Message queues

→ MQs can be used to pass messages b/w related or unrelated processes executing on same machine. Message queues are somewhat like pipes & fifos but different in important aspects:

(1) message boundaries are preserved so that readers and writers communicate in units of messages.

~~message queues are kernel persistent.~~

→ A MQ can be thought of as a linked list of messages in the kernel space. Process with adequate permission can put messages onto the queue & process with adequate permission can remove messages from the queue.

→ In pipes, a process read or write streams of bytes, while in MQ a process read or write a complete delimited message. It is not possible to read a partial message leaving rest behind in IPC object.

→ In pipe, a write makes no sense unless a reader also exists. In MQ, there is no requirement that some reader must be waiting before a process writes a message to the queue.

→ MQs are priority driven. Queue always remains sorted with the oldest message of the highest priority at front.

→ A process :

Creating or opening a message queue
int msgget (key_t key, int msgflag)

→ to create a brand new MQ. Or to get the identifier of an existing queue, we use the msgget() system call which on success returns a unique MQ identifier.

→ If a MQ associated with the IPC argument, key already exist, the call returns the identifier of the existing MQ otherwise it creates a new MQ.

→ We can use IPC_PRIVATE as a IPC argument: A parent process creates MQ prior to performing fork(); so the child inherits the returned MQ identifier so that parent & child can communicate using this MQ. For unrelated process, we can use IPC_Private but in that case the creator process has to write the returned MQ identifier in a file that can be read by the other process.

→ The 2nd argument msgflag is normally IPC_CREAT | 0666

→ Instead of IPC_PRIVATE constant as IPC argument, process can use the ftok() library call with same arguments to generate a unique key which can be used by msgget() func^n.

int msgsnd (int msgid, const void * msgp, size_t msgsz,
int msgflg);

→ It is used to send a message to the mq identifies by its 1st argument, which is the mq identifier.

→ 2nd argument is a pointer to a structure of type msgbuf having following 2 fields:

```
struct msgbuf
    { long mtype; // used to retrieve a message by type
      char mtext[512];
    };
```

→ 3rd argument msgsz is the size of mtext field in the structure msgbuf.

→ 4th argument msgflag can be 0 or IPC_NOWAIT. i.e. if mq has reached its max. no. of messages, in that case, the sender will wait until the mq does not have enough space for new msg. If it is 1 means the sender process will immediately return and will not add the msg in mq.

int msgrcv (int msgid, void * msgp, size_t maxmsgsz,
long msgtype, int msgflag)

→ The msgrcv() system call reads & removes a msg from mq identified by its 1st argument msgid & copies its contents into the buffer pointed by its 2nd argument msgp

→ 3rd argument maxmsgsz, specifies the maximum space available in the mtext field.

→ The 4th argument msgtype is used to specify as to which msg is to be removed & returned to the called caller back.

This is achieved by the specifying msgtype field of the struct msgbuf

msgtype == 0 — 1st msg from mq is removed & returned

msgtype > 0 — 1st msg from queue whose mtype field equals to msgtype is removed and returned to calling process

msgtype < 0 — 1st msg of the lowest mtype field less than or equal to absolute value of msgtype is removed & returned.

msg position in queue	msgtype	msg body
1	300	—
2	100	—
3	200	—
4	400	—
5	100	—

If msgtype = 0, i.e. msgrcv(id, &msg, maxmsgsz, 0, 0)
the receiver will receive msg in following order:

1 (mtype = 300)
2 (mtype = 100)
3 (mtype = 200)
4 (mtype = 400)
5 (mtype = 100)

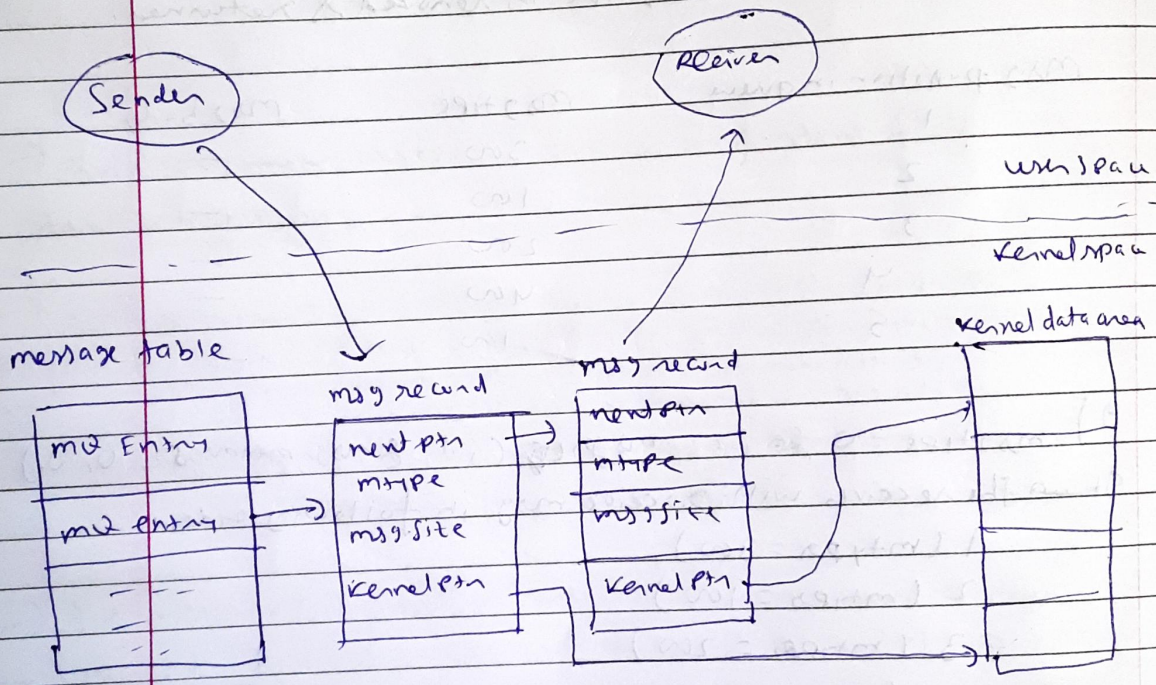
If msgrcv(id, &msg, maxmsgsz, 100, 0)
the receiver will receive msg in following order

2 (mtype = 100)
5 (mtype = 100)

if msgrcv(id, &msg, maxmsgsz, -3, 0)
the receiver will receive msg on following order:

2 (mtype = 1)
5 (mtype = 1)
3 (mtype = 2)
1 (mtype = 3)

next call of msgrcv will block the process.
Here, to notice that one msgrcv call will retrive only one msg. to retrive multiple messages like above, multiple msgrcv calls are needed.



→ to see the max. no. of ~~messages~~ Md can be created by the kernel

$ cat /proc/sys/kernel/msgmni
32000

→ to see max. bytes that can be written in a msg of a mQ.

$ cat /proc/sys/kernel/msgmax
8192

$ ipcs -q

$ gcc sender.c -o sende
$ ./sender
$ ipcs -q