# **CSSE2310: 2016 exam answers**

## [**UQAttic**](http://uqattic.net)

## **Get more out of your study time. Join UQAttic's revision chat.**

#### [**Other exam papers**](https://drive.google.com/drive/folders/1k04K5RHL61k3ae2jJ362MPdN5xy3Bqsp)

### Please **contribute** to these documents.

### If you're looking for an effective way to familiarise yourself with the course material, you can't go past collaborating with fellow students. We have laboured to put these up, and so at the very least point out where you think we are wrong!

### You'll get more out of the course, you'll do better in the exam, and other students will benefit from your input as well.

### To get editing permissions, simply go to the [chatroom](http://uqattic.net) and provide us with your Google Account address.

### **Style.**

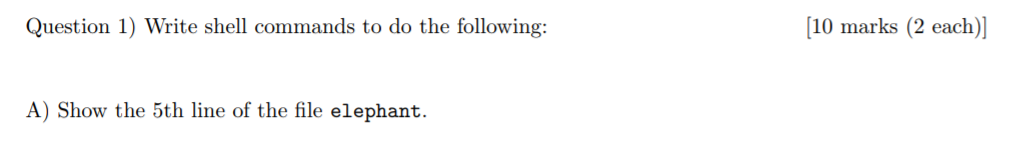
### Type answers in blue beneath each question.

### If you're unsure of your answer, highlight your answer text then hit Ctrl+Alt+M to create a comment beside the text. Once you're satisfied with the answer, click the "Resolve" button on the comment.

### If you want some extra explanation from someone else on their answer, highlight the other person's answer and repeat the procedure above.

### **Communicate.**

### Head over to [uqattic.net](http://uqattic.net/) and click "Chat Now!". You'll find a chatroom full of students just like you. Talk about a revision document (like this one) or swap prep tips. If you have your own IRC client, point it to irc.uqattic.net, port 6667, channel #attic.

****

**cat elephant | head -5 | tail -1**

****

**cat elephant | cut -d‘ ’ -f 1 | grep mouse | wc -l**

****

**ps -u $USER | grep bash**

**# I think this is more correct:**

**pgrep -l -u$USER bash**

****

**for file in $(ls)**

**do**

**if [ ${#file} -ge 8 ]**

**then**

**echo $file**

**fi**

**done**

**# Or**

**ls -a1d ????????\* | grep -v ^d**

****

**PATH=$PATH:/srd/bin**

****

**const char\* foo;**

****

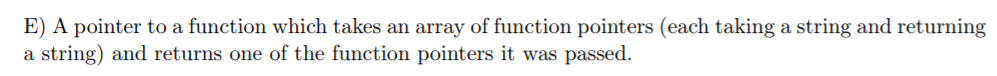
**unsigned long ~~long~~ int foo;**

****

**double foo[12];**

****

**char\* (\*foo)(void\*, void\*, void\*, char\*);**

****

**(char\*) (\*(\*foo)((char\*)\*[](char\*))) (char\*)**

**{Blue is for foo as pointer to function**

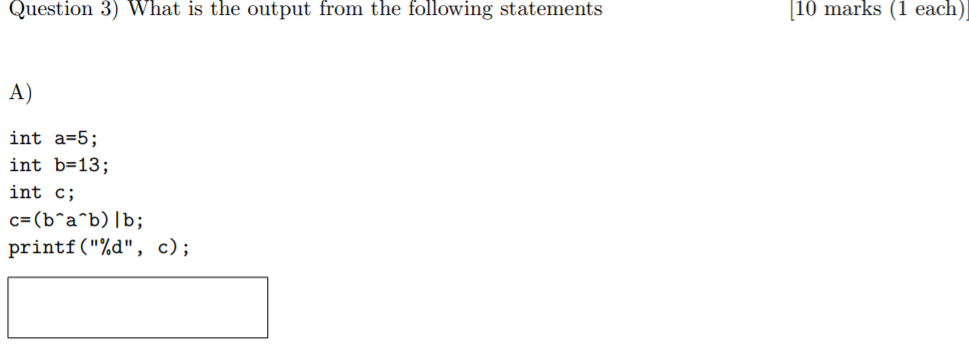
**With Pink representing it’s parameter**

**The red part is for the return}**

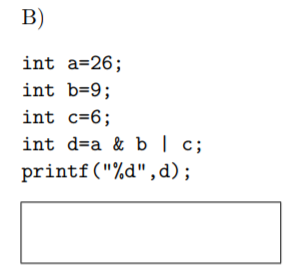
**~~(char\*)(\*)(char\*) (\*foo) ((char\*)(\*)(char\*) \*); // definitely unsure on this answer~~**

**typedef char\* (\*func\_t)(char\*);**

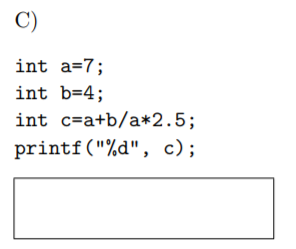
**func\_t (\*func)(func\_t\*);**

****

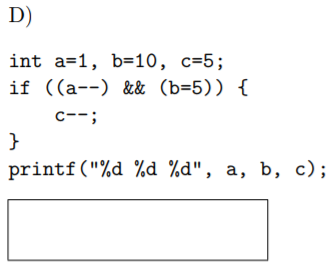
**13**

****

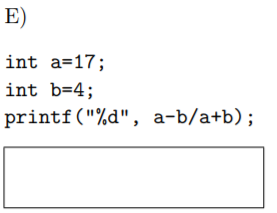
**14**

****

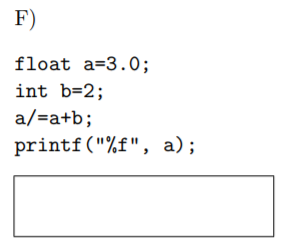
**7**

****

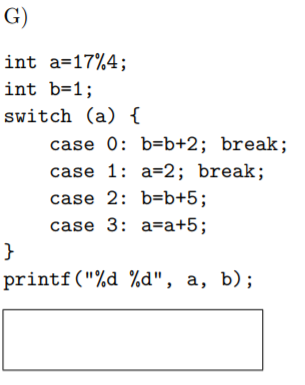
**0 5 4**

****

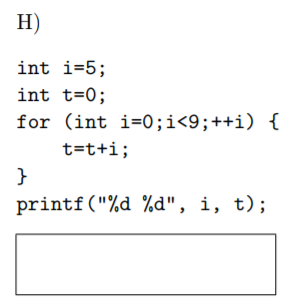
**21**

****

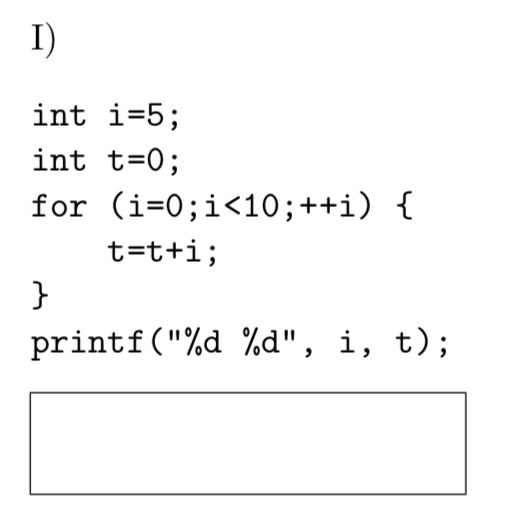
**0.600000**

****

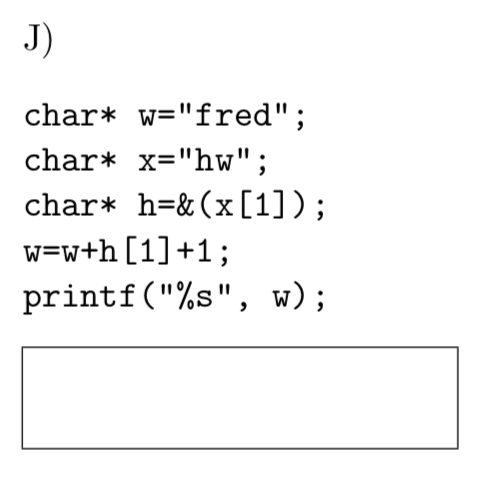
**2 1**

****

**5 36**

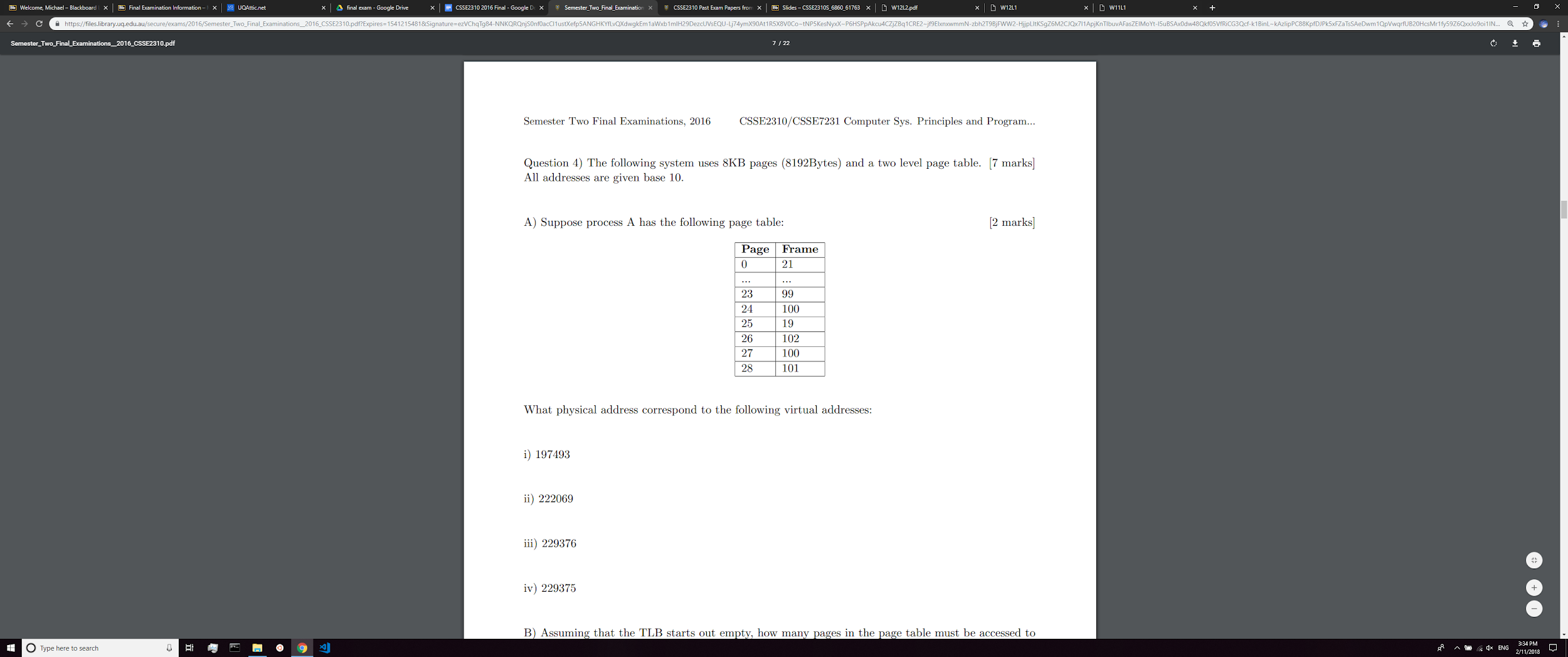
****

**10 45**

****

**red**

**4.**

****

**i)**

197493 = 24\*8192 + 885

=> 100\*8192 + 885 = **820085**

**ii)**

222096 = 27\*8192 + 885

=> 100\*8192 + 885 = **820085**

**This is weird because both 24 and 27 point to the same frame,**

**iii)**

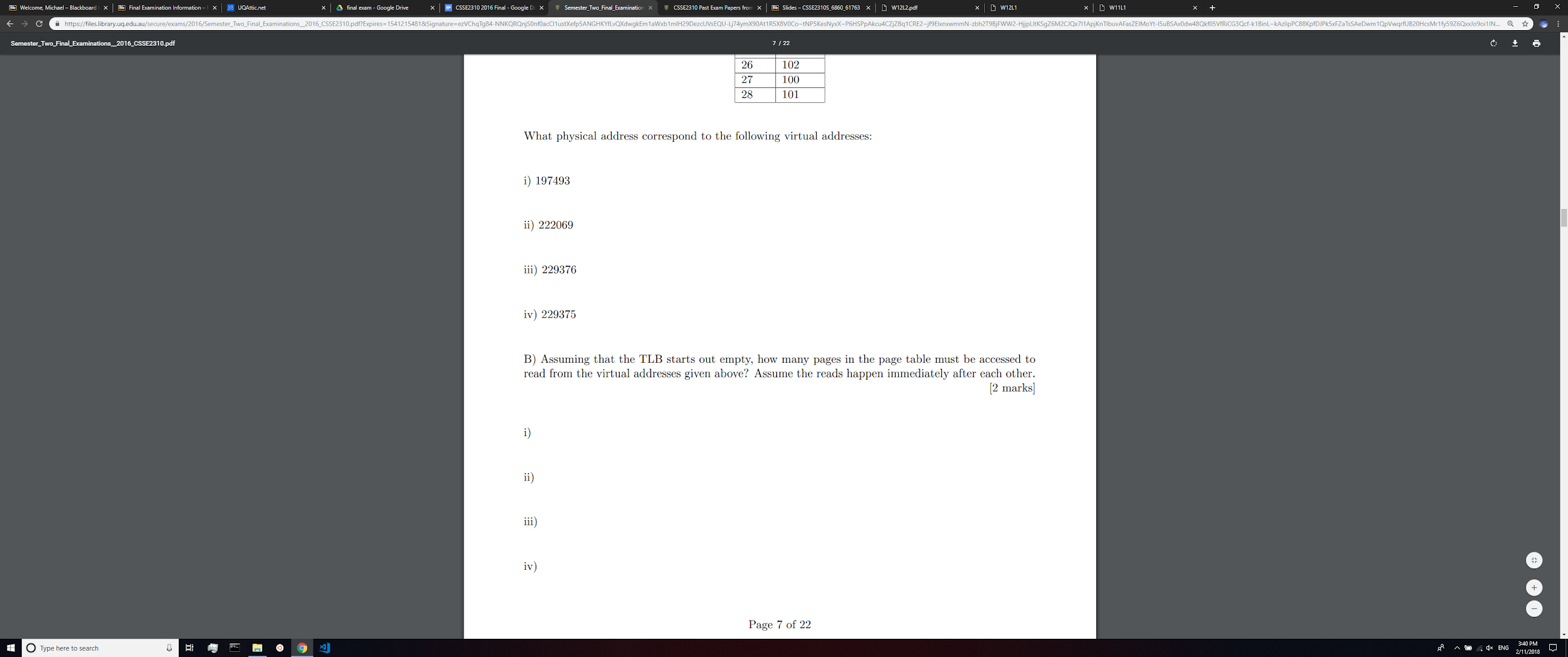
229376 = 28\*8192 + 0

=> 101\*8192 + 0 = **827392**

**iv)**

229375 = 27\*8192 + 8191

=> 100\*8192 + 8191 = **827391**

****

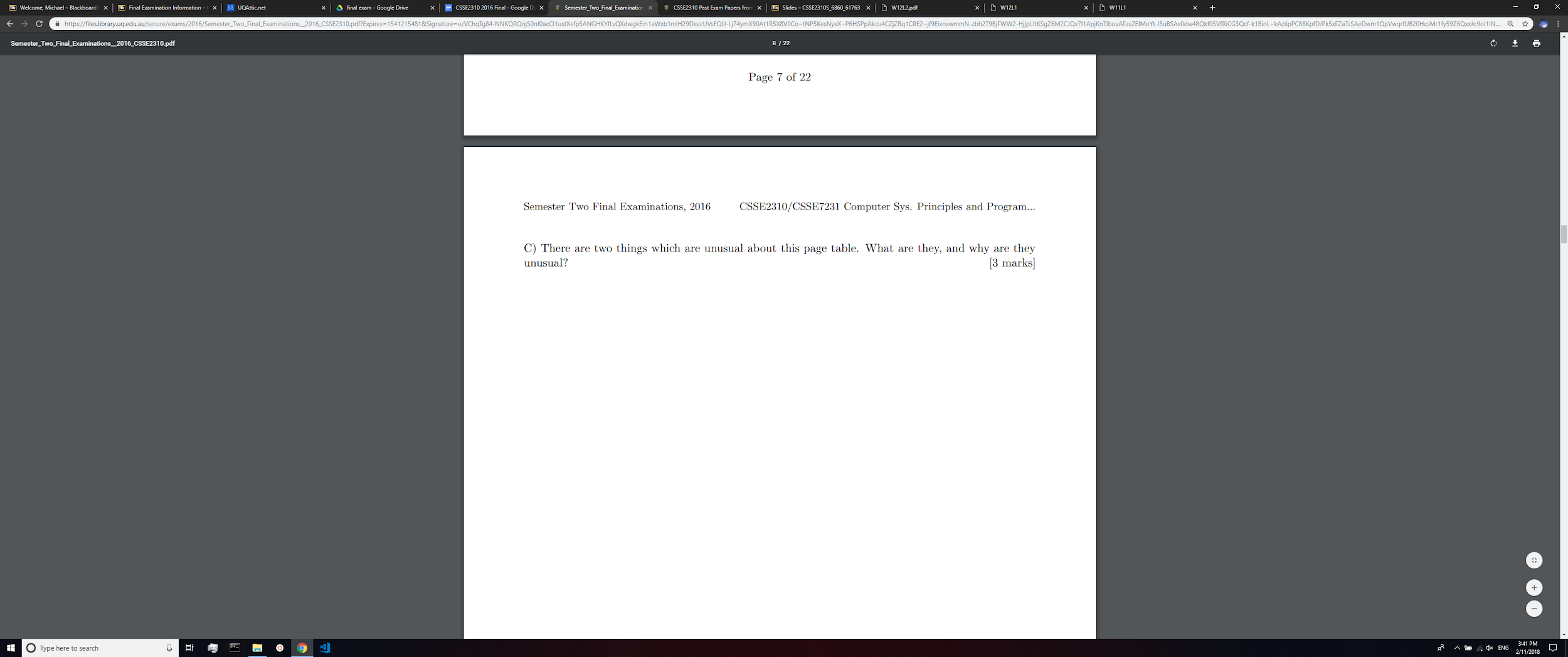
**B)**

**i) 1**

**ii) 1**

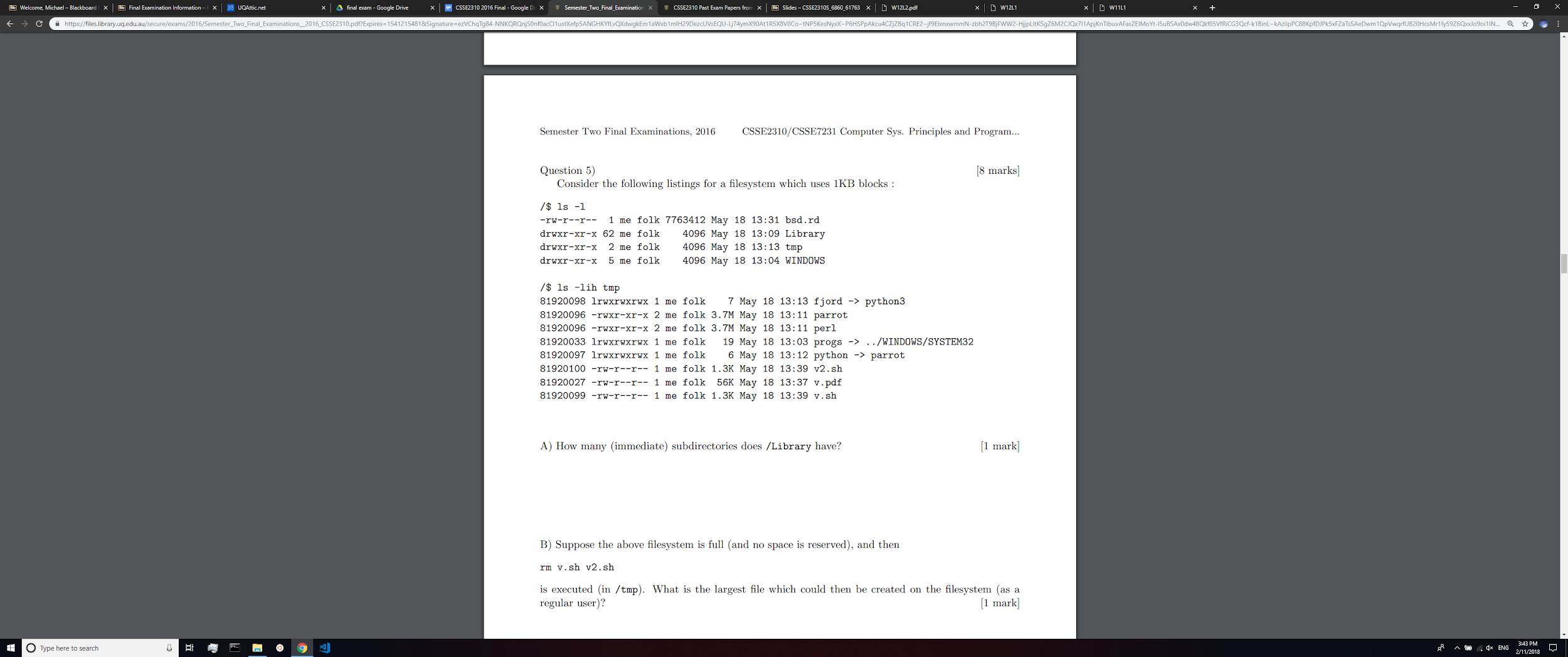
**iii) 2**

**iv) 2**

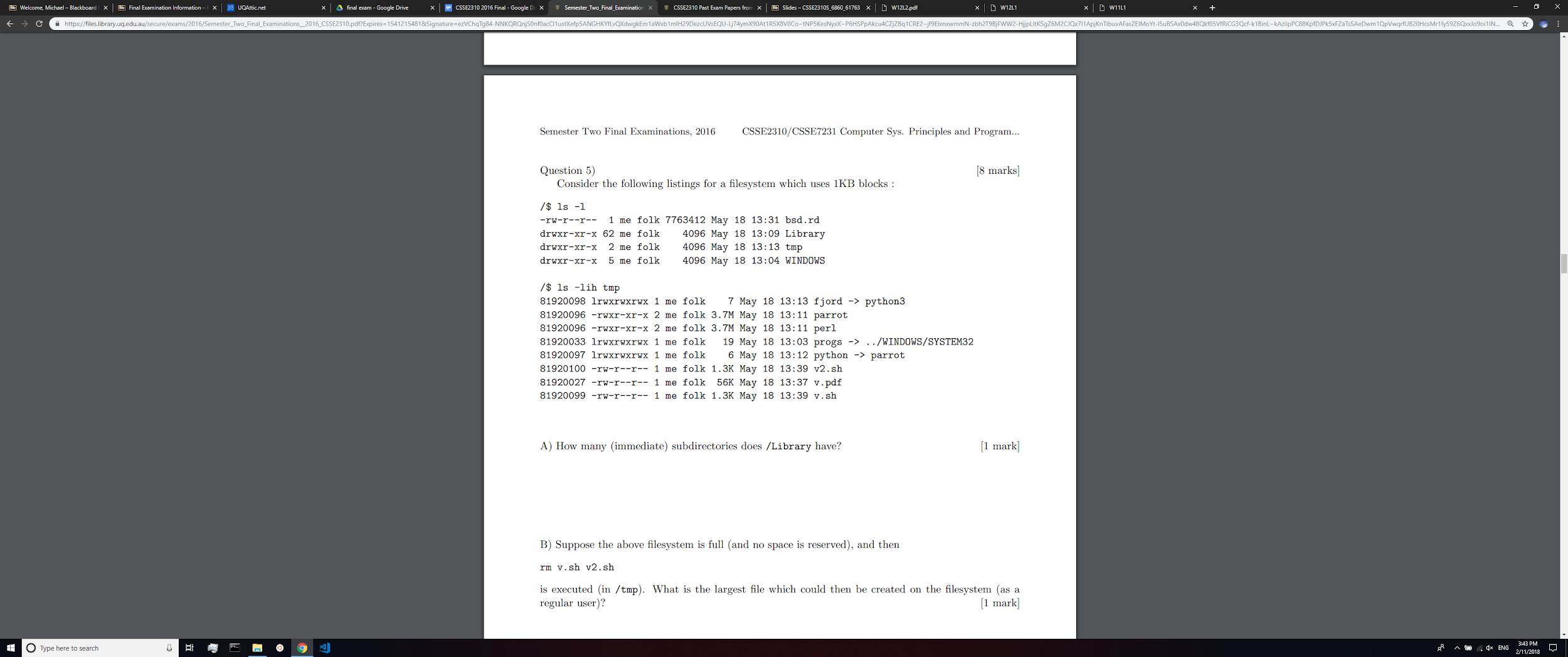
****

-Two pages map to the same physical address, wasting one page

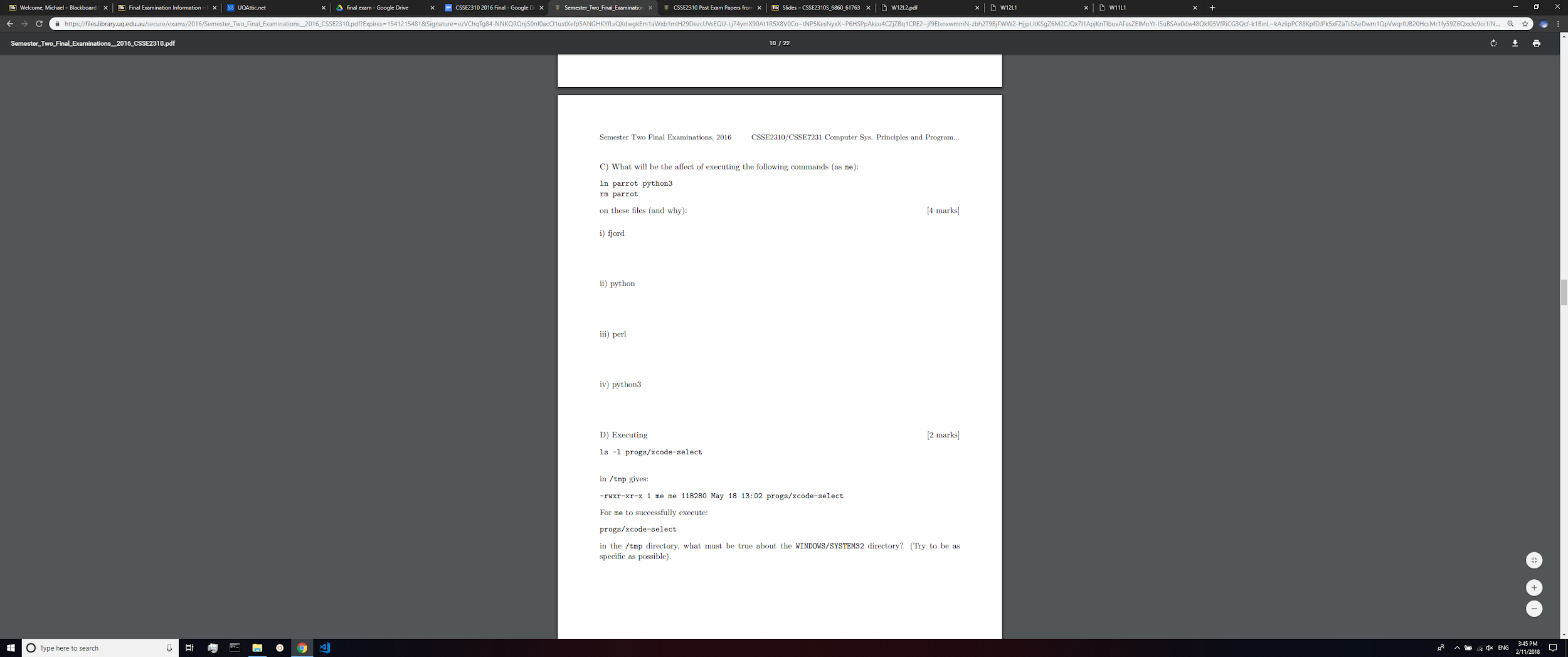
-Page 0 is a valid page, usually reserved for error checking



60



Since blocks are 1KB big, would the answer really be 2 \* ceil(1.3) = 4KB ? [+1]



i)

Removing parrot doesn't mean python3 is removed I thought. So fjord can still access python3 meaning it is unaffected. +2

ii)

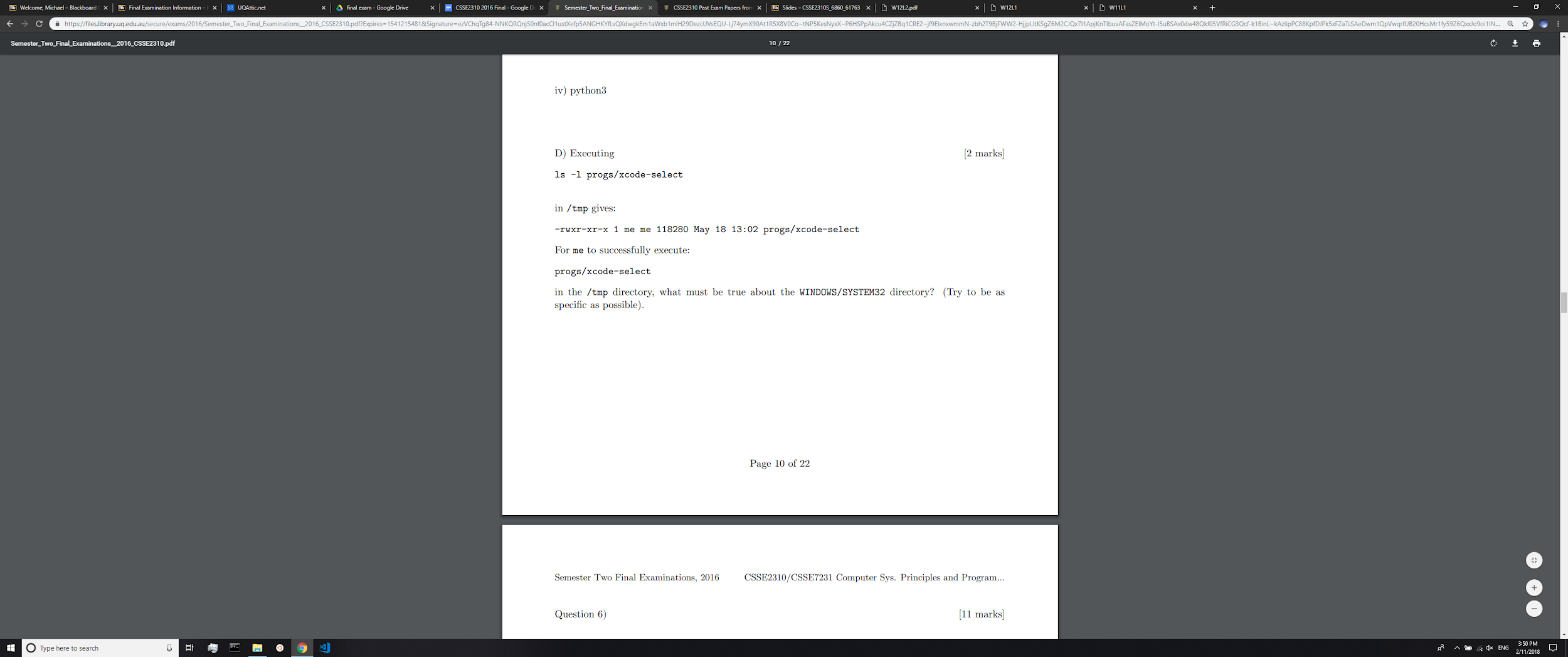
Dangling soft link

iii)

Perl is hard linked. Therefore when parrot is removed, the inode counter goes down and that is it.

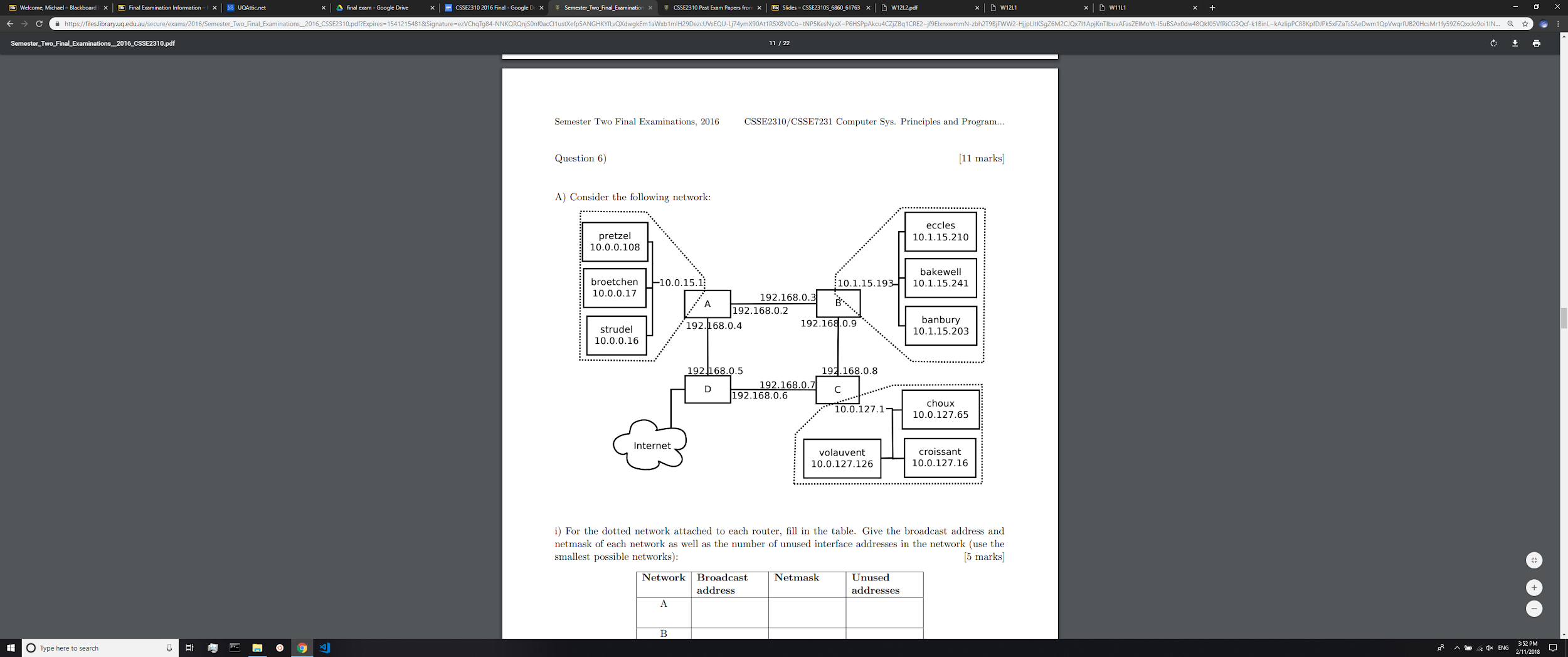
iv)

Nothing (bc of inode count being greater than 1)



The user ‘me’ is allowed to execute (change directory into) WINDOWS/SYS/32 and thus has the right execute bits set on that directory

(if anyone has a better answer pls share)



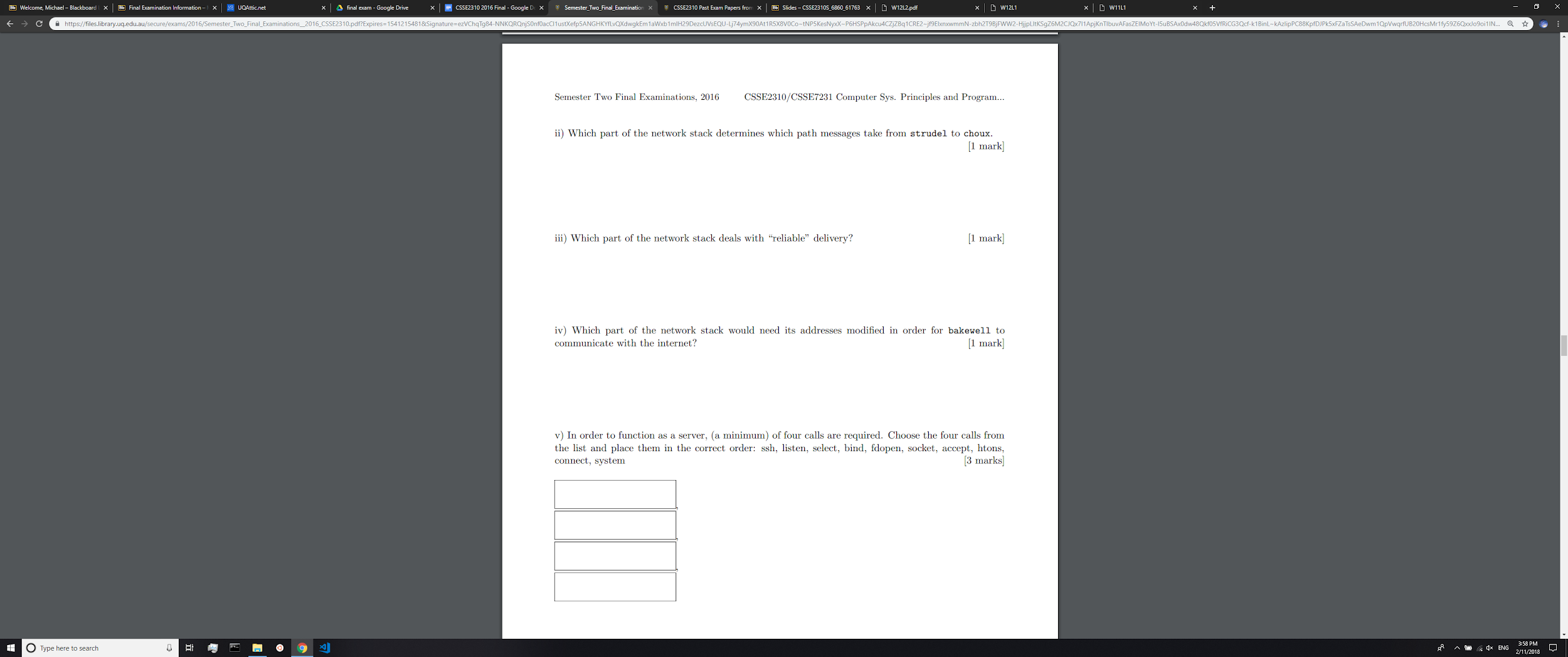
|  |  |  |  |
| --- | --- | --- | --- |
| Network | Broadcast | Netmask | Unused Addresses |
| A | 10.0.0.127  10.0.15.255 | 255.255.255.128  255.255.240.0 | 127**-2**-3=122 (**-2** for network + broadcast)  4090 |
| B | 10.1.15.192  10.1.15.255 [+1] | 255.255.255.192 | 63-2-3 = 58  (2^(31-notation))-2 |
| C | 10.0.127.0  10.0.127.127 | 255.255.255.127  255.255.255.128 | 127-2-3=122 |

**I believe the answers in green are correct, can anyone else weigh in? +3**

**If we have to consider the router addresses also then yes, I agree. +1**

**I believe that the answers in black are correct since that is what’s used on the tutes.**

**(char\*)**



ii)

Layer 3

iii)Layer 4 (TCP)

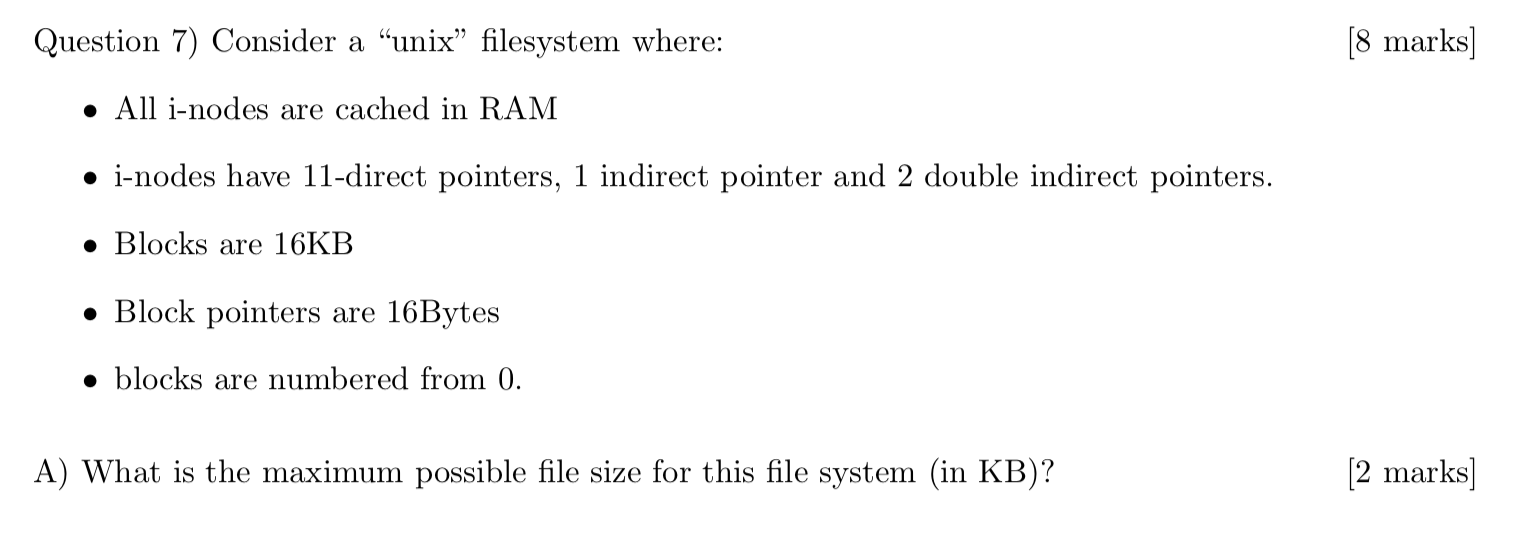
iv)

Layer 1

How is this layer 1? Layer 1 doesn't deal with addresses, it's just the physical layer (wires, radio waves, etc.). I would say layer 2.

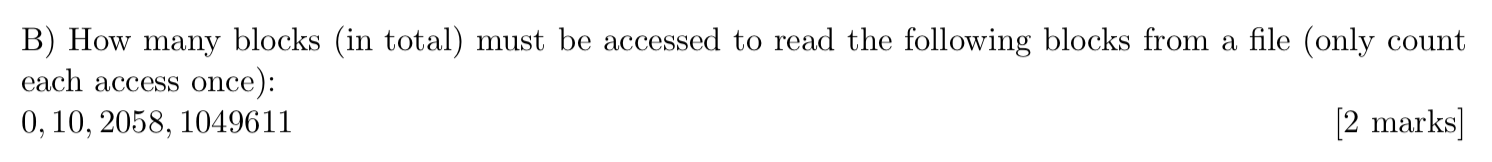
v)

|  |
| --- |
| socket |
| bind |
| listen |
| accept |



**Assume KiB since KB is not a thing**

**(KiB = kibibyte, kB = kilobyte, KB = ??)**

****

**0: 1 read**

**10: 1 read**

**2058: 3 reads**

**1049611: 2 reads**

**total = 7 reads**

Alternative:

0: 1 for top level inode + 1 for block 0

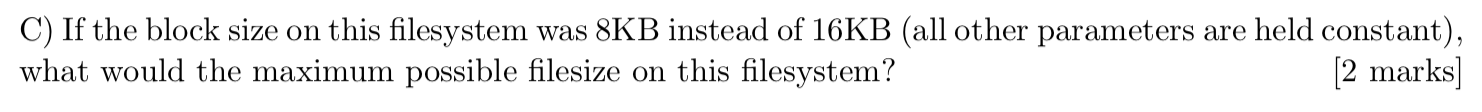
10: 1 for block 10 (inode is cached)

2058: 1 for double indirect + 1 single indirect off that double + 1 for block 2058

1049611: 1 for the single indirect off that double + 1 for block 1049611

Total = 8

**+2**

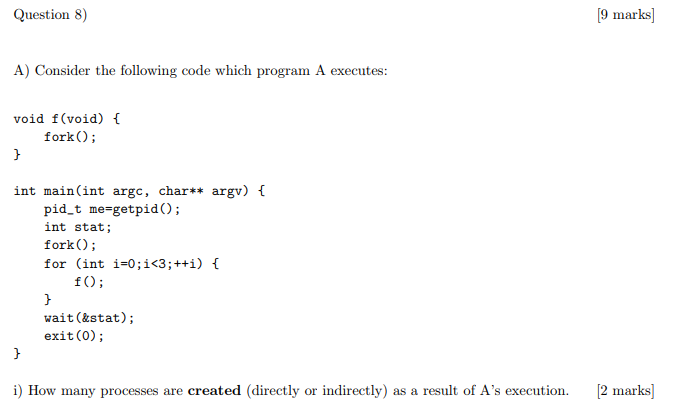
****

****

**faster access?**

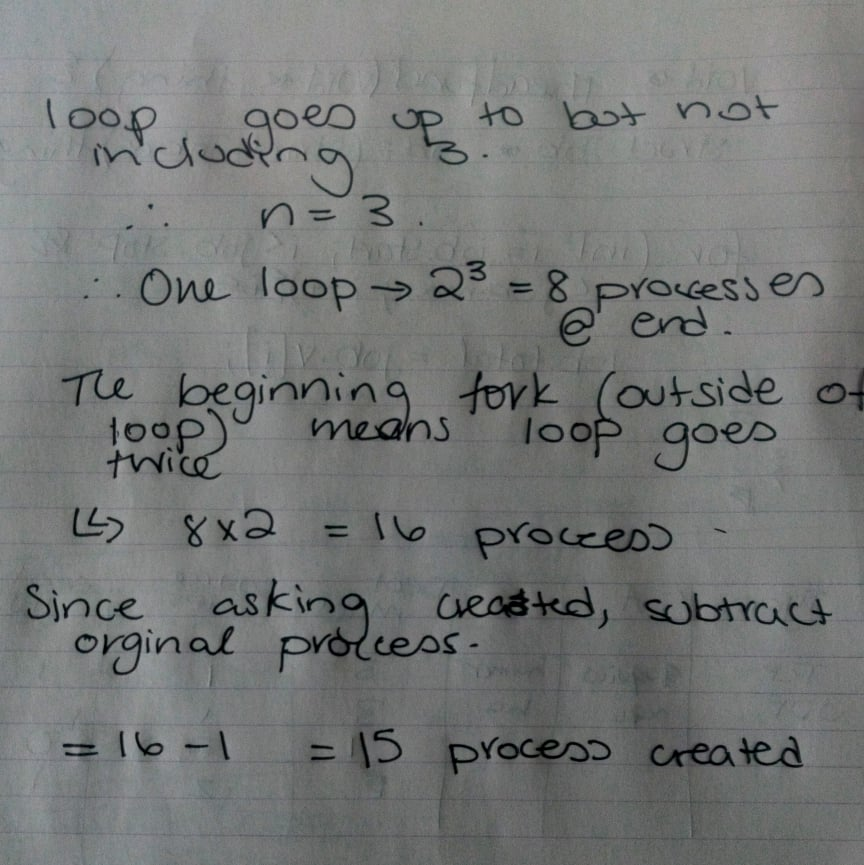
**Not as much wasted space in end blocks?**

**^^ fancy terminology: Less internal fragmentation**

****

**2 \* (23) - 1 = 15 processes**

**// please ignore my grammar/spelling but hope this helps**

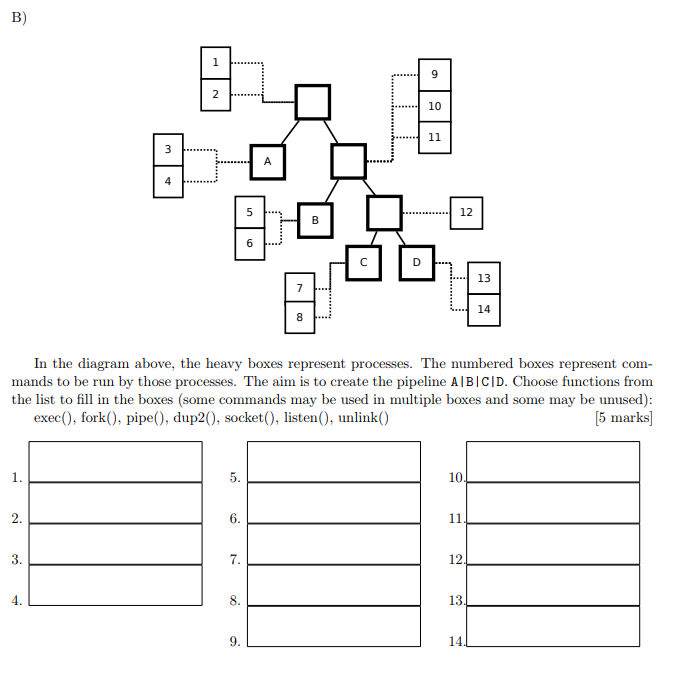
****

****

**7 processes (15 - 8)**

****

**kill (pid\_t pid, int signum);**

****

|  |  |  |
| --- | --- | --- |
| **1. pipe** | **5. dup2** | **10. pipe** |
| **2. fork** | **6. exec** | **11. fork** |
| **3. dup2** | **7. dup2** | **12. fork** |
| **4. exec** | **8. exec** | **13. dup2** |
|  | **9. dup2** | **14. exec** |

**Q9**

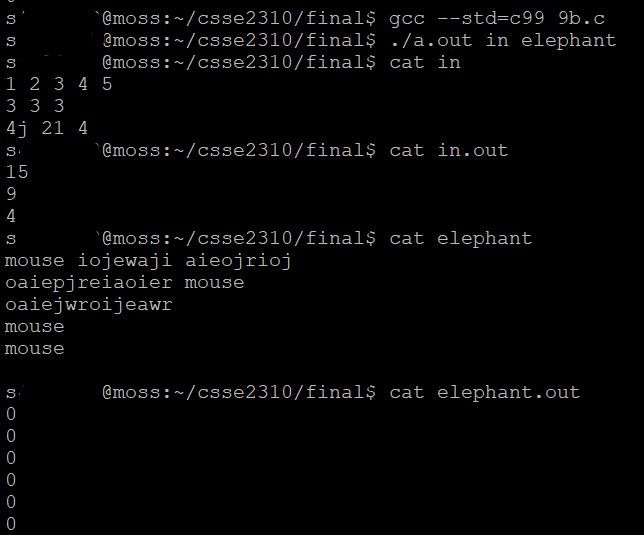
A) Write a function int sumline(const char\* line) which adds up the space separated integers in a string. sumline("12 2 3 5") would return 22. The function should process as much of the string as it can, but it will not move past a character which is not a space or part of a base 10 integer.

B) Write a function void sumfile(FILE\* in, FILE\* out) which sums the integers on each line in in and outputs each sum to a new line of out. You may call the sumline function from the previous part and assume that it works correctly.

C) Write a program which takes one or more filenames as commandline parameters. For each filename f, create an output file f.out and sums up the lines in f and stores the results in f.out. You may call the sumfile() function from the previous part and assume that it works correctly. Your program should use a separate process to deal with each input file. If any input/output pair can not be opened, continue processing the other pairs.

**Person 1: A/B/C**

**//I tested this & it works (if you wanna test it make sure to include libraries)**

****

#define LINE\_LENGTH 81

#define FILE\_INDEX 1

**int sumline(const char\* line) {**

int total = 0, i = 0, num = 0;

char current, delim = ' ';

while (1) {

current = line[i++];

if (current == delim) { // end of number

total += num;

num = 0;

} else if (isdigit(current)) {

num \*= 10;

num += (current - '0');

} else { // invalid or end of string

total += num;

break;

}

}

return total;

}

**void sumfile(FILE\* in, FILE\* out) {**

// assume in & out != NULL

char line[LINE\_LENGTH];

while (!feof(in)) {

if (!fgets(line, LINE\_LENGTH, in)) {

break;

}

fprintf(out, "%d\n", sumline(line));

}

fflush(out);

}

**int main(int argc, char\*\* argv) {**

int totalFiles = argc - FILE\_INDEX, status;

pid\_t\* pids;

if (!totalFiles) {

return 1; //no files given

}

pids = (pid\_t\*) malloc(sizeof(pid\_t) \* totalFiles);

for (int i = 0; i < totalFiles; i++) {

pids[i] = fork();

if (!pids[i]) { //child

handle\_file(argv[i + FILE\_INDEX]);

return 0;

}

}

for (int i = 0; i < totalFiles; i++) {

waitpid(pids[i], &status, 0);

}

free(pids);

return 0;

}

**void handle\_file(char\* file) {**

**// in the spirit of 2310’s 50 line limit… made a helper function for part c :)**

FILE\* in, \*out;

char\* outFileName;

in = fopen(file, "r");

if (!in) { //invalid in file

return;

}

outFileName = (char\*) malloc(sizeof(file) + sizeof(".out\0"));

sprintf(outFileName, "%s.out", file);

out = fopen(outFileName, "w");

free(outFileName);

if (!out) { //invalid out file

return;

}

sumfile(in, out);

fclose(in);

fclose(out);

}

**Person 2: A/B/C**

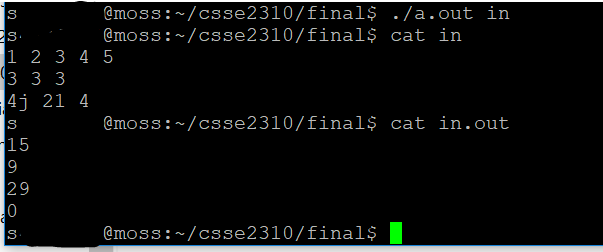
**YOU HAVEN’T TESTED THIS - It doesn’t work**

**in.out should look like:**

15

9

4

****

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>

int sumline(const char\* line) {

// Get our own copy of the string

char \*stringIn = (char \*) calloc(strlen(line) + 1, sizeof(char));

strcpy(stringIn, line);

int index = 0, sum = 0, numIn = 0;

char \*prev = stringIn, \*next = NULL;

while (1) {

if (next != NULL) {

prev = stringIn + index + 1;

}

index++;

while ((stringIn[index] != ' ') && (stringIn[index] != '\0')) {

index++;

}

if (stringIn[index] == '\0') {

if (sscanf(prev, "%d", &numIn) == 1) {

sum += numIn;

}

break;

}

next = stringIn + index;

stringIn[index] = '\0';

if (sscanf(prev, "%d", &numIn) != 1) {

break;

}

sum += numIn;

}

return sum;

}

void sumfile(FILE\* in, FILE\* out) {

char \*lineIn = (char \*) calloc(80, sizeof(char)), byte = -1;

for (int index = 0; !feof(in); memset(lineIn, 0, 80), index = 0) {

while (((byte = fgetc(in)) != '\n') && !feof(in)) {

lineIn[index] = byte;

index++;

}

fprintf(out, "%d\n", sumline(lineIn));

fflush(out);

}

free(lineIn);

return;

}

int main(int argc, char \*\*argv) {

for (int index = 1; index < argc; index++) {

if (!fork()) {

// Child

FILE \*in, \*out;

if ((in = fopen(argv[index], "r")) == NULL) {

exit(1);

}

char \*stringOut = (char \*) calloc(strlen(argv[index]) +

strlen(".out") + 1, sizeof(char));

strcpy(stringOut, argv[index]);

strcat(stringOut, ".out");

out = fopen(stringOut, "w");

sumfile(in, out);

free(stringOut);

fclose(in);

fclose(out);

exit(0);

}

}

exit(0);

}

ATTEMPT 3 (WORKS)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int sumline(const char\*);

void sumfile(FILE\*, FILE\*);

/\*

\* Reads filenames off commandline and makes a new child

\* for each which processes the file given from the command

\* line and outputs it to a file called filename.out

\* (where filename is the filename given to the child).

\* Returns 1 if no filenames given. All children die,

\* but never checks.

\*/

int main(int argc, char\*\* argv) {

if (argc < 1) {

fprintf(stderr, "Usage: ./q1 filename filename ...\n");

return 1; // Not enough file names given

}

for (int i = 1; i < argc; i++) { // Go through every filename given

if ((fork()) == 0) {

// Child

// Build the ouput name, +5 for ".out\0"

char\* outputName = malloc(sizeof(char) \* strlen(argv[i]) + 5);

FILE\* input = fopen(argv[i], "r");

if (input == NULL) { // Couldn't open it

free(outputName);

return 0;

}

sprintf(outputName, "%s.out", argv[i]);

FILE\* output = fopen(outputName, "w"); // Creates the file

sumfile(input, output);

free(outputName);

return 0; // Every child should die; never asked to ensure they do

}

}

return 0;

}

/\*

\* Function which takes two FILE pointers, and reads from the

\* first line by line outputting the sum of the integers on

\* the line read to the file specified by out.

\* Will stop once the end of file has been reached in

\* file in.

\*/

void sumfile(FILE\* in, FILE\* out) {

char buffer[80]; // Q says we can assume max 79 chars + 1 for \0

while (1) {

if (fgets(buffer, 80, in) == NULL) { // Read a line

break; // returns NULL on EOF, exit here

} // Feof won't work here as it won't be set until the next bad read

fprintf(out, "%d\n", sumline(buffer));

fflush(out);

}

fclose(in);

fclose(out); // Close these here

}

/\*

\* Function which takes a single nul terminated string,

\* and sums space separated integers until a non digit/space char.

\* Uses strtol to convert the value, and checks ahead of each

\* parsed int for only spaces and ints; if so, continues, if not,

\* breaks and returns.

\*

\* As per the exam question, this "Adds up space separated integers

\* in a string, processing as much as it can but will stop on

\* characters that are not a space or a base 10 integer".

\* Never specified how many spaces could be in between, and the wording

\* implies it keeps reading n spaces ("Not move past a character which

\* is not a space" -> moves past spaces always).

\*/

int sumline(const char\* line) {

char\* editLine = malloc(sizeof(char) \* strlen(line) + 1); // +1 for \0

strcpy(editLine, line);

int sum = 0;

char\* memory = editLine; // Hacky way to free later lol

char\* ptr;

do { // Do this until the end of the string ideally

int index = 0, valid = 1;

while (1) { // Parse the next piece of the string until next digit

if (isdigit(editLine[index])) {

break; // Found a digit and nothing else returned

} else if (editLine[index] == ' ') {

index++;

continue; // Read past spaces

} else { // It's an invalid character, stop here

valid = 0; // Set flag

break;

}

}

if (!valid) {

break; // Above piece of string had non legal chars; exit loop

} // and stop processing this string

sum += (int) strtol(editLine, &ptr, 10); // Above check ensures

// strtol has at least 1 digit

editLine = ptr; // Move string forward to keep processing

} while (\*ptr != '\0'); // Fixes valgrind issue of uninit

free(memory);

return sum;

}