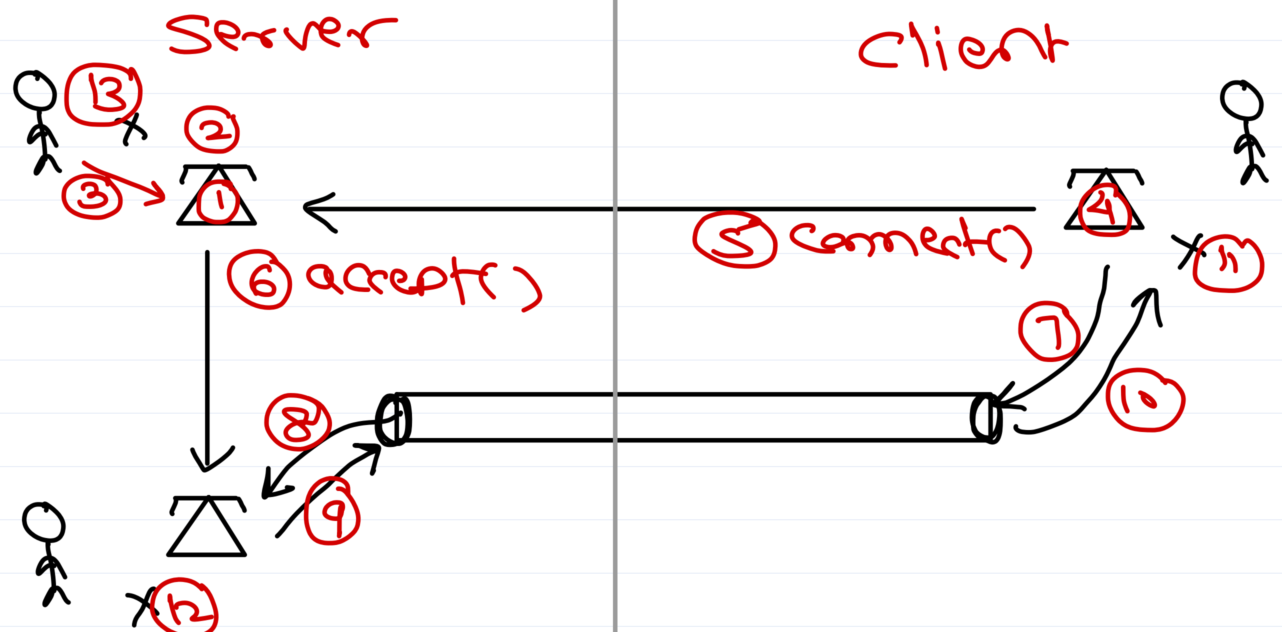
SUNBEAM

Embedded Operating Systems

Trainer: Nilesh Ghule

Sockets



1 socket()
2 bind()
3 listen()
6 accept()
8 read()
9 write()
12 close()
13 shutdown()

4 socket()
5 connect()
7 write()
10 read()
11 close()

UNIX (BSD) Socket Programming

SERVER

Create server socket. 1

```
serv_fd = socket(AF_INET, SOCK_STREAM, 0);
```

Assign socket address (IP addr + port no). 2

~~serv_fd~~ ret
```
= bind(serv_fd, &serv_addr, sizeof(serv_addr));
```

Listen to the server socket. 3

```
listen(serv_fd, 5);
```

Accept client connection (blocked until client connects).

```
cli_fd = accept(serv_fd, &cli_addr, sizeof(cli_addr));
```
& 6

Read data from client.

```
read(cli_fd, buf, size);
```
8

Send data to client.

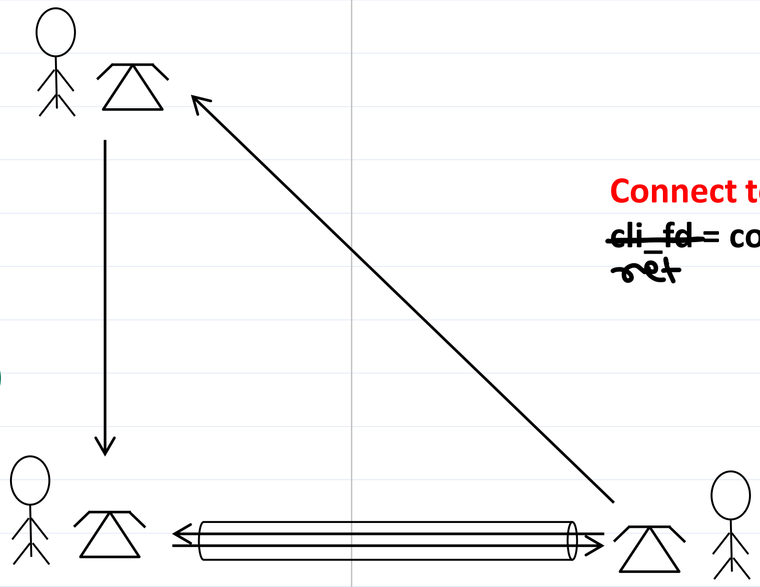
```
write(cli_fd, buf, size);
```
9

Close client socket.

```
close(cli_fd);
```
12

Shutdown the server socket.

```
shutdown(serv_fd, SHUT_RDWR);
```
13



CLIENT

4 Create client socket.

```
cli_fd = socket(AF_INET, SOCK_STREAM, 0);
```

Connect to the server. 5

~~cli_fd~~ ret = connect(~~serv_fd~~ cli_fd, &serv_addr, sizeof(serv_addr));

Send data to server.

7
```
write(cli_fd, buf, size);
```

Read data from server.

10
```
read(cli_fd, buf, size);
```

Close client socket.

11
```
close(cli_fd);
```

Multiple clients

Server

```
sfd = socket(—);
bind(sfd, —, —);
listen(sfd, 5);
while(1) {
    cfd = accept(sfd, —, —);
    ret = ~~fork();~~ new thread;
    if(ret == 0) {
        read(cfd, —, —);
        for each cfd in list:
        write(cfd, —, —);
    }
    else
        add cfd in linked list.

    close(—);
}
```

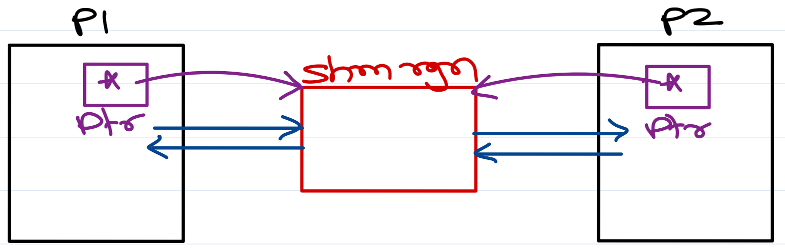
client

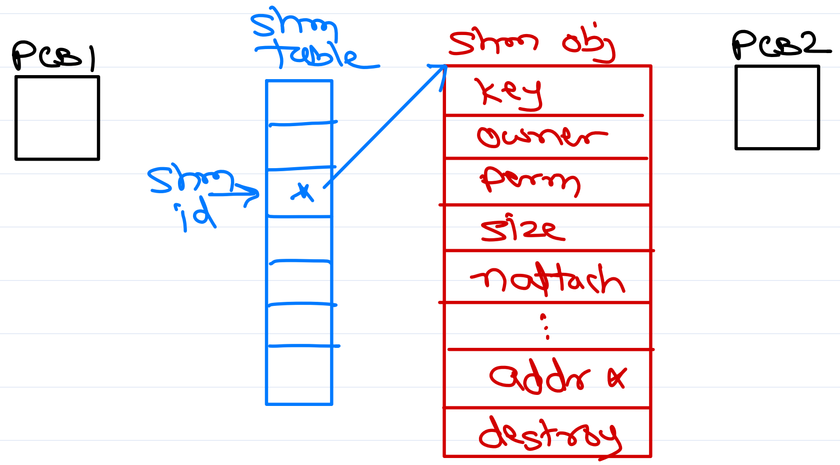
```
cfd = socket(—);
connect(cfd, —, —);
new thread
    ↳ read(cfd, —, —);
        print(—)

scanf(—);
write(—);

close(c)
```

Shared memory





① shmid = shmget(key, size, flags);
ⓐ creates shm obj and init it with given values / default values.
ⓑ store its addr into shm table & return its index i.e. shmid.

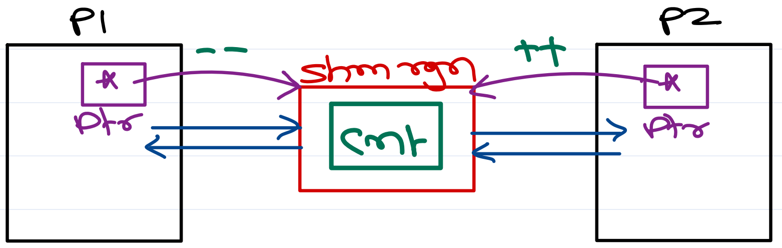
② shmctl(shmid, IPC_RMID, NULL);
ⓐ marks given shm obj for destruction.
ⓑ if no process is attached to shared memory (i.e. nattach = 0), then shm is deleted.

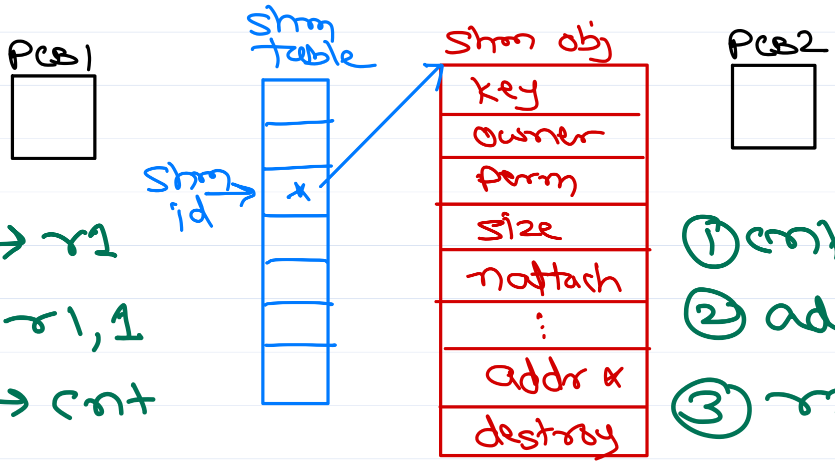
③ shmctl(shmid, IPC_STAT, &shmid_ds);
ⓐ get info about shm in out param.

④ ptr = shmat(shmid, NULL, 0);
ⓐ attach shm rgn to VAS of current process i.e. create VAD and page table entries.
ⓑ increment nattach field in shm obj
ⓒ return pointer (virtual addr) to shm rgn.
arg2 - which virtual addr of process to be mapped with shm rgn. NULL: take default.

⑤ shmdt(ptr);
ⓐ detach shm rgn from the process. i.e. delete VAD and page table entries.
ⓑ decrement nattach field in shm obj
ⓒ if nattach field is zero and shm is marked for destruction, then destroy the shm rgn.

Synchronization — Semaphore

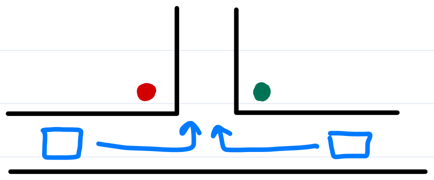




① cnt → r1
② sub r1, 1
③ r1 → cnt

① cnt → r2
② add r2, 1
③ r2 → cnt

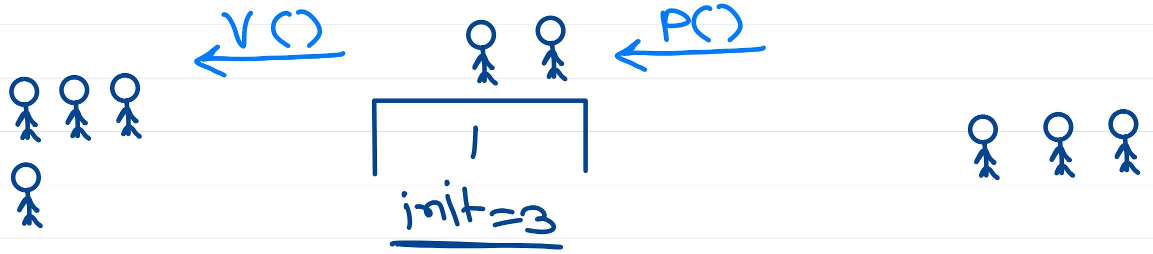
race condn: multiple process accessing same resource at the same time.



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

Semaphore → sync primitive

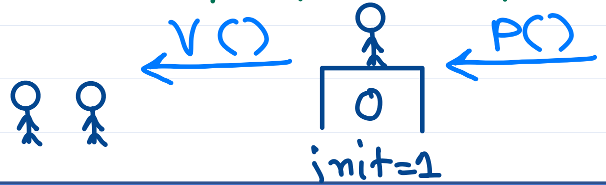
- is a counter → operations: inc / dec
  - V — signal op
  - P — wait op
- dec op: decr count by 1.
  - if count < 0, the block curr. process
- inc op: incr count by 1.
  - if one/processes are blocked, wake up one of them.



Semaphore types

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

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



SUNBEAM

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>