

15213 Recitation

Caches and Style

Rohan Aletty

10/03/2011

Today's Outline

- ❖ Buflab Review
- ❖ Coding Style
- ❖ Caches
- ❖ Cachelab
- ❖ Recap

Today's Outline

- ❖ Buflab Review
- ❖ Coding Style
- ❖ Caches
- ❖ Cachelab
- ❖ Recap

Buflab

- Due tomorrow night!
 - Hopefully, most of you have completed the firecracker stage by now
- Requires a pretty heavy understanding of:
 - x86 calling convention
 - stack discipline
 - buffer overflow

Buflab Review

- The idea of buffer overflow:
 - (1) Fill up the buffer with random junk
 - (2) Overwrite the elements past the buffer
return address, ebp, etc.
- What happens when we overwrite the return address?

Buflab Review (2)

Suppose we have a function such that it has:

```
char buf[8];  
strcpy(buf, userInput);
```

Stack structure:


Stack Content		Explanation
0x90 91 04 08	Return address	Next instruction in previous stack frame.
0x34 8B 68 55	Old %ebp	
buf[7] buf[6] buf[5] buf[4]	Last bytes of buffer	This is a buffer that holds will hold the first 8 characters of a user input string.
buf[3] buf[2] buf[1] buf[0]	First bytes of buffer	

Buflab Review (3)

- Say the user gives an 8-char string with some more stuff...
 - an overwrite of ebp
 - a pointer to the start of exploit code
 - the exploit code itself
- What will happen when the program runs?

Buflab Review (4)

Basic Idea:

Stack Content		Explanation	
exploit code	Start of exploit code	This address is 0x55688BA0.	
exploit code		The exploit code could potentially be something that is very harmful. Even worse, you may never find out about the exploitation and what was harmed.	
exploit code			
exploit code			
0xA0 8B 68 55	Return address	This returns to start of exploit code.	
0x34 8B 68 55	Old %ebp		
buf[7] buf[6] buf[5] buf[4]	Last bytes of buffer		
buf[3] buf[2] buf[1] buf[0]	First bytes of buffer		

Buflab Review (5)

- Make sure you guys read the writeup!
- Talks about the logistics of the lab:
 - how to use hex2raw
 - how to run bufbomb (from gdb too)
 - how to make your cookie
 - and most of your other setup questions
- If you have any other questions, come to office hours.

Buflab Tips

Aspects to consider:

- whether ebp needs to be uncorrupted
- the endianness of the input
- where to place of cookie
- handling random stack positions (nitro)
 - read up on the “nop sled”

Ways to Thwart Buffer Overflow

- Stack Randomization
 - Vary the position of the stack on every run of the program
 - Causes variation in stack addresses
- Corruption Detection
 - Store a random “canary” value between buffer and the rest of the stack
 - Check canary to see if it is corrupted before restoring register state and returning from function

Today's Outline

- ❖ Buflab Review
- ❖ **Coding Style**
- ❖ Caches
- ❖ Cachelab
- ❖ Recap

Style

- We grade code style because your code needs to be readable and explainable.
- It's good practice for when it becomes necessary in other courses and in industry.
- Guidelines are non-exhaustive.

Documentation

- Good documentation is important
 - Explain tricky parts or large blocks of code
 - Talk about why a piece of code may not have worked
- Preface each function with a header that describes what it does
- The idea is also **not** to comment every line of code
- Documentation shows us that you know what your code does!

Code Readability

- Use whitespace well
 - Indent when using loops or conditional statements
- Keep each line length to under 80 characters
- Use descriptive variable names
 - Helps us understand the variable's function

Code Readability (2)

- Remember to clarify magic numbers by using “#define” in the beginning of the file
 - This way, you can change the #define statement instead of changing every instance of the number in the code
 - Or define the magic number as a static global variable
 - .. Or define the magic number as an enum
 - ... Or define the magic number as a local const variable
- Get rid of dead code (code that will not be run with the program)
 - i.e. printf statements, unused variables, etc.

Failure Conditions and Error Checking

- Don't assume correct inputs/outputs!
- Look at and handle the wrong cases as well
 - i.e. malloc may return NULL, wrong number of inputs can be put into program, the inputs themselves may not be valid
- There are many different ways to handle and resolve these errors

Memory and File Handling

- Whenever you dynamically allocate memory (using malloc or calloc), remember to free it at some point
 - There should be no memory leaks
- If you open a file, make sure it gets closed

Consistency

- On a last and very important style note, keep your code consistent
 - Don't document some places and not others, indent at random, use both descriptive and non-descriptive variable names, etc.
- It is very distracting to see random style changes when grading

Any More Issues With Style?

The style guideline is up on the course website for any further issues with style. Again, following those guidelines will earn you full credit.

<http://www.cs.cmu.edu/~213/codeStyle.html>

Today's Outline

- ❖ Buflab Review
- ❖ Coding Style
- ❖ Caches (Intro to tomorrow's lecture)
- ❖ Cachelab
- ❖ Recap

Today's Outline

- ❖ Buflab Review
- ❖ Coding Style
- ❖ Caches (Intro to tomorrow's lecture)
 - ❖ Locality of reference
 - ❖ Cache Memory Organization
- ❖ Cachelab
- ❖ Recap

Locality

Locality

- data we are referencing is near other recent data references or has been recently referenced itself

Temporal Locality

- A memory location that is recently accessed is likely to be accessed again in the near future.

Spatial Locality

- Nearby elements of a recently accessed memory location are likely to be accessed.

Locality (2)

Does this code exhibit good locality and why?

```
int sum_array1(int a[M][N])
{
    int i, j, sum = 0;

    for(j = 0; j < M; j++)
        for(i = 0; i < N; i++)
            sum += a[i][j];

    return sum;
}
```


Locality (3)

Is this better? Why?

```
int sum_array2(int a[M][N])
{
    int i, j, sum = 0;

    for(i = 0; i < M; i++)
        for(j = 0; j < N; j++)
            sum += a[i][j];

    return sum;
}
```

Summary of Locality

- Programs referencing the same variables have temporal locality
- Programs with lower stride reference patterns have good spatial locality
 - Stride-k means every k-th element of a contiguous vector is referenced => if k is large, why is this bad?
 - The lower the stride, the better (stride-1 is the best)

Today's Outline

- ❖ Buflab Review
- ❖ Coding Style
- ❖ Caches (Intro to tomorrow's lecture)
 - ❖ Locality of reference
 - ❖ Cache Memory Organization
- ❖ Cachelab
- ❖ Recap

Generic Cache Organization

- Consider a computer system having M unique memory addresses
- A cache can be organized for the system by having S cache sets, E cache lines per set, and B cache blocks per line
 - Each cache line has its own “tag” bits to identify that line
- There will also be a valid bit that will indicate whether or not the line is meaningful

Cache Address Structure

$M = 2^m$ unique addresses \Rightarrow m address bits

$S = 2^s$ cache sets \Rightarrow s “set” bits

E cache lines (per set) identified through t “tag” bits

$$t = m - (b + s)$$

$B = 2^b$ data blocks per line \Rightarrow b “block” bits

The total cache size (not including tag or valid bits) is:

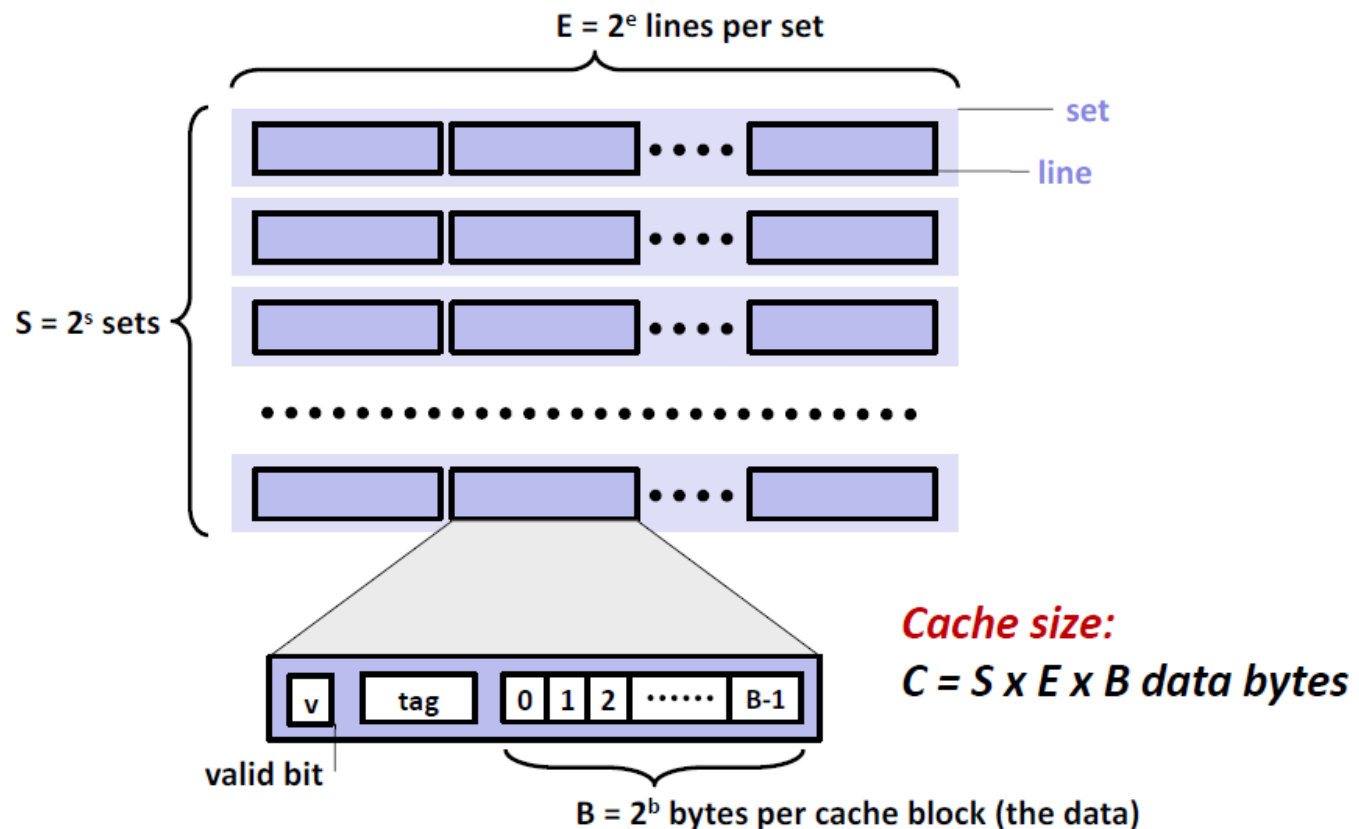
$$C = S * E * B$$

Address bit structure:



Cache Address Structure (2)

A visual diagram:



Cache Access

- When the CPU is instructed to read a word from an address in memory, it sends the address to the cache
- If the cache contains a copy of the word at the address, the word will be sent back to the CPU
 - Avoids going to the hard disk for the word
- The cache can tell if it has the word or not by simply examining the address sent to it by the CPU
 - Like a hash table (with a simple hash function)

How The Cache Accesses

- The s set bits index into the array of S sets in the cache
 - First set is 0, second set is 1, etc.
- We then use the tag bits (t) in the address to let us know if the cache line exists in the set
 - A line only contains the word if its tag bits are the same as the address's and if the valid bit is set
- Finally, we use the b block offset bits to index into the B-byte data block and retrieve the word

Confused Yet?

- If this structure is difficult to grasp now, it will become much clearer to you in the following lectures...
- A thorough understanding of this generic cache structure is essential for cachelab. Make sure you understand how a cache works before you try implementing one.

Today's Outline

- ❖ Buflab Review
- ❖ Coding Style
- ❖ Caches
- ❖ **Cachelab**
- ❖ Recap

Cachelab

Part 1:

You will be asked to create a cache simulator (not an actual cache!) that will record the hits, misses, and evictions of the cache.

Part 2:

You will be asked to write code to calculate some matrix operations that efficiently use the cache.

Cachelab (2)

You will not have any starter code for this lab...

- This is important because you will need to be able to create programs from scratch at some point in the future

Some necessary items for doing the lab include knowledge on:

- Parsing using getopt()
- Creating and using a Makefile
- How to open and read from a file

getopt

- `getopt()` automates parsing elements on the unix command line
 - Typically called in a loop to retrieve arguments
 - Its return value is stored in a local variable
 - When `getopt()` returns -1, there are no more options
- To use `getopt`, your program must include the header file `unistd.h`

getopt (2)

- A switch statement is used on the local variable holding the return value from getopt()
 - Each command line input case can be taken care of separately
 - “optarg” is an important variable => it will point to the value of the option argument
 - Will be useful in handling what (S,E,B) will be set to
 - The atoi() function will be needed here
- Also, remember to think about how to handle invalid inputs!

Makefile

- Makefiles are handy files that spare you the burden of recompiling many source files on the command line whenever you want to update your program.
 - Always entitled Makefile
 - Invoked using the “make” command
 - Tutorial on creating one given on later slide

fopen

- The `fopen()` function opens an I/O stream to a file and returns a pointer to that stream
 - Two parameters: filename, open type (r, w, etc.)
- In this lab, it will be useful for you to know how to open a file a read from it using `fopen()`
 - Will be used to open trace files
- Remember to use `fclose()` on any files you open!

fscanf

- The `fscanf()` function is just like the `scanf()` function except it can specify a stream to read from (`scanf` always reads from `stdin`)
 - Parameters: file pointer, format string with information on how to read file, and the rest are pointers to variables to storing data from file
 - Typically want to use this function in a loop until it hits the end of the file
- `fscanf` will be very useful in the lab to read from an external file containing all the traces

Tutorials

- There are many places online that give great tutorials on what we talked about (getopt, fscanf, Makefiles, etc.)
 - A few good ones are on the last [slide](#)
- If you were not already familiar with the above functions or what a Makefile is, it would be a good idea to start this lab early!

Today's Outline

- ❖ Buflab Review
- ❖ Coding Style
- ❖ Caches
- ❖ Cachelab
- ❖ **Recap**

Style

- Remember that coding style is a part of your final lab grade
- Your code needs to be functional as well as readable
- The style guideline is on the course website for a thorough explanation of style factors

Caches

- Locality refers to the fact that data we are referencing may have either been accessed recently (temporal) or near the recently accessed data (spatial).
- Caches take advantage of locality
- Generic cache structure involves the cache being split up into sets, lines, and blocks

Cachelab

- It will be given to you with no starter code
- Start early if you're not used to creating a program from scratch
- Remember to use getopt, a Makefile, fscanf, and fopen

Office Hours

As usual, office hours will be from 5:30 to 8:30 PM from Sunday through Thursday at Wean 5207.

Good luck on cachelab!

Tutorials

- Makefile:
 - <http://mrbook.org/tutorials/make/>
- getopt:
 - <http://www.gnu.org/s/hello/manual/libc/Getopt.html#Getopt>
- fscanf/fopen:
 - <http://www.crasseux.com/books/ctutorial/fscanf.html>
 - <http://www.gnu.org/s/hello/manual/libc/Opening-Streams.html>