Análise do Desempenho da TLB em Sistemas com Localidade de Memória

Aluno: Ronaldd Feliph Matias Costa
Matrícula: 122110574
Disciplina: Sistemas Operacionais
Professor: Thiago Emmanuel
Período: 2024.2

1. Introdução

Paginação é uma técnica de gerenciamento de memória amplamente utilizada em sistemas operacionais modernos. Nela, a memória virtual de um processo é dividida em blocos de tamanho fixo chamados de páginas, enquanto a memória física é organizada em quadros de página (page frames), como ilustrado na figura 1 e figura 2. Cada página virtual é mapeada para um quadro físico por meio de uma Tabela de Páginas (Page Table - PT). Assim, ao acessar um endereço de memória, o sistema operacional, com apoio da CPU, consulta essa tabela para localizar a posição correspondente na memória física. Esse mecanismo permite que o sistema operacional implemente, de forma eficiente, três funções essenciais: proteção de memória entre processos, alocação dinâmica e flexível de espaço, e tradução dos endereços virtuais para físicos.

	Memória Fisica	Quadro de Página
0	reservado para o SO	quadro de página 0 da memória fisica
16	(não utilizado)	quadro de página 1
32	(não utilizado)	quadro de página 2
48	página 0 do ES	quadro de página 2
80	página 2 do ES	quadro de página 4
80	(não utilizado)	quadro de página 5
112	página 1 do ES	quadro de página 7

SO = Sistema Operacional
ES = Espaço de endereçamemeto (do processo)

Figura 1: Divisão de um processo utilizando Paginação, representado no livro Three Easy Pieces.

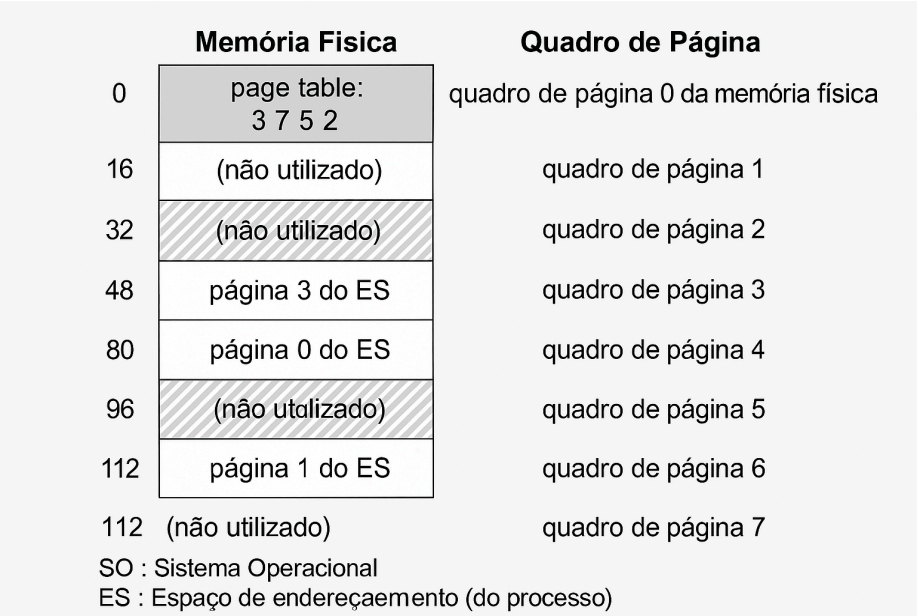

Figura 2: Page Table dividida em page frames, ilustrada no livro Three Easy Pieces.

O emprego da paginação traz diversas vantagens. Em primeiro lugar, cada página pertence a um processo específico, e o hardware fica responsável por validar acessos ilegais ou tentativas de acesso a páginas não pertencentes ao processo, garantindo assim a devida proteção. Além disso, as páginas podem ser carregadas em qualquer local da memória física, dispensando a necessidade de contiguidade e proporcionando uma alocação mais flexível. Por fim, cada acesso a um endereço virtual passa por um mecanismo único de localização do respectivo page frame, o que confere ao sistema uma forma de tradução unificada e facilita o gerenciamento de memória em ambientes multitarefa.

Por outro lado, a necessidade de consultar a Page Table em cada acesso pode introduzir uma sobrecarga, pois esse processo exige leituras adicionais na memória principal (que é significativamente mais lenta). Para amenizar esse problema, muitas arquiteturas utilizam um Translation Lookaside Buffer (TLB), que atua como memória cache do Memory Management Unit (MMU) para armazenar mapeamentos recentes de páginas para page frame, acelerando a busca e reduzindo a latência de tradução.

No decorrer deste trabalho, investigaremos como a TLB influencia o desempenho de sistemas de paginação e mostraremos que, independentemente do tamanho da entrada da TLB, o desempenho tende a se estabilizar. Chega-se, portanto, a um ponto em que aumentar o

tamanho da TLB não traz ganhos significativos, pois o desempenho permanece praticamente o mesmo. Para demonstrar esse comportamento, será realizada uma simulação em que um traço de acessos à memória com localidade é executado sobre diferentes tamanhos de TLB, medindo-se a taxa de acertos (*hit ratio*) em cada caso para observar a variação de desempenho ao longo dos testes.

2. Metodologia

A metodologia adotada para evidenciar o impacto da TLB no desempenho de um sistema de paginação consiste na realização de um experimento por simulação, cujos principais passos são:

2.2 Geração de um trace de acesso

Para analisar o impacto do tamanho da TLB no desempenho de sistemas de paginação, é essencial utilizar um *trace* de acessos à memória — isto é, uma sequência ordenada de endereços ou páginas que simula o comportamento real de programas em execução. Isso se justifica porque o desempenho da TLB depende diretamente dos padrões de acesso à memória, especialmente da presença de localidade temporal e espacial. Um *trace* bem construído permite reproduzir essas características de forma controlada, possibilitando a avaliação precisa da eficiência da TLB em diferentes configurações. Neste projeto[1], optamos por gerar esse *trace* de forma sintética e simplificada, representando cada página como um número inteiro correspondente ao seu identificador lógico.

O processo de geração do *trace* começa com a definição de três parâmetros principais, que controlam o comportamento e a distribuição dos acessos simulados:

- **Número total de acessos**: define quantos acessos serão gerados na sequência. Por exemplo, ao escolher 20.000, o algoritmo criará uma lista com exatamente 20.000 páginas acessadas.

- **Número máximo de páginas distintas**: determina o intervalo de identificadores de páginas que poderão ser acessadas. Um valor como 2.000 indica que os acessos estarão restritos ao intervalo de páginas numeradas de 0 a 1999, simulando um espaço

[1] https://github.com/RonalddMatias/SO-project/blob/main/trace_generator.py

de endereçamento virtual limitado.

- **Probabilidade de localidade**: controla a chance de que o próximo acesso esteja próximo do anterior. Um valor como 0,95 faz com que, em 95% das vezes, o algoritmo escolha uma página vizinha à última acessada, simulando localidade temporal e espacial — ou seja, padrões típicos de comportamento de programas reais, que frequentemente acessam os mesmos dados ou dados adjacentes na memória.

Esses parâmetros possibilitam o controle fino do grau de localidade do *trace*, permitindo gerar sequências que se aproximam dos padrões reais de acesso à memória, essenciais para testar o comportamento da TLB sob diferentes condições.

Inicialmente, a primeira página acessada é escolhida aleatoriamente dentro do intervalo definido, garantindo que o início da sequência não tenha nenhum viés específico. A partir do segundo acesso, a lógica aplicada em cada iteração depende da probabilidade de localidade. Caso um número aleatório entre 0 e 1 seja inferior à probabilidade estipulada, o novo acesso é gerado com base em um pequeno deslocamento (positivo ou negativo) em relação à última página acessada, normalmente entre -2 e +2. Esse comportamento simula localidade espacial, pois mantém os acessos dentro de uma região próxima da memória virtual. Por outro lado, caso o número gerado seja superior ou igual à probabilidade definida, o acesso será feito a uma página completamente aleatória, reforçando a diversidade do traço e garantindo que nem todos os acessos sejam previsíveis.

Conforme essa lógica é aplicada repetidamente, os identificadores de página são armazenados em sequência dentro de uma lista. Ao final do processo, essa lista constitui o *trace* de acessos completo, contendo exatamente o número de elementos definido inicialmente. O resultado é uma simulação de acessos à memória que incorpora características realistas de comportamento de programas — isto é, acessos intensivos a páginas recentes ou próximas, intercalados com saltos aleatórios ocasionais.

Embora esse modelo seja simplificado, ele é suficiente para capturar os efeitos práticos da localidade na performance de estruturas de cache como a TLB. Utilizar um traço gerado de forma controlada e reprodutível permite, além disso, aplicar a mesma sequência de acessos em experimentos com diferentes tamanhos de TLB, possibilitando comparações consistentes e justas entre os resultados obtidos.

2.3 Implementação da TLB

Para realizar os experimentos, foi desenvolvida uma implementação[2] simples de uma TLB, com tamanho configurável de acordo com o objetivo da simulação. A TLB foi estruturada para funcionar como uma memória cache de acessos rápidos, armazenando mapeamentos recentes entre páginas virtuais e páginas físicas. A estrutura interna da TLB adota a política de substituição LRU (*Least Recently Used*), que garante que, ao atingir sua capacidade máxima, o elemento menos recentemente utilizado seja descartado para dar lugar a uma nova entrada, todavia, isso é totalmente configurável, como visto em sala de aula, existem várias políticas quando a TLB está na sua capacidade máxima. Essa política foi escolhida por ser amplamente utilizada em sistemas reais, justamente por sua capacidade de se adaptar bem aos padrões de localidade temporal presentes na maioria das aplicações.

Internamente, a TLB foi representada por meio de uma estrutura de dados que mantém a ordem dos acessos, permitindo tanto consultas eficientes quanto a atualização da posição dos elementos conforme eles são utilizados. Dessa forma, sempre que uma nova página virtual é acessada, verifica-se se ela já se encontra na TLB. Caso esteja, considera-se um *hit*, e a página é movida para a posição mais recentemente usada. Se a página não estiver presente, ocorre um *miss*, e a TLB é atualizada com a nova entrada, respeitando o tamanho máximo estabelecido e aplicando a substituição LRU, caso necessário.

A simulação contabiliza cada acesso realizado e registra se ele foi atendido pela TLB (hit) ou exigiu uma consulta à tabela de páginas (miss). Ao final de cada execução, é possível calcular a **Taxa de Hits** (*hit ratio*) para um determinado tamanho de TLB, o que permite avaliar quantitativamente o impacto do seu tamanho na eficiência da tradução de endereços. Essa implementação, ao mesmo tempo simples e funcional, permite conduzir experimentos com diferentes configurações de TLB e serve de base para a análise dos resultados obtidos.

$$\text{Hit Ratio} = \frac{\text{Número de Hits}}{\text{Número de Hits} + \text{Número de Misses}}$$

Figura 3: Fórmula do Hit Ratio.

[2] https://github.com/RonalddMatias/SO-project/blob/main/tlb.py

2.4 Variação do Tamanho da TLB

Com o traço de acessos previamente gerado, a simulação foi executada diversas vezes, cada uma com um tamanho distinto de TLB. O objetivo dessa variação é observar como o desempenho da TLB se comporta à medida que sua capacidade de armazenamento aumenta. Para isso, foram testados tamanhos progressivos, iniciando em valores baixos, como 4 e 8 entradas, e avançando gradualmente para tamanhos maiores, chegando a centenas ou até milhares de entradas. Essa abordagem permite analisar se há, de fato, um ponto a partir do qual o aumento da TLB deixa de trazer ganhos significativos de desempenho.

Em cada execução da simulação, a métrica observada foi a **Taxa de Hits** (*hit ratio*), que corresponde à proporção de acessos à memória virtual que foram resolvidos diretamente pela TLB, sem a necessidade de consultar a tabela de páginas. Essa métrica é fundamental para avaliar a eficiência da TLB, pois indica o quanto da carga de trabalho foi atingida de forma rápida, evitando a sobrecarga associada ao processo de tradução convencional. Ao manter o mesmo traço para todas as simulações, garantimos uma base comparativa justa entre os diferentes tamanhos de TLB, isolando o impacto que essa variável tem sobre o desempenho final

3. Resultados e Discussão

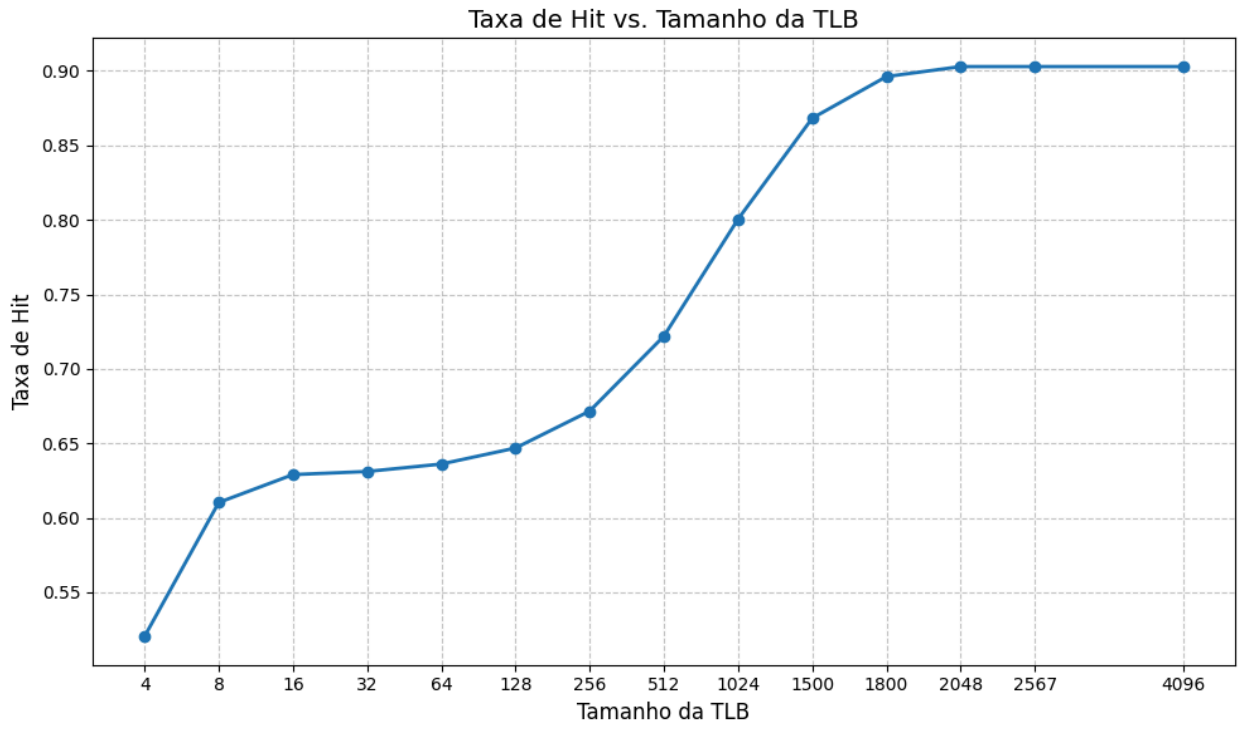

Figura 03: Gráfico comparativo entre a taxa de hit e o tamanho da TLB.

Após a implementação da TLB e a geração do traço de acessos com localidade, foi possível simular o comportamento da taxa de hits (*hit ratio*) para diferentes tamanhos de TLB. O objetivo deste experimento foi verificar até que ponto o aumento da capacidade da TLB contribui para o desempenho do sistema e identificar, se existente, um ponto de saturação em que o crescimento do número de entradas deixa de impactar significativamente a taxa de acertos.

O gráfico acima ilustra a variação da taxa de hits em função do tamanho da TLB. Como esperado, para tamanhos muito pequenos — como 4, 8 ou 16 entradas — a TLB é incapaz de reter uma quantidade significativa de mapeamentos ativos, o que resulta em uma baixa taxa de acertos. No entanto, à medida que o tamanho da TLB aumenta, a quantidade de hits cresce consideravelmente, evidenciando que a estrutura passa a armazenar mais páginas frequentemente acessadas, aproveitando melhor a localidade temporal e espacial do traço.

Contudo, essa melhora não é indefinida. Observa-se que, a partir de um certo ponto — em geral, após 2048 entradas, dependendo da configuração do traço — a curva de crescimento da taxa de hits começa a se estabilizar. Esse comportamento é conhecido como o "**joelho da curva**", e indica o **ponto de saturação** da TLB. A partir desse momento, mesmo dobrando o número de entradas, o ganho em desempenho torna-se marginal, pois a maioria dos acessos já está sendo atendida pela TLB existente. Isso ocorre porque, devido à localidade dos acessos, há um conjunto limitado de páginas que são reutilizadas com frequência. Portanto, uma TLB que consiga armazenar esse conjunto ativo já é suficiente para atender à maior parte das requisições.

4. Conclusão

Os resultados demonstraram que, embora o aumento do tamanho da TLB inicialmente leve a uma elevação acentuada na taxa de acertos, esse ganho tende a se estabilizar a partir de certo ponto — conhecido como ponto de saturação. A partir desse limiar, aumentar o número de entradas na TLB gera ganhos marginais, pois a maioria dos acessos já está sendo atendida pela estrutura existente.

Conclui-se, portanto, que a escolha do tamanho da TLB deve considerar os padrões de acesso da aplicação, já que uma TLB moderadamente dimensionada é suficiente para aproveitar a localidade e obter um desempenho próximo do ideal.

5. Referências

[1] ARPACI-DUSSEAU, Remzi H.; ARPACI-DUSSEAU, Andrea C. *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, 2018. Disponível em: https://pages.cs.wisc.edu/~remzi/OSTEP/. Acesso em: abr. 2025.

[2] ANDERSON, Thomas; DAHLIN, Michael. *Operating Systems: Principles and Practice*. 2nd ed. Recursive Books, 2014. Acesso em: abr. 2025.

[3] PYTHON SOFTWARE FOUNDATION. *The Python Language Reference*. Disponível em: https://docs.python.org/3/. Acesso em: abr. 2025.

[4] HUNTER, John D. *Matplotlib: A 2D Graphics Environment*. Computing in Science & Engineering, v. 9, n. 3, p. 90–95, 2007. Disponível em: https://matplotlib.org/. Acesso em: abr. 2025.