Task-10

```
#include <stdio.h>
struct pstruct
{
int fno;
int pbit;
}ptable[10];
int pmsize,lmsize,psize,frame,page,ftable[20],frameno;
void info()
{
printf("\n\nMEMORY MANAGEMENT USING PAGING\n\n");
printf("\n\nEnter the Size of Physical memory: ");
scanf("%d",&pmsize);
printf("\n\nEnter the size of Logical memory: ");
scanf("%d",&lmsize);
printf("\n\nEnter the partition size: ");
scanf("%d",&psize);
frame = (int) pmsize/psize;
page = (int) lmsize/psize;
printf("\nThe physical memory is divided into %d no.of frames\n",frame);
printf("\nThe Logical memory is divided into %d no.ofpages",page);
}
void assign()
{
int i;
for (i=0;i<page;i++)
{
ptable[i].fno = -1;
ptable[i].pbit= -1;
}
for(i=0; i<frame;i++)
```

```
ftable[i] = 32555;
for (i=0;i<page;i++)
{
printf("\n\nEnter the Frame number where page %d must be placed: ",i);
scanf("%d",&frameno);
ftable[frameno] = i;
if(ptable[i].pbit == -1)
{
ptable[i].fno = frameno;
ptable[i].pbit = 1;
}
}
printf("\n\nPAGE TABLE\n\n");
printf("PageAddressFrameNo. PresenceBit\n\n");
for (i=0;i<page;i++)
printf("%d\t\t%d\t\t%d\n",i,ptable[i].fno,ptable[i].pbit);
printf("\n\n\n\tFRAME TABLE\n\n");
printf("FrameAddressPageNo\n\n");
for(i=0;i<frame;i++)
printf("%d\t\t%d\n",i,ftable[i]);
}
void cphyaddr()
{
int laddr,paddr,disp,phyaddr,baddr;
printf("\n\n\n\tProcess to create the Physical Address\n\n");
printf("\nEnter the Base Address: ");
scanf("%d",&baddr);
printf("\nEntertheLogical Address: ");
scanf("%d",&laddr);
paddr = laddr / psize;
disp = laddr % psize;
```

```
if(ptable[paddr].pbit == 1 )
phyaddr = baddr + (ptable[paddr].fno*psize) + disp;
printf("\nThe Physical Address where the instruction present: %d",phyaddr);
}
void main()
{
info();
assign();
cphyaddr();
}
```

Task-9

```
#include <stdio.h>
#include <stdlib.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(avail[j] < need[i][j])
{
process = -1;
break;
}
}
}
```

```
if(process != -1)
break;
}
if(process != -1)
{
printf("\nProcess %d runs to completion!", process + 1);
safeSequence[count] = process + 1;
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!!");
}
```

Task-8

```
#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;
}
```