

## 操作系统

注意事项：

1. 答卷前，考生务必将自己的姓名和准考证号填写在答题卡上。
2. 回答选择题时，选出每小题答案后，用铅笔把答题卡对应题目的答案标号涂黑。如需改动，用橡皮擦干净后，再选涂其它答案标号。回答非选择题时，将答案写在答题卡上。写在本试卷上无效。
3. 考试结束后，将本试卷和答题卡一并交回。请认真核对监考员在答上所粘贴的条形码上的姓名、准考证号与您本人是否相符。

### 一、选择题 (共 15 题, 每题 1 分, 共 15 分)

1. 操作系统最主要的任务是( )  
A. 管理资源                      B. 并发                      C. 异步                      D. 虚拟
2. ( )是操作系统形成的标志。  
A. 单道批处理系统              B. 多道批处理系统              C. 分时系统              D. 实时系统
3. 进程从运行状态进入就绪状态的原因可能是( )。  
A. 被选中占有处理机      B. 等待某一事件      C. 等待的事件已发生      D. 时间片用完
4. 在信号量机制中，哪种信号量的 wait 操作，不满足“让权等待”。( )  
A. 整型信号              B. 记录型信号量              C. And 型信号量              D. 一般信号量集
5. 操作系统中的存储管理是指对( )的管理。  
A. 主存                      B. 辅存                      C. Cache                      D. 以上都不是
6. 在请求分页管理方式中，页表中的状态位(存在位)用来指示对应页( )。  
A. 是否被修改过                      B. 是否允许动态增长  
C. 是否已调入内存                      D. 是否已置换
7. 可变式分区分配方案中，某一作业完成后，系统收回其主存空间，并与相邻空闲区合并，为此需修改空闲区表，造成空闲区数减 1 的情况是( )。  
A. 无上邻空闲区，也无下邻空闲区  
B. 有上邻空闲区，但无下邻空闲区  
C. 有下邻空闲区，但无上邻空闲区  
D. 有上邻空闲区，也有下邻空闲区
8. 典型的分时系统采用的进程调度算法是( )  
A. 先来先服务              B. 短进程优先              C. 高优先权优先              D. 时间片轮转

9. 通道是一种( )。
- A. I/O 专用处理器                      B. 数据通道                      C. 软件工具                      D. I/O 端口
10. 基本分段存储管理方式中, 逻辑地址的地址格式是( )地址。
- A. 线性                      B. 一维                      C. 二维                      D. 三维
11. 动态重定位是在作业( )中进行的。
- A. 编译过程                      B. 链接过程                      C. 装入过程                      D. 执行过程
12. 如果允许不同用户的文件可以具有相同的文件名, 通常采用( )来保证按名存取的安全。
- A. 重名翻译机构                      B. 多级目录结构                      C. 建立指针                      D. 建立索引表
13. 位示图法可用于( )
- A. 磁盘空间的管理                      B. 磁盘的驱动调度  
C. 文件目录的查找                      D. 页式虚拟存贮管理中的页面调度
14. 哪一种文件的共享方式, 可以允许文件主随时删除共享文件? ( )
- A. FCB 拷贝复制共享                      B. 基于索引结点的共享方式  
C. 基于符号链的共享方式                      D. 以上都不对
15. 文件系统中, 目录管理的最基本功能是( )
- A. 实现虚拟存储                      B. 实现文件的按名存取  
C. 提高外存的读写速度                      D. 用于存储系统文件

## 二、 填空题 (共 10 空, 每空 1 分, 共 10 分)

1. 进程存在的唯一标志是 \_\_\_\_\_。
2. 处理机调度的层次包括: 高级调度、低级调度、\_\_\_\_\_ 调度。
3. 进程调度包括两种调度方式: 非抢占式进程调度方式、\_\_\_\_\_ 式进程调度方式。
4. 若 P、V 操作的信号量 S 初值为 2, 当前值为 -1, 则表示有 \_\_\_\_\_ 个等待进程。
5. 数据传输控制方式可分为程序 I/O 控制方式、中断控制方式、\_\_\_\_\_ 控制方式和 I/O 通道控制方式。
6. 在文件系统中, 能够唯一标识一个记录的数据项叫做 \_\_\_\_\_。
7. I/O 设备分类中, 按信息交换单位, 可以分为字符设备和 \_\_\_\_\_。
8. 记录式文件的逻辑结构包括: 顺序文件、索引文件、\_\_\_\_\_ 文件。
9. 从用户的角度, 观察到的文件的组织形式叫做文件的 \_\_\_\_\_ 结构。
10. 已知某分页系统主存容量为 64k, 页面大小 2k, 对一个 4 页大的作业, 第 0、1、2、3 页被分配到内存的 2、4、6、7 块中。将十进制逻辑地址 2600 转换成物理地址是 \_\_\_\_\_。

### 三、简答题 (共 3 题, 每题 5 分, 共 15 分)

1. 什么是操作系统? 操作系统的特征有哪些? (5 分)
2. 请画出三种基本状态的转换图, 并注明各状态转换时的条件。(5 分)
3. 简述假脱机打印机系统的实现过程。(5 分)

### 四、算法综合题 (共 6 题, 每题 10 分, 共 60 分)

1. 假设一个系统有 5 个进程, 它们的到达时间和服务时间如表 1 所示, 忽略 I/O 及其他开销时间, 计算先来先服务 FCFS、短进程优先 SPF 进行 CPU 调度, 请完成表 2。(10 分)

进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

表 1 进程到达和服务时间

算法	进程	A	B	C	D	E
FCFS	完成时间					
SPF	完成时间					

表 2

2. 系统中有 (A, B, C, D) 四种资源和五个进程。在银行家算法中某时刻出现下达资源分配情况:

Process	Allocation	Need	Available
P0	(0, 0, 3, 2)	(0, 0, 1, 2)	(1, 6, 2, 2)
P1	(1, 0, 0, 0)	(1, 7, 5, 0)	
P2	(1, 3, 5, 4)	(2, 3, 5, 6)	
P3	(0, 3, 3, 2)	(0, 6, 5, 2)	
P4	(0, 0, 1, 4)	(0, 6, 5, 6)	

试问:

- (1) 该状态是否安全? 请写出分析过程。(6 分)
- (2) 若进程 P2 提出请求 Request(1, 2, 2, 2), 系统能否分配给它资源? 写出分析过程 (4 分)
3. 在一个请求分页系统中, 假如一个作业的页面走向为 0、1、4、2、3、2、6、5、2、1, 分配给该作业的内存物理块数为 3, 分别采用最佳置换算法 (OPT) 和最近最久未使用 (LRU) 置换算法, 计算访问过程中所发生的缺页次数。(写出具体过程)
4. 对于移动头磁盘, 假设磁头现在位于 25 号磁道上, 且基于磁道号的磁盘访问请求序列 (按提出时间的先后次序排列) 为 39、62、18、20、28、100、130、90。试采用最短寻道时间优先算法 (SSTF) 和循环扫描算法 (CSCAN), 分别给出相关磁盘访问请求处理的先后次序, 并计算平均寻道长度。

5. 在 MS-DOS 中有两个文件 A 和 B, A 依次占用 1、2、6、4 盘块; B 依次占用 7、5、9、3 盘块。试问:
- (1) MS-DOS 文件系统采用的是哪种外存分配方式? (2 分)
  - (2) 画出在文件 A 和 B 中各盘块间的链接情况及 FAT 的情况。(8 分)
6. 在  $4 \times 100$  米接力赛中, 4 个运动员之间存在如下关系; 运动员 1 跑到终点把接力棒交给运动员 2; 运动员 2 一开始处于等待状态, 在接到运动员 1 传来的接力棒后才能往前跑, 他跑完 100 米后交给运动员 3, 运动员 3 也只有接到运动员 2 传来的棒后才能跑, 他跑完 100 米后交给运动员 4, 运动员 4 接到棒后跑完全程, 试用信号量机制 (P、V 操作) 描述上述过程。

## 参考答案

### 五、选择题答案

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	B	D	A	A	C	D	D	A	C	C	B	A	C	B

1. 操作系统的形成是一个逐步发展的过程，但**多道批处理系统**的出现被认为是操作系统形成的**关键标志**。理由如下：

1. 资源管理的复杂性显著增加：

- 在单道批处理系统中，内存中只有一个用户作业，CPU、内存等资源的管理相对简单，主要是一个作业完成后自动调入下一个作业。此时的“管理程序”功能还很有限。
- **多道批处理系统**允许多个作业同时驻留在内存中并并发执行。这就要求操作系统必须有效地管理和分配关键资源，如：
  - **CPU 管理**：需要调度算法来决定哪个作业获得 CPU。
  - **内存管理**：需要为多个作业分配和回收内存空间，并进行保护，防止作业间相互干扰。
  - **I/O 设备管理**：多个作业可能同时请求 I/O 操作，需要统一管理和调度。

2. 并发与并行的引入：

- 多道技术使得宏观上多个作业在并发执行（微观上 CPU 可能在它们之间快速切换），这极大地提高了系统资源的利用率（尤其是 CPU 和 I/O 设备的并行工作）。为了支持这种并发性，操作系统必须提供相应的机制，如进程/作业的建立、撤销、同步、通信等。

3. 操作系统核心功能的雏形出现：

- 为了实现多道程序设计，操作系统必须具备**中断处理、进程（或作业）调度、存储管理、设备管理、文件管理**等核心功能。这些功能的集合构成了现代操作系统的基本框架。

4. 从“监控程序”到“资源管理器”的转变：

- 单道批处理时代的管理程序更多地扮演“监控程序”的角色，负责作业的顺序加载和执行。
- 到了多道批处理时代，操作系统真正成为了系统资源的**管理者和调度者**，它负责协调各个并发作业对资源的需求，确保系统高效、有序地运行。

2. • **A. 被选中占有处理机**：这是进程从就绪状态进入运行状态的原因。
- **B. 等待某一事件**：这是进程从运行状态进入阻塞状态（或等待状态）的原因。例如，进程需要进行 I/O 操作，或者等待某个资源。
- **C. 等待的事件已发生**：这是进程从阻塞状态（或等待状态）进入就绪状态的原因。例如，I/O 操作完成，或者等待的资源可用了。
- **D. 时间片用完**：在采用时间片轮转调度的系统中，当一个正在运行的进程用完了分配给它的时间片后，即使它还没有完成任务，操作系统也会剥夺其处理机使用权，使其从运行状态转换为就绪状态，等待下一轮调度。此外，如果出现一个更高优先级的进程，当前运行的低优先级进程也可能被剥夺处理机，从运行态转到就绪态。

## 时间片 (Time Slice/Time Quantum)

时间片 (Time Slice 或 Time Quantum) 是操作系统中进程调度的一种基本单位，主要用于时间片轮转调度算法 (Round-Robin Scheduling)。

### 简单解释:

时间片就像是“让每个程序轮流用 CPU 的时间”。操作系统给每个运行的程序一小段时间 (这个就是“时间片”)，让它使用 CPU。到了时间，没运行完也要“让位”，换下一个程序上去。

### 举个例子:

假设你和 4 个朋友共用一台游戏机，每人每次只能玩 10 分钟：

- 你玩 10 分钟后必须让给下一个人
- 如果你 10 分钟内通关了就退出
- 如果没通关，下次轮到你再接着玩

这 10 分钟就是时间片。

### 时间片的关键点:


项目	说明
作用	防止某个进程长期占用 CPU，提升系统响应速度和公平性
大小	通常是几十毫秒，比如 10ms、50ms
太小会怎样?	上下文切换 (Context Switch) 太频繁，系统开销大
太大会怎样?	某些进程运行太久，响应变慢，失去公平性

### 哪些调度策略用到时间片?

- 时间片轮转 (Round-Robin) 调度算法：每个进程分到一个时间片，按顺序轮流执行。
- 多级反馈队列调度 (Multilevel Feedback Queue)：不同优先级的队列可能设置不同的时间片。

3. “让权等待”的含义是：当某个进程因为等待信号量而阻塞时，它会主动放弃处理器的执行权，让系统调度其他进程运行。

#### A. 整型信号

- 是最原始的信号量机制 (如早期 P/V 操作)，只维护一个整数值。
- 它的 wait(P) 操作通常是忙等 (busy waiting) 的形式，即不断循环检查信号量是否大于 0。
- 在这种情况下，进程并未让出处理器，而是在一直占用 CPU 等待条件满足。
- 所以它不满足“让权等待”机制 

B. 记录型信号量

- 除了计数器，还包含一个等待队列。
- wait 操作会在资源不足时将进程加入等待队列，并让出 CPU，等待调度唤醒。
- 满足让权等待机制。

C. And 型信号量

- 是一种扩展信号量机制，允许一个进程等待多个条件同时满足（如等待多个资源）。
- 实现中也涉及等待队列，进程会被阻塞并让出 CPU。
- 满足让权等待机制。

D. 一般信号量集

- 是操作系统中多个信号量组成的集合，支持对一组信号量的原子操作。
- 使用时也涉及进程阻塞、调度机制。
- 满足让权等待机制。

4. 主存管理是操作系统的核心功能之一，主要指对主存(RAM)的分配、回收、保护和地址映射等的管理。

A. 主存:

- 操作系统直接负责对主存的管理
- 包括内存分配、回收、分页/分段、虚拟存储器管理、地址转换等
- 正确选项 ✓

B. 辅存:

- 辅助存储(如硬盘)属于文件系统管理范畴
- 操作系统管理磁盘文件、目录等，但这是文件管理而非“存储管理”

C. Cache(高速缓存):

- Cache 是硬件层级(CPU 内部或主板上的硬件)
- 由硬件机制控制，操作系统不直接管理

D. 以上都不是:

- 错误，因为 A 是正确答案

5. 页表中的状态位(存在位)用来指示对应页是否已调入内存。

A. 是否被修改过:

- 这是修改位的作用，表示页是否被修改过

B. 是否允许动态增长:

- 动态增长通常与段式存储管理相关，不是页表的状态位

C. 是否已调入内存:

- 正确选项 ✓
- 存在位用于指示该页是否在主存中

D. 是否已置换:

- 置换是指将页从主存中移除到辅存，与存在位无关

6.

12. 当不同用户可以拥有相同的文件名时，操作系统需要一种机制来区分这些文件，防止文件访问混乱，同时保证按“文件名”访问时的安全性与唯一性。

各选项分析： B. 多级目录结构

- 这是标准解决方案
- 每个用户或应用都有自己独立的目录(如 /home/user1/, /home/user2/), 即使文件名相同, 目录路径不同就不会冲突
- 文件的完整路径名是唯一标识: 如 user1/report.txt 与 user2/report.txt 可以并存
- 支持按名存取且保证安全 ✓

A. 重名翻译机构 ✗

- 不是标准术语, 没有具体机制说明
- 更像是“假设的装置”, 非操作系统文件系统的实际组成部分

C. 建立指针 ✗

- 文件系统内部会使用指针(如 inode 指向磁盘块), 但不解决命名冲突
- 指针是为访问数据块服务的, 和“文件名唯一性”无关

D. 建立索引表 ✗

- 索引表(如 FAT 表、inode 表)用于定位文件内容, 不是解决“同名文件”的问题
- 索引表是对文件内容的索引, 不是对路径名或用户的分层管理

14. 文件共享的几种方式比较:

A. FCB 拷贝复制共享

- 复制全部 FCB 内容
- 创建独立副本
- 删除原文件不影响副本

B. 基于索引结点的共享 (硬链接)

- 指向同一个索引结点
- 增加链接计数
- 只有当最后一个链接删除时才能真正删除文件

C. 基于符号链的共享 (软链接) ✓

- 创建一个指向原文件的符号链接
- 原文件可以随时删除
- 删除后符号链接失效
- 类似 Windows 的快捷方式

D. 以上都不对

- 错误, C 是正确答案

15. 进程存在的唯一标志是 PCB 。

## 六、 填空题答案

1. 处理机调度的层次包括: 高级调度、低级调度、中级 调度。
2. 进程调度包括两种调度方式: 非抢占式进程调度方式、抢占 式进程调度方式。



3. 若 P、V 操作的信号量 S 初值为 2, 当前值为 -1, 则表示有 3 个等待进程。
4. 数据传输控制方式可分为程序 I/O 控制方式、中断控制方式、DMA 控制方式和 I/O 通道控制方式。
5. 在文件系统中, 能够唯一标识一个记录的数据项叫做 关键字 。
6. I/O 设备分类中, 按信息交换单位, 可以分为字符设备和 块 。
7. 记录式文件的逻辑结构包括: 顺序文件、索引文件、散列 文件。
8. 从用户的角度, 观察到的文件的组织形式叫做文件的 逻辑 结构。
9. 已知某分页系统主存容量为 64k, 页面大小 2k, 对一个 4 页大的作业, 第 0、1、2、3 页被分配到内存的 2、4、6、7 块中。将十进制逻辑地址 2600 转换成物理地址是 12792 。

## 七、解答题答案

1. 操作系统是一种系统软件, 用于管理计算机的硬件和软件资源, 控制程序的运行, 并提供用户一个方便, 安全, 可靠的工作环境和界面。

操作系统的四个特征包括:

- 并发性: 支持多个进程或线程并发执行, 尽管这些程序实际上是在交替执行的。
- 虚拟性: 通过某种技术, 操作系统可以将一个物理实体划分为多个逻辑上的对应物, 使得用户感觉像是在使用多个独立的计算机系统。
- 异步性: 在多道程序中, 程序的执行顺序和时间都是不确定的, 受到其他程序的影响。
- 共享性: 系统中的资源(如 CPU, 内存等)可以被多个程序共用, 而不是被单个程序独占。

2. 三种基本状态及转换:

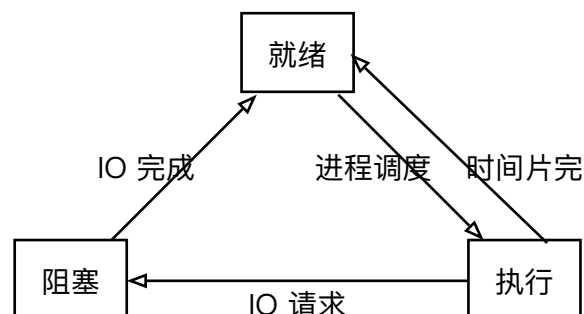


图 1

3. 当用户进程请求打印输长时; spooling 系统意节尚意为该进程 执行打印输出, 但并不是真正地 把打印机分配给该用户进程, 而只是为该进程做两项工作: 一项 是由输出进程在输出井中为之申请一个空闲的存储 空间, 并将要 打印的数据传送其中存放 另十项工作就是由输出进程再为用户进程申请一张空自的用户请求打印表, 并将用定的打印请求填入其中, 然后将该表挂到打印机的请求队列上。

### 假脱机打印 (Spooling – Simultaneous Peripheral Operations)

假脱机打印 (Spooling – Simultaneous Peripheral Operations On-Line) 系统是一种常见的技术, 用于提高打印效率和用户体验。其核心思想是将用户应用程序与慢速的打印机设备解耦, 通过在磁盘上建立一个缓冲区 (打印队列) 来实现。

以下是假脱机打印机系统的实现过程简述:

#### 1. 用户发起打印请求：

- 用户在应用程序（如 Word、浏览器等）中选择“打印”命令。
- 应用程序生成打印数据（可能是原始文本、PostScript、PCL 或特定于操作系统的中间格式如 EMF）。

#### 2. 操作系统/打印子系统介入：

- 应用程序并不直接将数据发送给物理打印机，而是将其发送给操作系统的打印子系统（通常称为打印后台处理程序或 Spooler 服务）。

#### 3. 数据暂存（Spooling）：

- **创建假脱机文件（Spool File）：** 打印后台处理程序接收到打印数据后，会在硬盘上的一个专用目录（称为假脱机目录或打印队列目录）中创建一个或多个临时文件。这些文件包含了实际要打印的内容。
- **创建控制文件（Control File）：** 同时，系统还会为该打印作业创建一个控制文件，其中包含元数据，如：
  - 请求打印的用户名
  - 打印优先级
  - 原始文档名
  - 打印份数、纸张大小等设置
  - 提交时间
  - 假脱机文件的路径
  - 目标打印机名称
- **加入打印队列：** 该打印作业（由假脱机文件和控制文件代表）被加入到目标打印机的打印队列中。打印队列通常是一个先进先出（FIFO）的列表，但也可以根据优先级进行调度。

#### 4. 应用程序释放：

- 一旦打印数据被完整地写入假脱机文件并加入队列，应用程序就可以认为打印任务已“提交”，控制权返回给用户，用户可以继续进行其他工作，无需等待物理打印机完成打印。

#### 5. 后台打印处理（Despooling）：

- **监控打印队列：** 打印后台处理程序会持续监控打印队列。
- **选择作业：** 当指定的物理打印机空闲时，后台处理程序会从队列中选择一个待打印的作业（通常是队首的作业，或优先级最高的作业）。
- **数据处理与发送：**
  - 后台处理程序读取假脱机文件中的数据。
  - 如果需要，它会调用相应的打印机驱动程序（Printer Driver）。打印机驱动程序负责将通用的打印数据（如 EMF）转换为打印机能够理解的特定指令（如 PCL、PostScript）。
  - 处理后的数据被逐步发送到物理打印机的端口（如 USB、网络端口）。
- **状态更新：** 后台处理程序会监控打印过程，更新作业状态（如正在打印、错误、完成等）。

#### 6. 打印完成与清理：

- 一旦物理打印机确认打印完成（或发生不可恢复的错误），后台处理程序会从队列中移除该作业。
- 通常情况下，对应的假脱机文件和控制文件会被删除，释放磁盘空间。
- 可能会记录打印日志。

#### 核心组件：

- **用户应用程序：** 发起打印请求。
- **打印后台处理程序 (Spooler Service/Daemon)：** 核心服务，管理打印队列、接收数据、与打印机通信。
- **假脱机目录 (Spool Directory)：** 硬盘上用于存放假脱机文件和控制文件的区域。
- **打印队列 (Print Queue)：** 待打印作业的逻辑列表。
- **打印机驱动程序 (Printer Driver)：** 将通用打印数据转换为打印机特定指令的软件。
- **物理打印机：** 实际执行打印的硬件设备。

通过这种方式，假脱机打印系统允许多个用户或应用程序同时“打印”到共享的打印机，而不会因为打印机的慢速而阻塞应用程序或用户。它提高了系统整体效率和用户体验。

## 八、 算法综合题答案

1.

算法	进程	A	B	C	D	E
FCFS	完成时间	3	9	13	18	20
SPF	完成时间	3	9	15	20	11

先来先服务 (FCFS) 调度算法 FCFS 算法按照进程到达的先后顺序进行调度。

- **进程 A:**
  - 到达时间: 0
  - 开始执行时间: 0
  - 服务时间: 3
  - **完成时间:  $0 + 3 = 3$**
- **进程 B:**
  - 到达时间: 2
  - A 在 3 完成，所以 B 的开始执行时间: 3 (因为 B 在 A 完成前已到达)
  - 服务时间: 6
  - **完成时间:  $3 + 6 = 9$**
- **进程 C:**
  - 到达时间: 4
  - B 在 9 完成，所以 C 的开始执行时间: 9 (因为 C 在 B 完成前已到达)
  - 服务时间: 4
  - **完成时间:  $9 + 4 = 13$**
- **进程 D:**
  - 到达时间: 6
  - C 在 13 完成，所以 D 的开始执行时间: 13 (因为 D 在 C 完成前已到达)
  - 服务时间: 5
  - **完成时间:  $13 + 5 = 18$**

- **进程 E:**
  - 到达时间: 8
  - D 在 18 完成, 所以 E 的开始执行时间: 18 (因为 E 在 D 完成前已到达)
  - 服务时间: 2
  - **完成时间:  $18 + 2 = 20$**

**FCFS 完成时间:** A: 3, B: 9, C: 13, D: 18, E: 20

2.短进程优先 (SPF) 调度算法 (非抢占式) SPF 算法在当前进程完成后, 从已到达的进程中选择服务时间最短的进程进行调度。

- **时间 0:**
  - 到达进程: A (服务时间 3)
  - A 开始执行。
  - A 完成时间:  $0 + 3 = 3$  (**A 完成**)
- **时间 3:**
  - A 完成。
  - 已到达进程: B (到达时间 2, 服务时间 6)
  - 只有一个 B, B 开始执行。
  - B 完成时间:  $3 + 6 = 9$  (**B 完成**)
- **时间 9:**
  - B 完成。
  - 已到达进程 (在时间 9 或之前):
    - C (到达时间 4, 服务时间 4)
    - D (到达时间 6, 服务时间 5)
    - E (到达时间 8, 服务时间 2)
  - 服务时间最短的是 E (2)。E 开始执行。
  - E 完成时间:  $9 + 2 = 11$  (**E 完成**)
- **时间 11:**
  - E 完成。
  - 已到达且未完成的进程:
    - C (到达时间 4, 服务时间 4)
    - D (到达时间 6, 服务时间 5)
  - 服务时间最短的是 C (4)。C 开始执行。
  - C 完成时间:  $11 + 4 = 15$  (**C 完成**)
- **时间 15:**
  - C 完成。
  - 已到达且未完成的进程:
    - D (到达时间 6, 服务时间 5)
  - 只有一个 D, D 开始执行。
  - D 完成时间:  $15 + 5 = 20$  (**D 完成**)

**SPF 完成时间:** A: 3, B: 9, C: 15, D: 20, E: 11

2. (1) 该状态是否安全? 请写出分析过程。

### 安全性算法步骤：

1. 初始化  $Work = Available = (1, 6, 2, 2)$ 。
2. 初始化  $Finish = [False, False, False, False, False]$  (对应 P0 到 P4)。
3. 寻找一个进程  $P_i$  满足以下两个条件：
  - $Finish[i] = False$
  - $Need[i] \leq Work$  (向量比较, 每个分量都要满足) 如果找到这样的  $P_i$ , 则:
    - $Work = Work + Allocation[i]$
    - $Finish[i] = True$
    - 重复步骤 3。 如果没有找到这样的  $P_i$ , 则转到步骤 4。
4. 如果所有进程的  $Finish[i]$  都为  $True$ , 则系统处于安全状态, 否则处于不安全状态。

### 执行过程：

- 初始:  $Work = (1, 6, 2, 2), Finish = [F, F, F, F, F]$
- 尝试 P0:
  - $Need_{P0} = (0, 0, 1, 2)$
  - $(0, 0, 1, 2) \leq (1, 6, 2, 2)$  ( $0 \leq 1, 0 \leq 6, 1 \leq 2, 2 \leq 2$ ) ? 是
  - P0 可以执行。
  - $Work = Work + Allocation_{P0} = (1, 6, 2, 2) + (0, 0, 3, 2) = (1, 6, 5, 4)$
  - $Finish = [T, F, F, F, F]$
  - 安全序列:  $\langle P0 \rangle$
- 尝试 P1 ( $Finish[P1]=F$ ):
  - $Need_{P1} = (1, 7, 5, 0)$
  - $(1, 7, 5, 0) \leq (1, 6, 5, 4)$  ( $1 \leq 1, 7 \leq 6?, 5 \leq 5, 0 \leq 4$ ) ? 否 (因为  $7 > 6$ )
- 尝试 P2 ( $Finish[P2]=F$ ):
  - $Need_{P2} = (2, 3, 5, 6)$
  - $(2, 3, 5, 6) \leq (1, 6, 5, 4)$  ( $2 \leq 1?, 3 \leq 6, 5 \leq 5, 6 \leq 4?$ ) ? 否 (因为  $2 > 1$  或  $6 > 4$ )
- 尝试 P3 ( $Finish[P3]=F$ ):
  - $Need_{P3} = (0, 6, 5, 2)$
  - $(0, 6, 5, 2) \leq (1, 6, 5, 4)$  ( $0 \leq 1, 6 \leq 6, 5 \leq 5, 2 \leq 4$ ) ? 是
  - P3 可以执行。
  - $Work = Work + Allocation_{P3} = (1, 6, 5, 4) + (0, 3, 3, 2) = (1, 9, 8, 6)$
  - $Finish = [T, F, F, T, F]$
  - 安全序列:  $\langle P0, P3 \rangle$
- 尝试 P1 ( $Finish[P1]=F$ ):
  - $Need_{P1} = (1, 7, 5, 0)$
  - $(1, 7, 5, 0) \leq (1, 9, 8, 6)$  ( $1 \leq 1, 7 \leq 9, 5 \leq 8, 0 \leq 6$ ) ? 是
  - P1 可以执行。
  - $Work = Work + Allocation_{P1} = (1, 9, 8, 6) + (1, 0, 0, 0) = (2, 9, 8, 6)$
  - $Finish = [T, T, F, T, F]$

▸ 安全序列:  $\langle P0, P3, P1 \rangle$

• 尝试 P2 (Finish[P2]=F):

▸ Need\_P2 = (2, 3, 5, 6)

▸  $(2, 3, 5, 6) \leq (2, 9, 8, 6)$  ( $2 \leq 2, 3 \leq 9, 5 \leq 8, 6 \leq 6$ ) ? 是

▸ P2 可以执行。

▸  $Work = Work + Allocation\_P2 = (2, 9, 8, 6) + (1, 3, 5, 4) = (3, 12, 13, 10)$

▸ Finish = [T, T, T, T, F]

▸ 安全序列:  $\langle P0, P3, P1, P2 \rangle$

• 尝试 P4 (Finish[P4]=F):

▸ Need\_P4 = (0, 6, 5, 6)

▸  $(0, 6, 5, 6) \leq (3, 12, 13, 10)$  ( $0 \leq 3, 6 \leq 12, 5 \leq 13, 6 \leq 10$ ) ? 是

▸ P4 可以执行。

▸  $Work = Work + Allocation\_P4 = (3, 12, 13, 10) + (0, 0, 1, 4) = (3, 12, 14, 14)$

▸ Finish = [T, T, T, T, T]

▸ 安全序列:  $\langle P0, P3, P1, P2, P4 \rangle$

**结论:** 由于所有进程的 Finish 标志都为 True, 系统找到了一个安全序列  $\langle P0, P3, P1, P2, P4 \rangle$ 。因此, 该状态是安全的。

(2) 若进程 P2 提出请求 Request\_P2(1, 2, 2, 2), 系统能否分配给它资源? 写出分析过程。

设 Request\_P2 = (1, 2, 2, 2)。

**步骤 1: 检查请求是否小于等于其需求** Need\_P2 \* Need\_P2 (从原表查得) = (2, 3, 5, 6)  
\* Request\_P2 = (1, 2, 2, 2) \* 比较: \*  $1 \leq 2$  (A) – True \*  $2 \leq 3$  (B) – True \*  $2 \leq 5$  (C) – True \*  $2 \leq 6$  (D) – True \* 该条件满足 (Request\_P2  $\leq$  Need\_P2)。

**步骤 2: 检查请求是否小于等于当前可用资源** Available \* Available (原状态) = (1, 6, 2, 2) \* Request\_P2 = (1, 2, 2, 2) \* 比较: \*  $1 \leq 1$  (A) – True \*  $2 \leq 6$  (B) – True \*  $2 \leq 2$  (C) – True \*  $2 \leq 2$  (D) – True \* 该条件满足 (Request\_P2  $\leq$  Available)。

由于以上两个条件均满足, 系统可以尝试分配资源。我们需要更新系统状态, 然后对新状态进行安全性检查。

**步骤 3: 更新系统状态 (假设分配)** \* New\_Available = Available - Request\_P2 = (1, 6, 2, 2) - (1, 2, 2, 2) = (0, 4, 0, 0) \* New\_Allocation\_P2 = Allocation\_P2 + Request\_P2 = (1, 3, 5, 4) + (1, 2, 2, 2) = (2, 5, 7, 6) \* New\_Need\_P2 = Need\_P2 - Request\_P2 = (2, 3, 5, 6) - (1, 2, 2, 2) = (1, 1, 3, 4)

其他进程的 Allocation 和 Need 保持不变。

**步骤 4: 对新状态进行安全性检查** \* 初始: Work = New\_Available = (0, 4, 0, 0), Finish = [F, F, F, F, F]

• 尝试 P0 (Finish[P0]=F):

▸ Need\_P0 = (0, 0, 1, 2)

▸  $(0, 0, 1, 2) \leq (0, 4, 0, 0)$  ( $0 \leq 0, 0 \leq 4, 1 \leq 0?$ ,  $2 \leq 0?$ ) ? 否 (因为  $1 > 0$  for C)

- 尝试 P1 (Finish[P1]=F):
  - Need\_P1 = (1, 7, 5, 0)
  - (1,7,5,0) <= (0,4,0,0) (1≤0?) ? 否 (因为 1 > 0 for A)
- 尝试 P2 (Finish[P2]=F):
  - New\_Need\_P2 = (1, 1, 3, 4)
  - (1,1,3,4) <= (0,4,0,0) (1≤0?) ? 否 (因为 1 > 0 for A)
- 尝试 P3 (Finish[P3]=F):
  - Need\_P3 = (0, 6, 5, 2)
  - (0,6,5,2) <= (0,4,0,0) (0≤0, 6≤4?) ? 否 (因为 6 > 4 for B)
- 尝试 P4 (Finish[P4]=F):
  - Need\_P4 = (0, 6, 5, 6)
  - (0,6,5,6) <= (0,4,0,0) (0≤0, 6≤4?) ? 否 (因为 6 > 4 for B)

在新的状态下，没有一个进程的 Need 小于或等于 Work = (0, 4, 0, 0)。因此，无法找到任何一个进程可以开始执行并最终释放资源。安全性算法无法找到一个安全序列。

**结论：** 由于假设分配资源给 P2 后，系统进入了**不安全状态**，所以系统**不能**将请求的资源 Request\_P2(1, 2, 2, 2) 分配给进程 P2。进程 P2 必须等待。

### 3. 作业的页面走向: 0, 1, 4, 2, 3, 2, 6, 5, 2, 1 分配的物理块数: 3

缺页是指当一个进程试图访问其虚拟地址空间中的某个页面，而该页面当前并未被加载到物理内存（RAM）中时，由硬件（通常是内存管理单元 MMU）产生的一种中断或异常。

最佳置换算法 Optimal Page Replacement Algorithm (OPT)

OPT 算法在需要置换页面时，选择未来最长时间内不会被访问的页面进行置换。

访问 页面	物理 块 1	物理 块 2	物理 块 3	是否 缺 页	换 出 页 面	缺页 次数 累计	备注 (未来访问情况)
						0	初始状态
0	0			是		1	
1	0	1		是		2	
4	0	1	4	是		3	
2	0	1	2	是	4	4	内存满。未来：3,2,6,5,2,1。0 不再访问，1 最后访问，4 不再访问。替换 4 (或 0)。选择 4。
3	3	1	2	是	0	5	内存满。未来：2,6,5,2,1。0 不再访问，1 最后访问，2 马上访问。替换 0。
2	3	1	2	否		5	页面 2 已在内存中
6	6	1	2	是	3	6	内存满。未来：5,2,1。3 不再访问，1 最后访问，2 下下个访问。替换 3。
5	5	1	2	是	6	7	内存满。未来：2,1。6 不再访问，1 下个访问，2 马上访问。替换 6。
2	5	1	2	否		7	页面 2 已在内存中
1	5	1	2	否		7	页面 1 已在内存中

**最佳置换算法 (OPT) 的缺页次数为：7 次。**

## 2.最近最久未使用 (LRU) 置换算法

LRU 算法在需要置换页面时，选择最近最久未被访问的页面进行置换。我们将使用一个栈来表示页面的使用情况，栈顶为最近使用的，栈底为最久未使用的。



访问页面	物理块 (栈底 -> 栈顶)	内存状态 (逻辑)	是否缺页	换出页面	缺页次数累计	备注 (栈更新: 访问则移到栈顶, 缺页则替换栈底并移到栈顶)
	[ , , ]				0	初始状态
0	[0, , ]	[0]	是		1	0 入栈
1	[0, 1, ]	[0,1]	是		2	1 入栈
4	[0, 1, 4]	[0,1,4]	是		3	4 入栈, 内存满。LRU:0, MRU:4
2	[1, 4, 2]	[1,4,2]	是	0	4	0 最久未使用, 被换出。2 入栈。LRU:1, MRU:2
3	[4, 2, 3]	[4,2,3]	是	1	5	1 最久未使用, 被换出。3 入栈。LRU:4, MRU:3
2	[4, 3, 2]	[4,3,2]	否		5	2 已在内存, 更新为 MRU。LRU:4, MRU:2
6	[3, 2, 6]	[3,2,6]	是	4	6	4 最久未使用, 被换出。6 入栈。LRU:3, MRU:6
5	[2, 6, 5]	[2,6,5]	是	3	7	3 最久未使用, 被换出。5 入栈。LRU:2, MRU:5
2	[6, 5, 2]	[6,5,2]	否		7	2 已在内存, 更新为 MRU。LRU:6, MRU:2
1	[5, 2, 1]	[5,2,1]	是	6	8	6 最久未使用, 被换出。1 入栈。LRU:5, MRU:1

最近最久未使用 (LRU) 置换算法的缺页次数为：8 次。

总结：\* 最佳置换算法 (OPT)：7 次缺页

4. 已知信息：\* 当前磁头位置：25 号磁道 \* 磁盘访问请求序列 (按到达时间)：39, 62, 18, 20, 28, 100, 130, 90 \* 为方便计算, 我们将请求序列按磁道号排序 (SSTF 不需要, 但对理解 SCAN 类有帮助)：18, 20, 28, 39, 62, 90, 100, 130。

#### 八、零、一、1. 最短寻道时间优先算法 (SSTF)

SSTF 算法选择与当前磁头位置最近的请求进行服务。

##### 1. 当前磁头位置：25

- 未处理请求：{39, 62, 18, 20, 28, 100, 130, 90}
- 与 25 的距离：
  - $|39-25|=14$
  - $|62-25|=37$
  - $|18-25|=7$
  - $|20-25|=5$
  - $|28-25|=3$  <- 最短
- 服务 28。寻道长度： $|28-25| = 3$ 。磁头移至 28。

2. 当前磁头位置：28

- 未处理请求：{39, 62, 18, 20, 100, 130, 90}
- 与 28 的距离：
  - $|39-28|=11$
  - $|62-28|=34$
  - $|18-28|=10$
  - $|20-28|=8$  <- 最短
- 服务 20。寻道长度： $|20-28| = 8$ 。磁头移至 20。

3. 当前磁头位置：20

- 未处理请求：{39, 62, 18, 100, 130, 90}
- 与 20 的距离：
  - $|39-20|=19$
  - $|62-20|=42$
  - $|18-20|=2$  <- 最短
- 服务 18。寻道长度： $|18-20| = 2$ 。磁头移至 18。

4. 当前磁头位置：18

- 未处理请求：{39, 62, 100, 130, 90}
- 与 18 的距离：
  - $|39-18|=21$  <- 最短
  - $|62-18|=44$
  - $|100-18|=82$
  - $|130-18|=112$
  - $|90-18|=72$
- 服务 39。寻道长度： $|39-18| = 21$ 。磁头移至 39。

5. 当前磁头位置：39

- 未处理请求：{62, 100, 130, 90}
- 与 39 的距离：
  - $|62-39|=23$  <- 最短
  - $|100-39|=61$
  - $|130-39|=91$
  - $|90-39|=51$
- 服务 62。寻道长度： $|62-39| = 23$ 。磁头移至 62。

6. 当前磁头位置：62

- 未处理请求：{100, 130, 90}
- 与 62 的距离：
  - $|100-62|=38$
  - $|130-62|=68$
  - $|90-62|=28$  <- 最短
- 服务 90。寻道长度： $|90-62| = 28$ 。磁头移至 90。

7. 当前磁头位置：90

- 未处理请求：{100, 130}
- 与 90 的距离：

- $|100-90|=10$  <- 最短
- $|130-90|=40$
- 服务 100。寻道长度： $|100-90| = 10$ 。磁头移至 100。

#### 8. 当前磁头位置：100

- 未处理请求：{130}
- 服务 130。寻道长度： $|130-100| = 30$ 。磁头移至 130。

**SSTF 结果：** \* 磁盘访问请求处理的先后次序： 28, 20, 18, 39, 62, 90, 100, 130 \* 总寻道长度： $3 + 8 + 2 + 21 + 23 + 28 + 10 + 30 = 125$  \* 平均寻道长度： $125 / 8 = 15.625$

### 八、零、二、2. 循环扫描算法 (CSCAN)

CSCAN 算法磁头沿一个方向移动，到达磁盘一端后，立即返回到磁盘另一端的起始位置，再继续沿相同方向扫描。我们假设磁头初始移动方向是磁道号增加的方向。（如果未给出磁盘最大磁道号，通常假设其扫描到当前方向上最后一个请求后，直接返回到磁盘的 0 号磁道，然后继续扫描。）

#### 1. 当前磁头位置：25。移动方向：增加。

- 请求序列中大于等于 25 的有：28, 39, 62, 90, 100, 130。
- 服务顺序 (递增)：28, 39, 62, 90, 100, 130。
- 磁头移动路径： $25 \rightarrow 28 \rightarrow 39 \rightarrow 62 \rightarrow 90 \rightarrow 100 \rightarrow 130$
- 寻道长度： $(28-25) + (39-28) + (62-39) + (90-62) + (100-90) + (130-100) = 3 + 11 + 23 + 28 + 10 + 30 = 105$

#### 2. 磁头到达 130 (当前方向最后一个请求)。

- 磁头立即返回到磁盘的起始位置 (0 号磁道)。
- 磁头移动路径： $130 \rightarrow 0$
- 寻道长度 (返回)： $|0-130| = 130$

#### 3. 磁头位置：0。移动方向：增加。

- 剩余未处理请求 (按磁道号排序)：18, 20。
- 服务顺序 (递增)：18, 20。
- 磁头移动路径： $0 \rightarrow 18 \rightarrow 20$
- 寻道长度： $(18-0) + (20-18) = 18 + 2 = 20$

**CSCAN 结果：** \* 磁盘访问请求处理的先后次序： 28, 39, 62, 90, 100, 130, 18, 20 \* 总寻道长度 (包括返回的寻道)： $105$  (服务 28 至 130) +  $130$  (从 130 返回 0) +  $20$  (服务 18 至 20) =  $255$  \* 平均寻道长度： $255 / 8 = 31.875$

**总结：**

算法	磁盘访问请求处理的先后次序	总寻道长度	平均寻道长度
SSTF	28, 20, 18, 39, 62, 90, 100, 130	125	15.625
CSCAN (向增加方向)	28, 39, 62, 90, 100, 130, 18, 20	255	31.875

5. (1) MS-DOS 文件系统主要采用的是链式分配 (Linked Allocation) 方式的一种变种，具体来说就是文件分配表 (File Allocation Table – FAT) 链接分配。  
(2)

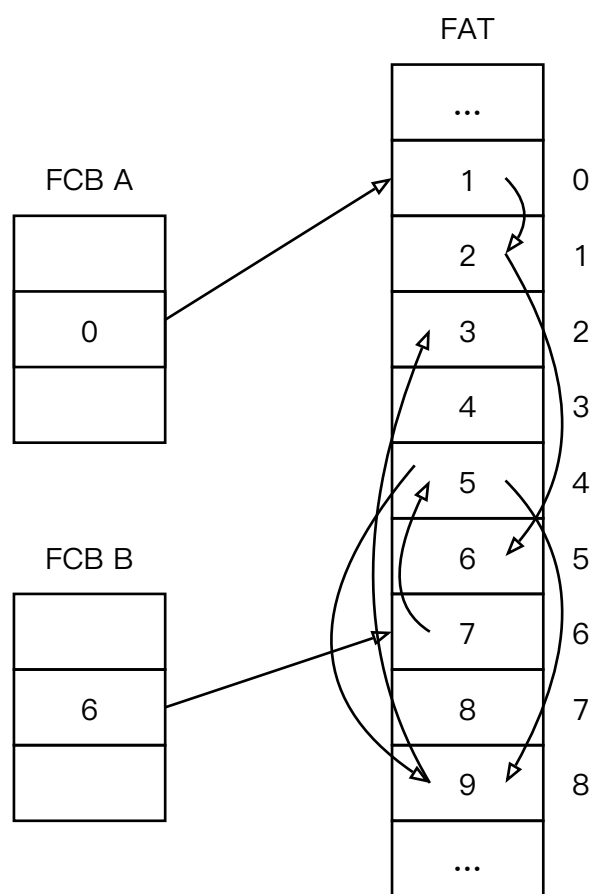


图 2 示意图

6.

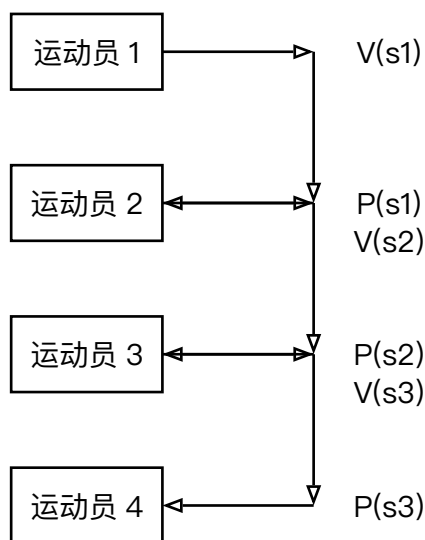


图 3 4x100 米接力赛信号量同步示意图

接力赛信号量同步代码

假设定义三个信号量：

- s1: 运动员 1 传给运动员 2 的同步信号
- s2: 运动员 2 传给运动员 3 的同步信号
- s3: 运动员 3 传给运动员 4 的同步信号

```
semaphore s1 = 0; // 初值为0  
semaphore s2 = 0; // 初值为0  
semaphore s3 = 0; // 初值为0
```

```
Runner1() {  
    run_100m();    // 跑100米  
    V(s1);        // 通知运动员2可以开始跑  
}
```

```
Runner2() {  
    P(s1);        // 等待运动员1的接力棒  
    run_100m();    // 跑100米  
    V(s2);        // 通知运动员3可以开始跑  
}
```

```
Runner3() {  
    P(s2);        // 等待运动员2的接力棒  
    run_100m();    // 跑100米  
    V(s3);        // 通知运动员4可以开始跑  
}
```

```
Runner4() {  
    P(s3);        // 等待运动员3的接力棒  
    run_100m();    // 跑完最后100米  
}
```