内存泄漏也称作“存储渗漏”，用动态存储分配函数动态开辟的空间，在使用完毕后未释放，结果导致一直占据该内存单元。直到程序结束。（其实说白了就是该内存空间使用完毕之后未回收）即所谓内存泄漏。

●

静态分配资源，所有进程在开始运行之前，一次性地申请其在整个运行过程所需的全部资源。但在分配资源时，只要有一种资源不能满足某进程的要求，即使它所需的其他资源都空闲，也不分配给该进程，而让进程等待。
在进程的等待期间，它并未占有任何资源，摒弃了“保持”条件，避免发生死锁。

●

进程占有的资源	线程占有的资源
地址空间	栈
全局变量	寄存器
打开的文件	状态
子进程	程序计数器
信号量	
账户信息	

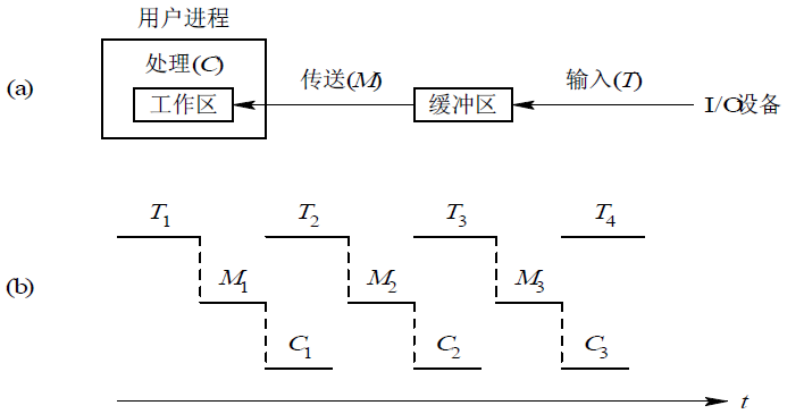
堆：　是大家共有的空间，分全局堆和局部堆。全局堆就是所有没有分配的空间，局部堆就是用户分配的空间。堆在操作系统对进程初始化的时候分配，运行过程中也可以向系统要额外的堆，但是记得用完了要还给操作系统，要不然就是内存泄漏。
栈：是个线程独有的，保存其运行状态和局部自动变量的。栈在线程开始的时候初始化，每个线程的栈互相独立，因此，栈是　thread safe 的。操作系统在切换线程的时候会自动的切换栈，就是切换　ＳＳ／ＥＳＰ寄存器。栈空间不需要在高级语言里面显式的分配和释放。
总结一下确实不会
进程占有的资源：地址空间，全局变量，打开的文件，子进程，信号量，账户信息
线程占有的资源：栈，寄存器，状态，程序计数器
最后关键：线程占有的资源不会共享，所以堆是共享的

●

1.多道批处理系统
在单道批处理系统中，内存中仅有一道作业，它无法充分利用系统中的所有资源，致使系统性能较差。
在多道批处理系统中，用户所提交的作业都先存放在外存上并排成一个队列，称为“后备队列”。然后，由作业调度程序按一定的算法从后备队列中选择若干个作业调入内存，使它们共享 CPU 和系统中的各种资源。
2.分时系统
分时系统与多道批处理系统之间有着截然不同的性能差别，它能很好地将一台计算机提供给多个用户同时使用，提高计算机的利用率。分时系统是指，在一台主机上连接了多个带有显示器和键盘的终端，同时允许多个用户通过自己的终端，以交互方式使用计算机，共享主机中的资源。
3.实时系统
所谓“实时”，是表示“及时”，而实时系统是指系统能及时响应外部事件的请求，在规定的时间内完成对该事件的处理，并控制所有实时任务协调一致的运行。其应用需求主要在实时控制和实时信息处理。

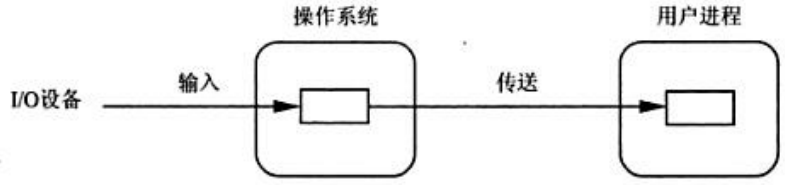
●

所谓多道程序设计指的是允许多个程序同时进入一个计算机系统的 主存储器 并启动进行计算的方法。也就是说，计算机内存中可以同时存放多道（两个以上相互独立的）程序，它们都处于开始和结束之间。从宏观上看是并行的，多道程序 都处于运行中，并且都没有运行结束；从微观上看是串行的，各道程序轮流使用 CPU，交替执行。引入多道程序设计技术 的根本目的是为了提高 CPU 的利用率，充分发挥计算机系统部件的 并行性 ，现代计算机系统都采用了多道程序设计技术。

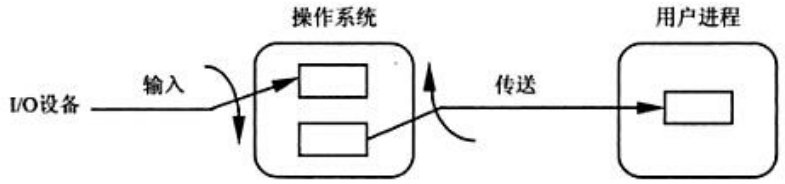


单缓冲是操作系统提供的一种最简单的缓冲形式，如图6.5(a)所示。当用户进程发出一个I/O请求时，操作系统便在内存中为它分配一个缓冲区。由于只设置了一个缓冲区，设备和处理机交换数据时，应先把要交换的数据写入缓冲区，然后，需要数据的设备或处理机从缓冲区取走数据，故设备与处理机对缓冲区的操作是串行的。

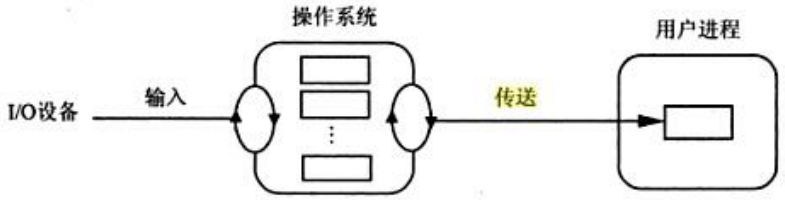
在块设备输入时，先从磁盘把一块数据输入到缓冲区，假设所花费的时间为t；然后由操作系统将缓冲区的数据传送到用户区，假设所花的时间为m；接下来CPU对这一块数据进行计算，假设计算时间为c；则系统对每一块数据的处理时间为max(c,t)+m。通常，m远小于t或c。如果没有缓冲区，数据将直接进入用户区，则每块数据的处理时间为t+c。在块设备输出时，先将要输出的数据从用户区复制到缓冲区，然后再将缓冲区中的数据写到设备。



(a) 单缓冲



(b) 双缓冲



(c) 循环缓冲

图 6.5 输入/输出缓冲方式

运行态：进程占用 CPU，并在 CPU 上运行；
就绪态：进程已经具备运行条件，但是 CPU 还没有分配过来；
阻塞态：进程因等待某件事发生而暂时不能运行； 进程在一生中，都处于上述 3 中状态之一。

运行---》就绪： 时间片用完。
就绪---》运行：运行的进程的时间片用完，调度就转到就绪队列中选择合适的进程分配 CPU
运行---》阻塞：发生了 I/O 请求或等待某件事的发生
阻塞---》就绪:进程所等待的事件发生，就进入就绪队列
P 操作是阻塞作用
V 操作是唤醒作用

●

Linux 进程间通信：管道、信号、消息队列、共享内存、信号量、套接字(socket)
Linux 线程间通信：互斥量（mutex），信号量，条件变量
Windows 进程间通信：管道、消息队列、共享内存、信号量 （semaphore） 、套接字(socket)
Windows 线程间通信：互斥量（mutex），信号量（semaphore）、临界区（critical section）、事件（event）

●

管道(pipe)：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。
有名管道 (named pipe) ： 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。
信号量(semophore) ： 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，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，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。
消息队列(message queue) ： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。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。
信号 (signal) ： 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。
共享内存(shared memory) ：共享内存就是映射一段能被其他进程所访问的内存，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，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号两，配合使用，来实现进程间的同步和通信。
套接字(socket) ： 套解口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同及其间的进程通信。

●

多道批处理主要的特点：
1.系统的吞吐量大
2.资源的利用率高
但由于多道批处理是整批的作业在运行，因此有时候要等很长时间才能运行完，所以效率很重要。
这里需要注意的是：多道批处理还不具有交互性。

●

单道批处理注重顺序性
多到批处理方式为了提高资源利用率和吞吐量，周转时间长，无交互能力
分时系统为了实现人机交互，特点是多路性独立性及时性和交互性
实时系统最明显的特征是实时性和可靠性

●

LRU:最近最少使用淘汰算法.
就像排队，被叫到但是队中没有就缺了，被叫到的队中有则不缺，再把其放在队尾，继续叫号。
缺页为：

4				缺 1
4	3			缺 1
4	3	2		缺 1
4	3	2	1	缺 1
3	2	1	4	不缺
2	1	4	3	不缺
1	4	3	5	缺 1

1	3	5	4	不缺
1	5	4	3	不缺
5	4	3	2	缺 1
4	3	2	1	缺 1
3	2	1	5	缺 1
2	1	5	4	缺 1

●

1.线程作为调度和分配的基本单位，进程作为拥有资源的基本单位
2.不仅进程之间可以并发执行，同一个进程的多个线程之间也可并发执行
3.进程是拥有资源的一个独立单位，线程不拥有系统资源，但可以访问隶属于进程的资源.
4.在创建或撤消进程时，由于系统都要为之分配和回收资源，导致系统的开销明显大于创建或撤消线程时的开销。

●

并发性、共享性、虚拟技术和异步性四大特征 。
最基本特征是并发性。

●

mrb 用于 win 平台
gpt 主要用于 mac（苹果），
MBR 分区表与 GPT 分区表的关系
与支持最大卷为 2 TB（Terabytes）并且每个磁盘最多有 4 个主分区（或 3 个主分区，1 个扩展分区和无限制的逻辑驱动器）的 MBR 磁盘分区的样式相比，GPT 磁盘分区样式支持最大卷为 18 EB（Exabytes）并且每磁盘的分区数没有上限，只受到操作系统限制（由于分区表本身需要占用一定空间，最初规划硬盘分区时，留给分区表的空间决定了最多可以有多少个分区，IA-64 版 Windows 限制最多有 128 个分区，这也是 EFI 标准规定的分区表的最小尺寸）。与 MBR 分区的磁盘不同，至关重要的平台操作数据位于分区，而不是位于非分区或隐藏扇区。另外，GPT 分区磁盘有备份分区表来提高分区数据结构的完整性。

●

虚拟性是 OS 的四大特性之一。如果说可以通过多道程序技术将一台物理 CPU 虚拟为多台逻辑 CPU，从而允许多个用户共享一台主机，那么，通过 SPOOling 技术便可将一台物理 I/O 设备虚拟为多台逻辑 I/O 设备，同样允许多个用户共享一台物理 I/O 设备。通过 SPOOling 技术 可以把一台物理 I/O 设备虚拟为多台逻辑 I/O 设备

●

首次适应算法（First Fit）：
从空闲分区表的第一个表目起查找该表，把最先能够满足要求的空闲区分配给作业，这种方法目的在于减少查找时间。为适应这种算法，空闲分区表（空闲区链）中的空闲分区要按地址由低到高进行排序。该算法优先使用低址部分空闲区，在低址空间造成许多小的空闲区，在高地址空间保留大的空闲区。

●

可重入码：当被多个线程调用的时候，不会引用任何共享数据，他们是线程安全的。（注意与线程安全做区别 可重入是线程安全的真子集）
重入代码(Reentry code)也叫纯代码(Pure code)是一种允许多个进程同时访问的代码。为了使各进程所执行的代码完全相同，故不允许任何进程对其进行修改。程序在运行过程中可以被打断，并由开始处再次执行，并且在合理的范围内（多次重入，而不造成堆栈溢出等其他问题），程序可以在被打断处继续执行，且执行结果不受影响。
可重入码是一种允许多个进程同时访问的代码。为了使各进程访问的代码相同，故不允许对其进行修改！

●

线程，有时被称为轻量级进程(Lightweight Process，LWP），是程序执行流的最小单元。一个标准的线程由线程 ID，当前指令指针(PC），寄存器集合和堆栈组成。另外，线程是进程中的一个实体，是被系统独立调度和分派的基本单位，线程自己不拥有系统资源，只拥有一点儿在运行中必不可少的资源，但它可与同属一个进程的其它线程共享进程所拥有的全部资源。在多线程 OS 中，线程是能独立 运行 的基本单位，因而也是独立调度和分派的基本单位。由于线程很“轻”，故线程的切换非常迅速且开销小（在同一 进程 中的）。一个线程可以创建和撤消另一个线程，

同一 进程中的多个线程之间可以并发执行。由于线程之间的相互制约，致使线程在运行中呈现出间断性。线程也有就绪、阻塞和运行三种基本状态。

1)进程：子进程是父进程的复制品。子进程获得父进程数据空间、堆和栈的复制品。2)线程：相对与进程而言，线程是一个更加接近与执行体的概念，它可以与同进程的其他线程共享数据，但拥有自己的栈空间，拥有独立的执行序列。两者都可以提高程序的并发度，提高程序运行效率和响应时间。线程和进程在使用上各有优缺点：线程执行开销小，但不利于资源管理和保护;而进程正相反。同时，线程适合于在 SMP 机器上运行，而进程则可以跨机器迁移。

是操作系统分配资源的单位；

线程是操作系统执行的单位。

进程有独立的地址空间，一个进程崩溃会，在保护模式下不会对其他进程产生影响，而线程只是一个进程中的不同执行路径。

线程是进程的一个实体，一个进程的多个线程共用这个进程分配到的资源。

主要区别是进程和线程是不同的操作系统资源管理的方式。

线程适合在 SMP 机器（多处理器）上运行，进程可以跨机器迁移。

多进程的程序比多线程的程序健壮。

进程间切换比线程间切换消费的资源多。

●

中断类型分为如下两大类：

一、强迫性中断：正在运行的程序所不期望的，来自硬件故障或外部请求。

1、I/O 中断：来自外部设备通道；

2、程序性中断：运行程序本身的中断，如 溢出、缺页中断、缺段中断、地址越界。

3、时钟中断

4、控制台中断

5、硬件故障

二、自愿性中断：用户在编程时要求操作系统提供的服务，使用访管指令或系统调用使中断发生。也称为访管中断。包括执行 I/O，创建进程，分配内存，信号量操作，发送/接收消息。

●

在分区分配方案中，回收一个分区时有几种不同的邻接情况，在各种情况下应如何处理？ 答：有四种：上邻，下邻，上下相邻，上下不相邻。

（1）回收分区的上邻分区是空闲的，需要将两个相邻的空闲区合并成一个更大的空闲区，然后修改空闲区表。

（2）回收分区的下邻分区是空闲的，需要将两个相邻的空闲区合并成一个更大的空闲区，然后修改空闲区表。

（3）回收分区的上、下邻分区都是空闲的（空闲区个数为 2），需要将三个空闲区合并成一个更大的空闲区（空闲区个数为 1 ），然后修改空闲区表、

（4）回收分区的上、下邻分区都不是空闲的，则直接将空闲区记录在空闲区表中。

系统回收主存时，按道理，空闲区会加 1，但是如果存在以下的两种情况，空闲区个数会有所不同：

（1）.当回收的主存与已有的空闲区存在上邻 或者 下邻的情况，将回收的主存与已有的空闲区合并，空闲区的个数不变；

（2）.当回收的主存与已有的空闲区存在上邻 和 下邻的情况，则回收的主存会将原来的空闲去中的两个空闲区合并成一个空闲区，即回收的主存起到了联通的作用，空闲区的个数不增，反而与之前相比，个数还减少了一个。

●

cpu 工作状态分为系统态（或称管理态，管态）和用户态（或称目态）。 引入这两个工作状态的原因是：为了避免用户程序错误地使用特权指令，保护操作系统不被用户程序破坏。具体规定为：当 cpu 处于用户态时，不允许执行特权指令，当 cpu 处于系统态时，可执行包括特权指令在内的一切机器指令

当 CPU 处于用户态时，不允许执行特权指令。当 CPU 处于系统态时，可执行包括特权指令在内的一切机器指令。

当一个任务（进程）执行系统调用而陷入内核代码中执行时，我们就称进程处于内核运行态（或简称为内核态）。

当进程在执行用户自己的代码时，则称其处于用户运行态（用户态）。

●

作业归还分区，要调整空闲区表，把空闲区表调整成空闲区长度递减的次序排列登记。可变分区分配方式下，当收

回主存时，应检查是否有与归还区相邻的空闲区，若有，则应合并成一个空闲区。
相邻可能有上邻空闲区、下邻空闲区、既上邻又下邻空闲区、既无上邻又无下邻空闲区四种情况。
有上邻空闲区，但无下邻空闲区.只修改上邻空闲区长度（为收回的空闲区长度与原上邻区长度之和），空闲区数不变
无下邻空闲区，但有下邻空闲区.改记录这个下邻空闲区记录的地址为收回空闲区的地址，长度为下邻空闲区的长度和收回空闲区的长度，空闲区数不变
有上邻空闲区，也有下邻空闲区.改记录上邻区记录的长度（为上邻区长度、下邻区长度和收回区长度之和），再把下邻区记录的标志位改为空，即空闲区数-1
无上邻空闲区，也无下邻空闲区.那么找一个标志位为空的记录，记下该回收区的起始地址和长度，且改写相应的标志位为未分配，表明该登记栏中指示了一个空闲区。 空闲区数+1

●

抢占式是优先级高的进程可以直接抢夺优先级低的进程；
如果静态优先权分配，就会导致低优先级的进程一直得不到资源，处于饥饿状态。

●

C 首次适应，是按地址排序，然后依次找到大小满足要求(比需要的大即可)的内存块，划分出一定大小，剩下空闲块的还在链上。
D 最佳适应，空闲块还是从小到大排序，找到大小最接近的内存块。摘链，多出来的再插入链表。
对于选项 C,D，空间地址是递增的，每个空白块的大小不确定，所以这两个算法只需在链表中查找所需大小的空白块，没有好不好实现一说。
对于选项 A，指针大小 4 字节（32 位），和指针所指向的空白块相比，其大小基本可以忽略，就算指针很多，占用了空间，但也说明了空白块也很多，所以对比下，指针所占空间基本可以忽略。
B 选项：在分配空间的时候，会进行空白块的查找。根据算法的不同，欲分配空间大小的不同，其查找的时间总是不容易计算的。

●

唤醒另一进程，说明有进程可被唤醒，就是说有进程在等待进行临界区，若 V 操作前只有一个进程在等待，则 V 操作之后 S=0，若 V 操作前有多个进程在等待，是 V 操作之后 S<0，所以，综合来说，S<=0

●

目录：
什么是 core 文件
如何开启关闭 core file
设置生成目录
如何使用 (gdb -c core)

1. core 文件的简单介绍
在一个程序崩溃时,它一般会在指定目录下生成一个 core 文件。core 文件仅仅是一个内存映象(同时加上调试信息)，主要是用来调试的。
2. 开启或关闭 core 文件的生成
用以下命令来阻止系统生成 core 文件:

```
ulimit -c 0
```

下面的命令可以检查生成 core 文件的选项是否打开:

```
ulimit -a
```

该命令将显示所有的用户定制，其中选项-a 代表“all”。
也可以修改系统文件来调整 core 选项
在/etc/profile 通常会有这样一句话来禁止产生 core 文件，通常这种设置是合理的:

```
# No core files by default
ulimit -S -c 0 > /dev/null 2>&1
```

但是在开发过程中有时为了调试问题，还是需要在特定的用户环境下打开 core 文件产生的设置
在用户的~/.bash_profile 里加上 ulimit -c unlimited 来让特定的用户可以产生 core 文件

如果 ulimit -c 0 则也是禁止产生 core 文件，而 ulimit -c 1024 则限制产生的 core 文件的大小不能超过 1024kb

3. 设置 Core Dump 的核心转储文件目录和命名规则

/proc/sys/kernel/core_uses_pid 可以控制产生的 core 文件的文件名中是否添加 pid 作为扩展，如果添加则文件内容为 1，否则为 0

/proc/sys/kernel/core_pattern 可以设置格式化的 core 文件保存位置或文件名，比如原来文件内容是 core-%e

可以这样修改:

echo "/corefile/core-%e-%p-%t" > /proc/sys/kernel/core_pattern

将会控制所产生的 core 文件会存放到/corefile 目录下，产生的文件名为 core-命令名-pid-时间戳

以下是参数列表:

%p - insert pid into filename 添加 pid

%u - insert current uid into filename 添加当前 uid

%g - insert current gid into filename 添加当前 gid

%s - insert signal that caused the coredump into the filename 添加导致产生 core 的信号

%t - insert UNIX time that the coredump occurred into filename 添加 core 文件生成时的 unix 时间

%h - insert hostname where the coredump happened into filename 添加主机名

%e - insert coredumping executable name into filename 添加命令名

4. 使用 core 文件

在 core 文件所在目录下键入:

gdb -c core

它会启动 GNU 的调试器，来调试 core 文件，并且会显示生成此 core 文件的程序名，中止此程序的信号等等

如果你已经知道是由什么程序生成此 core 文件的，比如 MyServer 崩溃了生成 core.12345，那么用此指令调试:

gdb -c core MyServer

以下怎么办就该去学习 gdb 的使用了

5. 一个小方法来测试产生 core 文件

直接输入指令:

kill -s SIGSEGV $$

6. 为何有时程序 Down 了，却没生成 Core 文件。

Linux 下，有一些设置，标明了 resources available to the shell and to processes。 可以使用

#ulimit -a 来看这些设置。 (ulimit 是 bash built-in Command)

-a All current limits are reported

-c The maximum size of core files created

-d The maximum size of a process 鈥樷 data segment

-e The maximum scheduling priority ("nice")

-f The maximum size of files written by the shell and its children

-i The maximum number of pending signals

-l The maximum size that may be locked into memory

-m The maximum resident set size (has no effect on Linux)

-n The maximum number of open file descriptors (most systems do not allow this value to be set)

-p The pipe size in 512-byte blocks (this may not be set)

-q The maximum number of bytes in POSIX message queues

-r The maximum real-time scheduling priority

-s The maximum stack size

-t The maximum amount of cpu time in seconds

-u The maximum number of processes available to a single user

-v The maximum amount of virtual memory available to the shell

-x The maximum number of file locks

从这里可以看出，如果 -c 是显示：core file size (blocks, -c)

如果这个值为 0，则无法生成 core 文件。所以可以使用：

#ulimit -c 1024 或者 #ulimit -c unlimited 来使能 core 文件。
如果程序出错时生成 Core 文件，则会显示 Segmentation fault (core dumped)。
7. Core Dump 的核心转储文件目录和命名规则:
/proc/sys/kernel/core_uses_pid 可以控制产生的 core 文件的文件名中是否添加 pid 作为扩展，如果添加则文件内容为 1，否则为 0
●
1、并发性：指多个进程实体同存于内存中，且在一段时间内同时运行。并发性是进程的重要特征，同时也成为操作系统的重要特征。
2、动态性：进程的实质是进程实体的一次执行过程，因此，动态性是进程最基本的特征。
3、独立性：进程实体是一个独立运行、独立分配资源和独立接受调度的基本单位。
4、异步性：指进程按各自独立的、不可预知的速度向前推进，或者说实体按异步方式运行。
●
要理解周转时间的含义，所谓的周转时间就是进程从开始到结束所经历的时间；
平均周转时间就是所有进程的周转时间除以进程的个数就 OK 了。
计算方法上面解析的很好。
●
“获得 spinlock ”：自旋锁是为了互斥同步访问的，就是说一个进程想访问一个互斥变量，但是这个变量目前被别人使用，那么自己就执行一个忙循环，也就是执行一个自旋锁
1. 等待状态就是阻塞状态
2. 当进程需要等待某个资源的石宏，进程进入阻塞状态
阻塞态就是等待态 执行态挂起变为就绪态 阻塞态唤醒变为就绪态，阻塞态和就绪态的区别在进程执行资源是否完全满足，满足为就绪，否则执行
●
周转时间：是指，从一个批处理作业提交时刻开始直到该作业完成时刻为止的统计的平均时间
响应时间：是指：作业等待时间+作业计算时间
运行时间：作业计算时间
周转时间：从一个批处理作业提交时刻开始直到该作业完成所经过的时间间隔（包括作业进入内存前的等待时间、在后备队列中的等待时间、占用 CPU 后的运行时间以及完成各种 IO 操作的时间）
响应时间：请求提交给系统到系统响应这个请求的时间间隔
●
在引入线程的操作系统中，线程是进程中的一个实体，是系统独立调度和分派的基本单位。但是线程自己基本上不拥有系统资源，所以它不是资源分配的基本单位，它只拥有一部分在运行中必不可少的与处理机相关的资源，如线程状态、寄存器上下文和栈等，它同样有就绪、阻塞和执行三种基本状态。它可与同属一个进程的其他线程共享进程所拥有的全部资源。一个线程可以创建和撤销另一个线程；同一个进程中的多个线程之间可以并发执行。由于用户线程不依赖于操作系统内核，因此，操作系统内核是不知道用户线程的存在的，用户线程是由用户来管理和调度的，用户利用线程库提供的 API 来创建、同步、调度和管理线程。所以，用户线程的调度在用户程序内部进行，通常采用非抢先式和更简单的规则，也无须用户态和核心态切换，所以速度很快。由于操作系统不知道用户线程的存在，所以，操作系统把 CPU 的时间片分配给用户进程，再由用户进程的管理器将时间分配给用户线程。那么，用户进程能得到的时间片即为所有用户线程共享。
●
在使用锁保证现场安全时可能会出现 活跃度 失败的情况主要包括 饥饿、丢失信号、和活锁、死锁 等。【多线程除了死锁之外遇到最多的就是活跃度问题了】
饥饿 ：指线程需要访问的资源 被永久拒绝 ，以至于不能再继续进行。解决饥饿问题需要平衡线程对资源的竞争，如线程的优先级、任务的权重、执行的周期等。
活锁 ：指线程虽然没有被阻塞，但由于某种条件不满足，一直尝试重试却始终失败。解决活锁问题需要对 重试机制 引入一些随机性。例如如果检测到冲突，那么就暂停随机的一定时间进行重试，这会大大减少碰撞的可能性。
●
我认为，活跃度失败意思就是调用不到了线程了，那么三种都有可能；

死锁也就是互相等着对方释放资源，结果谁也得不到；活锁可能发生让某一个线程一直处于等待状态，其他线程都可以调用到；饥饿我就感觉用抢占式说好说，每次来就执行优先级高的，那么优先级低的可能永远执行不到。

●

顺序执行的程序具有封闭性和可再现性。多道程序具有随机性，结果不可再现

●

同步信号量的初值一般设为 0；
互斥信号量的初值一般设为 1；
同步信号量的用途：防止被抢占 初始为空
低优先级的任务持有信号量，高优先级的任务需要这个信号量，只有当低优先级的任务 give（释放）信号量，高优先级的任务才能 take（获取）信号量。通过这种机制低优先级的任务就可以防止被高优先级的任务抢占。give 和 take 是分别在两个任务里做的。
互斥信号量的用途：对临界区上锁 初始为满
当一个任务想对临界区访问时，为了防止别的任务也对该临界区操作，它需要对该临界区上锁，即 take（获取）一个互斥的信号量，以保证独享。当该任务 take（获取）一个互斥的信号量以后，它仍然能被高优先级的任务抢占，但高优先级的用户仍然无法访问它已经上锁的临界区。而解锁也是由上锁的任务来做的。take 和 give 是在一个任务里完成的。

●

首先我们需要搞懂临界区和临界资源的概念，所谓临界区是指每个进程中访问临界资源的那段代码，临界资源是指一次仅允许一个进程使用的共享资源。通过概念我们可以很清晰的知道，我们可以把临界区想象为一个一个的线程，临界资源就是这一个个线程想要使用的共享资源。一个共享资源是可以对应多个线程的，因此一个临界资源可以对应多个临界区是正确的。
一个临界资源可以对应多个临界区。言下之意就是，存在多个集合，集合内线程对临界资源的使用是互斥的，集合间不互斥。每个集合代表一个临界区，但临界资源只有一个。

●

1、 多道：计算机内存中同时存放几道相对独立的程序。
宏观上并行：同时进入系统的几道程序都处于运行过程中，但都未运行完毕。微观上串行：各道程序轮流的使用 CPU，交替执行。

●

分段机制就是把虚拟地址空间中的虚拟内存组织成一些长度可变的称为段的内存块单元
分页机制把线性地址空间和物理地址空间分别划分为大小相同的块。这样的块称之为页。通过在线性地址空间的页与物理地址空间的页之间建立的映射，分页机制实现线性地址到物理地址的转换
实模式和保护模式是 CPU 的两种工作模式。一开始 PC 启动时 CPU 是工作在实模式下的，经过某种机制后，CPU 跳转到保护模式。其访问空间扩大了，要想从保护模式返回到实模式就只能重启。
当 x86 CPU 工作在保护模式时，可以使用全部 32 根地址线访问 4GB 的内存，因为 80386 的所有通用寄存器都是 32 位的，所以用任何一个通用寄存器来间接寻址，不用分段就可以访问 4G 空间中任意的内存地址。

●

虚存容量受两方面限制：1、指令中表示地址的 字长 ； 2、外存的容量
即取两者中的 MIN

●

A，进程执行完毕，系统需要把 CPU 时间分配给其他进程，引起进程调度
B，进入 IO 请求队列，进程需要暂停，等待 IO 访问结束才能继续执行，是进程调度
C，系统没有能力判断进程是否进入死循环，不会引起进程调度
D，进程调用阻塞原语，则会切换至等待状态，需要进程调度

●

进程被调度程序选中
时间片到
等待某一事件
等待的事件发生

A. 就绪态到执行态
B. 执行态到就绪态
C. 执行态到阻塞态
D. 阻塞态到就绪态

●

inode 是保存文件元信息的区域
一般情况下，文件名和 inode 号码是"一一对应"关系，每个 inode 号码对应一个文件名。但是，Unix/Linux 系统允许，多个文件名指向同一个 inode 号码。
这意味着，可以用不同的文件名访问同样的内容；对文件内容进行修改，会影响到所有文件名；但是，删除一个文件名，不影响另一个文件名的访问。这种情况就被称为"硬链接"（hard link）

●

一般情况下，文件名和 inode 号码是"一一对应"关系，每个 inode 号码对应一个文件名。但是，Unix/Linux 系统允许，多个文件名指向同一个 inode 号码。这意味着，可以用不同的文件名访问同样的内容；对文件内容进行修改，会影响到所有文件名；但是，删除一个文件名，不影响另一个文件名的访问。这种情况就被称为"硬链接"（hard link）。

●

通道是一个独立与 CPU 的专管输入/输出控制的处理机，它控制设备与内存直接进行数据交换。引入通道的目的是让数据的传输独立于 CPU，使 CPU 从繁重的 I/O 工作中解脱出来。它有自己的通道指令，这些指令受 CPU 启动，并在操作结束向 CPU 发出中断信号。通道技术主要是为了减轻 CPU 的工作负担，增加了计算机系统的并行工作程度。
虚拟存储器：它使得应用程序认为它拥有连续的可用的内存，而实际上，它通常是被分隔成多个物理内存碎片，还有部分暂时存储在外部磁盘存储器上，在需要时进行数据交换。
并行技术可以分为多进程编程和多线程编程。通常用 IPC 的形式来实现进程间的同步，如管道，信号量，消息队列或者共享存储。在所有 IPC 敏感词享存储器是最快的。
缓冲技术是为了缓和 CPU 和 I/O 设备速度不匹配的矛盾，提高 CPU 和 I/O 设备的并行性，在现代操作系统中，几乎所以的 I/O 设备在与处理机交换数据时都用了缓存区，并提供获得和释放缓冲区的手段。总的来说，缓冲区技术用到了缓冲区，而缓冲区的引入是为了缓和 CPU 和 I/O 设备速度不匹配，从而可以有效的减少 CPU 的终端频率，提高 CPU 和 I/O 设备的并行性

●

可抢占式调度是严格保证任何时刻，让具有最高优先数（权）的进程占有处理机运行，因此增加了处理机调度的时机，引起为退出处理机的进程保留现场，为占有处理机的进程恢复现场等时间（和空间）开销增大。
抢占式与非抢占式的对比：
非抢占式（Nonpreemptive） 让进程运行直到结束或阻塞的调度方式（容易实现，适合专用系统，不适合通用系统）
抢占式（Preemptive） 允许将逻辑上可继续运行的在运行过程暂停的调度方式，可防止单一进程长时间独占 CPU（系统开销大）

●

管程中的过程是原语操作，不可中断

●

虚拟性是 OS 的四大特性之一。如果说可以通过多道程序技术将一台物理 CPU 虚拟为多台逻辑 CPU，从而允许多个用户共享一台主机，那么，通过 SPOOling 技术便可将一台物理 I/O 设备虚拟为多台逻辑 I/O 设备，同样允许多个用户共享一台物理 I/O 设备。

●

线程共享的内容包括：
进程 代码段
进程 数据段
进程打开的文件描述符、
信号的处理器、
进程的当前目录和
进程用户 ID 与进程组 ID

线程独有的内容包括：

线程 ID

寄存器组的值

线程的堆栈

错误返回码

线程的信号屏蔽码

同一个进程中的线程是共享数据段的，进程是操作系统分配资源的单位，而线程是操作系统真正调度的单位，创建线程时并不会对数据线程进行重新分配。

线程有自己的 threadID、栈和寄存器集合的值，这些在线程上下文切换时，操作系统都会做好相应的保存后才能切换，栈和寄存器的值以及程序计数器的值表明了当前线程的执行过程，对这些进行保存就能保存好执行的现场，以便下次返回时继续执行。

●

当一进程因在记录型信号量 S 上执行 P（S）操作而被阻塞后，S 的值为（）。

＞0

＜0

≥0

≤0

1、若为整型信号量则选 D.

2、若为记录型型号量则选 B.

对于记录型信号量，当 s<0 的时候，请求进程会阻塞

对于整型信号量，当 s<=0 的时候，请求进程不会阻塞，而是进入盲等状态

记录型信号量是不存在“忙等”现象的进程同步机制， 当 S.value<0 时，表示该类资源已分配完毕，因此进程应调用 block 原语，进行自我阻塞，放弃处理机，并插入到该类资源的等待队列 S.L 中

●

用户空间与系统空间所在的内存区间不一样，同样，对于这两种区间，CPU 的运行状态也不一样。 在用户空间中，CPU 处于"用户态"；在系统空间中，CPU 处于"系统态"。

●

用于进程间通讯(IPC)的四种不同技术:

1. 消息传递(管道,FIFO,posix 和 system v 消息队列)

2. 同步(互斥锁,条件变量,读写锁,文件和记录锁,Posix 和 System V 信号灯)

3. 共享内存区(匿名共享内存区,有名 Posix 共享内存区,有名 System V 共享内存区)

4. 过程调用(Solaris 门,Sun RPC)

临界区是指线程的

Linux 下进程间通信的几种主要手段简介：

a)管道（Pipe）：即有名管道（named pipe）：管道可用于具有亲缘关系进程间的通信，有名管道克服了管道没有名字的限制，因此，除具有管道所具有的功能外，它还允许无亲缘关系进程间的通信；

b)信号（Signal）：信号是比较复杂的通信方式，用于通知接受进程有某种事件发生，除了用于进程间通信外，进程还可以发送信号给进程本身；linux 除了支持 Unix 早期信号语义函数 sigal 外，还支持语义符合 Posix.1 标准的信号函数 sigaction（实际上，该函数是基于 BSD 的，BSD 为了实现可靠信号机制，又能够统一对外接口，用 sigaction 函数重新实现了 signal 函数）；

c)Message（消息队列）：消息队列是消息的链接表，包括 Posix 消息队列 system V 消息队列。有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息。消息队列克服了信号承载信息量少，管道只能承载无格式字节流以及缓冲区大小受限等缺点。

d)共享内存：使得多个进程可以访问同一块内存空间，是最快的可用 IPC 形式。是针对其他通信机制运行效率较低而设计的。往往与其它通信机制，如信号量结合使用，来达到进程间的同步及互斥。

e)信号量（semaphore）：主要作为进程间以及同一进程不同线程之间的同步手段。

f)套接口（Socket）：更为一般的进程间通信机制，可用于不同机器之间的进程间通信。起初是由 Unix 系统的 BSD 分支开发出来的，但现在一般可以移植到其它类 Unix 系统上：Linux 和 System V 的变种都支持套接字。

Linux 线程间通信：互斥体，信号量，条件变量
Windows 线程间通信：临界区（Critical Section）、互斥量（Mutex）、信号量（Semaphore）、事件（Event）
Windows 进程间通信：管道、内存共享、消息队列、信号量、socket
Windows 进程和线程共同之处：信号量和消息（事件）
通信，指的进程/线程有交互，可以通过共享资源进行通知或数据传递。临界区则是一种概念，指的是访问公共资源的程序片段，并不是一种通信方式

●

页式-内零头
段式-外零头
操作系统在分配内存时，有时候会产生一些空闲但是无法被正常使用的内存区域，这些就是内存碎片，或者称为内存零头，这些内存零头一共分为两类：内零头和外零头。
内零头是指进程在向操作系统请求内存分配时，系统满足了进程所需要的内存需求后，还额外还多分了一些内存给该进程，也就是说额外多出来的这部分内存归该进程所有，其他进程是无法访问的。
外零头是指内存中存在着一些空闲的内存区域，这些内存区域虽然不归任何进程所有，但是因为内存区域太小，无法满足其他进程所申请的内存大小而形成的内存零头。
页式存储管理是以页为单位（页面的大小由系统确定，且大小是固定的）向进程分配内存的，例如：假设内存总共有 100K,分为 10 页，每页大小为 10K。现在进程 A 提出申请 56K 内存，因为页式存储管理是以页为单位进程内存分配的，所以系统会向进程 A 提供 6 个页面，也就是 60K 的内存空间，那么在最后一页中进程只使用了 6K，从而多出了 4K 的内存碎片，但是这 4K 的内存碎片系统已经分配给进程 A 了，其他进程是无法再访问这些内存区域的，这种内存碎片就是内零头。
段式存储管理是段（段的大小是程序逻辑确定，且大小不是固定的）为单位向进程进行内存分配的，进程申请多少内存，系统就给进程分配多少内存，这样就不会产生内零头，但是段式分配会产生外零头。
例如：假设内存总的大小为 100K，现在进程 A 向系统申请 60K 的内存，系统在满足了进程 A 的内存申请要求后，还剩下 40K 的空闲内存区域；这时如果进程 B 向系统申请 50K 的内存区域，而系统只剩下了 40K 的内存区域，虽然这 40K 的内存区域不归任何进程所有，但是因为大小无法满足进程 B 的要求，所以也无法分配给进程 B，这样就产生了外零头。请求段式存储管理是在段式存储管理的基础上增加了请求调段功能和段置换功能。
所以段式和请求段式存储管理会产生外零头，选 BD。

●

系统中的资源可以分为两类，一类是可剥夺资源，是指某进程在获得这类资源后，该资源可以再被其他进程或系统剥夺。例如，优先权高的进程可以剥夺优先权低的进程的 处理机 。又如，内存区可由 存储器管理 程序，把一个进程从一个存储区移到另一个存储区，此即剥夺了该进程原来占有的存储区，甚至可将一进程从内存调到外存上，可见， CPU 和 主存 均属于可剥夺性资源。另一类资源是不可剥夺资源，当系统把这类资源分配给某进程后，再不能强行收回，只能在进程用完后自行释放，如 磁带机 、打印机等。

●

在页面置换过程中的最糟糕的一种情形是，刚刚换出的页面马上又要换入主存，刚刚换入的页面马上就要换出主存，这种频繁的页面调度行为称为 抖动， 或 颠簸。

最佳置换算法所选择的淘汰页面将是以后永不使用的，或者是在最长时间内不再被访问的页面，这样可以保证最低的缺页率。但是人们目前无法预知进程在内存的若干页面中哪个是未来最长时间不再被访问的，因而该算法无法实现。这样说其完全避免抖动就无从谈起了。

●

UNIX 操作系统（尤尼斯），是一个强大的多用户、多任务 操作系统 ，支持多种 处理器架构 ，按照操作系统的分类，属于 分时操作系统
UNIX 系统是一个多用户，多任务的分时操作系统。
UNIX 的系统结构可分为三部分：操作系统内核（是 UNIX 系统核心管理和控制中心，在系统启动或常驻内存），系统调用（供程序开发者开发应用程序时调用系统组件，包括进程管理，文件管理，设备状态等），应用程序（包括各种开发工具，编译器，网络通讯处理程序等，所有应用程序都在 Shell 的管理和控制下为用户服务）。
UNIX 系统大部分是由 C 语言编写的，这使得系统易读，易修改，易移植。
UNIX 提供了丰富的，精心挑选的系统调用，整个系统的实现十分紧凑，简洁。

UNIX 提供了功能强大的可编程的 Shell 语言（外壳语言）作为用户界面具有简洁，高效的特点。
UNIX 系统采用树状目录结构，具有良好的安全性，保密性和可维护性。
UNIX 系统采用进程对换（Swapping）的内存管理机制和请求调页的存储方式，实现了虚拟内存管理，大大提高了内存的使用效率。
UNIX 系统提供多种通信机制，如：管道通信，软中断通信，消息通信，共享存储器通信，信号灯通信。

●

每个进程都有一个页表。在进程执行过程中，将其调入到内存中。

●

独占设备是不能同时使用的设备，即在一段时间内，该设备只允许一个进程独占，如行式打印机、读卡机、磁带机等

●

物理地址的计算公式为：
物理地址 = 内存块号*块长+页内地址
用户空间为 16 个页面，可知页号部分占 4 位；页长为 1KB，可知页内地址占 10 位。
102B(H) = 00 01 00 00 0010 1011 (b).
页内地址为：00 0010 1011(b).
块号为 2(d) = 0010(b)，块长为页长
最后的物理地址实际上是把内存块号当做高位地址，页内地址当做低位地址
0010 00 0010 1011 = 082B

16 个页面，2 的 4 次方=16，可知页号部分占 4 位，页长为 1k，2 的 10 次方=1k，页面大小（偏移）占 10 位
102B（H）十六进制表示法，B=11，所以 102B（H）=0001 0000 0010 1011
页内地址为：00 0010 1011
页号为：0100=4.查表可知页号 4 对应的块号为 2，
物理地址=物理块号*物理块号大小+页面位移
页号=（int）（逻辑地址/页面大小）
页面位移=逻辑地址%页面大小=0001 0000 0010 1011%100 0000 0000=0010 1011
所以逻辑地址 102B（H）所对应的物理地址=2*1k +0010 1011=1000 0010 1011
转化为 16 进制 082B（H）

●

在请求分页系统中，只要求将当前需要的一部分页面装入内存，便可以启动作业运行。在作业执行过程中，当所要访问的页面不在内存时，再通过调页功能将其调入，同时还可以通过置换功能将暂时不用的页面换出到外存上，以便腾出内存空间。

为了实现请求分页，系统必须提供一定的硬件支持。除了需要一定容量的内存及外存的计算机系统，还需要有页表机制、缺页中断机构和地址变换机构。请求分页系统的页表机制不同于基本分页系统，请求分页系统在一个作业运行之前不要求全部一次性调入内存，因此在作业的运行过程中，必然会出现要访问的页面不在内存的情况，如何发现和处理这种情况是请求分页系统必须解决的两个基本问题。为此，在请求页表项中增加了四个字段：
增加的四个字段说明如下：
状态位 P：用于指示该页是否已调入内存，供程序访问时参考。
访问字段 A：用于记录本页在一段时间内被访问的次数，或记录本页最近已有多长时间未被访问，供置换算法换出页面时参考。
修改位 M：标识该页在调入内存后是否被修改过。
外存地址：用于指出该页在外存上的地址，通常是物理块号，供调入该页时参考。

在请求分页系统中，每当所要访问的页面不在内存时，便产生一个缺页中断，请求操作系统将所缺的页调入内存。此时应将缺页的进程阻塞（调页完成唤醒)，如果内存中有空闲块，则分配一个块，将要调入的页装入该块，并修改页表中相应页表项，若此时内存中没有空闲块，则要淘汰某页（若被淘汰页在内存期间被修改过，则要将其写回外存)。

●

某一磁盘请求序列(磁道号):98、 183、 37、122、14、124、 65、 61，按照先来先服务 FCFS 磁盘调度对磁盘进行请求服务，假设当前磁头在 53 道上，则磁臂总移动道数为多少？

先来先服务 FCFS：按进程请求访问磁盘的先后次序进行调度

当前磁头在 53 道：

下一个磁道	移动磁道数
98	45
183	85
37	146
122	85
14	108
124	110
65	59
61	4

所以总的移动道数：45+85+146+85+108+110+59+4=642

●

信号量

信号量是最早出现的用来解决进程同步与互斥问题的机制。

信号量（Saphore）值表示相应资源的使用情况。信号量 S>=0 时，S 表示可用资源的数量。执行一次 P 操作意味着请求分配一个资源，因此 S 的值减 1；当 S<0 时，表示已经没有可用资源，S 的绝对值表示当前等待该资源的进程数。请求者必须等待其他进程释放该类资源，才能继续运行。而执行一个 V 操作意味着释放一个资源，因此 S 的值加 1；若 S<0，表示有某些进程正在等待该资源，因此要唤醒一个等待状态的进程，使之运行下去。

注意，信号量的值只能由 PV 操作来改变。

●

i++在两个线程里边分别执行 100 次，能得到的最大值和最小值分别是多少？

i++只需要执行一条指令，并不能保证多个线程 i++，操作同一个 i，可以得到正确的结果。因为还有寄存器的因素，多个 cpu 对应多个寄存器。每次要先把 i 从内存复制到寄存器，然后++，然后再把 i 复制到内存中，这需要至少 3 步。从这个意义上讲，说 i++是原子的并不对。

如此，假设两个线程的执行步骤如下：

1. 线程 A 执行第一次 i++，取出内存中的 i，值为 0，存放到寄存器后执行加 1，此时 CPU1 的寄存器中值为 1，内存中为 0；
2. 线程 B 执行第一次 i++，取出内存中的 i，值为 0，存放到寄存器后执行加 1，此时 CPU2 的寄存器中值为 1，内存中为 0；
3. 线程 A 继续执行完成第 99 次 i++，并把值放回内存，此时 CPU1 中寄存器的值为 99，内存中为 99；
4. 线程 B 继续执行第一次 i++，将其值放回内存，此时 CPU2 中的寄存器值为 1，内存中为 1；
5. 线程 A 执行第 100 次 i++，将内存中的值取回 CPU1 的寄存器，并执行加 1，此时 CPU1 的寄存器中的值为 2，内存中为 1；
6. 线程 B 执行完所有操作，并将其放回内存，此时 CPU2 的寄存器值为 100，内存中为 100；
7. 线程 A 执行 100 次操作的最后一部分，将 CPU1 中的寄存器值放回内存，内存中值为 2；
8. 结束！

所以该题目便可以得出最终结果，最小值为 2，最大值为 200。

●

首先介绍下动态重定位装入方式：

其运行环境：多道程序环境；

程序在运行过程中在内存的位置可能变动，装入程序把装入模块装入内存后，并不立即把装入模块中的相对地址转换为绝对地址，而是把这种地址转换推迟到程序真正执行时才进行。说白了，动态重定位装入方式，是在程序执行时由 CPU 硬件进行地址重定位。

特点：程序在内存中可以浮动，不要求整个应用程序占用连续控件；为使地址转换不影响指令的执行速度，需要一个重定位寄存器的支持。

可重定位装入方式：
运行环境：多道程序环境
程序目标模块的起始地址通常是从 0 开始的，程序中的其他地址也都是相对于起始地址计算的；根据内存的当前情况，将装入模块装入到内幕才能的适当位子；地址变换通常是装入时一次完成的，以后不再改变，所以是静态重定位。
特点：无需硬件支持；程序不能在内存中移动；要求程序的存储空间是连续的，不能把程序放在若干不连续区域

绝对装入方式：
环境：使用单批道程序环境
绝对装入需要实现知道程序驻留在内存的位置，程序按照装入模块中的地址，将程序和数据装入内存。所以程序中的逻辑地址与实际地址完全相同，当操作系统吧程序装入内存时，不需要对程序和数据进行地址修改；
特点：是 CPU 执行目标代码块，由于内存大小的限制，能装入内存冰法执行的进程数大大减少。

下面有关 raid 的工作模式，说法正确的有？
raid0：把要存放的数据分为很多部分分别存在 raid0 的各个硬盘中，不做备份
raid1：把用户写入硬盘的数据百分之百地自动复制到另外一个硬盘上
raid5: 把数据和相对应的奇偶校验信息存储到组成 RAID5 的各个磁盘上
raid0 至少需要 1 块硬盘，raid1 至少需要 2 块硬盘

A 正确
B 正确
C 正确 RAID 5 不对存储的数据进行备份，而是把数据和相对应的奇偶校验信息存储到组成 RAID5 的各个磁盘上，并且奇偶校验信息和相对应的数据分别存储于不同的磁盘上。
D 错误，raid0 至少需要 2 块盘
RAID 0：无差错控制的带区组
要实现 RAID0 必须要有两个以上硬盘驱动器，RAID0 实现了带区组，数据并不是保存在一个硬盘上，而是分成数据块保存在不同驱动器上。在所有的级别中，RAID 0 的速度是最快的。但是 RAID 0 没有冗余功能的，如果一个磁盘(物理)损坏，则所有的数据都无法使用。
RAID 1：镜象结构
当主硬盘损坏时，镜像硬盘就可以代替主硬盘工作。镜像硬盘相当于一个备份盘，可想而知，这种硬盘模式的安全性是非常高的，RAID 1 的数据安全性在所有的 RAID 级别上来说是最好的。但是其磁盘的利用率却只有 50%，是所有 RAID 级别中最低的。
RAID5：分布式奇偶校验的独立磁盘结构
RAID5 最大的好处是在一块盘掉线的情况下，RAID 照常工作，相对于 RAID0 必须每一块盘都正常才可以正常工作的状况容错性能好多了。因此 RAID5 是 RAID 级别中最常见的一个类型。RAID5 校验位即 P 位是通过其它条带数据做异或(xor)求得的。计算公式为 P=D0xorD1xorD2…xorDn，其中 p 代表校验块，Dn 代表相应的数据块，xor 是数学运算符号异或。
RAID10：高可靠性与高效磁盘结构
RAID 10 是先镜射再分区数据。是将所有硬盘分为两组，视为是 RAID 0 的最低组合，然后将这两组各自视为 RAID 1 运作。RAID 10 有着不错的读取速度，而且拥有比 RAID 0 更高的数据保护性。
●
raid0 也至少要两个磁盘。
●
UNIX 中有如下的通信方式：
1) 文件和记录锁定。

为避免两个进程间同时要求访问同一共享资源而引起访问和操作的混乱，在进程对共享资源进行访问前必须对其进行锁定，该进程访问完后再释放。这是 UNIX 为共享资源提供的互斥性保障。

2 ）管道。

管道一般用于两个不同进程之间的通信。当一个进程创建一个管道，并调用 fork 创建自己的一个子进程后，父进程关闭读管道端，子进程关闭写管道端，这样 提供了两个进程之间数据流动的一种方式。

3 ） FIFO 。

FIFO 是一种先进先出的队列。它类似于一个管道，只允许数据的单向流动。每个 FIFO 都有一个名字，允许不相关的进程访问同一个 FIFO。因此也成为命名管。

4）消息队列。

UNIX 下不同进程之间可实现共享资源的一种机制；UNIX 允许不同进程将格式化的数据流以消息形式发送给任意

进程。对消息队列具有操作权限的进程都可以使用 msget 完成对消息队列的操作控制。通过使用消息类型，进程可以按任何顺序读消息，或为消息安排优先级顺序。

5 ）信号灯。

作为进程间通讯的一种方法，它不是用于交换大批数据，而用于多进程之间的同步（协调对共享存储段的存取）。

6 ）共享内存。

通过信号灯实现存储共享（类似“红灯停、绿灯行”）

●

关中断 ：首先要知道中断是指当出现需要时，CPU 暂时停止当前程序的执行转而执行处理新情况的程序和执行过程。即在程序运行过程中，系统出现了一个必须由 CPU 立即处理的情况，此时，CPU 暂时中止程序的执行转而处理这个新的情况的过程就叫做中断。

而关中断是指在此中断处理 完成前，不处理其它中断。

●

os 调度的基本单位，线程在进程内部运行，线程是进程的一部分 （和其他线程是共享地址空间的）

●

常见进程间通信方式的比较：

管道：速度慢，容量有限

消息队列：容量受到系统限制，且要注意第一次读的时候，要考虑上一次没有读完数据的问题。

信号量：不能传递复杂消息，只能用来同步

共享内存区：能够很容易控制容量，速度快 ，但要保持同步，比如一个进程在写的时候，另一个进程要注意读写的问题，相当于线程中的线程安全，当然，共享内存区同样可以用作线程间通讯，不过没这个必要，线程间本来就已经共享了一块内存的。

●

进程间通讯的方式：

管道中还有命名管道和非命名管道之分，非命名管道只能用于父子进程通讯，命名管道可用于非父子进程，命名管道就是 FIFO，管道是先进先出的通讯方式。FIFO 是一种先进先出的队列。它类似于一个管道，只允许数据的单向流动。每个 FIFO 都有一个名字，允许不相关的进程访问同一个 FIFO，因此也成为命名管。

消息队列：是用于两个进程之间的通讯，首先在一个进程中创建一个消息队列，然后再往消息队列中写数据，而另一个进程则从那个消息队列中取数据。需要注意的是，消息队列是用创建文件的方式建立的，如果一个进程向某个消息队列中写入了数据之后，另一个进程并没有取出数据，即使向消息队列中写数据的进程已经结束，保存在消息队列中的数据并没有消失，也就是说下次再从这个消息队列读数据的时候，就是上次的数据!!!

信号量， 不能传递复杂消息，只能用来同步

共享内存，只要首先创建一个共享内存区，其它进程按照一定的步骤就能访问到这个共享内存区中的数据，当然可读可写；

几种方式的比较：

管道：速度慢，容量有限
消息队列：容量受到系统限制，且要注意第一次读的时候，要考虑上一次没有读完数据的问题。
信号量：不能传递复杂消息，只能用来同步
共享内存区：能够很容易控制容量，速度快，但要保持同步，比如一个进程在写的时候，另一个进程要注意读写的问题，相当于线程中的线程安全，当然，共享内存区同样可以用作线程间通讯，不过没这个必要，线程间本来就已经共享了一块内存的。

●

缓冲区有两块:高速缓存区（物理存在）和磁盘缓存区（逻辑存在，实际是内存一块），都不在外存（硬盘）

●

后调用的程序可能覆盖之前程序使用的内存，从而导致前面程序访问内存错误。所以用绝对地址编写的程序不适合多道程序系统运行

●

linux 提供了一种机制可以保证只要父进程想知道子进程结束时的状态信息， 就可以得到。这种机制就是: 在每个进程退出的时候,内核释放该进程所有的资源,包括打开的文件,占用的内存等。 但是仍然为其保留一定的信息(包括进程号 the process ID,退出状态 the termination status of the process,运行时间 the amount of CPU time taken by the process 等)。直到父进程通过 wait / waitpid 来取时才释放。

孤儿进程：一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程。孤儿进程将被 init 进程(进程号为 1)所收养，并由 init 进程对它们完成状态收集工作。

僵尸进程：一个进程使用 fork 创建子进程，如果子进程退出，而父进程并没有调用 wait 或 waitpid 获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵死进程。

僵死进程并不是问题的根源，罪魁祸首是产生出大量僵死进程的那个父进程，所以，解决方法就是 kill 那个父进程，于是僵尸进程就可以被 init 进程接收，释放。

●

死锁的 4 个条件
1.不可剥夺
2.互斥
3.请求和保持
4.环路等待
破坏一个即可破坏死锁

●

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动, 而程序是静态的

●

某系统有 n 台互斥使用的同类设备,3 个并发进程需要 3,4,5 台设备,可确保系统不发生死锁的设备数 n 最小为()
假设 3 个进程分别为 A,B,C,那么需要的最大的情况为:2,3,4 此时再多一个资源就可以打破死锁的环境,所以为 2+3+4+1=10

●

1、主存与辅存的区别：
概括的说，CPU 对所需要的数据进行计算时，要求很高的存储速度，且不需要能永久保存这些数据，高速存储设备的成本很高。
但其他设备对存储速度的要求不像 CPU 这么高，一般要求永久保存数据。一般低速的存储设备就可以满足，且低速的存储成本也低。
所以有主存和辅存之分：
内存（主存）直接给 CPU 提供存储，高速，低容量，价格贵，不能永久保存数据，断电消失，需要从辅存中重新调入数据。
外存（辅存）给主存提供数据，低速，大容量，价格低，能永久保存数据。
所以更高缓存的 CPU 和更大的内存能够大大提升系统的性能。

常见主存有：CPU 的高速缓存，电脑的内存条。
常见辅存有：硬盘、光盘、U 盘、磁盘、移动硬盘等等。
2、什么是虚拟存储：
根据程序执行的互斥性和局部性两个特点，我们允许作业装入的时候只装入一部分，另一部分放在磁盘上，当需要的时候再装入到主存，这样以来，在一个小的主存空间就可以运行一个比它大的作业。同时，用户编程的时候也摆脱了一定要编写小于主存容量的作业的限制。也就是说，用户的逻辑地址空间可以比主存的绝对地址空间要大。对用户来说，好像计算机系统具有一个容量很大的主存储器，称为“虚拟存储器”。
虚拟存储（Storage Virtualization）是指将多个不同类型、独立存在的物理存储体，通过软、硬件技术，集成转化为一个逻辑上的虚拟的存储单元，集中管理供用户统一使用。这个虚拟逻辑存储单元的存储容量是它所集中管理的各物理存储体的存储量的总和，而它具有的访问带宽则在一定程度上接近各个物理存储体的访问带宽之和。

●

可重定位分区分配 可以重整内存，将碎片内存进行“搬移”，将小块变为大块，这样就类似连续空间了直接被利用了。
从而也解决了“碎片”问题

●

A 错，进程拥有独立的地址空间；B 错，主线程和子线程是并行关系的时候，并没有依赖关系。父进程和子进程中，子进程是父进程的一个副本，创建子进程后，子进程会有自己的空间，然后把父进程的数据拷贝到子进程的空间里。运行时，谁先运行是不确定的，这由系统决定；C 错，多线程和多进程都会引起死锁，一般说的死锁指的是进程间的死锁。所以选 D
进程和线程的主要差别在于它们是不同的操作系统资源管理方式。进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程序健壮，但在进程切换时，耗费资源较大，效率要差一些。但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程。如果有兴趣深入的话，我建议你们看看《现代操作系统》或者《操作系统的设计与实现》。对就个问题说得比较清楚。

●

临界资源是指每次仅允许一个进程访问的资源。
属于临界资源的硬件有打印机、磁带机等,软件有消息缓冲队列、变量、数组、缓冲区等。 诸进程间应采取互斥方式，实现对这种资源的共享。

●

临界资源是针对共享资源来说的。 属于临界资源的硬件，比如，打印机，磁带机。 属于临界资源的软件有，缓冲队列，缓冲区等。

●

一部分存在磁盘上等待，一部分在主存上。 一旦发生缺页中断，先从主存里面找，有就直接置换，没有再从磁盘把缺失的页面置换到主存中。
因为进程访问具有局部性，所以同一个进程访问的页面都是来来回回都是那几个。磁盘调用发生的频率很低，所以不需要担心访问速度会变慢

●

cache，是内存高速缓存，而页表使用的缓存是特殊高速缓存，叫 TLB。
虚存访问时首先去 TLB 中查找相应页表项，找到对应实地址。
如果没有找到页表项，根据页号使用普通方法找实地址，而此时的页表可能庞大，所以会
采取虚存保存页表，即 B 选项。
当然也会有部分页表在内存中，即 C 选项。

●

按组织形式和处理方式可将文件分为以下几类：
1.普通文件：有 ASCII 码或二进制码组成的字符文件。
2.目录文件：由文件目录组成的，用来管理和实现文件系统功能的系统文件，通过目录文件可以对其他文件的信息进行检索。

3.特殊文件：特指系统中的各类 I/O 设备。

p280《新编操作系统习题与解析》 李春葆 清华大学出版社

●

原语是由若干个 机器指令 构成的完成某种特定功能的一段程序，具有不可分割性 • 即原语的执行必须是连续的，在执行过程中不允许被中断。

原语通常由若干条指令组成，用来实现某个特定的操作。通过一段不可分割的或不可中断的程序实现其功能。原语是操作系统的核心，它不是由进程而是由一组程序模块所组成，是操作系统的一个组成部分，它必须在管态(一种机器状态，管态下执行的程序可以执行特权和非特权两类指令，通常把它定义为操作系统的状态)下执行，并且常驻内存，而个别系统有一部分不在管态下运行。原语和广义指令都可以被进程所调用，两者的差别在于原语有不可中断性，它是通过在执行过程中关闭中断实现的，且一般由系统进程调用。许多广义指令的功能都可用目态(一种机器状态，通常把它作为用户程序执行时的状态)下运行的系统进程完成，而不一定要在管态下完成，例如文件的建立、打开、关闭、删除等广义指令，都是借助中断进入管态程序，然后转交给相应的进程，最终由进程实现其功能。引进原语的主要目的是为了实现进程的通信和控制。

●

虚拟存储器的最大容量 = min(内存+外存，2^n)。n 为计算机的地址总线位数。

虚存容量不是无限的，最大容量受内存和外存可利用的总容量限制,虚存实际容量受计算机总线地址结构限制

所谓虚拟存储器， 是指具有请求调入功能和置换功能， 能从逻辑上对内存容量加以扩充的一种存储器系统。 其逻辑容量由内存容量和外存容量之和所决定 ，其运行速度接近于内存速度，而每位的成本却又接近于外存。可见，虚拟存储技术是一种性能非常优越的存储器管理技术，故被广泛地应用于大、 中、 小型机器和微型机中。

●

错了好几次，终于明白了这道题的意思。

题目中物理内存是指实际的内存，而虚拟内存是指磁盘交换区，并不是真正的虚拟内存的意思。

所以对于页面置换算法来讲，当发生缺页中断时，都是要从内存中找到一个不需要的块换出去（对应物理内存的释放），然后将需要页面从磁盘的交换区中换进来（虚拟内存的分配）。

●

1. 需要页表，建立进程中的块（页）-》 内存中的块（页框）的映射关系；

2. “分页式存储管理是以块为单位进行存储分配的，每块的尺寸相同，在有存储请求时，只要系统中有足够的空闲块存在，就可以进行分配，分配哪一块都一样，无好坏之分”所以还需要 采用“存储分块表”和“位示图”的管理方法

●

固态硬盘相比于传统机械硬盘，没有旋转的盘片，寻址时不需要盘片转动和磁头移动，

也就是寻址速度远高于传统硬盘，表现在随机读取速度上远快于普通硬盘

●

与传统硬盘相比，固态硬盘具有“快速”、“轻量”、“耐震”、“省电”等特点！

1、固态硬盘不存在磁盘及磁头，控制器芯片事先掌握数据的存储位置，能够直接通过电信号来存取数据，数据读取速度更快。

2、 与传统硬盘相比，固态硬盘的构成部件较少，因此可实现轻量和小型化。

3、固态硬盘不存在磁头和磁盘，不会像传统硬盘那样出现磁头与磁盘相接触而发生碰撞的现象。因此，出现冲击及震动而导致数据损坏的可能性很小。

4、 采用固态硬盘减少电力消耗，实现更长使用时间。一般情况下，与搭配传统硬盘的电脑型号相比，运行时间会有一定延长

●

虚拟存储的实现式基于程序局部性原理，其实质是借助外存将内存较小的物理地址空间转化为较大的逻辑地址空间。

●

特权指令是作用于操作系统系统态的。这类指令只用于操作系统或其他系统软件，一般不直接提供给用户使用。

在管态的时候可以使用所有指令包括特权指令。目态可以使用除特权指令之外的所有指令。

●

SPOOLing 技术，大意是在独占设备（如打印机）内设置缓存。当有进程访问时，不是占有这台设备，而是在缓存中

为它分配一定的空间（进行排队），从而达到多进程共享设备的效果。
个人认为，题目的前半句正确，在不考虑 SPOOLing 等各种技术的前提下，后半句也正确。
然而随着科技发展，不可能变为可能。下面这句话是不是顺眼多了
尽管独占设备在一段时间内只允许一个进程使用，然而，多个并发进程仍然可以同时使用这类设备。

●

我们熟知的 Windows XP、Linux、Mac OS X 等都是多用户多任务分时操作系统，可见这个概念一直延续到了今天。它们最显著的特点就是可以让多个人使用同一台电脑而且不能互相窥探对方的秘密。当你使用电脑的时候可以边听音乐边看新闻，同时还能跟朋友聊天。只要你觉得你的大脑还能处理得过来，你还能让这台电脑同时干更多的事情。其实这类操作系统我们完全可以只用“分时”二字简要概述下来。因为“分时”就像它最初的定义那样：将电脑的时间资源适当分配给所有使用者身上，让所有使用者有独占机器的感觉。但是如果把“使用者”进行抽象，就不仅可以代表人，还可以将任务也理解为电脑时间资源的使用者，那么“分时”就是多用户和多任务的基础和前提。所以，既然是“分时”的，一般都会支持多用户和多任务

●

线程上下文切换和进程上下问切换一个最主要的区别

线程的切换虚拟内存空间依然是相同的，但是进程切换是不同的。这两种上下文切换的处理都是通过操作系统内核来完成的。内核的这种切换过程伴随的最显著的性能损耗是将寄存器中的内容切换出。
另外一个隐藏的损耗是上下文的切换会扰乱处理器的缓存机制。

简单的说，一旦去切换上下文，处理器中所有已经缓存的内存地址一瞬间都作废了。还有一个 显著的区别是当你改变虚拟内存空间的时候，处理的页表缓冲（processor’s Translation Lookaside Buffer (TLB)）或者相当的神马东西会被全部刷新，这将导致内存的访问在一段时间内相当的低效。但是在线程的切换中，不会出现这个问题。
进程间切换的步骤：
1，保存程序计数其以及其他寄存器。
2， 更新当前处于“运行态”的进程的进程控制块，把进程状态改为相应状态，更新其他相关域
3， 把被切换进程的进程控制块移到相关状态的队列
4， 选择另外一个进程开始执行，把该进程进程控制块的状态改为“运行态”
5， 恢复被选择进程的处理器在最近一次被切换出运行态时的上下文，比如载入程序计数器以及其他处理器的值
进程间切换伴随着两次模式切换（用户--内核，内核--用户）。
（同一进程内）线程间切换的步骤：
线程分两种，用户级线程和内核级线程
在用户级线程中，有关线程管理的所有工作都由应用程序完成，内核没有意识到线程的存在。
（同一进程内）用户级线程间切换时，只需要保存用户寄存器的内容，程序计数器，栈指针，不需要模式切换。
缺点：
1， 在进程的某个线程执行系统调用时，不仅该线程被阻塞，该线程所在进程的所有线程都被阻塞
2， 无法利用多处理器
在内核级线程中，有关线程的管理工作都是由内核完成的，应用程序部分没有线程管理的权限，只有一个接口（API)
（同一进程内）内核级线程间切换时，除了保存上下文，还要进行模式切换。
优点：
1， 可以利用多处理器
2， 线程阻塞不会导致进程阻塞

●

在一个分布式系统中，一组独立的计算机展现给用户的是一个统一的整体，就好像是一个系统似的。系统拥有多种通用的物理和逻辑资源，可以动态的分配任务，分散的物理和逻辑资源通过计算机网络实现信息交换。系统中存在一个以全局的方式管理计算机资源的分布式操作系统。通常，对用户来说，分布式系统只有一个模型或范型。在操作系统之上有一层软件中间件（middleware）负责实现这个模型。一个著名的分布式系统的例子是万维网（World Wide Web），在万维网中，所有的一切看起来就好像是一个文档（Web 页面）一样。

在计算机网络中，这种统一性、模型以及其中的软件都不存在。用户看到的是实际的机器，计算机网络并没有使这些机器看起来是统一的。如果这些机器有不同的硬件或者不同的操作系统，那么，这些差异对于用户来说都是完全可见的。如果一个用户希望在一台远程机器上运行一个程序，那么，他必须登陆到远程机器上，然后在那台机

器上运行该程序。

分布式系统和计算机网络系统的共同点是：多数分布式系统是建立在计算机网络之上的，所以分布式系统与计算机网络在物理结构上是基本相同的。

他们的区别在于：分布式操作系统的设计思想和网络操作系统是不同的，这决定了他们在结构、工作方式和功能上也不同。网络操作系统要求网络用户在使用网络资源时首先必须了解网络资源，网络用户必须知道网络中各个计算机的功能与配置、软件资源、网络文件结构等情况，在网络中如果用户要读一个共享文件时，用户必须知道这个文件放在哪一台计算机的哪一个目录下；分布式操作系统是以全局方式管理系统资源的，它可以为用户任意调度网络资源，并且调度过程是“透明”的。当用户提交一个作业时，分布式操作系统能够根据需要在系统中选择最合适的处理器，将用户的作业提交到该处理程序，在处理器完成作业后，将结果传给用户。在这个过程中，用户并不会意识到有多个处理器的存在，这个系统就像是一个处理器一样。

●

OS 的四个特性：并发性、共享性,虚拟性和异步性。

异步性： 在多道程序环境下，允许多个进程并发执行。但由于资源等因素的限制，进程的执行通常并非一气呵成，而是以走走停停的方式运行。内存中的每个进程在何时执行，何时暂停，以怎样的速度向前推进，每道程序总共需要多少时间才能完成，都是不可预知的。故而作业完成的先后次序与进入内存的次序并不完全一致，亦即进程是以异步方式运行的。所以异步性就是 OS 不确定性的原因，会导致 1. 程序的运行结果不确定 2. 程序的运行次序不确定 3. 程序多次运行的时间不确定

●

如果系统只有用户态线程，则线程对操作系统是不可见的，操作系统只能调度进程；

如果系统中有内核态线程，则操作系统可以按线程进行调度；

●

这是一个概念混淆题。。。关键词为程序。

意图考察程序和进程的区别：

进程由程序和数据两部分组成，进程是竞争计算机系统有限资源的基本单位，也是进程处理机调度的基本单位。

程序是静态的概念；进程是程序在处理机上一次执行的过程，是动态的概念。

一个程序可以作为多个进程的运行程序；一个进程也可以运行多个程序。

还有其他的区别，这里只列出相关的这三点。

不加程序 二字，只说设备驱动或者驱动的话，考虑到加上内存二字，说明是已经加载、正在使用的，一般理解就是驱动程序的进程。在这些进程的数据中包括了硬件设备特有的信息，便于辨别不同的设备。

但是题目又故意添加程序二字。。。。于是只有两台不同类型的设备，程序只有两种。

题目意思还包括同时使用了这四台设备。

●

文件系统中的最小存储单元是逻辑块，也就是这里所说的 Block；

block 越大，inode 越少，适合存储大文件的文件系统；block 越小，inode 越多，适合存储文件多而小的文件系统。

要格式化档案系统为 Ext3，亦可以使用命令 mkfs.ext3,block 块可以用 mkfs.ext3 -b 来制定快的大小，每个 block 块最多可存放一个文件;

block 存放文件的数据，每个 block 最多存放一个文件，而当一个 block 存放不下的情况下，会占用下一个 block。

●

一条大型机通道（channel）某种程度上类似于 PCI 总线（bus），它能将一个或多个控制器连接起来，而这些控制器又控制着一个或更多的设备（磁盘驱动器、终端、LAN 端口，等等。）

通道控制设备控制器，设备控制器控制设备

●

1， spinlock 介绍

spinlock 又称自旋锁，线程通过 busy-wait-loop 的方式来获取锁，任时刻只有一个线程能够获得锁，其他线程忙等待直到获得锁。spinlock 在多处理器多线程环境的场景中有很广泛的使用，一般要求使用 spinlock 的临界区尽量简短，这样获取的锁可以尽快释放，以满足其他忙等的线程。Spinlock 和 mutex 不同，spinlock 不会导致线程的状态切换(用户态->内核态)，但是 spinlock 使用不当(如临界区执行时间过长)会导致 cpu busy 飙高。

2 ，使用准则

Spinlock 使用准则：临界区尽量简短，控制在 100 行代码以内，不要有显式或者隐式的系统调用，调用的函数也尽量简短。例如，不要在临界区中调用 read,write,open 等会产生系统调用的函数，也不要去 sleep；strcpy，memcpy 等函数慎用，依赖于数据的大小。

●

A 固态硬盘使用存储芯片做存储，速度是传统硬盘的十倍以上
B 传统硬盘是机械式硬盘，有转动的盘片，防震效果差
C 固态硬盘因为不需要机械马达，功耗更低
D 固态硬盘寿命远高于传统硬盘

●

A 选项：先来先服务，先来的也就是是等待时间最长的那个啊，所以 A 一定会出现题目中的情况。
B 选项高响应比优先级调度，响应比=(等待时间+要求服务时间)/要求服务时间，这个一般是先来先服务和最短作业优先的结合体，可能会出现题目的要求
C 优先权调度算法，优先权的计算也是多种多样的，我们比较熟悉的就是把等待时间作为一个因素来计算优先级的，这就可能出现题目中的要求
D 最短作业优先，不跟等待时间挂钩，但是某一时刻的调度也可能存在最短作业就是等待时间最长的作业啊，比如来了 3 个短的作业和一个中等长度的作业，等三个短作业执行完毕，又来了 4 个时间很长的作业，此时，调度就会出现题目的要求，不是吗？
我不知道我的理解哪里错了？

●

银行家算法获得的安全序列不唯一。
活动状态和休眠状态的进程也可以在主存中，不一定在外存中。
操作系统的主要功能是实现对系统硬件和软件资源的管理。

●

A 请求系统服务。操作系统编制了许多不同功能的子程序，供用户程序执行中调用。这些由操作系统提供的子程序称为系统功能调用，简称系统调用。系统调用是操作系统为用户程序提供的一种服务界面，或者说，是操作系统保证程序设计语言能正常工作的一种支持。
本题答案有误，应该选择 A 选项。

●

（1）从静态存储区域分配：
内存在程序编译时就已经分配好，这块内存在程序的整个运行期间都存在。速度快、不容易出错，因为有系统会善后。例如全局变量，static 变量等。
（2）在栈上分配：
在执行函数时，函数内局部变量的存储单元都在栈上创建，函数执行结束时这些存储单元自动被释放。栈内存分配运算内置于处理器的指令集中，效率很高，但是分配的内存容量有限。
（3）从堆上分配：
即动态内存分配。程序在运行的时候用 malloc 或 new 申请任意大小的内存，程序员自己负责在何时用 free 或 delete 释放内存。动态内存的生存期由程序员决定，使用非常灵活。如果在堆上分配了空间，就有责任回收它，否则运行的程序会出现内存泄漏，另外频繁地分配和释放不同大小的堆空间将会产生堆内碎块。
一个 C、C++程序编译时内存分为 5 大存储区：堆区、栈区、全局区、文字常量区、程序代码区。

●

静态页式管理是一次性为要求内存的进程分配足够多的页面，无法将外存的空间利用起来实现虚存

●

抖动就是指当内存中已无空闲空间而又发生缺页中断时，需要从内存中调出一页程序或数据送磁盘的对换区中，如果算法不适当，刚被换出的页很快被访问，需重新调入，因此需再选一页调出，而此时被换出的页很快又要被访问，因而又需将它调入，如此频繁更换页面，以致花费大量的
时间，我们称这种现象为"抖动";所以在请求分页的时候如果处理不当则会发生抖动。

●

虚存=min（内存+辅存，逻辑地址）。

●

首次适应法：从空闲 分区表 的第一个表目起查找该表，把最先能够满足要求的空闲区分配给作业，这种方法目的在于减少查找时间。为适应这种算法，空闲分区表（空闲区链）中的空闲分区要按地址由低到高进行排序。该算法优先使用低址部分空闲区，在低址空间造成许多小的空闲区，在高 地址空间 保留大的空闲区。
最佳适应算法：从全部空闲区中找出能满足作业要求的、且大小最小的空闲分区的一种计算方法，这种方法能使碎片尽量小。
最坏适应分配算法：要扫描整个空闲分区或链表，总是挑选一个最大的空闲分区分割给作业使用。该算法要求将所有的空闲分区按其容量从大到小的顺序形成一空闲分区链，查找时只要看第一个分区能否满足作业要求。
循环首次适应算法： 该算法是首次适应算法的变种。在分配内存空间时，不再每次从表头(链首)开始查找，而是从上次找到空闲区的下一个空闲开始查找，直到找到第一个能满足要求的的空闲区为止，并从中划出一块与请求大小相等的内存空间分配给作业。该算法能使内存中的空闲区分布得较均匀。

●

一次只能被一个进程访问的资源称为临界资源。

●

虚拟内存的作用同物理内存一样，只不过是从硬盘存储空间划出的部分，来完成内存的工作，由于不是真正的内存，所以被称为虚拟内存。因为计算机所支持的最大内存是由该计算机的地址位数决定的，也就是计算机的最大寻址能力。例如，32 位机的寻址能力为 2 的 32 次方，大约为 4G。所以虚拟内存的大小也受计算机地址位数的限制。不过这到题单选 D 是由前提条件的，那就是磁盘空间足够大，否则严格意义上讲应该多一个选项 A。

●

线程之间的通信是为了同步（毕竟共享进程的资源），同步措施实施起来并不高效

●

话筒是将语音信号转化为电信号的装置，语音信号是连续的，转化的电信号也是连续的，因此是模拟信号
模拟的电信号在计算机内经过采样，量化，编码，最终形成音频数字信号

●

虚地址通常指的是逻辑地址，而程序执行所要访问的内存地址指的是物理地址!!!

●

并发与并行是两个不同的概念：
并发是指多个请求向服务器同时请求，服务器的响应过程是依次响应，或者轮转响应的；
并行是多个请求向多个服务器，各自服务各自的请求；
一定要注意区分这些细微的概念!
并发（Concurrent）：指两个或者多个事件在同一时刻发生；
并行（Parallel）：指两个或者多个事件在同一时间间隔内发生。

●

批处理系统主要指多道批处理系统，由于多道程序能交替使用 CPU，提高了 CPU 及其他系统资源的利用率，同时也提高了系统的效率。多道批处理系统的缺点是延长了作业的周转时间，用户不能进行直接干预，缺少交互性，不利于程序的开发与调试。

●

一般来说，操作系统可以分为五大管理功能部分：
1）设备管理：主要是负责内核与外围设备的数据交互，实质是对硬件设备的管理，包括对输入输出设备的分配，初始化，维护与回收等。例如管理音频输入输出。
2）作业管理：这部分功能主要是负责人机交互，图形界面或者系统任务的管理。
3）文件管理：这部分功能涉及文件的逻辑组织和物理组织，目录结构和管理等。从操作系统的角度来看，文件系统是系统对文件存储器的存储空间进行分配，维护和回收，同时负责文件的索引，共享和权限保护。而从用户的角度来说，文件系统是按照文件目录和文件名来进行存取的。
4）进程管理：说明一个进程存在的唯一标志是 pcb（进程控制块），负责维护进程的信息和状态。进程管理实质上是系统采取某些进程调度算法来使处理合理的分配给每个任务使用。
5）存储管理：数据的存储方式和组织结构。

●

信号量当前值为-（m-1），当前为-4，表示有 5 个进程共享同一临界资源，但是临界资源只能由一个进程享用，因此等待的进程数为 4，其实等待的进程数就是（m-1）。

●

线程占有的都是不共享的，其中包括：栈、寄存器、状态、程序计数器
线程间共享的有：堆，全局变量，静态变量；
进程占有的资源有：地址空间，全局变量，打开的文件，子进程，信号量、账户信息。

●

动态链接也要以段为单位进行管理。
动态链接 是指在作业运行之前，并不把几个 目标程序 段链接起来。要运行时，先将主程序所对应的 目标程序 装入内存并启动运行，当运行过程中又需要调用某段时，才将该段(目标程序)调入内存并进行链接。可见， 动态链接也要求以段作为管理的单位。

●

动态优先权是指在创建进程时所赋予的优先权，是可以随进程的推进或随其等待时间的增加而改变的，以便获得更好的调度性能。例如，我们可以规定，在就绪队列中的进程，随其等待时间的增长，其优先权以速率 a 提高。若所有的进程都具有相同的优先权初值，则显然是最先进入就绪队列的进程将因其动态优先权变得最高而优先获得处理机，此即 FCFS 算法。

●

分时系统的四个特点：
1、多路性。允许一台主机上联接多台终端，系统按分时原则为每个用户服务；
2、独立性。每个用户各占一个终端，独立操作，互不干扰；
3、及时性。用户的请求能在很短的时间内获得响应；
4、交互性。用户可通过终端与系统进行广泛的人机对话。
多用户分时系统要交互性良好。

●

文件的结构就是文件的组织形式，从用户观点所看到的文件组织形式成为文件的 逻辑结构，从实现观点看到的的文件在外存上的存放形式称为文件的物理结构，文件的逻辑结构与存储设备特性无关，但文件的物理结构与存储设备的特性有很大关系。

●

244 分成 2 进制是
010,100,100
所以是
-w- r-- r--

●

请求分段系统是在分段系统的基础上，增加了请求调段功能和分段置换功能所形成的分段式虚拟存储系统。分段式存储管理方式分配算法与可变分区的分配算法相似，可以采用最佳适应法、最坏适应法和首次适应法等分配算法。显然仍然要解决外碎片的问题。

首次 适应分配算法： 这种算法按分区序号从空闲分区表的第一个表目开始查找该表， 把最先找到的大于或等于作业大小的空闲分区分给要求的作业 。然后，再按照作业的大小，从该分区中划出一块内存空间分配给作业，余下的空闲分区仍留在空闲分区表中。如果查找到分区表的最后仍没有找到大于或等于该作业的空闲区，则此次分配失败。 优点：优先利用内存中低址部分的空闲分区，而高址部分的空闲分区很少被利用，从而保留了高址部分的大空闲区。为以后到达的大作业分配大的内存空间创造了条件。缺点：低址部分不断被划分，致使留下许多难以利用的、很小的空闲分区。

循环 首次 适应分配算法： 这种算法是由最先适应分配算法经过改进而形成的。在为作业分配内存时，不再每次从空闲分区表的第一个表项开始查找，而是从上次找到的空闲区的下一个空闲区开始查找，直至找到第一个能满足要求的空闲区为止，并从中划分出一块与请求大小相等的内存空间分配给作业。为实现该算法，应设置一起始查找指针，以指示下一次开始查找的空闲分区，并采用循环查找方式。即如果最后一个空闲分区的大小仍不能满足要求，则返回到第一个空闲分区进行查找。 优点 ：内存中的空闲区分布得更均匀，减少查找空闲分区的开销。 缺点 ：系统中缺乏大的空闲分区，对大作业不利。

最佳适应分配算法 ：该算法从所有未分配的分区中挑选一个 最接近作业大小且大于或等于作业的空闲分区分配给作业 ，目的是使每次分配后剩余的碎片最小。为了查找到大小最合适的空闲分区，需要查遍整个空闲分区表，从而增加了查找时间。因此，为了加快查找速度，要求将所有的空闲分区，按从小到大递增的顺序进行排序。这样，第一次找到的满足要求的空闲分区，必然是最佳的。 缺点 ：每次分配之后形成的剩余部分，却是一些小的碎片，不能被别的作业利用。因此，该算法的内存利用率是不高的。

最坏适应分配算法： 该算法从所有未分配的分区中挑选一个 最大的空闲分区分配给作业 ，目的是使分配后剩余的空闲分区足够大，可以被别的作业使用。为了查找到最大的空闲分区，需要查遍整个空闲分区表，从而增加了查找时间。因此，为了加快查找速度，要求将所有的空闲分区按从大到小递减的顺序进行排序。这样，第一次找到的空闲分区，必然是最大的。优点：最坏适应分配算法在分配后剩余的空闲分区可能比较大，仍能满足一般作业的要求，可供以后使用。从而最大程度地减少系统中不可利用的碎片。缺点：这种算法使系统中的各空闲分区比较均匀地减小，工作一段时间以后，就不能满足对较大空闲分区的分配要求。

●

采用段式管理的系统中，其逻辑地址分为段号和页内偏移量。

本题的地址一共 24 位，使用了 8 位表示段号，那么把剩下的 16 位全部用来表示段内偏移量就能使每段长度最大，如下图所示

从而每段允许的最大长度为 216

●

最优适应算法：通常将空闲区按长度递增顺序排列。查找时总是从最小一个空闲区开始，直到找到满足要求的分区为止。此算法保证不会分割一个更大的区域，使得装入大作业的要求容易得到满足。

补充：

最先适应算法：通常将空闲区按地址从小到大排列。查找时总是从低地址开始，可使高地址尽量少用，以保持一个大空闲区，有利于大作业的装入；缺点是内存低地址和高地址两端的分区利用不平衡，回收分区较麻烦。

最坏适应算法：通常将空闲区按长度递减顺序排列 。查找时从最大的一个空闲区开始，总是挑选一个最大的空闲区分割给作业使用，其优点是使剩下的空闲区不致于太小，这样有利于中小型作业，但不利于大作业。

这些都属于可变分区分配算法，当然还有下次适应分配算法和快速适应分配算法。

注：这些算法理解即可。

●

（1） 进程间通信方法有：文件映射、共享内存、匿名管道、命名管道、邮件槽、剪切板、动态数据交换、对象连接与嵌入、动态连接库、远程过程调用等

（2） 事件、临界区、互斥量、信号量可以实现线程同步

●

我们将对共享内存进行访问的程序片段称为临界区域(critical region)或临界区，实现临界区互斥的方案如 Peterson 解法：本质思想当一个进程想进入临界区时，先检查是否允许进入，若不允许，就原地等待直到允许为止。

考虑一台计算机有两个优先级不同的进程，一个 H 的优先级较高，L 较低，调度规则规定只要 H 处于就绪态就会运行，如果 L 处于临界区时 H 变为就绪态，比如刚刚结束了一个 I/O 操作，由于 H 就绪时 L 不会被调度，如果 H 采用了忙等待，由于 L 不被调度它将一直处于临界区，而 H 将一直等待下去，这也就是优先级反转的问题。

如果我们采用另一种策略，在一个进程不能进入临界区的时候将其挂起而不是进行忙等待，直到另一个进程将其 wakeup，那么处于临界区的就不会被中断。

个人认为选项 A 没有介绍互斥的策略，所以是错的。

选项 C：

任意时刻，由于程序局部性，往往在一个小的活动页面集合上工作，叫做工作集，如果工作集的大小超过了物理存储器的大小，那么程序将出现 thrashing,页面将不断换进换出。

所以 C 的解释是不准确的。

●

在多用户、多任务的 计算机系统 中特权指令必不可少。它主要用于 系统资源 的分配和管理，包括改变系统工作方式，检测用户的访问权限，修改虚拟存储器管理的段表、页表，完成任务的创建和切换等。

常见的特权指令有以下几种:

(1)有关对 I/O 设备使用的指令 如启动 I/O 设备指令、测试 I/O 设备工作状态和控制 I/O 设备动作的指令等。
(2)有关访问程序状态的指令 如对程序状态字(PSW)的指令等。
(3)存取特殊寄存器指令 如存取中断寄存器、时钟寄存器等指令。
(4)其他指令

●

单道程序系统几个特点：
1. 资源独占性
任何时候，位于内存中的程序可以使用系统中的一切资源，不可能有其他程序与之竞争
2. 执行的顺序性
内存中只有一个程序，各个程序是按次序执行的。在做完一个程序的过程中，不可能夹杂进另一个程序执行
3. 结果的可再现性
只要执行环境和初始条件相同，重复执行一个程序，获得的结果总是一样的
4. 运行结果的无关性
程序的运行结果与程序执行的速度无关。系统中的作业以串行的方式被处理，无法提高 CPU、内存的利用率

●

程序 IO 方式，是采用 busy-waiting 的方式，即 CPU 会采用轮询的方式来询问数据-------效果最差
中断 IO 方式，是设备控制器当取出一个数据之后向 CPU 发送一个中断，然后 CPU 将数据从控制器中取到 CPU 寄存器，再然后转移到内存中。这种方式，CPU 是以字节的方式来响应数据的。
DMA 方式，是 CPU 通过向 DMA 控制器设定若干参数，然后 DMA 打开了一条内存到设备的通道，这样，设备（内存）中的数据可以不通过ＣＰＵI 来进行数据交互。缺点是，DMA 是多少设备就需要多少 DMA，而且，DMA 方式下，CPU 的访问设备是以数据块为周期的。
到了 IO 通道方式，IO 通道相当于一个简单的处理机，有自己的指令，也可以执行指令。指令存储在内存。
IO 通道相当于一条 PCI 总线，一条 IO 通道可以连接所有的设备控制器。然后 CPU 向 IO 通道发出指令，IO 通道将会自动进行获取数据。
另外，IO 通道是以一组块为单位进行获取的。
所以，IO 通道方式需要最少的 CPU 干预

●

银行家算法
消进程法
资源静态分配法
资源分配图简化法
A.避免死锁
B.解除死锁
C.预防死锁
D.检测死锁

●

每个 inode 大小固定为 128bytes，每个文件仅会占用一个 inode。inode 除了记录文件的属性外，还有 12 个直接，一个间接，一个双间接，一个三间接来记录 block。
12 个直接指向：12*4K=48K
间接：间接记录 block 的区段可以指向一个 block，该 block 用于记录存储数据的 block。每条 block 号码的记录要用去 4bytes，因此一个 4K 的 block 可以记录 4K/4byte=1024 个 block 号码。所以可记录大小为 1024*4K=4M
双间接：同理，这个双间接区段可以指向一个 block，该 block 用于记录存储指向数据 block 的 block。4K 的 block 可以记录 1024 个 block 号码，而这 1024 个 block 中每一个都可以记录 1024 个指向数据的 block。所以可记录的总大小为 1024*1024*4K=4G
三间接：同理可得三间接区段可记录文件大小为 1024*1024*1024*4K=4T

●

同步为 0，互斥为 1
管理临界区时，对互斥资源访问，因此设置为 1。

●

时间片轮转法进行进程调度是为了()。多个终端都能得到系统的及时响应

●

采用索引这种结构，逻辑上连续的文件可以存放在若干不连续的物理块中，但对于每个文件，在存储介质中除存储文件本身外，还要求系统另外建立一张索引表。 索引结构既适用于顺序存取，也适用于随机存取，并且访问速度快，文件长度可以动态变化。 索引结构的缺点 是由于使用了索引表而增加了存储空间的开销。

●

线程通常被定义为一个进程中代码的不同执行路线。从实现方式上划分，线程有两种类型：“用户级线程”和“内核级线程”。 用户线程指不需要内核支持而在用户程序中实现的线程，其不依赖于操作系统核心，应用进程利用线程库提供创建、同步、调度和管理线程的函数来控制用户线程。这种线程甚至在象 DOS 这样的操作系统中也可实现，但线程的调度需要用户程序完成，这有些类似 Windows 3.x 的协作式多任务。另外一种则需要内核的参与，由内核完成线程的调度。其依赖于操作系统核心，由内核的内部需求进行创建和撤销，这两种模型各有其好处和缺点。用户线程不需要额外的内核开支，并且用户态线程的实现方式可以被定制或修改以适应特殊应用的要求，但是当一个线程因 I/O 而处于等待状态时，整个进程就会被调度程序切换为等待状态敏感词线程得不到运行的机会；而内核线程则没有各个限制，有利于发挥多处理器的并发优势，但却占用了更多的系统开支。 Windows NT 和 OS/2 支持内核线程。Linux 支持内核级的多线程
用户线程指不需要内核支持而在用户程序中实现的线程，其不依赖于操作系统核心，应用进程利用线程库提供创建、同步、调度和管理线程的函数来控制用户线程

●

用户线程的调度不需要经过内核态
用户线程指不需要内核支持而在用户程序中实现的线程，其不依赖于操作系统核心，应用进程利用线程库提供创建、同步、调度和管理线程的函数来控制用户线程

●

I/O 通道的目的是为了建立独立的 I/O 通道，使得原来一些由 CPU 处理的 I/O 任务由通道来承担，从而解脱 cpu。通道所能执行的命令局限于 I/O 操作的指令，也就是执行 I/O 指令集。
通道是独立于 CPU 的，使得原来由 CPU 处理的任务由通道来承担。

●

页表项（页描述子）中各个位的作用：
1. 页号
2. 块号（页框号）
3. 中断位： 用于判断该页是不是在内存中，如果是 0，表示该页面不在内存中，会引起一个缺页中断
4. 保护位（存取控制位）：用于指出该页允许什么类型的访问，如果用一位来标识的话：1 表示只读，0 表示读写
5. 修改位（脏位）：用于页面的换出，如果某个页面被修改过（即为脏），在淘汰该页时，必须将其写回磁盘，反之，可以直接丢弃该页
6. 访问位：不论是读还是写（get or set），系统都会设置该页的访问位，它的值用来帮助操作系统在发生缺页中断时选择要被淘汰的页，即用于页面置换
7. 高速缓存禁止位（辅存地址位）：对于那些映射到设备寄存器而不是常规内存的页面有用，假设操作系统正在循环等待某个 I/O 设备对其指令进行响应，保证硬件不断的从设备中读取数据而不是访问一个旧的高速缓存中的副本是非常重要的。即用于页面调入。

●

在请求分页存储管理中，刚被替换出去的页，立即又要被访问因无空,此时因无空闲内存，又要替换另一页，而后者又是下一次要被访问的页，于是系统需花费大量的时间忙于进行这种频繁的页面交换，致使系统的实际效率很低，这种现象称为抖动现象。一般都是由于置换算法不佳引起，因此选 A

●

字节多路通道是分时并行的，但不适用于高速设备。
数组选择通道有很高的传输速率，但只含有一个分配型子通道，一段时间只能执行一道通道程序
数组多路通道是上述两者的结合，兼具分时并行和高传输速率的优点

●

管道(pipe)：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。

信号量(semophore) ： 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，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，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。

消息队列(message queue) ： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。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。

共享内存(shared memory) ：共享内存就是映射一段能被其他进程所访问的内存，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，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号两，配合使用，来实现进程间的同步和通信。

套接字(socket) ： 套解口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同及其间的进程通信。

●

A，内存泄露是程序设计的 bug，不是操作系统的问题

B，内存泄露跟线程数无关

C，内存泄露是进程申请了内存却没有释放。导致占用内存无限上升

D，进程退出之前释放申请的内存，不代表进程运行过程中没有内存泄露

E，java 是自动管理内存的，但是也会有内存泄露，比如加入 HashMap 的对象 hash 值改变了就无法从 HashMap 中 remove，这就造成了内存泄露

●

地址寄存器长度为 24 位，其中页号占 14 位，则页内地址占 10 位，分页大小与主存块大小相同，因此， 主存的分块大小是 210 字节。

●

作业调度仅仅是创建进程将其调入内存

进程调度是获取 cpu 的

●

在请求分页系统中，页表中的改变位是供（）参考的。页面换出

●

线程共享的环境包括：进程代码段、进程的公有数据(利用这些共享的数据，线程很容易的实现相互之间的通讯)、进程打开的文件描述符、信号的处理器、进程的当前目录和进程用户 ID 与进程组 ID。

●

Block 原语是由被阻塞进程自我调用实现的，而 Wakeup 原语则是由一个与被唤醒进程相合作或被其他相关的进程调用实现的。

引起进程阻塞和唤醒的事件：

1、向系统请求共享资源失败。进程转变为阻塞状态。

2、等待某种操作完成。比如进程启动了 I/O 设备，必须等 I/O 操作完成后，进程才能继续。

3、 新数据尚未到达。对于合作的进程，如果一个进程 A 需要先获得另一进程 B 提供的数据后，才能对该数据进程处理，只要数据尚未到达，进程 A 只能阻塞。当进程 B 提供数据后，便可以去唤醒进程 A。

4、等待新任务到达，用于特定的系统进程。它们每当完成任务后，就把自己阻塞起来，等待新任务。

●

分时系统需要使用下面哪些技术

1.多道程序设计技术

2.终端命令解释程序

3.中断处理

4.系统调用

●

线程共享的内容包括：

1.进程代码段

2.进程的公有数据(利用这些共享的数据，线程很容易的实现相互之间的通讯)
3.进程打开的文件描述符、
4.信号的处理器、
5.进程的当前目录和
6.进程用户 ID 与进程组 ID
线程独有的内容包括：
1.线程 ID
2.寄存器组的值
3.线程的堆栈
4.错误返回码
5.线程的信号屏蔽码

●

（1） 虚拟存储器中，主存页面（或程序段）的替换
（2） Cache 中的块替换
（3） 虚拟存储器的快慢表中，快表的替换
（4） 虚拟存储器中，用户基地址寄存器的替换

●

多道程序执行（分为顺序执行 和 并发执行）。
如果是顺序执行： 可在现
如果是并发执行（伪并行）：不可在现。

●

线程通常被定义为一个进程中代码的不同执行路线。从实现方式上划分，线程有两种类型：“用户级线程”和“内核级线程”。 用户线程指不需要内核支持而在用户程序中实现的线程，其不依赖于操作系统核心，应用进程利用线程库提供创建、同步、调度和管理线程的函数来控制用户线程。这种线程甚至在象 DOS 这样的操作系统中也可实现，但线程的调度需要用户程序完成，这有些类似 Windows 3.x 的协作式多任务。另外一种则需要内核的参与，由内核完成线程的调度。其依赖于操作系统核心，由内核的内部需求进行创建和撤销，这两种模型各有其好处和缺点。用户线程不需要额外的内核开支，并且用户态线程的实现方式可以被定制或修改以适应特殊应用的要求，但是当一个线程因 I/O 而处于等待状态时，整个进程就会被调度程序切换为等待状态敏感词线程得不到运行的机会；而内核线程则没有各个限制，有利于发挥多处理器的并发优势，但却占用了更多的系统开支。 Windows NT 和 OS/2 支持内核线程。Linux 支持内核级的多线程

●

共享设备在同一时刻只能有一个作业调用。

●

LRU 算法是与每个页面最后使用的时间有关的。当必须置换一个页面时，LRU 算法选择过去一段时间里最久未被使用的页面。
LRU 算法是经常采用的页面置换算法，并被认为是相当好的，但是存在如何实现它的问题。LRU 算法需要实际硬件的支持。其问题是怎么确定最后使用时间的顺序，对此有两种可行的办法：
1.计数器。最简单的情况是使每个页表项对应一个使用时间字段，并给 CPU 增加一个逻辑时钟或计数器。每次存储访问，该时钟都加 1。每当访问一个页面时，时钟寄存器的内容就被复制到相应页表项的使用时间字段中。这样我们就可以始终保留着每个页面最后访问的“时间”。在置换页面时，选择该时间值最小的页面。这样做，不仅要查页表，而且当页表改变时（因 CPU 调度）要维护这个页表中的时间，还要考虑到时钟值溢出的问题。
2.栈。用一个栈保留页号。每当访问一个页面时，就把它从栈中取出放在栈顶上。这样一来，栈顶总是放有目前使用最多的页，而栈底放着目前最少使用的页。由于要从栈的中间移走一项，所以要用具有头尾指针的双向链连起来。在最坏的情况下，移走一页并把它放在栈顶上需要改动 6 个指针。每次修改都要有开销，但需要置换哪个页面却可直接得到，用不着查找，因为尾指针指向栈底，其中有被置换页。
上述来自：百度百科--页面置换算法

●

一个进程运行时由于自身或外界的原因而可能被中断，且断点是不固定的。一个进程被中断后，哪个进程可以运行

呢？被中断的进程什么时候能再去占用处理器呢？这是与进程调度策略有关的。所以，进程执行的相对速度不能由进程自己来控制，于是，就可能使并发进程在共享资源时出现与时间有关的错误。

●

多道程序系统是在计算机内存中同时存放几道相互独立的程序，使它们在管理程序控制之下，相互穿插的运行 (系统由一个程序转而运行另一个程序时需要使用中断机构中断正在运行的程序) 。 两个或两个以上程序在计算机系统中同处于开始和结束之间的状态。这就称为多道程序技术运行的特征：多道、宏观上并行、微观上串行。

●

多道批处理系统的核心技术：
作业调度：作业的现场保存和恢复
资源共享：资源的竞争和同步——互斥机制
内存使用：提高内存使用效率

●

Linux ext2/ext3 文件系统使用索引节点来记录文件信息，作用像 windows 的文件分配表。索引节点是一个结构，它包含了一个文件的长度、创建及修改时间、权限、所属关系、磁盘中的位置等信息。
Ext3 日志文件系统的特点：
1、高可用性
系统使用了 ext3 文件系统后，即使在非正常关机后，系统也不需要检查文件系统。宕机发生后，恢复 ext3 文件系统的时间只要数十秒钟。
2、数据的完整性:
ext3 文件系统能够极大地提高文件系统的完整性，避免了意外宕机对文件系统的破坏。在保证数据完整性方面，ext3 文件系统有 2 种模式可供选择。其中之一就是“同时保持文件系统及数据的一致性”模式。采用这种方式，你永远不再会看到由于非正常关机而存储在磁盘上的垃圾文件。
3、文件系统的速度:
尽管使用 ext3 文件系统时，有时在存储数据时可能要多次写数据，但是，从总体上看来，ext3 比 ext2 的性能还要好一些。这是因为 ext3 的日志功能对磁盘的驱动器读写头进行了优化。所以，文件系统的读写性能较之 Ext2 文件系统并来说，性能并没有降低。
4、数据转换
由 ext2 文件系统转换成 ext3 文件系统非常容易，只要简单地键入两条命令即可完成整个转换过程，用户不用花时间备份、恢复、格式化分区等。用一个 ext3 文件系统提供的小工具 tune2fs，它可以将 ext2 文件系统轻松转换为 ext3 日志文件系统。另外，ext3 文件系统可以不经任何更改，而直接加载成为 ext2 文件系统。
5、多种日志模式
Ext3 有多种日志模式，一种工作模式是对所有的文件数据及 metadata（定义文件系统中数据的数据,即数据的数据）进行日志记录（data=journal 模式）；另一种工作模式则是只对 metadata 记录日志，而不对数据进行日志记录，也即所谓 data=ordered 或者 data=writeback 模式。系统管理人员可以根据系统的实际工作要求，在系统的工作速度与文件数据的一致性之间作出选择。

●

Ext3 目前所支持的最大 16TB 文件系统和最大 2TB 文件，Ext4 分别支持
1EB（1,048,576TB， 1EB=1024PB， 1PB=1024TB）的文件系统，以及 16TB 的文件。

●

1. FIFO（First in First out），先进先出。其实在操作系统的设计理念中很多地方都利用到了先进先出的思想，比如作业调度（先来先服务），为什么这个原则在很多地方都会用到呢？因为这个原则简单、且符合人们的惯性思维，具备公平性，并且实现起来简单，直接使用数据结构中的队列即可实现。
2. LFU（Least Frequently Used）最近最少使用算法。它是基于“ 如果一个数据在最近一段时间内使用次数很少，那么在将来一段时间内被使用的可能性也很小 ”的思路。
3. LRU （Least Recently Used） 它的 实质 是，当需要置换一页时， 选择在最近一段时间里最久没有使用过的页面予以置换， 这种算法就称为 最久未使用算法。
4. （OPT） 最优置换（Optimal Replacement）是在 理论上 提出的一种算法。其 实质 是：当调入新的一页而必须预先置换某个老页时，所选择的老页应是将来不再被使用，或者是在最远的将来才被访问。采用这种页面置换算

法，保证有最少的缺页率。

5. （SCR） 第二次机会算法基本思想是与 FIFO 相同的，但是有所改进，避免把经常使用的页面置换出去。当选择置换页面时，检查它的访问位。如果是 0，就淘汰这页；如果访问位是 1，就给它第二次机会，并选择下一个 FIFO 页面。当一个页面得到第二次机会时，它的访问位就清为 0，它的到达时间就置为当前时间。如果该页在此期间被访问过，则访问位置 1。这样给了第二次机会的页面将不被淘汰，直至所有其他页面被淘汰过（或者也给了第二次机会）。因此，如果一个页面经常使用，它的访问位总保持为 1，它就从来不会被淘汰出去。

文字游戏… 不常用和不使用的问题 LFU 是最近不常用 LRU 最近不使用

●

1、固定分区存储管理

其基本思想是将内存划分成若干固定大小的分区，每个分区中最多只能装入一个作业。当作业申请内存时，系统按一定的算法为其选择一个适当的分区，并装入内存运行。由于分区大小是事先固定的，因而可容纳作业的大小受到限制，而且当用户作业的地址空间小于分区的存储空间时，造成存储空间浪费。

一、空间的分配与回收

系统设置一张“分区分配表”来描述各分区的使用情况，登记的内容应包括：分区号、起始地址、长度和占用标志。其中占用标志为“0”时，表示目前该分区空闲；否则登记占用作业名（或作业号）。有了“分区分配表”，空间分配与回收工作是比较简单的。

二、地址转换和存储保护

固定分区管理可以采用静态重定位方式进行地址映射。

为了实现存储保护，处理器设置了一对“下限寄存器”和“上限寄存器”。当一个已经被装入主存储器的作业能够得到处理器运行时，进程调度应记录当前运行作业所在的分区号，且把该分区的下限地址和上限地址分别送入下限寄存器和上限寄存器中。处理器执行该作业的指令时必须核对其要访问的绝对地址是否越界。

三、多作业队列的固定分区管理

为避免小作业被分配到大的分区中造成空间的浪费，可采用多作业队列的方法。即系统按分区数设置多个作业队列，将作业按其大小排到不同的队列中，一个队列对应某一个分区，以提高内存利用率。

2、可变分区存储管理

可变分区存储管理不是预先将内存划分分区，而是在作业装入内存时建立分区，使分区的大小正好与作业要求的存储空间相等。这种处理方式使内存分配有较大的灵活性，也提高了内存利用率。但是随着对内存不断地分配、释放操作会引起存储碎片的产生。

一、空间的分配与回收

采用可变分区存储管理，系统中的分区个数与分区的大小都在不断地变化，系统利用“空闲区表”来管理内存中的空闲分区，其中登记空闲区的起始地址、长度和状态。当有作业要进入内存时，在“空闲区表”中查找状态为“未分配”且长度大于或等于作业的空闲分区分配给作业，并做适当调整；当一个作业运行完成时，应将该作业占用的空间作为空闲区归还给系统。

可以采用首先适应算法、最佳（优）适应算法和最坏适应算法三种分配策略之一进行内存分配。

二、地址转换和存储保护

可变分区存储管理一般采用动态重定位的方式，为实现地址重定位和存储保护，系统设置相应的硬件：基址/限长寄存器（或上界/下界寄存器）、加法器、比较线路等。

基址寄存器用来存放程序在内存的起始地址，限长寄存器用来存放程序的长度。处理机在执行时，用程序中的相对地址加上基址寄存器中的基地址，形成一个绝对地址，并将相对地址与限长寄存器进行计算比较，检查是否发生地址越界。

三、存储碎片与程序的移动

所谓碎片是指内存中出现的一些零散的小空闲区域。由于碎片都很小，无法再利用。如果内存中碎片很多，将会造成严重的存储资源浪费。解决碎片的方法是移动所有的占用区域，使所有的空闲区合并成一片连续区域，这一技术称为移动技术（紧凑技术）。移动技术除了可解决碎片问题还使内存中的作业进行扩充。显然，移动带来系统开销加大，并且当一个作业如果正与外设进行 I/O 时，该作业是无法移动的。

3、页式存储管理

基本原理

1．等分内存

页式存储管理将内存空间划分成等长的若干区域，每个区域的大小一般取 2 的整数幂，称为一个物理页面有时称为块。内存的所有物理页面从 0 开始编号，称作物理页号。

2．逻辑地址

系统将程序的逻辑空间按照同样大小也划分成若干页面，称为逻辑页面也称为页。程序的各个逻辑页面从 0 开始依次编号，称作逻辑页号或相对页号。每个页面内从 0 开始编址，称为页内地址。程序中的逻辑地址由两部分组成：

逻辑地址
页号 p
页内地址 d

3．内存分配

系统可用一张“位示图”来登记内存中各块的分配情况，存储分配时以页面（块）为单位，并按程序的页数多少进行分配。相邻的页面在内存中不一定相邻，即分配给程序的内存块之间不一定连续。

对程序地址空间的分页是系统自动进行的，即对用户是透明的。由于页面尺寸为 2 的整数次幂，故相对地址中的高位部分即为页号，低位部分为页内地址。

3.5.2 实现原理

1．页表

系统为每个进程建立一张页表，用于记录进程逻辑页面与内存物理页面之间的对应关系。地址空间有多少页，该页表里就登记多少行，且按逻辑页的顺序排列，形如：

逻辑页号
主存块号

0
B0

1
B1

2
B2

3
B3

2．地址映射过程

页式存储管理采用动态重定位，即在程序的执行过程中完成地址转换。处理器每执行一条指令，就将指令中的逻辑地址（p,d）取来从中得到逻辑页号(p)，硬件机构按此页号查页表，得到内存的块号 B’，便形成绝对地址（B’,d），处理器即按此地址访问主存。

3．页面的共享与保护

当多个不同进程中需要有相同页面信息时，可以在主存中只保留一个副本，只要让这些进程各自的有关项中指向内存同一块号即可。同时在页表中设置相应的“存取权限”，对不同进程的访问权限进行各种必要的限制。

4、段式存储管理

基本原理

1．逻辑地址空间

程序按逻辑上有完整意义的段来划分，称为逻辑段。例如主程序、子程序、数据等都可各成一段。将一个程序的所有逻辑段从 0 开始编号，称为段号。每一个逻辑段都是从 0 开始编址，称为段内地址。

2．逻辑地址

程序中的逻辑地址由段号和段内地址（s,d）两部分组成。

3．内存分配

系统不进行预先划分，而是以段为单位进行内存分配，为每一个逻辑段分配一个连续的内存区（物理段）。逻辑上连续的段在内存不一定连续存放。

3．6．2 实现方法

1．段表

系统为每个进程建立一张段表，用于记录进程的逻辑段与内存物理段之间的对应关系，至少应包括逻辑段号、物理段首地址和该段长度三项内容。

2．建立空闲区表

系统中设立一张内存空闲区表，记录内存中空闲区域情况，用于段的分配和回收内存。

3．地址映射过程

段式存储管理采用动态重定位，处理器每执行一条指令，就将指令中的逻辑地址（s,d）取来从中得到逻辑段号(s)，硬件机构按此段号查段表，得到该段在内存的首地址 S’， 该段在内存的首地址 S’加上段内地址 d，便形成绝对地址（S’+d），处理器即按此地址访问主存。

5、段页式存储管理

页式存储管理的特征是等分内存，解决了碎片问题；段式存储管理的特征是逻辑分段，便于实现共享。为了保持页式和段式上的优点，结合两种存储管理方案，形成了段页式存储管理。

段页式存储管理的基本思想是：把内存划分为大小相等的页面；将程序按其逻辑关系划分为若干段；再按照页面的大小，把每一段划分成若干页面。程序的逻辑地址由三部分组成，形式如下：

逻辑地址
段号 s
页号 p
页内地址 d

内存是以页为基本单位分配给每个程序的，在逻辑上相邻的页面内存不一定相邻。

系统为每个进程建立一张段表，为进程的每一段各建立一张页表。地址转换过程，要经过查段表、页表后才能得到最终的物理地址。

●

所谓碎片是指内存中出现的一些零散的小空闲区域。由于碎片都很小，无法再利用。如果内存中碎片很多，将会造成严重的存储资源浪费。解决碎片的方法是移动所有的占用区域，使所有的空闲区合并成一片连续区域，这一技术称为移动技术（紧凑技术）。移动技术除了可解决碎片问题还使内存中的作业进行扩充。显然，移动带来系统开销加大，并且当一个作业如果正与外设进行 I/O 时，该作业是无法移动的。

●

在多道程序环境中，只有进程才能在系统中运行。因此为了使程序运行必须为其创建进程，而导致进程创建的时间典型的有四种：
1.用户登录。可以理解为，一个新的用户来了，需要为它提供服务，这个服务之前没有，所以要创建。
2.作业调度。系统会为调度的作业分配资源，从后备队列中将其放入内存中，并为其创建进程。
3.提供服务。
4.应用请求。

●

一般地，解决死锁的方法分为死锁的预防，避免，检测与恢复三种（注意：死锁的检测与恢复是一个方法）
死锁的预防是保证系统不进入死锁状态的一种策略。它的基本思想是要求进程申请资源时遵循某种协议，从而打破产生死锁的四个必要条件中的一个或几个，保证系统不会进入死锁状态。
死锁的避免，它不限制进程有关申请资源的命令，而是对进程所发出的每一个申请资源命令加以动态地检查，并根

据检查结果决定是否进行资源分配。就是说，在资源分配过程中若预测有发生死锁的可能性，则加以避免。这种方法的关键是确定资源分配的安全性。

死锁的检测与恢复 ， 一般来说，由于操作系统有并发，共享以及随机性等特点，通过预防和避免的手段达到排除死锁的目的是很困难的。这需要较大的系统开销，而且不能充分利用资源。为此，一种简便的方法是系统为进程分配资源时，不采取任何限制性措施，但是提供了检测和解脱死锁的手段：能发现死锁并从死锁状态中恢复出来。因此，在实际的操作系统中往往采用死锁的检测与恢复方法来排除死锁。 死锁检测与恢复是指系统设有专门的机构，当死锁发生时，该机构能够检测到死锁发生的位置和原因，并能通过外力破坏死锁发生的必要条件，从而使得并发进程从死锁状态中恢复出来。

●

传说中鸵鸟看到危险就把头埋在地底下。当你对某一件事情没有一个很好的解决方法时，那就忽略它，就像鸵鸟面对危险时会把它深埋在沙砾中，装作看不到。这样的算法称为“鸵鸟算法“。 我还以为鸵鸟算法就是看到他死锁了也不管他呢。。。。

●

在计算机科学中，鸵鸟策略是解决潜在问题的一种方法。假设的前提是，这样的问题出现的概率很低。比如，在操作系统中，为应对死锁问题，可以采用这样的一种办法。当系统发生死锁时不会对用户造成多大影响，或系统很少发生死锁的场合采用允许死锁发生的鸵鸟策略，这样一来可能开销比不允许发生死锁及检测和解除死锁的小。如果死锁很长时间才发生一次，而系统每周都会因硬件故障、编译器错误或操作系统错误而崩溃一次，那么大多数工程师不会以性能损失或者易用性损失的代价来设计较为复杂的死锁解决策略，来消除死锁。

●

分段是一组有逻辑意义的信息集合。

分段后，段表包含以下信息：段号+段长+段基地址+存取控制信息

所以段长是可以不固定的，但是每个段内地址是连续的。

这根分页有点区别，分页系统中，每个页面大小是固定的。

分段：不定长 连续

分页：定长 可能连续可能不连续

●

每按键一次，或鼠标点击一次，都产生一个中断，称为按键中断，执行中断响应程序，操作系统将按键消息加入消息队列

●

通道指令是用来执行 I/O 操作的指令，而一般指令是大众 CPU 指令。

●

内存屏障,也称内存栅栏，内存栅障，屏障指令等， 是一类同步屏障指令，使得 CPU 或编译器在对内存随机访问的操作中的一个同步点，使得此点之前的所有读写操作都执行后才可以开始执行此点之后的操作。 大多数现代计算机为了提高性能而采取乱序执行，这使得内存屏障成为必须。 语义上，内存屏障之前的所有写操作都要写入内存；内存屏障之后的读操作都可以获得同步屏障之前的写操作的结果。因此，对于敏感的程序块，写操作之后、读操作之前可以插入内存屏障。

内存屏障可以限制 CPU 对内存的访问，表现在编程语言上就是上锁！

●

1、什么是临界区？

答：每个进程中访问临界资源的那段程序称为临界区（临界资源是一次仅允许一个进程使用的共享资源）。每次只准许一个进程进入临界区，进入后不允许其他进程进入。

2、进程进入临界区的调度原则是：

①如果有若干进程要求进入空闲的临界区，一次仅允许一个进程进入。②任何时候，处于临界区内的进程不可多于一个。如已有进程进入自己的临界区，则其它所有试图进入临界区的进程必须等待。③进入临界区的进程要在有限时间内退出，以便其它进程能及时进入自己的临界区。④如果进程不能进入自己的临界区，则应让出 CPU，避免进程出现“忙等”现象。

●

实时操作系统要求响应有一个截止时间，必须在截止时间前响应（硬实时），或者偶尔出现超时的情况（软实时）；

RTOS 如 uCOS-II 采用的即是基于抢占式的优先级调度算法，后来的 uCOS-III 中增加了时间片轮询的调度方法，这时允许多个任务具有同一优先级（可以分时执行）。

●

进程调度最常用的一种简单方法，是把 CPU 分配给就绪队列中具有最高优先级的就绪进程。根据已占有 CPU 的进程是否可被抢占这一原则分为抢占式优先级调度算法和非抢占式优先级调度算法两种。
在抢占式调度中，会最先执行优先级最高的进程。

●

A. 可由 CPU 直接进行读取写入操作 //错误，CPU 不能直接读取磁盘上的数据，如果可以，那还要内存干嘛，何况 CPU 与磁盘的速度差距太大，如果真要直接访问磁盘的数据的话，那 CPU 的效率也太差了。

B. 须在 CPU 访问之前移入内存 //这个有争议，看怎么理解。如果理解成：CPU 访问的数据，需要先调入内存，然后再访问，那么就正确，如果理解成：CPU 要访问的数据，必须先在内存，那就错了，因为也许在内存中找不到数据，发生缺页，然后再调入内存。不过，我更趋于前者。 毕竟，不管有没有发生缺页，在 CPU 访问数据之前，数据都是要先调入内存的。

C. 必须由文件系统管理的 //我在想，有没有什么设备可以存储数据，但又没文件系统呢？我的理解是，把数据和文件嵌入到硬件里了。那么这个时候，是不需要文件系统管理的。

D. 必须由进程调度程序管理 //错误，由文件系统管理
E. 程序和数据必须为只读 //错误，肯定可以修改的了。
F. 程序和数据只能被一个进程独占 //错误，数据和程序是可以共享的。

●

Windows 中进程间通信方式有：File, 管道（Pipe），命名管道（named pipe），信号（Signal），消息队列（Message queue），共享内存（shared memory），内存映射（memory －mapped file），信号量（semaphore），套接口（Socket），命名事件。临界区事实上应该算是由信号量来保证的。

●

SPOOLing （即 外部设备 联机并行操作），即 Simultaneous Peripheral Operation On-Line 的缩写，它是关于慢速 字符设备 如何与计算机主机交换信息的一种技术，通常称为“ 假脱机 技术”。
SPOOLing 技术特点：
(1)提高了 I/O 速度.从对低速 I/O 设备进行的 I/O 操作变为对输入井或输出井的操作,如同脱机操作一样,提高了 I/O 速度,缓和了 CPU 与低速 I/O 设备速度不匹配的矛盾.
(2)设备并没有分配给任何进程.在输入井或输出井中,分配给进程的是一存储区和建立一张 I/O 请求表.
(3)实现了虚拟设备功能.多个进程同时使用一独享设备,而对每一进程而言,都认为自己独占这一设备,不过,该设备是逻辑上的设备.

●

多道程序系统的频繁切换会耗时，平均下来单个执行时间会大于无切换的单处理机系统

●

临界资源跟临界区的区别：
临界资源：只允许一个进程进行访问的资源，比如打印机；
临界区：使用临界资源的代码区；
所以这个题目的意思就是进程间互斥，针对同类临界资源的，而对应的代码段是独立的，所以各个进程只能访问各自的代码空间。

●

最先适应算法：依次判定后找到第一个满足要求的哈
最佳适应算法：对空闲区按从小到大排序，第一个满足的就是啦
最差适应算法：对空闲区按从大到小排序，第一个满足的就是啦
固定式分区算法：是分区的

●

死锁是多个进程或者多线程竞争互斥资源时长见的问题，假如两个进程 a,b，进程 a 占据了资源 S1,申请访问资源 S2，进程 b 占据了资源 S2，申请访问 S1,两个进程互不相让，都在等待对方进程释放该资源，这就造成了死锁

●

单道程序，即在计算机内存中只允许一个的程序运行，此时程序拥有足够的资源且没有其他程序的干扰，因此其具有封闭性和再现性，所谓再现性是指在另外一时刻重新运行程序会得到相同的结果。而多道程序中， 计算机内存中同时存放几道相互独立的程序，使它们在管理程序控制下，相互穿插运行，两个或两个以上程序在计算机系统中同处于开始到结束之间的状态, 这些程序共享计算机系统资源。程序受制于资源分配和 CPU 调度的影响，不具备封闭性和再现性

●

malloc/free 和 new/delete 的本质区别：

1.malloc/free 是 C/C++语言的标准库函数，new/delete 是 C++的运算符

2.new 能自动分配空间大小

3.对于用户自定义的对象而言，用 malloc/free 无法满足动态管理对象的要求

对象在创建的时候会自动调用构造函数，对象在消亡之前自动执行析构函数

由于 malloc/free 是库函数而不是运算符，不在编译器的控制范围，不能把构造函数和析构函数的任务强加于 malloc/free 。一次 C++需要一个能够对对象完成动态分配内存和初始化工作的运算符 new，以及一个释放内存的运算符 delete。简单来说就是 new/delete 能完成跟家详细的对内存的操作，而 malloc/free 不能。

●

独享设备：在一个用户作业未完成或退出之前，此设备不能分配给其他作业用。所有字符设备都是独享设备。如输入机、磁带机、打印机等——很明显：需要装驱动。

共享设备：多个用户作业或多个进程可以“同时”从这些设备上存取信息。软硬盘、光盘等块设备都是共享设备——无需驱动。

虚拟设备：通过软件技术将独享设备改造成共享设备。例如：通过 SPOOLing 技术将一台打印机虚拟成多台打印机——实质还是独享设备，需要驱动。

●

虚存的可行性基础是计算机中著名的局部性原理。 局部性原理表现在以下两个方面:☞ 时间局部性：如果程序中的某条指令一旦执行，不久之后该指令可能再次执行；如果某数据被访问过，不久之后该数据可能再次被访问。产生时间局部性的典型原因是程序中存在着大量的循环操作。 ☞ 空间局部性：一旦程序访问了某个存储单元，在不久之后，其附近的存储单元也将被访问，即程序在一段时间内所访问的地址，可能集中在一定的范围内，这是因为指令通常是顺序存放、顺序执行的，数据也一般是以向量、数组、表等形式聚簇存储的。

●

对于单缓冲：

假定从磁盘把一块数据输入到缓冲区的时间为 T，操作系统将该缓冲区中的数据传送到用户区的时间为 M，而 CPU 对这一块数据处理的时间为 C。由于 T 和 C 是可以并行的，当 T>C 时，系统对每一块数据的处理时间为 M 十 T，反之则为 M+C，故可把系统对每一块数据的处理时间表示为 Max(C, T)+M。

对于双缓冲：

系统处理一块数据的时间可以粗略地认为是 MAC(C, T)。如果 C<T，可使块设备连续输入（图中所示情况)；如果 C>T，则可使 CPU 不必等待设备输入。对于字符设备，若采用行输入方式，则采用双缓冲可使用户在输入完第一行之后，在 CPU 执行第一行中的命令的同时，用户可继续向第二缓冲区输入下一行数据。而单缓冲情况下则必须等待一行数据被提取完毕才可输入下一行的数据。

●

Unix 把进程分成两大类：

一类是系统进程，另一类是用户进程。系统进程执行操作系统程序，提供系统功能，工作于核心态。用户进程执行用户程序，在操作系统的管理和控制下执行，工作于用户态。进程在不同的状态下执行时拥有不同的权力。

在 Unix 系统中进程由三部分组成，分别是进程控制块、正文段和数据段。Unix 系统中把进程控制块分成 proc 结构和 user 结构两部分

proc 存放的是系统经常要查询和修改的信息，需要快速访问，因此常将其装入内存

●

产生死锁的原因主要是：

（1） 因为系统资源不足。

（2） 进程运行推进的顺序不合适。

（3） 资源分配不当等。

如果系统资源充足，进程的资源请求都能够得到满足，死锁出现的可能性就很低，否则
就会因争夺有限的资源而陷入死锁。其次，进程运行推进顺序与速度不同，也可能产生死锁。

产生死锁的四个必要条件：

（1） 互斥条件：一个资源每次只能被一个进程使用。

（2） 请求与保持条件(占有等待)：一个进程因请求资源而阻塞时，对已获得的资源保持不放。

（3） 不剥夺条件:进程已获得的资源，在未使用完之前，不能强行剥夺。

（4） 环路等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之
一不满足，就不会发生死锁。

●

短作业优先又称为“短进程优先”SPN(Shortest Process Next)：这是对 FCFS 算法的改进，其目标是减少平均周转时间。

定义：

对预计执行时间短的作业（进程）优先分派处理机。通常后来的短作业不抢先正在执行的作业。

SJF 的特点：

(1) 优点：

比 FCFS 改善平均周转时间和平均带权周转时间，缩短作业的等待时间；

提高系统的吞吐量；

(2) 缺点：

对长作业非常不利，可能长时间得不到执行；

未能依据作业的紧迫程度来划分执行的优先级；

难以准确估计作业（进程）的执行时间，从而影响调度性能。

●

因为一个进程已经拥有了足够多的资源，所以该进程可以运行，所以不会出现死锁的情况，但是该进程还继续占用更多的资源，就会造成资源的浪费，使得其他需要该资源的进程无法满足而处于就绪等待资源的状态，如果长期处在等待资源状态的话就会处于饥饿状态！

●

分页式存储管理可能将连续的指令放置在不同的页中，会发生换页中断，而分段、段页都是逻辑分配空间，段长可变，逻辑上相对连续的指令放在同一段中，不会像分页那样频繁换页操作。

●

一组生产者进程和一组消费者进程共享一个初始为空、大小为 n 的缓冲区，只有缓冲区没满时，生产者才能把消息放入到缓冲区，否则必须等待；只有缓冲区不空时，消费者才能从中取出消息，否则必须等待。由于缓冲区是临界资源，它只允许一个生产者放入消息，或者一个消费者从中取出消息。

关系分析。生产者和消费者对缓冲区互斥访问是互斥关系，同时生产者和消费者又是一个相互协作的关系，只有生产者生产之后，消费者才能消费，他们也是同步关系。

●

由于硬链接是有着相同 inode 号仅文件名不同的文件，因此硬链接存在以下几点特性：

文件有相同的 inode 及 data block；

只能对已存在的文件进行创建；

不能交叉文件系统进行硬链接的创建；

不能对目录进行创建，只可对文件创建；

删除一个硬链接文件并不影响其他有相同 inode 号的文件。

软链接与硬链接不同，若文件用户数据块中存放的内容是另一文件的路径名的指向，则该文件就是软连接。软链接就是一个普通文件，只是数据块内容有点特殊。软链接有着自己的 inode 号以及用户数据块。因此软链接的创建与使用没有类似硬链接的诸多限制：
软链接有自己的文件属性及权限等；
可对不存在的文件或目录创建软链接；
软链接可交叉文件系统；
软链接可对文件或目录创建；
创建软链接时，链接计数 i_nlink 不会增加；
删除软链接并不影响被指向的文件，但若被指向的原文件被删除，则相关软连接被称为死链接（即 dangling link，若被指向路径文件被重新创建，死链接可恢复为正常的软链接）。

●

常见的页面调度算法
（1）随机算法 rand（Random Algorithm）。
利用软件或硬件的随机数发生器来确定主存储器中被替换的页面。这种算法最简单，而且容易实现。但是，这种算法完全没用利用主存储器中页面调度情况的历史信息，也没有反映程序的局部性，所以命中率较低。
（2）先进先出调度算法（FIFO）
先进先出调度算法根据页面进入内存的时间先后选择淘汰页面，本算法实现时需要将页面按进入内存的时间先后组成一个队列，每次调度队首页面予以淘汰。它的优点是比较容易实现，能够利用主存储器中页面调度情况的历史信息，但是，它没有反映程序的局部性，因为最先调入主存的页面，很可能也是经常要使用的页面。
（3）最近最少调度算法 LFU（Least Frequently Used Algorithm ）
先进先出调度算法没有考虑页面的使用情况，大多数情况下性能不佳。根据程序执行的局部性特点，程序一旦访问了某些代码和数据，则在一段时间内会经常访问他们，因此最近最少用调度在选择淘汰页面时会考虑页面最近的使用，总是选择在最近一段时间以来最少使用的页面予以淘汰。算法实现时需要为每个页面设置数据结构记录页面自上次访问以来所经历的时间。
（4）最近最不常用调度算法 LRU（Least Recently Used Algorithm）
由于程序设计中经常使用循环结构，根据程序执行的局部性特点，可以设想在一段时间内经常被访问的代码和数据在将来也会经常被访问，显然这样的页面不应该被淘汰。最近最不常用调度算法总是根据一段时间内页面的访问次数来选择淘汰页面，每次淘汰访问次数最少的页面。算法实现时需要为每个页面设置计数器，记录访问次数。计数器由硬件或操作系统自动定时清零。
（5）最优替换算法 OPT（Optimal replacement Algorithm）
前面介绍的几种页面调度算法主要是以主存储器中页面调度情况的历史信息为依据的，他假设将来主存储器中的页面调度情况与过去一段时间时间内主存储器中的页面调度情况是相同的。显然，这种假设不总是正确的。最好的算法应该是选择将来最久不被访问的页面作为被替换的页面，这种算法的命中率一定是最高的，它就是最有替换算法。要实现 OPT 算法，唯一的方法就是让程序先执行一遍，记录下实际的页地址流情况。根据这个页地址流才能找出当前要被替换的页面。显然，这样做是不现实的。因此，OPT 算法只是一种理想化的算法，然而，它也是一种很有用的算法。实际上，经常把这种算法用来作为评价其它页面调度算法好坏的标准。在其它条件相同的情况下，哪一种页面调度算法的命中率与 OPT 算法最接近，那么，它就是一种比较好的页面替换算法。

●

解决死锁问题的几种方式:
一 ．预防死锁：会损坏系统性能
1. 资源一次性分配——破坏请求和等待条件：即让进程开始运行时获得全部的资源，在不能获得全部资源时进程等待；
2. 可剥夺资源——破坏不可剥夺条件 ：即当某进程新的资源未满足时，释放已占有的资源；
3. 资源有序分配法——破坏环路等待条件 ：系统给每类资源赋予一个编号，每一个进程按编号递增的顺序请求 资源，释放则相反；
二. 避免死锁：银行家算法
三. 检测死锁

四. 解除死锁：

1.剥夺资源

2.杀死进程

●

文件分配对应于文件的物理结构，是指如何为文件分配磁盘块。常用的磁盘空间分配方 法有三种：连续分配、链接分配和索引分配。

顺序分配：顺序 分配方法要求每个文件在磁盘上占有一组连续的块。

隐式链接分配： 每个文件对应一个磁盘块的链表；磁盘块分布在磁盘的任何地方，除最后一个盘块外，每一个盘块都有指向下一个盘块的指针，这些指针对用户是透明的。

显式链接分配：是指把用于链接文件各物理块的指针，显式地存放在内存的一张链接表中。 该表在整个磁盘仅设置一张，每个表项中存放链接指针，即下一个盘块号。 在该表中，凡是 属于某一文件的第一个盘块号，或者说是每一条链的链首指针所对应的盘块号，均作为文件 地址被填入相应文件的 FCB 的“物理地址”字段中。由于查找记录的过程是在内存中进行 的，因而不仅显著地提高了检索速度，而且大大减少了访问磁盘的次数。由于分配给文件的 所有盘块号都放在该表中，故称该表为文件分配表（File Allocation Table, FAT)。MS-DOS 采用的就是这种方式。

●

分时操作系统是一种联机的多用户交互式的操作系统。一般采用时间片轮转的方式使一台计算机为多个终端服务。对每个用户能保证足够快的响应时间，并提供交互会话能力。分时操作系统感觉每个用户独占操作系统一样。

●