**Objectives:**

The program aims to simulate three widely used page replacement algorithms—Optimal Replacement, First-In-First-Out (FIFO) Replacement, and Least Recently Used (LRU) Replacement. It will compute the number of page faults that occur under each algorithm for a given sequence of memory page requests. The goal is to analyze the performance of each algorithm under identical conditions, using user-specified inputs such as the number of page frames and the sequence of page requests. By understanding the behavior of these algorithms, users can evaluate which method is more efficient in minimizing page faults. Additionally, the program will enhance the understanding of memory management concepts in operating systems.

**Write a program to calculate page faults Using the following algorithms:**

1. **Optimal replacement**

**Code:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

int no\_of\_frames, no\_of\_pages;

// Read number of frames

cout << "Enter number of frames: ";

cin >> no\_of\_frames;

// Read number of pages

cout << "Enter number of pages: ";

cin >> no\_of\_pages;

vector<int> ref\_string(no\_of\_pages);

// Read page reference string

cout << "Enter page reference string: ";

for (int i = 0; i < no\_of\_pages; ++i) {

cin >> ref\_string[i];

}

vector<int> frames(no\_of\_frames, -1); // Initialize frames with -1 to indicate empty

int faults = 0;

cout<<"\n================================================================================\n";

for (int i = 0; i < no\_of\_frames; ++i)

cout << "frame[" << i << "]\t";

cout << "page fault";

cout<<"\n================================================================================\n";

for (int i = 0; i < no\_of\_pages; ++i) {

bool hit = false;

// Check if the page is already in the frames (hit)

for (int j = 0; j < no\_of\_frames; ++j) {

if (frames[j] == ref\_string[i]) {

hit = true;

break;

}

}

if (!hit) {

// Page fault occurs

// Check if there's any empty frame

bool inserted = false;

for (int j = 0; j < no\_of\_frames; ++j) {

if (frames[j] == -1) {

frames[j] = ref\_string[i]; // Insert the page in the empty frame

++faults;

inserted = true;

break;

}

}

// If no empty frame is found, apply optimal replacement

if (!inserted) {

// Determine which page to replace (optimal page replacement strategy)

int replace\_index = -1;

int farthest = -1;

for (int j = 0; j < no\_of\_frames; ++j) {

int future\_index = -1;

// Search for the next occurrence of the page in the future

for (int k = i + 1; k < no\_of\_pages; ++k) {

if (frames[j] == ref\_string[k]) {

future\_index = k;

break;

}

}

// If the page is never used again, replace it

if (future\_index == -1) {

replace\_index = j;

break;

}

// Track the page that is used farthest in the future

if (future\_index > farthest) {

farthest = future\_index;

replace\_index = j;

}

}

// Replace the page in the frames

frames[replace\_index] = ref\_string[i];

++faults;

}

}

// Print the frames

cout << "\n";

for (int j = 0; j < no\_of\_frames; ++j) {

cout << frames[j] << "\t\t";

}

if (!hit)

cout << " F"; // Mark page fault

}

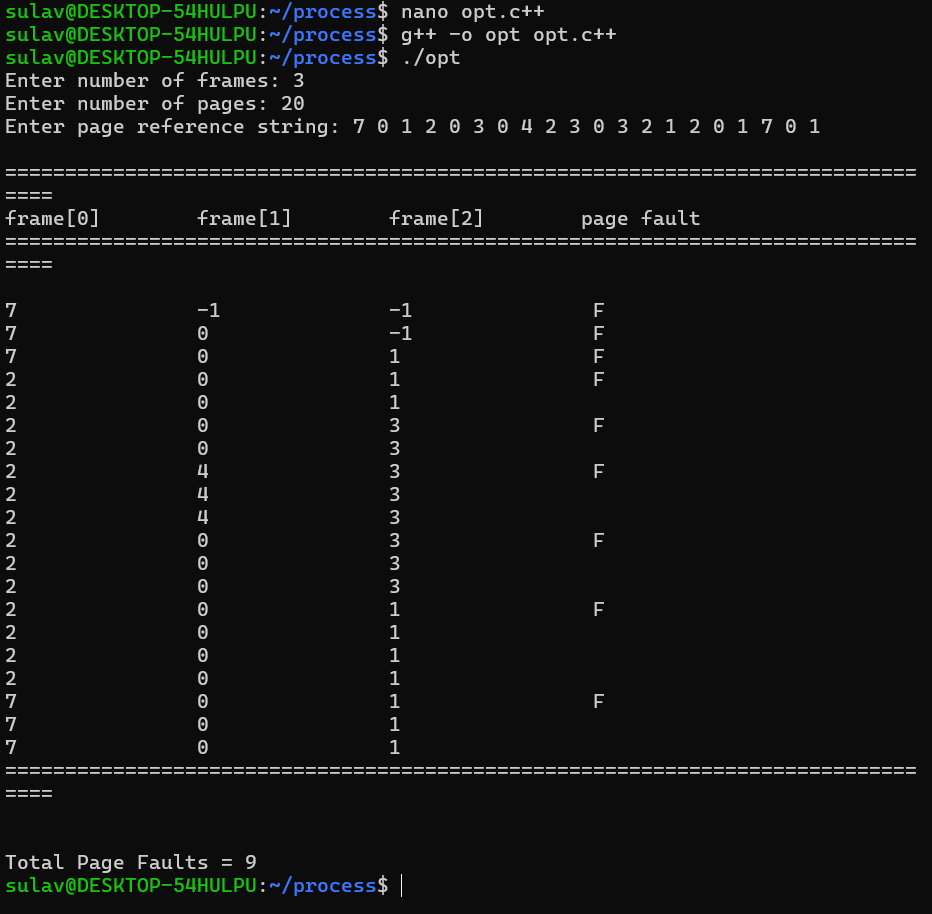
cout << "\n================================================================================\n";

cout << "\n\nTotal Page Faults = " << faults << endl;

return 0;

}

**Output:**

****

1. **FIFO replacement**

**Code:**

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

int main() {

int no\_of\_frames, no\_of\_pages;

// Read number of frames

cout << "Enter number of frames: ";

cin >> no\_of\_frames;

// Read number of pages

cout << "Enter number of pages: ";

cin >> no\_of\_pages;

vector<int> ref\_string(no\_of\_pages);

// Read page reference string

cout << "Enter page reference string: ";

for (int i = 0; i < no\_of\_pages; ++i) {

cin >> ref\_string[i];

}

vector<int> frames(no\_of\_frames, -1);

queue<int> page\_queue;

int faults = 0;

cout<<"\n================================================================================\n";

for (int i = 0; i < no\_of\_frames; ++i)

cout << "frame[" << i << "]\t";

cout << "page fault";

cout<<"\n================================================================================\n";

for (int i = 0; i < no\_of\_pages; ++i) {

bool hit = false;

// Check if the page is already in the frames

for (int j = 0; j < no\_of\_frames; ++j) {

if (frames[j] == ref\_string[i]) {

hit = true;

break;

}

}

if (!hit) {

if (page\_queue.size() == no\_of\_frames) {

// Remove the oldest page from the frames

int oldest\_page = page\_queue.front();

page\_queue.pop();

for (int j = 0; j < no\_of\_frames; ++j) {

if (frames[j] == oldest\_page) {

frames[j] = ref\_string[i];

break;

}

}

} else {

// Find an empty frame and add the new page

for (int j = 0; j < no\_of\_frames; ++j) {

if (frames[j] == -1) {

frames[j] = ref\_string[i];

break;

}

}

}

// Add the new page to the queue

page\_queue.push(ref\_string[i]);

++faults;

}

// Print the frames

cout << "\n";

for (int j = 0; j < no\_of\_frames; ++j) {

cout << frames[j] << "\t\t";

}

if (!hit)

cout << " F";

}

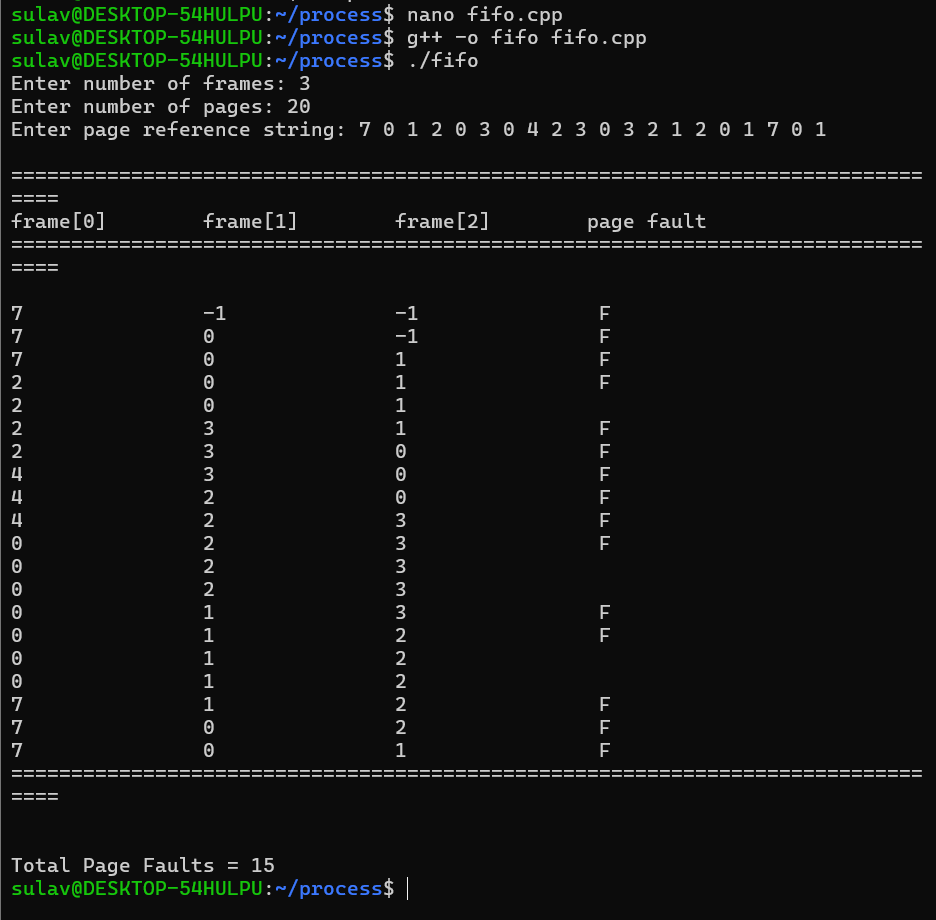
cout << "\n================================================================================\n";

cout << "\n\nTotal Page Faults = " << faults << endl;

return 0;

}

**Output:**

****

1. **LRU replacement**

**Code:**

#include <iostream>

#include <vector>

using namespace std;

int main() {

int no\_of\_frames, no\_of\_pages;

// Read number of frames

cout << "Enter number of frames: ";

cin >> no\_of\_frames;

// Read number of pages

cout << "Enter number of pages: ";

cin >> no\_of\_pages;

vector<int> ref\_string(no\_of\_pages);

// Read page reference string

cout << "Enter page reference string: ";

for (int i = 0; i < no\_of\_pages; ++i) {

cin >> ref\_string[i];

}

vector<int> frames(no\_of\_frames, -1); // -1 indicates an empty frame

vector<int> last\_used(1000, -1); // Track the last used index of pages (assume max page number is 1000)

int faults = 0;

cout << "\n================================================================================\n";

for (int i = 0; i < no\_of\_frames; ++i)

cout << "frame[" << i << "]\t";

cout << "page fault";

cout << "\n================================================================================\n";

for (int i = 0; i < no\_of\_pages; ++i) {

bool hit = false;

// Check if the page is already in the frames

for (int j = 0; j < no\_of\_frames; ++j) {

if (frames[j] == ref\_string[i]) {

hit = true;

break;

}

}

if (!hit) {

// If there is an empty frame, put the page in that frame

bool empty\_frame = false;

for (int j = 0; j < no\_of\_frames; ++j) {

if (frames[j] == -1) { // Empty frame found

frames[j] = ref\_string[i];

empty\_frame = true;

break;

}

}

// If no empty frame, replace the least recently used page

if (!empty\_frame) {

int lru\_index = -1, min\_last\_used = i;

for (int j = 0; j < no\_of\_frames; ++j) {

if (last\_used[frames[j]] < min\_last\_used) {

min\_last\_used = last\_used[frames[j]];

lru\_index = j;

}

}

frames[lru\_index] = ref\_string[i]; // Replace the least recently used page

}

++faults; // Increment page fault count

}

// Update the last used time for the current page

last\_used[ref\_string[i]] = i;

// Print the current frame status

cout << "\n";

for (int j = 0; j < no\_of\_frames; ++j) {

if (frames[j] == -1)

cout << "-\t\t"; // Empty frame

else

cout << frames[j] << "\t\t";

}

if (!hit)

cout << " F"; // Indicate a page fault occurred

}

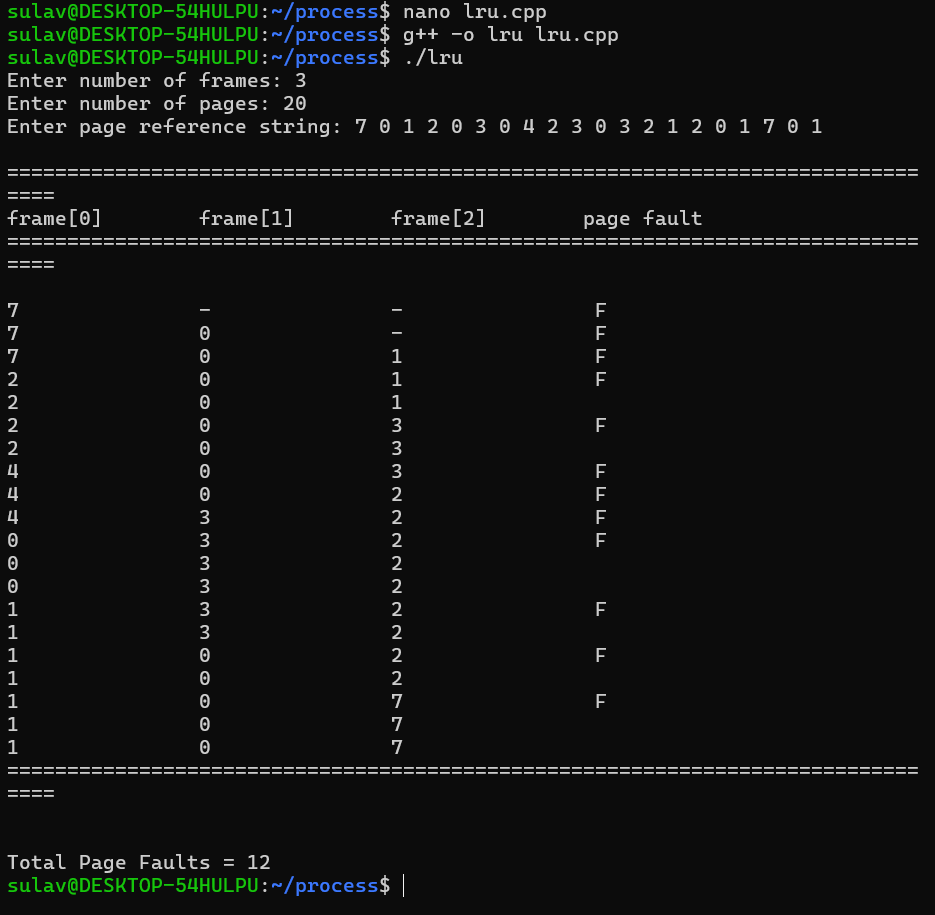
cout << "\n================================================================================\n";

cout << "\n\nTotal Page Faults = " << faults << endl;

return 0;

}

**Output:**

****

**Conclusion:**

The simulation of Optimal Replacement, FIFO Replacement, and LRU Replacement algorithms provides a comprehensive comparison of their efficiency in managing page faults. By analyzing the number of page faults for each algorithm with identical conditions, users gain insight into how different strategies affect memory management. The program’s results will reveal which algorithm is most effective under various scenarios, offering valuable guidance for optimizing memory usage. Furthermore, this exercise will deepen users' understanding of fundamental memory management concepts in operating systems. Ultimately, the insights gained from this analysis can inform better design decisions in system memory management.