1. Which of the following are more like policies, not mechanisms?
(a) The timer interrupt
(b) How long a time quantum should be
(c) Saving the register state of a process
(d) Continuing to run the current process when a disk I/O interrupt occurs

2. For a workload consisting of ten CPU-bound jobs of varying lengths (half run for 1 second, and the other half for ten seconds), which policy would result in the lowest total run time for the entire workload?
(a) Shortest Job First
(b) Shortest-Time to Completion First
(c) Round-robin with a 100 millisecond quantum
(d) Multi-level Feedback Queue

3. Assume the following schedule for a set of three jobs, A, B, and C:
A runs first (for 10 time units) but is not yet done
B runs next (for 10 time units) but is not yet done
C runs next (for 10 time units) and runs to completion
A runs to completion (for 10 time units)
B runs to completion (for 5 time units)
Which scheduling disciplines could not allow this schedule to occur?
(a) FIFO
(b) Round Robin
(c) STCF (Shortest Time to Completion First)
(d) Lottery Scheduling

4. A program's main function is as follows:

```
int main(int argc, char *argv[]) {
        char *str = argv[1];
        while (1)
        printf("%s", str);
        return 0;
}
```

Two processes, both running instances of this program, are currently running (you can assume nothing else of relevance is, except perhaps the shell itself). The programs were invoked as follows, assuming a "parallel command" as per our homework:
$main a &&& main b
Which of the following are possible screen captures of the command output?
(a) abababab ...
(b) aaaaaaaa ...
(c) bbbbbbbb ...
(d) aaaabbbb ...
(e) bbbbaaaa ...

5. Processes (or threads) can be in one of three states: Running, Ready, or Blocked. Which of the following examples is in Blocked state?
(a) Waiting in Domain Read() for a message from some other process to arrive.
(b) Spin-waiting for a variable x to become non-zero.
(c) Having just completed an I/O, waiting to get scheduled again on the CPU.
(d) Waiting inside of pthread cond wait() for some other thread to signal it.

6. Processes exist in a number of different states. We've focused upon a few (Running, Ready, and Blocked) but real systems have slightly more. For example, xv6 also has an Embryo state (used when the process is being created), and a Zombie state (used when the process has exited but its parent hasn't yet called wait() on it). Assuming you start observing the states of a given process at some point in time (not necessarily from its creation, but perhaps including that), which process states could you possibly observe? Note: once you start observing the process, you will see ALL states it is in, until you stop sampling.
(a) Running, Running, Running, Ready, Running, Running, Running, Ready
(b) Embryo, Ready, Ready, Ready, Ready, Ready
(c) Running, Running, Blocked, Blocked, Blocked, Running
(d) Running, Running, Blocked, Blocked, Blocked, Ready, Running
(e) Embryo, Running, Blocked, Running, Zombie, Running

7. The following code is shown to you:

```
int main(int argc, char *argv[]) {
        printf("a");
        fork();
        printf("b");
        return 0;
}
```

Assuming fork() might fail (by returning an error code and not creating a new process) and printf() prints its outputs immediately (no buffering occurs), what are possible outputs of the program?
(a) ab
(b) abb
(c) bab
(d) bba
(e) a

8. Which of the following will NOT guarantee that deadlock is avoided?
(a) Acquire all resources (locks) all at once, atomically
(b) Use locks sparingly
(c) Acquire resources (locks) in a fixed order
(d) Be willing to release a held lock if another lock you want is held, and then try the whole thing over again

9. Assume this attempted implementation of a lock:

```
void init(lock_t *mutex) {
    mutex->flag = 0; // 0 -> lock is available, 1 -> held
}
void lock(lock_t *mutex) {
    while (mutex->flag == 1);
    mutex->flag = 1;
}
void unlock(lock_t *mutex) {
    mutex->flag = 0;
}
```

Assume 5 threads are competing for this lock. How many threads can possibly acquire the lock? To answer: Fill in A for possible, B for not possible for following number of threads:
(a) 1
(b) 2
(c) 3
(d) 4
(e) 5

10. Assume the following list insertion code, which inserts into a list pointed to by shared global variable head:

```
int List_Insert(int key) {
```

```
        node_t *n = malloc(sizeof(node_t));
        if (n == NULL) { return -1; }
        n->key = key;
        n->next = head;
        head = n; return 0;
}
```

This code is executed by each of three threads exactly once, without adding any synchronization primitives (such as locks). Assuming malloc() is thread-safe (i.e., can be called without worries of data races) and that malloc() returns successfully, how long might the list be when these three threads are finished executing? (assume the list was empty to begin) To answer: Fill in A for possible, B for not possible. Assumes list empty at beginning.

(a) 0
(b) 1
(c) 2
(d) 3
(e) 4

11. A Semaphore is a useful synchronization primitive. Which of the following statements are true of semaphores?
(a) Each semaphore has an integer value
(b) If a semaphore is initialized to 1, it can be used as a lock
(c) Semaphores can be initialized to values higher than 1
(d) A single lock and condition variable can be used in tandem to implement a semaphore
(e) Calling sem post() may block, depending on the current value of the semaphore

12. Here is the classic semaphore version of the producer/consumer problem:

```
void *producer(void *arg) { // core of producer
    for (i = 0; i < num; i++) {
        sem_wait(&empty);
        sem_wait(&mutex);
        put(i);
        sem_post(&mutex);
        sem_post(&full);
    }
}

void *consumer(void *arg) { // core of consumer
    while (!done) {
        sem_wait(&full);
        sem_wait(&mutex);
        int tmp = get(i);
        sem_post(&mutex);
        sem_post(&empty);
        // do something with tmp ...
    }
}
```

For the following statements about this working solution, which statements are true?
(a) The semaphore full must be initialized to 0
(b) The semaphore full must be initialized to 1
(c) The semaphore empty must be initialized to 1
(d) The semaphore empty can be initialized to 1
(e) The semaphore mutex must be initialized to 1

13. One way to avoid deadlock is to schedule threads carefully. Assume the following characteristics of threads T1, T2, and T3:
• T1 (at some point) acquires and releases locks L1, L2
• T2 (at some point) acquires and releases locks L1, L3
• T3 (at some point) acquires and releases locks L3, L1, and L4

For which schedules below is deadlock possible?
(a) T1 runs to completion, then T2 to completion, then T3 runs
(b) T1 and T2 run concurrently to completion, then T3 runs
(c) T1, T2, and T3 run concurrently
(d) T3 runs to completion, then T1 and T2 run concurrently
(e) T1 and T3 run concurrently to completion, then T2 runs

14. The following multithreaded code snippet is supposed to add 'amount' to the global variable 'balance'. For each of the following, indicate whether the code snippet has a race condition, works correctly, or is broken in some other way (but not a race condition).

(a)
```
void update(int amount) {
    int tmp = amount;
    lock();
    tmp = tmp + balance;          //Race, Works, Broken?
    unlock();
    balance = tmp;
}
```

(b)
```
void update(int amount) {
    int tmp = amount;
    lock();
    tmp = tmp + amount;           //Race, Works, Broken?
    balance = tmp;
    unlock();
}
```

(c)
```
void update(int amount) {
    lock();
    balance = balance + amount;   //Race, Works, Broken?
    lock();
}
```

(d)
```
void update(int amount) {
    int *tmp = &balance;
    lock();
    *tmp = *tmp + amount;         //Race, Works, Broken?
    unlock();
}
```

(e)
```
void update(int amount) {
    int tmp = balance;
    lock();
    tmp = tmp + amount;           //Race, Works, Broken?
    unlock();
}
```

15. For a linked allocation scheme in a File System, each block of the file tells you where the next block is. Select those of the following which are true when using a linked allocation scheme.
(a) Sequentially overwriting all of the blocks of a file takes about the same time as overwriting the last block
(b) Each read of a random block of a file requires only about one I/O
(c) Computing the size of a file takes about as much time as reading the entire file
(d) Reading an entire file sequentially takes about the same time as reading just the last block of the file
(e) The size of the inode is smaller than in a more typical Unix file system inode

16. A process has associate state with it, which must be saved and restored when a context switch takes place. Select those of the following states that do not need to be saved and restored during a context switch:
(a) The general purpose (integer) registers
(b) The floating point registers
(c) The contents of the TLB
(d) The contents of the processes' address space
(e) The program counter

17. Assume we have a Broken RAID system which sometimes does not fill in the redundant parts of the disk correctly. Select those of the following examples which do not have correct parity/redundancy bits:
(a) BRAID Level 4:

Disk 0	Disk 1	Disk 2	Disk 3
0	0	0	0
0	1	1	1
0	1	0	1
1	1	1	1

(b) BRAID Level 5:

Disk 0	Disk 1	Disk 2	Disk 3
0	0	0	0
0	1	0	1
0	0	0	1
0	1	1	1

(c) BRAID Level 1 (Mirroring):

Disk 0	Disk 1	Disk 2	Disk 3
0	0	0	0
0	0	1	1
1	1	0	1
1	1	1	1

(d) BRAID Level 0:

Disk 0	Disk 1	Disk 2	Disk 3
0	0	0	0
0	1	1	1
0	1	0	1
1	1	1	1

18. For a specific RAID array, a read of a block takes about 10ms, a write of a block also takes about 10ms. This RAID is likely:
(a) RAID-0 (no protection)
(b) RAID-1 (mirroring)
(c) RAID-4 (parity disk)
(d) RAID-5 (rotating parity)

19. You are given a system with 64 bytes of physical memory, 4 byte pages, and 16-byte virtual address spaces. Which ranges of virtual address are valid?
(a) 0 … 64
(b) 0 … 16
(c) 4 … 7
(d) 12 … 15
(e) 80 … 89

20. What is throughput for each disk shown in Table for Random workload? Assume 4-KB read size.

	Disk A	Disk B
Capacity	300 GB	1 TB
RPM	15,000	7,200
Avg Seek	5 ms	8 ms
Max Transfer	120 MB/s	100 MB/s
Platters	4	4
Cache	16 MB	32 MB

21. Which statements are true about the multi-level page table?
(a) A multi-level page table may use more pages than a linear page table
(b) It's easier to allocate pages of the page table in a multi-level table (as compared to a linear page table)
(c) Multi-level page table lookups take longer than linear page table lookups
(d) With larger virtual address spaces, usually more levels are used
(e) TLBs are useful in making multi-level page tables even smaller

22. A translation lookaside buffer (TLB) is generally used to:
(a) translate virtual page numbers into physical page numbers
(b) translate physical page numbers into virtual page numbers
(c) make segmentation have the benefits of a pure paging approach
(d) translate the addresses generated by loads
(e) translate the addresses generated by stores
(f) translate the addresses generated by instruction fetches
(g) remove the need for a full-sized page table
(h) make translations happen quickly

23. Select proper uses of a TLB:
(a) To cache frequently accessed data
(b) To cache frequently used address translations
(c) To speed up the time to load data from memory
(d) To enforce protection across pages
(e) To help determine whether a page is resident in memory or not

24. Assume the following: Memory is only 100 bytes. Process A is loaded at address 0 and needs 10 bytes of memory, and Process B is loaded at address 30 and needs 20 bytes of memory. (note: there is no paging or segmentation, just simple base and bounds). Which of the following are broken decisions?
(a) Place new process C (needs 30 bytes of memory) at address 0
(b) Place new process C (needs 20 bytes of memory) at address 10
(c) Place new process C (needs 30 bytes of memory) at address 40
(d) Place new process C (needs 30 bytes of memory) at address 80
(e) Place new process C (needs 10 bytes of memory) at address 90

25. Apply true LRU replacement to the following reference string of page number references, assuming that 3 page frames are allocated to the process. Show which three frames are in memory after each page reference. Show which page each frame is allocated to (if any) after each reference.
Reference string: 3 2 4 5 4 1 2 4 3 4

26. An inode structure in a hypothetical file system has following structure:

4 direct block
1 Indirect block
1 double indirect block

Assume block size as 8 KB (8*1024 Bytes), and block pointer size as 8 Bytes. Calculate the maximum file size that this inode structure can support?