

Aluno: Lucas Otávio Vieira

Relatório final sobre o projeto de arquitetura de computadores:

Proposta 2, Simulador de Memória virtual

Introdução:

O conceito de memória virtual é de fundamental importância no estudo da arquitetura de computadores, principalmente para programadores, por reforçar o aprofundamento no entendimento de como determinado programa é executado, saber quais sistemas e componentes são utilizados para permitir e gerir essa execução, noções que auxiliam no desenvolvimento de um código mais otimizado e funcional. Um programa consiste num conjunto de dados e instruções a serem executadas pelo processador, para uma aplicação ser executada ela deve estar carregada na memória principal, assim como o sistema operacional. Em razão do particionamento da memória e outras particularidades que existem na execução de um programa, não é possível garantir que a aplicação estará sempre na mesma posição, o que nos impede de usar o endereço físico da memória como referência, de maneira que precisamos de outro tipo de endereço, o endereço lógico, que é o local que corresponde ao início do programa (convertido pelo processador num endereço físico).

Outro ponto importante para o entendimento da memória virtual é o conceito de paginação, que consiste na divisão da memória e do programa em pedaços de tamanho fixo, chamados de frames quando são pedaços de memória e de páginas quando são pedaços da aplicação, ambos possuem o mesmo tamanho e representam basicamente a mesma coisa. Esse processo todo tem a função de reduzir o desperdício de memória, pois, o desperdício máximo é referente a uma fração da última página alocada (caso a aplicação não necessite dessa página por inteiro).

A memória virtual se baseia no princípio de que não é necessário carregar todo o programa na memória de uma vez só, principalmente porque existem vários programas maiores do que a própria memória. Dessa maneira, é preciso dividir o programa em partes e carregá-las segundo a demanda, ou seja, aquelas que estão sendo utilizadas no momento. Nesse instante, a memória virtual entra em cena, atuando como uma extensão virtual da memória que permite o acesso a um espaço de endereçamento maior que o real. Isso possibilita o compartilhamento de informações entre a memória e vários programas, além de criar a ilusão de que existe mais memória do que a capacidade máxima da memória principal, função muito importante na execução de programas maiores que a memória.

Por meio da técnica de memória virtual é feita a tradução do espaço de endereçamento de um programa para seus endereços reais, um processo similar ao que ocorre na relação entre memória principal e memória cache, aqui a memória principal faria o papel da cache e a memória secundária seria a “memória principal”. Este trabalho tem por objetivo simular a técnica de memória virtual e exemplificar como todo esse processo ocorre na prática, exibindo alguns detalhes da sua execução e outras particularidades, como por exemplo, as técnicas de substituição de página, pagefaults, hit rate, tempo de processamento e o status da memória.

Solução implementada:

Para implementar o simulador foram utilizados conceitos de programação orientada a objetos na linguagem c++, principalmente por já ter utilizado essas ferramentas nas disciplinas de ITP e LP. O simulador possui uma biblioteca específica que contém a importação de todas as outras bibliotecas necessárias para rodar o programa, a classe endereço, seus métodos as funções externas necessárias para a simulação, dessa forma, na função main só é necessário incluir essa biblioteca e utilizar suas funções dentro de uma lógica que simule o papel da memória virtual.

A classe endereço foi criada para fazer a representação dos endereços e guardar outras informações úteis na implementação das técnicas de substituição e na coleta dos dados, além do código do endereço temos o guardamos o momento em que o endereço foi manipulado e o tipo de endereço. Uma pesquisa foi feita em relação a uma função capaz de retornar o tempo atual em milissegundos, e uma versão adaptada foi utilizada, devidamente referenciada nos comentários do código. Todas as outras funcionalidades vieram da aplicação dos conceitos de programação com recursos disponíveis no c++11 que facilitaram enormemente o trabalho.

Funções da biblioteca e breve descrição:

```
horaAtual(); // retorna o tempo atual em ms.  
temPagina(); // verifica se existem paginas vazias na memoria  
escreve(); // escreve um endereço numa pagina de memoria e retorna se conseguiu ou não  
busca(); // procura um endereço nas páginas da memória e retorna se encontrou ou não  
aleatoria(); // faz a substituição do endereço por meio da técnica random  
fifo(); // realiza a substituição do endereço utilizando a técnica fifo  
lru(); // executa a substituição do endereço através da técnica lru
```

Outras funções bastante utilizadas são as disponibilizadas pelo c++11 para a manipulação de vector e de strings. Além das bibliotecas de leitura de arquivos e que trabalham com tempo.

Execução da main(), a função principal:

Após o include da biblioteca criada utilizamos variáveis para a coleta de dados de contagem e dos dados fornecidos pela entrada, validações desses dados são feitas e se tudo ocorre como o previsto o simulador é executado. Para representar a memória utilizei um vector, que apesar de não ter um tamanho fixo definido, possui diversas outras funcionalidades importantes e para limitá-lo basta implementar uma simples lógica nas funções que usam o número de páginas da memória calculado a partir dos dados do usuário. Na representação dos endereços a escolha de strings é bem óbvia, por termos dados numéricos, letras, e espaços precisamos manipular de alguma maneira essa entrada para armazenarmos correntemente na nossa classe endereço, isso é feito utilizando as funções da biblioteca string fornecidas pelo c++11.

Caso o arquivo fornecido na entrada seja lido corretamente o percorremos linha por linha até o fim, manipulando os endereços nas páginas da memória (nosso vector) e incrementando os vetores necessários. Uma descrição sucinta da lógica seria:

- >lê linha do endereço
- >formata o endereço lido de acordo com a classe endereço
- >verifica o tipo de endereço
- >independentemente do tipo (leitura ou escrita) procura o endereço na memória
- >se acha atualiza hits e o tempo da lru
- >se não acha contabiliza pagefaults e tenta escrever o endereço no vector(memória)
- >se não consegue escrever utiliza uma das técnicas de memória para substituir a página
- >repete os passos anteriores até que não exista mais linha para ler

Após a leitura do arquivo ele é fechado, os contadores e as páginas da memória estão atualizados, cálculos com base nesses dados são feitos e os resultados são apresentados na tela.

Explicação das funções de substituição de página:

Os algoritmos para efetuar a substituição da página quando necessário são bem simples de implementar devido ao uso dos recursos do c++11 e de programação orientada a objetos.

Método aleatório(random)> Um índice randômico é gerado com base no tamanho do vector, (que será sempre o número total de páginas, pois essa função só é chamada quando não é possível mais escrever no vector) e na função rand(), utilizando o operador de resto da divisão, o que limita o range ao número de páginas. Após esse passo, o endereço passado é inserido na posição “sorteada” como já está formatado no tipo endereço o vector aceita sem problemas.

OBS: A semente da função que retorna o valor aleatório é atualizada a cada execução do programa, garantindo que é um valor diferente ao utilizarmos o retorno da função horaAtual como parâmetro passado.

Método fifo(first in first out)> Esse método de fila foi o mais simples de implementar, bastou eliminar a primeira posição através das funções do vector(erase e begin), o próprio objeto do tipo vector reorganiza os índices na ordem correta após a retirada do primeiro elemento, e o endereço passado é inserido na última posição do vetor via push_back().

Método lru(least recently used)> Esse foi o método mais trabalhoso de implementar, foi necessário criar um novo atributo na classe endereço para armazenar o tempo e buscar uma função externa que retornasse o tempo atual em milissegundos para se aproximar da velocidade do processador, e mesmo assim, quando temos poucos dados o processador é bastante rápido, não existindo diferença percebida na ordem dos milissegundos, no entanto, quando o número de dados é considerável existe uma diferença de 1 ou 2 milissegundos ou mais entre o primeiro endereço e o último endereço lido ou manipulado. Como cada endereço tem o atributo que armazena esse tempo em que foi atualizado aplicamos o algoritmo do trono com esse atributo, o primeiro endereço do vector será o mais antigo e o índice zero é capturado. Após, percorremos o resto do vector comparando os tempos para encontrar aquele que tem um tempo menor que o primeiro(mais antigo) para ocupar o trono de mais velho, e assim por diante até pegarmos o endereço mais velho de fato, a cada passo o índice desse endereço também é capturado e ao final desse processo atribuímos ao endereço localizado na posição mais antiga o novo endereço passado como argumento da função, atualizando também o tempo em que esse endereço passado foi trocado.

Análise de resultados:

Utilizando um arquivo de entrada com 216 linhas ou acessos, 104 de leitura e 112 de escrita, com cerca de 100 endereços diferentes e repetidos. Uma memória de 128 KB e páginas de 4 KB, temos 100% do vetor ocupado, pois o número de endereços diferentes ultrapassa o total de páginas da memória.

Usando a técnica LRU praticamente todos os dados variam pela inconstância do tempo. Para esse número de dados o tempo de processamento é muito rápido, levando cerca de 0~1 ms apenas, os endereços apresentam praticamente o mesmo tempo e temos uma taxa de acertos variando entre 14 e 15%, com baixas de 7% ou até mais dependendo de quanto tempo se espera pra executar novamente, provavelmente essas variações maiores são devido a alguma lentidão do processador ou a perda de temporalidade da cache quando demoramos muito a rodar novamente, quando isso ocorre percebemos um aumento no tempo de processamento que varia entre 3 e 5 ms. Em relação aos pagefaults temos cerca de 330~360 entre 155 e 169%.

Prints:

```
Número de paginas: 32
Ocupadas: 32
Vazias: 0
Uso das páginas da memória: 100%
R: 18 ou 56.25%
W: 14 ou 43.75%
```

Mais dados e estatísticas

```
Acessos: 216
Hit rate: 15.28%
Pagefaults: 334 (155%)
Tempo de processamento: 0 ms
```

```
Número de paginas: 32
Ocupadas: 32
Vazias: 0
Uso das páginas da memória: 100%
R: 13 ou 40.62%
W: 19 ou 59.38%
```

Mais dados e estatísticas

```
Acessos: 216
Hit rate: 7.87%
Pagefaults: 366 (169%)
Tempo de processamento: 3 ms
```

OBS: Na apresentação o número de pagefaults estava sendo contabilizado da forma incorreta devido a uma chave de if errada, uma parte do código estava comentada incorretamente, ao consertar isso o número de pagefaults voltou ao normal. A contagem incorreta do número de páginas escritas (por não ter entendido direito o que seria a escrita para o problema) também foi corrigida, bastou incrementar os contadores assim que o tipo de endereço era identificado e tirar dos outros lugares.

Utilizando o método randômico também observamos variações aleatórias, mas uma certa ineficácia para essa entrada específica com hit rate variando entre 1.3 e 4.6%, com picos de 5% e pagefaults na faixa de 380~395 de 176% a 182%. O tempo de processamento pouco varia, ficando sempre na faixa de 1 ms.

Prints:

```
Número de paginas: 32
Ocupadas: 32
Vazias: 0
Uso das páginas da memória: 100%
R: 18 ou 56.25%
W: 14 ou 43.75%
```

Mais dados e estatísticas

```
Acessos: 216
Hit rate: 1.389%
Pagefaults: 394 (182%)
Tempo de processamento: 1 ms
```

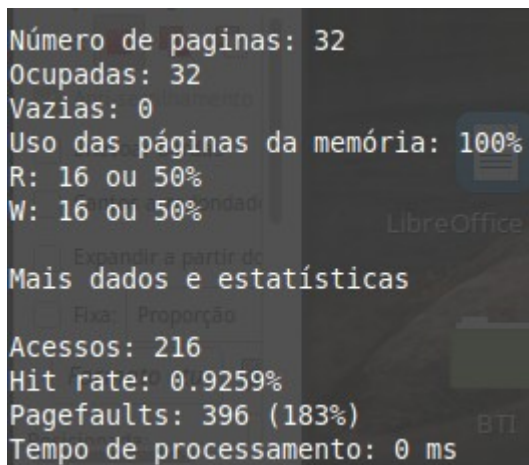
```
Número de paginas: 32
Ocupadas: 32
Vazias: 0
Uso das páginas da memória: 100%
R: 17 ou 53.12%
W: 15 ou 46.88%
```

Mais dados e estatísticas

```
Acessos: 216
Hit rate: 4.63%
Pagefaults: 380 (176%)
Tempo de processamento: 1 ms
```

Já a técnica fifo apresentou o pior resultado para essa entrada específica com dados constantes, inclusive os endereços que apareciam na memória eram os mesmos.

Print:



```
Número de paginas: 32
Ocupadas: 32
Vazias: 0
Uso das páginas da memória: 100%
R: 16 ou 50%
W: 16 ou 50%
Mais dados e estatísticas
Acessos: 216
Hit rate: 0.9259%
Pagefaults: 396 (183%)
Tempo de processamento: 0 ms
```

Conclusão:

O simulador foi construído utilizando as ferramentas que julguei mais adequadas, apesar de alguma dificuldade considero o resultado dentro da expectativa. Em relação aos resultados acredito que o LRU tenha se saído melhor porque explora bastante o princípio da temporalidade, como o processador executa muito rapidamente a diferença de tempo entre os endereços é pequena, mas mesmo assim fez alguma diferença na busca por ter uma lógica temporal embutida (retirar o mais antigo). O random apresentou um desempenho com alguma variação e não se saiu tão bem quanto o LRU, variando bastante, principalmente por seguir um princípio randômico, o 'acaso' seria o fator determinante, diminuindo consideravelmente o desempenho. Já o fifo se mostrou o pior para esse caso específico por ser linear, retirando sempre o primeiro e inserindo no final, se não existirem muitos endereços repetidos essa técnica não será muito eficaz, o que ocorre com essa amostra, dos 216 acessos cerca de 100 são endereços diferentes, e se repetem apenas uma vez. Vale lembrar que em virtude do elevado número de atividades no final do semestre não foi possível realizar mais simulações para obter mais resultados para comparar e chegar a conclusões mais embasadas, no entanto, no geral o desempenho provavelmente seguiria um padrão parecido com o analisado aqui.

Referências:

Utilizei os slides da disciplina para estudar o conceito de memória virtual. Além da wikipédia para alguns conceitos relacionados que me ajudaram a entender a proposta.

Em relação ao código utilizei as listas da disciplina de Linguagem de programação para relembrar a sintaxe do c++, principalmente na parte de leitura de arquivo e criação e utilização de bibliotecas personalizadas.

O cplusplus.com também foi uma fonte muito importante em relação ao uso das bibliotecas vector e string, lá pesquisei todas as funções dessas bibliotecas que utilizei no trabalho.

O stackoverflow.com também foi utilizado para pesquisar uma função que retornasse o tempo em milissegundos, assim como existe em java, devidamente referenciada nos comentários do código.

Link:

<https://stackoverflow.com/questions/19555121/how-to-get-current-timestamp-in-milliseconds-since-1970-just-the-way-java-gets>

(uma dessas funções foi adaptada para o meu código).

O site random.org também foi utilizado para gerar as strings não repetidas que representam os endereços.

Descrição da organização do código:

O projeto é bem simples e todos os arquivos estão localizados na mesma pasta, com o nome de Projeto arquitetura, ao extrair o arquivo com essa pasta zipada e extraí-lo teremos o arquivo.log, que foi o utilizado para a simulação. Dois arquivos .cpp, um é a implementação da biblioteca criada e o outro é a função main com a lógica do programa. O arquivo biblioteca.hpp é o arquivo de cabeçalho da biblioteca criado para declarar a classe endereço, importar as outras bibliotecas e especificar a assinatura das funções utilizadas. Os arquivos com extensão .o são os objetos que o c cria para executar o programa e ao rodar os comandos eles serão atualizados. Existe um arquivo READ-ME com as instruções de compilação e execução do programa, mas também colocarei aqui.

Compilação e execução:

Para compilar e executar o projeto é preciso ter o compilador g++ instalado, assim como o suporte pra versão 11 do c++. Ao extrair os arquivos teremos a pasta Projeto arquitetura com todos os arquivos dentro. Execute o terminal nessa pasta e os seguintes comandos:

```
g++ -std=c++11 -c biblioteca.cpp main.cpp
g++ -std=c++11 biblioteca.o main.o -o main
./main com a opção desejada no formato que consta na proposta
exemplos:
```

```
./main lru arquivo.log 4 128
./main fifo arquivo.log 4 128
./main random arquivo.log 4 128
```

OBS: o g++ chama o compilador e a opção -std=c++11 é utilizada para referenciar as funcionalidades do c++11, o primeiro comando compila os cpps e o segundo cria os objetos, já o terceiro executa o programa(função main) com as opções de entrada.