THE SYSTEMS QUALIFYING EXAM
SPRING 2002

5 Questions, 100 points

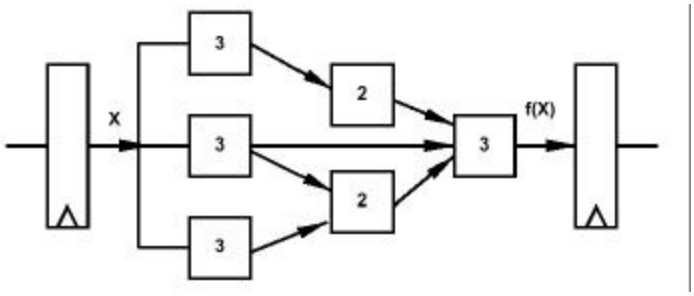
This exam consists of five (5) questions on five (5) pages.
You have 2 hours and 30 minutes.

1. [20 points] The robots assembling the International Space Station need to synchronize their actions in the following manner: certain joints on the solar-panel assembly consist of *n* structural elements (beams, other metal parts) which are welded together at the point where they meet. Suppose that *n+1* robots cooperate on this task: *n* to manipulate structural elements and 1 to weld them together. For the weld to hold, all *n* of the structural elements need to be in place before the welder starts to attach them together. These robots will need to repeat their actions *k* times, where *k* denotes the number of joints being welded.

Modeling each robot as a concurrent process in a shared memory space, write a *monitor* implementing the necessary synchronization. Also provide a skeleton for the *structural* process and the *welder* process, so that we can see exactly how the monitor entry points are called.

Note: people who have forgotten the monitor syntax often ask whether code using Java synchronized variables or general semaphores will be accepted. We would prefer to see monitors, because the semantics of a monitor are well-specified (assume the "signal-delayed" discipline, and that the signaled queue has priority over the entry queue). We will accept Java or semaphore-based solutions, but we do hope to see elegant, "obviously correct" solutions. The further you stray from that, the less likely you are to get the maximum credit.

2. The following diagram shows blocks of combinational logic that compute *f(X)* given X. The numbers in each block denotes the delay of that block in nanoseconds.



Blocks with a *vee* correspond to positive edge-triggered flip-flops. Assume, for simplicity, that flip-flops do not introduce any additional delay. You can only introduce flip-flops on the arrows shown in the figure.

a. [5 points] What is the latency of computing *f(X)*?

b. [5 points] Explain the role of the flip-flops shown in this circuit. Why are they needed?

c. [5 points] *Throughput* is related to the latency as shown you computed in part (a), but in a pipelined circuit, end-to-end latency is not the only factor that determines potential throughput. Briefly explain what other factors limit throughput for the circuit shown above.

d. [5 points] Additional flip-flops can be added to the circuit to improve its throughput. What is the best possible throughput one can achieve by adding flip-flops, and where should they be placed?

3. Provide concise, short answers to the following questions:

a. [5 points] The Internet is a packet-switched network with a *best-effort* delivery policy. Routers on the Internet may drop a packet at any time for any reason. In at most three sentences, discuss the ramifications of this design decision on the design of the network stacks at the communication endpoints, that is, non-router nodes at the edges of the network.

b. (b) [5 points] A team of network researchers proposes to re-architect the Internet to replace best-effort delivery with guaranteed delivery. They suggest modifying the routers to operate by using non-volatile storage and hop-by-hop acknowledgements. Under their proposed scheme, each hop (router on a path) receives a packet, stores it on disk or in non-volatile memory, and notifies the previous hop that the packet has been received. Each packet is thus stored in a persistent memory at all times. Even if a router crashes and restarts, their design allows these stored packets to be recovered and retransmitted. The researchers claim that, when using their scheme, a packet will not be lost in the network even if a router experiences a power failure or a crash/reboot occurs. This eliminates the need for packet-by-packet acknowledgements between the communication endpoints. They claim that their approach will greatly simplify TCP and improve performance because lost packets will be recovered more quickly. Do you agree with this assessment? Briefly describe why or why not.

c. [5 points] To reduce the amount of physical memory devoted to each application, the OS can place common library code (e.g. functions like *printf*) into physical memory locations that are shared by multiple applications. Explain how shared libraries can be implemented using the virtual memory system, assuming a normal (not an inverted) page table. Don't ignore protection issues or compilation issues, *but keep your answer as brief as possible.*

d. [5 points] For some applications, researchers have found it advantageous to introduce a small (1-8 entry), fully associative cache, called a *victim cache*, which stores the cache lines that have recently been evicted (i.e., replaced by other data) from the cache. This victim cache is introduced as another level in the memory hierarchy between the cache and physical memory: data not found in the cache may still be found in the victim cache for a short period of time, permitting faster handling of what would otherwise be costly cache-misses. Explain which types of cache misses (conflict, capacity, cold) this would help mitigate and why.

4. A. Concurrency Control. Consider the following schedule of transactions T1, T2, and T3 working on database objects A and B. (Rx(y) means that transaction Tx reads object y, Wx(y) means that transaction Tx writes object y, and Cx means that transaction x commits.)

S = BEGIN R3(A) R2(A) R1(B) W2(B) R1(A) W3(A) C2 C1 C3 END

The following questions ask where certain events may occur in this schedule. Assume that two-phase locking is used. Assume that a transaction can either obtain a shared lock or an exclusive lock on an object, but not both. Express your answers by giving the two events that must precede and follow the event in question. For example, transaction T1 must lock object A between BEGIN and R1(A).

a. [1 point] Where can T2 lock B?
b. [1 point] Where can T1 unlock A?
c. [1 point] When does T2 force the log to disk?
d. [2 point] Is this schedule conflict-serializable? If so, why? If not, why not?

B. Consider the following schema (keys are underlined):
Product(pid, name, type, mfgr, price), Buys(cid, pid), Customer(cid, cname, age, gender)

a. [2 points] Write the following query in relational algebra: "Find the names of all customers who have purchased at least three different products."
b. [2 points] Write the following query in SQL: "For each customer, list all the products that they have purchased and how often they have purchased them."

C. There are two types of indexing methods for relational database systems: tree-structured indexes such as B+-trees (or variants thereof) and hash-based indexes such as extendible hashing. Consider a relation R(A,B) with attributes A and B.

a. [2 points] In at most two sentences, describe one significant advantage of a tree-structured index over a hash-based index.
b. [2 points] In at most two sentences, describe one significant advantage of a hash-based index over a tree-structured index.
c. [2 points] Assume that you want to join two relations R1 and R2 on a common attribute A. When would you prefer a hash-join over a sort-merge join? Describe one specific scenario.

D. a. [2 points] Assume that the system scans the same very large relation several times. Discuss the advantages and disadvantages of LRU versus MRU as buffer manager replacement policy in at most three sentences.
b. [3 points] Assume that a buffer manager implements the no-steal and no-force policies, that is, the buffer manager does not allow pages to be written to disk before transactions that have written to the page have committed (no-steal), and the buffer manager does not necessarily flush all updates by a transaction to disk before the transaction commits (no-force). Assume that we have a recovery manager based on write-ahead logging. Describe the structure of an update log record under these assumptions.

5. [20 points] You have been hired by the Large-Concurrent-Systems-R-Us Company to review their code. Below is their **atomic_swap** procedure. It should take two stack data structures as arguments, pop an item from each, and push the items onto the opposite stack. If either stack is empty, the swap should fail and the stacks should be left as they were before the swap was attempted. The swap must appear to occur atomically – another thread performing a concurrent swap should not be able to observe that an item has been removed from one stack but not pushed onto the other one. In addition, the implementation must be concurrent – it must allow multiple swaps between unrelated stacks to happen in parallel. You may assume that

- stack1 and stack2 are valid pointers to stack objects,
- stack1 and stack2 never refer to the same stack,
- each stack has a 32-bit, unique identifier, accessible through *stackptr->id*, and
- the pop operation checks for an empty stack and returns NULL if the stack is empty.

Here is the current implementation. Users complain that the code is buggy.

```
extern Item *pop(Stack *);     // pops an item from a stack
extern void push(Stack *, Item *); // pushes an item onto a stack
extern void P(Semaphore *s);  // semaphore wait
extern void V(Semaphore *s);  // semaphore signal


void atomic_swap(Stack *stack1, Stack *stack2) {
      Item *item1;
      Item *item2; // items being transferred

      P(stack1->lock);
      item1 = pop(stack1);
      if(item1 !=  NULL) {
            P(stack2->lock);
            item2 = pop(stack2);
            if(item2 != NULL) {
                  push(stack2, item1);
                  push(stack1, item2);
            }
      }
      V(stack2->lock);
      V(stack1->lock);
}
```

a. [1 point] One of the problems with the given code is that it is prone to deadlock. Describe the circumstances under which the given implementation can deadlock.
b. [4 points] Identify two other, independent problems with this implementation.
c. [5 points] Suggest a deadlock-avoidance strategy appropriate for this kind of code, and indicate which of the four "conditions for deadlock" are avoided when that strategy is employed.
d. [10 points] Provide a correct, deadlock-free implementation that fixes all of the problems seen in parts (a) and (b). As in the original code, your solution should use the pop and push operations on stacks, and P (wait) and V (signal) operations on semaphores.