LAB MANUAL

FOR

LINUX PROGRAMMING

B TECH IV YEAR I SEMESTER

CSE

CVR COLLEGE OF ENGINEERING

(UGC Autonomous Institution)

ACCREDITED BY NBA & NAAC

(Approved by AICTE & Govt. of Telangana and Affiliated to JNTU, Hyderabad)

VASTUNAGAR, MANGALPALLI(V), IBRAHIMPATNAM (M),

R.R. DISTRICT. PIN: 501510

EMAIL: info@cvr.ac.in

WEB: http://www.cvr.ac.in

<u>LP LAB MANUAL Master COPY</u>

<u>15 batch</u>

LIST OF EXPERIMENTS:

1. Implement ‘cp and ‘mv’ shell commands using file related system calls.

2. Create a new file with 0666 access permissions and enable the close-on-exec flag.

4.Write a C Program to implement a UNIX **‘ls -l’** command using File related API & stat structure.

5. Write a C Program to implement a UNIX ‘ls –ls dir1’ command using directory related system calls.

6. Write a C program which creates a child process and the parent waits for child’s exit.

7. Write a C program to demonstrate the difference between the fork and vfork system calls.

8.Write a C program in which main process creates a child process and registers a signal handler to get the exit status of the child asynchronously.

9. Implement ‘ ls|wc -l -c –w ’ command using pipe and exec functions.

10. Establish bidirectional communication between sender program and receiver program using multiple FIFO’s.

11. Implement SVR based Message Queue IPC mechanism to establish asynchronous communication between two communicating processes.

12.Implement the following communication model:

- Process 1 creates a Message Queue resource.
- Process 2 enacts the server role
- Process 3 and 4 are clients
- Process 3 seeks ‘isprime’ service from the server by inserting the payload in the message queue
- Process 4 seeks ‘iseven’ service form the server by inserting the payload in the message queue
- Server retrieves the service request from the Message queue and inserts the reply
- Intended Client retrieves the response.

13. Implement shared Memory based communication model with the following features:
    a) Server and multiple clients communicate with each other through shared memory.
    b) Synchonization of SHM access is realized through semaphores.

14.Implement client/server model using socket API.

15. Implement concurrent server using fork based model while avoiding the zombie state of the client.

16. Implement a concurrent server model using pthread API.

17. Solve the producer consumer problem using pthread API.

18. Implement peer-to-peer communication model using socket API.

19. Solve the process synchronization on I/O using record locking mechanism.

20. Implement I/O multiplexing using select system call.

Course Outcomes: At the end of the course, the student should be able to

CO 1: Realize basic system calls and library functions on file operations.

CO 2: Model the process abstraction and process control

CO 3: Implement concurrent programs using process and thread API and establish communication among them.

CO 4: Implement and deploy scalable client-server architecture while utilizing relevant design patterns.

1. Implement ‘cp and ‘mv’ shell commands using file related system calls.

PROGRAM:

CP.c

```
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
int main()
{
        int fh_desr=0,fh_desw=0,sizer=0;
        char buf[30];

        fh_desr=open("src.txt",O_RDONLY);
        if(fh_desr<0)
        {
                printf("Source file not opened");
                return 0;
        }
        fh_desw=open("dest.txt",O_CREAT|O_WRONLY,0777);
        if(fh_desw<0)
        {
                printf("Destination file not opened");
                return 0;
        }


        while((sizer=read(fh_desr,buf,20))>0)
        {

                write(fh_desw,buf,sizer);
        }
        return 0;
}
```

MV.c

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

int main()
{
        int fh_desr=0,fh_desw=0,sizer=0;
        char buf[30];
        fh_desr=open("src1.txt",O_RDONLY);
        if(fh_desr<0)
        {
                printf("Source file not opened");
                return 0;
        }
fh_desw=open("dest1.txt",O_CREAT|O_WRONLY,0777);
        if(fh_desw<0)
        {
                printf("Destination file not opened");
                return 0;
        }


        while((sizer=read(fh_desr,buf,20))>0)
        {
                if(sizer==-1)
                {
                        printf("Error in reading the file");
                        return 0;
                }
                write(fh_desw,buf,sizer);
        }

        unlink("src1.txt");
        return 0;
}
```

2. Create a new file with 0666 access permissions and enable the close-on-exec flag.

```
#include <stdio.h>
#include<stdlib.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<fcntl.h>

int main( )
{

        int fd,old_fd,new_fd;

        fd = open(“close-exec.txt”,O_WRONLY|O_CREAT,0666);
```

```
            old_fd = fcntl(fd,F_GETFD);
            printf(“old fd = %d \n“,old_fd);

            fcntl(fd,F_SETFD,1);

            new_fd = fcntl(fd,F_GETFD);
             printf(“new fd = %d \n“,new_fd);

            close(fd);

}
```

4.Write a C Program to implement a UNIX **‘ls -l’** command using File related API & stat structure.

ls -l.c

```
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<stdio.h>
#include<time.h>
#include<pwd.h>
#include<grp.h>
#include<stdlib.h>

int main()
{
            struct stat st;
            struct passwd *pw;
            struct group *gr;


            if(stat("Text.txt",&st) < 0)
            {
                        printf("stat error");
            }

            if(S_ISREG(st.st_mode))
                        printf("-");
            else if(S_ISDIR(st.st_mode))
                        printf("d");
            else if(S_ISCHR(st.st_mode))
                        printf("c");
            else if(S_ISBLK(st.st_mode))
```

```
                printf("b");
        else if(S_ISFIFO(st.st_mode))
                printf("p");
        else if(S_ISLNK(st.st_mode))
                printf("L");
        else if(S_ISSOCK(st.st_mode))
                printf("s");

        if(S_IRUSR & st.st_mode)
                printf("r");
        else
                printf("-");

        if(S_IWUSR & st.st_mode)
                printf("w");
        else
                printf("-");

        if(S_IXUSR & st.st_mode)
                printf("x");
        else
                printf("-");

        if(S_IRGRP & st.st_mode)
                printf("r");
        else
                printf("-");

        if(S_IWGRP & st.st_mode)
                printf("w");
        else
                printf("-");

        if(S_IXGRP & st.st_mode)
                printf("x");
        else
                printf("-");

        if(S_IROTH & st.st_mode)
                printf("r");
        else
                printf("-");
if(S_IWOTH & st.st_mode)
                printf("w");
```

```
	else
		printf("-");
	if(S_IXOTH & st.st_mode)
		printf("x");
	else
		printf("-");
	printf(" %ld ",(long)st.st_nlink );
	pw = getpwuid(st.st_uid);
	gr = getgrgid(st.st_gid);
	printf(" %s ",pw->pw_name );
	printf(" %s ",gr->gr_name );
	printf(" %lld ",(long long)st.st_size );
	printf(" %s ",ctime(&st.st_ctime));
	printf("\n");
}
```

5. Write a C Program to implement a UNIX ‘ls –ls dir1’ command using directory related system calls.

```
#include <stdio.h>
#include <dirent.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <dirent.h>
#include <stdlib.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>
void printall(struct stat f_stat);
int main()
{
	printf("Give directory name: \n");
	char dname[100];
	scanf("%s",dname);
	DIR *dir=opendir(dname);
	struct dirent* d;
	if(dir)
		while((d=readdir(dir))!=NULL)
		{
			printf("%s ",d->d_name);
switch(d->d_type)
			{
				case DT_UNKNOWN:printf("unknown");	//unknown file type
				break;
```

```
                    case DT_REG:printf("r");            //regular file
                    break;

                    case DT_DIR:printf("d");            //directory file
                    break;

                    case DT_FIFO:printf("f");           //fifo file
                    break;

                    case DT_SOCK:printf("s");           //socket file
                    break;

                    case DT_CHR:printf("c");            //character device
                    break;

                    case DT_BLK:printf("b");            //block device
                    break;

                    case DT_LNK:printf("l");            //symbolic link
                    break;
                }
                struct stat f_stat;
                stat(d->d_name,&f_stat);
                printall(f_stat);               //prints ->(user,group,other permissions),(no of
hard links),(user id),(group id),(size of file),(time of modification)
            }
}
void printall(struct stat f_stat)
{
    char str[10];
    //user
    str[0]=f_stat.st_mode&S_IRUSR?'r':'-';
    str[1]=f_stat.st_mode&S_IWUSR?'w':'-';
    str[2]=f_stat.st_mode&S_IXUSR?'x':'-';
    //group
    str[3]=f_stat.st_mode&S_IRGRP?'r':'-';
    str[4]=f_stat.st_mode&S_IWGRP?'w':'-';
    str[5]=f_stat.st_mode&S_IXGRP?'x':'-';
//others
    str[6]=f_stat.st_mode&S_IROTH?'r':'-';
    str[7]=f_stat.st_mode&S_IWOTH?'w':'-';
    str[8]=f_stat.st_mode&S_IXOTH?'x':'-';
    str[9]='\0';
    printf("%s ",str);
```

```
        //hard link
        printf("%d ",f_stat.st_nlink);
        //group and user id -> include pwd.h and grp.h
        struct passwd *pw=getpwuid(f_stat.st_uid);
        struct group *gr=getgrgid(f_stat.st_gid);
        printf("%s %s ",pw->pw_name,gr->gr_name);
        //size of file
        printf("%d ",f_stat.st_size);
        //time of modification
        printf("%s ",ctime(&f_stat.st_mtime));
}
```

6. Write a C program which creates a child process and the parent waits for child's exit.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>             //for fork(),getpid(),getppid() etc..
#include <sys/wait.h>   //for wait() related function calls..
int main( )
{
        int child_pids[5];
        int i;

        for(i=0;i<5;i++)
        {
                if(fork()==0)
                {
                        printf("child(pid): %d of parent(pid): %d\n",getpid(),getppid());
                        exit(0);
                }
        }

        for(i=0;i<5;i++)
        {
                int cpid=wait(NULL);
                printf("parent (pid): %d waited for child(pid): %d\n",getpid(),cpid);
        }

        return 0;
}
```

7. Write a C program to demonstrate the difference between the fork and vfork system calls.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
```

```
#include <unistd.h>

void forktest( )
{
int a=3,b=2;

        if(fork( )==0)
        {
                a=a+1;
                b=b+1;
                _exit(1);
        }
        int cid=wait(NULL);
        printf("fork: a: %d b: %d\n",a,b);
}

void vforktest( )
{
        int a=3,b=2;

        if(vfork( )==0)
        {
                a=a+1;
                b=b+1;
                _exit(2);
        }
        int cid=wait(NULL);
        printf("vfork: a: %d b:%d\n",a,b);
}
int main( )
{
        forktest( );
        vforktest( );
        return 0;
}
```

8.Write a C program in which main process creates a child process and registers a signal handler to get the exit status of the child asynchronously.

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>             //for raise() signal()
#include <unistd.h>              //for fork()
#include <sys/wait.h>            //wait()
#include <signal.h>
```

```
pid_t cpid;
pid_t ppid;

void my_handler(int signum)
{
	int status;
	waitpid(cpid,&status,0);
	if(WIFEXITED(status))
	{
		int exit_status=WEXITSTATUS(status);
		printf("Exit status of the child was %d from handler!\n",exit_status);
	}
}

int main( )
{
	signal(SIGUSR1, my_handler);
	printf("hi from parent!\n");

	if((cpid=fork())==0)
	{
		printf("hi from child!!\n");
		ppid=getppid();
		kill(ppid,SIGUSR1);
		exit(0);
	}

	else
	{
		wait(NULL);
	}
}
```

9. Implement ‘ ls|wc -l -c –w ’ command using pipe and exec functions.

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

int main()
{
	int pid;
	int fds[2];
	if(pipe(fds) == -1)
	{
		printf("pipe error");
	}

	pid = fork();

	if(pid == 0)
	{
		close(fds[0]);
		dup2(fds[1],STDOUT_FILENO);
		execlp("ls","ls",NULL);
	}

	else
	{
		int status;
		wait(&status);
		close(fds[1]);
		dup2(fds[0],STDIN_FILENO);
		execlp("wc","wc","-l","-c","-w",NULL);
	}
}
```

10. Establish bidirectional communication between sender program and receiver program using multiple FIFO’s.

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

int main( )
{
	int client_to_server;
	char *myfifo = "/tmp/client_to_server_fifo";

	int server_to_client;
	char *myfifo2 = "/tmp/server_to_client_fifo";

	char str[BUFSIZ];
	printf("Input message to server: ");
	scanf("%s", str);
```

```
        /* write str to the FIFO */

        client_to_server = open(myfifo, O_WRONLY);
        server_to_client = open(myfifo2, O_RDONLY);
        write(client_to_server, str, sizeof(str));
        perror("Write:");                   //Very crude error check

        read(server_to_client,str,sizeof(str));
        perror("Read:");                    // Very crude error check
        printf("...received from the server: %s\n",str);
        close(client_to_server);
        close(server_to_client);
        /* remove the FIFO */
        return 0;
}
```

<u>server program</u>

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

int main()
{
        int client_to_server;
        char *myfifo = "/tmp/client_to_server_fifo";

        int server_to_client;
        char *myfifo2 = "/tmp/server_to_client_fifo";

        char buf[BUFSIZ];

        /* create the FIFO (named pipe) */

        mkfifo(myfifo, 0666);
        mkfifo(myfifo2, 0666);

        /* open, read, and display the message from the FIFO */

        client_to_server = open(myfifo, O_RDONLY);
        server_to_client = open(myfifo2, O_WRONLY);
        printf("Server ON.\n");
```

```
	while (1)
	{
		read(client_to_server, buf, BUFSIZ);
		if (strcmp("exit",buf)==0)
		{
			printf("Server OFF.\n");
			break;
		}

		else if (strcmp("",buf)!=0)
		{
			printf("Received: %s\n", buf);
			printf("Sending back...\n");

			write(server_to_client,buf,BUFSIZ);
		}

		/* clean buf from any data */
		memset(buf, 0, sizeof(buf));
	}

	close(client_to_server);
	close(server_to_client);
	unlink(myfifo);
	unlink(myfifo2);
	return 0;
}
```

11. Implement SVR based Message Queue IPC mechanism to establish asynchronous communication between two communicating processes.

receiver.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/msg.h>
#include <string.h>

struct msgbuf
{
	long mtype;
	char msgtxt[100];
};

int main( )
{
        struct msgbuf message;

        int messageid;
        key_t key;

        char msg[]="hello aliens!!";

        //ftok
        if((key=ftok("sender.c",’a’))==-1)
                perror("key generation failed");

        //msgid
        if((messageid=msgget(key,0666|IPC_CREAT))==-1)
                perror("msgid generation error");

        //msgrcv
        if(msgrcv(messageid,&message,sizeof(message),1,0)==-1)
                perror("error in sending message\n");
        else
                printf("received: %s \n",message.msgtxt);
}
```

sender.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/msg.h>
#include <string.h>

struct msgbuf
{
        long mtype;
        char msgtxt[100];
};

int main()
{
        struct msgbuf message;

        int messageid;
        key_t key;

        char msg[]="hello aliens!!";

        //ftok
        if((key=ftok("sender",'a'))==-1)
                perror("key generation failed");

        //msgid
        if((messageid=msgget(key,0666|IPC_CREAT))==-1)
                perror("msgid generation error");

        //msgsnd
        message.mtype=1;
        strcpy(message.msgtxt,msg);

        if(msgsnd(messageid,&message,sizeof(message),0)==-1)
                perror("error in sending message\n");
        else
                printf("sent: %s\n",message.msgtxt);
}
```

12.Implement the following communication model:

- Process 1 creates a Message Queue resource.
- Process 2 enacts the server role
- Process 3 and 4 are clients
- Process 3 seeks 'isprime' service from the server by inserting the payload in the message queue
- Process 4 seeks 'iseven' service form the server by inserting the payload in the message queue
- Server retrieves the service request from the Message queue and inserts the reply
- Intended Client retrieves the response.

process2.c

//12.PROCESS 2 or the server

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/types.h>
```

```
#include <sys/msg.h>
#include <string.h>
#include <unistd.h>                    //fork
#include <signal.h>

int isprime(int payload)
{
        if(payload>10)
                return 1;

        else
                return 0;
}

int iseven(int payload)
{
        if(payload%2==0)
                return 1;
        else
                return 0;
}

struct msgbuf
{
        long mtype;
        char msgtxt[100];
        char from[100];
        char to [100];
        char service[100];
        char reply[100];
        int payload;
};

int main( )
{
        struct msgbuf message;

        int messageid;
        key_t key;

        char msg[]="hello aliens!!";

        //ftok
        if((key=ftok("random",'a'))==-1)
```

```
		perror("key generation failed");

	//msgid
	if((messageid=msgget(key,0666|IPC_CREAT))==-1)
		perror("msgid generation error");

	//msgrcv
	if(msgrcv(messageid,&message,sizeof(message),1,0)==-1)
		perror("error in receiving message\n");

	else
	{

		printf("received: %s \n",message.msgtxt);
		printf("message is from: %s\n",message.from);
		printf("message is for: %s\n",message.to);
		printf("message is for sercvice: %s\n",message.service);

		if(strcmp(message.to,"server")==0)
		{
			strcpy(message.to,message.from);
			strcpy(message.from,"server");

			if(strcmp(message.service,"isprime")==0)
			{
				if(isprime(message.payload))
					strcpy(message.reply,"yes it is prime");
				else
					strcpy(message.reply,"no it is not prime");
			}

			else if(strcmp(message.service,"iseven")==0)
			{
				if(iseven(message.payload))
					strcpy(message.reply,"yes it is even");
				else
					strcpy(message.reply,"no it is not even");
			}

			printf("%s ",message.reply);

		}
	}
}
```

process3.c
//Process 3:static iseven request to process2(server)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/msg.h>
#include <string.h>
#include <sys/wait.h>
#include <unistd.h>
struct msgbuf
{
	long mtype;
	char msgtxt[100];
	char from[100];
	char to[100];
	char service[100];
	char reply[100];
	int payload;
};

int main( )
{
	struct msgbuf message;

	int messageid;
	key_t key;

	char msg[]="hello aliens1!!";

	//ftok
	if((key=ftok("random",'a'))==-1)
		perror("key generation failed");

	//msgid
	if((messageid=msgget(key,0666|IPC_CREAT))==-1)
		perror("msgid generation error");

	//msgsnd

	message.mtype=1;
	strcpy(message.msgtxt,msg);
	strcpy(message.from,"sender1");
	strcpy(message.to,"server");
	strcpy(message.service,"iseven");
	message.payload=25;
	if(msgsnd(messageid,&message,sizeof(message),0)==-1)
		perror("error in sending message\n");
	else
		printf("sent: %s\n",message.msgtxt);
	//sleep(10);

	if(vfork()==0)
	{
		sleep(5);
		execl("./p2",NULL);



	}
	else
	{
		wait(NULL);
	}
}
```

process4.c
//12.Process 4 makes dynamic request between prime or even to server

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/msg.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

struct msgbuf
{
	long mtype;
	char msgtxt[100];
	char from[100];
	char to[100];
	char service[100];
	char reply[100];
	int payload;
};

int main()
{
	struct msgbuf message;

	int messageid;
	key_t key;

	char msg[]="hello aliens!!";
	//ftok
	if((key=ftok("random",'a'))==-1)
		perror("key generation failed");

	//msgid
	if((messageid=msgget(key,0666|IPC_CREAT))==-1)
		perror("msgid generation error");

	//msgsnd

	message.mtype=1;
	strcpy(message.msgtxt,msg);
	strcpy(message.from,"sender");
	strcpy(message.to,"server");

	printf("1.isprime \n2.iseven\n");

	char choice;
	scanf("%c",&choice);

	switch(choice)
	{
		case '1':strcpy(message.service,"isprime");
		break;
		default:strcpy(message.service,"iseven");
		break;
	}

	printf("Give the payload: \n");
	scanf("%d",&message.payload);

	if(msgsnd(messageid,&message,sizeof(message),0)==-1)
		perror("message from client sent");
	else
		printf("request sent to msgque\n");
	if(vfork()==0)
	{
		sleep(5);
		execl("./p2",NULL);
	}
	else
	{
		wait(NULL);
	}
}
```

13. Implement shared Memory based communication model with the following features:
	a) Server and multiple clients communicate with each other through shared memory.
	b) Synchonization of SHM access is realized through semaphores.

PROGRAM:

b) SemShmWriter.C

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<sys/sem.h>
#include<unistd.h>

#define READDATA 0
#define WROTEDATA 1

int  main(void)
{
	key_t  key;
	int  shmid, semid, i ;
	int *pShm = NULL;

	struct sembuf sb;

	key = ftok("SemShmWriter.c" , 'M' );

	/*request the kernel to give block of shm */

	if (   (shmid = shmget ( key , sizeof(int) , IPC_CREAT | 0777 )) < 0 )
```

```
{
        perror("shmget");
        exit(1);
}

/* request the kernel to allot 2 sems */

if( (semid = semget(key , 2, IPC_CREAT | 0777 )) < 0 )
{
        perror("semget");
        exit(1);
}

/* attach the shm object to user addrress space */

if ( (pShm = (int*)shmat ( shmid , NULL , 0)) == NULL )
{
        perror("shmat");
        exit(1);
}

/* initialize READDATA to 1 */

semctl ( semid , READDATA , SETVAL , 1 );

/* write to the shm array*/

for ( i = 0 ; i < 15 ; i ++ )
{

        /*wait until the previous item is read */

        sb.sem_num = READDATA;
        sb.sem_op =  -1 ; /*wait */
        sb.sem_flg = SEM_UNDO;
        semop ( semid , &sb , 1 );

        printf("Wrote %d\n" , *(pShm) = (i+1)*10 );             /*Write to shm object */

        /*inform the reader that new item is written*/

        sb.sem_num = WROTEDATA;
        sb.sem_op =  1 ; /*release  */
        sb.sem_flg = SEM_UNDO;
```

```
                semop ( semid , &sb , 1 );
                sleep(2);
        }

        printf("Wrote the data to shm \n");

        /*detach shm */

        getchar();
        shmdt ( pShm );

        /* remove the IPC object shm */

        shmctl( shmid , IPC_RMID , NULL );
        return 0;

}
```

<u>SemShmReader.C</u>

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<sys/sem.h>
#include<unistd.h>

#define READDATA 0
#define WROTEDATA 1

int  main(void)
{
        key_t  key;
        int  shmid, semid, i ;
        int *pShm = NULL;

        struct sembuf sb;

        key = ftok("SemShmWriter.c" , 'M' );

        /*request the kernel to give block of shm */

        if (   (shmid = shmget ( key , sizeof(int) ,  0777 )) < 0 )
```

```
{
	perror("shmget");
	exit(1);
}

if( (semid = semget(key , 2,  0777 )) < 0 )
{
	perror("semget");
	exit(1);
}

/* attach the shm object to user addrress space */

if ( (pShm = (int*)shmat ( shmid , NULL , 0)) == NULL )
{
	perror("shmat");
	exit(1);
}

/* write to the shm array*/

for ( i = 0 ; i < 15 ; i ++ )
{

	/*wait until the new item is written */

	sb.sem_num = WROTEDATA;
	sb.sem_op =  -1 ; /*wait */
	sb.sem_flg = SEM_UNDO;
	semop ( semid , &sb , 1 );

	printf("Reader read: %d\n" , *(pShm) );  /*Write to shm object */

	/*inform the writer that item is read */

	sb.sem_num = READDATA;
	sb.sem_op =  1 ; /*release  */
	sb.sem_flg = SEM_UNDO;
	semop ( semid , &sb , 1 );

}

printf("Read all the data from shm \n");
```

```
        /*detach shm */

        getchar();
        shmdt ( pShm );

        /* remove the IPC object shm */

        shmctl( shmid , IPC_RMID , NULL );
        semctl(semid , READDATA , IPC_RMID, NULL );
        semctl(semid , WROTEDATA , IPC_RMID , NULL );
        return 0;

}
```

14.Implement client/server model using socket API.

<u>tcpserver.C</u>

```
#include<stdio.h>
#include<arpa/inet.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<unistd.h>
#include<signal.h>
#include<sys/wait.h>
#include<stdlib.h>

int main()
{
        int csfd,dsfd,i,j;
        pid_t pid;
        socklen_t size;
        char msg[10];
        struct sockaddr_in serversoc,clientsoc;


        csfd=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);

        serversoc.sin_family=AF_INET;
        serversoc.sin_port=htons(5001);
        serversoc.sin_addr.s_addr=htonl(INADDR_ANY);

        bind(csfd,(struct sockaddr *)&serversoc,sizeof(serversoc));
        listen(csfd,10);
        dsfd=accept(csfd,(struct sockaddr *)&clientsoc,&size);


        recv(dsfd,msg,15,0);
        printf("%s",msg);
        send(dsfd,"SERVER: Hai",15,0);

        close(dsfd);

}
```

<u>tcpclient.C</u>

```
#include<stdio.h>
#include<arpa/inet.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<unistd.h>
int main()
{
int csfd,dsfd,i,j;
socklen_t size;
struct sockaddr_in serversoc,clientsoc;
char msg[10];

csfd=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);

serversoc.sin_family=AF_INET;
serversoc.sin_port=htons(5001);
serversoc.sin_addr.s_addr=htonl(INADDR_ANY);

connect(csfd,(struct sockaddr *)&serversoc,sizeof(serversoc));
send(csfd,"CLIENT: Hello",15,0);
recv(csfd,msg,15,0);
printf("%s",msg);
close(csfd);
}
```

15. Implement concurrent server using fork based model while avoiding the zombie state of the client.

Conncurrent_server_fork.c

```
#include<stdio.h>
#include<sys/socket.h>
```

```
#include<netinet/in.h>
#include<arpa/inet.h>
#include<signal.h>
#include<sys/wait.h>
#include<string.h>
#include<unistd.h>
int main()
{
int size,size1,sockfd,connfd;
char buf[20];
int i;
sockfd=socket(AF_INET,SOCK_STREAM,0);
struct sockaddr_in ser,cl;
ser.sin_family=AF_INET;
ser.sin_port=htons(10001);
ser.sin_addr.s_addr=htonl(INADDR_ANY);
size=sizeof(ser);
i=bind(sockfd,(struct sockaddr*)&ser,size);
if(i==0)
{
listen(sockfd,5);
size1=sizeof(cl);
while((connfd=accept(sockfd,(struct sockaddr*)&cl,&size1)))
{
int pid;
pid=fork();
if(pid==0)
{
recv(connfd,buf,20,0);
printf("message=%s\n",buf);
}
else if(pid>0)
{
for(;;)
pause();
}
}
}
}
```

tcpclient.C

```
#include<stdio.h>
#include<arpa/inet.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<unistd.h>
int main()
{
        int csfd,dsfd,i,j;
        socklen_t size;
```

```
	struct sockaddr_in serversoc,clientsoc;
	char msg[10];

	csfd=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);

	serversoc.sin_family=AF_INET;
	serversoc.sin_port=htons(4005);
	serversoc.sin_addr.s_addr=htonl(INADDR_ANY);

	connect(csfd,(struct sockaddr *)&serversoc,sizeof(serversoc));
	send(csfd,"CLIENT:HAI",10,0);
	recv(csfd,msg,10,0);
	printf("%s",msg);

	//close(csfd);

}
```

16. Implement a concurrent server model using pthread API

```
#include<stdio.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<netinet/in.h>
#include<signal.h>
#include<sys/wait.h>
#include<pthread.h>
#include<stdlib.h>
void *doit(void *);
int main()
{
int a,size,fd,dsfd,i,pid;
int* new_sock;
fd=socket(AF_INET,SOCK_STREAM,0);
struct sockaddr_in ser,cl;
ser.sin_family=AF_INET;
ser.sin_port=htons(3017);
ser.sin_addr.s_addr=htonl(INADDR_ANY);
i=bind(fd,(struct sockaddr*)&ser,sizeof(ser));
if(i==0)
{
listen(fd,5);
size=sizeof(cl);
while((a=accept(fd,(struct sockaddr*)&cl,&size)))
{
pthread_t id;
new_sock=malloc(1);
*new_sock=a;
pthread_create(&id,NULL,doit,(void *)new_sock);
pthread_join(id,NULL);
//pthread_exit(NULL);
}
return 0;
}
}
void* doit(void* a)
{
int dsfd;
dsfd=*((int*)a);
char buf[20];
recv(dsfd,buf,sizeof(buf),0);
write(1,buf,sizeof(buf));
//pthread_exit(NULL);
}
```

17.Solve the producer consumer problem using pthread API.

PROGRAM

Procon.c

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


int buf[50];
int num=0;
int i=0;
int front=0,rear=0;
pthread_mutex_t m=PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cv1=PTHREAD_COND_INITIALIZER;
pthread_cond_t cv2=PTHREAD_COND_INITIALIZER;

void insert()
{
        buf[num]=i++;
        printf("produced: %d  ",i);
        rear=(rear+1)%50;
        num=num+1;
}

void delete()
{
        int d=buf[front];
        front=(front+1)%50;
        printf("consumed: %d  ",d);
```

```
            num-=1;
return ;
}

void * producer(void *arg)
{
            int j;

for(j=0;j<500;j++)
            {
            pthread_mutex_lock(&m);
            while(num==50)
            pthread_cond_wait(&cv1,&m);
            insert(i);
            pthread_cond_signal(&cv2);
            pthread_mutex_unlock(&m);

            }
}

void * consumer(void *arg)
{
            int i;
            for(i=0;i<500;i++)
            {
            pthread_mutex_lock(&m);
            while(num==0)
            pthread_cond_wait(&cv2,&m);
            delete();
            pthread_cond_signal(&cv1);
            pthread_mutex_unlock(&m);

            }

}

int main()
{           pthread_t ct,pt;
            void *ptr;
            void *ret;
            pthread_create(&pt,NULL,producer,ptr);
            pthread_create(&ct,NULL,consumer,ptr);
            pthread_join(pt,&ret);
            pthread_join(ct,&ret);
```

```
return 0;

}
```

18. Implement peer-to-peer communication model using socket API.

PROGRAM

Udpserver.c

```
#include<sys/socket.h>
#include<arpa/inet.h>
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/types.h>
#include<string.h>

int main() {
  int sfd, cfd;
  char buf[1024];
  struct sockaddr_in server, client;
  sfd = socket(AF_INET, SOCK_DGRAM, 0);

  server.sin_family = AF_INET;
  server.sin_port = htons(5007);
  inet_aton("127.0.0.1", &server.sin_addr);
  int b = bind(sfd, (struct sockaddr *) &server, sizeof (server));
  printf("BIND VALUE:%d\n", b);
  int l = sizeof (client);

    recvfrom(sfd, buf, 1024, 0, (struct sockaddr *) &client, &l);

    sendto(sfd, buf, sizeof(buf), 0, (struct sockaddr *) &client, sizeof (client));
    printf("MESSAGE FROM CLIENT:%s\n", buf);

  close(sfd);
}
```

Udpclient.c

```
#include<sys/socket.h>
#include<arpa/inet.h>
#include<stdio.h>
```

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

int main() {
   int sfd, cfd;
   char buf[1024];
   struct sockaddr_in server, client;
   sfd = socket(AF_INET, SOCK_DGRAM, 0);

   server.sin_family = AF_INET;
   server.sin_port = htons(5007);
   inet_aton("127.0.0.1", &server.sin_addr);

   int l = sizeof (client);

      printf("ENTER string\n");
      scanf("%s", buf);
      sendto(sfd, buf, sizeof(buf), 0, (struct sockaddr *) &server, sizeof (server));
      recvfrom(sfd, buf, 1024, 0, NULL, NULL);
      printf("RECEIVED FROM SERVER:%s\n", buf);

   close(sfd);
}
```

19. Solve the process synchronization on I/O using record locking mechanism.

PROGRAM

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

int main(int args,char* argv[])
{
        struct flock fl = { F_UNLCK,SEEK_SET,0,100,0};
        int fsize,offset,fd,pid;
        char buf[50];

        fd = open(argv[1],O_RDWR);
        printf("File is not locked by any process\n");
        printf("Press <Enter> to lock the file\n");
        printf("********************************");
```

```
	getchar();
	fl.l_type = F_WRLCK;
	fl.l_pid = getpid();
	if(fcntl(fd,F_SETLK,&fl) == -1)
	{
		printf("cannot set exclusive lock\n");
		exit(1);
	}
	else if(fl.l_type != F_UNLCK && fl.l_type != F_RDLCK)
		printf("file has been exclusively locked by process: %d\n",fl.l_pid);
	else
		printf("file is not locked\n");

	printf("Press <Enter> to Release the lock\n");

	getchar();

	fl.l_type =  F_UNLCK;
	printf("File has been Unlocked \n");

	// finding the offset value of last  50 bytes of data

	fsize = lseek(fd,0,SEEK_END);
	offset = fsize - 50;

	//setting the cursor to the offset found.

	lseek(fd,offset,SEEK_SET);

	// reading the last 50 bytes of data and printing

	read(fd,buf,50);
	printf("last  50 bytes of data in the file is: \n");
	printf("*******************************************");
	printf("%s \n",buf);
	return 0;
}
```

20. Implement I/O multiplexing using select system call.

PROGRAM
iomutiplexingserver.c

```
#include <sys/socket.h>
#include <sys/select.h>
```

```
#include <sys/types.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
int main()
{
            int fd,i;
            struct sockaddr_in server;
            fd_set fdset,readset;

            fd=socket(AF_INET,SOCK_STREAM,0);
            server.sin_family=AF_INET;
            server.sin_port=htons(4021);
            server.sin_addr.s_addr=inet_addr("127.0.0.1");
            bind(fd,(struct sockaddr*)&server,sizeof(server));
            int var;

            listen(fd,var);
            for(;;)
            {

            FD_SET(fd,&fdset);
            struct timeval a;
            a.tv_sec=5;
            a.tv_usec=0;
            readset=fdset;
            select(FD_SETSIZE,&readset,NULL,NULL,&a);
            for(i=0;i<FD_SETSIZE;i++)
            {
                  if(FD_ISSET(fd,&readset))
                  {
                        struct sockaddr_in client;
                        int size=sizeof(server);
                   int dfd=accept(fd,(struct sockaddr*)&client,&size);
                   char a[100];
                   if((i=recv(dfd,a,12,0))<0)
                         perror("rec error\n");
                   printf("%s\n",a);
                  }
            }
            }
            return 0;
}
```

<u>iomutiplexingclient.c</u>

```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
            int fd;
            struct sockaddr_in server;
            fd=socket(AF_INET,SOCK_STREAM,0);
            server.sin_family=AF_INET;
            server.sin_port=htons(4021);
            server.sin_addr.s_addr=inet_addr("127.0.0.1");
            connect(fd,(struct sockaddr *)&server,sizeof(struct sockaddr));
            char * a=(char *)malloc(10*sizeof(char));
            strcpy(a,"hello world");
            //char a[]="hello world";
        send(fd,a,strlen(a),0);
            printf("done");
            return 0;

}
```