Implementação De Um Sistema De Gerência de Memória

Felipe Guedes, Mateus Haas

Escola Politécnica– Pontifícia Universidade Católica do Rio Grande do Sul(PUCRS)
Porto Alegre – RS – Brazil

Resumo. *Este artigo descreve a implementação do sistema de gerência de memória implementado pelos alunos da Pontifícia Universidade Católica do Rio Grande do Sul durante o primeiro semestre de 2018. Primeiramente discutiremos o que é o sistema de gerência de memória, em seguida rapidamente abordaremos quais os requisitos do trabalho e o problema proposto, posteriormente abordaremos a chegada a solução. Por fim ocorrerá uma breve exposição dos resultados e algumas reflexões finais na conclusão.*

1. Introdução

A memória de um computador é gerenciada através de um "Gerente de Memória". Segundo Colleta (2018, p. 1) , este é responsável por cuidar de quais partes da memória estão em uso, quais estão disponíveis para serem alocadas por processos quando requisitarem e quais devem ser deslocadas quando não forem mais necessárias, também precisa gerenciar as trocas dos processos entre a memória principal e o disco quando a memória principal preencher o seu espaço disponível.

Tais atribuições tornam crucial a boa implementação de um gerente de memória, pois ele é de extrema importância para a estabilidade e velocidade de um sistema operacional. Duas das formas para se implementar um algoritmo que exerce a gerência de memória serão expostos a seguir.

O mais complexo e pertinente é o algoritmo LRU - *Least Recently Used* - ou em português "menos recentemente usado". Seu nome já é extremamente intuitivo, neste algoritmo a escolha sobre qual página devem ser desalocadas para a alocação em memória principal de uma memória requisitada por um processo recai sobre o tempo que a memória está no processador sem ser acessada. A memória que está a mais tempo sem ser utilizada deve sair para dar espaço para que a página que ela está utilizando seja alocada por um novo processo [Santos 2009, p. 27].

Outro modo de implementar tal solução é o modo aleatório onde não há uma escolha definida por um padrão bem definido como anteriormente. apenas escolhe ao acaso uma página para ser desalocada até que a necessidade do novo processo seja suprida.

2. Problema

Foi pedido para que os alunos implementassem um programa que rodasse com uma combinação de maneiras. O programa poderia rodar de forma manual ou aleatória, na forma manual seria fornecido um arquivo que segue um padrão estipulado pelo professor Avelino Zorzo onde seriam definidas as seguintes informações, o modo de execução, o algoritmo a ser utilizado o tamanha da página, o tamanho da memória física, o tamanho da área para armazenamento das páginas em disco e os processos.

Na forma aleatória o algoritmo deve executar de forma pseudo aleatória criando processos, terminando processos e processos requisitando mais memória de através de probabilidades estipuladas no código até que o programa não tenha mais memória ou uma probabilidade do programa acabar seja atingida.

3. Solução

A solução apresentada foi implementada em Java, que segundo Silberschatz (2008, p. 277) é linguagem extremamente difundida e utilizada por alguns sistemas operacionais. Nesta solução foi implementado um gerenciador de memória que é capaz de executar de forma simulada o comportamento de um gerenciador de memória real.

Esta simulação pode ocorrer das duas formas expostas na seção anterior do presente artigo, tanto a forma aleatória como a forma manual foram implementadas. A forma manual ocorre de forma sequencial sem mais necessidade de implementação de paralelismo, já a forma aleatória necessitou a criação de thread que de forma concorrente disputam por espaço de memória.

4. Resultados

Abaixo podemos ver a execução do algoritmos rodando no modo manual utilizando o algoritmo LRU. Está é a execução mais pertinente por esta razão é a que será apresentada neste artigo.

Um exemplo da execução deste algoritmo pode ser vista abaixo, o qual foi rodado com o arquivo de teste fornecido pelo professor e parou sua execução por falta de espaço de memória.

[C p1 16] - OK. Criou normal.
[C p2 18] - OK. Criou normal.
[C p3 10] - OK. Criou normal.
[A p1 14] - OK.
[A p1 20] - Segmentation fault.
[A p2 17] - OK.
[A p3 10] - Segmentation fault.

[M p1 8] - Espaço alocado com sucesso. Criou direto nova página.
[A p1 20] - OK.
[M p3 5] - Espaço alocado com sucesso. Não foi necessário criar página (tinha espaço sobrando).
[M p2 8] - Espaço alocado com sucesso. Criou nova página a partir de SWAP | R [p1(0) p1(1) p2(0) p2(1) p2(2) p3(0) p3(1) p1(2)] D [00 00];R [p2(3) p1(1) p2(0) p2(1) p2(2) p3(0) p3(1) p1(2)] D [p1(0) 00];
[A p1 1] - OK. SWAP necessário. R [p2(3) p1(1) p2(0) p2(1) p2(2) p3(0) p3(1) p1(2)] D [p1(0) 00];R [p2(3) p1(1) p1(0) p2(1) p2(2) p3(0) p3(1) p1(2)] D [00 p2(0)];
[A p2 20] - OK.
[A p1 5] - OK.
[A p1 17] - OK.
[A p1 20] - OK.
[A p2 7] - OK. SWAP necessário. R [p2(3) p1(1) p1(0) p2(1) p2(2) p3(0) p3(1) p1(2)] D [00 p2(0)];R [p2(3) p1(1) p1(0) p2(0) p2(2) p3(0) p3(1) p1(2)] D [p2(1) 00];
[A p2 9] - OK. SWAP necessário. R [p2(3) p1(1) p1(0) p2(0) p2(2) p3(0) p3(1) p1(2)] D [p2(1) 00];R [p2(3) p1(1) p1(0) p2(0) p2(2) p2(1) p3(1) p1(2)] D [00 p3(0)];
[A p2 18] - OK.
[A p2 25] - OK.
[A p3 0] - OK. SWAP necessário. R [p2(3) p1(1) p1(0) p2(0) p2(2) p2(1) p3(1) p1(2)] D [00 p3(0)];R [p2(3) p1(1) p1(0) p2(0) p2(2) p2(1) p3(0) p1(2)] D [p3(1) 00];
[A p3 12] - OK. SWAP necessário. R [p2(3) p1(1) p1(0) p2(0) p2(2) p2(1) p3(0) p1(2)] D [p3(1) 00];R [p2(3) p3(1) p1(0) p2(0) p2(2) p2(1) p3(0) p1(2)] D [00 p1(1)];
[A p1 6] - OK.
[A p1 18] - OK.
[A p1 8] - OK. SWAP necessário. R [p2(3) p3(1) p1(0) p2(0) p2(2) p2(1) p3(0) p1(2)] D [00 p1(1)];R [p2(3) p3(1) p1(0) p1(1) p2(2) p2(1) p3(0) p1(2)] D [p2(0) 00];
[M p2 10] - Espaço alocado com sucesso. Criou nova página a partir de SWAP | R [p2(3) p3(1) p1(0) p1(1) p2(2) p2(1) p3(0) p1(2)] D [p2(0) 00];R [p2(3) p3(1) p1(0) p1(1) p2(2) p2(4) p3(0) p1(2)] D [p2(0) p2(1)];
[A p2 20] - OK.
[M p1 10] - Não tem mais memória.

5. Conclusão

Com a implementação do algoritmo pudemos notar a importância desta parte do sistema operacional para a estabilidade e desempenho de um sistema operacional. O gerenciador de memória de um computador é bastante complexo e necessita lidar com bastante stress e ser tolerante a falhas, o que torna a sua implementação muito delicada, pois cada pequeno erro cometido na lógica contida no software terá um tremendo impacto no sistema operacional.

Foi muito interessante poder implementar o gerenciador de memória para poder compreender melhor seu funcionamento e pôr em prática os conhecimentos adquiridos nas aulas do professor Avelino Zorzo. O trabalho foi considerado bem sucedido pelos alunos por rodar sem falhas e atingir o resultado esperado.

Referências

COLETTA, Alex De Francischi. Gerenciamento de Memória. São Paulo. Disponível <http://alexcoletta.eng.br/artigos/gerenciamento-de-memoria/>. Acesso em: 22 de Junho de 2018.

SANTOS, Douglas. Avaliação Do Comportamento De Sistemas Em Situação De Thrashing. Curitiba. Disponível <https://www.ppgia.pucpr.br/pt/arquivos/mestrado/dissertacoes/2009/2009_douglas_santos.pdf>. Acesso em: 22 de Junho de 2018.

SILBERSCHATZ, Abraham. Sistemas operacionais com Java. Elsevier Brasil, 2008. Disponível <https://books.google.com.br/books?hl=pt-BR&lr=&id=Y8HMA2XngmwC&oi=fnd&pg=PA1&dq=sistemas+operacionais++JAVA&ots=l4H0FB7z19&sig=0460pMmm9vdl9DEuM1uOvbq7Edk#v=onepage&q=sistemas%20operacionais%20%20JAVA&f=false>. Acesso em: 22 de Junho de 2018.