25 年 408 考试大纲改动补充

(一)CPU 调度算法

1.基于公平原则的调度算法
   1）保证调度算法
   2）公平分享调度算法
2.实时调度算法
   1）分类
   2）最早截止时间优先算法（EDF）
   3）最低松弛度优先算法

(二)多处理器调度

1.多处理机操作系统
   1）特征
   2）功能对比
2.多处理机系统的类型
   1）按 CPU 之间耦合的紧密程度（了解）
   2）按系统中所用 CPU 是否相同
3.多处理机操作系统的进程调度
   1）对称多处理器 SMP 的进程分配方式
   2）对称多处理器 SMP 的负载均衡
   3）进程(线程)调度方式

(三)页框回收

1.伙伴算法（*Buddy*）
2.页框回收算法（*PFRA*）

(四)进程间通信：信号

1.信号（*Signal*）的概念
2.信号的来源
3.常见的信号
4.信号传递和响应
5.IPC 机制对比

25 年 408 考试大纲改动补充

（操作系统部分）

(一)CPU 调度算法

虽然删除了“典型调度算法”，但新增“CPU 调度算法”实际可考查范围更广（更偏）。
以下内容主要参考自《计算机操作系统（慕课版）》，**该补充仅作参考**。
除开已有的“先来先服务调度算法，短作业（短进程、短线程）优先调度算法，时间片轮转调度算法，优先级调度算法，高响应比优先调度算法，多级队列调度算法，多级反馈队列调度算法。”

1.基于公平原则的调度算法

考虑公平性，（对比其他的算法仅保证进程优先运行，而不考虑运行时间）。

1）保证调度算法

(1)算法描述

公平的分配处理机。如果在系统中有 n 个相同类型的进程同时运行，须保证每个进程都能获得相同的处理机时间，如 $1/n$。

(2)公平调度算法的系统具有的功能

①跟踪计算每个进程自创建以来已经执行的处理时间；
②计算每个进程应获得的处理机时间，即自创建以来的时间除以 n；
③计算进程获得处理机时间的比率，即进程实际执行的处理时间和应获得的处理机时间之比；
④比较各进程获得处理机时间的比率，例如，进程 A 的比率为 0.5，进程 B 的比率为 0.8，进程 C 的比率为 1.2，则通过比较发现，进程 A 的比率最低；
⑤调度程序应选择比率最小的进程，将处理机分配给它，并让它一直运行，直到它的比率超过最接近它的进程的比率为止。

2）公平分享调度算法

(1)引入——对用户不公平

如果各用户所拥有的进程数不同，分配给每个进程相同的处理机时间，就会发生对用户的不公平问题。

假如系统中仅有两个用户，用户 1 启动了 4 个进程，用户 2 只启动了 1 个进程，采用 RR 调度算法让每个进程轮流运行一个时间片的时间，这对进程而言很公平，但用户 1 和用户 2 得到的处理机时间分别为 80%和 20%，这对用户而言不公平。

(2)算法描述

该算法调度的公平性是针对用户的，即所有用户能获得相同的处理机时间或所要求的时间比例相同。因调度以进程为基本单位，必须考虑每个用户所拥有的进程数目。

(3)例子

系统中有两个用户，用户 1 有 4 个进程 A、B、C、D，用户 2 有 1 个进程 E。为保证两个用户能获得相同的处理机时间，则必须执行如下强制调度序列：

A E B E C E D E A E B E C E D E...

如果希望用户 1 所获得的处理机时间是用户 2 的两倍，则必须执行如下强制调度序列：

A B E C D E A B E C D E A B E C D E...

2.实时调度算法

实时调度必须要满足实时任务对截止时间的要求。

1）分类

(1)非抢占式调度算法

①**非抢占式轮转调度算法**。给若干个相同的对象各建立一个实时任务，并将它们排成一个轮转队列。调度程序每次选择队列中的第一个任务投入运行，任务完成后挂在轮转队列的末尾进行等待，调度程序再选择下一个队首任务运行。

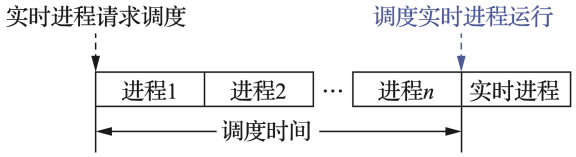
②**非抢占式优先级调度算法**。若系统中含有少数实时任务，可以为这些任务赋予较高的优先级。系统会把它们安排在就绪队列的队首，等待当前任务自我终止或运行完成后，再通过调度执行队首的高优先级进程。

(2)抢占式调度算法

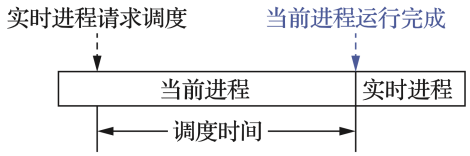
根据抢占发生时间的不同进行分类

①**基于时钟中断的抢占式优先级调度算法**。优先级更高的实时任务到达后，并不立即抢占当前任务的处理机，而是等到时钟中断发生后，调度程序才会剥夺当前任务的执行，将处理机分配给新到的高优先级任务。

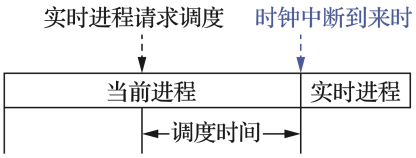
②**立即抢占的优先级调度算法**。要求 OS 具有快速响应外部中断事件的能力。一旦出现外部中断，只要当前任务未处于临界区，便能立即剥夺当前任务的执行，把处理机分配给请求中断的紧迫任务。



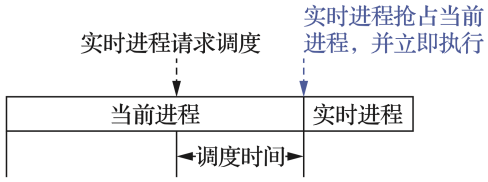
（a）非抢占式轮转调度算法



（b）非抢占式优先级调度算法



（c）基于时钟中断的抢占式优先级调度算法



（d）立即抢占的优先级调度算法

2）最早截止时间优先算法（EDF）

(1)算法描述

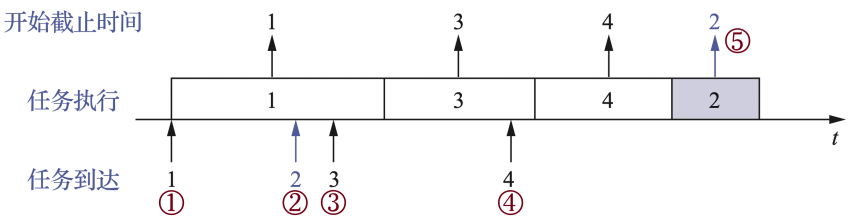
根据任务的截止时间来确定任务的优先级。截止时间越早，其优先级越高。

在系统中保持一个实时任务就绪队列，截止时间早的排在队列的前面，调度时选择就绪队列中的第一个任务。

既可用于抢占式调度方式，也可用于非抢占式调度方式中。

(2)非抢占式调度方式

用于非周期实时任务。（该部分仅了解，以抢占式为主）

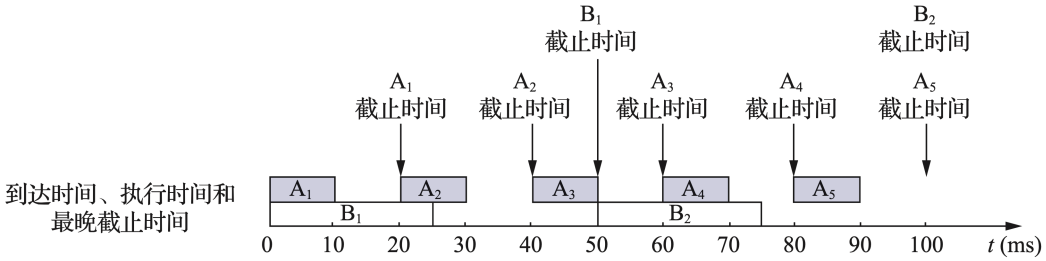


该例中具有 4 个非周期实时任务，它们先后到达。

①②③④分别表示任务 1、2、3、4 的到达。由于任务 3 的开始截止时间早于任务 2 的，故系统在执行完任务 1 后先调度任务 3 执行。在此期间任务 4 又到达了，其开始截止时间仍早于任务 2 的，故在任务 3 执行完后，系统又会先调度任务 4 执行，最后才调度任务 2 执行。

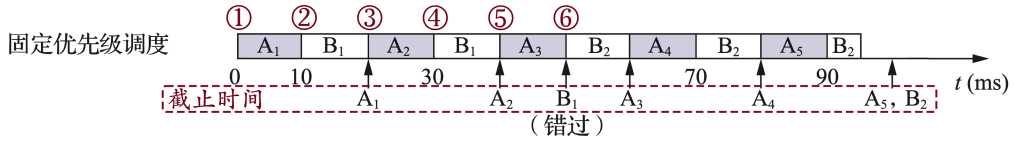
(3)抢占式调度方式

用于周期实时任务。



有两个周期实时任务，任务 A 和任务 B 的周期时间分别为 $20ms$ 和 $50ms$，每个周期的处理时间分别为 $10ms$ 和 $25ms$。上图画出了两个任务的到达时间、截止时间和执行时间。

1.假定任务 *A* 具有较高的优先级：



①$t=0ms$：先调度 A_1 执行；

②$t=10ms$：A_1 完成，调度 B_1 执行。

③$t=20ms$：又重新调度 A_2 执行，此时B_1处理了 $10ms$，剩余 $15ms$。

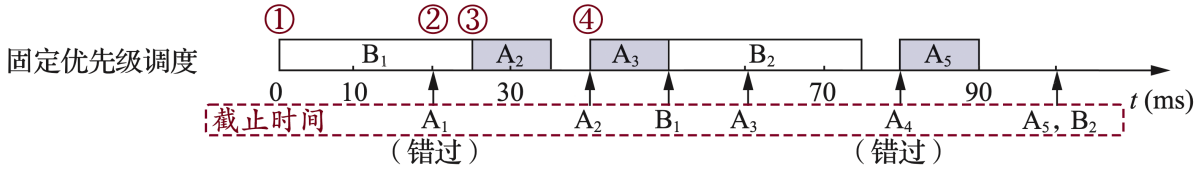
④$t=30ms$：A_2 完成，又调度 B_1 执行。

⑤$t=40ms$ 时，又调度 A_3 执行，此时B_1共计处理了 $20ms$，剩余 $5ms$。

⑥在 $t=50ms$ 时，虽然 A_3 已完成，但 B_1 已错过了它的最后期限。

这说明利用通常的优先级调度已经失败。

2.假定任务 *B* 具有较高的优先级：



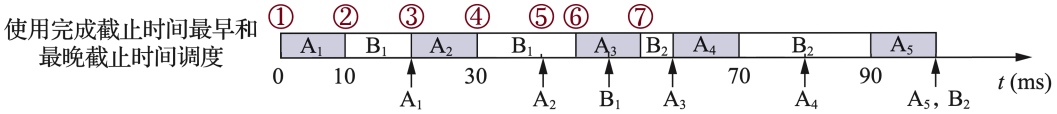
①$t=0ms$：先调度 B_1 执行；

②$t=20ms$：此时仍然在执行 B_1，但 A_1 已到达它的最后期限。

③$t=25ms$：B_1 完成，A_1 已错过它的最后期限。该算法失败。

④$t=40ms$：A_2 到达，此时没有其他的任务，则开始执行 A_2。

3.采用 EDF 算法



①$t=0ms$：A_1 和 B_1 同时到达，由于 A_1 的截止时间比 B_1 早，故调度 A_1 执行。

②$t=10ms$：A_1 完成，又调度 B_1 执行。

③$t=20ms$：A_2 到达，由于 A_2 的截止时间比 B_1 早，故 B_1 被中断而调度 A_2 执行。

④$t=30ms$：A_2 完成，又重新调度 B_1 执行。

⑤$t=40ms$：A_3 到达，但 B_1 的截止时间要比 A_3 早，因此仍让 B_1 继续执行直到完成。

⑥$t=45ms$：B_1 完成，调度 A_3 执行。

⑦$t=55ms$：A_3 完成，调度 B_2（$50ms$ 时到达）执行。

在该例中，利用 EDF 算法可以满足系统的要求。

3）最低松弛度优先算法

(1)算法描述

根据任务紧急程度（松弛度），来确定任务的优先级。任务的紧急程度越高，优先级就越高。

松弛度=必须完成时间−其本身的运行时间−当前时间

在系统中保持一个按松弛度排序的实时任务就绪队列，松弛度最低的任务排在队列的前面，调度时选择就绪队列中的第一个任务。

主要用于可抢占调度方式中。

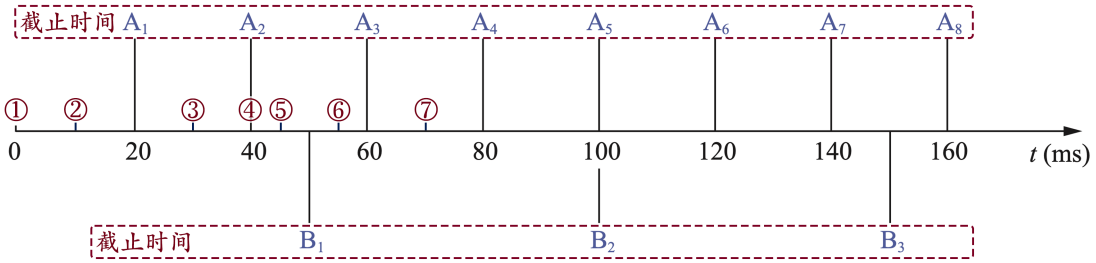
(2)松弛度

松弛度是任务的截止时间与当前时间的差值减去任务执行时间。如果一个任务的松弛度为0，表示该任务必须在当前时间立即执行，因为它没有时间缓冲。

例：一个任务在 $200ms$ 时必须完成，而它本身所需的运行时长是 $100ms$，因此调度程序必须在 $100ms$ 之前调度执行，该任务的松弛度为 $100ms$。

(3)例子

假如在一个实时系统中有两个周期性实时任务 A 和 B，任务 A 要求每 $20ms$ 执行一次，执行时长为 $10ms$，任务 B 要求每 $50ms$ 执行一次，执行时长为 $25ms$。



①$t=0ms$：A_1 必须在 $20ms$ 时完成，它本身运行需 $10ms$，则 A_1 的松弛度为$10ms$。B_1 必须在 $50ms$ 时完成，它本身运行需 $25ms$，则 B_1 的松弛度为 $25ms$，故先调度 A_1 执行。

②$t=10ms$：A_1 执行完毕，A_2 的松弛度$=40ms-10ms-10ms=20ms$。B_1 的松弛度为 $15ms$，故应选择 B_1 运行。

③$t=30ms$：A_2 的松弛度已减为 $0ms$（即 $40-10-30$），而 B_1 的松弛度为 $15ms$（即 $50-5-30$）。此时应抢占处理机给A_2运行。

④$t4=40ms$：A_3 的松弛度为 $10ms$，而 B_1 的松弛度仅为 $5ms$，故又应重新调度 B_1 执行。

⑤$t5=45ms$：B_1 执行完毕，此时 A_3 的松弛度为 $5ms$， B_2 的松弛度为 $30ms$，应调度 A_3 执行。

⑥$t6=55ms$：任务 A 尚未进入第 4 周期，而任务 B 已进入第 2 周期，故再调度 B_2 执行。

⑦$t7=70ms$：A_4 的松弛度已减至 $0ms$，而 B_2 的松弛度为 $20ms$，故应抢占 B_2 的处理机而调度 A_4 执行。

(二)多处理器调度

这部分内容有几率和计组 CPU 的多处理器联动。作为新考点，命题方向暂不明确（只能广撒网），以下拓展内容仅作了解即可。

1.多处理机操作系统

安装在多处理机系统上的 OS 称为**多处理机操作系统**。

1）特征

对比单处理机多道程序系统。

	单处理机多道程序系统	多处理机操作系统
并行性	为用户模拟多处理机环境，使程序能够并发执行	存在多个实处理机，可以使多个程序并行执行。
分布性	所有任务都在同一台处理机上执行，系统统一管理所有文件和资源；	①任务的分布：同一作业可划分为多个子任务，分配到多个处理机并行执行。 ②资源的分布：各处理机可拥有属于自己的本地资源（存储器、I/O 设备等）。 ③控制的分布：系统的每个处理单元上，都可配置有自己的 OS，负责控制本地进程的运行和管理本地资源，以及协调各处理机之间的通信和资源共享。
通信和同步性	——	不同处理机运行的不同进程间，也需要实现同步和通信
可重构性	一旦处理机发生故障，将引发整个系统的崩溃。	各处理机的结构和功能一般相同，某个资源发生故障时，系统能够自动切除故障资源并换上备份资源，以使系统能够继续工作。

2）功能对比

		单处理机多道程序系统	多处理机操作系统
进程管理	进程同步	各进程只能交替执行，不会发生两个进程同时访问系统中同一共享资源的情况	要解决同一处理机和多个不同的处理机程序并行执行所引发的同步问题。
	进程通信	同一台机器中的进程间的主要通信方式是共享存储器方式和直接通信方式。	涉及处理机之间的通信，采用间接通信方式
	进程调度	按照一定的算法，简单地从就绪队列中选择一个进程，并为之分配处理机的一个时间片，使之执行一段时间。	应考虑如何实现负载的平衡。 （后续重点学习）
存储器管理		地址变换机构、虚拟存储器功能	每个处理机有自己的本地存储器和共享的系统存储器。访问本地存储器的时间通常比访问系统存储器或其他处理机的远程存储器要短。 **功能**：①**地址变换机构**；②**访问冲突仲裁机构**；③**数据一致性机制**
文件管理		通常只有一个文件系统，所有的文件都存放在磁盘上，采用集中统一管理方式进行管理（集中式文件系统）。	①**集中式**：所有用户文件集中存放在一个处理机的文件系统中，由该处理机统一管理。 ②**分散式**：各处理机可管理自己的文件系统，但系统无法有效组织，无法实现文件共享。 ③**分布式**：文件分布在不同处理机上，逻辑上为一个整体，用户无需关心文件物理位置即可访问，需重点解决文件存取速度与保护问题。

2.多处理机系统的类型

1）按 CPU 之间耦合的紧密程度（了解）

(1)紧密耦合多处理机系统

①**共享内存系统**：所有 CPU 通过高速总线或交叉开关访问共享内存，访问时间为 $10ns\sim50ns$。

②**分布式存储系统**：每个 CPU 连接到独立的存储器模块，CPU 间通过消息通信互连，消息传递时间为 $10\mu s\sim50\mu s$。虽然较慢且软件复杂，但硬件方便。

(2)松散耦合多处理机系统

通过通道或通信线路连接，每台计算机拥有独立的存储器和 I/O 设备，并由各自的操作系统管理。计算机独立工作，信息交换时间为$10ms\sim50ms$。

2）按系统中所用 CPU 是否相同

(1)对称多处理机（SMP）系统（主要学习这种）

- 每个处理器功能和结构相同，且运行自己的调度程序。
- 进程可位于共同的就绪队列或各处理器的私有队列中。每个处理器的调度程序会检查共同队列来选择一个进程执行。
- 如果多个处理器同时访问或更新同一数据结构，必须确保：不同处理器不会选择同一进程，且进程不会从队列中丢失。
- 调度程序对共享资源的访问需要进行同步处理。

(2)非对称多处理机（ASMP）系统

- 包含多种不同类型的 CPU，让一个主处理器负责所有调度决定、I/O 处理以及其他系统活动，其他的处理器只负责执行用户代码。
- 优点：只有主处理器访问系统数据和资源，减少了数据共享和冲突。
- 缺点：主处理器故障会导致系统崩溃，且主处理器容易成为性能瓶颈。

3.多处理机操作系统的进程调度

在同构型多处理机操作系统中，进程可以分配到任意处理机上运行；而在非对称多处理机操作系统中，进程只能分配到特定的处理机执行。

1）对称多处理器 SMP 的进程分配方式

在对称多处理机操作系统中，所有处理机相同，因此可以将它们视为一个处理机池，调度程序可以将进程分配给池中的任何处理机。

特性	静态分配方式	动态分配方式
分配策略	进程在整个执行期间固定分配到一个处理机上运行	进程在每次调度时随机分配到空闲的处理机
就绪队列	每个处理机有独立的就绪队列	设有一个公共的就绪队列
调度开销	调度开销小	对于紧密耦合系统，调度开销小； 对于松散耦合系统，调度开销大。
处理机负载	可能出现处理机忙闲不均	负载均衡，消除忙闲不均现象
进程迁移	不发生迁移	进程可能从一个处理机迁移到另一个处理机
适用系统类型	适用于对称多处理机（SMP）	适用于紧密耦合和松散耦合多处理机系统
进程信息传输	不需要进程信息传输	对于松散耦合系统，需要额外传输进程信息

2）对称多处理器 SMP 的负载均衡

(1)负载均衡

在 SMP 系统中，保持处理器的负载平衡很重要，否则可能出现部分处理器空闲，其他处理器过载的情况。

对于处理器有私有可执行进程队列的系统，负载均衡是必要的；而对于公共队列系统，处理器可直接从公共队列获取进程，负载均衡通常不必特别设计。

(2)实现方法

为实现负载均衡，SMP 系统有两种动态进程分配方式：

●推迁移：定期检查每个处理器的负载，若发现不平衡，会将进程从过载处理器推送到空闲或较少负载的处理器。

●拉迁移：空闲处理器主动从繁忙处理器上拉取等待中的进程。

(3)处理器亲和性

当进程迁移到其他处理器时，原处理器的缓存内容会失效，新处理器的缓存需要重新填充。

概念	描述
缓存失效	当进程迁移到其他处理器时，原处理器缓存中与该进程相关的数据需被标记为无效。
缓存重新填充	新处理器需要重新从内存加载该进程所需的数据，导致性能下降。
处理器亲和性	进程对其运行的处理器具有依赖性或偏好，避免频繁迁移，减少缓存失效和重新填充的开销。
软亲和性	操作系统尝试让进程保持在同一处理器上运行，但不强制执行，进程可以迁移到其他处理器。
硬亲和性	通过系统调用，将进程限制在特定处理器或处理器子集上运行，不允许自由迁移。

(4)不足

负载均衡可能会影响处理器亲和性，即进程在同一处理器上运行时可利用缓存数据。如果进程频繁迁移，这一优势会丧失。不同系统对负载均衡的处理方式不同，有些系统在处理器空闲时总是拉取进程，而有些系统则仅在不平衡达到一定程度后才进行进程迁移。

3）进程(线程)调度方式

多处理机操作系统中的调度方式

(1)自调度机制

概述	直接继承自单处理机操作系统的调度机制。系统设置一个公共的进程或线程就绪队列，空闲的处理机可以从该队列中获取进程或线程运行。常用的调度算法可用 FCFS、最高优先级优先（HPF）、抢占式 HPF 等。
优点	调度算法灵活，可以轻松地使用单处理机操作系统中已有的各种调度算法；处理机利用率高，只要就绪队列非空，就不会出现处理机空闲或不均匀利用的问题。
缺点	系统瓶颈，多个处理机共享一个就绪队列，只能互斥访问；低效性，当线程在多个处理机之间切换时，可能会导致高速缓存的效率下降，因为线程在不同处理机上的缓存数据失效；线程切换频繁，合作型线程可能因为竞争处理机而频繁阻塞和切换。

(2)成组调度方式

通过将一个进程中的一组线程分配到一组处理机上执行，从而减少线程切换的频率。

分类	面向所有应用程序的平均分配	系统中的处理机时间被平均分配给每个应用程序。
	面向所有线程的平均分配	根据应用程序中线程的数量来分配处理机时间。
例子	若系统有 4 台处理机和 2 个应用程序（应用程序 A 有 4 个线程，应用程序 B 有 1 个线程）。 ● 面向所有应用程序的平均分配：每个应用程序分别占用 4 台处理机的一半时间（即 50%）。当应用程序 A 运行时，4 个处理机都在忙碌，而应用程序 B 运行时，只有 1 个处理机在忙碌，其他 3 个空闲。造成处理机时间的浪费。 ● 面向所有线程的平均分配：应用程序 A 和 B 共 5 个线程，应为应用程序 A 分配 4/5 的时间，为应用程序 B 分配 1/5 的时间。减少处理机时间的浪费。	
优点	①**减少线程阻塞**：如果一组相互合作的线程能并行执行，成组调度可以有效减少线程阻塞情况，从而降低线程切换频率，提升系统性能。 ②**降低调度频率**：每次调度可以同时处理一组线程的处理机分配问题，显著降低调度频率和调度开销。	

(3)动态调度方式

动态调度原则	说明
空闲则分配	当有一个或多个作业请求处理机时，若系统中存在空闲处理机，系统会立即将其分配给请求的作业，以满足作业需求。
新作业绝对优先	新到达的、尚未获得处理机的作业优先获取处理机。如果没有空闲处理机，系统会从已有多个处理机分配的作业中回收处理机，并将其分配给新作业。
保持等待	若系统无法满足作业对处理机的请求，作业会保持未完成状态，直到有处理机空闲可分配，或作业主动取消请求。

释放则分配	当作业释放一个或多个处理机时，系统扫描处理机请求队列。首先将处理机分配给新作业，其次按照先来先服务 (FCFS) 原则分配剩余的处理机。
优点	允许作业在运行期间根据需求动态调整线程数，系统处理机资源利用率提高。
缺点	复杂性较高，调度决策频繁变动，带来额外的开销，可能抵消其部分优势。

(三)页框回收

1.伙伴算法（*Buddy*）

这个概念第一次出现在 2024 真题中。了解即可。

(1)概念

所有分区的大小均为 $2^k(i \le k \le m)$：

2^i 表示最小分区的大小；2^m 表示最大分区的大小（通常是整个可分配内存的大小）。

系统中共有 m 个空闲分区链，按分区大小 2^k 排列，表示不同大小的空闲分区。

伙伴：两个块大小相同、地址连续、必须是同一个大块中分离出来的；

(2)分配

搜索最接近要求的、大小为 2^k 的空闲分区链，如果有空闲分区，直接分配；否则，搜索大小为 2^{k+1} 的空闲分区链，并将一个分区分成一对伙伴，其中的一个分配出去，另一个插到大小为 2^k 的空闲分区链中。

(3)回收

如果回收区的伙伴不在空闲分区链中（没有伙伴块），则只需将回收分区插入相应大小的空闲分区链；否则，需要将该分区及其伙伴合并成一个更大分区。

继续考察合并后的块在更大一级链表中是否有伙伴存在，直到不能合并或者已经合并到了最大的块。

将分区块插入对应空闲分区链。

(4)优先

I. **减少外部碎片**：伙伴系统通过合并相邻的空闲块有效减少外部碎片。
II. **高效的分配与释放**：由于采用 2 的幂次方进行内存分配，分配和释放操作非常迅速。
III. **快速合并**：当内存释放时，系统可以快速地检查是否能合并相邻的伙伴块，从而优化内存使用。

(5)缺点

I. **内部碎片**：由于内存块按 2 的幂次方分配，如果所需内存大小不是 2 的幂，会导致一定的内存浪费。
II. **合并受阻**：如果某个小块没有释放，可能阻碍相邻较大块的合并，导致内存无法有效回收。
III. **操作开销**：拆分和合并操作涉及大量链表和位图管理，增加了系统的计算开销。

2.页框回收算法（*PFRA*）

(1)概念

页框回收（*Page Frame Reclaiming Algorithm*，PFRA）是 Linux 内核中用于回收物理内存页的一种算法。当系统内存不足时，内核会触发页框回收算法，从用户态进程和内核高速缓存中“窃取”页框，以补充伙伴系统的空闲块列表。

主要目标是维持一个最小的空闲页框池，从而保证内核能够安全地从内存紧缺的状态中恢复过来。

(2)选择目标页

按照页框所含内容的不同，将页框区分为以下几种类型，并以不同的方式处理：

页类型	说明	回收操作
不可回收页	本身就空闲的页、保留页、内核动态分配页、进程内核态堆栈页、临时锁定页、内存锁定页	不允许也无需回收
可交换页	用户态进程的匿名页或与其他进程共享的页，这些页在硬盘上没有对应的区域	将页的内容保存在交换区
可同步页	用户态地址空间的映射页、存有磁盘文件数据且在高速缓存中的页、块设备缓冲区页、某些磁盘高速缓存的页（如索引节点高速缓存）——本来就在硬盘中有备份	检查页是否被修改过，如果被修改过则写回硬盘
可丢弃页	*slab* 机制保留的未使用页、文件系统目录项等缓存页	无需操作，可直接丢弃

(3)内存/页框回收

系统内存紧张的时候，就会进行回收内存（页框）的工作，主要有两类内存可以被回收：

类型	定义	回收方式	性能影响
文件页	内核缓存的磁盘数据（*Buffer*）和文件数据（*Cache*）。包含从磁盘读取的数据或待写入的数据。	●**干净页**：直接释放内存。 ●**脏页**：先写回磁盘后再释放内存。	●**干净页**：不会影响性能。 ●**脏页**：写回磁盘会发生磁盘 I/O，影响性能。
匿名页	没有实际载体的内存，如堆和栈数据。没有直接的文件或磁盘载体。	通过 ***Swap* 机制**：将不常访问的匿名页写到磁盘中，释放内存。再次访问时从磁盘读入内存。	影响系统性能（*Swap* 操作涉及磁盘读写）

(4)页框回收对比页面回收

区别点	页框回收	页面回收
作用对象	物理内存中的页框（*Page Frame*）	虚拟内存中的页面（*Virtual Page*）
回收内容	物理页框，主要是为了释放物理内存	虚拟页面，可能会涉及将页面交换到磁盘
回收机制	直接释放干净页，写回脏页，或者将匿名页交换到 *Swap* 中	通过虚拟内存管理策略（如 LRU、时钟算法）决定页面的回收与置换
目的	提供更多物理内存用于新进程或其他内核需求	管理虚拟内存，将不活跃的页面移出物理内存，腾出空间
典型场景	系统内存紧张时，内核通过 *kswapd* 等机制执行页框回收	虚拟内存管理系统自动管理页面回收与置换，响应内存压力

(四)进程间通信：信号

1.信号（*Signal*）的概念

- 信号是一种异步、非阻塞的通信机制，进程无需等待信号的到达即可接收。
- 信号也称为“用户态中断”，用于异步通知进程某个事件的发生。每个信号都有唯一的编号和对应的处理动作。当某个信号发生时，内核向相应的进程发送信号，并触发预设的信号处理函数或执行默认操作。
- 信号通常用于简单的事件通知，如终止、暂停进程，但不适合复杂的数据传输。
- 信号可以用于用户态进程与内核进程之间的交互，内核进程也可以通过信号通知用户进程某些系统事件。

2.信号的来源

(1)硬件来源

①**硬件信号触发**：某些硬件操作可能触发特定信号。如，SIGFPE 信号（浮点除零操作）。
②**硬件异常**：如内存访问越界（如段错误，SIGSEGV）或非法指令。

(2)软件来源

①**系统调用**：通过系统提供的函数，如 $kill()$、$alarm()$，显式地向进程发送信号。
②**用户输入**：用户通过终端输入特定字符组合（如 Ctrl+C），会向进程发送 SIGINT 信号。
③**子进程状态变化**：子进程结束或停止时，父进程会收到 SIGCHLD 信号，监控子进程状态。
④**定时器**：通过 $alarm()$ 或 $setitimer()$ 设置的定时器到期时，向进程发送 SIGALRM 信号。

3.常见的信号

信号名	含义	默认操作
SIGINT	该信号在用户键入 INTR 字符时发出，终端驱动程序发送此信号并送到前台进程中的每一个进程。	终止
SIGILL	该信号在一个进程企图执行一条非法指令时发出。	终止
SIGKILL	该信号用来立即结束程序的运行，并且不能被阻塞、处理和忽略。	终止
SIGALRM	该信号当一个定时器到时的时候发出。	终止
SIGSTOP	该信号用于暂停一个进程，且不能被阻塞、处理或忽略。	暂停进程
SIGCHLD	子进程改变状态时，父进程会收到这个信号	忽略
SIGABORT	调用 *abort* 函数生成的信号。该信号用于结束进程	终止

4.信号传递和响应

(1)信号产生

参考“信号的来源”。

(2)信号传递

信号产生后，如果信号没有被阻塞或忽略，内核会将信号传递给目标进程（正在等待信号或正在执行的进程）。

(3)信号处理

一旦进程接收到信号，内核会检查信号的处理方式。
①**忽略信号**：进程可忽略特定信号，但 SIGKILL 和 SIGSTOP 信号是不可忽略的。
②**捕捉信号**：进程可注册自定义的信号处理函数，当信号发生时，会执行这个处理函数。
③**执行默认操作**：若没有注册自定义处理函数，会执行默认行为（如终止或暂停进程）。

(4)信号处理过程

如果信号处理方式是自定义处理函数，进程将调用该函数来处理信号。
处理函数可以执行一系列操作，如修改全局变量、执行清理操作等。
在处理函数执行期间，进程可能会被阻塞在函数中。

(5)信号返回

信号处理函数执行完毕后，程序从处理函数中返回，恢复执行信号发生前的代码。

5.IPC 机制对比

IPC 机制	信号（*Signal*）	共享内存	管道（*Pipe*）	消息传递	信号量
类型	异步通知	数据传输	数据传输	数据传输	同步机制
数据传输能力	仅用于通知，无数据传输	高效数据传输，需额外同步机制	单向数据传输	支持数据传输	不直接传输数据，用于同步和互斥
同步机制	无同步能力	需额外同步	无同步能力	提供同步机制	强同步能力
主要用途	异常处理、控制	大量数据共享	进程间通信	跨进程通信	资源控制、同步
优缺点	优: 简单易用; 缺: 无数据传输	优: 适合大数据量; 缺: 需同步	优: 实现简单; 缺:单向传输，缓冲区有限	优: 可跨进程通信; 缺: 开销较大	优: 强同步和互斥能力; 缺: 需要额外管理
