# 340 review

- Operation system： 是用来管理电脑硬件的程序，它同样提供了一软件程序的基础并且让电脑用户和电脑硬件互动。
    - windows，linux，ubuntu，mac os from apple and android
- 在电脑硬件中拥有像cpu，memory或者i/o devices （input/out)
- 例如你想要打开word文档，你就需要os去告诉硬件去加载word文档，然后将word文档加载到主内存中。当在word文档里输入时，你又要让os告诉硬件让输入的东西通过显示器输出在用户面前。简单来说os让用户更简单的使用电脑硬件来完成自己的操作。
- functions of OS:
    - 用户与硬件的互动
    - 资源分配（allocation of resources)
    - 管理内存，安全等
- Goals of OS：
    - convenience（让用户更方便的使用硬件）
    - Efficiency（如果没有os，用户需要自己调配资源，读取或储存等。）

## Chapter 1

- Basic of operating system:普遍现在电脑系统包含1个或多个cpu和好几个设备连接共同接口来访问和共享内存。
    - cpu：计算机系统的大脑，它负责计算，处理等。
    - device：硬盘，鼠标，键盘和显示器等他们属于它们的controller。这些controller负责那些设备正常运行。
    - 这些controller与cpu连接着一条总线（bus），这条bus会同样连接到共享内存里。这两个东西可以同时（concurrently)执行任务。每一个设备如果想要运行就必须先加载倒主内存。他们可以同时执行任务是因为内存管理器会为设备或cpu提供他们需要的内存。在任何任务需要加载时，它必须加载到你的主内存中。
    - bootstrap program：（引导程序）一个初始的程序帮助电脑开机或重启。它储存在ROM（read-only memory) 中。它必须知道如何加载并运行OS。它必须位于并加载到os的核心部分(OS Kernel)。
    - Interrupt(中止）:打断事件的发生通常是由硬件或软件引起。硬件可以在任何时间通过系统bus触发中止命令给cpu
    - system call系统调用（monitor call）：通过软件的interrupt我们通常叫做system call。
    - 当cpu被打断时，他会停止他目前的任务并且立即转移它的执行力到fixed location（固定的地点）需要打断的任务将会被执行。在完成后，cpu会恢复并继续执行打断前的任务。
        - fixed location通常包含起始地址service routine（为什么打断以及打断后要做什么会写入service routine）打断的起始点。
    - 储存设备的层次结构（从前往后，越往后的越便宜也越慢但存储量会越大）
        - Registers（通过bit储存数据，例如0或者1因此他是最小的也是最快的）--->cache（比register 大一点并且慢一点）--->Main memory--->Electronic disk--->magnetic disk--->optical disk --->Magnetic Tapes

        - 由于main memory（RAM)的内存有限如果我们想要存储更多的数据我们就需要secondary memory。当我们需要读取大量数据时，我们会先从secondary memory搬到mian memory再执行。main memory或内存小的储存器是（volatile）易失性的而越大的储存器是非易失性的
            - Volatile意味着如果关机后数据会消失而not volatile就算关机了数据也不会消失
    - 存储同样也是许多样中的一个input/output设备在电脑中。大部分的系统代码是用来管理input/output的操作
        - device controller包含了local buff storage（本地缓冲存储） 和set of special purpose regisger（专用寄存器组）
        - 每个controller需要os给予设备驱动（device driver:了解controller具有什么样的性质)来运行。当device driver将适当的寄存器（register)加载到设备控制器，设备控制器会检查寄存器的内容来决定接下来要采取的行动
            - 控制器将转移设备的数据到本地的缓冲区。当控制器转移完数据后。设备控制器会通过中断来通知设备程序它已经完成了，之后设备驱动会把设备控制权返回给OS。这个打断运行方式的缺点在于如果只是搬运一些少量的数据还可以，但如果大量搬运数据就需要使用DMA(Diect memory accesss直接内存访问)
                - DMV可以不受cpu的干预直接把缓冲区的数据发送给内存。最大的优势在于由于cpu不会被每次的打断干预，它可以更高效率的执行别的工作。
    - Single processor system, mutiprocessor systems and clustered system(意味着每个系统持有的cpu数量)
        - single processor system：只有一个主要cpu来执行来自用户的(general purpose instruction)大众命令。计算机上同样还有一个微处理器来处理信息例如转换键盘的输入转换为别的code
        - Multiprocessor system:两个或更多的处理器。同样被称为平行系统(parallel system)。处理器们一起工作来分享计算机的总线，有时会共享clock, memory and peripheral devices（外围设备）。
            - 好处：Increased throughput（提高吞吐量：用来形容电脑的性能），Economy of scale（规模经济）, Increase reliability
                - DIfferent type of multiprocessor system:  symmetric multiprocessing(对称多处理) and Asymmetric multiprocessing(非对称多处理)
                    - Symmetric multiprocessing: Cpu彼此相似以及他们同时参与每一个processor
                    - Asymmetric multiprocessing: 其中一个cpu会占有主导权，其他的cpu会给予有主导权的cpu之下来单独处理每一个processor,例如cpu1主导cpu2,3,4。cpu2执行processor1，cpu3执行processor2等。
            - Clustered system:类似于multiprocessor system，但它结合了两个或更多的单个独立系统（而不是cpu）形成一个系统。并且它可以变成对称多处理（多个系统同时运行，互相监控）或者非对称处理（一个系统监管另一个系统运行）。
    - Multiprogramming and time sharing
        - Multiprogramming:有能力通过cpu运行多个程序。想要提高cpu的利用率通过整理运行任务让cpu永远都有一个任务在执行。当有很多任务需要OS执行时，cpu完成了任务1的作业时，任务1需要电脑的其他资源继续工作，而cpu可以立马去执行任务2的工作而不是等待任务1完全完成。但用户无法与OS互动当任务在执行时
        - Time sharing:相同点在于cpu可以多线程执行任务，但cpu是经常的在多个任务交换执行，用户同样可以在任务执行时与OS互动。简单来说它可以让多个用户操作。cpu的速度永远快于人的执行速度，因此当用户1需要一些output时，cpu已经完成它的命令并已经交换到另一个用户2的命令上了

## Chapter 2

- User Interface(CLI(command line interface) and GUI(graphical user interface)) :允许用户与操作系统或计算机互动
- Program exception(程序异常)：系统必须可以加载到内存并且运行程序
- I/O operation：背后有操作系统来帮助I/O device来与内存操作。
- File system manipulation(文件系统操作): 涉及到文件，如果操作或管理文件
- Communications: 通信在进程之间，它会告诉你什么时进程，每个进程可以互相通信，更有效率的方式完成
- Error detection：系统需要不断地检查错误是否发生在任何设备中。
- Resource allocation: 资源分配给不同的进程和用户。例如CPU,文件或者I/O
- Protection and security：数据应该被保护，一个进程在执行时，他不能被其他的任何操作或进程干扰
- System call: 提供一个操作系统的接口的服务。当用户模式下需要访问一些内存或硬件，system call就是帮助系统转换成核心模式来获取资源。
    - User mode: 如果使用用户模式，程序无法直接访问内存，硬件以及其他资源的内存。系统不会崩溃基于一个程序的崩溃。（safe mode)
    - Kernel mode: 直接可以访问用户模式无法访问的东西，它同样意味着Privileged mode(特权模式)。在核心模式下如果程序崩溃，整个系统崩溃是核心模式的一个缺点
        - Example关于如果system call 使用read的copy to another file（每一行步骤都需要system call)(system call每次都需要当我们需要读取内存，例如读取用户输入的filename)
            - 我们需要input file的名字（acquire input file name)
            - （write prompt to screen）让用户输入file的名字作为input
            - 最后需要接受用户给予的input data
            - 为了copy到另一个文件，我们需要输出的文件名
            - 显示输出文件的文件名
            - 再次接受用户介于的input 文件名
            - 打开input文件
            - 如果文件不存在，直接舍弃(abort)。
            - 创建output文件
            - 如果output文件已存在，直接舍弃
            - 读取input file的信息(这是一个loop因为需要一直读取文件里的data直到没有任何data可以读取)
            - copy到output file(当loop结束)
            - 关闭output file
            - 写copy任务完成
            - 关闭程序
- Process control, File Manipulation, Device management, Information Maintenance and communication
    - Process control : 用于控制进程的系统调用，进程完成时必须正常结束。如果进程出现错误必须停止或中止。
        - end, abort
        - load, execute
        - create process, terminate process
        - get process attributes, set process attributes
        - wait for time
        - wait event, signal event

        - allocate and free memory
    - File manipulation：操作或管理文件的系统调用
        - create file, delete file
        - open, close
        - read, write, reposition
        - get file attributes, set file attributes
    - Device manipulation管理和操作设备的系统调用
        - request device, release device
        - read,write,reposition
        - get device attributes, set device attributes
        - logically attach or detach devices(逻辑地附加或分离设备)
    - Information maintenance 任何信息在系统中都得保存或者更新
        - Get time or data, set time or date
        - get system data, set system data
        - get process, file, or device attributes
        - set process, file, or device attributes
    - Communications 系统调用用于连接每个进程与每个进程之间的通信
        - create, delete communication connection
        - send, receive messages
        - transfer status information
        - attach or detach remote devices
- Layered structure
    - Layer 0 is hardware as lowest layer. Top layer n is user interface
    - 这样子做的优点在于容易实现和调试。以及受到外层的保护，最内层的不会直接被用户询问。
    - 缺点在于非常空难的去设计因为你需要非常小心和具体的设计哪一个是特定层之上或者哪一个是特定层之下。又或者如果最上层的user interface需要访问最里层的hardware，我们需要用到许多的system call并且需要很多时间。
- Virtual machines：把计算机里的每一个硬件提供到几个不同的执行环境，然后为该执行环境创造一个错觉(illusion) 就像它们每一个都运行在它们自己的私人计算机上
    - 虚拟机的软件会认为自己核心模式而虚拟机其实是在用户模式上运行
    - 但虚拟机同样也有自己的核心模式和用户模式但基于计算机本体，虚拟机还是运行在用户模式上

## Chapter 3 & 4

- Process
    - 进程在执行时，可以将进程认为程序。早期的进程只有一个线程单位。现在的进程可以有多个线程例如多线程护腕！火法牛逼！
        - Process state 当进程在执行时，它会一直改变它的状态。new状态下的process会进入准备状态，等分配到处理时他进入运行状态如果没有别的需求进入结束状态。如果需要额外资源进入等待状态，当额外资源完成分配时再进入准备状态重复分配处理器进入运行状态。如果被打断会回来准备状态。
            - New: The process is being created
            - Running: Instructions are being executed
            - waiting：The process is waiting for some event to occur (例如等待I/O完成或者等待接受指令信号)
            - Ready：进程等待知道被分配给处理器
            - Terminated:进程完成了他的执行任务

- Thread：线程是这个进程执行的单位。进程可以用多个线程,线程越多，一次性执行的任务也越多。它包含了：
    - Thread ID
    - Program counter
    - Register set
    - Stack
        - 进程中的thread互相分享它们的：
            - code section
            - data section
            - other OS resources
        - 多线程好处：
            - Responsiveness（响应性）尽管一个线程堵塞或者其他问题。其他线程可以继续运行
            - Resource sharing：分享数据允许好几个不同的线程活跃在同样的地址空间
            - economy：更经济的创建context switch 线程
    - Multithreading models and hyperthreading
        - Two type of threads
            - user threads：被内核之上支持并且没有内核的支持下进行管理
            - kernel threads: 线程由操作系统直接支持和管理
        - Relationship in three common ways
            - many to one: 许多的user thread连接到1个kernel thread
                - 线程的管理完成是在用户空间里的所以比较效率
                - 全部的进程被堵塞如果有一个线程堵塞了系统调用 （缺点）
                - 尽管拥有多处理器我们还是无法核心线程也只会在其中一个里运行（缺点）
            - one to one： 一个用户线程连接着一个核心线程
                - 允许其他线程继续运行如果有一个线程堵塞了
                - 允许多线程运行在多处理器
                - 如果想要创建用户线程，就需要创建相应的核心线程（缺点）
                - 由于多个核心线程的创建可能会导致软件性能的负担因此线程的数量有限制（缺点）
            - many to many：多个用户线程连接多个核心线程
                - 多个用户线程可以连接相等或者更少一点的核心线程
                - 开发员可以任意创建几个用户线程并还可以平行运行在核心线程上
                - 尽管一个线程堵塞，核心线程可以让其他线程继续工作
- Fork() and exec() system call
    - Fork() system call是用来创建一个分离且副本的进程，创建一个相同的进程但是它们的进程id不一样。 fork()会创建程序的2^n次方
    - exec() 简单来说 用于将一个进程替换为另一个进程，然后它们会有相同的进程id
- Cancellation: 指线程在完成之前被取消，例如许多线程在数据库搜索答案，当有一个线程已经回馈了答案，其他的线程会被取消
    - Asynchronous cancellation: 一个线程立即结束目标线程
    - deferred cancellation：目标线程会不断的检查自己是否应该结束，它有机会自己有序的结束 （安全的）
    - cancellation将会困难因为：
        - 资源已经分配给了即将要被取消的线程(无法回收全部资源 )

        - 当两个线程在分享数据时，线程被结束将会得到不完整的数据
- process control block 进程控制块：用来表示操作系统中特定进程的东西
    - Process state
    - Process number
    - Program counter: 特定进程执行的下一行指令的地址
    - Registers:
    - Memory limits：决定了进程使用的内存
    - List of open files
    - CPU secheduling information：同样储存再PCB.帮助系统分配特定进程多少时间都是由scheduling决定的
    - accounting information：记录了哪些资源正在被使用
    - I/O status information：哪些输入/输出设备分配给特定的进程
- Context Switch： Represented in the PCB of the process
    - 当CPU被打断时，它需要执行优先级更高的任务。而被打断的任务的context会被保存直到优先级更高的任务完成。context会被恢复并且继续执行
- Process creation: 被创建的进程叫父进程，新的进程基于前一个进程下的叫子进程
    - PID类似于编号给每一个进程编码
    - 两种情况发生在父进程和子进程: 父进程与子进程一同执行，或者父进程会等待子进程完成后才执行
    - 两种情况发生在新进程的地址：子进程是父进程的副本（意味着子进程与父进程有同样的数据），或者子进程加载了一个新程序
- process terminates：当进程完成他的任务时，它会让OS删除它自己通过使用exit（)系统调用
    - 子进程结束后，它会回馈一个状态值给它的父进程。进程完成后,OS会解除分配资源给进程
    - 父进程可能会中止子进程运行的原因：1.子进程使用的资源量超出了它所分配的量。 2.给予子进程的任务不再需要。 父进程正在结束，所以子进程无法继续执行
- Interprocess communication(IPC进程间通信) 进程共同执行时可以有Independent processes 或者 cooperating processes
    - Independent process: 进程们执行时不会互相影响
    - Cooperating process: 进程们执行时会互相影响
        - 好处：共享数据，计算速度提高，模块化（modularity）（把系统分为各个模块设计分分工合作），方便（不同的进程同时运行分享信息）
            - 两种基础模式：shared memory 和 message passing
                - Shared memory：建立一个内存区域，P1可以把信息写入共享区，P2从共享区读取
                    - Two kinds of buffers: unbounded buffer and bounded buffer
                        - Unbounded buffer：没有实际的大小限制，成品可能需要等待，但生产可以一直生产
                        - Bounded buffer: 有固定大小的缓冲区。因此消费者必须等缓冲区为空的时候，而生产者也必须等待如果缓冲区满了
                - Message passing：进程相互交换信息，p1把写完的信息放进核心，核心再把信息发送给p2
                    - 好处在于通信时不需要分享同样地址空间
                    - Message-passing需要两个步骤：send and receive，而信息的大小可以是固定的或者可变的
                        - fixed size：系统等级直截了当，但使得程序任务更困难

        - variable size：系统等级需要更复杂的实现，但任务程序变得简单
    - 基于接受者和发送者准确的名字来发送信息。link会自动建立在想要通信的两个处理器之间