1) Producer's consumer's

```
#include<stdio.h>
#include<stdlib.h>
int full = 0, empty, n, buffer[20], item;
int in = 0, out = 0, mutex = 1;
void wait(int s){
    while (s<0)
    {
        printf("\nCannot add item\n");
        exit(0);
    }
    s--;
}
void signal(int s){
    s++;
}
void producer(){
    do
    {
        wait(empty);
        wait(mutex);
        printf("\nEnter an item: ");
        scanf("%d",&item);
```

```
24)             buffer[in] = item;
25)             in = in + 1;
26)             signal(mutex);
27)             signal(full);
28)         } while (in<n);
29)   }
30)   void consumer(){
31)         do
32)         {
33)             wait(full);
34)             wait(mutex);
35)             item = buffer[out];
36)             printf("\nItem consumed:
  %d",item);
37)             out = out + 1;
38)             signal(mutex);
39)             signal(empty);
40)         } while (out<n);
41)   }
42)   void main(){
43)         printf("Enter the value of n:
  ");
44)         scanf("%d",&n);
45)         empty = n;
```

```
46)       while (in<n)
47)       {
48)           producer();
49)       }
50)       while (out!=n)
51)       {
52)           consumer();
53)       }
54)   }
```

2) Reader's writer's

```
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
sem_t mutex, writeblock;
int data = 0, rcount = 0;
void *reader(void *arg){
    int f = ((int)arg);
    sem_wait(&mutex);
    rcount++;
    if (rcount==1)
        sem_wait(&writeblock);
    sem_post(&mutex);
```

```
    printf("\nData read by the reader %d is
%d\n", f, data);
    sleep(1);
    sem_wait(&mutex);
    rcount--;
    if (rcount==0)
        sem_post(&writeblock);
    sem_post(&mutex);
}
void *writer(void *arg){
    int f = ((int)arg);
    sem_wait(&writeblock);
    data++;
    printf("\nData written by writer %d is
%d\n", f, data);
    sleep(1);
    sem_post(&writeblock);
}
void main(){
    int i;
    pthread_t rtid[5], wtid[5];
    sem_init(&mutex, 0, 1);
    sem_init(&writeblock, 0, 1);
    for (i = 0; i <= 3; i++)
```

```
    {
        pthread_create(&wtid[i], NULL,
writer, (void*)i);
        pthread_create(&rtid[i], NULL,
reader, (void*)i);
    }
    for (i = 0; i <= 3; i++)
    {
        pthread_join(wtid[i], NULL);
        pthread_join(rtid[i], NULL);
    }
}
```

3) priority scheduling

```
#include <stdio.h>
void main()
{
    int i, j, n, t, turn[20], burst[20],
p[20], wt[20], c[20];
    float await, aturn, twait = 0, tturn =
0;
    printf("\nEnter the value of n:");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
```

```
    {
        printf("\n Enter the process no
burst and arrivaltime");
        scanf("%d", &c[i]);
        scanf("%d", &burst[i]);
        scanf("%d", &p[i]);
    }
    for (i = 0; i < n; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (p[i] > p[j])
            {
                t = p[i];
                p[i] = p[j];
                p[j] = t;
                t = burst[i];
                burst[i] = burst[j];
                burst[j] = t;
                t = c[i];
                c[i] = c[j];
                c[j] = t;
            }
        }
```

```
    }
    for (i = 0; i < n; i++)
    {
        if (i == 0)
        {
            wt[i] = 0;
            turn[i] = burst[i];
            tturn = tturn+turn[i];
        }

        else
        {
            turn[i] = turn[i - 1] +
burst[i];
            wt[i] = turn[i] - burst[i];
            twait = twait + wt[i];
            tturn = tturn + turn[i];
        }
    }
    await = twait / n;
    aturn = tturn / n;
    printf("pno\tbtime\tatime\twtime\tttime
");
    for (i = 0; i < n; i++)
```

```
    {
        printf("\n%d\t%d\t%d\t%d\t%d\n",
c[i], burst[i], p[i], wt[i], turn[i]);
    }
    printf("\n The average waiting time
is:%f", await);
    printf("\n The average turn around time
is:%f", aturn);
}
```

4) IPC

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void main(){
    int childid, fd[2], uid;
    char string[40];
    char buffer[40];
    pipe(fd);
    if((childid=fork())<0){
        printf("Error");
        exit(0);
```

```
    }
    if (childid==0)
    {
        close(fd[0]);
        printf("\nEnter string: ");
        gets(string);
        printf("\nChild process sends the
string %s", string);
        write(fd[1], string,
strlen(string)+1);
    }
    else{
        close(fd[1]);
        printf("\nParent Process");
        uid = read(fd[0], buffer,
sizeof(buffer));
        printf("\nParent process receives
the string %s", buffer);
    }
}
```

5) Environmental variables

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

```c
int main(int argc, char *args[], char
*env[]){
    int i;
    for (i = 0; env[i]!= NULL; i++)
    {
        printf("\n%s",env[i]);
    }
    return 0;
}
```

6) Dining philosopher

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/sem.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
int semid, shmid;
char *buf;
void dinphil(int);
void feelhungry(int);
void main()
{
```

```
    int i, pid;
    semid = semget((key_t)0x153, 1,
IPC_CREAT | 0600);
    shmid = shmget((key_t)0x133, 100,
IPC_CREAT | 0600);
    buf = (char *)shmat(shmid, NULL, 0);
    semctl(semid, 0, SETVAL, 2); // at a
time 2 processes can eat
    for (i = 0; i < 5; ++i)
        buf[i] = 0; // all the forks are
empty

    pid = fork();
    if (pid == 0)
    {
        pid = fork();
        if (pid == 0)
        {
            pid = fork();
            if (pid == 0)
            {
                pid = fork();
                if (pid == 0)
                    dinphil(3);
```

```
                else
                    dinphil(2);
            }
            else
                dinphil(1);
        }
        else
            dinphil(0);
    }
    semctl(semid, 0, IPC_RMID);
    shmctl(shmid, IPC_RMID, 0);
    printf("\n");
}

void dinphil(int n)

{
    int i;
    printf("\nPhilosopher  %d entered\n",
n);
    for (i = 0; i < 2; ++i)
    {
        printf("\nPhilosopher %d is
thinking....", n);
```

```
        sleep(1);
        feelhungry(n);
    }
    printf("\nPhilosopher %d is died...",
n);
}

void feelhungry(int n)
{
    struct sembuf sop;
    printf("\nPhilosopher %d is feeling
hungry..\n", n);
    sop.sem_num = 0; // first semaphore
    sop.sem_op = -1; // request for resouce
    sop.sem_flg = 0; // non blocking
    semop(semid, &sop, 1);
    if (buf[n] == 0 && buf[(n + 1) % 5] ==
0)
    {
        buf[n] = 1;     // using left fork
        buf[n + 1] = 1; // using right fork
        printf("\nPhilosopher %d is
eating...", n);
        sleep(3);
```

```
        printf("\nPhilosopher %d is
finished eating..", n);
        buf[n] = 0;             // free
        buf[(n + 1) % 5] = 0; // free
    }
    sop.sem_num = 0;
    sop.sem_op = 1; // release the resource
                    // sop.sem_flg=0;
                    //           //
semop(semid,&sop,1)
}
```

7) Echo server using pipes

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#define max 256
void main()
{

    int p1[2],p2[2],pid,n,i;
    char msg[max];
```

```
    pipe(p1);
    pipe(p2);
    pid=fork();
    if(pid==0)
    {
        close(p2[0]);
        close(p1[1]);
    for(i=1;i<=5;++i)
                {
    printf("\nEnter any msg:  ");
  fflush(stdout);
    scanf("%s",msg);
        write(p2[1],&msg,sizeof(msg));
           n=read(p1[0],&msg,max);
        printf("Server echoed back:
%s\n",msg);
                }
    }
    else
    {
        close(p1[0]);
        close(p2[1]);
    for(i=1;i<=5;++i)
                {
```

```
                        n=read(p2[0],&msg,max
);
        printf("\nServer received from
client is: %s\n",msg);
        write(p1[1],&msg,n);
                }
    }
}
```

8) home DIRECTORY

```
#include <stdio.h>
#include <stdlib.h>
extern char **environ;
int main()
{
    char *home;
    if ((home = getenv("HOME")) == NULL)
        printf("home is not defined");
    else
        printf("the value of HOME is :%s",
home);
}
```

9) Printing password

```
#include <pwd.h>
#include <stdio.h>
#include <sys/types.h>
main()
{
    struct passwd *ptr;
    int uid, gid;
    uid = getuid();
    gid = getgid();
    printf("\nUser ID and Group ID using
getuid(),getgid()");
    printf("\nUser ID = %d", uid);
    printf("\nGroup ID = %d", gid);
    printf("\nPassword information using
getpwuid()");
    ptr = getpwuid(uid);
    printf("\nUsername = %s\nPassword =
%s", ptr->pw_name, ptr->pw_passwd);
    printf("\nUser ID = %d\nGroup ID
=  %d", ptr->pw_uid, ptr->pw_gid);
    printf("\nHome Directory = %s\nShell =
%s", ptr->pw_dir, ptr->pw_shell);
```

```
    printf("\nPassword information using
getpwnam()");
    ptr = getpwnam("mca02-31");
    printf("\nUsername = %s\nPassword =
%s", ptr->pw_name, ptr->pw_passwd);
    printf("\nUser ID = %d\nGroup ID
=  %d", ptr->pw_uid, ptr->pw_gid);
    printf("\nHome Directory = %s\nShell =
%s", ptr->pw_dir, ptr->pw_shell);
}
```

10) Echo server message using message queue

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

struct
{
    long mtype;
```

```
    char mtext[512];
} send, recv;

main()
{
    int msqid, pid, i, n, n1;
    printf("how many message u t enter:");
    scanf("%d", &n1);
    msqid = msgget((key_t)123456, IPC_CREAT
| 0700);
    if (msqid < 0)
    {
        printf("msgget error: ");
        exit(1);
    }
    pid = fork();
    if (pid == 0)
    {
        for (i = 1; i <= n1; ++i)
        {
            printf("\nEnter a message:");
            fflush(stdin);
            scanf("%s", send.mtext);
            send.mtype = 1;
```

```
            msgsnd(msqid, &send,
strlen(send.mtext), 0);
            n = msgrcv(msqid, &recv, 512,
2, 0);
            recv.mtext[n] = '\0';
            printf("Message echoed: %s\n",
recv.mtext);
        }
        msgctl(msqid, IPC_RMID, 0);
    }
    else
    {
        for (i = 1; i <= n1; ++i)
        {
            n = msgrcv(msqid, &recv, 512,
1, 0);
            send.mtype = 2;
            strcpy(send.mtext, recv.mtext);
            msgsnd(msqid, &send, n, 0);
        }
    }
}
```

11) Copy file content from one file to another

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

int main(int argc,char *argv[])
{
    int fd1,fd2;
    char ch;
    if(argc!=3)
    {
        printf("\nUsage: mycp<src
file><dest file>");
        return;
    }
    fd1=open(argv[1],O_RDONLY);
    if(fd1<0)
    {
        perror(" ");
    return;
    }
          fd2=open(argv[2],O_WRONLY|O_CREAT
,0777);
```

```
    if(fd2<0)
    {
        perror(" ");
        return;
    }
    while(read(fd1,&ch,1)==1)
        write(fd2,&ch,1);
    close(fd1);
    close(fd2);
}
```

12) Create file

```
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
#include<string.h>
main()
{
    int fd,n,m;
    char buf1[100],buf2[100];
```

```
    fd=open("v.txt",O_WRONLY|O_CREAT,0600);

    printf("enter a string:");
    fflush(stdin);
    scanf("%s",buf1);
    n=write(fd,buf1,strlen(buf1));
    printf("\n no of bytes written: %d",n);
    close(fd);
    fd=open("v.txt",O_RDONLY);
    m=read(fd,buf2,n);
    printf("\n no of bytes read:%d",m);
    buf2[n]='\0';
    printf("\n readed string :%s\n",buf2);
    close(fd);
}
```