Chapter 8

代码可重入

* 可重入的定义

简单定义:"可以正确重复使用"，有两个关键：1，可以重复使用；2，并能正确使用。意味着在多次执行的时候能得到正确的值，并不受其他调用的影响。
官方定义：若一个程序或子程序可以“在任意时刻被中断然后操作系统调度执行另外一段代码，这段代码又调用了该子程序不会出错”，则称其为可重入（reentrant或re-entrant）的。即当该子程序正在运行时，执行线程可以再次进入并执行它，仍然获得符合设计时预期的结果。与多线程并发执行的线程安全不同，可重入强调对单个线程执行时重新进入同一个子程序仍然是安全的。
这里也有一段比较好的英文阐释：
A computer program or routine is described as reentrant if it can be safely called again before its previous invocation has been completed (i.e it can be safely executed concurrently)
可重入函数主要用于多任务环境中，一个可重入的函数简单来说就是可以被中断的函数，也就是说，可以在这个函数执行的任何时刻中断它，转入OS调度下去执行另外一段代码，而返回控制时不会出现什么错误；而不可重入的函数由于使用了一些系统资源，比如全局变量区，中断向量表等，所以它如果被中断的话，可能会出现问题，这类函数是不能运行在多任务环境下的。

* 产生背景

可重入概念是在单线程操作系统的时代提出的。一个子程序的重入，可能由于自身原因，如执行了jmp或者call，类似于子程序的递归调用；或者由于操作系统的中断响应。UNIX系统的signal的处理，即子程序被中断处理程序或者signal处理程序调用。所以，可重入也可称作“异步信号安全”。这里的异步是指信号中断可发生在任意时刻。 重入的子程序，按照后进先出线性序依次执行。

* 编写可重入代码注意的条件

若一个函数是可重入的，则该函数应当满足下述条件：

不能含有静态（全局）非常量数据。
不能返回静态（全局）非常量数据的地址。
只能处理由调用者提供的数据。
不能依赖于单实例模式资源的锁。
调用(call)的函数也必需是可重入的。
上述条件就是要求可重入函数使用的所有变量都保存在呼叫堆叠的当前函数栈（frame）上，因此同一执行线程重入执行该函数时加载了新的函数帧，与前一次执行该函数时使用的函数帧不冲突、不互相覆盖，从而保证了可重入执行安全。

多“用户/对象/进程优先级”以及多进程（Multiple processes），一般会使得对可重入代码的控制变得复杂。同时，IO代码通常不是可重入的，因为他们依赖于像磁盘这样共享的、单独的（类似编程中的静态、全域）资源。

* 与线程安全的关系

可重入与线程安全两个概念都关系到函数处理资源的方式。但是，他们有重大区别：可重入概念会影响函数的外部接口，而线程安全只关心函数的实现。大多数情况下，要将不可重入函数改为可重入的，需要修改函数接口，使得所有的数据都通过函数的调用者提供。要将非线程安全的函数改为线程安全的，则只需要修改函数的实现部分。一般通过加入同步机制以保护共享的资源，使之不会被几个线程同时访问。

操作系统背景与CPU调度策略：
可重入是在单线程操作系统背景下，重入的函数或者子程序，按照后进先出的线性序依次执行完毕。

多线程执行的函数或子程序，各个线程的执行时机是由操作系统调度，不可预期的，但是该函数的每个执行线程都会不时的获得CPU的时间片，不断向前推进执行进度。可重入函数未必是线程安全的；线程安全函数未必是可重入的。

例如，一个函数打开某个文件并读入数据。这个函数是可重入的，因为它的多个实例同时执行不会造成冲突；但它不是线程安全的，因为在它读入文件时可能有别的线程正在修改该文件，为了线程安全必须对文件加“同步锁”。
另一个例子，函数在它的函数体内部访问共享资源使用了加锁、解锁操作，所以它是线程安全的，但是却不可重入。因为若该函数一个实例运行到已经执行加锁但未执行解锁时被停下来，系统又启动该函数的另外一个实例，则新的实例在加锁处将转入等待。如果该函数是一个中断处理服务，在中断处理时又发生新的中断将导致资源死锁。`fprintf`函数就是线程安全但不可重入。
* 可重入锁

可重入锁也叫递归锁，它俩等同于一回事，指的是同一线程外层函数获得锁之后，内层递归函数仍然能获得该锁的代码，同一线程在外层方法获取锁的时候，再进入内层方法会自动获取锁。也就是说，线程可以进入任何一个它已经拥有的锁所同步着的代码块。`ReentrantLock` 和 `synchronized` 就是典型的可重入锁！

8.1 Name two differences between logical and physical addresses.

逻辑地址和物理地址的基本区别是，逻辑地址是由CPU从正在运行的程序的角度生成的。另一方面，物理地址是内存单元中存在的位置，可以物理访问。

CPU为程序生成的所有逻辑地址集称为逻辑地址空间。然而，映射到相应逻辑地址的所有物理地址的集合称为物理地址空间。编译时和加载时地址绑定方法生成相同的逻辑地址和物理地址，然而执行时的地址绑定方案生成不同的逻辑地址和物理地址

8.4 Consider a logical address space of 64 pages of 1,024 words each, mapped onto a physical memory of 32 frames.

1. **How many bits are there in the logical address?**
2. **How many bits are there in the physical address?**

    1.16 bits

    2.15 bits

8.5 What is the effect of allowing two entries in a page table to point to the same page frame in memory? Explain how this effect could be used to decrease the amount of time needed to copy a large amount of memory from one place to another. What effect would updating some byte on the one page have on the other page?

通过允许页面表中的两个条目指向内存中的同一页面帧，用户可以共享代码和数据。如果代码重新输入，可以通过共享文本编辑器、编译器和数据库系统等大型程序来节省大量内存空间。通过让不同的页面表指向相同的内存位置，可以“复制”大量内存。

然而，共享非重新输入代码或数据意味着任何有权访问该代码的用户都可以对其进行修改，这些修改将反映在其他用户的“副本”中。因此，应该使用“写时复制”。

8.6 Describe a mechanism by which one segment could belong to the address space of two different processes.

由于段表是基址寄存器的集合，当两个不同进程的段表中的条目指向同一个物理位置时，段可以被共享。这两个段表必须有相同的基址指针，而且两个进程中的共享段号必须相同。

8.7 Sharing segments among processes without requiring that they have the same segment number is possible in a dynamically linked segmentation system.

1. **Define a system that allows static linking and sharing of segments without requiring that the segment numbers be the same.**
2. **Describe a paging scheme that allows pages to be shared without requiring that the page numbers be the same.**

   1.可以通过共享页来实现，不同的段指向相同的页。

   2.不同页映射到相同的帧。

8.9 Explain the difference between internal and external fragmentation.

分配给进程固定大小的内存块时，如果分配的内存比进程需要内存大，就会产生内部碎片，这种碎片是被进程占用但是不被进程使用的区域，在这段空间释放之前，操作系统无法使用这段空间。

当可变大小的内存空间被分配给进程时，就会产生外部碎片，将空闲内存空间分为小的片段，当总的可用内存之和可以满足请求但是并不连续时，就会产生外部碎片问题。

8.12 Most systems allow a program to allocate more memory to its address space during execution. Allocation of data in the heap segments of programs is an example of such allocated memory. What is required to support dynamic memory allocation in the following schemes?

1. **Contiguous memory allocation**
2. **Pure segmentation**
3. **Pure paging**

   1.程序全部重新分配

   2.分配拓展段

   3.分配新分页

8.13 Compare the memory organization schemes of contiguous memory allocation, pure segmentation, and pure paging with respect to the following issues:

a. External fragmentation

b. Internal fragmentation

c. Ability to share code across processes

1. 连续内存分配：有；纯分段：有；纯分页：无
2. 连续内存分配：无；纯分段：无；纯分页：有
3. 连续内存分配：不行；纯分段：可以；纯分页：可以

8.14 On a system with paging, a process cannot access memory that it does not own. Why? How could the operating system allow access to other memory? Why should it or should it not?

分页系统上的一个地址是一个逻辑页号和一个偏移量。物理页是通过搜索一个基于逻辑页号的表来产生一个物理页号的。因为操作系统控制着这个表的内容，它可以限制一个进程只访问分配给该进程的物理页。一个进程不可能引用它不拥有的页面，因为该页面不在页表中。为了允许这种访问，操作系统只需要允许非进程内存的条目被添加到进程的页表中。当两个或更多的进程需要交换数据时，这是非常有用的，它们只是对相同的物理地址（可能在不同的逻辑地址）进行读写。这使得进程间的通信非常有效。

8.15 Explain why mobile operating systems such as iOS and Android do not support swapping.

闪存容量小，写入次数有限制，闪存与内存之间的吞吐量差。

8.18 Explain why address space identifiers (ASIDs) are used.

ASIDs在TLB中提供了地址空间保护，并支持TLB条目同时用于几个不同的进程。

如果TLB不支持独立的ASID，那么每次选择新的页表（上下文切换）时，必须刷新TLB以确保下一个执行进程不会使用错误的翻译信息。

8.19 Program binaries in many systems are typically structured as follows. Code is stored starting with a small, fixed virtual address, such as 0. The code segment is followed by the data segment that is used for storing the program variables. When the program starts executing, the stack is allocated at the other end of the virtual address space and is allowed to grow toward lower virtual addresses. What is the significance of this structure for the following schemes?

1. **Contiguous memory allocation**
2. **Pure segmentation**
3. **Pure paging**

   1.固定大小防止重新分配内存

   2.方便拓展

   3.方便新分配

8.20 Assuming a 1-KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers):

a. 3085

b. 42095

c. 215201

d. 650000

e. 2000001

Logical address (decimal)	Page # (decimal)	Offset (decimal)
3085	3	13
42095	41	111
215201	210	161
650000	634	784
2000001	1953	129

8.21 The BTV operating system has a 21-bit virtual address, yet on certain embedded devices, it has only a 16-bit physical address. It also has a 2-KB page size. How many entries are there in each of the following?

1. **A conventional, single-level page table**
2. **An inverted page table**
    1. `2^21/(2*1024)=1024`
    2. `2^16/(2*1024)=32`

8.23 Consider a logical address space of 256 pages with a 4-KB page size, mapped onto a physical memory of 64 frames.

1. **How many bits are required in the logical address?**
2. **How many bits are required in the physical address?**
    1. `12 + 8 = 20 bits`
    2. `12 + 6 = 18 bits`

8.28 Consider the following segment table:

What are the physical addresses for the following logical addresses?

a. 0,430

b. 1,10

c. 2,500

d. 3,400

e. 4,112

1. `219+430=649`
2. `2300+10=2310`
3. 非法
4. `1327+400=1727`
5. 非法

8.29 What is the purpose of paging the page tables?

页表本身可能会很大，以至于通过分页来分页表，可以简化内存分配问题（通过确保一切被分配为固定大小的页面，而不是可变大小的块），并且还启用交换当前未使用的页表部分。与之相关的缺点是，地址转换可能需要更多的内存访问。