NAME : JANHAVI GATTANI

BATCH : 2

PRN : 12311291

ROLL NO : 37

**OS LAB ASSIGNMENT – MEMORY MANAGEMENT**

1.PLACEMENT STRATEGIES

CODE:

#include <stdio.h>

#include <stdbool.h>

#define MAX 100

int memory[MAX], memSize;

int lastAllocatedIndex = 0; // Used in Next Fit

// Function to initialize memory with -1 (free)

void initializeMemory() {

for (int i = 0; i < memSize; i++)

memory[i] = -1;

}

// Function to display current memory status

void displayMemory() {

printf("Memory: ");

for (int i = 0; i < memSize; i++)

printf("%d ", memory[i]);

printf("\n");

}

// Helper function to check if block of size `size` can be allocated from index `start`

bool canAllocate(int start, int size) {

if (start + size > memSize)

return false;

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

if (memory[start + i] != -1)

return false;

}

return true;

}

// Allocates memory from `start` index

void allocate(int start, int size, int processId) {

for (int i = 0; i < size; i++)

memory[start + i] = processId;

printf("Process %d allocated at index %d\n", processId, start);

}

// FIRST FIT

void firstFit(int size, int processId) {

for (int i = 0; i <= memSize - size; i++) {

if (canAllocate(i, size)) {

allocate(i, size, processId);

return;

}

}

printf("No space for Process %d using First Fit\n", processId);

}

// NEXT FIT

void nextFit(int size, int processId) {

int i = lastAllocatedIndex;

int count = 0;

while (count < memSize) {

if (canAllocate(i, size)) {

allocate(i, size, processId);

lastAllocatedIndex = i + size;

return;

}

i = (i + 1) % memSize;

count++;

}

printf("No space for Process %d using Next Fit\n", processId);

}

// BEST FIT

void bestFit(int size, int processId) {

int bestIndex = -1, bestSize = MAX;

for (int i = 0; i < memSize;) {

if (memory[i] == -1) {

int j = i;

while (j < memSize && memory[j] == -1)

j++;

int blockSize = j - i;

if (blockSize >= size && blockSize < bestSize) {

bestSize = blockSize;

bestIndex = i;

}

i = j;

} else {

i++;

}

}

if (bestIndex != -1)

allocate(bestIndex, size, processId);

else

printf("No space for Process %d using Best Fit\n", processId);

}

// WORST FIT

void worstFit(int size, int processId) {

int worstIndex = -1, worstSize = -1;

for (int i = 0; i < memSize;) {

if (memory[i] == -1) {

int j = i;

while (j < memSize && memory[j] == -1)

j++;

int blockSize = j - i;

if (blockSize >= size && blockSize > worstSize) {

worstSize = blockSize;

worstIndex = i;

}

i = j;

} else {

i++;

}

}

if (worstIndex != -1)

allocate(worstIndex, size, processId);

else

printf("No space for Process %d using Worst Fit\n", processId);

}

// MAIN DRIVER

int main() {

int choice, processSize, processId = 1;

printf("Enter total memory size: ");

scanf("%d", &memSize);

initializeMemory();

while (1) {

printf("\n1. First Fit\n2. Next Fit\n3. Best Fit\n4. Worst Fit\n5. Display Memory\n6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

if (choice >= 1 && choice <= 4) {

printf("Enter process size: ");

scanf("%d", &processSize);

}

switch (choice) {

case 1: firstFit(processSize, processId++); break;

case 2: nextFit(processSize, processId++); break;

case 3: bestFit(processSize, processId++); break;

case 4: worstFit(processSize, processId++); break;

case 5: displayMemory(); break;

case 6: return 0;

default: printf("Invalid choice!\n");

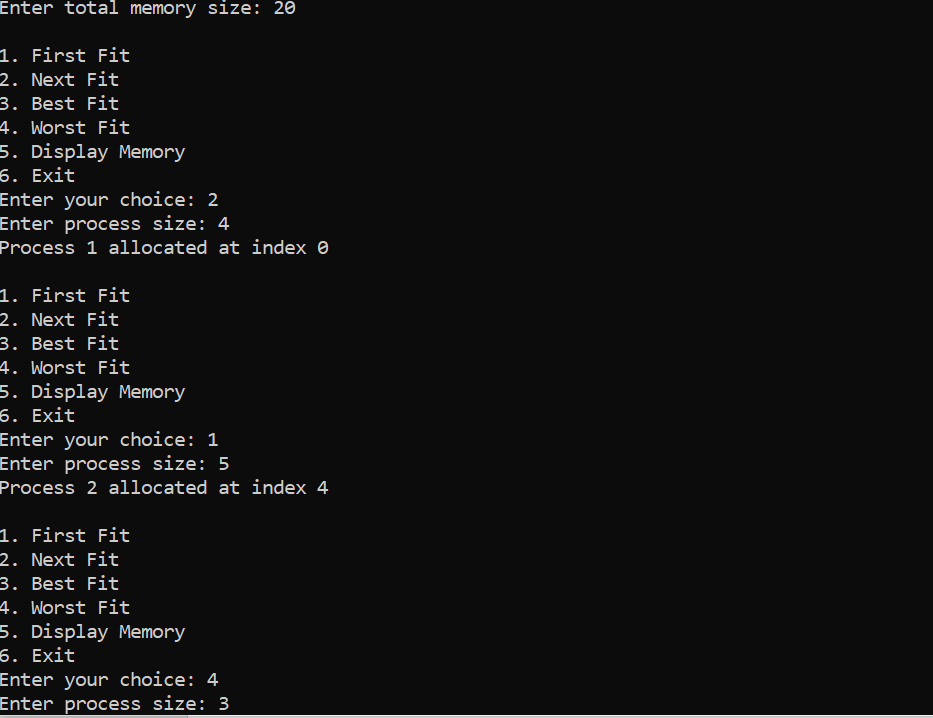
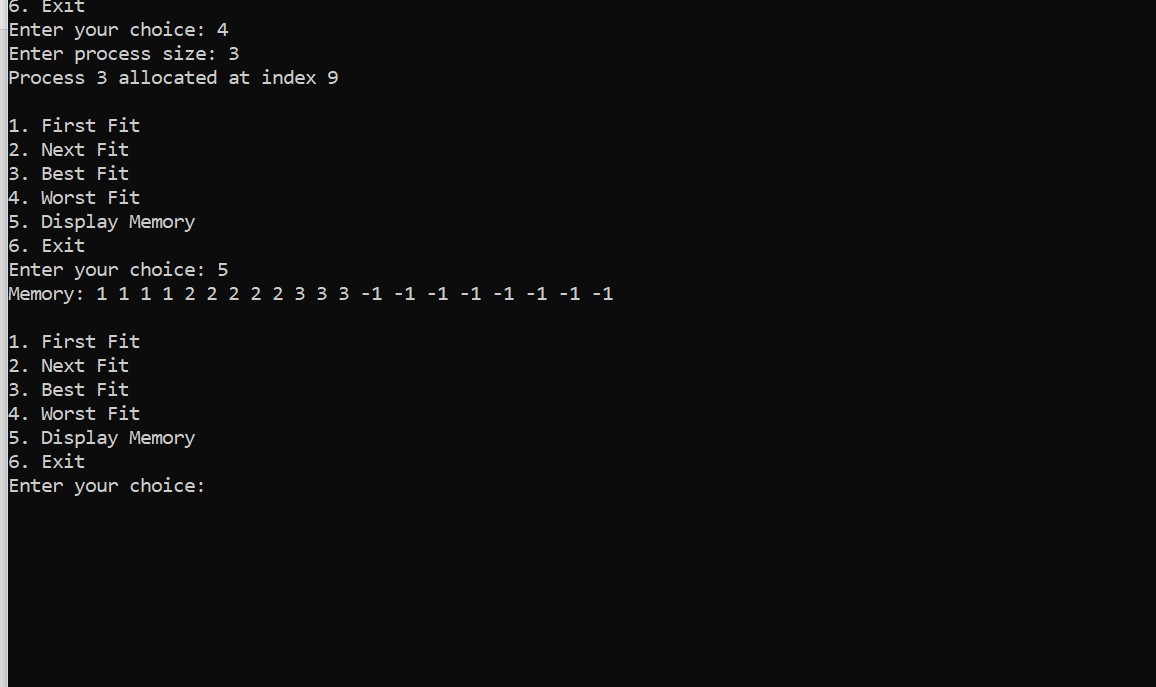
}

}

return 0;

}

OUTPUT :

2.BUDDY SYSTEM

CODE:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define MAX 7 // 2^7 = 128 max block size

// Free list for each block size (index 0 = size 1, ..., index 7 = size 128)

int freeList[MAX + 1];

void initializeMemory() {

for (int i = 0; i <= MAX; i++)

freeList[i] = 0;

freeList[MAX] = 1; // Start with one block of 128

}

int getBlockSizeIndex(int size) {

int i = 0;

int blockSize = 1;

while (blockSize < size && i <= MAX) {

blockSize <<= 1;

i++;

}

return (i > MAX) ? -1 : i;

}

// Allocate memory

void allocate(int size) {

int index = getBlockSizeIndex(size);

if (index == -1) {

printf("Cannot allocate size %d: exceeds maximum block size.\n", size);

return;

}

int i = index;

while (i <= MAX && freeList[i] == 0)

i++;

if (i > MAX) {

printf("No available block for size %d\n", size);

return;

}

// Split blocks down to required size

while (i > index) {

freeList[i]--;

freeList[i - 1] += 2;

i--;

}

freeList[index]--;

printf("Allocated block of size %d\n", 1 << index);

}

// Deallocate memory (simulate adding block back)

void deallocate(int size) {

int index = getBlockSizeIndex(size);

if (index == -1) {

printf("Invalid block size to deallocate.\n");

return;

}

freeList[index]++;

printf("Deallocated block of size %d\n", 1 << index);

// Try merging with buddy

while (index < MAX && freeList[index] >= 2) {

freeList[index] -= 2;

freeList[index + 1]++;

printf("Merged two blocks of size %d into size %d\n", 1 << index, 1 << (index + 1));

index++;

}

}

// Display memory status

void displayMemory() {

printf("Free blocks:\n");

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

printf("Size %3d: %d block(s)\n", 1 << i, freeList[i]);

}

}

// Driver

int main() {

int choice, size;

initializeMemory();

while (1) {

printf("\n1. Allocate Memory\n2. Deallocate Memory\n3. Display Free List\n4. Exit\n");

printf("Enter choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter size to allocate: ");

scanf("%d", &size);

allocate(size);

break;

case 2:

printf("Enter size to deallocate: ");

scanf("%d", &size);

deallocate(size);

break;

case 3:

displayMemory();

break;

case 4:

return 0;

default:

printf("Invalid choice.\n");

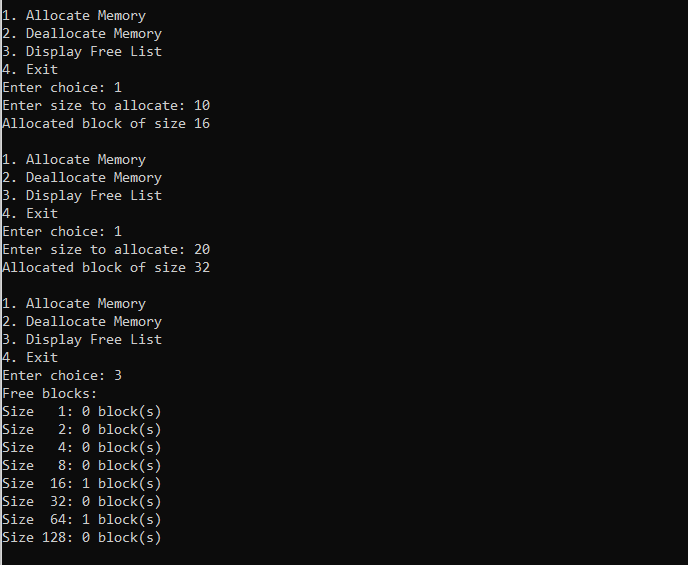
}

}

return 0;

}

OUTPUT:



3. Conversion of logical/virtual address using paging and segmentation.

CODE:

#include <stdio.h>

#include <stdlib.h>

int main() {

int choice;

printf("Select mode:\n1. Paging\n2. Segmentation\nEnter choice: ");

scanf("%d", &choice);

if (choice == 1) {

// Paging

int \*pageTable, numPages, pageSize, logicalAddress, pageNumber, offset, physicalAddress;

printf("\n--- Paging Mode ---\n");

printf("Enter number of pages: ");

scanf("%d", &numPages);

pageTable = (int \*)malloc(numPages \* sizeof(int)); // dynamic memory allocation

printf("Enter page size: ");

scanf("%d", &pageSize);

printf("Enter page table (frame numbers for each page):\n");

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

printf("Page %d -> Frame: ", i);

scanf("%d", &pageTable[i]);

}

printf("Enter logical address: ");

scanf("%d", &logicalAddress);

pageNumber = logicalAddress / pageSize;

offset = logicalAddress % pageSize;

if (pageNumber >= numPages) {

printf("Error: Invalid page number.\n");

} else {

physicalAddress = pageTable[pageNumber] \* pageSize + offset;

printf("Physical Address: %d\n", physicalAddress);

}

free(pageTable); // free dynamically allocated memory

}

else if (choice == 2) {

// Segmentation

int base[5], limit[5], segmentNumber, offset, physicalAddress;

printf("\n--- Segmentation Mode ---\n");

printf("Enter base and limit for 5 segments:\n");

for (int i = 0; i < 5; i++) {

printf("Segment %d - Base: ", i);

scanf("%d", &base[i]);

printf("Segment %d - Limit: ", i);

scanf("%d", &limit[i]);

}

printf("Enter segment number and offset: ");

scanf("%d%d", &segmentNumber, &offset);

if (segmentNumber < 5 && offset < limit[segmentNumber]) {

physicalAddress = base[segmentNumber] + offset;

printf("Physical Address: %d\n", physicalAddress);

} else {

printf("Error: Invalid segment number or offset exceeds limit.\n");

}

}

else {

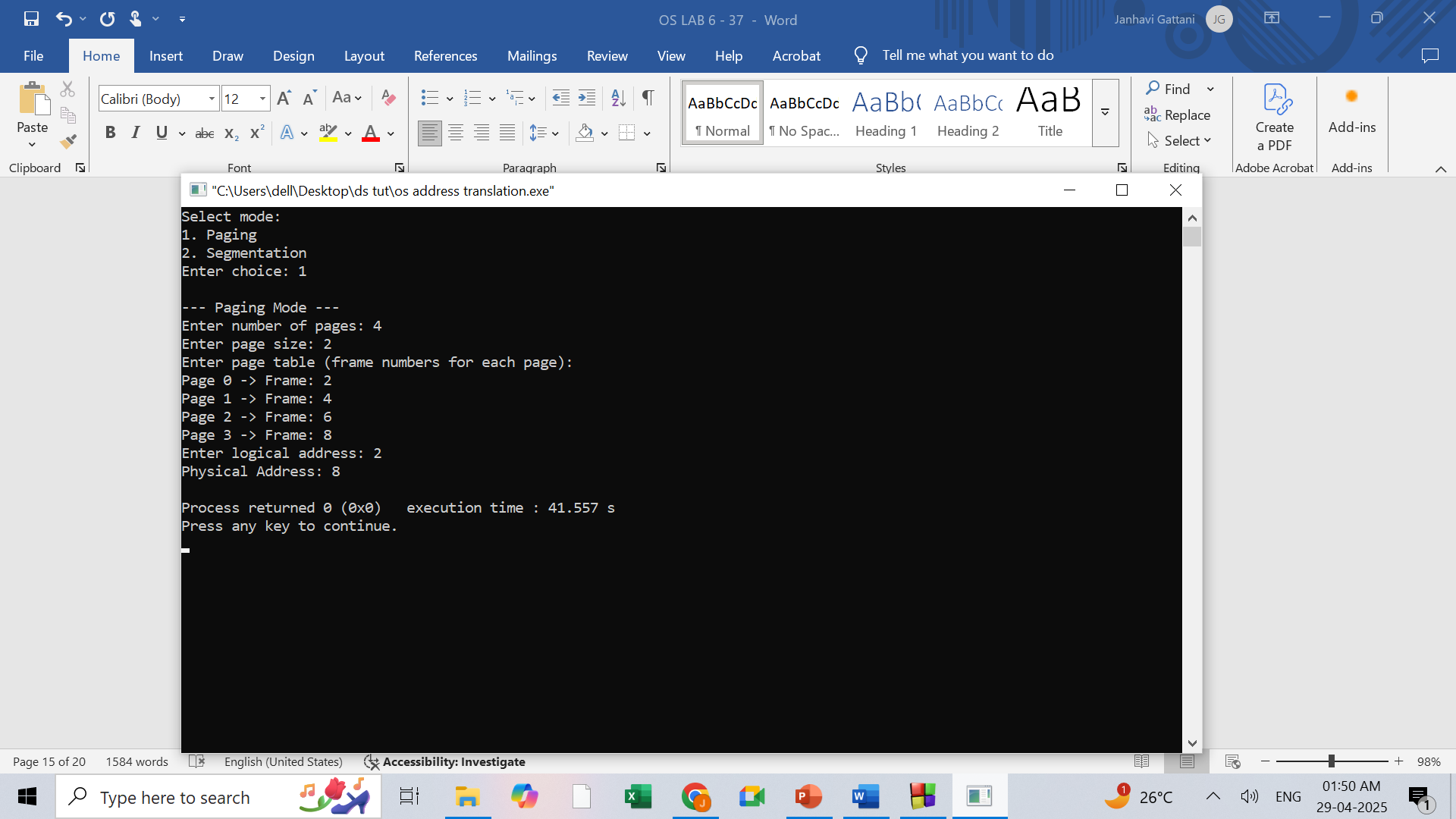
printf("Invalid choice.\n");

}

return 0;

}

OUTPUT:



4.PAGE REPLACEMENT ALGORITHMS

CODE:

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#include <stdbool.h>

#define MAX\_FRAMES 10

#define MAX\_PAGES 100

// Function to check if a page is present in frames

bool isPresent(int frames[], int n, int page) {

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

if (frames[i] == page)

return true;

return false;

}

// FIFO Algorithm

void FIFO(int pages[], int n, int frameCount) {

int frames[MAX\_FRAMES], front = 0, faults = 0;

for (int i = 0; i < frameCount; i++) frames[i] = -1;

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

if (!isPresent(frames, frameCount, pages[i])) {

frames[front] = pages[i];

front = (front + 1) % frameCount;

faults++;

}

}

printf("FIFO Page Faults: %d\n", faults);

}

// LRU Algorithm

void LRU(int pages[], int n, int frameCount) {

int frames[MAX\_FRAMES], recent[MAX\_FRAMES], faults = 0;

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

frames[i] = -1;

recent[i] = 0;

}

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

int page = pages[i];

if (isPresent(frames, frameCount, page)) {

for (int j = 0; j < frameCount; j++)

if (frames[j] == page)

recent[j] = i;

} else {

int lruIndex = 0;

for (int j = 1; j < frameCount; j++)

if (recent[j] < recent[lruIndex])

lruIndex = j;

frames[lruIndex] = page;

recent[lruIndex] = i;

faults++;

}

}

printf("LRU Page Faults: %d\n", faults);

}

// Optimal Algorithm

void Optimal(int pages[], int n, int frameCount) {

int frames[MAX\_FRAMES], faults = 0;

for (int i = 0; i < frameCount; i++) frames[i] = -1;

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

if (!isPresent(frames, frameCount, pages[i])) {

int farthest = -1, index = -1;

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

int k;

for (k = i + 1; k < n; k++) {

if (frames[j] == pages[k])

break;

}

if (k > farthest) {

farthest = k;

index = j;

}

}

frames[index == -1 ? 0 : index] = pages[i];

faults++;

}

}

printf("Optimal Page Faults: %d\n", faults);

}

// Driver Code

int main() {

int pages[MAX\_PAGES], n, frameCount, choice;

printf("Enter number of pages: ");

scanf("%d", &n);

printf("Enter the page reference string:\n");

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

scanf("%d", &pages[i]);

printf("Enter number of frames: ");

scanf("%d", &frameCount);

while (1) {

printf("\n1. FIFO\n2. LRU\n3. Optimal\n4. Exit\nChoose: ");

scanf("%d", &choice);

switch (choice) {

case 1: FIFO(pages, n, frameCount); break;

case 2: LRU(pages, n, frameCount); break;

case 3: Optimal(pages, n, frameCount); break;

case 4: return 0;

default: printf("Invalid choice!\n");

}

}

return 0;

}

OUTPUT:

