1. Identify one way in which an operating system helps make efficient use of a computing system. (1 pts.)

   many ways to choose from: scheduling the CPU, allocating memory, overlapping I/O and CPU execution, ...

2. Explain what it means to say that an operating system is “interrupt-driven”. (2 pts.)

   The only way to enter a traditional OS is through an interrupt.
   (I.e., system calls cause intentional interrupts, I/O completion and errors cause external interrupts, etc.)

3. The four generic actions that hardware performs in response to an interrupt are: (2 pts. each)

   1) Save PC and PSR
   2) Switch to kernel execution mode
   3) Disable or restrict further interrupts
   4) Load new PC from interrupt vector table

4. What instruction or other hardware mechanism does an operating system use to transfer control of the processor to a user thread? (2 pts.)

   IRET (interrupt return), which restore the save PC and PSR

Multithreaded Process/Thread. Circle only one of P or T. (1.5 pts. each)

5. P / T Has an associated SP (stack pointer).
6. P / T Has an associated PC (program counter).
7. P / T Is assigned the memory map (e.g., page table).
8. P / T Has an associated PSR (processor status register).
9. P / T Is the unit of resource allocation for resources other than the processor (e.g., open file).

   5, 6, 8 are Thread; 7, 9 are Process

True/False. Circle only one of T or F. (1.5 pts. each)

10. T / F An instruction that can only be executed in kernel mode is called a privileged instruction.
11. T / F For protection, jump to subroutine should be valid in user mode but invalid in kernel mode.
12. T / F To provide multiuser protection, hardware must have at least three execution modes.
13. T / F The interrupt vector table should be held in protected memory and not arbitrarily changed by users.
14. T / F When power is turned on, a processor will start executing a bootstrap loader in user mode.
15. T / F On current systems, threading is implemented directly by the kernel instead of at the user-library level.
16. T / F If you prevent deadlock, you also prevent starvation.

   10, 13, 15 are True; 11, 12, 14, 16 are False

Circle one or more of P, T, C, L, or S, as applies. (2.5 pts. each)

17. P / T / C / L / S  Blocks user threads by causing thread state change (i.e., no busy wait at user level).
18. P / T / C / L / S  Uses an address-watch register that snoops bus on shared-memory multiprocessor.
19. P / T / C / L / S  Machine instruction in which read-modify-write actions are performed as atomic unit

17 is S only; 18 is L only; 19 is T and C

20. Here is proposed code for software synchronization that uses a global “available” flag. (8 pts.)

```java
boolean available = true;  // initialization of available flag
T1S1 if( available ){
    T1S2 available = false;
} else{
    T1S3 while( !available );
    T1S4 available = false;
}
T1S5 CS1
T2S1 if( available ){
    T2S2 available = false;
} else{
    T2S3 while( !available );
    T2S4 available = false;
}
T2S5 CS2
T1S6 available = true;
T2S6 available = true;
```

Show a scenario in which this approach fails one of the three rules for mutual exclusion. Give the exact sequence using the T<i>S<j> statement ids so that it is clear the order in which the statements are executed. Explain what type of failure occurs.

Initially
available
true

T1S1 test is true
true
T2S1 test is true
true

T1S2 writes
false
T2S2 writes
false

T1S5 in CS
false
T2S5 in CS
false

both threads in CS at same time =>

failure of mutual exclusion

21. Fill in the blanks for the semaphore signal logic on a semaphore S. (10 pts.)

```java
if ( _____empty(S.blocked_list)______ ){
    S.value = _____S.value + 1______;
} else{
    select thread on _____S.blocked_list______ and move to _____ready_list______;
}
```

22. Fill in the blanks for the semaphore wait logic on a semaphore S. (7.5 pts.)

```java
if ( _____S.value == 0______){
    block calling thread and place on _____S.blocked_list______;
} else{
    S.value = _____S.value - 1______;
}
```
23. The system calls for semaphore operations discussed in class also made use of interrupt disabling and a spin lock at the beginning of each service routine, i.e.,

![Diagram showing the sequence of events involving interrupts and spin lock]

Explain why this was necessary. (7 pts.)

Interrupts are disabled to prevent interference from events on the current processor in updating shared data. A spin lock is used to prevent interference from events on other processors in updating shared data.

24. Describe the resource allocation pattern for message passing. Show the relevant pseudo code. (10 pts.)

Create a mailbox with one message for each resource; each message should contain a unique identifier that can be used to access the associated resource.

The code pattern for a process will then be:

```c
receive( msg, mbox );
< use resource identified by the resource id field in msg >
send( msg, mbox );
```

25. Give the failure scenario for priority inversion with three threads of differing priorities using semaphore-based mutual exclusion (i.e., with blocking). (6 pts.)

There are three threads, each with fixed priority: a low-priority thread (LPT), a medium-priority thread (MPT), and a high-priority thread (HPT).

There is a binary semaphore (mutex) initialized to 1. LPT and HPT wait on the semaphore; MPT does not use the semaphore.

LPT is the only ready thread, so LPT runs.
LPT enters a critical section after waiting on mutex.
HPT is resumed and preempts LPT.
HPT runs, waits on mutex, and is blocked.
LPT runs.
MPT is resumed and preempts LPT. Thus LPT cannot finish its critical section and signal mutex.
MPT can potentially run indefinitely and prevents LPT and therefore HPT from running.
26. How can the problem of priority inversion with semaphores in question 25 be avoided? (3 pts.)

Priority inheritance in which LPT would temporarily inherit the high priority of HPT while HPT is on the blocked list of the mutex semaphore, or by a dynamic-priority scheduling policy with priority aging in which MPT’s priority would decay while running and LPT’s priority would build while sitting in the ready list.

27. Initialize the semaphore values and add the appropriate calls to wait and signal so that the following code will work correctly for the Producer/Consumer with Bounded Buffer with n slots in the buffer. (10 pts.)

```pascal
var mutex: semaphore( ___1___ ); (* mutual exclusion of buffer access *)
empty: semaphore( ___N___ ); (* number of empty slots in buffer *)
full: semaphore( ___0___ ); (* number of items in buffer *)

procedure producer(i:integer);
begin
  var p_item: item_type;
  p_item = produce();
  wait(empty);
  wait(mutex);
  append(p_item); (* to shared buffer *)
  signal(mutex);
  signal(mutex);
  signal(full);
  signal(empty);
end;

procedure consumer(i:integer);
begin
  var c_item: item_type;
  c_item = take(); (* from shared buffer *)
  signal(mutex);
  signal(empty);
  consume(c_item);
end;
```

Extra credit.

XC-1. Use the compare and swap instruction to implement an optimistic concurrency increment to a shared, global value of ‘X’. That is, X=X+1. Your code can be C-like pseudo code for one of the threads. The other threads will be assumed to have the same type of code. (up to 10 pts.)

```pascal
local_X = X;
repeat {
  new_X = local_X + 1;
  rc = CAS( local_X, X, new_X );
} until( rc );
```

XC-2. Explain how transactional memory is an extension of the hardware support provided for LL/SC. (up to 5 pts.)

With transactional memory, the processor watches for write interference from other processors across multiple cache lines rather than watching for writes to the one memory word specified by the single address watch register.