SMP
Hyperthreading -- physical CPUs, logical CPUs

ls /boot/vmlinuz* # compressed kernel image, size?

2.x, 2.4,2.5,2.6
3.x, 4.x, 5.x

uname -r # major, minor, release, tagname
ls /lib/modules #dynamic modules

----
printf/scanf ==> write system call
scanf/cin ==> read

E.g. in x86 (32 bit)

sys call no --> EAX (Accumulator)
other params --> EBX, ECX, EDX, ESI, EDI
trap

```
int main() {               //stack frame of main / activation record
  int a,b, c;
  a=10, b=20;

}
int sum(int x,int y) {     //stack frame

}
```

API   -- Application Programming Interface

ABI -- Application Binary Interface

"Hello Kernel"	??	base addr
How to send more args	??	addr of structure
How to retrieve multiple results	??	

```
struct box b1;

struct box fetch(int x,int y,int z) {              //in efficient
   struct box temp;
   //fill temp.l, temp.b, temp.h with x,y,z
   return temp;
}

struct box& fetch(int x,int y,int z) {                  //unsafe
   struct box temp;
   //fill temp.l, temp.b, temp.h with x,y,z
   return &temp;
}

int fill(struct Box* pb,int x,int y,int z) {            //efficient & safe
   pb->l=x; pb->b=y;pb->h=z;        //fetch results thru params passed by re
   return SUCCESS;                                      //OUT param
}

struct Box b1;
fill(&b1, 10, 8, 5);
```

```
i=2;
i++*i++*i++                    //undefined behavior in C
                               //Hint:- sequence points, order of evaluation
24, 8, 27, 64
```

https://rules.sonarsource.com/c

Coding Standards - MISRA, SEI CERT
Sonarlint C rules

Static Analysis -- compliance with some coding standard (or mix)
Tools:- Polyspace, Klockwork, Sonarlint, PQRA QAC, LDRA tools
and many more
free/open source:- cppcheck, cpplint, clang-tidy

stdin	- 0
stdout	- 1
stderr	- 2

```
write(fd, buf, len)
write(1, buf, len)          ??
```

printf in C / echo in shell / cout in C++

==> write sytem call

scanf / read cmd / cin ==> read system call, fd as zero

Library calls vs System calls:-

- * ease of use -- library calls
- * portable -- lib calls
- * efficient -- mostly lib calls

```
for(i=1;i<=1000;i++)
   putchar(ch)          //write(1,&ch,1);
```

fflush, __fpurge

printf, scanf -- buffered i/o

x is 10, y is 20

strace

write(fd, str, len); ==> system call wrapper (part of std C library

* identify sys call no. unistd.h)
* store sys call no.,args in register
* trap

open system call wrapper ==> sys_open / do_open

For system calls without wrapper ==>

syscall (no, args)

syscall (SYS_write, fd, buf, len)

Process Management:-

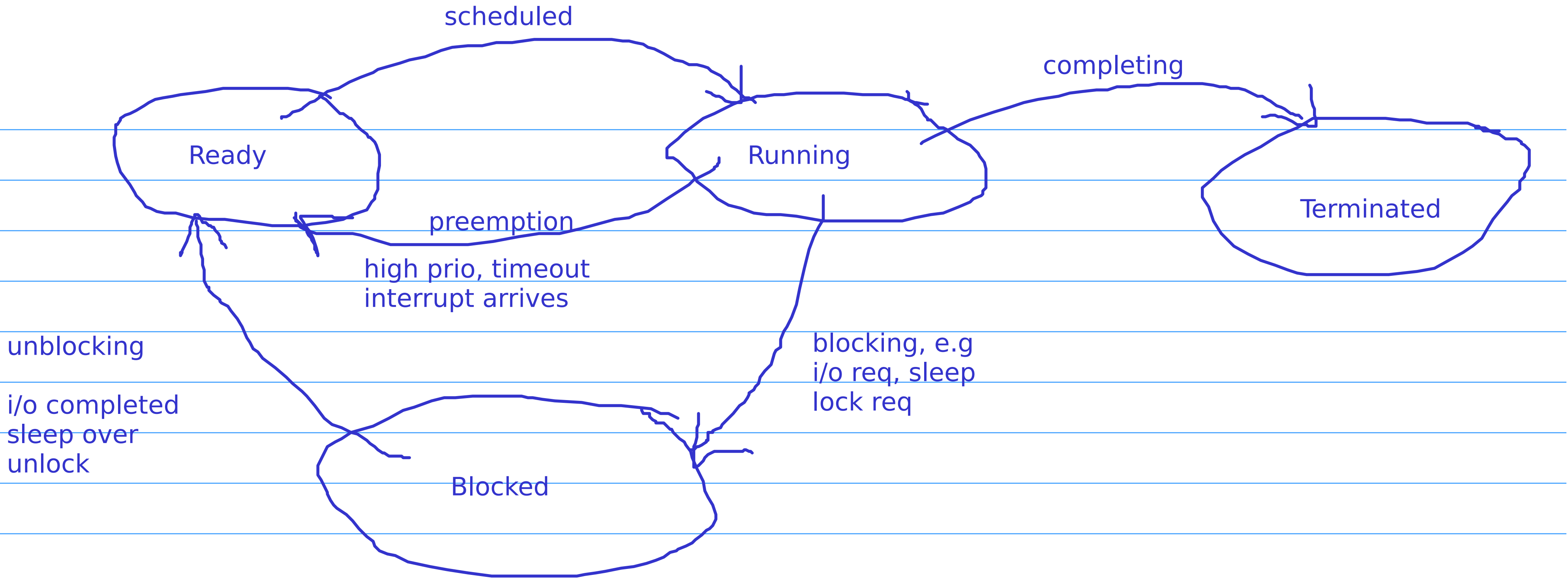
* what is a process? program loaded in memory for execution
* program - on disk (passive entity)

* program sections - code, idata followed by header
* process sections   - code , idata, udata, stack, heap, rodata
(user space)

* every program will have set of virtual resources (logical, multiplexing)

* kernel support for a process
    * process table/list   (typically linked list in Linux)
    * process descriptor/control block -- process attributes
    * unique id known as PID(one of the attributes)

Independent address space for every process , (stack for each process)



scheduled

completing

Ready

Running

Terminated

preemption

high prio, timeout
interrupt arrives

unblocking

i/o completed
sleep over
unlock

blocking, e.g
i/o req, sleep
lock req

Blocked

Context Switching:-

Context - snapshot of current execution
- typically reg values

Context Save Area
- somewhere in memory
- typically top of stack (identified by SP, BP/FP)

P1	P2
1500	1800
1504	1804
1508	1808
1512	1812
1516	

Context Saving -- from CPU to save area
Context Loading -- from save area to CPU

Process Hierarchy:-

parent - child process

a.out --> shell --> terminal --> --> init

init is origin of Linux process hierarchy, pid is 1

```
ps
ps -el             # lengthy listing, of all processes
ps aux             # different style

ps -e -o pid,ppid,stat,cmd               # observe first entry

pstree
pstree -np

top
```

Please Try Commands:- (TODO)

```
kill        <pid>
kill  -9    <pid>
killall     <pid>
pkill       <pid>
pgrep       <pname>

fg
bg
jobs

ctrl + Z
command &
```

System calls & lib calls:-

getpid, getppid, fork, waitpid, exec

exit, sleep

Normal & success	-- exit(0)
Normal & failure	-- exit with +ve failure
Abnormal	-- exceptions

```
execl("/usr/bin/cal", "cal", "2018", NULL);
```

Further topics:-
signals
threading
scheduling

IPC -- semaphores, mutex, shared mem, message queues, pipes

file system

memory management & mapping

SofPrayog links in Reading.md

YouTube ==> Shell Wave , Neso Academy