# CSSE2310: 2013 ‘Practice’ exam answers

**UQAttic**

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

**Other exam papers**

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 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 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.

**Q1) Write shell commands to do the following:**

**A) Delete all files with names beginning with A and ending in .c**
rm A*.c

**B) Show all lines in the file stuff which start with W**
grep ^W stuff

**C) A file nums consists of 4 space separated columns. Output columns 1, 3, 4 sorted by the last column**
cut -d ‘ ‘ -f1,3,4 nums | sort -k 3
OR
sort -k 4 nums | cut -d' ‘ -f1,3,4
OR
cat nums | cut -d' ‘ -f1,3,4 | sort -k 3

**D) Create a file c.c which is a copy of b.c**
cp b.c c.c

**E) For files f1, f2, f3, show all lines from any of them which contains all the words “song”, “river” and “terrible”**
cat f1 f2 f3 | grep song | grep river | grep terrible

**Q2) Write C to declare foo as:**

**A) An array of 12 integers**
int foo[12];

**B) A pointer to a positive integer**
unsigned int* foo;

**C) Another name for a small integer**
typedef short foo;

**D) A struct containing an integer called i and a string called s**

```
struct {
    int i;
    char* s;
} foo;
```

**NOTE: As question says a ‘struct’, it should be an instance of a struct, so foo goes at the end.**

**E) A pointer to a function which takes two floating point values and returns a string**
char* (*foo)(float, float);

**Q3)**

| | |
|---|---|
| **A.** | 2 |
| **B.** | 4 7 |
| **C.** | 3 6 |
| **D.** | 17 |
| **E.** | 9 |
| **F.** | 4 6 |
| **G.** | 8 2 |
| **H.** | 28 |
| **I.** | 8 9 |
| **J.** | 4 3 3 5 |
| **K.** | w ld |

**Q4) A system has 32bit virtual address, 4KB pages and page table entries are 4Bytes. It uses a two level page table.**

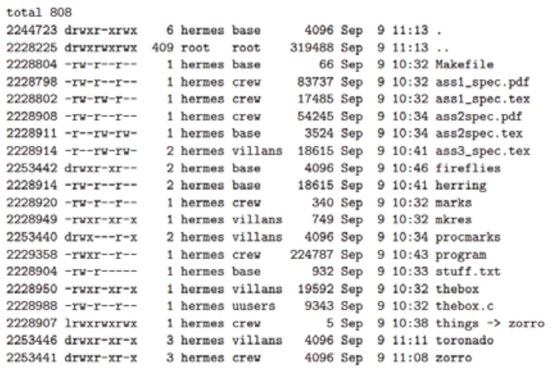
**A. Which pages do the following (decimal) addresses belong to?**
**11111, 22222, 9001, 404040**
2, 5, 2, 98

Eg.
4096 bytes in a page
Page = virtual address / page size
Page = 11111/4096 = 2.7
This means virtual address 11111 is on Page 2 (which is the third page, since numbering starts at 0).

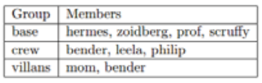
**B. what causes page faults?**
When an object/process/program is on disk and not in memory.

**C. What causes segmentation faults?**
-trying to access an invalid memory page
-writing to a read only page

**Q5) Consider the following directory listing:**



```
total 808
2244723 drwxr-xrwx    6 hermes base      4096 Sep  9 11:13 .
2228225 drwxrwxrwx  409 root   root    319488 Sep  9 11:13 ..
2228804 -rw-r--r--    1 hermes base        66 Sep  9 10:32 Makefile
2228798 -rw-r--r--    1 hermes crew     83737 Sep  9 10:32 ass1_spec.pdf
2228802 -rw-rw-r--    1 hermes crew     17485 Sep  9 10:32 ass1_spec.tex
2228908 -rw-r--r--    1 hermes crew     54245 Sep  9 10:34 ass2spec.pdf
2228911 -r--rw-rw-    1 hermes base      3524 Sep  9 10:34 ass2spec.tex
2228914 -r--rw-rw-    2 hermes villans  18615 Sep  9 10:41 ass3_spec.tex
2253442 drwxr-xr--    2 hermes base      4096 Sep  9 10:46 fireflies
2228914 -rw-r--r--    2 hermes base     18615 Sep  9 10:41 herring
2228920 -rw-r--r--    1 hermes crew       340 Sep  9 10:32 marks
2228949 -rwxr-xr-x    1 hermes villans    749 Sep  9 10:32 mkres
2253440 drwx---r-x    2 hermes villans   4096 Sep  9 10:34 procmarks
2229358 -rwxr--r--    1 hermes crew    224787 Sep  9 10:43 program
2228904 -rw-r-----    1 hermes base       932 Sep  9 10:33 stuff.txt
2228950 -rwxr-xr-x    1 hermes villans  19592 Sep  9 10:32 thebox
2228988 -rw-r--r--    1 hermes uusers    9343 Sep  9 10:32 thebox.c
2228907 lrwxrwxrwx    1 hermes crew         5 Sep  9 10:38 things -> zorro
2253446 drwxr-xr-x    3 hermes villans   4096 Sep  9 11:11 toronado
2253441 drwxr-xr-x    3 hermes crew      4096 Sep  9 11:08 zorro
```



| Group | Members |
|---|---|
| base | hermes, zoidberg, prof, scruffy |
| crew | bender, leela, philip |
| villans | mom, bender |

**A) what can `zoidberg` do to the following:**
**`>stuff.txt`**
read
**`>procmarks`**
read, execute

**B) Which users can modify all of the `.tex` files (without changing permissions)?**
the crew (bender, leela, philip)

**C)What command(s) could `mom` type to execute `program`?**
Copy program to home directory and then run it (mom can read program)
cp program ~/program
~/program

**D) What would change in the directory listing after hermes executed `rmdir toronado` (and why?)**
No change, since toronado is not an empty directory. rmdir does nothing.

**E) What command was used to create `things`?**
ln -s zorro things
**F) Given the following commands and their output:**

```
prompt> ls -l zorro/transport
lrwxrwxrwx 1 hermes base 21 Sep 9 11:20 zorro/transport -> ../fireflies/serenity

prompt> ls -l fireflies/serenity
-rwx--x--x 1 hermes base 1072966 Sep 9 10:46 fireflies/serenity
```

**Can bender run `./things/transport` ? (Why?)**
No because the fireflies directory doesn't give bender permissions to execute it (in other words access stuff in the folder)

**Q6) Network stuff**

**A)**

| | **Network Address** | **Subnet Mask** | **Broadcast Address** |
|---|---|---|---|
| **X1** | 10.10.96.0 | 255.255.254.0 | 10.10.97.255 |
| **X3** | 10.10.144.0 | 255.255.248.0 | 10.10.151.255 |
| **X4** | 10.11.64.32 ~~10.11.64.40.0~~ | 255.255.255.240 ~~Shouldn't this be 255.255.252 since only the last three bits differ?~~ | 10.11.64.47 ~~10.11.64.40.3~~ |

**B) What does the `bind()` function perform?**
bind(): After a socket has been created, it attaches a local address to it.

**C) X2 performs NAT for this network. What is NAT and why is it necessary?**
Network address translation: Needed when the number of IPs assigned to you is less than the total number of computers trying to access the internet. It assigns an entity/organisation a single IP

Wouldn't a safer answer be that some IPs are non -routable (http://jeff.nieusma.com/docs/network/non-routable-ip.html). If any machines with these IPs want to communicate with the rest of the internet, their address must be converted by the NAT.

**D) to which layers do the following belong:**

| **Term** | **Layer** |
|---|---|
| **MAC address** | link |
| **socket** | application |
| **IP Address** | network |
| **port** | transport |
| **UDP** | transport |
| **URL** | application |

**E) What is the purpose of a `network gateway`?**
Router interface connected to the local network. It's purpose is to send packets out of and receive to the local network.

**Q7) Consider a "unix" filesystem where:**
**>i-nodes have 10-direct pointers, 1 indirect pointer and 1 double indirect pointer.**
**>Blocks are 8KB**
**>Block pointers are 4Bytes**
**>blocks are numbered from 0.**

**A)Why is fragmentation a problem for linked filesystems but not for indexed filesystems?**
Indexed filesystems are sequential so adjacent blocks store data "in order". Linked filesystems hold pointers to data, so the data can be spread (physically) across the disk.

**B) How many blocks (in total) must be accessed to read the following blocks from a file: 9, 2053, 2057**
1 for 9, 2 for 2053, and 1 for 2057 (the pointer is stored from the 2053 read)
total of 4

**C)What is the maximum possible file size for this file system?**
total size = (10*8192) + (1*2048*8192) + (1*2048^2 * 8192) = 33,570,896 kB

**D) If an additional 2 double indirect pointers were added to the i-nodes on this system, what would be the increase in maximum file size?**
Increase is 2*2048^2 * 8192 = 67,108,864 kB

**Question 8: multiprocess/fork stuff**

**A)**
**Lower diagram shows a process being reaped by a process who isn't its parent (or init). This is impossible.**

**B)**
**wait(), with WNOHANG set.**

**C)**
**kill(pid_t pid, int sig)**

**D)**
**pthread_exit(void* value_ptr)**

**pthread_join(pthread_t thread, void** value_ptr)**

**E)**
**pthread_mutex_lock**
**pthread_mutex_unlock**

**F)**
~~**The down arrow indicates reaping of the child. So the top left, top right and bottom right are possible but not bottom left, for the same reason as (A).**~~

**JEA: Down arrow indicates a thread being joined. All of them are possible. Threads do not follow the same rules as processes, because they are different.**

**Isn't it that a thread cannot join a thread who isn't its child? So the bottom left is not correct**

**Question 9) code**
**Note: As the question states, the #includes aren't necessary in your answer. I put them in so I could run the code. Copy/Paste all 5 functions and play around with it for fun.**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

bool matchingLines(const char* string, const char* filename) {

       FILE *file;
       char line[81];

       if ((file = fopen(filename, "r")) == NULL) {
              return false;
       }

       while(fgets(line, 81, file) != NULL) {
              if (strstr(line, string) != NULL) {
                     printf("%s", line);
              }
       }

       return true;
}
```

**B)**
```
bool grepSearch(const char* string, const char* filename) {

       if (fork()) {       /* Parent */

              int status;
              wait(&status);

              if (WIFEXITED(status)) {
                     if (WEXITSTATUS(status) == 2) {
                            return false;
                     } else {
                            return true;
                     }
              }

       } else {      /* Child */

              execlp("grep", "grep", string, filename, NULL);
              exit(0);

       }

       return true;
}
```

**C)**
```
bool matchingLinesMany(const char* string, const char** filenames, int numfiles) {

       int i = 0;
       int status;
       bool result = true;

       for(i = 0; i < numfiles; i++) {
              if (fork() == 0) {/* Child */
                     if (matchingLines(string, filenames[i])) {
                            exit(0);
                     } else {
                            exit(2);
                     }
              }
       }
       for(i = 0; i < numfiles; i++) {
              wait(&status);
              if (WIFEXITED(status)) {
                     if (WEXITSTATUS(status) == 2) {
                            result = false;
                     }
              }
       }

       return result;
}


With threads (see next page):
```

```c
struct SearchData {
    char *string;
    char *filename;
};

void *match(void* arg) {
    struct SearchData* data = (struct SearchData*) arg;
    bool res = matchingLines(data->string, data->filename);
    pthread_exit((void*) res);
}

bool matchingLinesMany(const char* string, const char** filenames, int numfiles) {
    pthread_t* threads = malloc(numfiles * sizeof(pthread_t));
    struct SearchData* searches = malloc(numfiles * sizeof(SearchData));
    bool result = true;

    for (int i = 0; i < numfiles; ++i) {
        searches[i] = malloc(sizeof(struct SearchData));
        data->string = string;
        data->filename = filenames[i];
        pthread_create(&threads[i], NULL, match, (void*) data);
    }
    for (int i = 0; i < numfiles; ++i) {
        bool res;
        pthread_join(threads[i], (void**) &res);
        if (!res) {
            result = false;
        }
        free(threads[i]);
        free(searches[i]);
    }
    return result;
}
```

**D)**

```c
bool grepSearchMany(const char* string, const char** filenames, int numfiles) {

        int i = 0;
        int status;
        bool result = true;

        for(i = 0; i < numfiles; i++) {
                if (fork() == 0) { /* Child */
                        execlp("grep", "grep", string, filenames[i], NULL);
                        //should this line not be
                        result = grepsearch(string, filenames[i]);
                        exit(result);
                        exit(0);
                }
        }
        for(i = 0; i < numfiles; i++) {
                wait(&status);
                if (WIFEXITED(status)) {
                        if (WEXITSTATUS(status) == 2) {
                                result = false;
                        }
                }
        }
        return result;
}
```

**E)**

```c
int main(int argc, const char **argv) {

        if (argc < 4 || ((strcmp(argv[1], "M") != 0) &&
                    (strcmp(argv[1], "G") != 0))) {
                fprintf(stderr, "Bad params.\n");
                exit(1);
        }

        int i = 0;
        int (*func)(char*, char*, int); // function pointer avoids code duplication

        if (argv[1][0] == 'M') {
                func = matchingLinesMany;
        } else {
                func = grepSearchMany;
        }
        if (!func(argv[2], argv+3, argc - 3)) {
                fprintf(stderr, "Bad file.\n");
                exit(2);
        }
        return 0;
}
```