Exercise File Systems:

We have a machine with a 32-bit mono-processor and need to implement a file system for a UNIX operating system having a cache for blocks which allows to read (bread) and write (bwrite) disk blocks.

The file system that needs to be implemented has the following structure in disk:

1 block	**1 block**	**1** block	**n** blockes
Superblock	I-node block	Name Block	Data blocks
0	1	2	3 N

The requirements of the file system are the following:

1. The block size is 1024 bytes (1 KB).
2. The file system has a total of 800 blocks
3. There are only files (which belong to a root directory common for all of them). There can be, at most, 50 files.
4. The superblock takes up a disk block and it is stored in the first block (block nr. 0).
5. The superblock will be used to manage free and in-use blocks and free and unused i-nodes. It has a 50 characters' vector for i-nodes and a 800 characters' vector for blocks where each entry is set to 0 if the i-th block is free and '1' if it is in use.
6. Each i-node occupies 20 bytes
7. Each i-node contains among other things: the type (regular, symbolic link, etc.) and one simple-indirect block, which points to the blocks assigned to the file. Each pointer block occupies 4 bytes.
8. The name block contains an array of 50 entries, each of which corresponding to a string of 20 characters representing a file name.
9. Each file can be open by only one process and has a private file pointer for the process.

Answer the following questions:

a) Complete the design of the disk structures and state clearly the fields that the superblock and an i-node would require.
b) Design the in-memory structures which will allow users to work with the described file system.
c) Implement the pseudocode of the function that handles the bmap blocks.
d) Implement the pseudocode of the functions that handle the i-nodes: ialloc, ifree and namei.

a) **In-disk metadata structures:**

Superblock:

- int magic_number = 0x22223
- int blockSize = 1024 // R1
- int num_blocks = 800 // R2
- int inode_size = 20 // R6
- int num_inodes = 50 // R3
- char blocks_map[800] = {'1','1','0','0',...} // R5
- char inodes_map[50] = {'0','0',...} // R5
- char padding[1024-20-800-50]

Inode

- int block_ind // R7
- char type = REGULAR // R7
- int size = 0
- char padding[20-4-1-4]

int indirect_block [1024/4]

b) **In-memory metadata structures:**

Superblock sb
Inode inodes[50]
char names[50][20] // R8
Session sessions[50]

Where a session is a structure that includes the following fields:

- Int position // R9
- char opened // R9
- Int PID // R9

c) Bmap:

```
int bmap ( int inodo_id, int offset )
{
    int block[1024 / 4] ;
    int bid ;

    if (offset > inodes[inode_id].size)
        return -1;

    bid = offset / sb.block_size;
    if (bid > (1024/4))
        return -1;

    bread(inodes[inode_id].block_ind, bloque);
    return block[bid];
}
```

d) ialloc, ifree y namei:

```
int ialloc ( void )
{
  for (int=0; i< sb.num_inodes; i++)
  {
      if (sb.inodes_map[i] == '0') {
         memset(&(inodes[i]), 0, sizeof(inode));
         sb. inodes_map [i] = '1';
         return i;
      }
  }
  return -1;
}
```

```
void ifree ( int inode_id )
{
   sb.inodes_map [inode_id] = '0';
}


int namei ( char *fname )
{
    for (int=0; i< sb.num_inodes; i++)
    {
        if (! strcmp(names [i], fname))
            return i;
    }
    return -1;
}
```