Instructions:

In all questions circle the most correct answer.

In questions where knowledge of the computer's hardware is required, assume the following:

* A 32bit address space.
* 4KB pages with 4 byte page table entries.
* Uses two level page tables.
* L1 and L2 cache are considered part of memory.

1) Which lines in the following would cause warnings/errors under gcc -c -ansi -pedantic -Wall?

int f(char c) { [1]

if ((c<'a') || (c>'z')){ [2]

return; [3]

} [4]

char i; [5]

for (i=c;i<='z';++i){ [6]

printf("%c"); // print remaining [7]

} [8]

return 1; [9]

} [10]

(a) 1

(b) 2

(c) 3

(d) 5

(e) 6

(f ) 7

(g) 9

(h) 3, 5

(i) 6, 7

(j) 3, 5, 7

(k) 2, 3, 5

(l) 2, 5, 7

(m) 2, 6, 9

(n) 3, 5, 9

(o) 1, 2, 3, 5

(p) none of the above

Discussion:

There is an error on line 3 - return with no value. This is legal in C but only for void functions.

This function is supposed to return an int.

Line 5 is a declaration but we have had code already. This is a mixed declarations and code error in ansi-C

Line 7 has a c++ comment on it - Legal in c99 but not in ansi C. Also never gives argument for %c in printf statement.

This gives an answer of (j)

2) Which of the following svn commands do not require a commit?

(a) add

(b) remove

(c) status

(d) update

(e) diff

(f ) add, diff

(g) update, remove

(h) status, diff, update

(i) add, remove, update, diff, status

(j) We can't be certain/none of the above.

Discussion:

add needs one as does remove.

This rules out a,b,f,i

This leaves, status, update, diff.

That is, c,d,e,h,j

The only reason not to pick (h) is if you think one of those requires a commit.

(None of them do)

Commit goes to the repository update comes from the repository.

status and diff work on your working copy.

3) Consider the following program:

#include <stdio.h>

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

int i;

for (i=0;i<3;++i) {

fprintf(stderr, "%d is ",i);

if (i%2) {

fprintf(stderr, "even\n");

} else {

fprintf(stderr, "odd\n");

}

}

return 0;

}

The program is compiled and run with the following command:

./a.out | grep -v even

Which lines will appear on the console?

(a)nothing will appear

(b) 0 is even

1 is odd

2 is even

(c) 0 is even

2 is even

(d) 1 is odd

(e) The shell command is invalid

(f) none of the above.

Discussion:

The answer is none of the above for two reasons.

First (the intended one):

all the output is sent to standard error.

I've been very careful in lectures to always say that shell pipes connect the \_standard out\_ of a process to the standard input of the next process.

The pipe has no effect on standard error.

In this situation, everything printed will be displayed on the console.

Secondly, consider what happens to i==0.

0%2 is 0 which is false. Thus, it would print to stderr:

0 is odd

1 is even

2 is odd

4) We wish to find the 7th line in a file called data which contains the word “fish”. Which of the following lines would do this?

(a) grep data fish | head -7 | tail -1

(b) grep fish data | tail -7 | head -1

(c) cat data | head -7 | tail -1 | grep fish

(d) cat data | grep fish | head -7 | tail -1

(e) grep fish | head -7 | tail -1

(f ) head -7 | cat data | tail -1 | grep fish

(g) none of the above.

Discussion:

The order for grep is grep \_thing\_to\_look\_for\_ \_files\_to\_search\_

The second param is optional, if it is omitted, then grep reads from standard in.

All this should be fairly clear from the man page for grep [grep was on the list of commands you needed to know].

a- wrong because it searches for data in a file called fish.

b- wrong because it does the tail first. so would find the 7th last not the 7th from the beginning.

c- wrong it finds the 7th line but does not check if it contains fish until afterwards. This will either display the 7th line if it happens to contain fish or nothing.

d- correct. [remember grep can read from standard in].

e- will read from standard in rather than from the file. Apart from that it's correct.

f- again reads from standard in.

Note: This question is asking for the 7th line in the file that contains the word fish. That is, with a file like:

1 fish

2 fish

3 fish

4

5 fish

6

7

8 fish

9 fish

10 fish

11 fish

The result would be “10 fish”

5) After executing the following code fragment what are the values of w, x, y, z.

int w=0;

int x=6;

int y=3;

int z=2;

if (x=y==z){

z=4;

} else {

z=9;

}

if (w);{

w=7;

}

(a) w=7; x=0; y=2; z=4;

(b) w=0; x=6; y=2; z=4;

(c) w=7; x=0; y=3; z=9;

(d) w=0; x=0; y=3; z=4;

(e) w=7; x=6; y=2; z=9;

(f ) It is not legal C.

(g) Segmentation fault.

(h) We can't be certain/none of the above.

Discussion:

Some people had trouble with the mixed = and ==

This is legal in the same way you can mix + and -

The precedence chart in lectures shows that == has higher precedence so it will be evaluated first.

so if (x=y==z) becomes (x=(y==z))

(x=(3==2))

(x=0)

(0)

So the if is false

z=9

(and x==0)

The semi-colon after the if is legal [it's just an empty statement] so w=7 in all cases.

This leaves (c) as the answer.

6) What are the values of i and j after executing the following code.

int i=4;

int j=2;

switch (i++-j) {

case 3:i++; break;

case 1:j++; break;

case 2:j+=2;

case 5:i+=2;

default:i+=5;

}

(a) i=5; j=2;

(b) i=6; j=2;

(c) i=7; j=4

(d) i=9; j=2;

(e) i=11; j=2;

(f) i=12; j=4;

(g) It is not legal C.

(h) Segmentation fault.

(i) We can't be certain/none of the above.

Discussion:

Again people were unhappy with the idea of mixing operators (++ vs - ).

This is legal.

You are also allowed to have cases without break - this was explained in lectures.

In such cases you just keep executing until you hit a break, or reach the end of the switch statement.

The value of the expression in the switch expression is:

4-2 == 2

(The ++ has the side effect of changing i to 5.)

We enter the switch at case 2 (j=2) and execute j+=2, now j = 4.

No break -> Continue executing down the line, next is case 5, i+=2 -> i=7.

Next line is default case, i+=5 -> i=12

i=12, j=4

This gives the answer of (f)

7) What type are var, foo, baz in the following?

long\* var, foo[7], baz;

(a) var - pointer to long, foo - array of 7 pointers to long, baz - pointer to long

(b) var - pointer to long, foo - array of 7 longs, baz - long

(c) var - long, foo - array of 7 pointers to long, baz - long

(d) var - pointer to long, foo - array of 7 longs, baz - pointer to long

(e) var - long, foo - array of 7 longs, baz - long

(f ) It is not legal C.

(g) We can't be certain/none of the above.

Discussion:

As explained in lectures, the \* only affects the variable directly on its right.

So splitting the declarations up we get:

long\* var;

long foo[7];

long baz;

This gives the answer (b)

8) Which of the following declares foo to be:

a function which returns a string and takes a pointer to a function which takes a char and returns a char.

(a) char\* (\*foo)(char (\*)(char))

(b) char \*foo(char(\*)(char))

(c) char (\*foo)( (\*)())

(d) char\* (\*foo)(char (\*)(char))

(e) It is not possible to write such a declaration.

(f ) We can't be certain/none of the above.

Discussion:

The question asks for a \_function\_ which .....

alternative b was:

char \*foo(char(\*)(char))

In lectures I said that the position of the spaces around the \* does not matter.

So char\* x is the same as char \*x.

So b is equivalent to

char\* foo(char(\*)(char))

The rest of the answers (a,c,d) were all \_function pointers\_ not functions.

9) What is the result of the following code?

int x=2, y=3, z=5;

int\* py=&x;

int\* px=&y;

\*px=x+y;

z=\*px;

x=\*px-\*py;

\*px=y-\*py;

(a) x=0, y=3, z=5

(b) x=2, y=3, z=5

(c) x=3, y=2, z=5

(d) x=2, y=3, z=2

(e) x=2, y=3, z=3

(f ) It is not legal C.

(g) Segmentation fault;

(h) We can't be certain/none of the above.

Discussion:

You needed to notice that px points to y and py points to x

\*px=x+y ==> y=x+y

So y=5

z=5 (which it was already)

x=y-x ==> x=y-2==3

So x=3, y=5, z=5

\*px=y-\*py ==> y=y-x ==> y=5-3==2

So x=3, y=2, z=5

The answer is (c)

10) What is the result of the following code?

int x=0, y=2, z=4;

int \*px, \*py, \*pz;

py=&z;

pz=&y;

x=x+4;

\*px+=4;

(a) x=0, y=2, z=4

(b) x=4, y=2, z=4

(c) x=8, y=2, z=4

(d) x=4, y=4, z=4

(e) It is not legal C.

(f ) Segmentation fault;

(g) We can't be certain/none of the above.

Discussion:

px is not initialised.

That doesn't necessarily mean it points to an invalid page.

It \_could\_ be anything. That is the problem is uninitialised values.

Correct answer (g) we can't be certain.

11) Consider the following function:

void f(int x) {

x--;

fprintf(stderr, "B");

if (fork()) {

fprintf(stderr, "C");

if (x>1) {

f(x);

}

} else {

fprintf(stderr, "A");

}

fprintf(stderr, "B");

}

When f(3) is called, how many B's will be output?

(a) 2

(b) 3

(c) 4

(d) 5

(e) 6

(f ) 7

(g) 8

(h) 9

(i) none of the above.

Discussion:

The difficult part here is that when f(2) is called it forks so two processes return to f(3).

The best way to deal with this is just to draw a picture.

Please see my diagram linked in the exam area.

The answer was (f) - 7

12) When is a zombie process not created?

(a) When a segmentation fault occurs.

(b) When exit() is called.

(c) When the main() function returns.

(d) When its parent was wait()ing for it.

(e) none of the above.

Discussion:

As I said in lectures - zombies are always created.

They might not be around for very long but they are always made.

(e) - none of the above.

* ---- The last three questions need the system information from the front of the exam paper ----
* A 32bit address space.
* 4KB pages with 4 byte page table entries.
* Uses two level page tables.
* L1 and L2 cache are considered part of memory.

13) The CPU access the following (base 10) virtual addresses in sequence.

45055, 45056, 8392710, 45090, 8392714, 45091

How many accesses to memory are required?

(a) 4

(b) 5

(c) 6

(d) 9

(e) 10

(f ) 11

(g) 12

(h) 15

(i) 16

(j) none of the above.

Discussion:

Step one is to work out which pages the addresses fall in.

pages are 4KB (4096)

So we (integer) divide each address by 4096.

45055 ==> 10

45056 ==> 11

8392710==> 2049

45090 ==> 11

8392714 => 2049

45091 ==> 11

First location: to access this location we will need to find its physical address.

This means referring to the page table. There are two levels to the page table so we need two memory accesses to find the first location and one to do the access itself.

BUT now that mapping is in the TLB (Translation lookaside buffer).

The next two addresses will each require 3 memory accesses (for the same reason).

So we have done three and are currently at a total of 9 accesses.

Now we have three more addresses but they are all in the TLB now so they don't need additional lookups.

Gives a total of 12. Answer=(g)

14) What is the largest amount of memory the page table for a single process can occupy?

(a) 3072KB

(b) 4096KB.

(c) 4100KB.

(d) 1048576KB.

(e) 4GB.

(f ) none of the above.

Discussion:

(I suspect) A lot of people made a simple error here.

We know that for a 32bit virtual address space we will need 2^32/2^12 == 2^20 pages.

We know that each page can hold 4KB/4B entries.

So we need 2^20/2^10 pages worth of 2nd level table.

Which is 1024 table pages.

Each page is 4K so total pages take up 1024\*4K which is 4096K (\_of second level table\_)

You need to add an extra 4K to allow for the 1st level table.

So the answer is (c) 4100K

15) Given the following page table, map the (base 10) virtual addresses to physical addresses.

|  |  |
| --- | --- |
| 0 | 200 |
| 1 | 201 |
| 2 | 22 |
| 3 | invalid |
| ... | invalid |
| 25 | 24 |
| ... | invalid |

Addresses: 500, 8200, 102400

(a) 500, 8200, 102400

(b) 819700, 90120, 98304

(c) 823796, 90120, 102400

(d) 819700, 823304, 102400

(e) 823796, 823304, 98304

(f ) A page fault will occur

(g) A segmentation fault will occur

(h) none of the above

Discussion:

The table will always map virtual to physical.

The first step is to split the virtual address into page and offset:

500 = 500/4096 ; 500%4096

= 0 ; 500

8200= 2 ; 8

102400=25;0

Now page 0 maps to page 200 so 500

maps to 200\*4096+500=819700

Hmm ok that leaves (b) and (d) as options so far.

Page 25 maps to frame 24 which starts at 98304

So that rules out all the normal options.

Page 2 is valid, page 0 is valid and page 25 is valid

So that rules out page and segmentation faults.

The only option left is (b)