

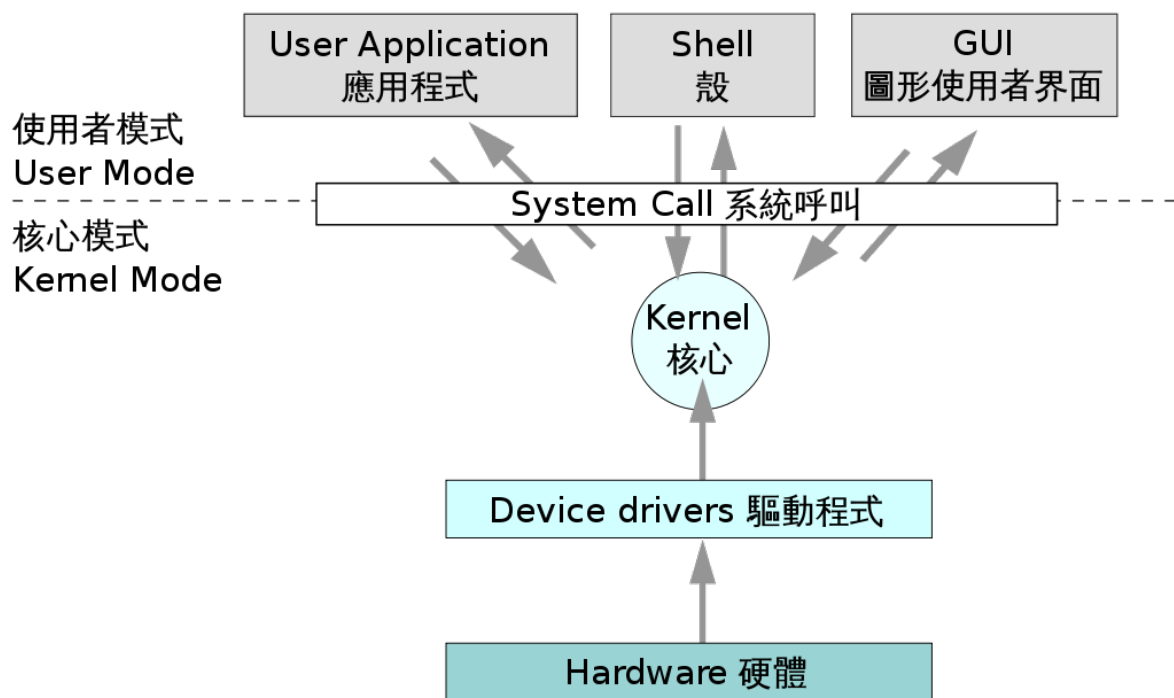
Linux系统在开发中的应用

军品三室 陈博凯

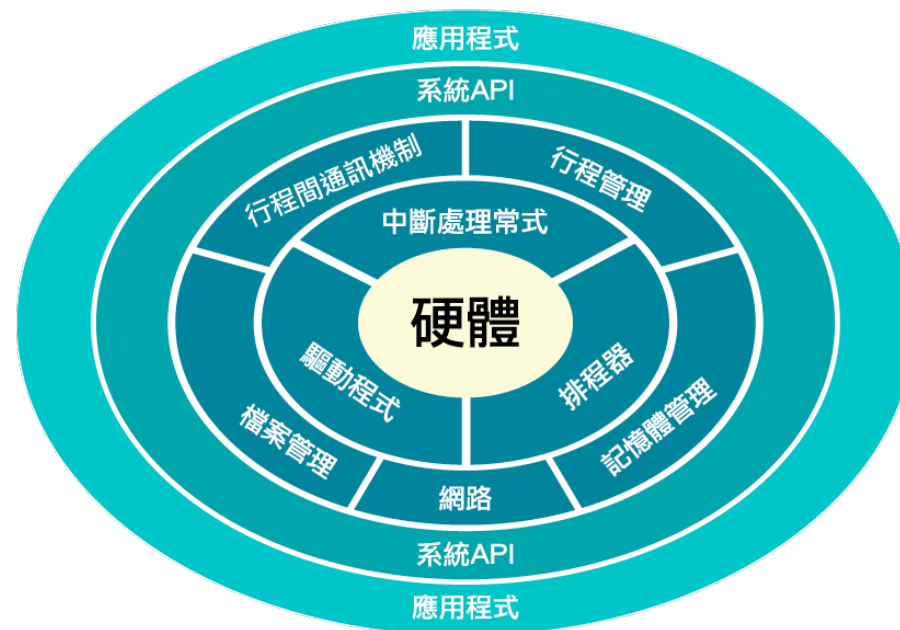
1. 了解操作系统演变过程
2. 掌握linux启动过程与模式转换
3. 了解Linux内核调度机制
4. 梳理开发时用到的操作系统的知识
5. 介绍演示构建的Linux操作系统
6. 延伸的操作系统介绍

操作系统：Operating System OS

是一组主管并控制[计算机](#)操作、运用和运行[硬件](#)、[软件资源](#)和提供公共[服务](#)来组织用户交互的相互关联的[系统软件程序](#)，同时也是计算机系统的内核与基石。



Linux 架構示意圖



操作系统演变过程

《洛杉矶时报》要选50名20世纪在经济界最有影响力的人物。第一名是三个人并列，分别是威廉肖克利，他发明的是晶体管，还有两位是诺伊斯和基尔比，他俩发明的是集成电路，也叫芯片。

在1957年夏天，肖克利实验室股份有限公司的7个家伙到旧金山的一家酒店里，会见了两个投资者。这七个哥们都是在IT历史上留下姓名的人，他们分别是戈登·摩尔（Gordon Moore）、C.谢尔顿·罗伯茨（C. Sheldon Roberts）、尤金·克莱纳（Eugene Kleiner）、维克多·格林尼许（Victor Grinich），朱利叶斯·布兰克（Julius Blank），杰·拉斯特（Jay Last）还有金-赫尔尼(Jean Hoerni)。

他们的大哥就是鲍勃·诺伊斯，正是这8个人从肖克利公司的出走，组建了仙童公司。仙童半导体之后成了硅谷所有公司的父亲，从仙童走出的人，创建了几百家公司，上市的公司就有92家。在1969年在森尼维尔举办的一次半导体工程师大会上，400名与会者当中，未曾在仙童半导体工作的只有24人。

这个公司成立后主营的业务是晶体管，后来转化为集成电路。10年的时间，创立了Intel、DEC、AMD、国家半导体等等硅谷公司。

在七八十年代，大型电脑公司真的是从CPU一路包到软件，IBM和DEC都是此列；Apple、Commodore之类的中小型电脑公司则是外购芯片，然后从整机包到软件。事实上在那个时代软件是硬件的附庸。

IBM PC 这个异类，IBM通过开放设计架构使得硬件得以标准化。这使得：

- PC 兼容机的成本降低，快速占据市场；
- PC 兼容机厂商迅速碎片化，整机厂商包括IBM逐渐失去话语权；
- 上下游的芯片厂商和软件厂商做大，并可以主导PC标准。

操作系统演变过程

第一、二代计算机（真空管和穿孔卡片、晶体管和批处理）
没有操作系统的概念 所有的程序设计都是直接操控硬件

第三代计算机（集成电路芯片和多道程序设计）
IBM公司试图通过引入system/360系列来同时满足科学计算和商业计算

第四代计算机（个人计算机）

1975年Unix诞生，对整个后续操作系统发展具有深远的影响

1980年DARPA（机构）TCP/IP在unix上诞生

1980年microsoft在unix上开发了上

1981年QDOS发布

1984年Minix系统发布

SUN：Solaris IBM*AIX HP：HP-UX随后几年陆续发布

1991年芬兰人开发了Linux，基于POSIX的多用户、多任务并支持多线程和多CPU的操作系统

发行商包括Slackware、Red Hat、Debian、Fedora、TurboLinux、Mandrake、SUSE、CentOS、Ubuntu、红旗、麒麟……

<https://liuyandong.com/>

刘延栋的赛博空间

主电台《忽软忽硬》

成一家之言，毕竟有言胜于无。

关于我

农村做题家，曾笃信“学好数理化，走遍天下都不怕”，在大学里热衷玩游戏，就去学计算机了。研究生搞网络和加密，就成了吃饭的饭碗。

年过35后，本着“人过留名，雁过留声”的态度，做了个电台叫“软件那些事儿”，在[网易云音乐](#)，[喜马拉雅App](#)和[YouTube](#)上。不要白来这个世间一遭。

由于被混球举报，我的公众号在2021年9月28日，被永久封号了。123年前的9月28日，谭嗣同去世，我正在编辑一篇纪念谭嗣同的文章，顺便介绍李敖的《北京法源寺》，手机突然收到了被封号的信息，随后，浏览器也退出了。文章没有备份下来。

后来我又注册了一个新的公众号，叫“忽软忽硬”，权当一个联系方式，以后的音频和文稿，都不会再在公众号上更新了，更新电台后，会在公众号上发一个通知。下面是新公众号的二维码。

操作系统演变过程

1991年，在[赫尔辛基](#)，[Linus Torvalds](#)开始那个后面成为了[Linux内核](#)的项目。最初它只是一个Torvalds用来访问大学里的大型Unix服务器的[虚拟终端](#)。他专门写了一个用于他当时正在用的硬件的，与操作系统无关的程序，因为他要用他那用80386处理器的新PC机的功能。开发是在[Minix](#)上，用至今仍为首选的[编译器](#)——[GCC](#)——来完成的。

Linux操作系统发展中的重要因素：

内核

社区

开源发展实验室和Linux基金会

相关公司



操作系统演变过程

批处理系统-》分时操作系统-》multics-》unix-》linux-》发行各种linux版

- 1、开源、不收费的操作系统，可自由传播(windows 操作系统收费，不开源)
- 2、没有任何的商业化版权制约(也会有商业发行版，但我们都不用)
- 3、linux支持多用户多任务多线程多cpu，主要用于企业环境

linux主要用于：

- 1、服务器
- 2、嵌入式开发
- 3、个人pc桌面

掌握linux启动过程与模式转换

POST（加电自检）→加载BIOS（Basic Input/Output System）→确定启动设备（Boot sequence）、加载Boot Loader→加载内核（kernel）→打印用户登录提示符

POST开机自检

该过程主要对计算机各种硬件设备进行检测，如CPU、内存、主板、硬盘、CMOS芯片

加载主引导目录（MBR）

CPU读取位于CMOS中的BIOS程序，读取MBR用于引导Boot loader，突破512字节，加载Boot loader程序分区

<https://cloud.tencent.com/developer/article/1114481>

掌握linux启动过程与模式转换

BootLoader

主流的是GRUB GRUB会把内核加载到内存去执行

init

工作相当多 /etc/rc.d/rc.sysinit中很多设置：包括网络、外设、文件系统、硬件初始化、存储器同步等等

打印用户登录提示符

业务逻辑代码的执行 主要是底层汇编代码到业务逻辑代码的转移。登录后，整个系统启动流程运行完毕。

[linux启动过程](#)

了解Linux内核调度机制

内核 (kernel) :

内核是硬件与软件之间的一个中间层。作用是将应用层序的请求传递给硬件,并充当底层驱动程序,对系统中的各种设备和组件进行寻址。

内核负责将可用的共享资源(CPU时间、磁盘空间、网络连接等)分配得到各个系统进程。

内核实现策略:

宏内核:内核的所有代码,包括子系统都打包到一个文件中。内核的每一个函数都可访问到所有其他部分。支持模块的动态裁剪。是Linux内核的策略实现。

微内核:最基本的功能由中央内核实现。所有其他的功能委托给独立进程,进程通过明确定义的通信接口与中心内核通信。

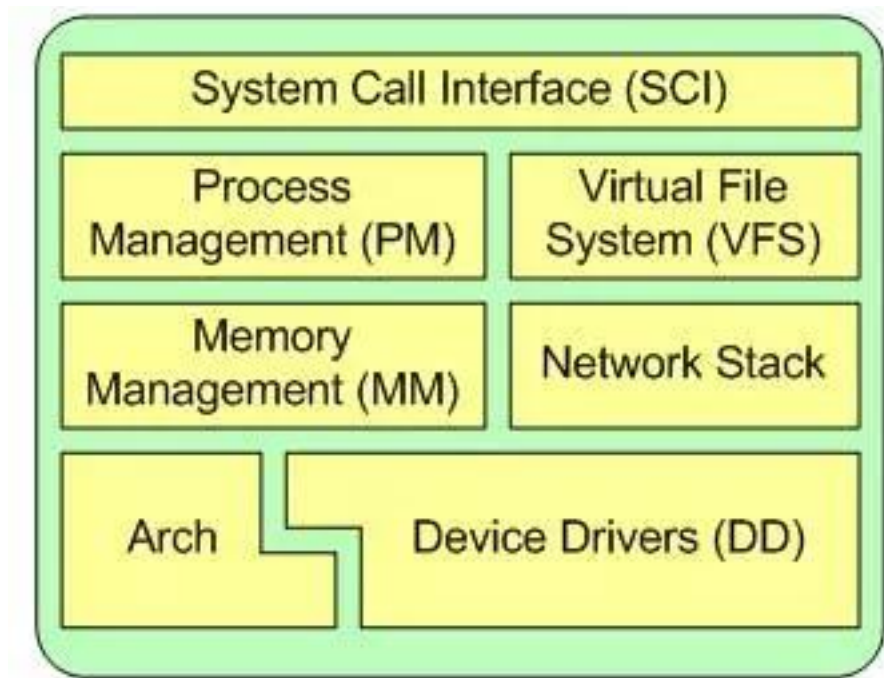
了解Linux内核调度机制

Linux内核的主要组件有：系统调用接口、进程管理、内存管理、虚拟文件系统、网络堆栈、设备驱动程序、硬件架构的相关代码。

进程管理：进程之间有共享CPU的需求。内核通过调度算法竞争CPU，固定时间进行操作。

内存管理：虚拟内存管理技术。建立虚拟地址和物理地址的映射，调用物理内存页。

设备驱动程序：能够运转特定的硬件设备，包括Bluetooth, I2C, serial等。



了解Linux内核调度机制 折中

导弹、卫星---实时性

手机、笔记本---节能

IO约束、CPU约束、前台任务、后台任务、响应时间、周转时间等等特性

最短作业优先 (SJF)

追求更少的平均时间,
最少的平均周转时间,
最少的平均平均带权周转时间

将每个进程与其下次 CPU 执行的长度关联起来。当 CPU 变为空闲时, 它会被赋给具有最短 CPU 执行的进程。如果两个进程具有同样长度的 CPU 执行, 那么可以由 FCFS 来处理。

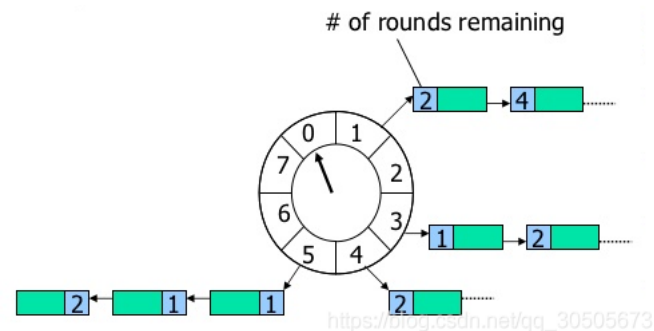
[SJF : http://c.biancheng.net/view/1244.html](http://c.biancheng.net/view/1244.html)

Linux 0.11的调度函数schedule()

```

void Schedule(void) //在kernel/sched.c中
{ while(1) { c=-1; next=0; i=NR_TASKS;
  p=&task[NR_TASKS];
  while(--i){ if((*p->state == TASK_RUNNING&&(*p)->counter>c)
    c=(*p)->counter, next=i; }
  if(c) break; //找到了最大的counter
  for(p=&LAST_TASK;p>&FIRST_TASK;--p)
    (*p)->counter=((*p)->counter>>1)
    +(*p)->priority; }
  switch_to(next);}

```



```

void do_timer(...) //在kernel/sched.c中
{ if(--current->counter>0) return;
  current->counter=0;
  schedule(); }
_timer_interrupt: //在kernel/system_call.s中
...
call _do_timer
void sched_init(void) {
  set_intr_gate(0x20, &timer_interrupt);
}

```

counter是典型的时间片，所以是轮转调度，保证了响应

```
while(--i){ if((*p->state == TASK_RUNNING&&(*p)->counter>c)
```

```
c=(*p)->counter, next=i; }
```

找counter最大的任务调度，counter表示了优先级

```
for(p=&LAST_TASK;p>&FIRST_TASK;--p)
```

```
(*p)->counter=((*p)->counter>>1)+(*p)->priority; }
```

counter代表的优先级可以动态调整

阻塞的进程再就绪以后优先级高于非阻塞进程，

counter保证了响应时间的界

每个进程只用维护一个**counter**变量，简单、高效

$$c(t) = c(t-1)/2 + p$$
$$c(0) = p$$
$$c(\infty) < 2p$$

经过IO以后，counter就会变大；IO时间越长，counter越大(为什么?)，照顾了

IO进程，变相的照顾了前台进程

后台进程一直按照counter轮转，近似了SJF调度

CPU调度：一个简单的算法折中了大多数任务的需求，这就是实际工作的schedule函数

$$c(1) = p/2 + p$$
$$c(2) = p/4 + p/2 + p$$
$$c(n) < 2p \quad \text{收敛}$$

二进制，左移右移方便计算处理。比除3更好

减法 不收敛

梳理开发时用到的操作系统的知识

Ubuntu是基于Debian，以桌面应用为主的Linux发行版。
Ubuntu有三个正式版本，包括桌面版、服务器版及用于物联网设备和机器人的Core版。

Ubuntu主要版本14.04 16.04 18.04

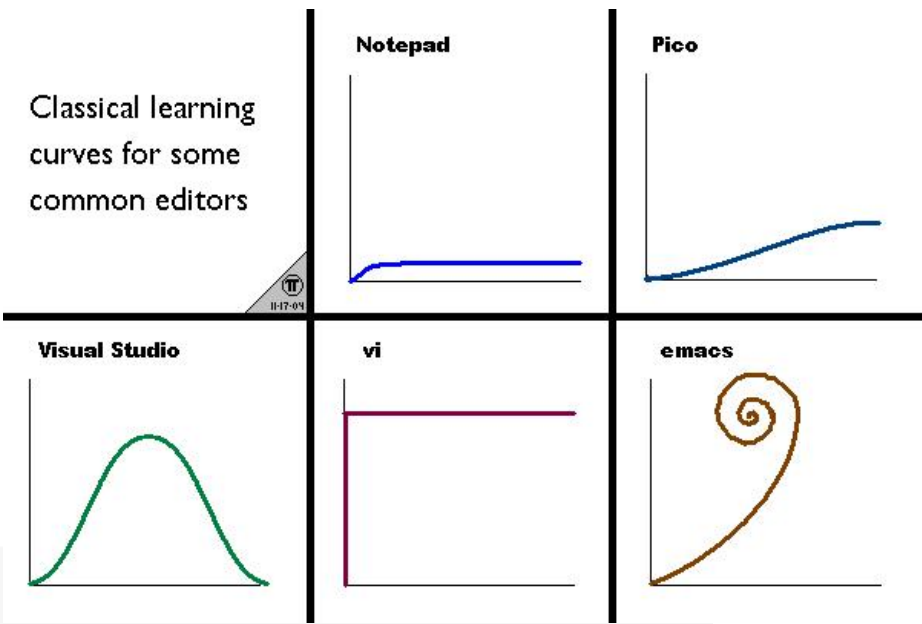
VMWare或者双系统安装

SSH VNC NoMachine

VI VIM

gedit EMACS jupyter notebook

sublime Visual Studio Code



- 1 常用的有如下插件需要安装：
- 2 C/C++。
- 3 C/C++ Snippets, 即 C/C++重用代码块。
- 4 C/C++ Advanced Lint,即 C/C++静态检测。
- 5 Code Runner, 即代码运行。
- 6 Include AutoComplete, 即自动头文件包含。
- 7 Rainbow Brackets, 彩虹花括号, 有助于阅读代码。
- 8 One Dark Pro, VSCode 的主题。
- 9 GBKtoUTF8, 将 GBK 转换为 UTF8。
- 10 ARM, 即支持 ARM 汇编语法高亮显示。
- 11 Chinese(Simplified), 即中文环境。
- 12 vscode-icons, VSCode 图标插件, 主要是资源管理器下各个文件夹的图标。
- 13 compareit, 比较插件, 可以用于比较两个文件的差异
- 14 DeviceTree, 设备树语法插件。
- 15 Markdown Preview Enhanced, markdown 预览插件。
- 16 Maridown pdf, 将.md 文件转换成其他格式。

Linux常用命令

clear ip addr date ~ pwd cd cd.. ls ls -l * mkdir rm

rm -f mv cp -r ping -c cat tail -f grep find chmod env

echo export source ps -ef

apt-get pip pip3 sudo python gcc code cmake..

catkin_make catkin_create_pkg roscpp nasm dd

介绍演示自己构建的Linux操作系统

构建过程中使用到的工具

Notepad++或Sublime：用于编辑asm文件、C文件和Makefile文件。

Linux虚拟机：利用Gcc组织编译工程。

nasm解释器：基于x86架构的汇编软件，用来编译汇编语音。

DD：用于给虚拟硬盘烧写数据。

gcc：GNU操作系统专门编写的一款编译器，用于将编程语言解释成机器语言。

Bochs：x86硬件平台模拟器 用于模拟从启动到运行的全过程。

Makefile：定义了一系列的规则来指定哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译，甚至于进行更复杂的功能操作，因为makefile就像一个Shell脚本一样，也可以执行操作系统的命令。

介绍演示自己构建的Linux操作系统

* 磁盘加载

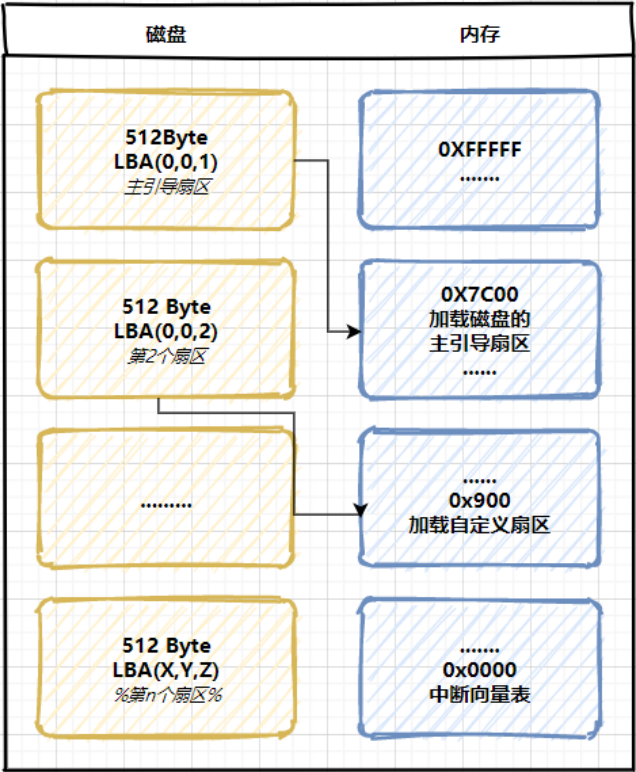
制作一个大小为512Bytes的软盘，用于存储指令文件。

* 指令文件

利用指令，设定程序的加载地址（7C00），初始化寄存器，调用子过程，然后死循环，这时，利用BIOS系统调用向屏幕输出字符串。

* 扇区加载

加载Loader程序，操作特定端口，并传入内存起始地址，利用LOOP将磁盘内容读到内存。



介绍演示自己构建的Linux操作系统

* 模拟加载kernel

通过loader文件的变化，更改读取的磁盘扇区偏移量和内存加载地址，来到达kernel起始位置。

kernel执行结束后，可认为所有与底层交互已经打通，已经顺利完成系统启动。

* 驱动与上层业务逻辑

对接底层硬件资源（asm），完成上层接口设计（c）。

nasm和gcc分别将其编译为elf文件

利用ld链接这两个二进制文件生成out文件。并指定程序的起始地址。

再将out文件转换为原始格式的二进制文件（bin）

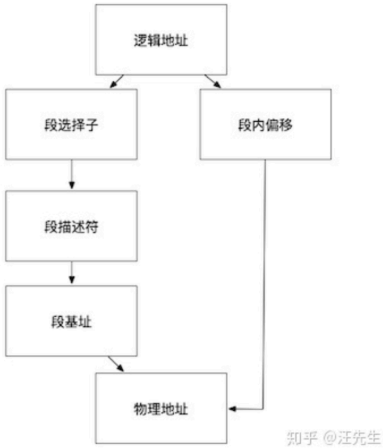
介绍演示自己构建的Linux操作系统

* 保护模式

实模式向保护模式转化。在保护模式下，每个应用程序的地址需要独立，逻辑上有4G（ 2^{32} ）的独立空间。利用内存管理单元，负责将虚拟页面映射至物理页面。

了解了上面这些术语后，现在来梳理下保护模式下的寻址方式：

- 1. 段寄存器存放段选择子；
- 2. CPU 根据段选择子从GDT中找到对应段描述符；
- 3. 从段描述符中取出段基址。
- 4. 根据之前的公式，结合段基址和段内偏移，计算出物理地址。



知乎 @汪先生

未开启分页时，保护模式寻址方式

这样实现了程序的内存隔离
可以让每个应用程序都感觉自己拥有4g内存。

介绍演示自己构建的Linux操作系统

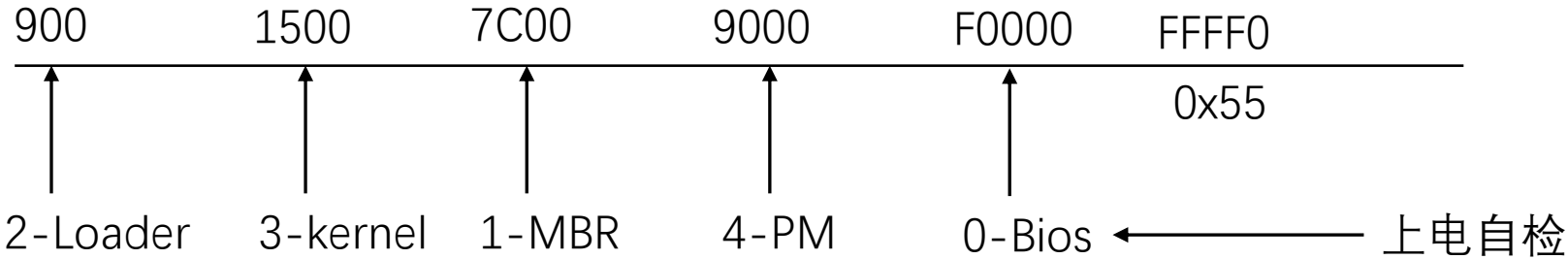
起始	结束	大小	用途
FFFF0	FFFFF	16 B	BIOS 入口地址, 此地址也属于 BIOS 代码, 这 16 字节的内容是为了执行跳转指令
F0000	FFFFF	64KB-16B	系统 BIOS 的地址范围实际上是 F000-FFFFF, 上面是入口地址, 所以单独列出
C8000	FFFFF	160KB	映射硬件适配器的 ROM 或内存映射式 I/O
C0000	C7FFF	32KB	显示适配器 BIOS
B8000	BFFFF	32KB	文本模式显示适配器
B0000	B7FFF	32KB	黑白显示适配器
A0000	AFFFF	64KB	彩色显示适配器
9FC00	9F000	1KB	EBDA(Extended BIOS Data Area) 扩展 BIOS 数据区
7E00	9FBFF	≈608KB	可区域用
7C00	7DFF	512B	MBR 被 BIOS 加载区域
500	7BFF	≈30KB	可区域用
400	4FF	256B	BIOS Data Area
000	3FF	1KB	Interrupt Vector Table 中断向量表

Bios用于上电自检后，管理各种硬件

Loader用于找到Kernel加载

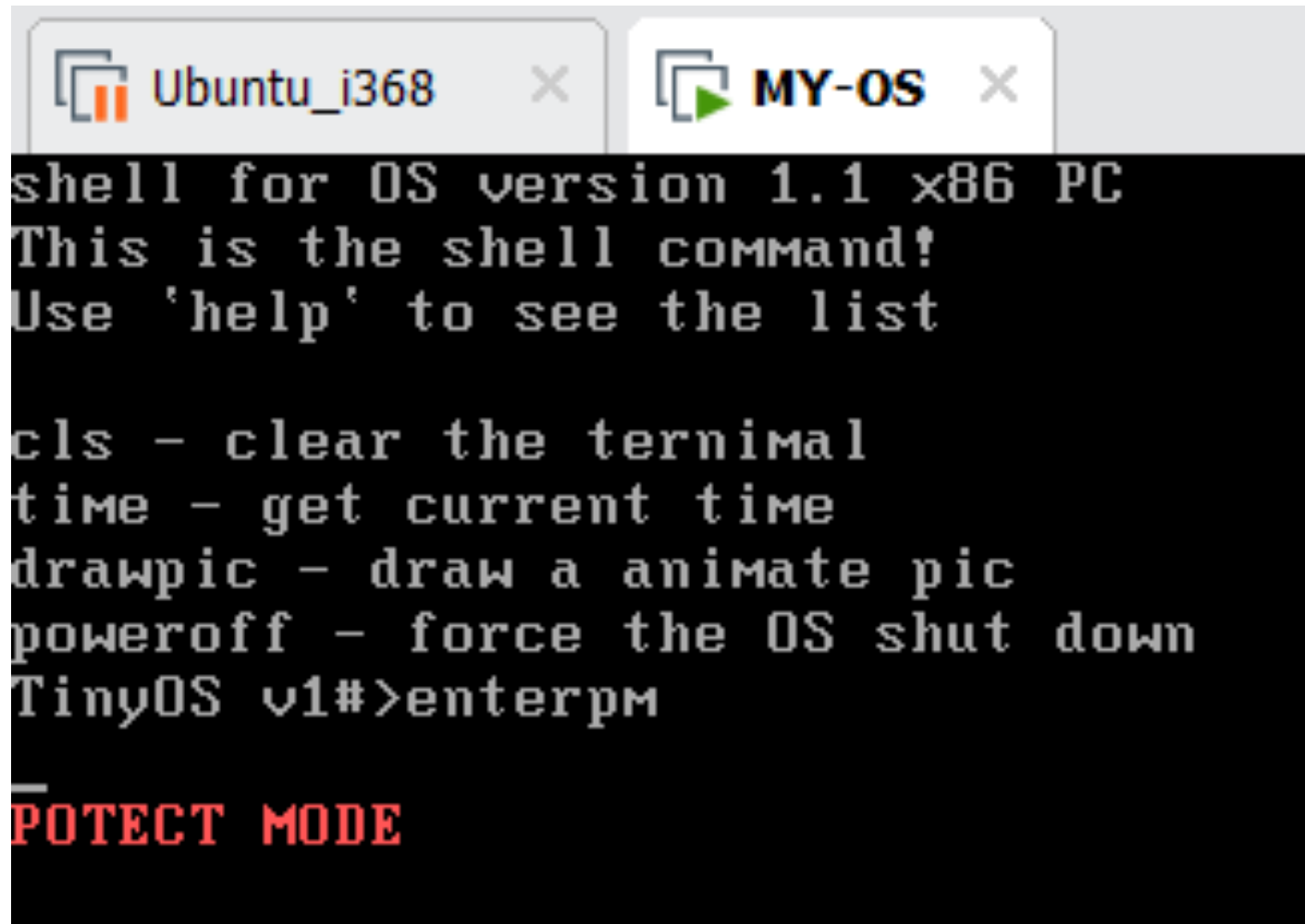
Kernel用于铺设软硬件交互

Protect Mode用于重新组织寻址方式



主引导记录

介绍演示自己构建的Linux操作系统



The screenshot shows a terminal window with two tabs: 'Ubuntu_i386' and 'MY-OS'. The 'MY-OS' tab is active, displaying a shell prompt for 'OS version 1.1 x86 PC'. The shell provides a list of commands: 'cls' (clear terminal), 'time' (get current time), 'drawpic' (draw an animated picture), and 'poweroff' (force OS shutdown). The user has entered 'enterpm', and the terminal displays 'PROTECT MODE' in red text.

```
shell for OS version 1.1 x86 PC
This is the shell command!
Use 'help' to see the list

cls - clear the terminal
time - get current time
drawpic - draw a animate pic
poweroff - force the OS shut down
TinyOS v1#>enterpm

PROTECT MODE
```

参照博文：<https://djh-sudo.github.io/Blog/#/OS/README>

延伸的操作系统介绍

μC-osII

μC/OS-II由Micrium公司提供，是一个可移植、可固化的、可裁剪的、占先式多任务实时内核，它适用于多种微处理器，微控制器和数字处理芯片（已经移植到超过100种以上的微处理器应用中）。同时，该系统源代码开放、整洁、一致，注释详尽，适合系统开发。^[1] μC/OS-II已经通过联邦航空局（FAA）商用航行器认证，符合航空无线电技术委员会（RTCA）DO-178B标准。

μC/OS-II可管理多达63个应用任务，并可以提供如下服务：

- 信号量
- 互斥信号量
- 事件标识
- 消息邮箱
- 消息队列
- 任务管理
- 固定大小内存块管理
- 时间管理

另外，在μC/OS-II内核之上，有如下独立模块可供用户选择：

- μC/FS文件系统模块
- μC/GUI图形软件模块
- μC/TCP-IP协议栈模块
- μC/USB协议栈模块

源码解析：<https://github.com/chenbokaix250/ForSomeForMySelf.git>

..	0_μCOS-II源码目录结构.md
os.h	10_优先级反转问题.md
os_core.c	11_os_mutex.c解析.md
os_dbg_r.c	12_os_sem.c解析.md
os_flag.c	13_os_mbox.c解析.md
os_mbox.c	1_os_cpu.h解析.md
os_mem.c	2_PendSV.png
os_mutex.c	2_os_cpu_a.asm解析.md
os_q.c	2_systick.png
os_sem.c	3_os_cpu_c.c解析.md
os_task.c	4_ucos_ii.h解析.md
os_time.c	5_TCB链表.png
os_tmr.c	5_TCB链表2.png
os_trace.h	5_ucos_ii中任务及TCB的理解.md
ucos_ii.c	5_任务控制关系.png
ucos_ii.h	5_控制块链接.png
	6_OSReadyGrp.png
	6_OS优先级.png
	6_ucos_ii中的任务就绪表.md
	7_ucos_ii的任务调度理解.md
	8_os_time.c解析.md
	9_os_task.c解析.md

延伸的操作系统介绍

OSEK

OSEK源于德语，英文意思是：“车载电子设备的开发系统和接口”，它是一个[标准](#)，用来产生嵌入式操作系统的规范，通讯协议栈，和汽车网络管理协议，也产生其他相关的规范。**OSEK**被设计来提供整车的各种电子控制单元的软件标准架构。

OSEK成立于1993，由一个德国的汽车公司联盟（宝马，博世，克莱斯勒，欧宝，[西门子](#)和大众）和[卡尔斯鲁厄大学](#)。1994年，法国的一个类似的项目VDX,由雷诺和标致汽车发起，也加入到这个联盟来。从此，正式名称为**OSEK/VDX**。**OSEK**是一个开放标准，由汽车工业的联盟来发布。**OSEK**的一部分被标准化为[ISO 17356](#)。

AUTOSAR与OSEK二者都是汽车电子软件的标准。

OSEK基于ECU开发，AUTOSAR基于整体汽车电子开发。

AUTOSAR中规定的操作系统就是OSEK，而通信和网络管理虽然和OSEK有区别，但思路一样的。

2.1 实时性

由于越来越多的微处理器被应用到汽车控制领域，如汽车刹车的防抱死系统、动力设备的安全控制等这些系统直接关系到人的生命安全，即使出现丝毫的差错也会导致危及生命安全的严重后果，因此要求操作系统具有严格的实时性。**OSEK**操作系统通过静态的系统配置、占先式调度策略、提供警报机制和优化系统运行机制以提高中断响应速度等手段来满足用户的实时需求。

2.2 可移植性

OSEK规范详细规定了操作系统运行的各种机制，并在这些机制基础上制定了标准的应用程序编程接口，使那些独立编写的代码能够很容易地整合起来，增强了应用程序的可移植性。**OSEK**还制定了标准的OIL，用户只需更改OIL配置文件中与硬件相关部分，便可实现不同微处理器之间的应用程序移植。通过这些手段，减少了用于维护应用程序软件和提高它的可移植性的花费，降低了应用程序的开发成本。

2.3 可扩展性

为了适用于广泛的目标处理器，支持运行在广泛硬件基础上的实时程序，**OSEK**操作系统具备高度模块化和可灵活配置的特性。它定义了不同的符合级别(Conformance Classes)，并采用对不同应用程序有可靠接收能力的体系结构，从而增强了系统的可扩展性。**OSEK**操作系统可以在很少的硬件资源(RAM,ROM,CPC时间)环境下运行，即便在8位微处理器上也是如此。

延伸的操作系统介绍

OSEK

主频的心跳，
驱动几个定时器的脉搏，
产生几个软硬件计数器的滴答。
一个计数器拉响了几个警报，
每个周期的轮回都激活一个任务和事件。
一个计数器启动了一个调度表，
每个超时的节点都激活若干任务和事件。
我是一个任务，
因为我优先级更高，所以抢占。
因为我们优先级平等，所以排队。
我若优先级最高，谁也别抢，我是山顶。
我若在访问资源，最好别抢，除非高过我的天花板。
我不想被别人抢，我要锁住调度器。
但我还是被抢，因为遇到了中断。
我是基础任务，
别人叫醒了我，我干完事情自己睡去。
除非有人要做更重要的事情，我只好让出工作台，
坐在旁边（Ready）。
我是扩展任务，
我在干活时需要等一个事件，我只好让出工作台，
假装睡觉（Wait）。
我是一个事件，
我有我的主人扩展任务。
请不要设置一个不属于你的事件，
这注定是不被察觉的失败。
请不要等待一个不属于你的事件，
那样只是白等止步不前。

一个高优先级任务等待事件，让出CPU给我的同伴
一个低优先级任务设置事件，再让CPU给我的主人
让我们一起同步。
我是一个资源，
一个任务开始访问我，把它的权限上升到天花板，
这个任务结束访问我，把它的权限恢复原状，
让优先级避免反转，
让任务等待避免死锁。
我是一个自旋锁，
我为不同核上共享资源而用，
在一个核上访问资源前先上锁，
那样另一个核上在访问资源时就会自旋等待。
相同的核上不要使用，那样起不到作用。
我是一个栈，
每个抢占的任务都要先来这里，
堆一捆柴火，
后来者居上，居上者优先，优先者早退。
请不要堆得太高，
那样有溢出风险。
我是一类中断，我从哪里来又回那里去。
我是二类中断，我走之后要求重新调度。
我是一个调度器，
在需要的时候，
我会更新就绪表，
取出优先级更高的任务，
切换运行。

延伸的操作系统介绍

ROS

ROS（机器人操作系统，Robot Operating System），是专为[机器人](#)软件开发所设计出来的一套电脑[操作系统](#)架构。它是一个开源的元级操作系统（后操作系统），提供类似于操作系统的服务，包括硬件抽象描述、底层驱动程序管理、共用功能的执行、程序间消息传递、程序发行包管理，它也提供一些工具和库用于获取、建立、编写和执行多机融合的程序。

ROS的运行架构是一种使用ROS通信模块实现模块间[P2P](#)的松耦合的网络连接的处理架构，它执行若干种类型的通讯，包括：

基于服务的同步[RPC](#)（远程过程调用）通讯；

基于Topic的异步数据流通讯，还有参数服务器上的数据存储。

[ROS WIKI](#)

“轻便”：ROS是设计得尽可能方便简易。不必替换主框架与系统，因为ROS编写的代码可以用于其他机器人软件框架中。毫无疑问的，ROS更易于集成与其他机器人软件框架。事实上ROS已完成与OpenRAVE、Orocos和Player的整合。

ROS-agnostic库：【agnostic：不可知论】建议的开发模型是使用clear的函数接口书写ROS-agnostic库。

语言独立性：ROS框架很容易在任何编程语言中执行。我们已经能在Python和C++中顺利运行，同时添加有[Lisp](#)、[Octave](#)和Java语言库。

测试简单：ROS有一个内建的单元/组合集测试框架，称为“roctest”。这使得集成调试和分解调试很容易。

扩展性：ROS适合于大型实时系统与大型的系统开发项目。

ROS2

ADLINK

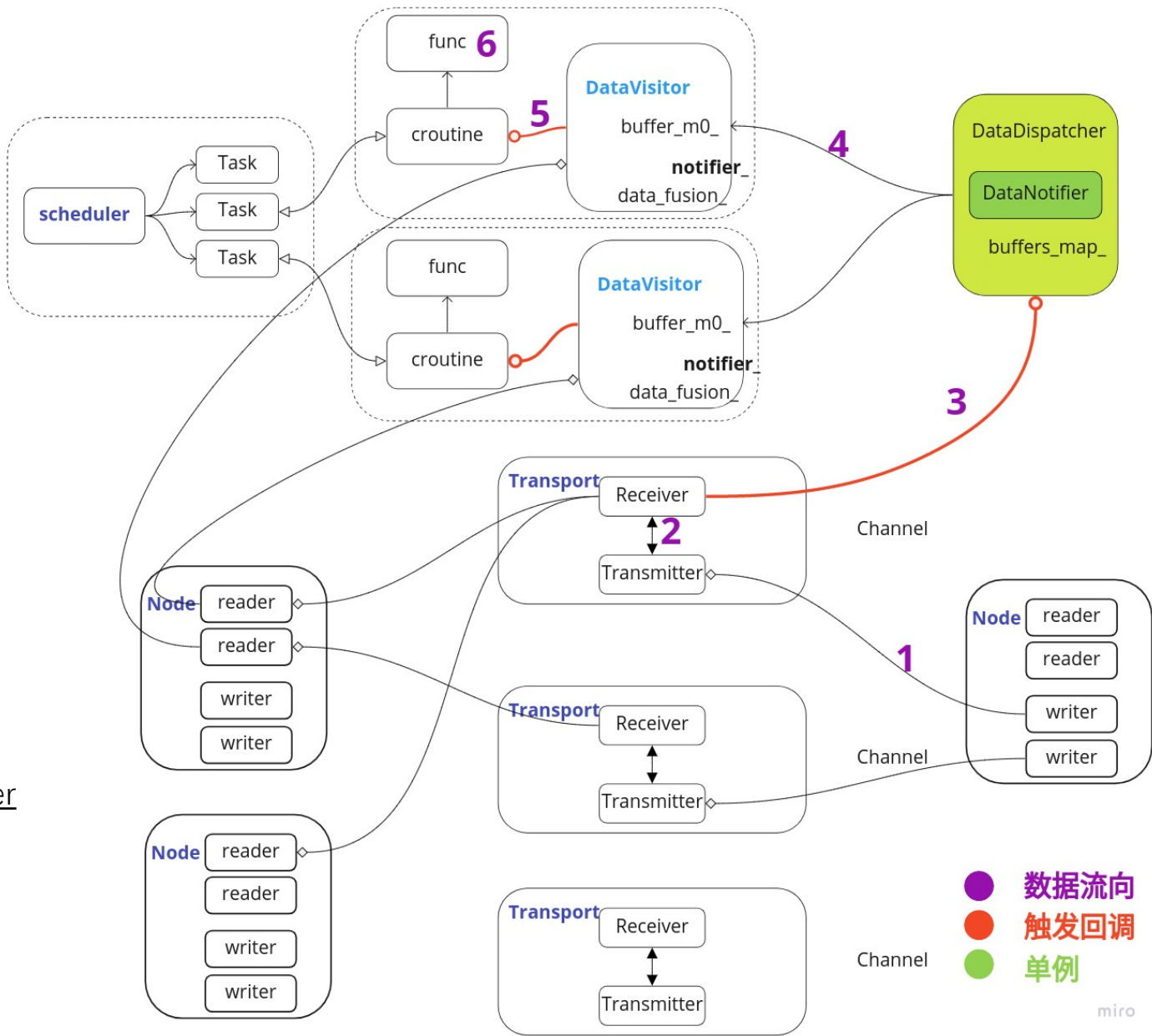


延伸的操作系统介绍

Cyber

Cyber是百度Apollo推出的替代Ros的消息中间件，自动驾驶中的各个模块通过Cyber进行消息订阅和发布，同时Cyber还提供了任务调度、录制Bag包等功能。通过Cyber实现了自动驾驶中间层，这里分为几小结分别对Cyber进行详细的分析。

https://github.com/chenbokaix250/Apollo_Analyze/tree/main/cyber



延伸的操作系统介绍

协程时代

从宏观上看，线程任务和协程任务都是并行的。从微观上看，线程任务是分时切片轮流执行的，这种切换是系统自动完成的，无需程序员干预；而协程则是根据任务特点，在任务阻塞时将控制权交给其他协程，这个权力交接的时机和位置，由程序员指定。由此可以看出，参与协程管理的每一个任务，必须存在阻塞的可能，且阻塞条件会被其它任务破坏，从而得以在阻塞解除后继续执行。

进程和线程和协程的区别

阻塞式读和写。当队列为空时，读会被阻塞，直到读出数据；当队列满时，写会被阻塞，直到队列空出位置后写入成功。因为队列具有阻塞式读写的特点，正好可以在协程中利用阻塞切换其他协程任务。



某个富豪（rich）手拿一沓钞票，随机取出几张，撒在地上（如果地上已经有钞票的话，就等有人捡走了再撒）；另有名为A、B、C的三个幸运儿（lucky），紧盯着撒钱的富豪，只要富豪把钱撒到地上，他们立刻就去捡起来。

延伸的操作系统介绍

协程时代

如果用协程实现上述功能的话，我们可以用长度为1的协程队列来存放富豪每一次抛撒的钱。一旦队列中有钱（队列满），富豪就不能继续抛撒了，抛撒被阻塞，协程控制权转移。三个幸运儿中的某一个获得控制权，就去读队列（捡钱），如果队列中没有钱（队列空），捡钱被阻塞，协程控制权转移。依靠队列的阻塞和解除阻塞，一个富豪和三个幸运儿可以顺利地分配完富豪手中的钞票。为了让这个过程可以慢到适合观察，可以在富豪抛钱之前，再增加一个随机延时。当然，这个延时不能使用time模块的sleep()函数，而是使用协程模块asyncio的sleep()函数。下面是完整的撒钱-捡钱代码。

```
import asyncio, random

async def rich(q, total):
    """任性的富豪，随机撒钱"""

    while total > 0:
        money = random.randint(10,100)
        total -= money
        await q.put(money) # 随机生成[10,100]之间的整数
        print('富豪潇洒地抛了%d块钱'%money)
        await asyncio.sleep(3*random.random()) # 在0-3秒之间随机延时

async def lucky(q, name):
    """随时可以捡到钱的幸运儿"""

    while True:
        money = await q.get()
        q.task_done()
        print('%s捡到了%d块钱!'%(name, money))

async def run():
    q = asyncio.Queue(1)

    producers = [asyncio.create_task(rich(q, 300))]
    consumers = [asyncio.create_task(lucky(q, name)) for name in 'ABC']

    await asyncio.gather(*producers,)
    await q.join()

    for c in consumers:
        c.cancel()

if __name__ == '__main__':
    asyncio.run(run())
```

延伸的操作系统介绍

富豪

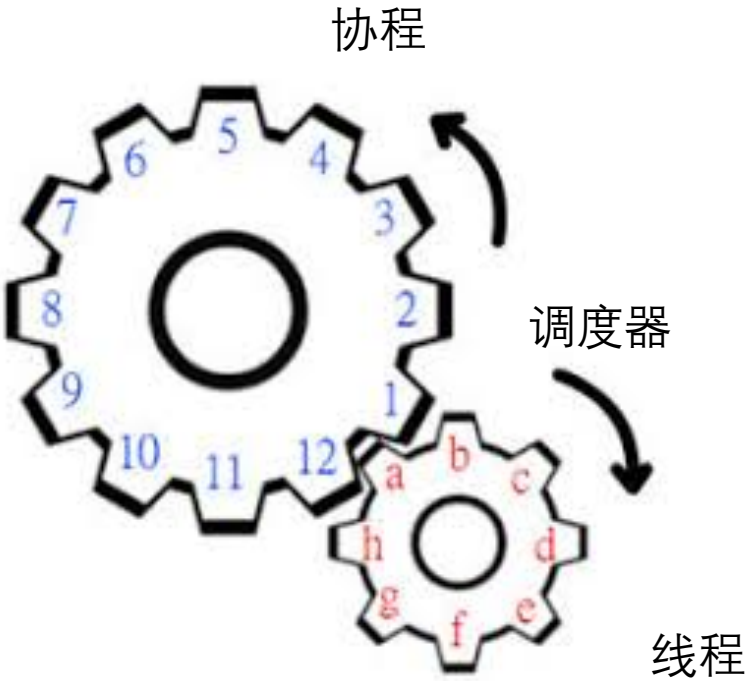
撒钱

穷人

捡钱

[深入浅出协程](#)

[GPM模型](#)



操作系统未来的发展

未来的操作系统一定是基于新场景的

像历史一样，从Windows到ios、android，未来的操作系统会像历史一样迁移，从PC到智能手机，再到IoT。

结论1：基于新的应用场景。

未来的操作系统是可以弹性伸缩的

目前装载在我们电脑和手机中的操作系统，多少功能是每个人都完整用过的呢？实际上操作系统中大部分功能并没有被调用。

结论2：根据产品思维，需求是以用户和客户的需求为主，未来操作系统一定是按需装载。

未来的操作系统将彻底分离存储、传输和计算

目前的操作系统负责数据传输、数据存储和数据计算为主，未来的操作系统这些功能将进一步彻底分离。

结论3：分离思想的实现。

未来的操作系统与SaaS产品、与B端产品、云生态产品的功能融合

云产品、B端产品、智能汽车产品将催生一个新的未来操作系统。

结论4：未来操作系统将会平台一体化。

感谢聆听

2022.2.22 陈博凯