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

int main()
{

  pid_t pid;
  pid=fork();//above code is same for child process too
  if(pid==0)
  {
    printf("child process created\n");
   }
   else if(pid>0)
   {
     printf("parent process\n");
   }
   }
```

.....

2)write a program to return fork value of parent and child

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

int main()
{
  int i;
  pid_t pid;
  pid=fork();
  if(pid==0)
  {
    printf("child process created \n");
    printf("%d fork value",pid);
    }
  else if(pid>0)
    {
       printf("parent created \n");
        printf("%d is fork  value\n",pid);
       }
 }
```

.....

4)write a program to print pid of the process child and parent

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

```
int main()
{
  pid_t pid;
  pid=fork();
  if(pid==0)
  {
    printf("child process\n");
    printf("%d pid of process\n",getpid());
  }
  else if(pid>0)
  {
     printf("parent process\n");
    printf("%d pid of process\n",getpid());
  }
  }
 ........................................
```

3)write a program to create a new process and make the parent wait for the child

```
#include<stdio.h>

#include<sys/types.h>

#include<sys/wait.h>

#include<unistd.h>

#include<stdlib.h>


int main()

{

     int child=fork();

     int exitStatus;

  if(child==0)

     {

         printf("child: iam running\n");

         printf("child: i have PD :%d\n",getpid());//return the process id

          sleep(4);

       exit(100);

     }//code for child process

  else
```

```
     {

          printf("Parent : iam running and waiting for child to
finish\n");

          int childPid=wait(&exitStatus);

          printf("parent : Child finished execution,it had the PID:
%d\n",childPid);

  }

}
```

.....

5)write program to create a new process and display process id of parent process of both child and parent of parent process

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

int main()
{
  pid_t p, pid,ppid;
  p=fork();

  if(p==0)
  {
    printf("\nchild process created");
    pid=getpid();
    printf(":\npid of child process is %d",pid);
    ppid=getppid();
    printf(":\npid of parent of child process is %d",ppid);
    printf("\n child terminated\n");
  }
  if(p>0)
  {
    printf("\n parent process created");
    pid=getpid();
    printf(":\npid of parent process is %d",pid);
    ppid=getppid();
    printf(":\npid of parent of parent process is %d",ppid);
    wait(NULL);
    printf("\n parent terminated");
  }
  return 0;
  }
```

.....

6)write a program to create an orpahn process

```
//sleep in child
#include<stdio.h>
#include<stdlib.h>
```

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

int main()
{
  pid_t p;
  p=fork();

  if(p==0)
  {
    printf("child process crested\n");
    sleep(10);
    printf("child process terminated\n");
  }
  if(p>0)
  {
    printf("parent process created\n");
    printf("parent process terminated\n");
    }
    }
```

.....

7) write a program to create zombie process(runs in the background and not available to interact with the user

```
//sleep in parent
//exit(0) in child
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>

int main()
{
  pid_t p;
  p=fork();

  if(p>0)
 {
   printf("parent created\n");
   sleep(10);
   }
   else
   {
     exit(0);
     }

     }
```

.....

8) write a program to create new process and make child different from the parent hint execlp sys call

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

```
#include<unistd.h>
#include<sys/wait.h>

int main()
{
  pid_t p;
  p=fork();
  if(p>0)
  {
    printf("parent process created\n");
    wait(NULL);
    printf("\n parent terminated\n");
    }
    if(p==0)
    {
      execlp("/bin/ls","ls",NULL);
      }
      }
```

.....

9) write a program to illustarte thread concept

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

void* th(void*); //code for thread

int main()
{
   pthread_t tid;//type pthread_t
   int t;
   printf("\n main thread\n");//started
   printf("\nmain thread calling for child thread");

   t=pthread_create(&tid,NULL,th,NULL);
//return value 0 for success  2nd para attribute of a thread
//3rd para code of our thread
// pass anything to thread in 4th parameter

   pthread_join(tid,NULL); //join a calling thread as wait for child
thread to terminate null child thread can pass value to aparent
thread
   //chlid has been tarminated
   printf("\nMain terminated\n");
   printf("%d\n",t);

   return 0;
}
 void* th(void *y) //actual code
 {
   printf("\nchild  thread created");
   printf("\n  child thread terminated");
 }
```

.....

10) write a program illustrate sharing of variables among thraeds

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

void* th(void*);

int i ;// i belongs to process
int main()
{
 pthread_t th1,th2; //2 threads created

 pthread_create(&th1,NULL,th,NULL);
 pthread_create(&th2,NULL,th, NULL);
 pthread_join(th1,NULL);
 pthread_join(th2,NULL);

 }

 void* th(void* p)
 {
   ++i;
   printf("\nthread %d",i);
    pthread_exit(NULL);
  }
```

.....

11) write a program to illustrate thread cancellation

```
#include <stdio.h>

#include <pthread.h>

#include <unistd.h>



void* th(void*);


int main()

{

    pthread_t th1;

    printf("main thread\n");

    pthread_create(&th1,NULL,th,NULL);

    sleep(2);

    printf("terminating thread\n");
```

```
    pthread_cancel(th1);
    printf("terminated\n");
    pthread_join(th1,NULL);
}
void* th(void*)
{
     while(1)
       {
           printf("child running\n");
           sleep(1);
       }

}
```

.....

12) write a program to illustrate IPC using a pipe

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

int main()
   {
       int a;
       pid_t parent;
       char c[20];
       int fd[2];
       a=pipe(fd);
        if(a==-1)
          {
```

```
              printf("cannot create pip\n");

     }
      parent=fork();

       if(parent)
         {
              write(fd[1],"good",strlen("good")+1);
          }
      else
        {
             read(fd[0],c,sizeof(c));
            printf("%s read by chlidd\n",c);

        }


}
```

.....

13) write a program to illustrate IPC shared memory

```
//client pro for shared memory
#include<unistd.h>
#include<stdio.h>
#include<sys/shm.h>
#include<sys/ipc.h>
#include<sys/types.h>
#include<stdlib.h>

int main()
{
   key_t key1=4321;
   size_t size=28;
   int i;
   char *shm,*s;
   int shmid;
   char y;

    shmid =shmget(key1,size,IPC_CREAT|0666);
  shm=shmat(shmid,NULL,0);

  if(shmid<0||shm<0)
  {
```

```
    printf("\n Can't create or attach \n");
    exit(1);
  }
  else
   {
       printf("%s",shm);
       *shm='*';
       }
       return 0;
   }
 //shmat return starting loc
*****************************************
//server part for shared memory
#include<stdio.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<sys/types.h>
#include<unistd.h>

int main()
{
  key_t key1=4321;
  size_t size=28;
  int i;
  char *shm,*s;
  int shmid;
  char y;

  shmid =shmget(key1,size,IPC_CREAT|0666);
  shm=shmat(shmid,NULL,0);

  if(shmid<0||shm<0)
  {
    printf("\n Can't create or attach \n");
    exit(1);
  }
  else
   {
     s=shm;
     for(i=0;i<25;i++)
     *shm++= (char) (65+i);
     *shm =NULL;

      while(*s!='*')
       {
       sleep(0);
       }
       return 0;
   }
 }
.....................................................................
.....
14)write a program to illustrate IPC using Message Queues
```

```
//msg queue reciver program
//read and puts into &m.mtext msg is stored
//-o is uesd to rename
//cc msgrcv.c -o msgrcv -lrt ( l means link)
//./msgrcv
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/msg.h>

struct msgbuf
{
  long mtype;
  char mtext[20];
  };
  int main()
 {
   struct msgbuf m;
   int msgid=msgget(189,IPC_CREAT|0666);
   m.mtype=1;
   msgrcv(msgid,&m,20,1,0);
   printf("the msg is :%s,\n %d",m.mtext,msgid);
   }
**********************************************************
// cc -o msgsend msgsend.c -lrt
//./msgsend
// sender program
#include<stdio.h>
#include<sys/msg.h>
#include<sys/ipc.h>
struct msgbuf
{
 long mtype;
 char mtext[20];
};
int main()
{
  struct msgbuf m;
  int msgid=msgget(189,IPC_CREAT|0666);
  m.mtype=1;
  printf("enter a msg");
  //gets(m.mtext);
  scanf("%s",m.mtext);
  msgsnd(msgid,&m,20,0);
}
.....................................................................
......
15)Write a program to illustarte process scheduling FCFS algorithm

//fcfs
#include<stdio.h>


int main()
```

```
{
    int i,n,bt[20],wt[20],tat[20];
  float totwt=0,totat=0;

  printf("enter no of process:");
       scanf("%d",&n);

  for(i=0;i<n;i++)
  {
          printf("\nenter brust time for p%d: ",i);
      scanf("%d",&bt[i]);
    }

  wt[0]=0;

  for(i=1;i<n;i++)
  {
         wt[i]=wt[i-1]+bt[i-1];
  }

  for(i=0;i<n;i++)
     {
      tat[i]=bt[i]+wt[i];
      totwt +=wt[i];
        totat +=tat[i];
    }
  printf("process bt wt tat\n");
  for( i=0;i<n;i++)
      {
```

```
            printf("P%d\t\t%d\t%d\t%d\n",i,bt[i],wt[i],tat[i]);

        }

           printf("\nAverage Waiting Time = %.2f", totwt / n);

           printf("\nAverage Turnaround Time = %.2f\n", totat / n);
 }
 ..............................................................
.
16)write a program to illustarte process scheduling SJF algorithm
//sjf
#include<stdio.h>
int n,pid[9],bt[9];

int wt[9],twt,tat[9],ttat;
float awt,atat;

      int main()
{
   printf("\n *** Shortest Job First *** \n");
   getdata(); //to collect inputs from user
   sort();
   wait(); //to calculate waiting times
   turn();//to calculate turn around times
   display();// to print calculations table & Gantt Chart

 }
 getdata()
 {
    printf("\n ENter the number of processes =");
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
      printf("\n Enter Brust Time of Pid-%d =",i);
      pid[i]=i;
      scanf("%d",&bt[i]);
    }
}
sort()
{
  int temp;
  for (int i=1;i<=n;i++)
  {
    for(int j=i+1;j<=n;j++)
    {
      if(bt[i]>bt[j])
      {
        temp=bt[i];
        bt[i]=bt[j];
        bt[j]=temp;

        temp=pid[i];
        pid[i]=pid[j];
```

```
        pid[j]=temp;
      }
    }
  }
}

wait()
{
  wt[1]=0; twt=0;
  for(int i=2;i<=n;i++)
  {
   wt[i]=wt[i-1]+bt[i-1];
   twt+=wt[i];
   }
   awt=(float)twt/n;
}

turn()
{
  ttat=0;
  for(int i=1;i<=n;i++)
  {
    tat[i]=wt[i]+bt[i];
    ttat+=tat[i];
  }
  atat=(float)ttat/n;
}
display()
{
  printf("\n PID \t BT \t WT \t TAT");
  for(int i=1;i<=n;i++)
  {
     printf("\n %d \t %d \t %d \t %d",pid[i],bt[i],wt[i],tat[i]);
  }
 printf("\n\n Total WT= %d",twt);
 printf("\n Avg WT =%f",awt);
 printf("\n\n Total TAT =%d ",ttat);
 printf("\n Avg .TAT =%f",atat);

 printf("\n\n 0");//Gantt Chart
 for(int i=1;i<=n;i++)
 {
   printf(" <P%d> %d",pid[i],tat[i]);
   printf("\n\n");
  }
 }
.....................................................
17)//producerconsumer

#include<unistd.h>
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
sem_t b,empty,full;
```

```
int buf=0;
void* prod()
{
  while(1)
  {
     sem_wait(&empty);
     sem_wait(&b);
     buf++;
     printf("produced buf=%d\n",buf);
     sleep(1);
     sem_post(&b);
     sem_post(&full);
     if(buf==5)
     sleep(20);
  }
}
void* cons()
{
 while(1)
 {
   sleep(5);
   sem_wait(&full);
   sem_wait(&b);
   buf--;
   printf("\nconsumed buf=%d",buf);
   sleep(1);
   sem_post(&b);
   sem_post(&empty);
  }
}
void main()
{
  pthread_t p,c;
  sem_init(&empty,0,5);
  sem_init(&full,0,0);
  sem_init(&b,0,1);
  pthread_create(&p,NULL,prod,NULL);
  pthread_create(&c,NULL,cons,NULL);
  pthread_join(p,NULL);
  pthread_join(c,NULL);
}
```

.....

18) program for //readers writers program

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

// Shared data
int data = 0;  // The data that readers and writers will access

// Semaphores
```

```
sem_t mutex;  // Mutex for protecting reader count
sem_t write_lock;  // Semaphore to allow only one writer at a time
int reader_count = 0;  // Number of active readers

// Reader function
void* reader(void* arg) {
    // Reader tries to read the data
    sem_wait(&mutex);  // Entry section for readers
    reader_count++;
    if (reader_count == 1) {
        sem_wait(&write_lock);  // Block writers if the first reader
enters
    }
    sem_post(&mutex);  // Exit section for readers

    // Reading the data
    printf("Reader %ld: Reading data = %d\n", (long)arg, data);
    sleep(1);  // Simulate reading

    sem_wait(&mutex);  // Entry section for readers
    reader_count--;
    if (reader_count == 0) {
        sem_post(&write_lock);  // Allow writers when no readers are
left
    }
    sem_post(&mutex);  // Exit section for readers

    return NULL;
}

// Writer function
void* writer(void* arg) {
    // Writer tries to write the data
    sem_wait(&write_lock);  // Block if another writer is writing

    // Writing to the data
    data++;
    printf("Writer %ld: Writing data = %d\n", (long)arg, data);
    sleep(2);  // Simulate writing

    sem_post(&write_lock);  // Release the write lock

    return NULL;
}

int main() {
    pthread_t readers[5], writers[3];
    long i;

    // Initialize semaphores
    sem_init(&mutex, 0, 1);  // Initialize mutex to 1 (binary
semaphore)
    sem_init(&write_lock, 0, 1);  // Initialize write lock to 1
(binary semaphore)
```

```
    // Create reader threads
    for (i = 0; i < 5; i++) {
        pthread_create(&readers[i], NULL, reader, (void*)i);
    }

    // Create writer threads
    for (i = 0; i < 3; i++) {
        pthread_create(&writers[i], NULL, writer, (void*)i);
    }

    // Wait for threads to finish
    for (i = 0; i < 5; i++) {
        pthread_join(readers[i], NULL);
    }
    for (i = 0; i < 3; i++) {
        pthread_join(writers[i], NULL);
    }

    // Destroy semaphores
    sem_destroy(&mutex);
    sem_destroy(&write_lock);

    return 0;
}
```

.....

19)//dining philosophers program

```
# include<stdio.h>
# include<pthread.h>
# include<semaphore.h>
# include<unistd.h>
pthread_mutex_t chopstick[5];
pthread_t philosopher[5];
void* runner(void*arg)
{
int i=*(int*)arg;
printf("Philosopher %d is thinking\n",i);
sleep(2);
pthread_mutex_lock(&chopstick[i]);
pthread_mutex_lock(&chopstick[(i+1)%5]);
sleep(3);
pthread_mutex_unlock(&chopstick[i]);
pthread_mutex_unlock(&chopstick[(i+1)%5]);
printf("Philosopher %d finisher eating \n",i);
}
int main()
{
int i;
for(i=0;i<5;i++)
{
```

```
pthread_create(&philosopher[i],NULL,runner,&i);
sleep(1);
}
for(i=0;i<5;i++)
{
pthread_join(philosopher[i],NULL);
}
}
```

.....

20) Bankers algorithm

.....

21)//FIFO page replacemt algorithm

```
#include <stdio.h>

void fifoPageReplacement(int pages[], int n, int frames) {
    int frame[frames];
    int index = 0, pageFaults = 0;

    // Initialize frames to -1 (indicating empty)
    for (int i = 0; i < frames; i++) {
        frame[i] = -1;
    }

    printf("\nPage Reference\tFrames\n");

    for (int i = 0; i < n; i++) {
        int found = 0;

        // Check if the page is already in the frame
        for (int j = 0; j < frames; j++) {
            if (frame[j] == pages[i]) {
                found = 1;
                break;
            }
        }

        // If the page is not found, replace the oldest page
        if (!found) {
            frame[index] = pages[i];
            index = (index + 1) % frames; // Circularly increment
index
            pageFaults++;
        }
```

```
        // Print the current state of frames
        printf("%d\t\t", pages[i]);
        for (int j = 0; j < frames; j++) {
            if (frame[j] != -1)
                printf("%d ", frame[j]);
            else
                printf("- ");
        }
        printf("\n");
    }

    printf("\nTotal Page Faults: %d\n", pageFaults);
}

int main() {
    int n, frames;

    printf("Enter the number of pages: ");
    scanf("%d", &n);

    int pages[n];
    printf("Enter the page reference sequence: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &pages[i]);
    }

    printf("Enter the number of frames: ");
    scanf("%d", &frames);

    fifoPageReplacement(pages, n, frames);

    return 0;
}
```