操作系统期中作业

一. 简答题：

1. 说明抽象资源和物理资源之间的区别，并举出两个例子。

答：区别：物理资源是有限的资源（我认为可以说是物理存在的资源），抽象资源是将物理资源转变成多个逻辑上的对应物，或者把多个物理资源转变成单个逻辑上的对应物的资源。

例子：

资源类别	物理资源	抽象资源
处理机	CPU	进程
存储器	主存	虚存

2. 操作系统有哪些架构，说明各自的特点。

答：有单体系统，层次式系统，微内核，客户机-服务器模式，虚拟机，外核等等。

各自的特点：

架构	**特点**
单体系统	全部操作系统在内核态以单一程序的方式运行，整个操作系统以过程集合的方式编写，链接成一个大型可执行二进制文件
层次式系统	层次式系统的上层软件都是在下一层软件的基础之上构建的
微内核	为了实现高可靠性，将操作系统划分成小的，良好定义的模块，只有其中一个模块——微内核运行在内核态，其余模块作为普通用户程序运行
客户机-服务器模式	将进程划分成两类：服务器（每个服务器提供某种服务），客户端（使用这些服务）
虚拟机	是裸机硬件的复制品，理论上可以运行一台裸机可以运行的任何类型的操作系统。
外核	运行在内核态，为虚拟机分配资源，并检查试图使用这些资源的目的

3. 什么是用户态，什么是内核态，这两种状态如何切换？

答：内核态：内核态是操作系统管理程序执行时所处的状态，能够执行包含特权指令在内的一切指令，能够访问系统内所有的存储空间。

用户态：用户态是用户程序执行时处理器所处的状态，不能执行特权指令，只能访问用户地址空间。

如何切换：

1. **从用户态切换到内核态的方法：**系统调用、异常和外部中断。

   切换方式一般为：保存上下文，然后跳转。不同的 ISA 提供的接口不太一样。

2.从内核态切换到用户态：一般是恢复上下文，然后用硬件指令切换特权级，跳转等等。

4. 简要描述什么是策略与机制的分离，举例说明。

答：机制是指实现某一功能的具体执行框架，策略是在机制的基础上借助某些算法来达到该功能。策略与机制的分离就是将一件事情的做什么（由机制决定），怎么做（由策略决定）相分离，达到一个更灵活的解构（我的个人观点）。

举例：调度机制与调度策略的分离：比如说数据库中假设内核使用优先级调度算法，但提供一条可供进程设置（并改变）优先级的系统调用。调度机制位于内核，调度策略由用户决定。

5. 什么是进程控制块（PCB），它有哪些主要内容？

答：进程控制块是一个被操作系统用来存储关于进程数据的数据结构，帮助程序成为一个能与其他进程并发执行的进程。

主要内容：寄存器，pc，正文段指针，数据段指针，堆栈段指针，进程状态，进程 ID 等等。

6. 简述进程的生命周期。

答：（1）首先是被创建进入就绪态（不同操作系统创建 API 不同）
（2）就绪 -> 执行：调度器调度，分配了处理机后
（3）执行 -> 就绪：调度器调度（由中断引起后）
（4）执行 -> 阻塞：因等待某种时间发生而无法继续进行（比如 I/O 或事件等待）
（5）阻塞 -> 就绪：I/O 或事件完成
（6）运行 -> 终止：

7. 什么是系统调用？简述系统调用流程。

答：系统调用是指运行在用户空间的程序向操作系统内核请求需要更高权限运行的服务，提供用户程序与操作系统之间的接口。

简述流程：从用户程序调用陷入指令跳转到系统调用陷入机构，首先保存 CPU 现场，取系统功能号找到入口地址表相应入口地址，通过地址跳转到系统调用处理子程序。子程序执行完后，结束处理，恢复现场，回到用户程序中（一般回到用户程序下一条指令）。

8. 进程间有哪些通信方式，各自有什么特点？

答：进程间通信主要包括管道、系统 IPC（包括消息队列、信号量、信号、共享内存等）

各自特点：

方式	特点
管道	由两个文件描述符引用，一个表示读端，一个表示写端。规定数据从管道的写端流入管道，从读端流出
共享内存	直接对内存进行操作来实现通信
信号量	使用一个整型变量来累计唤醒次数，一般用于同步

9. 在用户态实现线程的最大优点是什么？最大缺点是什么？

答：最大优点：用户级线程包可以在不支持线程的操作系统上实现。
最大缺点：难以实现阻塞系统调用

10. 内核采取一系列动作以便在两个进程之间进行上下文切换，请描述一下这些过程。

答：描述过程：
（1）挂起一个进程，将这个进程在 CPU 中的状态（上下文信息）存储于内存的 PCB 中
（2）调度器选择一个进程进行调度。在 PCB 中获取此进程上下文并将其在 CPU 寄存器中恢复。
（3）跳转到程序计数器指向位置，恢复该进程。

11. 最短作业优先算法在什么情况下是最优的？怎么证明？

答：最短作业优先算法在所有作业可以同时运行的情况下最优。
证明： 假设被调度进程集合有 n 个元素.对这 n 个进程的任意排列{P0,P1,P2,...,Pn-1},便是一种调度方案。设 T1 为任意进程调度方案产生的平均周转时间，T1=[X0+(X0+X1)+...+(X0+X1+...+Xn-1)]/n。Xi 是第 i+1 个执行的进程的运行时间。
对 X0,X1,...,Xn-1 递增排序,得 Xk0， X k1 …… ， X k n-1。设 T2 位 SJF 策略生成的调度方案的平均周转时间,则:
T2 = [Xk0 + (Xk0 + X k1)+……+(Xk0 + X k1 + ……+ X k n-1]/n
对比构成 T1 和 T2 的每个累加项,后者永远不大于前者，所以 T2 是 T1 中最小值.故,能够产生 T2 的调度方案必为理论最优，而 T2 调度方案是最短作业优先算法生成的,得证.

12. 简述实时系统中调度问题的单调速率算法。
答：简述算法:
在进入系统时，每个周期性任务会分配一个优先级，此优先级与其周期成反比。当较低优先级进程正在运行且较高优先级进程可以运行时，较高优先级进程将会抢占低优先级。

13. 怎么解决多级反馈队列调度算法中的饿死问题？

答：加入一个规则：经过一段时间，就将系统中所有工作重新加入最高优先级队列。

14. 什么是虚拟地址（空间）和物理地址（空间）？

答： 虚拟地址（空间）：在一个带虚拟内存的系统中，操作系统从一个有 N=pow(2,n) 个地址的地址空间中生成虚拟地址并使之对于进程可以访问，这个地址空间称为虚拟地址空间。现代系统通常支持 32 位或者 64 位虚拟地址空间。

物理地址（空间）：是指主存的某个可以让数据总线访问的特定存储单元的内存地址（空间）。

15. 请简述分页式存储管理的基本实现原理。

答：虚拟地址空间被划分成大小相等的页面，硬盘与主存之间按页面为单位交换信息。进程中每个虚拟页面都有一个对应的页表项，当装入位为 1 时，表示此页被调入主存，装入位为 0，表示此页没有被调入主存。当用虚拟地址访存时，首先根据页表基址寄存器中的内容找到对应页表起始位置，然后以虚拟地址高位字段的虚页号作为索引，找到对应页表项。如果装入位为 1，则根据页表项中的物理页号和虚拟地址低字段的页内地址获得物理地址。如果装入位为 0，则跳转到缺页异常程序从磁盘中载入页面修改装入位与物理页号等等，再根据页表项中的物理页号和虚拟地址低字段的页内地址获得物理地址。可以通过 TLB（快表）提高访问页表速度。

16. 假设物理内存足够大，虚拟内存是否还有存在的必要？如果不使用虚拟内存抽象，只使用物理内存寻址，会对操作系统的内存管理带来哪些改变？

答：有存在必要。

带来的改变：1. 更加复杂的存储地址分配：每个进程有不一样的物理地址，在从磁盘加载后要分配不同的物理地址，为了使程序正常运行，可能每个程序都要完全地保留一块地址空间的所有内容，如何分配会成为一个很复杂的问题。

2. 不同内存区域权限的管理。如何保证内存分配后，一个进程里的程序不会篡改另一个进程的内存会是个好问题。

3. 进程不再需要地址翻译（不用页表项等等），可以用物理地址直接访问。

4. 加载的变化。在虚拟内存中，很多是在要访问时才从磁盘加载到内存，并不保留位置在内存，而实地址必须保留位置。

17. 什么是优先级反转问题？怎么解决？

答：优先级反转问题：由于多进程共享资源，具有最高优先权的进程被低优先级进程阻塞，反而使具有中优先级的进程先于高优先级的进程执行，导致系统的崩溃。

解决方法：优先级继承：将低优先级任务的优先级提升到等待它所占有的资源的最高优先级任务的优先级。当高优先级任务由于等待资源而被阻塞时，此时资源的拥有者的优先级将会自动被提升。

18. 简述下管程中的 Brinch Hansen、Hoare、Mesa 语义区别。

答：管程中的 Brinch Hansen、Hoare、Mesa 语义区别：

管程中，在 signal 唤醒另一个进程使之成为 RUNNABLE 后：

Brinch Hansen：执行 signal 的进程必须立即退出管程，即 signal 语句一定作为管程过程最后一个语句。

Hoare：让新唤醒的进程运行，挂起另一个。

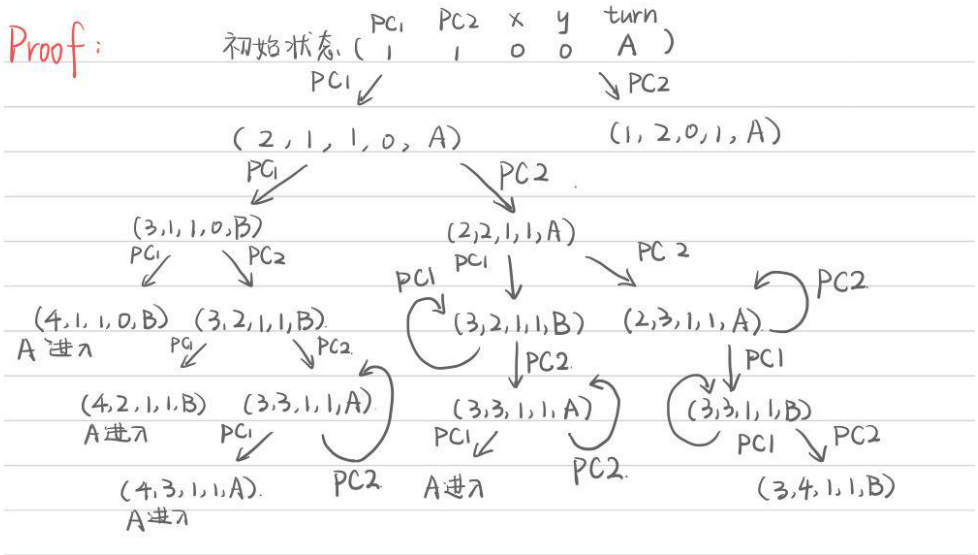
Mesa：让发信号者继续运行，并且只有在发信号者退出管程后，才允许等待的进程开始运行。

19. 简要说明 Peterson 算法的原理，Peterson 算法符合 safety 和 liveness 性质吗？说明理由。

答：原理：Peterson 实现的是两个进程之间访问临界区的互斥。在进入临界区前，每个进程首先要设置标识为 1，然后在一个公共变量设置对方进程的进程号。将循环到进程号不是自己，或者对方标识不为 1 才能进入。访问完成后将自己标识设为 0.如果没有两个进程同时访问时，没在访问的进程的标识为 0，故可以顺利进入。如果两个进程同时访问，先设置公共变量的将被后设置公共变量覆盖，这个时候先设置的那个进程满足条件将先进入。后设置的会陷入循环，直到先的出来，故保证了互斥。

符合。理由：画状态机如下：

其中 pc1,pc2 分别是两个进程的 pc ，x 表示 pc1 中想要访问时设置的标识，y 表示 pc2 中想要访问的标识，turn 标识现在在哪个进程中执行。



以上图中画出了所有可能的状态，发现确实没有任何坏的情况发生，证明了 safety，发现确实实现了互斥，证明了 liveness。

20. 简要说明信号量、互斥锁、条件变量的区别和联系。

答：区别：

（1）互斥锁是主要用来上锁（互斥），条件变量主要用来等待（同步），信号灯即可用于上锁，也可用于等待。

（2）互斥锁，条件变量一般是二值状态，信号量可以有很多数值。

（3）由于信号量有一个与之关联的状态（它的计数值），信号量挂出操作总是被记住。然而当向一个条件变量发送信号时，如果没有线程等待在该条件变量上，那么该信号将丢失。

联系：

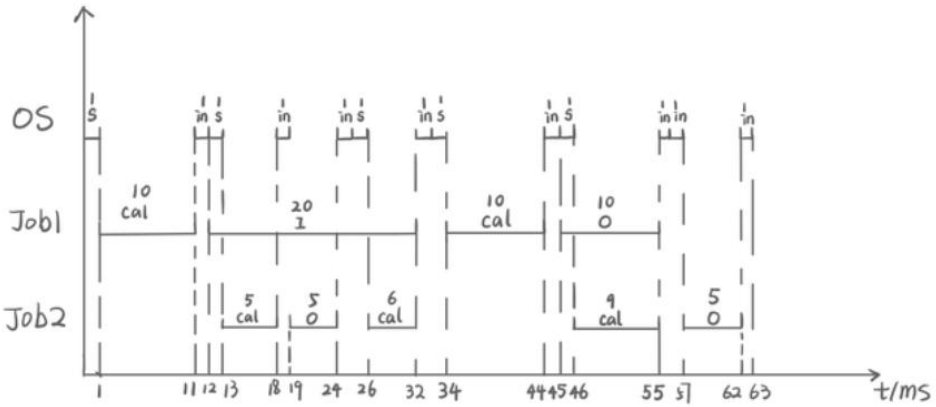
（1）条件变量一般会和互斥锁一起使用。

（2）都会在一定情况下造成阻塞（而不是像自旋锁一样忙等待）。信号量在数字降为 0 还要往下时会被阻塞，条件变量通过 wait 阻塞，互斥锁在加锁时被阻塞。

二. 应用题：

1. 在一个具有一个中央处理器（CPU）、一台输入设备（I）和一台输出设备（O）的计算机系统上同时运行两个计算任务（分别为 Job1 和 Job2），其执行内容如下： Job1：计算(10ms)，输入(20ms)，计算(10ms)，输出(10ms)； Job2: 计算(5ms)，输出(5ms)，计算(15ms)，输出(5ms)。 设中央处理器与输入/输出设备可并行工作，Job1 的优先级高于 Job2，高优先级计算任务可抢占低优先级计算任务，每次中断处理花费 1ms，每次调度程序执行花费 1ms，试画出各计算任务和操作系统内核（简称 OS）之间的时间关系图。

答: 用 s 表示调度(schedule), 用 in 表示中断(interrupt), 用 cal 表示计算(calculate), i 表示使用输入设备中，o 表示使用输出设备中，可作图如下:



注：CPU 调度系统空闲进程时处于 OS 中并未在图中画出，应该在 57-58 秒。

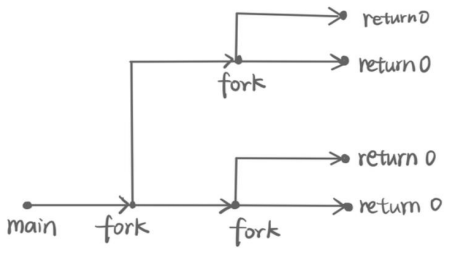
2. Windows 的 CreateProcess 和 Linux 的 fork 都具有创建进程的功能，但语义稍有不同，请分析以下三种情况，说明 CreateProcess 和 fork 哪个更加合适，并解释原因。
   a) Shell 接收用户输入的命令 ls，并创建一个新的进程执行该命令。
   b) Web 服务器收到请求，并创建一个新的进程处理该请求。
   c) 父进程创建一个子进程，并利用管道通信。

   答：CreateProcess 产生一个新进程并且在这个进程上加载一个新程序，fork 会产生一个新进程，并且父进程和子进程会并发跑 fork 之后的内容。(从某种意义上，CreateProcess 是 fork 和 execve 的结合)
   对于 a) : CreateProcess 更适合。原因：接下来将创建一个新进程执行 ls 命令，相当于重新开始跑 ls 程序，CreateProcess 适用但是 fork 并不能重置程序。
   对于 b)：CreateProcess 更适合。原因：接下来将创建一个新进程，执行处理请求的程序，CreateProcess 可以将此程序跑，但是 fork 只能继续在原程序后跑。
   对于 c) : fork 更合适。原因：子进程并没有重新加载什么程序，故用 fork 合适。

3. 下面代码总共创建多少进程（包含该父进程）。

```
#include <stdio.h>
#include <unistd.h>
int main()
{
/* fork a child process */
fork();
/* fork another child process */
fork();
return 0;
}
```

   答：画进程图如下：



   可得一共创建了四个进程。

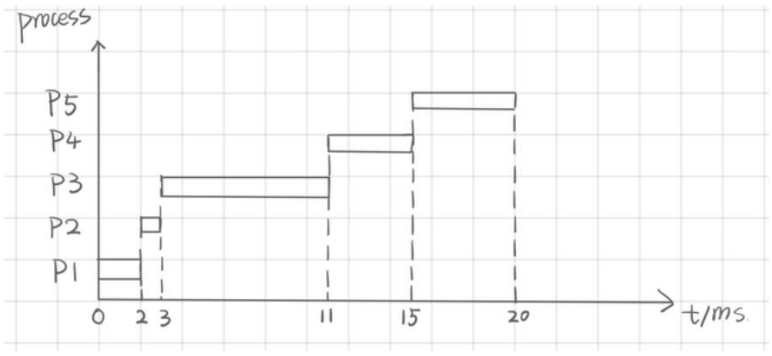
4. 假设有如下一组进程，他们的 CPU 执行时间以毫秒来计算：

进程	执行时间	优先级
P1	2	2
P2	1	1
P3	8	4
P4	4	2
P5	5	3

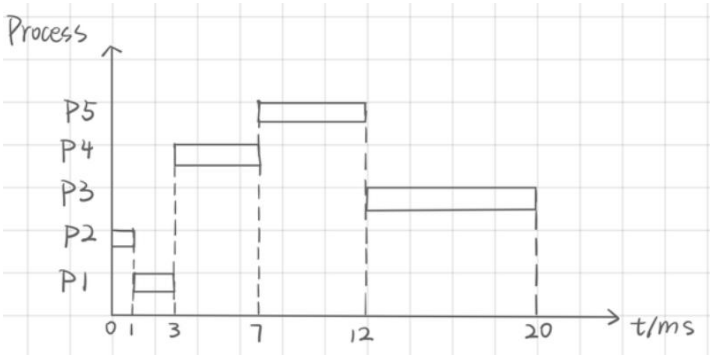
假设进程按照 P1, P2, P3, P4, P5 顺序在 0 时刻到达。

a） 画出甘特图，分别演示采用调度算法 FCFS、SJF、非抢占式优先级和 RR（时间片为 2）的进程执行。

答：1. FCFS

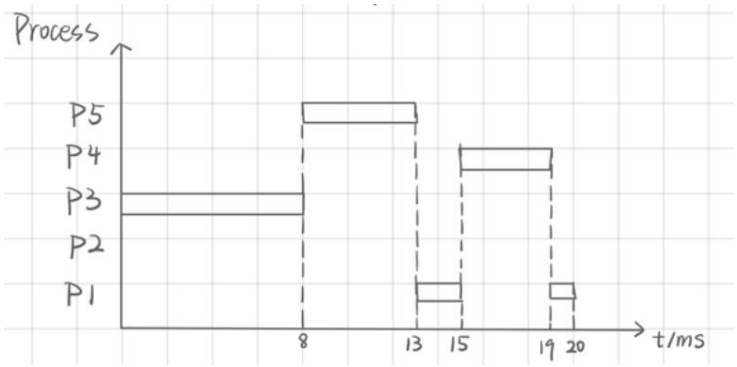


2. SJF

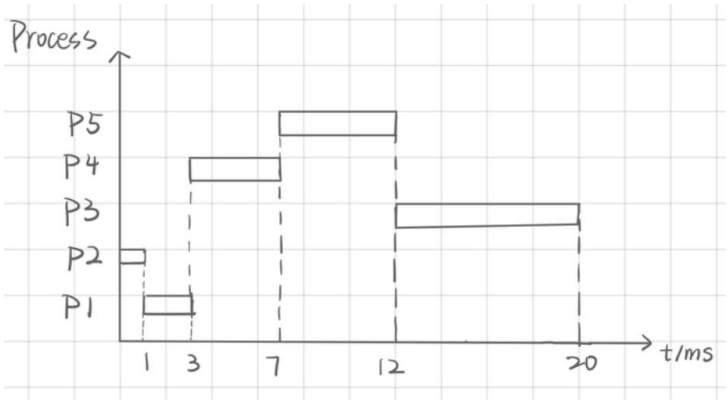


3. 非抢占式优先级

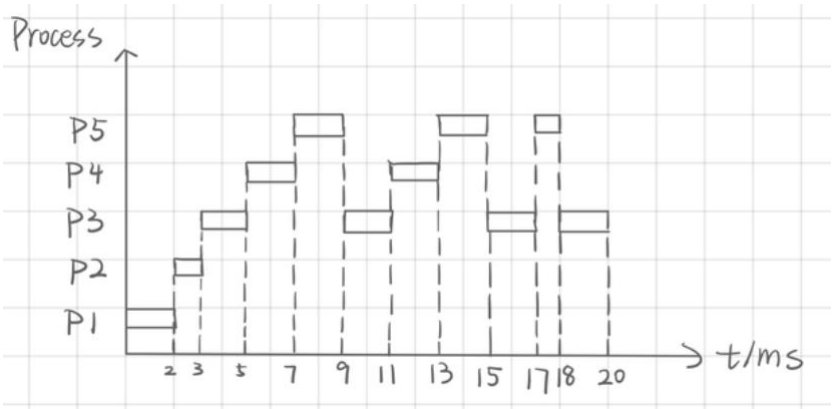
（1）如果数字大代表优先级高



（2）如果数字小代表优先级高



4. RR



b）对于问题 a，给出每种算法的周转时间。

答：（注：这里非抢占优先级按数字小优先级高算）

进程周转时间 / 算法	P1	P2	P3	P4	P5	周转总时间
FCFS	2	3	11	15	20	51
SJF	3	1	20	7	12	43
非抢占式优先级	3	1	20	7	12	43
RR	2	3	20	13	18	56

c） 哪一种算法的平均等待时间最小？

答：注：这里非抢占优先级按数字小优先级高算）

进程等待时间 / 算法	P1	P2	P3	P4	P5	平均等待时间
FCFS	0	2	3	11	15	31/5=6.2
SJF	1	0	12	3	7	23/5=4.6
非抢占式优先级	1	0	12	3	7	23/5=4.6
RR	0	2	12	9	13	36/5=7.2

综上，SJF 和非抢占式优先级平均等待时间最小，为 4.6

5. 现在运行 20 个 I/O 密集型任务和 1 个 CPU 密集型任务。假设 I/O 密集型任务每 1ms 的 CPU 计算就进行一次 I/O 操作，并且每个 I/O 操作需要 10ms 来完成。另假设上下文切换开销为 0.1ms，所有的进程都是长时间运行的任务，请讨论在下列条件下轮转调度程序的 CPU 利用率。
   a）时间片为 1ms
   b）时间片为 5ms

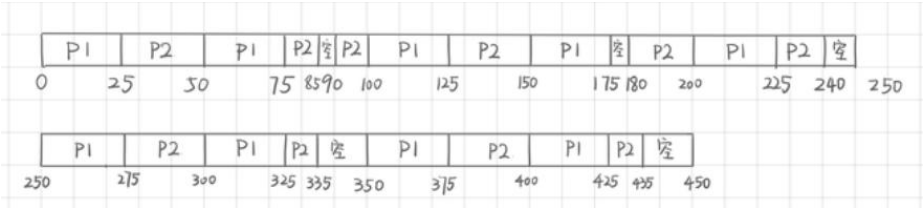
答：由于 I/O 操作需要 10ms，对于 ab 题而言所有任务都在再次轮到前变为就绪态。

对于 a) ：时间片为 1ms 时，无论哪个进程调度，都会花费上下文切换时间 0.1ms，所以 CPU 利用率是 1/1.1*100%=91%

对于 b)：时间片为 5ms 时，I/O 密集型任务用不完时间片，每工作 1ms 将花费 0.1ms 进行上下文切换。对于 CPU 密集型任务，将用完时间片，即每工作 5ms 花费 0.1ms 进行上下文切换。所以每个所有任务轮一边的循环要花费 20*1.1+5.1=27.1(ms)，其中 CPU 实际工作时间为 20+5=25(ms)，故 CPU 利用率为 25/27.1*100% = 92.3%

6. 考虑两个实时任务 P1 和 P2，期中 P1 的周期 50ms，而 P2 是 90ms，P1 的处理时间是 25ms，而 P2 是 35ms，P1 和 P2 的截止时间都是在下个迭代来临之前。讨论使用单调速率算法的情况，如果成功，解释一下为什么能成功，如果失败，解释一下为什么失败。

答：由于 25%50+35%90 <1 ,所以似乎可以调度任务以满足截至期限。运用单调速率算法：由于 P1 周期短，所有分配更高的优先级。P1 和 P2 的公共周期为 450ms，作 450ms 的 Gantt 图如下：



发现没有超出截至期限的任务，故成功。

7. 假设一个支持分页的计算机系统有 36 位的虚拟地址，页面大小为 8KB，每个页表项占用 4 Bytes

a) 虚拟地址空间中共有多少个页面？

b) 该系统可访问的最大物理地址空间为多少？

c) 如果进程的平均大小为 8GB，此时应选择一级、二级还是三级页表？为什么？在你选择的方案下，页表的平均大小是多少？

答：a) 8KB 为 2 的 13 次方字节，故虚拟地址空间中有 $2^{36-(3+10)}$，即 2^{23} 个页面。

b) 假设页表项中除了物理页号之外的其他位（比如装入位，修改位等等）占 x bit，则系统可以访问的最大物理地址为物理页号和页内地址全为 1 的情况，即 $2^{(3+10)+32-x+1}$ -1，即 2^{46-x} - 1。如果除物理页号以外的其他位占 0bit，则可访问的最大地址空间为 2^{46} - 1。（但是我觉得这种情况不可能发生，装入位一定要有）

c) 我选择二级页表。由于一个进程为 8GB，故至少要用到 2^{20} 个页表项。首先比较一级页表和二级页表。不妨假设 23（36 减去 13，即虚拟地址位数减去业内偏移位数）bit 的虚拟页号中，如果是二级页表，高 3 位为 PT1 域，低 20 位为 PT2 域。由于一个进程的大小为 8GB，对于二级页表，则只有 1 一级页表项用到，此页表项将索引到有 2^{20} 个页表项的二级页表。可以计算各级页表需要的内存:

一级页表：2^{25} 字节，即 32MB

二级页表：$2^{3+2}+2^{20+2}$ 字节，约为 4MB

故可以看到，虽然复杂度增加，但二级页表的大小优化较为明显，

比较二级页表和三级页表，由于 8GB 得使用三级页表也至少要 2^{20} 个页表项，故大小和二级页表差不多但是复杂度大于二级页表，故我选二级页表。

页表的平均大小：可得选择我的方案时，一共有 2 个页表（一个一级一个二级），故页表的平均大小为：$2^{5}+2^{21}$ 字节，约为 2MB

8. 对于一个有 4 个页框的机器，其每一页对应的载入时间、最近一次访问时间、以及每个页面的 Reference 和 Modify 位如下表所示。此时，FIFO、第二次机会、NRU 和 LRU 算法分别会选择哪个页面进行置换？

页面	载入时间	最近一次访问时间	R	M
0	126	280	1	0
1	230	265	0	1
2	140	270	0	0
3	110	285	1	1

答：

算法	选择置换的页面
FIFO	3
第二次机会	2
NRU	2
LRU	1

9. 将课上所实现的强制轮转法扩展到可以处理多个进程。

答：对于 n 个进程 Pi（i=0,1,...,n-1），可实现强制轮转法如下：
Process i 的程序主体为：

```
//process Pi
while(TRUE) {

    while (turn != i);
    critical_section();
    turn = (turn + 1) % n;
    noncritical_section();

}
```

10. 将课上所述利用条件变量解决生产者-消费者的算法扩展到多个生产者和多个消费者的情形下。

答：

```
int BUFFERSIZE = 100;
int count = 0 ;
condition full,empty;
int mutex = 0;
```

对于 producer:

```
void producer(void) {
    int item;
    while(TRUE) {
        produce_item(&item);
        mutex_lock(mutex);
        while(count == BUFFER_SIZE) //将课件上if改成while
            wait(full, mutex);
        //上面的while确保了跳出循环时,count 一定不为 BUFFER_SIZE
        enter_item(item); // put item in buffer
        count++;
        //当刚刚是empty现在有1个时，向所有consumer发信号
        if(count == 1) broadcast(empty);
        mutex_unlock(mutex);

    }
}
```

对于 consumer:

```
void consumer(void) {
    int item;
    while(TRUE) {
        mutex_lock(mutex);
        while(count == 0) //将课件上if改成while
            wait(empty,mutex);
        //上面的while确保了跳出循环时,count 一定不为0
        remove_item(item); //remove it from buffer
        count--;
        //当刚刚是full现在是full-1个时，向所有producer发信号
        if(count == BUFFER_SIZE -1) broadcast(full);
        mutex_unlock(mutex);
        print_consume_item(&item); // print item
    }
}
```