**Exp No.** 14 **Name:** PONNURI ANIRUDDHA

**Date**: 08/11/2022 **Reg No:** RA2112704010015

**Sequential and Indexed file Allocation**

**Aim: --**

To implement Sequential and Indexed file Allocation

**Procedure: --**

**Sequential file Allocation:**

Files are normally stored on the disks. So the main problem is how to allocate space to those files. So that disk space is utilized effectively and files can be accessed quickly. Three major strategies of allocating disc space are in wide use. Sequential, indexed and linked.

In this allocation strategy, each file occupies a set of contiguously blocks on the disk. This strategy is best suited. For sequential files, the file allocation table consists of a single entry for each file. It shows the filenames, starting block of the file and size of the file. The main problem with this strategy is, it is difficult to find the contiguous free blocks in the disk and some free blocks could happen between two files

**Indexed Allocation**

In this scheme, a special block known as the Index block contains the pointers to all the blocks occupied by a file. Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block. The directory entry contains the address of the index block.

Advantages:

* This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
* It overcomes the problem of external fragmentation.

Disadvantages:

* The pointer overhead for indexed allocation is greater than linked allocation.
* For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization.

**Algorithm (Sequential File Allocation):**

Step 1: Start the program.

Step 2: Get the number of memory partition and their sizes.

Step 3: Get the number of processes and values of block size for each process.

Step 4: First fit algorithm searches all the entire memory block until a hole which is big enough is encountered. It allocates that memory block for the requesting process.

Step 5: Best-fit algorithm searches the memory blocks for the smallest hole which can be allocated to requesting process and allocates it.

Step 6: Worst fit algorithm searches the memory blocks for the largest hole and allocates it to the process.

Step 7: Analyses all the three memory management techniques and display the best algorithm which utilizes the memory resources effectively and efficiently.

Step 8: Stop the program.

**CODE (Sequential file Allocation)**: --

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <string.h>*

*#define TOTAL\_DISK\_BLOCKS 32*

*#define TOTAL\_DISK\_INODES 8*

*#ifndef MAX*

*#define MAX 15*

*#endif*

*int blockStatus[TOTAL\_DISK\_BLOCKS]; // free = 0*

*int blockStart;*

*struct file\_table {*

*char fileName[20];*

*int startBlock;*

*int fileSize;*

*int allotStatus;*

*};*

*struct file\_table fileTable[TOTAL\_DISK\_BLOCKS - TOTAL\_DISK\_INODES];*

*int AllocateBlocks(int Size) {*

*int i = 0, count = 0, inList = 0, nextBlock = 0;*

*int allocStartBlock = TOTAL\_DISK\_INODES;*

*int allocEndBlock = TOTAL\_DISK\_BLOCKS - 1;*

*// check whether sufficient free blocks are available*

*for (i = 0; i < (TOTAL\_DISK\_BLOCKS - TOTAL\_DISK\_INODES); i++)*

*if (blockStatus[i] == 0)*

*count++;*

*if (count < Size)*

*return 1; // not enough free blocks*

*count = 0;*

*while (count < Size) {*

*nextBlock = (rand() % (allocEndBlock - allocStartBlock + 1)) + allocStartBlock;*

*for (i = nextBlock; i < (nextBlock + Size); i++) {*

*if (blockStatus[i] == 0)*

*count = count + 1;*

*else {*

*count = 0;*

*break;*

*}*

*}*

*}*

*blockStart = nextBlock;*

*for (int i = 0; i < Size; i++) {*

*blockStatus[blockStart + i] = 1;*

*}*

*if (count == Size)*

*return nextBlock; // success*

*else*

*return 1; // not successful*

*}*

*void main() {*

*int i = 0, j = 0, numFiles = 0, nextBlock = 0, ret = 1, totalFileSize = 0;*

*char s[20];*

*//-- -*

*char \*header[] = {"FILE\_fileName", "FILE\_SIZE", "BLOCKS\_OCCUPIED"};*

*printf("File allocation method: SEQUENTIAL\n");*

*printf("Total blocks: %d\n", TOTAL\_DISK\_BLOCKS);*

*printf("File allocation start at block: %d\n", TOTAL\_DISK\_INODES);*

*printf("File allocation end at block: %d\n", TOTAL\_DISK\_BLOCKS - 1);*

*printf("Size (kB) of each block: 1\n\n");*

*printf("Enter no of files: ");*

*scanf("%d", &numFiles);*

*//numFiles = 3;*

*for (i = 0; i < numFiles; i++) {*

*//-- -*

*printf("\nEnter the name of file #%d: ", i+1);*

*scanf("%s", fileTable[i].fileName);*

*printf("Enter the size (kB) of file #%d: ", i+1);*

*scanf("%d", &fileTable[i].fileSize);*

*//strcpy(fileTable[i].fileName, "testfile");*

*srand(1234);*

*ret = AllocateBlocks(fileTable[i].fileSize);*

*//-- -*

*if (ret == 1) {*

*exit(0);*

*} else {*

*fileTable[i].startBlock = ret;*

*}*

*}*

*//-- -*

*printf("\n%\*s %\*s %\*s\n", -MAX, header[0], -MAX, header[1], MAX, header[2]);*

*//Seed the pseudo-random number generator used by rand() with the value seed*

*srand(1234);*

*//-- -*

*for (j = 0; j < numFiles; j++) {*

*printf("\n%\*s %\*d ", -MAX, fileTable[j].fileName, -MAX, fileTable[j].fileSize);*

*for(int k=0;k<fileTable[j].fileSize;k++) {*

*printf("%d%s", fileTable[j].startBlock+k, (k == fileTable[j].fileSize-1) ? "" : "-");*

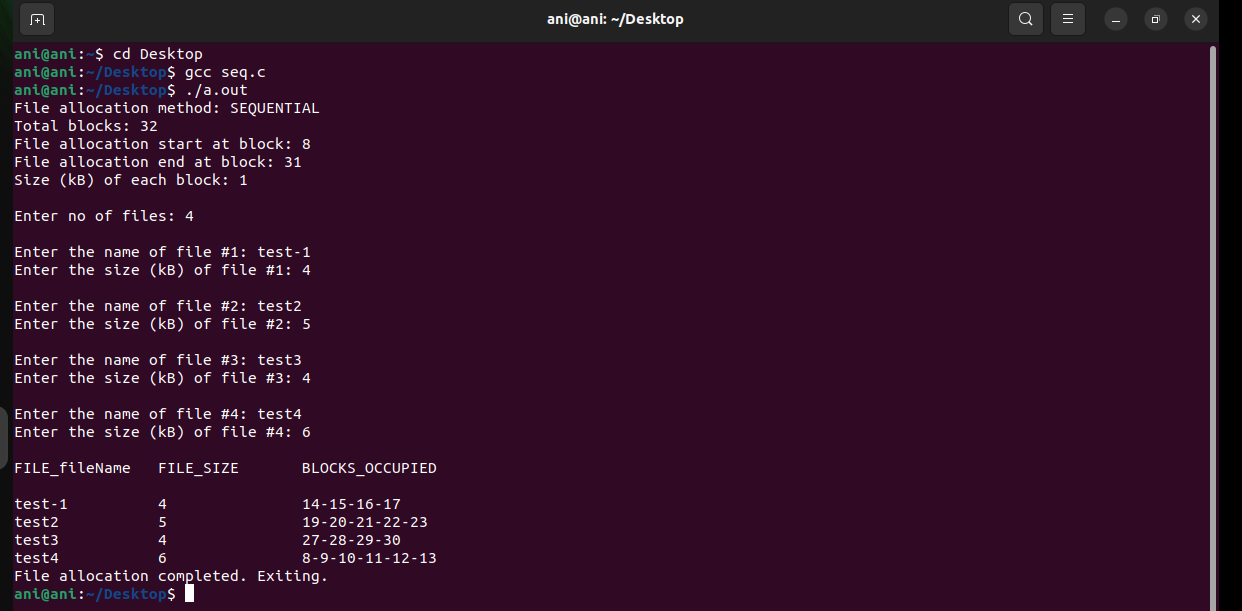
*}*

*}*

*printf("\nFile allocation completed. Exiting.\n");*

*}*

**OUTPUT (Sequential file Allocation): --**



CODE (**Indexed Allocation**): --

*#include<stdio.h>*

*#include<stdlib.h>*

*void main()*

*{*

*int f[50], index[50],i, n, st, len, j, c, k, ind,count=0;*

*for(i=0;i<50;i++)*

*f[i]=0;*

*x:printf("Enter the index block: ");*

*scanf("%d",&ind);*

*if(f[ind]!=1)*

*{*

*printf("Enter no of blocks needed and no of files for the index %d on the disk : \n", ind);*

*scanf("%d",&n);*

*}*

*else*

*{*

*printf("%d index is already allocated \n",ind);*

*goto x;*

*}*

*y: count=0;*

*for(i=0;i<n;i++)*

*{*

*scanf("%d", &index[i]);*

*if(f[index[i]]==0)*

*count++;*

*}*

*if(count==n)*

*{*

*for(j=0;j<n;j++)*

*f[index[j]]=1;*

*printf("Allocated\n");*

*printf("File Indexed\n");*

*for(k=0;k<n;k++)*

*printf("%d-------->%d : %d\n",ind,index[k],f[index[k]]);*

*}*

*else*

*{*

*printf("File in the index is already allocated \n");*

*printf("Enter another file indexed");*

*goto y;*

*}*

*printf("Do you want to enter more file(Yes - 1/No - 0)");*

*scanf("%d", &c);*

*if(c==1)*

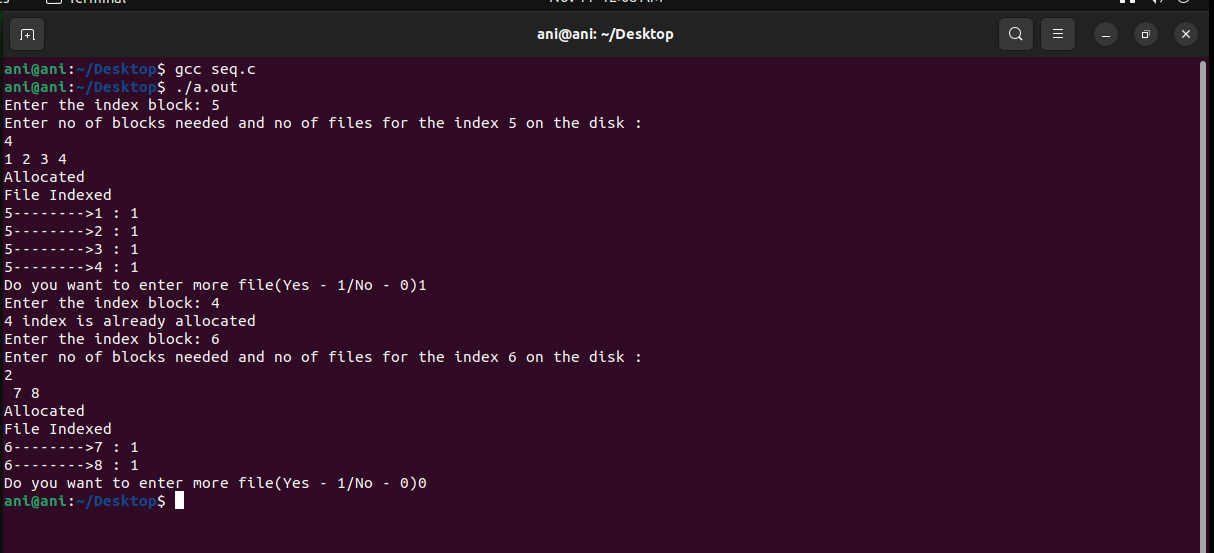
*goto x;*

*else*

*exit(0);*

*}*

**OUTPUT (Indexed Allocation): --**



**RESULT: --**

Successfully implemented Sequential and Indexed file Allocation in c program