**第二章作业补充题**

1. 请回答下列问题。

(1) 为什么在操作系统中要引入线程？内核级线程与用户级线程的区别是什么？

Ans： 首先，在操作系统中引入线程的主要原因是为了提高系统的响应速度和资源利用率。相比于进程，线程的创建、撤销和切换具有更小的时空开销，这使得系统能够更快速地响应用户请求。同时，线程之间可以共享同一进程的资源，从而提高了资源的利用效率，使系统具备更强的可扩展性。

此外，内核级线程与用户级线程之间存在明显的区别。内核级线程的创建、撤销、调度和管理都由操作系统内核负责，每个内核级线程都有一个线程控制块在内核空间进行管理。操作系统可以感知和控制内核级线程的执行。而用户级线程则由用户应用程序创建和管理，完全存在于用户空间，操作系统内核对其一无所知，用户应用程序负责调度和执行这些线程。

(2) Provide two programming examples in which multithreading does not provide better performance than a single-threaded solution.

Ans：a）任何一种顺序程序不适合线程化，比如一个几乎全都是if语句的程序。

b）shell程序也不适合线程化因为它必须密切监视自己的工作的空间。

(3) Under what circumstances does a multithreaded solution using multiple kernel threads provide better performance than a single-threaded solution on a single-processor system?

Ans:多核线程：当一个内核线程出现问题时，可以切换至另一个线程，以便进程可以继续运行，如果是单核就会全部停下，无法继续运行。

(4) Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single-processor system? Explain.

Ans：如果多个用户级线程只能映射到一个内核级线程，那么多线程系统无法充分利用多处理器系统中的不同处理器。操作系统将视其为单个进程，并不会在不同的处理器上同时调度该进程的不同线程。但如果多个用户级线程可以映射到多个内核级线程，那么在多处理器系统上处理该进程的效率将高于单处理器系统。

(5) Consider a multicore system and a multithreaded program written using the many-to-many threading model. Let the number of user-level threads in the program be greater than the number of processing cores in the system. Discuss the performance implications of the following scenarios.

a. The number of kernel threads allocated to the program is less than the number of processing cores.

b. The number of kernel threads allocated to the program is equal to the number of processing cores.

c. The number of kernel threads allocated to the program is greater than the number of processing cores but less than the number of user-level threads.

Ans：a. 如果内核线程数少于处理器数，一些处理器将保持闲置，因为调度器只能将内核线程映射到相应的处理器，这意味着系统未能充分利用所有处理器的性能，尽管相对于单处理器系统而言，性能已有所提升。

b. 当内核线程数量与处理器数量相等时，有可能实现同时利用所有处理器的情况。然而，当内核线程在内核内部遇到阻塞时，相应的处理器将处于空闲状态，这将进一步提高多处理器系统的利用率。

c. 如果内核线程数量多于处理器数量，那么一个被阻塞的内核线程可能会被替换，以支持另一个已准备好执行的内核线程，从而提高多处理器系统的利用率。这种情况下，系统更灵活地分配资源，以优化性能。

2. 请解答下列问题。

(1) 在一个使用多级反馈队列（如课程示例所示）的系统中，一个只使用CPU的进程的执行时间为40秒。如果第一个队列时间片为2，每级时间片增加5个时间单元，那么这个进程运行结束前会被中断多少次，结束时处于哪级队列？

Ans：初始时，进程位于第一队列，具有时间片2和执行时间2。然后进入第二队列，时间片增加至7，累计执行时间变为2+7。接着进入第三队列，时间片增加至12，累计执行时间变为2+7+12。由于只有三个队列，所以在第三级队列轮转执行。继续在第三队列执行，时间片仍为12，累计执行时间变为2+7+12+12。最后仍在第三队列，时间片为12，累计执行时间为2+7+12+12+7。因此，总共发生了5次中断，进程最终结束时仍在第三级队列。这种轮转执行方式确保了任务在不同队列中公平分配时间片，提高了多任务处理的效率。

(2) 某进程调度程序采用基于优先数(priority)的调度策略，即选择优先数最小的进程运行，进程创建时由用户指定一个nice作为静态优先数。为了动态调整优先数，引入运行时间cpuTime和等待时间waitTime，初值均为0。进程处于执行态时，cpuTime定时加1，且waitTime置0；进程处于就绪态时，cpuTime置0，waitTime定时加1。请回答下列问题。

　 (a) 若调度程序只将nice的值作为进程的优先数，即priority=nice，则可能会出现饥饿现象，为什么？

Ans：当进程在运行过程中，不断有新的高优先级进程加入，可能会导致一些低优先级的进程长时间无法获得CPU资源，从而陷入饥饿状态，无法执行。

(b) 使用nice、cpuTime和waitTime设计一种动态优先数计算方法，以避免产生饥饿现象，并说明waitTime的作用。

Ans：使用以下动态优先数计算方法进行计算

priority = nice - waitTime + cpuTime

WaitTime使得长时间等待的进程优先数减小使得优先级更高，从而避免饥饿现象。