Simulação de Memória Virtual

Sistemas Operacionais
Trabalho Prático 2

Bernardo Dutra Lemos - 2022043949
Raphael A. Carreiro Mendes - 2022043809

20 de dezembro de 2024

1 Introdução

O objetivo do presente relatório é mostrar a implementação de um programa que simula um sistema de memória virtual. Isso envolve a implementação de políticas de substituição de páginas e da tabela de páginas. O trabalho explora diferentes implementações, tanto para a política de substituição de páginas quanto para a tabela de páginas, implementados na linguagem C.

As políticas de substituição de páginas utilizadas no presente trabalho são **aleatórias**, **FIFO** (Primeiro a Entrar é o Primeiro a Sair), **LRU** (Menos Recentemente Utilizado) e **Segunda Chance** (Semelhante ao FIFO, porém com uma fila circular e um bit de sobrevida). Detalhes sobre a implementação de cada método podem ser encontrados nas seções posteriores.

Em relação as possíveis tabelas de páginas, se tem as tabelas Densas, Hierárquicas de 2 e 3 níveis e a Invertida. Bem como no caso anterior, detalhes de implementação podem ser encontrados nas seções posteriores.

Para realizar as simulações, é utilizado um arquivo de registro de acessos a memória por parte de um processo, contendo o endereço acessado e a operação realizada (leitura ou escrita) no endereço. Nesse contexto, é realizado um estudo sobre a performance de cada uma das possíveis implementações em função do tempo de execução, da quantidade de *page faults* e de *páginas sujas* escritas em disco, para cada um desses programas.

2 Metodologia e Desenvolvimento

A presente seção trata da implementação do simulador de memória virtual e das decisões de projeto dos autores do trabalho, a respeito da implementação das memória, políticas de substituição de páginas e tabela de páginas, bem como dos Tipos Abstratos de Dados (**TADs**) utilizados na implementação.

2.1 Leitura dos Dados de Entrada

A leitura dos dados é feito a partir de um arquivo, no formato "END OP", onde endereço é o endereço de memória em hexadecimal e a operação é uma letra R para casos de leitura, ou uma letra W para casos de escrita. O nome do arquivo deve ser passado

como argumento na execução do programa, como específicado no Apêndice A do presente documento.

2.2 Tipos Abstratos de Dados

A implementação conta com alguns Tipos Abstratos de Dados, representados por *structs* no código em C. Esses **TADs** são:

- **Frame:** Representa os quadros da memória. Contém uma posição da tabela, que referencia aquele quadro, bem como bits de validade, sujo, último acesso e referência para auxiliar a implementação das políticas.
- **Page:** Representa as páginas, da tabela de páginas, contendo um endereço de memória para o quadro que contém aquela página e um bit de validade para saber se a página está na memória.
- **Memory:** Representação geral da memória, contendo um *array* de *Frames* , o tamanho da memória, a política de reposição e alguns itens auxiliares para a implementação.

Esses são os principais **TADs** da implementação, porém não são os únicos. Cada tabela tem suas particularidades e suas implementações, então nos diferentes arquivos que implementam cada uma dessas tabelas existem **TADs** adicionais, que substituem a implementação de *Page*, utilizados pelas tabelas hierárquicas de nível 2 e 3, que contam com **TADs** específicos para facilitar a implementação dos níveis.

2.3 Tabela de Páginas

Como abordado na introdução, para a tabela de páginas existem 4 possibilidades distintas, sendo elas

- **Densa:** Um grande array que contém todas as páginas possíveis, sendo facilmente indexada pelo endereço lógico para acessar o quadro mapeado para aquela página.
- **Hierárquica de 2 níveis:** Um array de tamanho menor inicialmente alocado, onde cada posição desse array, é um outro array, numa estrutura semelhante à uma matriz. Observe que o segundo array armazena o endereço físico da página em questão. Além disos, esse segundo array é alocado sob demanda, conforme o programa necessita de mais páginas.
- **Hierárquica de 3 níveis:** Exatamente como a hierárquica de 2 níveis, porém agora todas as posições do segundo array também são arrays, como uma estrutura tridimensional. Como anteriormente, as informações sobre o endereço físico alocado para aquela página estão no 3° nível, bem como o 2° nível e o 3° nível são alocados sob demanda do programa.
- **Tabela Invertida:** Para a última opção da tabela, o array se torna de tamanho semelhante a memória, e a forma como foi implementado foi simplesmente adicionando um campo adicional no TAD *Frame*, que representa o endereço lógico que faz referência ao quadro. Dessa forma, ao representar a memória como vários *Frames*, a criação da tabela invertida é feita automaticamente.

Sobre a implementação das páginas, além dos fatos apresentados anteriormente existem algumas considerações adicionais sobre decisões de projeto. A primeira delas é sobre o tamanho de cada nível nas tabelas hierárquicas de 2 e 3 níveis. Após a remoção do bit de offset relacionado ao tamanho da página, o restante é dividido igualmente em 2 ou 3 partes, dependendo da hierárquica a ser utilizada.

Caso o resultado da divisão não seja inteiro, por exemplo, 15 bits, a atribuição é feita de modo que as tabelas dos níveis mais internos tenham mais bits, nesse caso, 7 bits para o nível 1 e 8 bits para o nível 2. Para ilustrar no caso de 3 níveis, que após a remoção do offset resta 13 bits, a divisão feita pelo programa será 4 bits para nível 1, 4 bits para nível 2 e 5 bits para nível 3. Se aumentarmos para 14, adicionamos o bit extra no nível 2, naturalmente, obtendo 4 bits para nivel 1, 5 bits para nível 2 e 5 bits para nível 3.

Por fim, cada nível da hierarquia conta com um TAD específico. Para ajudar na implementação, as entradas dos níveis não-finais, i.e., nível 1 no caso de 2 níveis e níveis 1 e 2 em caso de 3 níveis contam com um bit de alocação, para verificar se os níveis posteriores estão alocados ou não.

2.4 Políticas de Substituição de Páginas

As políticas de substituição de páginas tem um papel crucial no sistema de memória virtual, sendo responsáveis por explorar a localidade dos programas e evitar ao máximo o overhead de acesso ao disco para carregar os dados em memória. Como apresentado no início do presente documento, temos políticas diferentes de substituição de páginas, descritas abaixo.

- **LRU:** Para cada *Frame* da memória, é mantida uma informação referente à última vez que ele foi acessado, e sempre que é preciso realizar uma troca na memória, o frame que foi acessado a mais tempo é substituido pelo novo.
- **FIFO**: O *Frame* escolhido para sair é o primeiro a ter entrado na memória. É implementado usando uma ideia de fila.
- **Segunda chance**: Cada *Frame* é associado a um bit de referência. Quando um frame é escolhido para substituição, verifica-se o valor do bit de referência:
  - Se o bit de referência for 0, o *Frame* é removido.
  - Se o bit de referência for 1, o *Frame* recebe uma "segunda chance". O bit é zerado, e o *Frame* é movido para o final da fila, indicando que ele foi acessado recentemente.
- **RANDOM**: Seleciona uma posição aleatória da memória, com base nas funções rand() e srand() e substitui.

As decisões de projeto associadas à implementação das políticas de substituição envolvem os TADs apresentados anteriormente. Para suporte das políticas de LRU e Segunda Chance, o *Frame* contém 2 campos, um chamado *last access* e outro chamado *ref bit*.

Para a implementação da fila, foi utilizado simplesmente um campo adicional acoplado na própria memória, chamado de *first*, que auxilia na política de FIFO. Por fim, para a política aleatória, a única decisão envolve usar a função time() em conjunto com srand(), para aleatorizar o experimento em diferentes execuções.

2.5 Contabilização de Erros

Para contabilizar a quantidade de *page faults*, um contador é incrementado todas as vezes em que uma política de substituição de páginas é acionada. Além disso, quanto uma substituição de páginas é efetuada e a política seleciona uma página com o bit de *dirty* igual à 1, ocorre o incremento do contador de escritas, sendo as duas métricas de avaliação de qualidade para as políticas de reposição.

2.6 Custo de Memória

Para definir o custo de memória associado as tabelas, todo e qualquer custo de memória comum entre as diferentes tabelas foi desprezado, sendo levado em consideração apenas as características relevantes para o gasto de memória. A ideia será explicitada abaixo. OBS: (4 bytes = tamanho de um inteiro)

- **Densa:** $\frac{2^{32-offset}*4}{1024}$ KB
- **Hierárquica de 2 níveis:** $\frac{2^{32-offset-level1}*alloc_2*4}{1024}$ KB, onde $alloc_2$ é a quantidade de tabelas de nível 2 que foram alocadas.
- **Hierárquica de 3 níveis:** $\frac{2^{32-offset-level1-level2}*alloc_3*4}{1024}$ KB, onde $alloc_3$ é a quantidade de tabelas de nível 3 que foram alocadas.
- **Tabela Invertida:** $\frac{memsize*4}{1024}$ KB onde $memsize$ é a quantidade de *Frames*.

2.7 Tempo

Para analisar o tempo, considerou-se apenas o tempo em que o programa lê os endereços do arquivo de log e executa. Tempos adicionais, como alocação inicial das tabelas, bem como a desalocação da memória ao final da execução não foram considerados.

3 Análise dos Resultados

A análise dos resultados envolve análise de tempo, quantidade de *page faults* para análise das políticas de reposição, e uma análise do gasto de memória. Além disso, durante as análises, sempre é proposto um cenário onde o tamanho da memória é fixo, variando-se a página e outro onde a página é fixa, variando-se o tamanho da memória. Propõe-se uma divisão em tópicos dessas análises individualmente para entender como as decisões de algoritmos e tabelas influenciam em cada um desses aspectos, finalizando em uma análise de prós e contras e observações relevantes.

3.1 Análise de Page Faults

Com base na Figura 1, é possível observar que ao aumentar o tamanho das páginas a quantidade de hits aumenta. Isso é esperado, visto que nossa memória é fixa e o aumento da página reduz a quantidade total de quadros, ainda é mais interessante ter uma quantidade boa de quadros, antes de tentar aumentar os tamanhos da página. Esse comportamento também se repete na Figura 2.

É válido ressaltar que, geralmente, um aumento das páginas causa uma redução da quantidade de misses, mas no contexto do presente trabalho prático isso não é realidade,

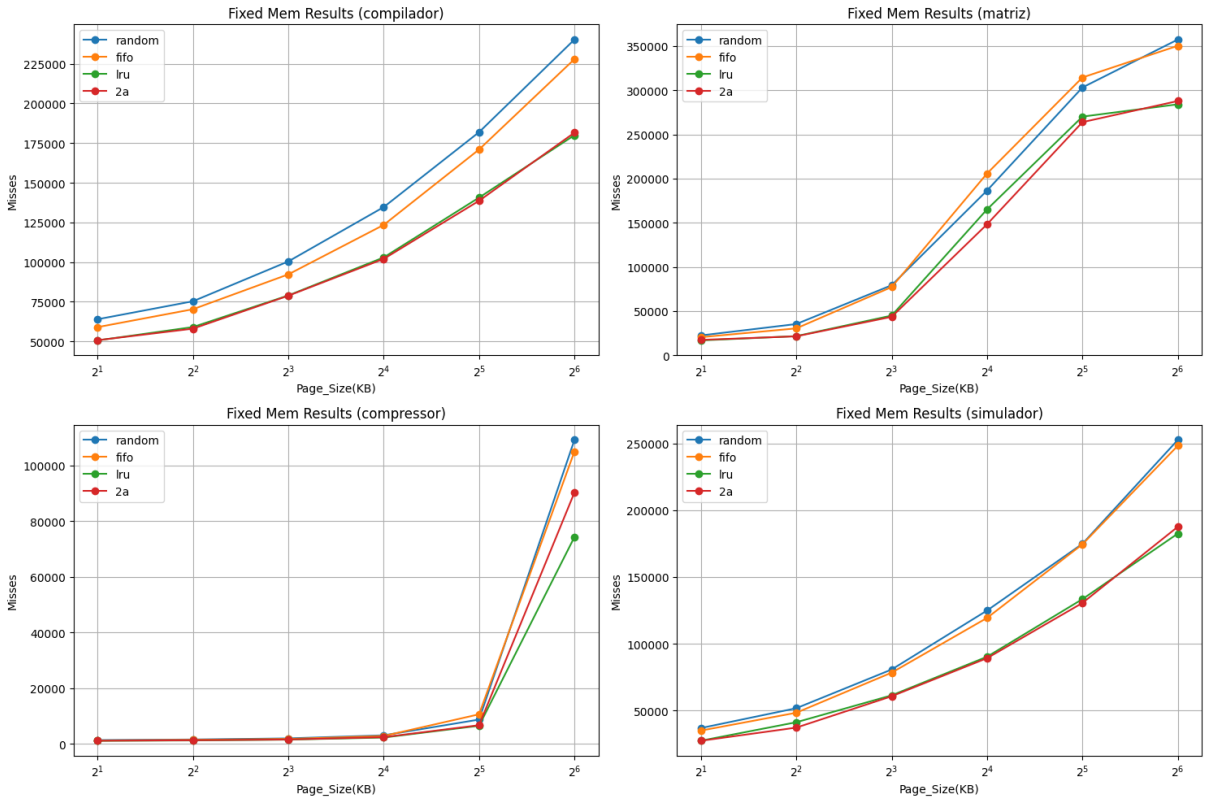

Figura 1: Misses por política de reposição variando o tamanho da página - Memória fixa 256KB.

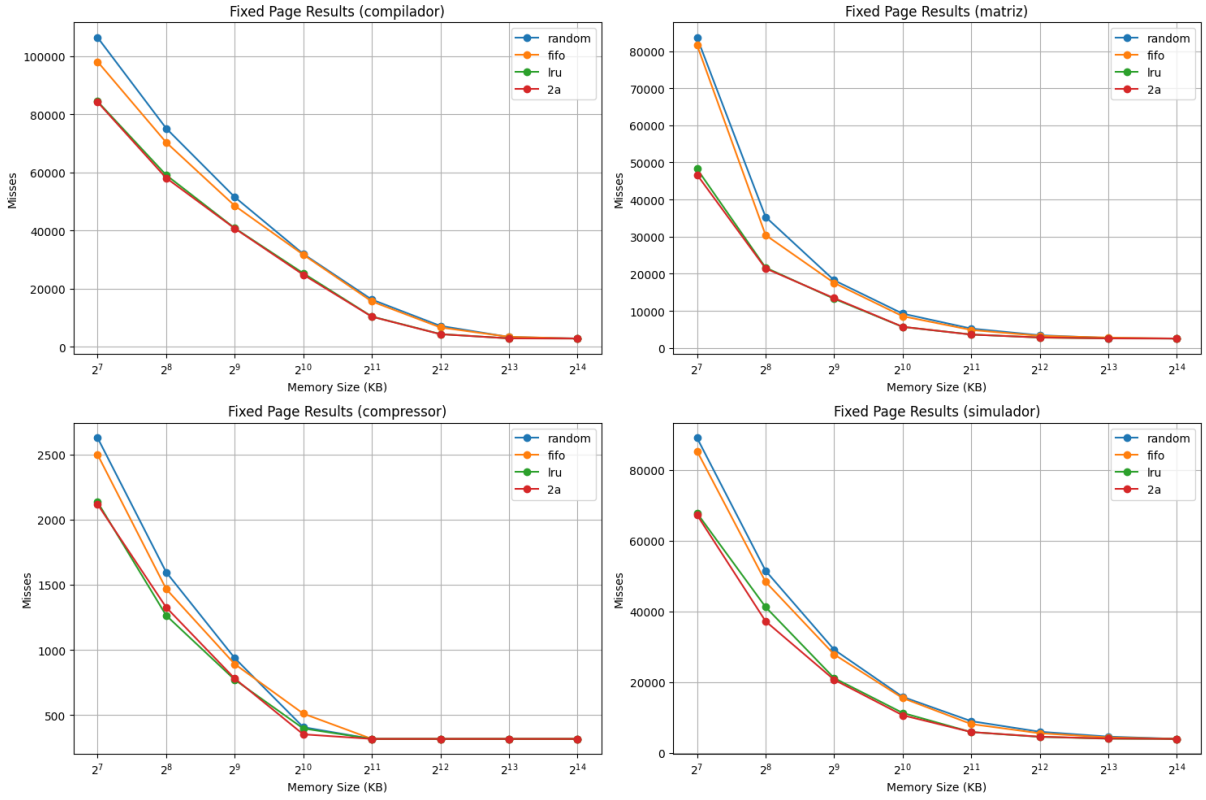

Figura 2: Misses por política de reposição variando o tamanho da memória - Página fixa 4KB.

pois a redução do número de quadros não vêm acompanhada de uma grande melhora na exploração da localidade de referência espacial proporcionada pelo aumento das páginas.

Por fim, analisando as políticas de reposição, é possível observar que as políticas aleatórias e FIFO são as piores, em todos os casos onde existe uma frequente reposição de páginas e disputa por memória. Enquanto as políticas de Segunda Chance e LRU apresentam as menores quantidades de *page faults*, sendo difícil definir um melhor, visto que em muitos momentos os valores são próximos. O comportamento para *page faults* se repete para escritas na memória.

3.2 Análise de Tempo para Políticas de Reposição

O objetivo desse tópico é compreender como as diferentes políticas de reposição podem influenciar no tempo de execução do algoritmo. É esperado que não existe uma diferença muito grande entre as políticas, visto que até mesmo políticas que deveriam ser custosas, como Segunda Chance, são implementadas eficientemente com uso de estruturas auxiliares.

Observe que em ambas as Figuras 3 e 4 a política aleatória consumiu mais tempo que as outras. Provavelmente, devido a forma como a geração de números aleatórias funciona por parte das funções utilizadas deve ser a causa desse maior tempo.

Numa perspectiva geral, considerando variações estatísticas com base no comportamento pouco padronizado do tempo de execução do programa, bem como a escala pequena de tempo ser suscetível a fatores externos, é possível afirmar que o teste em particular é inconclusivo.

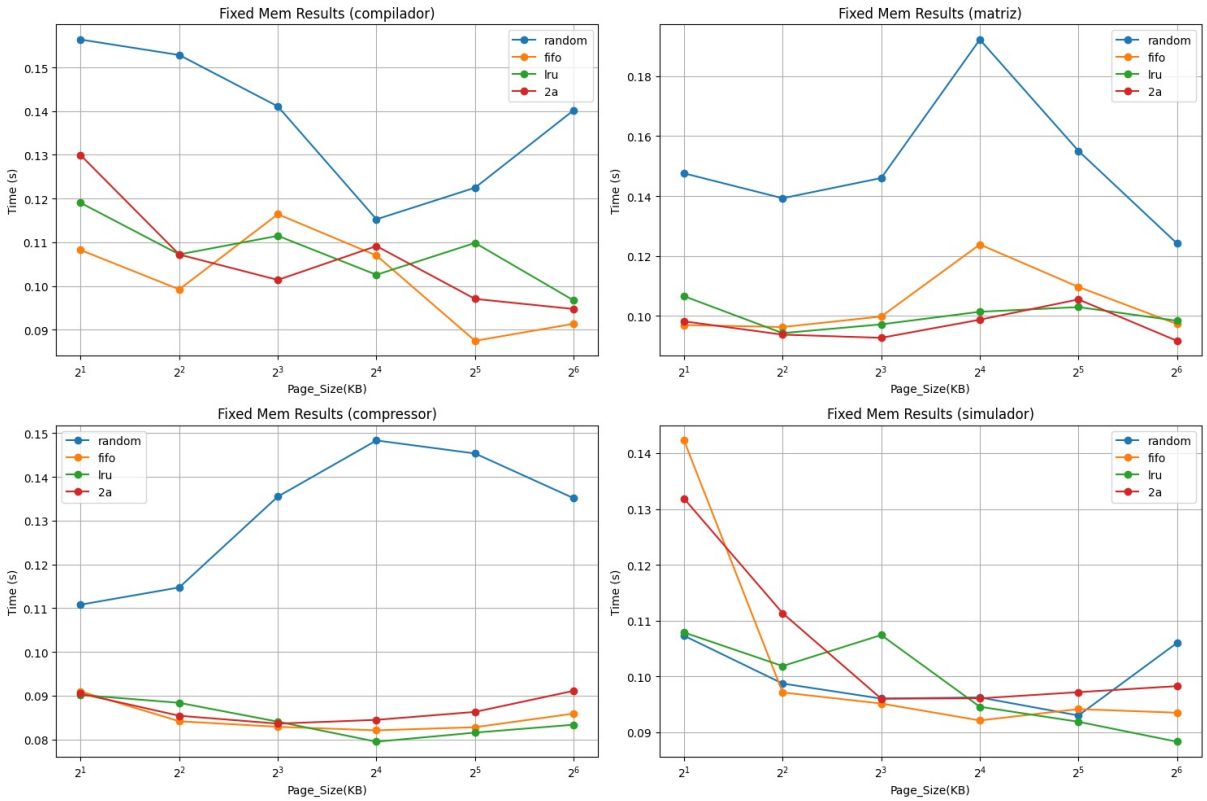

Figura 3: Tempo de execução por política de reposição variando o tamanho da página - Memória fixa 256KB.

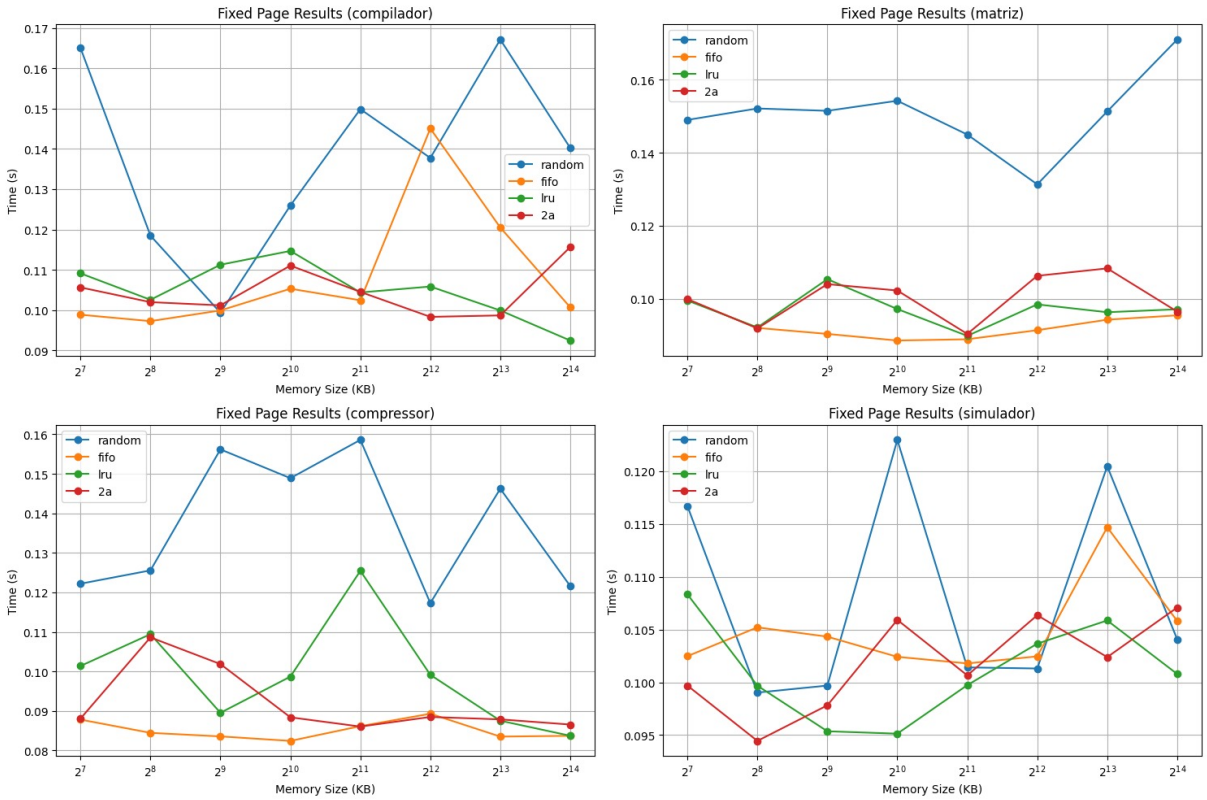

Figura 4: Tempo de execução por política de reposição variando o tamanho da memória - Página fixa 4KB.

3.3 Análise de Tempo para Tabela de Páginas

Para essa análise, é esperado que as tabelas hierárquicas e densas não tenham uma variação grande, porém a densa seja mais rápida que a hierárquica de 2 níveis, enquanto essa por sua vez é mais rápida que uma hierárquica de 3 níveis. Para a tabela invertida, esperamos um tempo alto, que seja proporcional ao tamanho da memória.

Analisando os gráficos da Figura 5, observe que um aumento do tamanho das páginas vem acompanhado de uma redução no tempo de execução da tabela invertida, o que é totalmente esperado, visto que esse aumento de página reduz a quantidade de quadros da memória. Na Figura 6, o tempo de execução para a tabela invertida aumenta, o que também é esperando, pois o tamanho da memória aumenta. Lembrando que a tabela invertida é uma busca na memória, então esse comportamento é completamente esperando.

Por fim observe que entre todos os programas e em ambas as figuras, os métodos de tabela hierárquica de 3 níveis tem tempos maiores do que as tabelas hierárquicas de 2 níveis, que por sua vez tem tempos maiores do que as tabelas densas. Na próxima seção, é feita uma análise para entender o uso de memória das diferentes tabelas, e o dilema entre custo de memória e rápido tempo de execução que envolve a escolha das tabelas.

3.4 Análise de Espaço para Tabela de Páginas

Para a análise de espaço, consideramos as tabelas densas como referença para definir o custo de espaço outras tabelas, visto que a densa apresenta o maior custo espacial possível, que pode ser confirmado no Tópico 2.6.

Além disso, é válido ressaltar que a Tabela Invertida apresenta gasto de memória muito baixo, proporcional ao tamanho da memória, que na nossa simulação nunca apresenta grandes tamanhos. Então, ao realizar o cálculo e obter a medida em KB, acaba por ter

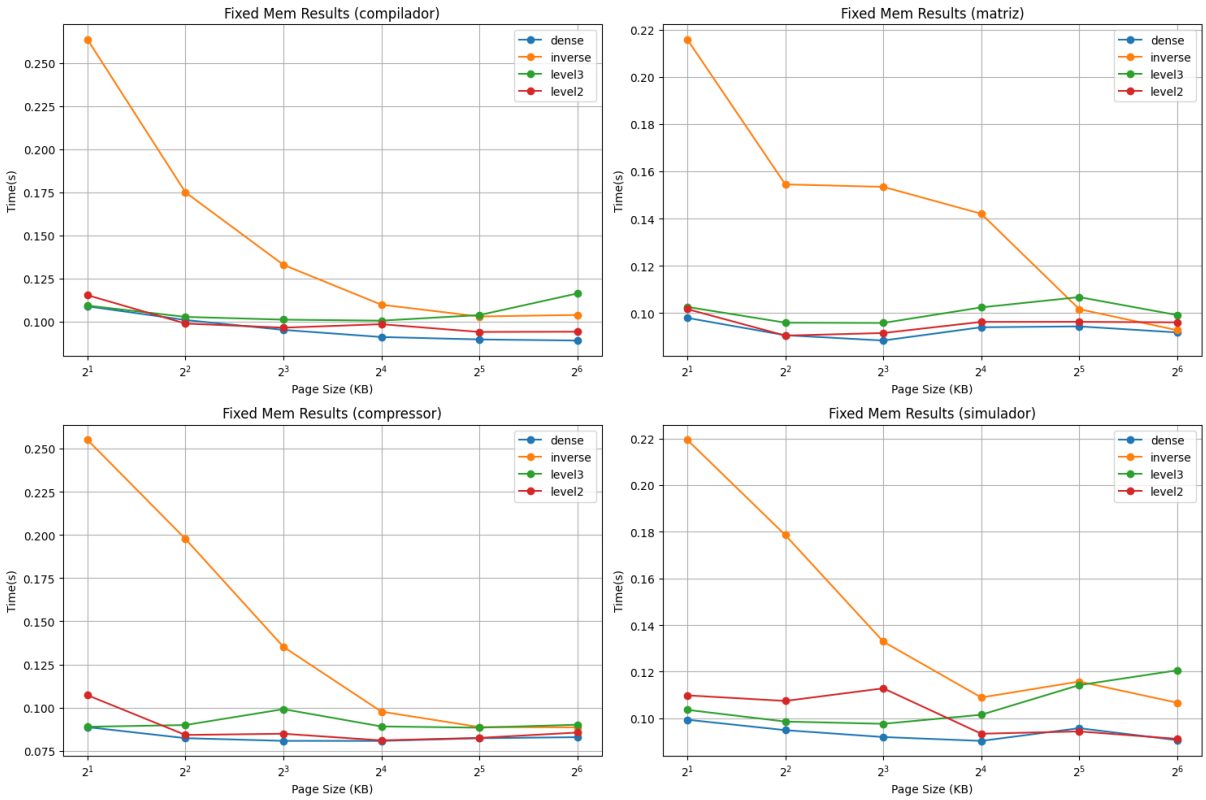

Figura 5: Tempo de execução por Tabela de Páginas variando o tamanho da página - Memória fixa 256KB.

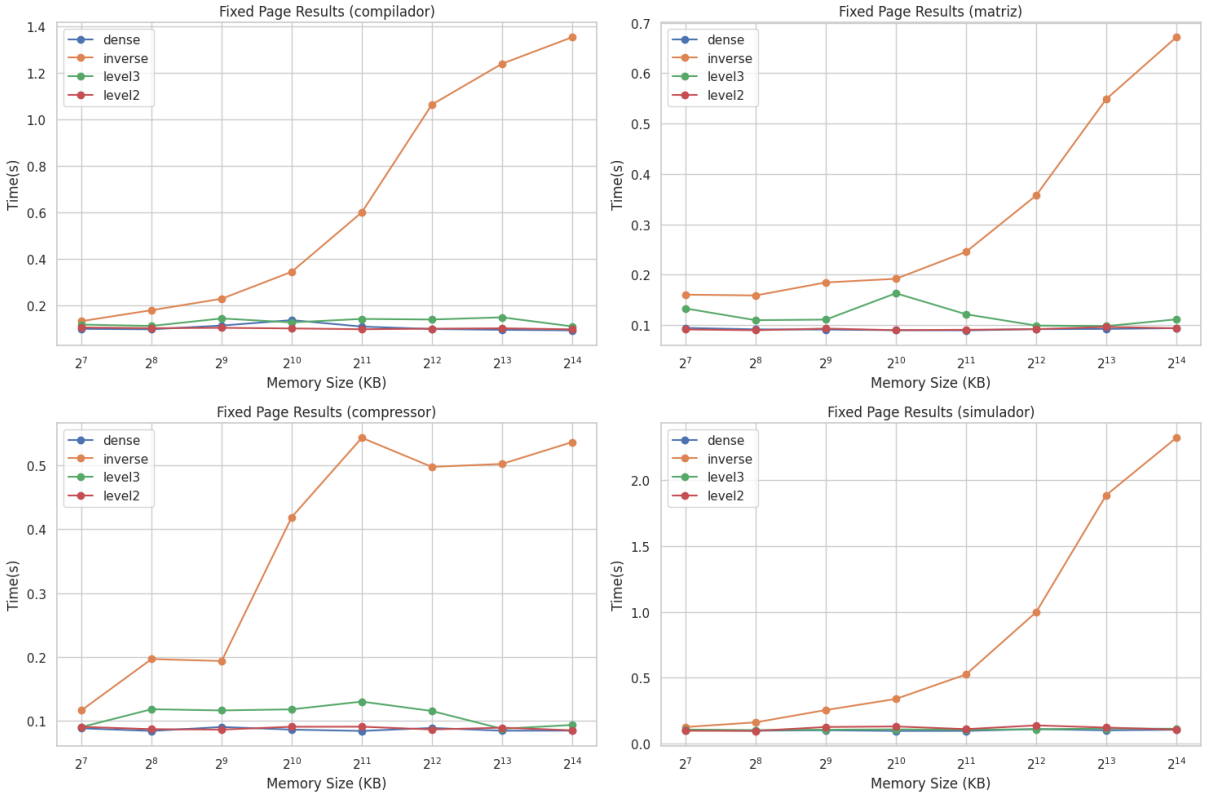

Figura 6: Tempo de execução por Tabela de Páginas variando o tamanho da memória - Página fixa 4KB.

valores menores que 1 KB, sendo arredondados para 0.

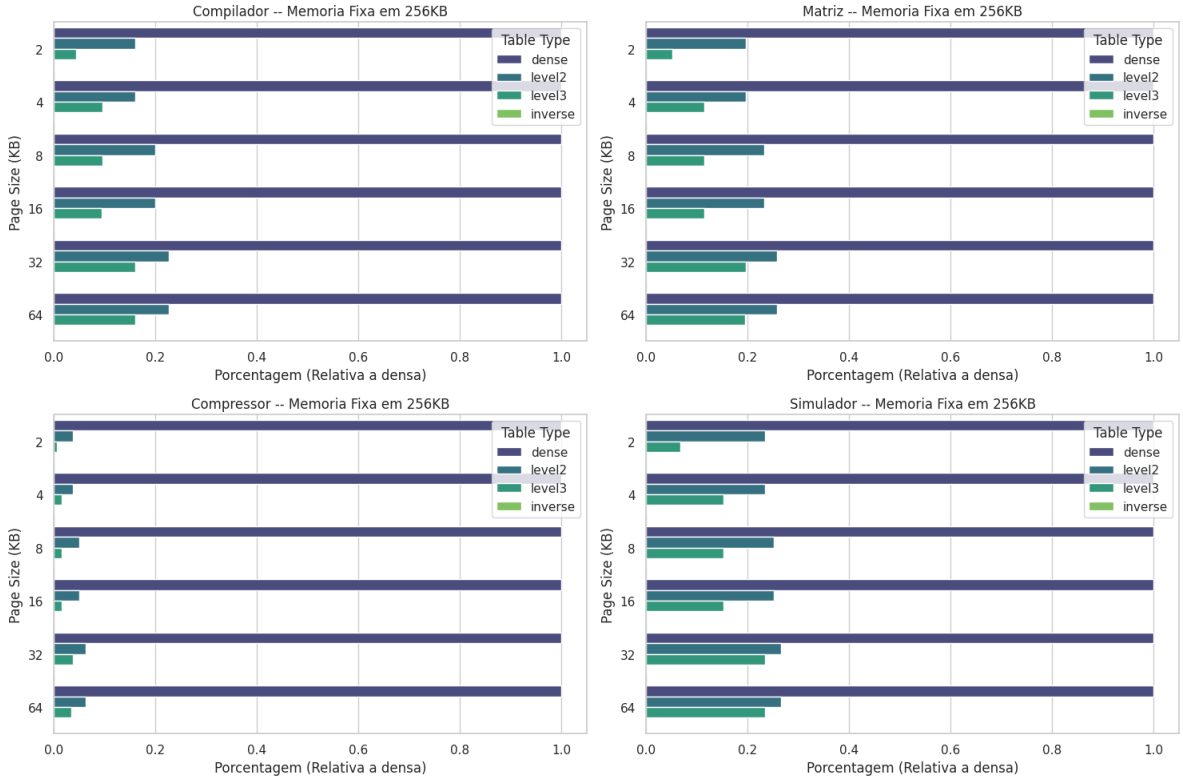

Figura 7: Memória ocupada por Tabela de Páginas variando o tamanho da página - Memória fixa 256KB.

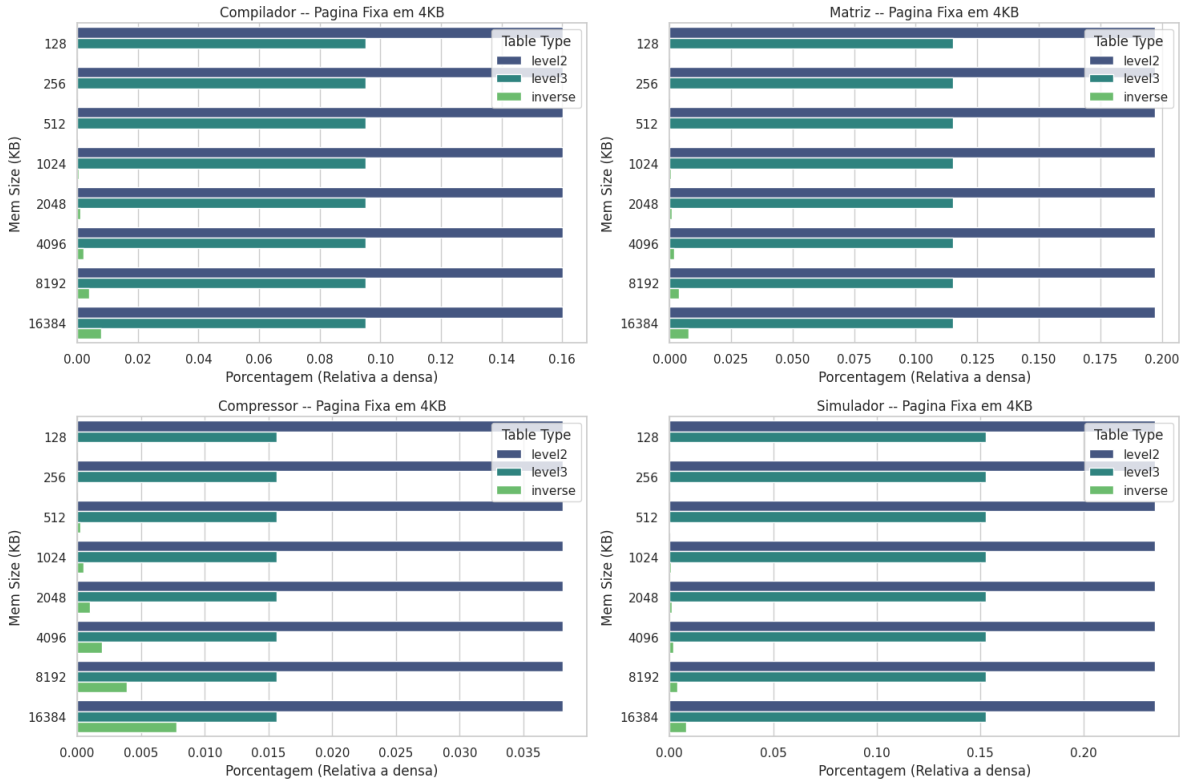

Figura 8: Memória ocupada por Tabela de Páginas variando o tamanho da página - Página fixa 4KB.

Analisando a Figura 7, é possível observar um comportamento muito interessante que ocorre ao aumentar as páginas nas tabelas hierárquicas. Naturalmente, ao aumentar o tamanho das páginas e observar as equações da Seção 2.6, é intuitivo pensar que um maior offset, causando pelo aumento do tamanho da página, será acompanhado de uma redução na memória total gasta pela tabela. Mas vejam que os gráficos nos dizem outra coisa.

A razão desse comportamento é porque, apesar da haver sim uma redução no tamanho das páginas que são alocadas nos níveis finais, a redução dos bits faz com que exista uma probabilidade maior de tabelas de nível 2 ou nível 3 serem alocadas, pois o mapeamento do endereço com os bits restante se tornou mais denso, ou seja, endereços minimamente distantes podem resultar em alocações de novas tabelas nos níveis finais. Além disso, observe que os valores de memória ocupados pela Inversa são desprezíveis no exemplo.

Avançando agora para a Figura 8, o tamanho de todas as tabelas hierárquicas são constantes, o que é esperado, visto que dependem exclusivamente do tamanho das páginas e do arquivo de registro utilizado. Aumentar ou reduzir a memória não impacta nas tabelas.

Porém, esse comportamento se altera, agora que tratamos das tabelas invertidas, que tem complexidade de espaço em função do tamanho da memória, agora tendo valores não desprezíveis de gasto de memória nesse último caso.

4 Conclusão

A partir de uma retrospectiva geral das simulações, é possível concluir que o trabalho foi satisfatório do ponto de vista dos resultados, sendo todas as simulações concluídas com sucesso e fornecendo uma experiência aos projetistas a respeito da implementação, tanto das políticas de troca de página quanto das diferentes implementações de tabela de páginas.

Nesse contexto, análise dos resultados permiu uma análise profunda sobre as políticas de substituição de páginas em função dos misses. Existem políticas que de fato causam mais misses do que outras, sendo as políticas de Segunda Chance e LRU as com melhor performance nesse sentido. Além disso, no contexto de misses, é possível observar que um aumento das páginas e redução da quantidade de quadros vem acompanhado de um aumento dos misses, pois apesar de proporcionar uma melhor localidade espacial devido a páginas maiores, esse benefício não proporciona ganhos quando se tem memórias com pouca quantidade de quadros, sendo necessário realizar reposições com mais frequência.

Sobre a implementação dos diferentes tipos de tabela de páginas, é possivel fazer uma análise e entender melhor o compromisso existente entre o tempo de execução e o gasto de espaço. É possível observar de forma muito clara que uma redução forte no gasto de espaço vem acompanhada de altos tempos de execução, observados através do comportamento da Tabela Inversa. Em oposição, se tem a Tabela Densa, que apresenta baixos tempos de execução, mas o maior gasto de memória dentre todas as opções, sendo inviáveis para sistemas de 64 bits, por exemplo. Em última análise, existem as opções intermediárias com as tabelas hierárquicas, que vem acompanhadas de um overhead adicional de acesso porém boas economias no custo espacial, o que é bem interessante num cenário onde não desejamos extremos.

Ainda no contexto das tabelas, também foi possível observar essa pequena "anomalia"causada pelo aumento do tamanho da página, onde era esperado que as tabelas hierárquicas ocupassem menos memória mas acabou tendo um efeito contrário devido as probabilidades maiores de se alocar uma nova tabela para pequenas distâncias entre endereços.

Por fim, o trabalho como um todo apresentou resultado satisfatório e permitiu aos projetistas, além de terem contato com a implementação dos diferentes métodos, a capacidade de fazer análises profundas sobre compromissos entre custo de memória x velocidade de execução, além dos comportamentos envolvendo mudanças no tamanho das páginas e no tamanho da memória.

A Apêndice - Instruções de Compilação e Execução

A.1 Pré-requisitos

Antes de compilar o projeto, você precisará de:

- O compilador GCC instalado em seu sistema.
- As bibliotecas necessárias para o projeto (`-lm` para operações matemáticas).

A.2 Passos para Compilar e Executar

1. Compile o projeto usando o Makefile:

   ```
   make run
   ```

2. Execute o programa:

   ```
   ./bin/tp2virtual [algoritmo] [caminho do arquivo] [Tamanho da pagina]
   [Tamanho da memória] [tipo de tabela]
   ```

   onde

   - 'algoritmo' = lru — 2a — fifo — random
   - 'tipo de tabela' = dense — level2 — level3 — inverse
   - O quinto parâmetro (tipo de tabela) é opcional e, caso não seja declarado, repetirá a simulação para todos os tipos de tabela.

3. Limpeza dos arquivos gerados:

   ```
   make clean
   ```

Isso deve compilar e executar o programa com sucesso!