**CSSE2310: 2017 exam cancers**

## [**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.**

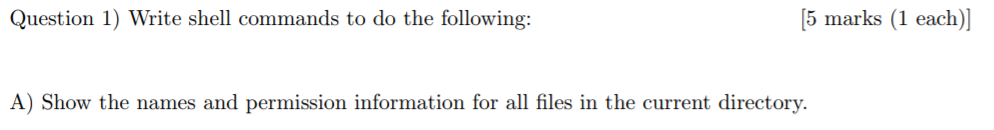
### Type answers in blue beneath each question.

### 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.



ls -al

ls -l should alright just fine, cause it is just asking for all files “in the current directory”, no need to show the permission inform for neither the current directory nor its parent directory.

Ls -la is required to show hidden files starting with a .

I disagree here: its asking for “files” … not sure if we are allowed to include directories [which ls -la gives directories as well so is incorrect imo]

Under unix, all directories are files. That is, all directories are just files that contain lists of other files. +1



let count=0

for i in \*.c; do let count=count+1; done;

echo $count

^^ works but feel like there is an easier way

easier way:

ls -l \*.c | wc -l

#wc -l counts the number of lines, and it has white space. To remove them:

ls -l \*.c | wc -l | tr -d ' '

ls -1 \*.c | wc -l # this works really nicely



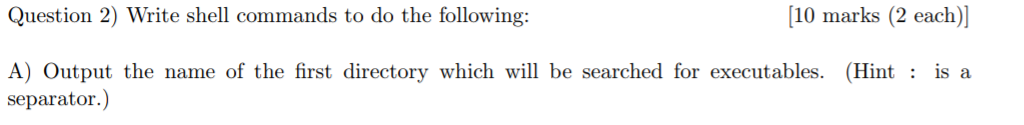
grep chip bill | head -1



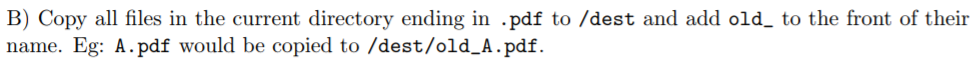
cat bill | grep chip | grep fish



cat dict | tail -4 | sort



echo $PATH | cut -d ‘:’ -f 1

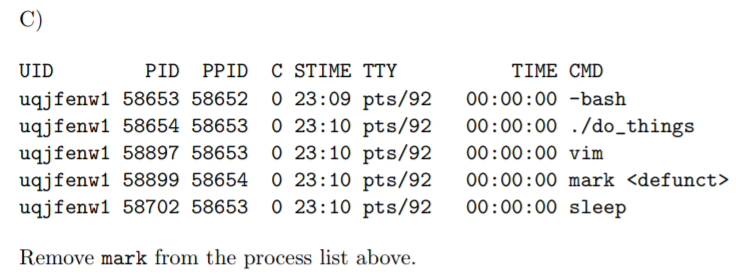


for file in \*.pdf

do cp $file ./dest/old\_$file

done;

#tested working answer [tested + 3]



wait 58899

I think this is case specific, and wait doesn’t do it:

s4493939@moss:3$ ps l

F UID PID PPID PRI NI VSZ RSS WCHAN STAT TTY TIME COMMAND

0 2108318 7604 120763 20 0 153196 1492 - R+ pts/5 0:00 ps l

0 2108318 120763 120750 20 0 116128 2068 do\_wai Ss pts/5 0:00 -bash

0 2108318 127630 120763 20 0 4364 528 do\_sig T pts/5 0:00 ./austerity 10 10 deck ./bad\_program ./bad\_program

0 2108318 127631 127630 20 0 0 0 do\_exi Z pts/5 0:00 [bad\_program] <defunct>

0 2108318 127632 127630 20 0 0 0 do\_exi Z pts/5 0:00 [bad\_program] <defunct>

s4493939@moss:3$ wait 127631

-bash: wait: pid 127631 is not a child of this shell

The only way I could do it was killing the parent process by force (waiting on it also doesn’t work):

s4493939@moss:3$ kill -9 127630  
s4493939@moss:3$ ps l  
F UID PID PPID PRI NI VSZ RSS WCHAN STAT TTY TIME COMMAND  
0 2108318 9378 120763 20 0 153196 1492 - R+ pts/5 0:00 ps l  
0 2108318 120763 120750 20 0 116128 2068 do\_wai Ss pts/5 0:00 -bash  
[1]+ Killed ./austerity 10 10 deck ./bad\_program ./bad\_program  
s4493939@moss:3$ ps l  
F UID PID PPID PRI NI VSZ RSS WCHAN STAT TTY TIME COMMAND  
0 2108318 9493 120763 20 0 153196 1492 - R+ pts/5 0:00 ps l  
0 2108318 120763 120750 20 0 116128 2068 do\_wai Ss pts/5 0:00 -bashpp

I think the answer lies in doing something (either kill or kill -9) with the parent process, not interacting with the zombie cos it’s not your child and you can’t kill it.

So `kill -9 58654` then? +1 this worked also kill 58654



svn diff thing.c fred.c

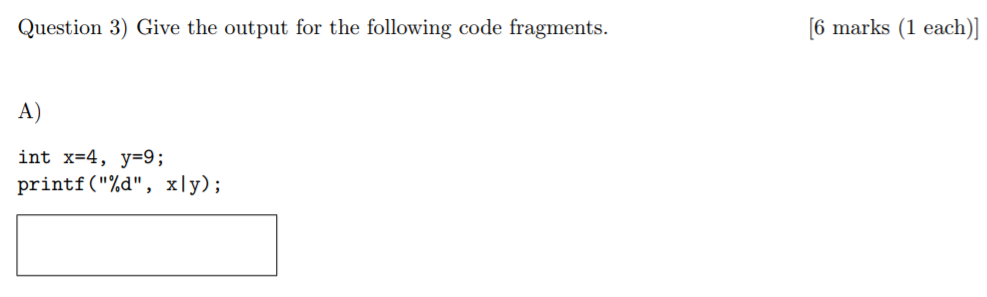


svn rm old.h

svn commit -m “” <- could you just do svn commit? Do you need the -m flag?

You don’t need the -m flag.

d



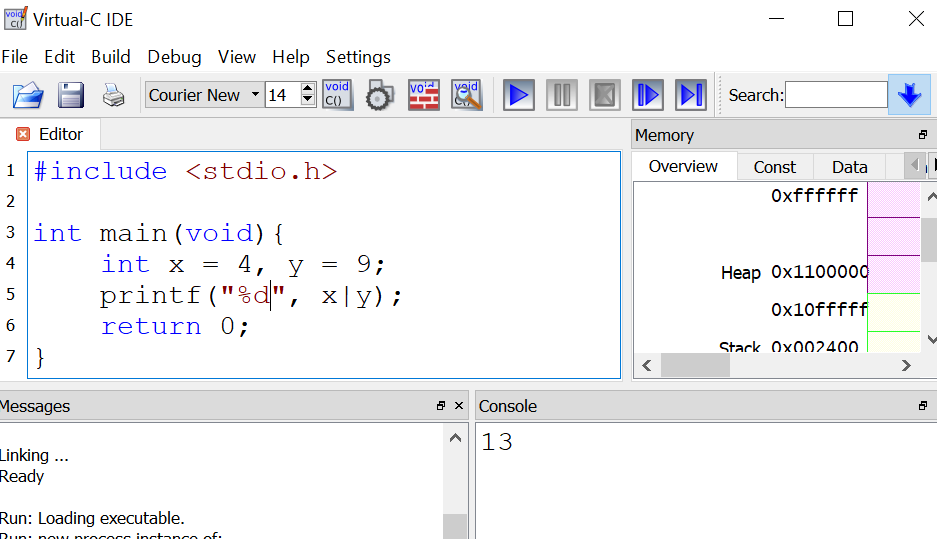
13 [+2]

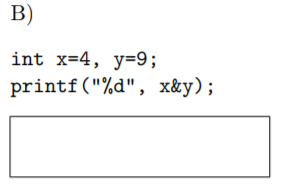
Why is this not 15? Because C.

0100

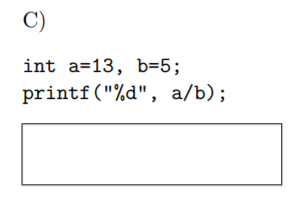
| 1001

=1101 = 13

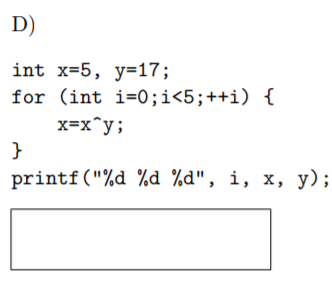




0 [+3]

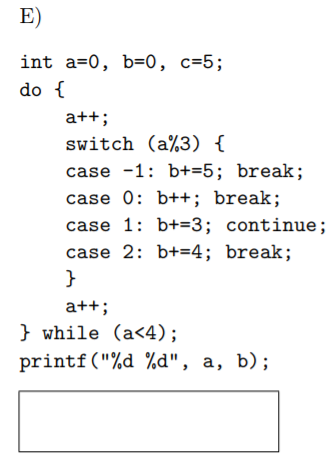


2 [+1]

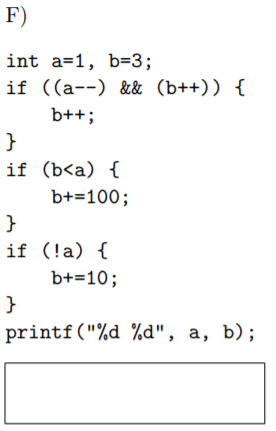


Error -> int i is out of scope of the print statement [+5]

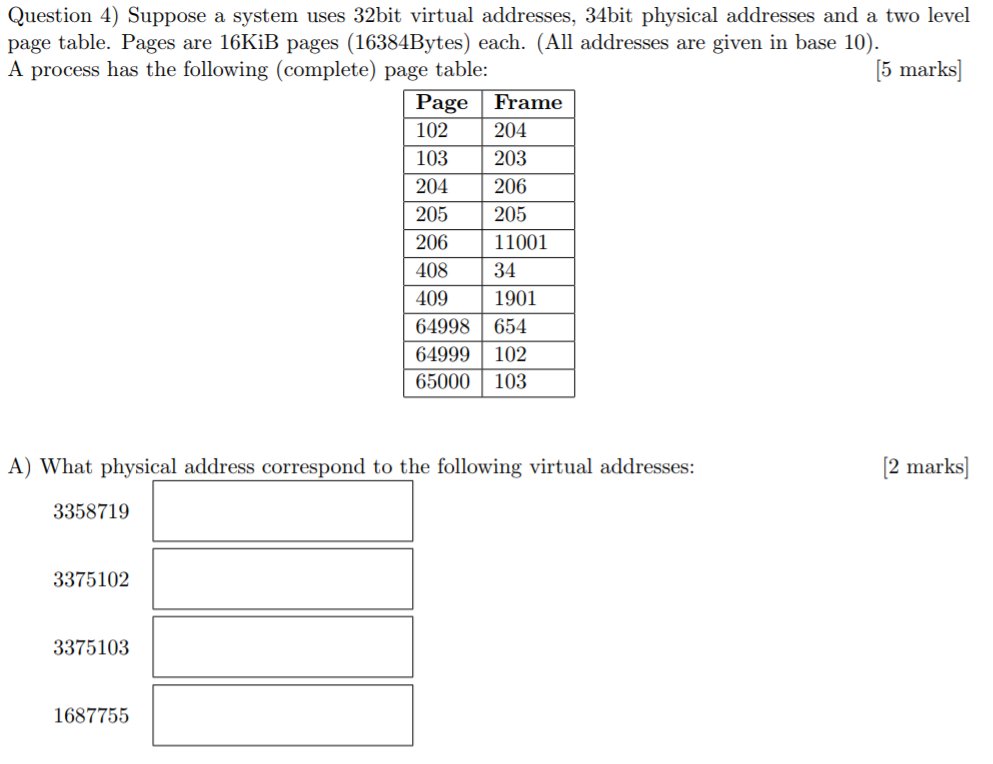
<https://repl.it/@WiseBaldOne/ForestgreenPiercingCodes> I agree with purple.



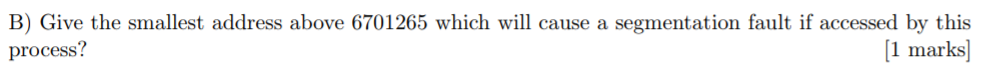
4 10



0 15



1. ~~3,391,493~~ 3,391,487 [+3]
2. 3,375,102 [+1]
3. 3,375,103 [+1]
4. 3,326,155 [+1]

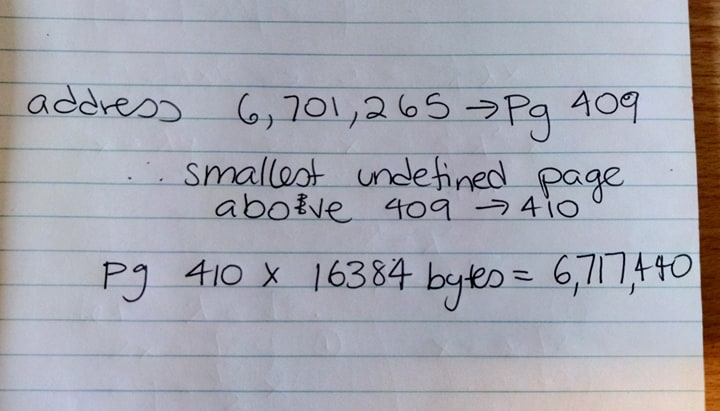
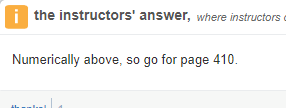


6,717,440

Lemme know if the calculation is wrong but :) [+2]

in the question it says 'above' so pwhy are we taking page after the 409? Shouldn't we take it above 409 that will be 407

407 \* 16384 = 6,668,288 <- proved to be wrong as it is asking for the “address above 6701265” instead of address above 409…..

Answer from tutor on piazza ---> 

i) 2 (VM 204)

ii) 2 (VM 205)

iii) 1 (VM 205)

iv) 2 (VM 103)

^Anyone know how to do this <3. Yes (look at link)

<https://uni.joeladdison.com/csse2310/virtual-memory/memory-accesses> <- shows how to do this but it suggests that for a 2 page table the answers should be:

i) 3 accesses but only 2 pages accessed; Ans = 2 [+4]

[access from 1st page level; access from 2nd page level; access to physical memory]

ii) 3 accesses but only 2 pages accessed; Ans = 2 [+4]

[access from 1st page level; access from 2nd page level; access to physical memory]

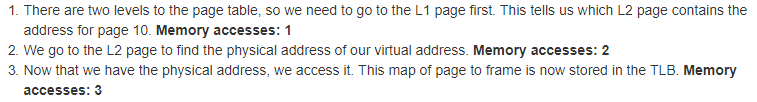
iii) 1 access but doesn’t look at page table; Ans = 0 [+6]

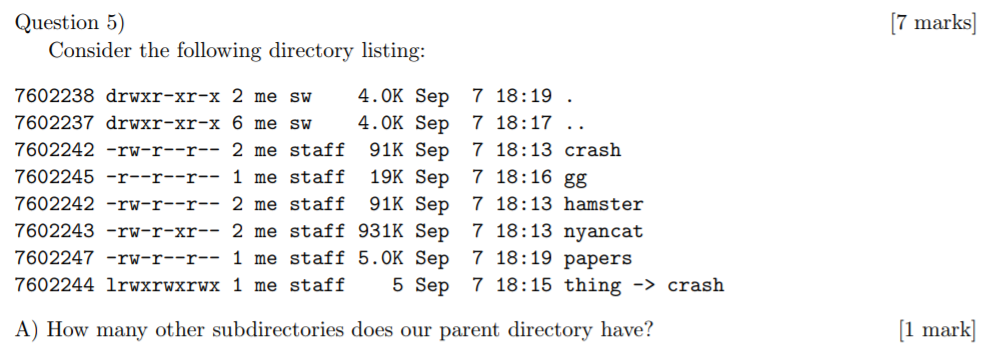
[access to physical memory (since stored in TLB)]

iv) 3 accesses but only 2 pages accessed; Ans = 2 [+4]

[access from 1st page level; access from 2nd page level; access to physical memory]

SCREEN SHOT FROM ABOVE SITE ^^





(3 assuming that this current directory is not in “other”)

6 - 2 (the . and ..) - 1 (current dir) = 3 [+2]



no one (thing is a soft link to crash which is not executable) [+1]



Group staff, not me, not everyone [+1]

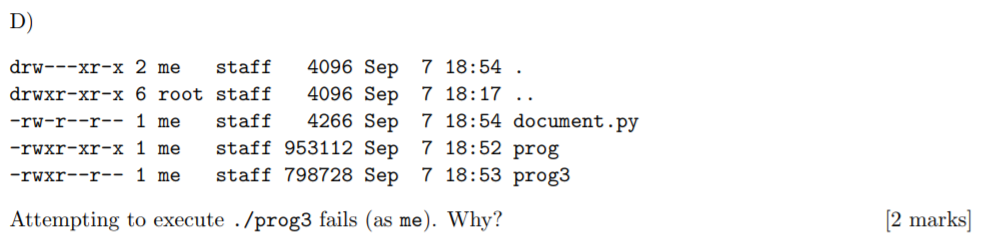


the files gg (7602245), hamster (same as crash, 7602242) as well as papers (7602247) will be lost. That’s because nyancat has 2 references but the other one is not in the same directory, and thing is just a soft link to crash so doesn’t count.

That makes **19K + 91K + 5K = 115 KB**

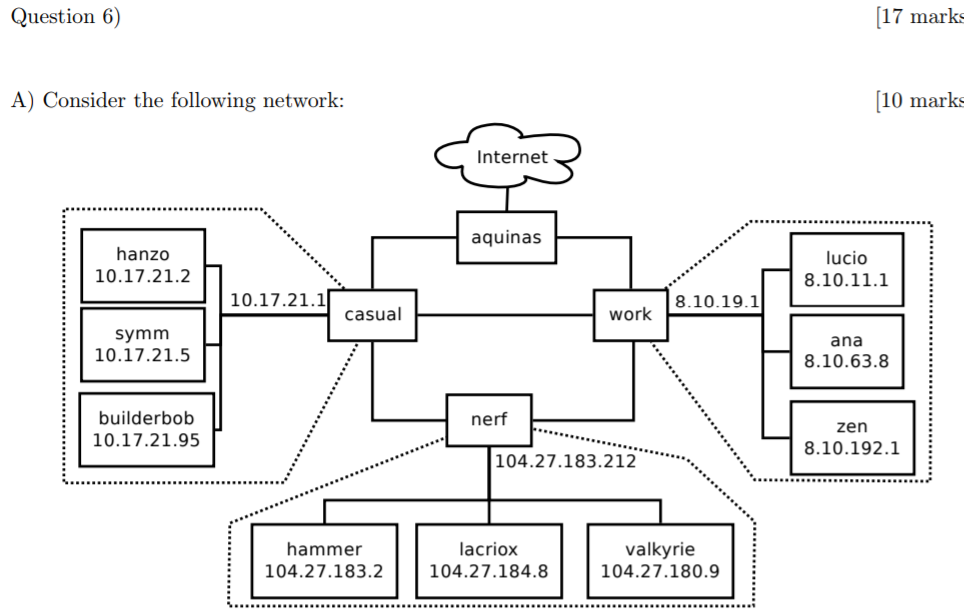
**Seems to me the directory is 4.0K and thing is 5 bytes so it will be 119K + 5 = 121861 bytes(one is current directory and one is parent directory)**

My dad beats me



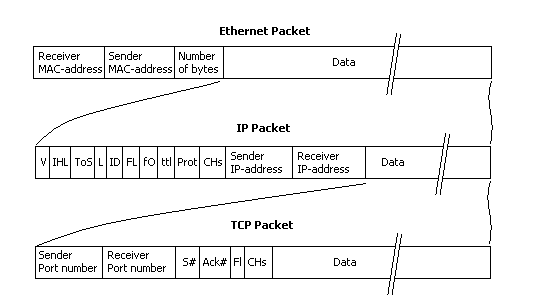
Prog3 may be set with executable permissions but not actually be an executable program.

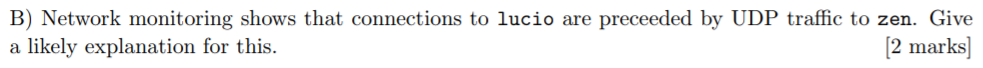
You can’t access anything in a directory without x, even if you know the directory’s name. [+1]



|  | Broadcast address | Network Mask | CIDR |
| --- | --- | --- | --- |
| Casual | 10.17.21.127 | 255.255.255.128 | 10.17.21.0/25 |
| Work | 8.10.255.255 | 255.255.0.0 | 8.10.0.0/16 |
| Nerf | 104.27.191.255 | 255.255.240.0 | 104.27.176.0/20 |

| TYPE | A | B | DEST MAC | SOURCE IP | DEST IP |
| --- | --- | --- | --- | --- | --- |
| TCP segment | aquinas | Hammer | 3 | 1 [+3] 2[+5] It | 1[+3] |
| UDP datagram | aquinas | Hammer | 3 | 1[+2] 2[+4] | 1[+3] |
| Ethernet frame | zen | work | 1 | 0 [+6] | 0 [+6] |
| TCP segment | symm | uq.edu.au | Leave blank | 2 | 1 |





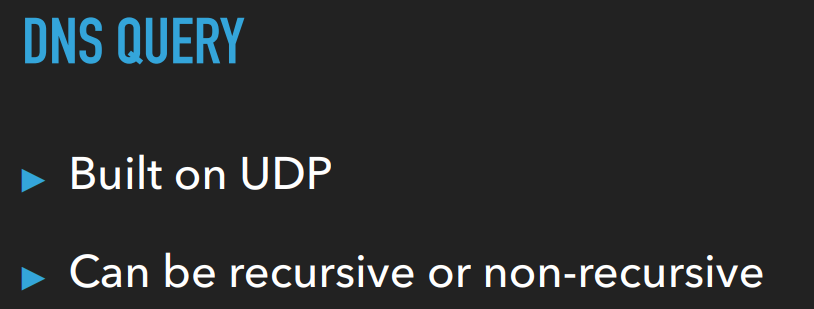
UDP has no error checking and is just streamed information. Packets therefore get processed faster. Please update if incorrect.

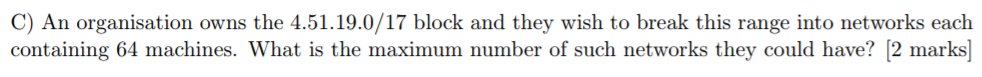
I think it’s because zen is a DNS server containing the information about lucio’s IP, so incoming messages ask zen using UDP what lucio’s IP is before they talk to lucio. Zen is just a client machine unless otherwise specified - how did you figure out zen is a DNS server? I think the question is hinting at it simply because DNS requests are in UDP from what I remember of the lecture and that is the best explanation I can think of for why the zen traffic always occurs before lucio (i.e. it’s a “likely explanation”). Also not sure why we’d assume client. I see your point. I think you’re right. Wtf lmao I feel like UDP being processed faster is still a decent answer? Idk about this cooked shit.

I also found on wikipedia that since UDP has no handshaking or even direct connection to a specific address (it’s said to be a connectionless protocol). Meaning that in the nature of UDP, it’s possible for the messages to lucio to be broadcasted to the subnets of work. But not 100% sure.

**(^^^So should we consider the pink font to be the right answer for this??)**

**Guys: Pink is Right [+1]**

****

****

**(2^(32-17) - 2)/64 = 511**

**(am i right? I subtracted by 2 since we have 2 reserved IP addresses; so the number of networks they could use is 511) [+1]**

**(is the reserved IP address unique?)**

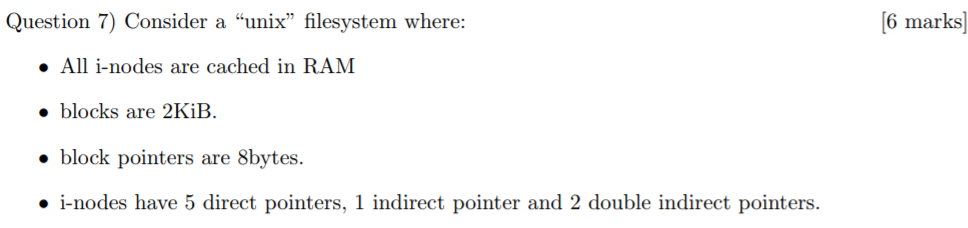
**2^(32-17)/128 = 256 networks**

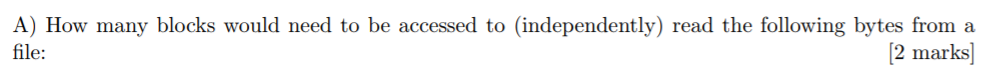
**It asks for 64 machines, which I dont think considers network and broadcast. Therefore you need at least 66 IP addresses for each network block. The next largest block of IP addresses you can have is 128. [+5]**

**^^ Joel said that this is correct**

****

| **SSH** | **5** |
| --- | --- |
| **Port** | **4** |
| **MAC** | **3 & 2** |
| **routing** | **3** |
| **putty** | **5** |
| **ethernet** | **2** |

****

****

**Ptrs/ Block = 28**

**Block # before double pointer = 5 + 28 = 261 block**

**10,200: ~~2 reads~~ 1 read [+3]**

~~10,200 / 256 = 39th block~~ 10,200 / 2,048 = 4th

block Block # before double indirect pointer = 5 + 2**8** = 261 block



**10,205: ~~2 reads~~ 1 read [+2]** Wouldn’t this be 0 reads? [cached ….] No caching. Pls elaborate Question says each access is independent, implying that there is no caching. how about it saying “all inodes are cahced in ram?” The i-node is the first thing with the 5 direct, 1 indirect and 2 double indirect, not the blocks of pointers. I see. Last q: if it doesn’t say “independpent” do we assume that there is caching? In the revision lecture they said you do assume I believe.

~~10,205 / 256 = 39th block~~ 10,205 / 2,048 = 4th block

^^Would this need a read? This block was read just prior?

^^Says independently - i node cached but reads are not

^^ Thanks!

^^ No problems - I don’t see how its just 1 read though…

^^Because it’s a direct pointer block. So as soon as you access this 1 block you get your data and in the question it has asked for the number of BLOCKS accessed.

10205+(8\*4) / 2048 = 4th block. I think it makes sense to include the pointer size of each of the pointers?

**51,025: 2 reads [+3]**

~~51025/ 256 = 199th block~~ 51025 / 2048 = 24th block

Inode -> ptr -> data

**102,050: ~~3 reads~~ 2 reads [+2]**

~~102,050/ 256 = 398th block~~ 102050 / 2048 = 49th block

inode -> ptr -> ~~ptr ->~~ data

**Doesn’t it say that the inode is cached in RAM? (so ans = 1, 0, 2 ,1) [+1]**

**Question says independently so no caching**

**inode is cached, loading blocks is not cached (independently). (1, 1, 2, 2) [+2]**

****

**5/1/2 file storage**

**Ptrs/Block = 2\*\*8**

**Why is 2^11?**

**[(5 + 28 + (2 \* 28 \* 28))] \* 211**

**= 268,969,984 bytes [+8]**

**If we include the size of pointers I think we’d have**

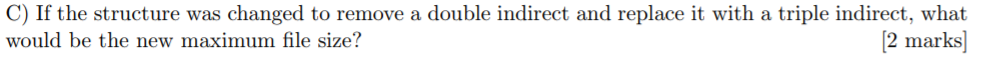
**[(5 + 28 + (2\*28 \* 28))] \* 211 - (23 \* 8) - (23 \* 28) - (23 \*2\* 28 \* 28)**

**(size of 8 pointers in 1st table) - size of 28 pointers in first indirect**

**- size of 28 \* 28 pointers in double indirect**

**=268,969,984 - 1,050,688 =267,919,296. EDIT: tutors confirmed don’t do this. Pointer size is just overhead and not our problem.**

**I don’t think you have to subtract off the pointer space because the original calculation isn’t including them anyway.**

****

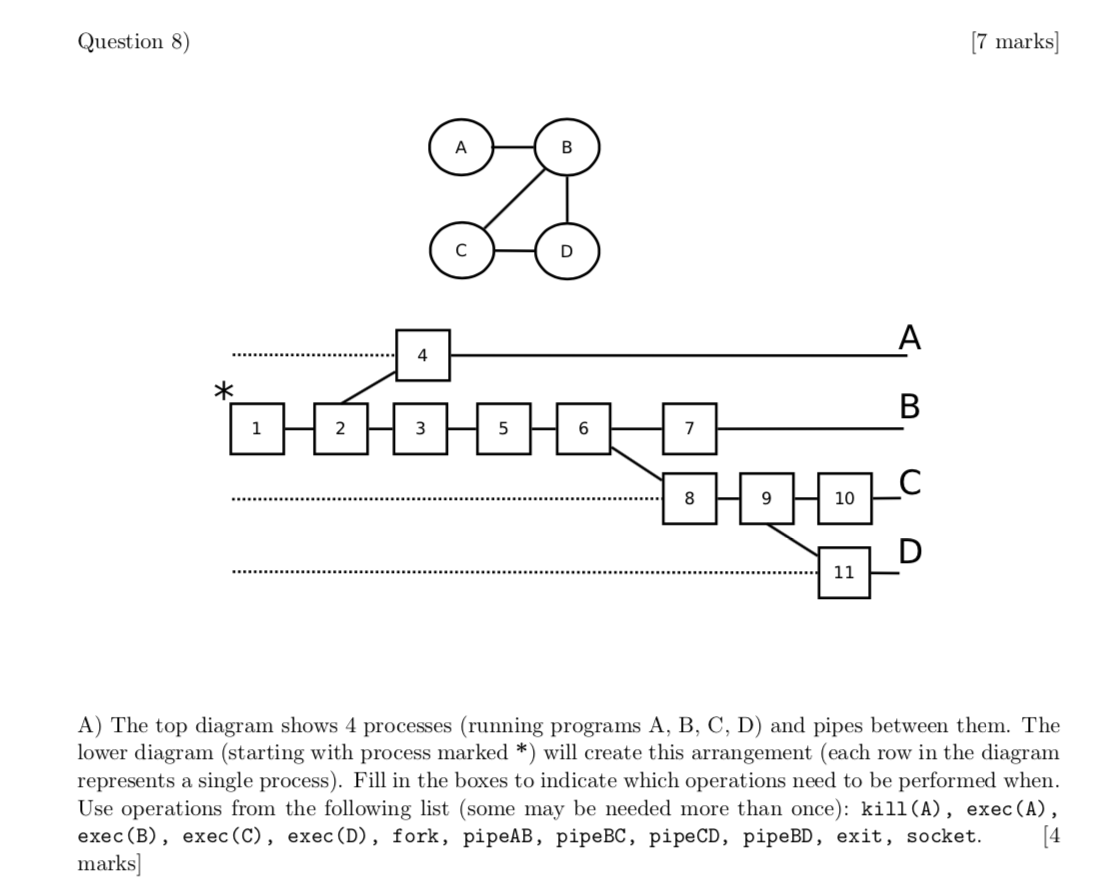
**[5 + 28 + (28 \* 28) + (28 \* 28 \* 28)] \* 211**

**= 34,494,490,624 bytes [+8]**

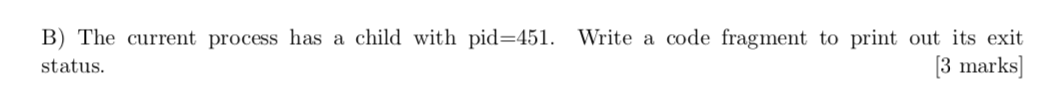
**Likewise = 34,494,490,624 - 134,744,128 = 3.4260\*10^10 bytes [-1]**

**????**

**5\*2048+1\*2048\*256+1\*2048\*256\*256+1\*256\*256\*256\*2048 / 1024 = 33,686,026 KiB [+1]**

**^ Same answer, just in KiB**

1. **pipeAB**
2. **fork**
3. **pipeBC**
4. **exec(A)**
5. **pipeBD**
6. **fork**
7. **exec(B)**
8. **pipeCD**
9. **fork**
10. **exec(C)**
11. **exec(D) [+1]**

****

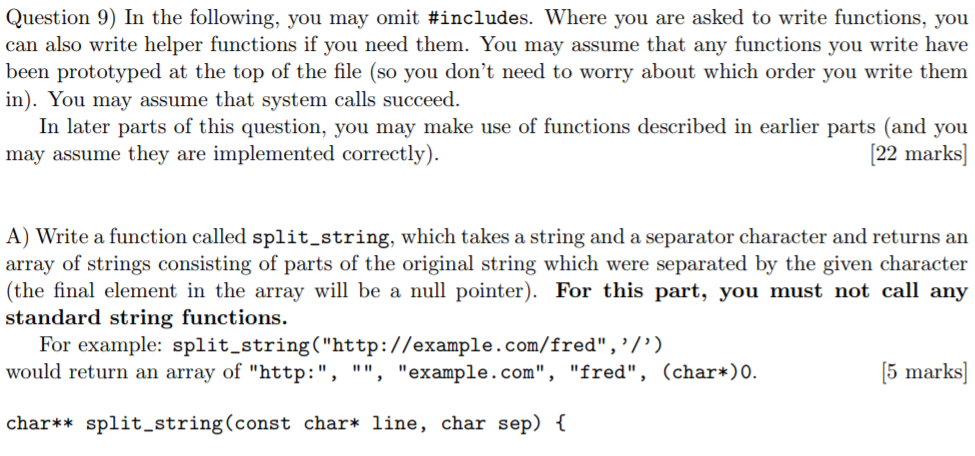
**int status;**

**waitpid(451, &status, 0);**

**if (WIFEXITED(status)) {**

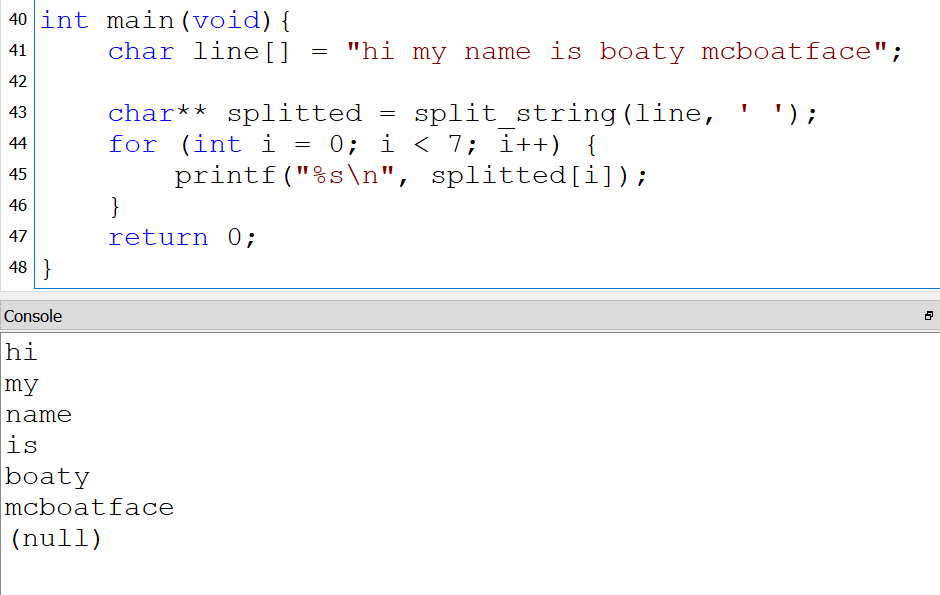
**printf("%d\n", WEXITSTATUS(status));**

**} [+1]**

****

**//assumes line != NULL**

**//tested and works**

****

**char\*\* separated;**

**int wordCount = 0, letterCount = 0;**

**char\* word;**

**separated = (char\*\*) malloc(sizeof(char\*));**

**word = (char\*) malloc(sizeof(char));**

**for (int i = 0; line[i] != ‘\0’; i++) {**

**if (line[i] == sep) {**

**word[letterCount] = '\0';**

**separated[wordCount++] = word;**

**separated = (char\*\*) realloc(separated, sizeof(char\*) \* (wordCount + 1));**

**letterCount = 0;**

**word = (char\*) malloc(sizeof(char));**

**} else { //continue word**

**word[letterCount++] = line[i];**

**word = (char\*) realloc(word, sizeof(char) \* (letterCount + 1));**

**}**

**}**

**// add most recent word**

**word[letterCount] = '\0';**

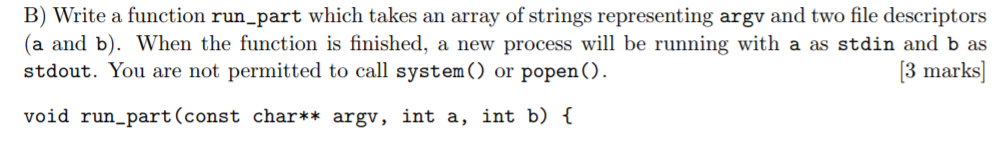
**separated[wordCount++] = word;**

**separated = (char\*\*) realloc(separated, sizeof(char\*) \* (wordCount +1));**

**separated[wordCount] = NULL;**

**return separated;**

**}**

****

**void run\_part(const char\*\* argv, int a, int b) {**

**// error checking - probably not actually needed for mark**

**if(a < 0 || b < 0 || !argv) {**

**return;**

**}**

**if(!fork()) { // (fork() can be -1 so it may be necessary to do error check)**

**dup2(a, STDIN\_FILENO);**

**dup2(b, STDOUT\_FILENO);**

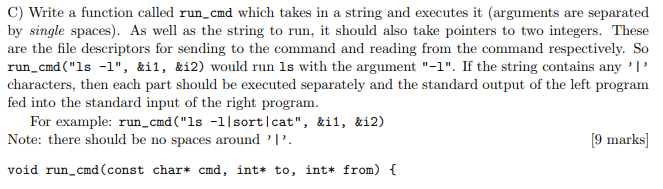
**close(a); // not sure if these are needed**

**close(b);**

**execvp(argv[0], argv);**

**}**

**}**

****

**// i have no idea if this would work**

**#define READ 0**

**#define WRITE 1**

**void run\_cmd(const char\* cmd, int\* to, int\* from) {**

**if(&from < 0 || &to < 0 || !cmd) {**

**return;**

**}**

**const char\*\* cmds = split\_string(cmd, ‘|');**

**int i = 0;**

**int childIn = \*to, childOut;**

**while(cmds[i]) {**

**int fd[2];**

**pipe(fd); // Create pipe between commands seperated by |**

**childOut = fd[WRITE];**

**if(!fork()) { // child**

**if(i == cmdLen - 1) { // ending pipe**

**childOut = \*from;**

**}**

**run\_part(split\_string(cmds[i], “ “), childIn, childOut);**

**} else { // parent**

**childIn = childOut; // redirect child output to next child input**

**}**

**i++;**

**}**

**}**

****

**Michael’s Solution**

I think this is what they had in mind. Not sure how people are meant to do this in 20min. Someone pls try and find a better solution (possibly recursion on the fds?).

/\*\*

\* Enumerates reading and writing for pipes

\*/

enum ReadAndWrite {

READ = 0,

WRITE = 1

};

/\*\*

\* A function used to split commands separated by sep (this

\* is a little different to string\_split since this function will also give

\* me the number of substrings and end the array with a null character).

\*

\* const char \*line:

\* The line that we want to separate.

\*

\* char sep:

\* The character use to separate the string.

\*

\* char \*\*output:

\* A pointer to an array of string where the output will be stored.

\*

\* int \*numSub:

\* A pointer to an integer that will store the number of substring

\* when splitting.

\*/

void split\_and\_count(const char \*line, char sep, char \*\*\*output, int \*numSub) {

// Create an array of string to store all the split strings

int numSubstrings = 1, index = 0, stringLen = 1;

char \*\*substrings = (char \*\*) calloc(numSubstrings, sizeof(char \*));

char byte = -1;

substrings[0] = (char \*) calloc(stringLen, sizeof(char));

while ((byte = line[index]) != '\0') {

if (byte != sep) {

stringLen++, index++;

substrings[numSubstrings - 1] =

realloc(substrings[numSubstrings - 1], stringLen \* sizeof(char));

substrings[numSubstrings - 1][stringLen - 2] = byte;

substrings[numSubstrings - 1][stringLen - 1] = '\0';

} else {

numSubstrings++, stringLen = 1;

substrings = realloc(substrings, numSubstrings \* sizeof(char \*));

substrings[numSubstrings - 1] = (char \*) calloc(stringLen, sizeof(char));

index++;

}

}

// End with a NULL character

numSubstrings++;

substrings = realloc(substrings, numSubstrings \* sizeof(char \*));

substrings[numSubstrings - 1] = NULL;

\*output = substrings;

// Exclude NULL

if (numSub != NULL) {

\*numSub = (numSubstrings - 1);

}

return;

}

void run\_cmd(char\* cmd, int\* to, int\* from) {

// Separate the command out into an array

char \*\*split;

int numSubString, index = 0;

split\_and\_count(cmd, '|', &split, &numSubString);

// Set up a list for our pipe

int conn[2], pendingRead = -1;

pid\_t child\_id;

for (index = 0; index < numSubString; index++) {

if (pipe(conn) == -1)

err(1,"pipe");

if ((child\_id = fork()) == 0) {

// There's nothing to read from

if (index == 0) {

close(conn[READ]);

if(dup2(\*from, STDIN\_FILENO) == -1)

err(1,"dup2 child grep");

} else {

if(dup2(pendingRead, STDIN\_FILENO) == -1)

err(1,"dup2 child grep");

}

// Output to Stdout if this is the last command

if ((index + 1) == numSubString) {

close(conn[WRITE]);

if(dup2(\*to, STDOUT\_FILENO) == -1)

err(1,"dup2 child");

} else {

// Else write to the next program

if(dup2(conn[WRITE], STDOUT\_FILENO) == -1)

err(1,"dup2 child");

}

// Separate the program string into a list containing the program

// name at the start followed by prog arguments

char \*\*program;

split\_and\_count(split[index], ' ', &program, NULL);

execvp(program[0], program);

err(1, "exec child");

}

if (pendingRead != -1) {

close(pendingRead);

}

close(conn[WRITE]);

if ((index + 1) == numSubString) {

close(conn[READ]);

} else {

// We have something to read from now

pendingRead = conn[READ];

}

}

}

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

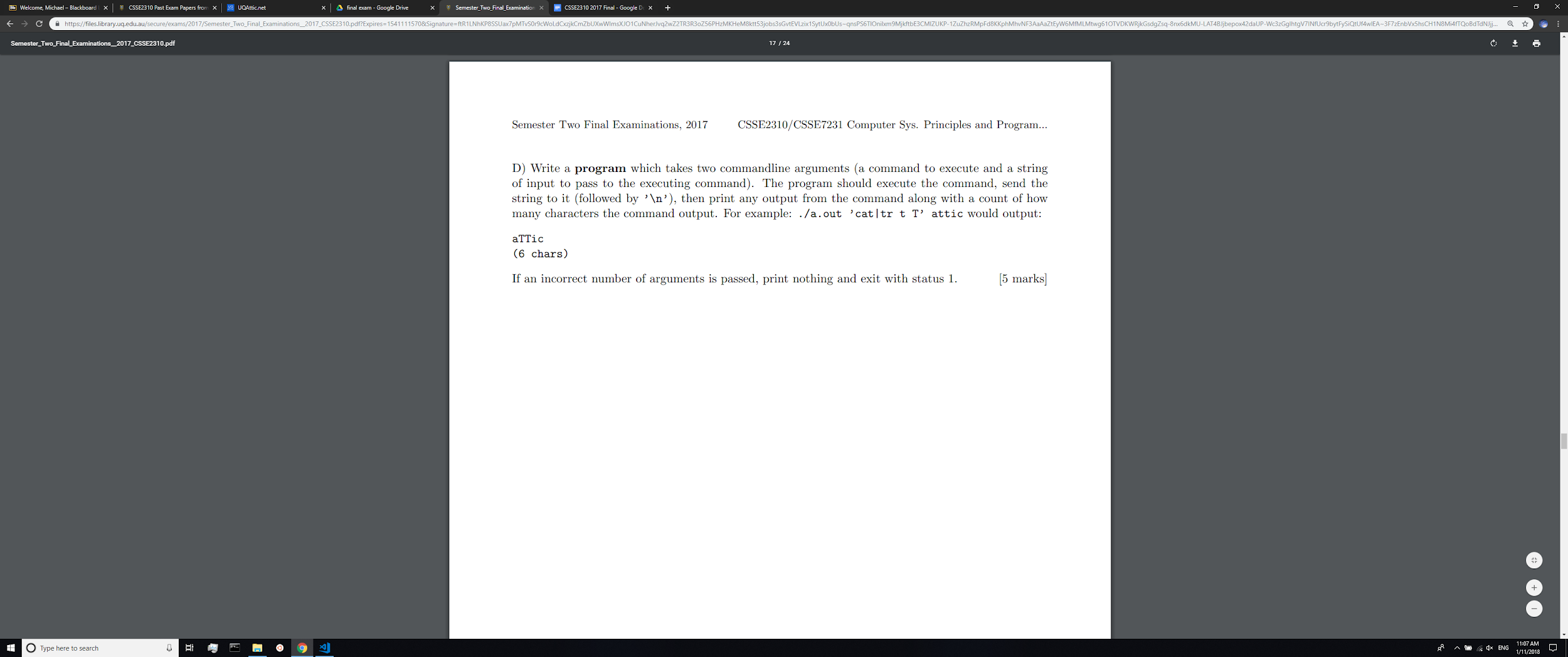
int in = STDIN\_FILENO, out = STDOUT\_FILENO;

// Test

run\_cmd("ls|sort", &out, &in);

return 0;

}



// no idea if this would work

#define DEFAULT\_BUFFER 10

char\* get\_line(FILE\* f) {

int size = DEFAULT\_BUFFER, i = 0, c;

char\* line = malloc(sizeof(char) \* DEFAULT\_BUFFER);

while(!feof(f)) {

if(i > size - 2) {

size \*= 2;

line = realloc(line, sizeof(char) \* size);

}

line[i++] = fgetc(f);

}

if(i) {

line[i] = ‘\0’;

return line;

}

return NULL;

}

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

if(argc != 3) {

return 1;

}

int fdIn[2], fdOut[2];

pipe(fdIn);

pipe(fdOut);

if(!fork()) {

run\_cmd(argv[1], &fdOut[READ], &fdIn[WRITE]);

exit(0);

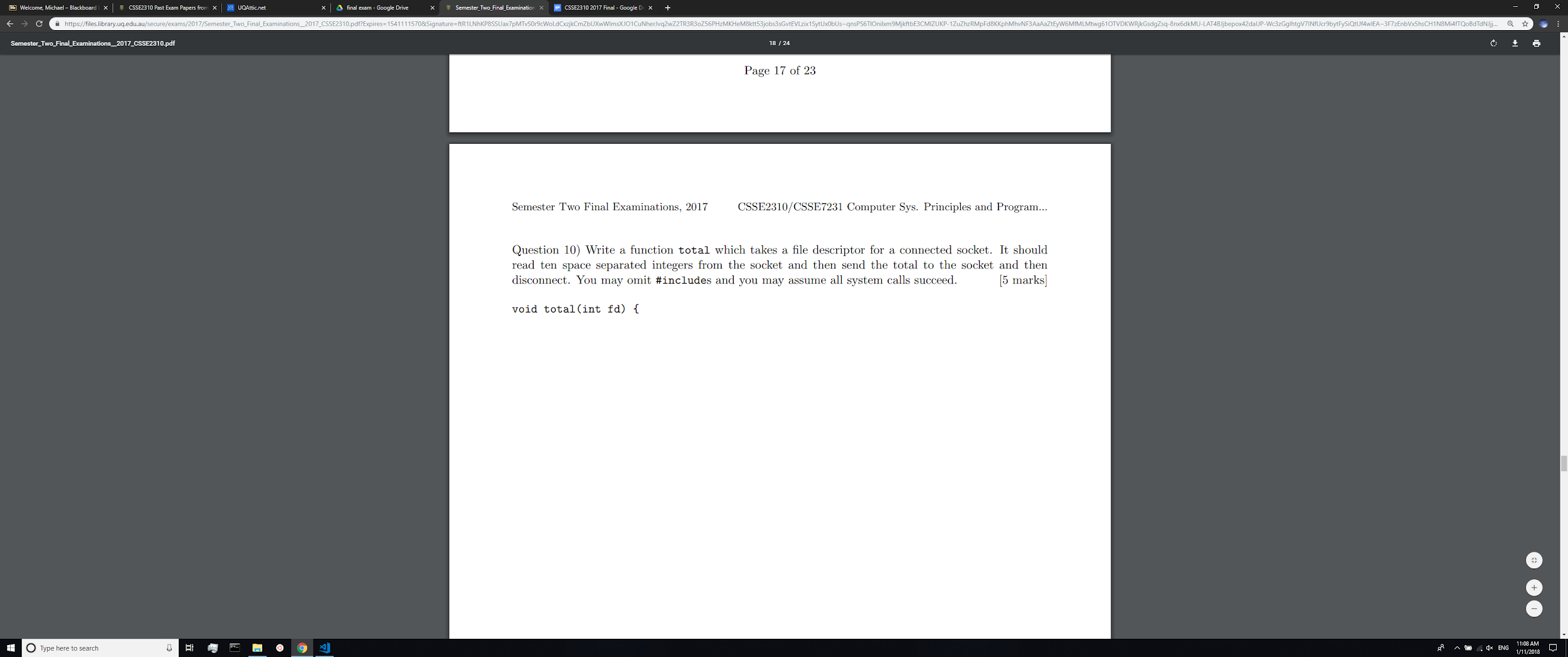
}

FILE\* f = fdopen(fdIn[READ], “r”);

printf(“%s\n”, get\_line(f));

return 0;

}



void total(int fd) {

int numLength = 10;

int \*intList = (int \*) calloc(numLength, sizeof(int));

// We need both read and write permissions

FILE \*getFile = fdopen(fd, "w+");

fscanf(getFile, "%d %d %d %d %d %d %d %d %d %d",

intList + 0, intList + 1, intList + 2, intList + 3, intList + 4,

intList + 5, intList + 6, intList + 7, intList + 8, intList + 9);

int index = 0, sum = 0;

for (index = 0; index < numLength; index++) {

sum += intList[index];

}

// Send back the total (don't forget to flush!)

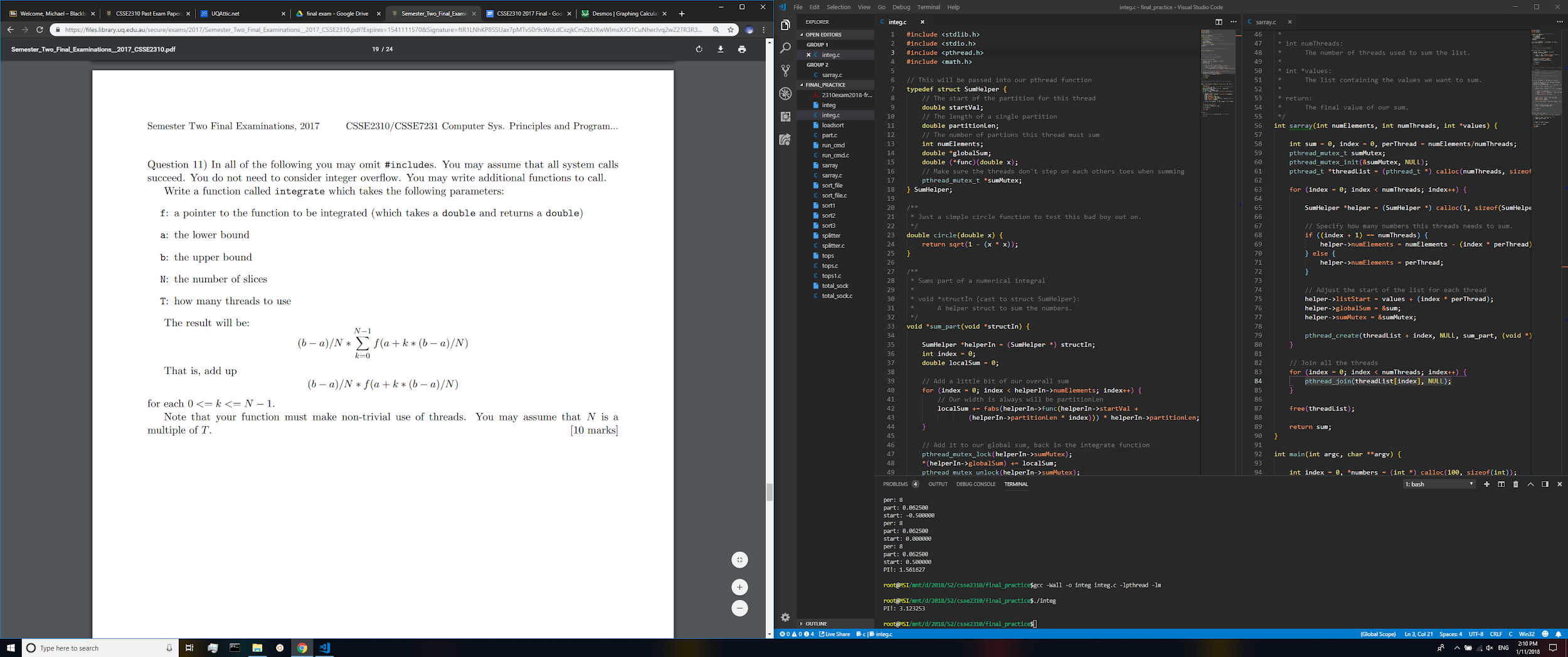
fprintf(getFile, "%d", sum);

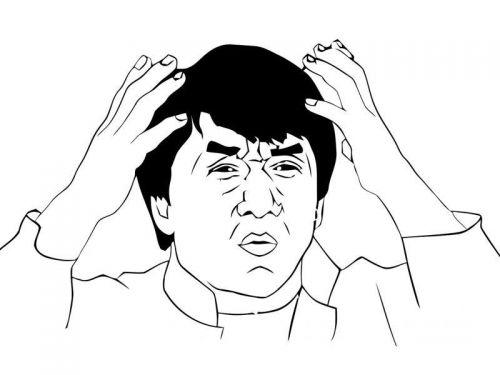
fflush(getFile);

fclose(getFile);

return;

}



.

Again, dunno how the hell you’re suppose to do this in 20mins or whatever. [+1] If you have a more elegant solution pls share it.

#include <stdlib.h>

#include <stdio.h>

#include <pthread.h>

#include <math.h>

// This will be passed into our pthread function

typedef struct SumHelper {

// The start of the partition for this thread

double startVal;

// The length of a single partition

double partitionLen;

// The number of partitions this thread must sum

int numElements;

double \*globalSum;

double (\*func)(double x);

// Make sure the threads don't step on each others toes when summing

pthread\_mutex\_t \*sumMutex;

} SumHelper;

/\*\*

\* Just a simple circle function to test this bad boy out on.

\*/

double circle(double x) {

return sqrt(1 - (x \* x));

}

/\*\*

\* Sums part of a numerical integral

\*

\* void \*structIn (cast to struct SumHelper):

\* A helper struct to sum the numbers.

\*/

void \*sum\_part(void \*structIn) {

SumHelper \*helperIn = (SumHelper \*) structIn;

int index = 0;

double localSum = 0;

// Add a little bit of our overall sum

for (index = 0; index < helperIn->numElements; index++) {

// Our width is always will be partitionLen

localSum += fabs(helperIn->func(helperIn->startVal +

(helperIn->partitionLen \* index))) \* helperIn->partitionLen;

}

// Add it to our global sum, back in the integrate function

pthread\_mutex\_lock(helperIn->sumMutex);

\*(helperIn->globalSum) += localSum;

pthread\_mutex\_unlock(helperIn->sumMutex);

free(helperIn);

return NULL;

}

/\*\*

\* Oh shiiiiiit, we have to math. Why is this only worth 10 marks!?

\* Why Joel?! WHY!!!?

\*/

double integrate(double (\*f)(double x), int a, int b, int N, int T) {

double sum = 0;

// This division should be a whole number (by the assumption given)

int k = 0, perThread = N/T;

double partitionLen = ((double) (b - a)) / ((double) N);

pthread\_mutex\_t sumMutex;

pthread\_mutex\_init(&sumMutex, NULL);

pthread\_t \*threadList = (pthread\_t \*) calloc(T, sizeof(pthread\_t));

for (k = 0; k < T; k++) {

SumHelper \*helper = (SumHelper \*) calloc(1, sizeof(SumHelper));

helper->func = f;

helper->globalSum = &sum;

helper->numElements = perThread;

helper->partitionLen = partitionLen;

helper->startVal = (((double) (b - a)) / ((double) T) \* k) + a;

helper->sumMutex = &sumMutex;

pthread\_create(threadList + k, NULL, sum\_part, (void \*) helper);

}

// Join all of our threads

for (k = 0; k < T; k++) {

pthread\_join(threadList[k], NULL);

}

free(threadList);

return sum;

}

int main() {

printf("PI!: %f\n", 2 \* integrate(circle, -1, 1, 32, 4));

return 0;

}