# EX:11 FILE ALLOCATION TECHNIQUES

-S.Vishakan CSE-C 18 5001 196

## SOURCE CODE – CONTIGUOUS ALLOCATION:

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

typedef struct
{ //structure to maintain memory
    int block;
    int not_free;
    char file[50];
} memory;

struct Element
{ //structure to maintain linked list
    int block;
    int size;
    char file[50];
    struct Element *next;
};

typedef struct Element element;

memory mem[100];
element *table = NULL;
int mem_size, block_size, num_blocks;

void initMemory();
void printDirectory();
int checkFreeSpace(int size);
int checkContiguous(int index, int size);
void insertFile(char file[], int size, int block);
void allocateFiles();
int allocateFile(char name[], int size);
void printFreeBlocks();

int main(void)
{
    int opt = 1;

    printf("\n\t\tContiguous Allocation\n");
    while (opt != 0)
```

```
    {
        printf("\n\n\t\tMain Menu\n");
        printf("\n\t1. Initialise Memory\n\t2. Print Free Blocks\n\t3. Allocate Files\n\t4. Print
Directory\n\t0. Exit\n\tYour Choice -> ");
        scanf("%d", &opt);
        switch (opt)
        {
        case 1:
            initMemory();
            break;
        case 2:
            printFreeBlocks();
            break;
        case 3:
            allocateFiles();
            break;
        case 4:
            printDirectory();
            break;
        case 0:
            printf("\n\tThank You!\n");
            break;
        default:
            printf("\nInvalid Option!");
            break;
        }
    }

    return 0;
}

void initMemory()
{ //initialising memory space
    int i = 0;

    printf("\nEnter size of memory in KB: ");
    scanf("%d", &mem_size);
    printf("\nEnter the size of a block in KB: ");
    scanf("%d", &block_size);
    num_blocks = mem_size / block_size;
    printf("\nNo. of Blocks : %d\n", num_blocks);

    for (i = 0; i < num_blocks; i++)
    {
        mem[i].block = i;
        mem[i].not_free = 0;
        strcpy(mem[i].file, "---");
    }
}

void printFreeBlocks()
{ //printing free blocks
```

```
    int i = 0;
    printf("\nFree Blocks: ");
    for (i = 0; i < num_blocks; i++)
    {
        if (mem[i].not_free == 0)
        {
            printf("%d ", i);
        }
    }
    printf("\n");
}

void printDirectory()
{ //displaying the file allocation details
    element *temp = table;

    printf("\n\n-------------------------------------------------");
    printf("\n|\tFile\t|\tBlock\t|\tSize\t|\n");
    printf("-------------------------------------------------\n");

    while (temp != NULL)
    {
        printf("|\t%s\t|\t%d\t|\t%d\t|\n", temp->file, temp->block, temp->size);
        temp = temp->next;
    }

    printf("-------------------------------------------------\n");
}

int checkFreeSpace(int size)
{ //checking for contiguous free space
    int i = 0, j = 0;

    for (i = 0; i < num_blocks;)
    {
        if (mem[i].not_free == 0)
        {
            int j = i;
            while (mem[j].not_free == 0 && j < num_blocks)
            {
                if (j - i + 1 == size)
                {
                    return i;
                }
                j++;
            }
            i += (j + 1);
        }
        else
        {
            i++;
        }
```

```
    }

    return -1; //no free space
}

int checkContiguous(int ind, int size)
{ //checking for contiguous blocks from a given index point
    int i = 0;

    if (mem[ind].not_free == 0)
    {
        for (i = ind; i < ind + size && i < num_blocks; i++)
        {
            if (mem[i].not_free == 1)
            {
                return 0;
            }
        }
    }
    else
    {
        return 0;
    }

    if (ind + size - 1 < num_blocks)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

void insertFile(char file[], int size, int block)
{ //inserting a new file into the linked list
    element *new_node = (element *)malloc(sizeof(element));
    new_node->next = NULL;
    new_node->block = block;
    new_node->size = size;
    strcpy(new_node->file, file);

    if (table == NULL)
    {
        table = new_node;
    }
    else
    {
        new_node->next = table;
        table = new_node;
    }
}
```

```
int allocateFile(char name[], int size)
{ //allocating memory space for a new file
   int flag = 0, blocks = size / block_size;
   int index = 0, i = 0;

   if (size * 1.0 / block_size != blocks)
   {
      blocks++;
   }

   size = blocks;

   if (checkFreeSpace(size) >= 0)
   {
      while (1)
      {
         index = random() % (num_blocks);
         if (checkContiguous(index, size) == 1)
         {
            flag = 1;
            i = index;

            do
            {
               mem[i].not_free = 1;
               strcpy(mem[i].file, name);
               i++;
            } while (i < blocks + index);
            insertFile(name, blocks, index);
            break;
         }
      }
   }

   else
   { //insufficient memory
      printf("\nNot enough memory to allocate File %s.", name);
   }

   return flag;
}

void allocateFiles()
{ //allocating files
   int i = 0;
   char file[50];
   int size, num_files;

   printf("\nEnter the no. of files to allocate: ");
   scanf("%d", &num_files);
```

```
    for (i = 0; i < num_files; i++)
    {
        printf("\nEnter the File Name: ");
        scanf("%s", file);
        printf("\nEnter the Size of the File: ");
        scanf("%d", &size);
        allocateFile(file, size);
    }
}
```

# *OUTPUT – CONTIGUOUS ALLOCATION:*

```
(base) vishakan@Legion:~/Desktop/Operating-Systems/Ex 11 File Allocation Techniques$ gcc ContiguousAllocation.c -o c
(base) vishakan@Legion:~/Desktop/Operating-Systems/Ex 11 File Allocation Techniques$ ./c

                Contiguous Allocation


                Main Menu

        1. Initialise Memory
        2. Print Free Blocks
        3. Allocate Files
        4. Print Directory
        0. Exit
        Your Choice -> 1

Enter size of memory in KB: 20

Enter the size of a block in KB: 2

No. of Blocks : 10


                Main Menu

        1. Initialise Memory
        2. Print Free Blocks
        3. Allocate Files
        4. Print Directory
        0. Exit
        Your Choice -> 2

Free Blocks: 0 1 2 3 4 5 6 7 8 9
```

Main Menu

1. Initialise Memory
2. Print Free Blocks
3. Allocate Files
4. Print Directory
0. Exit

Your Choice -> 3

Enter the no. of files to allocate: 3

Enter the File Name: A

Enter the Size of the File: 4

Enter the File Name: B

Enter the Size of the File: 4

Enter the File Name: C

Enter the Size of the File: 4

Main Menu

1. Initialise Memory
2. Print Free Blocks
3. Allocate Files
4. Print Directory
0. Exit

Your Choice -> 2

Free Blocks: 0 5 8 9

Main Menu

1. Initialise Memory
2. Print Free Blocks
3. Allocate Files
4. Print Directory
0. Exit

Your Choice -> 4

| File | Block | Size |
|---|---|---|
| C | 1 | 2 |
| B | 6 | 2 |
| A | 3 | 2 |

Main Menu

1. Initialise Memory
2. Print Free Blocks
3. Allocate Files
4. Print Directory
0. Exit

Your Choice -> 0

Thank You!

# *SOURCE CODE – LINKED ALLOCATION:*

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

struct List
{ //structure for maintaining linked list of blocks
   int block;
   struct List *next;
};

typedef struct List list;

typedef struct
{ //structure for maintaining file details
   char name[50];
   int size;
   int blocks;
   list *head;
} file;

int free_blocks[100], mem_size;
int no_free, no_blocks, no_files, block_size;

file files[100];

void initMemory();
list *makeNode(int block);
void printFreeBlocks();
void allocateFiles();
void printDirectory();
void printList(list *head);

int main(void)
{
   int opt = 1;

   printf("\n\t\tLinked Allocation\n");
   while (opt != 0)
   {
      printf("\n\n\t\tMain Menu\n");
      printf("\n\t1. Initialise Memory\n\t2. Print Free Blocks\n\t3. Allocate Files\n\t4. Print
Directory\n\t0. Exit\n\tYour Choice -> ");
      scanf("%d", &opt);
      switch (opt)
      {
      case 1:
         initMemory();
         break;
      case 2:
```

```
            printFreeBlocks();
            break;
        case 3:
            allocateFiles();
            break;
        case 4:
            printDirectory();
            break;
        case 0:
            printf("\n\tThank You!\n");
            break;
        default:
            printf("\nInvalid Option!");
            break;
        }
    }

    return 0;
}

list *makeNode(int block)
{ //making a new node for a block
    list *new_node = (list *)malloc(sizeof(list));
    new_node->block = block;
    new_node->next = NULL;
    return new_node;
}

void printList(list *head)
{ //printing list of blocks for a file
    list *temp;

    printf("\nBlock List: ");
    for (temp = head; temp != NULL; temp = temp->next)
    {
        printf("%d", temp->block);
        if (temp->next != NULL)
        {
            printf(" -> ");
        }
    }
    printf("\n");
}

void printFreeBlocks()
{ //printing free blocks
    int i = 0;

    printf("\nFree Blocks: ");
    for (i = 0; i < no_blocks; i++)
    {
        if (free_blocks[i] == 0)
```

```
        {
            printf("%d ", i);
        }
    }
    printf("\nFree Space: %d KB", no_free * block_size);
}

void printDirectory()
{ //printing the directory info
    int i = 0;

    printf("\nDirectory Structure: \n");
    for (i = 0; i < no_files; i++)
    {
        printf("\nFile : %s", files[i].name);
        printList(files[i].head);
        printf("\n");
    }
}

void initMemory()
{ //initialising memory space
    int i = 0, rand_block = 0;

    printf("\nEnter the size of memory in KB: ");
    scanf("%d", &mem_size);
    printf("\nEnter the size of a block in KB: ");
    scanf("%d", &block_size);

    no_blocks = mem_size / block_size;
    no_free = no_blocks;
    printf("\nNo. of blocks : %d", no_blocks);
}

void allocateFiles()
{ //allocating space for files
    int i = 0, j = 0, rand_block = 0, count = 0;

    if (no_free == 0)
    {
        printf("\nMemory space exhausted!\n");
        return;
    }

    printf("\nEnter the no. of Files: ");
    scanf("%d", &no_files);

    for (i = 0; i < no_files; i++)
    {
        printf("\nEnter the File Name: ");
        scanf("%s", files[count].name);
        printf("\nEnter the File Size in KB: ");
```

```
        scanf("%d", &files[count].size);

        files[count].blocks = files[count].size / block_size;

        if (files[count].size * 1.0 / block_size > files[count].blocks)
        {
           files[count].blocks++;
        }

        if (files[count].blocks > no_blocks)
        { //unavailability of blocks
           printf("\nCannot allocate this file due to insufficient memory.\n");
           i--;
        }

        else
        { //making a list of blocks for the file
           list *temp, *prev;
           for (j = 0; j < files[count].blocks; j++)
           {
              rand_block = random() % no_blocks;
              if (free_blocks[rand_block] == 0)
              {
                 free_blocks[rand_block] = 1;
                 no_free--;
                 temp = makeNode(rand_block);
                 if (j == 0)
                 { //init. the header node
                    files[count].head = temp;
                    prev = files[count].head;
                 }
                 else
                 { //enqueue the other nodes
                    prev->next = temp;
                    prev = temp;
                 }
              }
              else
              {
                 j--;
              }
           }
           count++;
        }
        if (no_free == 0)
        { //if memory space is exhausted
           printf("\nMemory space exhausted!\n");
           no_files = count;
           break;
        }
     }
  }
```

# *OUTPUT – LINKED ALLOCATION:*

```
(base) vishakan@Legion:~/Desktop/Operating-Systems/Ex 11 File Allocation Techniques$ gcc LinkedAllocation.c -o l
(base) vishakan@Legion:~/Desktop/Operating-Systems/Ex 11 File Allocation Techniques$ ./l

        Linked Allocation


        Main Menu

    1. Initialise Memory
    2. Print Free Blocks
    3. Allocate Files
    4. Print Directory
    0. Exit
    Your Choice -> 1

Enter the size of memory in KB: 20

Enter the size of a block in KB: 2

No. of blocks : 10

        Main Menu

    1. Initialise Memory
    2. Print Free Blocks
    3. Allocate Files
    4. Print Directory
    0. Exit
    Your Choice -> 2

Free Blocks: 0 1 2 3 4 5 6 7 8 9
Free Space: 20 KB

        Main Menu

    1. Initialise Memory
    2. Print Free Blocks
    3. Allocate Files
    4. Print Directory
    0. Exit
    Your Choice -> 3

Enter the no. of Files: 3

Enter the File Name: A

Enter the File Size in KB: 10
```

Enter the File Name: B

Enter the File Size in KB: 5

Enter the File Name: C

Enter the File Size in KB: 2

Main Menu

1. Initialise Memory
2. Print Free Blocks
3. Allocate Files
4. Print Directory
0. Exit

Your Choice -> 4

Directory Structure:

File : A
Block List: 3 -> 6 -> 7 -> 5 -> 2

File : B
Block List: 9 -> 1 -> 0

File : C
Block List: 8

Main Menu

1. Initialise Memory
2. Print Free Blocks
3. Allocate Files
4. Print Directory
0. Exit

Your Choice -> 0

Thank You!

# *SOURCE CODE – INDEXED ALLOCATION:*

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

typedef struct
{ //struct for maintaining a file
   char name[50];
   int size;
   int blocks;
   int index;
} file;

typedef struct
{ //struct to maintain indexing
   int block_id;
   int blocks[100];
} index_block;

index_block indexer[100];
file files[100];

int free_blocks[100], mem_size;
int block_size, no_files, no_blocks, no_free;

void initMemory();
void printBlocks(int ind);
void allocateFiles();
void printFreeBlocks();
void printDirectory();
void printIndexBlock();

int main(void)
{
   int opt = 1;

   printf("\n\t\tIndexed Allocation\n");
   while (opt != 0)
   {
      printf("\n\n\t\tMain Menu\n");
      printf("\n\t1. Initialise Memory\n\t2. Print Free Blocks\n\t3. Allocate Files\n\t4. Print
Directory\n\t5. Print Index Blocks\n\t0. Exit\n\tYour Choice -> ");
      scanf("%d", &opt);
      switch (opt)
      {
      case 1:
         initMemory();
         break;
      case 2:
         printFreeBlocks();
```

```
            break;
        case 3:
            allocateFiles();
            break;
        case 4:
            printDirectory();
            break;
        case 5:
            printIndexBlock();
            break;
        case 0:
            printf("\n\tThank You!\n");
            break;
        default:
            printf("\nInvalid Option!");
            break;
        }
    }

    return 0;
}

void printBlocks(int ind)
{ //printing block information
    int j = 0;

    printf("\n\tIndex Block No. : %d\n\tData Blocks : ", indexer[ind].block_id);
    for (j = 0; j < files[ind].blocks; j++)
    {
        printf("%d ", indexer[ind].blocks[j]);
    }
}

void printFreeBlocks()
{ //printing free blocks
    int i = 0;

    printf("\nFree Blocks: ");
    for (i = 0; i < no_blocks; i++)
    {
        if (free_blocks[i] == 0)
        {
            printf("%d ", i);
        }
    }
    printf("\nFree Space: %d KB", no_free * block_size);
}

void initMemory()
{ //initialising memory
    int i = 0, rand_block = 0;
```

```
    printf("\nEnter the size of memory in KB: ");
    scanf("%d", &mem_size);
    printf("\nEnter the size of a block in KB: ");
    scanf("%d", &block_size);

    no_blocks = mem_size / block_size;
    no_free = no_blocks;
    printf("\nNo. of blocks : %d", no_blocks);

    /*for (i = 0; i <= no_blocks / 3; i++)
    {
        rand_block = random() % no_blocks;
        if (free_blocks[rand_block] == 1)
        {
            i--;
        }
        else
        {
            free_blocks[rand_block] = 1;
            no_free--;
        }
    }*/
}

void allocateFiles()
{ //allocating files to blocks
    int i, rand_block, count = 0, j = 0;

    if (no_free == 0)
    {
        printf("\nMemory space exhausted!\n");
        return;
    }

    printf("\nEnter the no. of files: ");
    scanf("%d", &no_files);

    for (i = 0; i < no_files; i++)
    {
        printf("\nEnter the Name of File %d: ", i + 1);
        scanf("%s", files[count].name);
        printf("\nEnter Size of File %d in KB: ", i + 1);
        scanf("%d", &files[count].size);

        files[count].blocks = files[count].size / block_size;
        if (files[count].size * 1.0 / block_size > files[count].blocks)
        {
            files[count].blocks++;
        } //finding appropriate no. of blocks for the file

        if (files[count].blocks + 1 > no_free)
        {
```

```
            printf("\nCannot allocate file %d\n", i + 1);
            i--;
        }
        else
        {
            do
            {
                rand_block = random() % no_blocks;
            } while (free_blocks[rand_block] == 1);

            indexer[count].block_id = rand_block; //choosing the index block
            files[count].index = rand_block;
            free_blocks[rand_block] = 1;
            no_free--;

            for (j = 0; j < files[count].blocks; j++)
            { //allocating the file's blocks
                rand_block = random() % no_blocks;
                if (free_blocks[rand_block] == 0)
                {
                    free_blocks[rand_block] = 1;
                    no_free--;
                    indexer[count].blocks[j] = rand_block;
                }
                else
                {
                    j--;
                }
            }
            count++;
        }

        if (no_free == 0)
        {
            printf("\nMemory space exhausted!\n");
            no_files = count;
            break;
        }
    }
}

void printDirectory()
{ //printing out directory info
    int i = 0;

    printf("\nFile Allocation: \n");
    for (i = 0; i < no_files; i++)
    {
        printf("\nFile: %s\n", files[i].name);
        printBlocks(i);
        printf("\n");
    }
```

```
}

void printIndexBlock()
{ //printing out index blocks
    int i = 0;
    printf("\n\tFile Indexing\n");
    printf("\n---------------------------------");
    printf("\n|\tFile\t|\tIndex\t|");
    printf("\n---------------------------------");
    for (i = 0; i < no_files; i++)
    {
        printf("\n|\t%s\t|\t%d\t|", files[i].name, files[i].index);
    }
    printf("\n---------------------------------\n");
}
```

# *OUTPUT – INDEXED ALLOCATION:*

```
(base) vishakan@Legion:~/Desktop/Operating-Systems/Ex 11 File Allocation Techniques$ gcc IndexedAllocation.c -o i
(base) vishakan@Legion:~/Desktop/Operating-Systems/Ex 11 File Allocation Techniques$ ./i

            Indexed Allocation


            Main Menu

        1. Initialise Memory
        2. Print Free Blocks
        3. Allocate Files
        4. Print Directory
        5. Print Index Blocks
        0. Exit
        Your Choice -> 1

Enter the size of memory in KB: 20

Enter the size of a block in KB: 2

No. of blocks : 10

            Main Menu

        1. Initialise Memory
        2. Print Free Blocks
        3. Allocate Files
        4. Print Directory
        5. Print Index Blocks
        0. Exit
        Your Choice -> 2
```

Free Blocks: 0 1 2 3 4 5 6 7 8 9
Free Space: 20 KB

Main Menu

1. Initialise Memory
2. Print Free Blocks
3. Allocate Files
4. Print Directory
5. Print Index Blocks
0. Exit
Your Choice -> 3

Enter the no. of files: 3

Enter the Name of File 1: A

Enter Size of File 1 in KB: 10

Enter the Name of File 2: B

Enter Size of File 2 in KB: 2

Enter the Name of File 3: C

Enter Size of File 3 in KB: 4

Cannot allocate file 3
Enter the Name of File 3: C

Enter Size of File 3 in KB: 3

Cannot allocate file 3
Enter the Name of File 3: C

Enter Size of File 3 in KB: 2

Memory space exhausted!

Main Menu

1. Initialise Memory
2. Print Free Blocks
3. Allocate Files
4. Print Directory
5. Print Index Blocks
0. Exit
Your Choice -> 2

Free Blocks:
Free Space: 0 KB

Main Menu

1. Initialise Memory
2. Print Free Blocks
3. Allocate Files
4. Print Directory
5. Print Index Blocks
0. Exit
Your Choice -> 4

File Allocation:

File: A

Index Block No. : 3
Data Blocks : 6 7 5 2 9

File: B

Index Block No. : 1
Data Blocks : 0

File: C

Index Block No. : 8
Data Blocks : 4

Main Menu

1. Initialise Memory
2. Print Free Blocks
3. Allocate Files
4. Print Directory
5. Print Index Blocks
0. Exit
Your Choice -> 5

File Indexing

| File | Index |
|---|---|
| A | 3 |
| B | 1 |
| C | 8 |

Main Menu

1. Initialise Memory
2. Print Free Blocks
3. Allocate Files
4. Print Directory
5. Print Index Blocks
0. Exit
Your Choice -> 0

Thank You!

*/