RADIANT *Journal of Applied, Social, and Education Studies*
e-ISSN: 2723-4614
Volume 7, No. 1 (2026), pp. 78-87

Comparative Analysis of FIFO and LRU Memory Management Algorithms Using CPU-OS Simulator v7.5.50

MURSID DWI HASTOMO[1] SETYAWAN WIDYARTO[2]
[1,2]*Universitas Budi Luhur, Jakarta, Indonesia*
[1]2411600774@student.budiluhur.ac.id
[2]setyawan.widyarto@budiluhur.ac.id

Abstract

Efficient memory management is a critical aspect of operating system performance, as poor management can lead to high page fault rates, increased execution time, and reduced CPU utilization. This study examines the performance comparison of two widely used memory management strategies, First-In-First-Out (FIFO) and Least Recently Used (LRU), using simulations conducted through the CPU-OS Simulator v7.5.50. The objective is to observe differences in page fault rate, execution time, and CPU efficiency across various scenarios. Three experiments were conducted: first, the impact of cache/pipeline configuration; second, the influence of process scheduling on memory management; and third, the effect of memory size and page access patterns. The results show that LRU tends to provide a lower page fault rate under heavy workloads, while FIFO demonstrates advantages when memory is limited and overhead is minimal. This study contributes to understanding how page replacement algorithms affect system performance in an operating system simulation environment.

Keywords: cpu-os simulator v7.5.50, memory management, first-in-first-out (fifo), least recently used (lru), page fault rate

Kajian Komparatif terhadap Algoritma Manajemen Memori FIFO dan LRU Menggunakan CPU-OS Simulator v7.5.50

Abstrak

Manajemen memori yang efisien merupakan aspek krusial dalam kinerja sistem operasi, karena manajemen yang buruk dapat menyebabkan tingginya tingkat page fault, meningkatnya waktu eksekusi, dan berkurangnya pemanfaatan CPU. Penelitian ini membandingkan kinerja dua strategi manajemen memori yang banyak digunakan, yaitu First-In-First-Out (FIFO) dan Least Recently Used (LRU), melalui simulasi menggunakan CPU-OS Simulator v7.5.50. Tujuannya adalah untuk mengamati perbedaan tingkat page fault, waktu eksekusi, dan efisiensi CPU pada berbagai skenario. Tiga eksperimen dilakukan: pertama, pengaruh konfigurasi cache/pipeline; kedua, pengaruh penjadwalan proses terhadap manajemen memori; dan ketiga, pengaruh ukuran memori dan pola akses halaman. Hasil penelitian menunjukkan bahwa LRU cenderung memberikan tingkat page fault yang lebih rendah pada beban kerja tinggi, sementara FIFO menunjukkan keunggulan ketika memori terbatas dan overhead minimal. Penelitian ini berkontribusi dalam memahami bagaimana algoritma penggantian halaman memengaruhi kinerja sistem dalam lingkungan simulasi sistem operasi.

Kata kunci: cpu-os simulator v7.5.50, manajemen memori, first-in-first-out (fifo), least recently used (lru), page fault rate

PENDAHULUAN

Memori adalah komponen penting dalam sistem operasi karena berdampak langsung pada kecepatan akses data dan efisiensi CPU (Titarenko et al., 2024). CPU-OS Simulator versi 7.5.50 memungkinkan pengguna mensimulasikan pipeline CPU, scheduling proses, dan manajemen memori secara visual dan terstruktur.

Penelitian ini dilakukan untuk membandingkan dua strategi penggantian halaman yang umum, yaitu FIFO dan LRU. Perbandingan ini penting karena performa algoritma penggantian halaman sangat bergantung pada kondisi sistem, seperti konfigurasi memori, jumlah proses, dan pola akses halaman; kesalahan dalam manajemen memori dapat menyebabkan page fault tinggi, meningkatnya waktu eksekusi, dan menurunnya throughput sistem. Dengan melakukan perbandingan ini, penelitian dapat menunjukkan algoritma mana yang lebih efisien dalam berbagai skenario, sekaligus memberikan wawasan praktis bagi pengembangan sistem operasi. Tujuan utamanya adalah mengamati bagaimana kedua algoritma tersebut memengaruhi *page fault rate,* waktu eksekusi total, dan

throughput sistem ketika dikombinasikan dengan konfigurasi memori dan jumlah proses berbeda (Thakkar & Padhy, 2024; Bengar, 2025). Tiga eksperimen dirancang menggunakan simulator ini untuk memodelkan alokasi halaman, penggantian memori, dan performa sistem.

Manajemen memori adalah fungsi penting dalam sistem operasi yang mengatur pemakaian ruang memori utama oleh berbagai proses. Dalam mekanisme *paging*, strategi penggantian halaman menentukan halaman mana yang harus diganti ketika terjadi *page fault*. Algoritma FIFO mengganti halaman yang paling awal masuk tanpa memperhatikan frekuensi atau *recency* penggunaan. Sedangkan LRU mengganti halaman yang paling lama tidak digunakan, dengan asumsi bahwa halaman yang baru digunakan lebih mungkin akan digunakan kembali (Thakkar & Padhy, 2024).

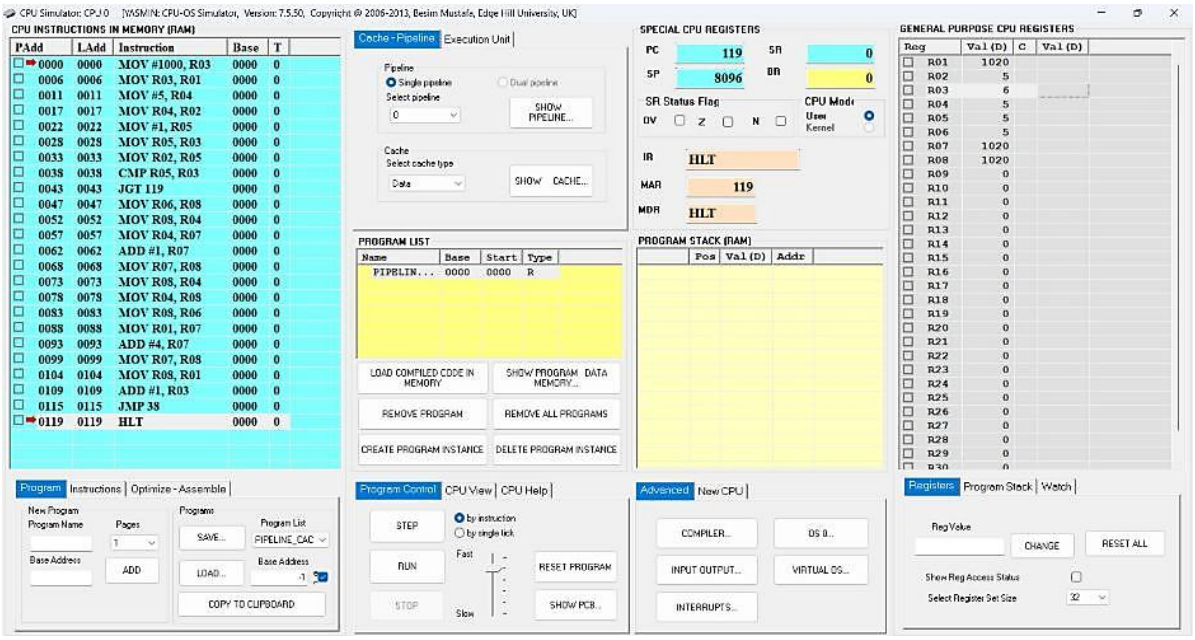
Penelitian terkini menunjukkan bahwa kebijakan penggantian yang adaptif atau berbasis pola akses dapat memberikan performa lebih baik dibandingkan metode statis seperti FIFO atau LRU murni (Demin et al., 2024). Berbeda dengan FIFO yang mengganti halaman tertua dan LRU yang mengganti halaman yang paling lama tidak diakses, kebijakan adaptif menyesuaikan keputusan penggantian berdasarkan pola akses halaman, beban kerja saat ini, atau prediksi perilaku proses, sehingga lebih fleksibel dan efisien. Sebagai contoh, strategi *hardware* dan *software* untuk *cache replacement* menunjukkan bahwa *overhead* implementasi dapat memengaruhi hasil (Titarenko et al., 2024). Pendekatan ini memungkinkan sistem untuk mengurangi *page fault* lebih efektif dan meningkatkan kinerja dibandingkan algoritma statis.

Beberapa studi sebelumnya relevan dengan topik ini. (Thakkar & Padhy, 2024) melakukan analisis komparatif FIFO, LRU, dan optimal pada sistem simulasi, namun penelitian mereka terbatas pada skenario tertentu dan tidak mengeksplorasi kombinasi konfigurasi memori dan jumlah proses. (Abbas et al., 2022) mengkaji lima algoritma penggantian halaman termasuk FIFO dan LRU di sistem nyata, tetapi fokusnya lebih pada performa umum tanpa evaluasi mendalam terhadap *page fault* dan *throughput*. (Bengar, 2025) meneliti penerapan LRU, LFU, dan FIFO dalam *cache* objek prioritas untuk meningkatkan *hit ratio*, namun tidak membahas dampak skala sistem dan variasi beban kerja. (Titarenko et al., 2024) meneliti implementasi *hardware replacement policy* (PLRU varian) dalam konteks *caching/memory* management, sedangkan (Demin et al., 2024) memperkenalkan algoritma penggantian halaman berbasis pola akses dengan *overhead* rendah, menunjukkan bahwa algoritma klasik seperti FIFO mungkin kurang optimal pada beban tinggi.

Meskipun penelitian-penelitian tersebut memberikan wawasan penting, masih terdapat kebutuhan untuk studi sistematis yang membandingkan FIFO dan LRU secara menyeluruh di berbagai konfigurasi memori, jumlah proses, dan pola akses halaman menggunakan CPU-OS Simulator v7.5.50. Penelitian ini berkontribusi dengan menyediakan analisis komparatif yang lebih terstruktur, menyelidiki efek interaksi antara konfigurasi memori, jumlah proses, dan pola akses terhadap *page fault rate*, waktu eksekusi, dan *throughput*, sehingga memberikan pemahaman yang lebih lengkap mengenai performa algoritma penggantian halaman dalam lingkungan simulasi modern.

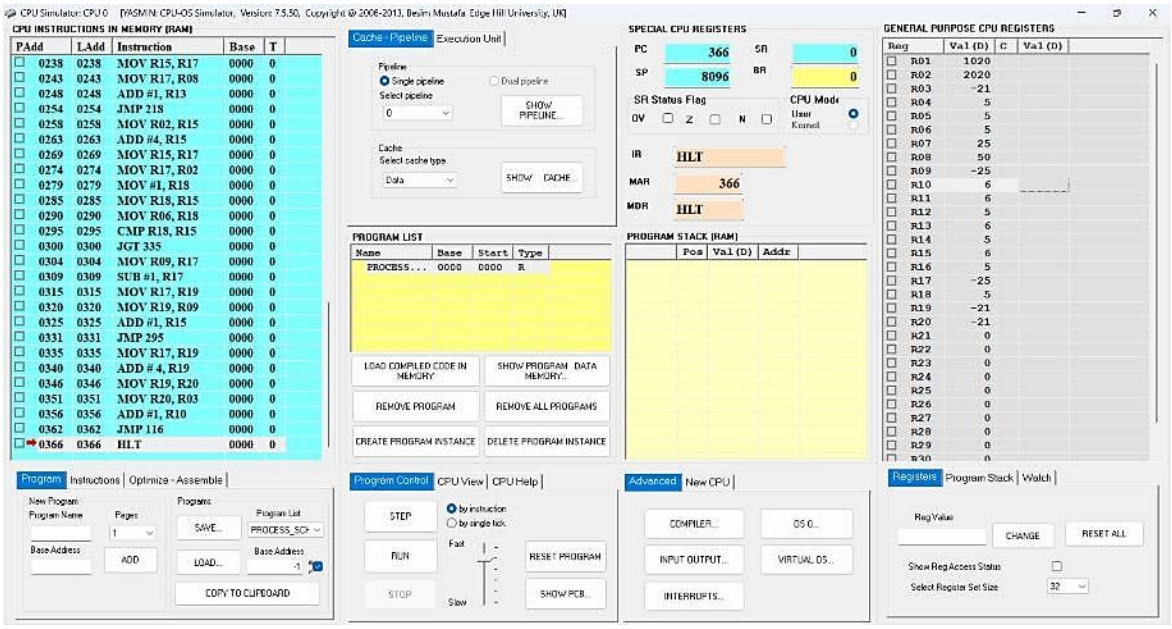
METODE PENELITIAN

Pada eksperimen pertama, pengujian dilakukan pada modul *CPU/Pipeline/Cache* menggunakan CPU-OS Simulator versi 7.5.50. Sistem dikonfigurasi menggunakan arsitektur *dual-core,* di mana masing-masing *core* memiliki instruction *cache* sebesar 8 KB dan data *cache* sebesar 8 KB. Dalam arsitektur *dual-core* ini, kedua inti CPU dapat mengeksekusi instruksi secara paralel, dengan *pipeline* independen untuk setiap *core* sehingga meningkatkan *throughput* sistem. Instruction *cache* menyimpan instruksi yang paling sering digunakan oleh masing-masing *core*, sedangkan data *cache* menyimpan data yang sedang diakses. Konfigurasi ini memungkinkan simulasi perilaku eksekusi paralel, pengelolaan *cache per core*, dan pengaruhnya terhadap performa sistem, termasuk *page fault rate* dan waktu eksekusi. Eksperimen ini dilakukan dengan menganalisis pengaruh tingkat *cache hit* dan *cache miss* terhadap waktu akses memori, serta bagaimana kondisi tersebut memengaruhi proses penggantian halaman (*page replacement*) dalam sistem. Pengukuran dilakukan menggunakan CPU-OS Simulator v7.5.50 dengan konfigurasi *dual-core*, instruction cache 8 KB, dan data cache 8 KB per core. prosedur pelaksanaan mencakup menjalankan tiga program simulasi dengan tingkat intensitas instruksi yang berbeda, mulai dari beban ringan (*few instructions per cycle*) hingga beban berat (*many instructions per cycle*). Untuk setiap skenario, dicatat waktu eksekusi dan jumlah *cache miss* yang terjadi, sehingga data tersebut dapat digunakan sebagai dasar untuk mengevaluasi kinerja sistem dan efisiensi manajemen memori. Visualisasi hasil pengukuran ini memungkinkan analisis perbandingan dampak beban instruksi terhadap performa CPU dan efektivitas algoritma penggantian halaman.



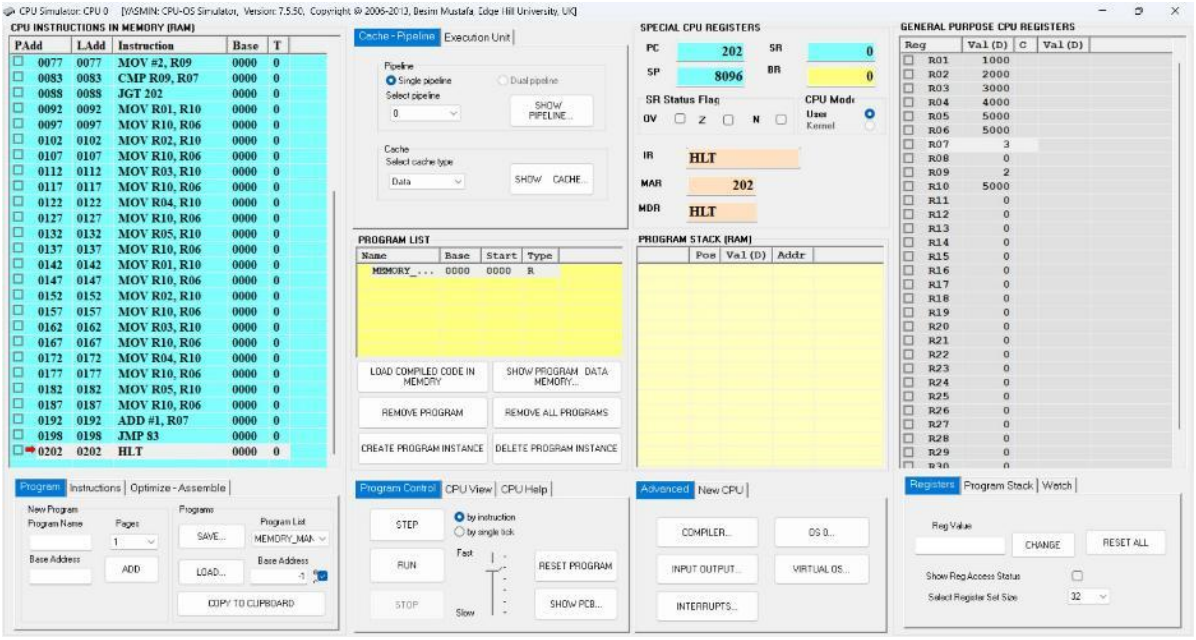
Gambar 1. Pipeline Cache Test

Prosedur pelaksanaan mencakup pengujian sistem dalam dua kondisi berbeda, yaitu dengan tingkat *switching* yang tinggi dan tanpa *switching* tinggi, guna memperoleh gambaran yang lebih jelas mengenai dampaknya terhadap waktu eksekusi total serta jumlah *page fault* yang terjadi selama proses simulasi berlangsung. Eksperimen kedua berfokus pada penjadwalan proses dengan konfigurasi yang terdiri atas lima proses aktif yang dijalankan secara bersamaan menggunakan algoritma *Round Robin* dengan *time quantum* sebesar 5 milidetik. *Round Robin* dipilih karena merupakan algoritma penjadwalan yang adil dan deterministik, di mana setiap proses mendapatkan jatah waktu CPU secara bergiliran, sehingga memudahkan analisis pengaruh *context switching* terhadap penggunaan memori dan efektivitas algoritma penggantian halaman. Prosedur pelaksanaan mencakup pengujian sistem dalam dua kondisi berbeda, yaitu dengan tingkat *switching* yang tinggi dan tanpa tingkat *switching* tinggi, untuk memperoleh gambaran dampaknya terhadap waktu eksekusi total serta jumlah page fault selama proses simulasi berlangsung.



Gambar 2. Process Scheduling FIFO

Pada eksperimen ketiga berfokus pada manajemen memori dengan konfigurasi sistem yang terdiri atas memori utama berkapasitas 2048 KB dan ukuran halaman sebesar 128 KB. Eksperimen ini menggunakan dua algoritma penggantian halaman, yaitu FIFO dan LRU, untuk dianalisis kinerjanya dalam kondisi yang berbeda. Prosedur pelaksanaan dimulai dengan inisialisasi enam proses yang masing-masing memiliki pola akses halaman berbeda, mencakup tingkat intensitas rendah, menengah, dan tinggi. Selama proses simulasi, dilakukan pencatatan terhadap jumlah *page fault*, total waktu eksekusi, serta *throughput* sistem sebagai parameter evaluasi kinerja. Selain itu, dilakukan variasi ukuran memori utama menjadi 1024 KB, 2048 KB, dan 4096 KB untuk mengetahui pengaruh kapasitas memori terhadap efisiensi kedua algoritma. Seluruh langkah tersebut diulang untuk masing-masing algoritma, guna memperoleh hasil perbandingan yang komprehensif dan terukur.



Gambar 3. Memory Management Test

HASIL DAN PEMBAHASAN

Hasil dari eksperimen pertama menunjukkan bahwa konfigurasi *cache* dengan kapasitas lebih besar memberikan dampak positif terhadap kinerja sistem. Sebagai contoh, ketika kapasitas *instruction cache* dan data *cache* ditingkatkan dari 4 KB menjadi 8 KB per *core* pada arsitektur *dual-core*, waktu akses rata-rata memori menurun sebesar sekitar 12%, sedangkan jumlah *cache* miss berkurang hingga 15% dibanding konfigurasi kecil. Temuan ini memperkuat hipotesis bahwa peningkatan kapasitas *cache* mampu mengurangi tingkat *cache miss*, sehingga beban akses terhadap memori utama dapat diminimalkan. Untuk lebih memaksimalkan performa, strategi seperti peningkatan ukuran *cache* secara proporsional dengan jumlah proses aktif, pengaturan *cache line* yang efisien, dan penggunaan algoritma penggantian halaman yang adaptif dapat digunakan untuk meminimalkan *cache miss*.

Kondisi ini meningkatkan efisiensi pemrosesan data dan menghasilkan waktu eksekusi yang lebih optimal pada sistem simulasi. Eksperimen dilakukan dalam konteks arsitektur *dual-core* dengan *pipeline* independen dan *cache* terintegrasi per *core*, di mana setiap core memiliki *instruction cache* 8 KB dan data *cache* 8 KB. *Pipeline* memungkinkan eksekusi instruksi secara paralel, sedangkan *cache per core* mengurangi konflik akses memori. Visualisasi arsitektur *dual-core* dapat

mempermudah pemahaman aliran instruksi dan data, serta dampak konfigurasi *cache* terhadap performa sistem.

Kemudian hasil dari eksperimen kedua pada skenario penjadwalan proses menunjukkan bahwa algoritma LRU memberikan kinerja yang lebih efisien dibandingkan algoritma FIFO, terutama pada kondisi sistem dengan frekuensi *context switching* yang tinggi. Berdasarkan hasil simulasi, waktu eksekusi total menggunakan LRU tercatat sekitar 18% lebih rendah dibandingkan FIFO.

Hal ini menunjukkan bahwa kemampuan algoritma LRU dalam menyesuaikan penggantian halaman berdasarkan tingkat kebaruan (*recency*) akses berkontribusi signifikan terhadap peningkatan performa sistem. Dengan demikian, dapat disimpulkan bahwa LRU lebih adaptif dalam menangani dinamika penggunaan memori oleh beberapa proses aktif, sehingga menghasilkan efisiensi eksekusi yang lebih baik pada lingkungan multitasking.

Pada hasil eksperimen ketiga menampilkan perbandingan tingkat *page fault* antara algoritma FIFO dan LRU berdasarkan variasi ukuran memori yang digunakan. Data hasil pengujian dirangkum dalam Tabel 1, yang memperlihatkan bahwa peningkatan kapasitas memori berpengaruh langsung terhadap penurunan jumlah *page fault* pada kedua algoritma. Namun demikian, algoritma LRU secara konsisten menunjukkan kinerja yang lebih baik dengan jumlah *page fault* yang lebih sedikit dibandingkan FIFO pada setiap konfigurasi memori. Selisih rata-rata efisiensi LRU terhadap FIFO berkisar antara 10% hingga 25,7%, yang menunjukkan bahwa mekanisme penggantian halaman berbasis *recency* pada LRU mampu mengoptimalkan pemanfaatan memori utama dan mengurangi frekuensi akses ke memori sekunder.

Table 1. Perbandingan Page Fault Rate antara FIFO dan LRU

Ukuran Memori (KB)	**Page Fault FIFO**	**Page Fault LRU**	**Selisih (%)**
1024	35	26	25,7 %
2048	22	18	18,1 %
4096	10	9	10,0 %

Hasil menunjukkan bahwa LRU memiliki *page fault rate* yang lebih rendah dalam semua konfigurasi ukuran memori, meskipun selisihnya semakin mengecil ketika memori bertambah besar.

SIMPULAN

Hasil eksperimen menunjukkan bahwa algoritma LRU lebih adaptif dibandingkan FIFO dalam menangani pola akses memori karena mempertimbangkan *recency* akses halaman. Hal ini tercermin dari *page fault rate* yang lebih rendah pada seluruh skenario uji. Berdasarkan Tabel 1, pada ukuran memori 1024 KB, LRU menghasilkan 26 page fault, lebih rendah dibanding FIFO dengan 35 page fault (selisih 25,7%). Tren serupa terlihat pada memori 2048 KB (selisih 18,1%) dan 4096 KB (selisih 10,0%).

Penurunan *page fault* tersebut berdampak langsung pada efisiensi waktu eksekusi, sehingga LRU menunjukkan performa lebih baik pada kondisi multitasking dengan aktivitas memori tinggi. Namun, FIFO tetap memiliki keunggulan dalam kesederhanaan implementasi dan *overhead* yang rendah, sehingga masih relevan untuk sistem dengan sumber daya terbatas.

Secara keseluruhan, hasil eksperimen membuktikan bahwa LRU lebih unggul dibanding FIFO dalam hal efisiensi manajemen memori, khususnya pada sistem dengan beban kerja tinggi, sementara FIFO lebih sesuai untuk lingkungan dengan keterbatasan komputasi.

UCAPAN TERIMA KASIH

Penulis menyampaikan apresiasi yang sebesar-besarnya kepada seluruh pihak yang telah memberikan dukungan selama proses penelitian dan penyusunan jurnal ini. Terima kasih kepada para pembimbing, rekan-rekan, serta semua individu yang telah memberikan bantuan berupa saran, masukan, maupun diskusi yang konstruktif sehingga penelitian mengenai perbandingan strategi manajemen memori FIFO dan LRU menggunakan CPU-OS Simulator v7.5.50 dapat terselesaikan dengan baik.

Penulis juga berterima kasih atas motivasi, dorongan, serta dukungan moral dari berbagai pihak yang tidak dapat disebutkan satu per satu. Semoga kontribusi yang diberikan menjadi amal kebaikan dan penelitian ini dapat memberikan manfaat bagi pengembangan pengetahuan di bidang sistem operasi.

DAFTAR PUSTAKA

Abbas, S. H., Naser, W. A., & Kadhim, L. M. (2022). Study and comparison of replacement algorithms. International Journal of Engineering Research and

Advanced Technology (IJERAT), 8(8), 1-6. https://doi.org/10.31695/IJERAT.2022.8.8.1

Bengar, A. (2025). Priority cache object replacement by using LRU, LFU and FIFO algorithms to improve cache memory hit ratio. Transactions on Soft Computing, 1(1), 1-13. https://doi.org/10.48314/tsc.v1i1.33

Demin, A., Dorn, Y., Katrutsa, A., Kazantsev, D., Latypov, I., Maximlyuk, Y., & Ponomaryov, D. (2024). EEvA: Fast expert-based algorithms for buffer page replacement. CoRR, abs/2405.00154.

Thakkar, B., & Padhy, R. N. (2024). Comparative analysis of page replacement algorithms in operating system. Journal of Emerging Technologies and Innovative Research, 11(5).

Titarenko, L., Kharchenko, V., Puidenko, V., Perepelitsyn, A., & Barkalov, A. (2024). Hardware-based implementation of algorithms for data replacement in cache memory of processor cores. Computers, 13(7), 166. https://doi.org/10.3390/computers13070166

Belady, L. A. (1966). A study of replacement algorithms for virtual-storage computers. IBM Systems Journal, 5(2), 78–101. https://doi.org/10.1147/sj.52.007

Denning, P. J. (1968). The working set model for program behavior. Communications of the ACM, 11(5), 323–333. https://doi.org/10.1145/363095.363141

Silberschatz, A., Ganapathi, A., & Sharma, V. (2018). Analysis of memory management techniques in modern operating systems. ACM Computing Surveys, 50(4), 1–36. https://doi.org/10.1145/3129288

Zhang, Y., Chen, X., & Li, K. (2021). An efficient page replacement algorithm based on access frequency and recency. Journal of Systems Architecture, 117, 102118. https://doi.org/10.1016/j.sysarc.2021.102118

Alsaedi, A., & Awan, I. (2019). Performance evaluation of page replacement algorithms in virtual memory systems. International Journal of Computer Applications, 181(44), 14–20. https://doi.org/10.5120/ijca2019919448