Objective

A few sample long-form questions to help students prepare for the cs 423 finals. Although this document does not include MCQs—because the in-class pop-quizzes provide many samples—the final exam will contain MCQs. In general, these sample questions are at least as difficult as (if not more difficult than) the ones in the final. Ditto for the pop-quiz MCQs.

Concurrency

This code simulates the bonding of water molecules. There are 2 types of threads: hydrogen (H) and oxygen (O). To form a water molecule (H_20), exactly 2 H threads and 1 O thread must come together. Thus, no thread can call the bond() API until a full set of these 3 threads has arrived. Once a set of these 3 threads has arrived, they must all call bond(), then exit to allow the next molecule to form. We use POSIX semaphores: sem_wait(&sem) and sem_post(&sem); the argument X can be ignored.

Here's the first attempt at the code:

```C/C++
#include <semaphore.h>

sem_t h_ready;     // Initialized to 0
sem_t o_ready;     // Initialized to 0
sem_t mutex;       // Initialized to 1 (Protects shared counts)
int h_count = 0;
int o_count = 0;

void* hydrogen_thread(void* arg) {
    sem_wait(&mutex);
    h_count++;
    if (h_count >= 2 && o_count >= 1) {
        // We have a full set!
        sem_post(&h_ready);
        sem_post(&o_ready);
        h_count -= 2;
        o_count -= 1;
        sem_post(&mutex);
    } else {
        sem_post(&mutex);
    }
```

```
    sem_wait(&h_ready);
    bond();
    return NULL;
}

void* oxygen_thread(void* arg) {
    sem_wait(&mutex);
    o_count++;
    if (h_count >= 2 && o_count >= 1) {
        // We have a full set!
        sem_post(&h_ready);
        sem_post(&h_ready);
        h_count -= 2;
        o_count -= 1;
        sem_post(&mutex);
    } else {
        sem_post(&mutex);
    }

    sem_wait(&o_ready);
    bond();
    return NULL;
}
```

1. There is one scenario in which 3 threads (2H and 1O) arrive but one of them stays blocked forever. Describe the exact sequence of arrivals and executions that would lead up to that situation. Explain which thread gets stuck & where.
2. Rewrite the code (in both the H and O threads, if necessary) to fix this bug. Explain why your fix works. **Hint**: *the code inside the `if` condition needs a small change in both threads.*
3. Does your fix guarantee strict ordering—do water molecules get created by the H and O threads in the order of their arrival? To be specific, say many threads arrive very close to each other, and we number the threads in the order of their first scheduling, i.e., H_0 gets to run at least once before H_1, which in turn gets to run at least once before H_2, and so on; ditto for O_0, O_1, O_2, and so on. Then, does your fix guarantee that the i^{th} water molecule is formed by threads H_{2i}, H_{2i+1}, and O_i? Explain your answer.

File Systems

You are using your favourite editor/IDE to code and test your MP. The file open in your editor is xyz.c, which is currently 16382 characters large; btw, 16382 = 4*4096 - 2. The file is currently stored locally in a file system on your laptop. You notice a bug in your code, and insert an extra line in your code at the very beginning of xyz.c to include the stdio.h header file.

```
#include <stdio.h>
```

1. How many bytes did you insert in that file, i.e., what is the new size of xyz.c?
2. The editor application had already invoked the open() syscall to obtain the file handle to xyz.c and issued multiple read() syscalls with that file handle to display the file to you. What system calls does the editor application make to persist your new changes to storage? Please specify the arguments for the system call—ie, for a write you must specify the offset, size, and the content.

Remote FS

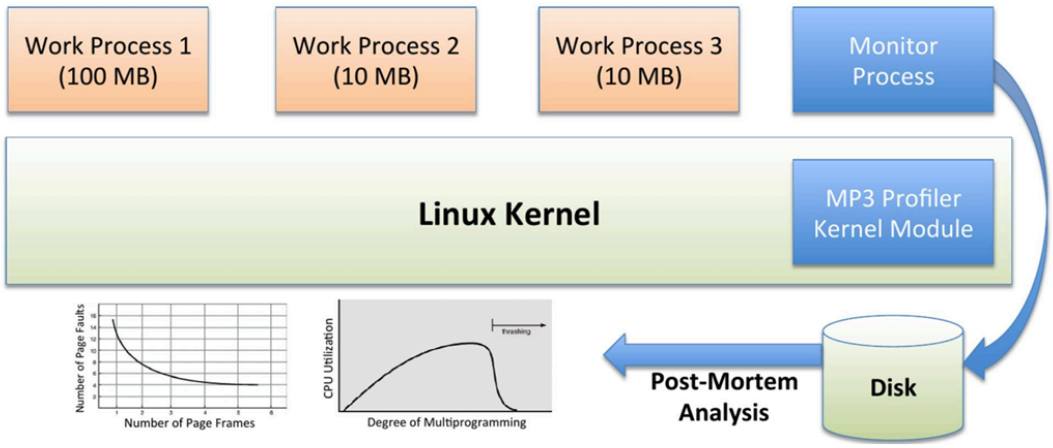
Let us say there are 2 client nodes (C1 and C2) communicating with an NFSv3 server (S). Each client uses write-buffering to improve performance. We know that NFS v3 is a stateless protocol, and to maintain a semblance of order, NFS uses flush-on-close semantics. A developer observes some strange behavior from her application, with the following sequence of events with increasing time:

1. T0: C2 opens an empty file my.log, reads the contents, and sees an empty file.
2. T1: C1 opens the same file my.log, writes the string “Alpha”, and then calls close().
3. T2: C2 opens the same file my.log immediately after, reads the contents, and sees an empty file.
4. T3: C2 waits 60 seconds, opens the my.log again, reads the contents, and now sees “Alpha”.
5. T4: C1 opens my.log, writes “Beta”, but the NFS server S crashes and reboots before C1 calls close().

1. Explain why C1 saw an empty file at T2, but the correct data at T3. In your answer, explain flush-on-close semantics, and explain the role of client-side attribute caching.
2. When S reboots at T4 and loses all information in its RAM, does C1 need to perform a “re-login” or a re-open (of my.log file) handshake with S to continue its session? Contrast this behavior with a system that’s using a stateful protocol, like NFSv4 or SMB.

MP3

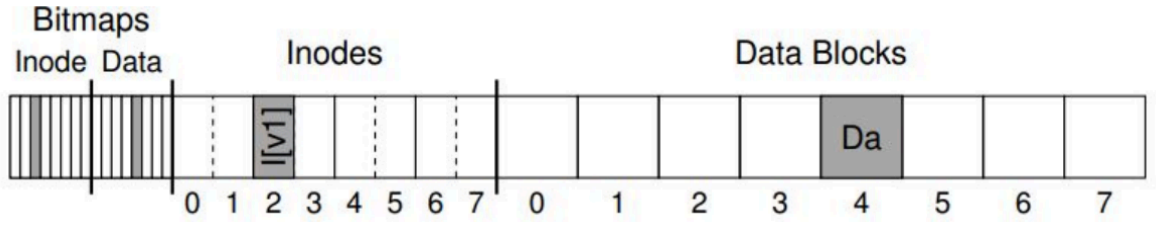
In MP3, we implemented a lightweight tool that captured information about registered work processes, and stored them into a memory buffer. We used this to perform 2 case studies to analyze page fault rates and CPU utilization.



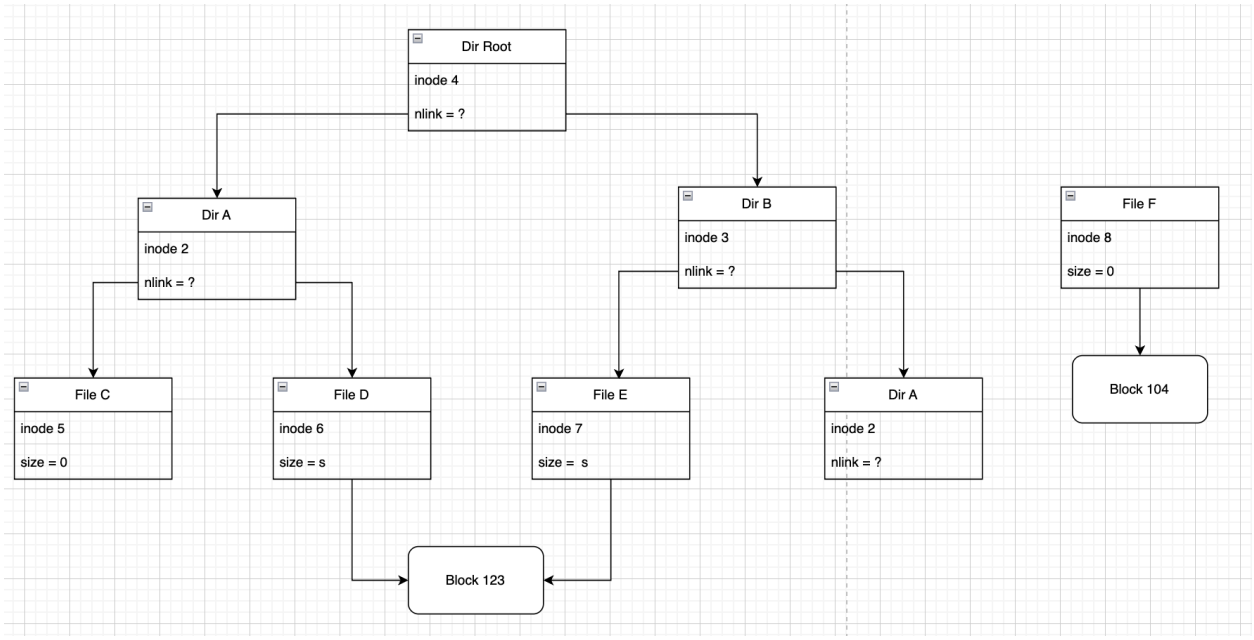
1. MP3 required the use of the vmalloc (instead of the kmalloc) kernel function to allocate the memory buffer used to store the profiled information about the registered processes. Explain this preference, especially when we allocate a large amount of memory.
2. MP3 also required the setting of the PG_reserved bit on this allocated buffer. Describe one potential problem that you would encounter if that bit was not set as instructed.
3. In case study#2, we created N instances of the work process and plotted CPU utilization on the y axis with increasing N on the x-axis (as the degree of multiprogramming). We noted a drop in CPU utilization once N increased past a certain point. Please explain that reason in a few sentences.

MP4

1. Provide a definition of file system consistency. And explain how an inconsistency can occur using the example of appending to a file (pic below).



2. Is a tool like myfsck needed for a modern file system that uses journaling? Explain your answer.
3. Consider a file system with block size “s”. Please fill in the *nlink* counts in some of the inodes and point out all inconsistencies (that myfsck would discover) in this diagram.


Dir Root
inode 4
nlink = ?
Dir A
inode 2
nlink = ?
Dir B
inode 3
nlink = ?
File F
inode 8
size = 0
File C
inode 5
size = 0
File D
inode 6
size = s
File E
inode 7
size = s
Dir A
inode 2
nlink = ?
Block 104
Block 123