1.
   - False
   - True
   - True
   - True

     False

     Notes
     - Hard link is a directory entry. Not inode.
   - False
   - False

     True

     Notes
     - Moe didn't see the word 'never result'
   - False
   - Omitted. Topic not covered in class

2. Omitted. Question not in scope of test

3. Omitted. Question not in scope of test

4. a)
   - 5
   - 36
   - 2
   - 3

   Correct Solution
   - 5
   - 37 (36 inode and data blocks + 1 indirect block)
   - 2
   - 3

   b)
   - the number of links in `emptydir` would change from 1 to 2
   - `emptydir`'s data block would add directory entry for `bdir`
   - `bdir` would have 1 data block of size 4 KiB
   - `bdir` would have link count of 2
   - `bdir`'s size field in inode is 4096
   - `bdir`'s data block would be ticked as allocated in data bitmap
   - `bdir`'s inode would be ticked as allocated in inode bitmap

c)
- Data block (bdir)
  - Is done first so there won't be inconsistency when crash occurs
- Inode bitmap (bdir)
- Inode (emptydir)
  - Are done so multiple inodes won't be pointing to where the inode of `bdir` is
  - They minimizese damage done to existing file system
- Data bitmap (bdir)
- Inode (bdir)
  - Are done so multiple inodes won't be pointing the same data block

Correct Solution

- Data block (bdir)
  - Is done first so there won't be inconsistency when crash occurs
- Inode bitmap (bdir) + Data bitmap (bdir)
- Inode (emptydir)
- Inode (bdir)
  - Are done so multiple file paths won't be pointing to where the inode of `bdir` is
  - They minimizese damage done to existing file system

5. a) An interrupt is raised, and a program will fail

   b) When the number of data that's being put to disk is large, then it is not a problem.
   But, when what's being written to disk is small in size, then per small amount of data, a data block + extent is required.
   This generates internal fragmentation, and since disk arm has to move back and forth between data block and extent in inode to read multiple data blocks, the speed would be slow.

   c) No. The only time when a program moves to a blocked state is when a signal or a system call has been made. This happens while it's executing instruction, or when in RUNNING state.

   Notes

   - Realized that I forgot to see 'READY' state

   d) Omitted because topic not covered in class

6. When a fork system call has been made what happens is

   - an interrupt is made
   - User-register of a program is saved to kernel stack

- The process is moved from user mode to kernel mode
- Child process is generated (done by copying PCB block)
- Execute Return-from-trap instruction
- Resume parent and start child process from fork