Operating systems

Question Bank Solution (Unit 2)

Q2. Explain why interrupts are not appropriate for implementing synchronization primitives in multiprocessor systems.

Answer:
Interrupts are not sufficient in multiprocessor systems since disabling interrupts only prevents other processes from executing on the processor in which interrupts were disabled; there are no limitations on what processes could be executing on other processors and therefore the process disabling interrupts cannot guarantee mutually exclusive access to program state.

Q3. Write two short methods that implement the simple semaphore `wait()` and `signal()` operations on global variable `S`.

Ans:
```
wait (S) {
  while (S <= 0);

  S--;

}

signal (S) {

  S++;

}
```

Q10. *Describe the four conditions that must hold simultaneously in a system if a deadlock is to occur.*

Ans: For a set of processes to be deadlocked: at least one resource must remain in a non-sharable mode, a process must hold at least one resource and be waiting to acquire additional resources held by other processes, resources in the system cannot be pre-empted, and a circular wait has to exist between processes.

Q11. Is it possible to have concurrency but not parallelism? Explain.

Answer:
Yes. Concurrency means that more than one process or thread is progressing at the same time. However, it does not imply that the processes are running simultaneously. The scheduling of tasks allows for concurrency, but parallelism is supported only on systems with more than one processing core.

- It only makes sense to create as many threads as there are blocking system calls, as the threads will be spent blocking. Creating additional threads provides no benefit. Thus, it makes sense to create a single thread for input and a single thread for output.

- Four. There should be as many threads as there are processing cores. Fewer would be a waste of processing resources, and any number > 4 would be unable to run.

13. Consider a multiprocessor system and a 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.
b. The number of kernel threads allocated to the program is equal to the number of processors.
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.

Answer:
When the number of kernel threads is less than the number of processors, then some of the processors would remain idle since the scheduler maps only kernel threads to processors and not user-level threads to processors. When the number of kernel threads is exactly equal to the number of processors, then it is possible that all of the processors might be utilized simultaneously. However, when a kernel-thread blocks inside the kernel (due to a page fault or while invoking system calls), the corresponding processor would remain idle. When there are more kernel threads than processors, a blocked kernel thread could be swapped out in favour of another kernel thread that is ready to execute, thereby increasing the utilization of the multiprocessor system.