Operating Systems (CSE 316) CA2

Name: Yusupha Sinjanka

Reg no: 12111133

Section: K21AR

Roll no: A27

Submitted to: Rajan Kakkar

Date: 28/03/23

Question 9 Solution:

METHODOLOGY

1. Choose a programming language: You will need to choose a programming language that is suitable for implementing the simulation program. In my case I chose C programming.
2. Define the data structures: You will need to define data structures that represent the simulated disk, the files, and the directory structure. The disk can be represented as an array of blocks, and the files can be represented as structures that contain information about the file name, size, and location on the disk.
3. Choose a file allocation algorithm: You will need to choose a file allocation algorithm, such as contiguous allocation or linked allocation, that will be used to allocate space for files on the simulated disk.
4. Implement the file system manager: You will need to implement a file system manager that handles file operations such as creating, deleting, renaming, and moving files. The file system manager should use the chosen file allocation algorithm to allocate and deallocate space for files on the simulated disk.
5. Simulate file operations: You will need to simulate file operations by randomly generating files of varying sizes and performing file operations such as creating, 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.

6. Evaluate the simulation results: You should evaluate the simulation results by analysing the average amount of fragmentation and the number of wasted disk blocks under different scenarios and with different file allocation algorithms. You should also identify the trade-offs involved in different file allocation algorithms and consider the factors that affect the performance of the file system.
7. Write a report: You should write a report that summarizes your findings and observations, including the performance of the file system under different scenarios and the trade-offs involved in the different file allocation algorithms.

IMPLEMENTATION CODE

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


#define MAX_FILES 1000

#define MAX_BLOCKS 10000

#define BLOCK_SIZE 512


int disk_blocks[MAX_BLOCKS];

int used_blocks = 0;

int wasted_blocks = 0;


struct file {

   int start_block;

   int num_blocks;

};
```

```
struct file files[MAX_FILES];
int num_files = 0;

void initialize_disk() {
    int i;
    for (i = 0; i < MAX_BLOCKS; i++) {
        disk_blocks[i] = 0;
    }
}

int find_free_blocks(int num_blocks) {
    int i, j, count;
    for (i = 0; i < MAX_BLOCKS - num_blocks; i++) {
        count = 0;
        for (j = i; j < i + num_blocks; j++) {
            if (disk_blocks[j] == 0) {
                count++;
            } else {
                break;
            }
        }
        if (count == num_blocks) {
            return i;
        }
    }
    return -1;
}
```

```
void allocate_blocks(int start_block, int num_blocks) {
    int i;
    for (i = start_block; i < start_block + num_blocks; i++) {
        disk_blocks[i] = 1;
    }
    used_blocks += num_blocks;
    wasted_blocks += BLOCK_SIZE * (num_blocks - 1);
}

void deallocate_blocks(int start_block, int num_blocks) {
    int i;
    for (i = start_block; i < start_block + num_blocks; i++) {
        disk_blocks[i] = 0;
    }
    used_blocks -= num_blocks;
    wasted_blocks -= BLOCK_SIZE * (num_blocks - 1);
}

void add_file(int num_blocks) {
    int start_block = find_free_blocks(num_blocks);
    if (start_block == -1) {
        printf("Error: no free blocks for file\n");
        return;
    }
    files[num_files].start_block = start_block;
    files[num_files].num_blocks = num_blocks;
    num_files++;
    allocate_blocks(start_block, num_blocks);
}
```

```
void remove_file(int index) {
    deallocate_blocks(files[index].start_block, files[index].num_blocks);
    int i;
    for (i = index; i < num_files - 1; i++) {
        files[i] = files[i + 1];
    }
    num_files--;
}

void print_stats(int time) {
    int i, frag = 0;
    for (i = 0; i < num_files; i++) {
        frag += files[i].num_blocks - 1;
    }
    float avg_frag = (float) frag / num_files;
    printf("Time: %d\n", time);
    printf("Used blocks: %d\n", used_blocks);
    printf("Wasted blocks: %d\n", wasted_blocks);
    printf("Average fragmentation: %.2f\n", avg_frag);
}

int main() {
    int i;
    int time;
    initialize_disk();
    srand(time);
    for (time = 1; time <= 10; time++) {
        // Add some random files
```

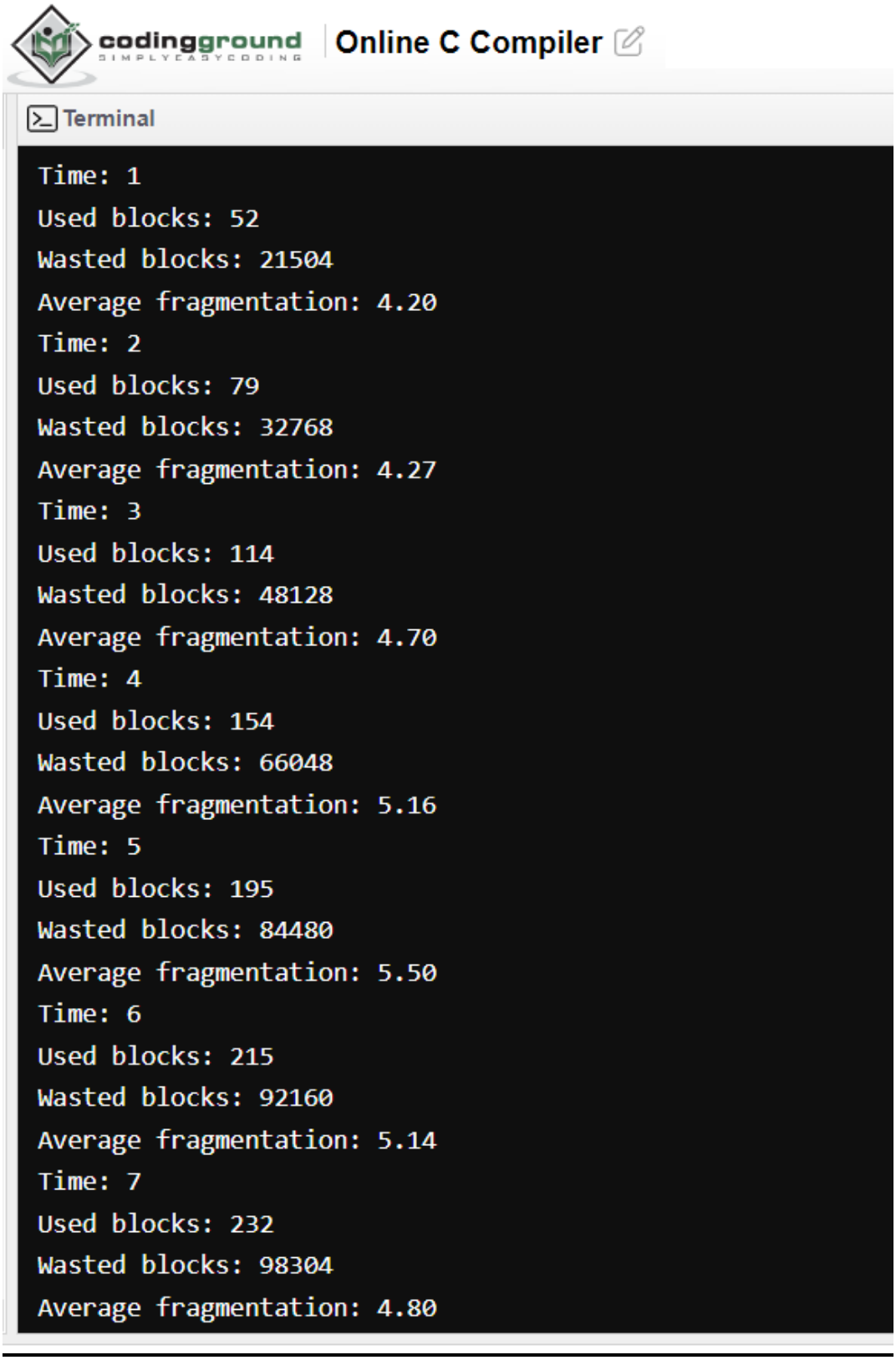
```
    for (i = 0; i < 10; i++) {
        add_file(rand() % 10 + 1);
    }
    print_stats(time);
    // Remove some random files
    for (i = 0; i < 5; i++) {
        if (num_files > 0) {
            remove_file(rand() % num_files);
        }
    }
  }
  return 0;
}
```

NOTE:

Here, **'MAX_BLOCKS**' represents the maximum number of blocks on the simulated disk, and **'MAX_FILENAME'** represents the maximum length of a file name. The **'file'** structure contains information about a file, including its name, size, and the index of the first block it occupies on the disk. The **'disk_block'** structure represents a block on the disk and contains a pointer to the next block in a file (for linked allocation).

'files' is an array of file structures that stores information about the **'files'** on the disk, and **'disk'** is an array of **'disk_block'** structures that represents the simulated disk. The **'allocate_blocks'** function finds a contiguous sequence of free blocks on the disk (for contiguous allocation) or returns the index of the first

OUTPUT SCREENSHOT



Terminal

```
Used blocks: 154
Wasted blocks: 66048
Average fragmentation: 5.16
Time: 5
Used blocks: 195
Wasted blocks: 84480
Average fragmentation: 5.50
Time: 6
Used blocks: 215
Wasted blocks: 92160
Average fragmentation: 5.14
Time: 7
Used blocks: 232
Wasted blocks: 98304
Average fragmentation: 4.80
Time: 8
Used blocks: 251
Wasted blocks: 105472
Average fragmentation: 4.58
Time: 9
Used blocks: 283
Wasted blocks: 119296
Average fragmentation: 4.66
Time: 10
Used blocks: 306
Wasted blocks: 128512
Average fragmentation: 4.56
```