Buffer - Substituição de Páginas FIFO - Implementação e Estatísticas

Nome: Brendon Hudson Cardoso Nº USP: 10284879

Resumo

A proposta do trabalho é implementar e testar uma implementação de um buffer implementando uma das políticas de substituição de páginas descritas no capítulo de Gerenciamento de Memória da bibliografia estudada na disciplina, utilizando técnicas de programação orientada a objetos, multi-threading para melhor performance e políticas de substituição de páginas. O desenvolvimento tem como finalidade gerar relatórios completos com um padrão de saída definido que poderão ser comparados com os relatórios gerados pelo grupo parceiro de turma que em paralelo com o auxílio de treinamento de redes neurais buscaram implementar o algoritmo ótimo de substituição de páginas.

Introdução

O problema ronda a computação desde seus primórdios, a memória. Sempre houve um desbalanceamento entre o processamento e a memória, sendo a memória o gargalo dos sistemas computacionais, devido seus altos preços e limitações de espaço. Uma vez que a memória é usada de maneira eficaz para carregar os programas, não havendo a necessidade de constantes consultas ao armazenamento principal que geralmente são discos rígidos, consideravelmente lentos, podem ocorrer falta de páginas na memória, nesse ambiente o sistema operacional precisa escolher qual página vai remover da memória (que é limitada) para buscar no armazenamento a página faltante e carregá-la para a memória. Embora seja possível escolher aleatoriamente a página que será removida no caso de falta de página, o sistema pode ganhar um desempenho muito melhor se for escolhida para descarte uma página que não é frequentemente usada e que provavelmente não será requisitada tão brevemente, visando esse desempenho, pesquisadores e teóricos investiram seus trabalhos no desenvolvimento de bons algoritmos de substituição, sendo um deles o FIFO, escolhido para a implementação desse projeto.

Dada uma implementação eficaz do algoritmo ótimo de substituição de páginas já existente e seu relatório de colisões, porcentagens de acerto e testes com diferentes tamanhos de buffer para definidos conjuntos de arquivos, seria necessário para comparar uma implementação de um dos demais algoritmos mais comuns e consolidados para substituição de páginas no buffer, logo, visando um sólido aprendizado das teorias e técnicas obtidas ao longo da disciplina, como o gerenciamento de Processos pelo sistema operacional, seus conflitos, o gerenciamento da Memória, entre outros, optei por implementar um dos algoritmos ainda usados, porém, de implementação não muito complexa para não ser prejudicado com uma grande curva de aprendizado, portanto o algoritmo escolhido foi o “Primeiro a Entrar, Primeiro a Sair”, mais conhecido e citado como FIFO (First In First Out). A escolha do algoritmo também foi motivada por ser um dos algoritmos de substituição de páginas com o menor custo, diferente da proposta do algoritmo ótimo que para se aproximar dele o custo é altíssimo, haja vista o treinamento das redes neurais.

Motivado a gerar essas estatísticas e obter dados reais que explicitem a diferença real e não teórica como estudada na bibliografia principal da disciplina de políticas de substituição tão distintas, junto do outro grupo, foram decididos pontos importantes para que a comparação fosse a mais fiel possível, como, usarmos a mesma linguagem de programação, criar nosso buffer em cima de uma mesma base de dados e rodar os mesmos testes.

Problema

Dado o problema conhecido do gerenciamento de memória que é a dificuldade de uma eficaz substituição de páginas na memória pelo sistema operacional, estudados algoritmos consolidados para resolver tal problema, analisados custo e benefício de cada um deles, em união com o outro grupo de projeto do mesmo capítulo, vimos que não há comparações sólidas de fato com números reais para observarmos o desempenho de diferentes algoritmos.

Abordagem

O problema foi resolvido com a implementação de dois dos algoritmos de paginação, o ótimo com o auxílio de redes neurais (pelo outro grupo) e um dos com mais baixo custo, FIFO (por mim). E com eles realizar uma simulação do funcionamento de ambos buffers e gerar relatórios completos para comparação.

Desenvolvimento

Primeiramente foi estudado o material sobre gerenciamento de memória pelo Sistema Operacional, com ênfase nos itens referentes a paginação e algoritmos de paginação.

Com uma base sólida de como acontece a paginação, o porquê dela existir e seus diferentes algoritmos com diferentes políticas de substituição, foi escolhido para implementação um deles, o FIFO.

Em seguida foi decidido o formato do relatório que ambos grupos precisaram gerar para que fosse possível eficazes comparações entre as diferentes implementações dos algoritmos de paginação, então, decidimos levar em conta: o número de buscas, as colisões no buffer (falta de páginas no buffer), a frequência das colisões, o tamanho do buffer usado, o tamanho da base de dados e a proporção entre o buffer usado e a base de dados.

A base de dados escolhida foi o MNIST, uma base com 60.000 números feitos a mão, comumente usada para avaliação de métodos de processamento de imagem. Essa database foi escolhida para esse projeto pois é aberta ao público, mas infelizmente não é grande o suficiente para encher a memória RAM de um computador comum, sendo assim, o próximo passo, simular uma RAM pequena.

Para melhorar a velocidade de execução do projeto, as imagens ficam armazenadas todas em RAM e o buffer é apenas visto como um conceito, são contadas as colisões e as falhas no buffer e então é mostrada a eficiência do buffer. Além disso, ler as imagens do MNIST se provou um desafio por conta de seu formato compactado específico, dessa forma ambos grupos compartilharam trechos de códigos que compreendem a descompactação e leitura desses arquivos para manter o mesmo padrão, porém, com significativas diferenças pois, para um foi necessário ainda o treinamento de uma rede neural com os arquivos e o outro descompactou em uma estrutura de dados adequada para implementação do algoritmo FIFO, uma fila.

O acesso a base de dados, como definido em projeto com ambos grupos, é feito de maneira aleatória 100.000 vezes para cada buffer testado, gerando assim sólidos resultados para boas comparações.

Com o esqueleto do projeto concluído e decisões de projeto importantes tomadas, foi iniciado a implementação do algoritmo na linguagem Python, utilizando classes para abstração dos conceitos de Páginas, Disco e Buffer, cada um com seus estados e comportamentos definidos em atributos e métodos, de maneira que a abstração do disco acontece pelo carregamento do arquivo de base de dados que no caso são as imagens do ‘.idx’, para cada imagem é instanciado um objeto da classe Página que contém seu identificador e seu conteúdo, e as páginas são guardadas como um array num objeto da classe Disco, semelhante a essa é a

abstração da memória do buffer, que também armazena páginas, porém, com um limite de páginas a serem armazenadas e com sua devida política de substituição nos casos de faltas (colisões).

Para melhor eficiência dos testes foi implementada threads no arquivo de simulação, processando a simulação em diferentes threads os testes que por contar com grande número de páginas não sobrecarregam um único core agiliza o retorno dos resultados (mesmo assim pode ser demorado devido a grande quantidade de páginas e operações de paginação que ocorrem na simulação).

Vale acrescentar que durante o desenvolvimento foram aplicadas algumas técnicas de TDD, um desenvolvimento baseado em testes usando Doctests do Python, onde algumas funções nasceram como testes primeiro e depois foram implementadas.

Seguem alguns testes feitos com o buffer e seu desempenho formatados:

Base de dados de 10000 imagens

Buffer contendo no máximo 1 imagens ou 0.01% da base de dados, após 100000 consultas, 99.989% não estavam presentes no buffer.
Buffer contendo no máximo 2 imagens ou 0.02% da base de dados, após 100000 consultas, 99.974% não estavam presentes no buffer.
Buffer contendo no máximo 10 imagens ou 0.1% da base de dados, após 100000 consultas, 99.919% não estavam presentes no buffer.
Buffer contendo no máximo 100 imagens ou 1.0% da base de dados, após 100000 consultas, 98.985% não estavam presentes no buffer.
Buffer contendo no máximo 500 imagens ou 5.0% da base de dados, após 100000 consultas, 95.081% não estavam presentes no buffer.
Buffer contendo no máximo 1000 imagens ou 10.0% da base de dados, após 100000 consultas, 90.062% não estavam presentes no buffer.
Buffer contendo no máximo 5000 imagens ou 50.0% da base de dados, após 100000 consultas, 51.77% não estavam presentes no buffer.
Buffer contendo no máximo 10000 imagens ou 100.0% da base de dados, após 100000 consultas, 10.0% não estavam presentes no buffer.
Base de dados de 60000 imagens

O relatório completo está no arquivo Report.txt encontrado na pasta raíz do projeto.

Conclusão

Os maiores desafios encontrados durantes o desenvolvimento do projeto foram os de integrar de certa forma as soluções de ambos grupos, com algoritmos diferentes, ideias diferentes, mas, unidos de maneira que fosse possível análise e comparação real dos resultados. Outro obstáculo foi a separação da equipe, infelizmente a partir da entrega 2 continuei o desenvolvimento do trabalho sozinho, com uma grande curva de aprendizado ainda para alcançar tanto no conceito de paginação quanto as ferramentas específicas da linguagem de programação necessárias para a implementação do projeto.

Finalmente, concluído o projeto, o software que foi entregado consiste em um simulador de buffer com que implementa como política de paginação o algoritmo FIFO funcionando corretamente e com testes já bem definidos em seu arquivo ‘main.py’. Ao final da execução dos testes são gerados diversos relatórios detalhando cada passo. Como o número de acessos é significamente grande para todos os casos (100.000) e nos menores testes utilizo uma base com 10.000 páginas e no maior 60.000 páginas, dependendo do poder de processamento do computador pode ser que a execução do programa possa ser mais lenta.

Referências

Sistemas Operacionais Modernos - 4º Edição - ANDREW S., TANENBAUM HERBERT.
The MNIST database of handwritten digits.

Anexo 1:

Há duas maneiras de executar o projeto uma vez que sua pasta raíz foi extraída para algum diretório de seu computador.

1. Utilizando a Virtual Enviroment (venv), ambiente python com o mínimo necessário criado na pasta raíz do projeto para execução do mesmo sem problemas de compatibilidade com bibliotecas locais:
    a. Abra um terminal (ou prompt de comando)
    b. Caminhe até a pasta raíz do projeto
    c. execute o comando: **source venv/bin/activate**
    d. Se tudo correu bem, antes do nome do seu usuário apareceu a marca **(venv)**
    e. Agora é só iniciar a simulação com o comando no mesmo diretório raíz: **python main.py**
    f. E então aguardar os resultados. (Resultados esses já compilados no arquivo de texto “Report.txt”)

2. Utilizando seu ambiente de desenvolvimento local (Python e módulos):
    a. Abra um terminal (ou prompt de comando)
    b. Caminhe até a pasta raíz do projeto
    c. Certifique-se de ter instalado em sua máquina o módulo *numpy* com o comando: **pip install numpy**
    d. Inicie a simulação executando: **python main.py**
    e. Aguarde os resultados. (Os mesmos do arquivo de texto “Report.txt”)

Mais detalhes sobre o projeto estão bem documentados em cada classe e em cada função nos arquivos de código fonte. Variáveis globais como tamanho do buffer para os testes podem ser facilmente modificadas no “main.py”.

Anexo 2:

Compilação de alguns Resultados obtidos (Report.txt):

Relatórios do Buffer - Algoritmo de Paginação FIFO

Base de dados de 10000 imagens

Buffer contendo no máximo 1 imagens ou 0.01% da base de dados, após 100000 consultas, 99.989% não estavam presentes no buffer.
Buffer contendo no máximo 2 imagens ou 0.02% da base de dados, após 100000 consultas, 99.974% não estavam presentes no buffer.
Buffer contendo no máximo 10 imagens ou 0.1% da base de dados, após 100000 consultas, 99.919% não estavam presentes no buffer.
Buffer contendo no máximo 100 imagens ou 1.0% da base de dados, após 100000 consultas, 98.985% não estavam presentes no buffer.
Buffer contendo no máximo 500 imagens ou 5.0% da base de dados, após 100000 consultas, 95.081% não estavam presentes no buffer.
Buffer contendo no máximo 1000 imagens ou 10.0% da base de dados, após 100000 consultas, 90.062% não estavam presentes no buffer.
Buffer contendo no máximo 5000 imagens ou 50.0% da base de dados, após 100000 consultas, 51.77% não estavam presentes no buffer.
Buffer contendo no máximo 10000 imagens ou 100.0% da base de dados, após 100000 consultas, 10.0% não estavam presentes no buffer.

Base de dados de 60000 imagens

Buffer contendo no máximo 1 imagens ou 0.0016666666666666668% da base de dados, após 100000 consultas, 99.998% não estavam presentes no buffer.
Buffer contendo no máximo 2 imagens ou 0.0033333333333333335% da base de dados, após 100000 consultas, 99.997% não estavam presentes no buffer.
Buffer contendo no máximo 10 imagens ou 0.016666666666666666% da base de dados, após 100000 consultas, 99.986% não estavam presentes no buffer.
Buffer contendo no máximo 100 imagens ou 0.16666666666666666% da base de dados, após 100000 consultas, 99.825% não estavam presentes no buffer.
Buffer contendo no máximo 500 imagens ou 0.8333333333333334% da base de dados, após 100000 consultas, 99.142% não estavam presentes no buffer.
Buffer contendo no máximo 1000 imagens ou 1.6666666666666667% da base de dados, após 100000 consultas, 98.347% não estavam presentes no buffer.
Buffer contendo no máximo 5000 imagens ou 8.333333333333334% da base de dados, após 100000 consultas, 91.89% não estavam presentes no buffer.

Buffer contendo no máximo 10000 imagens ou 16.666666666666668% da base de dados, após 100000 consultas, 84.153% não estavam presentes no buffer.

Buffer contendo no máximo 6 imagens ou 0.01% da base de dados, após 100000 consultas, 99.996% não estavam presentes no buffer.
Buffer contendo no máximo 60 imagens ou 0.1% da base de dados, após 100000 consultas, 99.926% não estavam presentes no buffer.
Buffer contendo no máximo 600 imagens ou 1.0% da base de dados, após 100000 consultas, 99.034% não estavam presentes no buffer.
Buffer contendo no máximo 6000 imagens ou 10.0% da base de dados, após 100000 consultas, 90.432% não estavam presentes no buffer.
Buffer contendo no máximo 60000 imagens ou 100.0% da base de dados, após 100000 consultas, 48.836% não estavam presentes no buffer.

Precisão: 0.0001

Buscas: 100000
Colisões no buffer: 99990
Frequência de colisão no buffer: 0.9999

Tamanho do buffer: 6
Tamanho da database: 60000
Tamanho do buffer / Tamanho da database: 0.0001

Precisão: 0.00107

Buscas: 100000
Colisões no buffer: 99893
Frequência de colisão no buffer: 0.99893

Tamanho do buffer: 60
Tamanho da database: 60000
Tamanho do buffer / Tamanho da database: 0.001

Precisão: 0.00976

Buscas: 100000
Colisões no buffer: 99024
Frequência de colisão no buffer: 0.99024

Tamanho do buffer: 600
Tamanho da database: 60000
Tamanho do buffer / Tamanho da database: 0.01

Precisão: 0.0978

Buscas: 100000
Colisões no buffer: 90220
Frequência de colisão no buffer: 0.9022

Tamanho do buffer: 6000
Tamanho da database: 60000
Tamanho do buffer / Tamanho da database: 0.1

Precisão: 0.51325

Buscas: 100000
Colisões no buffer: 48675
Frequência de colisão no buffer: 0.48675

Tamanho do buffer: 60000
Tamanho da database: 60000
Tamanho do buffer / Tamanho da database: 1.0