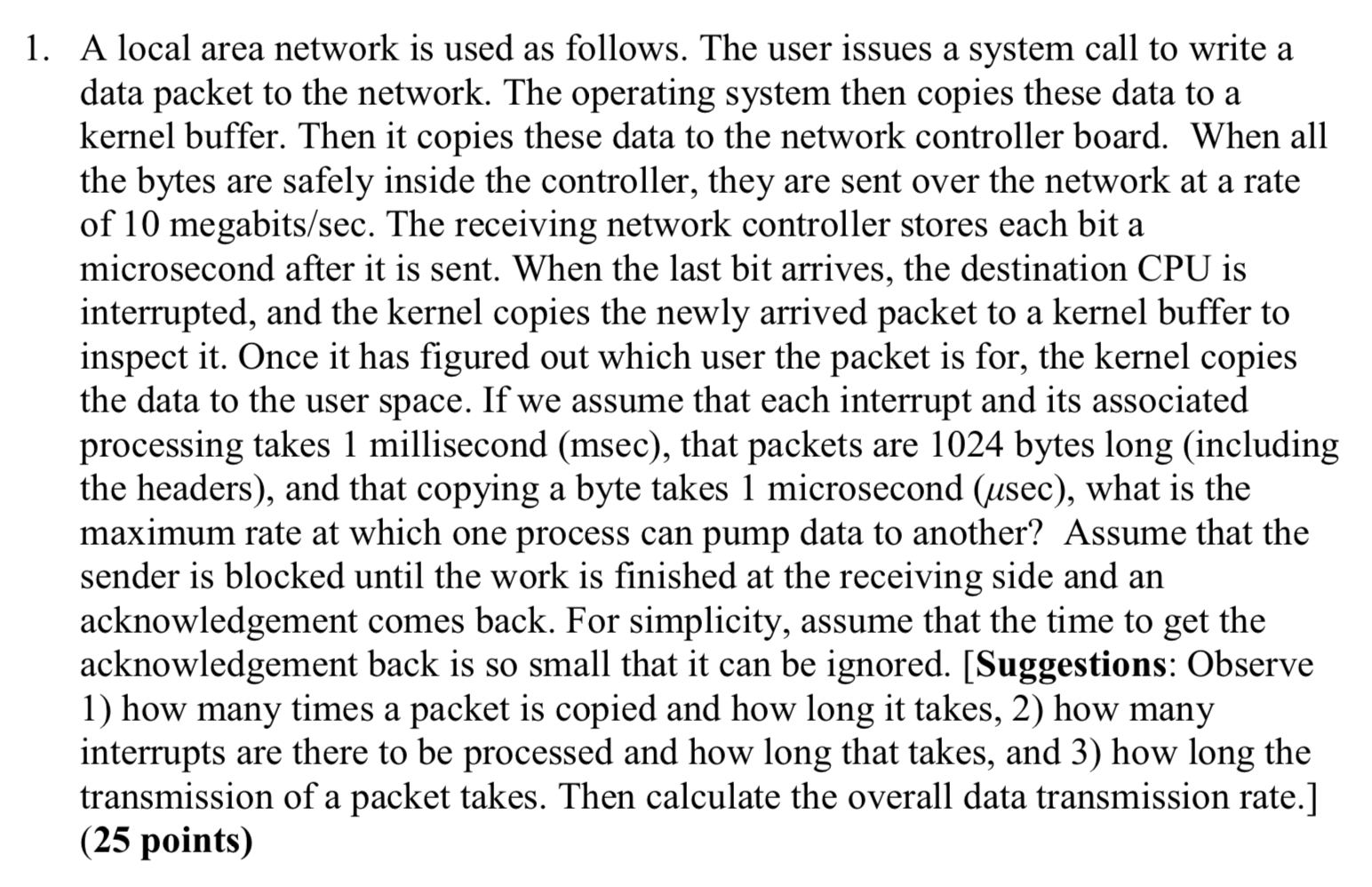
CS 520 Homework 5 | CWID 10430147 | Divyendra Patil | Username: dpatil3  
Date: 11/17/2017



**Solution**:

A packet is copied at below 4 places:

1. Machine 1:
   * 1. User buffer to Kernel buffer
     2. Kernel buffer to Network controller
2. Machine 2:
   * 1. Network controller to Kernel buffer
     2. Kernel buffer to User buffer

Now to copy 1024 Bytes, it takes (1024 x 4)/1000 = 4.096 msec.

There are two system calls:

1. One trap in machine 1 to copy data to network.
2. One interrupt in machine 2 to copy data from network controller to kernel buffer.

And these system calls take 2ms collectively & the Total transmission period takes

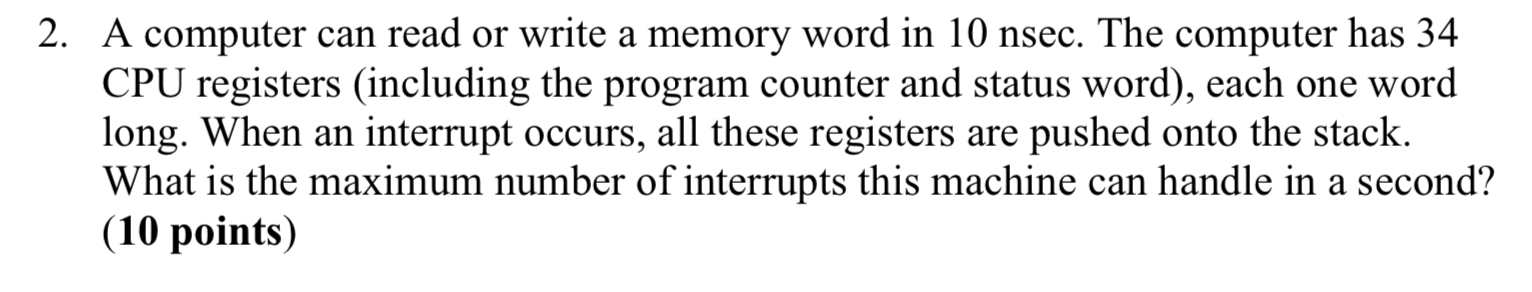
1024 x 8 / 10000000 = 0.8192 msec.

A received bit is stored after 1 microsecond delay. Therefore, there will be 1.024 msec delay for a packet.

So overall for a packet to travel from source to destination, it takes

4.096 + 2 + 0.8192 = 6.9152 msec

Hence, the total data rate is (1024x8x1000)/ 6.9152 = 1,184,637 bit/sec.



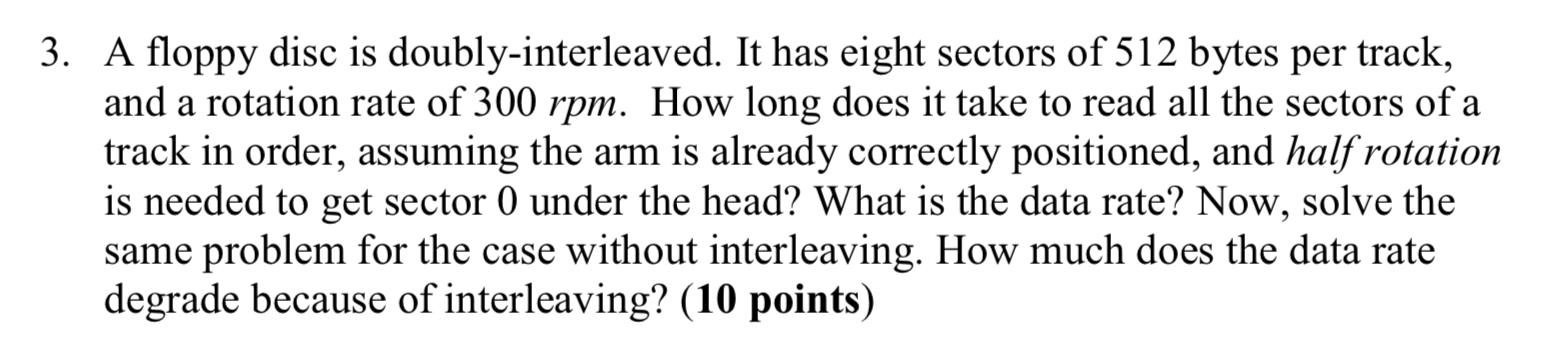
While interrupt handling all 34 CPU registers are to be pushed to stack.

After the completion of Interrupt service routine, 34 CPU registers needs to be copied back & after copying registers to stack a new PC, pointing to start of Interrupt service routing, has to be loaded from memory. Basically, Returning from the interrupt requires fetching 34 words from the stack.

Hence total I/O operations are 34 +34 = 68.

For these 68 operations, it will take 680 nsec = 680 × sec

Hence total Interrupt handled by system = = 1.47 × if we assume that no work is done for each interrupt.



We are given that the 0.5 rotation and the arm is correctly positioned and there are 8 sectors in the disk and the rotation rate is 300rpm.

**For interleaved disk**

Rotation is 0.5(initial) + 2.75(to get to all the sectors from 0) = 3.25 rotation

We are given that the RPM is 300 which is 300 rounds(rotation) per minute

For 3.25 rotation, the time taken is 60\*(3.25/300) = 0.650 seconds

Total time taken to read all the sectors from the initial sector is 0.65 seconds

The total transfer rate is 4096/0.65 = 6302 bytes/sec

**For non-interleaved**

The total rotation needed is = 0.5 + 1(To get to all the sectors from 0) = 1.5

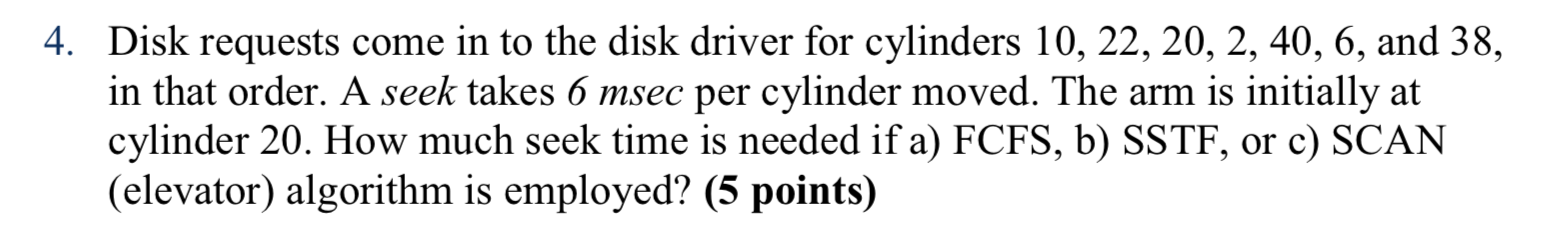
We are given that the RPM is 300 which is 300 rounds(rotation) per minute

For 1.5 rotation, the time taken is = 60 \*(1.5/300) = 0.3 seconds

Total time taken to read all the sectors from the initial sector is 0.3 seconds

The total transfer rate is 4096/0.3 = 13653.3 bytes/sec

**The total degrading due to interleaving is 46 (6302/13653) percent**



**Solution**:

First come first serve:

The order of reading cylinders will be 10, 22, 20, 2, 40, 6, 38

The total move step is 10 + 12 + 2 + 18 + 38 + 34 + 32 = 146 cylinders × 6 milliseconds = 876 milliseconds.

Shortest seek time first:

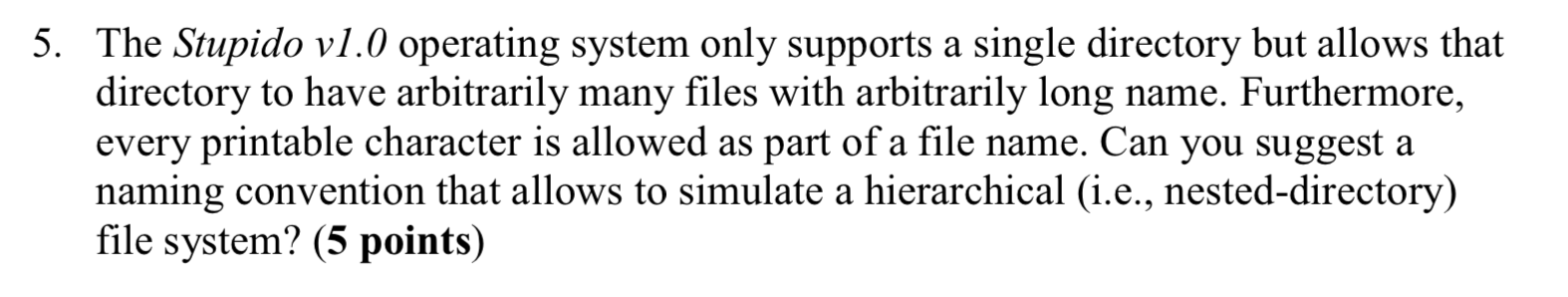
The order of reading cylinders will be 20, 22, 10, 6, 2, 38, 40

The total move step is 0 + 2 + 12 + 4 + 4 + 36 + 2 = 60 cylinders × 6 milliseconds = 360 milliseconds.

SCAN(elevated):

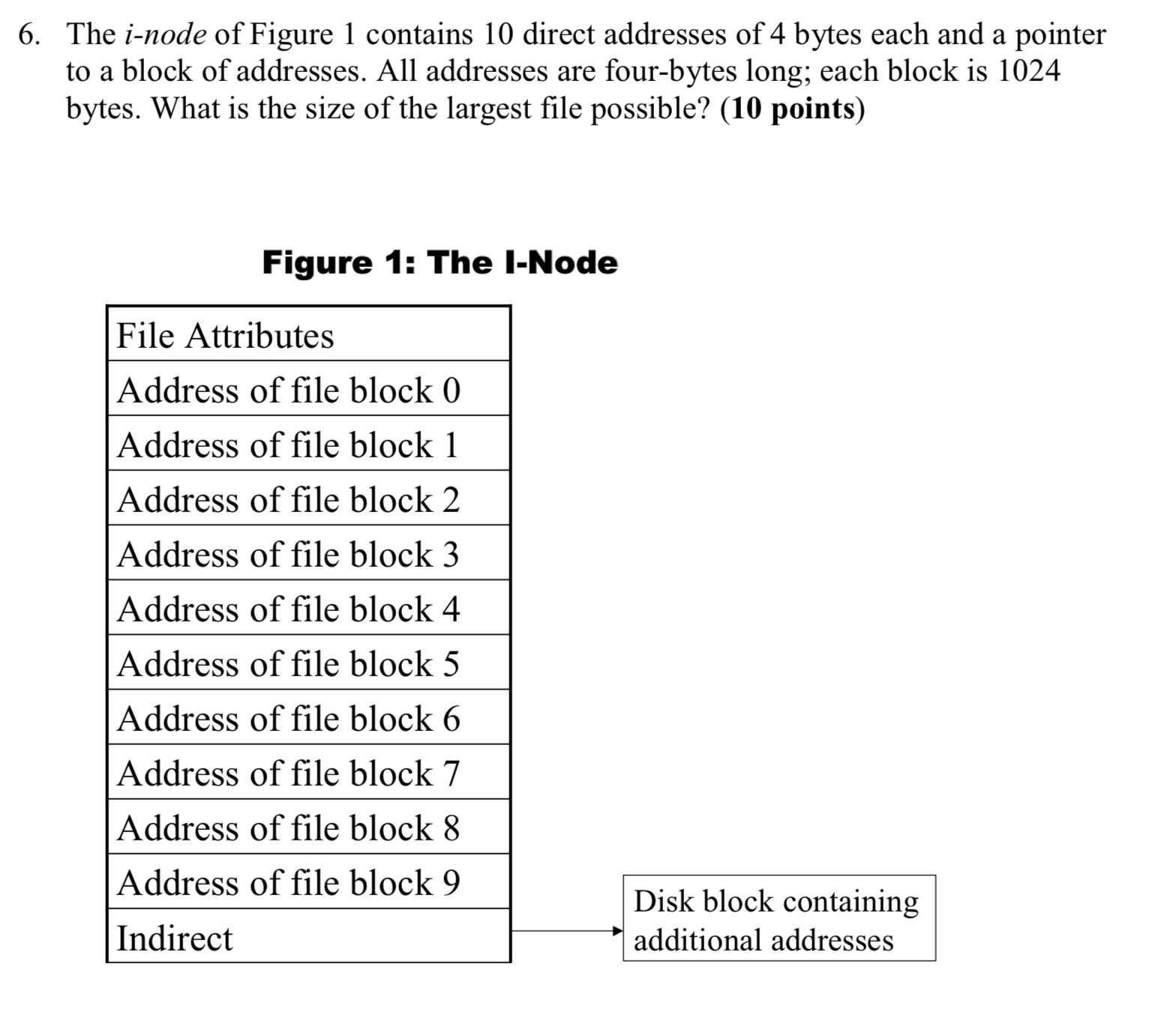
The order of reading cylinders will be 20, 22, 38, 40, 10, 6, 2

The total move step is 0 + 2 + 16 + 2 + 30 + 4 + 4 = 58 cylinders × 6 milliseconds = 348 milliseconds



As file name can be of any length the easiest way to name a file would be to combine names of desired directory structure and actual file name instead of creating the sub directories.

For example, the WorkFile.pdf file will have the name as “/home/DivyendraPatil/WorkDocuments/Office/WorkFile.pdf”

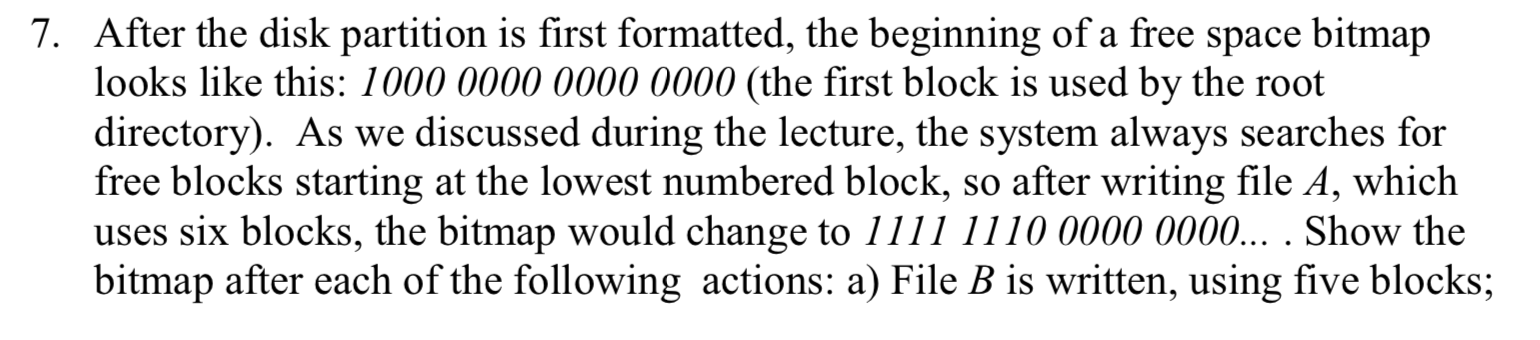


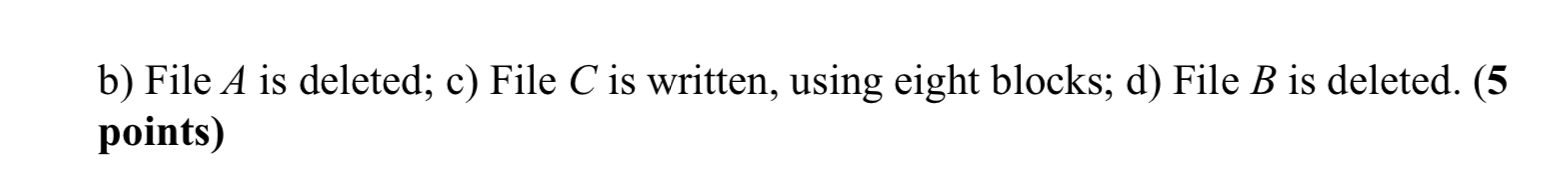
The Number of addresses per block = 1024 / 4= 256

These are also called as number of indirect blocks in the file.

There are 10 direct addresses & therefore the total number of addresses present are 266.

Each address is of 1024 bytes hence total size of the file pointed by these addresses is 266 × 1024 = 272,384 bytes or 266 KB.





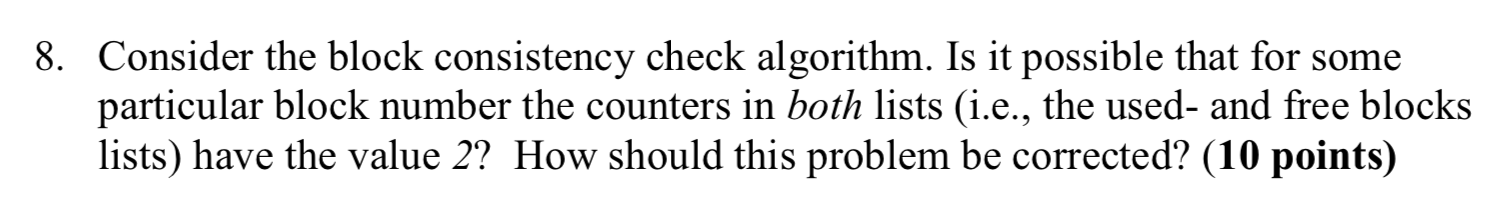
Original free space bitmap of system after writing file A is

1111 1110 0000 0000

* 1. File B is written, using five blocks: 1111 1111 1111 0000
  2. File A is deleted: 1000 0001 1111 0000
  3. File C is written, using 8 blocks. Here the assumption made is that File C is can be divided in two discontinuous parts: 1111 1111 1111 1100

If the file cannot be divided into two parts, then last 4 bytes may be lost.

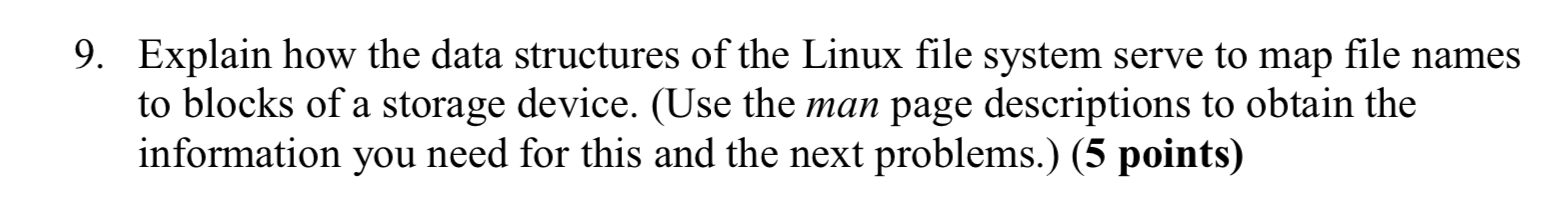
* 1. File B is deleted: 1111 1110 0000 1100



It is **not** possible to have value 2. Because if the value is present in both free list and block list the block is marked as suspicious.

But in case, by some error/glitch the values get written on both the list, then another free block should be found & contents of suspicious block should be copied there.

Then remove the contents from suspicious block & re run the block consistency check algorithm to correct the values of the lists.



The Unix environment is divided physical disks into logical disks called partitions. Each partition is a standalone file system. Each disk device is given its own major device number and each partition has its own minor device number. The major/minor device number combination serves as a handle into device switch table. That is, the major number acts as an index and minor number is used to driver router to get specific instance of a device.

Each file system contains a linear array of **inodes** (short for index nodes). There is one to one mapping of files to inodes and vice versa. An inode is identified by its ‘inode number’, which contains the information needed to find the inode itself on the disk Thus, while users think of files in terms of file names, Unix thinks of files in terms of inodes.

An inode is the **handle** to a file and contains the following information:

* file ownership indication
* file type (e.g., regular, directory, special device, pipes, etc.)
* file access permissions. May have setuid (sticky) bit set.
* time of last access, and modification
* number of links (aliases) to the file
* pointers to the data blocks for the file
* size of the file in bytes (for regular files), major and minor device numbers for special devices.

An integral number of inodes fits in a single data block.

Information the inode does not contain:

* path (short or full) name of file

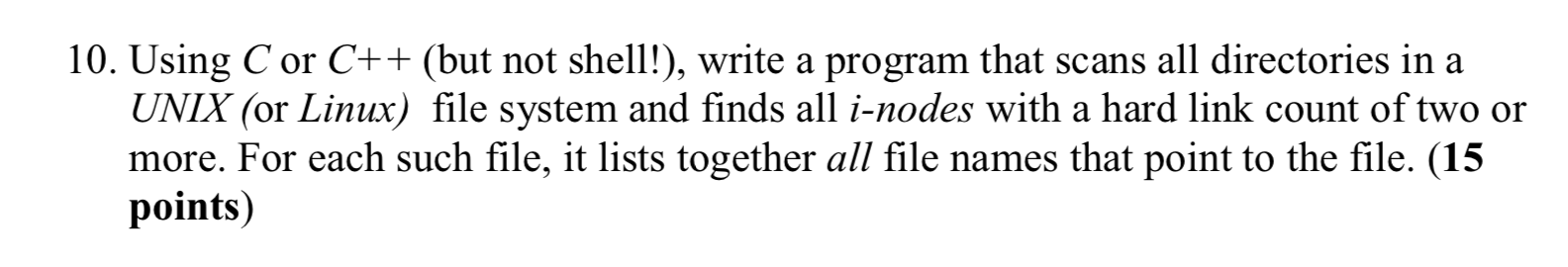
Internally, Unix stores directories in files. The file type (of the inode) is marked ``directory'', and the file contains pairs of name/inode numbers.

The Unix file system allocates data blocks (blocks that contain a file's contents) one at a time from a pool of free blocks. Unix uses 4K blocks. Moreover, a file's blocks are scattered randomly within the physical disk.

Inodes include pointers to the data blocks. Each inode contains 15 pointers:

* The first 12 pointers point directly to data blocks
* The 13th pointer points to an indirect block, a block containing pointers to data blocks
* The 14th pointer points to a doubly-indirect block, a block containing 128 addresses of singly indirect blocks
* The 15th pointer points to a triply indirect block (which contains pointers to doubly indirect blocks, etc.)

These features changes with each new version of file system type implemented such as “The Berkerly Fast File System” or new “EXT4”



An inode in Unix contains following details:

struct stat {

dev\_t st\_dev; /\* ID of device containing file \*/

ino\_t st\_ino; /\* inode number \*/

mode\_t st\_mode; /\* protection \*/

nlink\_t st\_nlink; /\* number of hard links \*/

uid\_t st\_uid; /\* user ID of owner \*/

gid\_t st\_gid; /\* group ID of owner \*/

dev\_t st\_rdev; /\* device ID (if special file) \*/

off\_t st\_size; /\* total size, in bytes \*/

blksize\_t st\_blksize; /\* blocksize for file system I/O \*/

blkcnt\_t st\_blocks; /\* number of 512B blocks allocated \*/

time\_t st\_atime; /\* time of last access \*/

time\_t st\_mtime; /\* time of last modification \*/

time\_t st\_ctime; /\* time of last status change \*/

};

Within a directory, each entry is an inode with the following details

struct dirent {

/\* when \_DARWIN\_FEATURE\_64\_BIT\_INODE is NOT defined \*/

ino\_t d\_ino; /\* file number of entry \*/

\_\_uint16\_t d\_reclen; /\* length of this record \*/

\_\_uint8\_t d\_type; /\* file type, see below \*/

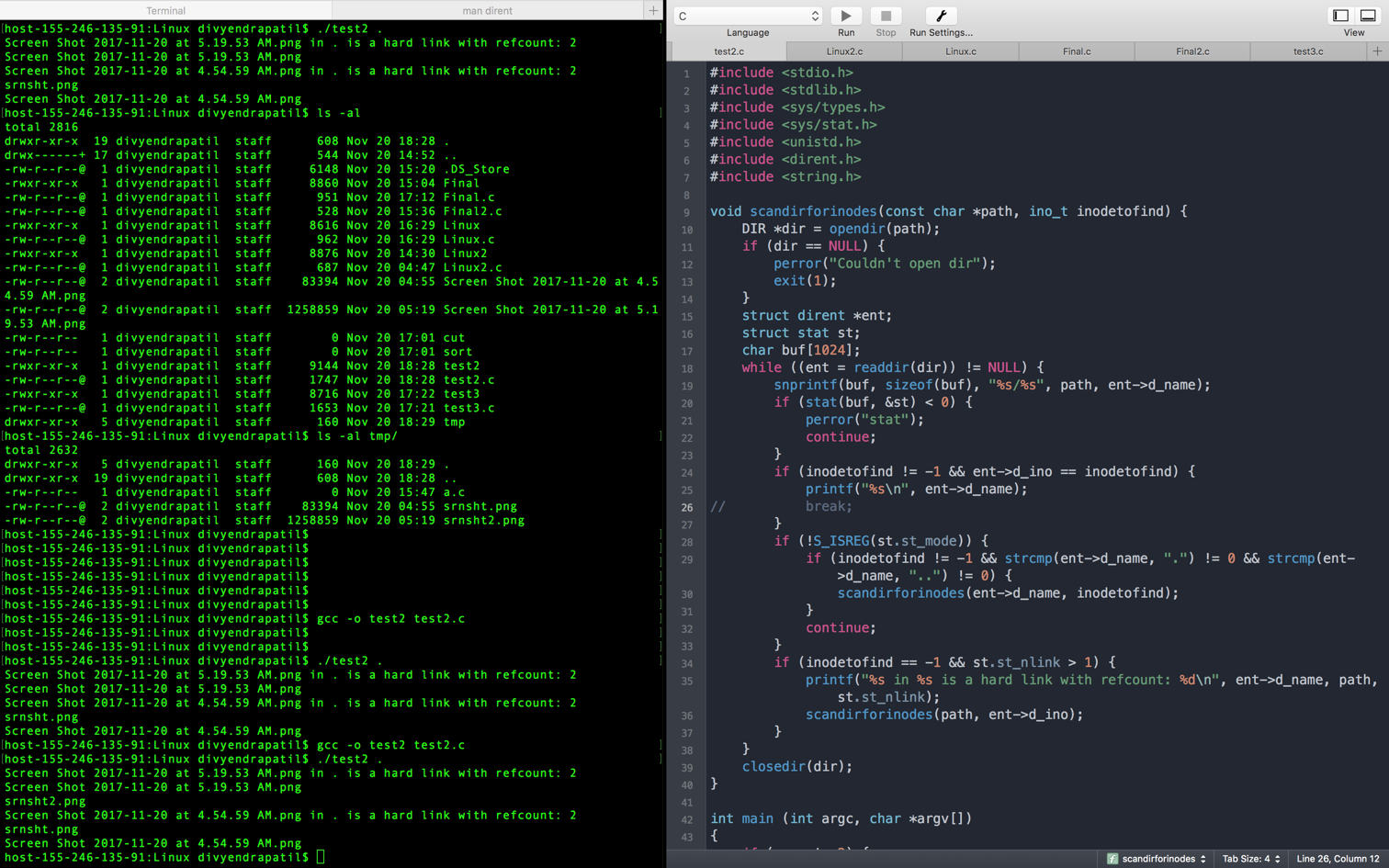
\_\_uint8\_t d\_namlen; /\* length of string in d\_name \*/

char d\_name[255 + 1]; /\* name must be no longer than this \*/

};

In the submitted program, we open a directory and scan all the inodes within the directory & whenever we find an inode with hardlink count greater than 1, we start to scan all the other inodes pointing to that particular inode within same directories and its sub directories.

As seen in the screenshot provided, we have manually created two hardlinks for testing purposes which were then listed out by our program.



#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#include <dirent.h>

#include <string.h>

void scandirforinodes(const char \*path, ino\_t inodetofind) {

DIR \*dir = opendir(path);

if (dir == NULL) {

perror("Couldn't open dir");

exit(1);

}

struct dirent \*ent;

struct stat st;

char buf[1024];

while ((ent = readdir(dir)) != NULL) {

snprintf(buf, sizeof(buf), "%s/%s", path, ent->d\_name);

if (stat(buf, &st) < 0) {

perror("stat");

continue;

}

if (inodetofind != -1 && ent->d\_ino == inodetofind) {

printf("%s\n", ent->d\_name);

}

if (!S\_ISREG(st.st\_mode)) {

if (inodetofind != -1 && strcmp(ent->d\_name, ".") != 0 && strcmp(ent->d\_name, "..") != 0) {

scandirforinodes(ent->d\_name, inodetofind);

}

continue;

}

if (inodetofind == -1 && st.st\_nlink > 1) {

printf("%s in %s is a hard link with refcount: %d\n", ent->d\_name, path, st.st\_nlink);

scandirforinodes(path, ent->d\_ino);

}

}

closedir(dir);

}

int main (int argc, char \*argv[])

{

if (argc != 2) {

fprintf(stderr, "Usage: ./%s directory\n\n", argv[0]);

exit(1);

}

const char \*path = argv[1];

scandirforinodes(path, -1);

return 0;

}

Final Output:

