Concurrency

1. One H thread and one O thread arrive, increment h_count and o_count, and each end up waiting on h_ready and o_ready, respectively. The 2nd H thread, it calls sem_post for one H and one O. One H thread (either the one that was waiting or the 2nd one) will remain blocked at sem_wait(&h_ready). The other H thread and the O thread end up calling bond() and exiting.
2. Both threads share the same code for the if statement. Regardless of whether an H or an O thread comes last, exactly 3 threads (2 H and 1 O) must be permitted to call bond(). By calling the 3 sem_posts, the fix guarantees that exactly two H and 1 O thread get to go through the last sem_wait and call bond().

```
C/C++
            For the H_thread:
            if (h_count >= 2 && o_count >= 1) {
                  sem_post(&h_ready);
                  sem_post(&h_ready); ← new code
                  sem_post(&o_ready);
                  …
            }
            For the O_thread:
            if (h_count >= 2 && o_count >= 1) {
                  sem_post(&h_ready);
                  sem_post(&h_ready); ← new code
                  sem_post(&o_ready);
                  …
            }
```

3. No, there is no such guarantee. Say H_0 and O_1 are waiting on the final sem_wait. H_1 posts thrice (thus waking up H_0 and O_1, which both call bond), and then gets interrupted just before calling sem_wait(&h_ready) and goes to sleep. H_2 runs and goes through sem_wait(&h_ready) and calls bond. Thus, H_0, O_1, and H_2 bond to form the 0th molecule and break the ordering.

File Systems & Storage

1. 16382+19 (don't forget the "\n" at the end of the line) = 16401 = 4*4096 + 17
2. A write with offset 0, size 16401, and the new line followed by the original contents of xyz.c.

Remote FS

1. NFS guarantees that when a file is closed at C1, all dirty blocks are flushed to the server. However, C2 likely used the cached attributes of my.log (from T0) for the read in T2, and when it checked with S at T2, S had not yet received the close() operation. By T3, when C2 rechecked with S the attributes had changed, and so C2 refetched my.log to get the accurate contents.
2. No. Because NFSv3 is stateless, every request (READ/WRITE) contains all the information necessary for the operation, such as file handle, offset, etc. The client simply retries until the server responds. In a system with a stateful protocol, the server would lose the “session” state, requiring the client to restart the connection and re-acquire the file handle with a new NFS OPEN.

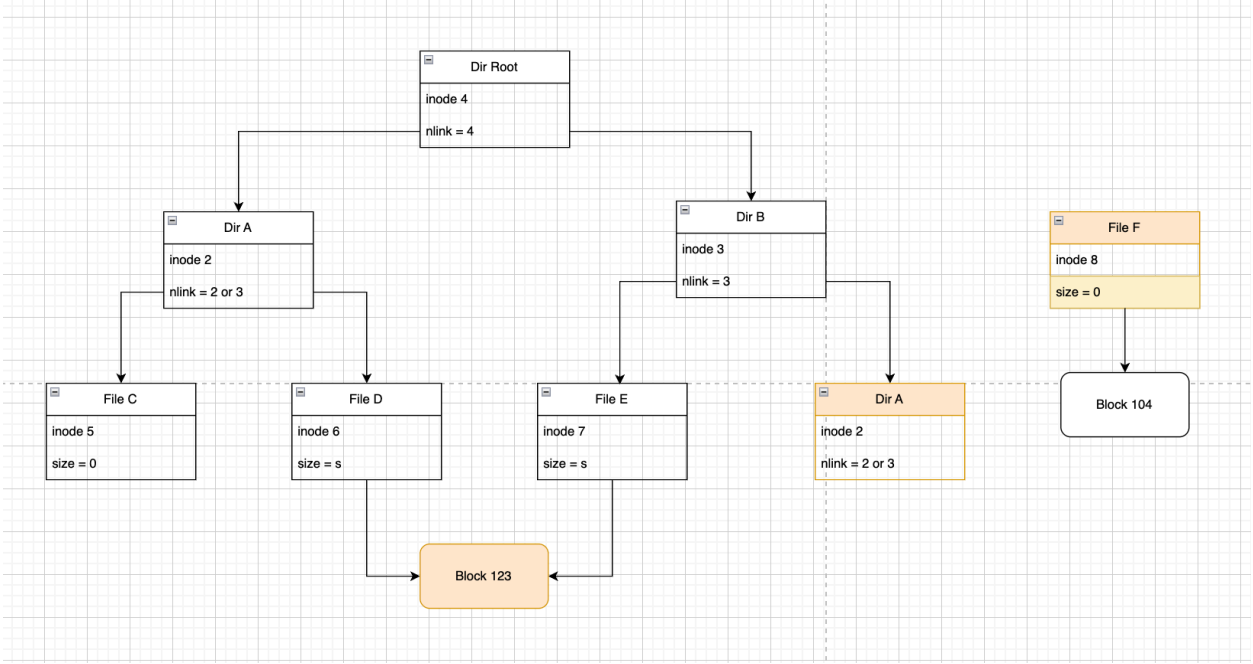
MP3

1. Both functions return dynamically allocated memory that is guaranteed to be contiguous in virtual memory space. However, only kmalloc() returns allocated memory that is also guaranteed to be contiguous in physical memory. When allocating a large amount of memory, kmalloc() may potentially fail because of the unavailability of contiguous free physical memory. MP3’s memory buffer does not require contiguity in physical memory, so we use vmalloc() instead.
2. Setting PG_reserved bit is an indication to the kernel’s memory management system that the page must not be swapped out reclaimed. If unset, MP3’s memory buffer may get swapped out to storage, which would result in a general slow down in the running of MP3.
3. As N increases, the memory consumed by all work processes increases. This results in the overcommitment of the physical memory resource, which leads to increased swapping, which in turn leads to thrashing. Thrashing causes a significant drop in CPU utilization because the system spends more of its time swapping memory pages between RAM and storage. In other words, the CPU spends more time waiting on the pages of its work process to get loaded into memory than actually executing its code.

MP4

1. A file system is consistent when each piece of its metadata accurately matches all other metadata and data in the file system. In other words, all data structures in the file system are valid, coherent, and free of contradictions. Updates to the file system when appending to a file: (1) a new data block Da for the appended bytes, (2) block bitmap update to mark Da allocated (3) inode is updated with a new pointer to Da, new size, and new last-modify timestamp. If only (1) and (3) get persisted to the file system, then

the Da will be marked as a free block in the bitmap even though the inode points to it. This is an inconsistency. It can even morph into a different inconsistency if subsequently Da (as a free block) gets allocated for a different inode; two inodes will then point to Da.

2. Even though journaling prevents any inconsistencies in the file system due to crashes, inconsistencies may get created due to (1) bugs in the file system code, or (2) media corruption that remains undetected. In such cases, a tool like myfsck is needed to make the file system consistent again.
3. 

Missing nlink counts are marked in the figure. Inconsistencies are

1. Dir A appears in two places with the same inode

   Because A appears in 2 places and is an obvious corruption, we can accept both answers (nlink=2 and nlink=3).
2. File D and File E have different inodes but point to the same data block (Block 123)
3. File F is orphaned–it does not exist in any directory
4. File F has size 0 but should be “s”