What is the Course About?

- (Systems) Knowledge is a great enabler!
  - Power -☺
  - How hardware (processors, memories, disk drives, network infrastructure) plus software (operating systems, compilers, libraries, network protocols) combine to support the execution of application programs
  - How you as a programmer can best use these resources
  - Become more effective programmers
    - Able to find and eliminate bugs efficiently
    - Able to understand and tune for program performance

- Not really integers and reals
- Is $x^2 >= 0$?
  - Lets see
- What about x+y+z
  - (1e20 + -1e20) + 3.14 --> 3.14
  - 1e20 + (-1e20 + 3.14) -->

```
        a += b;
    }
```
Actual : a = 1000.220703

- Computer Arithmetic
  - Does not generate random values
    - Arithmetic operations have important mathematical properties
  - Cannot assume all "usual" mathematical properties
    - Due to finiteness of representations
    - Integer operations satisfy "ring" properties
      - Commutativity, associativity, distributivity
    - Floating point operations satisfy "ordering" properties
      - Monotonicity, values of signs
  - Observation
    - Need to understand which abstractions apply in which contexts
    - Important issues for compiler writers and serious application programm

Programmer Centric Course

- By knowing more about the underlying system, you can be more effective as a programmer
  - Enable you to
    - Write programs that are more reliable and efficient
    - Incorporate features that require hooks into OS
      - E.g., concurrency, signal handlers
  - Cover material in this course that you won't see elsewhere
  - Not just a course for dedicated hackers
    - We bring out the effective programmer [hidden hacker 🤔] in everyone!

Reality is More than Big-Oh

- Constant factors matter too!
- And even exact op count does not predict performance
  - Easily see 10:1 performance range depending on how code written
  - Must optimize at multiple levels: algorithm, data representations, procedures, a loops
- Must understand system to optimize performance
  - How programs compiled and executed
  - How to measure program performance and identify bottlenecks
  - How to improve performance without destroying code modularity and genera

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

4.3ms

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

81.8ms

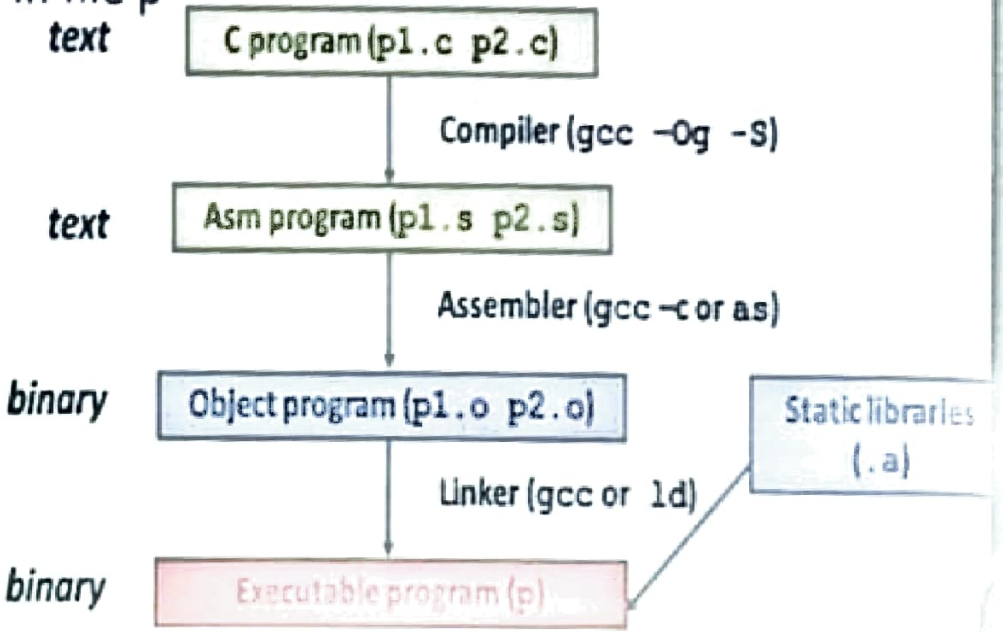
2.0 GHz Intel Core i7 Haswell

A Program lives in Society ☹

- They need to get data in and out
  - I/O system critical to program reliability and performance
- They communicate with each other over networks
  - Many system-level issues arise in presence of network
    - Concurrent operations by autonomous processes
    - Coping with unreliable media
    - Cross platform compatibility
    - Complex performance issues

Turning C into Object Code

- Code in files p1.c p2.c
- Compile with command: gcc –Og p1.c p2.c -o p
  - Use debugging-friendly optimizations (-Og)
  - Put resulting binary in file p



Compiling Into Assembly

- Machine Specific
  - gcc -Og -S compileExample.c

```
long plus(long x, long y);

void sumstore(long x, long y,
              long *dest)
{
    long t = plus(x, y);
    *dest = t;
}
```

```
        .file   "compileExa
        .text
        .globl  sumstore
        .type   sumstore, @
sumstore:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbx
        .cfi_def_cfa_offset
        .cfi_offset 3, -16
        movq    %rdx, %rbx
        call    plus@PLT
        movq    %rax, (%rbx)
        popq    %rbx
        .cfi_def_cfa_offset 8
        ret
        .cfi_endproc
.LFE0:
        .size   sumstore, .-sumstore
        .ident  "GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
        .section        .note.GNU-stack,"",@progbits
        .section        .note.gnu.property,"a"
        .align 8
        .long   1f - 0f
        .long   4f - 1f
        .long   5
```

```
sumstore:
    pushq   %rbx
    movq    %rdx, %rbx
    call    plus
    movq    %rax, (%rbx)
    popq    %rbx
    ret
```

And machine Code

- Assembler
  - Translates `.s` into `.o`
  - Binary encoding of each instruction
  - Nearly-complete image of executable code
  - Missing linkages between code in different files
- Linker
  - Resolves references between files
  - Combines with static run-time libraries
    - E.g., code for **`malloc`**, **`printf`**
  - Some libraries are *dynamically linked*
    - Linking occurs when program begins execution

```
*dest = t;
```

- **C Code**
  - Store value `t` where designated by `dest`

```
movq %rax, (%rbx)
```

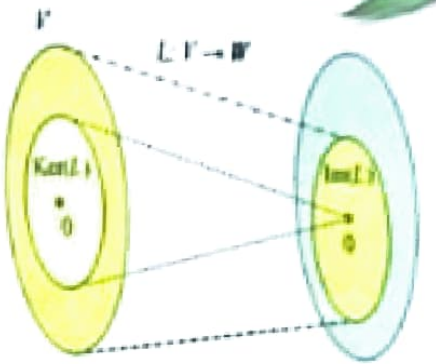
- **Assembly**
  - Move 8-byte value to memory
    - Quad words in x86-64 parlance
  - Operands:
    - `t`: Register `%rax`
    - `dest`: Register `%rbx`
    - `*dest`: Memory M[`%rbx`]

```
0x40059e:  48 89 03
```

- **Object Code**
  - 3-byte instruction
  - Stored at address `0x40059e`

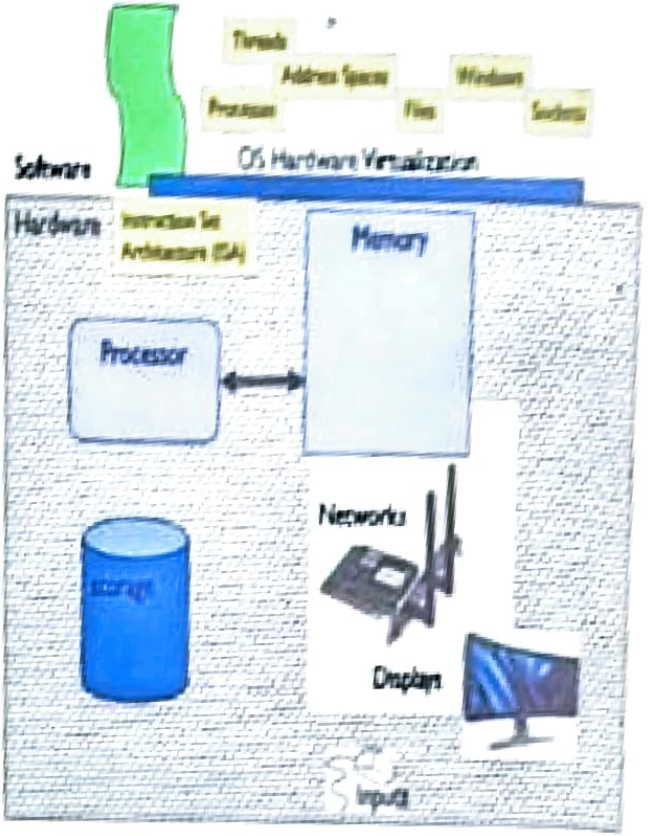
What is an OS?

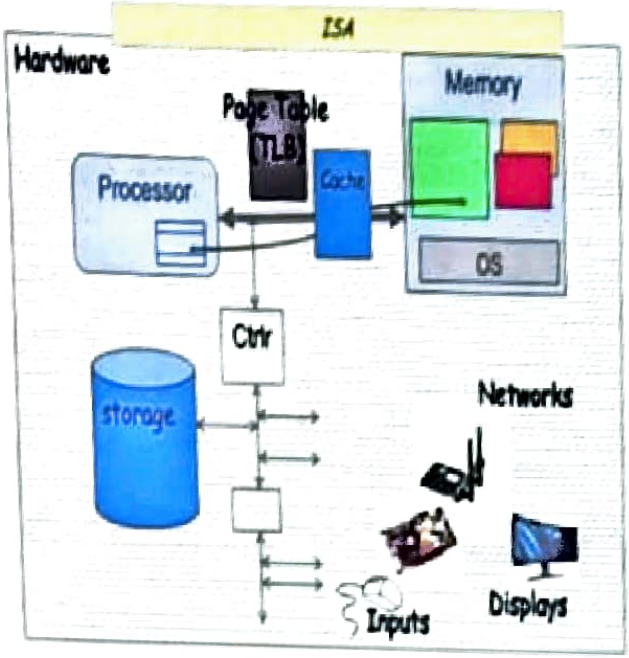
- Something to do with
  - Memory Management
  - I/O Management
  - CPU Scheduling
  - Communications?
    - Does Email belong in OS?
    - Browser
  - Multitasking/multiprogramming?
- No universal definition
  - Kernel



OS in a Nutshell

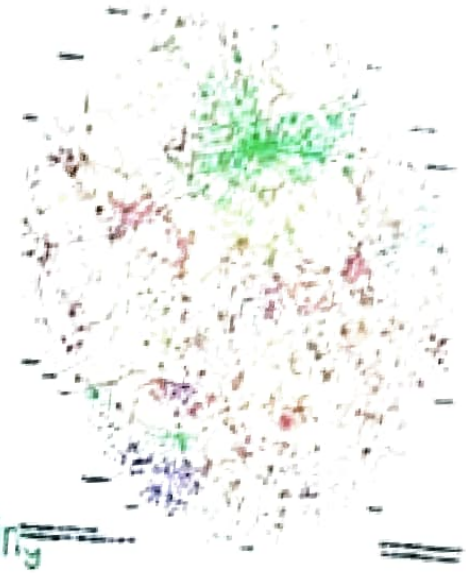
- Computer Architecture
- Magician
  - Create an illusion
    - Infinite memory, dedicated machine
    - Higher level objects: files, users, messages
    - Masking limitations, virtualization





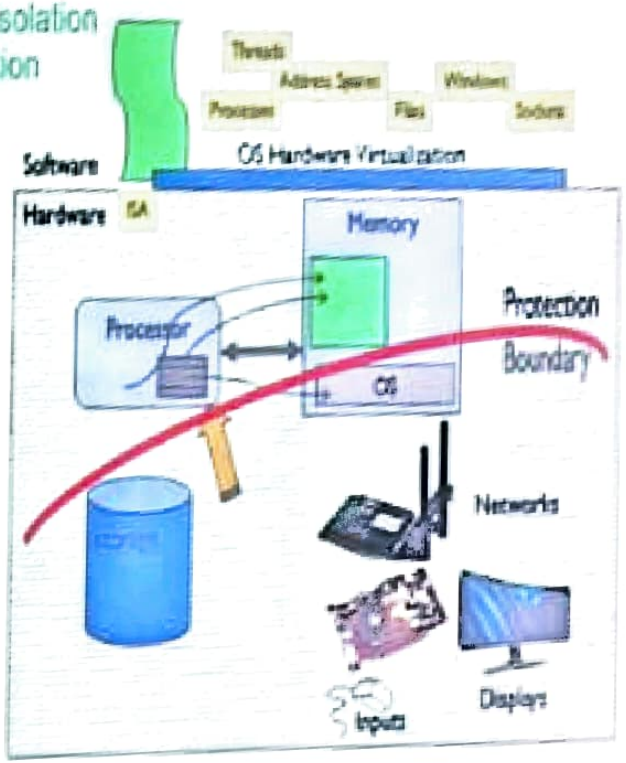
What is an OS?

- What is an Operating System?
  - And – what is it not?
- What makes Operating Systems so exciting?
- Is this a relevant course in the ML Era?
  - Actually Yes
    - Some of you will actually design and build operating systems or components of them.
    - Many of you will create systems that utilize the core concepts in operating systems.
    - All of you will build applications, etc. that utilize operating systems



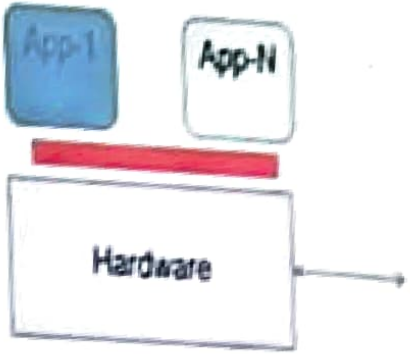
Then we Need to Switch

- Context Switch
  - Referee Role
    - Manage sharing of resources, Protection, Isolation
      Resource allocation, isolation, communication



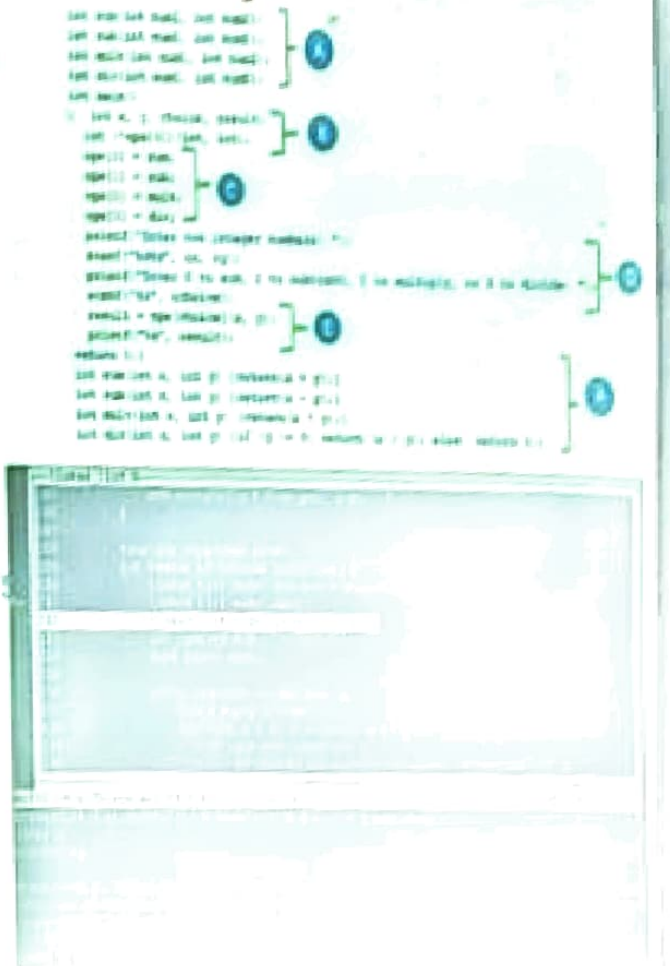
So What is an OS?

- Layer of software that provides application software access to hardware resources
  - Abstraction the hardware
  - Provides Protection to shared resources
    - Access
  - Application
    - Security and authentication
  - Mechanisms for Communication



So How should you be Prepared

- OS is all about Programming (in C)
  - Why?
    - Efficiency
    - Wasted cycles
- Hence this require you to be very comfortable with programming and debugging C
  - Pointers (including function pointers, void*)
  - Memory Management (malloc, free, stack vs heap)
  - Debugging with GDB



be Prepared

```
int sum(int num1, int num2);
int sub(int num1, int num2);
int mult(int num1, int num2);
int div(int num1, int num2);
int main()
{  int x, y, choice, result;
   int (*ope[4])(int, int);
   ope[0] = sum;
   ope[1] = sub;
   ope[2] = mult;
   ope[3] = div;
   printf("Enter two integer numbers: ");
   scanf("%d%d", &x, &y);
   printf("Enter 0 to sum, 1 to subtract, 2 to multiply, or 3 to divi
   scanf("%d", &choice);
   result = ope[choice](x, y);
   printf("%d", result);
return 0;}
 int sum(int x, int y) {return(x + y);}
```

A

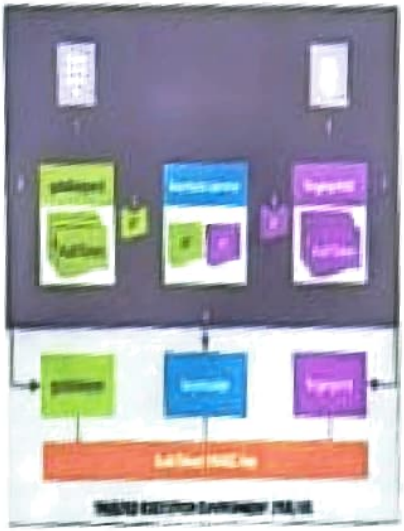
B

C

E

Tying Things Together

- Glue
  - Common services
    - Storage, Window system, Networking
    - Sharing, Authorization
    - Look and feel



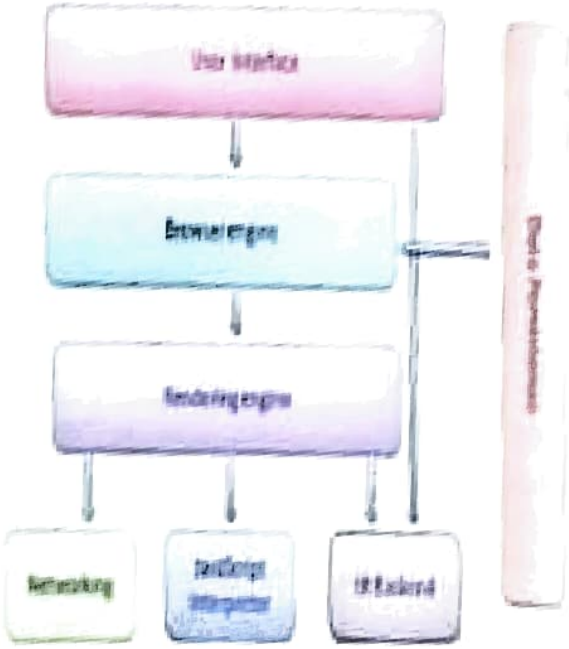
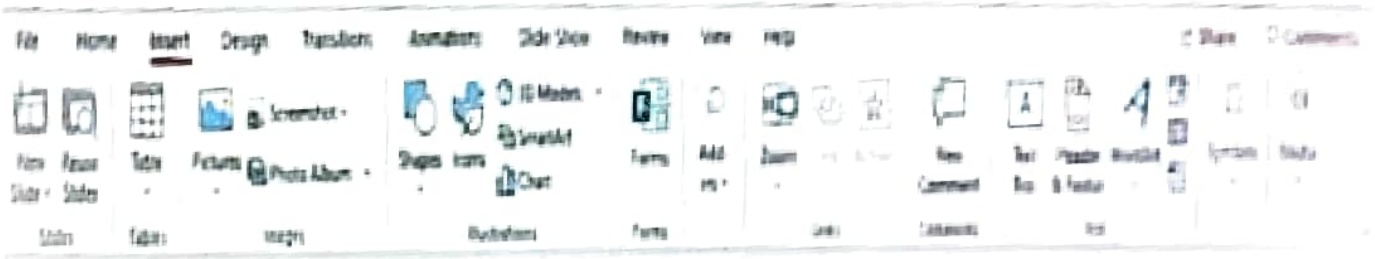
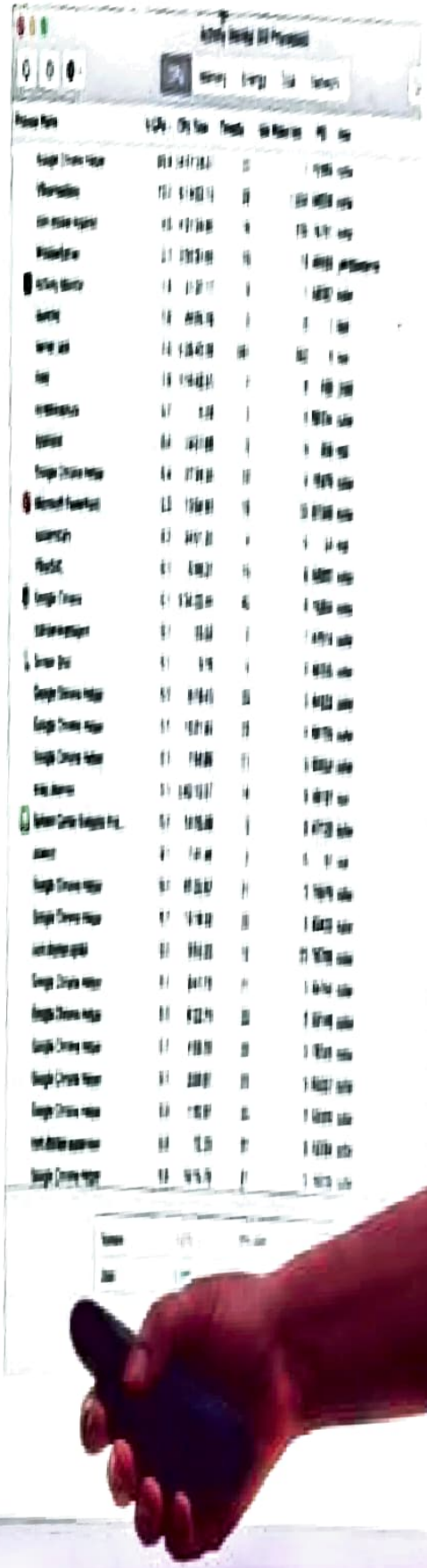


Figure : Browser components



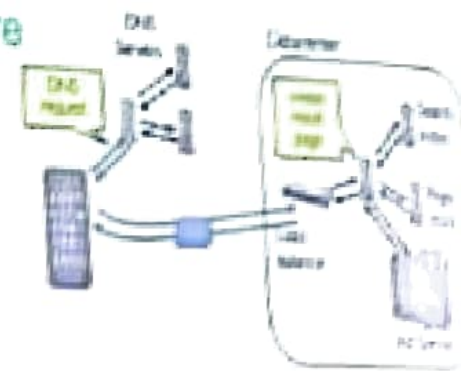
Oops - New Terms

- Process
  - Address Space
  - One or more *threads* of control
  - Additional system state associated with it
- Thread:
  - locus of control (PC)
    - Its registers (processor state when running)
  - And its "stack" (SP)
    - As required by programming language runtime



What is an OS?

- What is an Operating System?
  - And – what is it not?
- What makes Operating Systems so exciting?
- Is this a relevant course in the ML Era?
  - Actually Yes
    - Some of you will actually design and build operating systems or components of them.
    - Many of you will create systems that utilize the core concepts in operating systems.
    - All of you will build applications, etc. that utilize operating systems
  - The better you understand their design and implementation, the better use you'll make of them



Bare Metal to Display -☺

- Lets write
  - test.S
- Compile
  - gcc
- Link
  - ld
- Create a bootable USB
  - dd if=/dev/zero of=usb.img bs=512 count=2880
- Copy the test.bin file to the image
  - dd if=test.bin of=usb.img
- Execute
  - Any Virtual Machine
    - Box
      - bochs
    - Homework (non Graded)
      - VirtualBox or VMWare

```
.code16                     #generate 16-bit code
.text                       #executable code location
        .globl _start;
_start: #code entry point
        movb $'X' , %al     #character to print
        movb $0x0e, %ah     #bios service code to print
        int $0x10           #interrupt the cpu now
        . = _start + 510    #mov to 510th byte from 0 pos
        .byte 0x55          #append boot signature
        .byte 0xaa          #append boot signature
```

as test.S -o test.o

ld –Ttext 0x7c00 --oformat=binary test.o –o test.bin

But Assembly – surely a Joke

- So lets try C
  - Performance
- Build the executable
  - Compile
  - Link
  - Copy

```
/*generate 16-bit code*/
__asm__(".code16\n");
/*jump boot code entry*/
__asm__("jmpl $0x0000, $main\n");
/* user defined function to print series of characters terminated by null character */
void printString(const char* pStr) {
while(*pStr) {
        __asm__ __volatile__ ( "int $0x10" : : "a"(0x0e00 | *pStr), "b"(0x0007) );
        ++pStr; }
}
void main() {
        /* calling the printString function passing string as an argument */
        printString("Hello, World");
}
```

- gcc -c -g -Os -march=i686 -ffreestanding -Wall -Werror test.c -o test.o
- ld -static -Ttest.ld -nostdlib --nmagic -o test.elf test.o
- objcopy -O binary test.elf test.bin

Steps for Introducing a System Call

- Ensure that you are in the linux-xx.xx.y directory
- Create your own directory
  - mkdir info
    cd info/
  - Create processInfo.h
    - asmlinkage long sys_listProcessInfo(void);
  - Create processInfo.c
  - Write Makefile and change top level Makefile
    - obj-y:=listProcessInfo.o
    - core -y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ info/
  - Alter the /arch/x86/entry/syscalls/syscall_64.tbl
    - 548 common processInfo sys_processInfo
  - Alter /include/linux/syscalls.h
    - Add asmlinkage long sys_hello(void)
  - Compile
    - sudo make && sudo make modules_install && sudo make install
  - Test

```
#include<linux/kernel.h>
#include<linux/init.h>
#include<linux/sched.h>
#include<linux/syscalls.h>
#include "processInfo.h"

asmlinkage long sys_listProcessInfo(void){ struct task_struct *process;

for_each_process(process){

printk(
"Process: %s\n \
PID_Number: %ld\n \
Process State: %ld\n \
Priority: %ld\n \
RT_Priority: %ld\n \
Static Priority: %ld\n \
Normal Priority: %ld\n",\
process->comm, \
(long)task_pid_nr(process), \
(long)process->state, \
(long)process->prio, \
(long)process->rt_priority, \
(long)process->static_prio, \
(long)process->normal_prio \
);


if(process->parent)
printk(
"Parent process: %s, \
PID_Number: %ld",\
process->parent->comm, \
(long)task_pid_nr(process->parent) \
);

printk("\n\n");

}

}
```