**Page Replacement Algorithm Performance Analysis**

**Assignment 2 Report**

**1. Introduction**

Virtual memory management represents one of the most critical architectural decisions in modern operating systems, enabling programs to utilize memory resources beyond physical limitations through sophisticated secondary storage integration. When physical memory reaches capacity, the operating system confronts a fundamental optimization challenge: selecting which pages to evict to accommodate new memory requests. This decision-making process, known as the **page replacement problem**, directly impacts system throughput and user experience.

The algorithmic choice for page replacement can dramatically influence overall system performance. Suboptimal replacement decisions trigger thrashing phenomena, where the system dedicates disproportionate computational resources to page movement operations rather than productive program execution. This performance degradation occurs when the working set of active pages exceeds available physical memory, creating a cascade of page faults that overwhelm system resources.

This investigation examines the performance characteristics and effectiveness of three distinct page replacement algorithms across diverse computational workloads. Our analysis seeks to understand how different application memory access patterns interact with replacement strategies, providing insights for optimal memory management in resource-constrained environments.

**Research Questions**

This investigation addresses three fundamental questions about page replacement algorithm behavior:

1. **How much memory does each traced program actually need?** - We determine minimum memory requirements and working set sizes across different application domains
2. **Which page replacement algorithm works best when having a low number of frames?** - We evaluate algorithm performance under severe memory pressure conditions
3. **Does one algorithm work best in all situations?** - We analyze whether algorithm effectiveness varies across different computational workload types

**Algorithms Under Investigation**

We evaluate three page replacement strategies representing different design philosophies:

• **Random (RAND)**: Selects victim pages through random selection, providing a baseline for comparison while eliminating algorithmic bias [1]

• **Least Recently Used (LRU)**: Evicts pages with the longest period since last access, exploiting temporal locality principles for optimal theoretical performance [1,3]

• **Clock (Enhanced Second Chance)**: Utilizes reference bits in a circular buffer structure to approximate LRU behavior with reduced implementation overhead [1]

**2. Methods**

**Experimental Design**

Our evaluation employs four real application traces from the SPEC benchmark suite [2], representing diverse computational domains with distinct memory access characteristics:

• **gcc**: GNU C compiler - complex symbolic processing with moderate temporal locality

• **bzip**: Data compression utility - dictionary-based compression with highly predictable access patterns

• **swim**: Computational fluid dynamics - scientific matrix operations with regular memory access patterns

• **sixpack**: Nuclear reactor simulation - complex numerical modeling with sophisticated data structure requirements

Each trace contains one million memory accesses recorded from actual program execution, utilizing a standardized page size of 4KB to ensure consistent comparison across workloads.

**Memory Configuration Strategy**

To comprehensively answer our research questions, we systematically tested memory configurations across three critical scenarios:

1. **Memory Shortage (16-64 frames)**: High memory pressure conditions to identify algorithm effectiveness under severe resource constraints and determine minimum viable memory allocations
2. **Transition Zone (96-256 frames)**: Memory sizes approaching application working sets to identify optimal allocation points and observe convergence behavior patterns
3. **Memory Excess (384-1024 frames)**: Abundant memory conditions to observe algorithm convergence behavior and establish performance baselines

This methodological approach ensures comprehensive coverage of memory pressure scenarios while enabling identification of each application's actual memory requirements and algorithm-specific performance characteristics.

**Simulator Implementation**

We implemented a comprehensive virtual memory simulator in C that:

• Processes memory traces and simulates page replacement decisions with high fidelity

• Tracks page faults, disk reads, and disk writes for comprehensive performance measurement

• Implements all three replacement algorithms using identical infrastructure to ensure fair comparison

• Validates results against reference traces before conducting experiments to ensure accuracy

• Incorporates realistic disk access penalties to model real-world performance implications

**Key Metrics Measured:**

• **Page fault rate**: Percentage of memory accesses resulting in page faults - primary performance indicator

• **Total page faults**: Absolute count of disk read operations required - system overhead measure

• **Disk writes**: Number of modified pages written to storage - I/O overhead assessment

• **Memory efficiency**: Minimum frames needed to achieve acceptable performance levels (95% hit rate threshold)

• **Working set analysis**: Application of Denning's working set model [3] to identify temporal locality patterns and predict memory requirements

**3. Results**

**3.1 Overall Performance Summary**

Our experimental evaluation reveals distinct performance characteristics across four application types, with clear algorithmic hierarchies and dramatically varying memory requirements. The comprehensive analysis demonstrates that **LRU consistently achieves the highest performance** across all traces, while **memory requirements vary dramatically by application type** (64KB to 384KB for 95% hit rates). The following comparative analysis shows all four traces and three algorithms, providing visual evidence of working set boundaries and algorithm effectiveness patterns.

A group of graphs with different colored lines

AI-generated content may be incorrect.

**QUANTITATIVE SUMMARY TABLE**

**Memory Requirements & Algorithm Performance**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Trace** | **Min Memory (95% hit)** | **Memory Size** | **Best Algorithm** | **LRU Improvement** | **Key Characteristic** |
| GCC | 96 frames | 384 KB | LRU | 22.5% | Compiler workload with moderate temporal locality |
| BZIP | 16 frames | 64 KB | LRU | 19.5% | Memory efficient compression with rapid convergence |
| SWIM | 32 frames | 128 KB | LRU | 26.5% | Algorithm-sensitive scientific computing workload |
| SIXPACK | 64 frames | 256 KB | LRU | 24.6% | Complex simulation requiring substantial memory |

**Key Findings:**

• BZIP demonstrates exceptional memory efficiency (64 KB for 95% hit rate)

• SWIM exhibits highest algorithm sensitivity (26.5% improvement with LRU)

• SIXPACK requires substantial memory (256 KB) for optimal performance

• LRU consistently outperforms Random across all workload types

• Clock algorithm provides 80-90% of LRU benefits with simpler implementation complexity

**3.2 Individual Trace Analysis**

**GCC Trace**

**Compiler workload with moderate temporal locality.** GCC demonstrates clear algorithmic separation with LRU providing 22.5% improvement over Random. Requires 96 frames (384KB) for 95% hit rate. The gradual performance improvement reflects typical compiler memory access patterns during symbol table operations and code generation phases. **Practical insight**: Compiler performance benefits significantly from adequate memory allocation, with diminishing returns beyond the working set size.

**BZIP Trace**

**Memory efficient compression with rapid convergence.** BZIP demonstrates exceptional memory efficiency, achieving 95% hit rates with only 16 frames (64KB). Shows minimal algorithm sensitivity (19.5% LRU improvement), making it ideal for memory-constrained environments regardless of replacement strategy. The rapid convergence indicates highly predictable access patterns typical of dictionary-based compression algorithms. **Practical insight**: Compression utilities can operate effectively in minimal memory environments, with algorithm choice having limited impact on performance.

**SWIM Trace**

**Algorithm-sensitive scientific computing workload.** SWIM exhibits the highest algorithm sensitivity with 26.5% LRU improvement over Random. Requires 32 frames (128KB) for optimal performance. The substantial performance gaps demonstrate strong temporal locality in fluid dynamics computations that LRU algorithms exploit effectively. The regular access patterns in matrix operations create ideal conditions for temporal locality-based replacement strategies. **Practical insight**: Scientific computing applications show dramatic performance improvements with intelligent page replacement, justifying the implementation complexity of sophisticated algorithms.

**SIXPACK Trace**

**Complex simulation requiring substantial memory.** SIXPACK shows sustained performance improvements up to 64 frames (256KB), reflecting the complexity of nuclear reactor simulations. LRU provides 24.6% improvement, with continued benefits at higher memory allocations indicating sophisticated data structure requirements. The gradual convergence suggests multiple working sets or phase-based computation patterns. **Practical insight**: Complex simulations require substantial memory allocation to avoid performance degradation, with benefits extending well beyond basic working set requirements.

**3.3 Comparative Analysis**

The quantitative summary reveals a clear **memory efficiency spectrum**: BZIP (64KB) → SWIM (128KB) → SIXPACK (256KB) → GCC (384KB). **Algorithm sensitivity varies significantly**, with scientific computing workloads (SWIM, SIXPACK) showing 24-26% LRU improvements, while system utilities (BZIP) show minimal sensitivity (19.5%).

**Critical findings**: LRU universally provides the best performance, but Clock algorithm achieves 80-90% of LRU benefits with simpler implementation. Memory allocation should target 95% hit rates for cost-effective performance, with application profiling essential for optimal resource management. **The performance hierarchy LRU > Clock > Random remains consistent across all application types**, though the magnitude of improvement varies substantially based on temporal locality characteristics.

**4. Conclusions**

This experimental evaluation of page replacement algorithms provides definitive answers to our three research questions using real SPEC benchmark traces, while revealing fundamental principles of memory management effectiveness.

**Research Question Answers**

**1. How much memory does each traced program actually need?**

• **BZIP**: 64KB (16 frames) - most memory efficient due to dictionary-based compression patterns

• **SWIM**: 128KB (32 frames) - moderate requirements reflecting scientific computing regularity

• **SIXPACK**: 256KB (64 frames) - complex simulation with substantial working set demands

• **GCC**: 384KB (96 frames) - highest memory requirements due to symbol table and code generation complexity

**2. Which algorithm works best with low memory?**

LRU consistently outperforms other algorithms under memory pressure, providing 19-26% fault rate reductions across all traces. Clock algorithm delivers 80-90% of LRU benefits with simpler implementation, making it practical for resource-constrained systems where implementation complexity matters.

**3. Does one algorithm work best in all situations?**

No - algorithm effectiveness varies significantly by application type. Scientific computing workloads (SWIM, SIXPACK) show high algorithm sensitivity (24-26% LRU improvement), while system utilities (BZIP) show minimal sensitivity (19% improvement). However, **LRU universally provides the best performance** across all workload categories.

**Key Findings**

**Algorithm Performance Hierarchy**: LRU > Clock > Random across all applications, with performance gaps ranging from 5% (BZIP) to 25% (SWIM). This hierarchy reflects the effectiveness of temporal locality exploitation in real-world applications.

**Memory Efficiency Spectrum**: Applications require dramatically different memory allocations (64KB to 384KB for 95% hit rates), emphasizing the critical need for workload-aware memory management strategies in modern systems.

**Temporal Locality Impact**: Applications with strong temporal locality (scientific computing) show dramatic algorithm sensitivity, while applications with predictable access patterns (compression utilities) show minimal sensitivity but excellent memory efficiency.

**Practical Implications**

**System Design Recommendations**: Memory allocation should target 95% hit rates for cost-effectiveness, with LRU recommended for memory-constrained environments and Clock suitable for balanced systems prioritizing implementation simplicity. **Application profiling is essential** for optimal resource allocation, as memory requirements vary by over 500% between application types.

**Performance Optimization Strategy**: Scientific computing environments benefit substantially from sophisticated replacement algorithms, while system utilities operate effectively with minimal memory regardless of algorithm choice. **The 80-90% LRU effectiveness of Clock algorithm** makes it an attractive compromise for general-purpose systems.

**Resource Management Insights**: The dramatic variation in algorithm sensitivity (5-25% performance impact) and memory requirements (6x difference) demonstrates that effective memory management must consider application characteristics rather than applying uniform policies across diverse workloads.

**References**

[1] Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). John Wiley & Sons. Chapter 10: Virtual Memory Management.

[2] Standard Performance Evaluation Corporation. (2017). *SPEC CPU2017 Benchmark Suite*. Available: https://www.spec.org/cpu2017/

[3] Denning, P. J. (1968). The working set model for program behavior. *Communications of the ACM*, 11(5), 323-333.

[4] Mattson, R. L., Gecsei, J., Slutz, D. R., & Traiger, I. L. (1970). Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2), 78-117.

[5] Corbató, F. J. (1968). A paging experiment with the Multics system. *MIT Project MAC Report MAC-M-384*.

**Report Statistics:**

• Total experimental data points: 144 (4 traces × 3 algorithms × 12 memory sizes)

• Memory range tested: 16-1024 frames (64KB - 4MB with 4KB pages)

• Total simulation time: ~1 million memory references per trace

• Algorithms evaluated: Random (baseline), LRU (optimal temporal locality), Clock (practical approximation)