Multithreaded Process / Thread. Circle only one of MTP or T. (1 pt. each)
1. MTP / T This object contains a table of open files.
2. MTP / T This object is associated with a single PC (program counter).
3. MTP / T In a paging system, this object has a one-to-one association with a page table.
   1 and 3 are MTP; 2 is T

Compare & Swap / Load Linked & Store Conditional / Transactional Memory. Circle only one of CS, LLSC, or TM. (1 pt. each)
4. CS / LLSC / TM Uses an address register to snoop on the memory bus.
5. CS / LLSC / TM Watches for changes in multiple cache lines to determine outcome.
6. CS / LLSC / TM Compares value in register to value in memory to determine outcome.
   4 is LLSC; 5 is TM; 6 is CS

Semaphore / Conditional Critical Region / Monitor. Circle one or more of Sem, CCR, or Mon, as applies. (1 pt. each)
7. Sem / CCR / Mon Has scheduling guarantees on when a signaled thread will run.
8. Sem / CCR / Mon Provides a signal operation that has no effect when the wait queue is empty.
9. Sem / CCR / Mon Provides language structures that enforce mutually exclusive access to shared variables.
   7 and 8 are Mon only; 9 is both CCR and Mon

Short-Term / Medium-Term / Long-Term Scheduling. Circle only one of S, M, or L. (1 pt. each)
10. S / M / L Is called a swapper.
11. S / M / L Is called a dispatcher.
12. S / M / L Typically runs after every interrupt.
   10 is M; 11 and 12 are S

FCFS / RR / MLFQ / SRTN. Circle one or more of F, R, M, S, as applies. (1 pt. each)
13. F / R / M / S Is preemptive.
14. F / R / M / S Uses time slices.
15. F / R / M / S Uses multiple queues.
   13 is R, M, and S; 14 is R and M; 15 is M only

Fully live / Starvation / Deadlock. Circle only one of D, S, or L. (1 pt. each)
16. F / S / D No process under consideration can make forward progress.
17. F / S / D All processes under consideration can make forward progress.
18. F / S / D At least one process under consideration can make forward progress but at least one cannot.
   16 is D; 17 is F; 18 is S

Paging / Segmentation. Circle one or both of P or S, as applies. (1 pt. each)
19. P / S Can have external fragmentation.
22. P / S Process memory image is divided into variable-length regions that match the program structure.
   19 and 22 are S only; 20 and 21 are both P and S
Scheduling Mechanism / Replacement Mechanism / Scheduling Policy / Replacement Policy. Circle only one of SM, RM, SP or RP. (1 pt. each)

23. SM / RM / SP / RP RR.
24. SM / RM / SP / RP LRU.
25. SM / RM / SP / RP CPU timer.
27. SM / RM / SP / RP Reference bit.

23 is SP; 24 is RP; 25 and 26 are SM; 27 is RM

Directory entry / Open File Table Entry / Process Control Block. Circle only one of D, OFTE, or PCB, as applies. (1 pt. each)

28. D / OFTE / PCB  Contains a pointer to the current record in the file.
29. D / OFTE / PCB  This object is created when an existing file is opened.
30. D / OFTE / PCB  This object is located using a character string containing the filename.

28 and 29 are OFTE; 30 is D

Use this table for questions 36-41 for the Banker’s algorithm. Treat each question independently. (1 pt. each)

<table>
<thead>
<tr>
<th>process</th>
<th>max_demand</th>
<th>allocated</th>
<th>remaining_claim</th>
<th>unused units = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

31. T / F Can safely grant request of P1 for 1 unit.
32. T / F Can safely grant request of P2 for 1 unit.
33. T / F Can safely grant request of P1 for 2 units.
34. T / F Can safely grant request of P2 for 2 units.
35. T / F Can safely grant request of P3 for 2 units.

31, 32, and 34 are True; 33 and 35 are False

36. Under what condition(s) is it permissible to execute a privileged instruction? (1 pt.)

When the CPU is in kernel execution mode

37. Here is the code for our first approach to software synchronization using a single lock flag.

    initially: lock = false
    T1S1    while(lock);
    T1S2    lock = true;
    T1S3    CS1
    T1S4    lock = false;
    T2S1    while(lock);
    T2S2    lock = true;
    T2S3    CS2
    T2S4    lock = false;

Show a scenario in which this approach fails one of the three rules for mutual exclusion. (Give the exact sequence using the T<&gt;i>S&lt;j&gt; statement ids in a way that the order of execution is clear.) (4 pts.)

Initially

<table>
<thead>
<tr>
<th>lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
</tr>
</tbody>
</table>

T1S1 falls through false false T2S1 falls through
T1S2 writes lock true true T2S2 writes lock
T1S3 in CS true true T2S3 in CS ] both threads in CS at same time =>
T1S4 ] failure of mutual exclusion
38. Why does the Test & Set instruction allow us to use a single lock flag for mutual exclusion? (2 pts.) Show the pseudo-code for a spin lock using the Test & Set instruction. (up to 5 pts. extra credit)

Test & Set is an atomic instruction that locks the memory bus and performs read-modify-write (RMW) actions without any intervening memory accesses.

```c
while( test_and_set( lock ) );
<critical section>
lock = false;
```

39. Fill in the blanks for the semaphore wait logic. (3 pts.)

```c
if ( S.value == 0 ){
    block calling thread and place on S.blocked_list;
}else{
    S.value = S.value - 1;
}
```

40. Fill in the blanks for the semaphore signal logic. (3 pts.)

```c
if ( empty( S.blocked_list ) ){
    S.value = S.value + 1;
}else{
    select thread on S.blocked_list and move to ready_list;
}
```

41. A friend says that the sleep() and wakeup() system calls in early Unix were not as powerful as semaphores for synchronizing processes. Do you agree? If so, explain what they lacked. If not, then explain how you would convince your friend that these Unix system calls have the same power as semaphores. (2 pts.)

You should disagree and show your friend how semaphores can be implemented with sleep() and wakeup().

While not required for the answer, here is the code from the lecture notes:

```c
void wait( struct sem *s ){
    s->val--;
    if( s->val < 0 ){ add( s->queue, getpid() ); sleep( getpid() ); } }

void signal( struct sem *s ){
    s->val++;
    if( s->val <= 0 ) wakeup( get( s->queue ) );
}
```
42. What two types of basic synchronization design patterns are used in the bounded-buffer (producer-consumer) design pattern? (2 pts.)

   mutual exclusion and notification

43. Consider the bounded-buffer procedures “put” and “get”. Add semaphore declarations and the necessary wait and signal operations to produce a correctly-synchronized solution of the bounded-buffer problem. (3 pts.)

   var       buf: array[0..N-1] of item_type;
   in: integer initial_value( 0 );
   out: integer initial_value( 0 );

   (* semaphores and their initial values *)

   mutex:   semaphore initial_value( 1 );
   empty_slots: semaphore initial_value( N );
   full_slots: semaphore initial_value( 0 );

   procedure put( input item: item_type )
   begin
      wait( empty_slots );
      wait( mutex );
      buf[in] := item;
      in := (in + 1) mod N;
      signal( mutex );
      signal( full_slots );
   end

   procedure get( output item: item_type )
   begin
      wait( full_slots );
      wait( mutex );
      item := buf[out];
      out := (out + 1) mod N;
      signal( mutex );
      signal( empty_slots );
   end

44. Consider the bounded-buffer monitor “buffer” and the skeletons of the monitor procedures “put” and “get”. Add condition variable declarations and the necessary if-then statements and wait and signal operations on the condition variables to produce a correctly-synchronized solution of the bounded-buffer problem using a Brinch-Hansen-style monitor. (3 pts.)

   buffer = monitor
   var       buf: array[0..N-1] of item_type;
   in: integer;
   out: integer;
   count: integer;

   (* condition variables *)

   not_full: condition;
   not_empty: condition;

   procedure put( input item: item_type )
   begin
      if count = N then not_full.wait;
      buf[in] := item;
      in := (in + 1) mod N;
      count := count + 1;
      not_empty.signal
   end

   procedure get( output item: item_type )
   begin
      if count = 0 then not_empty.wait;
      item := buf[out];
      out := (out + 1) mod N;
      count = count - 1;
      not_full.signal
   end
begin (* monitor initialization *)
in := 0;
out := 0;
count := 0;
end (* monitor *)

45. Give the individual departure times (dep) and turnaround times (trn) for the following three processes using
   a) First Come, First Served (FCFS), and
   b) Shortest Remaining-Time Next (SRTN) where any ties (i.e., equal remaining times) are broken in FCFS order.
   Assume that zero time is required for a process to arrive, to be added to the queue, or to be dispatched. (6 pts.)

<table>
<thead>
<tr>
<th>process</th>
<th>arrival</th>
<th>service</th>
<th>FCFS dep</th>
<th>FCFS trn</th>
<th>SRTN dep</th>
<th>SRTN trn</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>5</td>
<td>5_5</td>
<td>6_6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>6</td>
<td>11_10</td>
<td>12_11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>1</td>
<td>12_10</td>
<td>3_1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 1 2 3 4 5 6 7 8 9 10 11 12

46. Explain what processor affinity means and why it would be used in a scheduling policy. (2 pts.)

Processor affinity is a policy for multiprocessor scheduling in which a thread is dispatched to the same
processor on which it last executed.

Running on the same processor allows reuse of cache lines that might be left over from the previous
execution. (In a NUMA system, affinity may be node-oriented and can be used to optimize the number of
local memory accesses vs. the number of remote memory accesses.)

(While not needed for the answer, processor affinity can be hard or soft, that is, it can be required without
exception or it can be a goal for the dispatcher to try to meet. Hard affinity may be needed for a system
with asymmetric processors.)

47. Explain the problem of priority inversion in the three-thread case and what can be done to prevent it. (4 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.

1) LPT is the only ready thread, so LPT runs.
2) LPT enters a critical section after waiting on mutex.
3) HPT is resumed and preempts LPT.
4) HPT runs, waits on mutex, and is blocked.
5) LPT runs.
6) MPT is resumed and preempts LPT. Thus LPT cannot finish its critical section and signal mutex.
7) MPT can potentially run indefinitely and prevents LPT and therefore HPT from running.

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.
48. Consider a 10-page virtual memory with virtual pages numbered from 0 to 9, 10-word pages with words numbered from 0 to 9, a 4-page physical memory with page frames numbered from 0 to 3.

<table>
<thead>
<tr>
<th>VPN</th>
<th>P</th>
<th>R</th>
<th>W</th>
<th>X</th>
<th>M</th>
<th>PFN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

VPN - presence bit  
R - read permission  
W - write permission  
X - execute permission  
M - modified bit  
PFN - page frame number

Starting each time from this page table, what is the result of these virtual address accesses - give either the physical address or the system action (e.g., page fault or protection violation). (1 pt. each)

- a. inst. fetch 13  
  physical address 23
- b. read 52  
  page fault
- c. write 32  
  page fault
- d. write 13  
  protection violation

49. The Xerox Sigma 7 was a computer system with paged memory. It used word addressability rather than byte addressability. It had a page size of 512 words, a virtual address space of 131,072 (=128*1024) words, and a physical memory of 131,072 (=128*1024) words. (Do not convert words to bytes in your answers.) (5 pts.)

- a) How many bits are required in the virtual address of a word?
  \[ \log_2(131,072) = \log_2(128 \times 1024) = \log_2(2^7 \times 2^{10}) = \log_2(2^{17}) = 17 \]

- b) What is the division of bits in the virtual address between the virtual page number and the offset?
  for offset bits => \[ \log_2(512) = \log_2(2^9) = 9 \]
  for VPN bits => 17 – 9 = 8

- c) How many bits are required in the physical address of a word?
  same as in part (a), 17 bits

- d) What is the division of bits in the physical address between the page frame number and the offset?
  for offset bits => \[ \log_2(512) = \log_2(2^9) = 9 \]
  for PFN bits => 17 – 9 = 8

- e) The Sigma 7 used a normal page table. How many entries did the page table need?
  For a normal page table, there will be one entry per VPN; so, \(2^8 = 256\) entries

50. What is the formal definition of a working set in a paged memory system? (2 pts.)

the set of pages referenced in the last \(\Delta\) units of time

51. Are there times that a page write-permission violation can indicate something other than a protection error? If so, explain. (2 pts.)

Yes, this can signal the first write to a copy-on-write page. The OS must check a COW bit in the PTE or in a shadow page table before the violation is determined to be a protection error.
52. Other than performance, give a reason why you would page-align an object in virtual memory. (1 pt.)

An object would need to be page-aligned in order to start a new protection domain (e.g., read-only).

53. Explain the difference between a logical record and a physical record. (2 pts.)

A logical record is the unit of access by the application. (This can be as small as a byte at a time.)
A physical record is the unit of access to the I/O device. (This is typically thousands of bytes at a time.)

54. A friend recommends that all files be stored as sets of contiguous sectors on a disk to speed sequential processing. Identify and explain any drawbacks to this type of file organization. (3 pts.)

1) Contiguous organization for files will cause external fragmentation as files change and thus require compaction (this is, defragmentation).
2) It is difficult to know how many sectors to allocate when a file is first created.
3) Copying a file to a larger set of free sectors on disk is required when the file cannot grow in place.

55. Identify the three things you would find in an open file table entry (that is, the per-open control block created for a process when a file is opened). (3 pts.)

1) access permission under which the file was opened
2) pointer to current record
3) pointers to first and last records (for sequential) or to i-node (for indexed)

56. What is the difference between an access control list and a capability list? (2 pts.)

An ACL is held by an object and contains a list of users and their access permissions.
A capability list is held by a user and contains a list of objects and their access keys.

57. Identify two specific instances where we studied context-dependent names. (2 pts.)

virtual addresses and unqualified (or partially-qualified) file names

58. For each of the instances in 57, identify how you would locate the appropriate naming table that is used to map the name to an object. (2 pts.)

For a virtual address, the page or segment table base address is held in the PCB. This address is loaded into a control register in the CPU on dispatch.

For an unqualified or partially-qualified file name, the current working directory is held in one of the variables maintained by the command interpreter. This variable is initialized when a user logs in.

59. Identify a common hardware-managed cache used by the address translation logic on a paged-memory computer system. What does this cache contain? (1 pt.)

The TLB (translation lookaside buffer) holds copies of page or segment table entries.
(Some of the Motorola processors call this cache an address translation cache, or ATC for short.)

60. Identify a common software-managed cache used by the I/O subsystem of the OS on many computer systems. What does this cache contain? (1 pt.)

The disk cache holds blocks of data that are have been read from the disk or will be written to the disk.