Micro Project Report

On

Memory Management Simulation

Diploma Computer Engineering

Semester 4

(Advance Object-Oriented Programming – 4340701)

Group Members		
Sr. No.	**Enrollment No.**	**Student Name**
1	**226090307001**	**Aditya Pithva**
2	**226090307023**	**Smit Dhokiya**
3	**226090307038**	**Darshan Jadav**
4	**226090307065**	**Manthan Makani**

Guided By: -

Mr. S. R. Bhalgama

Lecturer, CE Department,

C. U. Shah Polytechnic, Surendranagar

Index

Introduction to Project

In the realm of computer science, efficient memory management plays a pivotal role in ensuring smooth system operation. The Memory Management Unit (MMU) acts as the central control unit, allocating and overseeing memory usage for various processes. This project delves into simulating an MMU, implementing key memory allocation algorithms, and exploring techniques to optimize memory utilization.

The core objective of this project is to provide a comprehensive simulation environment for understanding memory management concepts and evaluating different allocation strategies. It achieves this by offering functionalities to:

- **Simulation Memory Allocation**:- The project implements three widely used memory allocation algorithm:First Fit,Best Fit and Worst Fit. Users can choose the designed algorithm to observe its impact on memory allocation decisions.
- **Process Memory Management:-** The simulation allow users to define pre-running processes with specific memory requirements.Additionally, it enables users to introduce new processes dynamically, simulated real-world scenarios.
- **Memory Fragmentation Analysis:**- Fragmentation occurs when free memory compaction functionality. This process rearranges allocated memory blocks,consolidating free memory into contiguous chuncks, thereby improving allocation efficiency.
- **Process Termination and Memory Release:**- The simulation mimics process termination by allowing users to specify a process ID for memory deallocation. This function realistically portrays memory management tasks during process lifecycle.

This project is implemented using Java programming language and presents a user-friendly command-line interface for interaction. Users can configure RAM size, define pre-existing processes, and select the preferred memory allocation algorithm. The interface also provides options to execute memory compaction, release memory for terminated processes, and analyze memory fragmentation.

This project serves as a valuable learning tool for students, educators, and professionals interested in computer systems and memory management strategies. It allows for experimentation and analysis, fostering a comprehensive understanding of memory management concepts and their practical applications.

Functional Requirement

- Allocate Memory
- Add Processes
- Release Memory
- Display RAM Status
- Compact Memory
- Analyze Fragmentation
- Handle Error Gracefully
- Validate User Input
- Exit System
- Support Multiple Allocation Algorithm
- Manage Waiting Queue
- Automate Process Allocation
- Handle Concurrent Access
- Provide Feedback to Users

Applications

- **Operating Systems**:- Your memory management system could be integrated into operating systems to efficiently allocate and deallocate memory for running processes. This ensures optimal performance and resource utilization, contributing to the overall stability and responsiveness of the system.
- **Embedded Systems**:- Embedded systems with limited memory resources can benefit from your memory management system to optimize memory usage and accommodate multiple tasks or applications running concurrently. This is particularly useful in IoT devices, smart appliances, and automotive systems.
- **Gaming Consoles**:- Gaming consoles can utilize your memory management system to allocate memory for game assets, textures, and other resources dynamically during gameplay. This ensures smooth performance and immersive gaming experiences by optimizing memory usage and minimizing loading times.
- **Mobile Applications**:- Mobile applications running on smartphones and tablets can benefit from your memory management system to optimize memory usage and enhance battery life. By efficiently managing memory resources, mobile apps can deliver better performance and responsiveness, improving the user experience.
- **Industrial Control Systems**:- Industrial control systems, such as those used in manufacturing and process automation, can leverage your memory management system to allocate memory for controlling and monitoring equipment and processes in real-time. This ensures efficient resource utilization and reliable operation of industrial systems.
- **Educational Environments**:- Educational institutions and training centers can use your memory management system to teach students about memory management concepts and algorithms. By providing hands-on experience with memory allocation and deallocation, students can gain practical knowledge and skills in system programming and software development.
- **Scientific Computing**:- Scientific computing applications, such as simulations and data analysis, often require large amounts of memory for processing complex datasets. Your memory management system can ensure efficient memory allocation and utilization, enabling scientists and researchers to perform computational tasks effectively.

Code

```java
import java.util.Scanner;
import java.util.Map;
import java.util.HashMap;
import java.util.Queue;
import java.util.LinkedList;
public class Main{
    public static Short nextId = 1;
    public static Queue<E_process> waitingQueue = new LinkedList<E_process>();
    public static void add_process(int pre_run_process,Process[] p,int[] array_ram,Scanner sc){
        for(Short i=0;i<pre_run_process;i++){
            System.out.println("Enter details of process "+(i+1)+" (size,start_address):");
            Short size = checkNegativeInput(sc);
            Short startaddress = checkNegativeInput(sc);
            Short id = nextId++;
            p[i] = new Process(id,size,startaddress);
            for(Short j=startaddress;j<startaddress+size;j++){
                if(array_ram[j] != 0){
                    System.out.println(x:"This space is already occupied by another process...!");
                    break;
                }else{
                    array_ram[j] = id;
                }
            }
        }
    }
    public static void print_ram(int[] array_ram){
        System.out.println(x:"--------------------RAM--------------------");
        for(Short i=0;i<array_ram.length;i++){
            System.out.print("["+array_ram[i]+"]");
```

```java
    public static void First_Fit(Scanner sc, int[] array_ram, E_process[] ep) {
        System.out.println(x:"Enter number of processes you want to add:");
        Short a_p = checkNegativeInput(sc);

        for (Short i = 0; i < a_p; i++) {
            System.out.print("Enter size of process" + (i + 1) + ":");
            Short size =  checkNegativeInput(sc);
            Short id = nextId++;
            ep[i] = new E_process(size, id);
        }
        for (Short i = 0; i < a_p; i++) {
            Short count = 0;
            Short startAddress = 0;
            boolean allocated = false;
            for (Short j = 0; j < array_ram.length; j++) {
                if (array_ram[j] == 0) {
                    if (count == 0) {
                        startAddress = j;
                    }
                    count++;
                    if (count == ep[i].getsize()) {
                        for (int k = startAddress; k < startAddress + ep[i].getsize(); k++) {
                            array_ram[k] = ep[i].getid();
                        }
                        allocated = true;
                        break;
                    }
```

```java
public static void Best_Fit(Scanner sc, Short id, int[] array_ram, E_process[] ep) {
    System.out.print(s:"Enter number of processes you want to add:");
    Short a_p = checkNegativeInput(sc);
    for (Short i = 0; i < a_p; i++) {
        System.out.print("Enter size of process " + (i + 1) + ":");
        Short size = checkNegativeInput(sc);
        id = nextId++;
        ep[i] = new E_process(size, id);
    }
    Map<Integer, Integer> available_blocks = new HashMap<>();
    for (Short i = 0; i < array_ram.length; i++) {
        if (array_ram[i] == 0) {
            int blockSize = 0;
            int start = i;
            while (i < array_ram.length && array_ram[i] == 0) {
                blockSize++;
                i++;
            }
            if (blockSize > 0) {
                available_blocks.put(start, blockSize);
            }
        }
    }
    for (Short i = 0; i < a_p; i++) {
        int processSize = ep[i].getsize();
        int[] allocationInfo = allocateBestFit(available_blocks, processSize);
        int allocateStart = allocationInfo[0];
        int blockSize = allocationInfo[1];
```

```java
public static void Worst_fit(Scanner sc, Short id, int[] array_ram, E_process[] ep){
    Short a_p = (short) checkNegativeInput(sc);
    for(int i=0; i<a_p; i++){
        System.out.print("Enter size of process "+(i+1)+":");
        Short size = checkNegativeInput(sc);
        id = nextId++;
        ep[i] = new E_process(size, id);
    }

    for(int i = 0; i < a_p; i++){
        int processSize = ep[i].getsize();
        int worstFitStart = allocateWorstFit(array_ram, processSize);
        if(worstFitStart != -1){
            // Allocate memory for the process
            for(int j = worstFitStart; j < worstFitStart + processSize; j++){
                array_ram[j] = ep[i].getid();
            }
        } else {
            waitingQueue.add(ep[i]);
            System.out.println("Memory allocation failed for process "+ep[i].getid()+". No suitable
            block available");
        }
    }
    print_ram(array_ram);
}

private static int allocateWorstFit(int[] array_ram, int processSize){
    int worstFitStart = -1;
    int worstFitSize = -1;
```

```java
public static void compact_memory(int[] array_ram,Queue<E_process> waitingQueue){
    Short id = 0;
    for(Short i=0;i<array_ram.length;i++){
        if(array_ram[i]!= 0){
            array_ram[id++] = array_ram[i];
        }
    }
    for(Short i=id;i<array_ram.length;i++){
        array_ram[i] = 0;
    }
    while(!waitingQueue.isEmpty()){
        E_process temp =  waitingQueue.poll();
        // Now, find a suitable place in RAM to allocate the process
        boolean allocated = false;
        for(Short j=0; j<array_ram.length; j++){
            if(array_ram[j] == 0){
                Short size = temp.getsize();
                boolean fits = true;
                for(Short k=j; k<j+size; k++){
                    if(k >= array_ram.length || array_ram[k] != 0){
                        fits = false;
                        break;
                    }
                }
                if(fits){
                    for(Short k=j; k<j+size; k++){
                        array_ram[k] = temp.getid();
```

```java
public static void main(String[] args) {
    Process[] p = new Process[100];
    Scanner sc = new Scanner(System.in);
    boolean exit = false;
    while (!exit) {
        try {
            System.out.print(s:"Enter size of your RAM:");
            Short size_of_ram = (short) checkRam(sc);
            final int[] array_ram = new int[size_of_ram];
            E_process[] ep = new E_process[100];
            System.out.print(s:"Enter no. of pre-running processes:");
            Short pre_run_process = checkNegativeInput(sc);
            add_process(pre_run_process, p, array_ram, sc);
            print_ram(array_ram);
            while (true) {
                System.out.println(x:"\n1)Allocation in First Fit");
                System.out.println(x:"2)Allocation in Best Fit");
                System.out.println(x:"3)Allocation in Worst Fit");
                System.out.println(x:"4)Compact Memory");
                System.out.println(x:"5)Release Memory for Process");
                System.out.println(x:"6)Memory Fragmentation Analysis");
                System.out.println(x:"7)Exit");
                System.out.print(s:"Enter your choice:");
                Short choice = checkNegativeInput(sc);
                switch (choice) {
                    case 1:
```

Screenshots of Working Project

```
Enter size of your RAM:20
Enter no. of pre-running processes:2
Enter details of process 1 (size,start_address):
3
0
Enter details of process 2 (size,start_address):
4
8
--------------------RAM--------------------
[1][1][1][0][0][0][0][0][2][2][2][2][0][0][0][0][0][0][0][0]
1)Allocation in First Fit
2)Allocation in Best Fit
3)Allocation in Worst Fit
4)Compact Memory
5)Release Memory for Process
6)Memory Fragmentation Analysis
7)Exit
Enter your choice:1
Enter number of processes you want to add:
1
Enter size of process1:3
```

```
Enter number of processes you want to add:
1
Enter size of process1:3
--------------------RAM--------------------
[1][1][1][3][3][3][0][0][2][2][2][2][0][0][0][0][0][0][0][0]
1)Allocation in First Fit
2)Allocation in Best Fit
3)Allocation in Worst Fit
4)Compact Memory
5)Release Memory for Process
6)Memory Fragmentation Analysis
7)Exit
Enter your choice:3
Enter number of processes you want to add:1
Enter size of process 1:1
--------------------RAM--------------------
[1][1][1][3][3][3][0][0][2][2][2][2][4][0][0][0][0][0][0][0]
1)Allocation in First Fit
2)Allocation in Best Fit
3)Allocation in Worst Fit
4)Compact Memory
5)Release Memory for Process
6)Memory Fragmentation Analysis
7)Exit
Enter your choice:4
```

```
--------------------RAM--------------------
[1][1][1][2][2][2][2][3][3][3][3][3][3][3][3][3][3][0][0][0]
1)Allocation in First Fit
2)Allocation in Best Fit
3)Allocation in Worst Fit
4)Compact Memory
5)Release Memory for Process
6)Memory Fragmentation Analysis
7)Exit
Enter your choice:6
Memory Fragmentation Analysis:
Number of free blocks: 1
Total free memory size: 3

1)Allocation in First Fit
2)Allocation in Best Fit
3)Allocation in Worst Fit
4)Compact Memory
5)Release Memory for Process
6)Memory Fragmentation Analysis
7)Exit
Enter your choice:
```

```
1)Allocation in First Fit
2)Allocation in Best Fit
3)Allocation in Worst Fit
4)Compact Memory
5)Release Memory for Process
6)Memory Fragmentation Analysis
7)Exit
Enter your choice:5
Enter process id to release memory:2
Memory released successfully...!
--------------------RAM--------------------
[1][1][1][0][0][0][0][3][3][3][3][3][3][3][3][3][3][0][0][0]
1)Allocation in First Fit
2)Allocation in Best Fit
3)Allocation in Worst Fit
4)Compact Memory
5)Release Memory for Process
6)Memory Fragmentation Analysis
7)Exit
Enter your choice:7
PS D:\Projects\Memory_Allocation_Techniques_Java>
```

References

- https://www.w3schools.com/java/default.asp
- https://www.javatpoint.com/collections-in-java
- https://chatgpt.com/
- Programming with Java : E Balagyrysamy
- Modern Operating System : Andrew Tanenbaum