目录

Q1: 操作系统的功能有哪些？ 3
Q2: 操作系统的类型有哪些？ 3
Q3: 分布式操作系统和网络操作系统的区别是什么？ 3
Q4: 并行性和并发性的区别？ 3
Q5: 什么是双模工作方式，引入原因是什么，什么时候会引起模式转变？ 3
Q6: 什么是系统调用？简述系统调用的实现过程. 3
Q7: 什么是进程？ 4
Q8: 说明进程和程序之间的区别和联系 4
Q9: 从资源共享、进程创建和进程结束方面谈谈父进程和子进程的关系. 4
Q10:请画出五状态进程图，并说明进程的状态及其相互间的转换关系. 4
Q11:进程和线程的异同？ 5
Q12:多线程编程的优点 5
Q13:从内核角度看，内核级线程和用户级线程有什么不同？ 5
Q14:什么是用户线程和内核线程？它们之间的映射关系有哪些？ 6
Q15:临界区应遵循哪些访问原则？ 6
Q16:记录型信号量中对 P、V 操作的定义. 6
Q17:CPU 调度可发生在哪些情况下？哪些是可抢占式和非抢占式调度？ 7
Q18:生产者消费者问题 7
Q19:哲学家就餐问题. 8
Q20:读者写者问题 9
Q21:水果问题. 10
Q22:用管程解决生产者消费者问题 12
Q23:何为死锁？产生进程死锁产生的原因和必要条件是什么？ 13
Q24:死锁处理的基本策略. 13
Q25:死锁恢复的主要方法. 14
Q26:简述逻辑地址和物理地址的地址绑定策略 14
Q27:什么是地址的动态重定位和静态重定位？ 14
Q28:区分内存装入模块装入内存的几种方式. 15
Q29:请说明缺页中断的过程，并说明与一般中断的区别. 15
Q30:局部分配和全局分配的特点. 15
Q31:什么是抖动（颠簸），如何采用工作集模型防止抖动？ 16
Q32:两种调页策略是什么. 16

Q33:什么是文件，什么是文件系统？ 16
Q34:文件访问方式....17
Q35:文件的逻辑结构. 17
Q36:文件分配方式（文件的物理结构）及各自的优缺点？ 18
Q37:什么是文件控制块，文件控制块的典型组成. 18
Q38:什么是索引结点，索引结点的作用....18
Q39:文件目录的作用是什么？一个目录表目应包含哪些信息？ 19
Q40:文件目录项、文件目录、目录文件之间的差别和联系. 19
Q41:为什么要在设备管理种引入缓冲技术？ 19
Q42:什么是与设备无关性？有什么好处？ 19
Q43:各种磁盘调度算法的性能及特点. 20
Q44:什么是假脱机技术？ 20
Q45:什么是虚拟设备？请说明 SPOOLing 系统是如何实现虚拟设备的. 20

Q1：操作系统的功能有哪些？
A： 进程管理、内存管理、设备管理、文件系统、人机接口．

Q2：操作系统的类型有哪些？
A： 批处理系统、分时系统、实时系统、其他：网络操作系统、分布式作系统．

Q3：分布式操作系统和网络操作系统的区别是什么？
A： 分布式操作系统与网络操作系统本质上的不同之处在于分布式操作系统中，若干台计算机相互协同完成同一任务．

Q4：操作系统的结构有哪些，每种结构的简单例子？
A： 简单结构、分层方法、微内核和模块
微内核：Mach
模块：UNIX、Linux 和 MAC OS．

Q4：并行性和并发性的区别？
A： 并行性和并发性是既相似又有区别的两个概念．并行性是指两个或多个事件在同一时刻发生，并发性是指两个或多个事件在同一时间间隔内发生．

在多道程序环境下，并发性是指在一段时间内，宏观上有多个程序在同时运行，但在单处理器系统中每个时刻却仅有一道程序执行，故微观上这些程序只能分时地交替执行．倘若在计算机系统中有多个处理器，则这些可以并发执行的程序便被分配到多个处理器上，实现并行执行，即利用每个处理器来处理一个可并发执行的程序．

Q5：什么是双模工作方式，引入的原因是什么，什么时候会引起模式的转变？
A： 为了确保 OS 和所有其它程序和数据不受任何故障程序影响，CPU 至少需要两重独立的操作模式：

内核模式：操作系统管理程序运行时的状态，较高的特权级别．当 CPU 处于系统模式时，程序可以执行特权指令，访问所有资源，并可以改变处理器状态．

用户模式：用户程序运行时的状态，较低的特权级别．当 CPU 处于用户状态时，程序只能执行非特权指令．

用户模式->内核模式：

1．用户程序要求操作系统的服务，即系统调用．

2．发生一次中断．

3．用户程序中产生了一个错误状态．

3．用户程序中企图执行一条特权指令．

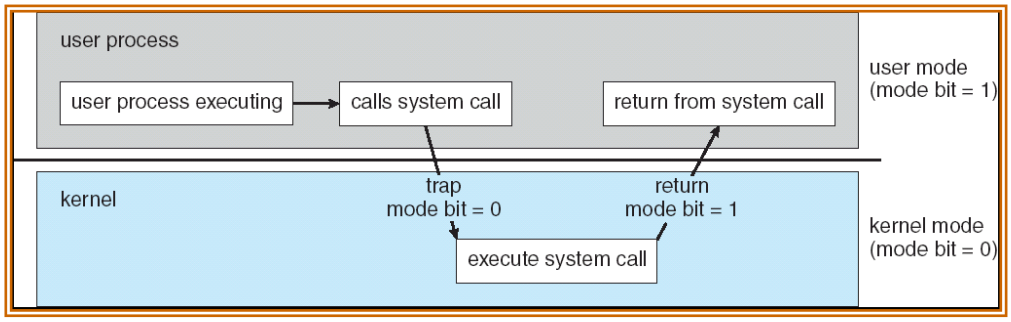
内核模式->用户模式：

从内核模式转向用户模式由一条指令实现，这条指令也是特权指令．一般是中断返回指令．

Q6：什么是系统调用？简述系统调用的实现过程．
A： 系统调用是操作系统提供给编程人员的唯一接口．编程人员利用系统调用，在源程序一级动态请求和释放系统资源，调用系统中已有的系统功能来完成那些

与机器硬件部分相关的工作以及控制程序的执行速度等.

实现过程：用户在程序中使用系统调用，给出系统调用名和函数后，即产生一条相应的陷入指令，通过陷入处理机制调用服务，引起处理机中断，然后保护处理机现场，取系统调用功能号并寻找子程序入口，通过入口地址表来调用系统子程序，然后返回用户程序继续执行.



Q7：什么是进程？

A： 进程（Process）是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位，是操作系统结构的基础．在早期面向进程设计的计算机结构中，进程是程序的基本执行实体；在当代面向线程设计的计算机结构中，进程是线程的容器．程序是指令、数据及其组织形式的描述，进程是程序的实体.

Q8：说明进程和程序之间的区别和联系.

A： 进程和程序是既又区别又有联系的两个概念.

1．进程是动态的，程序是静态的．程序是一组有序的指令集合，是一个静态的概念；进程则是程序及其数据在计算机上的一次执行，是一个动态的集合．离开了程序，进程就失去了存在的意义，但同一程序在计算机上的每次运行将构成不同的进程.

2．一个进程可以执行多个程序.

3．一个程序可以被多个进程执行.

4．程序员可以长期保存，进程只能存在一段时间．程序是永久存在的，进程则有从创建到消亡的生命周期.

Q9：从资源共享、进程创建和进程结束三个方面谈谈父进程和子进程的关系.

A： 资源共享方面：父进程创建子进程时，子进程可能从操作系统那里直接获得资源，也可以从父进程那里获得．父进程可能必须在子进程之间共享资源.

进程创建方面：子进程被父进程创建时，除了得到各种物理和逻辑资源外，初始化数据（或输入）由父进程传递给子进程.

进程结束方面：子进程使用了超过它所分配的资源，或者分配给子进程的任务已不再需要，父进程将终止子进程．父进程退出，如果父进程终止，那么操作系统（处特殊的操作系统）不允许子进程继续.

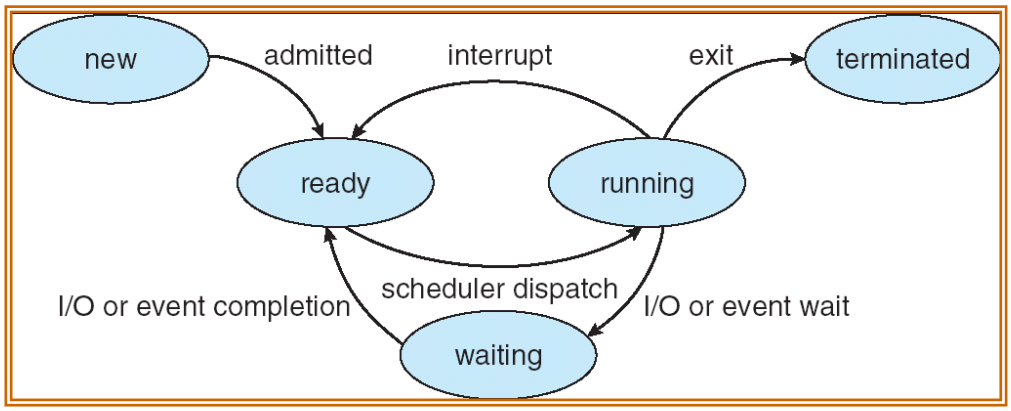
Q10:请画出五状态进程图，并说明进程的状态及其相互间的转换关系.

A： 就绪–运行：被调度程序选中

运行–就绪：时间片到时,或有更高优先级的进程出现

运行–等待：等待某事件发生

等待–就绪：等待的事件发生了



Q11:进程和线程的异同？

A: **1．调度**

线程是调度的基本单位，同一进程中的线程切换不会引起进程切换．

2．并发

在引入线程的操作系统中不仅进程之间可以相互切换亦可并发执行．

3．资源拥有

进程拥有资源，是资源分配的基本单位；

线程不拥有资源，但它可以访问创建它的进程的资源．

4．系统开销

创建或撤销进程时系统都要为之分配和回收资源，操作系统所付出的开销大于在创建或撤销线程时的开销．

5．上下文切换

线程的上下文切换的代价比进程小很多．

Q12:多线程编程的优点．

A: **1．响应度高：**对一个交互程序采用多线程能够增加对用户的响应程度．

2．资源共享：线程共享所属进程的内存和资源．

3．经济性：由于线程共享它们所属进程的资源，所以创建和切换线程更为经济．

4．多处理器结构体系结构的运用：多线程能够充分使用多处理器结构，以便每个进程能并行运行在不同的处理器上．不管有多少 CPU，单线程进程只能运行在一个 CPU 上，在多 CPU 上使用多线程增加了并发功能．

Q13:从内核角度看，内核级线程和用户级线程有什么不同？

A: 用户级线程仅存在于用户级中，它的创建、撤消和切换都不利用系统调用实现，与内核无关，相应的，内核也不知道有用户级线程存在．

内核级线程依赖于内核，无论用户进程中的线程还是系统进程中的线程，其创建、撤消、切换都由内核实现．在内核中保留了一张线程控制块，内核根据控制块感知线程的存在并对其进行控制．

1．线程的调度与切换速度：内核支持线程的调度和切换与进程的调度和切换十分相似．对于用户级线程的切换，通常是发生在一个应用程序的多线程之

间，这时，不仅无须通过中断进入 OS 的内核，而且切换的规则也远比进程调度和切换的规则简单．因此，用户级线程的切换速度特别快．

2．系统调用：当传统的用户进程调用一个系统调用时，要由用户态转入核心态，用户进程将被阻塞．当内核完成系统调用而返回时，才将该进程唤醒，继续执行．而在用户级线程调用一个系统调用时，由于内核并不知道有该用户级线程的存在，因而把系统调用看作是整个进程的行为，于是使该进程等待，而调度另一个进程执行，同样是在内核完成系统调用而返回时，进程才能继续执行．如果系统中设置的是内核支持线程，则调度是以线程为单位．当一个线程调用一个系统调用时，内核把系统调用只看作是该线程的行为，因而阻塞该线程，于是可以再调度该进程中的其他线程执行．

Q14:什么是用户线程和内核线程？它们之间的映射关系有哪些？

A: 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.

内核线程：由操作系统内核创建和撤销. 内核维护进程及线程的上下文信息以及线程切换. 一个内核线程由于 I/O 操作而阻塞，不会影响其它线程的运行.

用户线程与和心态线程之间的映射关系有一对一、多对一、多对多．

Q15:临界区应遵循哪些访问原则？

A: 空闲则入

忙则等待

有限等待

让权等待

Q16:记录型信号量中对 P、V 操作的定义．

A:

```
P(semaphore* S)
{
    S->value--;
    if (value < 0)
    {
        add this process to S->list
        block();
    }
}

V(semaphore* S)
{
    S->value++;
    if (value <= 0)
    {
        remove a process P from S->list
        wakeup(P);
```

```
    }
}
```

Q17:CPU 调度可发生在哪些情况下？哪些情况是可抢占式调度？哪些是非抢占式调度？

A: 1．正在执行的进程执行完毕．

2．执行中进程自己调用阻塞原语．

3．执行中进程调用了 P 原语操作，从而因资源不足而被阻塞；或调用了 V 原语操作激活了等待资源的进程队列．

4．执行中进程提出 I/O 请求后被阻塞．

5．在分时系统中时间片已经用完．

6．在执行完系统调用，在系统程序返回用户进程时，可认为系统进程执行完毕，从而可调度选择一新的用户进程执行．

7．就绪队列中的某进程的优先级变的高于当前执行进程的优先级，从而也将引发进程调度．

可抢占式调度：7

非抢占式调度：1，2，3，4，5，6

Q18:一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；只有缓冲区不为空时，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，或一个消费者从中取出消息．

A:

```
semaphore mutex = 1; // 临界区互斥信号量
semaphore empty = n; // 空缓冲区同步信号量
semaphore full = 0;  // 满缓冲区同步信号量

producer()
{
    while (1)
    {
        ...
        // produce an item in nextp
        ...
        P(empty);
        P(mutex);
        ...
        // add nextp to buffer
        ...
        V(mutex);
        V(full);
    }
}
```

```
consumer()
{
    while (1)
    {
        P(full);
        P(mutex);
        ...
        // remove an item from buffer
        ...
        V(mutex);
        V(empty);
        ...
        // comsume the item
        ...
    }
}
```

Q19:一张圆桌上坐着 5 个哲学家,每两个哲学家之间的桌子上摆一根筷子，桌子中间是一碗米饭．哲学家们倾注毕生精力用于思考和进餐,哲学家在思考时，并不影响他人．只有当哲学家饥饿的时候，才试图拿起左右两根筷子(一根一根的拿起)．如果筷子在他人手里,则需等待．饥饿的哲学家只有同时拿到了两只筷子才能开始进餐,当进餐完毕时,放下筷子继续思考．

```
semaphore chopsticks[5] = {1, 1, 1, 1, 1};
semaphore mutex = 1;

P(i)
{
    while(1)
    {
        P(mutex);
        P(chopsticks[i]);
        P(chopsticks[(i + 1) % 5]);
        V(mutex);
        ...
        // eat
        ...
        v(chopsticks[i]);
        v(chopsticks[(i + 1) % 5]);
        ...
        // think
        ...
    }
}
```

Q20:有读者和写者两组并发进程，共享一个文件，当两个或以上的读进程同时访问共享数据时不会产生副作用，但若某个写进程和其他进程（读进程或写进程）同时访问共享数据时则可能导致数据不一致的错误．因此要求：①允许多个读者可以同时对文件执行读操作；②只允许一个写者往文件中写信息；③任一写者在完成写操作之前不允许其他读者或写者工作；④写者执行写操作前，应让已有的读者和写者全部退出．

A:

```
// 读者优先
int count = 0;
semaphore mutex = 1; //互斥信号量
semaphore rw = 1; //互斥信号量

writer()
{
    while (1)
    {
        P(rw);
        ...
        // writing
        ...
        V(rw);
    }
}

reader()
{
    while (1)
    {
        P(mutex);
        if (count==0)
            P(rw);
        count++;
        V(mutex);
        ...
        // reading
        ...
        P(mutex);
        count--;
        if (count==0)
            V(rw);
        V(mutex);
    }
}
```

```
// 写者优先
int count=0;
semaphore mutex=1; // 互斥信号量
semaphore rw=1; // 互斥信号量

writer ()
{
    while (1)
    {
        P(rw);
        ...
        // writing
        ...
        V(rw);
    }
}

reader ()
{
    while (1)
    {
        P(mutex);
        if (count==0)
            P(rw);
        count++;
        V (mutex);
        ...
        // reading
        ...
        P (mutex) ;
        count--;
        if (count==0)
            V(rw);
        V (mutex);
    }
}
```

Q21:桌子上有一只盘子，每次只能向其中放入一个水果．爸爸专向盘子中放苹果，妈妈专向盘子中放橘子，女儿专等吃盘子中的苹果，儿子专等吃盘子中的橘子．只有盘子为空时，爸爸或妈妈就可向盘子中放一只水果；仅当盘子中有自己需要的水果时，儿子或女儿可以从盘子中取出．

A:

```
semaphore plate = 1;  // 互斥信号量
```

```
semaphore apple = 0;  // 同步信号量
semaphore orange = 0; // 同步信号量

father()
{
    while (1)
    {
        ...
        // prepare an apple
        ...
        P(plate);
        ...
        // put the apple on the plate
        ...
        V(apple);
    }
}

daughter()
{
    while (1)
    {
        P(apple);
        ...
        // take an apple from the plate
        ...
        V(plate);
        ...
        // eat the apple
        ...
    }
}

mother()
{
    while (1)
    {
        ...
        // prepare an orange
        ...
        P(plate);
        ...
        // put the orange on the plate
        ...
```

```
        V(orange);
    }
}

son()
{
    while (1)
    {
        P(orange);
        ...
        // take an orange from the plate
        ...
        V(plate);
        ...
        // eat the orange
        ...
    }
}
```

Q22:用管程解决生产者消费者问题.

A:

```
monitor pc
{
    int count = 0;
    condition empty;
    condition full;

    void add(item)
    {
        if (count == BUFFER_SIZE)
            wait(empty);
        put_item(item);
        count++;
        if (count == 1)
            signal(full);
    }

    Item remove()
    {
        if (count == 0)
            wait(full);
        item = remove_item();
        count--;
        if (count == BUFFER_SIZE - 1)
```

```
            signal(empty);
        return item;
    }
}

void producer()
{
    while (true)
    {
        item = produce();
        pc.add(item);
    }
}

void consumer()
{
    while (true)
    {
        item = pc.remove();
        consume(item);
    }
}
```

Q23:何为死锁？产生进程死锁产生的原因和必要条件是什么？

A: 死锁定义:

所谓死锁是指多个进程在执行过程中，由于竞争资源因竞争共享资源而造成的一种僵局，若无外力作用，它们都将无法推进下去.

死锁原因:

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

死锁必要条件:

1. 互斥：一个资源每次只能被一个进程使用
2. 占有并等待：一个进程因请求资源而阻塞时对占有的资源保持不放
3. 不可抢占：进程已获得的资源在末使用完之前不能抢占
4. 循环等待：若干进程之间形成一种头尾相接的循环等待资源关系

Q24:死锁处理的基本策略.

A: 1. 预防死锁

通过设置一些限制条件，去破坏产生死锁的必要条件，以防止发生死锁.

2. 避免死锁

在资源分配过程中，使用某种方法避免系统进入不安全的状态，从而避免发生死锁.

3. 死锁的检测及解除

无需采取任何限制措施，允许死锁的发生，通过系统的检测机制及时检测出死锁的发生，然后采取某种措施，解除死锁．

预防死锁和避免死锁都属于事先预防策略，但是预防死锁的限制条件比较严格，实现较为简单，但往往导致系统的效率低，资源利用率低；避免死锁的限制条件相对宽松，资源分配后需要通过算法来判断是否进入不安全状态，实现起来较为复杂．

Q25:死锁恢复的主要方法．

A: 资源抢占：从进程中选择牺牲品，抢占其资源给其他进程使用，直到死锁环被打破为止．

1．选择适当的牺牲品：可以按优先级高低和撤销进程代价的高低来进行．

2．回滚：被牺牲的进程必须回滚到某个安全状态，以便从该状态重启进程．

3．饥饿：防止同一个进程总是被抢占，得不到资源，从而发生饥饿问题．

进程终止：终止所有的死锁进程或者一次终止一个直到死锁环解除为止．

Q26:简述逻辑地址和物理地址的地址绑定策略．

A: 编译时（compile time）：如果在编译时就知道进程将在内存中的驻留地址，那么可以生成绝对代码．如果将来开始地址发生变化，那么就必须重新编译代码．

加载时（load time）：当编译时不知道进程将驻留在内存的什么地方，那么编译器就必须生成可重定位代码．绑定会延迟到加载时才进行．如果开始地址发生变化．只需要重新加载用户代码已引入改变值．

执行时（execution time）：如果进程在执行时可以从一个内存段移到另一个内存段，那么绑定必须延迟到执行时才发生．绝大多数通用计算机操作系统采用这种方法．

Q27:什么是地址的动态重定位和静态重定位？

A: 静态重定位：静态重定位是在目标程序装入内存时，由装入程序对目标程序中的指令和数据的地址进行修改，即把程序的逻辑地址都改成实际的地址．对每个程序来说，这种地址变换只是在装入时一次完成，在程序运行期间不再进行重定位．

优点：无需增加硬件地址转换机构，便于实现程序的静态连接．在早期计算机系统中大多采用这种方案．

缺点：

1．程序的存储空间只能是连续的一片区域，而且在重定位之后就不能再移动，这不利于内存空间的有效使用．

2．各个用户进程很难共享内存中的同一程序的副本．

动态重定位：动态重定位是在程序执行期间每次访问内存之前进行重定位．这种变换是靠硬件地址变换机构实现的．通常采用一个重定位寄存器，其中放有当前正在执行的程序在内存空间中的起始地址，而地址空间中的代码在装入过程中不发生变化．

优点：

1．程序占用的内存空间动态可变，不必连续存放在一处．

2．比较容易实现几个进程对同一程序副本的共享使用．

缺点：需要附加的硬件支持，增加了机器成本．

现在一般计算机系统中都采用动态重定位方法．

Q28:区分内存装入模块装入内存的几种方式．

A: 1.绝对装入方式（Abosolute Loading Mode）

在编译时，如果程序知道将驻留在内存的什么位置，那么，编译程序将产生绝对地址的目标代码．绝对装入方式按照装入模块中的地址，将程序和数据装入内存．装入模块被装入内存后，由于程序中的逻辑地址和实际内存地址完全相同，所以不需要对程序和数据的地址进行修改．

2.可重定位装入方式（Relocation Loading Mode）

绝对装入方式能将目标模块装入到内存中事先指定的位置．在多道程序环境下，编译程序不可能预知所编译的模块应该放在内存的何处，因此，绝对装入方式只适用于单道程序环境．在多道程序环境下，所得到的目标模块起始地址通常从 0 开始，程序中的其他地址都是相对于起始地址 0 计算的，此时应该采用可重定位装入方式，根据内存当时的情况，将装入模块装入到内存的合适位置．

值得注意的是：在采用重定位装入方式将程序装入内存后，会使装入模块中的所有逻辑地址和内存的物理地址不同，解决方法是：在装入的时候，将装入模块中指令和数据的逻辑地址修改为物理地址，这一过程就叫 重定位 ．又因为地址变换是在装入时一次性完成的，以后不会再改变，所以称为 静态重定位．

3.动态运行时装入方式（Dynamic Run-time Loading）

静态重定位方式可将装入模块装入到内存中任何允许的位置，所以可以用于多道程序环境；但是这种装入方式并不允许程序运行时在内存中移动位置．因为程序一旦移动的话，就必须修改程序和数据的绝对地址．然而，实际情况是，程序在运行的过程中在内存的位置可能经常要改变，此时就应该采用动态重定位的方式．

动态重定位的装入程序在把模块装入内存后，并不会立即把模块中的相对地址转换为物理地址，而是 把这种地址转换推迟到程序真正要执行时才进行．因此，装入内存后的所有地址仍然是相对地址，为了使地址转换不影响指令的执行速度，这种方式需要一个重定位寄存器来存储模块在内存中的起始地址．

Q29:请说明缺页中断的过程，并说明与一般中断的区别．

A: 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）．

缺页中断作为中断同样要经历，诸如保护 CPU 环境、分析中断原因、转入缺页中断处理程序、恢复 CPU 环境等几个步骤．但与一般的中断相比，它有以下两个明显的区别：

1．在指令执行期间产生和处理中断信号，而非一条指令执行完后，属于内部中断．

2．一条指令在执行期间，可能产生多次缺页中断．

Q30:局部分配和全局分配的特点．

A: 全局替换：允许一个进程从所有帧集合中选择一个置换帧，而不管该帧是否已分配给其他进程，即一个进程可以从另一个进程中拿到帧．

局部替换：每个进程仅从其自己的分配帧中进行选择．

全局替换中单个进程不能控制页错误率，一个进程内的页集合不但取决于本身的进程调页行为，还取决于其他进程的调页行为．

局部替换中进程内存中的页只受该进程本身的调页行为所影响，这一定程度上会限制抖动的发生，但是因为局部置换不能使用其他进程的不常用内存，所以会阻碍一个进程．因此全局置换通常有更好的系统吞吐量．

Q31:什么是抖动（颠簸），如何采用工作集模型防止抖动？

A: 抖动：在页面置换过程中的一种最糟糕的情形是，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．如果一个进程在换页上用的时间多于执行时间，那么这个进程就在颠簸．

频繁的发生缺页中断（抖动），其主要原因是某个进程频繁访问的页面数目高于可用的物理页帧数目．虚拟内存技术可以在内存中保留更多的进程以提高系统效率．在稳定状态，几乎主存的所有空间都被进程块占据，处理机和操作系统可以直接访问到尽可能多的进程．但如果管理不当，处理机的大部分时间都将用于交换块，即请求调入页面的操作，而不是执行进程的指令，这就会大大降低系统效率．

工作集：工作集是指在某段时间间隔内，进程要访问的页面集合．经常被使用的页面需要在工作集中，而长期不被使用的页面要从工作集中被丢弃．为了防止系统出现抖动现象，需要选择合适的工作集大小．

工作集模型的原理是让操作系统跟踪每个进程的工作集，并为进程分配大于其工作集的物理块．如果还有空闲物理块，则可以再调一个进程到内存以增加多道程序数．如果所有工作集之和增加以至于超过了可用物理块的总数，那么操作系统会暂停一个进程，将其页面调出并且将其物理块分配给其他进程，防止出现抖动现象．

Q32:两种调页策略是什么．

A: 预调页策略：根据局部性原理，一次调入若干个相邻的页可能回避一次调入一页更高效．但如果调入的一批页面中大多数都未被访问，则又是低效的．所以就需要采用以预测为基础的预调页策略，将预计在不久之后便会访问的页面预先调入内存．但面前预调页的成功率仅约 50%．故这种策略主要用于进程的首次调入时，由程序员指出应该先调入哪些页．

请求调页策略：进程在运行钟需要访问的页面不在内存而提出请求，由系统将所需页面调入内存．由这种策略调入的页一定会被访问，且这种策略比较易于实现呢，故在目前的虚拟存储器中大多采用此策略．它的缺点在于一次只调入一页，调入调出页面数多时会花费过多的 I/O 开销．

Q33:什么是文件，什么是文件系统？

A: 文件是以计算机硬盘为载体存储在计算机上的信息集合，它的形式很多样化，可以是文本文档、图片、程序等等．操作系统中负责管理和存储文件信息的软件机构称为文件系统，文件系统由三部分组成：与文件管理有关软件、被管理

文件以及实施文件管理所需的数据结构．

Q34:文件访问方式．

A: 顺序访问：最为简单的访问方式，文件信息按顺序，一个记录接着一个记录地加以处理．这种访问模式最为常用，例如编辑器和编译器通常是按照这种方式来访问文件的．

直接访问：文件由固定长度的逻辑记录组成，以允许程序按任意顺序进行快速读和写．直接访问方式是基于文件的磁盘模型，这是因为磁盘允许对任意文件块进行随机读和写．对于直接访问文件，读写顺序是没有限制的．直接读写访问文件可立即访问大量的信息，所以极为有用．数据库通常使用这种类型的文件．

其他访问方式：这些访问通常涉及创建文件索引．索引包括各块的指针．为了查找文件中的记录，首先搜索索引，再根据指针直接访问文件，以查找所需要的记录．

Q35:文件的逻辑结构．

A: 逻辑结构：从用户观点出发看到的文件的组织形式，是用户可以直接处理的数据及其结构．独立于文件的物理特性．也称为文件组织．根据逻辑结构，文件可以分为两个大类：无结构文件和有结构文件．

1．无结构文件

最简单的文件组织形式．将数据按照顺序组织成记录（记录指的是一组相关数据项的集合）并积累保存，是有序相关信息项的集合．以字节为单位．对记录的搜索只能是穷举搜索．适用于存储源代码，目标代码等．

2．有结构文件

顺序文件：文件中的记录一个接一个的顺序排列，记录可以是定长的或变长的，可以顺序存储或以链表形式存储，在访问时需要顺序搜索文件．

索引文件：对于定长记录文件，如果要查找第 **i** 个记录，可直接根据下式计算来获得第 **i** 个记录相对于第一个记录的地址：

$$A_i = i \times L$$

然而，对于可变长记录的文件，要查找第 **i** 个记录时，必须顺序地查找前 **i-1** 个记录，从而获得相应记录的长度 **L**，然后才能按下式计算出第 **i** 个记录的首址：

$$A_i = \sum_{i=0}^{i-1} L_i + i$$

变长记录文件只能顺序查找，系统开销较大．为此可以建立一张索引表以加快检索速度，索引表本身是定长记录的顺序文件．在记录很多或是访问要求高的文件中，需要引入索引以提供有效的访问．实际中，通过索引可以成百上千倍地提高访问速度．

索引顺序文件：索引顺序文件是顺序和索引两种组织形式的结合．索引顺序文件将顺序文件中的所有记录分为若干个组，为顺序文件建立一张索引表，在索引表中为每组中的第一个记录建立一个索引项，其中含有该记录的关键字值和指向该记录的指针．

散列文件：给定记录的键值或通过 **Hash** 函数转换的键值直接决定记录的物理地址．这种映射结构不同于顺序文件或索引文件，没有顺序的特性．

Q36:文件分配方式（文件的物理结构）及各自的优缺点？

A: 1．连续分配

连续分配方法要求每个文件在磁盘上占用一组连续的块．

优点：实现简单、存取速度快，支持随机访问．

缺点：文件长度不宜动态增加，因为一个文件末尾后的盘块可能已经分配给其他文件，一旦需要增加，就需要大量移动盘块．此外，反复增删文件后会产生外部碎片（与内存管理分配方式中的碎片相似），并且很难确定一个文件需要的空间大小，因而只适用于长度固定的文件．

2．链接分配

每个文件是磁盘块的链表；分布在磁盘的任何地方．目录包括文件第一块的指针和最后一块的指针．

优点：简单，仅需要起始地址；无外碎片，显著提高磁盘利用率；文件动态增长时磁盘块可以动态分配；对文件的增删查改非常方便．

缺点：无法实现随机访问；可靠性差（指针丢失、错误指针）；指针需要空间．

解决方式：多个块组成簇（Cluster），按簇分配．减少了指针占用空间比例，代价是增加了内碎片．

文件分配表(FAT)：在每个卷的起始位置存储 FAT，记录文件块顺序，每块在 FAT 表中有一项，FAT 表可以通过块号码来索引．缓存的 FAT 改善了随机访问时间．

3．索引分配

以文件为对象，把所有相关指针放在一起，即索引块．

索引分配需要索引表，支持随机访问和动态存取，并且没有外碎片，但索引块的负担较重．

Q37:什么是文件控制块，文件控制块的典型组成．

A: 文件控制块（File Control Block，FCB）是用来存放控制文件所需要的各种信息的数据结构，以实现「按名存取」．

文件控制块的典型组成：

1．文件权限

2．文件大小

3．文件日期（创建、访问、写）

4．文件所有者、组、访问控制列表

5．文件数据块或文件控制指针

Q38:什么是索引结点，索引结点的作用．

A: 在检索目录文件的过程中，只用到了文件名，仅当找到一个目录项（查找文件名与目录项中文件名匹配）时，才需要从该目录项中读出该文件的物理地址．也就是说，在检索目录时，文件的其他描述信息不会用到，也不需调入内存．因此，UNIX 采用了文件名和文件描述信息分开的方法，文件描述信息单独形成一个称为索引结点的数据结构．在文件目录中的每个目录项仅由文件名和该文件对应的索引结点编号构成．

系统读取文件，首先在目录中查找文件名对应的索引结点编号，再通过索引结点编号获取索引结点信息，最后根据索引结点信息，找到文件数据块读出数

据.

Unix 中 iNode 的组成:

1．文件长度：以字节为单位

2．文件主标志符：拥有该文件的个人或小组的标志符

3．文件权限：文件的读、写、执行权限

4．文件时间戳：文件最近存取时间、最近修改时间和 iNode 最近修改时间

5．文件链接计数：所有指向该文件的文件名的指针计数

6．文件物理地址：直接或间接的方式指向文件的数据块．直接块指向数据，间接块指向索引．

Q39:文件目录的作用是什么？一个目录表目应包含哪些信息？

A: 目录在用户需要的文件和文件名之间提供一种映射，目录管理要实现「按名存取」．

目录相关操作：搜索文件、创建文件、删除文件、遍历目录、重命名文件和跟踪文件系统等等．

内容有：文件名，文件号，用户名，文件地址，文件长度，文件类型，文件属性，共享计数，文件的建立日期，保存期限，最后修改日期，最后访问日期，口令，文件逻辑结构，文件物理结构．

Q40:文件目录项、文件目录、目录文件之间的差别和联系．

A: 文件目录项:为了使文件控制块与文件一一对应，而其中的每一个文件控制块被称为文件目录项．

文件目录：为实现「按名存取」，必须建立文件名与文件的映射关系，体现这种映射关系的数据结构称为文件目录．

目录文件：存放文件的一组相关信息的集合．

他们三者之间的关系是包含与被包含的关系，但也存在区别，目录文件存储有文件目录项，文件目录也包含文件目录项的相关信息，但是目录文件并不包含全部的文件目录．

Q41:为什么要在设备管理种引入缓冲技术？

A: 解决设备间速度的不匹配；

减少 CPU 的中断频率；

提高 CPU 和 I/O 设备之间的并行性．

Q42:什么是与设备无关性？有什么好处？

A: 为了提高 OS 的可适应性和可扩展性，在现代 OS 中都毫无例外地实现了设备独立性，也称设备无关性．

基本含义：应用程序独立于具体使用的物理设备．为了实现设备独立性而引入了逻辑设备和物理设备两概念．

在应用程序中，使用逻辑设备名称来请求使用某类设备；而系统在实际执行时，还必须使用物理设备名称．

优点:

1．增加设备分配时的灵活性

2．易于实现 I/O 重定向（用于 I/O 操作的设备可以更换），而不必改变应

用程序.

Q43:各种磁盘调度算法的性能及特点.

A: 1.先来先服务(First Come First Served, FCFS)调度:根据进程请求访问磁盘的先后顺序进行调度.

优点:简单,公平.

缺点:效率不高,相邻两次请求可能会造成最内到最外的柱面寻道,使磁头反复移动,增加了服务时间,对机械也不利.

2.最短寻道优先(Shortest Seek Time First, SSTF)调度:选择调度处理的磁道是与当前磁头所在磁道距离最近的磁道,以使每次的寻找时间最短.

优点:改善了磁盘平均服务时间.

缺点:造成某些访问请求长期等待得不到服务,即出现"饥饿"现象.

3.扫描(SCAN)调度:在磁头当前移动方向上选择与当前磁头所在磁道距离最近的请求作为下一次服务的对象.

优点:克服了最短寻道优先的缺点,既考虑了距离,同时又考虑了方向 .

缺点:存在一个请求刚好被错过而需要等待较长时间的问题.

4.循环扫描(Cicular SCAN, C-SCAN)调度:在扫描算法的基础上规定磁头单向移动来提供服务,回返时直接快速移动至起始端而不服务任何请求.

优点:兼顾较好的寻道性能和防止饥饿现象,同时减少一个请求可能等待的最长时间.

缺点:-

5.LOOK(C-LOOK)调度:SCAN(C- SCAN)调度的改进,即磁头移动只需要到达最远端的一个请求即可返回,不需要到达磁盘端点.

Q44:什么是假脱机技术?

A: 假脱机技术即SPOOLing(Simultaneous Peripheral Operating On-Line).SPOOLing是指在联机情况下实现的同时外围操作,也称假脱机输入输出操作,它是操作系统中的一项将独占设备改为共享设备的技术.SPOOLing是低速输入输出设备与主机交换的一种技术,通常也称为"假脱机真联机",它的核心思想是以联机的方式得到脱机的效果.

SPOOLing技术是用于将一台独占设备改造成共享设备的一种行之有效的技术.当系统中出现了多道程序后,可以利用其中的一道程序,来模拟脱机技术输入时的外围控制机的功能,把低速输入输出设备上的数据传送到高速磁盘上;再用另一道程序来模拟脱机输出时外围控制机的功能,把数据从磁盘传送到低速输出设备上.这样便可以在主机的直接控制下,实现脱机输入、输出功能.

Q45:什么是虚拟设备?请说明SPOOLing系统是如何实现虚拟设备的.

A: 虚拟设备是指通过虚拟技术,可将一台独占设备变换成若干台逻辑设备,供若干个用户(进程)同时使用.由于多台逻辑设备实际上并不存在,而只是给用户的一种感觉,因此被称为虚拟设备.

SPOOLing系统中,作业执行前,操作系统已将作业通过独占设备预先输入到磁盘或磁鼓上一个特定的存储区域(称之为"输入井")存放好,称为"预输入",此后,作业执行使用数据时不用再启动独占设备读入,而把这一要求转换成从辅助存储器中读入.另一方面,作业执行中,也不必直接启动独占设备输出数据,而

只要将作业输出数据写入磁盘或磁鼓中的特定存储区域（称之为"输出井"）存放，当作业执行完毕后，由操作系统通过相应的输出设备来组织信息输出，称为"缓输出"．这样做可以提高独占设备的利用率，缩短作业的执行时间．这样，由于一台设备可以和辅助存储器中的若干存储区域相对应，所以在形式上就好像把一台输入设备（或输出设备）变成了许多虚拟的输入设备（或输出设备），即把一台不能共享的输入设备（或输出设备）转换成了一台可共享的缓冲输入设备（或输出设备），使用户产生一个"错觉"，好像他们各自都有一台专用的字符设备，从而实现虚拟设备．