Trabalho Prático - Sistemas Operacionais
Simulação de um sistema de memória virtual

Beatriz Reis, Izadora Ganem, e Rafael Martins

Univerisdade Federal de Minas Gerais

Este relatório contempla um resumo da implementação do simulador de memória virtual, as decisões de projeto da equipe e uma análise do desempenho dos algoritmos de subtituição de páginas e dos diferentes tipos de tabela de páginas.

1 Estrutura geral

O projeto é organizado da seguinte maneira:

```
project-root/
 bin/
 include
     dense.h
     hier2.h
     hier3.h
     inverted.h
     mem_address.h
 obj/
 src/
     dense.c
     hier2.c
     hier3.c
     inverted.c
     main.c
     mem_address.c
 Makefile
```

Os arquivos `dense.c, hier2.c, hier3.c` e `inverted.c` contêm a implementação de cada tipo de tabela de páginas, enquanto os arquivos .h contêm sua definição. Em cada arquivo, existem 4 funções, uma para cada algoritmo de reposição de páginas, nomeadas `table-type_algorithm`.

O arquivo `mem_address.c` contém a implementação de algumas estruturas comuns a todos os algoritmos, definidas para auxiliar na implementação dos algoritmos. São elas:

- Struct `mem_address`:

```
struct mem_address {
    unsigned addr;
    unsigned first;
    unsigned second;
    unsigned third;
    unsigned time;
    unsigned dirty;
    char rw;
};
```

A estrutura `mem_address` representa o endereço de memória lido, com seus atributos:

- **addr**: endereço;
- **first**: primeiro "trecho" do endereço após divisão dos bits;
- **second**: segundo "trecho" do endereço após divisão dos bits;
- **third**: terceiro "trecho" do endereço após divisão dos bits;
- **dirty**: bit que indica se a página foi ou não suja, isto é, se foi realizada sobre ela uma operação de escrita (do tipo 'W').
- **rw**: caractere que indica a operação realizada (escrita, 'W', ou leitura, 'R').

- Struct `page_time`:

```
struct page_time {
    unsigned page;
    unsigned first;
    unsigned second;
    unsigned third;
    unsigned time;
};
```

A estrutura `page_time` representa o endereço que está atualmente na memória, com seus atributos:

- **page**: endereço;
- **first**: primeiro "trecho" do endereço após divisão dos bits;
- **second**: segundo "trecho" do endereço após divisão dos bits;
- **third**: terceiro "trecho" do endereço após divisão dos bits;
- **time**: momento em que o endereço foi adicionado à memória ou em que é acessado (a depender o algoritmo escolhido).

2 Decisões de projeto

Implementação

Na implementação dos algoritmos, todas as tabelas são vetores do tipo `mem_address`, para permitir o controle dos atributos de cada endereço. Além disso, para as tabelas `densa, hierárquica 2 níveis` e `hierárquica 3 níveis`, foi criado um vetor do tipo `page_time` para controlar o tempo das páginas que estão atualmente na memória e otimizar a execução do programa.

A execução de todos os algoritmos é orientada pela mesma lógica, cuja implementação segue as especificidades de cada algoritmo e tabela de páginas:

As demais estruturas foram explicadas no próprio código.
Para cada linha do arquivo de entrada:

1. Leitura do endereço `addr` e da operação `rw` realizada;
2. Verificação da tabela: se `addr` está na memória, acontece um hit e terminamos a iteração;

3. Verificação da tabela: se ainda existem espaços vazios, acontece um page fault. Incluímos `addr` na tabela e terminamos a iteração;

4. Determinação de um endereço `choice` a ser escrito de volta na memória, já que o endereço não está na memória e não existem espaços vazios, ou seja, acontece um page fault. Atualizamos a tabela, retirando `choice` e incluindo `addr` na tabela.

Além disso, toda vez é lida uma operação de escrita, o bit `dirty` do endereço é marcado como 1. Sempre que um endereço for escrito de volta na memória e ele estiver marcado como `dirty`, é incrementada a quantidade de páginas escritas.

Estratégias de robustez

- Erro nos argumentos: interrupção da execução e exibição de mensagem de erro.
- Entrada sem operação, apenas com endereço: a execução segue normalmente, porém não serão contadas páginas sujas, uma vez que a operação é desconhecida.

As demais decisões foram explicadas no próprio código.

3 Análise de desempenho

A análise de desempenho foi realizada considerando o comportamento de cada algoritmo de substituição de páginas em diferentes condições. Foram avaliados o número de falhas de página (*page faults*) e o tempo de execução para diferentes tamanhos de memória e tipos de tabela de páginas. Essa abordagem permitiu identificar os trade-offs entre eficiência de substituição e custo computacional.

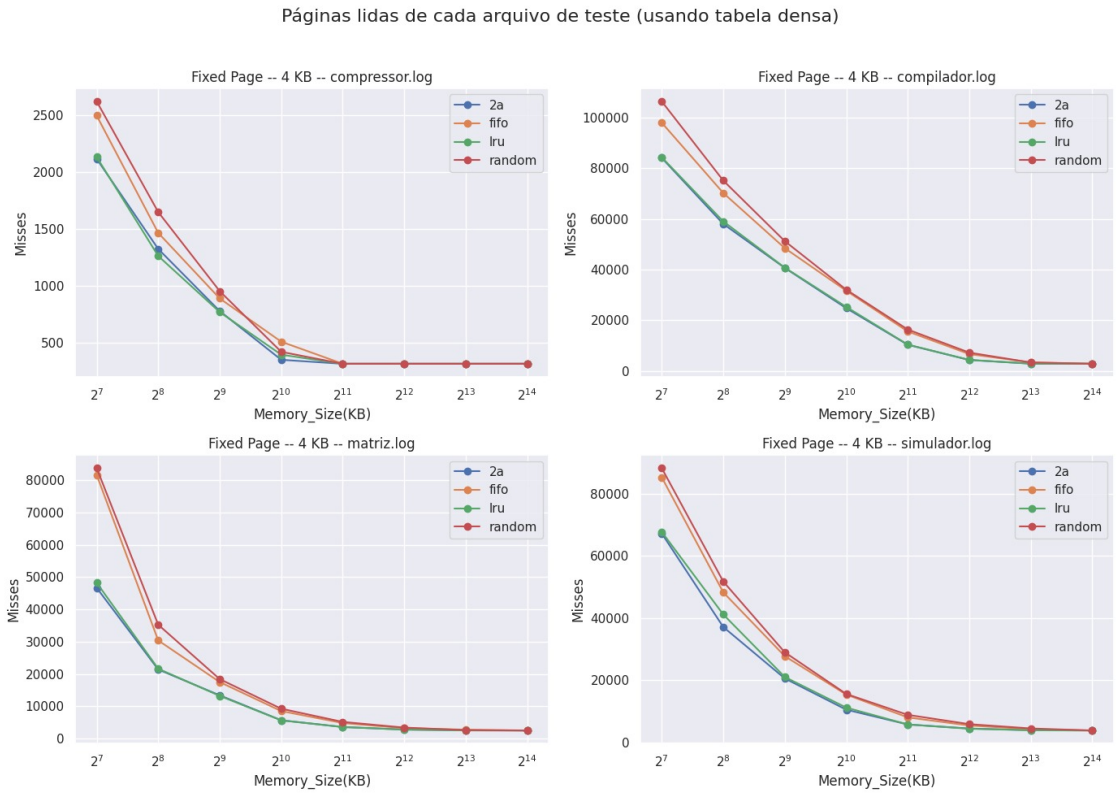

Figura 1: Páginas lidas de cada arquivo (tabela densa)

Os gráficos da Figura 1 exibem uma comparação entre o número de page faults por método de substituição, considerando um tamanho fixo de 4KB para as páginas. Nesse sentido, é possível observar que os gráficos refletem o esperado de acordo com os estudos em sala de aula: o método aleatório, por não contar com critério para minimizar o número de misses, tende a ter mais page faults. O método FIFO também não se mostrou otimizado, ao contrário dos métodos LRU e segunda chance, que, por contarem com substituições de página que buscam antecipar tentativas de uso da memória, conseguem diminuir o número de misses obtido.

É interessante observar também que, com o aumento da memória, a diferença entre os métodos de substituição passa a ser insignificante, tendo em vista que há espaço suficiente para que menos falhas de página aconteçam.

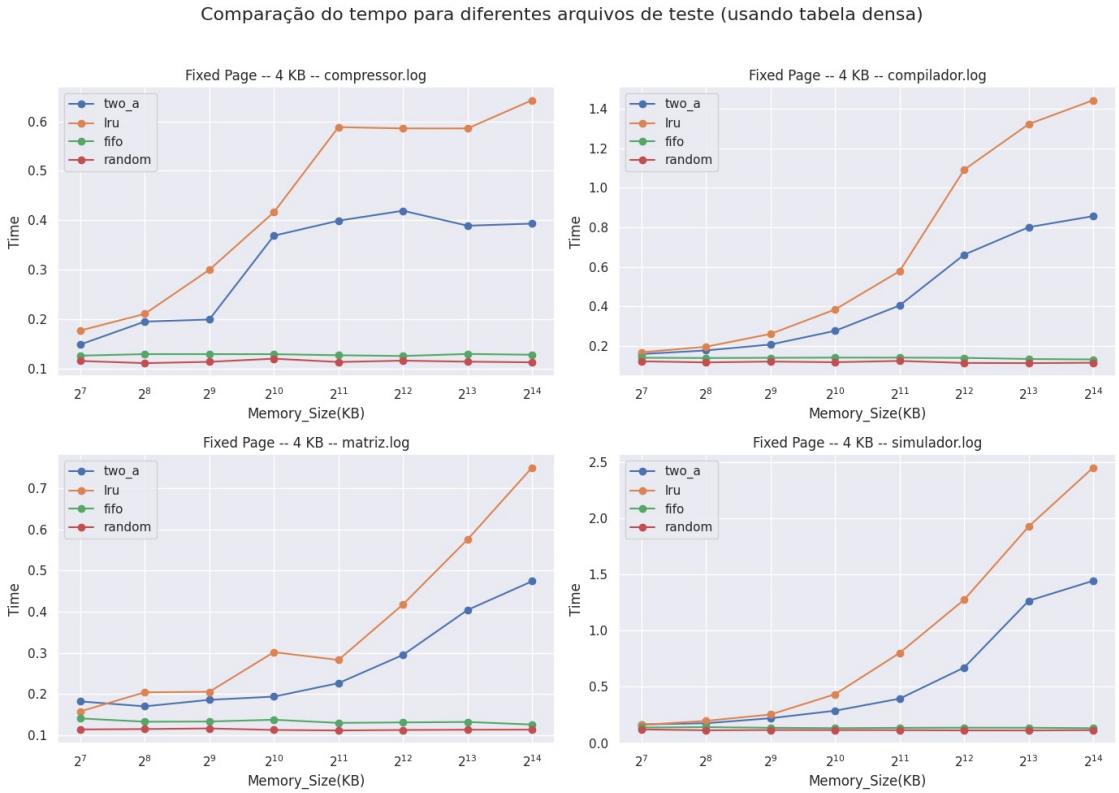

Figura 2: Comparação entre arquivos (tabela densa)

A Figura 2, por sua vez, evidencia as diferenças de desempenho entre os algoritmos. Observa-se que, de maneira inversa ao que ocorre com o número de page faults, o tempo de execução dos algoritmos LRU e segunda chance é consideravelmente maior que do FIFO e aleatório, que se mantêm com um tempo curto independentemente do tamanho de memória utilizada. Esses resultados evidenciam o trade-off entre tempo de execução e falhas de página: algoritmos com lógicas mais complexas para maximizar o número de hits obtidos tendem a ser mais custosos, enquanto aqueles mais rápidos sofrem com um maior número de misses.

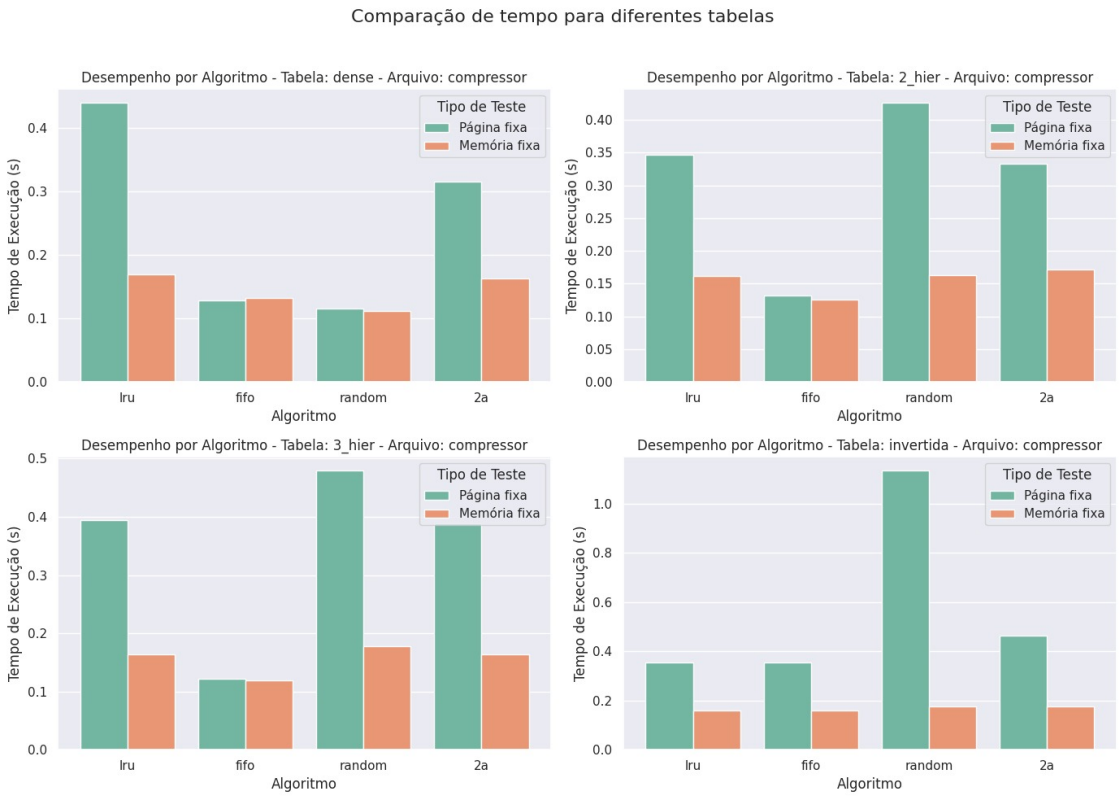

Figura 3: Comparação de tempo para diferentes tabelas

Finalmente, a Figura 3 compara o desempenho dos métodos de substituição em diferentes tipos de tabelas (densa, hierárquica de 2 e 3 níveis, e invertida).

Na tabela densa, LRU foi o mais lento, enquanto FIFO e aleatório tiveram os melhores tempos, com segunda chance em posição intermediária. Na tabela hierárquica de 2 níveis, o padrão foi semelhante, com LRU novamente como o mais lento e FIFO e aleatório mais rápidos. Na hierárquica de 3 níveis, o comportamento se manteve, com LRU atrás dos demais e FIFO e aleatório como os melhores. Na tabela invertida, o algoritmo aleatório teve um desempenho atípico com o tamanho da página fixa, com tempo elevado, enquanto FIFO e Segunda Chance foram as opções mais rápidas.

Em resumo, é possível observar que, mesmo com as vantagens supracitadas, LRU é consistentemente o mais lento, enquanto FIFO se mostrou o mais rápido e confiável na maioria dos casos.

4 Conclusão

Este trabalho nos permitiu implementar um simulador de memória virtual e explorar diferentes algoritmos de substituição de páginas e tipos de tabelas. Observou-se o trade-off entre desempenho em tempo de execução, além do número de falhas de página para cada combinação testada.

Por fim, o trabalho nos forneceu uma base prática para o entendimento as políticas gerenciamento de memória, uma vez que os resultados obtidos confirmaram a teoria discutida em sala.

5 Instruções de compilação e execução

O diretório contém um Makefile, conforme instruído nas especificações. Portanto, para compilar o programa, basta digitar `make` na linha de comando.

Para executar o programa, basta digitar na linha de comando:

```
./bin/tp2virtual <algorithm> <arquivo> <page-size> <mem-size> <table-type> <debug>
```

Onde os parâmetros são:

- algorithm: A política de substituição escolhida. As opções são:
  - **fifo**
  - **lru**
  - **random**
  - **2a** (segunda chance)
- arquivo: A entrada sendo testada. Basta incluir o nome do arquivo.
- page-size: Tamanho da página. Deve ser fornecido em KB.
- mem-size: Tamanho total da memória física disponível. Deve ser fornecido em KB.
- table-type: Tipo de tabela de páginas escolhida. As opções são:
  - **inverted**
  - **dense**
  - **2hierarchical** (tabela hierárquica com 2 níveis)
  - **3hierarchical** (tabela hierárquica com 3 níveis)
- debug: Parâmetro opcional que determina a saída em modo de depuração. Para ativar a saída mais detalhada, deve ser apenas a letra `d`.

Referências

[1] Second chance algorithm. Disponível em: `https://www.geeksforgeeks.org/second-chance-or-clock-page-replacement-policy/`

[2] Slides da disciplina Sistemas Operacionais (2024/2).