AVL Trees
Heap Sort
Hash Tables

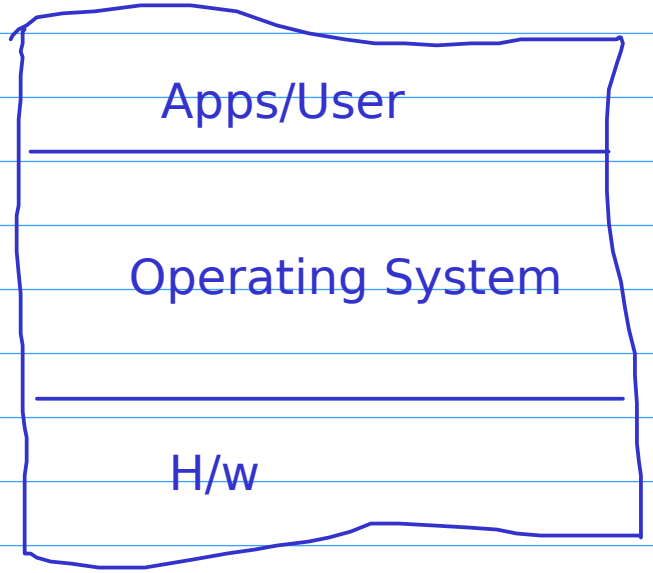
Self-paced:- sorting & searching techniques

Linux System Programming -- User Space Programming

process management
threading
scheduling
signals
IPC - semaphores, mutex, shared memory, message queues
File Systems - pipes,fifos
Virtual Memory -- page tables, mmap etc

i0: 1500
i1: 1504
i2: 1508
i3: 1512
i4: 1516

Introduction to OS:-
* Interface b'n hardware and application (users)
* Resource Manager
* Platform/Basis for executing apps



OS Services:-
* process management
* memory management
* file system management (Files)
* Disk Management (Storage)
* I/O Services
* Network Management
* Protection security
* Exception Handling

Computer Architecture -- High Level Components

```
int a,b,c;
c  = a + b;
```

CPU/Processor -- exec core, Registers, clock, cache
Memory (RAM) -- load/store operation
I/O Devices , including storage devices

Bus -- interconnecting path (System Bus, Data Bus/Address Bus)
power supply

Higher cache levels -- lesser cost, lesser performance, increased capacity
---

CPU Regs ==> Special purpose - stack pointer, frame/base pointer
program counter(PC)/instruction pointer(IP)
FLAGS register / Program Status Word (PSW)
==> General Purpose - Accumulator, others

FLAGS/PSW ==> status bits, control bits

mode bit ==> 0 or 1

supervisor mode (unlimited/unrestricted/privileged mode)
* Can access hardware
* Can access entire instruction set
* Can access entire memory

normal mode (limited/restricted/unprivileged mode)
* no/minimal hardware access
* can access subset of instruction set (typically ALU,CU operations)
* can access subset of memory

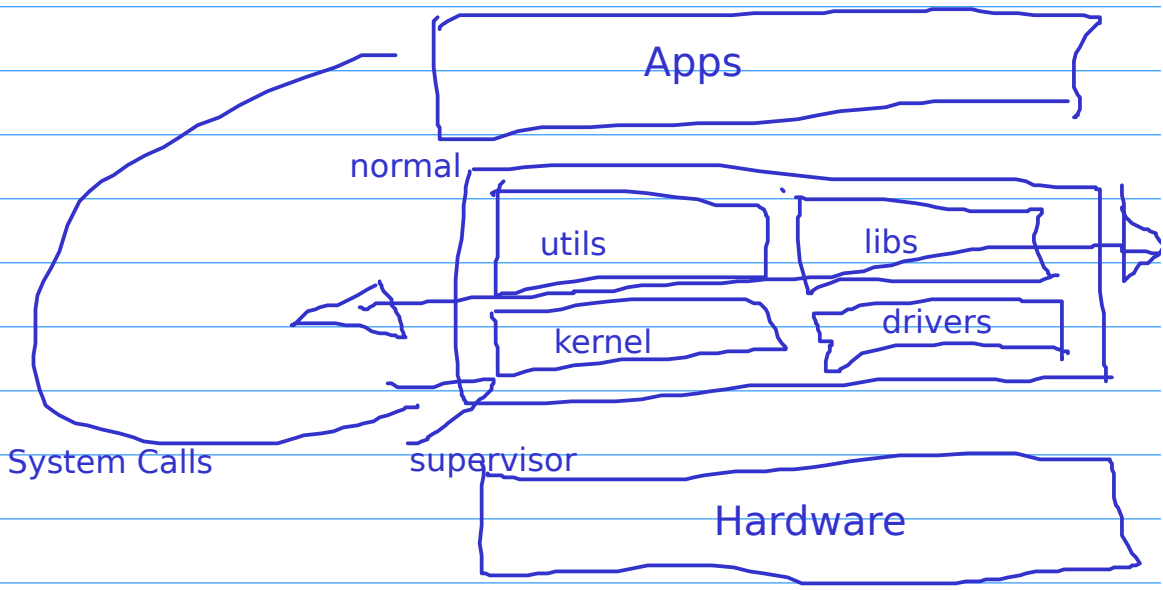
If no mode bit (i.e. low end systems) --> single mode of operations (flat mode)

Trap instruction (s/w interrupt) by CPU:-
normal mode to supervisor mode transition
e.g. int 80h / sysenter in INTEL
swi / svc in ARM

OS Architecture:-
* std apps (utils)
* libraries
* kernel (core component), minimal h/w access - CPU & memory
* device drivers

Kernel:-
* Core component of OS (nutshell
* Provides all services
* Resides in memory all times

* Kernel Mode
* User Mode



Memory Partitioning

Kernel Space (Around 1 GB/25%)
User space (Rest of 75%/3 GB)

mode vs space
user mode -- user space access only
kernel mode -- kernel space + user space (entire memory)

Types of Kernel (self - study):-
* Monolithic Kernel
* Micro kernel
* Modular Kernel

Linux Kernel -- Modular Kernel architecture
-- collection of modules
-- all modules run in supervisor mode(kernel mode)
-- static modules vs dynamic modules

Every device driver is a module

ls /boot/vmlinuz* # compressed kernel image
uname -r

modular kernel:-
    core + static modules ==> /boot/vmlinuz (kernel image)
    dynamic modules ==> /lib/modules

interfacing channel b'n user mode and kernel mode? System Calls

In Linux/Unix ==> Everything is a file

/dev/psaux ==> PS/2 device
/dev/ttyS0 ==> UART device (Serial interface)

Device Files -- special files representing I/O devices

char devices -- char by char, arbitrary size of data
block devices -- block size only (chunk by chunk)

ls -l /dev # starting letter -c or b

I/O Mechanisms:-
* Simple/Direct i/o -- wait until data is ready (1)
* Interrupt Driven I/O -- trigger on event (2)
* Polling -- periodical checking status (3)

Multitasking

FLAGS/PSW ==> interrupt pending bit (1 when interrupt arrives)
  ==> interrupt enable/masking bit

PC/IP ==> check the pending and jump to PC/IP

Interrupts:-
Asynchronous event -- caused by I/O devices, Timers, H/w failures (or CPU)
Utlmost priority -- handle priority
Interrupt Service Routine(ISR) / Handler
ISR Table / Interrupt Vector

TODO:- context saving in case of preemption (emergency save)

System Calls:-

way of consuming kernel service (offerings)
communicating with kernel

bridge b'n user space and kernel space
defined in kernel space , executed in kernel mode
requested/invoked by usespace component

system call calling conventions (protocols) -- ABI
not identified by name/address
indentified by unique number (std header files)

A1 -- requested I/O
A2 -- running when
interrupt arrives
A3 -- other

Scheduler call

```
c=sum(a,b);

int sum(int x,int y) {
   int res=x+y;
   return res;
}
```

Accumulator register

```
int fd=4;
char str[]="Hello Linux";
int len=12;

write(fd,str,len);              //sys call no.for write is 21 (as an example)
```

Trap instruction can't carry system call number

System call invocation:- (ABI -- Application Binary Application)

identify sys call number (from headers)
sys call number --> designated general purpose register (accumulator)
store arguments in other g.p. regs (in x86 - ebx,ecx,edx,esi,edi,xxx)
trap instruction

if arguments are not primitive -- store base addr in register
How to send more values than available regs .. pack them as struct

Ref:- man 2 syscall , for ABI details

```
write(fd,str,len);      // system call wrapper in userspace      0 - stdin
                        // defined in std C library              1 - stdout
                        // prototyped in unistd.h                2 - stderr

printf("abcdxyz\n");

char str[]="abcdxyz\n";
write(1, str, 8);

printf   (vs)  write
----------------------
TODO:-
        open, read, write, close (git repo --> intro --> code)
        strace

        Library API vs System Calls
        ---
        Process Management
        examples - fork etc
```