Quiz 2 Solutions

1. What are the two models of interprocess communication? What are the strengths and weakness of the two approaches?
   Ans.
   The two models for IPC are:
   a. Shared-memory model.
   Strength:
   1. Shared memory communication is faster the message passing model when the processes are on the same machine.
   Weaknesses:
   1. Different processes need to ensure that they are not writing to the same location simultaneously.
   2. Processes that communicate using shared memory need to address problems of memory protection and synchronization.
   b. Message-passing model.
   Strength:
   1. Easier to implement than the shared memory model
   Weakness:
   1. Communication using message passing is slower than shared memory because of the time involved in connection setup.

2. Provide two programming examples in which multithreading does not provide better performance than a single-threaded solution.
   Ans.
   Example 1:

```
string filenames[100];
/* Create threads 1 to 100 */
create_threads();
/* each thread creates a particular file on completion whose name
is passed to the program executed by the thread */
while (1)
{
not_created = FALSE;
for i = 0 .. 99
{
if (not(exists(filename[i])) { not_created = TRUE;}
}
if (not_created == FALSE)
{
printf("Done!");
return;
}
}
```

   This is the pseudo code for a multithreaded program which does not perform any better than a single threaded solution to the same problem.
   Example 2:
   If the computation times of the processes do not differ significantly, multithreading is not very useful in comparison to a single-threaded solution.

3. Describe the action taken by a thread library to context switch between user-level threads.
   Ans.
   To context switch between user-level threads, the thread library saves the context of the old thread in its TCB and loads the saved context of the new thread. The TCB contains the Stack Pointer, Program Counter, Register values and the current state of the thread.

4. Which of the following components of program state are shared across threads in a multithreaded process?
   a. Register values
   b. Heap memory
   c. Global variables
   d. Stack memory

   Ans.
   b. Heap memory and
   d. Global variables.

5. Consider a multiprocessor system and multithreaded program written using the many-to-many threading model. Let the number of user-level threads in the program be more than the number of processors 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 processors.
   Ans.
   Each kernel thread can be run on a separate processor simultaneously along with the other kernel threads. Though the number of user level threads is more than the number of processors, the number of these threads of a particular process than can be run parallelly is constrained by the number of kernel threads. Since the number of these kernel threads is less than the number of processors, some of the CPUs are left idle.

   b. The number of kernel threads allocated to the program is equal to the number of processors.
   Ans.
   Each kernel thread can handle an user level thread. Since the number of kernel threads is equal to number of processors each of them can run on a separate processor. If any thread performs a blocking system call, there are no more kernel threads that can be used to map a waiting user level thread. Thus, though there is an idle CPU, it cannot be used by this process.

   c. The number of kernel threads allocated to the program is greater than the number of processors but less than the number of user-level threads.
   Ans.
   In case of the blocking system call mentioned in (b) above, a waiting user level thread (mapped to a kernel thread) can be run on the CPU. This improves performance. Thus, the idle time of the CPU is reduced.

6. Discuss how the following pairs of scheduling criteria conflict in certain settings.
   a. CPU utilization and response time
   Ans.
   In order to increase the CPU utilization, the CPU should be used for performing user jobs (rather than for book-keeping activities). Thus, the time slices should be longer. This causes the scheduler to execute infrequently. Thus, a new process that enters the system will stay longer in the ready queue before being allocated the CPU. This causes an increase in the response time.

   b. Average turnaround time and maximum waiting time
   Ans.

Turnaround time is the amount of time between the arrival of the job and its completion. Waiting time is the amount of time spent in the ready queue. In order to decrease the maximum waiting time for each process, the time slices should be short. This results in a larger number of context switches and thus an increase in the turnaround time for each job. Thus there is a conflict.

c. I/O device utilization and CPU utilization
Ans.
Consider a system with a single I/O bound process and a large number of CPU bound processes. The I/O bound process typically has many very short CPU bursts. After finishing an I/O burst, the I/O bound process enters the ready queue. Since there are a large number of CPU bound processes, the CPU device is busy running one of these processes. Since the I/O bound process has a very short CPU burst, in order to increase I/O device utilization, the scheduler should preempt one of the CPU bound processes and allot the CPU to run the short burst of the I/O bound process. This happens as frequently as the CPU burst of the I/O bound process occurs. Thus, an attempt to increase the I/O device utilization causes context switches which are wasteful in terms of CPU utilization. The CPU utilization would be higher if the CPU bound jobs are not preempted.

7. Which of the following scheduling algorithms could result in starvation? Explain why.
   a. First-come, first-served
   b. Shortest job first
   c. Round robin
   d. Priority
   Ans.
   a. Shortest job first and
   b. Priority scheduling algorithm can result in starvation.

8. The traditional UNIX scheduler enforces an inverse relationship between priority number and priorities: The higher the number, the lower the priority. The scheduler recalculates process priorities once per second using the following function:

   Priority = (recent CPU usage / 2) + base

   Where base = 60 and recent CPU usage refers to a value indicating how often a process has used the CPU since priorities were last recalculated. Assume that recent CPU usage for P1 is 40, process P2 is 18,and process P3 is 10. What will be the new priorities for these three processes when priorities are recalculated? Based on this information, does the traditional UNIX scheduler raise or lower the relative priority of a CPU-bound process?
   Ans.
   Priority(P1) = 80
   Priority(P2) = 69
   Priority(P3) = 65

   The relative priority of a CPU-bound process is decreased as a result of this recalculation.

9. A real time system has four periodic processes:

   P1 with computation time 20 and period 40
   P2 with computation time 60 and period 500
   P3 with computation time 5 and period 20
   P4 with computation time 12 and period 100

Is this task set schedulable on a uniprocessor? Using what priority assignment algorithm is it schedulable? Explain your answer.
Ans.
For the task set to be schedulable, the following condition must hold:

$$\frac{C1}{P1} + \frac{C2}{P2} + \frac{C3}{P3} + \frac{C4}{P4} < 1$$
$$= \frac{20}{40} + \frac{60}{500} + \frac{5}{20} + \frac{12}{100}$$
$$= 0.5 + 0.12 + 0.25 + 0.12$$
$$= 0.99 < 1$$

The processes can be scheduled using Earliest Deadline First algorithm.

10. Under what circumstances will the "booting" sequence fail? Indicate how we can ensure that "re"boot will eventually succeed.
Ans.
Booting fails when:
a. POST (Power On Self Test fails) because the necessary hardware requirements are not met.
b. The boot sector of the Operating System is corrupted.
Booting can be made to succeed by providing the necessary hardware and providing a valid bootable device to boot from.

11. Are there any differences in OS to mouse driver interactions for an optical mouse vs. a mechanical mouse with a roller ball? Explain why (not)?
Ans.
There are no differences between OS to mouse driver interactions in the two cases. The mouse driver which receives the data representing the movement of the mouse ensures that an uniform interface is provided to the OS in both these cases.

12. The states a thread can be in are the same as that for a process. True or false? Explain.
Ans.
True.
Like a process, a thread can be in any one of several states: running, blocked, ready, or terminated. A running thread currently has the CPU and is active. A blocked thread is waiting for some event to unblock it. For example, when a thread performs a system call to read from the keyboard, it is blocked until input is typed. A thread can block waiting for some external event to happen or for some other thread to unblock it. A ready thread is scheduled to run as soon as it gets the CPU. The transitions between thread states are the same as the transitions between process states.