**PRACTICAL No. – 06**

**CODE -**

#include <stdio.h>

#include <stdlib.h>

int main() {

FILE \*filePointer;

char data[100];

filePointer = fopen("file1.txt", "w");

if (filePointer == NULL) {

printf("File could not be opened.\n");

return -1;

}

printf("Enter some text to write to the file: ");

fgets(data, sizeof(data), stdin);

fprintf(filePointer, "%s", data);

fclose(filePointer);

filePointer = fopen("file1.txt", "r");

if (filePointer == NULL) {

printf("File could not be opened.\n");

return -1;

}

printf("Contents of the file:\n");

while (fgets(data, sizeof(data), filePointer) != NULL) {

printf("%s", data);

}

fclose(filePointer);

return 0;

}

**OUTPUT -**

Enter some text to write to the file: Hello, I am a student from MIT.

Contents of the file:

Hello, I am a student from MIT.

**PRACTICAL No. – 07**

**CODE –**

#include <stdio.h>

#include <stdlib.h>

int main() {

FILE \*sourceFile, \*destinationFile;

char ch;

sourceFile = fopen("file1.txt", "r");

if (sourceFile == NULL) {

printf("Error: Unable to open source file.\n");

return -1;

}

destinationFile = fopen("file2.txt", "w");

if (destinationFile == NULL) {

printf("Error: Unable to open destination file.\n");

fclose(sourceFile);

return -1;

}

while ((ch = fgetc(sourceFile)) != EOF) {

fputc(ch, destinationFile);

}

if (fclose(sourceFile) != 0) {

printf("Error: Unable to close source file.\n");

return -1;

}

if (fclose(destinationFile) != 0) {

printf("Error: Unable to close destination file.\n");

return -1;

}

printf("File copied successfully!\n");

return 0;

}

**OUTPUT -**

File copied successfully!

**PRACTICAL No. – 08**

**CODE - First Fit**

#include <stdio.h>

#include <stdlib.h>

#define MEMORY\_SIZE 100

typedef struct Node {

int size;

int allocated;

struct Node\* next;

} Node;

Node\* head = NULL;

void initializeMemory() {

head = (Node\*)malloc(sizeof(Node));

head->size = MEMORY\_SIZE;

head->allocated = 0;

head->next = NULL;

}

void firstFit(int size) {

Node\* current = head;

while (current != NULL) {

if (current->allocated == 0 && current->size >= size) {

current->allocated = 1;

printf("Memory allocated using First Fit starting at address %p\n", current);

return;

}

current = current->next;

}

printf("Memory allocation failed using First Fit\n");

}

void displayMemory() {

Node\* current = head;

printf("Memory Status:\n");

while (current != NULL) {

printf("Block: %p, Size: %d, Allocated: %d\n", current, current->size, current->allocated);

current = current->next;

}

printf("\n");

}

int main() {

initializeMemory();

displayMemory();

firstFit(20);

displayMemory();

firstFit(30);

displayMemory();

return 0;

}

**OUTPUT –**

Memory Status:

Block: 0x7ffcbdb00000, Size: 100, Allocated: 0

Memory allocated using First Fit starting at address 0x7ffcbdb00000

Memory Status:

Block: 0x7ffcbdb00000, Size: 20, Allocated: 1

Block: 0x7ffcbdb00030, Size: 80, Allocated: 0

Memory allocated using First Fit starting at address 0x7ffcbdb00030

Memory Status:

Block: 0x7ffcbdb00000, Size: 20, Allocated: 1

Block: 0x7ffcbdb00030, Size: 30, Allocated: 1

Block: 0x7ffcbdb00060, Size: 50, Allocated: 0

**PRACTICAL No. – 08**

**CODE - Best Fit**

#include <stdio.h>

#include <stdlib.h>

#define MEMORY\_SIZE 100

typedef struct Node {

int size;

int allocated;

struct Node\* next;

} Node;

Node\* head = NULL;

void initializeMemory() {

head = (Node\*)malloc(sizeof(Node));

head->size = MEMORY\_SIZE;

head->allocated = 0;

head->next = NULL;

}

void bestFit(int size) {

Node\* current = head;

Node\* bestFitBlock = NULL;

int minFragmentation = MEMORY\_SIZE + 1;

while (current != NULL) {

if (current->allocated == 0 && current->size >= size) {

int fragmentation = current->size - size;

if (fragmentation < minFragmentation) {

minFragmentation = fragmentation;

bestFitBlock = current;

}

}

current = current->next;

}

if (bestFitBlock != NULL) {

bestFitBlock->allocated = 1;

printf("Memory allocated using Best Fit starting at address %p\n", bestFitBlock);

} else {

printf("Memory allocation failed using Best Fit\n");

}

}

void displayMemory() {

Node\* current = head;

printf("Memory Status:\n");

while (current != NULL) {

printf("Block: %p, Size: %d, Allocated: %d\n", current, current->size, current->allocated);

current = current->next;

}

printf("\n");

}

int main() {

initializeMemory();

displayMemory();

bestFit(20);

displayMemory();

return 0;

}

**OUTPUT –**

Memory Status:

Block: 0x7ffcbdb00000, Size: 100, Allocated: 0

Memory allocated using Best Fit starting at address 0x7ffcbdb00000

Memory Status:

Block: 0x7ffcbdb00000, Size: 20, Allocated: 1

Block: 0x7ffcbdb00030, Size: 80, Allocated: 0

**PRACTICAL No. – 08**

**CODE - Worst Fit**

#include <stdio.h>

#include <stdlib.h>

#define MEMORY\_SIZE 100

typedef struct Node {

int size;

int allocated;

struct Node\* next;

} Node;

Node\* head = NULL;

void initializeMemory() {

head = (Node\*)malloc(sizeof(Node));

head->size = MEMORY\_SIZE;

head->allocated = 0;

head->next = NULL;

}

void worstFit(int size) {

Node\* current = head;

Node\* worstFitBlock = NULL;

int maxFragmentation = -1;

while (current != NULL) {

if (current->allocated == 0 && current->size >= size) {

int fragmentation = current->size - size;

if (fragmentation > maxFragmentation) {

maxFragmentation = fragmentation;

worstFitBlock = current;

}

}

current = current->next;

}

if (worstFitBlock != NULL) {

worstFitBlock->allocated = 1;

printf("Memory allocated using Worst Fit starting at address %p\n", worstFitBlock);

} else {

printf("Memory allocation failed using Worst Fit\n");

}

}

void displayMemory() {

Node\* current = head;

printf("Memory Status:\n");

while (current != NULL) {

printf("Block: %p, Size: %d, Allocated: %d\n", current, current->size, current->allocated);

current = current->next;

}

printf("\n");

}

int main() {

initializeMemory();

displayMemory();

worstFit(20);

displayMemory();

return 0;

}

**OUTPUT -**

Memory Status:

Block: 0x7ffcbdb00000, Size: 100, Allocated: 0

Memory allocated using Worst Fit starting at address 0x7ffcbdb00000

Memory Status:

Block: 0x7ffcbdb00000, Size: 20, Allocated: 1

Block: 0x7ffcbdb00030, Size: 80, Allocated: 0

**PRACTICAL No. – 09**

**CODE -**

#include <stdio.h>

#include <stdbool.h>

#define NUM\_FRAMES 3

#define NUM\_PAGES 10

int main() {

int pages[NUM\_PAGES] = {1, 2, 3, 4, 5, 1, 2, 6, 7, 8};

int frames[NUM\_FRAMES];

bool isInMemory[NUM\_PAGES] = {false};

int pageFaults = 0;

int nextFrameIndex = 0;

for (int i = 0; i < NUM\_FRAMES; ++i) {

frames[i] = -1;

}

for (int i = 0; i < NUM\_PAGES; ++i) {

int currentPage = pages[i];

bool pageFault = true;

for (int j = 0; j < NUM\_FRAMES; ++j) {

if (frames[j] == currentPage) {

pageFault = false;

break;

}

}

if (pageFault) {

++pageFaults;

frames[nextFrameIndex] = currentPage;

isInMemory[currentPage - 1] = true;

nextFrameIndex = (nextFrameIndex + 1) % NUM\_FRAMES;

}

printf("Page reference: %d\t Memory: ", currentPage);

for (int j = 0; j < NUM\_FRAMES; ++j) {

if (frames[j] != -1) {

printf("%d ", frames[j]);

} else {

printf("- ");

}

}

printf("\n");

}

printf("Total page faults: %d\n", pageFaults);

return 0;

}

**OUTPUT –**