CSE316: OPERATING SYSTEMS

CA3-SIMULATION BASED ASSIGNMENT

Final Report

NAME : SAI SASANK LAKIMSETTI

REGISTRATION NUMBER : 12209302

SECTION : K22SR

ROLL NUMBER : 68

GROUP : 2

BRANCH : B.Tech (Computer Science and Engineering)

SCHOOL : COMPUTER SCIENCE AND ENGINEERING

Submitted to:

Akhil Sohal (29462)

Topic: Contiguous Memory Allocation

Question:

A Company has large number of employees who work on various domains and create different types of files such as documents, Excel sheets, and images. Company wants to implement a new file system that can manage the disk space professionally and handle the high volume of file operations. Problem: Create a simulation program that simulates a file system for the Company. The program should include a file system manager that uses a specific file allocation algorithm to allocate space for files on a simulated disk. They should also include a mechanism for deleting, renaming and moving files. The simulation should run for a set amount of time and record the average amount of fragmentation and the number of wasted disk blocks at the end of each time unit. The students should also consider the following factors in their simulation:

• The employees will be creating and editing different types of files (e.g. documents, spreadsheets, images) with varying file sizes.

• The employees will be frequently adding and deleting files.

• The company wants to minimize the amount of wasted disk space. At the end of the simulation, the students should provide a report on their findings and observations, including the performance of the file system under different scenarios and the trade-offs involved in the different file allocation algorithms.

Expected outcomes:

1. File allocation: The program should demonstrate the ability to allocate space for files on a simulated disk using the chosen file allocation algorithm.

2. File deletion and renaming: The program should include a mechanism for deleting and renaming files.

3. Wasted disk space: The program should also record the number of wasted disk blocks at the end of each time unit. Wasted disk space refers to blocks of disk space that are no longer being used by any files.

4. Simulation results: The program should run the simulation for a set amount of time and display the results, including the number ofwasted disk blocks, at the end of each time unit. The students should also experiment with different input scenarios and algorithms to compare the results and see how different factors affect the performance of the file system

Approach:

- Contiguous Allocation of memory in disk states that the every part of the file should be allocated in the contiguous manner in the disk i.e. if a file is storing in the disk then it needs to be present in the disk in contiguous blocks, not in scattered blocks.
- I've used **Array** data structure as it stores data items in contiguous manner.
- In the simulation code, I've used four functions which helps in initializing a disk, allocating file to a disk, deleting file from disk and displaying the blocks of disk whether the block in a disk is empty or allocated with files.
- I've used switch case statements in the code so that it would be easy to do contiguous memory allocation.
- The disk size is fixed i.e. in the code we have defined the disk size earlier.
- A structure is defined which contains the character array to represent file name with maximum limit of 20 characters, two integer variables, one represents the starting block of the disk we want to fit the file in and the other represents the ending block of the disk we want to fit the file in.
- The user need to mention the initial block and ending block of the disk that he/she wants to fit the file in.
- If the file size is greater than the remaining disk size, the file won't be allocated to the disk.

- '**X**' represents that the block is allocated to a particular part of a file/ whole file and '**.**' Represents that the particular block is empty and no part of the file or full file is allocated to it.
- When we want to delete any file from the disk, we need to specify the serial number of file i.e. the order of file in the file list. If it exists in the disk, it will delete the file from disk and empty the space and display message that the file got deleted from the disk. If not, it will display that the file doesn't exist in the file.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define DISK_SIZE 1000
#define MAX_FILES 50
typedef struct File
{
   char filename[20];
   int start;
   int end;
} File;
int disk[DISK_SIZE];
int num_of_files = 0; //initially setting it to zero. later increase and decrease according to the task performed
File files[MAX_FILES]; // an array used to store files that were allocated in

//Function to initialize disk
void initialiseDisk()
{
   for (int i = 0; i < DISK_SIZE; i++)
   {
      disk[i] = -1; //block is empty
   }
}
//Function to allocate files in the disk
int allocateFile(File file)
{
   int file_size = file.end - file.start + 1;
   int remaining_space = 0;
   for (int i = 0; i < DISK_SIZE; i++)
   {
      if (disk[i] == -1)
      {
         remaining_space++;
      }
   }
```

```
    if (file_size > remaining_space)
    {
        printf("ERROR!: File size greater than remaining disk space. No allocation of the file will done.\n");
        return 0;
    }

    for (int block = file.start; block <= file.end; block++)
    {
        if (disk[block] != -1)
        {
            printf("ERROR!: Disk Space not available.\n");
            return 0; // file allocation error
        }
        else
        {
            disk[block] = file_size; //allocation of disk space to the file
        }
    }
    num_of_files++;
    files[num_of_files - 1] = file; // file added to the list
    return 1; // successful allocation of disk space to the file
}
//Function to delete a file from disk
void deleteFile(File fileToDelete)
{
    char deletedFilename[20];
    for (int block = fileToDelete.start; block <= fileToDelete.end; block++)
    {
        disk[block] = -1;  //deallocation of disk space where the deleted file is situated earlier
    }
    // Find the corresponding filename in the list of files
    for (int i = 0; i < num_of_files; i++)
    {
        if (files[i].start == fileToDelete.start && files[i].end == fileToDelete.end)
        {
            strcpy(deletedFilename, files[i].filename); //copying the deleted filename from original list
            // Remove the file from the list of files
            if (i != num_of_files - 1)
            {
                files[i] = files[num_of_files - 1];
            }
            break;
        }
    }
    num_of_files--;  //decrease value of number of files in the list
    printf("File '%s' successfully deleted.\n", deletedFilename);
}
//Function to display disk status whether disk is allocated or empty
void displayDisk()
{
    printf("Disk Status:\n");
    for (int i = 0; i < DISK_SIZE; i++)
    {
        if (disk[i] == -1)
        {
            printf(". "); // Block is empty
        }
        else
        {
            printf("X "); // Block is allocated to any file
        }
        //no error message after allocation of files less than the maximum number of files
        if ((i + 1) % 20 == 0)
        {
            printf("\n");
        }
    }
}

int main()
{
    initialiseDisk();  //Initializing disk with some units of space  (1000 - as defined earlier)
    int choice;
    while (1)
    {
        printf("\n1. Allocate a file\n2. Delete a file\n3. Display\n4. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
            {
                if (num_of_files >= MAX_FILES)
                {
                    printf("Error: Maximum number of files reached.\n");
                }
                else
                {
                    File file;
                    printf("Enter filename: ");
                    scanf("%s", file.filename);
                    printf("Enter starting block of file: ");
                    scanf("%d", &file.start);
                    printf("Enter ending block of file: ");
                    scanf("%d", &file.end);
                    if (allocateFile(file))
                    {
                        printf("Space for File '%s' allocated successfully\n", file.filename);
                    }
                    else
                    {
                        printf("Space not available for File '%s'\n", file.filename);
                    }
                }
                break;
            }

            case 2:
            {
                if (num_of_files > 0)
                {
                    int deleteNum;
                    printf("Enter the S.No. of file to delete: ");
                    scanf("%d", &deleteNum);
                    if (deleteNum >= 1 && deleteNum <= num_of_files)
                    {
                        File fileToDelete = files[deleteNum - 1];
                        deleteFile(fileToDelete);
                    }
                    else
                    {
                        printf("Invalid file number.\n");
                    }
                }
                else
                {
                    printf("No files to delete.\n");
                }
                break;
            }

            case 3:
                displayDisk();
                break;

            case 4:
                exit(0);

            default:
                printf("Invalid option.\n");
                break;
        }
    }
    return 0;
}
```