**Write a program to simulate Memory placement strategies – best fit, first fit.**

package MemoryAllocation;

import java.util.Scanner;

public class MemoryAllocation {

// Method to allocate memory using First fit algorithm

static void firstFit(int blockSize[], int m, int processSize[], int n) {

int allocation[] = new int[n];

// Initially no block is assigned to any process

for (int i = 0; i < allocation.length; i++) {

allocation[i] = -1;

}

// Pick each process and find suitable blocks

for (int i = 0; i < n; i++) {

boolean allocated = false; // Flag to check if process is allocated

for (int j = 0; j < m; j++) {

if (blockSize[j] >= processSize[i]) {

allocation[i] = j;

blockSize[j] -= processSize[i]; // Reduce available memory in this block

allocated = true;

break; // Stop once the process is allocated

}

}

}

// Print results for FirstFit

printResults(processSize, allocation, n, blockSize);

}

// Method to allocate memory using Best fit algorithm

static void bestFit(int blockSize[], int m, int processSize[], int n) {

int allocation[] = new int[n];

// Initially no block is assigned to any process

for (int i = 0; i < allocation.length; i++) {

allocation[i] = -1;

}

// Pick each process and find the best fit block for it

for (int i = 0; i < n; i++) {

int bestIdx = -1;

for (int j = 0; j < m; j++) {

if (blockSize[j] >= processSize[i]) {

if (bestIdx == -1) {

bestIdx = j;

} else if (blockSize[bestIdx] > blockSize[j]) {

bestIdx = j; // Choose the block with the smallest remaining space

}

}

}

if (bestIdx != -1) {

allocation[i] = bestIdx;

blockSize[bestIdx] -= processSize[i]; // Reduce available memory in this block

}

}

// Print results for BestFit

printResults(processSize, allocation, n, blockSize);

}

// Helper method to print the results for both FirstFit and BestFit

static void printResults(int processSize[], int allocation[], int n, int blockSize[]) {

// Additional details

System.out.println("Name: Ketan Devraj");

System.out.println("Roll No.: TACO22122");

System.out.println("\nProcess No.\tProcess Size\tBlock no.\tRemaining Block Size");

// Output process allocation details

for (int i = 0; i < n; i++) {

System.out.print(" " + (i + 1) + "\t\t" + processSize[i] + "\t\t");

if (allocation[i] != -1) {

// Print the block number allocated

System.out.print((allocation[i] + 1) + "\t\t");

} else {

System.out.print("Not Allocated\t");

}

// If allocated, print the remaining block size after allocation

if (allocation[i] != -1) {

System.out.print("Block " + (allocation[i] + 1) + " remaining: " + blockSize[allocation[i]]);

}

System.out.println();

}

}

// Driver Method

public static void main(String[] args) {

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

int m = blockSize.length;

int n = processSize.length;

Scanner sc = new Scanner(System.in);

System.out.println("Choose Allocation Strategy:");

System.out.println("1. First Fit");

System.out.println("2. Best Fit");

int choice = sc.nextInt();

// Switch case to choose between First Fit and Best Fit

switch (choice) {

case 1:

System.out.println("First Fit Allocation:");

firstFit(blockSize.clone(), m, processSize, n);

break;

case 2:

System.out.println("Best Fit Allocation:");

bestFit(blockSize.clone(), m, processSize, n);

break;

default:

System.out.println("Invalid choice.");

break;

}

sc.close();

}

}

**Write a program to simulate Memory placement strategies –next fit and worstfit.**

package MemoryAllocation;

import java.util.\*;

public class MemoryAllocation {

// Method to allocate memory using Next fit algorithm

static void nextFit(int blockSize[], int m, int processSize[], int n) {

int allocation[] = new int[n];

Arrays.fill(allocation, -1); // Initially no block is allocated to any process

int lastAllocated = 0; // Variable to track the last allocated block

// Next Fit: Pick each process and find a suitable block starting from last allocated block

for (int i = 0; i < n; i++) {

boolean allocated = false;

// Search from lastAllocated block to the end of the blocks

for (int j = lastAllocated; j < m; j++) {

if (blockSize[j] >= processSize[i]) {

allocation[i] = j;

blockSize[j] -= processSize[i]; // Reduce available memory in this block

lastAllocated = j; // Update last allocated block

allocated = true;

break;

}

}

// If no block is found in the remaining blocks, start from the beginning of the blocks

if (!allocated) {

for (int j = 0; j < lastAllocated; j++) {

if (blockSize[j] >= processSize[i]) {

allocation[i] = j;

blockSize[j] -= processSize[i];

lastAllocated = j;

break;

}

}

}

}

// Print the results for NextFit

printResults(processSize, allocation, n, blockSize);

}

// Method to allocate memory using Worst fit algorithm

static void worstFit(int blockSize[], int m, int processSize[], int n) {

int allocation[] = new int[n];

Arrays.fill(allocation, -1); // Initially no block is allocated to any process

// Worst Fit: Pick each process and find the worst block to allocate

for (int i = 0; i < n; i++) {

int wstIdx = -1;

for (int j = 0; j < m; j++) {

if (blockSize[j] >= processSize[i]) {

if (wstIdx == -1 || blockSize[wstIdx] < blockSize[j]) {

wstIdx = j;

}

}

}

if (wstIdx != -1) {

allocation[i] = wstIdx;

blockSize[wstIdx] -= processSize[i]; // Reduce available memory in this block

}

}

// Print the results for WorstFit

printResults(processSize, allocation, n, blockSize);

}

// Helper method to print the results for all allocation strategies

static void printResults(int processSize[], int allocation[], int n, int blockSize[]) {

System.out.println("Name: Ketan Devraj");

System.out.println("Roll No.: TACO22122");

System.out.println("\nProcess No.\tProcess Size\tBlock no.\tRemaining Block Size");

// Print the details of each process

for (int i = 0; i < n; i++) {

System.out.print(i + 1 + "\t\t\t\t" + processSize[i] + "\t\t\t\t");

if (allocation[i] != -1) {

// Print allocated block number (1-based index)

System.out.print((allocation[i] + 1) + "\t\t\t");

} else {

System.out.print("Not Allocated\t\t\t");

}

// Print the remaining size of the allocated block

if (allocation[i] != -1) {

System.out.print("Block " + (allocation[i] + 1) + " remaining: " + blockSize[allocation[i]]);

}

System.out.println();

}

}

// Driver program

public static void main(String[] args) {

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

int m = blockSize.length;

int n = processSize.length;

Scanner sc = new Scanner(System.in);

System.out.println("Choose Allocation Strategy:");

System.out.println("1. Next Fit");

System.out.println("2. Worst Fit");

int choice = sc.nextInt();

// Switch case to choose between Next Fit and Worst Fit

switch (choice) {

case 1:

System.out.println("Next Fit Allocation:");

nextFit(blockSize.clone(), m, processSize, n);

break;

case 2:

System.out.println("Worst Fit Allocation:");

worstFit(blockSize.clone(), m, processSize, n);

break;

default:

System.out.println("Invalid choice.");

break;

}

sc.close();

}

}

**Write a program to simulate Memory placement strategies – best fit, next fit.**

package MemoryAllocation;

import java.util.\*;

public class MemoryAllocation {

// Method to allocate memory using Best fit algorithm

static void bestFit(int blockSize[], int m, int processSize[], int n) {

int allocation[] = new int[n];

Arrays.fill(allocation, -1); // Initially no block is allocated to any process

// Best Fit: Pick each process and find the best block to allocate

for (int i = 0; i < n; i++) {

int bestIdx = -1;

for (int j = 0; j < m; j++) {

if (blockSize[j] >= processSize[i]) {

if (bestIdx == -1 || blockSize[bestIdx] > blockSize[j]) {

bestIdx = j;

}

}

}

if (bestIdx != -1) {

allocation[i] = bestIdx;

blockSize[bestIdx] -= processSize[i]; // Reduce available memory in this block

}

}

// Print the results for BestFit

printResults(processSize, allocation, n, blockSize);

}

// Method to allocate memory using Next fit algorithm

static void nextFit(int blockSize[], int m, int processSize[], int n) {

int allocation[] = new int[n];

Arrays.fill(allocation, -1); // Initially no block is allocated to any process

int lastAllocated = 0; // Variable to track the last allocated block

// Next Fit: Pick each process and find a suitable block starting from last allocated block

for (int i = 0; i < n; i++) {

boolean allocated = false;

// Search from lastAllocated block to the end of the blocks

for (int j = lastAllocated; j < m; j++) {

if (blockSize[j] >= processSize[i]) {

allocation[i] = j;

blockSize[j] -= processSize[i]; // Reduce available memory in this block

lastAllocated = j; // Update last allocated block

allocated = true;

break;

}

}

// If no block is found in the remaining blocks, start from the beginning of the blocks

if (!allocated) {

for (int j = 0; j < lastAllocated; j++) {

if (blockSize[j] >= processSize[i]) {

allocation[i] = j;

blockSize[j] -= processSize[i];

lastAllocated = j;

break;

}

}

}

}

// Print the results for NextFit

printResults(processSize, allocation, n, blockSize);

}

// Helper method to print the results for both BestFit and NextFit

static void printResults(int processSize[], int allocation[], int n, int blockSize[]) {

System.out.println("Name: Ketan Devraj");

System.out.println("Roll No.: TACO22122");

System.out.println("\nProcess No.\tProcess Size\tBlock no.\tRemaining Block Size");

// Print the details of each process

for (int i = 0; i < n; i++) {

System.out.print(i + 1 + "\t\t\t\t" + processSize[i] + "\t\t\t\t");

if (allocation[i] != -1) {

// Print allocated block number (1-based index)

System.out.print((allocation[i] + 1) + "\t\t\t");

} else {

System.out.print("Not Allocated\t\t\t");

}

// Print the remaining size of the allocated block

if (allocation[i] != -1) {

System.out.print("Block " + (allocation[i] + 1) + " remaining: " + blockSize[allocation[i]]);

}

System.out.println();

}

}

// Driver program

public static void main(String[] args) {

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

int m = blockSize.length;

int n = processSize.length;

Scanner sc = new Scanner(System.in);

System.out.println("Choose Allocation Strategy:");

System.out.println("1. Best Fit");

System.out.println("2. Next Fit");

int choice = sc.nextInt();

// Switch case to choose between Best Fit and Next Fit

switch (choice) {

case 1:

System.out.println("Best Fit Allocation:");

bestFit(blockSize.clone(), m, processSize, n);

break;

case 2:

System.out.println("Next Fit Allocation:");

nextFit(blockSize.clone(), m, processSize, n);

break;

default:

System.out.println("Invalid choice.");

break;

}

sc.close();

}

}

**Write a program to simulate Memory placement strategies –first fit and worstfit.**

package MemoryAllocation;

import java.util.\*;

public class MemoryAllocation {

// Method to allocate memory using First fit algorithm

static void firstFit(int blockSize[], int m, int processSize[], int n) {

int allocation[] = new int[n];

Arrays.fill(allocation, -1); // Initially no block is allocated to any process

// First Fit: Pick each process and find the first block that can accommodate it

for (int i = 0; i < n; i++) {

boolean allocated = false;

// Search for the first block that can accommodate the current process

for (int j = 0; j < m; j++) {

if (blockSize[j] >= processSize[i]) {

allocation[i] = j;

blockSize[j] -= processSize[i]; // Reduce available memory in this block

allocated = true;

break;

}

}

// If no block is found for the current process, it remains unallocated

if (!allocated) {

allocation[i] = -1;

}

}

// Print the results for FirstFit

printResults(processSize, allocation, n, blockSize);

}

// Method to allocate memory using Worst fit algorithm

static void worstFit(int blockSize[], int m, int processSize[], int n) {

int allocation[] = new int[n];

Arrays.fill(allocation, -1); // Initially no block is allocated to any process

// Worst Fit: Pick each process and find the block with the largest remaining space

for (int i = 0; i < n; i++) {

int wstIdx = -1;

for (int j = 0; j < m; j++) {

if (blockSize[j] >= processSize[i]) {

if (wstIdx == -1 || blockSize[wstIdx] < blockSize[j]) {

wstIdx = j;

}

}

}

if (wstIdx != -1) {

allocation[i] = wstIdx;

blockSize[wstIdx] -= processSize[i]; // Reduce available memory in this block

}

}

// Print the results for WorstFit

printResults(processSize, allocation, n, blockSize);

}

// Helper method to print the results for all allocation strategies

static void printResults(int processSize[], int allocation[], int n, int blockSize[]) {

System.out.println("Name: Ketan Devraj");

System.out.println("Roll No.: TACO22122");

System.out.println("\nProcess No.\tProcess Size\tBlock no.\tRemaining Block Size");

// Print the details of each process

for (int i = 0; i < n; i++) {

System.out.print(i + 1 + "\t\t\t\t" + processSize[i] + "\t\t\t\t");

if (allocation[i] != -1) {

// Print allocated block number (1-based index)

System.out.print((allocation[i] + 1) + "\t\t\t");

} else {

System.out.print("Not Allocated\t\t\t");

}

// Print the remaining size of the allocated block

if (allocation[i] != -1) {

System.out.print("Block " + (allocation[i] + 1) + " remaining: " + blockSize[allocation[i]]);

}

System.out.println();

}

}

// Driver program

public static void main(String[] args) {

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

int m = blockSize.length;

int n = processSize.length;

Scanner sc = new Scanner(System.in);

System.out.println("Choose Allocation Strategy:");

System.out.println("1. First Fit");

System.out.println("2. Worst Fit");

int choice = sc.nextInt();

// Switch case to choose between First Fit and Worst Fit

switch (choice) {

case 1:

System.out.println("First Fit Allocation:");

firstFit(blockSize.clone(), m, processSize, n);

break;

case 2:

System.out.println("Worst Fit Allocation:");

worstFit(blockSize.clone(), m, processSize, n);

break;

default:

System.out.println("Invalid choice.");

break;

}

sc.close();

}

}