UNIVERSIDADE ESTADUAL DE MARINGÁ

Ives Yuji Hirose RA:130282

Olga Maria RA:130002

Arquitetura e Organização de Computadores: Algoritmos de substituição de cache.

Maringá

2023

SUMÁRIO:

1 INTRODUÇÃO

O presente relatório tem como objetivo elucidar sobre a memória cache, unidade de armazenamento e instruções localizada no processador e suas formas de substituição,introduzindo algoritmos essenciais para manipulação da memória cache.

2 INSTALAÇÃO:

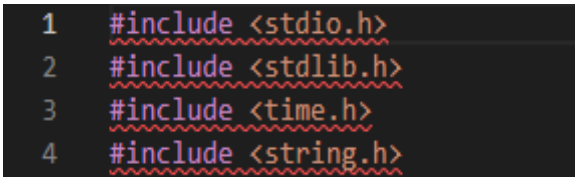
O programa de substituição de cache foi feito na linguagem C, pelo programa CodeBlocks e VSCode

LINK PARA COMPILADOR VSCODE
VSCode
LINK PARA COMPILADOR CODEBLOCKS
CodeBlocks

Após baixar o programa é necessário abrir o arquivo .c pelo compilador de sua preferência e em seguida executar o arquivo pelo comando F6 no teclado. Logo em seguida é aberto um Prompt de comando que exige a interação do usuário.

2 METODOLOGIA E UTILIZAÇÃO:

Começamos pela seleção de bibliotecas que foram utilizadas para a criação do programa.

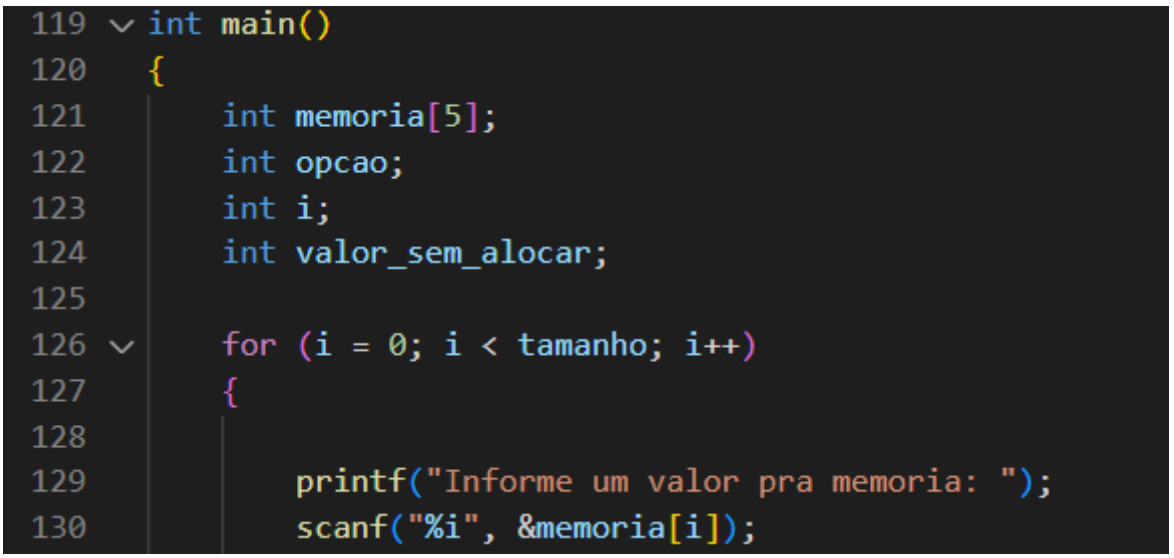


```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <string.h>
```

(figura 01) biblioteca utilizando o programa VsCode.

<stdio.h> Responsável pela adição de três variáveis(int, float, char) e funções básicas de leitura entrada e saída.
<stdlib.h> Utilizada para alocação e manipulação de memória
<time.h> Especializada na manipulação de referências temporais sejam minutos segundos meses dias anos ou até séculos
<string.h> Associado a manipulação de strings

Em seguida partimos para criação da interface do programa fazendo uma interação com o usuário através do “for” (Estrutura de repetição que executa várias instruções) solicitando ao mesmo, números inteiros para alocação na memória.



```
119 int main()
120 {
121     int memoria[5];
122     int opcao;
123     int i;
124     int valor_sem_alocar;
125
126     for (i = 0; i < tamanho; i++)
127     {
128
129         printf("Informe um valor pra memoria: ");
130         scanf("%i", &memoria[i]);
```

(figura 02) entrada de dados utilizando o programa VsCode.

Assim que o usuário informa os cinco números, é solicitado mais uma vez um outro número inteiro, quando ele é informado o programa identifica que não há

espaço suficiente para alocação desse valor no vetor, é aberto uma nova interface com várias opções para o usuário alocar esse valor que ficou fora do vetor. Por fim, o usuário escolhe uma das opções feita pela estrutura de comparação denominada "switch", ocasionando numa série de chamadas de funções para cada caso (LFU, FIFO, LRU e aleatório).

```
142         switch (opcao)
143         {
144         case 1:
145             acessa_memoria(memoria);
146             LRU(memoria, valor_sem_alocar);
147             imprime(memoria);
148             break;
149         case 2:
150             FIFO(memoria, valor_sem_alocar);
151             imprime(memoria);
152             break;
153         case 3:
154             acessa_memoria(memoria);
155             LFU(memoria, valor_sem_alocar);
156             imprime(memoria);
157             break;
158         case 4:
159             aleatorio(memoria, valor_sem_alocar);
160             imprime(memoria);
161             break;
162         default:
163             break;
164         }
165     }
166 }
```

(figura 03) switch utilizando o programa VsCode.

FIFO:

Para criação do First in / First Out, criamos uma função que retira o valor do vetor da posição 0, ou seja do primeiro local da memória, e alocamos o valor que o usuário informou.

```
87  void FIFO(int vetor[5], int valor_a_alocar)
88  {
89      vetor[0] = valor_a_alocar;
90
91      printf("\n Alocado na primeira posicao: \n\n");
92  }
```

(figura 04) função FIFO utilizando o programa VsCode.

Antes da substituição pelos métodos LFU e LRU, o programa precisa encontrar quem foi o menos recentemente acessado e quem foi o menos frequentemente acessado, para isso utilizamos em ambos os casos uma função auxiliar que pede ao usuário que acesse a memória, para que as funções de contagem de menos frequente e recente sejam executadas

```
void acessa_memoria(int memoria[5]){
    char querAcessar[1];
    strcpy(querAcessar,"s");
    int posicao;

    pulaLinha();
    printf("Para sabermos qual o menos recente e o menos frequente, preciso que acesse sua memoria \n\n");

    while((strcmp(querAcessar,"s")==0)){
        printf("\nQual posicao quer ver na memoria: ");
        scanf("%i", &posicao);

        for(int i = 0; i < tamanho; i++){
            if(posicao == i){
                printf("Posicao %i contem o valor: %i\n", posicao, memoria[i]);

                menosFrequente(posicao);
                menosRecente(posicao);
            }
        }
        printf("Continuar acessando: (s/n) ");
        scanf("%s", querAcessar);
        pulaLinha();
    }
    if(strcmp(querAcessar,"n")==0){
        printf(" \n Terminou os acessos \n");
    }
}
```

(figura 05) função da memória utilizando o programa VsCode.

A lógica desta função é pedir que o usuário acesse a memória, escolhendo qual posição ele quer que seja impresso na tela, esta posição vai ser passada como parâmetros para as função menos recente e menos frequente, até que o usuário não queira mais acessar, assim será executado as funções LRU e LFU.

LFU:

Para encontramos o valor menos frequente utilizamos a função que conta os acessos em cada posição, para isso foi criado um vetor com todos os índices

zerados, indo de 0 a 4, quando o usuário escolhe uma posição para ser impressa, esta posição, recebe um incremento, ou seja ela foi acessado mais uma vez.

```
58  void menosFrequente(int posicaoAcessada){
59      contador_da_frequencia_acessos[posicaoAcessada]++;
60  }
61
```

(figura 06) função contadora de frequência utilizando o programa VsCode.

Após o usuário parar de acessar a memória, a função LFU vai comparar os valores dentro do vetor auxiliar chamado de "contador da frequência acessos", e encontrar o menor valor, este vai ser o menos frequentado pelo usuário, e esta mesma posição do vetor auxiliar, vai ser a posição da memória onde será alocado o valor que o usuário deseja.

```
81  void LFU (int memoria[5], int valor_a_alocar){
82      int menosAcessadoFrequentemente = 0;
83
84      for (int i = 1; i < tamanho; i++) {
85          if (contador_da_frequencia_acessos[i] < contador_da_frequencia_acessos[menosAcessadoFrequentemente]) {
86              menosAcessadoFrequentemente = i;
87          }
88      }
89      memoria[menosAcessadoFrequentemente] = valor_a_alocar;
90      pulaLinha();
91      printf("Substituicao pela posicao %i, que foi menos acessado frequentemente: \n", menosAcessadoFrequentemente);
92  }
93
```

(figura 07) Função LFU utilizando o programa VsCode.

LRU:

Last Recently Used (LRU) é um método de cache utilizado na remoção do dado menos utilizado recentemente. Esse conceito é muito utilizado em páginas da internet. Para criação do recurso, utilizamos uma função auxiliar que conta o tempo que foi acessado cada posição na memória, este tempo é guardado em um vetor auxiliar, chamado de “contador de acessos recentes”, cada posição acessada vai receber o valor de tempo, e este a cada acesso é incrementado, ou seja, se acessar a posição X e depois a posição Y, a Y terá um valor maior que X, pois acessada mais recentemente.

```
54  void menosRecente(int posicaoAcessada){
55      tempo++;
56      contador_de_acessos_recentes[posicaoAcessada] = tempo;
57  }
```

(figura 08) função contadora recente utilizando o programa VsCode.

Por fim, a função LRU, compara os valores, até encontrar o menor, e este é o menos recentemente acessado, e esta posição do vetor auxiliar, será a mesma posição do vetor memória que será alocado o valor do usuário.

```
63  void LRU (int memoria[5], int valor_a_alocar){
64      int menosAcessadoRecentemente = 0;
65
66      for (int i = 1; i < tamanho; i++) {
67          if (contador_de_acessos_recentes[i] < contador_de_acessos_recentes[menosAcessadoRecentemente]) {
68              menosAcessadoRecentemente = i;
69          }
70      }
71      memoria[menosAcessadoRecentemente] = valor_a_alocar;
72      pulaLinha();
73      printf("Substituicao pela posicao %i, que foi o usado menos recentemente: \n", menosAcessadoRecentemente);
74
75  }
```

(figura 09) função LRU utilizando o programa VsCode.

Escolha Aleatória:

A escolha aleatória tem como objetivo trocar o valor que o usuário quer na memória, por uma posição aleatória, para isso usamos uma função já criada no C, puxada pelas bibliotecas, a função rand(), ela cria uma valor aleatório, que varia de 0 a 4, e para que o valor não seja sempre repetido, tivemos que utilizar a função srand() que não deixa a rand repetir os valores, após a geração de um valor aleatório, ele é alocado em uma variável para que naquela posição seja alocado último o valor que o usuário pediu.

```
void aleatorio(int vetor[5], int valor_a_alocar){
    srand( (unsigned)time(NULL) );
    int posicaoAleatorio = (rand() % 4);
    vetor[posicaoAleatorio] = valor_a_alocar;

    printf("Alocado na posicao %i, aleatoriamente: \n", posicaoAleatorio);
}
```

(figura 10) função aleatória utilizando o programa VsCode.

CONSIDERAÇÕES FINAIS:

Neste trabalho foi concluído a importância da memória cache cada vez mais presente no nosso cotidiano e como ela é essencial na matéria de Arquitetura e Organização de Dados. Durante a criação do código foi constatada a dificuldade na incorporação da memória cache com o contador para cada um dos algoritmos LFU e LRU. Além disso desenrolou-se com certa facilidade o algoritmo de substituição de Escolha aleatória e FIFO por já terem comandos e estruturas evidentes.

REFERÊNCIAS BIBLIOGRÁFICAS:

HTTPS://ACERVOLIMA.COM/LFU-FORMULARIO-COMPLETO/. *In*: RODRIGUES, Jardel *et al*. **Estudo Comparativo de Simuladores de Memória Cache**. [*S. l.*], 2011. Disponível em: https://www.researchgate.net/profile/Jardel-Rodrigues/publication/325540636_Estudo_Comparativo_de_Simuladores_de_Memoria_Cache/links/5b1400f44585150a0a65813d/Estudo-Comparativo-de-Simuladores-de-Memoria-Cache.pdf. Acesso em: 07 abr. 2023.

Olibário,DesCOMPlica, Oliba!. **Algoritmos de Substituição de Cache - FIFO, LRU, LFU e Aleatório.** Youtube, 02 jul. 2020. Disponível em: <https://www.youtube.com/watch?v=f-5-tJlkSXY&t=561s>. Acesso em: 07 abr. 2023.

Least Frequently Used (LFU) Cache Implementation. [*S. l.*], 24 fev. 2023. Disponível em: https://www.geeksforgeeks.org/least-frequently-used-lfu-cache-implementation/. Acesso em: 7 abr. 2023.