# Bansilal Ramnath Agarwal Charitable Trust’s

Vishwakarma Institute of Technology, Pune-37

*(Anautonomous Institute of Savitribai Phule Pune University)*



**Department of Computer Engineering**

|  |  |
| --- | --- |
| **Division** | **CS** |
| **Batch** | **B1** |
| **Roll no.** | **90** |
| **Name** | **Aditya Shrinivas Kurapati** |
| **PRN No** | **12320184** |

* **Memory Management requirements, Memory Partitioning: Fixed, Dynamic Partitioning**

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#define MEMORY\_SIZE 1024

#define FIXED\_PARTITION\_SIZE 256

// Structure to represent a memory block

typedef struct MemoryBlock {

int size;

bool allocated;

} MemoryBlock;

// Fixed partitioning function

void fixedPartitioning() {

MemoryBlock memory[MEMORY\_SIZE / FIXED\_PARTITION\_SIZE];

// Initialize memory blocks

for (int i = 0; i < MEMORY\_SIZE / FIXED\_PARTITION\_SIZE; ++i) {

memory[i].size = FIXED\_PARTITION\_SIZE;

memory[i].allocated = false;

}

// Allocate memory

int num\_processes, process\_size;

printf("Enter the number of processes: ");

scanf("%d", &num\_processes);

printf("Enter the size of each process:\n");

for (int i = 0; i < num\_processes; ++i) {

scanf("%d", &process\_size);

bool allocated = false;

for (int j = 0; j < MEMORY\_SIZE / FIXED\_PARTITION\_SIZE; ++j) {

if (!memory[j].allocated && memory[j].size >= process\_size) {

memory[j].allocated = true;

allocated = true;

printf("Process %d allocated to memory block %d\n", i + 1, j + 1);

break;

}

}

if (!allocated) {

printf("Insufficient memory to allocate process %d\n", i + 1);

}

}

}

// Dynamic partitioning function

void dynamicPartitioning() {

int memory[MEMORY\_SIZE];

int num\_processes, process\_size;

// Initialize memory

for (int i = 0; i < MEMORY\_SIZE; ++i) {

memory[i] = -1; // -1 indicates unallocated memory

}

// Allocate memory

printf("Enter the number of processes: ");

scanf("%d", &num\_processes);

printf("Enter the size of each process:\n");

for (int i = 0; i < num\_processes; ++i) {

scanf("%d", &process\_size);

bool allocated = false;

for (int j = 0; j < MEMORY\_SIZE; ++j) {

if (memory[j] == -1) { // Find a free block

int k;

for (k = j; k < j + process\_size; ++k) {

if (memory[k] != -1) {

break; // Block is too small, move to next free block

}

}

if (k ==

j + process\_size) { // Allocate memory if the block is large enough

for (int l = j; l < j + process\_size; ++l) {

memory[l] = i; // Allocate process ID to memory block

}

printf("Process %d allocated to memory starting from block %d\n",

i + 1, j);

allocated = true;

break;

}

}

}

if (!allocated) {

printf("Insufficient memory to allocate process %d\n", i + 1);

}

}

}

int main() {

int choice;

// Display menu

printf("Memory Management Techniques\n");

printf("1. Fixed Partitioning\n");

printf("2. Dynamic Partitioning\n");

printf("Enter your choice: ");

scanf("%d", &choice);

// Perform selected operation

switch (choice) {

case 1:

fixedPartitioning();

break;

case 2:

dynamicPartitioning();

break;

default:

printf("Invalid choice\n");

}

return 0;

}  
  
