Synchronization

a. Show that, if the wait() and signal() semaphore operations are not executed atomically, then mutual exclusion may be violated.

Answer: A wait operation atomically decrements the value associated with a semaphore. If two wait operations are executed on a semaphore when its value is 1, if the two operations are not performed atomically, then it is possible that both operations might proceed to decrement the semaphore value thereby violating mutual exclusion.

b. Show how to implement the wait() and signal() semaphore operations in multiprocessor environments using the TestAndSet() instruction. The solution should exhibit minimal busy waiting.

```c
int guard = 0;
int semaphore value = 0;
wait()
{ while (TestAndSet(&guard) == 1);
  if (semaphore value == 0) { atomically add process to a queue of processes
    waiting for the semaphore and set guard to 0;
  } else { semaphore value--;
    guard = 0;
  }
}

signal()
{ while (TestAndSet(&guard) == 1);
  if (semaphore value == 0 &&
    there is a process on the wait queue)
    wake up the first process in the queue
    of waiting processes
  else
    semaphore value++;
  guard = 0;
}
c. 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.

**Answer:** If a user-level program is given the ability to disable interrupts, then it can disable the timer interrupt and prevent context switching from taking place, thereby allowing it to use the processor without letting other processes to execute.

**Memory management**

a. Discuss the hardware support required to support demand paging.
There are primary and secondary memory (swapping space).
Hardware to support page map table such as page table base register.
Hardware to generate page fault interrupt.
In order to improve performance, hardware translation look-aside buffer can also be used.

b. Assume that we have a demand-paged memory. The page table is held in registers. It takes 8 milliseconds to service a page fault if an empty frame is available or if the replaced page is not modified and 20 milliseconds if the replace page is modified. Memory-access time is 100 nanoseconds.
Assume that the page to be replaced is modified 70 percent of the time. What is the maximum acceptable page-fault rate for an effective access time to no more than 200 nanoseconds?

**Answer:**

\[
200 \text{ nanosec} = (1 - P) \times 100 \text{ nanosec} + (0.3P) \times 8 \text{ millisec} + (0.7P) \times 20 \text{ millisec}
\]

\[
0.1 = -0.01P + 2400P + 14000P
\]

\[
0.1 = 16400P
\]

\[
P = 0.000006
\]

c. Suppose that your replacement policy (in a paged system) is to examine each page regularly and to discard that page if it has not been used since the last examination. What would you gain and what would you lose by using this policy rather than LRU or second-chance replacement?

**Answer:** Such an algorithm could be implemented with the use of a reference bit. After every examination, the bit is set to zero; set back to one if the page is referenced. The algorithm would then select an
arbitrary page for replacement from the set of unused pages since the last examination.

The advantage of this algorithm is its simplicity - nothing other than a reference bit need be maintained. The disadvantage of this algorithm is that it ignores locality by only using a short time frame for determining whether to evict a page or not. For example, a page may be part of the working set of a process, but may be evicted because it was not referenced since the last examination (i.e. not all pages in the working set may be referenced between examinations.)