1. (20 pts) **Short answer questions.**

(a) (4 pts) In what type of applications would circuit switching be preferable to packet switching and vice versa?

**Solution:** Circuit switching supports applications in which any interruption of service or out of order delivery would be very noticeable, for example: video or voice applications. Packet switching gets good performance for short bursty communication, such as sending email across a WAN.

(b) (4 pts) What is a distributed system?

**Solution:** A distributed system is a set of physically separate processors with distinct memories, separate clocks, and independent operating systems connected by one or more communication links.

(c) (4 pts) Name 4 disk scheduling algorithms. What component of I/O time do they reduce, and under what type of loads?

**Solution:** FIFO - First-in-First-out, SSTF - Shortest Seek Time First, Scan, and C-Scan. Except for FIFO, their goal is to reduce seek time on the disk under high loads. Under very low load, all schemes revert to FIFO.

(d) (4 pts) What kinds of tradeoffs do the different approaches to distributed file systems make? Name at least two.

**Solution:** One tradeoff is performance versus consistency between file versions. The higher degree consistency the worse performance tends to get. Another tradeoff is performance and fault tolerance. For example, keeping state at the server yields better performance but makes recovery after faults more costly and time consuming.

(e) (4 pts) What are the 4 conditions for deadlock? What makes deadlock detection more difficult in a distributed system?

**Solution:**

1. **Mutual Exclusion** - at least one thread must hold a resource in non-sharable mode, i.e., the resource may only be used by one thread at a time.

2. **Hold and Wait** - at least one thread holds a resource and is waiting for other resource(s) to become available. A different thread holds the resource(s).

3. **No Preemption** - A thread can only release a resource voluntarily; another thread or the OS cannot force the thread to release the resource.

4. **Circular wait** - A set of waiting threads \( \{t_1, \ldots, t_n\} \) where \( t_i \) is waiting on \( t_{i+1} \) (\( i = 1 \) to \( n \)) and \( t_n \) is waiting on \( t_1 \).

Deadlock in a distributed system is more difficult to detect because the algorithm needs to acquire a global system snapshot, which because of message delays is never entirely accurate.
2. (20 pts) **Distributed Systems**

(a) (5 pts) Describe the logical clock scheme for keeping time in a distributed system with causally-related events.

**Solution:**

i. Each processor $i$ maintains a logical clock $lc_i$.

ii. Whenever an event occurs locally, $lc_i = lc_i + 1$.

iii. When process $X$ sends a message to $Y$, it also sends a *time stamp* $lc_x$ on the message.

iv. When $Y$ receives this message from $X$:

   if $lc_y \leq lc_x$

   then $lc_y = lc_x + 1$  

   (Y’s clock was behind X’s)

   else do nothing

(b) (15 pts) Classify each of the following statements as true or false and *briefly* justify your answer.

i. (5 pts) An event $a$ that happened before event $b$ in physical time will occur before $b$ in logical time as well.

   **Solution:** False. If $a$ and $b$ on different processors with independent local clocks, one clock may indicate a smaller time stamp for a given real time.

ii. (5 pts) If each process increments its logical clock value by a different amount (rather than incrementing the logical clock value by 1 on each event), the logical ordering of causally related events would not hold.

   **Solution:** False. The increment does not affect the algorithm since sent messages will still be received at a later time and thus the relative ordering stays the same.

iii. (5 pts) If the delay of a message sent from process $x$ to process $y$ is always twice as much as the delay of a message sent from process $y$ to $x$, the logical ordering of events scheme will not work.

   **Solution:** False. Time stamping happens when the message is received and is thus independent of network delays.
3. (20 pts) Distributed Coordination.

(a) (2 pts) For what types of work loads does token passing to send messages or manage critical sections work well? not work well?

Solution: High loads where everyone wants the token yield good performance for these algorithms, since one message to send the token to a neighbor is all the overhead that is incurred. Lightly loaded systems will not work as well since the token passing will incur overhead even though no one wants to use it.

(8 pts) In those situations in which it does not work well, design a mechanism to detect and improve this situation.

Solution: The simpliest way to adjust for a lightly loaded system is for participating processes to timestamp every time they see the token, and slow the token down if they are seeing it to frequently. If the last time, the token came by this process is the expected time to send the message around the ring plus some fudge factor, then hold the token for some period.

(b) (2 pts) In the Bully Election Algorithm, how many election messages may be sent?

Solution: Each process can try to elect itself simultaneously, which would result in \( n(n-1)/2 \) initial “elect me” messages from lower to higher numbered processes, and an additional \( n(n-1)/2 \) messages back to from higher to lowered numbered processes that say “I am higher, make me coordinator,” yielding \( n(n-1) \), or an \( O(n^2) \) algorithm.

(8 pts) Modify this algorithm to prevent the lower numbered processes from holding an election when an election is going on at a higher level.

Solution: When process A tries to elect itself coordinator, it sends an “elect me” message to the higher numbered processes and an “election is happening” message to the lower numbered processes. Now when a process receives an “election is happening” message it waits some time \( T \) to here who is the new coordinator. If no message comes, it tries to elect itself. This solution in the worst case is no better than the Bully solution, but in the expected case should be better.

4. (20 pts) Memory Management. Compare the issues of storage allocation in a file system with that of main memory allocation. What are the similarities and differences?

Solution: One of the main, and important similarities between the two is the tradeoff between external and internal fragmentation. The design of choice in both systems is to prefer a small amount of internal fragmentation by making the data discrete via the very similar techniques of paging in memory or indexed block layout on disk, rather than using static or dynamic allocation in memory and contiguous layout on disk. Paging and indexed layout split the data into discrete elements (i.e. a page or a block) and then use tables to look up the requested offset. Similarly, static and dynamic contiguous allocation are similar to contiguous file layout, since the techniques place the data contiguously, all lookups are offsets from the first position of the data, and if the data grows bigger than the available space, it must be moved. For both we need techniques like First-Fit, Worst-Fit, or Best-Fit, to find blocks in which the data fit and compaction to reclaim fragmented space.

A difference between the two is that virtual memory needs a backing store to work properly, thus supporting a partial image of the process in memory. Of course, that backing store is the
disk and it must contain a complete version of anything it stores (e.g., files, processes, etc.). The costs between the two systems are also different. RAM has one fixed overhead; disk has 3 components: seek time, rotational delay, and latency, and thus benefits from optimizations on the first two, which are unnecessary in RAM.
5. (20 pts) **Synchronization.** Solve the following variation of the candy shop problem with a monitor with Mesa-style semantics. Write a routine **Enter** for the candy shop, in which a customer enters, takes a number (assume an infinite supply), and waits until a sales person calls their number. Write a routine **Service** in which a sales person waits on a customer based on the order of arrival. If no customer is in the store, the sales person waits until one takes a ticket, and then services him or her.

**Solution:**

```cpp
class CandyShop {
    public:
        void Enter(); void Service();
    private:
        int number; // ticket number
        int serve; // current number being waited on
        Lock lock; // control access to shared variables
        CVar server, ticket;
    }
CandyShop::CandyShop(int n) {
    number = 0;
    serve = 0;
    lock = FREE; // lock available
}
CandyShop::Enter() {
    lock.Acquire();
    number++;
    ticket.Signal(); // Signal that a ticket has been taken
    server.Wait(); // Wait for a server to call your number
    lock.Release();
}
CandyShop::Service() {
    lock.Acquire();
    while (number == serve) { // If no customer is in the store,
        ticket.Wait(); // wait on ticket.
    }
    serve++;
    // Wait on a customer
    server.Signal();
    lock.Release();
}
```