Output :- readend = P[0] = 3

writeend = P[1] = 4

Enter the string ----

In child Before read ----

→ hello

In child Data = hello

⇒ In Parent Scanf is blocking the process.

1) pipe is empty then read is blocked.
2) pipe is Full then write is blocked.

⊛ Loopback :-

⇒ If writing the data in pipe after this data is read by same process is called Loopback.

```
# include " header.h "
main()
{
    int p[2];
    if (Pipe(p)<0)
    {
        Perror ("Pipe");
        return;
    }
    if ( fork()==0)
    {
        Char s[20];
        Printf (" In child before read---\n");
        read (P[0], s, sizeof(s));
        for(i=0 ; s[i] ; i++) .
            if ( s[i] >='a' && s[i] <='z')
                s[i] = s[i] -32 ;
```

```
    write ( P[1] , s , strlen(s) + 1) ;
  }
  else
  {
        Char  a[20] ;
        Printf (" Enter the string ---\n") ;
        Scanf (" %s ", a) ;
        write ( P[1] , a , strlen(a) + 1) ;
        Sleep(1) ;
        read ( P[0] , a , strlen(a) + 1) ;
        Printf ( " Data = %s\n" , a) ;
  }
```

⇒ If a process attempts to read from an empty pipe then read will block until data is available.

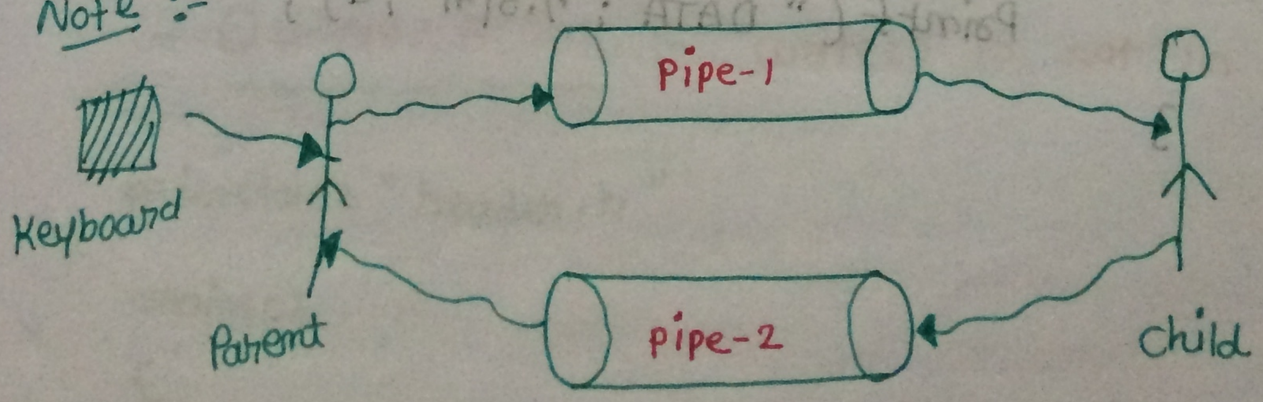
⇒ If a process attempts to write to a full pipe, the write() block until sufficient data has been read from the pipe to allow the write to complete.

Program :-

Parent Scan the data from the Keyboard and put the data in pipe 1 Then child read the data from pipe-1.

- Child convert the data and put in pipe-2 and parent read the data from pipe-2 and display on the screen.

Note :-



```
# include " header.h "

main()
{
    int P[2], i;
    int P1[2];

        if( pipe(P)<0 || pipe(P1)<0)
        {
            perror(" pipe ");
            return;
        }
        if (fork()==0)
        {
            char s[20];
            Printf(" In Child Before the read ----\n");
            read (P[0], s, sizeof(s));
            for(i=0; s[i]; i++)
                if(s[i]>='a' && s[i]<='z')
                    s[i] = s[i] -32;
            write (P1[1], s, strlen(s)+1);
        }
        else
        {
            char a[20];
            Printf(" Enter the String ----\n");
            gets(a);
            write (P[1], a, strlen(a)+1);
            read (P1[0], a, sizeof(a));
            Printf(" DATA : %s\n", a);
        }

}
```

program :- write a program to find out the size of the pipe in our Kernel Version.

Que :- How to find out our Kernel Version. (command).

→ uname -r → Kernel Version (Release)

uname -a →

uname -V → Kernel Version.

```
# include " header.h "
main()
{
    char ch='a';
    int p[2], c=0;
        pipe(p);
    while (write(p[1], &ch, 1) != -1)
    {
        c++
        Printf(" %d\n", c);
    }
    Printf(" Bye---\n");
}
```

output :- after sometime pipe is full than write() is block

Then o/p is 65536

Flags :-

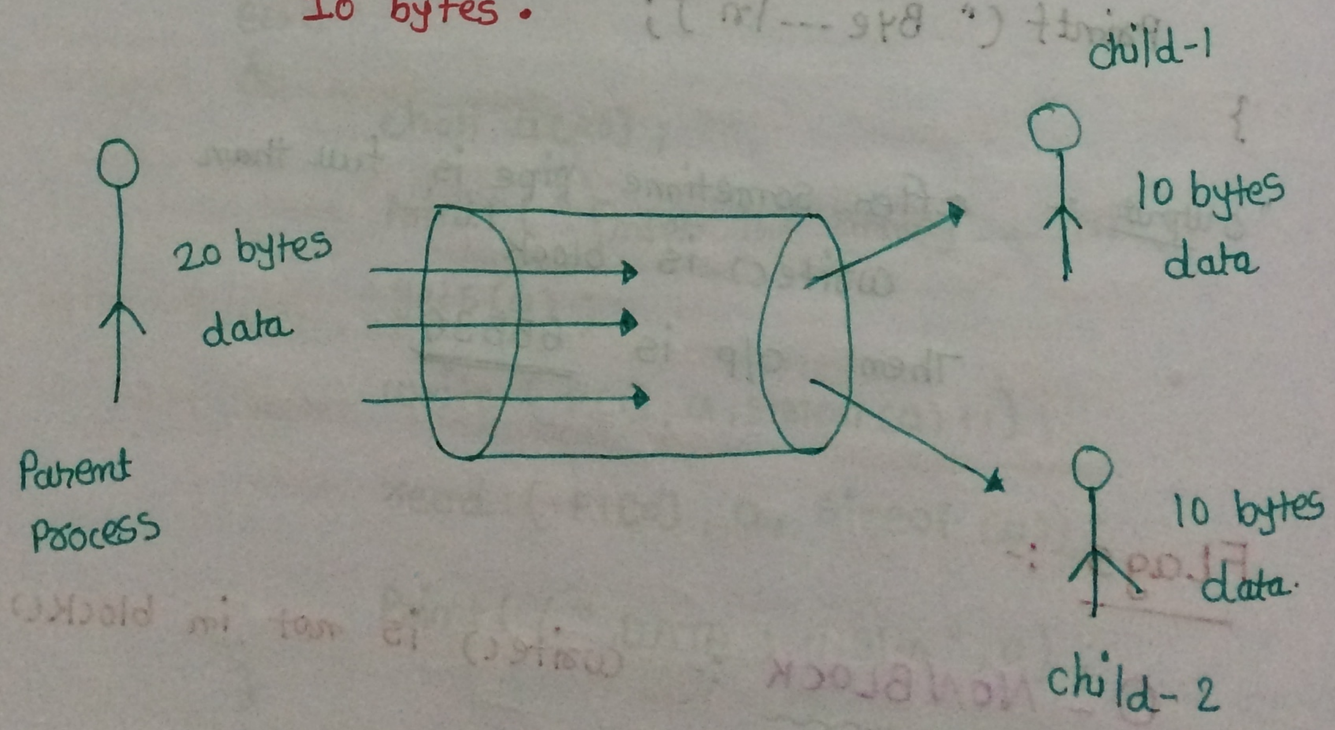
O_NONBLOCK :- write() is not in block() state

```
# include " header.h "
main()
{
```

```
char ch='a';
int p[2], c=0;
Pipe2(p, O_NONBLOCK);
while (write(p[1], &ch, 1) != -1)
{
    c++;
}
Printf("%d\n", c);
Printf("Bye---\n");
}
```

⇒ The communication channel provided by a pipe is a byte stream : there is no concept of message boundaries.

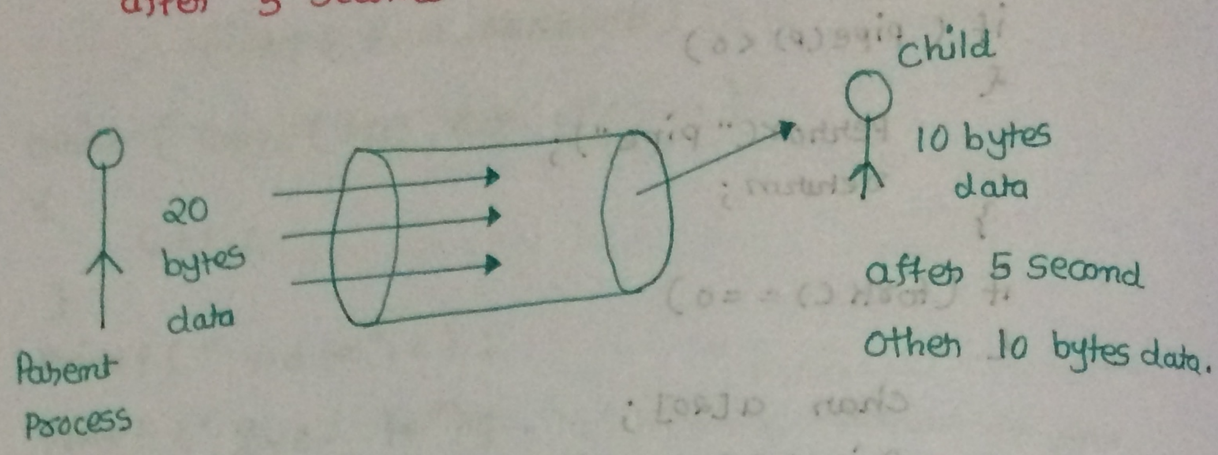
Program-1 :- Write a parent can write 20 bytes data in pipe. 1 child is read from pipe only 10 bytes. 2 child is read only 10 bytes.



```
# include " header.h "

main()
{
        int  p[2];
            if ( pipe(p) < 0 )
            {
                perror(" pipe ");
                return ;
            }
            if (fork () = = 0 )
            {
                char  a[20] ;
                Printf (" In child_1 ---- \n") ;
                bzero (a, sizeof(a)) ;
                read ( p[0], a, 5) ;
                Printf (" In child_1 = %s \n", a) ;
            }
            else
            {
                if ( fork () = = 0 )
                {
                    Char  b[20] ;
                    Printf (" In child_2 --- \n") ;
                    bzero ( b, sizeof (b)) ;
                    read ( p[0], b, 5) ;
                    Printf (" In child_2 = %s \n" b ) ;
                }
                else
                {
                    char  m[20] ;
                    int s ;
                    Printf (" Enter the String --- \n") ;
                    scanf (" %s", m) ;
                    write ( p[1], m, strlen(m) + 1 ) ;
                }
            }
}
```

Program-2 :- Parent can write 20 bytes data in a pipe child can first read only 10 bytes after 5 second other 10 bytes data read.




```
# include "header.h"
main()
{
        int P[2], i ;
        if (Pipe(P) < 0)
        {
            perror ("Pipe") ;
            return ;
        }
        if (fork() == 0)
        {
            char S[20] ;
            Printf ("In child before Read----\n") ;
            bzero (S, Sizeof(S)) ;
            read (P[0], S, 10) ;
            Printf (" Data_1 = %s\n", S) ;
            sleep(5) ;
            bzero (S, Sizeof(S)) ;
            read (P[0], S, 10) ;
            Printf (" Data_2 = %s\n", S) ;
        }
        else
        {
            char a[20] ;
            int s ;
            Printf (" Enter the String----\n") ;
            Scanf (" %s ", a) ;
            write (P[1], a, strlen(a) +1) ;
            wait (&s) ;
}       }
```

⇒ If all file discriptors referring to the write end of a pipe have been closed, then an attemp to read() from the pipe see end-of-File (read will return 0).

⇒ If all the File discriptors referring to the read end of the pipe have been close then a write() will be cause a SIGPIPE signal to be generated for the calling process.

⇒ If the calling process is ignoring this signal, then write() fails with the error EPIPE.

advantages :-

1) communicate with only related file.

**** FIFO ****

⇒ Communicate with unrelated file.

1) mkfifo
2) open
3) read
4) write

⊛ $ man 7 fifo

fifo → first-in-first-out special file (named pipe).

⇒ A FIFO special file is similar to a pipe except that it is accessed as part of the file system. It can be open by multiple process for reading or writing. when process are exchanging data via the FIFO, the kernel passes all data internally without writing it to the file system.