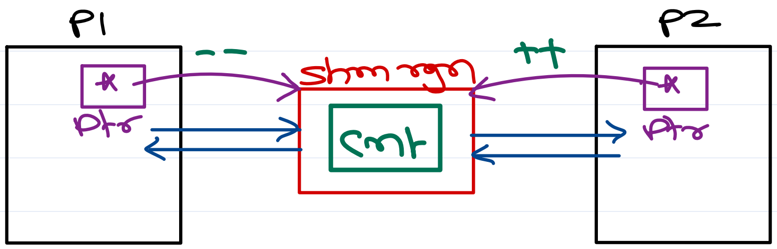
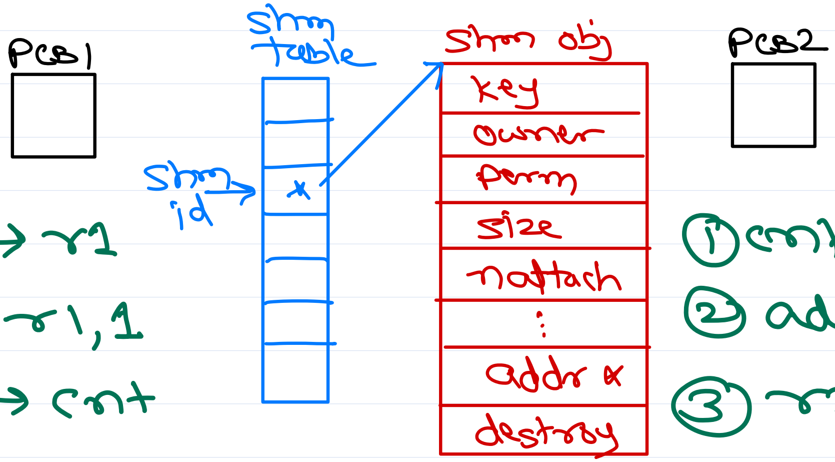
SUNBEAM

Embedded Operating Systems

Trainer: Nilesh Ghule

Synchronization — Semaphore

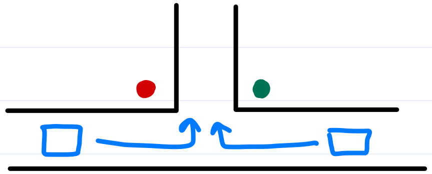




① cnt → r1
② sub r1, 1
③ r1 → cnt

① cnt → r2
② add r2, 1
③ r2 → cnt

race condn: multiple process accessing same resource at the same time.

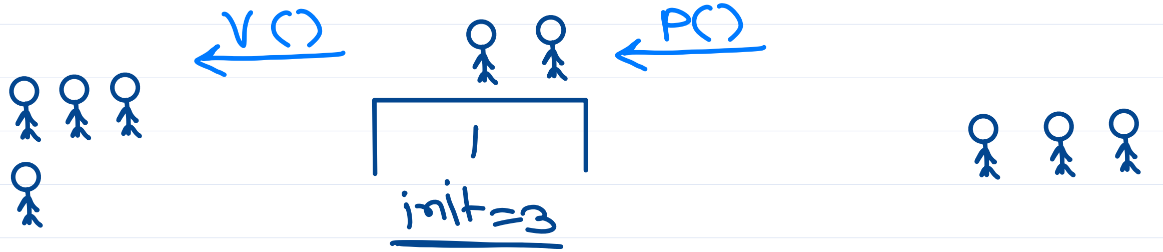


one/more processes can be blocked in order to provide safe access to other processes - synchronization.

Semaphore → sync primitive

- is a counter → operations: inc / dec
- dec op: decr count by 1.
  - if count < 0, the block cur. process.
- inc op: incr count by 1.
  - if one/processes are blocked, wake up one of them.

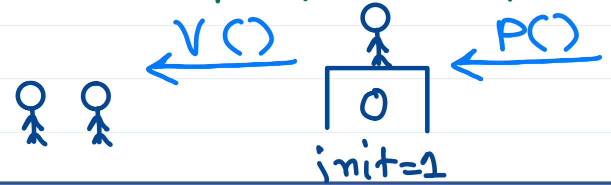
inc / dec
V / P
signal op / wait op



Semaphore types

① counting semaphore: resource/processes counting. init value = n.

② binary semaphore: mutual exclusion or event/condition. init value = 1/0.



Semaphore

① mutual exclusion

S=1

P1	P2
P(s);	P(s);
access res.	access res.
V(s);	V(s);

② Condition / event

S=0

```
P1            P2
—             —
—             —
P(s);         —
—             —
—             V(s);
—             —
```

③ Counting

S=n

P1	P2	...	Pn
P(S);	P(S);		P(S);
access res;	access res;	...	access res;
V(S);	V(S);		V(S);

* Producer Consumer Problem

① producer process - produce data.

② consumer process - consume data.

③ producer → circular buffer → consumer

④ only one process should read/write into circular buffer.

- mutual exclusion. → sm=1

⑤ if buffer is empty, block consumer.

- count filled slots → filled=0 → sf=0

⑥ if buffer is full, block producer.

- count empty slots → empty=0 → se=n

0. sm=1
1. sf=0
2. se=n

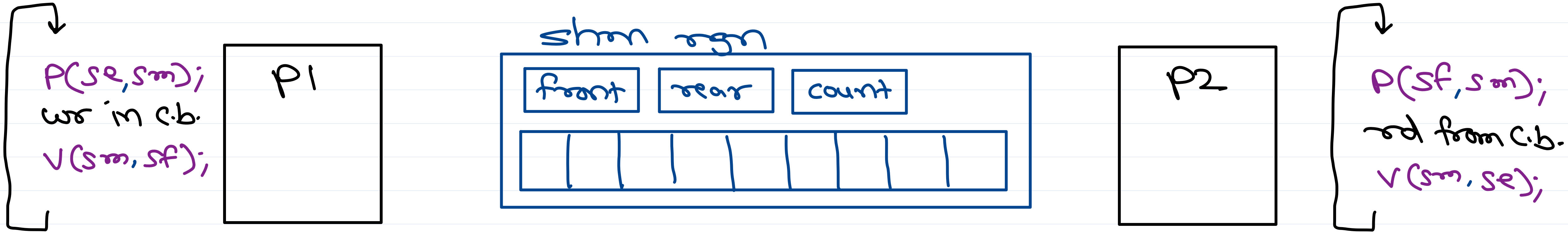
producer

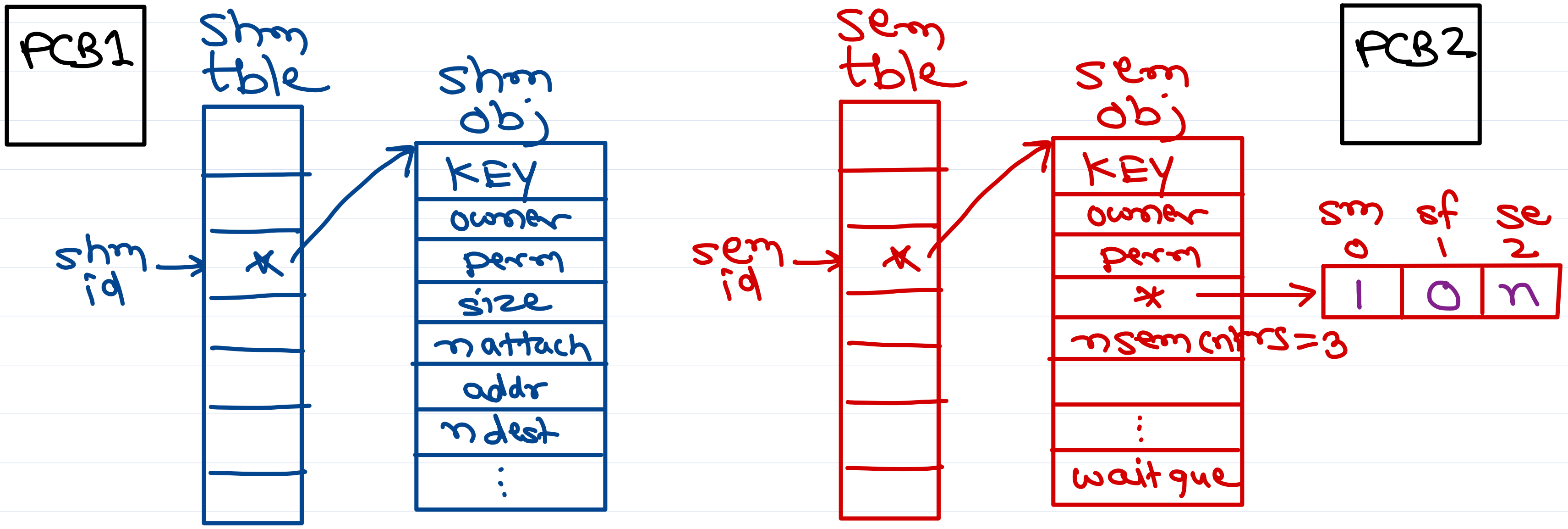
```
P(se);
P(sm);
wr in c.b.
V(sm);
V(sf);
```

consumer

```
P(sf);
P(sm);
rd from c.b.
V(sm);
V(se);
```

Producer Consumer Problem





Deadlock

P1

1. open the file.
2. connect printer.
3. print file.
4. disconnect printer.
5. close the file .

P2

1. connect printer.
2. open the file.
3. print file
4. close the file.
5. disconnect printer.

Deadlock characteristics

1. no preemption
2. mutual exclusion
3. hold & wait
4. circular wait

Deadlock avoidance

process should inform OS about resources required before actually allocating them.

OS maintains data of all resources & processes and take decision about allow/deny the resource.

Avoidance algorithms:

1. safe state
2. resource allocation graph.
3. bankers algorithm

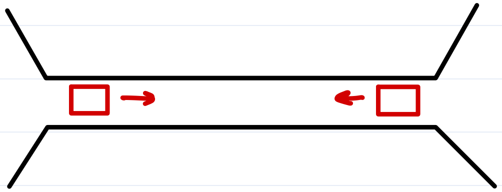
sf=1
sp=1

P1

```
P(sf);
P(sp);
print file;
V(sp);
V(sf);
```

P2

```
P(sp);
P(sf);
print file;
V(sf);
V(sp);
```



Deadlock prevention

✓ design system so that one of the deadlock condn never holds true - So deadlock is prevented.

sf=1;
sp=1;

P1

```
P(sf,sp);
print file;
V(sf,sp);
```

P2

```
P(sf,sp);
print file;
V(sf,sp);
```

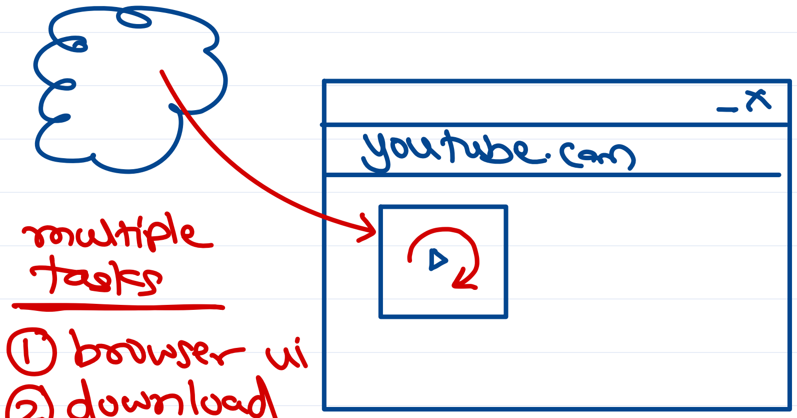
Sys V semaphore allow multiple ops to be done at same time. So hold & wait is never true.

deadlock detection

deadlock recovery

Processes involved in deadlock are permanently blocked i.e. in wait queue.

Multi threading



multiple tasks

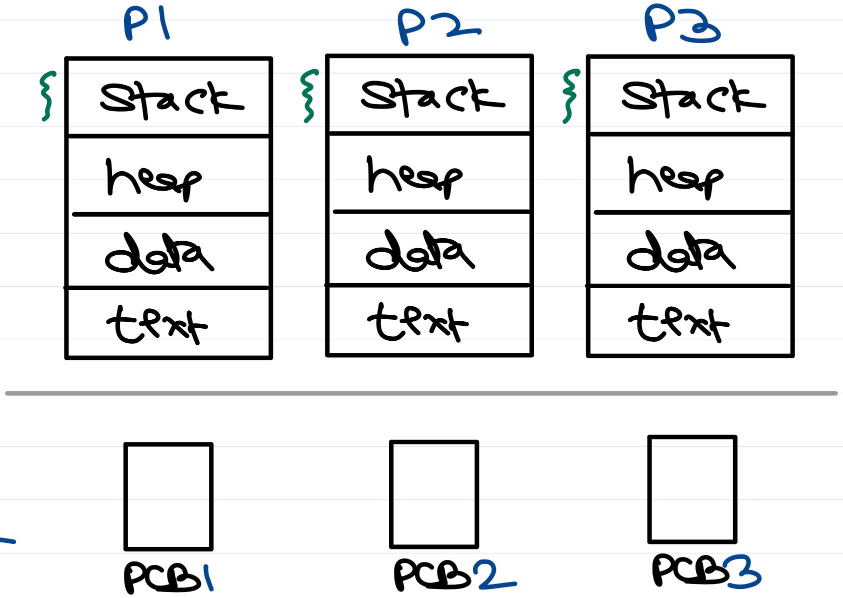
① browser ui
② download
③ play

In modern OS, process is like a container that holds resources required for execution; while thread is unit of execution/scheduling.

For each process one thread is created by default, called as main thread.

cmd> ps -e -m -o pid,tid,nlwp,cmd

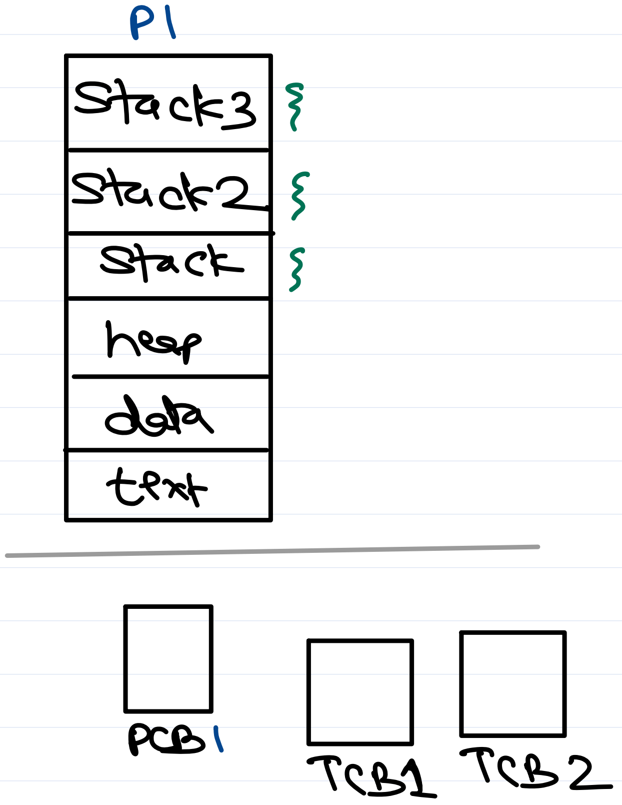
Process based multi-tasking



PCB: info about resources
e.g. pid, exit status, memory info, files info, ipc info (signal), ...

TCB: info about execution
e.g. tid, sched info (time, algo, priority), exec. ctx, kernel stack.

thread based multi-tasking



Thread is a light-weight process.

SUNBEAM

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>