CH4：

4.2 Under which circumstances does a multithreaded solution using multiple kernel threads provide better performance than a single-threaded solution on a single-processor system?

When a kernel thread is blocked or performing a lengthy operation, another kernel thread can be switched in to use the interleaving time in a useful manner.

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

No, the kernel thread is not aware of the user-level threads that are created. Therefore, it is not able to run the user-level threads on different processors.

4.13 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.

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

The scheduler can only schedule user level processes to the kernel threads, and since some of the processes are not mapped to the kernel threads, they will be idle, taking more time to finish the processes.

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

The processors will utilize all the processors in the system. However, when a kernel tread blocked by a system call, the corresponding processor would remain idle, lower system usage.

1. The number of kernel threads allocated to the program is greater than the number of processing cores.

All of the processes will be working simultaneously. If a thread gets blocked by a system call, the scheduler may be swapped to another thread that is not blocked, maximizing system utilization.

CH5：

5.6 A variation of the round-robin scheduler is the regressive round-robin scheduler.

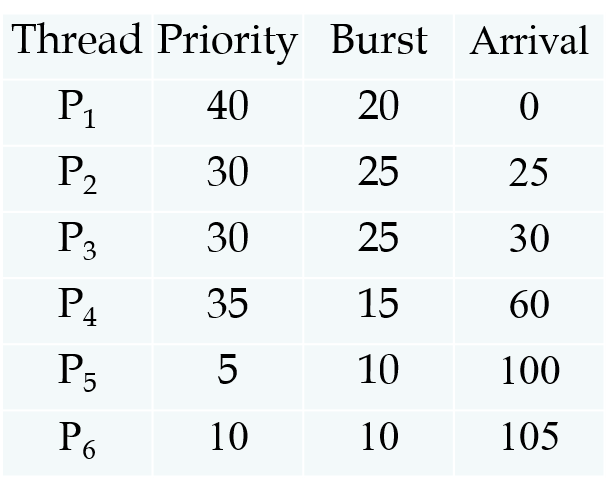
* This scheduler assigns each process a time quantum and a priority.
* The initial value of a time quantum is 50 milliseconds
* However, every time a process has been allocated to the CPU and uses it entire time quantum (does not block for I/O), 10 milliseconds is added to its time quantum, and its priority level is boosted.
* (The time quantum for a process can be increased to a maximum of 100 milliseconds.)
* When a process blocks before using its entire time quantum, its time quantum is reduced by 5 milliseconds, but its priority remains the same

What type of process (CPU-bound, I/O-bound) does the regressive round-robin scheduler favor? Explain the reasons.

This scheduler would favor CPU-bound processes as they are rewarded with a longer time quantum as well as priority boost whenever they consume an entire time quantum. This scheduler does not penalize I/O bound processes as they are likely to block for I/O before consuming their entire time quantum, but their priority remains the same.

5.8 The following processes are being scheduled using a preemptive round-robin scheduling algorithm. Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed below, the system also has an idle task (which consumes no CPU resources and is identified as Pidle). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

(a) Show the scheduling order of the processes using a Gantt chart.  
(b) What is the turnaround time for each process?  
(c) What is the waiting time for each process?  
(d) What is the CPU utilization rate?



a)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | id | P2 | P3 | P2 | P3 | P4 | P2 | P3 | id | P5 | P6 | P5 |

20 25 35 45 55 60 75 80 90 100 105 115 120

b) P1：20-0=20 P2：80-25=55 P3：90-30=60 P4：75-60=15

P5：120-100=20 P6：115-105=10

c) P1：0 P2：30 P3：35 P4：0 P5：10 P6：0

d) (20+25+25+15+10+10)/120=0.875

5.10 Which of the following scheduling algorithms could result in starvation?

* First-come, first-served
* Shortest job first
* Round-robin
* Priority

Please explain the reasons.

Priority and Shortest job first. Since FCFS and RR does not favor any process, they are starvation free. In priority based scheduling, the higher priority process would execute before the lower priority process. Low priority processes may never execute.

5.15 Explain the differences in how much the following scheduling algorithms discriminate in favor of short processes:  
(a) FCFS  
(b) RR  
(c) Multilevel feedback queues

a) Waiting time is large if longer processes come in before the short processes.

b) All processes have equal time burst, so short jobs can leave the system faster once they are finished.

c) Multilevel feedback queues work similarly to RR algorithm. They discriminate favorably toward short jobs.

CH6：

6.4 Explain why implementing synchronization primitives by disabling interrupts is not appropriate in a single-processor system if the synchronization primitives are to be used in user-level programs.

If a user-level program can disable interrupts, then it can disable timer interrupt and prevent context switching from taking place. When critical section is used by a long process, other processes may never enter critical section, so they may never execute.

6.10 The implementation of mutex locks provided in Section 6.5 suffers from busy waiting. Describe what changes would be necessary so that a process waiting to acquire a mutex lock would be blocked and placed into a waiting queue until the lock became available.

To prevent the diminishing efficiency of operating systems, spinlock is introduced. Spinlock enable thread to run one processor and other threads can concurrently run over other processors without getting interrupted.

6.11 Assume that a system has multiple processing cores. For each of the following scenarios, describe which is a better locking mechanism – a spinlock or a mutex lock where waiting processes sleep while waiting for the lock to become available:

* The lock is to be held for a short duration.
* The lock is to be held for a long duration.
* A thread may be put to sleep while holding the lock.

Short duration lock：Spinlock may finish faster in short duration while mutex lock requires suspending and awakening the waiting process.

Long duration lock：Mutex lock allows other processing core to schedule another process while the locked process waits.

Thread may be put to sleep while holding the lock：Mutex lock can prevent waiting process to be spinning while waiting for the other process to wake up.