Department of Computer Science and Engineering

S.G.Shivanirudh , 185001146, Semester IV

3 April 2020

UCS1411 - Operating Systems Laboratory

Lab Exercise 11: File Allocation Techniques

Objective:

Develop a C program to implement the various file allocation techniques.

Code:

Q.To write a C program to implement the various file allocation techniques.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>

struct Item{
    char *file_name;
    int size;
    int no_of_blocks;
    int file_block_table[20];
    int start_block_id;
    int end_block_id;
};

typedef struct Item File;

struct Box{
    int block_id;
```

```
    struct Box* next;
    struct Box* next_file_block;
    File f;
};

typedef struct Box Block;

void initialiseFile(File *f){
    f->file_name=(char*)malloc(sizeof(20));
    strcpy(f->file_name,"Free");
    f->no_of_blocks=0;
    for(int i=0;i<20;i++)
        f->file_block_table[i]=-1;
    f->start_block_id=-1;
    f->end_block_id=-1;
}

void initialiseBlock(Block* b){

    b->block_id=-1;
    b->next=NULL;
    b->next_file_block=NULL;
    initialiseFile(&b->f);
}

void acceptFile(File *f){
    printf("\nEnter file name: ");scanf(" %[^\n]",f->file_name);
    printf("\nEnter file size: ");scanf("%d",&f->size);
}

void displayFile(File *f){
    printf("\n%s %d %d\n",f->file_name,f->size,f->no_of_blocks);
}

//Insert into Linked List
void insert(Block *LL, Block *n){

    Block *stmp=LL;
    while(stmp->next){
        stmp=stmp->next;
    }
    n->next=stmp->next;
    stmp->next=n;
}

//Count number of contiguous free blocks from a given block
int countBlock(Block *LL,int block_id){
    Block *tmp=LL;
    tmp=tmp->next;
    while(tmp){
        if(tmp->block_id==block_id)
            break;
        else
            tmp=tmp->next;
    }

    int block_count=1;
```

```
    while(tmp){
        if(strcmp(tmp->f.file_name,"Free")!=0)
            break;
        else{
            block_count++;
            tmp=tmp->next;
        }
    }
    return block_count;
}

void Display(Block *LL){
    Block *tmp=LL;
    tmp=tmp->next;
    while(tmp){
        printf("%d %s\n",tmp->block_id,tmp->f.file_name);
        tmp=tmp->next;
    }
}

struct Dir{
    int no_of_files;
    File file_list[10];
};

typedef struct Dir Directory;

void initialiseDir(Directory *d){
    d->no_of_files=0;
    for(int i=0;i<10;i++)
        initialiseFile(&d->file_list[i]);
}
```

```
#include "LinkedList.h"

//Contiguous Allocation
/*
Logic:
For each file do the following:
1. Generate a random number between 1 to n.
2. Check for continuous number of needed file free blocks starting
    from that random block no.
3. If free then allot that file in those continuous blocks and
    update the directory structure.
4. If the block is not free, repeat from step 1.
5. If not enough continuous blocks are free then flag an error.
6. The Directory Structure should contain Filename, Starting Block,
     length (no. of blocks).
*/
void ContinguousAllocation(Block *MM,Directory *dir,int
    MM_block_count){
    srand(time(0));

    Block *tmp=MM;
    Block *start=tmp;
    int pos=-1;
    int block_count=0;
    int attempt_count=0;
```

```
22
23      for(int i=0;i<dir->no_of_files;i++){
24
25          attempt_count=0;
26          do{
27
28              if(attempt_count>=MM_block_count)
29                  break;
30              attempt_count++;
31              tmp=start->next;
32              pos=rand()%MM_block_count;
33              for(int j=0;j<pos;j++)
34                  tmp=tmp->next;
35              block_count=countBlock(MM,tmp->block_id);
36
37
38          }while(strcmp(tmp->f.file_name,"Free")!=0 || block_count <
        dir->file_list[i].no_of_blocks);
39
40          if(attempt_count>=MM_block_count || pos<0 ){
41
42              printf("\n Error: No enough memory for file %s. \n",dir
        ->file_list[i].file_name);
43              continue;
44          }
45          else{
46
47              for(int j=0;j<dir->file_list[i].no_of_blocks;j++){
48
49                  strcpy(tmp->f.file_name,dir->file_list[i].file_name
        );
50                  tmp=tmp->next;
51              }
52              dir->file_list[i].start_block_id=pos;
53
54          }
55      }
56
57      printf("\n Directory Structure: \n");
58      printf("%-10s|%-12s|%-6s\n","File Name","Start Block","Length")
        ;
59      for(int i=0;i<dir->no_of_files;i++){
60          if( dir->file_list[i].start_block_id>=0)
61              printf("%5s%5s|%6d%6s|%3d%3s\n",dir->file_list[i].
        file_name," ",dir->file_list[i].start_block_id," ",dir->
        file_list[i].no_of_blocks," ");
62      }
63
64  }
65
66  //Linked Allocation
67  /*
68  Logic:
69
70  For each file do the following:
71  1. Generate a random number between 1 to n.
72  2. Check that block is free or not.
```

```
3. If free then allot it for file. Repeat step 1 to 3 for the
    needed number of blocks for the file and
   create linked list in Main memory using the field Link to Next
    File block.
4. Update the Directory entry which contains Filename, Start block
    number, Ending Block Number.
5. Display the file blocks starting from start block number in
    Directory upto ending block number
   by traversing the Main memory Linked list using the field Link
    to Next File block.
*/
void LinkedAllocation(Block *MM,Directory *dir,int MM_block_count){

    srand(time(0));

    Block *tmp=MM;
    Block *start=tmp;
    int pos=-1;
    int attempt_count=0;

    for(int i=0;i<dir->no_of_files;i++){

        for(int j=0;j<dir->file_list[i].no_of_blocks;j++){

            attempt_count=0;
            do{

                if(attempt_count>=MM_block_count)
                    break;
                attempt_count++;
                tmp=start->next;
                pos=rand()%MM_block_count;
                for(int k=0;k<pos;k++)
                    tmp=tmp->next;

            }while(strcmp(tmp->f.file_name,"Free")!=0);

            if(attempt_count>=MM_block_count || pos<0 ){

                printf("\n Error: No enough memory for file %s. \n"
    ,dir->file_list[i].file_name);

                if(j!=0){

                    Block *del=MM;
                    del=del->next;
                    while(del){

                        if(strcmp(del->f.file_name,dir->file_list[i
    ].file_name)==0)
                            strcpy(del->f.file_name,"Free");
                        del=del->next;
                    }
                }
                break;
            }
            else{
```

```
123
124                 strcpy(tmp->f.file_name,dir->file_list[i].file_name
    );
125                 tmp->next_file_block=NULL;
126                 if(j==0){
127                     dir->file_list[i].start_block_id=pos;
128                 }
129                 else{
130                     Block *ins=start->next;
131                     for(int k=0;k<dir->file_list[i].start_block_id;
    k++)
132                         ins=ins->next;
133                     while(ins->next_file_block)
134                         ins=ins->next_file_block;
135
136                     tmp->next_file_block=ins->next_file_block;
137                     ins->next_file_block=tmp;
138                 }
139                 if(j == dir->file_list[i].no_of_blocks-1){
140                     dir->file_list[i].end_block_id=pos;
141                 }
142             }
143         }
144
145     }
146
147     printf("\n Directory Structure: \n");
148     printf("%-10s|%-12s|%-12s\n","File Name","Start Block","End
    Block");
149     for(int i=0;i<dir->no_of_files;i++){
150         if( dir->file_list[i].start_block_id>=0)
151             printf("%5s%5s|%6d%6s|%6d%6s\n",dir->file_list[i].
    file_name," ",dir->file_list[i].start_block_id," ",dir->
    file_list[i].end_block_id," ");
152     }
153
154     for(int i=0;i<dir->no_of_files;i++){
155
156         if(dir->file_list[i].start_block_id>=0){
157
158             Block *tmp=MM;
159             tmp=tmp->next;
160             for(int j=0;j<dir->file_list[i].start_block_id;j++)
161                 tmp=tmp->next;
162             printf("\nFile: %s\n",dir->file_list[i].file_name);
163             while(tmp){
164                 printf("%d ",tmp->block_id);
165                 tmp=tmp->next_file_block;
166             }
167             printf("\n");
168         }
169     }
170
171 }
172
173 //Indexed Allocation
174 /*
```

```
Logic:
For each file do the following:
1. Generate a random number between 1 to n as a block for index block.
2. Check if it is free else repeat index block selection.
3. Generate needed number of free blocks in random order for the file and store those block numbers
   in index block as array in File block table array.
4. Display the Directory structure which contains the filename and index blocknumber. Display the
   File Details by showing the File Block Table in the index block.
*/
void IndexedAllocation(Block *MM,Directory *dir,int MM_block_count){

    srand(time(0));

    Block *tmp=MM;
    Block *start=tmp;
    int pos=-1;
    int attempt_count=0;

    for(int i=0;i<dir->no_of_files;i++){
        int file_block_ctr=0;
        for(int j=0;j<dir->file_list[i].no_of_blocks+1;j++){

            attempt_count=0;
            do{

                if(attempt_count>=MM_block_count)
                    break;
                attempt_count++;
                tmp=start->next;
                pos=rand()%MM_block_count;
                for(int k=0;k<pos;k++)
                    tmp=tmp->next;

            }while(strcmp(tmp->f.file_name,"Free")!=0);

            if(attempt_count>=MM_block_count || pos<0 ){

                printf("\n Error: No enough memory for file %s. \n",dir->file_list[i].file_name);

                if(j!=0){

                    Block *del=MM;
                    del=del->next;
                    while(del){

                        if(strcmp(del->f.file_name,dir->file_list[i].file_name)==0)
                            strcpy(del->f.file_name,"Free");
                        del=del->next;
                    }
                }
                dir->file_list[i].start_block_id=-1;
```

```
226                 break;
227             }
228             else{
229 
230                 strcpy(tmp->f.file_name,dir->file_list[i].file_name
    );
231 
232                 if(j==0){
233                     dir->file_list[i].start_block_id=pos;
234                 }
235                 else{
236                     Block *ins=start->next;
237                     for(int k=0;k<dir->file_list[i].start_block_id;
    k++)
238                         ins=ins->next;
239                     ins->f.file_block_table[file_block_ctr++]=pos;
240                 }
241             }
242         }
243 
244     }
245 
246     printf("\n Directory Structure: \n");
247     printf("%-10s|%-12s\n","File Name","Index Block");
248     for(int i=0;i<dir->no_of_files;i++){
249         if( dir->file_list[i].start_block_id>=0)
250             printf("%5s%5s|%6d%6s\n",dir->file_list[i].file_name,"
    ",dir->file_list[i].start_block_id," ");
251     }
252 
253     for(int i=0;i<dir->no_of_files;i++){
254 
255         if(dir->file_list[i].start_block_id>=0){
256 
257             Block *tmp=MM;
258             tmp=tmp->next;
259             for(int j=0;j<dir->file_list[i].start_block_id;j++)
260                 tmp=tmp->next;
261             printf("\nFile: %s\n",dir->file_list[i].file_name);
262             for(int j=0;j<dir->file_list[i].no_of_blocks;j++)
263                 printf("%d ",tmp->f.file_block_table[j]);
264             printf("\n");
265         }
266     }
267 }
268 
269 void main(){
270     int MMsize;
271     int block_size;
272     int MM_block_count;
273 
274     Directory dir;
275     initialiseDir(&dir);
276 
277     printf("\nEnter size of Main Memory: ");scanf("%d",&MMsize);
278     printf("\nEnter size of block: ");scanf("%d",&block_size);
279 
```

```
    MM_block_count = MMsize/block_size;

    Block *MM=(Block*)malloc(sizeof(Block));
    initialiseBlock(MM);

    for(int i=0;i<MM_block_count;i++){
        Block *tmp=(Block*)malloc(sizeof(Block));
        initialiseBlock(tmp);
        tmp->block_id=i;
        insert(MM,tmp);
    }

    printf("\nEnter the number of files: ");scanf("%d",&dir.
    no_of_files);
    for(int i=0;i<dir.no_of_files;i++){
        acceptFile(&dir.file_list[i]);
        if(dir.file_list[i].size % block_size)
            dir.file_list[i].no_of_blocks = (dir.file_list[i].size
    / block_size)+1;
        else
            dir.file_list[i].no_of_blocks = (dir.file_list[i].size
    / block_size);
    }

    for(int i=0;i<dir.no_of_files;i++){
        displayFile(&dir.file_list[i]);
    }

    int option;
    do{
        printf("\n Select Allocation Algorithm: ");
        printf("\n 1.Contiguous Allocation ");
        printf("\n 2.Linked Allocation \n 3.Indexed Allocation ");
        printf("\n 0.Exit \n Your choice: ");scanf("%d",&option);

        if(option==1){
            ContinguousAllocation(MM,&dir,MM_block_count);

        }
        else if(option==2){
            LinkedAllocation(MM,&dir,MM_block_count);
        }
        else if(option==3){
            IndexedAllocation(MM,&dir,MM_block_count);
        }
        else if(option){
            printf("\n Invalid option. \n");
        }
        else;
    }while(option);
}
```

Output:

```

Enter size of Main Memory: 500

```

```
Enter size of block: 10

Enter the number of files: 3

Enter file name: file1

Enter file size: 203

Enter file name: file2

Enter file size: 154

Enter file name: file3

Enter file size: 50

 Select Allocation Algorithm:
 1.Contiguous Allocation
 2.Linked Allocation
 3.Indexed Allocation
 0.Exit
 Your choice: 1

 Directory Structure:
File Name |Start Block |Length
file1     |    10      | 21
file2     |    34      | 16
file3     |     5      |  5

 Select Allocation Algorithm:
 1.Contiguous Allocation
 2.Linked Allocation
 3.Indexed Allocation
 0.Exit
 Your choice: 2

 Directory Structure:
File Name |Start Block |End Block
file1     |    45      |    39
file2     |     6      |    23
file3     |    30      |    37

File: file1
45 3 25 22 41 38 49 8 36 42 46 35 5 18 31 14 19 21 12 11 39

File: file2
6 2 24 15 9 0 13 16 20 44 40 29 17 32 33 23

File: file3
30 27 48 34 37

 Select Allocation Algorithm:
 1.Contiguous Allocation
 2.Linked Allocation
 3.Indexed Allocation
 0.Exit
 Your choice: 3
```

```
61
62  Directory Structure:
63 File Name |Index Block
64 file1     |     9
65 file2     |    18
66 file3     |    41
67
68 File: file1
69 33 46 17 7 39 28 24 25 1 13 0 26 10 21 15 23 19 40 11 44 45
70
71 File: file2
72 22 12 38 43 47 6 5 8 34 32 42 37 14 48 16 35
73
74 File: file3
75 30 36 31 4 3
76
77  Select Allocation Algorithm:
78  1.Contiguous Allocation
79  2.Linked Allocation
80  3.Indexed Allocation
81  0.Exit
82  Your choice: 0
```