Assignment 5

(Each question is worth 20 points)

1. A toy disk has two tracks with 10 blocks each and rotates counter-clockwise. Blocks 0-9 are on the inner track and blocks 10-19 are on the outer track. Suppose the track-to-track move time for the read-write head is long enough for 1.7 blocks to rotate past the head. Draw a disk and label the blocks such that there is the minimal disk skew that allows us to read block 10 after block 9 is read with minimal delay.

2. Suppose we wish to support hard links in an indexed file system. Write pseudocode to show how an implementation of hard link creation and removal would work. Assume the existence of the following filesystem functions:
   - get_inode(path_to_file) – Given a file path, **return** the block number of its root i-node.
   - read_block(block_num) – Read data from the specified block, **return** the corresponding data from block number block_num. Assume data returned for an i-node can be used as an inode object (an instance of an i-node data structure).
   - write_block(block_num, block_data) – Write block data to specified block.
   - get_directory(path_to_file) – Given a file path, return the containing directory of a file path or the current directory if only a filename is given.
   - get_filename(path_to_file) – Return the filename portion of a file path.
   - dir_add(dir, filename, inode_blk) – Add filename to the specified directory with a root i-node block of inode_blk.
   - dir_remove(dir, filename) – Remove filename from the specified directory.
   - inode_release(inode_blk) – Given the block number of the root i-node associated with a file, return all i-nodes associated with this file to the free i-node pool and all data blocks to the free data block pool.

   You may propose a field in the i-node metadata that would let the operating system know when a file should be deleted.

   Implement:
   - **hardlink(existingFilePath, targetLinkPath)** - Create a hard link to an existing file path
   - **remove(targetLinkPath)**

   Hint:
   - Hard links to a same file share the root inode of the file

   You do **not** need to worry about error checking, also **assume**:
   ```
   // i-node has a reference count field refcount that
   // tracks number of references to a file and
   // is initialized to 1 when it is first created.
   ```

3. Assume the file allocation table below has a single directory file in block **5** that contains files **A** and **B** starting on blocks **2** and **1** respectively.

Entry	0	1	2	3	4	5	6	7	8	9	10
Pointer		6	3	4	8	-1	7	-1	9	-1	

   a. List the blocks associated with:
      - the directory
      - file A
      - file B

   b. A two-block file starting on block **10** is added to the directory. Show the resulting file allocation table.

4. An i-node file system support **10 direct** blocks, **3 single indirect** blocks, and **3 double indirect** blocks. To simplify the math, assume that all indirect block types (single or double) have room for **100** pointers.
   a. Assuming **1 KB** blocks, what is the largest file supported by the system?
   b. Write pseudocode to implement the following function (you do not need to worry about error checking):

      // Given block number of the root inode of a file and a byte offset into the file,
      // return the block number of the data block containing the byte.
      block_num seek(block_num root_inode, unsigned long bytes)

      Assume:
      a. A read_block function as in question 2.

      b. The root node structure has 3 arrays containing the block numbers for the direct, single indirect, and double indirect blocks:

```
#define DIRECTN 10
#define SNGL_INDIRECTN 3
#define DBL_INDIRECTN 3
#define NUMBLOCKS 100
block_num direct[DIRECTN];
block_num indirect[SNGL_INDIRECTN];
block_num doubleIndirect[DBL_INDIRECTN];
```

      c. Each **indirect** i-node structure (Single and double indirect nodes) has an array of block numbers as:

```
block_num blockNumber[NUMBLOCKS];
```

5. Is the following disk operation idempotent? Truncate file foobar.txt to 2096 bytes. Justify your answer.

6. Suppose a disk block free list is implemented as an array of **32 bit** integers. Write pseudocode to implement the following function:

```
// Given a bitmap stored in an array of words (1 word has 32 bits), find the bit_idx'th bit
// and set it to 1 (new_state = true) or 0 (false).
void bit_state(uint32_t words[], int bit_idx, bool new_state)
```

7. Suppose hardware supports a **single** timer device. Write the following pseudocode:

```
/*
 * bool setalarm(timestamp, processId)
 * A system call uses this interface to set an alarm for calling process
 * (processId) at specified time.
 *
 * Assume access to any data structures you may need.
 * (real interface would be more complicated with register or stack arguments
 * specified)
 */

/* void alarm()
 * Interrupt service routine that is called when the hardware timer expires.

 * Sets the next alarm (if any).  Alarms are set on this hardware by writing
 *    the alarm deadline timestamp to memory location 0x3A00.  Alarms
 *    To enable the alarm after setting a deadline, set bit 4 of memory location
 *    0x3A02.
 *
 * Sends a SIGALRM to a specified process using function
 *    sendsignal(sig, processId)
 */
```

Important:

- You **must** support multiple alarms using the single timer device.
- You need not worry about setting two alarms for the same time.

You may assume the use of any abstract data types you wish, just state how they are initialized.