X IPC in System V (Unix) —:

- Three mechanisms : messages (allowing sending arbitrary data streams) shared memory (share virt. addr. spc.) & semaphores (synchronizat
- Mechanisms contain tables whose entries describe instances.
- Each entry contains numeric key to uniquely identify entry.
- A get syscall for every mechanism to create new entry or retrieve existing one (params : key, flags) :
  - Search for table & entry named by key.
  - IPC_PRIVATE flag assures return of unused entry.
  - IPC_CREAT flag to create new entry if entry not exists.
  - IPC_EXCL to raise error on existent key w/ IPC_CREAT.
- For IPC mechanisms, index of entry into table of data structures :
  index = descriptor % (# entries in table).
- Descriptors not reused until certain conditions to prevent multiple consumers accessing data at same location.
- IPC entries also consist of permissions structures :
  - User ID & Group ID of process which created the entry.
  - UID, GID of entry (set using control syscall).
  - R, W & X perms for user, group & others.
- Entries also consist of status information :
  - PID of last process which updated entry.
  - Timestamp of last access or update.
  - Info recorded for actions : send, recv., attach, etc.
- A control syscall aids in :
  - Querying status entry (& copy entry data to usr spc.)
  - Set status info.
  - Remove entry from system.

1) Query : Check read permission, copy if allowed else raise error.
2) Set Params : Checks UID of process is of SU or has UID of creator of entry. If yes, copies user data to entry, & set fields.

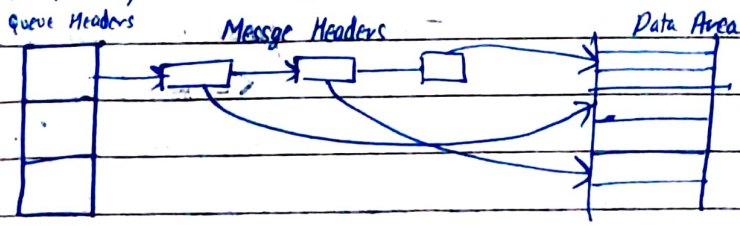
3) Remove : Only allowed to su or creator. Descriptor # incremented to ensure next entry assignment returns different descriptor.

× System Calls for Messages — : <sys/ipc.h>, <sys/msg.h>

- msgget : Returns / creates (if flags set) descriptor for msg queue.
- msgctl : set / return params assoc. w/ descriptors & rm descriptors
- msgsnd : Send message
- msgrcv : receive message

⊕ msgget : <msgqid> = msgget (<key>, <flags>);
- msgqid : descriptor of queue containing shared messages
- key, flags : get syscall params
- Messages stored on linked list (queue), unique for each descriptor, msgqid helps index from array of message queue headers.

× Queue Headers contain information on :
- Ptr to linked list (of msg queue). (first & last msg ptrs)
- Size of msg queue (# of msgs, total # of bytes)
- Max size (Capacity) of msg queue (max # of bytes)
- PID of last process to send & receive
- timestamps of last msgsnd, msgrcv & msgctl operations.



⊕ msgsnd : msgsnd (<msgqid>, <msg>, <size>, <flag>);
- msgqid : descriptor
- msg : struct object containing msg & type.
- size : sizeof msg (content of msg, not struct)
- flag : control wait behavior if no space.

0 : wait until space available
IPC_NOWAIT : exit if no space. (no wait incurred)

- Checks : write permissions, avl. memory, space in queue & +ve msg type value.
- Actions : Allocs space for msg., copy data from usr spc, alloc msg header & add to end of linked list.
: Records message type & size in msg header.
: Link message header w/ data & update stat fields in queue header
: Awaken processes waiting for message or specific message of type.
: Process sleeps if no space on queue (if flag = 0)

⊕ msgrcv : <count> = msgrcv (<id>, <msg>, <maxct>, <type>, <flag>);
id : msg queue descriptor
msg : user struct to hold message
maxct : size of data array in message
type : type of msg to read (0 = any message).
flag : controls if process should wait if no msgs on queue.
(IPC_NOWAIT : exit on no message)
type value = +ve : msg of specific type (first such msg.)
0 : first msg (any type) on queue
-ve : first msg on queue s.t. type of msg <= abs(type param) ~~and return closest such msg (type closest to value)~~.

- Actions : If avl. msg, adjust msg size or return error if user struct size is too small. Copy message content from kernel space to user space, update stat info. & unlink from message queue.
- MSG_NOERROR flag : exit on no message, but return no error. (Useful for size constraint errors).
(msg truncated as per given size & removed from queue)

⊕ msgctl : msgctl (<id>, <cmd>, <mstatbuf>),
- id : descriptor of msg queue
- cmd : type of command / action
- mstatbuf : addr. of struct object to hold data for modification or retrieval. (control params)

Commands :
IPC_RMID : remove queue associated w/ msgid.

× IPC Using Shared Memory ←:

- Processes communicate by sharing parts of vmt. addr. spc. & performing I/O in the shared memory (for data content).
- Syscalls : <sys/shm.h>
  - shmget, : Create new region of shared memory or return existing
  - shmat, : Attaches region to virtual space.
  - shmdt, : Detaches region from virtual space.
  - shmctl : Manipulates params associated w/ shared memory.

⊕ shmget : <shmid> = shmget (<key>, <size>, <flag>),
- size : # bytes in shared region
- key : shared memory descriptor key
- flag : control flags (IPC_CREAT, etc).

Actions : Search shared memory table for key, if entry found & permissions match returns descriptor, else on non-existent region w/ IPC_CREAT, creates new region using allocreg
: Memory not allocated until process attaches the region to addr. spc.
: Permissions, size & ptr to region table entry saved in shared memory table entry & flag set to indicate no associated memory.
: Flag set to indicate non-freeing of region when last attached process exits set in region table entry.

⊕ shmat: (<id>, <addr shmat (<id>, <addr>, <flags>);
- id : shared memory descriptor
- addr : virt. addr. to attach shared memory at
- flags : options for region (read-only, round off addr, etc)
- virt addr : Actual addr at which shared region was attached.

Actions : Check validity of descriptor & permissions. As per flags, round off addr if requested & check if addr is valid if specified, otherwise select any virt_addr. Next attach region to addr. spc. (attachreg). If region attached for first time, allocate page tables (growreg) & return virt addr.

- Shared memory placed before beginning of stack region.
(data region not viable as may be resized via brk)
(end of stack region not viable as stack may grow)

⊕ shmdt : shmdt (<addr>);
- addr : addr of attached region.

Actions : search for process region attached at addr, & detach it from addr. spc. (detach reg)
Region tables do not point to shared memory table, so kernel searches for entry of region in shared mem tbl. & adjusts time

⊕ shmctl : shmctl (<id>, <cmd>, <shmstatbuf>);
- id : shared memory descriptor
- cmd : command, type of operation (remove, query, etc)
- shmstatbuf : control parameters / memory to hold results

Actions :
- Remove : frees entry & refers region table entry :
  - If no process has region attached, free region entry and all resources using freereg.
  - If region attached (ref. count > 0), flag ~~detroyed~~ to prevent freeing when last process detaches, cleared.

× IPC - Semaphores —:

× Locking via lockfile : process creates lock file via creat to lock resource. creat fails if file exists, so process may assume another process has resource locked.
- Disadvantages :
  - Process not aware when resource freed.
  - Lockfiles remain upon system crash, rendering resource locked.

× Solution : Dekker's Algorithm (& Dijkstra's) :
- Integer valued objects (counting semaphore)
  - +ve : resources avl.
  - -ve : # processes waiting.
- Atomic operations : (Only one op. succeeds on semaphore at a time):
  - P : decrements value if > 0.
  - V : increments value

× Semaphores in System V : generalization of dijkstra's operations.
- Consist of : Value (Current res. avl.)
  - : PID of process (most recent) which modified semaphore.
  - : # proc. waiting for semaphore to increase
  - : # proc. waiting for semaphore to have value 0.
- Syscalls : <sys/sem.h>
  - semget : Create / gain access to semaphore
  - semctl : Perform control operations on semaphore
  - semop : Manipulate value of semaphore

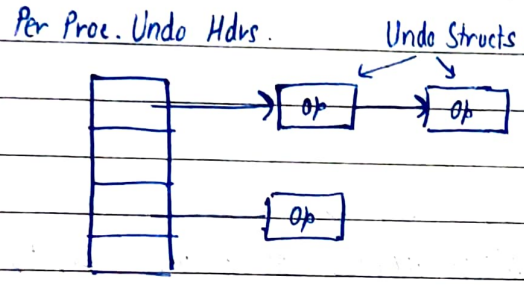
⊕ semget : <semid> = semget (<key>, <nsems>, <flag>);
- key : semaphore key for descriptor (IPC_PRIVATE : create new)
- nsems : # semaphores to create (0 if existing is used)
- flag : permissions, creation flags (IPC_CREAT), etc.

- nsems > 0 imply creation of array of semaphores pointed by entry.

⊕ semop : semop (<semid>, <sembufs>[], <nops>);
- sembufs (struct sembuf *) :
pointer to array of semaphore operations (size = nops)
struct sembuf {
ushort sem_num ; // index of semaphore array to change
short sem_op ;
short sem_flg ; // IPC_NOWAIT, SEM_UNDO, etc.
}
sem_op > 0 : add value to semaphore (unlock)
= 0 : wait until semval == 0 (all resources in use)
< 0 : subtract value from semaphore (lock)
- IPC_NOWAIT : no wait, return -1
- else : wait until semval >= |sem_op|
× SEM_UNDO : revert operations by preserving undo information
(undo stack) (for use on process termination & crash)

⊕ semctl semctl (<id>, <semnum>, <cmd>, <arg>);
- id : semaphore descriptor
- semnum : index of semaphore array to operate on / # of semaphores
- cmd : operation :
- GETALL : Get values of semaphores into arg.
- SETALL : Set values of semaphores from arg.
- IPC_RMID : remove semaphore w/ given id.
- arg : union type to hold or provide data for syscall
: union semun {
int val ; // SETVAL
struct semid_ds * buf // IPC_STAT, IPC_SET
ushort * array // SETALL, GETALL
}

x SEM_UNDO : upon process exit, revert all operations performed by process.
- Table of operations per process for all processes maintained in an undo structure.



Undo Structure

op = Contains:
- Semaphore index (of array)
- Semaphore descriptor
- Adjustment Value

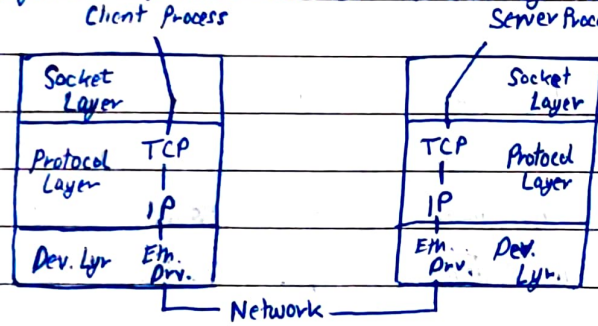
- If undo entry found in undo structure, adjustment value adjusted (on adjust value = 0, remove entry
- Else, for decrement operations, possibly undo entry non-existent, so create new entry w/ opp. (negated) value for adjustment.

X Network Communication using IPC :-

x Kernel communication structure :

- Socket Layer (IP + Port = Socket)
- Protocol Layer
- Device Layer

- Socket Layer : Interface b/w syscalls & lower layer
- Protocol Layer : Protocol modules for comm. (TCP, IP)
- Device Layer : Device drivers controlling network devices.



x TCP / UDP :

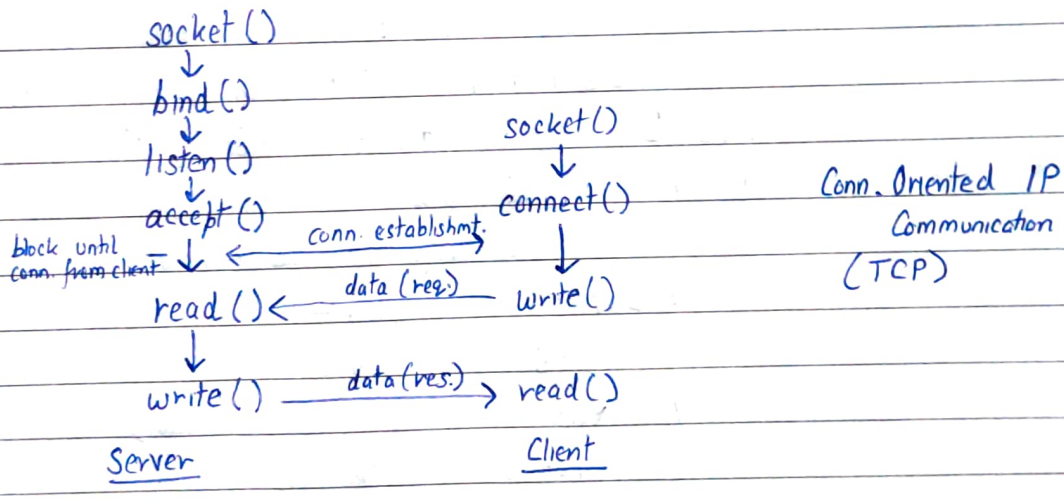
- End to end transport
- TCP : Connection Oriented, reliable byte stream (virtual circuit)
- UDP : Unsequenced, connectionless, unreliable data stream, less expensive to setup (datagram)

x Socket

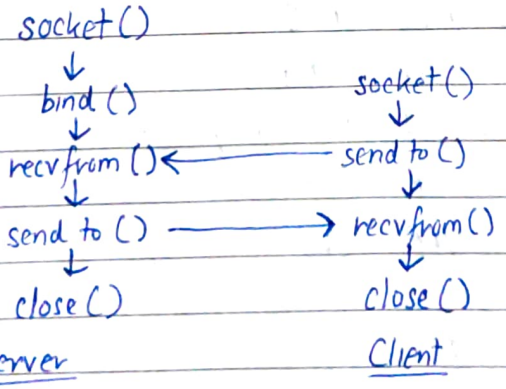
- Abstraction of communication endpoint.
- Internal connections & routes maintained by kernel.
- Communication via client-server model :
  - Server listens to socket (one end of duplex communication)
  - Client communicates to server over another socket (other end of duplex communication)

- Types :
  - Stream Sockets / Virtual Circuit (TCP) (SOCK_STREAM)
  - Datagram Sockets (UDP) (SOCK_DGRAM)
    - Best effort datagram service, messages upto 65,500 B.
- Extend conventional UNIX I/O facilities :
  - File descriptors for network comm.
  - Extension of read & write syscalls.

Client - Server Architecture :



Conn. Oriented IP Communication (TCP)



Conn. less IP Communication (UDP)

bind : Associates name w/ socket.
socket : Creates new socket.

× Syscalls

- socket : int socket(int domain, int type, int protocol);
  <sys/socket.h> [Creates a new socket for communication]

  Returns : socket fd, -1 on error

  Args :

  type : type of socket to determine comm. characteristics.
  - SOCK_DGRAM : UDP (fix len, conn. less)
  - SOCK_RAW : Datagram interface to IP
  - SOCK_SEQPACKETS : fix len, seq. reliable packets.
  - SOCK_STREAM : TCP (conn. oriented, reliable byte stream)

  protocol : 0 for default protocol, otherwise specifies protocol.

  domain : determines nature of communication, including address format.
  - AF_INET : IPv4
  - AF_INET6 : IPv6
  - AF_UNIX : Unix domain (Local)
  - AF_UNSPEC : Unspecified

  Protocols :
  - IPPROTO_IP : IPv4
  - IPPROTO_IPV6 : IPv6
  - IPPROTO_ICMP : ICMP
  - IPPROTO_RAW : Raw IP packets
  - IPPROTO_TCP : TCP
  - IPPROTO_UDP : UDP

× IPv4 AF_INET sockets :

```
struct sockaddr_in {
    short sin_family;   // AF_INET, etc.
    unsigned short sin_port;
    struct in_addr sin_addr;
```

```
    char sin_zero[8];
};
struct in_addr {
    unsigned long s_addr;
};
struct sockaddr {
    unsigned short sa_family;    // AF_INET, etc.
    char sa_data[14];            // 14B protocol addr.
};
```

⊕ bind : int bind (int sockfd, struct sockaddr* addr, int addrlen);
<sys/socket.h> [Associates address with socket]
Args : sockfd : Socket FD from socket (2).
addr : ptr. to struct with info. about addr : port & IP.
port must be ≥ 1024. (0-1023 = reserved)
addrlen : sizeof(*addr), size of address in bytes.

⊕ listen : int listen (int sockfd, int backlog);
<sys/socket.h> [Announces willingness to accept connection requests]
Args : sockfd : Socket FD
backlog : # of connections allowed on incoming queue.
- Backlog represents # of outstanding connect requests to enqueue on behalf of process (Once queue is full, future requests are rejected)

⊕ accept : int accept (int sockfd, struct sockaddr* addr, socklen_t* addrlen),
<sys/socket.h> [Accept a connecting socket]
Args : sockfd : listening socket descriptor
addr : (Optional, Nullable) address of connecting client
addrlen (Optional, Nullable) size of addr struct given
- Blocks until a connection request arrives or is not-pending.
Returns : 0 success
-1 : error

⊕ close : int close (int sockfd);
<sys/socket.h> [Closes conn to socket descriptor]

⊕ connect : int connect (int sockfd, struct sockaddr * addr, int addrlen);
<sys/socket.h> [In conn oriented service, connect to a socket]
Args : sockfd : client socket fd
addr : server addr details
addrlen : size of server addr struct in bytes.
- Used for establishing connection b/w client & server sockets.

⊕ shutdown : controls over socket connection close.
int shutdown (int sockfd, int how);
<sys/socket.h>
Args : how : SHUT_RD : disable read from socket.
SHUT_WR : disable write to socket.
SHUT_RDWR : disable I/O.

(TCP) ⊕ send : ssize_t send (int sockfd, const void * buf, size_t nbytes, int flags);
Returns : 0 if success (msg sent), -1 on failure.
Args : flags : Controls behavior of send call.
MSG_DONTROUTE : Prevent routing outside local network.
MSG_DONTWAIT : Enable nonblock operation.
- Used for TCP comm. (sockfd on server known).

(TCP/UDP) ⊕ sendto : ssize_t sendto (int sockfd, const void * buf, size_t nbytes, int flags, struct sockaddr * destaddr, socklen_t destlen);
- Similar to send (2), but allows specifying a dest. addr. for connectionless sockets.
- For conn-oriented sockets, dest is ignored.