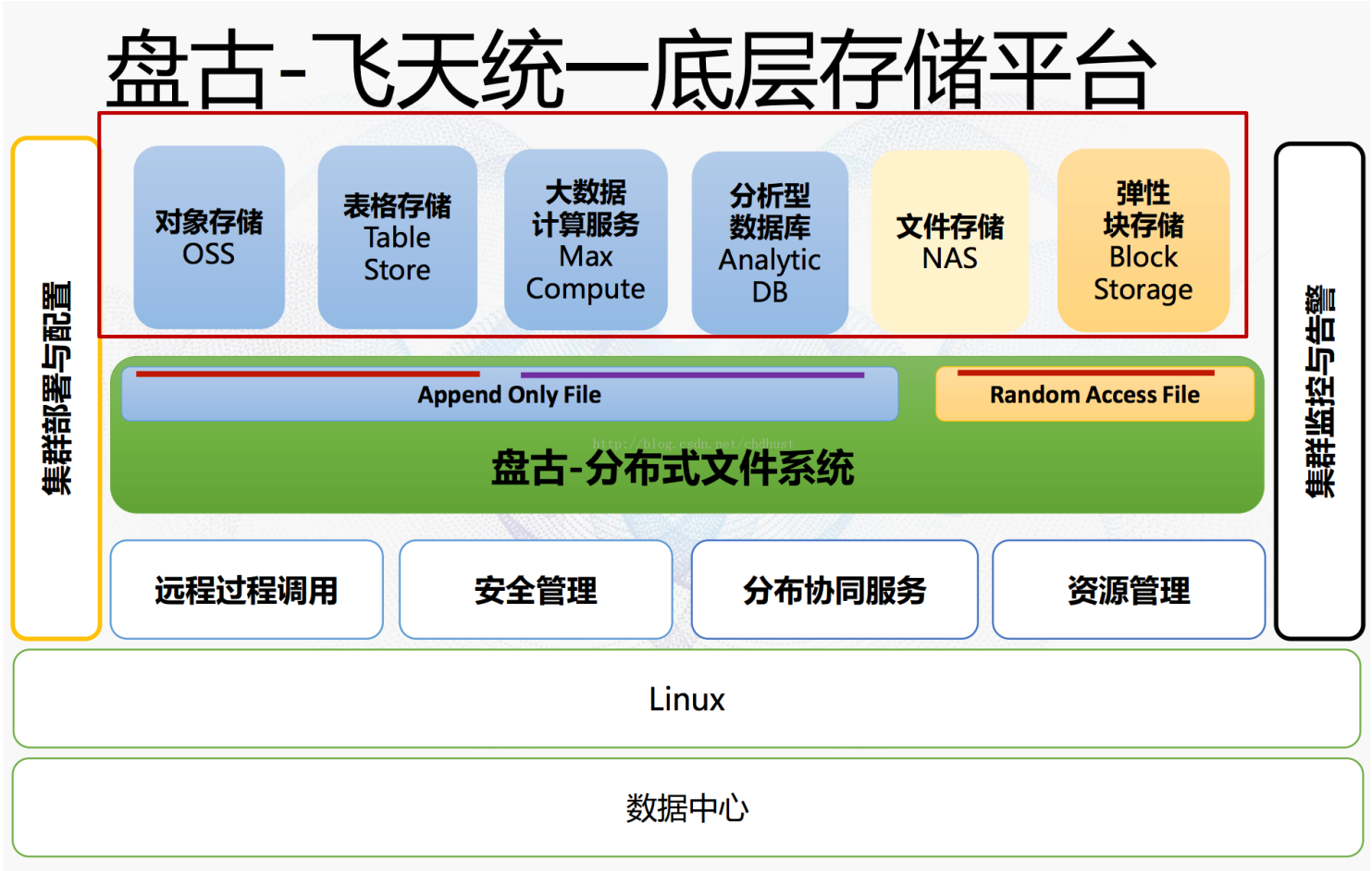
2.6 操作系统结构

5. 分布式特征：阿里飞天OS的整体结构

云引擎 **ACE**（AliCloud Engine）是**阿里云**推出的应用托管平台，支持托管Java、PHP、Node.js、Python等语言开发的Web应用和API服务。只需上传开发完成的应用代码，**ACE**会自动完成资源分配、运行环境初始化、程序启动、负载均衡配置、运行状况监控、日志采集、自动伸缩等一系列连贯的部署工作。

2.6 操作系统结构

5. 分布式特征：阿里飞天OS的整体结构



2.6 操作系统结构

5. 分布式特征：AliOS, AliOS-Things

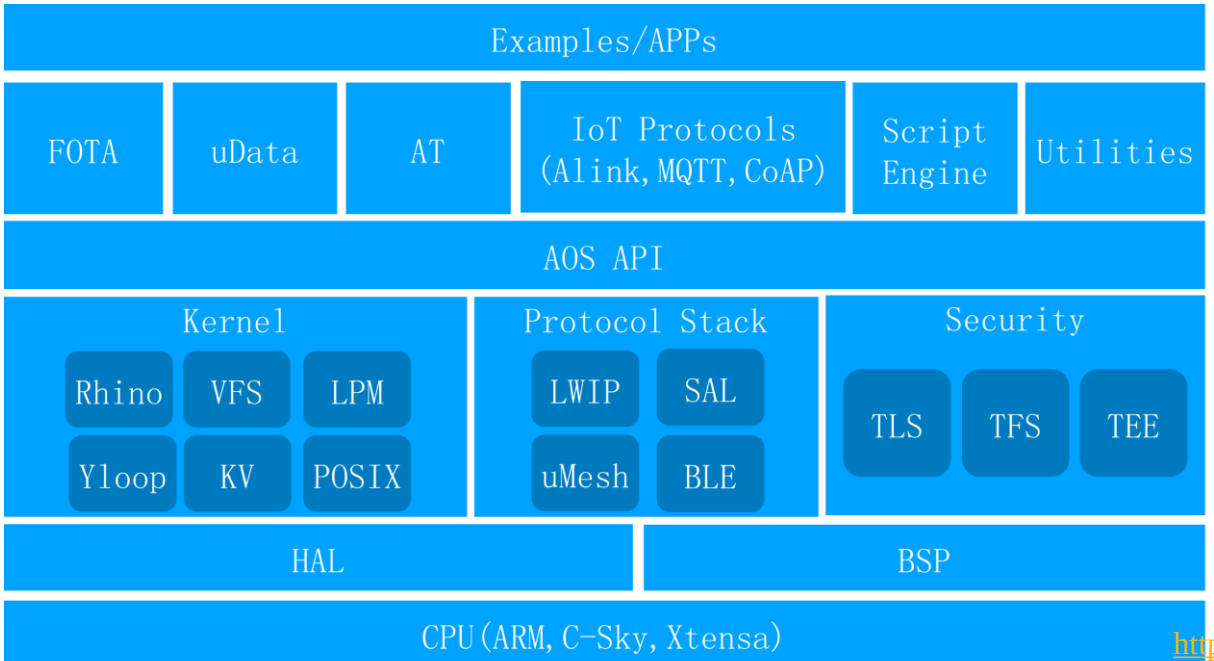
- AliOS拥有高效的系统内核,基于云的应用框架和先进的安全机制。AliOS以驱动万物智能为目标，可应用于智联网汽车、智能家居、手机、Pad等智能终端，为行业提供一站式IoT解决方案，构建IoT云端一体化生态，使物联网终端更加智能。AliOS 正在定义一个不同于PC和移动时代的物联网操作系统。
- 构建物联设备智能中枢为核心，充分利用云端协同计算、融合智能感知、生态互联互通等优势
- AliOS Things是面向IoT领域的轻量级物联网嵌入式操作系统。致力于搭建云端一体化IoT基础设备。具备极致性能，极简开发、云端一体、丰富组件、安全防护等关键能力，并支持终端设备连接到阿里云Link，可广泛应用在智能家居、智慧城市、新出行等领域。

2.6 操作系统结构

5. 分布式特征：AliOS, AliOS-Things

AliOS Things的架构可以适用于分层架构和组件化架构。从底部到顶部，AliOS Things包括：

- 板级支持包（BSP）：主要由SoC供应商开发和维护
- 硬件抽象层（HAL）：比如WiFi和UART
- 内核：包括Rhino实时操作系统内核、Yloop（异步事件框架），VFS（虚拟文件系统），KV 存储（ Key-Value持久化存储的轻量级组件）

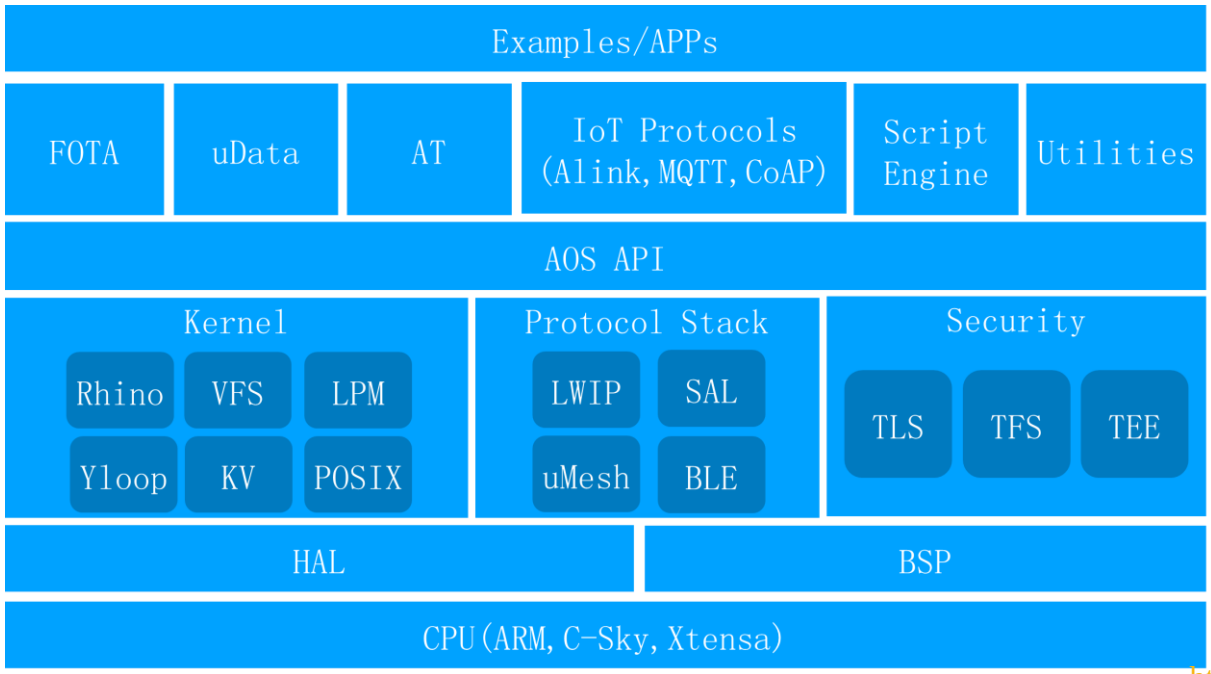


https://developer.aliyun.com/article/616983

2.6 操作系统结构

5. 分布式特征：AliOS, AliOS-Things

- 协议栈：包括TCP/IP协议栈（LwIP），uMesh网络协议栈
- 安全：安全传输层协议（TLS），可信服务框架（TFS）、可信运行环境（TEE）
- AOS API：提供可供应用软件和中间件使用的API
- 中间件：包括常见的物联网组件和阿里巴巴增值服务中间件
- https://github.com/alibaba/AliOS-Things



https://developer.aliyun.com/article/616983

2.6 操作系统结构

5. 分布式特征：百度DureOS

- DuerOS拥有海量数据，能通过自然语言完成对硬件的操作与对话交流， 为用户提供完整的服务链条。
- 作为一款开放式的操作系统， DuerOS通过云端大脑时刻进行自动学习让机器具备人类的语言能力。

2.6 操作系统结构

5. 分布式特征：百度DureOS

- DuerOS是百度人工智能技术的集大成者，拥有业界领先的人工智能技术和先天优势。
- 算法：拥有建立在超大规模神经网络、万亿级参数、千亿级样本上的人工智能算法；
- 计算：数十万服务器和中国最大GPU集群的计算能力；
- 数据：累计了全网万亿网页、数十亿搜索、百亿级图像视频和定位数据；
- 语音识别：语音识别准确率97%以上；
- 图像识别：人脸识别准确率99.7%；
- 自然语言处理：能用自然语言与用户交流，理解用户意图；
- 用户画像：拥有近10亿用户画像。
- DuerOS可以广泛适用于家居、随身、车载等多种场景，支持音箱、电视、冰箱、手机、机器人、车载、手表等多种硬件设备。

2.7 虚拟化技术

- 计算机是由运算器、控制器、存储器、输入、输出这五部分组成。
- 使用操作系统来驱动这些设备，让它们协同合作，这就变成了一台完整的计算机。
- 随着计算任务的变化，我们对计算机有了更多的需求。

1）已经装了windows系统，你却想让他再跑linux

2）你现在的电脑计算能力不够了，但是加上你同学的电脑就可以了。

也就是说，我们想要计算机的资源能够具有很强的弹性伸缩能力，可任意增长，也可任意消减。

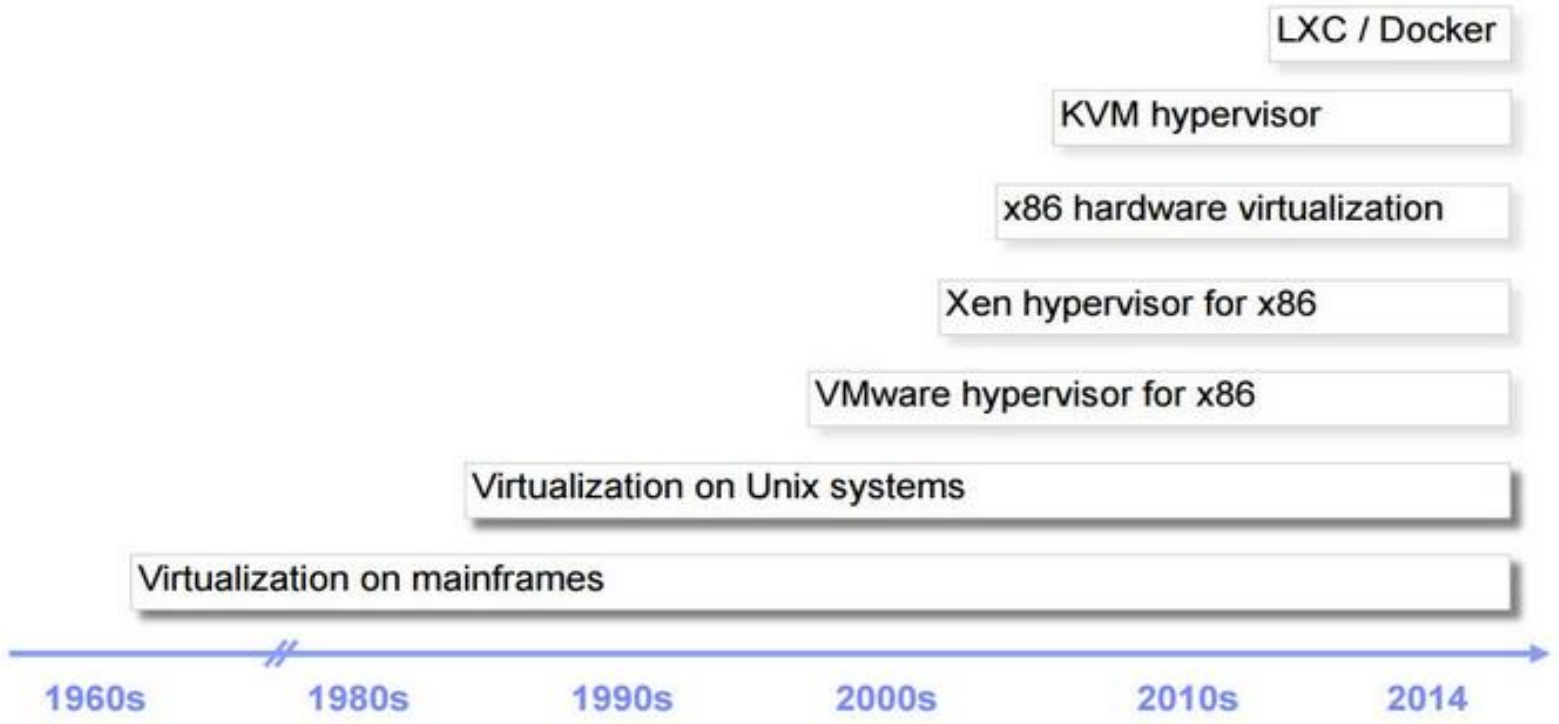
2.7 虚拟化技术

- 使用软件的方法重新定义和划分IT资源，可以实现IT资源的动态分配、灵活调度、跨域共享，提高IT资源利用率，使IT资源能够真正成为社会基础设施，服务于各行各业中灵活多变的应用需求。

- 虚拟化目前有两个方向，其一是把一个物理机虚拟成多个独立的逻辑虚拟机；其二是把若干分散的物理机虚拟为一个大的逻辑虚拟机。

2.7 虚拟化技术

- 虚拟化技术其实是很早就提出来的概念，其技术迭代如下图所示。
- 现在处于容器级虚拟化阶段，也就是将应用程序、开发软件包、依赖环境等统一打包到容器中，将整个容器部署至其他的平台或者服务器上。

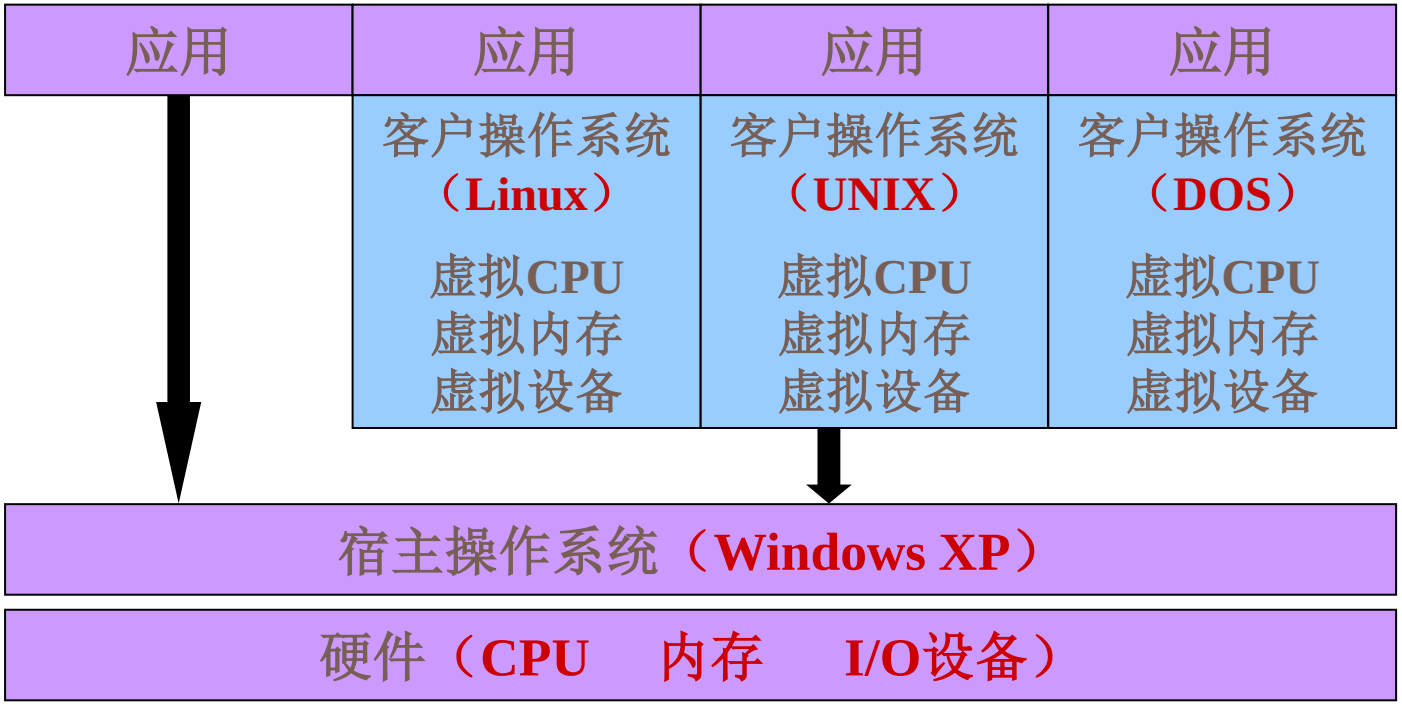


2.8 主机虚拟化-虚拟机

- **主机级别的虚拟化**是通过虚拟化技术，可以将物理资源转变为逻辑资源(虚拟机)，应用程序运行在虚拟资源上。因此，**对于主机级虚拟化，一般是模拟出硬件环境，模拟出cpu、内存、硬盘、网卡等资源，然后在这些虚拟资源之上安装合适的操作系统来控制这些资源。**
- **在当前硬件提供的VMM之上**，可以模拟出逻辑的计算机虚拟环境，然后安装操作系统，使其成为一台逻辑上虚拟的计算机主机，该主机有自己的系统内核，有自己的用户空间，可以在自己的用户空间内跑各种各样的应用程序。

2.8主机虚拟化-虚拟机

虚拟机：是一种操作系统技术中的特殊结构。

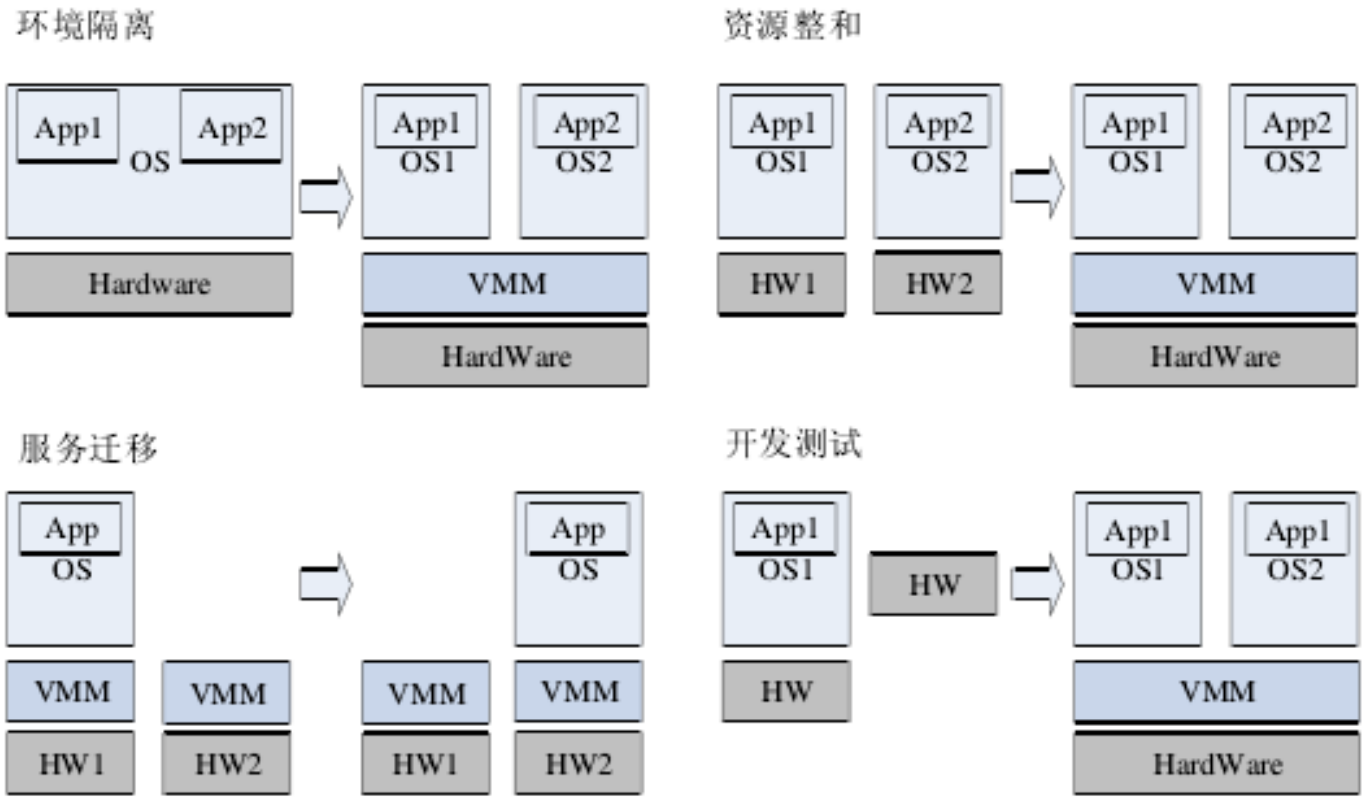


VMWare结构模型

2.8主机虚拟化-虚拟机

虚拟机：虚拟机监控器VMM+虚拟机VM

典型的云计算平台：openstack、阿里云



2.8主机虚拟化-虚拟机

虚拟机实现技术：纯软件虚拟化与硬件支持的虚拟化

- 传统x86机器的虚拟化：虚拟机监视器软件(VMM）代表客户操作系统来监听中断与执行特定指令，效率低。
- VT-x是intel运用Virtualization虚拟化技术中的一个指令集，是CPU的硬件虚拟化技术，VT可以同时提升虚拟化效率和虚拟机的安全性，在x86平台上的VT技术，一般称之为VT-x。

2.8主机虚拟化-虚拟机

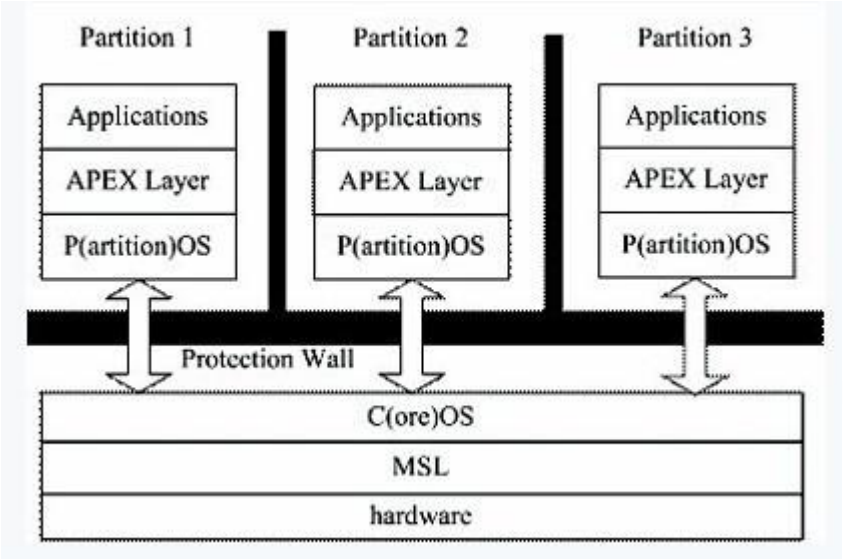
虚拟机实现技术

- 为了建立这种两个操作模式的架构，VT-x设计了一个Virtual-Machine Control Structure（**VMCS，虚拟机控制结构**）的数据结构；
- 包括了Guest-State Area（客户状态区）和Host-State Area（主机状态区），用来保存虚拟机以及主机的各种状态参数；
- 并提供了VM entry和VM exit两种操作在虚拟机与VMM之间切换
- **用户可以通过在VMCS的VM-execution control fields里面指定在执行何种指令/发生何种事件的时候，虚拟机就执行VM exit，从而让VMM获得控制权，因此VT-x解决了虚拟机的隔离问题，又解决了性能问题。**

2.8主机虚拟化-分区操作系统Arinc653

- **在传统的嵌入式实时操作系统中，内核和应用都运行在同一特权级，应用程序可以无限制的访问整个系统地址空间。因此在某些情况下，应用的潜在危险动作会影响其他应用和内核的正常运行，甚至导致系统崩溃或者误操作。**
- 为了满足航空电子对高可靠性、高可用性以及高服务性的要求, 1997年1月ARINC发布了ARINC653（航空电子应用软件标准接口），并于2003年7月发布ARINC653 Supplement 1，对区间管理、区间通信及健康监测部分进行了补充说明，用以规范航空电子设备和系统的开发。 分区（Partitioning）是ARINC653中一个核心概念。
- **采用arinc653标准的操作系统设计原理将传统操作系统分为两级，一个是CoreOS，作用是区间化以及区间的管理和调度，CoreOS的上层就是POS，即分区操作系统，在POS的上层才是应用程序的执行。**

2.8主机虚拟化-分区操作系统Arinc653



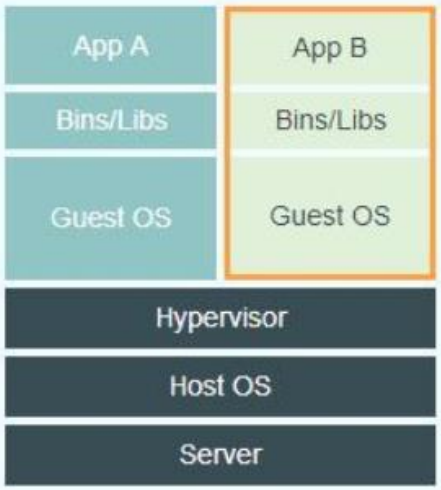
2.9 容器虚拟化-Docker容器

容器虚拟化

- 容器是一种虚拟化的方案，和传统的虚拟机(通过中间层“guest OS”运行服务)不同，Docker直接运行在操作系统之上。因此容器虚拟化也被称之为操作系统虚拟化。
- 容器虚拟化技术相当于把操作系统进行虚拟化，把物理的操作系统模拟为逻辑上的多个操作系统，不同的操作系统有**自己的用户空间**，实现了应用程序间的隔离。

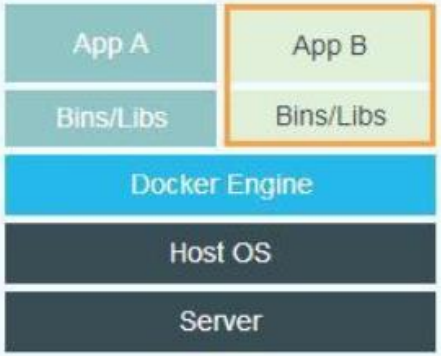
设想：一个应用程序运行需要复杂的运行环境（基础软件、组件、配置文件、系统环境变量等），将其部署或需要动态部署和迁移时怎么办？

2.9 Docker容器与虚拟机结构对比



Virtual Machines

Each virtualized application includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may weigh 10s of GB.



Docker

The Docker Engine container comprises just the application and its dependencies. It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.

2.9 Docker容器

容器虚拟化——Docker使用场景（资源隔离与资源配额）

- 1）使用Docker容器开发、测试、部署服务: Docker本身是轻量级的，所以本地开发人员可以构建、运行并分享Docker容器，容器可以在开发环境中创建，然后提交到测试，在到生产环境（大企业大系统开发过程支持）。
- 2）创建隔离的运行环境: 在很多企业应用中，同一服务的不同版本可能服务于不同的用户，使用Docker很容易创建不同的环境来运行不同版本的服务
- 3）搭建测试环境: Docker轻量化，开发者很容易在本地搭建测试环境，来测试程序在不同系统下的兼容性，甚至集群式的测试环境。
- 4）构建多用户的基础设施平台服务(PaaS)，进行高性能、超大规模的宿主机服务部署以及硬件资源能力分配。

2.9 Docker容器

容器技术的优劣

- 举一个例子来说明容器技术的优越性，比如说同时要在虚拟机和容器上都部署Nginx服务器，分析下这二者之间的区别。
  - 传统虚拟化：硬件服务器-HostOS-**VMM-GuestOS**-Nginx WEB服务
  - Docker虚拟化：硬件服务器-HostOS-**Docker**-Nginx WEB服务
  - 传统的虚拟化技术多了一层要去启动虚拟机的操作系统的流程，而虚拟机的操作系统运行时非常消耗内存资源的。
  - 在资源的利用上，以及响应速度上，容器虚拟化技术要优越于主机虚拟化技术。

2.9 Docker容器

容器技术的目标

- Docker目标提供简单轻量级的虚拟化方式（Docker的启动是毫秒/秒级的）
- 职责的逻辑分离：开发人员只需要关注容器中运行的程序，运维人员只需要关注对容器的管理。
- 快速高效的开发周期：缩短代码从开发、测试到部署上线的生命周期
- **多面向微服务的架构：Docker推荐单个容器只运行一个应用程序/进程，这样就形成了一个分布式的应用程序模型，避免服务之间的互相影响。实现高内聚，低耦合。**

动态部署,动态扩容

2.9 Docker容器

Docker微服务架构涉及的技术体系：

- 高性能：消息队列MQ、RxJava异步并发、分布式缓存、本地缓存、Http的Etag缓存、使用Elasticsearch优化查询、CDN等等。
- 可用性：容器服务集群、RxJava的熔断处理、服务降级、消息的幂等处理、超时机制、重试机制、分布式最终一致性等等。
- 伸缩性：服务器集群的伸缩、容器编排Kubernetes、数据库分库分表、Nosql的线性伸缩、搜索集群的可伸缩等等。
- 扩展性：基于Docker的微服务本身就是为了扩展性而生！
- 安全性：JPA/Hibernate,SpringSecurity、高防IP、日志监控、Https、Nginx反向代理、HTTP/2.0等等

本章主要内容回顾

讨论操作系统结构，一般包括3个方面：

- 操作系统所提供的服务
  - 2.1 操作系统服务
- 操作系统为用户和程序员提供的接口
  - 2.2 操作系统的用户界面
  - 2.3 系统调用（重点）
  - 2.4 系统调用类型
  - 2.5 系统程序
- 操作系统结构及其相互关系
  - 2.6 操作系统结构
  - 2.7 虚拟化技术
  - 2.8 虚拟机与分区操作系统
  - 2.9 Docker容器