

A brief intro to

CS61

Systems Programming and Machine Organization

Prof. Matt Welsh

Harvard University

December 13, 2007



CS61

New course in Spring 2008: Tuesday/Thursday 2:30-4:00

Prerequisites: CS50 (or C programming experience)

Can be used for CS concentration breadth requirement (“middle digit”)

Can be used for CS secondary area requirement

You can, and should, take both CS51 and CS61 at the same time!

What is CS61 all about?

- Revealing the mystery of how machines really work!
- Getting “under the hood” of programming at the machine level
- Understanding what affects the performance of your programs:
 - Processor architecture
 - Caching and memory management
 - Processes, threads, and synchronization
- Writing rock solid (and fast) systems code

Why CS61?

- Huge gap between the *concepts* of programming and the *reality*
- This gap is more profound when you start programming in higher-level languages: Java, C++, Scheme, etc.
- Need to understand how machines really work to grasp:
 - Operating Systems
 - Databases
 - Processor Architecture
 - Compilers
 - Networks
- ... and even just to be a good programmer, even if you don't become a Computer Scientist.

*** STOP: 0x0000001E (0x80000003,0x80106fc0,0x8025ea21,0xfd6829e8)
 Unhandled Kernel exception c0000047 from fa8418b4 (8025ea21,fd6829e8)

Dll Base Date Stamp - Name	Dll Base Date Stamp - Name
80100000 2be154c9 - ntoskrnl.exe	80400000 2bc153b0 - hal.dll
80258000 2bd49628 - ncrc710.sys	8025c000 2bd49688 - SCSIPORT.SYS
80267000 2bd49683 - scsidisk.sys	802a6000 2bd496b9 - Fastfat.sys
fa800000 2bd49666 - Floppy.SYS	fa810000 2bd496db - Hpfs_Rec.SYS
fa820000 2bd49676 - Null.SYS	fa830000 2bd4965a - Beep.SYS
fa840000 2bdaab00 - i8042prt.SYS	fa850000 2bd5a020 - SERMOUSE.SYS
fa860000 2bd4966f - kbdclass.SYS	fa870000 2bd49671 - MOUCLASS.SYS
fa880000 2bd9c0be - Videoprt.SYS	fa890000 2bd49638 - NCC1701E.SYS
fa8a0000 2bd4a4ce - Vga.SYS	fa8b0000 2bd496d0 - Msfs.SYS
fa8c0000 2bd496c3 - Npfs.SYS	fa8e0000 2bd496c9 - Ntfs.SYS
fa940000 2bd496df - NDIS.SYS	fa930000 2bd49707 - wlan.sys
fa970000 2bd49712 - TDI.SYS	fa950000 2bd5a7fb - nbfs.sys
fa980000 2bd72406 - streams.sys	fa9b0000 2bd4975f - ubnbs.sys
fa9c0000 2bd5bffd - usbser.sys	fa9d0000 2bd4971d - netbios.sys
fa9e0000 2bd49678 - Parallel.sys	fa9f0000 2bd4969f - serial.SYS
faa00000 2bd49739 - mup.sys	faa40000 2bd4971f - SMBTRSUP.SYS
faa10000 2bd6f2a2 - srv.sys	faa50000 2bd4971a - afd.sys
faa60000 2bd6fd80 - rdr.sys	faaa0000 2bd49735 - bowser.sys

Address	dword	dump	Dll Base	- Name
801afc20	80106fc0	80106fc0	00000000	00000000 80149905 : fa840000 - i8042prt.SYS
801afc24	80149905	80149905	ff8e6b8c	80129c2c ff8e6b94 : 8025c000 - SCSIPORT.SYS
801afc2c	80129c2c	80129c2c	ff8e6b94	00000000 ff8e6b94 : 80100000 - ntoskrnl.exe
801afc34	801240f2	80124f02	ff8e6df4	ff8e6f60 ff8e6c58 : 80100000 - ntoskrnl.exe
801afc54	80124f16	80124f16	ff8e6f60	ff8e6c3c 8015ac7e : 80100000 - ntoskrnl.exe
801afc64	8015ac7e	8015ac7e	ff8e6df4	ff8e6f60 ff8e6c58 : 80100000 - ntoskrnl.exe
801afc70	80129bda	80129bda	00000000	80088000 80106fc0 : 80100000 - ntoskrnl.exe

Kernel Debugger Using: COM2 (Port 0x2f8, Baud Rate 19200)
 Restart and set the recovery options in the system control panel
 or the /CRASHDEBUG system start option. If this message reappears,
 contact your system administrator or technical support group.

How many times have you seen this?

```
% gcc -o myawesomeprogram map.c  
  
% ./myawesomeprogram  
segmentation fault - core dumped  
  
%
```

```
% gdb myawesomeprogram core
```


```
(gdb) where
```

```
#0 0x00001fea in main ()
```

```
(gdb) disass
```

```
Dump of assembler code for function main:
```

```
0x00001fc6 <main+0>:      push    %ebp
0x00001fc7 <main+1>:      mov     %esp,%ebp
0x00001fc9 <main+3>:       sub     $0x28,%esp
0x00001fcc <main+6>:       movl    $0x200, (%esp)
0x00001fd3 <main+13>:      call   0x3005 <dyld_stub_malloc>
0x00001fd8 <main+18>:      mov     %eax,-0x10(%ebp)
0x00001fdb <main+21>:      movl    $0x0,-0xc(%ebp)
0x00001fe2 <main+28>:      jmp     0x1ff2 <main+44>
0x00001fe4 <main+30>:      mov     -0xc(%ebp),%eax
0x00001fe7 <main+33>:      add     -0x10(%ebp),%eax
0x00001fea <main+36>:      movb    $0x42, (%eax)
0x00001fed <main+39>:      lea     -0xc(%ebp),%eax
0x00001ff0 <main+42>:      incl    (%eax)
0x00001ff2 <main+44>:      cmpl    $0x270ffff,-0xc(%ebp)
0x00001ff9 <main+51>:      jle     0x1fe4 <main+30>
0x00001ffb <main+53>:      leave
0x00001ffc <main+54>:      ret
End of assembler dump.
```



Hacking into my account...

- Say I left a program in my home directory that would run a shell as “mdw” if you gave it the right password.

```
% cd /home/mdw
% ./mdwshell
% Enter the password: ****
Congratulations! Running shell...

$ whoami
mdw
```


How would you figure it out?

- Brute force guess? No dice ...

```
% cd /home/mdw
% ./mdwshell
% Enter the password: lameguess
Sorry, wrong!
Emailing President Faust...

%
```

How would you figure it out?

- What if you could read the executable file?

```
% cat mdwshell
ELF@4E4 ($!444ààüüüüü$LD(( (
Qåtd/lib/ld-linux.so.2GNU)E
¬KãÀgUa Hy\²0B)óA9U 6N@``;
__gmon_start__libc.so.6_IO_stdin_
usedfflushexec1putnàinprintfgets
stdoutmalloc__libc_start_mainGLIB
C_2.0ii U%00åSfîè[ÃX<“üÿÿÿ...Òtèèù
´X[ÉÃÿ5èÿ%ìÿ%ðhéàÿÿÿÿ%ôéÐÿÿÿÿ%øhé
Àÿÿÿÿ%ühé°ÿÿÿÿ%hé ÿÿÿÿ%h(éÿÿÿÿh0é
ÿÿÿÿ%h8épÿÿÿ1í^%00áfäðPTRhðhQVhGèÿ
```

How would you figure it out?

- What if you could read the executable file?

```
% od -x mdwshell
0000000 457f 464c 0101 0001 0000 0000 0000 0000
0000020 0002 0003 0001 0000 8440 0804 0034 0000
0000040 0f8c 0000 0000 0000 0034 0020 0007 0028
0000060 0024 0021 0006 0000 0034 0000 8034 0804
0000100 8034 0804 00e0 0000 00e0 0000 0005 0000
0000120 0004 0000 0003 0000 0114 0000 8114 0804
0000140 8114 0804 0013 0000 0013 0000 0004 0000
0000160 0001 0000 0001 0000 0000 0000 8000 0804
```

Disassembly?

```
% objdump -d mdwshell
```

```
...
```

```
080484f0 <check_password>:
```

80484f0:	55	push	%ebp
80484f1:	89 e5	mov	%esp,%ebp
80484f3:	83 ec 14	sub	\$0x14,%esp
80484f6:	a1 14 98 04 08	mov	0x8049814,%eax
80484fb:	89 45 fc	mov	%eax,-0x4(%ebp)
80484fe:	eb 21	jmp	8048521
8048500:	8b 45 08	mov	0x8(%ebp),%eax
8048503:	0f b6 10	movzbl	(%eax),%edx
8048506:	8b 45 fc	mov	-0x4(%ebp),%eax
8048509:	0f b6 00	movzbl	(%eax),%eax
804850c:	38 c2	cmp	%al,%dl
804850e:	74 09	je	8048519
8048510:	c7 45 ec 00 00 00 00	movl	\$0x0,-0x14(%ebp)
8048517:	eb 29	jmp	8048542
8048519:	83 45 08 01	addl	\$0x1,0x8(%ebp)
804851d:	83 45 fc 01	addl	\$0x1,-0x4(%ebp)
8048521:	8b 45 08	mov	0x8(%ebp),%eax
8048524:	0f b6 00	movzbl	(%eax),%eax
8048527:	0f be c0	movsbl	%al,%eax

```
...
```

Disassembly

<code>push</code>	<code>%ebp</code>	► Put the ebp register on the stack
<code>mov</code>	<code>%esp,%ebp</code>	► Copy stack pointer to ebp register
<code>sub</code>	<code>\$0x14,%esp</code>	► Subtract 20 bytes from stack pointer
<code>mov</code>	<code>0x8049814,%eax</code>	► Move 0x8049814 to eax register
<code>mov</code>	<code>%eax,-0x4(%ebp)</code>	► Move eax register to local variable
<code>jmp</code>	<code>8048521</code>	► Jump to address 0x804521
<code>mov</code>	<code>0x8(%ebp),%eax</code>	► Copy 2 nd argument to eax register

Disassembly

<code>push %ebp</code>	➤ Put the ebp register on the stack
<code>mov %esp,%ebp</code>	➤ Copy stack pointer to ebp register
<code>sub \$0x14,%esp</code>	➤ Subtract 20 bytes from stack pointer
<code>mov 0x8049814,%eax</code>	➤ Move 0x8049814 to eax register
<code>mov %eax,-0x4(%ebp)</code>	➤ Move eax register to local variable
<code>jmp 8048521</code>	➤ Jump to address 0x804521
<code>mov 0x8(%ebp),%eax</code>	➤ Copy 2 nd argument to eax register

- Hmmmm ... seems kind of complex.

- Until you take CS61 that is...

- What's this?

```
mov     0x8049814,%eax
```

Looks interesting.

Disassembly

```
% objdump -s mdwshell
...
Contents of section .data:
 8049808 00000000 00000000 00970408 ac860408 .....
```

↑
This is 0x8049808

↑
This is 0x8049814


- This is what the “check_password” routine is looking at. How do we read it?
- Well, the x86 is a **little-endian** processor...
 - Meaning, a four-word byte is stored with the **least** significant byte **first!**
 - So, ac 86 04 08 == 0x080486ac
 - Hmm, that looks like another memory address....

Disassembly

This is 0x80486a4

This is 0x80486ac

Must be the password



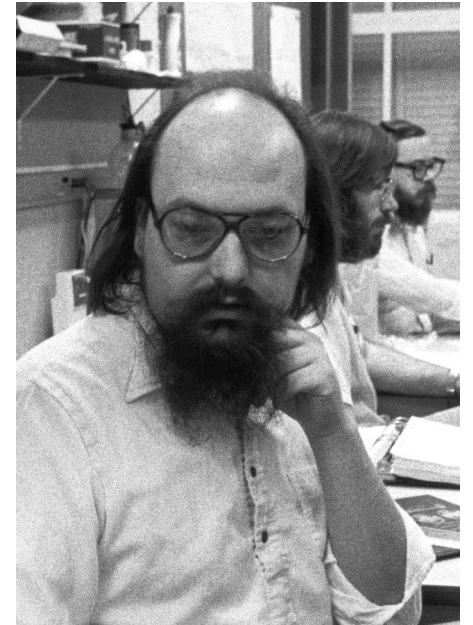
```
80486a4 03000000 01000200 74616b65 63733631 .....takecs61
80486b4 00456e74 65722074 68652070 61737377 .Enter the passw
80486c4 6f72643a 2000436f 6e677261 74756c61 ord: .Congratula
80486d4 74696f6e 7321002f 62696e2f 73680059 tions!../bin/sh.Y
```


“Hack this Binary” Contest

- 1) Go to the CS61 web page:
<http://www.eecs.harvard.edu/~mdw/course/cs61>
- 2) Click on “Binary hacking contest” link at the top
- 3) First one to hack the binary wins a prize (TBD)
- 4) Have fun :-)

Ken Thompson's Compiler Hack

- Ken Thompson – Co-inventor of UNIX
- Won Turing Award in 1983 (with Dennis Ritchie)
- During his award lecture, made a stunning admission...



Thompson's Compiler Hack

- Early days of UNIX: Thompson hacked the “login” program
 - Would accept a “magic” password to let him login on any UNIX system
 - Really helpful for debugging ...
- Problem: The source code for “login.c” was widely distributed
 - The whole system was “open source” (before we had that term...)
 - So, anyone could find the backdoor code!
- So, he hacked the C compiler...
 - C compiler would recognize that it was compiling “login.c”
 - Insert the backdoor code in at compile time

Thompson's Compiler Hack

- Now the backdoor was in the compiler code.
What if someone read that?
- He hacked the *compiler* to recognize when it was compiling *itself*
 - The compiler was itself implemented in C.
 - (Chicken and egg problem: How did they write the first C compiler?)
- The compiler would insert the backdoor code into itself!
 - So when the compiler compiles itself, it would insert the backdoor code to recognize when it was compiling login.c, to insert the backdoor code to check for the magic password. Got it?
- He then deleted the original compiler source code.
 - The backdoor could only be found in the *binary*!

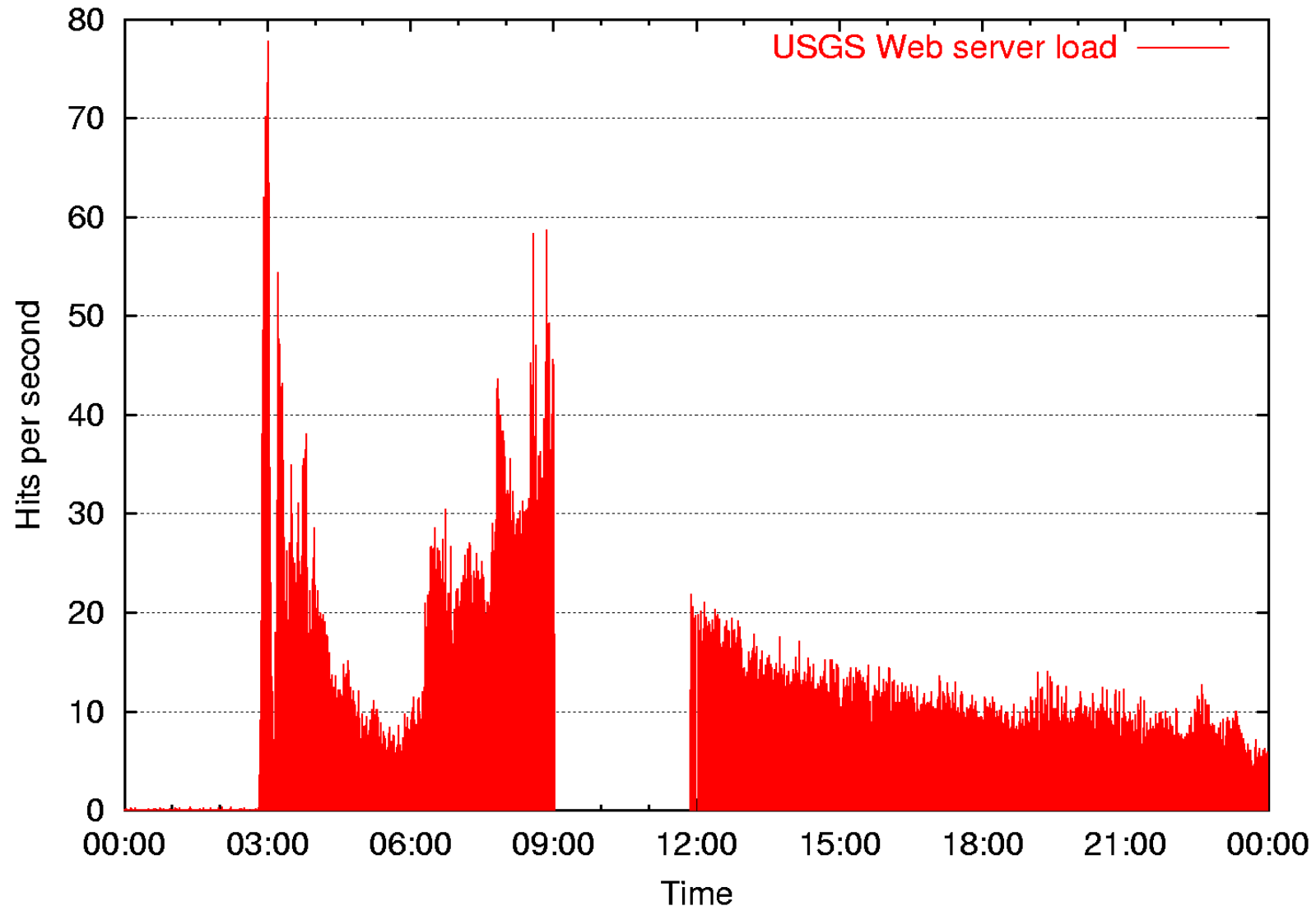
Building Robust Internet Servers

- Imagine you are running Amazon, Yahoo, or Google.
- How do you design your software to handle massive load?
- Tens of thousands of clicks *a second*
- Extremely bursty demand
- Service must be up and running 24/7 with no (discernible) outages
- This is a hard problem!

Overload on the Internet

- Any reasonably popular service on the Internet will experience overload
 - User populations are approximately infinite
 - User demand can be highly correlated (“flash crowds”)
- The peak load can be **orders of magnitude** larger than the average!
- Some examples:
 - CNN on Sept 11, 2001: 30,000 hits/sec, down for 2.5 hours
 - E*Trade sued in class-action lawsuit for not executing trades on time
 - Final Fantasy XI launch in Japan caused all servers to go down for 2 days
- The “Slashdot effect” -- daily frustration to nerds everywhere

God's version of the Slashdot effect



- M7.1 earthquake at 3:00 a.m., October 16, 1999

Why take CS61?

- Learn how machines really work.
 - Use gdb and objdump like an expert.
- Debug the hardest (and most interesting) bugs.
 - Stuff that only makes sense when you can read assembly.
- Hacking binaries for fun and profit.
 - How did the iPhone get jailbroken? The Code Red virus spread so quickly?
- Measure and improve the performance of your programs.
 - Understand memory hierarchies, processor pipelines, and parallelism.
- Write concurrent, multi-threaded programs like a pro.
 - The basis for every application and server on the Internet today.

Er, this sounds really hard...

- CS61 is **not** intended to be a heavy workload course.
 - Challenging, but fun.
 - Intended for everyone who has taken CS50 – not just CS concentrators
- Five lab assignments – can work in pairs:
 - 1) Defusing a binary bomb
 - 2) Hacking a buffer overrun bug
 - 3) Implementing dynamic memory allocation
 - 4) Writing your own UNIX shell
 - 5) Building a concurrent Internet service.
- Two midterms in class. Take-home final exam. That's it.



CS61



CS51



Or take both!

Topics to be covered

- Intel x86 assembly language programming
 - Registers, memory, control flow, procedures, data structures
- Performance measurement and program optimization
- Linking and loading
- Memory hierarchy, caching, and dynamic memory allocation
- UNIX systems programming: files, pipes, signals, processes
- Threads and synchronization
- UNIX sockets programming
- Implementing concurrent servers

Questions?

- Email me! mdw@eecs.harvard.edu
- Or drop by Maxwell Dworkin 233