PROGRAM 8:

(i)BUDDY ALGORITH:

```
#include<stdio.h>
#include<stdlib.h>
int tree[2050],i,j,k;
void segmentalloc(int,int), makedivided(int),makefree(int),printing(int,int);
int place(int),power(int,int);
int main()
{
int totsize,cho,req;
//clrscr();
for(i=0;i<80;i++) printf("%c",5);
printf("**BUDDY SYSTEM REQUIREMENTS**\n");
for(i=0;i<80;i++) printf("%c",5);
printf("Enter the Size of the memory:\n");
scanf("%d",&totsize);
while(1)
{
for(i=0;i<80;i++) printf("%c",5);
printf("\n**BUDDY SYSTEM**\n");
for(i=0;i<80;i++) printf("%c",5);
printf("1)Locate the process into the Memory\n");
printf("2)Remove the process from Memory\n");
printf("3)Tree structure for Memory allocation Map\n");
printf("4)Exit\n");
for(i=0;i<80;i++) printf("%c",5);
printf("Enter your choice:\n");
scanf("%d",&cho);
switch(cho)
{
case 1:
printf(" ");
```

```
for(i=0;i<80;i++)
printf("%c",5);
printf("\nMEMORY ALLOCATION\n");
for(i=0;i<80;i++) printf("%c",5);
printf("Enter the Process size:\n");
scanf("%d",&req);
segmentalloc(totsize,req);
break;
case 2:
for(i=0;i<80;i++) printf("%c",5);
printf("\nMEMORY DEALLOCATION\n");
for(i=0;i<80;i++) printf("%c",5);
printf("Enter the process size:\n");
scanf("%d",&req);
makefree(req);
break;
case 3:
for(i=0;i<80;i++) printf("%c",5);
printf("\nMEMORY ALLOCATION MAP\n");
for(i=0;i<80;i++) printf("%c",5);
printing(totsize,0);
for(i=0;i<80;i++) printf("%c",5);
break;
default: return(0);
}
}
}
void segmentalloc(int totsize,int request)
{
int flevel=0,size;
size=totsize;
if(request>totsize)
{
printf("%c RESULT:",2);
printf("The system don't have enough free memory\n");
```

```
printf("Suggession:Go for VIRTUAL MEMORY\n");
return;
}
while(1)
{
if(request<size && request>(size/2))
break;
else
{
size/=2;
flevel++;
}
}
for(i=power(2,flevel)-1;i<=(power(2,flevel+1)-2);i++)
if(tree[i]==0 && place(i))
{
tree[i]=request;
makedivided(i);
printf("Result: Successful Allocation\n");
break;
}
if(i==power(2,flevel+1)-1)
{
printf("Result:\n");
printf("The system don't have enough free memory\n");
printf("Suggession: Go for VIRTUAL Memory Mode\n");
}
}
void makedivided(int node)
{
while(node!=0)
{
node=node%2==0?(node-1)/2:node/2;
tree[node]=1;
}
```

```
}
int place(int node)
{
while(node!=0)
{
node=node%2==0?(node-1)/2:node/2;
if(tree[node]>1)
return 0;
}
return 1;
}
void makefree(int request)
{
int node=0;
while(1)
{
if(tree[node]==request)
break;
else
node++;
}
tree[node]=0;
while(node!=0)
{
if(tree[node%2==0?node-1:node+1]==0 && tree[node]==0)
{
tree[node%2==0?(node-1)/2:node/2]=0;
node=node%2==0?(node-1)/2:node/2;
}
else break;
}
}
int power(int x,int y)
{
int z,ans;
```

```
if(y==0) return 1;
ans=x;
for(z=1;z<y;z++)
ans*=x;
return ans;
}
void printing(int totsize,int node)
{
int permission=0,llimit,ulimit,tab;
if(node==0)
permission=1;
else if(node%2==0)
permission=tree[(node-1)/2]==1?1:0;
else
permission=tree[node/2]==1?1:0;
if(permission)
{
llimit=ulimit=tab=0;
while(1)
{
if(node>=llimit && node<=ulimit)
break;
else
{
tab++;
llimit=ulimit+1;
ulimit=2*ulimit+2;
}
}
printf(" %d ",totsize/power(2,tab));
if(tree[node]>1)
printf("---> Allocated %d\n",tree[node]);
else if(tree[node]==1)
printf("---> Divided\n");
else printf("---> Free\n");
```

```
printing(totsize,2*node+1);
printing(totsize,2*node+2);
}
}
```