# **If you want to edit join UQAttic group.**

**Don’t email me requesting access, read the instructions below.**

# **CSSE2310: 2019 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.**

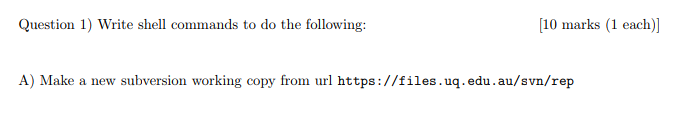
### Type answers in blue beneath each question.

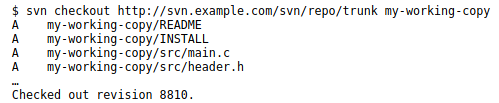
### If you're unsure of your answer, highlight your answer text then hit Ctrl+Alt+M to create a comment beside the text. Once you're satisfied with the answer, click the "Resolve" button on the comment.

### 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.

svn checkout <https://files.uq.edu.au/svn/rep> new-copy (what does new-copy do?) creates a new working copy. I was tempted to write svn checkout …., but since it says ‘a new working copy’, this will checkout the contents of the SVN under a new directory named ‘new-copy’:



FROM: <http://svnbook.red-bean.com/en/1.7/svn.tour.initial.html>



gcc -g start.c



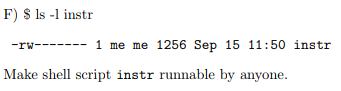
gcc -lm a.c b.c



ls \*.c



rep “rose” data > matches



chmod ugo+rx instr



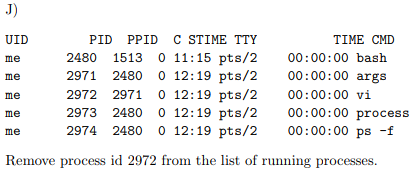
Cut -d ‘:’ -f 2 x.cols



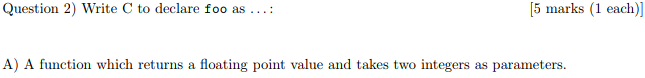
Grep “chocolate” data | grep “icecream”



Grep “muffin” data | sort | head -3 (Can also use grep -m3 “muffin” data)



Kill 2972



float (\*foo)(int, int);  
float foo(int, int); Just a function, not function pointer [+3]



bool foo;



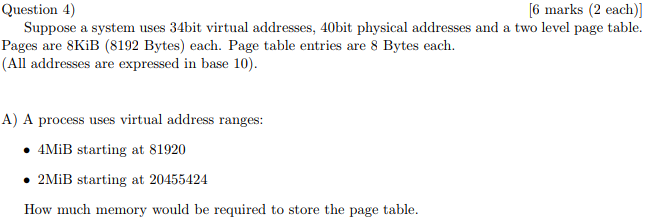
short\* foo;



Volatile unsigned int foo;



Void (\*foo)(void);



Pages used by 1MiB from 81920:

First page: Last page:

Pages used by 2MiB from 20455424:

First page: Last page:

Each ‘page of table’ holds (8\*1024)/8B PTE’s = 1024 PTE’s

200 … 522 are in the first page, + 1 page

2497 … 2753 are in the third page, + 1 page

2 pages to of table to reference the 4MiB and 2MiB, +1 for the first level page table

3 pages required for the page table, memory required = 8KiB \* 3 = 24 KiB

~~??????????????????~~

Joel’s answer:

We know that each page of table can store 1024 entries (8KiB/8B).

4MiB = (2^22) and takes (2^22/2^13) = 2^9 pages = 512 pages

2MiB = 2^8 pages = 256 pages

So where in the address space do those end up?

81920 is at the beginning of page 10.

Page 0 of table will store entries for page numbers from 0 to 1023 so all the entries from the first block belong on table page 0.

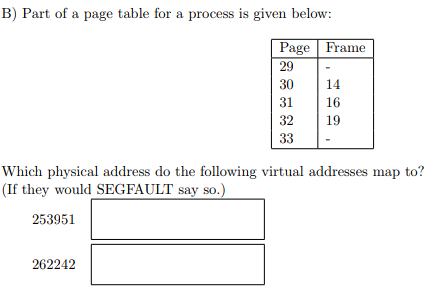
20455424 is on page 2497

Page 2 of table will store entries for page numbers from 2048 to 3072.

2497+256 = 2753l.

Page 2 will hold all the entries for the second block.

So that's 3 pages in total or 24KiB



**253951**: page # = floor(253951/(8\*1024)) = 30…. 30 maps to frame 14

Offset = VA mod page\_sz = 253951 mod 8\*1024 = 8191

Physical address = frame # \* page\_sz + offset

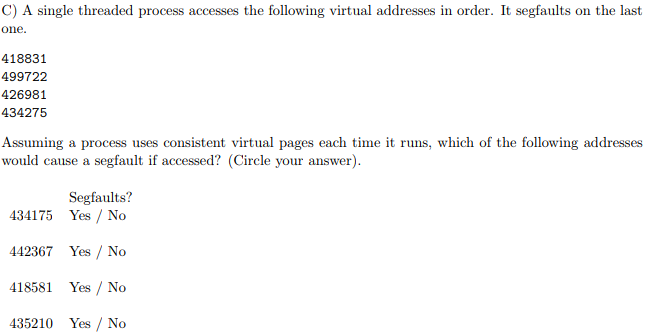
Physical address = 14 \* 8\*1024 + 8191 = **122879** [+2]

**262242**: page # = floor(262242/(8\*1024)) = 32 ….32 maps to frame 19

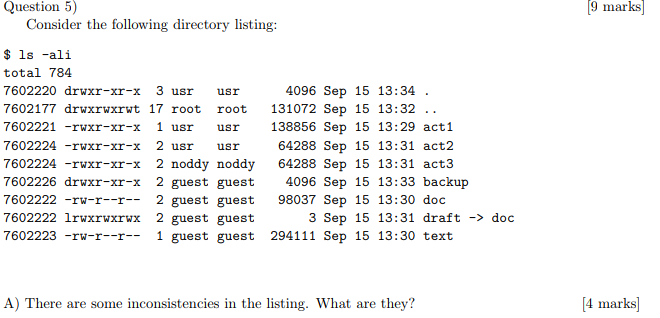
Offset = VA mod page\_sz = 262242 mod 8\*1024 = 98

Physical address = frame # \* page\_sz + offset

Physical address = 19 \* 8\*1024 + 98 = **155746** [+2]



No, Yes, No, Yes [+1]

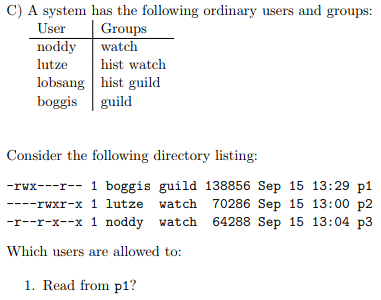


act2 and act3 are hardlinked but have different user and group  
draft is a symlink to doc but they share inode number

Also t is not a valid permission for parent “..” directory



0, # subdirectories = links - 2 shouldn’t be 15? “..” = 17 - 2 = 15 [+1] Actually, wouldn’t it be 14? Since it specifies ‘other’ subdirectories (i.e. not including the directory we’re looking at) [+2][+1] Good point.



Boggis, lutze, lobsang, noddy (?)  
I think this is right if user permissions override group permissions

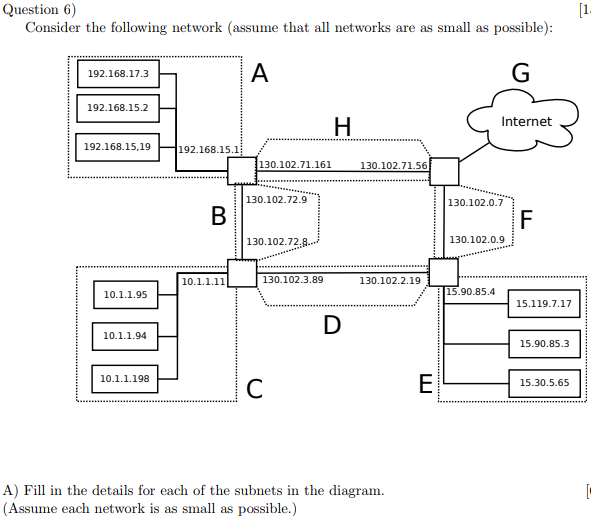
Boggis, lutze, noddy because group permissions also override everyone else's permissions. So Assuming ‘hist’ is a seperate group to ‘guild’ and not ‘hist guild’ lobsang is in group ‘guild’ so he gets group permissions. [+2]



Noddy (?) [+2]



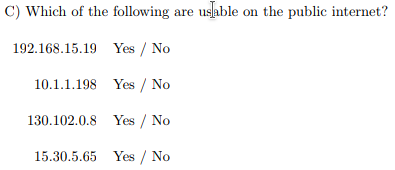
Lutze, lobsang, boggis (?) [+2]



| Network | Netmask | Broadcast | CIDR |
| --- | --- | --- | --- |
| A | 255.255.224.0 | 192.168.31.255 | 192.168.15.1/19 ?  192.168.0.0/19 [+5] |
| B | 255.255.255.240  255.255.255.254[+1]  255.255.255.252 | 130.102.72.15  130.102.72.9  130.102.72.11 | 130.102.72.0/28  130.102.72.8/31[+1]  130.102.72.8/30 |
| C | 255.255.255.0 | 10.1.1.255[+1]  10.1.0.255 | 10.1.1.0/24[+1]  10.1.0.0/24 |
| D | 255.255.254.0 | 130.102.3.255[+1]  130.102.1.255 | 130.102.2.0/23[+1]  130.102.0.0/23 |



| A | 8186 [+2] |
| --- | --- |
| E | 8388602 [+2] |
| F | 12 [+2] |



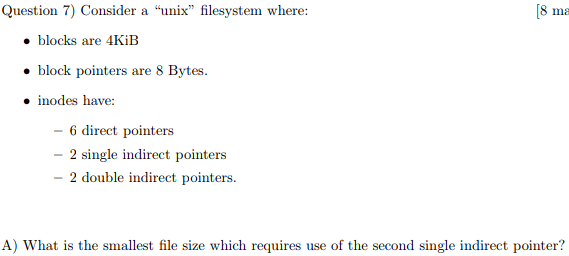
No, No, Yes, Yes [+1] (See non-routable IPs)



I’m not sure but I think:  
Host bits available: 32 - 19 = 13  
Host machines: 2^13 = 8192  
Network containing 128 + 2 (network & broadcast) requires 8 bits of host addresses  
floor(8192 / (2^8)) = 32 networks [+1]



| svn | Layer 5 (application) |
| --- | --- |
| wifi | Layer 2 (Link) ?? [+1] Physical? (it is EM waves) |
| IPv4 | Layer 3 (Network) |
| UDP | Layer 4 (transport) |
| HTTP | Layer 5 (application) |
| MAC | Layer 2 (link) |



24576 Bytes ? (24KiB) [+1]

----------------------------------------------------------

Pointers\_per\_block = (4\*1024)/8 = 512

First 6 direct pointers (6 \* 4KiB = 24KiB). First single indirect = pointers\_per\_block \* 4KiB = 512 \* 4KiB

So, 24KiB + (512 \* 4KiB) = 2072KiB which is the largest value that would not use a second single indirect pointer.

So the first to use the second single indirect pointer would be 2072KiB + 1B = 2121729B

(thoughts ??????)

Don’t think you need to add the extra byte because index starts at 0, so block 518 will be the start of the use of second indirect. 6 direct blocks + 512 for first single indirect = 518 \* 4KiB = 2072KiB [+2]



Other file properties except name?  
Size? Location? Permissions? [+1]

Known as metadata



2101272KiB [+3]



Could be unused fragments inside the direct and single indirect portions, meaning more of the double indirect is used than strictly necessary ?

Question 8)

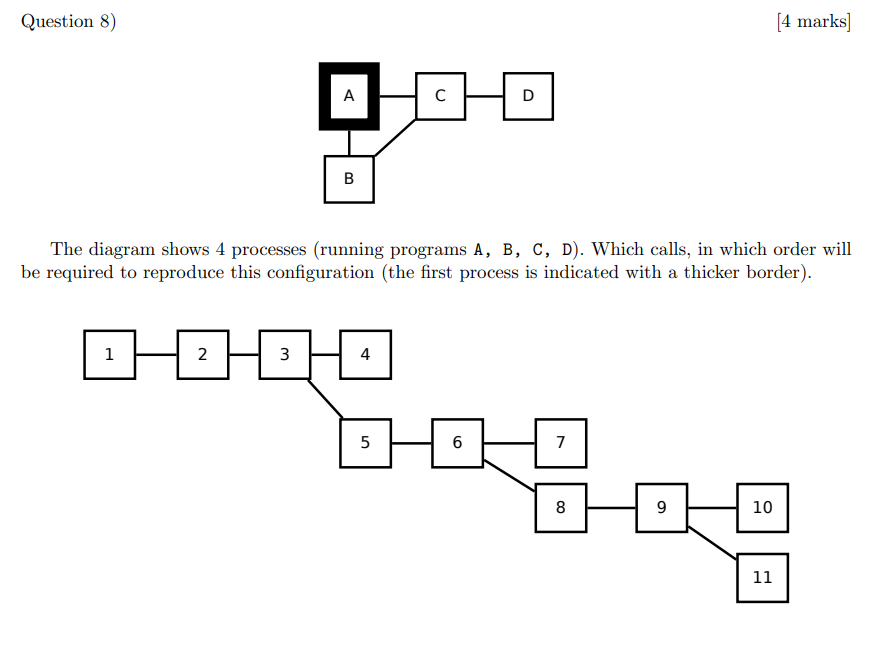
There are four process A, B, C, D. The processes are connected by pipes as follows:

* A to B
* A to C
* B to C
* C to D

The diagram shows 4 processes (running programs A, B, C, D). Which calls, in which order will be required to reproduce the configuration above The first process is the one which ends up running A).

There are eleven numbered boxes in four rows. Each row represents a process and the boxes on that row are operations performed by that process (in order).

* First row: 1, 2, 3, 4
* Second row: 5, 6, 7 (box 3 on the previous row connects to 5).
* Third row: 8, 9, 10 (box 6 on the previous row connects to 8).
* Fourth row: 11 (box 9 on the previous row connects to 11).



Fill in the correct operations in the boxes below: fork, execA, execB, execC, execD, fctl, pipeAB, pipeBC, pipeAC, pipeCD, pipeAC, sigaction

| 1. pipeAB | 5. pipeBC | 9. fork |
| --- | --- | --- |
| 2. pipAC | 6. fork | 10. execC |
| 3. fork | 7. execB | 11. execD |
| 4. execA | 8. pipeCD |  |

[+1]

Question 9)

Where you are asked to write functions, you can also write helper functions or types if you need them. You may assume that system calls succeed.

In later parts of this question, you may make use of functions described in earlier parts (and you may assume they are implemented correctly).

We will be bucket sorting some integers in the range [0, …, *N*]. Make an array of ints of size *N* + 1. Each time a number *x* is seen(), increment array[x]. To output the sorted sequence, loop through the array looking for indices which have a non-zero total.

Eg:

Bucket b;

init(&b, 5);

seen(&b, 0);

seen(&b, 4);

seen(&b, 0);

seen(&b, 3);

Would result in a data structure like this:

A Bucket struct containing an int\* member and any other members you think are necessary. The int\* member points to an array of five values storing: 2, 0, 0, 1, 1, 0.

Printing would output:

0,0,3,4

A) Write a typedefed struct type declaration for Bucket. You should look at parts **B–E** to see what you might need.

OK, this doesn’t work but here are my initial thoughts:

typedef struct {

int\* bucket;

int upperLimit;

sem\_t\* lock;

int fd;

} Bucket;

B) Implement void init(Bucket\* b, int upperLimit)

void init(Bucket\* b, int upperLimit) {  
 sem\_init(b->lock, 0, 1);  
 b = malloc(sizeof(Bucket));  
 b->upperLimit = upperLimit;  
 b->bucket = calloc(upperLimit, sizeof(int));  
}

Don’t need to malloc the Bucket, it was passed in as a pointer. It is the caller’s responsibility to pass it in.

C) Implement void cleanup(Bucket\* b)

Not sure if this is what they’re looking for or not [+1]

void cleanup(Bucket\* b) {

free(b->bucket);

free(b);

}

No need to free b. Mabe it was declared on the stack elsewhere and freeing is invalid. It is the caller’s responsibility to dea with the memory for b (but you must free its members in this function).

D) Implement void print\_sorted(Bucket\* b, FILE\* f) : output comma separated values E) Implement void seen(Bucket\* b, int value) : silently reject any values which are out of range. Your implementation should be thread-safe.

Yeah, I know, this is where we start to segfault and fall apart

void print\_sorted(Bucket\* b, FILE\* f) {

char output[200];

for (int i = 0; i < b->upperLimit; ++i) {

if (b->bucket[i] > 0) {

for (int j = 0; j < b->bucket[i]; ++i) {

sprintf(output, "%c,", i);

}

}

}

output[strlen(output) - 1] = '\0';

fprintf(f, "%s\n", output);

}

F) Implement void do\_thing(int fd, Bucket\* b) where fd is a connected socket file descriptor. Read a line of text (no more than 200 chars) from that socket. The line will contain space separated integers. Add each integer to the bucket, output the number of values read to the socket and close it.

void do\_thing(int fd, Bucket\* b) {

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

FILE\* from = fdopen(fd, "r");

char input[200];

int count = 0;

fgets(input, 200, from);

char\* currentNum = input;

char\* nextNum;

int number = (int)strtol(input, &nextNum, 10);

while (currentNum != nextNum) {

seen(b, number);

count++;

number = (int)strtol(input, &nextNum, 10);

}

fprintf(to, "%d\n", count);

fflush(to);

fclose(to);

fclose(from);

}

G) Implement void startup(int limit, int conc, int\* conv) where

* limit is the largest number expected for a Bucket.
* conv is an array of connected socket file descriptors.
* conc is the number of descriptors in the array.

The function should create a Bucket, run an instance of do\_thing() for each descriptor in separate threads. Once all of the threads have finished, print out the values in the bucket to stdout.

If anyone can fix this or give a better solution we are forever in your debt.

How we’re meant to get this to compile and run in under 3 hours I have no idea. Haha, half the time is used to understand the question

void\* thread\_wrapper(void\* bucket) {

Bucket\* b = (Bucket\*)bucket;

init(b, b->upperLimit);

do\_thing(b->fd, b);

return NULL;

}

void startup(int limit, int conc, int\* conv) {

pthread\_t\* threads = malloc(sizeof(pthread\_t) \* conc);

Bucket\* b = malloc(sizeof(Bucket));

for (int t = 0; t < conc; ++t) {

b->upperLimit = limit;

b->fd = conv[t];

pthread\_create((pthread\_t\*)threads[t], 0,  
 thread\_wrapper, (void\*)b);

}

for (int t = 0; t < conc; ++t) {

pthread\_join(threads[t], 0);

}

print\_sorted(b, stdout);

}

Question 10)

A)

Write a C program to execute the following shell command without using a shell. You are not permitted to call system(). You may omit #includes. You may assume that all system calls succeed.

./translate mode 7 < source > output

Is translate a shell command? Is mode?

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

execvp(argv[1], argv);

}

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

char\* procName = "./translate";

char\* procArg1 = "mode";

char\* procArg2 = "7";

char\* inFile = "source";

char\* outFile = "output";

int inFD = open(inFile, O\_RDONLY);

dup2(inFD, STDIN\_FILENO);

int outFD = creat(outFile, 0777);

dup2(outFD, STDOUT\_FILENO);

execlp(procName, procName, procArg1, procArg2, NULL);

}

B) Running svn status produces the following output:

? Z.h

M X.c

D P.c

A B.c

What affect would svn commit have on the repository?

? - No effect, not under version control

M - will attempt to merge with repo version of X.c?

D - will remove P.c from repo

A - will add B.c to repo

I feel like after the ridiculous assignment workload that it is only fair we get an exam so large it is not possible to finish it… [+1\*10^9999999999] Agreed.. I thought this exactly when i saw the integrate question (Q11) from the 2017 exam

**Exam prep ideas**

Any ideas that you have to prepare for the programming part of the exam (e.g. to save time)

* A whole collection of utility functions like str\_is\_numeric, sort, count\_occurrences (of characters in a string), trim\_newline, char\_to\_int, int\_to\_char, digits\_in\_int, etc
* C files ready to go with int main() and the usual includes ready

I imagine every second will count on the exam [+1][+1]

Good luck to all. Anyone gets a 7, I will worship them as a God.

typedef enum Grades {

FAIL = 0,

MORTAL = 4,

KING = 5,

IMMORTAL = 6,

GOD = 7,

} Grades;

void get\_grades(int grade)

{

if (grade < MORTAL) {

printf("GPA %d: See you next year!\n", grade);

} else if (grade < IMMORTAL) {

printf("GPA %d: The mission, the nightmares.. they're finally.. over.\n", grade);

} else if (grade == IMMORTAL) {

printf("GPA %d: You are an immortal\n", grade);

} else { //grade == GOD

printf("GPA %d: People must worship you as a GOD\n", grade);

}

fflush(stdout);

}

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

{

if (argc != 2)

{

printf("Usage: ./getGrades grade\n");

return 1;

}

int grade;

sscanf((const char\*)argv[1], "%d", &grade);

get\_grades(grade);

return 0;

}

int calculate\_final\_grade(double ass1, double ass2, double ass3, double ass4, double final) {

return 4;

}