1. a) No. Trap instruction occurs at user level, and it is responsible for moving process from user mode to kernel mode.

   Correct Solution

   No. Trap instruction must also needs to executed called at user level (e.g. system call), to move process from user mode to kernel mode

   b) This question is omitted. It is not covered in class

   c) No. If the type of access is read for both threads then there will be no concurrency error

   Correct Solution

   No. If both threads are reading the shared variable then it's fine, and furthermore, it's also fine when only one of two threads are updating the shared variable and reading whichever shared variable, before and after update, by the other thread doesn't matter

   d) No. Limited direct execution means running a process in CPU but with limited permission, and it can be thought as baby proofing CPU, so bad code won't harm the system.

   Correct Solution

   Limited direct execution means a process can be run on CPU without controller, but with condition where user-level process cannot execute previleged instructions.

   e) No. Indexed based system uses block pointers in inode, and block pointer can be pointing data blocks in the data region.

   Correct Solution

   No. Each data block in indexed based file system uses block pointer in inode, and a block pointer can point to any data blocks in data region, so external fragmentation cannot occur

   f) No. Extent-based file system requires only extent + length to get to a particular byte in file, and this differs from indexed-based system which uses many indirect pointers in between (which adds disk access).

2. a)
   - Process counter
   - Process state
   - Process ID
   - Process Register

   b) Before going from user to kernel mode, the following must be saved

   - User/Process Register
   - Stack pointer
   - Frame pointer
   - I/O Information

- Process ID
- Process State

so that upon executing return-from-trap instruction, the process can resume where it has left.

Correct Solution

Before going from user to kernel mode, the following must be saved

- Registers
  - Stack pointer
  - Program counter
  - User register
- Kernel State

so that upon executing return-from-trap instruction, the process can resume where it has left.

c) This question is omitted. It is not covered in class

3.
- `b) The mutex is held by this thread while it is blocked in pthread_con`
- `c) The mutex is held by this thread whenever it checks the value of writing`
- `d) The mutex is held by this thread when the while loop terminates.`

Correct Solution

- `c) The mutex is held by this thread whenever it checks the value of writing`
- `d) The mutex is held by this thread when the while loop terminates.`

Notes

- `pthread_cond_wait` only puts thread with mutex to sleep, but it doesn't hold it

4. a) 4. one for the root directory, one each for `a, b` and `c`

b) 3. one for the root directory, one for `a`, and the other for `b`

c) At least 1 single indirect block. This is because inode can only store 12 data blocks using direct pointers. The next 1022 many 4KB blocks are pointed by pointers in single indirect block

d) No other data blocks.

Correct Solution

a) 1. 4 inodes are needed: one for the root directory, one each for `a`, `b` and `c`. But, with 64 bytes per inode, all can be fit in one inode block

b) 3. one for the root directory, one for `a`, and the other for `b`

c) At least 1 single indirect block. This is because inode can only store 12 data blocks using direct pointers. The next 1022 many 4KB blocks are pointed by pointers in single indirect block. This is true because with size of 4 byte per pointer, single indirect block can hold 1024 pointers.

d) 1. This is the 4KB data block that is read and loaded into buffer

5. a)
   - `a) inode bitmap`
   - `b) data block bitmap`
   - `c) new directory inode`
   - `d) new directory data block` (because of directory entry containing . and ..)
   - `e) existing directory inode`
   - `f) existing directory data block`

b) data block (because directory entry is added), size (because directory entry is added), time accessed (because existing directory is read before adding directory entry), time modified (because contents in directory is changed).

c) **Inode Bitmap and Data Block Bitmap**

   - `b) Data leak`
   - `c) Inode leak`

   New Directory Inode

   - `a) No inconsistency`

   New Directory Inode

   - `a) No inconsistency`

   Inode Bitmap, Data Block Bitmap, Existing Directory data, New Directory inode, and New Directory data

   - `e) inconsistent inode data`

   Inode Bitmap and New Directory inode

   - `f) Something points to garbage`

   New Directory inode, Existing Directory inode, and Existing Directory data

   - `d) Multiple file paths may point to same inode`
   - `f) Something points to garbage`