FILE ALLOCATION

```
kaushik@kaushik-AcerPower-Series:~/OS $ cat filealloc.c
#include<stdio.h>
#include<malloc.h>
#include<string.h>
typedef struct Bnode* Block;
struct Index
{
Block Address[20];
};
struct Bnode
{
int ID;
char Filename[20];
Block Next;
Block LinkedNext;
struct Index Table;
};
struct Directory
{
char Filename[20];
int length,startBlock,endBlock,IndexBlock;
};
Block BCreate()
{
Block head;
head=(struct Bnode*)malloc(sizeof(struct Bnode));
head->Next=NULL;
return head;
}
void BlockInsert(Block B,int i)
{
Block NewBlock,pos;
pos=B;
while(pos->Next!=NULL)
pos=pos->Next;
NewBlock=(struct Bnode*)malloc(sizeof(struct Bnode));
strcpy(NewBlock->Filename,"Free");
NewBlock->ID=i;
NewBlock->Next=NULL;
NewBlock->LinkedNext=NULL;
pos->Next=NewBlock;
return;
}
//CONTIGUOUS ALLOCATION
int ContiguousCheck(Block B,int start,int number)
{
Block pos=B->Next;
int i=0;
while(pos!=NULL&&pos->ID!=start)
pos=pos->Next;
while(pos!=NULL&&i<number)
if(strcmp(pos->Filename,"Free")!=0)
return 0;
else
{
i++;
pos=pos->Next;
if(pos==NULL)
return 0;
}
return 1;
}
```

```
void ContiguousAllocate(Block B,char FileName[],int start,int number)
{
Block pos=B;
int i;
while(pos!=NULL&&pos->ID!=start)
pos=pos->Next;
for(i=0;i<number;i++)
{
strcpy(pos->Filename,FileName);
pos=pos->Next;
} return;
}
//LINKED ALLOCATION
int LinkedCheck(Block B,int BlockNo)
{
Block pos=B->Next;
while(pos!=NULL&pos->ID!=BlockNo)
pos=pos->Next;
if(strcmp(pos->Filename,"Free")==0)
return 1;
else return 0;
}
Block LinkedAllocate(Block B,char Filename[],int BlockNo,Block prevMemBlock)
{
Block pos=B;
while(pos!=NULL&&pos->ID!=BlockNo)
pos=pos->Next;
strcpy(pos->Filename,Filename);
if(prevMemBlock!=NULL)
prevMemBlock->LinkedNext=pos;
return pos;
void DisplayLinkedList(Block B,int start,int end)
{
Block pos=B;
while(pos!=NULL&&pos->ID!=start)
pos=pos->Next;
while(1)
{
printf("->Block-%d ",pos->ID);
if(pos->ID==end)
return;
pos=pos->LinkedNext;
} return;
}
//INDEXED ALLOCATION
void IndexTableAllocate(Block B,int BlockNo)
{
Block pos=B;
while(pos!=NULL&&pos->ID!=BlockNo)
pos=pos->Next;
strcpy(pos->Filename,"IndexTable");
return;
}
void IndexAllocate(Block B,char Filename[],int BlockNo,int TableNo,int i)
{
Block in,pos=B;
while(pos!=NULL&&pos->ID!=BlockNo)
pos=pos->Next;
strcpy(pos->Filename,Filename);
in=B;
while(in!=NULL&&in->ID!=TableNo)
in=in->Next;
```

```
in->Table.Address[i]=pos;
return;
}
void DisplayIndexedBlock(Block B,int IndexNo,int length)
{
Block pos=B,in;
int i;
while(pos!=NULL&&pos->ID!=IndexNo)
pos=pos->Next;
for(i=0;i<length;i++)
printf("DataBlock%d,",pos->Table.Address[i]->ID);
printf("\b");
return;
}
void main()
{
int i,j,random,check,MemSize,BlockSize,BlockNo,FileNo,option,FileSize,count;
struct Directory D[20];
Block B,prevMemBlock;
B=BCreate();
printf("\nFILE ALLOCATION\n");
printf("Enter the Memory Size(in Kb): ");
scanf("%d",&MemSize);
printf("Enter the Block Size(in Kb): ");
scanf("%d",&BlockSize);
BlockNo=MemSize/BlockSize;
printf("Number of Memory Blocks available: %d\n",BlockNo);
for(i=0;i<BlockNo;i++)
BlockInsert(B,i);
printf("Enter the number of files: ");
scanf("%d",&FileNo);
printf("\n");
for(i=0;i<FileNo;i++)
{
printf("File Name and Size(in Kb): ");
scanf("%s%d",D[i].Filename,&FileSize);
D[i].length=FileSize/BlockSize;
}
printf("\nFILE ALLOCATION TECHINQUE:\n1. CONTIGUOUS ALLOCATION\n2. LINKED
ALLOCATION\n3. INDEXED ALLOCATION\nEnter an option: ");
scanf("%d",&option);
switch(option)
{
case 1://CONTIGUOUS ALLOCATION
printf("\nCONTIGUOUS ALLOCATION\n");
for(i=0;i<FileNo;i++)
{
count=0;
CALLOCATE:
random=rand()%BlockNo;
check=ContiguousCheck(B,random,D[i].length);
if(check)
{
ContiguousAllocate(B,D[i].Filename,random,D[i].length);
D[i].startBlock=random;
} else
{
count++;
if(count>=5)
{
printf("\nMemory not available for %s",D[i].Filename);
D[i].startBlock=-1;
```

```
} else
goto CALLOCATE;
}
}
printf("\n\n DIRECTORY\n FILE NAME START BLOCK LENGTH");
for(i=0;i<FileNo;i++)
printf("\n %9s %11d %6d",D[i].Filename,D[i].startBlock,D[i].length);
printf("\n");
break;
case 2://LINKED ALLOCATION
printf("\nLINKED ALLOCATION\n");
for(i=0;i<FileNo;i++)
{
prevMemBlock=NULL;
for(j=0;j<D[i].length;j++)
{
LALLOCATE:
random=rand()%BlockNo;
check=LinkedCheck(B,random);
if(check)
{
if(j==0) D[i].startBlock=random;
if(j==(D[i].length-1)) D[i].endBlock=random;
prevMemBlock=LinkedAllocate(B,D[i].Filename,random,prevMemBlock);
} else
goto LALLOCATE;
}
}
printf("\n\n DIRECTORY\n FILE NAME START BLOCK END BLOCK");
for(i=0;i<FileNo;i++)
printf("\n %9s %11d %9d",D[i].Filename,D[i].startBlock,D[i].endBlock);
printf("\n");
for(i=0;i<FileNo;i++)
{
printf("\n%s :",D[i].Filename);
DisplayLinkedList(B,D[i].startBlock,D[i].endBlock);
}
break;
case 3://INDEXED ALLOCATION
printf("\nINDEXED ALLOCATION\n");
for(i=0;i<FileNo;i++)
{
TABLEALLOCATE:
random=rand()%BlockNo;
check=LinkedCheck(B,random);
if(check)
{
IndexTableAllocate(B,random);
D[i].IndexBlock=random;
} else
goto TABLEALLOCATE;
for(j=0;j<D[i].length;j++)
{
IALLOCATE:
random=rand()%BlockNo;
check=LinkedCheck(B,random);
if(check)
IndexAllocate(B,D[i].Filename,random,D[i].IndexBlock,j);
else
goto IALLOCATE;
}
}
```

```
printf("\n\n DIRECTORY\n FILE NAME INDEX BLOCK");
for(i=0;i<FileNo;i++)
printf("\n %9s %11d",D[i].Filename,D[i].IndexBlock);
printf("\n\n FILE BLOCK TABLE\n FILE NAME BLOCK INDEXED");
for(i=0;i<FileNo;i++)
{
printf("\n %9s ",D[i].Filename);
DisplayIndexedBlock(B,D[i].IndexBlock,D[i].length);
}
break;
}
}
```

```
kaushik@kaushik-AcerPower-Series:~/OS $ gcc -o -f filealloc.c
kaushik@kaushik-AcerPower-Series:~/OS $./f
FILE ALLOCATION
Enter the Memory Size(in Kb): 64
Enter the Block Size(in Kb): 2
Number of Memory Blocks available: 32
Enter the number of files: 5
File Name and Size(in Kb): fil1.# ## #e1 10
File Name and Size(in Kb): file2 6
File Name and Size(in Kb): file3 12
File Name and Size(in Kb): file4 20
File Name and Size(in Kb): file5 8
FILE ALLOCATION TECHINQUE:
1. CONTIGUOUS ALLOCATION
2. LINKED ALLOCATION
3. INDEXED ALLOCATION
Enter an option: 1
CONTIGUOUS ALLOCATION
Memory not available for file4
DIRECTORY
FILE NAME START BLOCK LENGTH
file1 7 5
file2 19 3
file3 12 6
file4 -1 10
file5 27 4

kaushik@kaushik-AcerPower-Series:~/OS $./f
FILE ALLOCATION
Enter the Memory Size(in Kb): 64
Enter the Block Size(in Kb): 2
Number of Memory Blocks available: 32
Enter the number of files: 5
File Name and Size(in Kb): file1 10
File Name and Size(in Kb): file2 6
File Name and Size(in Kb): file3 12
File Name and Size(in Kb): file4 20
File Name and Size(in Kb): file5 8
FILE ALLOCATION TECHINQUE:
1. CONTIGUOUS ALLOCATION
2. LINKED ALLOCATION
3. INDEXED ALLOCATION
Enter an option: 2
LINKED ALLOCATION
DIRECTORY
FILE NAME START BLOCK END BLOCK
file1 7 17
file2 31 12
file3 13 3
```

file4 28 29
file5 30 15
file1 :->Block-7 ->Block-6 ->Block-9 ->Block-19 ->Block-17
file2 :->Block-31 ->Block-10 ->Block-12
file3 :->Block-13 ->Block-26 ->Block-11 ->Block-18 ->Block-27 ->Block-3
file4 :->Block-28 ->Block-2 ->Block-20 ->Block-24 ->Block-8 ->Block-22 ->Block-14 ->Block-23 ->Block-5 ->Block-29
file5 :->Block-30 ->Block-1 ->Block-16 ->Block-15

kaushik@kaushik-AcerPower-Series:~/OS $ ./f
FILE ALLOCATION
Enter the Memory Size(in Kb): 64
Enter the Block Size(in Kb): 2
Number of Memory Blocks available: 32
Enter the number of files: 5
File Name and Size(in Kb): file1 10
File Name and Size(in Kb): file2 6
File Name and Size(in Kb): file3 12
File Name and Size(in Kb): file4 20
File Name and Size(in Kb): file5 8
FILE ALLOCATION TECHINQUE:
1. CONTIGUOUS ALLOCATION
2. LINKED ALLOCATION
3. INDEXED ALLOCATION
Enter an option: 3
INDEXED ALLOCATION
DIRECTORY
FILE NAME INDEX BLOCK
file1 7
file2 10
file3 11
file4 24
file5 21
FILE BLOCK TABLE
FILE NAME BLOCK INDEXED
file1 DataBlock6,DataBlock9,DataBlock19,DataBlock17,DataBlock31
file2 DataBlock12,DataBlock13,DataBlock26
file3 DataBlock18,DataBlock27,DataBlock3,DataBlock28,DataBlock2,DataBlock20
file4 DataBlock8,DataBlock22,DataBlock14,DataBlock23,DataBlock5,
DataBlock29,DataBlock30,DataBlock1, DataBlock16, DataBlock15
file5 DataBlock25,DataBlock0,DataBlock4,DataBlock0
kaushik@kaushik-AcerPower-Series:~/OS $