Embedded Operating Systems

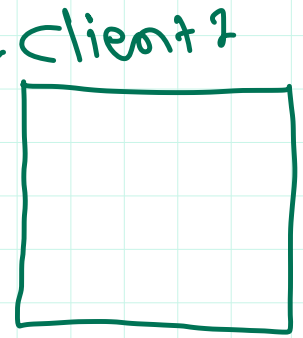
Trainer: Nilesh Ghule

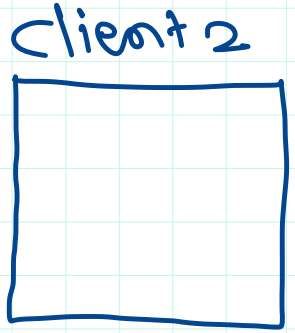
Sockets : Handling multiple clients

server

```
signal (SIGCHLD, sigchld_handler);
sfd = socket();
bind (sfd,...);
listen (sfd,...);
while(1) {
    cfd = accept (sfd,...);
    // accept() failed -> Interrupted SysCall.
    if(cfd<0) {
        if(errno== EINTR)
            continue;
    }
    if(cfd > 0) {
        pid = fork();
        if (pid==0) {
            handle_client(...);
            exit(0);
        }
    }
}

void sigchld_handler(...){
    wait(&s);
}
```

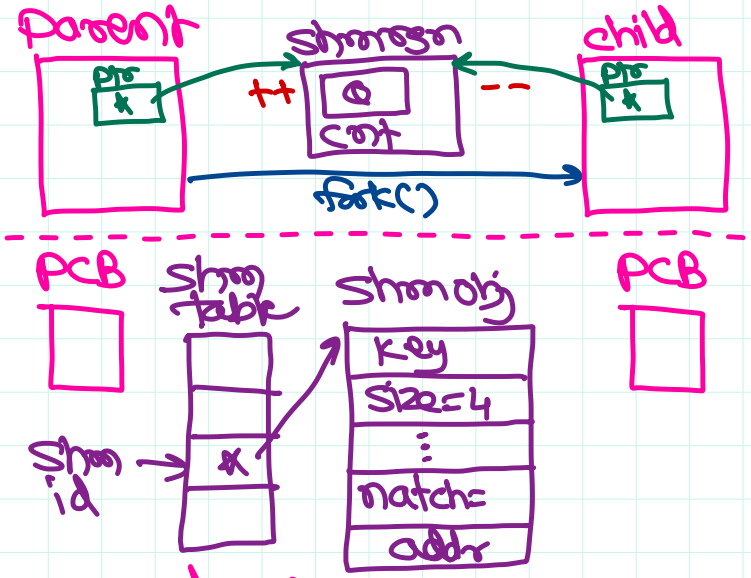




child 1 ×

child2()

```
handle_client(cfd,...) {
    read (cfd,...);
    write(cfd,...);
    close(cfd);
}
```



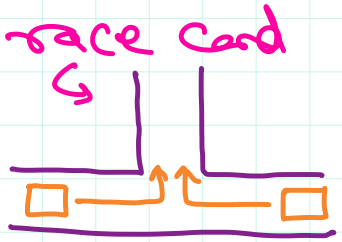
Synchronization: is a process of coordinating multiple concurrent processes to ensure that they can access shared resources without data inconsistency/corruption.

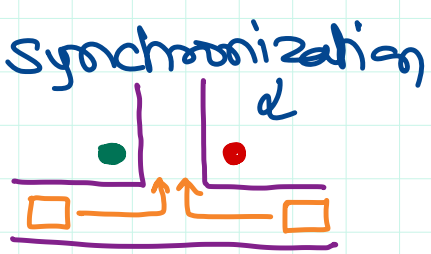
Usually this is done by blocking one/more processes while a process is using resource.

Sync is done using OS sync objs/primitives:

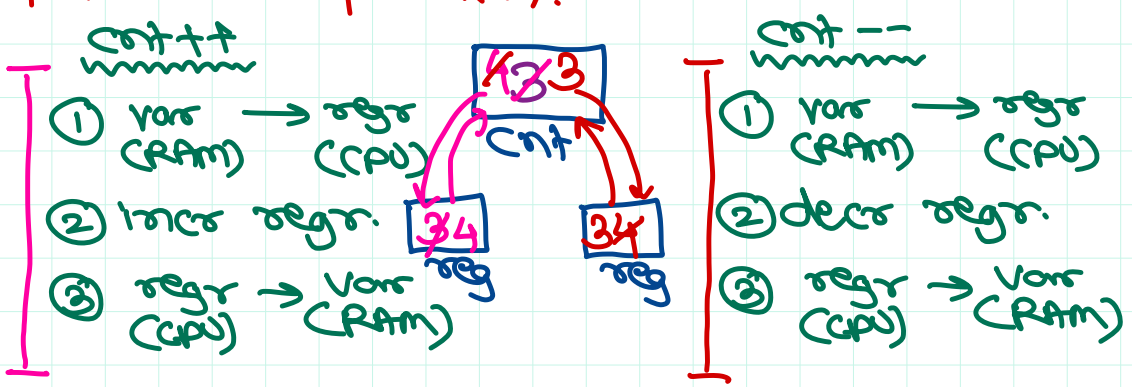
① Semaphore ② mutex ③ cond variable

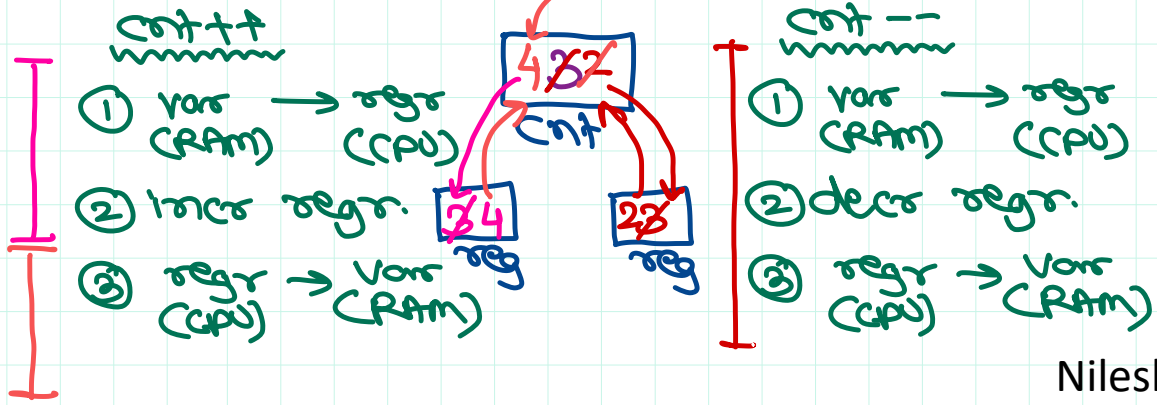
race cond: when a common resource accessed by multiple processes at the same time, it may yield unexpected result due to resource corruption.





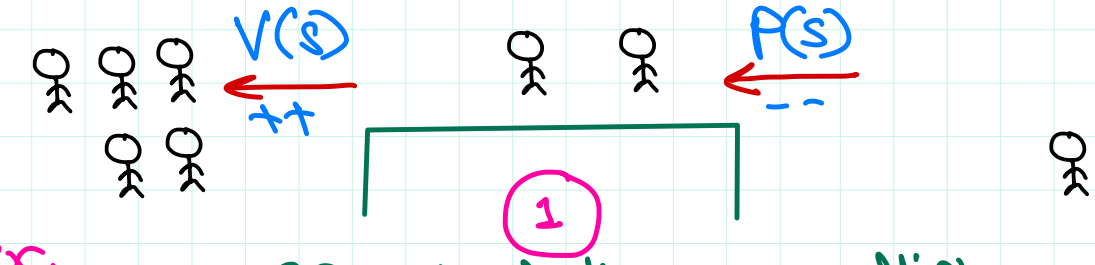
peterson's problem:





Semaphore

- sync object
- Dijkstra designed.
- sema is a counter.
  - decr op - wait op - P op.
    - decr cnt by 1.
    - if cnt < 0, block the current process.
  - incr op - signal op - V op
    - incr cnt by 1.
    - if one/more processes are blocked, one of the process is woken up.
- sema types
  - counting sema - count resource, availability, or processes.
  - binary sema - two states of sema (locked / unlocked)
    locked = 0, unlocked = 1



OS scheduling → multiple queues

1. job queue a.k.a. process table/list → all processes
2. ready queue a.k.a. run list → processes ready for execution on CPU.
3. wait queue(s) → per IO device, IPC mech, sync objs, ...

process block/sleep/wait:

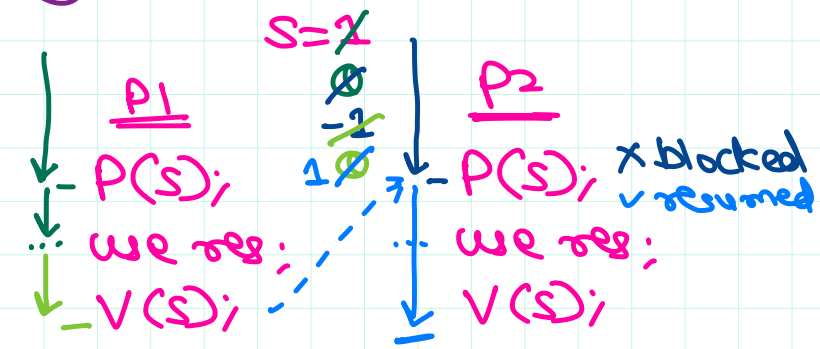
- PCB removed from ready queue
- pcb.state = waiting
- PCB is now added to wait queue.

process wake up

- PCB removed from wait queue.
- pcb.state = ready
- PCB is added to ready queue.

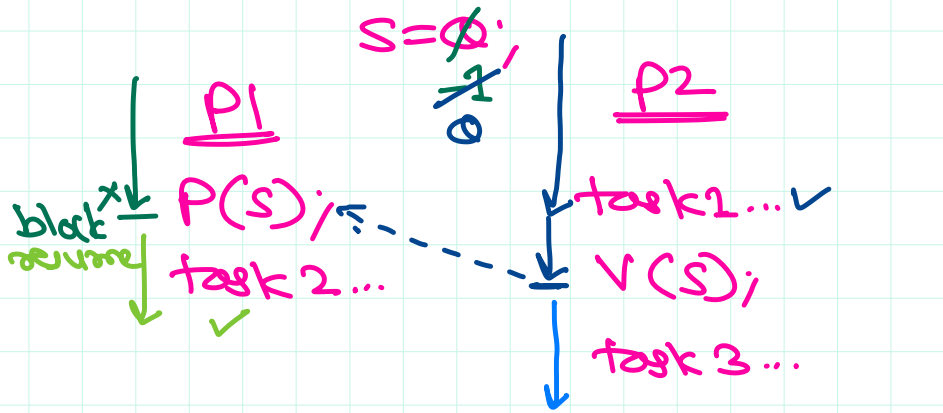
Semaphore Usage

① Mutual Exclusion



mutual exclusion → only one process can access resource at a time.

③ Flag/event



A process (P1) waits for another process (P2) to complete certain task (task1) before P1 continues its work (task2).

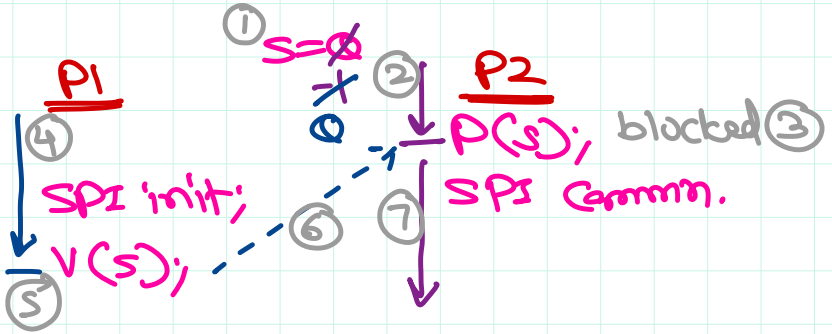
② Counting

S=n

P1	P2	...	Pn
P(S);	P(S);		P(S);
use res;	use res;		use re
V(S);	V(S);		V(S);

task 1 (P1) → SPI init

task 2 (P2) → SPI send/recv



resource → uart

tasks (P1, P2, ...) → use uart (tx)

Px S=1

P(S);

tx on uart;

V(S);

5 uarts to share betn 20 tasks.

array

U0	U1	U2	U3	U4
0/1	0/1	0/1	0/1	0/1

uarts.

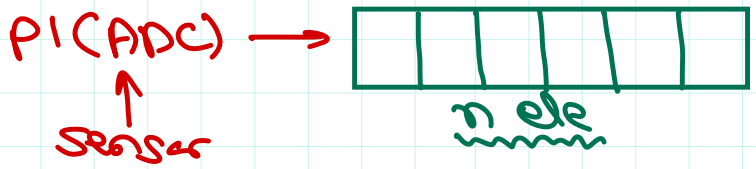
inuse

sem_avail = 5

sem_mut = 1

ADC task (P1) → read ADC sensor → producer

UART task (P2) → send ADC read on UART → consumer.



1. only producer/consumer should access buffer at a time.
2. if buffer is full, producer should wait.
3. if buffer is empty, consumer should wait.

emp = n sidx = 0

mut = 1 sidx = 1

fil = 0 sidx = 2

Producer

```
P(emp);
P(mut);
~ get data
~ write in buf
V(mut);
V(fil);
```

Consumer

```
P(fil);
P(mut);
~ read the buf
~ use data
V(mut);
V(emp);
```

```
int acquire_uart() {
P(sem_avail);
P(sem_mut);
for(i=0; i<5; i++){
  if(inuse[i]==0)
    inuse[i]=1;
    V(sem_mut);
    return i;
}
}
```

```
void release_uart(i){
inuse[i]=0;
V(sem_avail);
}
```

task

```
{ i = acquire_uart();
  use uart[i];
  release_uart(i);
```

UNIX Semaphore (Sys V)

One sema obj may have one/ more sema counters.

① Create a sema

```
semid = semget(key, nsems, flag);
```

key: 1234, nsems: ③, flag: IPC_CREAT

② init sema

```
unsigned short initvals[] = {0, 1, 0};
union semun su;
su.array = initvals;
semctl(semid, 0, SETALL, su);
```

initvals: {wait cnt, mutex, rel cnt}

③ destroy sema

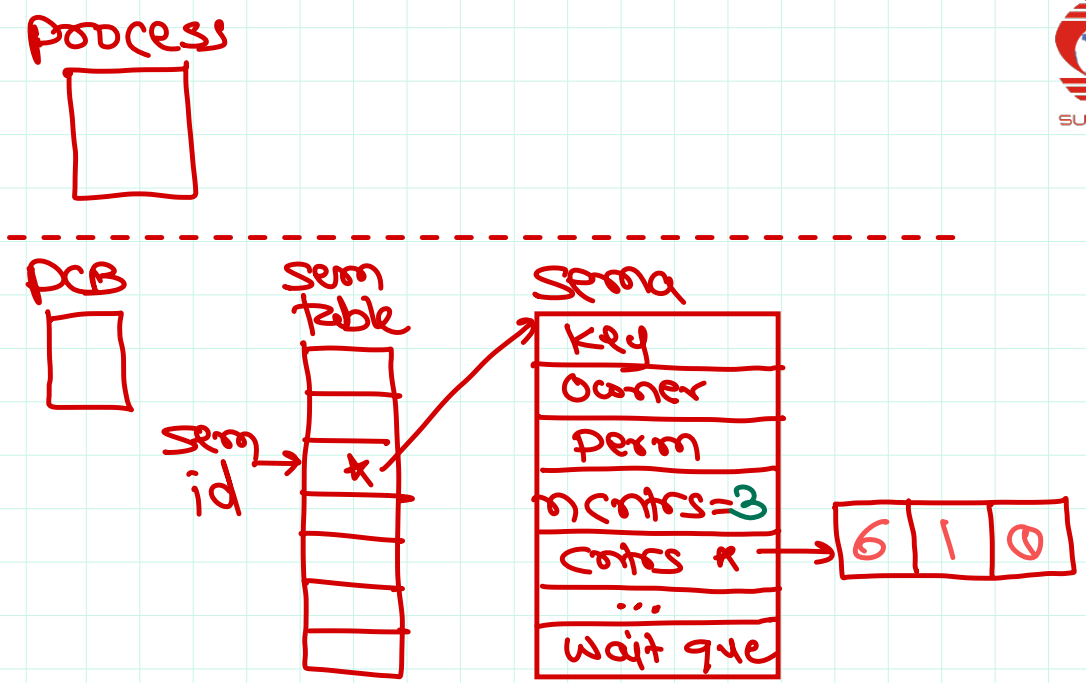
```
semctl(semid, 0, IPC_RMID);
```

④ sema op - P or V

sembuf struct
- num = sem idx
- op = +1 or -1 ; (+1 → V, -1 → P)
- flg = 0

```
struct sembuf ops[2];
ops[0].sem_num = 0;
ops[0].sem_op = -1;
ops[0].sem_flg = 0;
ops[1].sem_num = 1;
ops[1].sem_op = +1;
ops[1].sem_flg = 0;
semop(semid, ops, 2);
```

P(S[0]) V(S[1])



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>