Examples of
Monolithic -- Traditional Unix
Micro -- RTOS like QNX

Hybrid Approach -- Modular Approach
Modules -- static vs dynamic modules

lsmod, /lib/modules

Simple system calls:-

```
fd=open("sample.txt",O_WRONLY|O_CREAT, 0666);
(or)
fd=open("sample.txt",O_WRONLY);
(or)
fd=open("simple.txt",O_RDONLY);
//system call wrapper in user space
-----
char msg[] = "Hello Unix!";
nbytes=   write(fd, msg, len);
```

system calls return -ve value on failures

0 or +ve means successful

TODO:-

wrsample.c

rdsample.c

alpha.txt/simple.txt ==> 36 bytes

```
int len=10;
nbytes=read(fd,buf,len); //A...J          , nbytes=10
nbytes=read(fd,buf,len); //K...T          , nbytes=10
nbytes=read(fd,buf,len); //U..0123  , nbytes=10
nbytes=read(fd,buf,len); //456789         , nbytes=6
nbytes=read(fd,buf,len); //nbytes will be zero, end of file
```

printf/fprintf vs write

POSIX std

lib APIs vs system calls:-

* interoperability (across OS)
* ease of use
* efficient

```
strace   <command>

strace  cal
strace  ./a.out
```

Process Management:-

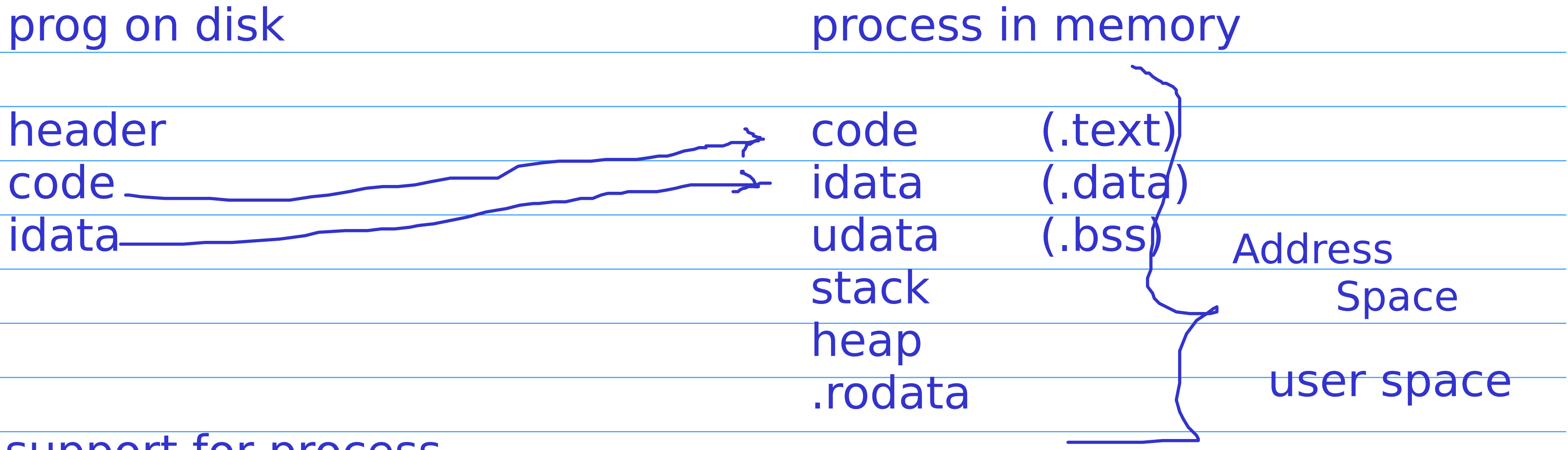
Concept of a Process:-

what is a process?

program loaded in memory for execution -- user oriented (user space)

program on disk   -->   process in memory   (loading)

flow of control -- function calls



prog on disk

header
code
idata

process in memory

code (.text)
idata (.data)
udata (.bss)
stack
heap
.rodata

Address Space

user space

kernel support for process

==> unique ID (pid - process id)

==> data structure with process attributes

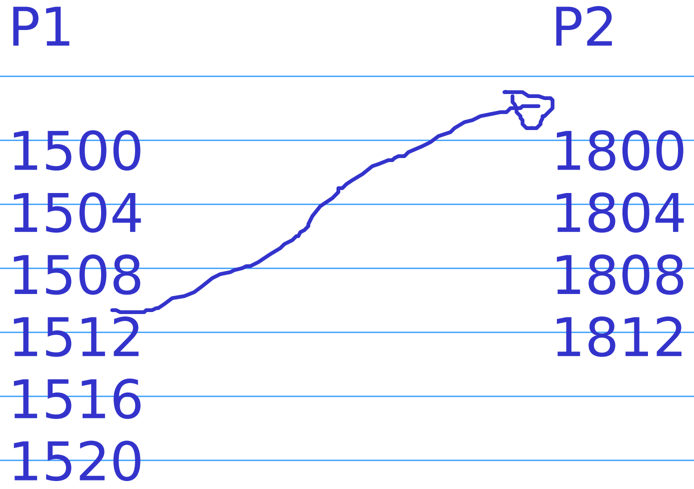
- process control block (PCB) / process descriptor(PD)

==> list of all processes (process table/process list)

Some attributes of a process:-

* pid (process id)	* open file info -- mode
* ppid (parent pid)	* memory description
* name (cmd)	* i/o description
* Priority	* ownership (user & group
* Policy	- accounting)
* Timing info	* exit status

* context of process (reg values)



P1	P2
1500	1800
1504	1804
1508	1808
1512	1812
1516	
1520	

Blocking (on own) - 1

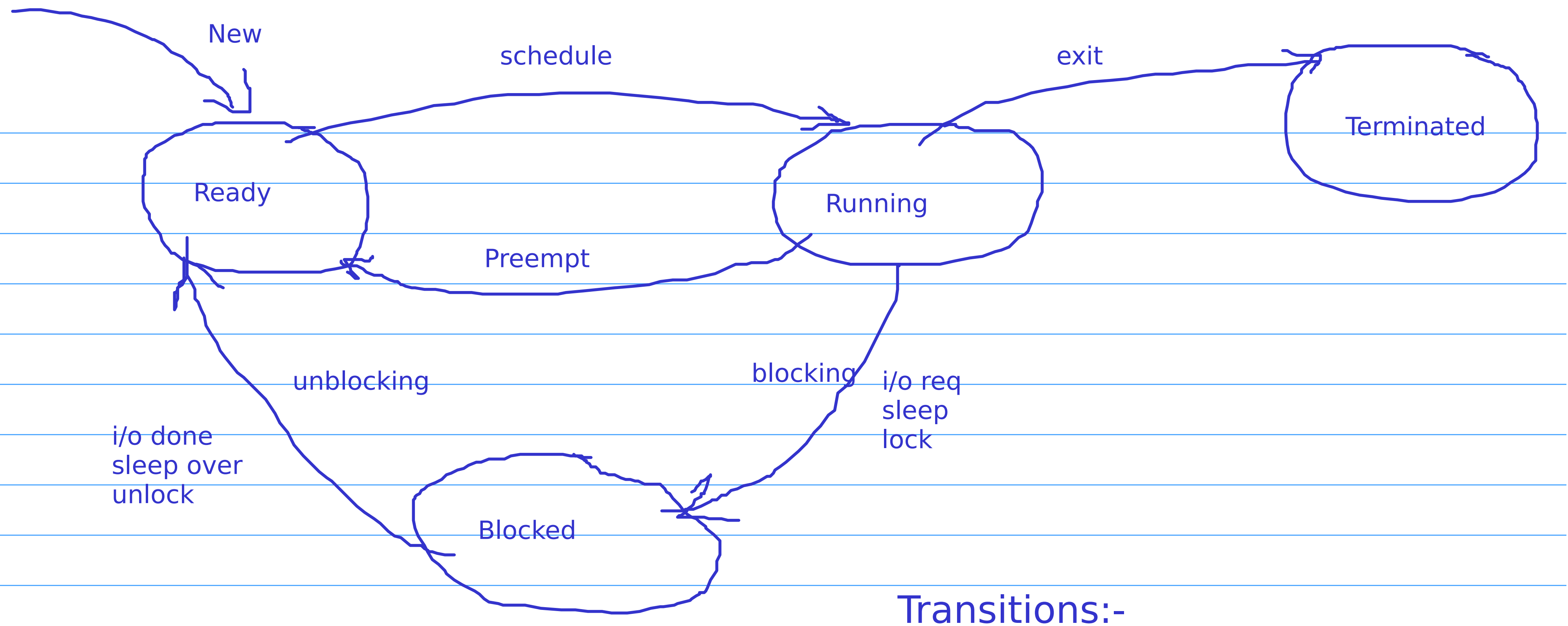
High Prio - 2
Timeout - 3
Interrupt arrives - 4

Preemption - 2, 3, 4

context area -- backup(snapshot) of registers
... typically in memory
... on top of process stack in X86

cotext saving -- copying CPU reg values in context area
context loading -- filling CPU regs as per context area (saved reg values)
context switching -- context saving + context loading

every process will have independent stack (stack frames)
stack frames -- local vars, parameters of functions



Transitions:-
* new
* scheduled
* blocking
* unblocking
* terminating
* preemption

States:-
* Ready
* Running
* Blocked
* Terminated

Process Hierarchy:-
* parent - child relationship
* origin process
    init/systemd in unix/linux
    PID is 1

terminal
  --> shell
    --> commands

webminal.org
repl.it
katacoda.com [Ubuntu Playground]
CoCalc Linux

Commands:-

```
ps
ps   aux
ps   -el
ps   -e -o pid, ppid, stat, cmd

pstree
pstree -np

top                          # q - quit
```

System Calls & APIs

getpid
getppid
fork
waitpid
execl, execlp
exit

sleep
exit

fork - create a new process, known as child

```
int ret;
ret=fork();
if(ret<0)
   perror("fork");
if(ret==0)  {
  //some code for child
  exit(0);
}
else  {  //ret > 0
  //some code for parent
}
```

Example1 - simple fork

Example2a - concurrency, lengthy loop, no delay
Example2b - concurrency, small loop, with delay(sleep)

If parent terminates before child, init/systemd becomes parent of running child (reparenting/adopting)

waitpid:-

At end of parent:-

```
        waitpid(-1, &status, 0);
        //print WEXITSTATUS(status);
```

waitpid 1st param:

- -1 means , waiting for any one child
- +ve val means, specific child

execl:-

```
if(ret==0) {                       //child
  execl("/usr/bin/cal","cal",NULL);
  printf("Thank you");             //not reachable if execl succeeds
}
else {
      //waitpid
      //print status
}
```

/

usr

which cal
which gcc
which ls

```
//execl("/usr/bin/cal","cal","10", "2015", NULL);
//execlp("cal","cal", "10", "2015", NULL);                  //refer PATH variable
```

execl, execlp
execv, execvp
execle, execvpe

TODO:-   kernel, system calls & process management (concepts ,hands-on)

Further:-
* Signal Handling
* Threads
* Scheduling

Beginning Linux Programming - Richard Stones/Neil Mathew (11-15/16)
The Linux Programming Interface(TLPI) by michael kerrsick