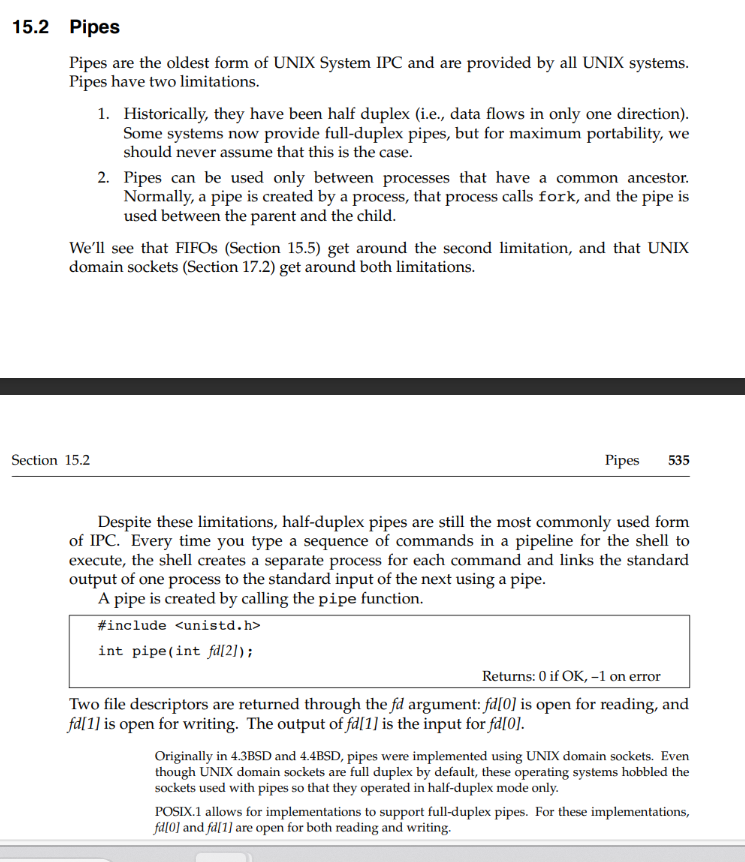
**DAY -24(29-11-2024)**

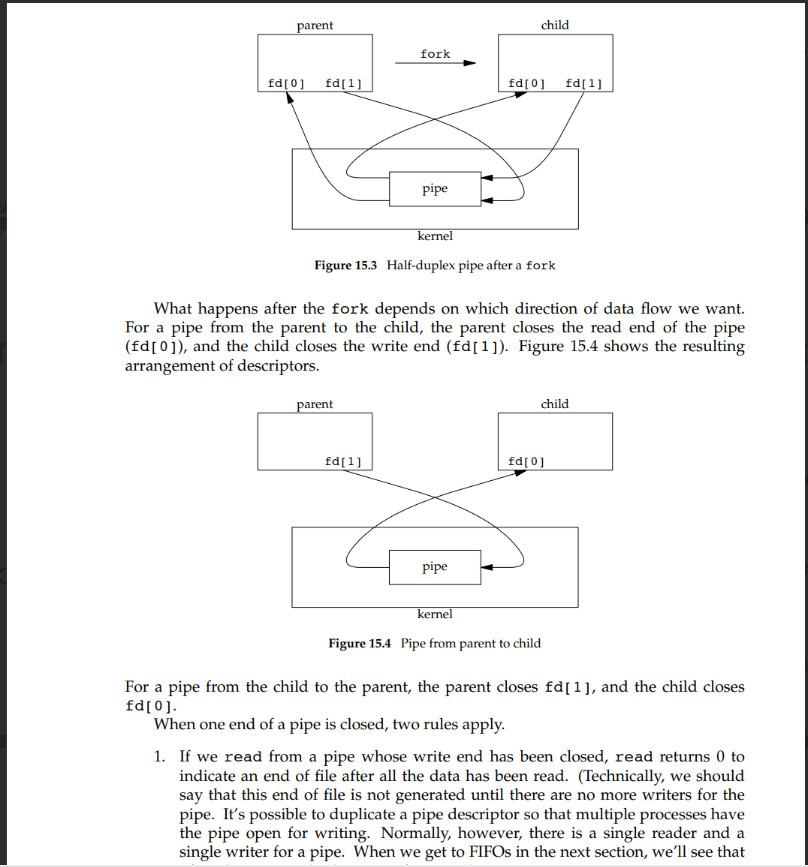
**REFER AUPE-3RD PDF FOR IPC (downloads)**

* **PIPES:**
* Fd[0] is open for reading
* Fd[1] is open for writing
* The output of fd[1] is the input for fd[0].



A screenshot of a computer

Description automatically generated



A screenshot of a document

Description automatically generated

* Program 1:

#include<stdio.h>

2 #include<stdlib.h>

3 #include<string.h>

4 #include<sys/wait.h>

5 #include<sys/types.h>

6 #include<unistd.h>

7 #define BUFF 1024

8 int main()

9 {

10 int fd[2];

11 pid\_t cpid;

12 char line[BUFF];

13 pipe(fd);

14 cpid=fork();

15 if(cpid==0)

16 {

17 //child process

18 close(fd[0]);// redaing is closed

19 write(fd[1],"HELLO WORLD",12);

20 exit(EXIT\_SUCCESS);

21 }

22 else

23 {

24 //parent process

25

26 close(fd[1]);//writing is closed

27 read(fd[0],line,BUFF);

28 printf("line read :%s \n",line);

29 }

30 printf("\n ended program\n");

31 }

Output:

line read :HELLO WORLD

ended program

Program 2:

#include<stdio.h>

2 #include<stdlib.h>

3 #include<string.h>

4 #include<sys/wait.h>

5 #include<sys/types.h>

6 #include<unistd.h>

7 #define BUFF 1024

8 int main()

9 {

10 int fd[2];

11 pid\_t cpid;

12 char line[BUFF];

13 pipe(fd);

14 int status;

15 cpid=fork();

16 if(cpid==0)

17 {

18 //child process

19 close(fd[0]);// redaing is closed

20 // write(fd[1],"HELLO WORLD",12);

21 exit(EXIT\_SUCCESS);

22 }

23 else

24 {

25 //parent process

26 wait(&status);

27 printf("\n child with %d id exited with status %d ",cpid,(status));

28 close(fd[1]);//writing is closed

29

30 read(fd[0],line,BUFF);

31 printf("line read :%s \n",line);

32 }

33 printf("\n ended program\n");

34 }

* IPC(inter process communication)
* Ipcs
* It contains key (which is unique)

user44@trainux01:~$ ipcs

------ Message Queues --------

* key msqid owner perms used-bytes messages
* ------ Shared Memory Segments --------

key shmid owner perms bytes nattch status

* ------ Semaphore Arrays --------

key semid owner perms nsems

* #include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

int msgget(key\_t key, int msgflg);

* The msgget() system call returns the System V message queue identifier associated with the value of the key argument. A new message queue is created if key has the value IPC\_PRIVATE or key isn't IPC\_PRIVATE, no message queue with the given key key exists, and IPC\_CREAT is specified in msgflg.
* If msgflg specifies both IPC\_CREAT and IPC\_EXCL and a message queue already exists for key, then msgget() fails with errno set to EEXIST.

A black background with white text

Description automatically generated

* Message send:
* #include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

int msgsnd(int msqid, const void \*msgp, size\_t msgsz, int msgflg);

ssize\_t msgrcv(int msqid, void \*msgp, size\_t msgsz, long msgtyp, int msgflg);

* The msgsnd() and msgrcv() system calls are used, respectively, to send messages to, and receive messages from, a System V message queue. The calling process must have write permission on the message queue in order to send a message, and read permission to receive a message.
* ftok - convert a pathname and a project identifier to a System V IPC key
* #include <sys/types.h>

#include <sys/ipc.h>

key\_t ftok(const char \*pathname, int proj\_id);

* A computer screen shot of a program

  Description automatically generated
* Program:

#include<stdio.h>

2 #include<sys/ipc.h>

3 #include<sys/wait.h>

4 #include<sys/types.h>

5 int main()

6 {

7 key\_t key=ftok("/home2/user44/day23",90013);

8

9 printf("%u",key);

10

11 int msgid;

12 msgid=msgget((key\_t)300,IPC\_CREAT| IPC\_EXCL);

1. printf("\n msg queue with %d id\n",msgid);

14

15 printf("\n\n");

16 return 0;

17 }

* **message send:**

program:

#include "myheader.h"

#define PERMS 0666

typedef struct msgText{

long msgtype;

char txtMsg[1024];

}MSG;

int main()

{

int msgid, len=0;

MSG msg1;

msgid = msgget((key\_t)15,IPC\_CREAT|PERMS);

if(msgid < 0){

perror("msgget ");

\_exit(EXIT\_SUCCESS);

}

printf("\nMSG queue created with %d id\n",msgid);

printf("\nTO send msg to msgid = %d\n",msgid);

msg1.msgtype = 1;

strcpy(msg1.txtMsg,"Hi Bhima, Hope you are doing good\n");

len = strlen(msg1.txtMsg);

if(msgsnd(msgid,&msg1,len,IPC\_NOWAIT)==-1)

{

perror("msgsnd ");

return (EXIT\_FAILURE);

}

printf("\n\n");

return 0;

}

A computer screen shot of a black screen

Description automatically generated

A screenshot of a computer screen

Description automatically generated

Program 2:

#include "myheader.h"

#define PERMS 0666

typedef struct msgText{

long msgtype;

char txtMsg[1024];

}MSG;

int main()

{

int msgid, len=0;

MSG msg1;

msgid = msgget((key\_t)15,IPC\_CREAT|PERMS);

if(msgid < 0){

perror("msgget ");

\_exit(EXIT\_SUCCESS);

}

printf("\nMSG queue created with %d id\n",msgid);

printf("\nTO send msg to msgid = %d\n",msgid);

msg1.msgtype = 1;

strcpy(msg1.txtMsg,"Hi Bhima3, Hope you are doing good\n");

len = strlen(msg1.txtMsg);

if(msgsnd(msgid,&msg1,len,IPC\_NOWAIT)==-1)

{

perror("msgsnd ");

return (EXIT\_FAILURE);

}

printf("\n\n");

return 0;

}

**Message receive:**

Program 1:

#include "myheader.h"

#define BUFF 1024

#define PERMS 0666

typedef struct msgText{

long msgtype;

char txtMsg[1024];

}MSG;

int main()

{

int msgid, len=0;

MSG msg1;

msgid = msgget((key\_t)15,IPC\_CREAT|PERMS);

if(msgid < 0){

perror("msgget ");

\_exit(EXIT\_SUCCESS);

}

printf("\nMSG queue created with %d id\n",msgid);

printf("\nTO recv msg from msgid = %d\n",msgid);

if(msgrcv(msgid,&msg1,BUFF,0,0)<0)

{

perror("msgrcv ");

return (EXIT\_FAILURE);

}

printf("\nReceived message: Type: %ld\n",msg1.msgtype);

printf("\nReceived message: Msg: %s\n",msg1.txtMsg);

printf("\n\n");

return 0;

}

A screenshot of a computer program

Description automatically generated

Program 2:

#include "myheader.h"

#define BUFF 1024

#define PERMS 0666

typedef struct msgText{

long msgtype;

char txtMsg[1024];

}MSG;

int main()

{

int msgid, len=0;

MSG msg1;

msgid = msgget((key\_t)15,IPC\_CREAT|PERMS);

if(msgid < 0){

perror("msgget ");

\_exit(EXIT\_SUCCESS);

}

printf("\nMSG queue created with %d id\n",msgid);

printf("\nTO recv msg from msgid = %d\n",msgid);

if(msgrcv(msgid,&msg1,BUFF,0,IPC\_NOWAIT)<0)

{

perror("msgrcv ");

if(msgctl(msgid,IPC\_RMID,0)<0)

{

perror("msgctl ");

\_exit(EXIT\_SUCCESS);

}

return (EXIT\_FAILURE);

}

printf("\nReceived message: Type: %ld\n",msg1.msgtype);

printf("\nReceived message: Msg: %s\n",msg1.txtMsg);

printf("\n\n");

return 0;

}

**Message control:**

Program:

#include "myheader.h"

#define PERMS 0666

int main()

{

int msgid;

msgid = msgget((key\_t)13,IPC\_CREAT|PERMS);

if(msgid < 0){

perror("msgget ");

\_exit(EXIT\_SUCCESS);

}

printf("\nMSG queue created with %d id\n",msgid);

printf("\nTO remove msgid = %d from ipc table\n",msgid);

if(msgctl(msgid,IPC\_RMID,0)<0)

{

perror("msgctl ");

\_exit(EXIT\_SUCCESS);

}

printf("\n\n");

return 0;

}

To check the msqid use command ipcs -q

HEADER FILE:

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <unistd.h>

#include <signal.h>

#include <sys/ipc.h>

#include <sys/types.h>

#include <sys/msg.h>

#include <string.h>

* SHARED MEMORY:
* Shared memory allows two or more processes to shareagiven region of memory. This is the fastest form of IPC, because the data does not need to be copied between the client and the server. The only trick in using shared memory is synchronizing access to a given region among multiple processes. If the server is placing data into a shared memory region, the client shouldn’t try to access the data until the server is done. Often, semaphores are used to synchronize shared memory access.
* Once a shared memory segment has been created, a process attaches it to its address space by calling shmat.
* #include <sys/shm.h>

void \*shmat(int shmid, const void \*addr, int flag);

Returns: pointer to shared memory segment if OK, −1 on error

The address in the calling process at which the segment is attached depends on the addr argument and whether the SHM\_RND bit is specified in flag.

* • If addr is 0, the segment is attached at the first available address selected by the kernel. This is the recommended technique.
* • If addr is nonzero and SHM\_RND is not specified, the segment is attached at the address given by addr.
* • If addr is nonzero and SHM\_RND is specified, the segment is attached at the address given by (addr − (addr modulus SHMLBA)). The SHM\_RND command stands for ‘‘round.’’ SHMLBA stands for ‘‘low boundary address multiple’’ and is always a power of 2. What the arithmetic does is round the address down to the

next multiple of SHMLBA

**program 1 : reading**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdlib.h>

#include <string.h>

#define PERMS 0666

#define MAXBUF 1024

int main()

{

key\_t key = 10003;

int shmid;

char msg[] = "Hello Bhima, WellCome to C Programmming class";// = NULL; //(char \*)malloc(MAXBUF);

int len = strlen(msg);

shmid = shmget(key,len+1,PERMS|IPC\_CREAT);

char \*ptr = NULL;

if(shmid < 0)

{

perror("shmget()");

printf("\nDirectly use this shm mem to read and write");

exit(EXIT\_FAILURE);

}

printf("\nShared mem id =%d\n",shmid);

// strcpy(msg,"Hello, I am Shared Memory Segment");

// msg = (char \*)shmat(shmid, (void \*)0,0);

ptr = (char \*)shmat(shmid,(void \*)0,0);

ptr = msg;

shmdt(ptr);

return 0;

}

**Program 2:**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdlib.h>

#include <string.h>

#define PERMS 0666

#define MAXBUF 1024

int main()

{

key\_t key = 10003;

int shmid;

// char msg[] = "Hello Bhima, WellCome to C Programmming class";// = NULL; //(char \*)malloc(MAXBUF);

shmid = shmget(key,46,PERMS|IPC\_CREAT);

char \*ptr = NULL;

if(shmid < 0)

{

perror("shmget()");

printf("\nDirectly use this shm mem to read and write");

exit(EXIT\_FAILURE);

}

printf("\nShared mem id =%d\n",shmid);

ptr = (char \*)shmat(shmid,(void \*)0,0);

printf("\nMsg = %s\n",ptr);

shmdt(ptr);

return 0;

}

* Synchronization(making other person to wait for a while once the first thread is release then other one is going) 🡪 wait, mutex
* **SEMAPHORE:**
* #include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

int semget(key\_t key, int nsems, int semflg);

* Program:

#include <stdio.h>  
#include <string.h>  
#include<sys/sem.h>  
#include<unistd.h>  
#include <stdlib.h>  
#include <sys/ipc.h>  
#define MAX\_BUFF 1024  
#define PERMS 0660

int main(){  
    int semid, nsem = 0;  
    key\_t key = ftok("/home2/user53/day24",11101);

    nsem = 1;  
    semid = semget(key,nsem,PERMS|IPC\_CREAT);

    if(semid<0)  
    {  
       perror("semget()");  
       \_exit(EXIT\_FAILURE);

    }  
    printf("\nSemaphore is created with ID %d",semid);

    printf("\n\n");  
    return 0;

}

#include <sys/types.h>

**Semctl:**

#include <sys/ipc.h>

#include <sys/sem.h>

int semctl(int semid, int semnum, int cmd, ...);

DESCRIPTION

semctl() performs the control operation specified by cmd on the System V semaphore set identified by semid, or on the semnum-th semaphore of that

set. (The semaphores in a set are numbered starting at 0.)