③ $ man memset

memset → fill memory with a constant byte.

```
#include <string.h>

Void * memset ( Void *s , int c , size_t n);
```

Return Value :-

The memset() function returns a pointer to the memory area s.

Discription :-

→ The memset() function fills the first n bytes of the memory area pointed to by s with the constant byte c.

→ In bzero() we put only '\0'. In memset we put any other char (means 'm') or '\0'.

```
#include " header.h "
main()
{
    char a[200] ;
    int fd , ret ;
    fd = open (" data ", O_RDONLY);
    if (fd<0)
    {
        Perror (" open ");
        return ;
    }
Printf (" fd = %d \n", fd) ;
    bzero ( a, sizeof(a));
ret = read ( fd, a, 5) ;
    Perror (" read ");
Printf (" Data = %s \n", a) ;
Printf (" ret = %d \n", ret) ;
}
```

(4) $ man 2 write

write → write to a file discriptor.

```
#include <unistd.h>

ssize_t write (int fd, const void *buf, size_t count);
```

⇒ write() writes upto count bytes from the buffer pointed buf to the file refered to by the file discriptor fd.

⇒ The number of bytes written may be less than count if for example there is insufficient space on the underlying physical medium.

Success → return no. of bytes written

Failure → -1 return and errno.

①
```
char s[] = "hello";
write (fd, s, strlen(s));
```

②
```
int i = 10;
write (fd, &i, sizeof (i));
```

③
```
int a[5] = {10, 20, 30, 40, 50}
write (fd, a, sizeof (a));
```

④
```
char ch = 'a';
write (fd, &ch, 1);
```

⑤
```
struct st V = { 10, "abc", 23.5 }
write (fd, &V, sizeof (st));
```

① program :-

```
#include <stdio.h>
#include <unistd.h>
main()
{
    int fd ;
        fd = open ("data", O_RDONLY | O_TRUNC | O_APPEND |
                                        O_CREAT , 0644) ;
    Printf (" Hello World \n") ;
    printf (" fd = %d \n", fd);
}
```

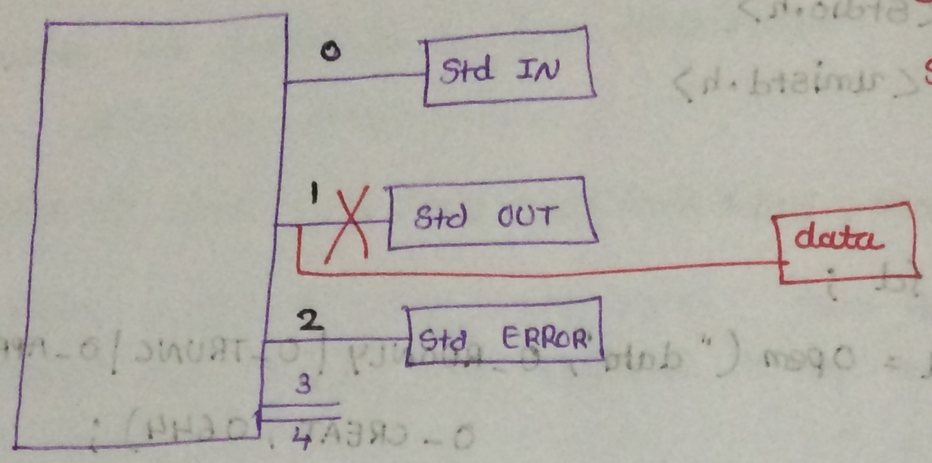
output

```
Hello world
fd = 3
```

② program :-

```
#include <stdio.h>
#include <unistd.h>
main()
{
    int fd ;
    Close (1) ;
        fd = open ("data", O_RDONLY | O_TRUNC | O_APPEND
                                | O_CREAT , 0644) ;
    Printf (" Hello world \n");
    Printf (" fd = %d \n ", fd) ;
}
```

output :- o/p is not display on screen
it is directly store in a data file.



Close(1) means Stdout is free.

⇒ Printf is not only display the result related to system call it is store the data in file also redirect the file

ls → display list of file

ls > abc then after cat abc

→ list of file copy in the abc file.

ls > abc

→ Significant :- Output redirect.

Program :- write data on a file.

```
#include " header.h"
main()
{
    int a[5] = {10,20,30,40,50}; int i;
     close(1);
      open("data", O_WRONLY | O_TRUNC | O_CREAT,
                                        0644);
```

```
for (i=0 ; i<5 ; i++) ;
    Printf (" %d ", a[i]);

}
```

Output — data is not display on screen

data "store in file"

10 20 30 40 50

Program — read from the file :-

```
#include "header.h"
main()
{   int a[5] ;
    close (0);
        open (" data ", O_RDONLY );
    for (i=0 ; i<5 ; i++)
        Scanf (" %d ", &a[i] );

    for (i=0 ; i<5 ; i++)
        Printf (" %d ", a[i] );

    Printf ("\n");

}
```

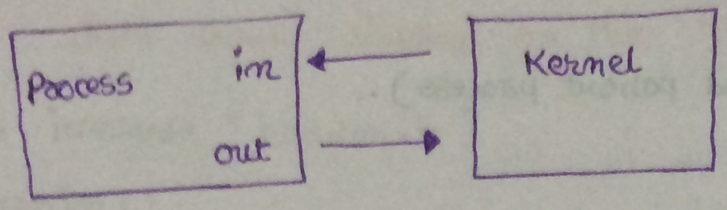
output :- display on the screen.

10 20 30 40 50

⇒ Scanf() is not used for only Scan the data but it is also used for the scan the data from the file.

*** IPC Mechanism ***

⇒ a pipe is a method of connecting the standard output of one process to the standard input of other.



1) Pipe
2) Named pipe
3) Massage queue
4) Shared memory
5) Semaphome

} Exchange data between two process. It is used for transfering the data.

} → It is used for Synchronization.

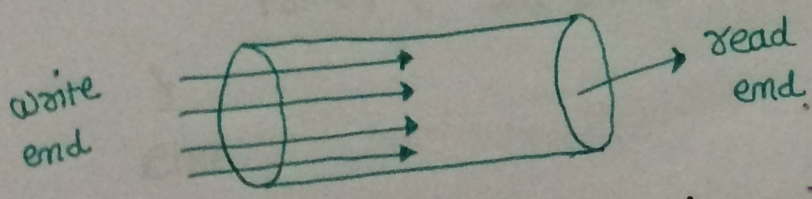
⇒ IPC is Kernel supported mechanism.

⊛ $ man 7 pipe

Pipe → overview of pipes and FIFOS.

⇒ pipes and FIFOS provide a unidirectional interprocess communication channel.

⇒ A pipe has a read end and a write end.



Only one direction can data flow.

⇒ Data written to the write end of the pipe can be read from the end of the pipe.

⇒ A pipe is Create using pipe().

⇒ which Creates a new pipe and returns two file discriptors

- one file discriptors is referring read end.
- other file discriptors is referring to the write end.

⇒ pipes can be used to create a communication channel between related process.

(child and parent process).

man 2 pipe

```
#include <unistd.h>
```

1) `int pipe (int pipefd[2]);`

2) `int pipe2 (int pipefd[2], int flags);`

> Pipe is used for interprocess communication.

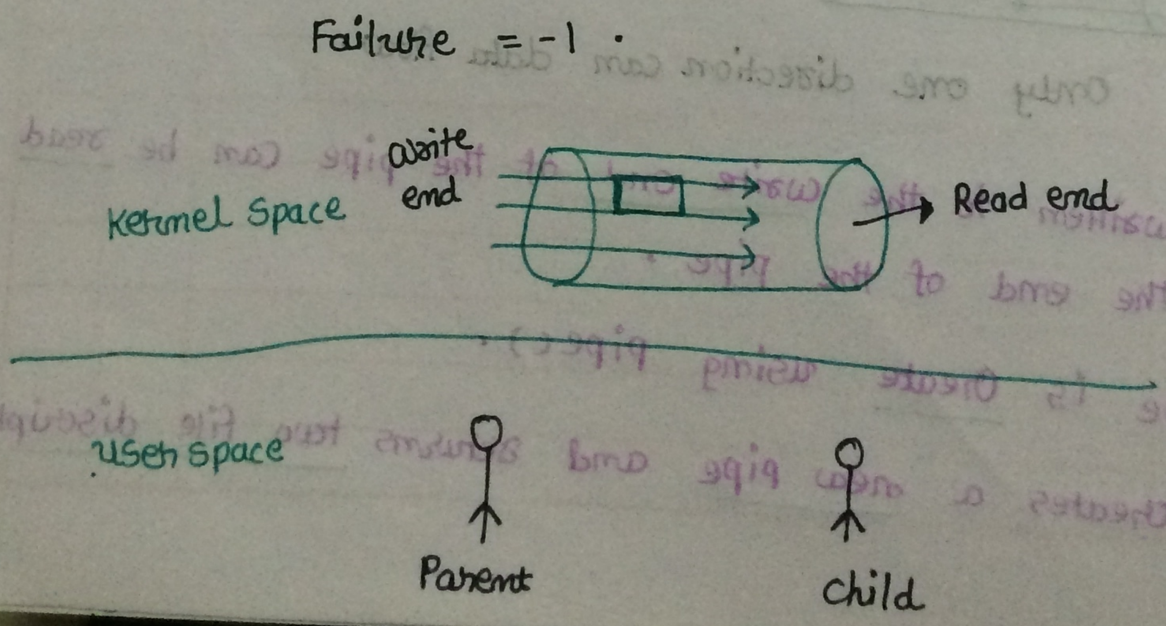
⇒ pipefd is used for return two file discriptor

> P[0] = read-end.
>
> P[1] = write-end.

* Return Value :-

Success = 0.

Failure = -1.



⇒ Child is read the data after Kernel is delet the data (or clear the data).

program :- Parent is scan the data from the Keyboard and put in the pipe. Child is read the data and display on the screen.

```
# include "header.h"
main()
{
    int p[2];
     if (pipe(p) <0)
     {
         perror (" pipe");
         return ;
     }
  Printf (" readend = p[0] = %d\n writeend = p[1] = %d\n",
                                        p[0], p[1]);

  if (fork() == =0)
  {
      char s[20];
      Printf (" In child Before read ---\n");
      read (p[0], s, sizeof(s));
      Printf (" In child Data = %s\n", s);
  }
  else
  {
      char a[20];
      Printf (" Enter the string---\n");
      scanf (" %s", a);
      write (p[1], a, strlen(a)+1);
  }
```