CSE 127 Week 2 Discussion

Zijie Zhao

This discussion is being recorded

PA0: GDB + x86

- Sunday, April 10 at 6:00pm
- Group submissions
  - Piazza @68
- Goal is to prepare you for the next assignment

Virtual Machine

- Weird stack trace on startup and system doesn't start
  - In advanced boot options, try booting using sysvinit or switch to an older kernel
- VirtualBox throws an error on startup
  - This varies, but on windows it is most likely because you haven't enabled Hyper-V, which there are resources to do here
- SSH is not required, but you need a way to transfer your solution out of the VM

GDB

- Gnu DeBugger
- Allows you to "see" inside your program
  - See registers, memory access, instructions
  - Breakpoints allow you to pause execution at any point

GDB Demo

GDB

- b main → add breakpoint
- info frame → print info about the current stack frame
- x/10x $ebp+4 → show as hex
- x/10i $eip → show as instructions
- x/5c name → show as char
- x/10xw, x/10xh, x/10xb → unit size word(4 bytes)/half(2 bytes)/byte
- disass main → disassemble a function
- tui enable → enable text user interface
- layout src/asm → show source code/assembly
- tui reg general → show registers
- set $ebp = 123 → set value for a register
- set {int}0xfff12345 = 123 → set value for a memory region

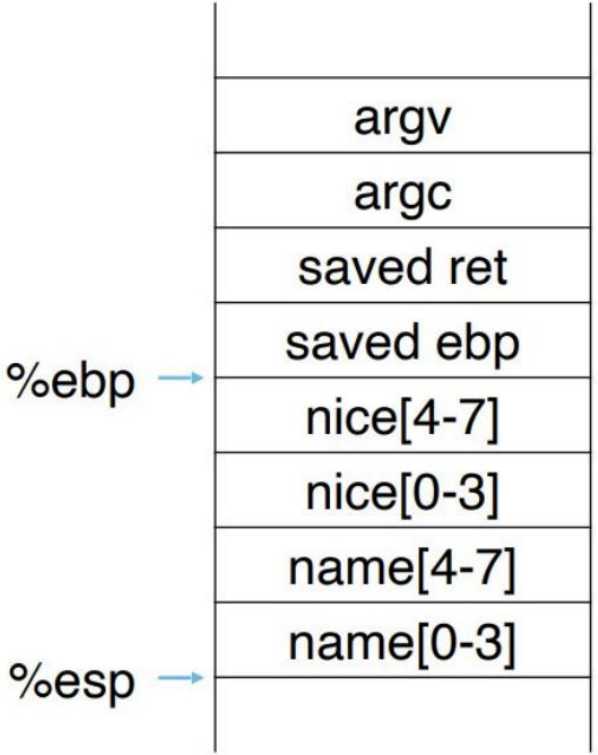
More resources here

echo

- Write a simplified version of the echo utility using the example code provided
- Use only raw x86 assembly code
- Hints:
  - Strings are terminated by a null byte (a null byte has value 0x0)
  - You might need to write a loop
  - You can make more than one system call
  - You can append a `-g` flag to the `ASFLAGS` in `Makefile` to get debugging information generated, but you need to make sure your program also work without the flag

x86 Registers

- %esp, or the Stack Pointer
  - Designates the top of the stack
  - Grows from high to low memory addresses
- %ebp, or the Frame Pointer/Base Pointer
  - Points to middle of stack frame(to the saved base pointer)
  - Doesn't move as function calls are made



x86 Registers

- %eip, or the Instruction Pointer
  - Holds the address of the next instruction to be executed

%eip →

```
pushl   %ebp
movl    %esp, %ebp
subl    $40, %esp
movl    16(%ebp), %eax
movl    %eax, -28(%ebp)
movl    %gs:20, %eax
movl    %eax, -12(%ebp)
xorl    %eax, %eax
```

x86 Registers

- inc %eax → eax
- inc (%eax) → *eax
- inc 4(%eax) → *(eax + 4)
- inc 4(%eax, %ebx, 2) → *(eax + 4 + %ebx * 2)

x86 Instructions

- movl
- cmpb
- je, jne, jmp
- add, sub, inc, dec
- int 0x80

x86 Instructions

- Byte (B)
  - 8-bits
- Word (W)
  - 16-bits = 2 bytes
- Double word (L)
  - 32-bits = 4 bytes
- Quad word (Q)
  - 64-bits = 8 bytes