vi p2.c

```
# include "header.h"
main()
{
    int f1, fd, i ;
    char s[100] ;
        mkfifo ("f1", 0644) ;
        Perror ("mkfifo") ;
        fd = open ("f1", O_RDONLY);
        Perror ("open") ;
    f1 = open ("data1", O_WRONLY | O_TRUNC | O_CREAT, 0644);
    bzero (s, sizeof(s)) ;
    read (fd, s, sizeof(s)) ;
    for( i=0 ; s[i] ; i++)
    {
        if (s[i] >= 'a' && s[i] <= 'z')
            s[i] = s[i] - 32 ;
    }
    write (f1, s, strlen(s) +1) ;
}
```

* advantages :-

1) unrelated communication.

* disadvantages :-

1) writing process write the data reading should available.
   - reading process read the data writing should available.

2) one writing process in terminal, many reading process are in 2nd terminal
   - CPU allocated the time it is first read after other are in waiting.
     (no addressing mechanism).

⊛ Overcome this disadvantages by Using a "Message queue."

**** Message Queue ****

1) msgget();
2) msgsnd();
3) msgrcv();
4) msgctl();

① msgget() :- msgget() is useful for the creating as well as opening the message queue.

② msgsnd() :- It is useful for sending the message ~~from~~ in to the message queue.

③ msgrcv() :- It is useful for receiving the message from the message queue.

④ msgctl() :- It is useful for doing some control operation in message queue.

⊛ $ man msgget

- msgget → get a message queue identifier.
- #include <sys/types.h>
  #include <sys/ipc.h>
  #include <sys/msg.h>

int msgget(key_t key, int msgflag);

⇒ The msgget() system call returns the message queue identifier associated with the value of the key argument.

- New message queue create $\boxed{\text{Value} = \text{IPC_PRIVATE}}$
- No message Flag (queue) with given key $\boxed{\text{Value} = \text{IPC_CREAT}}$

Program :-

```
#include "header.h"
main()
{
    int id;
        id = msgget (4, IPC_CREAT | 0644);
        perror("msgget");
    Printf("id = %d \n", id);
}
```

Output :- msgget : Success
id = 0

important

Que How to show message queue in Kernel ?

Ans :- ipcs -q

1) Before run the program $ ipcs -q

------ Message queue ------

Key	msgid	owner	perm.	used-bytes	message
-	-	-	-	-	-

2) after Run the program :- $ ipcs -q

Key	msgid	owner	per	used bytes	message
0X0000004	0	Savam	644	0	0
0X0000005	32768	Savam	644	0	0

* remove the message queue using 2 Ways :-

1) ipcrm -Q 4 → keyno.

Q → Keynum.
q → msgid

2) ipcrm -q 0 → msgid

* $ man msgop :-

```
# include <sys/types.h>
# include <sys/ipc.h>
# include <sys/msg.h>
```

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

2) ssize_t msgrcv (int msgid, void *msgp, size_t msgsz, long msgtyp, int msgflg);

life of the msg queue cheat if and life of end upto system is shutdown. (because it is store in Kernel.).

→ The msgp argument is a pointer to caller-defined structure of the following general form.

```
Struct msgbuf
{
    long mtype ;   // message type must be >0
    char mtext[1] ; // message data
};
```

* program :- (Sender Process) :-

```
# include " header.h "
Struct msgbuf
{
    int mtype ;
    char data[20] ;
};
main ( int argc , char ** argv )
{
    Struct msgbuf V ;
    int id ;
        if (argc != 3)
        {
            Printf (" Usage : ./a.out type data\n");
            return ;
        }
    id = msgget ( 5 , IPC_CREAT | 0644) ;
    Perror (" msgget ") ;
    Printf (" id = %d\n ", id) ;

    V. mtype = atoi (argv[1]) ;
    Strcpy (v.data , argv[2]) ;

    msgsnd (id , &V, strlen (v.data)+1 , 0) ;

    Perror (" msgsnd ") ;
}
```

* program (Receiver process) :-

```
# include "header.h"
struct msgbuf
{
    int mtype;
    char data [20];
};
main (int argc, char **argv)
{
    struct msgbuf v;
    int fd;
      if (argc != 2)
      {
          Printf (" Usage : ./rcv type \n");
          return;
      }
    id = msgget (5, IPC-CREAT | 0644);
    Perror (" msgget");
    Printf (" id = %d \n", id);

    msgrcv (id, &v, sizeof (v.data), atoi (argv[1]), 0);
    Perror (" msgrcv");
    Printf (" DATA : %s \n", v.data);
}
```

⇒ again you enter same type (means no types of message in message queue) Then "msgrcv() is block."

Flags :-

① **IPC_NOWAIT :-**

Used Return immediately if no message of the requested type is in the queue. The system call fails with errno set to ENOMSG.

② **MSG_EXCEPT :-**

Used with msgtyp greater than 0 to read the First message in the queue with message type that differs from msgtyp.

③ **MSG_NOERROR :-**

To truncate the message text if longer than msgsz bytes.

→ In sender, message queue is Full then use IPC_NOWAIT.

→ In receiver, message is not available in given type then block the rcv() so use the IPC_NOWAIT.

① **IF msgtyp +Ve :-**

The first message in the queue is read.

② **IF msgtyp +Ve :-**

The first message in the queue of type msgtyp is read, unless MSG_EXCEPT was specified in msgflg in which case the first message in the queue of type not equal to msgtyp will be read.

③ **IF msgtyp -Ve :-**

The first message in the queue with the lowest type less than or equal to the absolute value of msgtyp will be read.

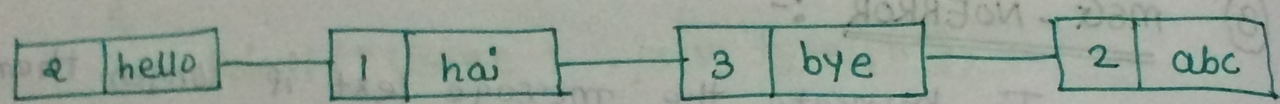
Ex:- msgrcv (—, —, —, 3, 0);

Then type of 3 msg is received.

Ex:- msgrcv (—, —, —, 3, MSG_EXCEPT);

→ without 3 type, in other first message is drop which is received.

Ex:- msgrcv (—, —, -3, 0);



⇒ o/p is "hai".

⇒ if -3 then absolute of -3 is 3. Then (less or equal) to 3 in that type lower type message is received.

Example Pass the structure :-

```
struct A
{
    int i ;
    float f ;
    char ch ;
} ;
```