191220154 张涵之 第一章作业

* What are the two main functions of an operating system?
- 向下（对硬件）：管理资源。在裸机上直接和硬件交互，负责管理、控制、扩充和改造机器硬件，资源的调度与分配、信息的存取与保护、并发活动的协调与控制等工作；
- 向上（对软件）：提供接口。和上层的支撑和应用软件交互，把它们与硬件隔离，为程序员提供编程接口、功能支撑、运行环境，使计算机系统完整、可用和高效。

* On early computers, every byte of data read or written was handled by the CPU (i.e., there was no DMA). What implications does this have for multiprogramming?
- 多道程序设计的指允许多个程序同时进入一个计算机系统的主存储器并启动进行计算的方法，其目的是分析程序运行时的特征，提高 CPU 的利用率，从而充分发挥计算机硬部件的并行性。如果没有 DMA，全部读写操作均由 CPU 完成，则进行 I/O 操作时 CPU 是完全被占用的，多道程序设计将无法实现：无论程序执行多少 I/O 操作，CPU 都是 100%忙碌的。

* What is the difference between kernel and user mode? Explain how having two distinct modes aids in designing an operating system.
- 当处理器处于内核态时，操作系统管理程序运行，处理器运行可信系统软件，全部机器指令都可在处理器上执行，程序可访问所有内存单元和系统资源，可改变处理器状态；
- 当处理器处于用户态时，正在运行非可信应用程序，此时无法执行特权指令，并且访问仅限于当前处理器上执行程序所在的地址空间；
- 这样就能防止操作系统程序受到应用程序的侵害。

* What is a trap instruction? Explain its use in operating systems.
- 陷阱也称自陷或陷入，执行陷阱指令时，CPU 调出特定程序进行相应处理，处理结束后返回到陷阱指令的下一条指令执行。陷阱的作用是在用户和内核之间提供一个像过程一样的接口，称为系统调用，用户程序利用这个接口可方便地使用操作系统内核提供的一些服务。如 IA-32 处理器中的 int 和 sysenter、MIPS 处理器中的 syscall 等都属于陷阱指令。

* What type of multiplexing (time, space, or both) can be used for sharing the following resources: CPU, memory, disk, network card, printer, keyboard, and display?
- Time multiplexing: CPU, network card, printer and keyboard.
- Space multiplexing: memory and disk.
- Both: display

* To a programmer, a system call looks like any other call to a library procedure. Is it important that a programmer know which library procedures result in system calls? Under what circumstances and why?
- 系统调用和函数调用：
    1. 调用形式和实现方式不同
    - 系统调用：由功能调用号决定内核服务例程入口地址，在内核态执行
    - 函数调用：转向的地址固定不变、在用户态执行
    2. 被调用代码的位置不同
    - 系统调用：动态调用，服务例程位于操作系统内核中

- 函数调用：静态调用，调用程序和被调用代码处于同一程序

3. 提供方式不同

- 系统调用：由操作系统统一提供
- 函数调用：编程语言提供，取决于语言提供的函数功能

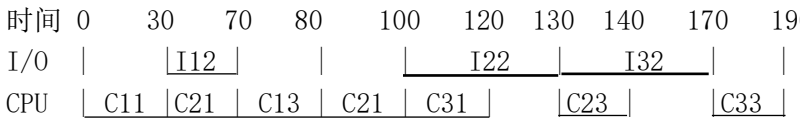
- 综上所述，可见系统调用需在内核态下执行，而函数调用只在用户态下执行。如果程序员想要改变处理器状态，或者执行其他用户态下不支持、无权限的操作，则 TA 需要知道哪些函数是通过系统调用实现的；反之，如果 TA 并不打算改变处理器状态、访问内存单元或系统资源，则 TA 需要特别小心避免执行系统调用，防止造成意料之外的错误。
- 此外，系统调用为动态调用，调用代码较远（位于内核中），且涉及状态的切换，而函数调用是静态调用，调用代码较近（位于同一程序中），不需要切换状态。因此如果其他条件相同（不考虑运行时的操作权限和内存、资源的安全问题），则程序员如果想要提高程序运行的速度和性能，则应该尽可能使用函数调用而避免系统调用。

* Explain how separation of policy and mechanism aids in building microkernel-based operating systems.

- 机制与策略分离原则是让程序中的独立部分分别来实现机制与策略，这有助于内核保持短小和良好结构，这种软件适应性好且易于开发。通过将机制植入操作系统而将策略留给上层软件（如应用进程），即使需要改变策略，系统本身也可以保持不变；即使策略模块必须保留在内核中，也应尽可能与机制隔离，这样策略模块中的变化就不会影响机制模块。

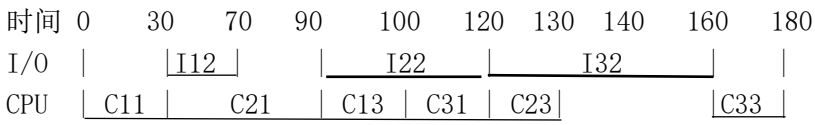
应用题 3

1） 忽略调度执行时间，抢占式多道运行：



花费 190ms，比单道运行节省 260 - 190 = 70ms。

2） 忽略调度执行时间，非抢占式多道运行：



花费 180ms，比单道运行节省 260 - 180 = 80ms。

3）抢占式多道运行，调度时间 1ms，花费 198ms，节省 62ms。

C11：0~30　　C13：73~83　　I12：31~71
C21：32~71，84~105　　C23：137~147　　I22：106~136
C31：107~127　　C33：178~198　　I32：137~177

4）非抢占式多道运行，调度时间 1ms，花费 189ms，节省 71ms。

C11：0~30　　C13：95~105　　I12：31~71
C21：32~71，72~93　　C23：129~139　　I22：94~124
C31：106~124，125~127　　C33：169~199　　I32：128~168

应用题 6

按照抢占式多道运行分析，有：

CPU：0~20 Job3，20~30 Job2，30~40 Job1，40~50 Job2，50~60 Job3，
60~70 空闲，70~80 Job1，80~90 空闲；

I1：0~20 Job2，20~40 空闲，40~70 Job1，70~90 Job3；

I2：0~30 Job1，30~50 空闲，50~90 Job2；

1） Job1 从投入到完成需 90ms，Job2 需 90ms，Job3 需 90ms；

2） Job1 执行过程中 CPU 空闲时段为 60~70，80~90ms，Job1 从投入到完成 CPU 利用率为(90 - 20) / 90 = 77.8%，同理 Job2 为 77.8%，Job3 为 77.8%；

3） 设备 I1 空闲时段为 20~40，利用率为(90 - 20) / 90 = 77.8%，
设备 I2 空闲时段为 30~50，利用率为(90 - 20) / 90 = 77.8%。