4 problems. 70 points. *Closed book. Closed notes. No electronic device.* *Write your name above.*

1. [20 points] A byte-addressable single-level paging system has 48-bit virtual address, 32-bit physical address, and page size of 16 KB. Page table entries are aligned to 32-bit addresses (i.e., the least significant 5 bits are zero). Each page table entry has the following fields: page number; 1-bit "present" field; 12 bits for protection/accessed/etc.

a. Draw the page map table for a process. Show the number of entries, the fields and their sizes.

b. The hardware has a TLB of 6 entries managed with LRU replacement. Draw the TLB, showing its fields and their sizes. Indicate which part of the TLB is asociatively searched.

c. A process is allocated three physical pages, numbered 20, 21, 22. Initially, virtual page 0 is mapped to physical page 20 and no other virtual page is mapped to a physical page. The process makes the sequence of memory accesses shown below in the first column; entry i-R (i-W) means page i read (write). Assume LRU replacement.

In the second column, indicate the virtual pages in physical memory after the access at the left is over.

Give the number of page faults and the number of disk page transfers at the end.

Access	virtual pages in physical memory
0-R	0
1-R	
9-W	
0-W	
6-R	
1-W	
4-R	
9-R	

2. [20 points] This question concerns adding *medium-term scheduling* to GeekOS. Assume GeekOS with semaphores and demand-paging (i.e., GeekOS with projects 3 and 4) and user programs that make use of both features (i.e., calls semaphores and grows stack pages).

Assume you have a `Freeze()` function that (1) intelligently chooses a a user process in the runnable queue or a wait queue, and (2) moves all its virtual memory into a separate "frozen" file, freeing all its physical pages and paging file pages. Assume you have an `Unfreeze()` function that (1) intelligently chooses a user process in the frozen queue, and (2) restores it.

a. Outline how your `Freeze()` function chooses a process to freeze. What extra information (if any) will GeekOS collect for this purpose.
b. Outline how your `Unfreeze()` function chooses a process to unfreeze. What extra information (if any) will GeekOS collect for this purpose.

Be precise and concise. You'll lose points for rambling.

3. [20 points] Solve the readers-writers problem without busy waiting, using **semaphores** and no other synchronization construct (no atomic read-modify-write, no disabling interrupts, no PCBs, no wait/wakeup, etc.).

Specifically, given functions `f()` and `g()` that always return, write down functions `cf()` and `cg()` that can be called simultaneously by multiple threads such that:

1. (`cf()` calls `f()` exactly once) and (`cg()` calls `g()` exactly once).
2. The following holds at any time:
   (no thread in `f()` and at most 1 thread in `g()`) or (0 or more threads in `f()` and no thread in `g()`).
3. (every call to `cf` eventually enters `f`) and (every call to `cg` eventually enters `g`)
4. Allow multiple simultaneous calls to `f()`.

Be neat and clear. You lose points if I can't understand your code in a reasonable time.

4. [10 points] Solve problem 3 with two changes: (a) allow multiple simultaneous `g()` calls, and (b) use **awaits** instead of semaphores (no other synchronization construct, no busy waiting).

Specifically, given `f()` and `g()`, obtain `cf()` and `cg()` such that:

1. Same as in problem 3.
2. The following holds at any time:
   (no thread in `f()` and 0 or more threads in `g()`) or (0 or more threads in `f()` and no thread in `g()`).

3–4. Same as in problem 3.

5. Allow multiple simultaneous calls to `g()`.