Universidade Federal
de São João del-Rei

Implementação de um Simulador de Memória Virtual

Sistemas Operacionais

Professor: Rafael Sachetto Oliveira

Pedro Garcia
Matheus Tavares Elias

1. INTRODUÇÃO

Este trabalho consiste na implementação de um simulador de memória virtual em linguagem C. O objetivo é replicar as estruturas de um mecanismo de gerência de memória virtual, processando uma sequência de endereços de memória acessados por um programa real e simulando o processo de carga e substituição de páginas.

O simulador recebe como entrada o tipo de política de substituição de páginas, um arquivo contendo os endereços de memória acessados, o tamanho da página e o tamanho da memória. Ele executa as entradas dos logs, realizando as operações necessárias de carga e substituição de páginas de acordo com a política escolhida. Ao final da execução, o simulador gera um relatório com informações relevantes, como o número de defaults, páginas sujas, leituras de memória e escritas de memória realizadas.

Essa implementação permite a análise e compreensão do funcionamento de um sistema de memória virtual, contribuindo para o estudo de sistemas operacionais e estruturas de gerenciamento de memória. O objetivo final é fornecer um simulador eficiente, capaz de processar a sequência de acessos à memória e gerar o relatório final em um tempo razoável, de preferência na ordem de segundos.

2. FORMATO DE ENTRADA E SAÍDA

O simulador de memória virtual, chamado tp2virtual, é um programa implementado em C que recebe como entrada um arquivo contendo uma sequência de endereços de memória acessados por um programa real. Cada linha do arquivo contém um endereço de memória representado em formato hexadecimal, seguido por uma letra "R" ou "W", indicando se o acesso foi de leitura ou escrita, respectivamente.

Para iniciar a execução do simulador, são fornecidos quatro argumentos na linha de comando: o algoritmo de substituição de páginas a ser utilizado ("lru", "nru" ou "segunda_chance"), o arquivo contendo os acessos à memória, o tamanho de cada página ou quadro de memória em kilobytes (no intervalo de 2 a 64), e o tamanho total da memória física disponível para o processo, também em kilobytes (no intervalo de 128 a 16384, correspondendo a 16 MB). Um exemplo : ./tp2virtual <politica_subs> <arquivo> <tamanhopagina> <tamanhoMemoria>

Durante a execução, o programa processa cada acesso à memória, atualizando os bits de controle de cada quadro, detectando faltas de páginas (page faults) e simulando o processo de carga e substituição de páginas. Ao final da simulação, são coletadas estatísticas para gerar um relatório com as seguintes informações:

- Configuração utilizada: inclui o algoritmo de substituição escolhido, o nome do arquivo de entrada, o tamanho da memória e o tamanho das páginas/quadros.

- Número total de acessos à memória contidos no arquivo.
- Número de page faults ocorridos durante a simulação.
- Número de páginas "sujas" que precisaram ser escritas de volta no disco, (É importante ressaltar que páginas sujas existentes no final da execução não são consideradas nesse valor).

O relatório é apresentado de forma clara e organizada para fornecer uma visão abrangente do desempenho do simulador.

3. ESTRUTURA DO CÓDIGO

A estrutura do código é composta por dois arquivos: main.c e memory.c, que contém as implementações principais.

3.1. MAIN

No arquivo main.c, temos a função main, responsável por iniciar a execução do simulador de memória virtual. Ele recebe os argumentos da linha de comando, verifica se a quantidade mínima de argumentos foi fornecida e exibe informações sobre a configuração utilizada. Em seguida, chama as funções para criar a memória, executar as entradas do arquivo de log, gerar o relatório e liberar a memória alocada (que serão explicadas mais detalhadamente no arquivo memory).

3.2. MEMORY

No arquivo memory.c, encontramos a implementação das funções relacionadas à memória virtual. Temos funções para construir e destruir a estrutura de memória, calcular o valor de s (quantidade de bits descartados), criar os frames de página e inicializá-los como vazios.

Em seguida, são implementados três algoritmos de substituição de páginas: NRU (Not Recently Used), LRU (Least Recently Used) e Second Chance. Cada algoritmo possui sua própria função que recebe a memória, seu tamanho e, em alguns casos, outros parâmetros necessários. Essas funções são responsáveis por escolher e retornar o índice do frame a ser substituído.

A função *algoChooser* recebe o nome do algoritmo fornecido e escolhe a função correspondente para a substituição de páginas. Em caso de nome inválido, a função imprime uma mensagem de erro e termina a execução.

A função *frameIndex* é utilizada para encontrar o índice do frame que possui o identificador virtual dado. Se não encontrar, retorna -1.

A função *execMem* executa cada endereço no arquivo de log. Ela lê o endereço e a operação de leitura/escrita, calcula o identificador da página,

verifica se a página está presente na memória e, em caso negativo, trata o page fault. Para isso, verifica se há espaço na tabela de frames para adicionar a página ou se é necessário escolher uma página para substituir. Em seguida, atualiza as informações do frame e incrementa os contadores correspondentes.

Por fim, a função *report* é responsável por gerar o relatório da execução, exibindo informações como o número total de acessos, o número de page faults, o número de páginas "sujas" e a quantidade de leituras e escritas realizadas.

Essa estrutura do código permite a execução do simulador de memória virtual, aplicando diferentes algoritmos de substituição de páginas e fornecendo estatísticas relevantes para análise do desempenho do simulador.

4. DECISÕES DE PROJETO

Ao analisar o tratamento de linhas que não continham comandos, mas apenas manipulação de arquivos. Foi verificado que as especificações não forneciam uma instrução explícita sobre como lidar com esse tipo de problema.

Para resolver essa ambiguidade, foi adotada uma decisão de projeto de simplesmente ignorar essas linhas durante a execução do interpretador. Ao fazer isso, evitamos processar informações desnecessárias, o que resulta em um consumo reduzido de recursos e um tempo de execução mais eficiente.

Essa abordagem permite uma maior flexibilidade na estrutura do arquivo de log, permitindo a inclusão de informações adicionais, comentários ou outros dados irrelevantes para a execução do simulador. Ao ignorar essas linhas, o interpretador pode se concentrar apenas nos comandos válidos, tornando o processo de interpretação mais preciso e direcionado.

Em suma, ao lidar com linhas que não possuem comandos, apenas manipulação de arquivos, o interpretador adota a estratégia de ignorá-las, garantindo eficiência e foco nos comandos relevantes para a execução do simulador.

5. RECURSOS NÃO IMPLEMENTADOS

Analisando as especificações pedidas, não foram identificados explicitamente nenhum recurso que tenha deixado de ser implementado. No entanto, vale ressaltar que o código se concentra principalmente na implementação do simulador de memória virtual, com algoritmos de substituição de página e tratamento de page faults.

Com base nisso, é possível afirmar que certos recursos comumente encontrados em simuladores de memória virtual não foram implementados.

Como por exemplo o suporte a múltiplos processos, este trabalho não apresenta suporte para a execução de vários processos simultaneamente. O simulador opera com uma única sequência de comandos a partir de um arquivo de log. Além de não utilizar políticas avançadas de substituição de página, o código implementa apenas três algoritmos de substituição de página (NRU, LRU e Second Chance), mas não inclui outros algoritmos considerados usuais, como FIFO (First-In, First-Out), MFU (Most Frequently Used) ou LFU (Least Frequently Used). Como também podemos citar o tratamento de alocação de memória, não há implementação de tabelas de páginas, mapas de bits ou outros mecanismos relacionados à alocação e desalocação dinâmica de memória.

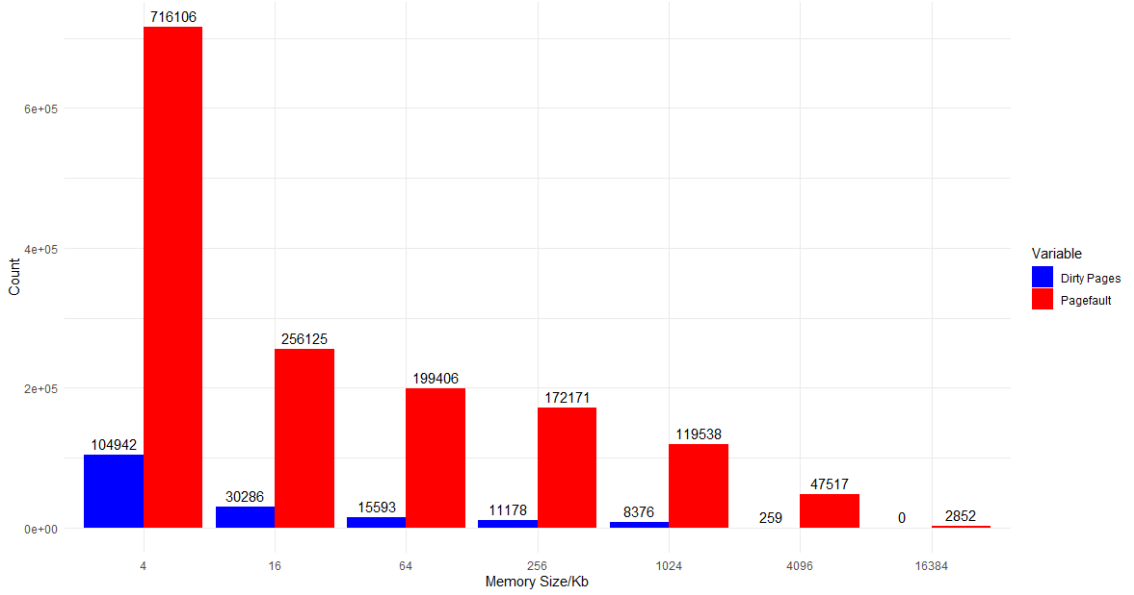
É importante ressaltar que a ausência desses recursos é devido ao escopo específico que foi pedido. Cabendo a quem for utilizar este código avaliar a necessidade e a viabilidade de implementar esses recursos adicionais, levando em consideração as limitações existentes.

6. TESTES

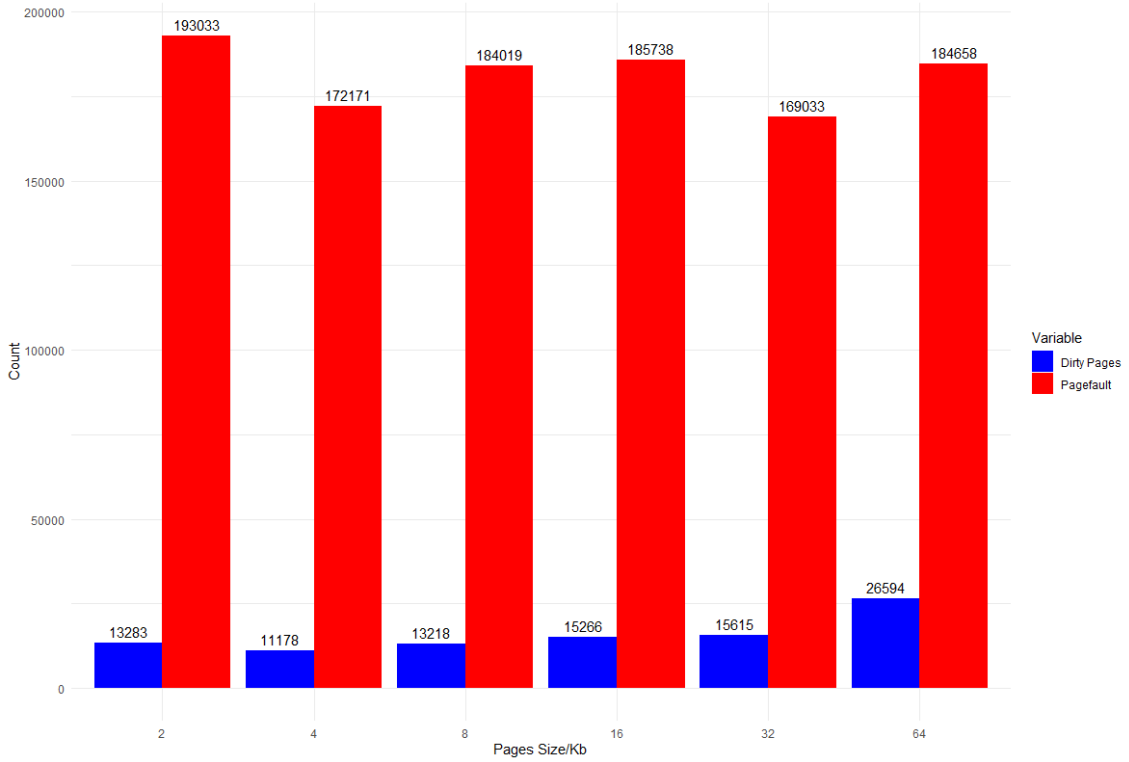
Para montar os gráficos utilizamos o log do compilador que representa bem todos os outros logs. O log do compilador foi escolhido por fornecer uma visão geral dos resultados e tendências esperadas em relação ao desempenho da memória virtual.

6.1. NRU

6.1.1 Memória variável e Paginas 4k

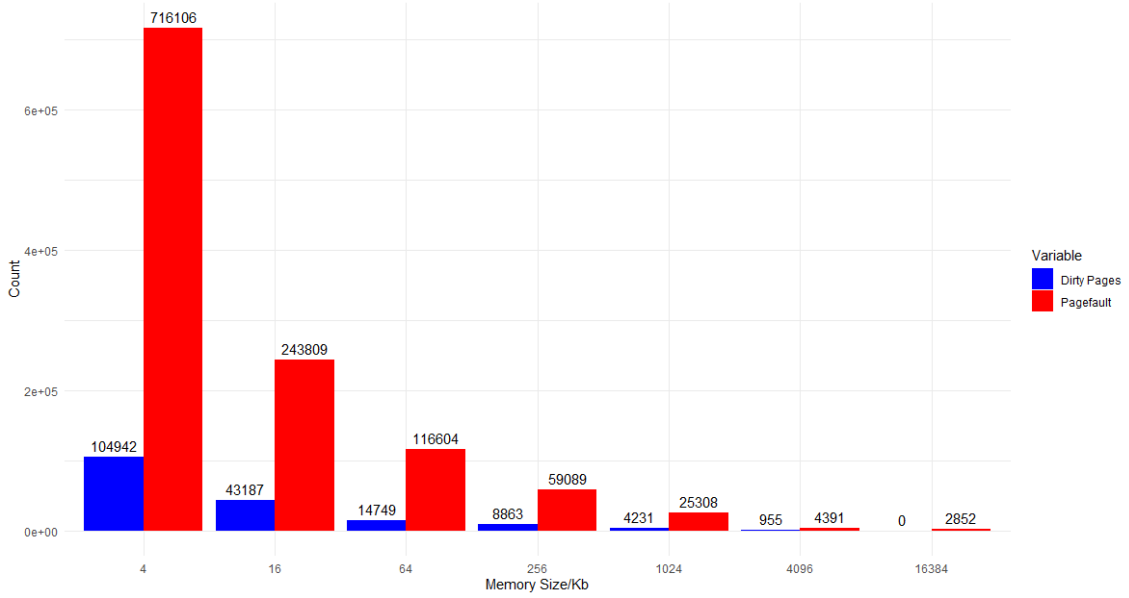


6.1.2 Memória constante em 256kb e Páginas 2 à 64k

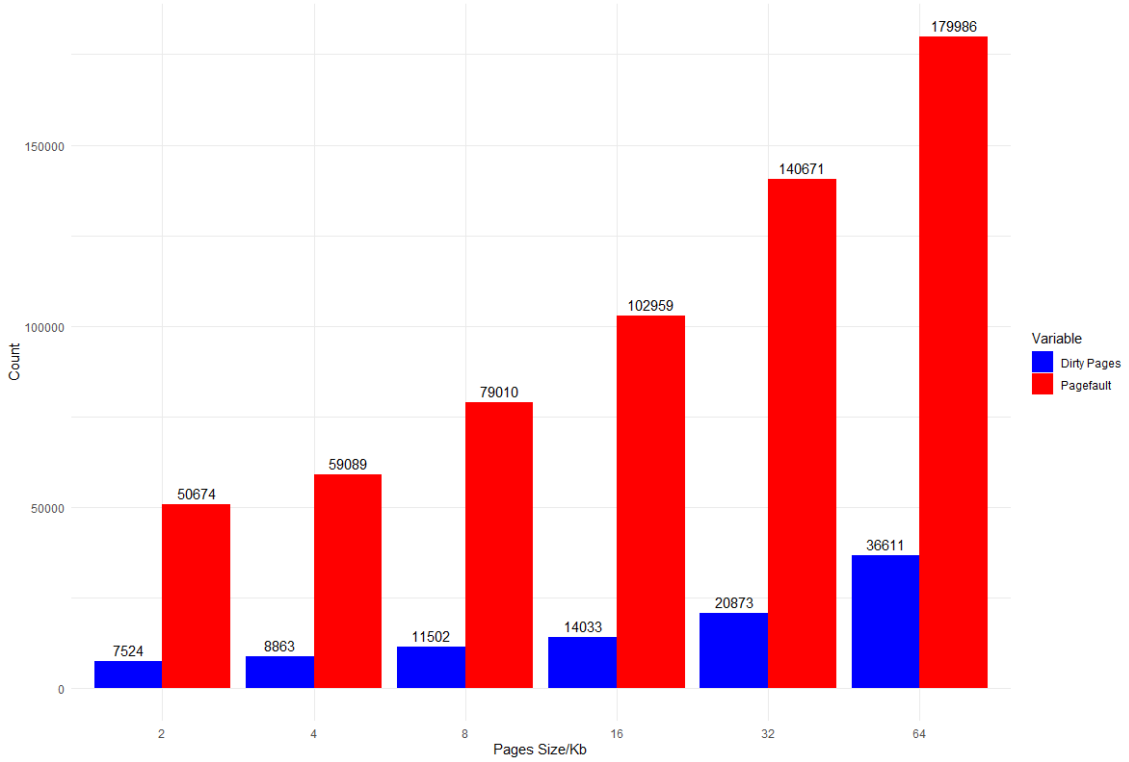


6.2. LRU

6.2.1 Memória variável e Páginas 4k

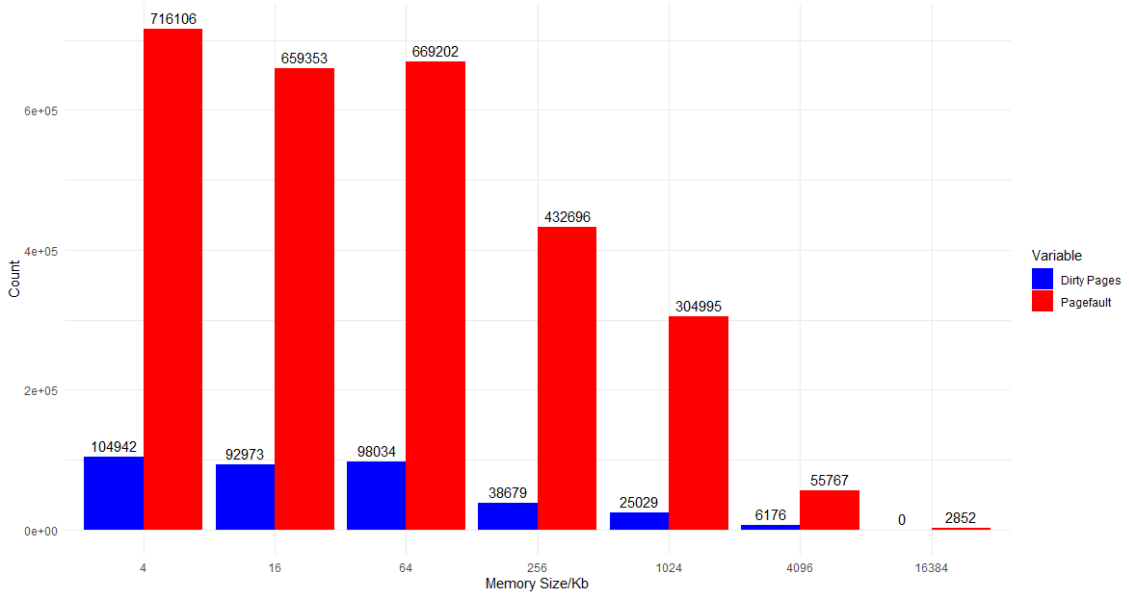


6.2.2 Memória constante em 256kb e Páginas 2 à 64k

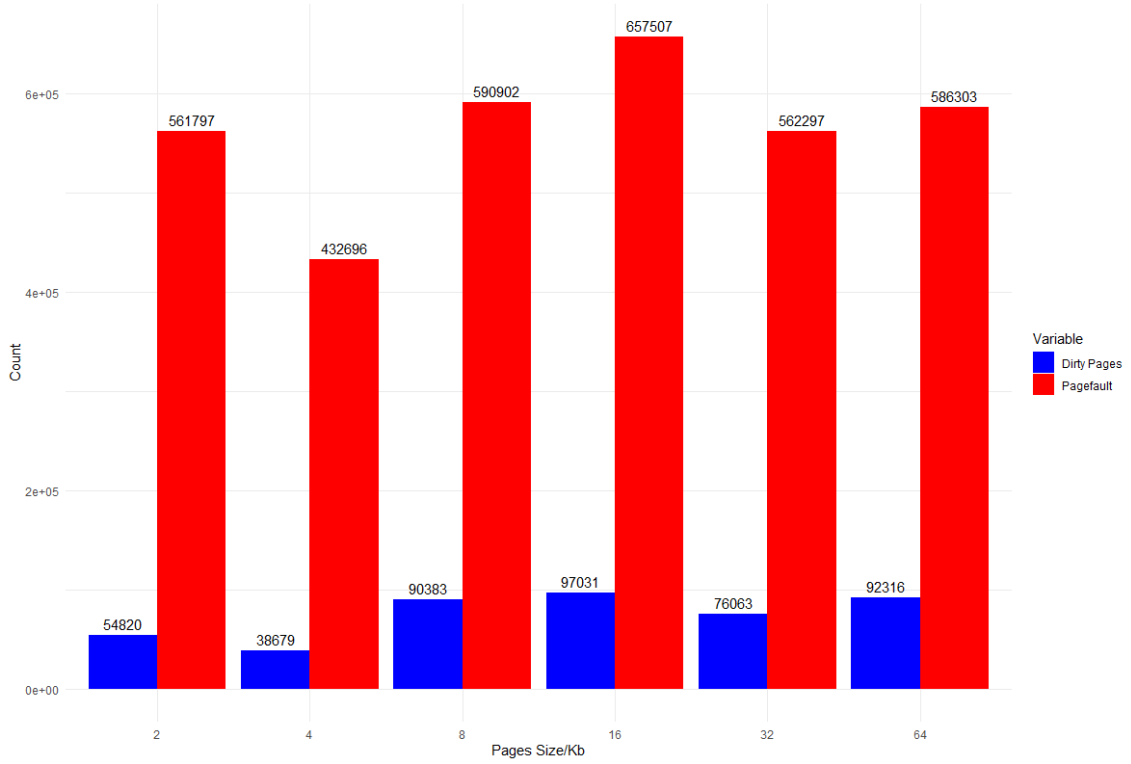


6.3. Second Chance

6.3.1 Memória variável e Páginas 4k



6.3.2 Memória constante em 256kb e Páginas 2 à 64k



Podemos perceber que com tamanho de página fixa em 4kb, a medida que o tamanho da memória cresce o número de page faults logicamente diminui em todas as políticas e em todos os logs. Isso acontece pois a memória fica tão grande que acaba não precisando fazer tantas substituições e todas as páginas já estão carregadas na memória.

No caso da memória constante e tamanho de página variável, o algoritmo que mais se destaca é o LRU pois para tamanhos de páginas pequenos a quantidade de page faults é baixa. Porém, à medida que o tamanho da página se aproxima do tamanho da memória todos os algoritmos aumentam a quantidade de page fault. Esse comportamento é esperado pois quanto maior a página menos páginas poderão existir simultaneamente na memória e precisará fazer mais substituições.

7. CONCLUSÃO

Percebemos que embora o simulador implementado seja capaz de realizar o processamento da sequência de acessos à memória e gerar o relatório final com as informações solicitadas, é importante destacar que alguns recursos comumente encontrados em simuladores de memória virtual não foram implementados neste projeto. Esses recursos incluem o suporte a

múltiplos processos, políticas avançadas de substituição de página e tratamento de alocação de memória. Para futuras melhorias e desenvolvimentos do projeto, é recomendado avaliar a necessidade e a viabilidade de implementar esses recursos adicionais, considerando o escopo e as limitações existentes.

Em relação aos testes realizados, observamos que o tamanho da página e da memória têm um impacto significativo no número de page faults. Com o tamanho da página fixo em 4kb, à medida que o tamanho da memória aumenta, o número de page faults diminui em todas as políticas e em todos os logs. Isso ocorre porque, com uma memória maior, há espaço suficiente para armazenar a maioria das páginas necessárias, reduzindo a necessidade de substituições frequentes.

No caso do tamanho de memória fixo e tamanho de página variável, o algoritmo LRU se destaca, especialmente para tamanhos de página menores, resultando em uma baixa quantidade de page faults. No entanto, à medida que o tamanho da página se aproxima do tamanho da memória, todos os algoritmos apresentam um aumento no número de page faults. Esse comportamento era esperado, pois quanto maior a página, menos páginas podem ser armazenadas simultaneamente na memória, levando a um maior número de substituições.

Esses resultados enfatizam a importância de escolher adequadamente o tamanho da página e da memória, bem como a política de substituição de páginas, para otimizar o desempenho do sistema de memória virtual. É essencial considerar as características do programa em execução e as limitações do hardware para tomar decisões adequadas sobre a configuração do sistema.

Em resumo, o simulador de memória virtual desenvolvido neste projeto proporcionou uma compreensão mais aprofundada sobre o impacto do tamanho da página, tamanho da memória e política de substituição de páginas no desempenho do sistema. Os testes realizados demonstraram as tendências esperadas e destacaram a importância de ajustar esses parâmetros de acordo com as necessidades específicas do sistema em questão.