

## PROGRAM1:

The program provided doesn't work and the parent process waits indefinitely for the child process to execute. This is because each parent and child process has its own memory space for the variables and thus is independent of what happens in other process.

The solution for this is:

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

void main()
{
    char c;
    int pfd[2];
    if (pipe(pfd) > 0)
    {
        printf("error");
    }
    if (!fork())
    {
        close(pfd[0]);
        printf("press a key");
        scanf("%c", &c);
        write(pfd[1], &c, 1);
        exit(0);
    }
    else
    {
        close(pfd[1]);
        read(pfd[0], &c, 1);
        printf("I received the character %c",c);
        exit(0);
    }
}
```

```
press a keyD
I received the character D[1] + Done
gine-In-aizmc0r2.jlk" 1>"/tmp/Microsoft-I
```

## Program2:

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

void main()
{
    int pfd[2];
    if (pipe(pfd) < 0)
        printf("error ");
    if (!fork())
    {
        char data;
        printf("I'm child\n");
        printf("press any key to exit.....\n");
        scanf("%c",&data);
        write(pfd[1], &data, 1);
        printf("child exiting\n");
    }
    else
    {
        char data;
        read(pfd[0], &data, 1);
        printf("I'm parent\n");
        printf("received %c from child\n", data);
        printf("parent exiting.....\n");
        exit(0);
    }
}
```

```
I'm child
press any key to exit.....
k
child exiting
I'm parent
received k from child
parent exiting.....
[1] + Done
```

In this program pipe is used to pass a character from child process to the parent process. The read is a blocking call so the parent process waits till it receives a character from child process. Also, in above output we observe that parent finished execution earlier so the child process become orphan and its last print statement occurs at new command line.

### Program3:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#define msgsz 16
void main() {
    char *msg1= "hello one");
    char *msg2= "hello two");
    char *msg3= "hello three");
    char inbuf[msgsz];
    int p[2],j;
    pipe(p);
    write(p[1],msg1,msgsz);
    write(p[1],msg2,msgsz);
    write(p[1],msg3,msgsz);
    for(j=0;j<3;j++){
        read(p[0],inbuf,msgsz);
        printf("%s\n",inbuf);
    }
    exit(0);
}
```

```
hello one
hello two
hello three
[1] + Done
tmp/Microsoft-MIEngine-Out-d232dj
```

In the given program, both ends of pipe is used by same process first to write some characters and then to read those written characters. This can be modified as follows to write from the child process and read it from parent.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#define msgsz 16

void main()
{
    char *msg1 = "hello one";
    char *msg2 = "hello two";
    char *msg3 = "hello three";
    char inbuf[msgsz];
    int p[2];
    pipe(p);
    if(!fork())
    {
        printf("\n message sent from child:%s",msg1);
        fflush(stdout);
        write(p[1],msg1,msgsz);
        printf("\nMessage sent from child:%s",msg2);
        fflush(stdout);
        write(p[1],msg2,msgsz);
        printf("\nmessage sent from child:%s",msg3);
    }
```

```

        fflush(stdout);
        write(p[1],msg3,msgsz);
    }
    else{
        char inbuff[msgsz];
        for(int i=0;i<3;i++){
            read(p[0],inbuf,msgsz);
            printf("\nMessage received in parent: %s",inbuf);
            fflush(stdout);
        }
    }
}
}

```

```

message sent from child:hello one
Message received in parent: hello one
Message sent from child:hello two
Message received in parent: hello two
message sent from child:hello three
Message received in parent: hello three[1] + Done
Microsoft-MIEngine-In-r05cle1l.4rw" 1>"/tmp/Micro

```

Message can be sent from parent and then received from child just by changing !fork() to fork() in the if statement in the above program code.

#### Program4:

```

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

void main()
{
    int p[2],pid;
    pipe(p);
    pid = fork();
    if (pid == 0)
        printf("in the child p[0] is %d p[1] is %d\n", p[0], p[1]);
    else
        printf("in the parent p[0] is %d p[1] is %d\n", p[0], p[1]);
}

```

```

in the parent p[0] is 3 p[1] is 4
in the child p[0] is 3 p[1] is 4
[1] + Done
"/usr
tmp/Microsoft-MIEngine-Out-twxyzlno.fs

```

Swapping contents within parent and child process in program 2 we get:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
void main()
{
    int pfd[2];
    if (pipe(pfd) < 0)
        printf("error ");
    if (!fork())
    {
        char data;
        printf("I'm parent\n");
        printf("press any key to exit.....\n");
        scanf("%c",&data);
        write(pfd[1], &data, 1);
        wait(NULL);
        printf("parent exiting\n");
    }
    else
    {
        char data;
        read(pfd[0], &data, 1);
        printf("I'm child\n");
        printf("received %c from parent\n", data);
        printf("child exiting.....\n");
        exit(0);
    }
}
```

```
I'm parent
press any key to exit.....
i
parent exiting
I'm child
received i from parent
child exiting.....
[1] + Done                               "/u
tmp/Microsoft-MIEngine-Out-hxyz2wyv.
```

Here it is observed that the parent process exits first making the child process orphan which leads the shell to think that the program has been completed but then the output from child process appears afterwards. To rectify this problem a wait system call should be inserted in parent process before exiting.

## Program5:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#define msgsz 16

void main() {
    char *msg="hello1";
    char inbuf[msgsz];
    int p[2],pid,j;
    pipe(p);
    pid=fork();
    if(pid>0)
    {
        close(p[0]);
        write(p[1],msg1,msgsz);
    }
    if(pid==0)
    {
        close(p[1]);
        read(p[0],inbuf,msgsz);
        printf("%s\n",inbuf);
    }
    exit(0);
}
```

```
hello1
[1] + Done                               "/usr/
tmp/Microsoft-MIEngine-Out-pswmmzom.r46
suyog@suyog-VirtualBox:~/Desktop/OS lab
```

## After modification

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#define msgsz 16

int main() {
    char *msg = "hello1";
    char inbuf[msgsz];
    int p[2], pid, j;
    pipe(p);
    pid = fork();
    if (pid > 0) {
        if (close(p[0]) == 0) {
            printf("Read file descriptor closed successfully!\n");
        }
        if (read(p[0], inbuf, msgsz) == -1) {
```

```

        printf("Unable to read from pipe!\n");
    } else {
        printf("Successfully read content from pipe!\n");
        printf("In parent, read data : %s\n", inbuf);
    }
    write(p[1], msg, msgsz);
}
if (pid == 0) {
    if (close(p[1]) == 0) {
        printf("Write file descriptor closed successfully!\n");
    }
    if (write(p[1], msg, msgsz) == -1) {
        printf("Unable to write to pipe!\n");
    } else {
        printf("Successfully written content to pipe!\n");
    }
    read(p[0], inbuf, msgsz);
    printf("In child, data read : %s\n", inbuf);
}
exit(0);
}

```

```

Read file descriptor closed successfully!
Unable to read from pipe!
Write file descriptor closed successfully!
Unable to write to pipe!
In child, data read : hello1

```