//SHELL SCRIPTING TO IMPLEMENT FIBONACCI SERIES//

```
#! /bin/bash

echo "Input number of terms "
read N
a=0
b=1
echo "The Fibonacci series is : "

for (( i=0; i<N; i++ ))
do
        echo -n "$a "
        fn=$((a + b))
        a=$b
        b=$fn
done
```

//SHELL SCRIPTING TO IMPLEMENT ARITHMETIC OPERATION USING CASE//

```
#!/bin/bash

echo "Enter A"
read A
echo "Enter B"
read B
echo "Enter operation to be performed:"
echo "1)Addition 2)Substraction 3)Multiplication 4)Division "
read op
case $op in
1) c=`expr $A + $B` ;;
2) c=`expr $A - $B` ;;
3) c=`expr $A * $B` ;;
4) c=`expr $A / $B` ;;
5) echo "Invalid option"
esac
echo "Result:"
echo $c
```

```
//C PROGRAM TO DEMONSTRATE WORKING OF FORK GETPID GETPPID//

#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
int main()
{
        int pid;
        pid = fork();
        if(pid == -1)
        {
                perror("fork failed");
                exit(0);
        }
        if (pid == 0)
        {
                printf("\n child process is under execution ");
                printf(" \n Process id of the child process is %d ",getpid());
                printf("\n process id of the parent process is %d",getppid());
        }
        else
        {
                printf("\n Parent process is under execution ");
                printf("\n Process id of the parent process is %d ",getpid());
                printf("\n Process id of the child process in parent is %d",pid);
                printf("\n Process id of the parent of parent is %d",getppid());
        }
        return(0);
}
```

```
//C PROGRAM TO FIND MODE OF A FILE USING STAT SYSTEM CALL//

#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
#include<time.h>
void printfileproperties(struct stat stats);
int main()
{
        char path[100];
        struct stat stats;

        printf("Enter source file path: ");
        scanf("%s",path);

        if (stat(path,&stats)==0)
        {
                printfileproperties(stats);
        }
        else

        {
                printf("unable to get file properties.\n");
                printf("please check whether '%s' file exits .\n ",path);
        }
        return 0;
}

void printfileproperties(struct stat stats)
{
        struct tm dt;
        printf("\n File access:\n ");
        if(stats.st_mode & R_OK)
                printf("read\n");
        if(stats.st_mode & W_OK)
                printf("write\n");
        if(stats.st_mode & X_OK)
                printf("execute\n");
}
```

```
//IMPLEMENT PRODUCER CONSUMER//

#include <stdio.h>
#include <stdlib.h>
int mutex = 1;
int full = 0;
int empty = 10, x = 0;
void producer()
{
  --mutex;
    ++full;
    --empty;
    x++;
printf("\nProducer produces item %d",x);
    ++mutex;
}
void consumer()
{
 --mutex;
    --full;
    ++empty;
printf("\nConsumer consumes item %d",x);
    x--;
   ++mutex;
}
int main()
{
 int n, i;
printf("\n1. Press 1 for Producer""\n2. Press 2 for Consumer""\n3. Press 3 for Exit");
for (i = 1; i > 0; i++) {
printf("\nEnter your choice:");
scanf("%d", &n);
switch (n) {
    case 1:
        if ((mutex == 1)&& (empty != 0)) {
                 producer();}
        else {
          printf("Buffer is full!");
              }
    break;
    case 2:
        if ((mutex == 1)&& (full != 0)) {
                 consumer();
```

```
            }
        else {
              printf("Buffer is empty!");
            }
    break;
    case 3:
            exit(0);
    break;
            }
}
}
```

//IMPLEMENT IPC USING SHARED MEMORY//

/TO WRITE/

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
int i;
void *shared_memory;
char buff[100];
int shmid;
shmid=shmget((key_t)2345, 1024, 0666|IPC_CREAT);
//printf("Key of shared memory is %d\n",shmid);
shared_memory=shmat(shmid,NULL,0);
printf("Process attached at %p\n",shared_memory);
printf("Enter some data to write to shared memory\n");
read(0,buff,100);
strcpy(shared_memory,buff);
printf("You wrote : %s\n",(char *)shared_memory);
}
```

/TO READ/

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
int i;
void *shared_memory;
char buff[100];
int shmid;
shmid=shmget((key_t)2345, 1024, 0666);
//printf("Key of shared memory is %d\n",shmid);
shared_memory=shmat(shmid,NULL,0);
printf("Process attached at %p\n",shared_memory);
printf("Data read from shared memory is : %s\n",(char *)shared_memory);
}
```

```
//PROGRAM TO STIMULATE COMMAND ls//

#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<dirent.h>
int main(int argc,char**argv)
{
struct dirent **namelist;
int n;
if(argc<1)
{
exit(EXIT_FAILURE);
}
else if(argc==1)
{
n= scandir(".",&namelist,NULL,alphasort);
}
else
{
n= scandir(argv[1],&namelist,NULL,alphasort);
}
if(n<0)
{perror("scandir");
exit(EXIT_FAILURE);
}
else{
while(n--)
{
printf("%s\n",namelist[n]->d_name);
free(namelist[n]);
}
free(namelist);
}
exit(EXIT_SUCCESS);
}
```

```
//PROGRAM TO IMPLEMENT FCFS

#include<stdio.h>
int main()
{
   int AT[10],BT[10],WT[10],TT[10],n;
   int burst=0,cmpl_T;
   float Avg_WT,Avg_TT,Total=0;
   printf("Enter number of the process\n");
   scanf("%d",&n);
   printf("Enter Arrival time and Burst time of the process\n");
   printf("AT\tBT\n");
   for(int i=0;i<n;i++)
   {
      scanf("%d%d",&AT[i],&BT[i]);
   }

   // Logic for calculating Waiting time
   for(int i=0;i<n;i++)
   {
      if(i==0)
         WT[i]=AT[i];
      else
         WT[i]=burst-AT[i];
      burst+=BT[i];
      Total+=WT[i];
   }
   Avg_WT=Total/n;

   // Logic for calculating Turn around time
   cmpl_T=0;
   Total=0;
   for(int i=0;i<n;i++)
   {
      cmpl_T+=BT[i];
      TT[i]=cmpl_T-AT[i];
      Total+=TT[i];
   }
   Avg_TT=Total/n;

   // printing of outputs

   printf("Process ,Waiting_time ,TurnA_time\n");
```

```
    for(int i=0;i<n;i++)
    {
        printf("%d\t\t%d\t\t%d\n",i+1,WT[i],TT[i]);
    }
    printf("Average waiting time is : %f\n",Avg_WT);
    printf("Average turn around time is : %f\n",Avg_TT);
    return 0;
}
```

```
//PROGRAM TO IMPLEMENT SJF

#include<stdio.h>
int main()
{
 int BT[10],AT[10],Pid[10],WT[10],TT[10];
 int n;
printf(" Input the number of process from  \n");
scanf("%d",&n);
printf("Input the burst time  & arival time \n");
printf("Pid|BT|AT \n");
for (int i=0; i<n; i++){
scanf("%d",&Pid[i]);
scanf("%d",&BT[i]);
scanf("%d",&AT[i]);
}

//for sorting (burst time);

for (int i=0; i<n-1; i++)
   for(int j=0; j<n-i-1; j++)
{
        if (BT[j+1] < BT[j])
        {
                int burst_t;
                burst_t = BT[j];
                BT[j] =BT[j+1];
                BT[j+1] = burst_t;

                int arival_t;
                arival_t = AT[j];
                AT[j] = AT[j+1];
                AT[j+1] = arival_t;
int pro_id;
pro_id=Pid[j];
Pid[j] = Pid[j+1];
Pid[j+1] = pro_id;

        }

}
/*
for (int i=0; i<n; i++)
{
int total;
total += BT[i];
 }
 */
WT[0]=0;
TT[0]=BT[0];
float total_w=0;
float total_t=BT[0];
for (int i=1; i<n; i++)
{
//waiting time
WT[i]=WT[i-1]+BT[i-1];
TT[i] = WT[i] + BT[i];
total_t +=(float)BT[i]+ (float)WT[i];
}

float avg_w,avg_t;
for (int i=0;i<n;i++)
{
total_w = (float)total_w + (float)WT[i];
}

avg_w= (float)total_w/(float) n;
avg_t = (float)total_t/(float)n;

printf("Process id = Pid,Burst time = BT, Arival time = AT");
printf("\n process schedule :\n|Pid|AT|BT|WT|TT|\n");

for(int i=0; i<n; i++)
{
 printf("|%d|%d|%d|%d|%d|\n",Pid[i],AT[i],BT[i],WT[i],TT[i]);
}

printf("\n average turn around time: %f \n  average waiting time : %f \n",avg_t,avg_w);
}
```

```
//PROGRAM TO IMPLEMENT ROUND ROBIN

#include<stdio.h>
void main()
{
int i ,nop,y,quant,at[10],bt[10],temp[10],sum=0,tat=0,count=0,wt=0;
float avg_wt,avg_tat;
printf("Input total number of process \n");
scanf("%d",&nop);
y=nop;

//for process arival and burst time
for (i=0;i<nop;i++)
{//repeat till it meets the number of process
        printf("Input the arrival time and burst time of the process[%d] \n",i+1);
printf("Arrival time \t:");
scanf("%d",&at[i]);
printf("\n Burst time \t:");
scanf("%d",&bt[i]);
temp[i] =bt[i];//will be used to check whether the process is completed or not in future
}

printf("Enter the time quanta for the process \t:");
scanf("%d",&quant);
printf("\n Process no \t\t Burst time \t\t TAT \t\t Wating time ");

i=0;
for (sum =0;y!=0;)
{
        if (temp[i]<=quant && temp[i]>0)
        {
                sum =sum + temp[i];
                temp[i]=0;
                count =1;
        }
        else if (temp[i]>0)
        {
                temp[i] =temp[i]-quant;
                sum =sum +quant;
        }
        if (temp[i] == 0 && count ==1)
        {
                y--;
        printf("\n Process NO[%d]\t\t[%d]\t\t\t\t%d\t\t\t\t\t%d",i+1,bt[i],sum-at[i],sum-at[i]-bt[i]);
        wt =wt +sum-at[i]-bt[i];
        tat = tat + sum -at[i];
        count =0;
    }
    if (i==nop-1)
    {
        i=0;
    }
    else if (at[i+1]<=sum)
    {
        i++;
    }
    else
    {
        i=0;
    }
}
avg_wt =wt *1.0/nop;
avg_tat = tat* 1.0/nop;
printf("\n Average Wating time \t%f:",avg_wt);
printf("\n Average turn around time \t%f:",avg_tat);
}
```

//PROGRAM TO IMPLEMENT MEMORY MANAGEMENT//

/MVT/

```
 #include<stdio.h>
int main()
{
int ms,mp[10],i, temp,n=0;
char ch = 'y';
printf("\nEnter the total memory available (in Bytes)-- ");
scanf("%d",&ms);
temp=ms;
for(i=0;ch=='y';i++,n++)
{
printf("\nEnter memory required for process %d (in Bytes) -- ",i+1);
scanf("%d",&mp[i]);
if(mp[i]<=temp)
{
printf("\nMemory is allocated for Process %d ",i+1);
temp = temp - mp[i];
}
else
{
printf("\nMemory is Full");
break;
}
printf("\nDo you want to continue(y/n) -- ");
scanf(" %c", &ch);
}
printf("\n\nTotal Memory Available -- %d", ms);
printf("\n\n\tPROCESS\t\t MEMORY ALLOCATED ");
for(i=0;i<n;i++)
printf("\n \t%d\t\t%d",i+1,mp[i]);
printf("\n\nTotal Memory Allocated is %d",ms-temp);
printf("\nTotal External Fragmentation is %d",temp);
return (0);
}
```

```
**/MFT/**

 #include<stdio.h>
int main()
{
int ms, bs, nob, ef,n, mp[10],tif=0;
int i,p=0;
printf("Enter the total memory available (in Bytes) -- ");
scanf("%d",&ms);
printf("Enter the block size (in Bytes) -- ");
scanf("%d", &bs);
nob=ms/bs;
ef=ms - nob*bs;
printf("\nEnter the number of processes -- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter memory required for process %d (in Bytes)-- ",i+1);
scanf("%d",&mp[i]);
}
printf("\nNo. of Blocks available in memory -- %d",nob);
printf("\n\nPROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL
FRAGMENTATION");
for(i=0;i<n && p<nob;i++)
{
printf("\n %d\t\t%d",i+1,mp[i]);
if(mp[i] > bs)
printf("\t\tNO\t\t---");
else
{
printf("\t\tYES\t%d",bs-mp[i]);
tif = tif + bs-mp[i];
p++;
}
}
if(i<n)
printf("\nMemory is Full, Remaining Processes cannot be accomodated");
printf("\n\nTotal Internal Fragmentation is %d",tif);
printf("\nTotal External Fragmentation is %d",ef);
return(0);
}
```

```
//IMPLEMENT PROGRAM FOR DEADLOCK AVOIDANCE//

// Banker's Algorithm
#include <stdio.h>
int main()
{
   // P0, P1, P2, P3, P4 are the Process names here

   int n, m, i, j, k;
   n = 5;                        // Number of processes
   m = 3;                         // Number of resources
   int alloc[5][3] = {{0, 1, 0},  // P0 // Allocation Matrix
                  {2, 0, 0},  // P1
                  {3, 0, 2},  // P2
                  {2, 1, 1},  // P3
                  {0, 0, 2}}; // P4

   int max[5][3] = {{7, 5, 3},  // P0 // MAX Matrix
                {3, 2, 2},  // P1
                {9, 0, 2},  // P2
                {2, 2, 2},  // P3
                {4, 3, 3}}; // P4

   int avail[3] = {3, 3, 2}; // Available Resources

   int f[n], ans[n], ind = 0;
   for (k = 0; k < n; k++)
   {
      f[k] = 0;
   }
   int need[n][m];
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < m; j++)
         need[i][j] = max[i][j] - alloc[i][j];
   }
   int y = 0;
   for (k = 0; k < 5; k++)
   {
      for (i = 0; i < n; i++)
      {
         if (f[i] == 0)
         {
                int flag = 0;
                for (j = 0; j < m; j++)
                {
                    if (need[i][j] > avail[j])
                    {
                        flag = 1;
                        break;
                    }
                }
                if (flag == 0)
                {
                    ans[ind++] = i;
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y];
                    f[i] = 1;
                }
            }
        }
    }
    int flag = 1;
    for (int i = 0; i < n; i++)
    {
        if (f[i] == 0)
        {
            flag = 0;
            printf("The following system is not safe");
            break;
        }
    }
    if (flag == 1)
    {
        printf("Following is the SAFE Sequence\n");
        for (i = 0; i < n - 1; i++)
            printf(" P%d ->", ans[i]);
        printf(" P%d", ans[n - 1]);
    }
    return (0);
}
```

```
// IMPLEMENT PROGRAM FOR DEADLOCK DETECTION //

**/SAFETY ALGO/**
#include<stdio.h>
int main()
{
int mark[20];
int i,j,np,nr;
int alloc[10][10],request[10][10],avail[10],r[10],w[10];
printf("\nEnter the no of process: ");
scanf("%d",&np);
printf("\nEnter the no of resources: ");
scanf("%d",&nr);
for(i=0;i<nr;i++)
{
printf("\nTotal Amount of the Resource R%d: ",i+1);
scanf("%d",&r[i]);
}
printf("\nEnter the request matrix:");
for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&request[i][j]);
printf("\nEnter the allocation matrix:");
for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&alloc[i][j]);
/*Available Resource calculation*/
for(j=0;j<nr;j++)
{
avail[j]=r[j];
for(i=0;i<np;i++)
{
avail[j]-=alloc[i][j];
}
}
//marking processes with zero allocation
for(i=0;i<np;i++)
{
int count=0;
 for(j=0;j<nr;j++)
   {
     if(alloc[i][j]==0)
       count++;
```

```
      else
        break;
    }
 if(count==nr)
 mark[i]=1;
}
// initialize W with avail
for(j=0;j<nr;j++)
    w[j]=avail[j];
//mark processes with request less than or equal to W
for(i=0;i<np;i++)
{
int canbeprocessed=0;
 if(mark[i]!=1)
{
   for(j=0;j<nr;j++)
    {
      if(request[i][j]<=w[j])
        canbeprocessed=1;
      else
         {
         canbeprocessed=0;
         break;
          }
     }
if(canbeprocessed)
{
mark[i]=1;
for(j=0;j<nr;j++)
w[j]+=alloc[i][j];
}
}
}
//checking for unmarked processes
int deadlock=0;
for(i=0;i<np;i++)
if(mark[i]!=1)
deadlock=1;
if(deadlock)
printf("\n Deadlock detected");
else
printf("\n No Deadlock possible");
}
```

```
//FILE MANIPULATION USING C - COPYING CONTENT FROM ONE FILE TO ANOTHER

#include <stdio.h>
#include <stdlib.h> // For exit()
int main(){
  FILE *fptr1, *fptr2;
  char filename[100], c;
  printf("Enter the filename to open for reading ");
  scanf("%s",filename);
  // Open one file for reading
  fptr1 = fopen(filename, "r");
  if (fptr1 == NULL){
    printf("Cannot open file %s ", filename);
    exit(0);
  }
  printf("Enter the filename to open for writing ");
  scanf("%s", filename);
  // Open another file for writing
  fptr2 = fopen(filename, "w");
  if (fptr2 == NULL){
    printf("Cannot open file %s ", filename);
    exit(0);
  }
  // Read contents from file
  c = fgetc(fptr1);
  while (c != EOF){
    fputc(c, fptr2);
    c = fgetc(fptr1);
  }
  printf("Contents copied to %s", filename);
  fclose(fptr1);
  fclose(fptr2);
  return 0;
}
```