Further Topics:-

Hint:- futex

Named Pipes (FIFOs)
Message Queues
Scheduling

Virtual Memory
Shared Memory

Named Pipes/FIFOs

mkfifo command
mkfifo lib API
mknod system call

```
mkfifo /tmp/p1
ls -l /tmp/p1

#In terminal-1
cat /tmp/p1

#In terminal-2
echo "abcdxyz" > /tmp/p1
```

```
mkfifo /tmp/p2
```

writer.c:-
```
fd=open("/tmp/p2", O_WRONLY);
//err handling
write(fd,msg,len);
//err handling
close(fd);
```

reader.c:-
```
fd=open("/tmp/p2", O_RDONLY);
//err handling
read(fd,buf,maxlen);
//print buf
//err handling
close(fd);
```

file <xxxxx>   # based on header

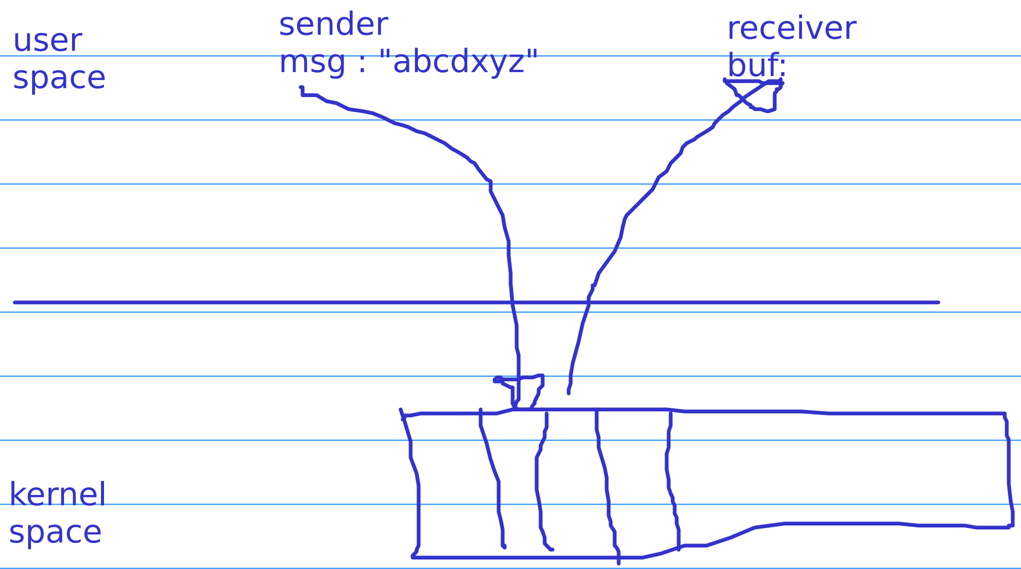
IPC:-   Synchronization, Data Exchange

Sys V Semaphores:-
        semget, semop, semctl
POSIX Semaphores:-
        sem_init, sem_open, sem_destroy
        sem_wait, sem_post
Sys V Message Queues:-
        msgget, msgsnd, msgrcv, msgctl
POSIX Message Queues:-
        mq_open, mq_send, mq_receive, mq_unlink

Semaphores
Message Queues
Shared Memory
|
Sys V Variant
POSIX Variant

POSIX

Message Queues:-
Purpose:- Data Exchange



Internal sync support
Typically FIFO
Discrete messages:-

send m1(20)
send m2(10)

recv with 40 bytes
--> m1 : 20 byes
recv with 15 bytes
--> m2 : 10 bytes
recv
--> block

Steps:-

```
int mqfd;
mqfd = mq_open("/mqsimple", O_RDWR);
mqfd = mq_open("/mqsimple", O_RDWR|O_CREAT,0666,NULL);
```

sender:-

```
char msg[] = "Hello POSIX";
int len=strlen(msg);
mq_send(mqfd, msg, len, 5);
```

receiver:-
```
char buf[MAXLEN];
int maxlen=8192;
int prio=5;
mq_receive(mqfd, buf, maxlen, &prio);
```

close:-
```
mq_close(mqfd);
```

Default msg len : 8192
During receive - maxlen >= msg len
//through attr, can override default msg len

```
mqd_t mqid;
struct mq_attr attr;
attr.mq_msgsize=512;
attr.mq_maxmsg=10;
mqid=mq_open("/mque2",O_WRONLY|O_CREAT,0666,&attr);
```

linker flag:-
        -lrt
```
gcc sender.c -lrt -o sender
```

```
ls /dev/mqueue
cat /dev/mqueue/mqsimple
```

```
man mq_overview
```

mq_setattr
mq_getattr

mq_unlink

Scheduling:-

When do CPU becomes free? When switching required?

* task completion A
* blocking call e.g sleep, read, mq_receive, sem_wait B
* timeout C
* interrupt arrives, end of ISR D

A & B only -- non preemptive
C & D also -- preemptive

Note:- process never sleep/block in userspace
blocking always occur in kernel space (system call)

Classical Policies:-
FCFS/FIFO
Priority Based
Round Robin (Time slice)

Short Job First

O(1) complexity expected -- idle scheduling

Q1 -- telephonic : Priority (real time) FCFS
Q2 -- user interactive : RR (fairness) RR
Q3 -- background/maintenence : FCFS (simpliity) PRIO + FCFS
: RR with larger quantum PRIO + RR

----
processes may move across queues
- explicit change in priority/process
- aging
- downgrade (lot of CPU usage)

Priority policies :- different queues for each priority (goal O(1) complexity)

Scheduling in Linux:-
* Real time policies : 100 priority levels : 0 - 99 (0-min, 99-max)
  SCHED_FIFO : Priority + Fifo
  SCHED_RR : Priority + RR
* Non real time policies
  SCHED_OTHER : Time sharing algo
  : Modified Round Robin

```
ps -e -o pid,ppid,stat,policy,pri,cmd

    sudo chrt -f  5 ./a.out
    sudo chrt -r 15 ./a.out
```

True/Complete Fairness:-

* Every process should be scheduled atleast once, before any other process
  goes for second turn

Limitation of Round Robin:-

Time slice - 2 units

	Time	CPU	Queue
p1 - 0, 4 units	0	p1	empty
p2 - 1, 6 units	1	p1	p2, p3
p3 - 1	1.9	p1	p2,p3,p4
p4 - 1.9	2	p2	p3,p4,p1
p5 - 3	3	p2	p3,p4,p1,p5

Time sharing:-

* Two queues -- active, expired
* Preempted process (timeout) will go back to expired
* New processes are added to active queue
* Scheduler always pickup from Active queue
* When active Q is empty, swapping occurs b'n active & expired

* Priority is applicable , 40 levels
* Pair of active & expires for each prio level

* Dynamic time slice -- proportional to priority

Nice vals -- controls priority in TS Algo (inverse relation)

Range : -20 to +19
default nice value : 0

+6, less prior than 0
-9, more prior than 0

-20 : Highest, 0 : default, +19: lowest

ni	:	0	5	-6	-20	+19	(-20 to +19 scale)
ps o/p	:	19	14	25	39	0	(0 to 39 scale, 0-low, 39-high)

```
nice -n 5 ./a.out
nice -n -6 ./a.out
#renice to change ni value of existing
```

lowering nice vals, -ve ni values are not allowed for normal user
(only for admin/root user)

APIs:- sched_setscheduler,sched_setparam

SMT or Hyperthreading:-
Switching b'n two threads at h/w level

nproc
cat /proc/cpuinfo

AMP : asymmetric, co-processor
e.g. GPU, ARM + DSP

Scheduling in SMP
* SMP : multiple CPU cores, identical

* Common ready Q or private ready Qs ?

* Modern kernels (e.g. Linux kernel > v2.6)
    ==> private ready Q
* CPU Affinity (No Migration), reason - private cache entries

Migration -- discard CPU cache, rebuild on other CPU

Load Balancing Techniques, e.g. Pull Migration, Push Migration

```
taskset 0x1 ./receiver
taskset -p 0x4 <pid>
```

Core i3
Core i5
Core i7

phy	threads	
2	2	
2	4	*
4	4	
4	8	*

SMT
Symmetric Multi Threading
e.g. Hyperthreading

0x1	-- CPU0
0x2	-- CPU1
0x4	-- CPU2
0x8	-- CPU3
0x6	-- 1 or 2

sched_setaffinity, sched_getaffinity
pthread_setaffinity_np, pthread_attr_setaffinity_np

Memory Management / Virtual Memory:-

Myths / Facts:-
* 1 GB RAM + 2 GB swap area   # swap area -- from disk
virtual memory size? 3 GB

```
1 GB        ==>        2 ^ 30
4 GB        ==>        2 ^ 32
```

```
Address < 1GB                              ==> RAM
Address > 1GB, but < 3 GB      ==> Not Applicable
```

Whichever address, CPU will access RAM only (not swap area for active load/store)

Processes (pages) in swap area are not under active execution

Swap area -- backing store

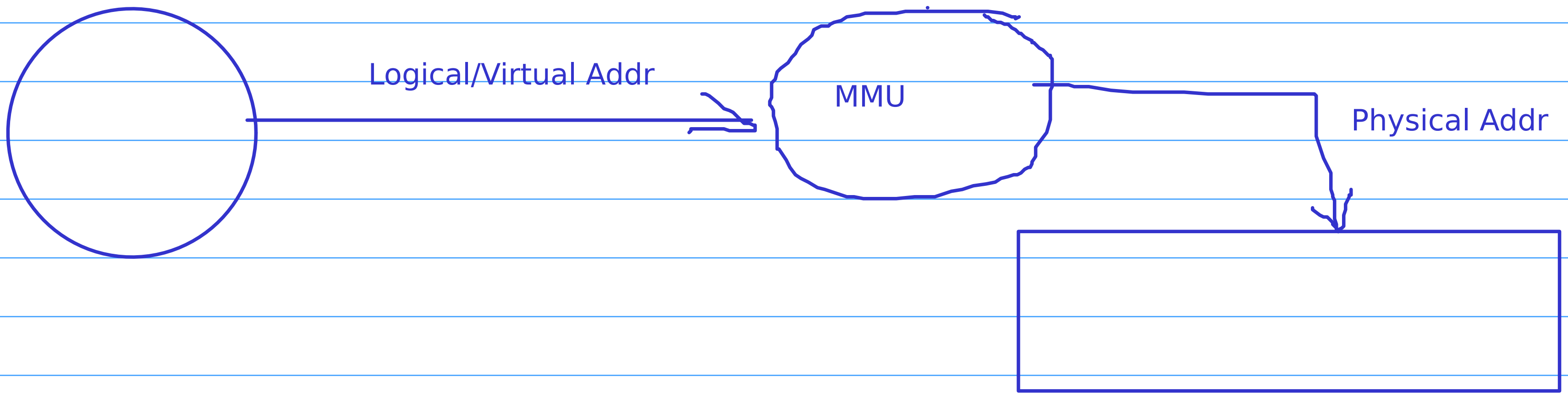
4 GB RAM + 0 swap area   ==> 4 GB of Virtual Mem

Virtual Memory Management:-
* Allocate/Deallocate memory for processes (stack, heap, code, idata, udata)

challenges:-
* How to handle scalable memory
* holes in memory - how to manage with non contiguous memory
* Expand physical memory (+swap area)
* Protection b'n processes (Addresss space)

Memory Mapping:-

compiler & linker generates logical addresses, which are issued by CPU



Pages, Paging:-

* Physical memory is divided into equal size partitions -- Frames
* Logical memory is also divided into same size parts -- Pages
* Page/Frame size is fixed by h/w design

* Pages will be contiguous, but not frames (not necessary)
* Mapping of page no.s and frame no.s --> Page Table
* Page Table -- collection of Page Table Entries (PTEs)

In x86
page size
4 KB

0-4095	- p0	- 15
4096-8191	- p1	- 25
8192-12287	- p2	- 20
12287-16383	- p3	- 12

Logical addr: 4200 ==> p1, offset: 104
Logical addr: 4500 ==> p1, offset: 404
Logical addr: 8400 ==> p2, offset: 208