基本特征

1. 并发

并发性是指宏观上在一段时间内能同时运行多个程序，而并行性则指同一时刻能运行多个指令。

并行需要硬件支持，如多流水线或者多处理器。

操作系统通过引入进程和线程，使得程序能够并发运行。

2. 共享

共享是指系统中的资源可以供多个并发进程共同使用。

有两种共享方式：互斥共享和同时共享。

互斥共享的资源称为临界资源，例如打印机等，在同一时间只允许一个进程访问，需要用同步机制来实现对临界资源的访问。

3. 虚拟

虚拟技术把一个物理实体转换为多个逻辑实体。

主要有两种虚拟技术：时分复用技术和空分复用技术。例如多个进程能在同一个处理器上并发执行使用了时分复用技术，让每个进程轮流占有处理器，每次只执行一小个时间片并快速切换。

4. 异步

异步是指进程不是一次性执行完毕，而是走走停停，以不可知的速度向前推进。

系统调用

如果一个进程在用户态需要用到操作系统的一些功能，就需要使用系统调用从而陷入内核，由操作系统代为完成。可以由系统调用请求的功能有设备管理、文件管理、进程管理、进程通信、存储器管理等

中断分类

1. 外中断

由 CPU 执行指令以外的事件引起，如 I/O 结束中断，表示设备输入/输出处理已经完成，处理器能够发送下一个输入/输出请求。此外还有时钟中断、控制台中断等。

2. 异常

由 CPU 执行指令的内部事件引起，如非法操作码、地址越界、算术溢出等。

3. 陷入

在用户程序中使用系统调用。

大内核和微内核

1. 大内核

大内核是将操作系统功能作为一个紧密结合的整体放到内核。

由于各模块共享信息，因此有很高的性能。

2. 微内核

由于操作系统不断复杂，因此将一部分操作系统功能移出内核，从而降低内核的复杂性。移出的部分根据分层的原则划分成若干服务，相互独立。

因为需要频繁地在用户态和核心态之间进行切换，所以会有一定的性能损失。

第二章 进程管理

1. 进程

进程是操作系统进行资源分配的基本单位。

进程控制块 (Process Control Block, PCB) 描述进程的基本信息和运行状态，所谓的创建进程和撤销进程，都是指对 PCB 的操作。

2. 线程

一个进程中可以有多个线程，线程是独立调度的基本单位。

同一个进程中的多个线程之间可以并发执行，它们共享进程资源。

3. 区别

拥有资源：进程是资源分配的基本单位，但是线程不拥有资源，线程可以访问隶属进程的资源。

调度：线程是独立调度的基本单位，在同一进程中，线程的切换不会引起进程切换，从一个进程内的线程切换到另一个进程中的线程时，会引起进程切换。

系统开销：由于创建或撤销进程时，系统都要为之分配或回收资源，如内存空间、I/O 设备等，所付出的开销远大于创建或撤销线程时的开销。类似地，在进行进程切换时，涉及当前执行进程 CPU 环境的保存及新调度进程 CPU 环境的设置，而线程切换时只需保存和设置少量寄存器内容，开销很小。

通信方面：进程间通信 (IPC) 需要进程同步和互斥手段的辅助，以保证数据的一致性。而线程间可以通过直接读/写同一进程中的数据段（如全局变量）来进行通信。

进程状态的切换

阻塞状态是缺少需要的资源从而由运行状态转换而来，但是该资源不包括 CPU，缺少 CPU 会让进程从运行态转换为就绪态。

只有就绪态和运行态可以相互转换，其它的都是单向转换。就绪状态的进程通过调度算法从而获得 CPU 时间，转为运行状态；而运行状态的进程，在分配给它的 CPU 时间片用完之后就会转为就绪状态，等待下一次调度。

1. 批处理系统中的调度

1.1 先来先服务

first-come first-serverd（FCFS）。

调度最先进入就绪队列的作业。

有利于长作业，但不利于短作业，因为短作业必须一直等待前面的长作业执行完毕才能执行，而长作业又需要执行很长时间，造成了短作业等待时间过长。

1.2 短作业优先

shortest job first（SJF）。

调度估计运行时间最短的作业。

长作业有可能会饿死，处于一直等待短作业执行完毕的状态。如果一直有短作业到来，那么长作业永远得不到调度。

1.3 最短剩余时间优先

shortest remaining time next（SRTN）。优先完成剩余时间最短的作业。

2. 交互式系统中的调度

2.1 优先级调度

除了可以手动赋予优先权之外，还可以把响应比作为优先权，这种调度方式叫做高响应比优先调度算法。

响应比 = (等待时间 + 要求服务时间) / 要求服务时间 = 响应时间 / 要求服务时间

这种调度算法主要是为了解决 SJF 中长作业可能会饿死的问题，因为随着等待时间的增长，响应比也会越来越高。

2.2 时间片轮转

将所有就绪进程按 FCFS 的原则排成一个队列，每次调度时，把 CPU 分配给队首进程，该进程可以执行一个时间片。当时间片用完时，由计时器发出时钟中断，调度程序便停止该进程的执行，并将它送往就绪队列的末尾，同时继续把 CPU 分配给队首的进程。

时间片轮转算法的效率和时间片的大小有很大关系。因为每次进程切换都要保存进程的信息并且载入新进程的信息，如果时间片太小，进程切换太频繁，在进程切换上就会花过多时间。

2.3 多级反馈队列

如果一个进程需要执行 100 个时间片，如果采用轮转调度算法，那么需要交换 100 次。多级队列是为这种需要连续执行多个时间片的进程考虑，它设置了多个队列，每个队列时间片大小都不同，例如 1,2,4,8,..。进程在第一个队列没执行完，就会被移到下一个队列。这种方式下，之前的进程只需要 7 （包括最初的装入）的交换。

每个队列的优先权也不同，最上面的优先权最高。因此只有上一个队列没有进程在排队，才能调度当前队列上的进程。

3. 实时系统中的调度

实时系统要求一个服务请求在一个确定时间内得到响应。

分为硬实时和软实时，前者必须满足绝对的截止时间，后者可以容忍一定的超时。

进程同步

1. 临界区

对临界资源进行访问的那段代码称为临界区。

为了互斥访问临界资源，每个进程在进入临界区之前，需要先进行检查。

2. 同步与互斥

同步指多个进程按一定顺序执行；

互斥指多个进程在同一时刻只有一个进程能进入临界区。

3. 信号量

信号量（Semaphore） 是一个整型变量，可以对其执行 down 和 up 操作，也就是常见的 P 和 V 操作。

down : 如果信号量大于 0 ，执行 -1 操作；如果信号量等于 0，将进程睡眠，等待信号量大于 0；

up ：对信号量执行 +1 操作，并且唤醒睡眠的进程，让进程完成 down 操作。

down 和 up 操作需要被设计成原语，不可分割，通常的做法是在执行这些操作的时候屏蔽中断。

如果信号量的取值只能为 0 或者 1，那么就成为了 互斥量（Mutex） ，0 表示临界区已经加锁，1 表示临界区解锁。

进程通信

进程通信可以看成是不同进程间的线程通信，对于同一个进程内线程的通信方式，主要使用信号量、条件变量等同步机制。

1. 管道

写进程在管道的尾端写入数据，读进程在管道的首端读出数据。管道提供了简单的流控制机制，进程试图读空管道时，在有数据写入管道前，进程将一直阻塞。同样地，管道已经满时，进程再试图写管道，在其它进程从管道中移走数据之前，写进程将一直阻塞。

Linux 中管道是通过空文件来实现。

管道有三种：

普通管道：有两个限制，一是只能单向传输；二是只能在父子进程之间使用；

流管道：去除第一个限制，支持双向传输；

命名管道：去除第二个限制，可以在不相关进程之间进行通信。

2. 信号量

信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止一个进程正在访问共享资源时，其它进程也访问该资源。主要作为进程间以及同一进程内不同线程之间的同步手段。

3. 消息队列

消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

4. 信号

信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

5. 共享内存

共享内存就是映射一段能被其它进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。

共享内存是最快的 IPC 方式，它是针对其它 IPC 运行效率低而专门设计的。它往往与其它通信机制（如信号量）配合使用，来实现进程间的同步和通信。

6. 套接字

套接字也是一种进程间通信机制，与其它通信机制不同的是，它可用于不同机器间的进程通信。

死锁的必要条件

互斥：每个资源要么已经分配给了一个进程，要么就是可用的。

占有和等待：已经得到了某个资源的进程可以再请求新的资源。

不可抢占：已经分配给一个进程的资源不能强制性地被抢占，它只能被占有它的进程显示地释放。

环路等待：有两个或者两个以上的进程组成一条环路，该环路中的每个进程都在等待下一个进程所占有的资源。

鸵鸟策略

把头埋在沙子里，假装根本没发生问题。

因为解决死锁问题的代价很高，因此鸵鸟策略这种不采取任务措施的方案会获得更高的性能。当发生死锁时不会对用户造成多大影响，或发生死锁的概率很低，可以采用鸵鸟策略。

大多数操作系统，包括 Unix，Linux 和 Windows，处理死锁问题的办法仅仅是忽略它。

死锁预防

在程序运行之前预防发生死锁。

1.破坏互斥条件

例如假脱机打印机技术允许若干个进程同时输出，唯一真正请求物理打印机的进程是打印机守护进程。

2. 破坏占有和等待条件

一种实现方式是规定所有进程在开始执行前请求所需要的全部资源。

3.破坏不可抢占条件

4.破坏环路等待

给资源统一编号，进程只能按编号顺序来请求资源。

虚拟内存

每个程序拥有自己的地址空间，这个地址空间被分割成多个块，每一块称为一页。这些页被映射到物理内存，但不需要映射到连续的物理内存，也不需要所有页都必须在物理内存中。当程序引用到一部分不在物理内存中的地址空间时，由硬件执行必要的映射，将缺失的部分装入物理内存并重新执行失败的指令。

1. 分页

大部分虚拟内存系统都使用分页技术。把由程序产生的地址称为虚拟地址，它们构成了一个虚拟地址空间。例如有一台计算机可以产生 16 位地址，它的虚拟地址空间为 0~64K，然而计算机只有 32KB 的物理内存，因此虽然可以编写 64KB 的程序，但它们不能被完全调入内存运行。

虚拟地址空间划分成固定大小的页，在物理内存中对应的单元称为页框，页和页框大小通常相同，它们之间通过页表进行映射。

程序最开始只将一部分页调入页框中，当程序引用到没有在页框的页时，产生缺页中断，进行页面置换，按一定的原则将一部分页框换出，并将页调入。

2. 分段

分段的做法是把每个表分成段，一个段构成一个独立的地址空间。每个段的长度可以不同，并且可以动态增长。

每个段都需要程序员来划分。

3. 段页式

用分段方法来分配和管理虚拟存储器。程序的地址空间按逻辑单位分成基本独立的段，而每一段有自己的段名，再把每段分成固定大小的若干页。

用分页方法来分配和管理实存。即把整个主存分成与上述页大小相等的存储块，可装入作业的任何一页。

程序对内存的调入或调出是按页进行的，但它又可按段实现共享和保护。

4. 分页与分段区别

对程序员的透明性：分页透明，但是分段需要程序员显示划分每个段。

地址空间的维度：分页是一维地址空间，分段是二维的。

大小是否可以改变：页的大小不可变，段的大小可以动态改变。

出现的原因：分页主要用于实现虚拟内存，从而获得更大的地址空间；分段主要是为了使程序和数据可以被划分为逻辑上独立的地址空间并且有助于共享和保护。