Gerencia de Memória: Paginação com Algoritmos de Substituição LRU e Aleatório

Israel Deorce Vieira Júnior[1]

[1] Faculdade de Informática – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre, RS - Brazil

{israel.deorce@acad.pucrs.br}

Resumo. *Este relatório descreve uma alternativa de implementação para o exercício proposto no segundo trabalho da disciplina de Sistemas Operacionais. O exercício envolve o desenvolvimento de um programa que simule um gerenciador de memória utilizando técnica de paginação, com algoritmos LRU (Menos Recentemente Utilizado) e aleatório como algoritmos de substituição de páginas. Desenvolvemos a solução proposta utilizando a linguagem de programação Java orientada a objetos, com auxílio de uma estrutura de tabela de páginas. Apresentaremos a solução de forma textual explicativa, acompanhada de pseudocódigo e imagens ilustrativas.*

1. Introdução

A memória de um computador é um recurso importante, e um bom sistema operacional precisa gerenciá-lo com bastante cuidado para proporcionar um melhor desempenho e maior espaço pelo menor custo. Por isso, é comum que sistemas operacionais modernos intercalem processos entre a memória principal e o disco através da técnica de Swapping. Além disso, através da técnica de Paginação, é possível organizar os espaços da memória virtual em páginas, e os espaços da memória física em frames, permitindo que processos sejam divididos em partes e que somente as partes utilizadas ocupem espaço nas memórias mais rápidas.

A atividade proposta consiste na implementação de um programa que simula a falta de páginas ($PageFault$) de processos em um sistema operacional, que ocorre quando um programa tenta acessar uma página que está na memória virtual e, por isso, precisa ser trazido para a memória física. O programa a ser implementado deve ler um arquivo de entrada com instruções que construirão o sistema. Os arquivo no formado de texto foram organizados com os seguintes campos:

1. Tipo de leitura do arquivo: O tipo de leitura pode ser de duas formas:
   (a) $sequencial$: As leituras dos comandos do arquivo devem ser feitas na sequência em que elas aparecem no arquivo;
   (b) $aleatorio$: Os processos criados deverão executar de forma concorrente através de threads, intercalando de forma randômica entre os comandos de acesso à uma página e alocação de memória extra.
2. Tipo de algoritmo de troca: lru (Menos Frequentemente Usado) ou $aleatorio$;
3. Tamanho das páginas: Valor inteiro que representa o tamanho fixo das páginas do sistema;
4. Tamanho da memória física: Valor inteiro múltiplo do tamanho das páginas;

5. Tamanho da memória virtual: Tamanho da área para armazenamento das páginas em disco;
6. Sequência de N comandos relativos à processos Px específicos, e uma posição ou quantidade T de espaço na memória. Cada um desses comandos pode vir compostos das seguintes formas:
   (a) C: Cria um processo Px de tamanho T
   (b) A: Acessa um processo Px na memória na sua subdivisão de valor T
   (c) M: Aloca um valor T de memória extra para um processo Px

Neste trabalho, buscaremos demonstrar o nosso entendimento alcançado durante a atividade, descrevendo o conceito de paginação, dos algoritmos de substituição de páginas "LRU"e "Aleatório", a solução encontrada e os resultados obtidos enviados em anexo no formato de arquivos de texto.

2. Paginação e Swapping

O método básico de se implementar páginas envolve quebrar a memória principal em blocos de tamanho fixo chamados de frames, e quebrar a memória virtual (memória do disco) em blocos do mesmo tamanho chamados de páginas [Silberschatz et al. 1991].

Quando processos são criados, é dado a eles um número de páginas necessário para comportar todo o programa. Para que um processo seja executado de maneira eficiente, é necessário que as páginas do mesmo que serão executadas estejam alocadas na memória física. Por isso, sempre que for feito um acesso à uma página ou uma alocação extra de memória para um determinado processo, se este processo estiver na memória virtual, temos uma situação de $PageFault$ e será preciso realizar o processo de $Swapping$.

O Swapping é uma técnica que transfere páginas ou programas do disco para a memória principal e vice-versa. Quando acontece um acesso à uma determinada página, ou uma alocação de memória extra para um determinado processo, as páginas referenciadas devem ser trazidas para a memória física, pois estas têm maior chance de serem utilizadas novamente.

Na nossa solução implementada em linguagem Java, definimos a classe $Pagina$, que implementa o método $swap$ para uma página Pa. Este método recebe por parâmetro as referências para as memórias física e virtual, além do novo índice que é a primeira posição onde se encontra a página Pb a qual trocará de posição com Pa. De posse dessas informações, o swap das páginas pode ser feito através de um loop simples, descrito a seguir:

```
1 procedimento swap
2   para i ← 0 ate < quantidade de índices na memoria faca
3     memNova[novoIndice + i] ← memAntiga[indicesAntigos[i]]
4     memAntiga[indicesAntigos[i]] ← null
5     indicesNovos[i] ← novoIndice + i
6   fim para
7   alteramos o bit que informa em qual memoria a página está
8   adicionamos a outra página na memória antiga
9 fim
```

3. Algorimo de Substituição da página menos recentemente usada (LRU ou MMU)

Ueyama explica em sua vídeo-aula, que a ideia do algoritmo LRU é que as páginas muito usadas ultimamente provavelmente serão usadas novamente nos próximos momentos [Uivesp 2017]. Portanto, este algoritmo prioriza a troca de páginas (remoção de uma página da memória principal) da página que permaneceu em desuso pelo maior tempo.

Normalmente, na implementação deste algoritmo deve-se manter uma lista encadeada de todas as páginas na memória, mantendo a página que foi mais recentemente usada no início da lista, e a menos recentemente usada no final desta lista. Contudo, este é um processo custoso e exige atualização constante da lista.

Existem outras maneiras de se implementar este algoritmo. Uma dessas maneiras é utilizando uma tabela de páginas que pode ser implementada a nível de hardware, como em uma unidade de gerenciamento de memória (MMU). Tanenbaum et. al. descreve esta solução:

- *"Contudo, existem outras maneiras de se implementar o algoritmo MRU com auxílio de hardware especial. Vamos considerar o modo mais simples primeiro: ele requer que o hardware seja equipado com um contador de 64 bits, que é incrementado automaticamente após a execução de cada instrução. Além disso, cada entrada da tabela de páginas deve ter um campo extra para armazenar o valor do contador. Após cada referência à memória, o valor atual do contador é armazenado nesse campo adicional da entrada da tabela de páginas correspondente à página que acabou de ser referenciada. Quando ocorre uma falta de página, o sistema operacional examina todos os campos contadores da tabela de páginas a fim de encontrar o menor deles. A página correspondente a esse menor valor será a 'menos recentemente usada'".*

  [Tanenbaum 2003]

O maior problema do algoritmo LRU, é que pode acontecer de páginas de processos que foram frequentemente referenciadas no início, e que porém, pararam de serem referenciadas após um determinado tempo, acabam ficando na memória física por um tempo maior do que deveriam, já que elas provavelmente não serão mais referenciadas.

4. Tabela de Páginas

A tabela de páginas é uma estrutura de dados que visa auxiliar o mapeamento das páginas presentes na memória virtual e na memória física. Cada processo presente em um sistema operacional possui as informações de suas páginas armazenas em uma dessas tabelas. Para facilitar o nosso trabalho, o processo de swapping e de execução dos algoritmos de substituição de páginas, associamos para cada página na tabela de páginas (Figura 1), os seguintes componentes:

1. Bit de Residência: Se o campo contiver o valor 1, então a página está presente na memória RAM. Se o valor for 0, significa que a página se encontra alocada na memória virtual.
2. Índice na memória: A tabela também armazena o local exato inicial (posição zero) de uma página na memória. O índice válido será o apresentado na memória cujo o Bit de Residência indicar.

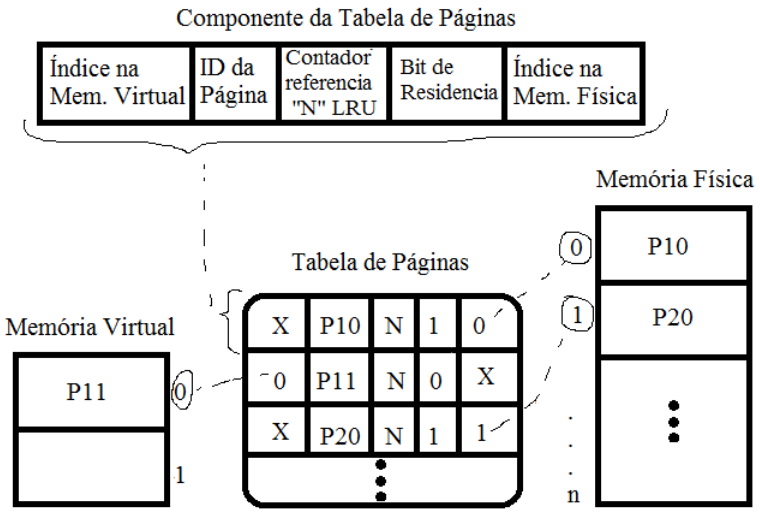

Figura 1. Tabela de Páginas e o mapeamento das páginas

3. ID da página: código identificador da página, que também identifica o processo a qual esta página pertence. Por exemplo, a página de "ID"$P10$ pertence ao processo $P1$ e é a página de número 0, enquanto a página de "ID"$P11$ também pertence ao processo $P1$, porém é a página de número 1.
4. Contador LRU: Contador para auxiliar o algoritmo de substituição LRU, cujo valor é incrementado após cada referência à memória.

No nosso programa, cada vez que uma página é referenciada na memória, a tabela de páginas é consultada para descobrir se será necessário realizar o processo de swap e, dependendo do algoritmo de substituição que escolhermos utilizar, os campos expostos na tabela nos ajudarão na escolha da página correta para substituir. No caso do LRU, a página escolhida para ir para a memória física será aquela cujo bit de residência indicar 1 e o valor do contador de referência for o menor dentre todas as páginas da tabela.

5. Algorimo de Substituição de Páginas Aleatório Simulação com Threads

O Algoritmo de Substituição de Páginas Aleatório, faz o swap de uma página randômica na memória quando acontece um $PageFault$. O maior benefício deste algoritmo é que ele elimina a necessidade de se manter a referência de páginas. Porém, ele pode acabar removendo da memória, páginas importantes para melhoria dos acessos, o que pode gerar uma instabilidade na quantidade de $PageFaults$.

Para ajudar na simulação do programa, utilizamos a técnica de $Threads$. Cada processo foi associado à uma $Thread$ que intercala entre acesso e alocação de memória extra à uma página na memória. Para garantir a sincronização e a não existência de $deadlocks$, utilizamos técnicas de programação típicas da linguagem Java, como a inclusão da palavra-chave $synchronized$ para os métodos, e o método $join()$, que obriga o sistema a esperar o final da execução da Thread para continuar a sua execução normal [Oracle 2016].

6. Conclusão

Este relatório apresentou os resultados obtidos na realização da atividade de desenvolvimento de um programa que simula um gerenciador de memória utilizando técnica de paginação, com algoritmos LRU (Menos Recentemente Utilizado) e aleatório como algoritmos de substituição de páginas. Também, descrevemos alguns conceitos paginação, swapping, simulados de forma sequêncial e aleatória com o auxílio de threads e tabelas de páginas. Os resultados obtidos na execução da solução descrita, foram anexados à entrega deste relatório em formado $.txt$. Com isso, é plausível afirmar que, houve uma considerável melhora em nossos conhecimentos da disciplina de Sistemas Operacionais, mais especificamente na parte de gerenciamento de memória e técnica de paginação.

Referências

Oracle (2016). *Java API - Comparator*.

Silberschatz, A., Peterson, J. L., and Galvin, P. B. (1991). *Operating System Concepts*. Wiley, 9th edition.

Tanenbaum, A. S. (2003). *Sistemas Operacionais Modernos*. Prentice-Hall do Brasil, 2th edition.

Uivesp, J. U. (2017). *Curso de Sistemas Operacionais para Engenharia da Computação; Disponível em: https://www.youtube.com/watch?v=Q8Zqj EafmN-clist=PLxI8Can9yAHeK7GUEGxMsqoPRmJKwI9Jwindex=18*.