Report on FIFO Page Replacement

**Introduction**

This program simulates a FIFO (First-In-First-Out) page replacement algorithm and generates page reference sequences with a given probability of repeated accesses. The purpose is to evaluate page faults under different conditions using local and global FIFO strategies.

**Function: generate\_page**

**Purpose:** Generates multiple sequences of page references based on a probability that determines whether the next page remains the same or changes.

**Explanation:**

1. The function initializes a vector of vectors (pages) to store multiple page sequences.
2. srand(time(0)) is used to ensure randomization across runs.
3. The range of valid page numbers is determined by higher\_range - lower\_range + 1.
4. A loop runs for num\_of\_page\_sequences, generating individual sequences:
   * A random starting number is selected.
   * For each subsequent page reference, a probability check determines whether to repeat the current page or select a new one.
   * A do-while loop ensures that when a new page is chosen, it differs from the previous one.
5. The function returns the generated sequences.

**Function: fifo\_page\_replacement**

**Purpose:** Simulates the FIFO page replacement algorithm for a single sequence and calculates the number of page faults.

**Explanation:**

1. Uses a queue (frame) to maintain page order and an unordered\_set (current\_pages) for fast lookups.
2. Iterates through the page reference sequence:
   * If the page is not in the set, a page fault occurs.
   * If the frame is full, the oldest page is removed from both the queue and the set.
   * The new page is then added to both.
3. The function returns the total page faults encountered.

**Function: fifo\_global**

**Purpose:** Simulates a global FIFO page replacement strategy where multiple processes share the same memory.

**Explanation:**

1. Similar to fifo\_page\_replacement, but processes are iterated sequentially.
2. Keeps track of the total pages processed.
3. If a page is not in memory and the frame is full, the oldest page is removed.
4. The function returns the total number of page faults encountered.

**Main Function**

**Purpose:** Coordinates the program execution by generating page sequences and evaluating them under FIFO policies.

**Explanation:**

1. Defines parameters for sequence generation (length, number of sequences, range, and probability).
2. Calls generate\_page to create test data.
3. Prints the generated sequences for verification.
4. Loops through each sequence:
   * Calls fifo\_page\_replacement with a frame size of 3 and prints page fault results.
   * Calls fifo\_global with a frame size of 4 and prints results.
5. The output provides insight into how FIFO performs for different sequences.

**Conclusion**

The program successfully simulates FIFO page replacement strategies. The local FIFO strategy manages each process separately, while the global FIFO strategy shares frames across all processes. By analyzing page fault counts, this program provides a clear comparison of memory management efficiency under varying conditions.

Source Code

#include <iostream>

#include <vector>

#include <queue>

#include <cstdlib>

#include <ctime>

#include <unordered\_set>

using namespace std;

// Function to generate page sequences with locality of reference

vector<vector<int>> generate\_page(int page\_size, float probability, int length\_of\_sequence, int num\_of\_page\_sequences, int lower\_range, int higher\_range) {

vector<vector<int>> pages(num\_of\_page\_sequences);

srand(time(0)); // Ensure different sequences each run

int range\_size = higher\_range - lower\_range + 1; // Total valid numbers

for (int i = 0; i < num\_of\_page\_sequences; i++) {

int current\_num = rand() % range\_size + lower\_range; // Start with a random number in range

pages[i].push\_back(current\_num);

for (int j = 1; j < length\_of\_sequence; j++) {

if ((rand() % 100) < (probability \* 100)) {

// Stay on the same number based on probability

pages[i].push\_back(current\_num);

}

else {

// Pick a new number in range (different from the current one)

int new\_num;

do {

new\_num = rand() % range\_size + lower\_range;

} while (new\_num == current\_num); // Ensure it's different

pages[i].push\_back(new\_num);

current\_num = new\_num; // Update current number

}

}

}

return pages;

}

// Function to simulate FIFO page replacement for a single process

int fifo\_page\_replacement(const vector<int>& pages, int frame\_size) {

int page\_faults = 0;

queue<int> frame; // FIFO queue for maintaining order

unordered\_set<int> current\_pages; // Set for quick page lookups

for (int page : pages) {

// If the page is NOT already in memory

if (current\_pages.find(page) == current\_pages.end()) {

// If frame is full, remove the oldest page (FIFO)

if (frame.size() == frame\_size) {

int oldest\_element = frame.front(); // Save the element before deleting so we can remove it from the set

frame.pop();

current\_pages.erase(oldest\_element); // Remove from the set

}

// Add the new page

frame.push(page);

current\_pages.insert(page);

page\_faults++;

}

}

return page\_faults;

}

// Function to simulate FIFO page replacement using a global frame pool shared by all processes

int fifo\_global(const vector<vector<int>>& processes, int frame\_size) {

int page\_faults = 0;

queue<int> frame; // FIFO queue for global replacement

unordered\_set<int> current\_pages; // Set for tracking pages in memory

for (const vector<int>& pages : processes) {

for (int page : pages) {

if (current\_pages.find(page) == current\_pages.end()) {

// If frame is full, remove the oldest page (FIFO)

if (frame.size() == frame\_size) {

int oldest = frame.front();

frame.pop();

current\_pages.erase(oldest);

}

// Add the new page

frame.push(page);

current\_pages.insert(page);

page\_faults++;

}

}

}

return page\_faults;

}

int main() {

int length\_of\_sequence = 10;

int num\_of\_page\_sequences = 3;

int lower\_range = 0;

int higher\_range = 9;

float probability = 0.3; // 30% chance to stay on the same number

int local\_frame\_size = 3; // Each process has 3 frames in local FIFO

int global\_frame\_size = 4; // All processes share 4 frames in global FIFO

// Generate page sequences

vector<vector<int>> pages = generate\_page(10, probability, length\_of\_sequence, num\_of\_page\_sequences, lower\_range, higher\_range);

// Print the generated sequences

cout << "\nGenerated Page Sequences:\n";

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

cout << "Sequence " << i << ": ";

for (int num : pages[i]) {

cout << num << " ";

}

cout << endl;

}

// Calculate and print page faults for local FIFO

cout << "\nLocal FIFO Page Faults:\n";

for (int i = 0; i < num\_of\_page\_sequences; i++) {

cout << "Process " << i << " Page Faults: " << fifo\_page\_replacement(pages[i], local\_frame\_size) << endl;

}

// Calculate and print page faults for global FIFO

cout << "\nGlobal FIFO Page Faults:\n";

cout << "Total Page Faults: " << fifo\_global(pages, global\_frame\_size) << endl;

return 0;

}