SHELL PROGRAMS
SEARCHING A SUBSTRING IN GIVEN TEXT

PROGRAM:

```
echo Enter main string:
read main
l1=`echo $main | wc -c`
l1=`expr $l1 - 1`
echo Enter sub string:
read sub
l2=`echo $sub | wc -c`
l2=`expr $l2 - 1`
n=1
m=1
pos=0
while [ $n -le $l1 ]
do
a=`echo $main | cut -c $n`
b=`echo $sub | cut -c $m`
if [ $a = $b ]
then
n=`expr $n + 1`
m=`expr $m + 1`
pos=`expr $n - $l2`
r=`expr $m - 1`
if [ $r -eq $l2 ]
then
break
fi
else
pos=0
m=1
n=`expr $n + 1`
fi
done
echo Position of sub stringin main string is $pos
```

OUTPUT:

```
Enter main string:

This is a shell pgm

Enter sub string:
shell
Position of sub stringin main string is 11
```

MENU BASED MATH CALCULATOR

PROGRAM:

```
echo " Menu Based Calculator"
echo "Enter the Operands"
read a
read b
echo "Enter the Operator"
read o
case $o in
"+" ) echo "$a + $b" = `expr $a + $b`;;
"-" ) echo "$a + $b" =  `expr $a - $b`;;
```

```
"*" ) echo "$a + $b" =   `expr $a * $b`;;
"/" ) echo  "$a + $b" =  `expr $a / $b`;;
* ) echo " Inavlid Operation"
esac
```

OUTPUT:

```
Menu Based Calculator
Enter the Operands
4
6
Enter the Operator
+
4  +  6  =  10
```

CONVERTING ALL FILENAMES FROM LOWERCASE TO UPPERCASE

PROGRAM

```
for i in *
do
echo Before Converting to uppercase the filename is
echo $i
j=`echo $i | tr '[a-z]' '[A-Z]'`
echo After Converting to uppercase the filename is
echo $j
mv $i $j
done
```

OUTPUT

```
Before Converting to upper case the filename  is
cse.sh
After Converting to uppercase the filename is
CSE.SH
```

PRINTING PATTERN USING LOOP STATEMENT

PROGRAM

```
echo "Enter the Limit "
read n
echo "Pattern"
for (( i = 1 ; i < $n ; i++ ))
do
for (( j = 1 ; j <= i ; j++ ))
do
echo -n " $ "
done
echo " "
done
```

OUTPUT

```
Enter the Limit

3

Pattern

 $
 $  $
 $  $  $
```

CONVERTING THE FILENAME FROM UPPERCASE TO LOWERCASE

PROGRAM

```
echo  -n  "Enter the Filename"
read filename
if [  !  -f  $filename ];
then
echo "Filename $filename does not exists"
exit 1
fi
tr  '[A-Z]' '[a-z]' <  $filename
```

OUTPUT

```
Enter the Filename
CSE.sh

cse.sh
```

SHOWING VARIOUS SYSTEM INFORMATION

PROGRAM

```
echo "SYSTEM INFORMATION"
echo "Hello ,$LOGNAME"
echo "Current Date is = $(date)"
echo "User is 'who I am'"
echo "Current Directory = $(pwd)"
echo "Network Name and Node Name = $(uname -n)"
echo "Kernal Name =$(uname -s)"
echo "Kernal Version=$(uname -v)"
echo "Kernal Release =$(uname -r)"
echo "Kernal OS =$(uname -o)"
echo "Proessor Type = $(uname -p)"
echo "Kernel Machine Information = $(uname -m)"
echo "All Information =$(uname -a)"
```

OUTPUT

```
SYSTEM INFORMATION
Hello, 3CSE-A
Current date is = Mar 17 08:38:58 IST 2014
Kernal Name = Linux
User is Who I am
Current Directory = 11scs122
Network name and Node name = linuxmint
Kernal Versio n= #1-Ubuntu SMP Fri Apr 16 08:10:02 UTC 2010
Kernal OS = GNU/Linux
kernal release =2.6.32-21-generic
Kernal Processor Type = 2.6.33.85.fcl3.i686.PAE
Kernal All Information = Linux main lab 2.6.33.85.fcl.3 i686.PAE
= #1-Ubuntu SMP Fri Apr 16 08:10:02 UTC 2010
 I686 i686 i686 GNU/Linux
```

==========

C PROGRAMS

IMPLEMENTATION OF PROCESS SCHEDULING MECHANISM

FIRST COME FIRST SERVE SCHEDULING

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int nop, wt[10], twt, tat[10], ttat, i, j, bt[10], t;
    float awt, atat;
    awt = 0.0;
    atat = 0.0;
    printf("Enter the no. of processes :  ");
    scanf("%d", &nop);
    for(i=0; i<nop; i++)
    {
        printf("Enter the burst time for process %d:  ",i+1);
        scanf("%d", &bt[i]);
    }
    wt[0] = 0;
    tat[0] = bt[0];
    twt = wt[0];
    ttat = tat[0];
    for(i=1; i<nop; i++)
    {
        wt[i] = wt[i-1]+bt[i-1];
        tat[i] = wt[i]+bt[i];
        twt+=wt[i];
        ttat+=tat[i];
    }
    awt = (float)twt/nop;
    atat = (float)ttat/nop;
    printf("\nProcess ID\tBurst Time\tWaiting Time\tTurn Around Time\n");
    for(i=0; i<nop; i++)
    {
        printf("%d\t\t%d\t\t%d\t\t%d\n", i+1, bt[i], wt[i], tat[i]);
    }
    printf("\nTotal waiting time = %d", twt);
    printf("\nTotal turn around time = %d", ttat);
    printf("\nAverage waiting time = %f", awt);
    printf("\nAverage turn around time = %f", atat);
    return 0;
}
```

SHORT JOB FIRST SCHEDULING

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main()
{
int nop,wt[10],twt,tat[10],ttat,i,j,bt[10],t;
float awt,atat;
clrscr();
```

```
awt=0.0;
atat=0.0;
printf("Enter the no.of process:");
scanf("%d",&nop);
for(i=0;i<nop;i++)
{
printf("Enter the burst time for process %d: ", i);
scanf("%d",&bt[i]);
}
for(i=0;i<nop;i++)
{
for(j=i+1;j<nop;j++)
{
if(bt[i]>=bt[j])
{
t=bt[i];
bt[i]=bt[j];
bt[j]=t;
}
}
}
wt[0]=0;
tat[0]=bt[0];
twt=wt[0];
ttat=tat[0];
for(i=1;i<nop;i++)
{
wt[i]=wt[i-1]+bt[i-1];
tat[i]=wt[i]+bt[i];
twt+=wt[i];
ttat+=tat[i];
}
awt=(float)twt/nop;
atat=(float)ttat/nop;
printf("\nProcessid\tBurstTime\tWaitingTime\tTurnaroundTime\n");
for(i=0;i<nop;i++)
printf("%d\t\t%d\t\t%d\t\t%d\n",i,bt[i],wt[i],tat[i]);
printf("\nTotal Waiting Time:%d\n",twt);
printf("\nTotal Around Time:%d\n",ttat);
printf("\nAverage Waiting Time:%f\n",awt);
printf("\nAverage Total Around Time:%f\n",atat);
getch();
}
```

PRIORITY QUEUE SCHEDULING

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main(){
char s[21][21],chng[20];
int wt[21],a[21],n,i,j,temp,trn[21],p[21];
float tot,t;
printf("Enter the no.of process :");
scanf("%d",&n);
for(i=1;i<=n;i++){
printf("Enter process id and time and priority :");
scanf("%s%d%d",&s[i],&a[i],&p[i]);}
wt[0]=0;
```

```
a[0]=0;
t=tot=0;
for(i=1;i<=n;i++){
for(j=i+1;j<=n;j++){
if(p[i]>p[j]){
temp=a[i];
a[i]=a[j];
a[j]=temp;
temp=p[i];
p[i]=p[j];
p[j]=temp;
strcpy(chng,s[i]);
strcpy(s[i],s[j]);
strcpy(s[j],chng);}}}
printf(" \n process\t burst time\t waiting time\t turn around time\t priority :");
for(i=1;i<=n;i++){
wt[i]=wt[i-1]+a[i-1];
trn[i]=wt[i]+a[i];
printf("%s\t%d\t%d\t%d\t%d\n",s[i],a[i],wt[i],trn[i],p[i]);
tot=tot+wt[i];
t=t+trn[i];}
printf("Average waiting time=%f Average turn around time=%f",(tot/n),(t/n));
getch();
}
```

READER - WRITER PROBLEM

PROGRAM

```
#include<stdio.h>
int x = 1, rc = 0, readcount = 1;
void p(int * a) {
    while ( * a == 0) {
        printf("busy wait");
    }
    * a = * a - 1;
}
void v(int * b) {
    * b = * b + 1;
}
void p1(int * c) {
    while ( * c == 0) {
        printf("busy wait");
    }
    * c = * c - 1;
}
void v1(int * d) {
    * d = * d + 1;
}
void reader() {
    int flag = 1;
    while (flag == 1) {
        p( & readcount);
        rc = rc + 1;
        if (rc == 1)
            p1( & x);
        v( & readcount);
        printf("\n Reader is reading");
        p( & readcount);
        rc = rc - 1;
        if (rc == 0)
            v1( & x);
        v( & readcount);
```

```
        flag = 0;
    }
}
void writer() {
    p1( & x);
    printf("\n Writer is writing");
    v1( & x);
}
int main() {
    reader();
    writer();
    reader();
    writer();
}
```

DINING PHILOSOPHER'S PROBLEM

PROBLEM

```
#include<stdio.h>
#include<conio.h>
#define LEFT (i+4) %5
#define RIGHT (i+1) %5
#define THINKING 0
#define HUNGRY 1
#define EATING 2
int state[5];
void put_forks(int);
void test(int);
void take_forks(int);
void philosopher(int i)
{
if(state[i]==0)
{
take_forks(i);
if(state[i]==EATING)
printf("\n Eating in process....");
put_forks(i);
}
}
void put_forks(int i)
{
state[i]=THINKING;
printf("\n philosopher %d completed its works",i);
test(LEFT);
test(RIGHT);
}
void take_forks(int i)
{
state[i]=HUNGRY;
test(i);
}
void test(int i)
{
if(state[i]==HUNGRY && state[LEFT]!=EATING && state[RIGHT]!=EATING)
{
printf("\n philosopher %d can eat",i);
state[i]=EATING;
}
}
void main()
{
int i;
clrscr();
```

```
for(i=1;i<=5;i++)
state[i]=0;
printf("\n\t\t\t Dining Philosopher Problem");
printf("\n\t\t.....");
for(i=1;i<=5;i++)
{
printf("\n\n the philosopher %d falls hungry\n",i);
philosopher(i);
}
getch();
}
```

MEMORY MANAGEMENT SCHEME

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main()
{
int f3[20],f2[20],r[20],r1[20],ms,bod,sb[20],nsb[20],nsb1[20],np,sp[20];
int f[20],i,j,l,k,z[20],s=0;
clrscr();
printf("enter the memory size:");
scanf("%d",&ms);
printf("\n enter the number of block of division of memory:");
scanf("%d",&bod);
printf("enter the size of each block:");
for(i=1;i<=bod;i++)
{
printf("\nBlock[%d]:",i);
scanf("%d",&sb[i]);
f[i]=1;
f2[i]=1;
f3[i]=1;
r[i]=1;
r1[i]=1;
z[i]=sb[i];
}
printf("\nenter the number of process:");
scanf("%d",&np);
printf("\nenter the size of each process:");
for(i=1;i<=np;i++)
{
printf("\nprocess[%d]:",i);
scanf("%d",&sp[i]);
}
printf("\n  FIRST FIT  ");
printf("\n *********  ");
for(i=1;i<=np;i++)
{
for(j=1;j<=bod;j++)
{
if((sb[j]>=sb[i]) && (f[j]!=0))
{
printf("\n Process p[%d] is allocated to Block[%d]",i,j);
f[j]=0;
z[j]=sb[j]-sp[i];
s++;
goto l1;
}
}
printf("\n process p[%d] cannot be allocated",i);
```

```
l1:
printf(" ");
}
printf("\n\n Remaining space left in each block \n");
printf("\n     ********************************** \n");
for(i=1;i<=bod;i++)
{
printf("\n Block[%d]: free space =%d",i,z[i]);
}
printf("\n\nUnallocated Blocks");
printf(" \n ******************");
for(i=1;i<=bod;i++)
{
if(f[i]!=0)
{
printf("\n Block [%d] unallocated",i);
}
}
if(s==bod)
printf("\n No Block is left unallocated");
getch();
clrscr();
s=0;
getch();
printf("\n\n BEST FIT ");
printf("\n   ******** ");
for(i=2;i<=bod;i++)
{
for(j=1;j<i;j++)
{
if(sb[i]>=sb[j])
r[i]++;
else
r[j]++;
}
}
for(i=1;i<=bod;i++)
{
nsb[r[i]]=sb[i];
z[r[i]]=sb[i];
}
for(i=1;i<=np;i++)
{
for(j=1;j<=bod;j++)
{
if((nsb[j]>=sp[i]) && (f2[j]!=0))
{
for(k=1;k<=bod;k++)
{
if(r[k]==j)
l=k;
}
printf("\nProcess p[%d] is allocated to Block[%d]",i,l);
f2[j]=0;
z[j]=nsb[j]-sp[i];
s++;
goto l2;
}
}
printf("\n process p[%d] cannot be allocated",i);
l2:
printf(" ");
}
printf("\n free space in each block \n");
printf("   ********************** \n");
```

```
for(i=1;i<=bod;i++)
printf("\nBlock [%d]: free space =%d",i,z[r[i]]);

printf("\n\nUnallocated Blocks");
printf(" \n ******************");
for(i=1;i<=bod;i++)
{
if(f2[r[i]]!=0)
{
printf("\n Block [%d] unallocated",i);
}
}
if(s==bod)
printf("\n No Block is left unallocated");
getch();
clrscr();
s=0;
getch();
printf("\n\n WORST FIT ");
printf("\n   ********* ");
for(i=2;i<=bod;i++)
{
for(j=1;j<i;j++)
{
if(sb[i]<=sb[j])
r1[i]++;
else
r1[j]++;
}
}
for(i=1;i<=bod;i++)
{
nsb1[r1[i]]=sb[i];
z[r1[i]]=sb[i];
}
for(i=1;i<=np;i++)
{
for(j=1;j<=bod;j++)
{
if((nsb1[j]>=sp[i]) && (f3[j]!=0))
{
for(k=1;k<=bod;k++)
{
if(r1[k]==j)
l=k;
}
printf("\nProcess p[%d] is allocated to Block[%d]",i,l);
f3[j]=0;
z[j]=nsb1[j]-sp[i];
s++;
goto l3;
}
}
printf("\n process p[%d] cannot be allocated",i);
l3:
printf(" ");
}
printf("\n free space in each block \n");
printf("   ************************ \n");
for(i=1;i<=bod;i++)
printf("\nBlock [%d]: free space =%d",i,z[r1[i]]);

printf("\n\nUnallocated Blocks");
printf(" \n ******************");
for(i=1;i<=bod;i++)
```

```
{
if(f3[r1[i]]!=0)
{
printf("\n Block [%d] unallocated",i);
}
if(s==bod)
printf("\n No Block is left unallocated");
getch();
printf("\n");
}
}
```

BANKERS ALGORITHM

PROGRAM

```
#include <stdio.h>
int main()
{
int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];
int p, r, i, j, process, count;
count = 0;

printf("Enter the no of processes : ");
scanf("%d", &p);

for(i = 0; i< p; i++)
        completed[i] = 0;

printf("\n\nEnter the no of resources : ");
scanf("%d", &r);

printf("\n\nEnter the Max Matrix for each process : ");
for(i = 0; i < p; i++)
{
        printf("\nFor process %d : ", i + 1);
        for(j = 0; j < r; j++)
                scanf("%d", &Max[i][j]);
}

printf("\n\nEnter the allocation for each process : ");
for(i = 0; i < p; i++)
{
        printf("\nFor process %d : ",i + 1);
        for(j = 0; j < r; j++)
                scanf("%d", &alloc[i][j]);
}

printf("\n\nEnter the Available Resources : ");
for(i = 0; i < r; i++)
                scanf("%d", &avail[i]);


        for(i = 0; i < p; i++)
                for(j = 0; j < r; j++)
                        need[i][j] = Max[i][j] - alloc[i][j];

do
{
        printf("\n Max matrix:\tAllocation matrix:\n");
        for(i = 0; i < p; i++)
        {
```

```
                for( j = 0; j < r; j++)
                        printf("%d  ", Max[i][j]);
                printf("\t\t");
                for( j = 0; j < r; j++)
                        printf("%d  ", alloc[i][j]);
                printf("\n");
        }

        process = -1;

        for(i = 0; i < p; i++)
        {
                if(completed[i] == 0)//if not completed
                {
                        process = i ;
                        for(j = 0; j < r; j++)
                        {
                                if(need[i][j]>avail[j])
                                {
                                        process = -1;
                                        break;
                                }
                        }
                }
                if(process != -1)
                        break;
        }

        if(process != -1)
        {
                printf("\nProcess %d runs to completion!", process + 1);
                safeSequence[count] = process ;
                count++;
                for(j = 0; j < r; j++)
                {
                        avail[j] += alloc[process][j];
                        alloc[process][j] = 0;
                        Max[process][j] = 0;
                        completed[process] = 1;
                }
        }
}while(count != p && process != -1);


if(count == p)
{
        printf("\nThe system is in a safe state!!\n");
        printf("Safe Sequence : < ");
        for( i = 0; i < p; i++)
                        printf("%d  ", safeSequence[i]);
        printf(">\n");
}
else
        printf("\nThe system is in an unsafe state!!");
return 0;
}
```

PRODUCER CONSUMER PROBLEM

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int mutex = 1, full = 0, empty = 3, x = 0;
```

```
main() {
  int n;
  void producer();
  void consumer();
  int wait(int);
  int signal(int);
  printf("\n 1. Producer \n 2. Consumer \n 3. Exit\n\n");
  while (1) {
    printf("\n Enter your choice : ");
    scanf("%d", & n);
    switch (n) {
    case 1:
      if ((mutex == 1) && (empty != 0))
        producer();
      else
        printf("\n Buffer is full\n");
      break;
    case 2:
      if ((mutex == 1) && (full != 0))
        consumer();
      else
        printf("\n Buffer is empty\n");
      break;
    case 3:
      exit(0);
      break;
    }
  }
}
int wait(int s) {
  return (--s);
}
int signal(int s) {
  return (++s);
}
void producer() {
  mutex = wait(mutex);
  full = signal(full);
  empty = wait(empty);
  x++;
  printf("\n Producer produces the item %d\n", x);
  mutex = signal(mutex);
}
void consumer() {
  mutex = wait(mutex);
  full = wait(full);
  empty = signal(empty);
  printf("\n Consumer consumes item %d\n", x);
  x--;
  mutex = signal(mutex);
}
```

PROBLEM

MEMORY MANAGEMENT SCHEME - PAGING

PROGRAM USING C++

```
#include<stdio.h>

#define MAX 50
int main() {
    int page[MAX], i, n, f, ps, off, pno;
    printf("\nEnter the no of  pages in memory: ");
    scanf("%d", & n);
    printf("\nEnter page size: ");
    scanf("%d", & ps);
    printf("\nEnter no of frames: ");
    scanf("%d", & f);
    for (i = 0; i < n; i++)
        page[i] = -1;
    printf("\nEnter the page table\n");
    printf("(Enter frame no as -1 if that page is not present in any frame)\n\n");
    printf("\npageno\tframeno\n-------\t-------");
    for (i = 0; i < n; i++) {
        printf("\n\n%d\t\t",i);
        scanf("%d", & page[i]);
    }
    printf("\n\nEnter the logical address(i.e,page no & offset):");
    scanf("%d%d", & pno, & off);
    if (page[pno] == -1)
        printf("\n\nThe required page is not available in any of frames");
    else
        printf("%d,%d", page[pno], off);
    return (1);
}
--------------------------------------------------------------------------
memory allocation strategy

--------------------------------------------------------------------------
#include<stdio.h>
#include<conio.h>

int main() {
    int p[20], f[20], min, minindex, n, i, j, c, f1[20], f2[20], f3[20], k = 0, h = 0, flag,
t = 0, n1;
    printf("\nEnter the number of memory partitions : ");
    scanf("%d", & n);
    printf("\nEnter the number of process : ");
    scanf("%d", & n1);
    for (i = 0; i < n; i++) {
        printf("\n Enter the memory partition size %d: ", i + 1);
        scanf("%d", & f[i]);
        f1[i] = f[i];
        f2[i] = f[i];
        f3[i] = f[i];
    }
    for (i = 0; i < n; i++) {
        printf("\n Enter the page size %d: ", i + 1);
        scanf("%d", & p[i]);
    }
    do {
        printf("\n1. First fit\n");
        printf("\n2. Best fit\n");
        printf("\n3. Worst fit\n");
        printf("\nEnter your choice : ");
        scanf("%d", & c);
        switch (c) {
        case 1:
            for (i = 0; i < n1; i++) {
                for (j = 0; j < n; j++) {
```

```
                    f1[i] = 0;
                    if (p[i] <= f[j]) {
                        f1[i] = f[j];
                        f[j] = 0;
                        break;
                    }
                }
            }
            break;
        case 2:
            for (i = 0; i < n1; i++) {
                min = 9999;
                minindex = -1;
                for (j = 0; j < n; j++) {
                    if (p[i] <= f2[j] && f2[j] != 0 && min > f2[j]) {
                        min = f2[j];
                        minindex = j;
                    }
                }
                f1[i] = f[minindex];
                f2[minindex] = 0;
            }
            break;
        case 3:
            for (i = 0; i < n1; i++) {
                f1[i] = 0;
                for (j = 0; j < n; j++) {
                    if (p[i] < f3[j]) {
                        k++;
                        if (k == 1)
                            f1[i] = f3[j];
                        if (f1[i] <= f3[j]) {
                            flag = 1;
                            f1[i] = f3[j];
                            h = j;
                        }
                    }
                }
                k = 0;
                if (flag == 1)
                    f3[h] = 0;
            }
            break;
        default:
            printf("\n Out of choice");
        }
        printf("\n----------\n\n");
        printf("\n|Page          |Frame          |Free \n");
        printf("\n----------\n");
        t = 0;
        for (i = 0; i < n1; i++) {
            h = f1[i] - p[i];
            if (h < 0)
                h = 0;
            printf("\n%d\t\t%d\t\t%d", p[i], f1[i], h);
            t = t + h;
        }
        printf("\n----------\n");
        printf("\n Total free space in memory: %d\n\n", t);
    }
    while (c < 4);
}
```