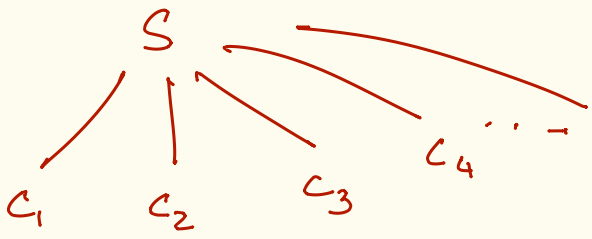
# CS744 LECTURE 15

* Recap:
  - data is exchanged by processes in units of packets.
    packet : headers + data / payload.
    └ protocol specific information.
  - communicating processes have unique addresses / endpoints
    - IP address, port number, MAC-address.
    - every packet on the network has this information.
  - multiple protocols exist for communication.
    - tcp, ip, udp, ethernet, 802.11, POP, IMAP, http, telnet, ftp, sctp, rtp, ....
    - protocol processing / network stack part of OS
  - the socket abstraction allows processes to read / write data / payload after setup & config.
  - connection-oriented & connection-less sockets.
    - recv, send
      read, write
      - <receiver ip, sender ip, receiver port, sender port>
        4-tuple identifies connection (socket at each end).
    - recvfrom, sendto
      - receiver ip, receiver port identifies socket.

* socket, bind, listen, connect, accept
  recv, send, recvfrom, sendto, read, write, close

# read & write or recv/send are blocking calls.




server

```
socket()
bind()
listen()
while (1) {
    newfd = accept();
    read(newfd, ....);
    write(newfd, ....);
}
```

blocking call

(i) scale to multiple connections is a multi-process server

```
socket, bind, listen
sockfd
while (1) {
    newfd = accept(sockfd, ...);
    if (fork() == 0) {          // child
        close(sockfd);
        // handle connection, read & write using newfd
        exit();
    }
    close(newfd);
}
```

(ii) improve efficiency using a single CPU.

- event driven IO model.
- select, poll, epoll

* usage of select.

⊛ select is used to monitor a set of file descriptors to check if they are ready for read, write, error logging etc.

```
fd_set  conn_fdset;
while (1) {
    conn_fdset.clear();        // reset fds.
    conn_fdset.add( );         // socket fd
                               // socket fds to monitor
    ready_cnt = select(MAXFDS, conn_fdset, ....);
                               // socket accept
    for (i=0; i< MAXFDS, i++) {
        if ( conn_fdset[i].status == READY) {
            // do work for the socket descriptor
            // socket();
            // accept or read/write
        }
    }
} // end of while
```

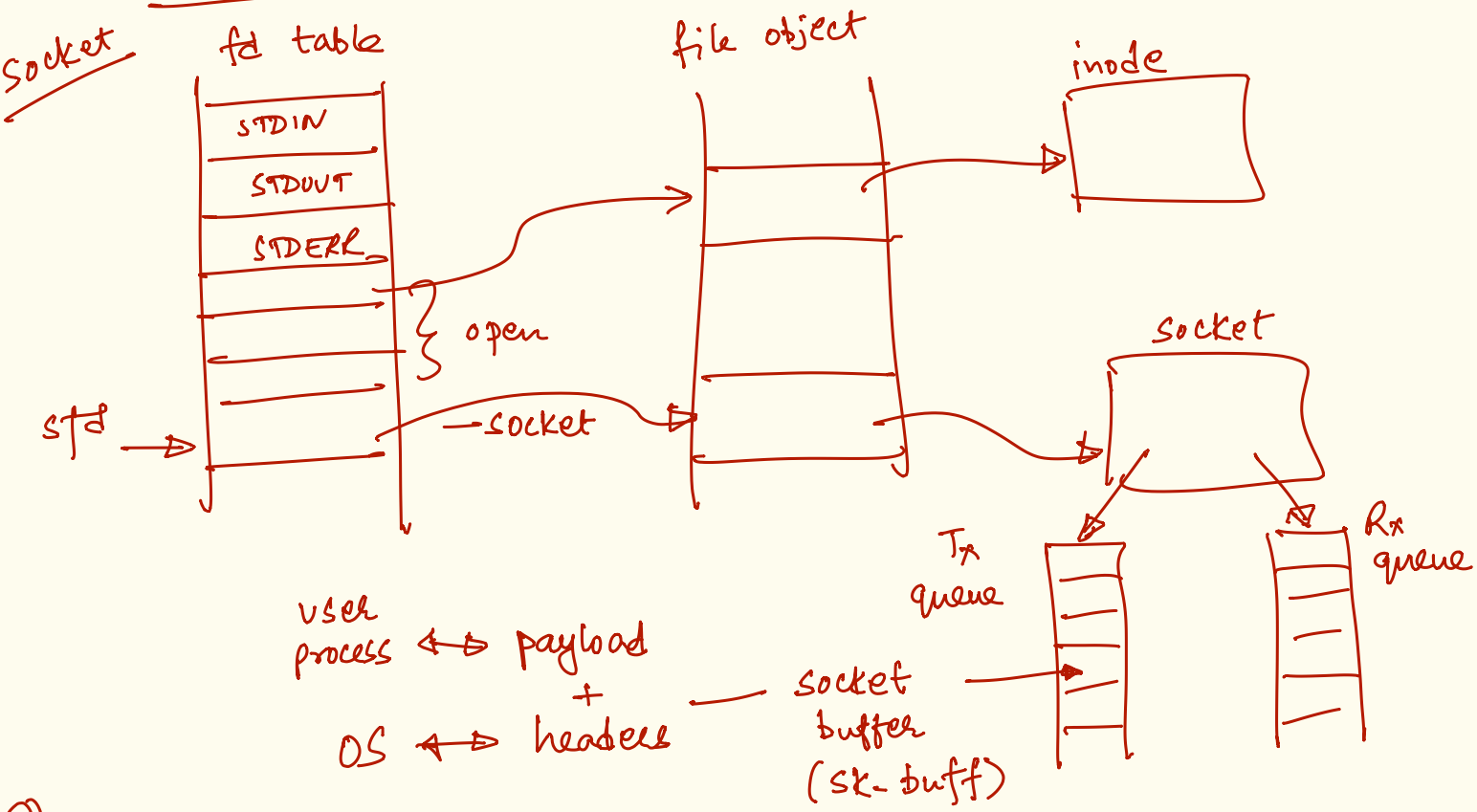
add a set/number of fds. | loop

poll - very similar to select & also sets status to FALSE if fd not ready.

⊛ epoll → returns only the set of ready fds.

```
epfd = epoll_create();
epoll_add(epfd, client_fd);        } loop for set of fds.
while (1) {
    ready_cnt = epoll_wait(epfd, events, ...);
    for (i=0; i< ready_cnt, i++); {
        client.sfd = events[i].fd;     // do work on fd
```

# ☆ OS view of the network IO sequence.



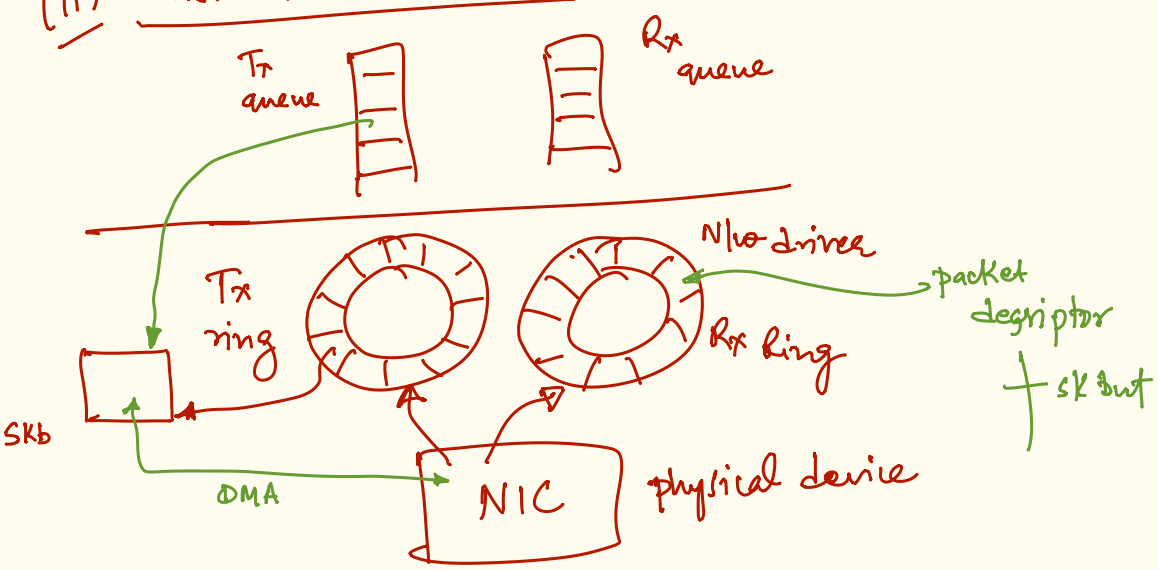

## ⊛ Read/Recv via a sockfd

- ~ Dequeue skb for Rx queue
- ~ copy payload of skb to user-space VA.
- ~ skb freed (can be reused)
- ~ pkt on n/w → OS processes pkt headers → adds skb to Rx queue of appropriate socket.

## ⊛ Send/Tx

- ~ new skb is created/added to Tx queue.
- ~ copy payload to skb from user space VA
- ~ OS processing adds headers & updates skb.
- ~ Device driver uses skb to send pkt. on the NIC.
- ~ Tx done, skb is freed.

(ii) network driver




Receive / Read.

~ driver initializes Rx Ring w/ free/available packet decriptors.

— on pkt arrival, NIC finds free pkt. desc, DMA's pkt. to location/memory

~ device generates interrupt

— OS handles interrupt.

└ network (protocol) processing + socket abstraction implementation.

~ add/replace pkt. decriptor in Rx Ring.