第三章　I/O系统

翁楚良

https://chuliangweng.github.io

2023 春　ECNU

第三章 I/O系统 提纲

- 3.1 I/O硬件原理
- 3.2 I/O软件原理
- 3.3 死锁
- 3.4 MINIX3 I/O概述
- 3.5 MINIX3 块设备
- 3.6 RAM盘
- 3.7 磁盘
- 3.8 终端

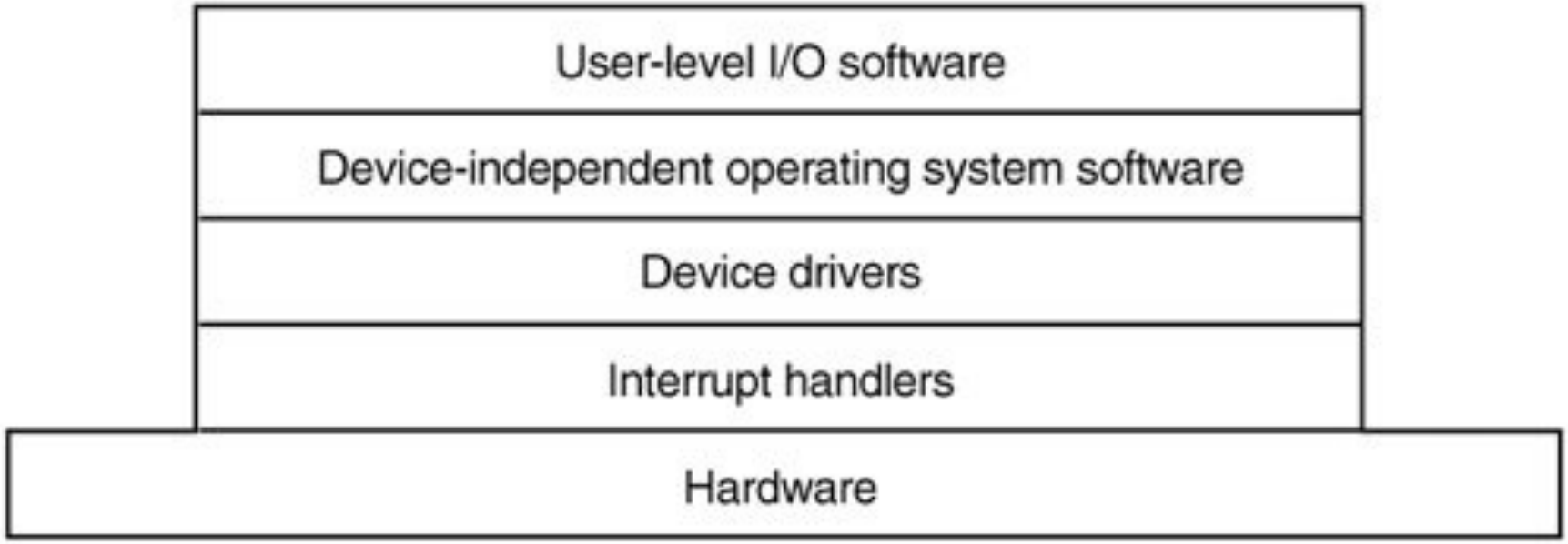
I/O软件原理

- I/O软件目标
- 中断处理器
- 设备驱动程序
- 设备无关I/O软件
- 用户空间I/O软件

I/O软件目标

- 设备无关性
  - 程序员写出的软件无需修改便能读出软盘、硬盘以及CD-ROM等不同设备上的文件
- 统一的命名
  - 一个文件或设备名将简单地只是一个字符串或一个整数，而完全不依赖于设备。
  - 在UNIX和MINIX 3中，所有的磁盘可以以任何方式集成到文件系统层次结构中去，用户也不必知道哪个各字对应着哪个设备。
- 容错功能
  - 错误应在尽可能接近硬件的地方处理，低层软件可以自行处理错误，尽可能向上层软件透明
- 协同同步与异步传输
  - 多数物理I/O是异步传输，用户接口是阻塞的，需要使之协同
- 设备共享
  - 某些设备可同时被多个用户使用，另一些设备则在某一时刻只能供一个用户专用

I/O软件系统的层次结构



I/O软件原理

- I/O软件目标
- 中断处理器
- 设备驱动程序
- 设备无关I/O软件
- 用户空间I/O软件

中断处理器

- 中断需要尽量加以屏蔽，需将其放在操作系统的底层进行处理，以便其余部分尽可能少地与之发生联系。
  - 屏蔽中断的最好方法是将每一个进行I/O操作的进程挂起，直至I/O操作结束并发生中断。
  - 进程自己阻塞的方法有：执行信号量的DOWN操作、条件变量的WAIT操作；或接收一条消息等等。
- 当中断发生时，中断处理程序执行相应的操作，然后解除相应进程的阻塞状态。
  - 一些系统中是执行信号量的UP操作，在管程中可能是对一个条件变量执行SIGNAL操作，另一些系统则可能是向阻塞的进程发一条消息，总之其作用是将刚才被阻塞的进程恢复执行。

I/O软件原理

- I/O软件目标
- 中断处理器
- 设备驱动程序
- 设备无关I/O软件
- 用户空间I/O软件

设备驱动程序

- 设备驱动程序中包括了所有与设备相关的代码。每个设备驱动程序只处理一种设备，或者一类紧密相关的设备
- 设备驱动程序的功能是从与设备无关I/O软件中接收抽象的请求，并负责执行该请求。
- 设备驱动程序放在系统内核中，可以获得良好的性能，但会影响系统的可靠性；MINIX3将其作为用户模式的进程，以提高其可靠性

驱动程序的工作流程

- 执行一条I/O请求的第一步，是将它转换为更具体的形式。
  - 例如对磁盘驱动程序，它包含：计算出所请求块的物理地址、检查驱动器电机是否在运转、检测磁头臂是否定位在正确的柱面等等。
  - 简言之，它必须确定需要哪些控制器命令以及命令的执行次序。
- 一旦决定应向控制器发送什么命令，驱动程序将向控制器的设备寄存器中写入这些命令。
  - 某些控制器一次只能处理一条命令，另一些则可以接收一串命令并自动进行处理。
- 这些控制命令发出后，存在两种情况
  - 驱动程序需等待控制器完成一些操作，所以驱动程序阻塞，直到中断信号到达才解除阻塞。
  - 另一种情况是操作没有任何延迟，所以驱动程序无需阻塞。
    - 例如：在有些终端上滚动屏幕只需往控制器寄存器中写入几个字节，无需任何机械操作，所以整个操作可在几微秒内完成。

I/O软件原理

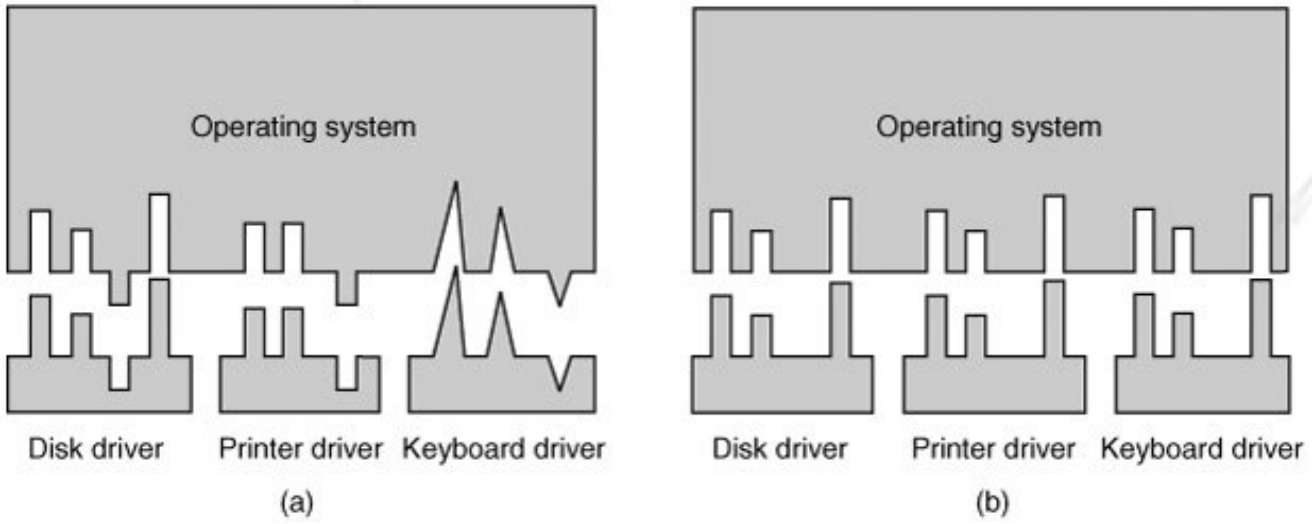
- I/O软件目标
- 中断处理器
- 设备驱动程序
- 设备无关I/O软件
- 用户空间I/O软件

设备无关I/O软件

- 设备无关软件的基本功能是执行适用于所有设备的常用I/O功能，并向用户层软件提供一个一致的接口。
  - 设备无关软件和设备驱动程序之间的精确界限在各个系统都不尽相同。对于一些以设备无关方式完成的功能，在实际中由于考虑到执行效率等因素，也可以考虑由驱动程序完成。
- 设备无关I/O软件的功能包括
  - 设备驱动程序的统一接口
  - 缓冲
  - 错误报告
  - 分配和释放专用设备
  - 提供与设备无关的块大小

设备驱动程序的统一接口

- I/O设备和驱动程序的对外接口需要尽可能的相同
- 在标准接口情况下，可以方便地加入新的驱动程序



设备驱动程序的统一接口

- 设备无关I/O软件负责将设备名映射到相应的驱动程序
  - 在UNIX中，一个设备名，如/dec/tty00 唯一地确定了一个i-节点，其中包含了主设备号（major device number），通过主设备号就可以找到相应的设备驱动程序。i-节点也包含了次设备号（minor device number），它作为传给驱动程序的参数指定具体的物理设备。
- 设备作为命名对象出现在文件系统中，通过对文件的常规保护规则，确保对设备的访问权限。

缓冲

- 对于块设备，硬件每次读写均以块为单元，而用户程序则可以读写任意大小的单元。如果用户进程写半个块，操作系统将在内部保留这些数据，直到其余数据到齐后才一次性地将这些数据写到盘上。
- 对字符设备，用户向系统写数据的速度可能比向设备输出的速度快，所以需要进行缓冲。超前的键盘输入同样也需要缓冲。

错误报告

- 错误处理多数由驱动程序完成，多数错误是与设备紧密相关的，因此只有驱动程序知道应如何处理（如重试、忽略、严重错误）。
  - 一种典型错误是磁盘块受损导致不能读写。驱动程序在尝试若干次读操作不成功后将放弃，并向设备无关软件报错。从此处往后错误处理就与设备无关了。
    - 如果在读一个用户文件时出错，则向调用者报错即可。
    - 如果是在读一些关键系统数据结构时出错，比如磁盘使用状况位图，则操作系统只能打印出错信息，并终止运行。

分配和释放专用设备

- 一些设备，如CD-ROM记录器，在同一时刻只能由一个进程使用。
- 这要求操作系统检查对该设备的使用请求，并根据设备的忙闲状况来决定是接受或拒绝此请求。
  - 一种简单的处理方法是通过直接用OPEN打开相应的设备文件来进行申请。若设备不可用，则OPEN失败。关闭独占设备的同时将释放该设备。

提供与设备无关的块大小

- 不同磁盘的扇区大小可能不同，设备无关软件屏蔽了这一事实并向高层软件提供统一的数据块大小
  - 如将若干扇区作为一个逻辑块，这样上层软件只和逻辑块大小相同的抽象设备交互，而不管物理扇区的大小。
  - 如有些字符设备对字节进行操作(Modem)，另一些则使用比字节大一些的单元（网卡），这类差别也可以进行屏蔽。

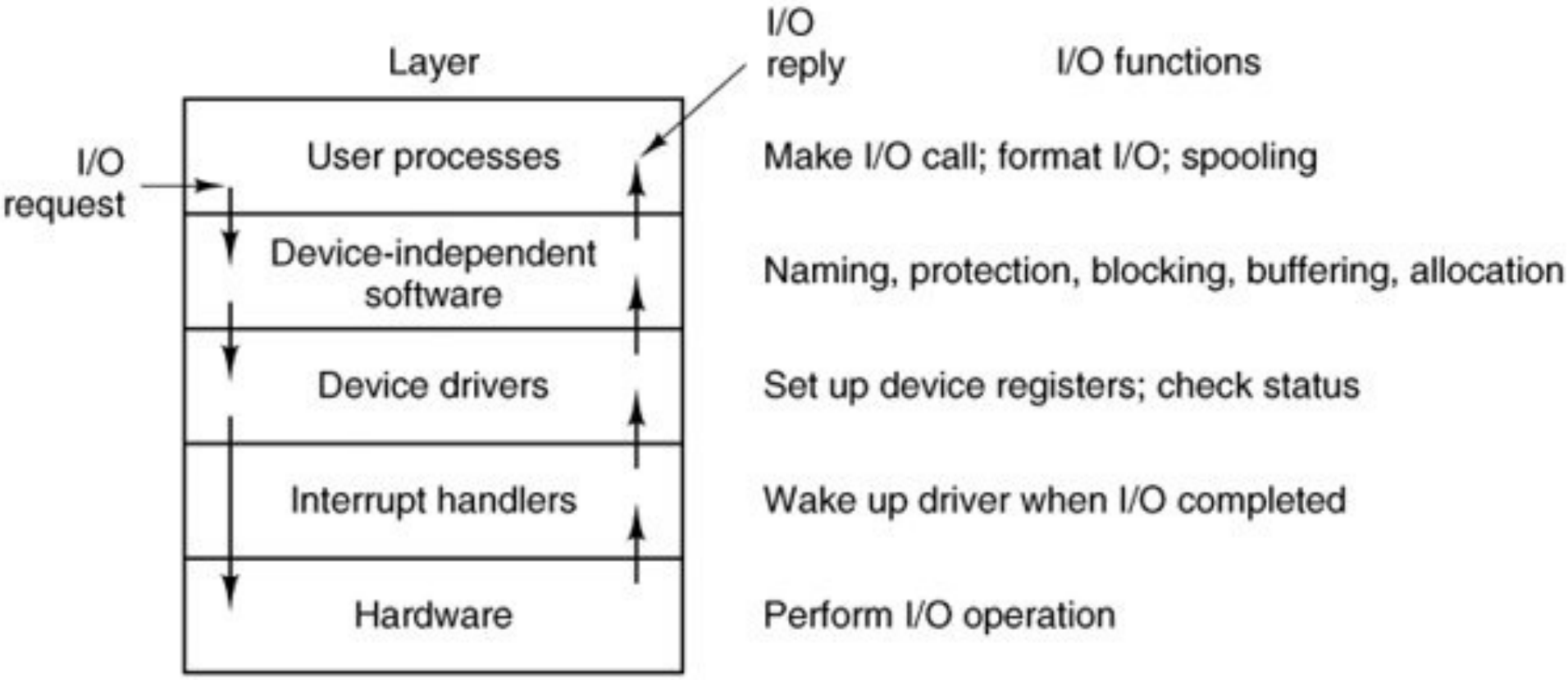
I/O软件原理

- I/O软件目标
- 中断处理器
- 设备驱动程序
- 设备无关I/O软件
- 用户空间I/O软件

用户层I/O软件

- I/O相关的库例程
  - 主要工作是提供参数给相应的系统调用并调用之
- 假脱机系统(spooling)
  - Spooling是在多道程序系统中处理专用I/O设备的一种方法
    - 创建一个特殊的守护进程daemon以及一个特殊的目录，称为 Spooling 目录。
    - 打印一个文件之前，进程首先产生完整的待打印文件并将其放在Spooling目录下。
    - 由该守护进程进行打印，只有该守护进程能够使用打印机设备文件。

I/O系统的层次结构及功能



第三章 I/O系统 提纲

- 3.1 I/O硬件原理
- 3.2 I/O软件原理
- 3.3 死锁
- 3.4 MINIX3 I/O概述
- 3.5 MINIX3 块设备
- 3.6 RAM盘
- 3.7 磁盘
- 3.8 终端

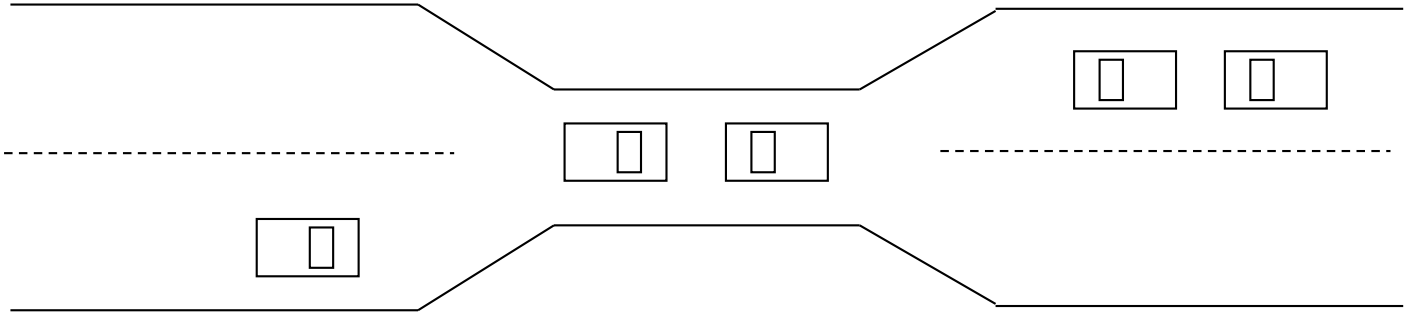
概述

- 死锁是指系统中多个进程无限制地等待永远不会发生的条件
- Example
  - System has 2 tape drives.
  - P0 and P1 each hold 1 tape drive and each needs another one.
- Example
  - semaphores *A* and *B*, initialized to 1

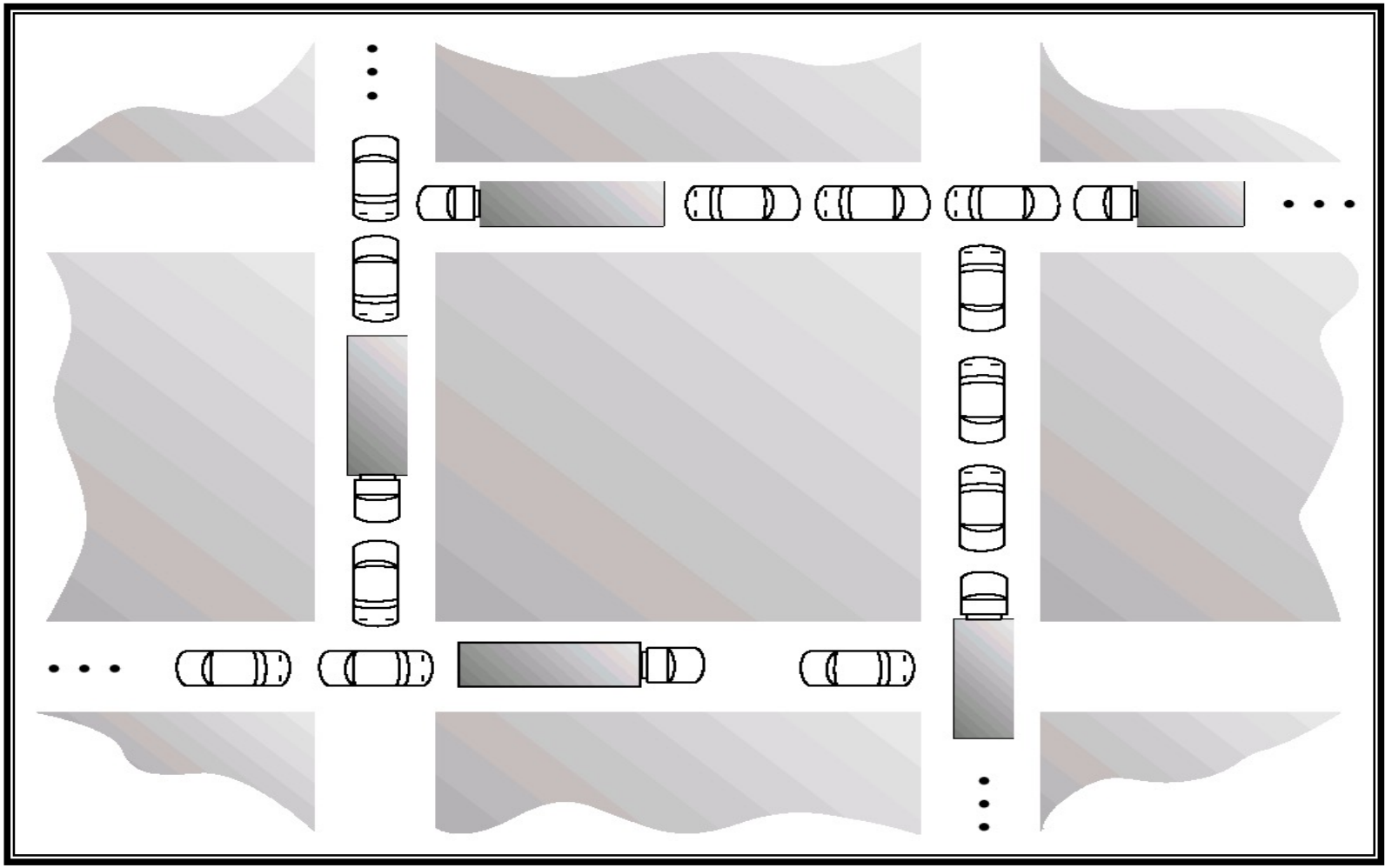
P_0	P_1
wait (A);	*wait(B)*
wait (B);	*wait(A)*

Bridge Crossing Example

- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible.



Traffic Gridlock/Deadlock

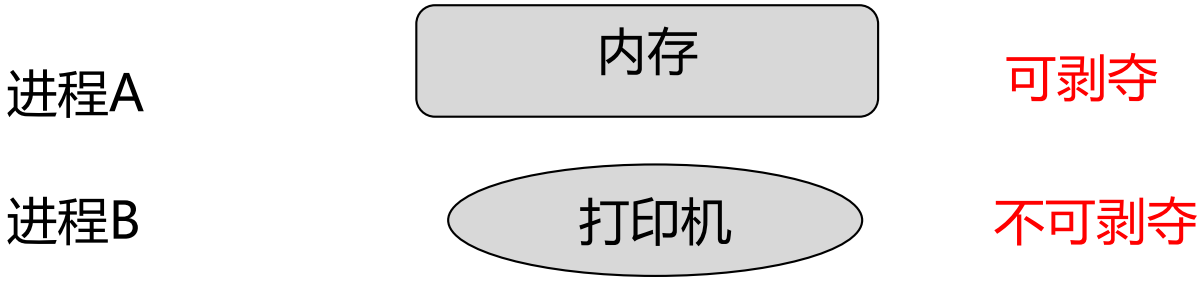


死锁

- 资源
- 死锁原理
- 鸵鸟算法
- 死锁检测与恢复
- 死锁预防
- 避免死锁

资源

- 进程对设备、文件等获得独占性的访问权时有可能会发生死锁，为了尽可能地通用化，将这种需排它使用的对象称为资源。
- 资源有两类：可剥夺式和不可剥夺式
  - 可剥夺式资源可从拥有它的进程处剥夺而没有任何副作用，存储器是一类可剥夺资源。
  - 不可剥夺资源无法在不导致相关计算失败的情况下将其从属主进程处剥夺



死锁

- 资源
- 死锁原理
- 鸵鸟算法
- 死锁检测与恢复
- 死锁预防
- 避免死锁

System Model

- Resource types $R_1, R_2, \ldots, R_m$

  CPU cycles, memory space, I/O devices

- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
  - request
  - use
  - release
- deadlock: A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause

死锁的必要条件

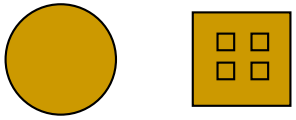
- 死锁发生的必要条件
  - 互斥：任一时刻只允许一个进程使用资源
  - 请求和保持：进程在请求其余资源时，不主动释放已经占用的资源
  - 非剥夺：进程已经占用的资源，不会被强制剥夺
  - 环路等待：环路中的每一条边是进程在请求另一进程已经占有的资源。

Resource-Allocation Graph (1)

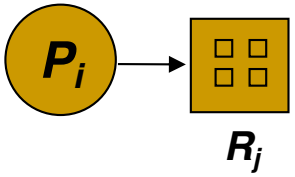
- A set of vertices V and a set of edges E.
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system.
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system.
- request edge – directed edge $P_i \rightarrow R_j$
- assignment edge – directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph (2)

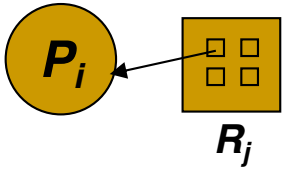
- Process



Resource Type with 4 instances

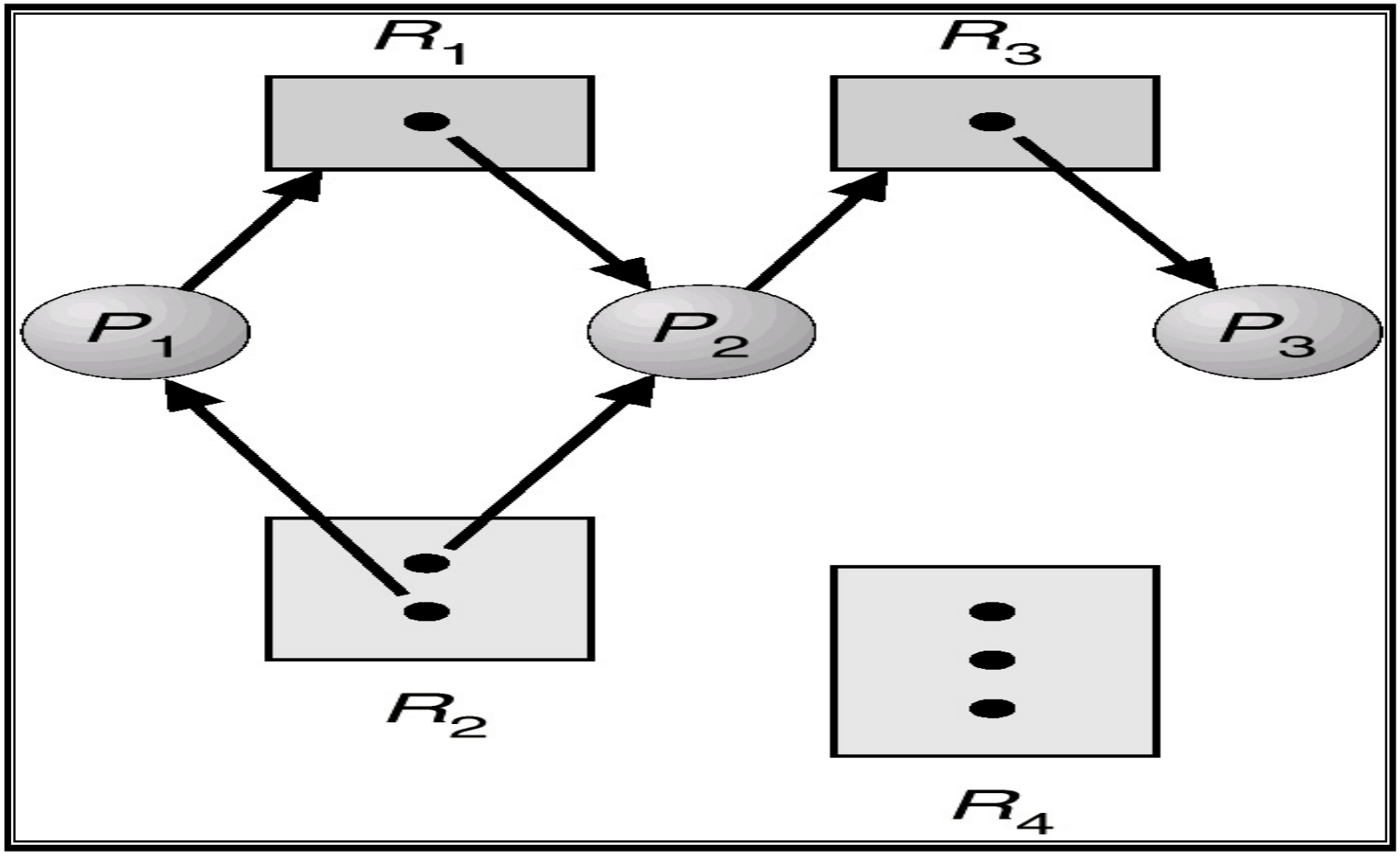


- P_i requests instance of R_j

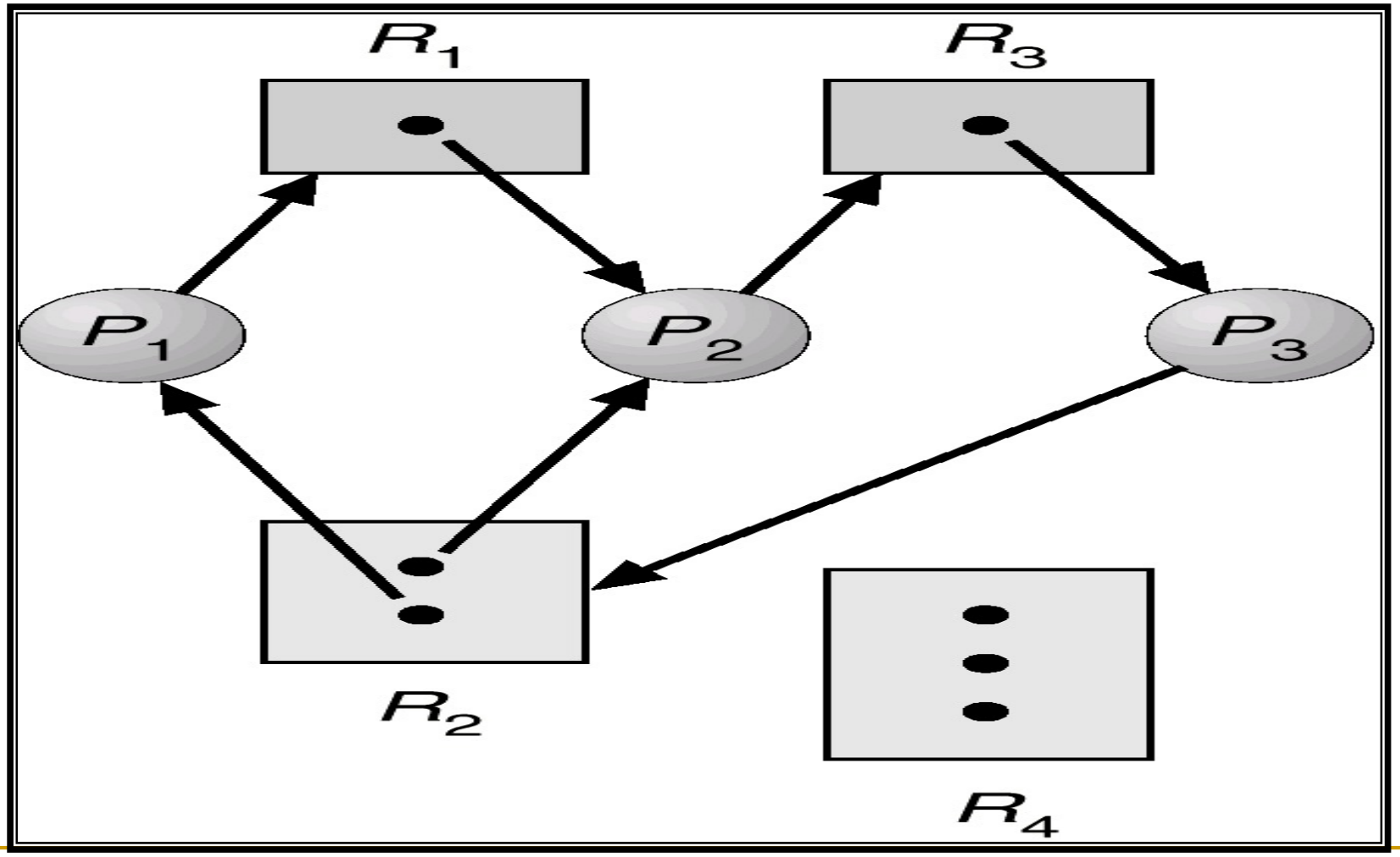


- P_i is holding an instance of R_j

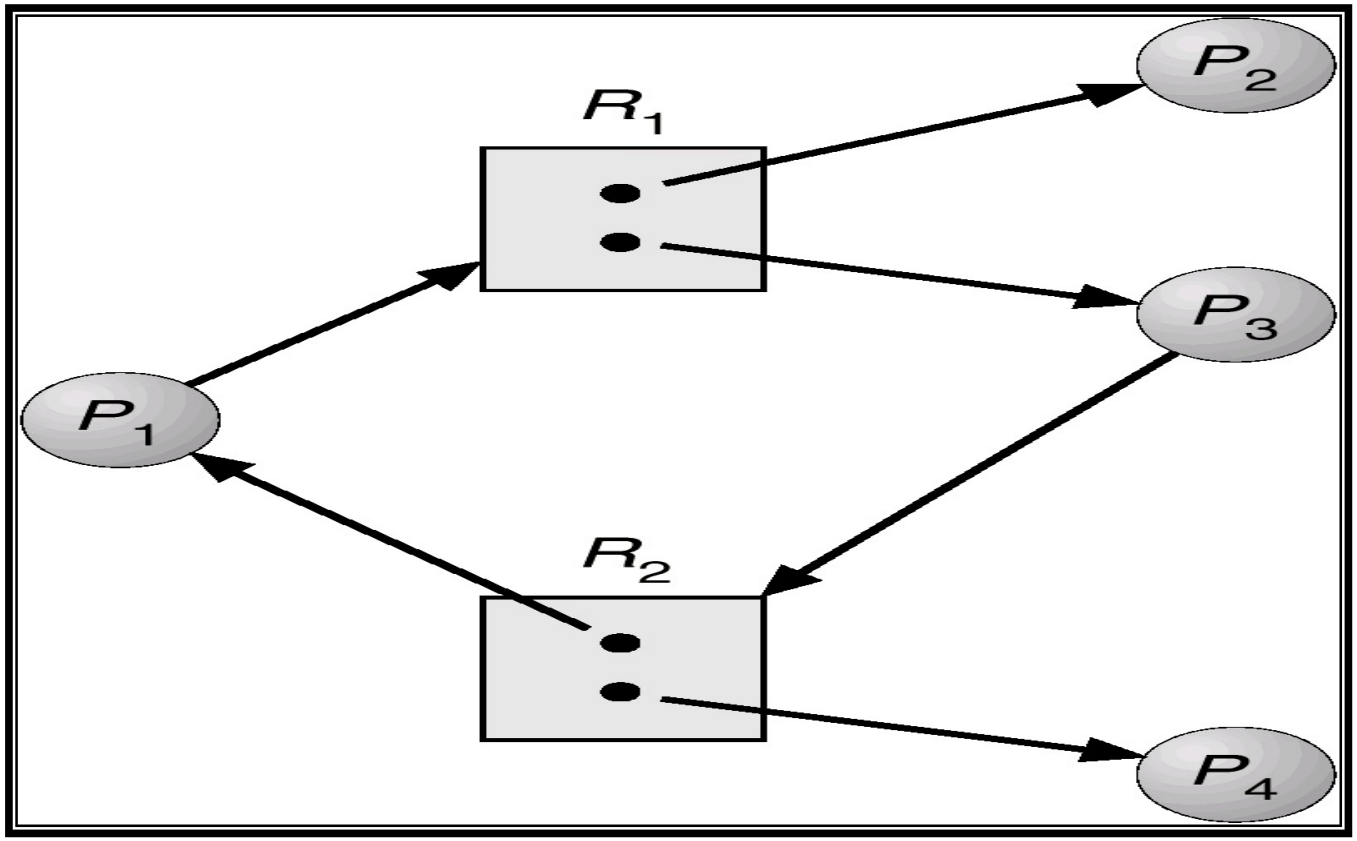
Example of a Resource Allocation Graph



Resource Allocation Graph With A Deadlock



Resource Allocation Graph with a cycle but no deadlock



Basic Facts

- If graph contains no cycles ⇒ no deadlock.
- If graph contains a cycle ⇒
  - if only one instance per resource type, then deadlock.
  - if several instances per resource type, possibility of deadlock.

Methods for Handling Deadlocks

- Ensure that the system will *never* enter a deadlock state.
- Allow the system to enter a deadlock state and then recover.
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

处理死锁的有效方法

方法	资源分配策略	各种可能模式	主要优点	主要缺点
预防 Prevention	保守的；宁可资源闲置（从机制上使死锁条件不成立）	一次请求所有资源<条件 2>	适用于作突发式处理的进程；不必剥夺	效率低；进程初始化时间延长 剥夺次数过多；多次对资源重新起动 不便灵活申请新资源
		资源剥夺<条件 3>	适用于状态可以保存和恢复的资源	
		资源按序申请<条件 4>	可以在编译时（而不必在运行时）就进行检查	
避免 Avoidance	是“预防”和“检测”的折衷（在运行时判断是否可能死锁）	寻找可能的安全的运行顺序	不必进行剥夺	使用条件：必须知道将来的资源需求；进程可能会长时间阻塞
检测 Detection	宽松的；只要允许，就分配资源	定期检查死锁是否已经发生	不延长进程初始化时间；允许对死锁进行现场处理	通过剥夺解除死锁，造成损失