Kavon Cacho
822794135

Assignment 5

SINGLE EXAMINEE AFFIDAVIT

"I, the undersigned, promise that this exam submission is my own work. I recognize that should this not be the case; I will be subject to plagiarism penalties as outlined in the course syllabus."

Student Name: Kavon Cacho
RED ID: 822794135
Date: 04/27/2022

Kavon Cacho
822794135

```
QUESTION 1

createHardlink(srcPath, linkPath) {

    fileName = getFileName(linkPath) //get file name

    directory = getDir(linkPath) //get directory to file

    iNode = getINodeBlockNum(srcPath) //get block num for the actual file's iNode

    addFileToDir(directory, fileName, iNode) //add new file with root iNode

    dataBlock = readDatablock(iNode) //read dataBlock at iNode so we have metadata

    dataBlock.metaData_refCount++ //increment reference count so we know to delete it at some point

    writeDatablock(iNode, bD) //write the data

}

removeHardlink(linkPath) {

    fileName = getFileName(linkPath) //get file name

    directory = getDir(linkPath) //get directory to file

    iNode = getINodeBlockNum(linkPath) //get block num for hard link's iNode

    removeFileFromDir(directory, fileName) //remove hard link's filePath

    dataBlock = readDatablock(iNode) //read dataBlock at iNode so we have metadata

    dataBlock.metaData_refCount-- //decrement reference count so we know it will get deleted

    releaseInode(iNode) //delete

}
```

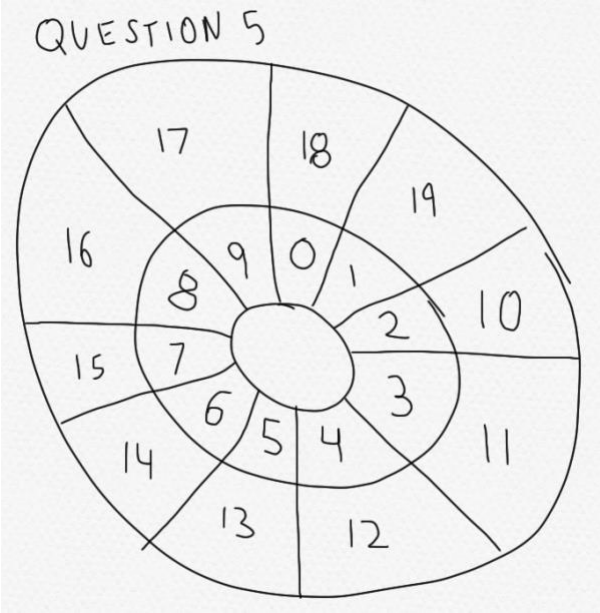
Kavon Cacho
822794135

```
QUESTION 2

A.

4KB / 8B = 512 total pointers
(10 + 3*512^1 + 2*512^2) * 4096 = 2153816064 bytes or about 2 gigabytes

B.

BlockNum findBlockOfOffset(BlockNum rootInodeBlk, unsigned long nthByte) {

    BlockNum targetInode = readBlockData(rootInodeBlk)
    long offset = bytes / 1024

    if (offset < NUMOFDIRECT) { //direct block

        return targetInode.directBlocks[offset]

    } else if (offset < (NUMOFBLOCKS * NUMOFSINGLEDIRECT)) { //single indirect block

        return readBlockData(targetInode.singleIndirect[offset]).blockNumber[offset]

    } else { //double indirect block

        return readBlockData(targetInode.doubleIndirect[offset]).blockNumber[offset]

    }

}
```

```
QUESTION 3

This disk operation is idempotent. This is because if it's a function, we would only need
to call the function once. Calling it again would not do anything if all of the whitespaces
have been already been replaced with asterisks. Hence, it is idempotent.
```

Kavon Cacho
822794135

```
QUESTION 4

int findFreeblocks(int words[], int numOfNeededBlocks) {

    //variables in whole method's scope so it is tracked throughout the whole process, even between blocks
    int currHoleSize, currHoleWordIndex, currHoleBitIndex, targetHoleWordIndex, targetHoleBitIndex = 0;

    for (int i = 0; i < sizeof words; ++i) { //iterate through each word

        for (int j = 0; j < BITSPERWORD; ++j) { //iterate through each bit in each word

            int bit = extractBit(j) //handy helper method that does all of bitmask and bitwise stuff :^)

            if (bit == 0) { //hole
                currHoleSize++
                if (currHoleSize == 1) { //beginning sequence of 0's, store index
                    holeWordIndex = i
                    holeBitIndex = j
                }
                if (currHoleSize == numOfNeededBlocks) { //our target
                    targetHoleWordIndex = holeWordIndex
                    targetHoleBitIndex = holdBitIndex
                    return targetHoleBitIndex
                }

            } else { //hole ends. reset values
                currHoleSize = 0
                holeWordIndex = 0
                holeBitIndex = 0
            }

        }

    }

    return -1 //this is reached when we don't find a hole of the desired size

}
```



Kavon Cacho
822794135

```
QUESTION 6

bool addAlarm(alarmTimestamp, processId) {

    bool minheap[] //used to hold deadlines

    for (int i = 0; i < sizeof(minheap); ++i) {

        if minheap[i] > alarmTimestamp { //if deadline exceeds timestamp

            minheap.pop //remove nearest future alarm time
            minheap.insert(alarmTimestamp) //insert newest timestamp
            minheap.heapify //reorganize minheap after the changes

        }

    }

}

void setoffAlarm() {

    Register* StatusRegister, CommandRegister

    something

}
```