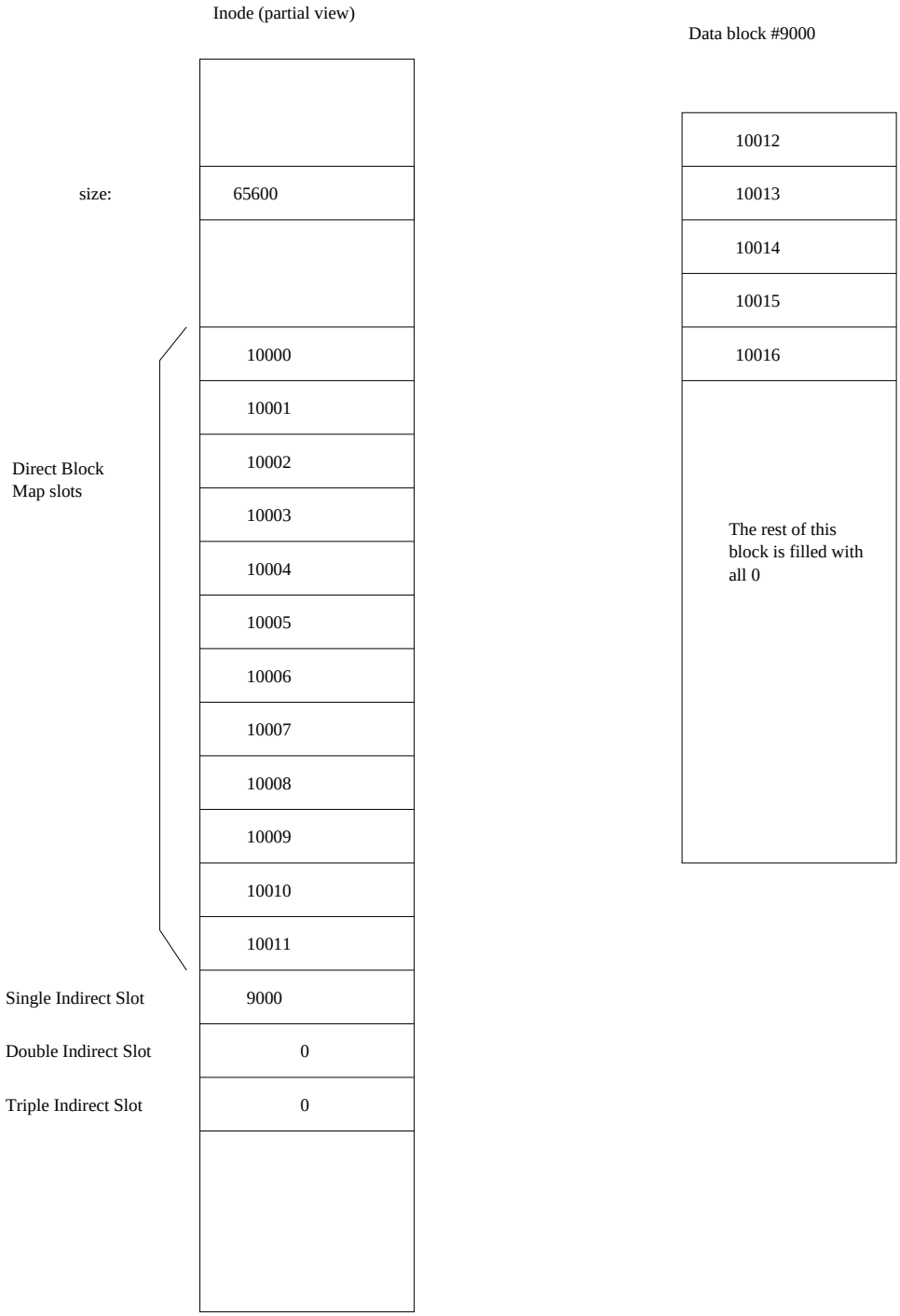
Problem 1 -- File allocation

The following diagram depicts part of the information contained inside an inode which represents an ordinary file. Specifically, it shows the block allocation area of the inode and the size. Also shown is one data block which is being used as an indirect block. This is a Linux EXT3 filesystem and block sizes are the default 4K. *Note: all values in the diagram below are expressed in decimal (base 10)*



a)What is the number of the data block (express in decimal) which contains byte 4100 decimal (0x1004 hexadecimal)?

b) Ditto, for byte 65535 dec (0xFFFF hex)

c) If instead of EXT3, this were an EXT4 filesystem using extent-based allocation, but the same blocks were allocated to the file in the same order, how many extent descriptors would be required? Where would the extent descriptor(s) be stored? Justify your answer.

d) Now, back to EXT3, and we modify the file as follows: we `lseek` to file offset 409600 (decimal) bytes and then `write` a single byte. How many additional data blocks does the kernel have to allocate? Justify your answer.

Problem 2 -- Exploratory Questions

Answer each of the following with a paragraph or two **in your own words**. After you read the lecture notes, you'll have enough material to answer. **Don't just copy the lecture notes verbatim**, use your own words.

A) In browsing the inode table of an EXT3 filesystem I observe an inode with inode type `S_IFDIR` and `nlink`==4. What does this link count tell us about the contents of this directory inode?

B) There is a directory say `/home/pcooper/stuff` which has about 50,000 files. The first time I `open("/home/pcooper/stuff/12345.txt",O_RDONLY)` it takes a "long time" but when I open the same file again a few seconds later the `open` system call returns much faster. Explain.

C) We have an EXT3 filesystem with journalling turned on, using mode `data=ordered`. Some schlemeil trips over the power cord while the system has filesystem activity. When the system powers back up, `fsck` examines the journal and finds this:

```
--BEGIN TRANSACTION ID#1234--
Copy of block containing inode #9999 type==FILE size==0 nlink==1
Copy of block containing entry 9999 in free inode bitmap, showing i9999 allocated
Copy of block containing new directory entry referring to ino9999
Copy of block containing ino100 (parent of i9999) with updated mtime
--COMMIT TRANSACTION ID #1234--
--BEGIN TRANSACTION ID#1235--
Copy of block containing free block bitmap entry for block #5555, showing d5555 allocated
Copy of disk block containing inode #9999, now size==4096, mtime updated
--COMMIT TRANSACTION ID #1235--
```

i) From examining the journal, what does it appear was happening?

ii) Before we "replay" the above two journal entries, can we say what state inode #9999 is on the disk?

iii) After the "replay" would there be any perceptable corruption of the filesystem?

iv) Reconsider question (iii) but this time assume the filesystem journal mode was set to `data=writeback`

D) I mount a filesystem:

```
# mount /dev/sdb1 /mnt/usbdrives/volume01
# ls -ladi /mnt/usbdrives/volume01
```

Where `/mnt/usbdrives/volume01` was an empty directory within the root filesystem (aka root volume) and is inode #10. What is the inode # reported by the ls command? Explain your answer.

E) A user runs the command `mv /A/B/F1 /A/C/F2;` where `F1` is a fairly large file. This command runs very quickly. Then the user runs `mv /A/C/F2 /A/Z/F3;` but this takes quite a while to run. Why might that be the case?

SUBMISSION: Submit your answers to problems 1 / 2 in the format of either a PDF or plain text document.

Problem 3 -- Recursive filesystem statistics checker

Write a program which recursively explores the filesystem starting from a given point, which is the sole argument to the program. The order in which the filesystem tree is explored (e.g. depth-first) is not important. Tabulate the following statistics and report at the end:

a) # of inodes of each type (e.g. directory, file, symlink, etc.) encountered

b) for all regular files encountered, the sum of their sizes, and the sum of the number of disk blocks allocated for them. The ratio gives you, in a sense, the spatial efficiency of the filesystem

c) # of inodes (except, of course, directories) which have a link count of more than 1.

d) # of symlinks encountered that did not (at least at the time of the exploration) resolve to a valid target

e) # of directory entries encountered which would be "problematic" to enter at the keyboard. This means any name which contains non-printable characters, non-ASCII characters, or special characters that confuse the shell unless properly escaped.

Error handling: you might encounter errors, especially if you try to explore a part of the filesystem where you don't have permission. Do not terminate the entire program because of an error. Report the error and take reasonable action to continue, if possible.

SUBMISSION: Your code, plus a screenshot (must be READABLE without a magnifying glass) or plain text session output, showing your code running.