Chapter 1

- 通用计算机系统：一个或多个 CPU 和若干设备控制器，公共总线提供了共享内存的访问。
- 引导程序（BootStrap Program）一般位于 ROM 或 EEPROM，它初始化系统的各组件（CPU 寄存器、设备控制器、内存内容），计算机开机后操作系统最终被加载到 RAM 中。
- 中断：系统发生某个异步/同步事件后，处理器暂停正在执行的程序，转去执行处理该事件程序的过程。外中断（中断）：I/O 中断、时钟中断； 内中断（异常）：系统调用、缺页、断点、程序性异常（如算数溢出等）。现代操作系统是中断（Interrupt Driven）驱动的。
- 存取速度：磁带<光盘<硬盘<固态硬盘<内存<高速缓存<寄存器
- 内核态：模式位（0），能够访问系统所有资源，执行特权指令（I/0、时钟、控制控制寄存器），使用内核栈。

Chapter 2

- 系统调用与库函数的区别：API 是一组函数定义，说明了如何获得一个给定的服务；系统调用则是通过软中断向内核发出一个明确的请求。系统调用的实现是在内核完成的，用户态函数是在函数库实现的。
- 系统调用传参方法：寄存器、内存的块或表（地址通过寄存器传递）、堆栈。
- 系统调用的类型：进程控制、文件管理、设备管理、信息维护、通信、保护。
- 操作系统的结构：①简单结构（MS-DOS、最初的 UNIX）：单线程，没有很好地区分功能的接口和层次，应用程序能访问 I/O 会导致系统崩溃，没能提供双模式和硬件保护②分层方法：有硬件支持，使用自顶向下的方法，最底层 0 为硬件，最高层 N 为用户接口，需要执行更长时间③微内核④可加载内核模块⑤混合系统。
- 微内核的主要功能是为客户端程序和运行在用户空间中的各种服务提供通信，只有进程通信、内存管理、CPU 调度等最基本功能由微内核实现。特点是便于扩展操作系统，安全可靠，但性能受损。
- 权限命令：Set value of timer; Clear memory; Turn off interrupts; Modify entries in device-status table; Access I/O device 非权限命令：Read the clock; Issue a trap instruction; Switch from user to kernel mode.

Chapter3

- 进程是资源的分配单位，线程是资源的调度单位。
- 进程是正在执行的程序（Unit of Resource ownership & Unit of Dispatching）：包含代码、PC、寄存器、数据段、堆栈、堆。①代码段 ②数据段：存放已初始化全局变量（静态变量和全局变量）③BSS 段：包含程序中未初始化的全局变量④堆⑤栈。
- 进程运行时可能发生①发出 I/0 请求②时间片用完③创建子进程④等待中断
- PCB：1.Process state 2.Program counter 3.CPU registers 4.CPU scheduling information 5.Memory-management information 6.Accounting information 7.I/O status information 8.page table or relocation register and limit register 9.file open table
- 长期调度程序（long-term scheduler, job scheduler）从作业池中选择放入就绪队列的进程，其控制着程序的多道性。短期调度程序（short-term scheduler, CPU scheduler）从就绪队列中选择下一个要执行的进程并分配 CPU。中期调度程序：可将进程从内存或 CPU 竞争中移出，之后重新调入内存，并从中断处开始执行。
- 上下文切换：保存程序计数器和其他寄存器的值，更新 PCB 的信息，把 PCB 移入就绪等队列，选择另一个进程执行并更新 PCB，更新内存管理的数据结构，恢复上下文。【在保存当前进程的内核状态和 CPU 寄存器之前，需要先把该进程的虚拟内存、栈等保存下来；而加载了下一进程的内核态后，还需要刷新新进程的虚拟内存和用户栈】
- 父进程（pid>0，实际上是子进程的 pid），子进程（pid=0）。子进程复制父进程的数据段，BSS 段，堆空间，栈空间，文件描述符，但是对于文件描述符关联的内核文件表项（即 struct file 结构体），代码段则是采用共享的方式。但是只进程只复制当前在运行的父进程。
- 进程可以通过系统调用 wait()返回状态值到父进程，系统调用 exit()用于终止进程。僵尸进程：当子进程已经终止时，其父进程没有调用 wait（因为虽然资源被释放，但是还在进程表的条目中）。 孤儿进程（orphan process）：如果父进程没有调用 wait()就终止，那么 init 进程将会作为孤儿进程的父进程。
- 父进程终止子进程的原因：①子进程使用了超过它分配的资源②分配给子进程的任务不再需要③父进程正在退出，且操作系统不允许无父进程的子进程继续运行（级联终止）
- 进程间通信（IPC）：共享内存，消息传递。管道：普通管道（单向，父子关系，又称匿名管道），命名管道（双向）。信号（SIGNAL）。

Chapter4

- 每个线程有独立的线程 ID，程序计数器，寄存器组和堆栈。是进程内一个执行单元或一个可调度实体。有执行状态（状态转换），不运行时保存上下文，有一个执行栈，有一些局部变量的静态存储，可存取所在进程的内存和其他资源，可以创建、撤消另一个线程。同一进程内的所有线程共享进程的地址空间。
- 用户线程（user thread）：位于内核之上，管理无需内核支持，用户线程切换不需要内核特权，一对多模型中一个线程发起系统调用而阻塞，则整个进程在等待。内核线程（kernel thread）由操作系统直接管理，一个线程发起系统调用而阻塞，不会影响其他线程。
- 多对一模型：映射多个用户线程到一个内核线程。优点：①不需要操作系统支持②可以调整调度策略以满足应用程序（用户级别）的需求③由于没有系统调用，因此降低了线程操作的开销；缺点：①无法利用多处理器（没有真正的并行性）②一个线程阻塞时整个进程阻塞。
- 一对一模型：映射每一个用户线程到内核线程中。①每个内核级线程可以在多处理器上并行运行②当一个线程阻塞时，可以调度进程中的其他线程；缺点①线程操作的开销更高②操作系统必须随着线程数量的增加而很好地扩展。
- 多对多模型：多路复用多个用户级线程到同样数量或数量更少的内核线程上。
- 双层模型：在多对多模型的基础上允许绑定某个用户线程到一个内核线程。
- 线程池：速度更快，限制可用线程的数量，将要执行的任务从创建任务的机制中分离出来。

Chapter 5

- 非抢占式（Nonpreemptive）调度：①Switches from running to waiting state. ④Terminates. 抢占式（Preemptive）调度：②Switches from running to ready state.（被抢占） ③Switches from waiting to ready.（主动去抢占）
- Dispatcher：①切换上下文②切换到用户模式③跳转到用户程序的合适位置
- CPU Burst: 对长作业更好，少量长 CPU 执行； I/O Burst：对短作业更好，大量短 CPU 执行
- 周转时间（turnaround time）：进程从提交到完成所经历的时间；等待时间：进程在就绪队列中等待的时间总和；响应时间：提出请求到首次被响应的时间。响应比 R=(waitting time + execution time)/ execution time
- 先来先服务（FCFS）：有利于长进程（或 CPU Bound），而不利于短进程（或 I/O Bound），平均等待时间往往很长。
- 最短作业优先（SJF）：对预计执行时间短的作业（进程）优先分派处理器，是最优的。最短剩余时间优先（SRTF）是基于抢占的 SJF 算法。估算下一次 CPU Burst 的长度：$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$。 有利于 I/O 密集型的作业。非抢占式会导致饥饿。
- 优先级调度（priority scheduling）：存在无穷阻塞或饥饿的问题，但是高响应比优先：满足短任务优先，随着长作业等待时间增加，响应比变大，机会增大，因此不会导致饥饿。
- 时间片轮转（RR）：就绪进程按照 FCFS 原则，排成队列，每次调度时将 CPU 分派给队首进程，让其执行一个时间片（time slice）。其不可能导致饥饿现象。
- 多级队列调度：每个作业归入一个队列，不同队列可有不同的优先级、时间片长度、调度策略等。多级反馈队列算法允许进程在队列中迁移，把占用 CPU 多的进程移动到低优先级的队列，能够阻止饥饿的发生。

Chapter 6

- 临界区解决方案必须满足：①互斥（mutual exclusion）②空闲让进（progress）：如果没有进程在临界区中执行，并且有进程需要进入临界区，那么只有那些不在剩余区内执行的进程可以参加选择，以便确定谁能下次进入临界区，而且这种选择不能无限推迟③有限等待（bounded waiting）：从一个进程做出进入临界区的请求直到这个请求允许为止，其他进程进入临界区的次数具有上限。
- Peterson Solution: 不会产生死锁，也不会产生饥饿现象
  Algorithm1: 在 Pi 出让临界区之后，Pj 使用临界区之前，Pi 不可能再次使用临界区（因为必须在 Pj 使用后 turn 才会改变）
  Algorithm2: 当 P0 执行了 flag[0] = true 后， 然后 P1 执行了 flag[1] = true，这样两个进程都无法进入临界区
  Algorithm 2-1:两个进程可能同时进入临界区，当 flag[0]=flag[1]=false 且 P0 执行了第一行后。
- Bakery Algorithm: Before entering its critical section, process receives a number. Holder of the smallest number enters the critical section. If processes Pi and Pj receive the same number, if i < j, then Pi is served first; else Pj is served first. 解决了饥饿问题。
  choosing[i]=true，表示进程 i 正在获取它的排队登记号；
  number[i]是进程 i 的当前排队登记号。如果值为 0，表示进程 i 未参加排队，不想获得该资源。
- Synchronization Hardware: ①TestAndSet ②Swap
- Semaphores: 当进程发现信号量不为正时，选择阻塞自己而不是忙等待（让权等待！）。 wait-P signal-V
- Bounded-Buffer Problem：假设缓冲池有 n 个缓冲区，mutex=1; empty=n; full=0;
- Reader-Writer Problem: rw_mutex = 1; mutex = 1; read_count = 0;
- Dining-Philosophers Problem：chopstick[i]=1
- 自旋锁：当一个 进程在临界区中，其他进程在进入临界区时必须连续循环地调用 acquire()，会产生忙等待，但是不需要上下文切换，通常用于多处理器系统。

Chapter 7

- 死锁必要条件：①mutual exclusion②hold and wait③no preemption④circular wait 死锁预防指的是打破死锁的 4 个条件之一（条件①不可打破）
- 系统资源分配图：申请边 Pi→Rj， 分配边 Rj→Pi。如果分配图没有环就没有死锁，如果有环不一定有死锁，如果环+每个资源类型刚好有一个实例则死锁。
- 只有存在一个安全序列，系统才处于安全状态，非安全状态不一定导致死锁
- （死锁避免）资源分配图算法：假设 Pi 申请 Rj，只有将申请边变成分配边并且不会导致分配图形成环时，才允许分配。
- （死锁避免）银行家算法

Chapter 8 & Chapter 9

- 逻辑地址：程序在编译后，每个目标模块都从 0 号单元开始编址，该相对地址称为逻辑地址；线性地址：逻辑地址加上相应段的基地址，若没有启用分页机制，那么线性地址直接就是物理地址，如果启用了分页机制，则需要再变换一次产生物理地址。
- 地址绑定：源程序→编译→链接（形成逻辑地址）→加载（静态重定位）→执行（动态重定位）
- 静态重定位：作业装入内存时，必须一次性分配给要求的全部内存空间，一旦进入内存，整个运行期间就不能在内存中移动。对重定位的存储管理方式，整个系统只需一个重定位寄存器。
- 分区存储管理：固定分区（fixed partition）可以采用静态重定位，不会产生外部碎片（external fragmentation），会产生内部碎片；可变分区（variable

partition），不会产生内部碎片（internal fragmentation），会产生外部碎片。

- 分段式存储管理：在编译用户程序时，编译器会根据程序自动构造段，有利于程序的动态链接；会产生外部碎片，不会产生内部碎片；段中可重入代码可以共享。
- 分页式存储管理：会产生内部碎片，不会产生外部碎片，但是每个进程平均只产生半个块大小的内部碎片；页表中引入有效位（valid bit）和脏位（dirty bit）
- TLB 有效内存访问时间EAT = p * ma + (1 − p)(2ma)
- 缺页处理：①Operating system looks at another table to decide:(1)Invalid reference→abort (2)Just not in memory ②Get empty frame ③Swap page into frame④Reset tables ⑤Set validation bit = v ⑥Restart the instruction that caused the page fault
- Page Fault Rate = p. EAT = (1 − p) * ma + p * (缺页错误时间)
- Copy-on-Write allows both parent and child processes to initially share the same pages in memory until either process modifies. More efficient process creation.
- First-In-First-Out Algorithm：use a FIFO queue to hold all pages in memory. When a page is brought into memory, insert it at the tail. When a free frame is needed, we replace the page at the queue. **Belady's Anomaly**
- Optimal Page Replacement OPT/MIN：replace the page that won't be used for the longest period of time
- Least Recently Used： associates with each page the time of that page's last use. When a page must be replaced, choose the page that has not been used for the longest period of time.
- LRU-Approximation Page Replacement: Reference Bit - with each page associate a bit, initially = 0, when page is referenced, bit set to 1.
  ①Additional Reference Bits：Use right-shift history byte for each page, current reference bit shift into the left-most bit. choose the page with lowest number.
  ②Second-Chance Algorithm(Clock Algorithm)：FIFO + reference bit. Circular Queue. Inspect the current frame, if the reference bit is set, reset it, and skip to next frame; otherwise, replace it.
  ③Enhanced Second-Chance Algorithm：使用有序对(引用位,修改位)，淘汰的顺序为(0,0) (0,1) (1,0) (1,1)
- Global Replacement: process selects from the set of all frames. High priority process can take a frame from a lower priority process. Results in greater system throughput. Problem: a process cannot control its own page-fault rate.
- Local Replacement: process selects from only its own set of allocated frames. the number of frames allocated to a process does not change.
- 组合方式：固定分配-局部置换，可变分配-局部/全局置换
- Buddy System:Allocates memory from fixed-size segment consisting of physically contiguous pages.
- Slab Allocator: Slab is one or more physically contiguous pages. Cache consists of one or more slabs. Single cache for each unique kernel data structure. When cache created, filled with objects marked as free. When structures stored, objects marked as used. If slab is full of used objects, next object allocated from empty slab. If no empty slabs, new slab allocated. Benefits include no fragmentation, fast memory request satisfaction.
- 分段：是变化大小的。每个段和逻辑单元有关，大小不同，并非连续。有相应的段表，奔腾当中分为 13 位 selector，1 位 global/local,2 位 protection，剩下是 offset。（32 位）
- 如何操作：段号来了，在段表中找到项，拿 base 和 offset 拼起来，用分段变线性，分页变物理地址。包括 linear address 的定义。分页机制中有两层分页，前一层有 page directory，分别为 10,10,12。

Chapter 10 ~12

- 文件属性：Name;Identifie;Type;Location;Size;Protection;Time, date, and user identification;
- 每个打开的文件具有如下信息：
  - File pointer: 文件指针对操作文件的每个进程是唯一的
  - File-open count
  - Disk location of the file: cache of data access information
  - Access rights: per-process access mode information
- 访问方法：Sequential Access; Direct Access; Indexed Sequential-Acess
- 硬链接相当于一个指针，指向文件的索引节点，系统不会增加 inode 节点。而符号链接类似于 Windows 的快捷方式，是不同的文件，数据块中存放原文件的路径。建立符号链接时引用计数直接复制，建立硬链接时，引用计数加 1。
- 逻辑文件系统：管理元数据（指文件系统的所有结构数据），管理目录结构，通过 FCB 来维护文件结构。
- 文件组织模块：知道文件及其逻辑块和物理块，空闲空间管理器。
- 基本文件系统：向合适的设备驱动程序发送一般命令就可对磁盘上的物理块进行读写。
- 打开文件的过程：系统调用 open 将文件名传递到逻辑系统，搜索系统范围内的打开文件表以确定某文件是否已被其他进程所使用。如果是，就在单个进程的打开文件表中创建一项，并指向现有系统范围的打开文件表。如果否，根据给定的文件名搜索目录结构，部分目录结构通常缓存在内存中以加快目录操作找到文件后，其 FCB 会复制到内存的整个系统开放文件表中。
- VFS： Provides an object-oriented way of implementing file systems. Allows the same system call interface(API) to be used for different types of file systems. The API is to the VFS interface, rather than any specific type of file system.Three Layers: ①file-system interface – based on open, close, read, write and on file descriptors. ②virtual file system.
- 连续分配(Contiguous allocation): 所需寻道时间最短，可随机访问，但是浪费空间，且文件不能增长。访问第 n 条记录需要访问 1 次磁盘。
- 链接分配(Linked allocation): 每个目录条目都有文件首个磁盘块的指针，没有空间浪费，但只能顺序访问文件且可能丢失指针。访问第 n 条记录需要访问 n 次磁盘。
- 索引分配(Indexed allocation): 将所有的指针放在索引块中，索引块的第 i 个条目指向文件的第 i 个块。目录包含索引块的地址。
  - 链接索引(Linked scheme)，索引块最后一个指针指向下一个索引块
  - 多级索引(Multi-level index)，以二级索引为例，一级索引块中每一个指针对应的二级索引块。m 级索引需要访问 m+1 次磁盘。
  - 组合方案：UNIX 索引块的前几个指针存在 iNode（索引结点）中，前 12 个指针指向直接块，接下来 3 个指针分别指向一级、二级、三级索引块。
- 位向量(bit map)：块空闲为 1，已分配为 0。按字查找第一个空闲块号公式为(每个字的位数)*(字数)+第一个值为 1 的偏移。盘块分配时，盘块号 b=n*(row-1)+col，盘块回收时 row=(b-1)/n+1, col=(b-1)%n+1
- 磁盘调度算法：①FCFS（不会饥饿）②SSTF（最短寻道时间优先）：处理距离当前磁头位置最短寻道时间的请求③SCAN：从磁臂的一端移动到另一端，到达另一端后反转④C-SCAN：到达另一端后直接返回，不处理回程请求⑤LOOK：只移动到一个方向上的最远请求为止
- 低级格式化(Low-level formatting, physical formatting): 将磁盘分成扇区以便控制器读写，每个扇区的数据结构通常由头部、数据区域和尾部组成，头部和尾部通常包括了扇区号、纠错代码。逻辑格式化(Logical formatting):建立文件系统的根目录，将初始文件系统的数据结构存储到磁盘上。
- 磁盘 I/O 按块完成，文件系统 I/O 按簇完成。
- BootStrap 程序的作用是从磁盘上调入完整的引导程序（存储在磁盘固定位置上的启动块），具有启动分区的磁盘称为启动磁盘。引导分区(boot partition)包含操作系统和设备驱动程序，win 将引导代码存在磁盘的第一个扇区，称为主引导记录(MBR)，引导首先运行 ROM 内存中的代码，其指示系统从 MBR 中读取引导代码，MBR 包含一个分区表和指示引导的分区的标志，当系统找到引导分区时，读取分区的第一个扇区(boot sector)，并继续余下的引导过程。
- RAID – multiple disk drives provides reliability via redundancy. Increases the mean time to failure.Frequently combined with NVRAM(Non-volatile RAM) to improve write performance. RAID is arranged into six different levels.
  RAID 0: non-redundant striping.
  RAID 1: mirrored disks.
  RAID 2: memory-style error correcting codes.
  RAID 3: bit-interleaved parity.
  RAID 4:block-interleaved parity.
  RAID 5: block-interleaved distributed parity.
  RAID 6: P + Q redundancy.

Chapter 13

- 设备驱动程序为 I/O 子系统提供了统一接口。
- Direct I/O instructions：使用特殊 I/O 指令针对 I/O 端口地址传输一个字节或字。I/O 指令触发总线线路，选择适当设备，并将位移入或移出设备寄存器。
- Memory-mapped I/O：设备控制寄存器被映射到处理器的地址空间，处理器执行 I/O 请求时通过标准数据传输指令读写映射到物理内存的设备控制器。
- I/O 端口通常由 4 个寄存器组成：①data-in register②data-out register③status register④control register
- 轮询（Polling）：①主机重复读取忙位直到清零（忙等待）②主机设置命令寄存器的写位，并写出一个字节到数据输出寄存器③主机设置命令就绪位④当控制器注意到命令就绪位时设置忙位⑤控制器读取命令寄存器，并看到写命令，则从输出寄存区中读取一个字节并准备向设备执行 I/O 操作⑥控制器清楚命令就绪位，清楚状态寄存器的故障位表示 I/O 成功，清除忙位表示完成。
- 设备控制器通过 IRL 发送信号引起中断，CPU 捕获中断并分派到中断处理程序，中断处理程序通过处理设备来清除中断。
- 非屏蔽中断（nonmaskable interrupt）：保留用于诸如不可恢复的内存错误等事件。可屏蔽中断：由 CPU 在执行关键的不可中断的指令序列前加以屏蔽。
- 中断向量：包含专门的中断处理程序的内存地址。 中断优先级：能够使 CPU 延迟处理低优先级中断而不屏蔽所有中断，这也可以让高优先级中断抢占低优先级中断处理。
- DMA：主机将 DMA 命令块写到内存，该块包含传输源地址的指针、目标地址的指针、传输字节数，CPU 将这个命令块的地址写到 DMA 控制器，DMA 控制器继续操作内存总线，将地址放到总线，在没有主 CPU 的帮助下执行传输。
- Block devices（块设备） include disk drives，Character devices （字符设备）include keyboards, mice, serial ports
- 内核与 I/O 有关的服务：I/O scheduling、buffering、caching、spooling（假脱机）、device reservation、and error handling.
- SPOOL（假脱机）：用来保存设备输出的缓冲，这些设备如打印机不能接收交叉的数据流。打印机虽然是独享设备，通过 SPOOL 技术，可以将它改造为一台可供多个用户共享的设备。
- I/O 系统层次：用户程序→系统调用处理程序→设备驱动程序→中断处理程序

==========实验部分==========