```
Script started on 2020-04-07 00:16:13+0530
 ]0;GAYU@GAYU: ~/Desktop/fs  [01;32mGAYU@GAYU [00m: [01;34m~/Desktop/fs [00m$ gcc floc.c -o f
 ]0;GAYU@GAYU: ~/Desktop/fs  [01;32mGAYU@GAYU [00m: [01;34m~/Desktop/fs [00m$ cat floc.c
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<time.h>
int main_memsize;
int block_size;
int n_blocks;
int free_count;
typedef struct MainMem
{
 int block_id;
 char filename[20];
 struct MainMem* next;
 struct MainMem* nextFB;
 int block_table[20];
}M;
typedef struct files
{
 char filename[20];
 int start;
 int end;
 struct files* next;
}F;
F* create2()
{
 F* head = (F*)malloc(sizeof(F));
 head->next = NULL;
 return head;
}

M* create()
{
 M* head = (M*)malloc(sizeof(M));
 head->next = NULL;
 return head;
}
void insertFile(F* head, char filename[20],int start,int end)
{
 F* newNode = (F*)malloc(sizeof(F));
 strcpy(newNode->filename,filename);
 newNode->start = start;
 newNode->end = end;
 F* temp = head;
 while(temp->next!=NULL)
  temp = temp->next;
 newNode->next = temp->next;
 temp->next = newNode;
}
void insertLast(M* head, int block_id,char filename[20],int bc[20])
{
 M* temp = head;
 M* newNode = (M*)malloc(sizeof(M));
 newNode->block_id = block_id;
 for(int i=0;i<20;i++)
  newNode->block_table[i] = bc[i];
 strcpy(newNode->filename,filename);
 while(temp->next!=NULL)
  temp = temp->next;
 newNode->next = temp->next;
 temp->next = newNode;
}
void display(F* flist) // mlist parameter not required
{
 F* temp2 = flist->next;
 printf("\n\n\t\tFILE LIST \n");
 while(temp2!=NULL)
 {
  printf("Filename : %s\t start : %d\t end : %d\n",temp2->filename,temp2->start,temp2->end);
  temp2 = temp2->next;
 }
}
void display2(M* dblist, F* flist)
{
 M* temp1 = dblist->next;
 M* tempi;
 F* temp2 = flist->next;
 printf("\n\n\tDIRECTORY \n");
 while(temp2!=NULL)
 {
  printf("File Name : %s\t Start : %d\t End : %d\n",temp2->filename,temp2->start,temp2->end);
  temp2 = temp2->next;
 }

 printf("Individual File listing \n");
 while(temp1!=NULL)
 {
  if(strcmp(temp1->filename,"Empty")!=0)
  {
   tempi = temp1->nextFB;
   printf("%s DataBlock %d",temp1->filename,temp1->block_id);
   while(tempi!=NULL)
   {
    printf("   DataBlock %d",tempi->block_id);
    tempi = tempi->nextFB;
   }
   printf("\n");
  }
  temp1 = temp1->next;
 }
}
void display3(M* mlist)
{
 M* temp1 = mlist->next;
 printf("Filename\t\tBlock Indexed\n");
 while(temp1!=NULL)
 {
  printf("%s\t\t\tDatablock %d\n",temp1->filename,temp1->block_id);
  for(int i=0;temp1->block_table[i]!=-1;i++)
   printf("\t\t\tDatablock %d\n",temp1->block_table[i]);
  temp1 = temp1->next;
 }
}
void contiguous(M* mlist, F* flist, char filename[20], int b_reqd)
{
        M* temp;
        M* freetemp;
        int i,j,k,rnum;
        int error = 0;
        srand(time(0));
        int check=1;
        int c =0;
        if(free_count < b_reqd)
 {
  printf("Not enough free memory ! Exiting \n");
  return;
 }

        while(check && free_count!=0)
        {
         rnum = (rand() % (n_blocks));
         temp = mlist->next;
          j = 0;
          while(j<rnum && temp!=NULL)
           {
            temp = temp->next;
            j++;
           }
          k = 0;
          freetemp = temp;
          int flag = 0;
          while(k < b_reqd)
           {
           if(strcmp(temp->filename,"Empty")!=0)
            {
             flag = 1;
             break;
            }
           temp = temp->next;
           k++;
           }
          if(flag == 1)
           continue;
          else
           {
  for(int l=0;l<b_reqd; l++)
   {
   strcpy(freetemp->filename,filename);
   freetemp = freetemp->next;
   c++;
  }
  insertFile(flist,filename,rnum,rnum+b_reqd-1);
  error = 1;
  break;
 }
 if(c == b_reqd)
  check = 0;
        }
 if(error == 0)
  printf("Not enough memory !\n");
 display(flist);
}
void linked(M* a_pool, M* mlist, F* flist, char filename[20], int b_reqd)
{
 int check = 1,rnum;
 M* temp;
 M* nFB;
 int base;
 int bc[20];
 int c=0;
 srand(time(0));
 if(free_count < b_reqd)
 {
  printf("Not enough free memory ! Exiting \n");
  return;
 }

 for(int i=0; i<b_reqd && free_count!=0;i++)
 {
  rnum = rand() % (n_blocks);
  temp = mlist->next;
  int i=0;
  while(i< rnum)
  {
   temp = temp->next;
   i++;
  }
  if(strcmp(temp->filename,"Empty")==0)
   {
    bc[c++] = temp->block_id;
    strcpy(temp->filename,filename);
    free_count--;
    break;
   }
 }
 //temp holds the value of first empty memory location
 int ncount = 0;
 while(check && free_count!=0)
 {
  rnum = rand() % (n_blocks);
  nFB = mlist->next;
  int i=0;
  while(i< rnum)
  {
   nFB = nFB->next;
   i++;
  }
  if(strcmp(nFB->filename,"Empty")==0)
   {
    bc[c++] = nFB->block_id;
    strcpy(nFB->filename,filename);
    free_count--;
    temp->nextFB = nFB;
    temp = nFB;
   }
  if(c == b_reqd)
  { check=0;
   insertFile(flist,filename,bc[0],bc[b_reqd-1]);
  }
 }
 if(check==1)
  printf("Not enough Memory ! \n");
 display2(mlist,flist);
}


void indexed(M* b_pool, M* mlist, F* flist, char filename[20], int b_reqd)
{
 int check = 1,rnum;
 M* temp;
 M* base;
 int bc[20];
 int c=0;
 srand(time(0));
 if(free_count < b_reqd)
 {
  printf("Not enough free memory ! Exiting \n");
  return;
 }
 for(int i=0; i<b_reqd && free_count!=0;i++)
 {
  rnum = rand() % (n_blocks);
  temp = mlist->next;
  int i=0;
  while(i< rnum)
  {
   temp = temp->next;
   i++;
  }
  if(strcmp(temp->filename,"Empty")==0)
   {
    strcpy(temp->filename,filename);
    free_count--;
    b_reqd--;
    base = temp;
    break;
   }
 }
 //temp holds the value of first empty memory location
 int ncount = 0;
 while(check && free_count!=0)
 {
  rnum = rand() % (n_blocks);
  temp = mlist->next;
  int i=0;
  while(i< rnum)
  {
   temp = temp->next;
   i++;
  }
  if(strcmp(temp->filename,"Empty")==0)
   {
    bc[c++] = temp->block_id;
    strcpy(temp->filename,filename);
    free_count--;
   }
  if(c == b_reqd)
  { check=0;
   for(i=b_reqd; i<20;i++)
    bc[i] = -1;
  insertFile(flist,filename,bc[0],bc[b_reqd-1]);
  insertLast(b_pool,base->block_id,base->filename,bc);
  }
 }
 if(check==1)
  printf("Not enough Memory ! \n");
 display3(b_pool);
}

void main()
{
 int fsize;
 int choice;
 int b_reqd;
 char filename[20];
 int bc[20];
 for(int i= 0; i < 20; i++)
  bc[i] = -1;
 M* memory_list = create();
 F* file_list = create2();
 M * a_pool =create();
 M * b_pool =create();
 printf("\t\tFILE ALLOCATION TECHNIQUES\n");
 printf("Main Memory Size : ");
 scanf("%d",&main_memsize);
 printf("\nBlock size : ");
 scanf("%d",&block_size);
 n_blocks = main_memsize / block_size;
 free_count = n_blocks;
 for(int i = 0; i < n_blocks; i++) // creating empty partitions
  insertLast(memory_list,i,"Empty",bc);
 do
 {
  printf("Enter file name : ");
  scanf("%s",filename);
  printf("\nFile Size : ");
  scanf("%d",&fsize);
  b_reqd = fsize/block_size;
  printf("1. Contiguos Allocation \n");
  printf("2. Linked Allocation \n");
  printf("3. Indexed Allocation \n");
  printf("4. Enter choice : ");
  scanf("%d",&choice);
  switch(choice)
  {
   case 1: contiguous(memory_list,file_list,filename,b_reqd);
           break;
   case 2: linked(a_pool,memory_list,file_list,filename,b_reqd);
           break;
   case 3: indexed(b_pool,memory_list,file_list,filename,b_reqd);
           break;
  }
 }while(choice!=4);
}
```

```
 ]0;GAYU@GAYU: ~/Desktop/fs  [01;32mGAYU@GAYU [00m: [01;34m~/Desktop/fs [00m$ ./f
  FILE ALLOCATION TECHNIQUES
Main Memory Size : 1   200

Block size : 10
Enter file name : Prasanna

File Size : rama            5   50
1. Contiguos Allocation
2. Linked Allocation
3. Indexed Allocation
4. Enter choice : 2


 DIRECTORY
File Name : Prasanna  Start : 2  End : 17
Individual File listing
Prasanna DataBlock 2   DataBlock 11   DataBlock 16   DataBlock 13   DataBlock 17
Prasanna DataBlock 11   DataBlock 16   DataBlock 13   DataBlock 17
Prasanna DataBlock 13   DataBlock 17
Prasanna DataBlock 16   DataBlock 13   DataBlock 17
Prasanna DataBlock 17
Enter file name : Craz            Sama            Raja

File Size : 70
1. Contiguos Allocation
2. Linked Allocation
3. Indexed Allocation
4. Enter choice : 3
Filename  Block Indexed
Raja   Datablock 4
```

```
   Datablock 14
   Datablock 19
   Datablock 7
   Datablock 5
   Datablock 3
   Datablock 15
Enter file name : baby   y   u

File Size : 30
1. Contiguos Allocation
2. Linked Allocation
3. Indexed Allocation
4. Enter choice : 3
Filename  Block Indexed
Raja   Datablock 4
   Datablock 14
   Datablock 19
   Datablock 7
   Datablock 5
   Datablock 3
   Datablock 15
babu   Datablock 6
   Datablock 18
   Datablock 0
Enter file name : Kronos

File Size : 20
1. Contiguos Allocation
2. Linked Allocation
3. Indexed Allocation
4. Enter choice : 1


  FILE LIST
Filename : Prasanna  start : 2  end : 17
Filename : Raja  start : 14  end : 15
Filename : babu  start : 18  end : 0
Filename : Kronos  start : 9  end : 10
Enter file name : kebab

File Size : 100
1. Contiguos Allocation
2. Linked Allocation
3. Indexed Allocation
4. Enter choice : 1
Not enough free memory ! Exiting
Enter file name : Ora         santhosh

File Size : 100
1. Contiguos Allocation
2. Linked Allocation
3. Indexed Allocation
4. Enter choice : 4
 ]0;GAYU@GAYU: ~/Desktop/fs  [01;32mGAYU@GAYU [00m: [01;34m~/Desktop/fs [00m$ exit
```

```
exit

Script done on 2020-04-07 00:17:51+0530
```