王道考研——操作系统

WWW.CSKAOYAN.COM

第二章 进程管理

历年真题考频统计

章节	索引	核心考点	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	小题考频	大题考频	章节考频
Ch1	1	操作系统的概念、特征和功能	#23							#23	#28	#23				#23	#23	6		27+3
	2	内核态与用户态			#24	#23	#28	#25	#24		&45				#23	#27	#26	8	1	
	3	中断、异常				#24			#23		#32 &45	#29		#25			#24	6	1	
	4	系统调用		#23							#24		#25		#32	#31		5		
	5	操作系统引导													&46	#24		1	1	
Ch2	6	进程与线程			#25	#31		#31		#24			#23	#29				6		52+16
	7	进程状态与进程控制		#24 #26				#26	#25			#27	#24		#24	#28	#27	9		
	8	处理机调度	#24		#23	#29 #30	#31	#23		&46	#23 #27	#24	#27	#26	#25 #27	#25	#29	15	1	
	9	进程同步与互斥	&45	#25 #27	#32 &45		&45	&47	&45	#27 #30 #31	&46	#25 #28 #32		#32	&45		&45	10	8	
	10	经典同步问题	&45		&45		&45		&45				&43	&45		&46			7	
	11	死锁	#25		#27	#27	#32	#24	#26	#25		#26	#30	#27	#31	#26		12		
Ch3	12	内存管理的概念			#30		#29											2		30+13
	13	连续分配管理方式	#26	#28							#25		#32					4		
	14	非连续分配管理方式	#27 &46	#29 &46			&46	#32	&46	#28	&45	&45	#28 #31	&46			#25	7	7	
	15	虚拟页式存储管理	&46	&46	#28 #29	#25 &45	#30	#30	#27 #30	#26 #29	&45	&45	#29	#28 &46	#28 #29	#29 #30	#28 #30	17	6	
Ch4	16	文件元数据和索引节点		#30			#26					&46		#31		&45		3	2	32+18
	17	文件的操作				#28	#23	#29			#31			#23	#30	&45	#31	6	1	
	18	文件的逻辑结构和物理结构	#28	#30	&46	&46	#24 #26	&46	#29	&47		&46		#24		&45		6	6	
	19	文件共享和文件保护	#30 #31								#30			#23				4		
	20	目录结构和操作		#31	&46		#23			&47					#30	&45		3	3	
	21	磁盘的组织与管理	#29	&45		#32		#27	#31		#26 #29	#30 #31	#26 &44		#26 &46			10	3	
Ch5	22	I/O控制方式															&46		1	12+2
	23	I/O软件的层次结构	#32	#32	#26	#26	#25							#30		#32	&46	7	1	
	24	高速缓存与缓冲区			#31		#27		#28									3		
	25	设备分配与回收															#32	1		
	26	SPOOLing 技术								#31								1		

第二章（进程管理）命题重点

【命题重点】

1．进程和线程的比较，内核支持线程和用户级线程。

2．进程的状态变化，进程的创建与终止，进程的阻塞与唤醒。

3．作业运行的顺序与甘特图，处理机调度的时机，各种调度算法的特点，特别是高响应比优先调度和多级反馈队列调度算法的原理。

4．进程的并发执行，临界区互斥的软件实现方法，信号量机制的原理，掌握掌握经典的同步互斥问题并能灵活应用。

5．死锁的判断、安全序列，银行家算法。

考点6：进程与线程

2009	2010	2011	2012	2013	2014	2015	2016
		• 小题 25	• 小题 31		• 小题 31		• 小题 24

2017	2018	2019	2020	2021	2022	2023
		• 小题 23	• 小题 29			

历年考频：小题×6、综合题×0

操作系统考点6

进程与线程

进程与线程

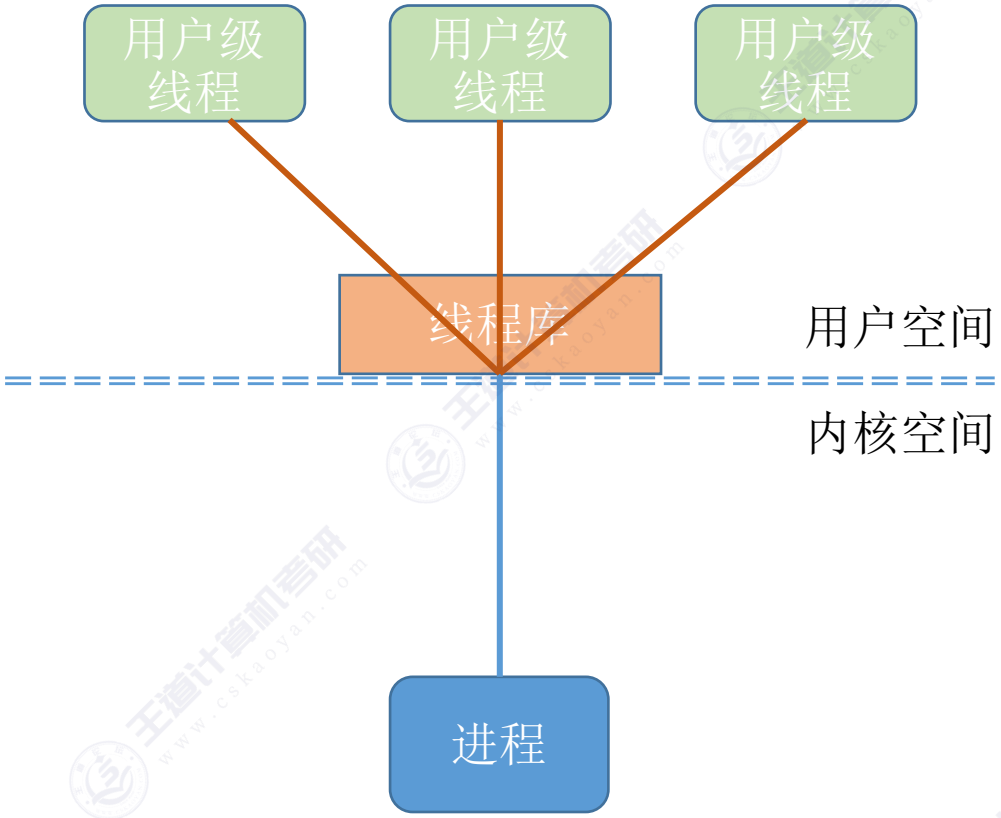
【考点笔记】进程与线程的比较

	进程	线程
映像组成	由程序段、相关数据段和 PCB 构成。	共享其隶属进程的进程映像，仅拥有线程 ID、寄存器集合和堆栈等。
并发性	在没有引入线程的系统中，进程是独立运行的基本单位。	线程是独立运行的基本单位①，一个进程可以拥有一个或多个线程。
资源分配	进程是资源分配和拥有的基本单位	线程自己基本不拥有系统资源，但它可访问所属进程所拥有的全部资源。
调度	在没有引入线程的操作系统中，进程是独立调度和分派的基本单位。	引入线程后的操作系统中，线程是独立调度和分派的基本单位。
通信	PV 操作；共享存储； 消息传递；管道通信。	同一进程的各线程直接读写进程数据段。 不同进程的线程之间通信属于进程间通信。
系统开销	进程切换时涉及当前 CPU 环境的保存及新进程 CPU 环境的设置，开销较大。	线程切换时只需保存和设置少量寄存器内容，开销很小。
地址空间	进程的地址空间之间互相独立	同一进程的各线程间共享进程的地址空间

①线程运行的独立性是指线程之间独立运行，但线程不能脱离进程运行

用户级线程 vs 内核级线程

用户级线程（User-Level Thread, ULT）



用户级线程由应用程序通过线程库实现。

所有的线程管理工作都由应用程序负责（包括线程切换）

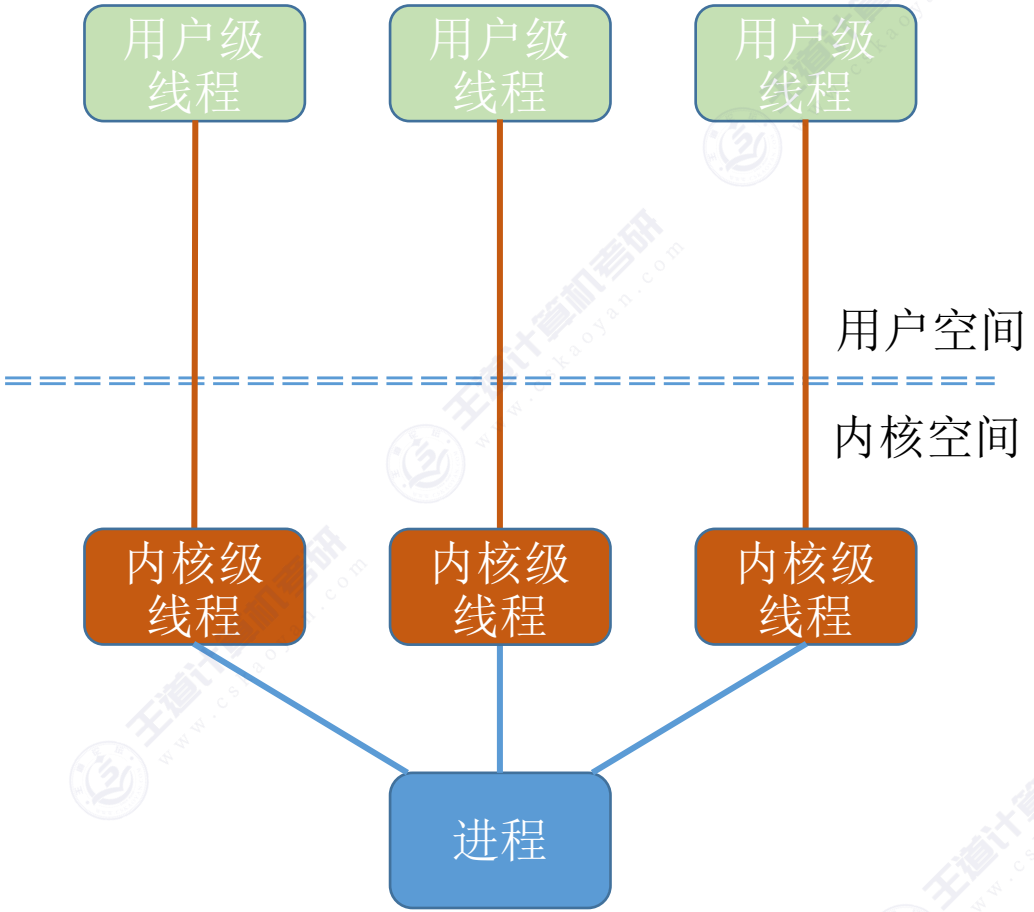
用户级线程中，线程切换可以在用户态下即可完成，无需操作系统干预。

在用户看来，是有多个线程。但是在操作系统内核看来，并意识不到线程的存在。（用户级线程对用户不透明，对操作系统透明）

可以这样理解，“用户级线程”就是“从用户视角看能看到的线程”

用户级线程 vs 内核级线程

内核级线程（Kernel-Level Thread, KLT, 又称“内核支持的线程”）

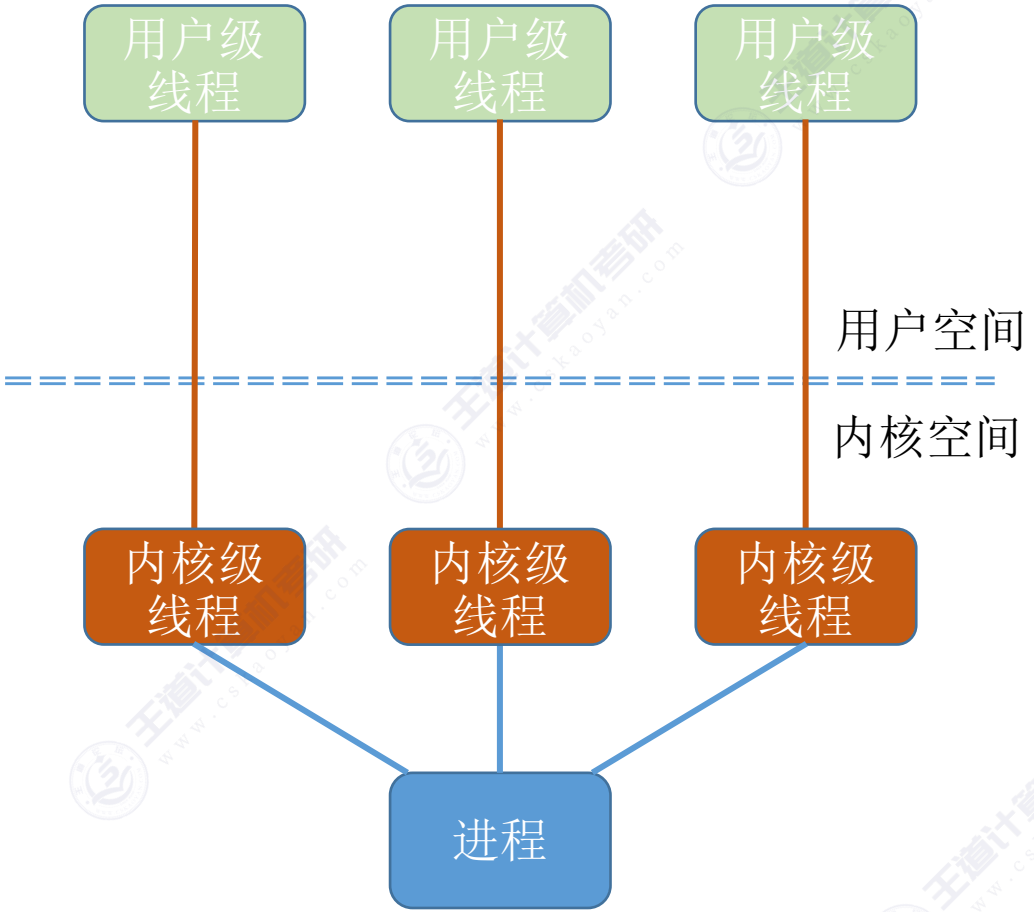


内核级线程的管理工作由操作系统内核完成。线程调度、切换等工作都由内核负责，因此内核级线程的切换必然需要在核心态下才能完成。

可以这样理解，“内核级线程”就是“从操作系统内核视角看能看到的线程”

用户级线程 vs 内核级线程

内核级线程（Kernel-Level Thread, KLT, 又称“内核支持的线程”）

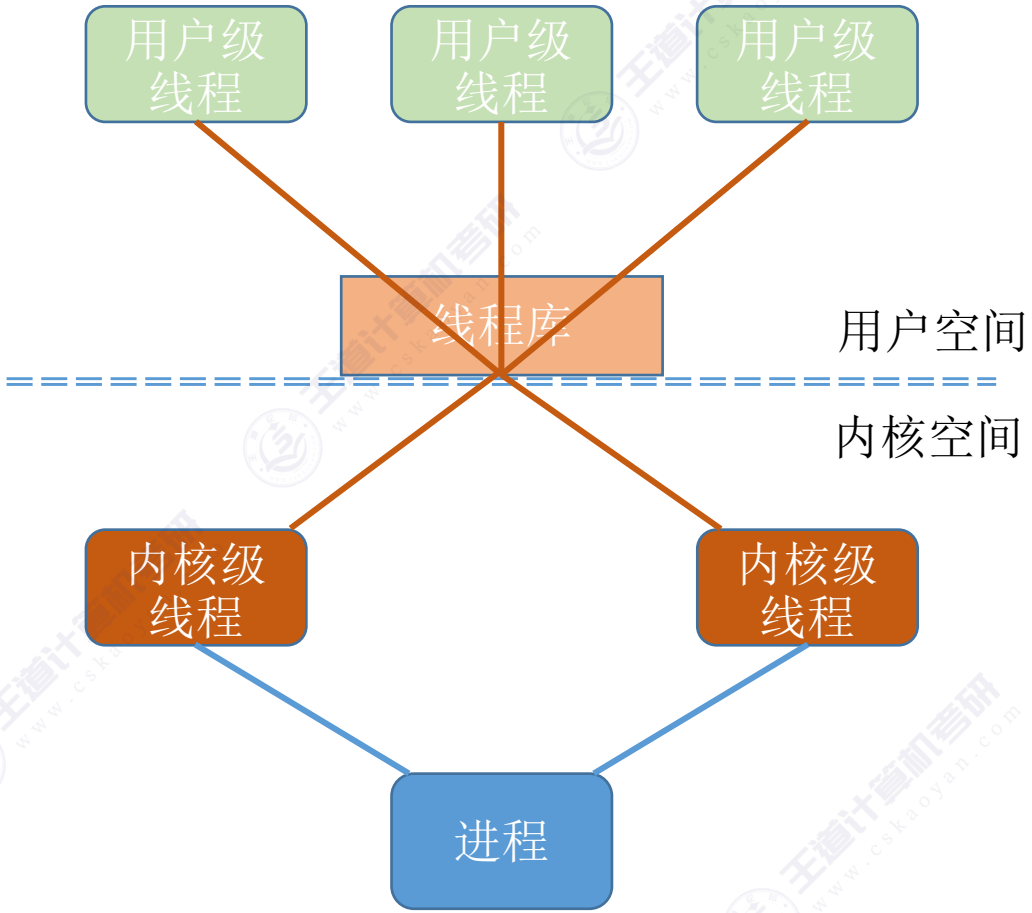


内核级线程的管理工作由操作系统内核完成。线程调度、切换等工作都由内核负责，因此内核级线程的切换必然需要在核心态下才能完成。

可以这样理解，“内核级线程”就是“从操作系统内核视角看能看到的线程”

用户级线程 vs 内核级线程

在同时支持用户级线程和内核级线程的系统中，可采用二者组合的方式：将n个用户级线程映射到m个内核级线程上（ n >= m）



重点重点重点：
操作系统只“看得见”内核级线程，因此只有内核级线程才是处理机分配的单位。

例如：左边这个模型中，该进程由两个内核级线程，三个用户级线程，在用户看来，这个进程中有三个线程。但即使该进程在一个4核处理机的计算机上运行，也最多只能被分配到两个核，最多只能有两个用户线程并行执行。

用户级线程 vs 内核级线程

【考点笔记】内核支持线程和用户级线程

线程	用户级线程	内核支持线程
定义	存在于用户空间中。线程的创建、撤消、同一进程的线程之间的切换等功能都在用户空间实现。内核不知道用户级线程的存在。	是在内核的支持下运行的。线程的创建、撤消和切换等都在内核空间实现。内核根据线程控制块感知线程的存在。
优点	线程切换不需要转换到内核空间（对同一进程的线程而言），从而节省了开销。 调度算法可以是进程专用的。 用户级线程的实现与操作系统平台无关，对于线程管理的代码都属于用户程序。	在多处理器系统中，内核能够同时调度同一进程中多个线程并行执行； 一个线程被阻塞了，内核可以调度该进程中的其他线程或其他进程中的线程运行； 内核本身也可以采用多线程技术，可以提高系统的执行速度和效率。
缺点	当一个线程被阻塞，同一进程内的所有线程都会被阻塞。 多线程应用不能利用多处理器系统的优点。内核以进程为单位分配 CPU，因此同一时刻一个进程中仅有一个线程能执行	同一个进程中，线程切换时，需要从用户态转到内核态进行，系统开销较大

考点7：进程状态与进程控制

2009	2010	2011	2012	2013	2014	2015	2016
	• 小题24 • 小题26				• 小题26	• 小题25	

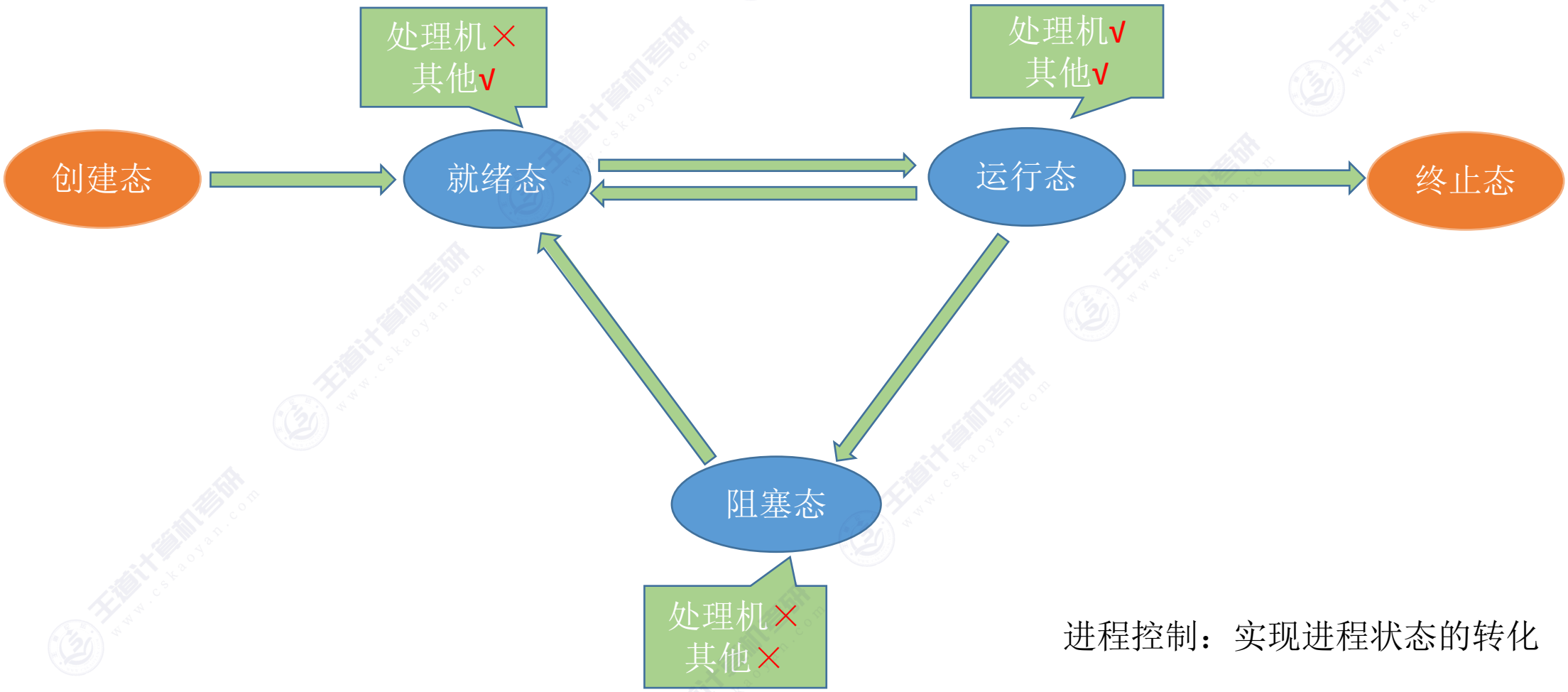
2017	2018	2019	2020	2021	2022	2023
	• 小题27	• 小题24		• 小题24	• 小题28	• 小题27

历年考频：小题×9、综合题×0

操作系统考点7

进程状态与进程控制

进程状态与进程控制



进程控制：实现进程状态的转化

进程状态与进程控制

通过撤销原语完成：就绪态/阻塞态/运行态→终止态→无

通过创建原语完成：无→创建态→就绪态

【考点笔记】进程的创建与终止

	进程创建	进程终止
引发事件	终端用户登录系统。 作业调度。 系统提供服务。 用户程序的应用请求。	正常结束。 异常结束，由于发生异常（如存储区越界）而终止进程。 外界干预，进程应外界请求而终止运行。
过程	申请空白 PCB。 为新进程分配资源。 初始化 PCB。 如果进程就绪队列能够接纳新进程，就将新进程插入到就绪队列。	根据被终止进程的标识符，检索 PCB，从中读出该进程的状态。 若该进程处于执行状态，则终止其执行，将处理器资源分配给其他进程。 若该进程还有子进程，则将子进程终止。 释放该进程所拥有的全部资源。 将该 PCB 从所在队列中删除。

进程状态与进程控制

【考点笔记】进程的阻塞与唤醒

	进程阻塞	进程唤醒
引发事件	请求系统服务。 启动某种操作。 新数据尚未到达。 无新工作可做。 （需要等待某个事件的发生时引发阻塞）	引发阻塞的事件完成。 （等待的事件发生时唤醒进程）
特点	阻塞：运行状态→阻塞状态 进程通过调用阻塞原语 block 把自己阻塞。进程的阻塞是进程自身的一种主动行为。	唤醒：阻塞状态→就绪状态 由完成相关事件的进程调用 wakeup 原语将阻塞进程唤醒，是被动完成的。

考点8：处理机调度

2009	2010	2011	2012	2013	2014	2015	2016
• 小题24		• 小题23	• 小题29 • 小题30	• 小题31	• 小题23		• 综合题46

2017	2018	2019	2020	2021	2022	2023
• 小题23 • 小题27	• 小题24	• 小题27	• 小题26	• 小题25 • 小题27	• 小题25	• 小题29

历年考频：小题×15、综合题×1

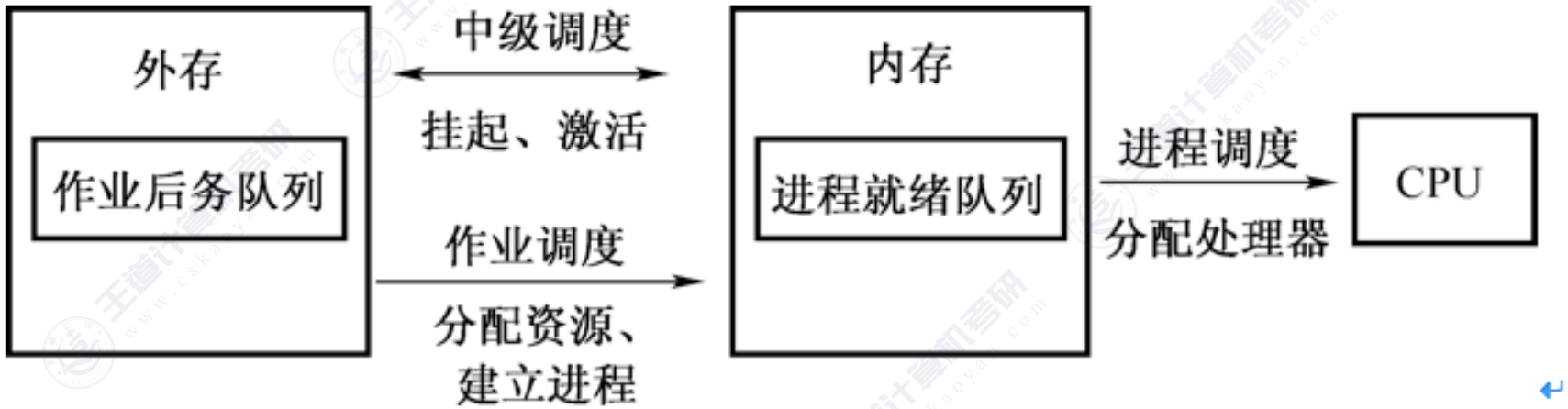
操作系统考点8

处理机调度

处理机调度

【考点笔记】调度的基本概念

调度从作业提交到完成可以分为：作业调度、中级调度和进程调度三个层次：



处理机调度

【考点笔记】调度的原则

指标
CPU 利用率
系统吞吐量
周转时间
平均周转时间
带权周转时间
平均带权周转时间
等待时间
响应时间

处理机调度

【考点笔记】调度的原则

指标	定义	原则
CPU 利用率	CPU 的工作时间在整个系统工作时间中所占的比例	高
系统吞吐量	系统吞吐量表示单位时间内 CPU 完成作业的数量	高
周转时间	周转时间=作业完成时间-作业提交时间	短
平均周转时间	平均周转时间=(作业 1 的周转时间+…+作业 n 的周转时间)/n	短
带权周转时间	带权周转时间=作业周转时间/作业实际运行时间	小
平均带权周转时间	平均带权周转时间=(作业 1 的带权周转时间+…+作业 n 的带权周转时间)/n	小
等待时间	进程处于等待处理器状态的时间之和	短
响应时间	从用户提交请求到系统首次产生响应所用的时间	短

处理机调度

【考点笔记】进程调度的优先级

根据进程创建后其优先级是否可以改变，可以将进程优先级分为以下两种：

1）静态优先级。在进程投入运行前就确定一个优先级，并且之后一直不变。

2）动态优先级。在进程运行过程中，根据进程情况的变化动态调整优先级。

通常：
系统进程优先级 高于 用户进程；
前台进程优先级 高于 后台进程；
I/O型进程（或称 I/O繁忙型进程）优先级 高于 计算型进程（CPU繁忙型进程）；

处理机调度

【考点笔记】不能进行处理机调度的情况

在操作系统内核程序运行时，如果某时发生了引起进程调度的因素，并不一定能够马上进行调度与切换。不能进行进程的调度与切换的情况有以下几种：

1）在处理中断的过程中：中断处理过程复杂，在实现上很难做到进程切换，而且中断处理是系统工作的一部分，逻辑上不属于某一进程，不应被剥夺处理机资源。

2）进程在操作系统内核程序临界区中：进入临界区后，需要独占式地访问共享数据，理论上必须加锁，以防止其他并行程序进入，在解锁前不应切换到其他进程运行。

3）其他需要完全屏蔽中断的原子操作过程中：如加锁、解锁、中断现场保护、恢复等原子操作。在原子过程中，连中断都要屏蔽，更不应该进行进程调度与切换。

如果在上述过程中发生了引起调度的条件，并不能马上进行调度和切换，应置系统的请求调度标志，直到上述过程结束后才进行相应的调度与切换。

处理机调度

重点思考：
①算法思想；②算法规则；③优缺点；④是否会导致饥饿

【考点笔记】典型的调度算法

算法	调度方法	特点
先来先服务调度算法	作业：每次从后备作业队列中选择最先进入该队列的作业调入内存。 进程：每次从就绪队列中选择最先进入该队列的进程，将处理器分配给它，使之投入运行	算法简单，但效率低； 有利于长作业不利于短作业； 有利于 CPU 繁忙型作业而不利于 I/O 繁忙型作业
短作业/进程优先调度算法	作业：每次从后备队列中选择一个或若干个估计运行时间最短的作业，将它们调入内存。 进程：每次从就绪队列中选择一个估计运行时间最短的进程，将处理器分配给它，使之投入运行	对长作业不利； 估计运行时间可能不准确； 平均等待时间、平均周转时间最少
高响应比优先调度算法	先计算后备作业队列中每个作业的响应比，从中选出响应比最高的作业投入内存。 $响应比R_P=\frac{等待时间+要求服务时间}{要求服务时间}$ 综合考虑了作业的等待时间和估计运行时间	有利于短作业； 长作业不至于产生饥饿； 等待时间越长，优先级越高

处理机调度

算法	调度方法	特点
优先级调度算法	作业：每次从后备作业队列中选择优先级最高的一个或几个作业，将它们调入内存。 进程：每次从就绪队列中选择优先级最高的进程，将处理器分配给它，使之投入运行	优先级高的先运行。优先级分为——静态优先级：创建时确定且不变。动态优先级：创建时确定并动态改变
时间片轮转调度算法	所有就绪进程按先来先服务排队，总是选择就绪队列中第一个进程运行一个时间片。然后释放处理器给下一个就绪进程，而被剥夺处理器的进程返回到就绪队列的末尾重新排队	使所有用户在给定时间内得到响应。时间片大：系统开销小但交互性差。时间片小：有利于短作业，但系统开销大
多级反馈队列调度算法（适用于分时系统进程调度）	① 设置多个就绪队列，优先级越高的队列进程的运行时间片越小。优先级按第 1～n 级队列递减。 ② 当一个新进程进入内存后，首先将它放入第 1 级队列的末尾，按 FCFS 原则排队等待调度。 当轮到位于第 i 级队列的某一进程执行时，若它在该时间片内完成，便可撤离系统，否则： ◆第 1～(n-1)级队列：将该进程转入下一级队列的末尾，等待下次调度运行。 ◆第 n 级队列：采用时间片轮转的方式运行。 ③ 仅当第 1～(i-1)级队列均为空时，才会调度第 i 级队列中的进程运行。 ④ 第 i 级队列中的进程运行时，若又有新进程进入第 1～(i-1)中的任一队列，则把正在运行的进程放回第 i 级队列的末尾，把处理器分配给新进程	终端型作业用户：短作业优先。 短批处理作业用户：周转时间较短。 长批处理作业用户：不会长期得不到处理

考点9：进程同步与互斥

2009	2010	2011	2012	2013	2014	2015	2016
• 综合题45	• 小题25 • 小题27	• 小题32 • 综合题45		• 综合题45	• 综合题47	• 综合题45	• 小题27 • 小题30 • 小题31

2017	2018	2019	2020	2021	2022	2023
• 综合题46	• 小题25 • 小题28 • 小题32		• 小题32	• 大题45		• 综合题45

历年考频：小题×10、综合题×8

操作系统考点9

进程同步与互斥

进程同步与互斥

为了实现对临界资源的互斥访问，同时保证系统整体性能，需要遵循以下原则：

1. 空闲让进。临界区空闲时，可以允许一个请求进入临界区的进程立即进入临界区；
2. 忙则等待。当已有进程进入临界区时，其他试图进入临界区的进程必须等待；
3. 有限等待。对请求访问的进程，应保证能在有限时间内进入临界区（保证不会饥饿）；
4. 让权等待。当进程不能进入临界区时，应立即释放处理机，防止进程忙等待。

互斥的四种软件实现方式——单标志法、双标志先检查法、双标志后检查法、Peterson算法。
记忆要点：结合生活经验理解代码背后的逻辑，不要一头钻进代码分析，这样不易理解也难以记忆。

进程同步与互斥

【考点笔记】临界区互斥的软件实现方法

对于两个进程 P0、P1 互斥访问临界区的软件实现方法：

方法	算法与特点

单标志法	设置一个公用整型变量 turn，用于指示被允许进入临界区的进程编号，即若 turn=0，则允许 P0 进程进入临界区。
	P0 进程：
	while(turn!=0);　　//进入区
	critical section;　　//临界区
	turn=1;　　//退出区
	remainder section;　　//剩余区
	P1 进程：
	while(turn!=1);
	critical section;
	turn=0;
	remainder section;

特点	确保进程互斥访问临界区。 两个进程必须交替进入临界区。

检查自己是否拥有访问临界区的权限

将临界区的访问权让给另一个进程

违背“空闲让进”

进程同步与互斥

flag[i] 表示 Pi 进程想要进入临界区的意愿。初始时 flag[0]=false, flag[1]=false

检查对方是否想进入临界区，如果对方想进入，自己就循环等待

方法	算法与特点
双标志法先检查	设置了一个数组 flag[2]，表示进程是否进入临界区。若 flag[i]值为 FALSE，表示 Pi 进程未进入临界区；值为 TRUE，表示 Pi 进程进入临界区（i=0 或 1）。 P0 进程：　　P1 进程： while(flag[1]); ①　　while(flag[0]); ②　　//进入区 flag[0]=TRUE; ③　　flag[1]=TRUE; ④　　//进入区 critical section; ⑤　　critical section; ⑥　　//临界区 flag[0]=FALSE;　　flag[1]=FALSE;　　//退出区 remainder section;　　remainder section;　　//剩余区
特点	进程不必交替进入，可连续使用； 两进程可能同时进入临界区，无法实现互斥访问。

上锁

违背“忙则等待”

进程同步与互斥

flag[i] 表示 Pi 进程想要进入临界区的意愿。初始时 flag[0]=false, flag[1]=false

方法	算法与特点
双标志法后检查	同样设置了 flag[2]，采用先设置自己标志后，再检测对应状态标志，若对方标志位 TURE，则进程等待；否则进入临界区。 P0 进程：　　P1 进程： flag[0]=TRUE; ①　　flag[1]=TRUE; ②　　//进入区　上锁 while(flag[1]); ③　　while(flag[0]);④　　//进入区　检查 critical section;　　critical section;　　//临界区 flag[0]=FLASE;　　flag[1]=FLASE;　　//退出区 remainder section;　　remainder section;　　//剩余区
特点	确保进程互斥访问临界区。 存在两个进程都进不了临界区的“饥饿”现象。

违背“空闲让进”、“有限等待”（会饥饿）

进程同步与互斥

turn 表示优先让哪个进程进入临界区。类似于“孔融让梨”

flag 表示各进程想要进入临界区的意愿。初始时 flag[0]=false, flag[1]=false

方法	算法与特点
皮特森算法	设置变量 turn 以指示不允许进入临界区的进程编号，每个进程在先设置自己的 flag 标志后再设置 turn 标志，不允许另一个进程进入。这时，再同时检测另一个进程 flag 标志和不允许进入标志 turn。若对方 flag 标志位为 TRUE 且 turn 显示不允许对方进入临界区，则进程进入临界区；否则等待。
特点	确保进程互斥访问临界区。 不会出现“饥饿”现象。

表明自己想进入临界区的意愿

表示可以优先让对方进入

```
P0 进程：                                  P1 进程：
flag[0]=TURE;turn=1;                       flag[1]=TRUE;turn=0;          //进入区
while(flag[1]&&turn==1);                   while(flag[0]&&turn==0);      //进入区
critical section;                          critical section;             //临界区
flag[0]=FLASE;                             flag[1]=FLASE;                //退出区
remainder section;                         remainder section;            //剩余区
```

若对方也想进入，且最后是自己做出了“谦让动作”，则自己等待

不满足“让权等待”原则，暂时不能进入临界区的进程还会占用处理机

turn 变量的设置——单标志法的思想
flag 数组的设置——双标志法的思想

进程同步与互斥

【考点笔记】信号量机制

整型信号量被定义为一个用于表示资源个数的整型量 S。当进程发现 S≤0 时，就会不断测试。因此进程处于忙等状态，未遵循“让权等待”原则。

记录型信号量遵循了“让权等待

```
typedef struct{
    int value;
    struct process *L;
} semaphore;
```

```
int S = 1;  //初始化整型信号量s，表示当前系统中可用的打印机资源数

void wait (int S) {    //wait 原语，相当于“进入区”
  while (S <= 0);      //如果资源数不够，就一直循环等待
  S=S-1;               //如果资源数够，则占用一个资源
}

void signal (int S) { //signal 原语，相当于“退出区”
  S=S+1;              //使用完资源后，在退出区释放资源
}
```

信号量的 P、V 操作的实质是“

号量（只能对信号量，不可以对普通

塞等待，“否”则什么都不做，而 V 操

则唤醒进程，“否”则什么也不做。

记录型信号量 S 中 value 值的意义：

- S.value 的初值表示系统中某类资源的总数。
- S.value<0 表示当前系统中已经没有可用的该类资源。
- S.value<0 时，其绝对值表示 S.L 中因等待该资源而阻塞的进程个数。

考点10：经典同步问题

2009	2010	2011	2012	2013	2014	2015	2016
• 综合题45		• 综合题45		• 综合题45		• 综合题45	

2017	2018	2019	2020	2021	2022	2023
		• 综合题43	• 综合题45		• 综合题46	

历年考频：小题×0、综合题×7

操作系统考点10

经典的同步问题

生产者-消费者问题

生产者、消费者共享一个初始为空、大小为n的缓冲区。
只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。
只有缓冲区不空时，消费者才能从中取出产品，否则必须等待。
缓冲区是临界资源，各进程必须互斥地访问。

```
semaphore mutex = 1;        //互斥信号量，实现对缓冲区的互斥访问
semaphore empty = n;        //同步信号量，表示空闲缓冲区的数量
semaphore full = 0;         //同步信号量，表示产品的数量，也即非空缓冲区的数量
```

```
producer (){
    while(1){
        生产一个产品;
        P(empty);
        P(mutex);
        把产品放入缓冲区;
        V(mutex);
        V(full);
    }
}
```

消耗一个空闲缓冲区

增加一个产品

```
consumer (){
    while(1){
        P(full);
        P(mutex);
        从缓冲区取出一个产品;
        V(mutex);
        V(empty);
        使用产品;
    }
}
```

消耗一个产品（非空缓冲区）

增加一个空闲缓冲区

能否改变两个P操作的顺序？

```
producer (){
    while(1){
        生产一个产品;
        P(mutex);        ①
        P(empty);        ②
        把产品放入缓冲区;
        V(mutex);
        V(full);
    }
}
```

mutex 的P操作在前

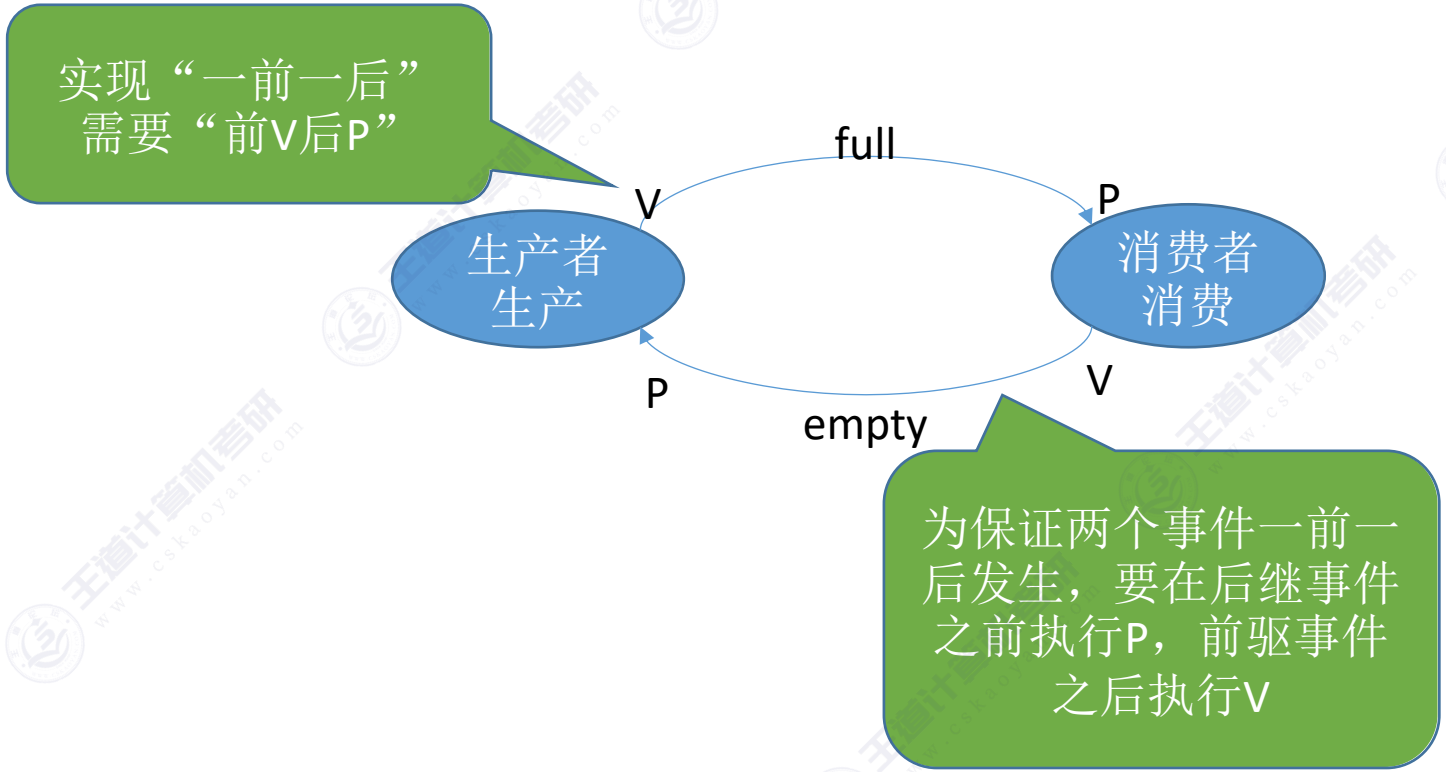
```
consumer (){
    while(1){
        P(mutex);        ③
        P(full);         ④
        从缓冲区取出一个产品;
        V(mutex);
        V(empty);
        使用产品;
    }
}
```

若此时缓冲区内已经放满产品，则 empty=0，full=n。按①②③的顺序执行会发生死锁
若缓冲区中没有产品，即full=0，empty=n。按③④① 的顺序执行就会发生死锁。
因此，实现互斥的P操作一定要在实现同步的P操作之后。

两个V操作交换顺序不会导致死锁

生产者-消费者问题

生产者消费者问题是一个互斥、同步的综合问题。
最难的是发现题目中隐含的两对同步关系。
缓冲区满时，生产者进程需要等待消费者进程取走产品；缓冲区空时，消费者进程需要等待生产者进程放入产品。这是两个不同的“一前一后问题”（即两个同步问题），因此也需要设置两个同步信号量。



易错点：实现互斥和实现同步的两个P操作的先后顺序

读者-写者问题

有读者和写者两组并发进程，共享一个文件，当两个或两个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。

写者与写者需要互斥，写者与读者也需要互斥，但读者与读者不互斥。

解决“读者与写者互斥，但读者与读者不互斥”的核心在于设置了一个计数器 count 用来记录当前正在访问共享文件的读进程数。我们可以用count 的值来判断当前进入的进程是否是第一个/最后一个读进程，从而做出不同的处理。

读者-写者问题

```
semaphore rw=1;        //用于实现对文件的互斥访问。表示当前是否有进程在访问共享文件
int count = 0;         //记录当前有几个读进程在访问文件
semaphore mutex = 1;//用于保证对count变量的互斥访问
```

```
writer (){
    while(1){
        P(rw);    //写之前"加锁"
        写文件…
        V(rw);    //写之后"解锁"
    }
}
```

思考：若两个读进程并发执行，则两个读进程有可能先后执行 P(rw)，从而使第二个读进程阻塞的情况。

如何解决：出现上述问题的原因在于对 count 变量的检查和赋值无法一气呵成，因此可以设置另一个互斥信号量来保证各读进程对count 的访问是互斥的。

```
reader (){
    while(1){
        P(mutex);       //各读进程互斥访问count
        if(count==0)
            P(rw);      //第一个读进程负责"加锁"
        count++;        //访问文件的读进程数+1
        V(mutex);
        读文件…
        P(mutex);       //各读进程互斥访问count
        count--;        //访问文件的读进程数-1
        if(count==0)
            V(rw);      //最后一个读进程负责"解锁"
        V(mutex);
    }
}
```

潜在的问题：只要有读进程还在读，写进程就要一直阻塞等待，可能“饿死”。因此，这种算法中，读进程是优先的

如何实现

```
semaphore rw=1;          //用于实现对文件的互斥访问
int count = 0;           //记录当前有几个读进程在访问文件
semaphore mutex = 1;     //用于保证对count变量的互斥访问
semaphore w = 1;         //用于实现"写优先"
```

分析以下并发执行 P(w) 的情况：
读者1→写者1→读者2
写者1→读者1→写者2

写者不会饥饿，是相对公平的先来先服务原则。有的书上把这种算法称为“读写公平法”。

```
writer (){
    while(1){
        P(w);
        P(rw);
        写文件…
        V(rw);
        V(w);
    }
}
```

```
reader (){
    while(1){
        P(w);
        P(mutex);
        if(count==0)
            P(rw);
        count++;
        V(mutex);
        V(w);
        读文件…
        P(mutex);
        count--;
        if(count==0)
            V(rw);
        V(mutex);
    }
}
```

读者-写者问题

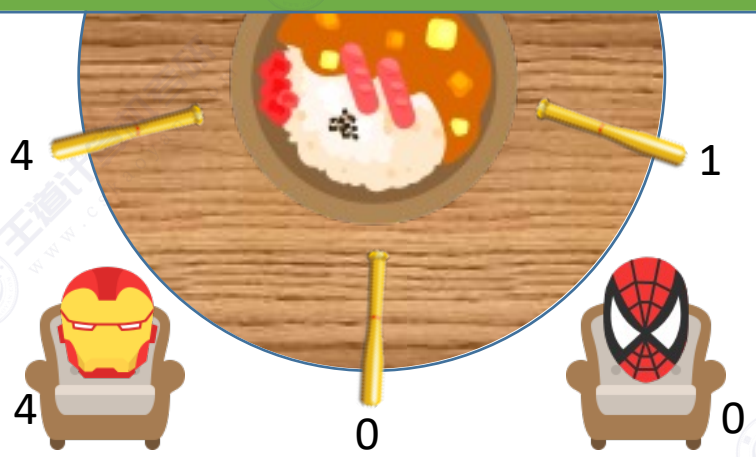
读者-写者问题为我们解决复杂的互斥问题提供了一个参考思路。
其核心思想在于设置了一个计数器 count 用来记录当前正在访问共享文件的读进程数。我们可以用 count 的值来判断当前进入的进程是否是第一个/最后一个读进程，从而做出不同的处理。
另外，对 count 变量的检查和赋值不能一气呵成导致了一些错误，如果需要实现“一气呵成”，自然应该想到用互斥信号量。
最后，还要认真体会我们是如何解决“写进程饥饿”问题的。

绝大多数的考研PV操作大题都可以用之前介绍的几种生产者-消费者问题的思想来解决，如果遇到更复杂的问题，可以想想能否用读者写者问题的这几个思想来解决。

哲学家进餐问题

一张圆桌上坐着5名哲学家，每两个哲学家之间的桌上摆一根筷子，桌子的中间是一碗米饭。哲学家们倾注毕生的精力用于思考和进餐，哲学家在思考时，并不影响他人。只有当哲学家饥饿时，才试图拿起左、右两根筷子（一根一根地拿起）。如果筷子已在他人手上，则需等待。饥饿的哲学家只有同时拿起两根筷子才可以开始进餐，当进餐完毕后，放下筷子继续思考。

mutex 保证了同一时刻最多只有一个哲学家正在“拿筷子”。也就是说，最多只有一个哲学家处于“请求和保持”的状态（占有了一只筷子的同时申请另一支筷子）。别的哲学家要么手里没有筷子，要么有两支筷子。手里有两支筷子的哲学家一定可以顺利运行结束，并释放筷子，唤醒处于“请求和保持”状态的哲学家




```
semaphore chopstick[5]={1,1,1,1,1};
semaphore mutex = 1;     //互斥地取筷子
Pi (){                   //i号哲学家的进程
    while(1){
        P(mutex);
        P(chopstick[i]);             //拿左
        P(chopstick[(i+1)%5]);       //拿右
        V(mutex);
        吃饭…
        V(chopstick[i]);             //放左
        V(chopstick[(i+1)%5]);       //放右
        思考…
    }
}
```

哲学家进餐问题

哲学家进餐问题的关键在于解决进程死锁。

这些进程之间只存在互斥关系，但是与之前接触到的互斥关系不同的是，每个进程都需要同时持有两个临界资源，因此就有“死锁”问题的隐患。

如果在考试中遇到了一个进程需要同时持有多个临界资源的情况，应该参考哲学家问题的思想，分析题中给出的进程之间是否会发生循环等待，是否会发生死锁。

考点11：死锁

2009	2010	2011	2012	2013	2014	2015	2016
• 小题25		• 小题27	• 小题27	• 小题32	• 小题24	• 小题26	• 小题25

2017	2018	2019	2020	2021	2022	2023
	• 小题26	• 小题30	• 小题27	• 小题31	• 小题26	

历年考频：小题×12、综合题×0

操作系统考点11

死锁

死锁

【考点笔记】死锁条件

产生死锁必须同时满足以下四个条件，只要其中任一个条件不成立，死锁就不会发生。

- 互斥条件：在一段时间内某资源仅为一个进程所占有。此时若有其他进程请求该资源，则请求进程只能等待。
- 不剥夺条件：进程所获得的资源在未使用完毕之前，不能被其他进程强行夺走，即只能由获得该资源的进程自己来释放。
- 请求和保持条件：进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源已被其他进程占有，此时请求进程被阻塞，但对自己已获得的资源保持不放。
- 循环等待条件：存在一种进程资源的循环等待链，链中每一个进程已获得的资源同时被链中下一个进程所请求。

1. 预防死锁。破坏死锁产生的四个必要条件中的一个或几个。
2. 避免死锁。用某种方法防止系统进入不安全状态，从而避免死锁（银行家算法）
3. 死锁的检测和解除。允许死锁的发生，不过操作系统会负责检测出死锁的发生，然后采取某种措施解除死锁。（Key：资源分配图——两种结点？两种边分别代表什么？什么是“可完全简化”？）

死锁

【考点笔记】系统安全状态

安全状态，是指系统能按某种进程推进顺序（P1, P2, …, Pn），为每个进程 Pi 分配其所需资源，直至满足每个进程对资源的最大需求，使每个进程都可顺序地完成。此时称 P1, P2, …, Pn 为安全序列。如果系统无法找到一个安全序列，则称系统处于不安全状态。

并非所有的不安全状态都是死锁状态，但当系统进入不安全状态后，便可能进入死锁状态；反之，只要系统处于安全状态，系统便可以避免进入死锁状态。

死锁

【考点笔记】银行家算法的数据结构

数据结构	描述
可利用资源矢量 Available	含有 M 个元素的数组，其中的每一个元素代表一类可用的资源数目。Available[j]=K，则表示系统中现有 Rj 类资源 K 个
最大需求矩阵 Max	N×M 矩阵，定义了系统中 N 个进程中的每一个进程对 M 类资源的最大需求。Max[i, j]=K，则表示进程 i 需要 Rj 类资源的最大数目为 K
分配矩阵 Allocation	N×M 矩阵，定义了系统中每一类资源当前已分配给每一进程的资源数。Allocation[i, j]=K，则表示进程 i 当前已分得 Rj 类资源的数目为 K
需求矩阵 Need	N×M 矩阵，表示每个进程尚需的各类资源数。Need[i, j]=K，则表示进程 i 还需要 Rj 类资源的数目为 K

进程	最大需求	已分配	最多还需要
P0	(7, 5, 3)	(0, 1, 0)	(7, 4, 3)
P1	(3, 2, 2)	(2, 0, 0)	(1, 2, 2)
P2	(9, 0 ,2)	(3, 0, 2)	(6, 0, 0)
P3	(2, 2, 2)	(2, 1, 1)	(0, 1, 1)
P4	(4, 3, 3)	(0, 0, 2)	(4, 3, 1)

Max矩阵　Allocation矩阵　Need矩阵

向量 Available = (2, 2, 1)
表示剩余可用资源数

死锁

向量 Available = (2, 2, 1)
表示剩余可用资源数

向量 Request $_i$ = (2, 1, 1)
表示进程 Pi 请求的资源数

进程	最大需求	已分配	最多还需要
P0	(7, 5, 3)	(0, 1, 0)	(7, 4, 3)
P1	(3, 2, 2)	(2, 0, 0)	(1, 2, 2)
P2	(9, 0 ,2)	(3, 0, 2)	(6, 0, 0)
P3	(2, 2, 2)	(2, 1, 1)	(0, 1, 1)
P4	(4, 3, 3)	(0, 0, 2)	(4, 3, 1)

Max矩阵　Allocation矩阵　Need矩阵

数据结构：
长度为 m 的一维数组 Available 表示还有多少可用资源
n*m 矩阵 Max 表示各进程对资源的最大需求数
n*m 矩阵 Allocation 表示已经给各进程分配了多少资源
Max – Allocation = Need 矩阵表示各进程最多还需要多少资源
用长度为 m 的一位数组 Request 表示进程此次申请的各种资源数

银行家算法步骤：
①检查此次申请是否超过了之前声明的最大需求数（需确保Need[i] ≥ Request $_i$）
②检查此时系统剩余的可用资源是否还能满足这次请求（需确保 Available ≥ Request $_i$）
③试探着分配，更改各数据结构（Available -= Request $_i$ ； Allocation[i] += Request $_i$ ； Need = Max - Allocation）
④用安全性算法检查此次分配是否会导致系统进入不安全状态

安全性算法步骤：
检查当前的剩余可用资源是否能满足某个进程的最大需求，如果可以，就把该进程加入安全序列，并把该进程持有的资源全部回收。
不断重复上述过程，看最终是否能让所有进程都加入安全序列。

注意：系统处于不安全状态未必死锁，但死锁时一定处于不安全状态。系统处于安全状态一定不会死锁。