**BVRIT HYDERABAD College of Engineering for Women**

**(UGC-Autonomous)**

(Approved by AICTE| Affiliated to JNTUH| Accredited by NAAC with Grade ‘A’& NBA for CSE,ECE,EEE,&IT)

Bachupally, Hyderabad-500090

**Department of Artificial Intelligence & Machine Learning**

**Subject: Operating Systems**

**Name of the student:BATHULA PADMASRI**

**RollNo:22WH1A6632**

**CASE STUDY ON:**

**Memory management game**

**Problem Statement-** Consider expanding the memory management game idea by introducing different types of memory allocation algorithms such as first-fit, best-fit, and worst-fit. Explain the pros and cons of each approach, offering users a comprehensive look into the complexities of memory management**.**

**Description-** In the memory management game, incorporating various allocation algorithms adds depth to the player's experience. Let's explore three common strategies: first-fit, best-fit, and worst-fit.

**1. \*First-Fit:\***

- Pros: Simple and efficient, as it allocates the first available block that is large enough.

- Cons: Can lead to fragmentation, both internal (unused memory within allocated blocks) and external (unused memory between allocated blocks).

**2. \*Best-Fit:\***

- Pros: Tries to minimize wasted memory by allocating the smallest available block that fits the request.

- Cons: May lead to fragmentation, as small gaps between blocks can accumulate over time, making it challenging to find suitable spaces for larger allocations.

**3. \*Worst-Fit:\***

- Pros: Allocates the largest available block, reducing the likelihood of future allocations being too large.

- Cons: High fragmentation potential, as smaller gaps between allocated blocks can result.

**PROGRAM-**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_BLOCKS 10

void displayMenu() {

printf("\nMemory Management Game Menu:\n");

printf("1. Allocate Memory\n");

printf("2. Deallocate Memory\n");

printf("3. Display Memory\n");

printf("4. Exit\n");

}

int main() {

int\* memoryBlocks[MAX\_BLOCKS] = {NULL};

int choice;

do {

displayMenu();

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

// Allocate Memory

for (int i = 0; i < MAX\_BLOCKS; ++i) {

if (memoryBlocks[i] == NULL) {

int size;

printf("Enter the size to allocate: ");

scanf("%d", &size);

memoryBlocks[i] = (int\*)malloc(size \* sizeof(int));

printf("Memory allocated successfully.\n");

break;

}

}

break;

case 2:

// Deallocate Memory

for (int i = 0; i < MAX\_BLOCKS; ++i) {

if (memoryBlocks[i] != NULL) {

free(memoryBlocks[i]);

memoryBlocks[i] = NULL;

printf("Memory deallocated successfully.\n");

break;

}

}

break;

case 3:

// Display Memory

printf("Memory Blocks:\n");

for (int i = 0; i < MAX\_BLOCKS; ++i) {

if (memoryBlocks[i] != NULL) {

printf("Block %d: Allocated\n", i + 1);

} else {

printf("Block %d: Free\n", i + 1);

}

}

break;

case 4:

// Exit

printf("Exiting the Memory Management Game.\n");

break;

default:

printf("Invalid choice. Please enter a valid option.\n");

}

} while (choice != 4);

// Deallocate any remaining memory before exiting

for (int i = 0; i < MAX\_BLOCKS; ++i) {

if (memoryBlocks[i] != NULL) {

free(memoryBlocks[i]);

}

}

return 0;

}

**OUTPUT-**

Memory Management Game Menu:

1. Allocate Memory

2. Deallocate Memory

3. Display Memory

4. Exit

Enter your choice: 1

Enter the size to allocate: 5

Memory allocated successfully.

Memory Management Game Menu:

1. Allocate Memory

2. Deallocate Memory

3. Display Memory

4. Exit

Enter your choice: 2

Memory deallocated successfully.

Memory Management Game Menu:

1. Allocate Memory

2. Deallocate Memory

3. Display Memory

4. Exit

Enter your choice: 3

Memory Blocks:

Block 1: Free

Block 2: Free

Block 3: Free

Block 4: Free

Block 5: Free

Block 6: Free

Block 7: Free

Block 8: Free

Block 9: Free

Block 10: Free

Memory Management Game Menu:

1. Allocate Memory

2. Deallocate Memory

3. Display Memory

4. Exit

Enter your choice: 4

Exiting the Memory Management Game.