OS-1

ISA（Instruction Set Architecture）指令集架构，区分硬件和软件

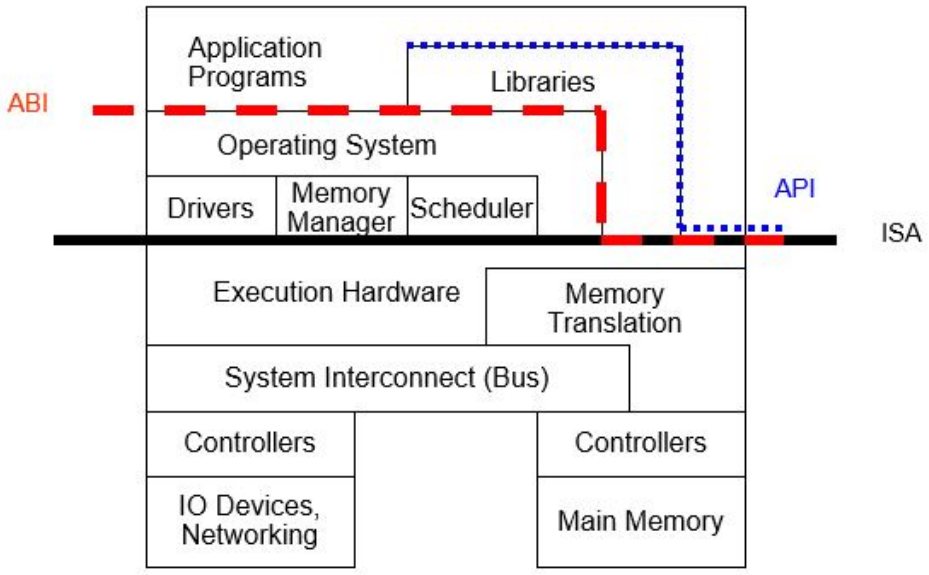
？？？ User ISA与System ISA的区别：系统ISA中有一些特权指令

ABI（Application Binary Layer）应用程序二进制接口，跟系统有关，定义了在相应系统上编译程序所需的规则（基本数据类型，通用寄存器的使用，参数的传递规则，以及堆栈的使用），想win10,win7,ubuntu16.04都有自己不同的ABI

让程序能接触到硬件跟系统服务，包含User ISA跟System Call

API（Application Programming Interface）程序调用库函数，read(),write()

– Level of Abstraction = Implementation layer
  • ISA, ABI, API



8 important problems

scale up(可扩展性)

Security & Privacy（安全和隐私)

Power Efficiency（用电的效率)

Mobility（流动性)

Write Correct Parallel Code(写正确的并行代码)

Scale Out(use distributed systems)分布式系统

Non-Volatile Storage(稳定的存储)

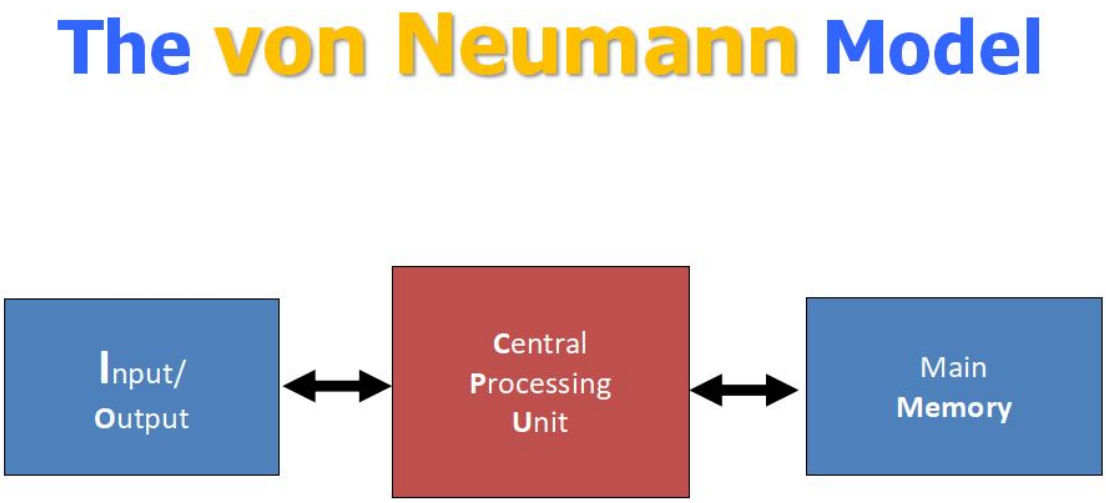
Virtualization(虚拟化)

OS-2

随机存取存储器（英语：**R**andom **A**ccess **M**emory，缩写：**RAM**），也叫**主存**，是与CPU直接交换数据的内部存储器。[1]它可以随时读写（刷新时除外，见下文），而且速度很快，通常作为操作系统或其他正在运行中的程序的临时数据存储媒介。

主存（Main memory）即计算机内部最主要的存储器，用来加载各式各样的程序与数据以供CPU直接运行与运用。由于DRAM的性价比很高，且扩展性也不错，是现今一般计算机主存的最主要部分。

PC架构：von Neumann Model（冯 诺伊曼模式）



- **I/O**: communicating data to and from devices
- **CPU**: digital logic for performing computation
- **Memory**: N words of B bits

程序运行时，程序的指令和数据会被读到主存里面，CPU会一条一条的执行主存里面的指令。

x86运行模式：

1. 实模式：

实地址模式（real-addess mode）——该模式以扩展的方式实现了8086CPU的程序运行环境（就像切换到保护模式和系统管理模式一样）。处理器在刚刚上电或者重启后的时候，处于实地址模式。

1. 保护模式：

保护模式（protected mode）——是处理器的根本模式，在保护模式下，可以为直接运行的实地址模式程序提供保护的、多任务的环境，这种特性被称作“虚拟8086模式（virtual 8086 mode）”,尽管“虚拟8086”模式并不是一种真正的处理器模式；virtual 8086 mode实际上是保护模式的一种属性，在保护模式下，可以向任何任务提供这种属性。

1. SMM：

系统管理模式（system manangement mode，SMM）——该模式提供操作系统或者执行程序一种透明的机制去实现平台相关的特性，例如电源管理和系统安全。当来自外部的或者APIC控制器的SMM中断pin脚被触发时，处理器在下列情况进入SMM。在SMM下，处理器切换到一个独立的地址空间，同时保存当前运行的程序或任务的上下文。SMM相关的代码可透明的执行。当SMM模式返回时，处理器返回SMI（system manangement interrupt）前的工作模式。SMM模式在Intel 386 SL和Intel 486 SL处理器时被引入，在Pentium家族时成为标准的IA-32架构的特性。

实模式保护模式SMM必读

x86控制寄存器：

1. CR0:

Bit	Name	Full Name	Description
0	PE	Protected Mode Enable	If 1, system is in protected mode, else system is in real mode
1	MP	Monitor co-processor	Controls interaction of WAIT/FWAIT instructions with TS flag in CR0
2	EM	Emulation	If set, no x87 floating point unit present, if clear, x87 FPU present
3	TS	Task switched	Allows saving x87 task context upon a task switch only after x87 instruction used
4	ET	Extension type	On the 386, it allowed to specify whether the external math coprocessor was an 80287 or 80387
5	NE	Numeric error	Enable internal x87 floating point error reporting when set, else enables PC style x87 error detection
16	WP	Write protect	When set, the CPU can't write to read-only pages when privilege level is 0
18	AM	Alignment mask	Alignment check enabled if AM set, AC flag (in EFLAGS register) set, and privilege level is 3
29	NW	Not-write through	Globally enables/disable write-through caching
30	CD	Cache disable	Globally enables/disable the memory cache
31	PG	Paging	If 1, enable paging and use the CR3 register, else disable paging

2. CR1:预留给之后用的，现在调用会抛exception

3. CR2:用于发生Page Fault报告出错信息。当发生页异常时，处理器把引起页异常的线性地址保存在CR2中。操作系统中的页异常处理程序可以检查CR2的内容，从而查出线性地址空间中的哪一页引起本次异常。
4. CR3:用于保存页目录表页面的物理地址，因此被称为PDBR。由于目录是页对齐的，所以仅高20位有效，低12 位保留供更加高级的处理器使用。向CR3中装入一个新值时，低12位必须为0；但从 CR3中取值时，低12位被忽略。每当用MOV指令重置CR3的值时，会导致分页机制高速缓冲区的内容无效，用此方法，可以在启用分页机制之前，即把PG 位置1之前，预先刷新分页机制的高速缓存。CR3寄存器即使在CR0寄存器的PG位或PE位为0时也可装入，如在实模式下也可设置CR3，以便进行分页机制的初始化。在任务切换时，CR3要被改变，但是如果新任务中CR3的值与原任务中CR3的值相同，那么处理器不刷新分页高速缓存，以便当任务共享页表时有较快的执行速度。
5. CR4：Used in protected mode to control operations such as virtual-8086 support, enabling I/O breakpoints, page size extension and machine check exceptions.

Bit	Name	Full Name	Description
0	VME	Virtual 8086 Mode Extensions	If set, enables support for the virtual interrupt flag (VIF) in virtual-8086 mode.
1	PVI	Protected-mode Virtual Interrupts	If set, enables support for the virtual interrupt flag (VIF) in protected mode.
2	TSD	Time Stamp Disable	If set, RDTSC instruction can only be executed when in ring 0, otherwise RDTSC can be used at any privilege level.
3	DE	Debugging Extensions	If set, enables debug register based breaks on I/O space access
4	PSE	Page Size Extension	If unset, page size is 4 KiB, else page size is increased to 4 MiB (if PAE is enabled or the processor is in Long Mode this bit is ignored[2]).
5	PAE	Physical Address Extension	If set, changes page table layout to translate 32-bit virtual addresses into extended 36-bit physical addresses.
6	MCE	Machine Check Exception	If set, enables machine check interrupts to occur.
7	PGE	Page Global Enabled	If set, address translations (PDE or PTE records) may be shared between address spaces.
8	PCE	Performance-Monitoring Counter enable	If set, RDPMC can be executed at any privilege level, else RDPMC can only be used in ring 0.
9	OSFXSR	Operating system support for FXSAVE and FXRSTOR instructions	If set, enables SSE instructions and fast FPU save & restore
10	OSXMMEXCPT	Operating System Support for Unmasked SIMD Floating-Point Exceptions	If set, enables unmasked SSE exceptions.
11	UMIP	User-Mode Instruction Prevention	If set, the SGDT, SIDT, SLDT, SMSW and STR instructions cannot be executed if CPL > 0[1].
12	LA57	(none specified)	If set, enables 5-Level Paging[3].

Bit	Name	Full Name	Description
13	VMXE	Virtual Machine Extensions Enable	see Intel VT-x
14	SMXE	Safer Mode Extensions Enable	see Trusted Execution Technology (TXT)
16	FSGSBASE	Enables the instructions RDFSBASE, RDGSBASE, WRFSBASE, and WRGSBASE.	
17	PCIDE	PCID Enable	If set, enables process-context identifiers (PCIDs).
18	OSXSAVE	XSAVE and Processor Extended States Enable	
20	SMEP[4]	Supervisor Mode Execution Protection Enable	If set, execution of code in a higher ring generates a fault
21	SMAP	Supervisor Mode Access Prevention Enable	If set, access of data in a higher ring generates a fault[5]
22	PKE	Protection Key Enable	See Intel® 64 and IA-32 Architectures Software Developer's Manual

Memory-management Register(内存管理寄存器)

1. GDTR（Global Descriptor Table Register）存放GDT的入口地址，**物理地址**
2. IDTR (Interrupt Descriptor Table Register)
3. TR(Task Register)

内存管理笔记(分页，分段，逻辑地址，物理地址与地址转换方式)

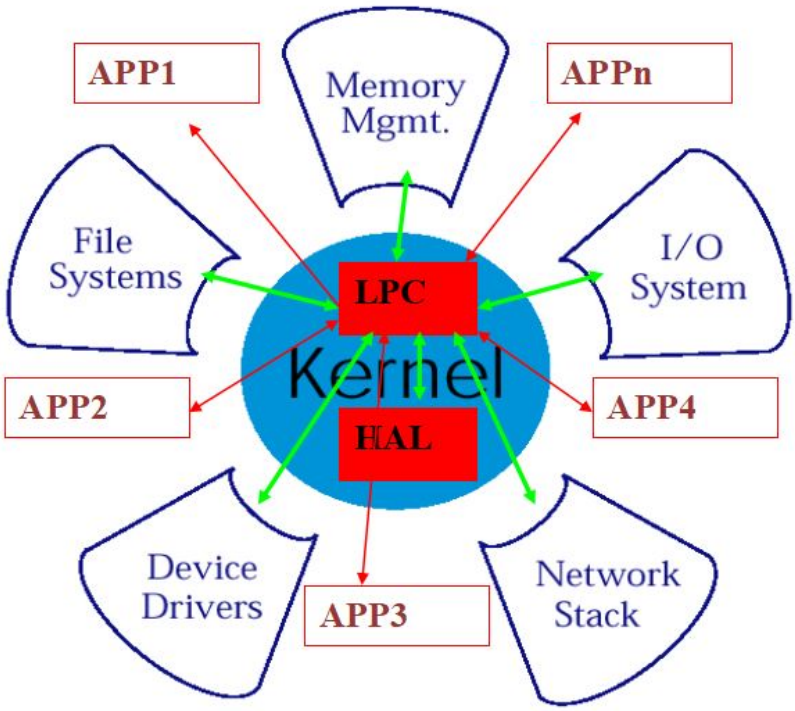
OS-3

系统设计要注意模块化，设计与实现分离，这样设计需要修改时，实现可以不用大改。

MicroKernel

1. 最小的分配单元：进程
2. 最小的执行单元：线程
3. 应用可以实现自己的CPU调度
4. IPC与线程间通信？？？

5. 对Syscall的处理：trap 到用户模式
6. 微内核把所有原来内核里面的系统服务都用进程来实现，要调用系统服务的时候，使用IPC



7. Redirection allows call traps to link directly to executable binaries without modifying the kernel!Just need an emulation library

L4 Microkernel

1. 之前的IPC的copy是先copy到kernel，然后kernel再copy到系统服务进程，在L4里面是直接copy到系统服务进程，不经过kernel

   send把数据copy到内核控制的一块区域，然后将这块区域映射到receiver的地址空间

2. 异步的IPC

Exokernel

Exokernel微内核的核心观点是：只要内核还提供对系统资源的抽象，就不能实现性能的最大优化 -- 内核应该支持一个最小的、高度优化的原语集，而不是提供对系统资源的抽象。从这个观点上来说，IPC也是一个太高级的抽象因而不能达到最高的性能。Exokernel微内核的核心是支持一个高度优化的原语名叫保护控制转移(protected control transfer, PCT)。PCT是一个不带参数的跨地址空间的过程调用，其功能类似于一个硬件中断。在PCT的基础上，可以实现高级的IPC抽象如RPC。在MIPS R3000处理器上，一个基于PCT的RPC实现了仅10μs的开销，而在同一硬件上运行的Mach RPC为95μs。

内核分配资源，应用自己来实现内存管理，文件系统等功能。

exokernel 为不同的应用分配硬件资源，并确保不会把一个资源同时分配给多个应用，达 到应用间隔离的效果。to separate protection from management： Exokernel 只负责资源的分配、回收等，而资源如 何使用由应用自己负责。

Exokernel are much smaller than a normal kernel (monolithic kernel). They give more direct access to the hardware, thus removing most abstractions

外内核的三种保护措施：

1. Secure Binding:
   应用需要什么资源，外内核给应用分配资源，将此资源与应用绑定，意思是不再将这个资源分配给其他应用
2. VIsible resource revocaton
   当应用使用完资源，内核会先通知应用再回收资源。传统是直接回收，不通知。
3. Abort
   当应用不归还资源时，内核强行收回。

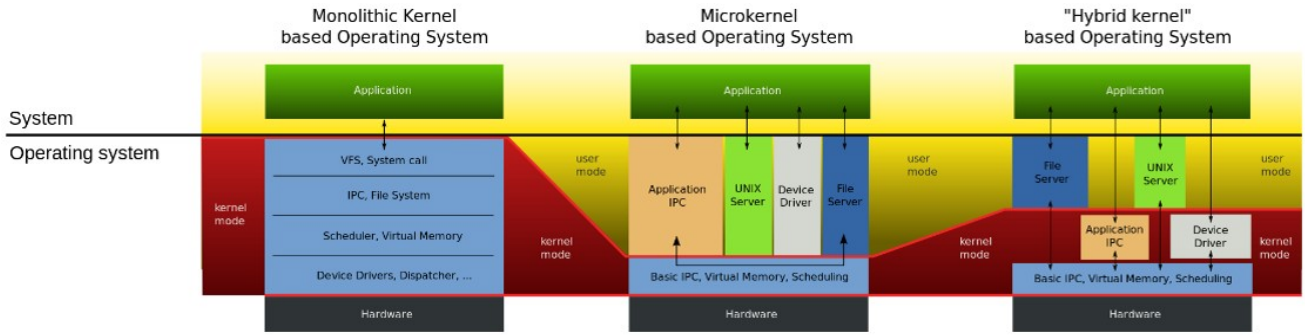
Library OS:

外内核不管操作系统是什么，他只分配资源，把操作系统抽象成链接库，库可以有多个，给进程提供抽象。外内核不提供关于文件系统，虚拟内存，IO的抽象，这些都由Library OS来提供。

Hybrid kernel

is a kernel architecturebased on combining aspects of microkernel and monolithickernel architectures used in computer operating systems.

就是单片内核+微内核



OS-4

PAE

Address translation: PAE-4k

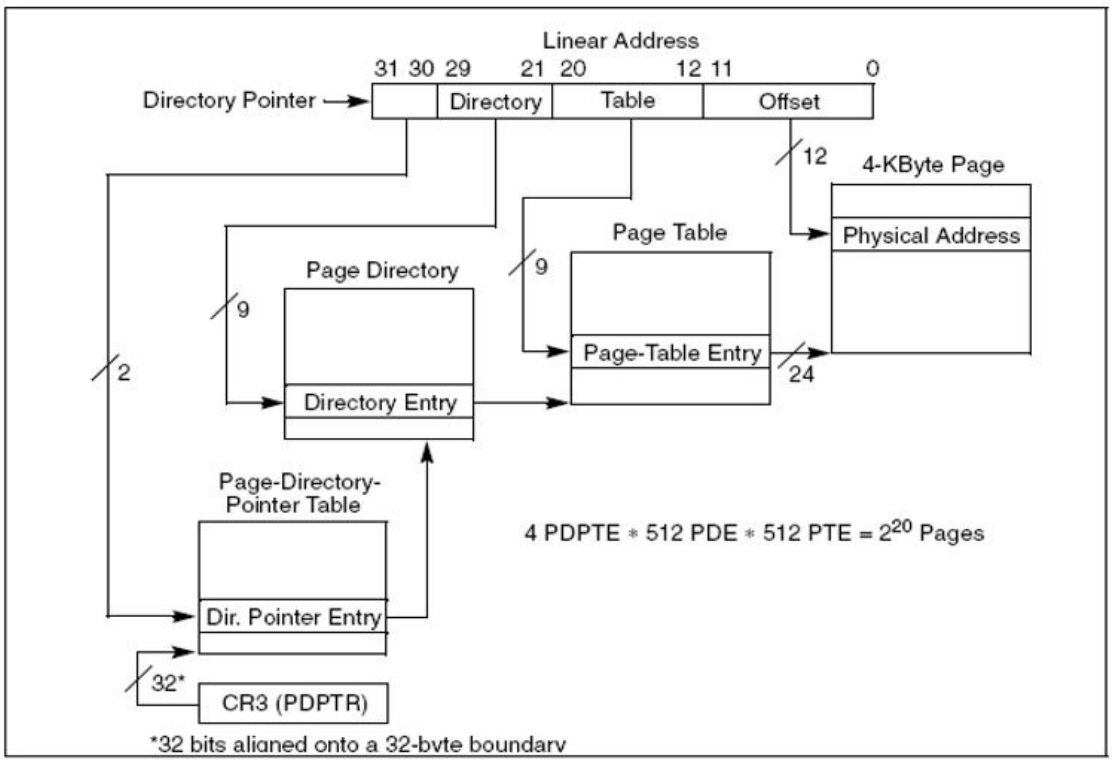


Figure 3-18. Linear Address Translation With PAE Enabled (4-KByte Pages)

OS-5

线程

线程跟地址空间是独立的，线程可以有自己独立的地址空间，也可以没有。

进程

进程包括程序计数器，栈，数据段

PCB:

- **The process scheduling state:** The state of the process in terms of "ready", "suspended", etc., and other scheduling information as well, like priority value, the amount of time elapsed since the process gained control of the CPU or since it was suspended. Also, in case of a suspended process, event identification data must be recorded for the event the process is waiting for.
- **Process structuring information:** process's children id's, or the id's of other processes related to the current one in some functional way, which may be represented as a queue, a ring or other data structures.
- **Interprocess communication information:** various flags, signals and messages associated with the communication among independent processes may be stored in the PCB.
- **Process Privileges** in terms of allowed/disallowed access to system resources.
- **Process State:** State may enter into new, ready, running, waiting, dead depending on CPU scheduling.
- **Process Number (PID):** A unique identification number for each process in the operating system (also known as Process ID).

- **Program Counter (PC):** A pointer to the address of the next instruction to be executed for this process.
- **CPU Registers:** Indicates various register set of CPU where process need to be stored for execution for running state.
- **CPU Scheduling Information:** indicates the information of a process with which it uses the CPU time through scheduling.
- **Memory Management Information:** includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
- **Accounting Information:** Includes the amount of CPU used for process execution, time limits, execution ID etc.
- **I/O Status Information:** Includes a list of I/O devices allocated to the process.

内核切换进程A到进程B，context switch：

1. 存储进程A的PCB到进程A的地址空间
2. 从B的地址空间里面读取B的PCB并加载

进程A从用户态到内核态执行，然后返回：

1. 内核存储进程A的PCB到进程A的栈上（就是进程A的地址空间），返回时再读取
2. 将返回地址压入内核栈，压入内核栈的有
    1. ss(栈段),cs(代码段)的基地址，用来将逻辑地址翻译成线性地址
    2. esp（栈顶指针），eip（程序计数器，指向下一条将被执行的指令），eflags(存了一些条件码（CF，ZF,SF,OF）ics.p124)

父进程与子进程：

1. 子进程复制了父进程的用户地址空间（text,data,bss,堆，栈五个段）
2. 子进程复制了父进程的打开文件表

IPC

1. shared memory:bounded buffer
2. message passing:send/receive

POXIS

1. POSIX表示可移植操作系统接口（Portable Operating System Interface of UNIX，缩写为 POSIX），POSIX标准定义了操作系统应该为应用程序提供的接口标准，是IEEE为要在各种UNIX操作系统上运行的软件而定义的一系列API标准的总称，其正式称呼为IEEE 1003，而国际标准名称为ISO/IEC 9945。

   POSIX标准意在期望获得源代码级别的软件可移植性。换句话说，为一个POSIX兼容的操作系统编写的程序，应该可以在任何其它的POSIX操作系统（即使是来自另一个厂商）上编译执行。

   POSIX 并不局限于 UNIX。许多其它的操作系统，例如 DEC OpenVMS 支持 POSIX 标准，尤其是 IEEE Std. 1003.1-1990（1995 年修订）或 POSIX.1，POSIX.1 提供了源代码级别的 C 语言应用编程接口（API）给操作系统的服务程序，例如读写文件。POSIX.1 已经被国际标准化组织（International Standards Organization，ISO）所接受，被命名为 ISO/IEC 9945-1:1990 标准。

OS-6

详细介绍了IPC

实现IPC的几种机制：

Method	Short Description	Provided by (operating systems or other environments)
File	A record stored on disk, or a record synthesized on demand by a file server, which can be accessed by multiple processes.	Most operating systems
Signal; also Asynchronous System Trap	A system message sent from one process to another, not usually used to transfer data but instead used to remotely command the partnered process.	Most operating systems
Socket	Data sent over a network interface, either to a different process on the same computer or to another computer on the network. Stream-oriented (TCP; data written through a socket requires formatting to preserve message boundaries) or more rarely message-oriented (UDP, SCTP).	Most operating systems
Unix domain socket	Similar to an internet socket but all communication occurs within the kernel. Domain sockets use the file system as their address space. Processes reference a domain socket as an inode, and multiple processes can communicate with one socket	All POSIX operating systems and Windows 10[2]
Message queue	A data stream similar to a socket, but which usually preserves message boundaries. Typically implemented by the operating system, they allow multiple processes to read and write to the message queue without being directly connected to each other.	Most operating systems
Pipe	A unidirectional data channel. Data written to the write end of the pipe is buffered by the operating system until it is read from the read end of the pipe. Two-way data streams between processes can be achieved by creating two pipes utilizing standard input and output.	All POSIX systems, Windows
Named pipe	A pipe implemented through a file on the file system instead of standard input and output. Multiple processes can read and write to the file as a buffer for IPC data.	All POSIX systems, Windows, AmigaOS 2.0+
Shared memory	Multiple processes are given access to the same block of memory which creates a shared buffer for the processes to communicate with each other.	All POSIX systems, Windows

Method	Short Description	Provided by (operating systems or other environments)
Message passing	Allows multiple programs to communicate using message queues and/or non-OS managed channels, commonly used in concurrency models.	Used in RPC, RMI, and MPI paradigms, Java RMI, CORBA, DDS, MSMQ, MailSlots, QNX, others
Memory-mapped file	A file mapped to RAM and can be modified by changing memory addresses directly instead of outputting to a stream. This shares the same benefits as a standard file.	All POSIX systems, Windows

LRPC

OS-7

这一张讲Exception，exception分为interrupt,trap,fault,abort

中断是异步的，因为你不知道什么时候会发生，其实fault也是，但是fault是同步的。

Nested Interrupt: 在处理中断的时候又发生了一次中断，不能超过三次（例子：处理page fault中断，发现page fault中断的代码也缺页，再次触发page fault，处理这个page fault会abort）

Triple Fault

- Things never to do in an OS #1: Swap out the page swapping code (triple-fault here we come) —Kemp

- When a fault occurs, the CPU invokes an exception handler.
- If a fault occurs while trying to invoke the exception handler, that's called a double fault, which the CPU tries to handle with yet another exception handler.
- If that invocation results in a fault too, the system reboots with a triple fault.

Bottom Half

当有多个IRQ要处理时，最多只能pengding一个IRQ，其他的IRQ就会被丢弃，所以使用Bottom Half，立即返回，意思是处理中断时间很短，中断本身延迟处理。

中断处理程序在接收到中断之后，把记录状态的寄存器压栈，找到对应的中断处理程序，创建一个微进程处理，返回。微进程并不会立即执行，而是等下一次Sysycall时会检查微进程有没有处理结束，没有的话就处理。

Bottom Half的四种处理机制（softirqs，tasklets，工作队列，内核线程）。

Comparing Approaches

	ISR	SoftIRQ	Tasklet	WorkQueue	KThread
Will disable all interrupts?	Briefly	No	No	No	No
Will disable other instances of self?	Yes	Yes	No	No	No
Higher priority than regular scheduled tasks?	Yes	Yes*	Yes*	No	No
Will be run on same processor as ISR?	N/A	Yes	Yes	Yes	Maybe
More than one run can on same CPU?	No	No	No	Yes	Yes
Same one can run on multiple CPUs?	Yes	Yes	No	Yes	Yes
Full context switch?	No	No	No	Yes	Yes
Can sleep? (Has own kernel stack)	No	No	No	Yes	Yes
Can access user space?	No	No	No	No	No

*Within limits, can be run by ksoftirqd

OS-8

DPL(descripter privilege level) 中断描述符所需要的特权级

CPL(current privilege level) 当前特权级

FLEX-SC

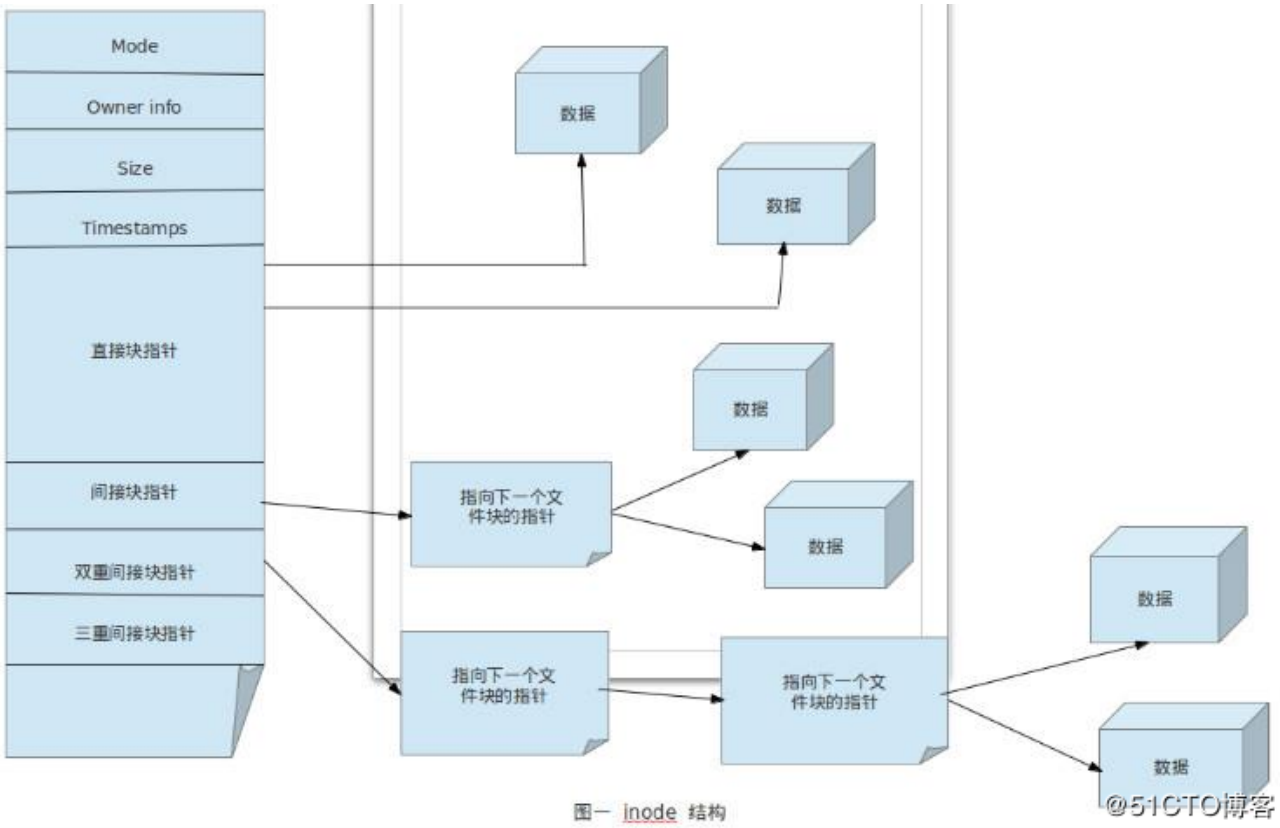
OS-9

I/O子系统

OS-10

文件系统

ext,ext2



图一 inode 结构

ext4,NTFS用extent(起始block+length)来代替单纯的block

用Btree来存文件，这样查询O(logN)

OS-11

FAT32, NTFS

FAT:由一个cluser找到下一个cluser

FAT32长文件名短文件名目录项

NTFS

FAT、HPFS 和 NTFS 文件系统概述

OS-12

File System Durability & Crash Recovery

- Write-back（回写模式）在数据更新时只写入缓存Cache。只在数据被替换出缓存时，被修改的缓存数据才会被写到后端存储。此模式的优点是数据写入速度快，因为不需要写存储；缺点是一旦更新后的数据未被写入存储时出现系统掉电的情况，数据将无法找回。
- **Write-misses写缺失的处理方式**
- 对于写操作，存在写入缓存缺失数据的情况，这时有两种处理方式：

- **Write allocate** (aka **Fetch on write**) - Datum at the missed-write location is loaded to cache, followed by a write-hit operation. In this approach, write misses are similar to read-misses.
- **No-write allocate** (aka **Write-no-allocate**, **Write around**) - Datum at the missed-write location is not loaded to cache, and is written directly to the backing store. In this approach, actually only system reads are being cached.

- Write allocate方式将写入位置读入缓存，然后采用write-hit（缓存命中写入）操作。写缺失操作与读缺失操作类似。
- No-write allocate方式并不将写入位置读入缓存，而是直接将数据写入存储。这种方式下，只有读操作会被缓存。
- 无论是Write-through还是Write-back都可以使用写缺失的两种方式之一。只是通常Write-back采用Write allocate方式，而Write-through采用No-write allocate方式；因为多次写入同一缓存时，Write allocate配合Write-back可以提升性能；而对于Write-through则没有帮助。

OS-13

ext3

ext3 支持三种日志模式，划分的依据是选择元数据块还是数据块写入日志，以及何时写入日志。

1. 日志模式（Journal）：**文件系统所有数据和元数据的改变都被记入日志。这种模式减少了丢失每个文件的机会，但需要很多额外的磁盘访问。例如：当一个新文件被创建时，它的所有数据块都必须复制一份作为日志记录。这是最安全但最慢的日志模式。

2. 预定模式（Ordered）：**只对文件系统元数据块的改变才记入日志，这样可以确保文件系统的一致性，但是不能保证文件内容的一致性。然而，ext3文件系统把元数据块和相关的数据块进行分组，以便在元数据块写入日志之前写入数据块。这样，就可以减少文件内数据损坏的机会；例如，确保增大文件的任何写访问都完全受日志的保护。这是缺省的 ext3 日志模式。

3. 写回（Writeback）：**只有对文件系统元数据的改变才被记入日志**，**对文件数据的更新与元数据记录可以不同步（相对Ordered模式而言），即ext3是支持异步的日志。

OS-14

闪存设备跟碟盘存储设备，在硬件上有不同的特性，例如：

- 抹除区块（Erasing blocks）：闪存的区块（block）在写入之前，要先做抹除（erase）的动作。抹除区块的时间可能会很长，因此最好利用系统闲置的时间来进行抹除。
- 耗损平均技术（Wear leveling）：闪存的区块有抹写次数的限制，重复抹除、写入同一个单一区块将会造成读取速度变慢，甚至损坏而无法使用，因此闪存设备的驱动程序需要将抹写的区块分散，以延长闪存寿命。用于闪存的文件系统，也需要设计出平均写入各区块的功能。
- 随机存取（Random access）：一般的硬盘，读写数据时，需要旋转磁盘，以找到存放的扇区，因此，一般使用于磁盘的文件系统，会作最优化，以避免搜索磁盘的作用。但是闪存可以随机存取，没有查找延迟时间，因此不需要这个最优化。

设计闪存文件系统的基本概念是，当存储数据需要更新时，文件系统将会把新的复本写入一个新的闪存区块，将文件指针重新指向，并在闲置时期将原有的区块抹除。例如JFFS2与YAFFS，都是这样设计。

FlexFS

OS-15

GFS

NFS

OS-16

虚拟化

Virtualizationcan be defined many ways. I will try to define it formally and also define itby giving a few examples. However loosely, virtualization is the addition of asoftware layer (the virtual machine monitor) between the hardware and theexisting software that exports an interface at the same level as the underlyinghardware.

Inthe strictest case the exported interface is the exact same as the underlyinghardware and the virtual machine monitor provides no functionality exceptmultiplexing the hardware among multiple VMs. This was largely the case in theold IBM VM/360 systems.

Howeverthe layer really can export a different hardware interface as the case incross-ISA emulators. Also the layer can provide additional functionality notpresent in the operating system.

Ithink of virtualization as the addition of a layer of software that can run theoriginal software with little or no changes.

OS-17

Virtualization: CPU and Memory

虚拟机想执行特权指令：trap and emulate

OS-18

I/O虚拟化

总结OS-16/17/18的一篇讲虚拟化的文章

OS-19

Scalable Locking

一个CAS操作的过程可以用以下c代码表示: [1]

```c
int cas(long *addr, long old, long new)
{
    /* Executes atomically. */
    if(*addr != old)
        return 0;
    *addr = new;
    return 1;
}
```

OS-20

读写锁

除了锁之外可以用一些原子操作例如fetch_and_add, compare_and_swap

OS-21

bug survey:

1. 违反原子性
2. 两个线程之间的执行顺序违反

OS-22

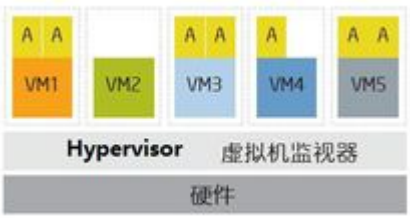
死锁

知识点

hypervisor

Hypervisor——一种运行在基础物理服务器和操作系统之间的中间软件

层,可允许多个操作系统和应用共享硬件。也可叫做VMM（virtual machine monitor），即虚拟机监视器。



Hypervisors是一种在虚拟环境中的"元"操作系统。他们可以访问服务器上包括磁盘和内存在内的所有物理设备。Hypervisors不但协调着这些硬件资源的访问，也同时在各个虚拟机之间施加防护。当服务器启动并执行Hypervisor时，它会加载所有虚拟机客户端的操作系统同时会分配给每一台虚拟机适量的内存，CPU，网络和磁盘。

Android Architecture

Programmed I/O

Programmed I/O是指用特定的IO指令实现从设备到CPU的数据传输。具体表现在Intel的x86平台上有访问端口的IN和OUT指令。

PIO是 CPU 与外围设备（如网卡、硬盘等）传输数据的一种方法。

DMA

直接内存访问（**D**irect **M**emory **A**ccess，**DMA**）是计算机科学中的一种内存访问技术。它允许某些计算机内部的硬件子系统（计算机外设），可以独立地直接读写系统内存，而不需中央处理器（CPU）介入处理 。在同等程度的处理器负担下，DMA是一种快速的数据传送方式。很多硬件的系统会使用DMA，包含硬盘控制器、绘图显卡、网卡和声卡。

MMU

MMU负责地址翻译，TLB是地址翻译的cache。

内存管理单元（英语：**memory management unit**，缩写为**MMU**），有时称作**分页内存管理单元**（英语：**paged memory management unit**，缩写为**PMMU**）。它是一种负责处理中央处理器（CPU）的内存访问请求的计算机硬件。它的功能包括虚拟地址到物理地址的转换（即虚拟内存管理）、内存保护、中央处理器高速缓存的控制，在较为简单的计算机体系结构中，负责总线的仲裁以及存储体切换（bank switching，尤其是在8位的系统上）。

现代的内存管理单元是以页的方式，分区虚拟地址空间（处理器使用的地址范围）的；页的大小是2的n次方，通常为几KB。地址尾部的n位（页大小的2的次方数）作为页内的偏移量保持不变。其余的地址位（address）为（虚拟）页号。内存管理单元通常借助一种叫做转译旁观缓冲器（Translation Lookaside Buffer，缩写为TLB）的相联高速缓存（associative cache）来将虚拟页号转换为物理页号。当后备缓冲器中没有转换记录时，则使用一种较慢的机制，其中包括专用硬件（hardware-specific）的数据结构（Data structure）或软件辅助手段。这个数据结构称为分页表，页表中的数据就叫做页表项（page table entry，缩写为PTE）。物理页号结合页偏移量便提供出了完整的物理地址。

页表或转换后备缓冲器中数据项包括的信息有：一、"脏位"（页面重写标志位，dirty bit）——表示该页是否被写过。二、"访问位"（accessed bit）——表示该页最后使用于何时，以便于最近最少使用页面置换算法（least recently used page replacement algorithm）的实现。三、哪种进程可以读写该页的信息，例如用户模式（user mode）进程还是特权模式（supervisor mode）进程。四、该页是否应被高速缓冲的信息。

有时，TLB或PTE会禁止对虚拟页的访问，这可能是因为没有物理随机存取存储器（random access memory）与虚拟页相关联。如果是这种情况，MMU将向CPU发出页错误（page fault）的信号。操作系统operating system）将进行处理，也许会尝试寻找RAM的空白帧，同时创建一个新的PTE将之映射到所请求的虚拟地址。如果没有空闲的RAM，可能必须关闭一个已经存在的页面，使用一些替换算法，将之保存到磁盘中（这被称之为页面调度（paging）。在一些MMU中，PTEs或者TLB也存在一些缺点，在这样的情况下操作系统将必须释放空间以供新的映射。

BIOS

BIOS是英文"Basic Input Output System"的缩略语，直译过来后中文名称就是"基本输入输出系统"。其实，它是一组固化到计算机内主板上一个ROM芯片上的程序，它保存着计算机最重要的基本输入输出的程序、系统设置信息、开机后自检程序和系统自启动程序。 其主要功能是为计算机提供最底层的、最直接的硬件设置和控制。

ROM

ROM是只读内存（Read-Only Memory）的简称，是一种只能读出事先所存数据的固态半导体存储器。其特性是一旦储存资料就无法再将之改变或删除。通常用在不需经常变更资料的电子或电脑系统中，资料并且不会因为电源关闭而消失。

RAM

RAM（random access memory）随机存储器。存储单元的内容可按需随意取出或存入，且存取的速度与存储单元的位置无关的存储器。这种存储器在断电时将丢失其存储内容，故主要用于存储短时间使用的程序。 按照存储信息的不同，随机存储器又分为静态随机存储器（Static RAM,SRAM)和动态随机存储器（Dynamic RAM,DRAM)。

CMOS

CMOS（Complementary Metal Oxide Semiconductor），互补金属氧化物半导体，电压控制的一种放大器件。是组成CMOS数字集成电路的基本单元。

BIOS与CMOS

BIOS与CMOS 区别　　由于CMOS与BIOS都跟电脑系统设置密切相关，所以才有CMOS设置和BIOS设置的说法。也正因此，初学者常将二者混淆。CMOS 是电脑主机板上一块特殊的RAM芯片，是系统参数存放的地方，而BIOS中系统设置程序是完成参数设置的手段。因此，准确的说法应是通过BIOS设置程序对CMOS参数进行设置。而我们平常所说的CMOS设置和BIOS设置是其简化说法，也就在一定程度上造成了两个概念的混淆。事实上，BIOS程序是储存在主板上一块 EEPROM Flash 芯片中的，CMOS存储器是用来存储BIOS设定后的要保存数据的，包括一些系统的硬件配置和用户对某些参数的设定，比如传统BIOS的系统密码和设备启动顺序等等 联系　　BIOS是一组设置硬件的电脑程序，保存在主板上的一块EPROM或EEPROM芯片中，里面装有系统的重要信息和设置系统参数的设置程序——BIOS Setup程序。而CMOS即：Complementary Metal Oxide Semiconductor——互补金属氧化物半导体，是主板上的一块可读写的RAM芯片，用来保存当前系统的硬件配置和用户对参数的设定，其内容可通过设置程序进行读写。CMOS芯片由主板上的钮扣电池供电，即使系统断电，参数也不会丢失。CMOS芯片只有保存数据的功能，而对CMOS中各项参数的修改要通过BIOS的设定程序来实现。BIOS与CMOS既相关又不同：BIOS中的系统设置程序是完成CMOS参数设置的手段；CMOS RAM既是BIOS设定系统参数的存放场所，又是 BIOS设定系统参数的结果。因此，完整的说法应该是"通过BIOS设置程序对CMOS参数进行设置"。由于 BIOS和CMOS都跟系统设置密初相关，所以在实际使用过程中造成了BIOS设置和CMOS设置的说法，其实指的都是同一回事，但BIOS与CMOS却是两个完全不同的概念，切勿混淆。

volatile

volatile是一个类型修饰符（type specifier），就像大家更熟悉的const一样，它是被设计用来修饰被不同线程访问和修改的变量。**volatile**的作用是作为指令关键字，确保本条指令不会因编译器的优化而省略，且要求每次直接读值。

volatile的变量是说这变量可能会被意想不到地改变，这样，编译器就不会去假设这个变量的值了。

System call

Linux 的系统调用通过 int 80h 实现，用系统调用号来区分入口函数。操作系统实现系统调用的基本过程是：

1. 应用程序调用库函数（API）；
2. API 将系统调用号存入 EAX，然后通过中断调用使系统进入内核态；

3. 内核中的中断处理函数根据系统调用号，调用对应的内核函数（系统调用）；
4. 系统调用完成相应功能，将返回值存入 EAX，返回到中断处理函数；
5. 中断处理函数返回到 API 中；
6. API 将 EAX 返回给应用程序。

应用程序调用系统调用的过程是：

1. 把系统调用的编号存入 EAX；
2. 把函数参数存入其它通用寄存器；
3. 触发 0x80 号中断（int 0x80）。

PTE中的dirty bit

表示这个页有没有没写过

page fault

MMU翻译后找到的PTE中有效位为0

地址空间

```
/*
 * Virtual memory map:                                   Permissions
 *                                                       kernel/user
 *
 *    4 Gig --------->  +------------------------------+
 *                      |       Memory-mapped I/O      | RW/--
 *    IOMEMBASE ---->   +------------------------------+ 0xfe000000
 *                      |                              | RW/--
 *                      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *                      :              .               :
 *                      :              .               :
 *                      :              .               :
 *                      |~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~| RW/--
 *                      |                              | RW/--
 *                      |   Remapped Physical Memory   | RW/--
 *                      |                              | RW/--
 *    KERNBASE ----->   +------------------------------+ 0xf0000000
 *                      |      Invalid Memory (*)      | --/--  PTSIZE
 *    KSTACKTOP ---->   +------------------------------+ 0xefc00000      --+
 *                      |     CPU0's Kernel Stack      | RW/--  KSTKSIZE   |
 *                      | - - - - - - - - - - - - - - -|                   |
 *                      |      Invalid Memory (*)      | --/--  KSTKGAP    |
 *                      +------------------------------+                   |
 *                      |     CPU1's Kernel Stack      | RW/--  KSTKSIZE   |
 *                      | - - - - - - - - - - - - - - -|                 PTSIZE
 *                      |      Invalid Memory (*)      | --/--  KSTKGAP    |
 *                      +------------------------------+                   |
 *                      :              .               :                   |
 *                      :              .               :                   |
 *    ULIM     ------> +------------------------------+ 0xef800000      --+
 *                      |  Cur. Page Table (User R-)   | R-/R-  PTSIZE

 *    UVPT      ---->   +------------------------------+ 0xef400000
```

```
 *                    |          RO PAGES            | R-/R-  PTSIZE
 *    UPAGES    ---->  +------------------------------+ 0xef000000
 *                    |           RO ENVS            | R-/R-  PTSIZE
 * UTOP,UENVS ------>  +------------------------------+ 0xeec00000
 * UXSTACKTOP -/       |     User Exception Stack     | RW/RW  PGSIZE
 *                     +------------------------------+ 0xeebff000
 *                     |       Empty Memory (*)       | --/--  PGSIZE
 *    USTACKTOP  --->  +------------------------------+ 0xeebfe000
 *                     |      Normal User Stack       | RW/RW  PGSIZE
 *                     +------------------------------+ 0xeebfd000
 *                     |                              |
 *                     |                              |
 *                     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *                     .                              .
 *                     .                              .
 *                     .                              .
 *                     |~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|
 *                     |     Program Data & Heap      |
 *    UTEXT -------->  +------------------------------+ 0x00800000
 *    PFTEMP ------->  |       Empty Memory (*)       |        PTSIZE
 *                     |                              |
 *    UTEMP -------->  +------------------------------+ 0x00400000      --+
 *                     |       Empty Memory (*)       |                   |
 *                     | - - - - - - - - - - - - - - -|                   |
 *                     |  User STAB Data (optional)   |                 PTSIZE
 *    USTABDATA ---->  +------------------------------+ 0x00200000        |
 *                     |       Empty Memory (*)       |                   |
 *    0 ------------>  +------------------------------+                 --+
 *
 * (*) Note: The kernel ensures that "Invalid Memory" (ULIM) is *never*
 *     mapped.  "Empty Memory" is normally unmapped, but user programs may
 *     map pages there if desired.  JOS user programs map pages temporarily
 *     at UTEMP.
 */
```

PIC

Programmable Interrupt Controller

I/O devices have (unique or shared) Interrupt Request Lines (IRQs)。IRQs are mapped by special hardware to interrupt vectors, and passed to the CPUThis hardware is called a Programmable Interrupt Controller (PIC)

ISR

中断处理服务就是Interrupt Handler

Linux启动流程

https://www.binss.me/blog/boot-process-of-linux/

TODO

？？？ User ISA与System ISA的区别：系统ISA中有一些特权指令

Question:Which software does ABI Emulation?

不懂

【已解决】OS-1 23-27

OS-2 33-41

【已解决】OS-3 14-15(害处) OS-17

chip是什么

虚拟机的IP是什么，虚拟化的原理?

【已解答】线性地址，逻辑地址，物理地址，虚拟地址的区别?

逻辑地址：段：偏移

线性地址：段描述符+偏移（因为段描述符被设置为0）所以线性地址数值上等于虚拟地址

虚拟地址；偏移

物理地址：线性地址地址翻译

【已解答】系统的存储，DRAM，SRAM，RAM，FLASH STORAGE

GDT里面存的什么，跟代码段 数据段这些有什么关系

HW的标准答案

【半解决】IPC与线程间通信

什么是trap（csapp 异常控制流）

【已解决】什么是IPC的port（就是消息队列）

对L4的微内核来说，什么是异步的IPC

【已解决】外内核如何确保安全，应用之间不会

【已解决】OS-4 11,14

【已解决】堆栈的区别？堆里面存的是全局变量（一些全局变量，一些被分配空间的field，被malloc的），栈一直向下增长，具体可参考memlayout

OS-10 30

OS-12 30,43

OS-13 20 xaction是什么

ext3 journal模式，什么时候commit，什么时候free，什么时候journal上的commit被写入disk

什么是FlexFS

OS-17 37 44-

critical sections 是什么

英语

voltage:电压

be prone to somethings:易于遭受某事

notorious：臭名昭著的

harness:利用，控制

faithfully:忠诚的

unified:统一的

decouple:解耦

infrastructure：基础设施

resident:定居的，常驻的，存于内存中的

detriment:损害 不利

primitive: 原始的，原函数

simultaneously： 同时

eliminate: 消除

Rendezvous:会合

revocation:撤销

conventional: 常用的，常规的

Cascade： 级联

Nested：嵌套的

reentrant： 折返

comprehensive：全面的

sophisticated：复杂的

paranoid：偏执

Obsolete：过时的，弃用的

negligible：微不足道

Durability:持久力，续航

tension：紧张，拉力

invariants : 不变

Durable (Persistence):耐用（持久性）

semantics：语义

barrier: 屏障

batch:批量

Isomorphism： 同构

Encapsulation：封装

Interposition：介入

Isolation：隔离

Mobility：流动性

violation:违反