UNIVERSIDADE FEDERAL DE MINAS GERAIS

Instituto de Ciências Exatas

Graduação em Sistemas de Informação

Gabriel Oliveira Sant'Ana

Mariana Assis Ramos

Trabalho Prático #2 – Memória Virtual

Belo Horizonte

2024

1. Introdução

A memória virtual é uma técnica de gerenciamento de memória que permite a um computador utilizar mais memória do que a fisicamente disponível, combinando hardware e software para simular memória adicional através do uso de espaço em disco. Isso possibilita que múltiplos programas rodem simultaneamente sem esgotar a memória RAM.

Neste trabalho prático, foi requisitado implementar um simulador de memória virtual, replicando as estruturas de um sistema de gerência de memória virtual. Para atender a essa demanda, foi essencial compreender o funcionamento da paginação na memória virtual. Criamos uma estrutura para simular a memória física e uma tabela de páginas para representar a memória virtual. Buscando entender a relação entre consumo de memória e desempenho, a tabela foi implementada usando quatro estruturas diferentes: uma tabela de páginas densa (sem hierarquia), tabelas hierárquicas com dois e três níveis, e uma tabela invertida. Além disso, abordamos algoritmos de substituição de páginas para casos de memória cheia, implementando LFU, FIFO, Random e 2ª Chance.

2. O projeto

O principal ponto deste trabalho é simular a memória virtual e, para isso, precisamos simular a tabela que mapeia os endereços virtuais para os físicos. O fluxo geral do código segue o mesmo independente da estrutura, no qual recebemos de entrada uma sequência de endereços virtuais acessados, onde em cada linha temos o endereço, o tipo de acesso (leitura ou escrita), e o tamanho da memória e da página. Inicializamos a memória e a tabela de páginas com os valores passados, tendo em mente que o tamanho da tabela de páginas é a quantidade de páginas que couber dentro dela. Em seguida, vamos lendo cada um desses endereços recebidos e fazendo o passo a passo.

Verificamos usando a tabela de páginas a validade do endereço. Caso o endereço seja válido, ou seja, exista na memória, é necessário atualizar as variáveis relativas a esse endereço, seja ele do tipo leitura ou escrita. Por exemplo, o ref usado no secondChance é alterado para 1. Em caso de LRU, também é necessário atualizar a lista auxiliar. Caso o endereço não seja válido, existem dois caminhos: se

existir espaço na memória, adicionamos o endereço nela; caso a memória esteja cheia, usamos um algoritmo de substituição que retira algum dos endereços da memória, desvalidando-o e adicionando o novo endereço. Por fim, atualizamos as estatísticas da página também.

3. Estruturas

Para simular a memória virtual, além da estrutura para a tradução de endereços (a **tabela de páginas**), foi necessária a criação de uma estrutura para simular a **memória física**. Optamos por não criar uma estrutura específica para o disco, simulando apenas a busca dos endereços nele, já que nele não ocorre manipulação direta. Assim, criamos as estruturas necessárias para o uso da memória virtual. Além disso, desenvolvemos uma estrutura de **fila duplamente encadeada** para o algoritmo de substituição Least Recently Used (**LRU**).

3.1. Estruturas Básicas

As estruturas mencionadas são formadas utilizando outras estruturas como base. Vamos detalhar cada uma dessas estruturas básicas a seguir:

3.1.1 - Page

Cada **tabela de páginas** é composta por **diversas páginas agregadas** de alguma forma, seja em formato de array, matriz ou outra configuração. Uma página é uma unidade de armazenamento fixa, tanto na memória física (RAM) quanto na memória virtual.

- **Page**

Esta estrutura é a base para as tabelas de páginas densas e hierárquicas, seja de dois ou três níveis. É nela que o sistema operacional mapeia diretamente os quadros (frames) da memória física. Nossa página é formada por três inteiros:

- ***changed***: indicador utilizado para identificar as “**dirty pages**” (páginas sujas).
- ***valid***: indica se o endereço está mapeado na memória
- ***addressInMemory***: corresponde ao endereço do quadro(frame) na memória

Optamos por não adicionar o **endereço virtual,** pois este será o **índice** da página na tabela de páginas.

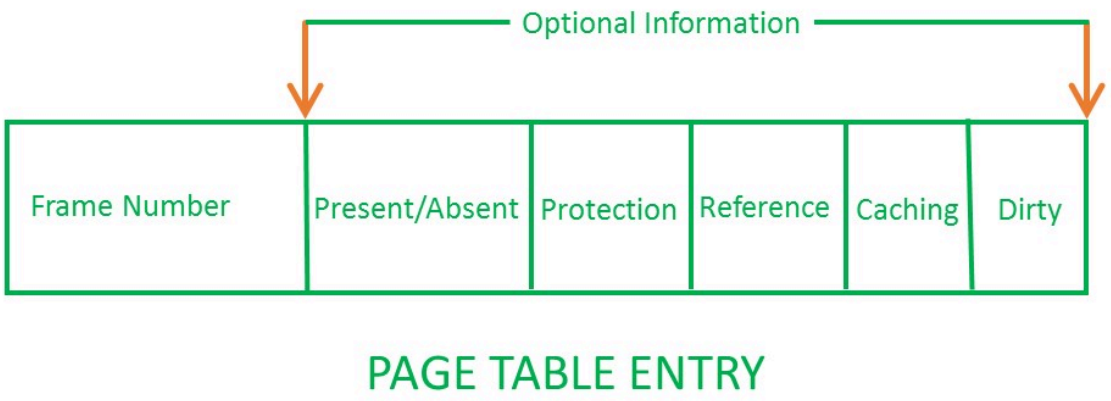


figura 1: Representação de uma página padrão

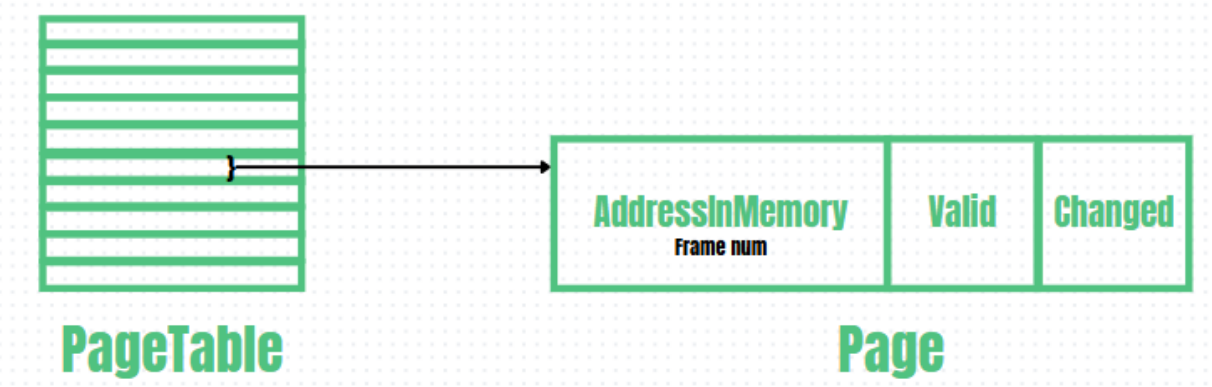


figura 2: Implementação de página utilizada

- **Inverted Page**

Na tabela de páginas invertida, é necessário utilizar uma estrutura de página diferente. A principal característica dessa página é ter o endereço virtual como um dos atributos, pois, como o **tamanho da tabela** é baseado na **memória física** (RAM), não podemos usar os índices dela para indicar o endereço virtual. Assim, salvamos os endereços virtuais diretamente na página. Esta página contém:

- ***addressVirtual***: o endereço virtual na memória virtual.
- ***changed***: indicador utilizado para identificar as "**dirty pages**" (páginas sujas).
- ***valid***: indica se o endereço está mapeado na memória.
- ***addressInMemory***: corresponde ao endereço do quadro (frame) na memória

3.1.2 - Frame

A **memória física** (RAM) também é dividida em **blocos** de tamanho fixo chamados **frames**, cujo tamanho corresponde ao das páginas na memória virtual. No nosso simulador, a estrutura dos frames é uniforme, independentemente do tipo de tabela de páginas empregada. Os frames são projetados com os seguintes atributos:

- ***addressInTable***: Indica o endereço do dado na tabela de páginas.
- ***changed***: Indica se o bloco de memória foi escrito pelo menos uma vez (usado para testes).
- ***lastAccess***: Armazena o momento do último acesso a este bloco de memória.
- ***ref***: Utilizado no algoritmo de substituição de páginas "Second Chance" (detalhado na *seção "4.3.4 Second Chance"*).

3.1.3 - Auxiliares

Além das estruturas principais, foi necessário desenvolver uma **lista duplamente encadeada** para implementar o algoritmo de substituição Least Recently Used (LRU), descrito em detalhe na seção *"4.3.3 Least Recently Used"*. A lista duplamente encadeada é composta por duas estruturas fundamentais:

- **Node**: Cada nó na lista contém:
- Dois ponteiros: um apontando para o próximo nó e outro para o nó anterior, permitindo a navegação bidirecional na lista.
- Um identificador (id): Representa o valor ou a referência armazenada no nó, associando-o ao frame ou à página relevante.
- **List**: Esta estrutura representa a lista encadeada como um todo, e inclui:
- Um inteiro (size) que indica o tamanho atual da lista.
- Dois ponteiros: um para o primeiro nó (head) e outro para o último nó (tail), facilitando inserções e remoções eficientes em ambas as extremidades da lista.

3.2. Tabela de páginas

Agora que temos como base o que são as páginas e as demais estruturas, podemos explicar as diferenças entre as implementações das tabelas.

3.2.1 - Densa

A tabela de páginas densa é a estrutura mais simples de gerenciamento de memória virtual. Nesta implementação, a tabela de páginas é um **array linear** onde cada índice corresponde a uma página virtual e o valor armazenado é o endereço do frame na memória física. O tamanho da tabela é fixo e determinado pelo número de páginas virtuais possíveis.

- **Implementação**:

Para implementar essa tabela, definimos um array de page chamado pageTable e um inteiro pageTableSize que indica o tamanho total da tabela. Cada entrada do array representa uma página virtual na estrutura page.

3.2.2 - Hierárquica de 2 níveis

A tabela de páginas hierárquica de 2 níveis organiza a memória virtual em duas camadas de tabelas de páginas. O endereço virtual é dividido em duas partes: bits de nível superior e bits de nível inferior. A tabela de nível superior contém ponteiros para tabelas de nível inferior, que por sua vez contêm os endereços dos frames na memória física.

- **Implementação**:

Para implementar esta estrutura, utilizamos um array de ponteiros para page, denominado level2PageTable. Os bits do endereço virtual são divididos entre firstLevelBits e secondLevelBits, que indicam a quantidade de bits alocados para os índices das tabelas de primeiro e segundo nível, respectivamente. Isso permite um mapeamento mais eficiente e reduzido da memória virtual, comparado à tabela densa.

3.2.3 - Hierárquica de 3 níveis

A tabela de páginas hierárquica de 3 níveis é uma extensão da abordagem de 2 níveis, adicionando uma camada extra. O endereço virtual é dividido em três partes: bits de nível superior, intermediário, e inferior. Esta estrutura é útil para sistemas com grandes espaços de endereçamento virtual.

- **Implementação**:

Nesta implementação, level3PageTable é um array de ponteiros para arrays de ponteiros para page. Os bits do endereço virtual são distribuídos entre firstLevelBits, secondLevelBits, e thirdLevelBits, com numBitsEndereco representando o total de bits no endereço virtual. Essa divisão permite uma organização ainda mais granular da memória virtual, facilitando o gerenciamento de grandes espaços de endereçamento.

3.2.4 - Invertida

A tabela de páginas invertida é uma abordagem alternativa onde a tabela de páginas é organizada com base nos frames de memória física, em vez de páginas virtuais. Cada entrada na tabela de páginas invertida armazena o endereço virtual correspondente ao frame de memória física.

- **Implementação**:

Para implementar a tabela invertida, utilizamos um array de page_IPT chamado invertedPageTable. Cada entrada da tabela inclui addressInMemory (endereço físico), addressVirtual (endereço virtual correspondente), valid, e changed. Essa estrutura reduz a quantidade de memória necessária para armazenar a tabela de páginas, pois o tamanho da tabela é proporcional à memória física disponível, e não ao espaço de endereçamento virtual.

4. Algoritmos de substituição

Nesta seção, detalhamos os algoritmos de substituição de página implementados no simulador. Esses algoritmos são fundamentais para decidir qual página na memória será substituída quando uma nova página precisa ser carregada e a memória está cheia.

4.1 - First In, First Out(FIFO)

O algoritmo FIFO é um dos métodos mais diretos de substituição de páginas. Ele substitui a página que está na memória há mais tempo, seguindo a lógica de que a primeira página a entrar será a primeira a sair. Quando uma nova página é carregada na memória, ela é adicionada ao final da fila. Para realizar a substituição,

a página na primeira posição da fila é removida, e a nova página é adicionada ao final, mantendo a ordem de chegada. Embora a implementação seja simples e fácil de entender, o algoritmo FIFO não considera o uso recente das páginas, o que pode levar à substituição de páginas que ainda são frequentemente utilizadas.

4.2 - Random

O algoritmo Random escolhe uma página aleatoriamente para ser substituída quando uma nova página precisa ser carregada na memória. Ele utiliza a função rand() para selecionar uma posição aleatória e então realiza a substituição diretamente, sem a necessidade de ajustar outros endereços. A principal vantagem desse algoritmo é sua simplicidade e rapidez na execução. No entanto, a substituição aleatória pode resultar em uma performance inconsistente, já que páginas frequentemente usadas podem ser removidas.

4.3 - Least Recently Used(LRU)

O algoritmo Least Recently Used (LRU) parte do princípio de que as páginas usadas mais recentemente provavelmente serão mais utilizadas novamente. Dessa maneira, quando ocorre um page fault o algoritmo elimina a página que está há mais tempo sem ser referenciada. Para implementar o LRU, foi criada uma estrutura chamada indexList, que armazena a ordem de referência de cada página na memória. Essa estrutura é uma lista encadeada onde cada nó armazena o índice da página na memória. Sempre que uma página é referenciada, ela é movida para o fim da lista.

Dessa forma, a página no início da lista é a que está há mais tempo sem ser referenciada. Quando uma substituição é necessária, o algoritmo então seleciona a página no início da lista (ou seja, a que está há mais tempo sem ser usada), substitui essa página na memória e atualiza a tabela de páginas e a indexList. Essa abordagem garante que as páginas recentemente usadas permaneçam na memória, aumentando a eficiência do sistema.

4.4 - Second Chance

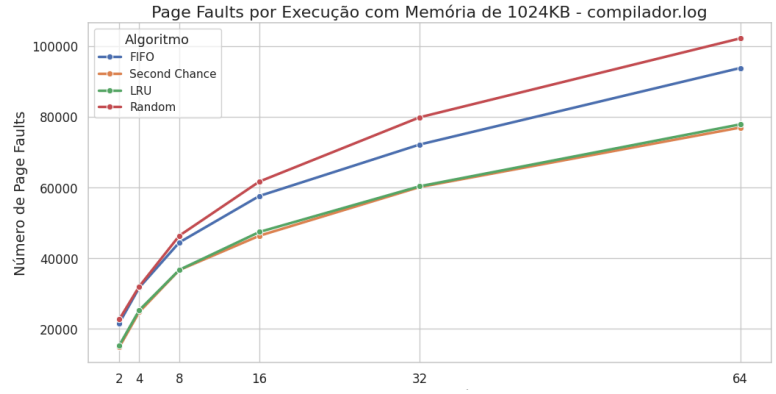
O algoritmo Second Chance é uma modificação do FIFO que melhora seu desempenho ao dar uma "segunda chance" às páginas antes de serem substituídas. Cada página possui um bit de referência que é verificado antes da remoção. Para realizar este processo, foi criada uma lista circular auxiliar que gerencia os bits de referência das páginas. Quando é necessário realizar uma substituição, a lista circular é percorrida e, se uma página com o bit de referência igual a um for encontrada, seu bit é alterado para zero, e a busca continua. Quando uma página com o bit de referência igual a zero é encontrada, ela é escolhida para substituição. A nova página é ativada na tabela de páginas, marcando seu endereço como válido e atualizando seu índice na memória. A lista circular é então rotacionada para apontar para a próxima página, assegurando uma gestão eficiente dos bits de referência. Dessa forma, o algoritmo Second Chance combina a simplicidade do FIFO com um mecanismo adicional que melhora a eficiência ao evitar a remoção de páginas que ainda podem ser úteis.

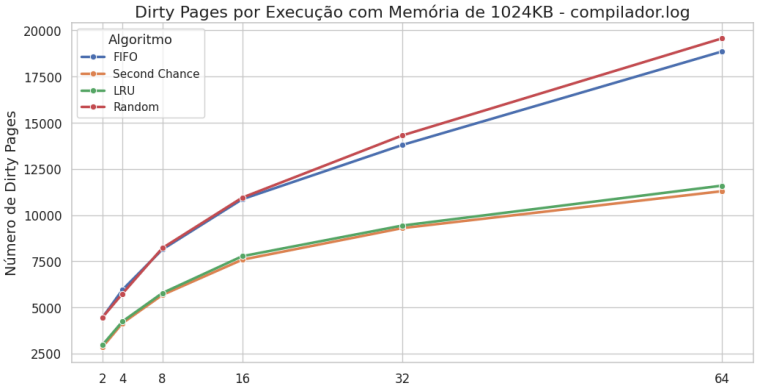
5. Testes

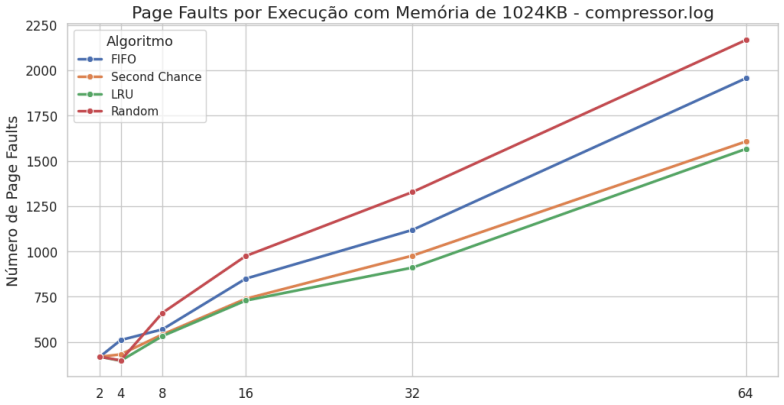
5.1. Comparação dos Algoritmos

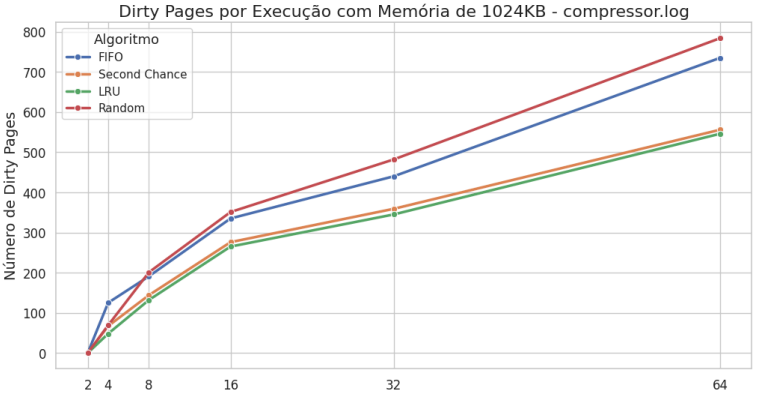
Nesta seção, analisamos o desempenho dos diferentes algoritmos de substituição de páginas implementados no simulador de memória virtual. Os algoritmos comparados são FIFO (First In, First Out), Second Chance, LRU (Least Recently Used) e Random. A análise é feita considerando dois principais indicadores de desempenho: page faults e dirty pages. As métricas são avaliadas em diferentes tamanhos de páginas (variando de 2 a 64) e de memória (variando de 128 a 16384), utilizando os logs de execução dos programas "compilador", "compressor", "matriz", e "simulador".

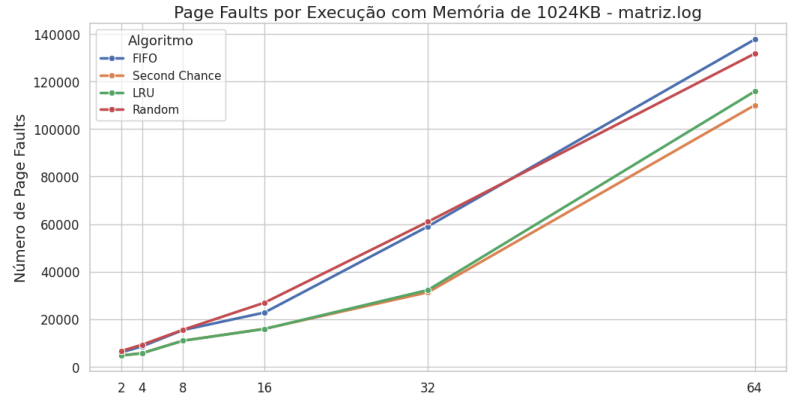
- **Análise variando o tamanho da página**

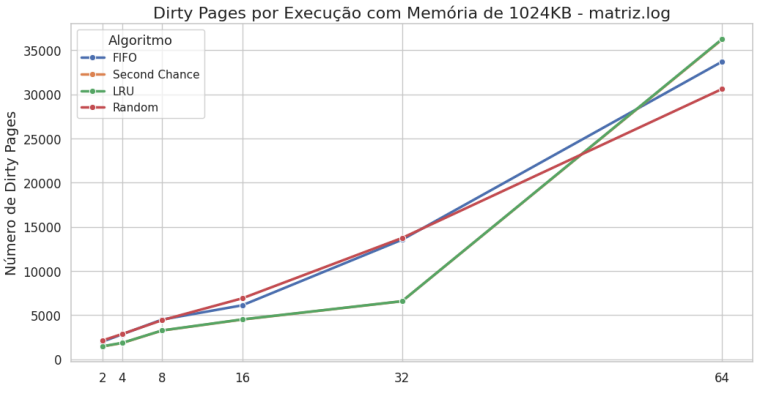


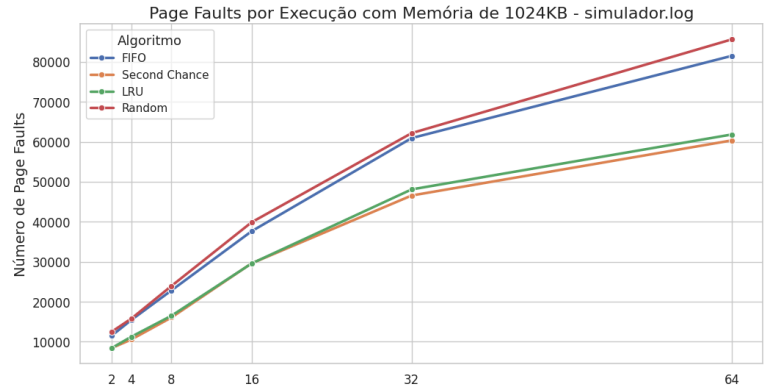












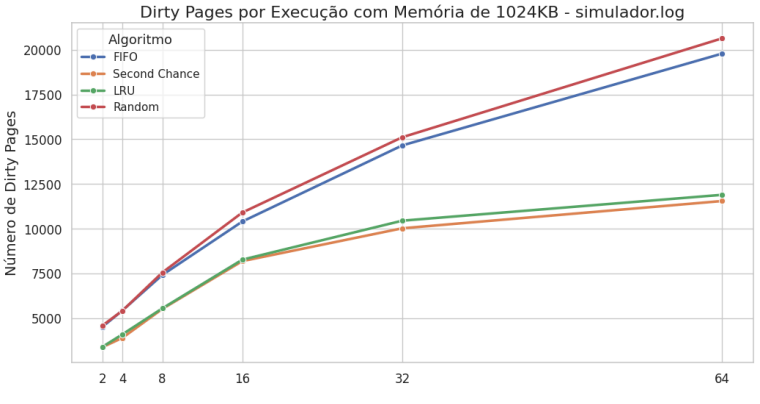


figura 3: gráficos com Page Fault/Dirty Pages dado o tamanho da Page

Memória fixa 1024KB - "compilador.log"								
tamanho da página(KB)	Algoritmo							
	FIFO		Second Chance		Least Recently Used - LRU		Random	
	Dirty Pages	Page Faults	Dirty Pages	Page Faults	Dirty Pages	Page Faults	Dirty Pages	Page Faults
2	4442	21639	2841	14834	2957	15330	4454	22683
4	5945	31698	4141	24802	4231	25308	5736	32032
8	8135	44465	5671	36654	5775	36707	8211	46366
16	10858	57600	7582	46356	7770	47424	10945	61682
32	13794	72140	9291	60121	9429	60336	14310	79838
64	18860	93778	11292	76935	11593	77837	19569	102153

Memória fixa 1024KB - "compressor.log"								
tamanho da página(KB)	Algoritmo							
	FIFO		Second Chance		Least Recently Used - LRU		Random	
	Dirty Pages	Page Faults	Dirty Pages	Page Faults	Dirty Pages	Page Faults	Dirty Pages	Page Faults
2	0	419	0	419	0	419	0	419
4	125	511	67	432	48	397	69	399
8	191	570	144	542	132	533	201	661
16	335	850	276	739	265	729	351	975
32	440	1119	359	977	345	911	482	1328
64	735	1956	556	1607	546	1566	784	2167

Memória fixa 1024KB - "matriz.log"								
tamanho da página(KB)	Algoritmo							
	FIFO		Second Chance		Least Recently Used - LRU		Random	
	Dirty Pages	Page Faults	Dirty Pages	Page Faults	Dirty Pages	Page Faults	Dirty Pages	Page Faults
2	1993	6097	1450	4659	1473	4744	2125	6647
4	2836	8551	1846	5729	1866	5673	2870	9260
8	4505	15410	3262	10961	3273	10928	4437	15523
16	6114	22815	4500	15965	4517	15904	6904	26967
32	13543	59061	6575	31323	6582	32282	13739	61009
64	33692	137751	36196	110074	36242	115886	30597	131764

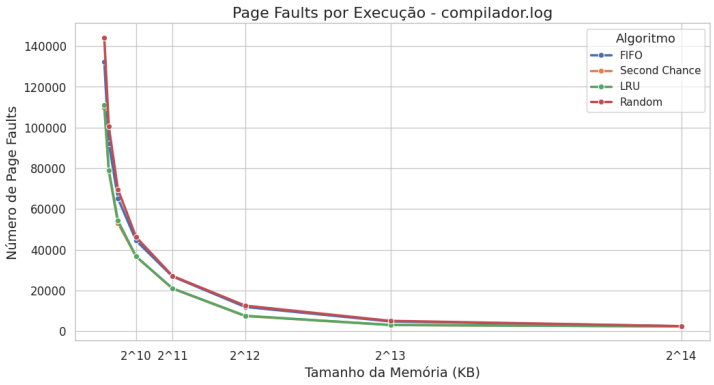
Memória fixa 1024KB - "simulador.log"								
tamanho da página(KB)	Algoritmo							
	FIFO		Second Chance		Least Recently Used - LRU		Random	
	Dirty Pages	Page Faults	Dirty Pages	Page Faults	Dirty Pages	Page Faults	Dirty Pages	Page Faults
2	4510	11449	3374	8356	3395	8364	4585	12433
4	5426	15440	3896	10550	4092	11240	5414	15779
8	7413	22816	5514	16043	5546	16479	7550	23931
16	10399	37606	8186	29578	8267	29547	10902	39849
32	14653	60922	10021	46557	10446	48088	15110	62165
64	19786	81506	11545	60347	11892	61829	20638	85620

figura 4: tabelas com diferença de Page Fault e Dirty Pages para diferentes algoritmos

Primeiramente, foram analisados os algoritmos a partir de uma variação no tamanho das páginas. Observamos que o algoritmo FIFO tem um desempenho semelhante ao algoritmo Random, ambos apresentando um número elevado de page faults. Isso ocorre porque o FIFO substitui a página mais antiga sem considerar se ela será necessária novamente em breve, o que frequentemente leva à remoção de páginas que serão acessadas em um futuro próximo. O algoritmo Random, por sua natureza aleatória, não tem uma estratégia para manter as páginas que são mais prováveis de serem reutilizadas, resultando em um número comparável de page faults.

Por outro lado, o algoritmo Second Chance mostra um desempenho semelhante ao LRU. O Second Chance é uma melhoria do FIFO, onde as páginas recebem uma segunda chance antes de serem substituídas, o que ajuda a reter páginas que foram acessadas recentemente. O LRU, por sua vez, substitui a página que não foi usada há mais tempo, garantindo que as páginas mais recentemente acessadas permaneçam na memória. Como resultado, tanto o Second Chance quanto o LRU apresentam um número menor de page faults em comparação com FIFO e Random, com o LRU geralmente tendo o menor número de page faults devido à sua abordagem mais precisa na retenção das páginas necessárias.

- **Análise variando o tamanho da memória**



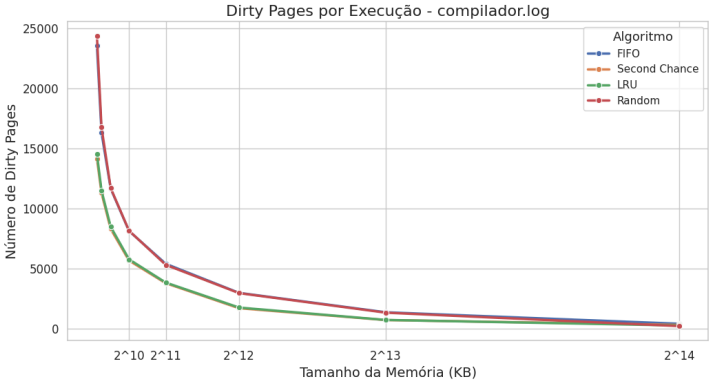


figura 5: gráficos com Page Fault/Dirty Pages dado o tamanho da Memória

Página fixa 8KB - "compilador.log"								
tamanho da memória(KB)	Algoritmo							
	FIFO		Second Chance		Least Recently Used - LRU		Random	
	Dirty Pages	Page Faults	Dirty Pages	Page Faults	Dirty Pages	Page Faults	Dirty Pages	Page Faults
128	23565	132386	14173	109863	14522	110909	24371	144007
256	16308	92279	11300	78449	11502	79010	16769	100470
512	11665	65270	8307	53104	8483	54286	11706	69677
1024	8135	44465	5671	36654	5775	36707	8156	46247
2048	5404	26928	3788	20987	3839	21091	5315	27178
4096	2979	11951	1701	7469	1756	7632	2957	12605
8192	1362	4825	714	3087	722	3107	1330	5123
16384	406	2494	312	2406	263	2391	210	2529

Página fixa 8KB - "compressor.log"								
tamanho da memória(KB)	Algoritmo							
	FIFO		Second Chance		Least Recently Used - LRU		Random	
	Dirty Pages	Page Faults	Dirty Pages	Page Faults	Dirty Pages	Page Faults	Dirty Pages	Page Faults
128	1126	3235	927	2869	897	2757	1199	3706
256	639	1804	534	1586	540	1558	681	2055
512	382	1030	313	920	316	934	412	1197
1024	191	570	144	542	132	533	201	661
2048	0	255	0	255	0	255	0	255
4096	0	255	0	255	0	255	0	255
8192	0	255	0	255	0	255	0	255
16384	0	255	0	255	0	255	0	255

Página fixa 8KB - "matriz.log"								
tamanho da memória(KB)	Algoritmo							
	FIFO		Second Chance		Least Recently Used - LRU		Random	
	Dirty Pages	Page Faults	Dirty Pages	Page Faults	Dirty Pages	Page Faults	Dirty Pages	Page Faults
128	47010	211346	45305	157937	46114	169540	39649	191914
256	17371	77190	8863	43407	9161	44904	17381	79045
512	6913	27104	5146	18933	5149	19141	7896	31949
1024	4505	15410	3262	10961	3273	10928	4437	15523
2048	2281	6672	1559	4710	1623	4745	2521	7742
4096	1359	3840	1004	3014	985	2950	1503	4333
8192	804	2778	636	2441	608	2398	834	2928
16384	138	2326	64	2225	74	2214	85	2238

Página fixa 8KB - "simulador.log"								
tamanho da memória(KB)	Algoritmo							
	FIFO		Second Chance		Least Recently Used - LRU		Random	
	Dirty Pages	Page Faults	Dirty Pages	Page Faults	Dirty Pages	Page Faults	Dirty Pages	Page Faults
128	30140	131827	18053	98135	18629	100880	30934	136696
256	17772	78338	12391	59363	12910	61335	18353	80917
512	11143	42421	8807	33915	8922	33827	11730	45065
1024	7413	22816	5514	16043	5546	16479	7550	23931
2048	4655	11884	3230	8144	3395	8556	4744	12482
4096	2829	6293	2095	4804	2094	4732	2961	7019
8192	2022	4528	1583	3815	1597	3784	2040	4844
16384	1169	3669	997	3418	997	3405	997	3756

figura 6: tabelas com diferença de Page Fault e Dirty Pages para diferentes algoritmos

Em uma segunda análise, considerando a variação da quantidade de memória, verificamos que o número de page faults diminui significativamente à medida que o tamanho da memória aumenta. O algoritmo LRU (Least Recently Used) se destaca ao gerar a menor quantidade de page faults, especialmente em tamanhos de memória menores. Isso ocorre porque o LRU substitui a página que não foi utilizada por mais tempo, fazendo um uso mais eficiente da localidade temporal. Conforme a memória disponível aumenta, a diferença de desempenho entre os algoritmos diminui, uma vez que menos substituições são necessárias, tornando a vantagem dos algoritmos mais sofisticados menos pronunciada.

Entretanto, é possível observar que os algoritmos FIFO (First In, First Out) e Random geram mais page faults em tamanhos menores de memória em comparação com os algoritmos de Second Chance e LRU. O FIFO, por exemplo, substitui sempre a página mais antiga, independentemente de quão frequentemente essa página é acessada, o que pode levar a remoções ineficientes. O algoritmo Random, que escolhe aleatoriamente a página a ser substituída, também apresenta um desempenho inferior devido à falta de qualquer consideração sobre a frequência de uso das páginas.

5.2. Comparação das Estruturas

É importante destacar como diferentes estruturas de tabela de páginas afetam o desempenho do sistema. As estruturas de tabela de páginas são fundamentais para a gestão da memória em sistemas operacionais, e cada uma tem suas próprias características que influenciam diretamente na eficiência do acesso à memória. Para avaliar o desempenho das diferentes estruturas de tabelas de páginas, realizamos diversos testes variando o tamanho das páginas e a quantidade de memória.

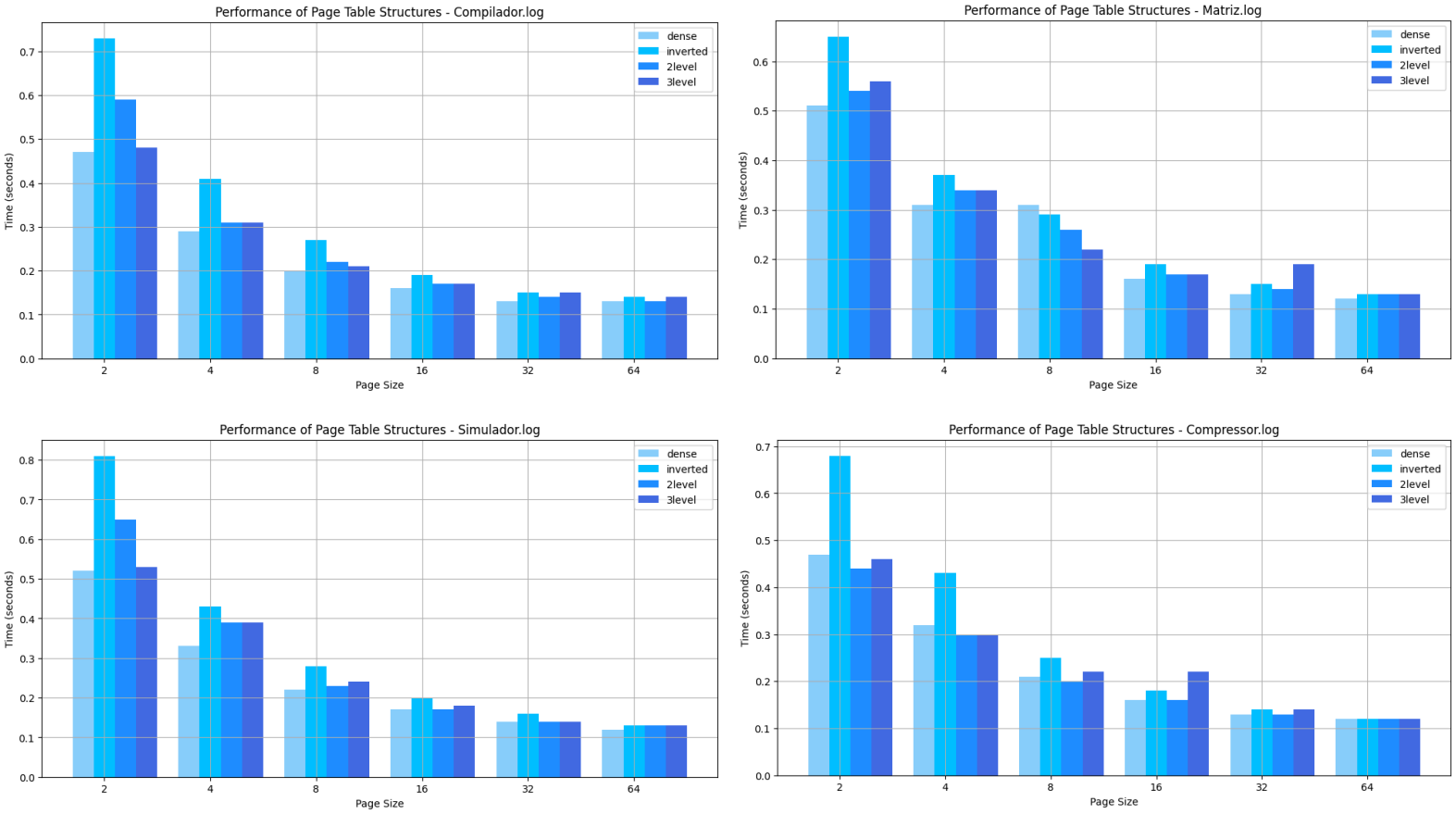
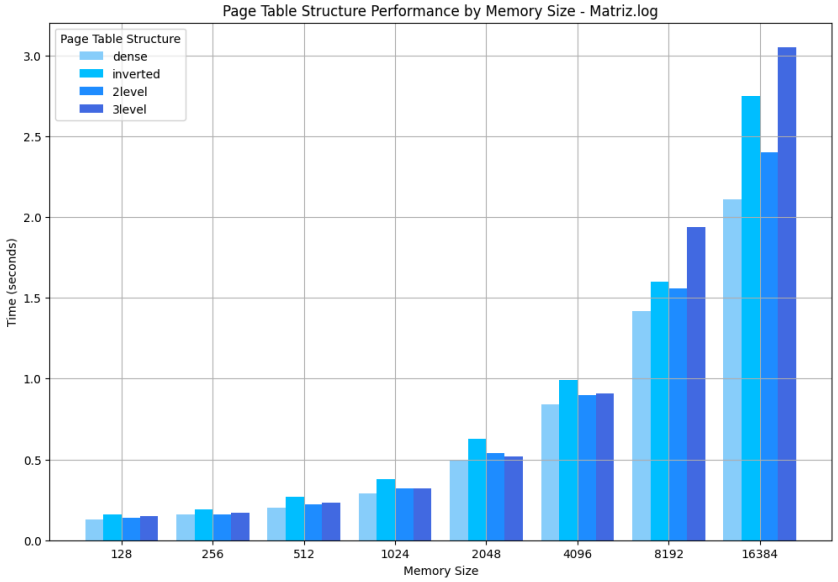


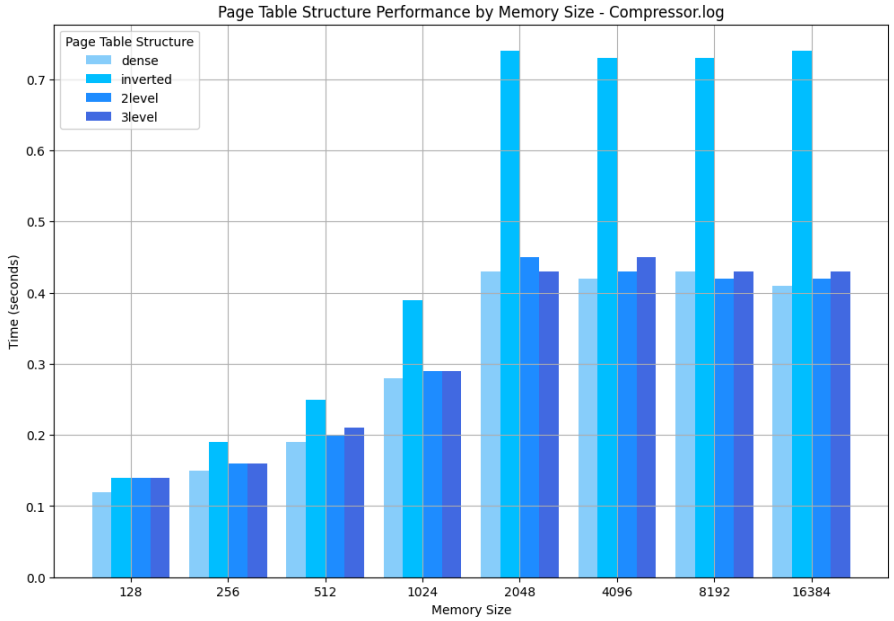
figura 7: Gráfico com as performances das diferentes Estruturas dado o tamanho da Page

Nessa primeira análise, foi observado o tempo de execução das quatro estruturas de tabelas de páginas (densa, invertida, 2 níveis e 3 níveis) enquanto o tamanho da página varia de 2 a 64. é evidente que a tabela de páginas invertida apresenta um tempo de execução significativamente maior para tamanhos de página menores (2 e 4). A tabela invertida requer uma busca mais complexa, já que, em vez de um acesso direto, é necessário pesquisar através da tabela para encontrar o

endereço virtual correspondente a um endereço físico. Com tamanhos de página menores, o número de páginas aumenta, exacerbando essa desvantagem e resultando em tempos de execução mais longos.

Em contraste, as tabelas hierárquicas (2 níveis e 3 níveis) e a tabela densa mostram uma redução considerável no tempo de execução à medida que o tamanho da página aumenta. Isso ocorre porque um aumento no tamanho da página diminui o número total de páginas, reduzindo a sobrecarga de gerenciamento de memória. A tabela densa, sendo a mais simples, se beneficia mais claramente dessa redução, mas as tabelas hierárquicas também demonstram bons resultados devido à sua eficiência na organização e busca de endereços em múltiplos níveis.





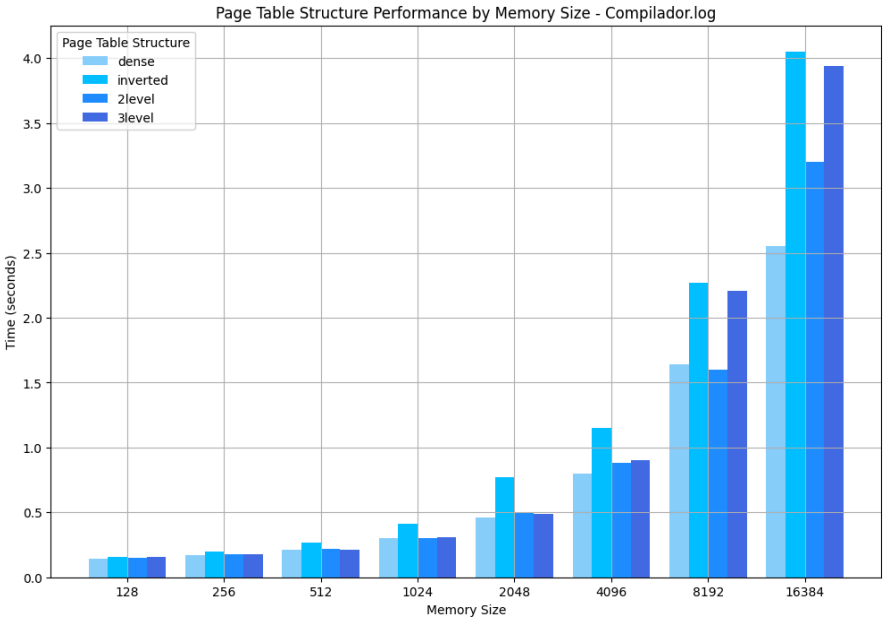


figura 8: Gráfico com as performances das diferentes Estruturas dado o tamanho da Memória

Em segunda análise, que avalia o desempenho das estruturas com diferentes tamanhos de memória, a tendência observada é semelhante. À medida que o tamanho da memória aumenta, todas as estruturas de tabelas de páginas apresentam um aumento no tempo de execução, mas em ritmos diferentes. A tabela de páginas densa, por exemplo, tem um crescimento mais moderado no tempo de execução, mantendo-se relativamente eficiente mesmo com grandes tamanhos de memória. Em contraste, a tabela de páginas invertida apresenta um crescimento exponencial no tempo de execução com o aumento do tamanho da memória. Isso se deve ao fato de que a tabela invertida é proporcional ao número de frames na memória física, e cada operação de busca se torna mais custosa conforme a memória aumenta. As tabelas hierárquicas de dois e três níveis apresentam um desempenho intermediário, beneficiando-se da estrutura hierárquica que divide o espaço de endereçamento, mas ainda assim mostrando um aumento no tempo de execução à medida que a memória cresce.

- **Análise das páginas sujas**

Além da análise dos page faults, avaliamos também o comportamento dos algoritmos em relação às dirty pages. Observamos que os algoritmos Second Chance e LRU se destacaram por minimizar a quantidade de dirty pages, fazendo

bom uso da localidade temporal. O algoritmo LRU, em particular, mostrou-se altamente eficiente ao garantir que as páginas mais recentemente modificadas permanecessem na memória, resultando em menos páginas sujas que precisam ser escritas de volta ao disco. O Second Chance, ao dar uma segunda oportunidade às páginas antes de substituí-las, também conseguiu reduzir significativamente a quantidade de dirty pages. Em contraste, os algoritmos FIFO e Random apresentaram um desempenho inferior. O FIFO, por substituir a página mais antiga sem considerar seu estado de modificação, frequentemente removia páginas sujas, aumentando o overhead de escrita no disco. O algoritmo Random, devido à sua natureza aleatória, teve um desempenho inconsistente, frequentemente resultando em um número elevado de dirty pages. Essa diferença de desempenho é mais pronunciada em ambientes com memória limitada, onde a eficiência na gestão de dirty pages é crucial para o desempenho geral do sistema.

6. Conclusões

A implementação do simulador de memória virtual com diferentes estruturas de tabela de páginas e algoritmos de substituição proporcionou uma compreensão aprofundada sobre a complexidade e a eficiência das diversas abordagens de gerenciamento de memória.

Os testes mostram que cada estrutura de tabela de páginas tem suas particularidades e impacta significativamente o desempenho do sistema. A tabela de páginas densa, apesar de sua simplicidade, mostrou-se eficiente para tamanhos de páginas maiores devido ao acesso direto aos endereços. As tabelas hierárquicas, tanto de 2 níveis quanto de 3 níveis, ofereceram um bom equilíbrio entre complexidade e desempenho, especialmente em sistemas com grandes espaços de endereçamento. Por outro lado, a tabela de páginas invertida, embora eficiente em termos de uso de memória física, apresentou um desempenho inferior em termos de tempo de execução, especialmente para tamanhos de páginas menores.

Os algoritmos de substituição também demonstraram diferenças significativas em suas eficiências. O algoritmo FIFO, apesar de simples, resultou em um elevado número de page faults, similar ao algoritmo Random, que, devido à sua aleatoriedade, não conseguiu otimizar o uso da memória. O algoritmo Least Recently Used (LRU) destacou-se por sua capacidade de minimizar page faults, ao reter páginas recentemente acessadas. O algoritmo Second Chance, uma melhoria do FIFO, mostrou-se uma solução intermediária eficaz, oferecendo um bom equilíbrio entre simplicidade e desempenho.

Através dos testes realizados, ficou claro que a escolha da estrutura de tabela de páginas e do algoritmo de substituição impacta diretamente na eficiência do gerenciamento de memória. Sistemas com grandes espaços de endereçamento e exigências de alta performance beneficiam-se de estruturas hierárquicas e algoritmos como o LRU, enquanto sistemas com restrições de recursos podem se beneficiar da simplicidade das tabelas densas e do algoritmo Second Chance.

Em suma, este trabalho evidenciou a importância de uma escolha criteriosa das técnicas de gerenciamento de memória, considerando as necessidades específicas do sistema e os recursos disponíveis. A implementação prática e a análise comparativa forneceram insights valiosos que podem ser aplicados no desenvolvimento de sistemas operacionais mais eficientes e robustos.

7. Instruções para uso

Após abrir o zip:

- direcione para o diretório principal “**UFMG-SO-virtual-memory-main**” onde se encontra o arquivo “***tp2virtual.c***”
- rode o código com: **make clean && make**
- para um teste pronto
    - **make test**
- para testar entradas específicas os argumentos são:

```
tp2virtual lru files/arquivo.log 4 128 densa
```

- respectivamente:

```
[exe] [algoritmo] [entradas] [page size] [memory size] [struct*]
```

- onde:
    - exe: o executável gerado com o Makefile
    - algoritmo: o algoritmo de substituição usado
        - fifo, lru, secondChance, random
    - entradas: arquivo de simulação de acesso na memória dentro da pasta ***“/files”***
        - compilador.log, compressor.log, matriz.log, simulador.log
    - page size: tamanho da página, em potência de 2, variando de 2 até 64 KB
    - memory size: tamanho da memória, em potência de 2, variando de 128 a 16384 KB
    - struct [OPCIONAL]: estrutura da tabela escolhida
        - dense, 2level, 3level, inverted

8. Referências

- https://www.geeksforgeeks.org/second-chance-or-clock-page-replacement-policy/
- https://www.javatpoint.com/os-page-table
- Aula 21 - Memória Virtual - YouTube