参考答案

一、名词术语解释（每题 3 分，共 24 分）

1. **系统调用**
   操作系统为用户程序提供的一组接口，使其能够请求内核提供诸如进程管理、文件操作、设备控制等服务。系统调用扩充机器功能、增强系统能力、方便用户使用。

2. **线程**
   是操作系统能够进行运算调度的最小单位。它被包含在进程之中，是进程中的实际运作单位。它是实现并发和提高程序响应性的重要机制。

3. **临界区**
   指在并发环境下，访问共享资源（如变量、设备）的程序段。为了防止数据竞争和不一致问题，任一时刻只能允许一个线程或进程进入临界区。

4. **分时系统**
   采用时间片轮转的方式为多个用户服务的一种操作系统，使得多个用户可以共享同一台计算机，每个用户在很短的时间片内获得 CPU 的使用权，从而使每个用户都感觉自己独占系统。

5. **死锁**
   当两个或多个进程互相占有对方需要的资源，并等待对方释放时，就会形成一种循环等待状态，导致各进程都无法继续执行，这种现象称为死锁。（各进程互相等待对方手里的资源，导致各进程都阻塞，无法向前推进）

6. **虚拟存储器**
   基于局部性原理，在程序装入时，可以将程序中很快会用到的部分装入内存，暂时用不到的部分留在外存，就可以让程序开始执行。在程序执行过程中，当所访问的信息不在内存时，由操作系统负责将所需信息从外存调入内存，然后继续执行程序。若内存空间不够，由操作系统负责将内存中暂时用不到的信息换出到外存。在操作系统的管理下，在用户看来似乎有一个比实际内存大得多的内存，这就是虚拟内存。

7. **进程同步**
   同步亦称直接制约关系，它是指为完成某种任务而建立的两个或多个进程，这些进程因为需要在某些位置上协调它们的工作次序而产生的制约关系。进程间的直接制约关系就是源于它们之间的相互合作。

8. **SPOOLing**
   一种假脱机技术，用于解决计算机主机与慢速字符设备之间信息交换的问题。

这项技术通过在输入和输出之间增加“输入井”和“输出井”的排队转储环节，实现了外围设备的同时联机操作。SPOOLing 技术的核心在于利用软件手段模拟脱机技术，使得多道程序技术得到支持，从而提高 I/O 速度，将独占设备改造为共享设备，并实现虚拟设备功能。

二、填空题（每题 2 分，共 14 分）

1. **静态重定位**是在**编译或装入内存时**完成，**动态重定位**则依靠**硬件（MMU）和程序在运行时**来完成。
2. 进入就绪状态的进程来自**运行状态**，**创建状态**，**阻塞状态**。
3. 死锁的四个必要条件中，“保持和等待”指的是**进程在占有部分资源的同时，又请求其他进程占有的资源，从而导致相互等待**。
4. 防止死锁的策略分为静态和动态两种，而银行家算法属于**动态死锁避免策略**，在每次资源申请时判断系统是否处于安全状态。
5. 若 M 个进程共享一互斥段，每次最多允许 N 个进程进入（M > N），则信号量的取值范围为**[N-M, N]（小于零说明有进程在等待）**。
6. 判断一个计算机系统是否具备虚拟存储系统，关键在于是否支持**页式或段式存储管理（具有请求调入和置换功能）及地址转换机制（相关硬件支持）**。
7. 操作系统向用户提供的接口主要有**用户接口和程序接口**。

三、判断题（每题 1 分，共 10 分）

1. **(b) 错**
   增加页面大小可能降低页表项数量，但缺页中断次数不仅与页面大小有关，还与程序局部性有关，不能简单地减少一半。
2. **(b) 错**
   虚拟存储器的大小受地址位数限制，而不是简单地将物理内存和硬盘容量相加。
3. **(b) 错**
   C 编译器代码一般是可重入的，多用户同时编译时，共享同一份代码段而不必各自复制。
4. **(b) 错**
   进程的运行状态还由操作系统调度算法决定，而非进程自行选择。
5. **(a) 对**

在请求分页存储管理中，系统可以在部分页面装入后开始执行程序，遇缺页中断时再动态调入所需页面。

6. **(a) 对**
   管程保证互斥的实现是在进程进入管程时由系统自动控制的。

7. **(a) 对**
   字节设备，一个字节一个字节读写的设备，不能随机读取设备内存中的某一数据。字符设备按照字符流的方式被有序访问，面向流的设备。

8. **(b) 错**
   进程挂起可能是等待 I/O 或其他事件，而死锁是资源循环等待，二者不能等同。

9. **(b) 错**
   优点是访问速度快，文件长度可以动态变化。缺点是存储开销大，因为每个文件有一个索引表，而索引表亦由物理块存储，故需要额外的外存空间。。

10. **(b) 错**
    多道程序系统的道数并非越多越好；过多会导致频繁的上下文切换和系统调度负担增大，从而影响整体性能。

四、简答题（每题 5 分，共 10 分）

1.题目：
系统有 8 个相同的资源，7 个进程共享，每个进程最多需 2 个资源，证明该系统不会发生死锁。

死锁产生的必要条件包括:
① **互斥条件**：资源一次只能由一个进程使用;
② **占有并等待**：进程在占有部分资源时等待其他资源;
③ **非抢占条件**：已分配的资源不能被抢占;
④ **循环等待**：进程之间形成资源等待环路。

在本题中，假设每个进程最多需要 2 个资源。即使 7 个进程都已占有 1 个资源，总共占用了 7 个资源，系统仍剩下 1 个空闲资源。

此时，总有至少一个进程能够获得其所需的第二个资源并顺利完成运行，完成后释放占有的资源，进而使其他进程得到满足。

因此，系统总能避免进入循环等待状态，从而不会产生死锁。

2. 题目：
如何判断一个实时系统是可调度的?

响应时间分析：需确保所有任务的响应时间均小于其截止时间（Deadline）。

任务模型和优先级：根据任务周期、执行时间、截止时间等参数，比如采用 Earliest Deadline First (EDF) 等调度算法，通过理论公式或仿真验证所有任务是否能在规定时间内完成。

系统负载与利用率：实时系统需满足利用率要求，通常有最大利用率阈值，超过此阈值则系统不可调度。

五、解答题（共 7 分）：

题目：
利用图示的硬件机构进行分页存储管理，主存（M1）中有 2 个页面框，辅存（M2）存储全部页面。页面访问序列为：

a, b, a, c, a, b, d, b, a, c, d

分别采用 FIFO 和 LRU 算法计算缺页中断率，并简述两者差异原因。

（1）FIFO 算法

FIFO 算法始终替换“最早进入内存”的页面。下表展示了每一步的情况：

步骤	访问页	帧状态（顺序表示 FIFO 队列，左侧为最早进入）	是否缺页	说明	累计缺页次数
1	a	[a, –]	是	将 a 插入空帧	1
2	b	[a, b]	是	将 b 插入空帧	2
3	a	[a, b]	否	a 已在帧中	2
4	c	[c, b]	是	替换队列中最早的 a（FIFO：a→c）	3
5	a	[c, a]	是	替换队列中最早的 b（FIFO：b→a）	4
6	b	[b, a]	是	替换队列中最早的 c（FIFO：c→b）	5
7	d	[b, d]	是	替换队列中最早的 a（FIFO：a→d）	6

步骤	访问页	帧状态（顺序表示 FIFO 队列，左侧为最早进入）	是否缺页	说明	累计缺页次数
8	b	[b, d]	否	b 已在帧中	6
9	a	[d, a]	是	替换队列中最早的 b（FIFO：b→a）	7
10	c	[a, c]	是	替换队列中最早的 d（FIFO：d→c）	8
11	d	[c, d]	是	替换队列中最早的 a（FIFO：a→d）	9

FIFO 结果： 共 9 次缺页，缺页率 9/11 ≈ 81.8%

（2）LRU 算法

LRU 算法每次替换“最近最久未使用”的页面。下表展示了各步操作及最近使用情况：

步骤	访问页	帧状态（附上最近使用时间）	是否缺页	说明	累计缺页次数
1	a	[a(1), –]	是	插入 a	1
2	b	[a(1), b(2)]	是	插入 b	2
3	a	[a(3), b(2)]	否	a 命中，更新 a 的时间为 3	2
4	c	[a(3), c(4)]	是	在 [a(3), b(2)]中，b 最久未用，替换 b → c	3
5	a	[a(5), c(4)]	否	a 命中，更新 a 的时间为 5	3
6	b	[a(5), b(6)]	是	在 [a(5), c(4)]中，c 最久未用，替换 c → b	4
7	d	[d(7), b(6)]	是	在 [a(5), b(6)]中，a 最久未用，替换 a → d	5
8	b	[d(7), b(8)]	否	b 命中，更新 b 的时间为 8	5

步骤	访问页	帧状态（附上最近使用时间）	是否缺页	说明	累计缺页次数
9	a	[a(9), b(8)]	是	在 [d(7), b(8)]中，d 最久未用，替换 d → a	6
10	c	[a(9), c(10)]	是	在 [a(9), b(8)]中，b 最久未用，替换 b → c	7
11	d	[d(11), c(10)]	是	在 [a(9), c(10)]中，a 最久未用，替换 a → d	8

LRU 结果： 共 8 次缺页，缺页率 8/11 ≈ 72.7%

(3)差异说明

FIFO 只考虑页面进入内存的顺序，未利用页面使用频率的信息，因此在该序列中出现 9 次缺页。

LRU 根据页面最近使用情况进行替换，通常更能利用程序的局部性，因而缺页次数稍少，为 8 次。

六、计算题（磁盘调度算法，共若干分）

题目描述：

磁盘有 400 道（0 ~ 399）；

当前磁头位置：202（刚完成 164 道请求后）；

请求序列（磁道号）：136, 198, 142, 223, 149, 198, 156, 245, 176；

（1）SSTF（最短寻道时间优先）算法

模拟步骤：

1. **初始位置 202**
   距离各请求：
   - 136: 66
   - 198: 4
   - 142: 60

- 223: 21
- 149: 53
- 198: 4
- 156: 46
- 245: 43
- 176: 26
  → 最接近的是 198（4 道）。

2. **更新到 198**
   剩余请求：136, 142, 223, 149, 198, 156, 245, 176
   距离：
   - 136: 62
   - 142: 56
   - 223: 25
   - 149: 49
   - 198: 0
   - 156: 42
   - 245: 47
   - 176: 22
     → 下一步选择 198（0 道移动）。

3. **保持在 198**
   剩余：136, 142, 223, 149, 156, 245, 176
   距离：
   - 136: 62
   - 142: 56
   - 223: 25
   - 149: 49
   - 156: 42
   - 245: 47

- 176: 22
  → 选择 176（22 道）。

4. **更新到 176**
   剩余：136, 142, 223, 149, 156, 245
   距离：
   - 136: 40
   - 142: 34
   - 223: 47
   - 149: 27
   - 156: 20
   - 245: 69
     → 选择 156（20 道）。
5. **更新到 156**
   剩余：136, 142, 223, 149, 245
   距离：
   - 136: 20
   - 142: 14
   - 223: 67
   - 149: 7
   - 245: 89
     → 选择 149（7 道）。
6. **更新到 149**
   剩余：136, 142, 223, 245
   距离：
   - 136: 13
   - 142: 7
   - 223: 74
   - 245: 96
     → 选择 142（7 道）。
7. **更新到 142**

剩余：136, 223, 245
距离：

- 136: 6
- 223: 81
- 245: 103
  → 选择 136（6 道）。

8. **更新到 136**
   剩余：223, 245
   距离：
   - 223: 87
   - 245: 109
     → 选择 223（87 道）。
9. **更新到 223**
   剩余：245
   距离：
   - 245: 22
     → 选择 245（22 道）。

总移动量计算：
4 + 0 + 22 + 20 + 7 + 7 + 6 + 87 + 22 = **175 道**

SSTF 调度序列： 198 → 198 → 176 → 156 → 149 → 142 → 136 → 223 → 245

（2）SCAN 算法（假定沿磁头移动方向不再有访问请求时，磁头沿相反方向移动）

假设当前磁头正向（由较小号向较大号方向）移动

- 分组：
  - 向上的请求：223, 245
  - 向下的请求：136, 198, 142, 149, 156, 176, 198

调度步骤：

1. 从 202 开始，向上依次服务 223 和 245。
   - 202 → 223：21 道
   - 223 → 245：22 道

2. 到达 245 后，磁头方向反转，服务所有小于 202 的请求。按照从大到小顺序依次为：198, 198, 176, 156, 149, 142, 136。
    - 245 → 198：47 道
    - 198 → 198：0 道
    - 198 → 176：22 道
    - 176 → 156：20 道
    - 156 → 149：7 道
    - 149 → 142：7 道
    - 142 → 136：6 道

总移动量计算：
21 + 22 + 47 + 0 + 22 + 20 + 7 + 7 + 6 = **152 道**

SCAN 调度序列： 上行：223, 245；反转后下行：198, 198, 176, 156, 149, 142, 136

七、P/V 操作题（同步与互斥问题）

题目背景：
系统中有三个进程：GET、PRO 和 PUT；两个缓冲区：BUF1（容量 11，初始有 2 个信息）和 BUF2（容量 5）。

GET 负责将输入信息送入 BUF1；

PRO 负责从 BUF1 中取数据，处理后送到 BUF2；

PUT 负责从 BUF2 中取数据并输出。

解答（伪 C 语言描述）：

定义信号量和互斥锁：

```
// 对于 BUF1
semaphore empty1 = 11 - 2;    // 空槽数（初始 BUF1 已有 2 个信息）
semaphore full1 = 2;          // 已填充信息数
semaphore mutex1 = 1;         // 互斥访问 BUF1
// 对于 BUF2
semaphore empty2 = 5;         // BUF2 初始为空
```

```
semaphore full2 = 0;              // 已填充信息数
semaphore mutex2 = 1;             // 互斥访问 BUF2
```

GET 进程：

```
void GET() {
    while (true) {
        // 获取新的输入信息（input_data）
        Data input_data = getInput();
        P(empty1);          // 等待 BUF1 有空位
        P(mutex1);          // 进入临界区
        insertToBUF1(input_data);  // 将信息放入 BUF1
        V(mutex1);          // 离开临界区
        V(full1);         // 增加 BUF1 中信息计数
    }
}
```

PRO 进程：

```
void PRO() {
    while (true) {
        // 从 BUF1 取数据进行处理
        P(full1);         // 等待 BUF1 中有数据
        P(mutex1);        // 互斥访问 BUF1
        Data d = removeFromBUF1();
        V(mutex1);
        V(empty1);
        Data processed = processData(d);  // 对数据进行处理
        // 将处理后的数据放入 BUF2
        P(empty2);        // 等待 BUF2 有空位
        P(mutex2);        // 互斥访问 BUF2
```

```
            insertToBUF2(processed);
            V(mutex2);
            V(full2);
        }
    }
```

PUT 进程：

```
void PUT() {
    while (true) {
        // 从 BUF2 取数据输出
        P(full2);          // 等待 BUF2 中有数据
        P(mutex2);         // 互斥访问 BUF2
        Data output_data = removeFromBUF2();
        V(mutex2);
        V(empty2);
        output(output_data);     // 输出数据
    }
}
```

说明：

- GET 进程通过 P(empty1) 确保不会向已满的 BUF1 中写入，同时使用 mutex1 保证对 BUF1 的互斥访问。
- PRO 进程既作为 GET 的消费者（从 BUF1 取数据）又作为 PUT 的生产者（将处理结果放入 BUF2），因此分别对 BUF1 和 BUF2 进行同步和互斥控制。
- PUT 进程从 BUF2 取数据时同样保证互斥访问。

这种设计确保了数据在各缓冲区之间按顺序传递，且避免了竞态条件，满足同步与互斥要求。
