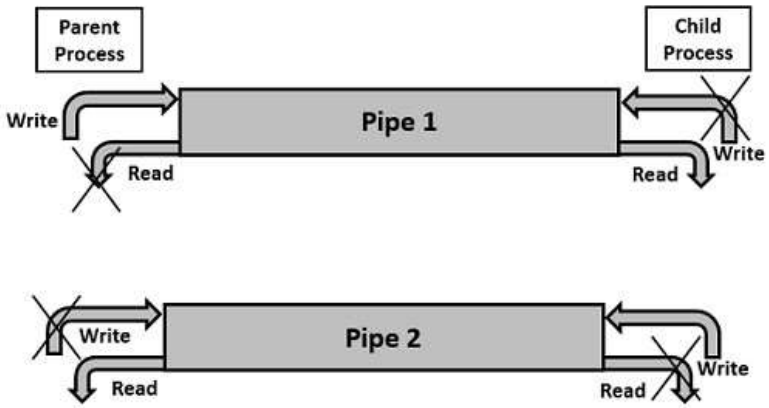
```
#include <sys/types.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#define BUFFER_SIZE 25
#define READ_END 0
#define WRITE_END 1

int main(void)
{
char write_msg[BUFFER_SIZE] = "Greetings";
char read_msg[BUFFER_SIZE];
int fd[2];
pid_t pid;

/* create the pipe */
if (pipe(fd) == -1) {
   fprintf(stderr,"Pipe failed");
   return 1;
}

/* fork a child process */
pid = fork();

if (pid < 0) { /* error occurred */
   fprintf(stderr, "Fork Failed");
   return 1;
}
```

```
    if (pid > 0) { /* parent process */
      /* close the unused end of the pipe */
      close(fd[READ_END]);

      /* write to the pipe */
      write(fd[WRITE_END], write_msg, strlen(write_msg)+1);

      /* close the write end of the pipe */
      close(fd[WRITE_END]);
    }
    else { /* child process */
      /* close the unused end of the pipe */
      close(fd[WRITE_END]);

      /* read from the pipe */
      read(fd[READ_END], read_msg, BUFFER_SIZE);
      printf("read %s",read_msg);

      /* close the write end of the pipe */
      close(fd[READ_END]);
    }

    return 0;
}
```

2 way communication using pipe system call

- ✓pipe1 - parent process writes ; child process reads
- ✓pipe2 - child process writes ; parent process reads.
- ✓parent and child process - unwanted ends are closed
- ✓First parent writes string onto the pipe ; read by child and displayed on screen
- ✓Second part - Child process writes a message; read by parent and prompted on the screen



```
#include<stdio.h>
#include<unistd.h>
 int main() {
int pipefds1[2], pipefds2[2];
int returnstatus1, returnstatus2;
int pid; char pipe1writemessage[20] = "Hi";
char pipe2writemessage[20] = "Hello";
char readmessage[20];
returnstatus1 = pipe(pipefds1);
if (returnstatus1 == -1)
{
printf("Unable to create pipe 1 \n");
return 1;}
returnstatus2 = pipe(pipefds2);
if (returnstatus2 == -1)
{
 printf("Unable to create pipe 2 \n");
return 1;}
```

```
pid = fork();
if (pid > 0) // Parent process
{   close(pipefds1[0]);     // Close the unwanted pipe1 read side
    close(pipefds2[1]);   // Close the unwanted pipe2 write side
    printf("In Parent: Writing to pipe 1 – Message is %s\n",
    pipe1writemessage);
write(pipefds1[1], pipe1writemessage,
    sizeof(pipe1writemessage)+1);
read(pipefds2[0], readmessage, sizeof(readmessage));
printf("In Parent: Reading from pipe 2 – Message is %s\n",
    readmessage);
}
```

```
else
{  close(pipefds1[1]);   // Close the unwanted pipe1 write side
   close(pipefds2[0]);   // Close the unwanted pipe2 read side
   read(pipefds1[0], readmessage, sizeof(readmessage));
printf("In Child: Reading from pipe 1 – Message is %s\n",
   readmessage); printf("In Child: Writing to pipe 2 – Message is
   %s\n", pipe2writemessage);
write(pipefds2[1], pipe2writemessage,
   sizeof(pipe2writemessage)+1);
}
return 0; }
```

OUTPUT

In Parent: Writing to pipe 1 – **Message is Hi**

In Child: Reading from pipe 1 – Message is Hi

In Child: Writing to pipe 2 – **Message is Hello**

In Parent: Reading from pipe 2 – **Message is Hello**

Exercises:

(1) Parent sets up a string which is read by child, reversed there and read back the parent

(2) Parent sets up string 1 and child sets up string 2. string 2 concatenated to string 1 at parent end and then read back at the child end.

(3) Substring generation at child end of a string setup at parent process end.

(4) String reversal and palindrome check using pipes / shared memory.