Programming Project 5 Report

Programming Project 5 Report

Author: Oscar Lopez
Class: CS 433 Operating Systems | Section 1
Professor: Dr. Zhang

Summary

This report documents the implementation of page replacement algorithms for Assignment 5. The project includes implementations of FIFO (First-In-First-Out), LIFO (Last-In-First-Out), and LRU (Least Recently Used). These algorithms are tested against common page reference traces and evaluated for runtime efficiency and correctness. All implementations conform to a unified interface and adhere to object-oriented design principles.

Implementation Details

The simulation is written in C++ and follows an object-oriented structure. A base class defines the common interface,
while each page replacement strategy inherits and implements algorithm-specific logic.

Architecture Overview

All algorithms derive from an abstract class `Replacement`, which provides a consistent interface. This design supports clean modularity and easy substitution of replacement strategies.

Class Hierarchy:

class Replacement (base class)
  FIFOReplacement
  LIFOReplacement
  LRUReplacement

Key Components

- Page Frame Table: Stores active pages
- Access Tracker: Used by LRU to track usage
- Eviction Logic: Specific to each algorithm
- Statistics: Tracks page faults and replacements

Algorithm-Specific Implementations

FIFOReplacement:
- Evicts the oldest page in a queue
- Time Complexity: O(1)

LIFOReplacement:
- Evicts the most recently inserted page using a stack
- Time Complexity: O(1)

LRUReplacement (Clock-based):
- Approximates LRU using a reference-bit clock buffer
- Uses a circular array and map for efficient lookup
- Time Complexity: O(n) in worst-case, O(1) amortized

Implementation-Specific Details

The following outlines the exact implementation techniques used in the C++ source files for each page replacement algorithm.

FIFOReplacement (fifo_replacement.cpp):
- Uses `std::queue<int> fifo_queue` to store the order of page arrivals.
- Pages are inserted using `fifo_queue.push()` and evicted using `fifo_queue.front()` then popped.
- A `std::unordered_set<int> page_table` is used for fast membership checking.
- Eviction occurs only when page_table reaches max frame size.

LIFOReplacement (lifo_replacement.cpp):
- Implements a stack structure using `std::vector<int> lifo_stack`.
- New pages are pushed to the back of the vector; the last page is evicted when needed.
- A `std::unordered_set<int> page_table` ensures O(1) lookup and removal.
- This structure leads to very high turnover under certain workloads.

LRUReplacement (lru_replacement.h / lru_replacement.cpp):
- Combines a fixed-size `int* clock_buffer` with a clock hand pointer for LRU approximation.
- A parallel `bool* reference_bits` tracks recent usage of pages.
- `std::unordered_map<int, int> page_to_index` maps pages to indices in the buffer.
- On access: updates the page's reference bit to true.
- On eviction: rotates clock hand and clears bits until it finds one with a 0.
- This approximates true LRU behavior with lower computational overhead.

Statistics and Profiling:
- Each algorithm logs page faults and evictions using counters defined in the base class `Replacement`.
- Execution time is captured using `std::chrono::high_resolution_clock` to ensure compliance with < 1s runtime for large inputs.
- All results are printed in a standardized format via the main simulator.

Unified Interface (replacement.h):
- All strategies implement `insert(int page)`, `evict()`, `exists(int page)`, and `access(int page)`.
- This design ensures algorithms are interchangeable at runtime and simplifies testing.

These implementation choices optimize each algorithm for clarity, performance, and ease of debugging.

Performance Metrics

Each algorithm tracks total references, page faults, and evictions.
Timing is captured using <chrono> to verify compliance with runtime requirements (< 1s on large inputs).

Sample Output - Small Trace:
Logical address: 1048576, page number: 1024, frame = 0, fault? 1
Logical address: 1049600, page number: 1025, frame = 1, fault? 1

Sample Output - Large Trace (LRU):
Number of references: 20000

Page faults: 1500
Replacements: 1490
Elapsed time: 1.1342 seconds

Features

- Multiple page replacement algorithms
- Clean inheritance-based design
- Precise statistics for evaluation
- Modular architecture

Advanced Features

- Clock-based approximation for LRU
- Runtime profiling using <chrono>
- Dynamic frame sizing via CLI
- Efficient memory and container management

Limitations and Drawbacks

- LRU implementation is not true LRU (uses Clock approximation)
- No modeling of I/O cost or dirty pages
- All pages assumed to be equal cost
- No demand-based prefetching

Compilation and Execution

Compilation Command:
g++ -std=c++17 main.cpp fifo_replacement.cpp lifo_replacement.cpp lru_replacement.cpp pagetable.cpp replacement.cpp -o pager_sim

Execution Command:
./pager_sim 1024 16

Input files required:
- small_refs.txt
- large_refs.txt

Conclusion

This project highlights key trade-offs in page replacement strategies. While FIFO and LIFO offer low runtime overhead,
they tend to underperform in real-world scenarios. The Clock-based LRU implementation provides a good balance between
runtime efficiency and replacement accuracy. Overall, the simulation demonstrates foundational OS memory management concepts.