SIMULAÇÃO DE CACHE COM THREADS

DISCENTES:
EFRAIN MARCELO PULGAR PANTALEON
FERNANDO LUCAS SOUSA SILVA
PEDRO LEANDRO BATISTA MARQUES
TEÓPHILO VITOR DE CARVALHO CLEMENTE

DOCENTE:
DIOGO PINHEIRO FERNANDES PEDROSA

NATAL/RN
2022

1. INTRODUÇÃO

O presente trabalho visa apresentar o programa multithread desenvolvido para realizar o mapeamento de uma cache. A partir de um arquivo disponibilizado pelo professor serão realizadas simulações de cache, onde nela, uma thread mestra irá criar threads de mapeamento para cada uma das 4 situações pedidas no roteiro do trabalho, assim em cada uma dessas threads iremos identificar acertos e falhas durante o mapeamento, seja ele um mapeamento direto ou mapeamento associativo. Ademais, como nosso grupo é composto por 4 componentes, como pedido no roteiro adicionamos mais 2 situações para deixar o trabalho mais completo, desse modo teremos 6 situações de mapeamento com threads. Desse modo, aplicamos de forma prática o aprendizado adquirido até então na disciplina.

2. METODOLOGIA

Para a realização deste trabalho, fizemos uso de pesquisas no material didático fornecido pelo professor e em referências do assunto na internet, como também de bibliotecas que possibilitasse o desenvolvimento do código do programa multithread aplicado à memória cache. Para melhor organização deste trabalho apresentaremos uma parte teórica resumida de mapeamentos, as bibliotecas e funções utilizadas no código e em seguida o processo de desenvolvimento do código em questão.

Primeiramente para desenvolvimento do programa era necessário o conhecimento sobre mapeamento de memórias cache, conteúdo esse visto na matéria de arquitetura de computadores, então para embasamento metodológico apresentaremos de forma breve como funciona cada um, o mapeamento direto e o associativo.

A função de mapeamento direto, funciona mapeando cada bloco da memória principal a apenas uma linha de cache (linhas fixas). Para encontrar a linha da cache a qual o bloco da memória principal será alocado, pegamos o número do bloco e dividimos pela quantidade de linhas da cache, pegando o resto da divisão para saber onde irá acontecer a alocação. Assim, ao percorrer os blocos da memória principal e aplicar a operação descrita, a função checa para qual linha irá alocar aquele determinado bloco.

Já o mapeamento associativo permite que os blocos da memória principal sejam alocados em qualquer linha da cache. Para identificarmos onde o bloco vai ser carregado, a cache interpreta um endereço da memória como sendo apenas composto pela campos Tag e Palavra, cada qual com suas respectivas interpretações. Então, a Tag identifica o bloco da memória principal e comparando simultaneamente a tag de cada linha, é possível determinar se aquele determinado já foi alocado na cache.

2.1 BIBLIOTECAS E FUNÇÕES

O funcionamento esperado de forma resumida, consiste na programação de uma thread mestre (ou mãe) que cria um thread de mapeamento para cada sessão de tarefas que vamos realizar, no nosso caso teremos 6 sessões. Dado isso, o thread de mapeamento busca as informações de sessão e mapeamento que devem ser realizadas de acordo com a programação, compila o mapeamento e o limpa após a execução da sessão.

Para tal usaremos as bibliotecas *pandas*, *random* e *numpy* da linguagem *Python*. A biblioteca *pandas* será usada para o tratamento e análise dos dados que receberemos do arquivo enderecos.dat, assim poderemos trabalhar e validar seus dados para o mapeamento da cache. A *random* será usada para o mapeamento associativo gerando números "aleatórios" dentro de uma distribuição pré-definida. Já a biblioteca *numpy*, bastante conhecida, será utilizada para intermediar o uso de vetores e as operações algébricas que sejam necessárias.

Para a construção dos mapeamentos em cada thread a serem utilizadas iremos criar classes para cada uma, desse modo, em cada classe será definido o tratamento e métodos a serem usados para realizar o mapeamento por threads.

2.2 CÓDIGO DESENVOLVIDO

A seguir apresentaremos o código no corpo do texto com suas respectivas explicações, porém se desejar conferir o código completo ele está disponível neste link: https://colab.research.google.com/drive/1CM60fgpB44z9Am7fRZWjYfFe1nQYj4oN?usp=sharing. Para rodar o código basta carregar o arquivo *enderecos.dat* no Google Colaboratory acima e executar as células em ordem e será possível obter os resultados a seguir apresentados.

Primeiramente criamos um script inicial (que será a thread mãe) que realiza a leitura de um arquivo texto chamado *enderecos.dat* (disponibilizado na atividade pelo SIGAA) e salva suas informações em uma estrutura de dados adequada para seguirmos com os próximos itens pedidos.

```
import pandas as pd

from random import randint

import numpy as np

cpu_requests = pd.read_csv("./enderecos.dat") # Lendo os dados de endereço de
memoria
```

```
# Setando dados iniciais

erros = 0

acertos = 0

enderecos = len(cpu_requests['0'])

memoria = []

for i in range(enderecos):

  lista = [False, None]

  memoria.append(lista)

arq = cpu_requests['0']

addresses = []

for i in arq:

  addresses.append(int(i))
```

Para o primeiro item (primeira thread) foi pedida uma simulação dos acertos e falhas de uma cache com mapeamento direto, sendo ela com 1000 linhas e cada linha com 4 endereços.

Inicialmente criamos uma classe que tem o papel de mapear o número de endereços que podem ser armazenados por linha. O método *push* insere um elemento em um posição que estava vazia nesta linha, mas caso a linha estivesse cheia, ele remove o primeiro elemento que foi inserido. O método *sroll_to_right* desloca os elementos da linha para a direita removendo o primeiro que foi inserido (FIFO).

```
class LinhaCacheMapeamentoDireto:

  def __init__(self,):

    self.flag = False

    self.columns = [None,None,None,None]
```

```
  def push(self,address):
    self.flag = True

    for index,value in enumerate(self.columns):
      if value == None:
        self.columns[index]=address

      elif index == len(self.columns)-1:
        self.scroll_to_right(address)

  def scroll_to_right(self,address):
    for index in range(len(self.columns)-1):
      self.columns[index] = self.columns[index+1]
    self.columns[3] = address

  def __repr__(self):
            columns = [f"{index} : {value}" for index,value in
enumerate(self.columns)]

    return f"{columns}"
```

Depois criamos uma classe para armazenar todas as linhas da memória cache. O método *setvalue _in_lin*e insere um endereço na linha da cache.

```
class CacheMapeamentoDireto:

  def __init__(self, number_lines=0):

    self.number_lines = number_lines

          self.memorie   =   [   LinhaCacheMapeamentoDireto()   for   index   in
range(self.number_lines)]



  def set_value_in_line(self,line,address):

      self.memorie[line].push(address)



  def __str__(self):

    rep = f"cache {self.number_lines} linhas\n"

    for index,line in enumerate(self.memorie):

      rep =rep+f" linha {index} e {line}\n"



    return rep
```

Para a segunda thread foi pedida uma simulação de acertos e falhas de uma cache com mapeamento totalmente associativo, sendo a cache com 1000 linhas e cada linha com 4 endereços.

```
from collections import deque

def SimulacaoMapeamentoCompletamenteAssociativo():

    tags = [-1] * 4
```

```
valid = [0] * 4

LRU = deque()

acertos = 0

erros = 0

total_instructions = 0



for i in range(0, 2):

  for addr in addresses:

        offset = addr % 4

        tag = addr // (4)

        if tag in tags:

            location = tags.index(tag)

            if location in LRU:

                LRU.remove(location)

            LRU.append(location)

            acertos += 1

        elif 0 in valid:

            location = valid.index(0)

            tags[location] = tag

            valid[location] = 1

            if i > 0:

                erros += 1
```

```
                if location in LRU:
                    LRU.remove(location)
                LRU.append(location)
            else:
                leastUsedLoc = LRU.popleft()
                tags[leastUsedLoc] = tag
                if i > 0:
                    erros += 1
                if leastUsedLoc in LRU:
                    LRU.remove(leastUsedLoc)
                LRU.append(leastUsedLoc)
            if i > 0:
                total_instructions += 1
    return acertos*1/2
```

Para a terceira thread foi pedida uma simulação de acertos e falhas de uma cache com mapeamento associativo em conjunto de 2 vias. Neste caso, a cache deve ser estruturada em 512 conjuntos. Cada via, de cada conjunto, tem 4 endereços de espaço.

Inicialmente criamos uma classe para mapear as vias das linhas da cache e o número de endereços que podem ser armazenados em cada via da linha.

```
    class LinhaCacheMapeamentoAssociativo2:
      def __init__(self,):
                                                self.way              =
```

```
[{"flag":False,"values":[None,None,None,None]},{"flag":False,"values":[None,No
ne,None,None]}]



        def push(self,address):

          for key,value in enumerate(self.way):

            line = [block for block in value["values"]]



            if None in line:

              empty_index = line.index(None)

              self.way[key]["values"][empty_index] = address

              self.way[key]["flag"] = True



            elif key == 1:

              rand_way = randint(0,1)

              rand_index = randint(0,3)

              self.way[rand_way]["values"][rand_index] = address

              self.way[rand_way]["flag"] = True



        def __repr__(self):

               columns = [f"way {index} : {value}" for index,value in
enumerate(self.way)]



          return f"{columns}"
```

Depois criamos uma classe para armazenar todas as linhas da memória cache. O método *setvalue _in_lin*e insere um endereço na linha da cache.

```
      class CacheMapeamentoAssociativo2:

        def __init__(self, number_lines=0):

          self.number_lines = number_lines

            self.memorie  =  [  LinhaCacheMapeamentoAssociativo2()  for  index  in
range(self.number_lines)]



        def set_value_in_line(self,line,address):

            self.memorie[line].push(address)



        def __str__(self):

          rep = f"cache {self.number_lines} linhas\n"

          for index,line in enumerate(self.memorie):

            rep =rep+f" linha {index} e {line}\n"



          return rep
```

Para a quarta thread foi pedida uma simulação de acertos e falhas de uma cache com mapeamento associativo em conjunto de 4 vias. Neste caso, a cache deve ser estruturada em 256 conjuntos. Cada via, de cada conjunto, tem 4 endereços de espaço.

Semelhante ao mapeamento associativo de 2 vias, criamos uma classe para realizar o mapeamento agora das 4 vias.

```
      class LinhaCacheMapeamentoAssociativo4:

        def __init__(self,):

                    self.way   =   [{"flag":False,"values":[None,None,None,None]},
{"flag":False,"values":[None,None,None,None]}

                                  ,  {"flag":False,"values":[None,None,None,None]},
{"flag":False,"values":[None,None,None,None]}]



        def push(self,address):

          for key,value in enumerate(self.way):

            line = [block for block in value["values"]]



            if None in line:

              empty_index = line.index(None)

              self.way[key]["values"][empty_index] = address

              self.way[key]["flag"] = True



            elif key == 1:

              rand_way = randint(0,1)

              rand_index = randint(0,3)

              self.way[rand_way]["values"][rand_index] = address

              self.way[rand_way]["flag"] = True



        def __repr__(self):
```

```
            columns = [f"way {index} : {value}" for index,value in
enumerate(self.way)]



        return f"{columns}"
```

Depois realizamos o armazenamento das linhas com a classe de mapeamento associativo 4.

```
    class CacheMapeamentoAssociativo4:

      def __init__(self, number_lines=0):

        self.number_lines = number_lines

          self.memorie = [ LinhaCacheMapeamentoAssociativo4() for index in
range(self.number_lines)]



      def set_value_in_line(self,line,address):

          self.memorie[line].push(address)



      def __str__(self):

        rep = f"cache {self.number_lines} linhas\n"

        for index,line in enumerate(self.memorie):

          rep =rep+f" linha {index} e {line}\n"



        return rep
```

Por fim, criamos as funções para contabilizar o número de acertos de cache e de falhas de cache, um acerto é quando o endereço solicitado pelo cpu está presente na linha da cache, e a falha de cache é quanto o endereço não está presente na linha da cache.

Para o Mapeamento Direto temos:

```
cache = CacheMapeamentoDireto(number_lines = 1000)



def SimulacaoMapeamentoDireto(cache=cache, cpu_requests=cpu_requests):

  hit=0

  fault=0

  flag = True

  for request in cpu_requests["0"]:

    line = request%cache.number_lines

    if not cache.memorie[line].flag:

      fault=fault+1



    else:

      if request in cache.memorie[line].columns:

        hit = hit+1

        flag=False

      else:

        fault= fault+1

        flag = True
```

```
    if flag:

      cache.set_value_in_line(line,request)



  return hit
```

Criamos nossas caches no sistema. Para a cache com mapeamento associativo em 2 vias, temos ela com 128 linhas e outra com 512 linhas.Já para a cache com mapeamento associativo em 4 vias, temos ela com 256 linhas e outra com 512 linhas.

```
      cache2_128 = CacheMapeamentoAssociativo2(number_lines = 128)

      cache2_512 = CacheMapeamentoAssociativo2(number_lines = 512)

      cache4_256 = CacheMapeamentoAssociativo4(number_lines = 256)

      cache4_512 = CacheMapeamentoAssociativo4(number_lines = 512)
```

Com elas criadas, criamos os métodos:

```
      def                 SimulacaoMapeamentoAssociativo2_128(cache=cache2_128,
cpu_requests=cpu_requests):

        hit=0

        fault=0

        flag = True

        for request in cpu_requests["0"]:

          line = request%cache.number_lines


                    flag_ways    =    set([block["flag"]    for    block       in
```

```
cache.memorie[line].way])

          if False in flag_ways:

            fault = fault + 1



          else:

              if request in cache.memorie[line].way[0]["values"] or request in
cache.memorie[line].way[1]["values"]:

               hit=hit+1

            else:

               fault = fault + 1



          if flag:

            cache.set_value_in_line(line,request)

        return hit
```

```
      def                SimulacaoMapeamentoAssociativo2_512(cache=cache2_512,
cpu_requests=cpu_requests):

        hit=0

        fault=0

        flag = True

        for request in cpu_requests["0"]:

          line = request%cache.number_lines
```

```
                    flag_ways    =    set([block["flag"]    for    block       in
cache.memorie[line].way])

          if False in flag_ways:

            fault = fault + 1



          else:

              if request in cache.memorie[line].way[0]["values"] or request in
cache.memorie[line].way[1]["values"]:

               hit=hit+1

            else:

               fault = fault + 1



          if flag:

            cache.set_value_in_line(line,request)

        return hit
```

```
      def                SimulacaoMapeamentoAssociativo4_256(cache=cache4_256,
cpu_requests=cpu_requests):

          hit=0

          fault=0

          flag = True

          for request in cpu_requests["0"]:

            line = request%cache.number_lines
```

```
                    flag_ways = set([block["flag"] for block in
cache.memorie[line].way])
            if False in flag_ways:
              fault = fault + 1

            else:
               if request in cache.memorie[line].way[0]["values"] or request in
cache.memorie[line].way[1]["values"]:
                 hit=hit+1
               else:
                 fault = fault + 1

            if flag:
              cache.set_value_in_line(line,request)
          return hit
```

```
      def SimulacaoMapeamentoAssociativo4_512(cache=cache4_512,
cpu_requests=cpu_requests):
          hit=0
          fault=0
          flag = True
          for request in cpu_requests["0"]:
```

```
            line = request%cache.number_lines



                        flag_ways   =   set([block["flag"]   for   block      in
cache.memorie[line].way])

            if False in flag_ways:

              fault = fault + 1



            else:

               if request in cache.memorie[line].way[0]["values"] or request in
cache.memorie[line].way[1]["values"]:

                 hit=hit+1

              else:

                 fault = fault + 1



            if flag:

              cache.set_value_in_line(line,request)

          return hit
```

3. RESULTADOS

Para obter os resultados, executamos nossas simulações criando funções para retornar os resultados de acertos e erros de uma forma mais legível.

```
      import threading

      import random
```

```
      import sys

      global x, x1, x2, x3, x4, x5



      def task1():

                                              print("\nIniciando          Thread
{}".format(threading.current_thread().name))

          a = SimulacaoMapeamentoDireto()

          global x

            x = "\nAcertos para a Thread 1 (Direto): {} ".format(a) + " -->
Porcentagem: " + str((a/enderecos)*100) + "%\nErros para a Thread 1: {}
".format(enderecos - a) + " --> Porcentagem: " + str(((enderecos -
a)/enderecos)*100) + "%\n"



      def task2():

                                              print("\nIniciando          Thread
{}".format(threading.current_thread().name))

          a = SimulacaoMapeamentoCompletamenteAssociativo()

          global x1

            x1 = "\nAcertos para a Thread 2 (Associativo): {} ".format(a) + "
--> Porcentagem: " + str((a/enderecos)*100) + "%\nErros para a Thread 2: {}
".format(enderecos - a) + " --> Porcentagem: " + str(((enderecos -
a)/enderecos)*100) + "%\n"



      def task3():

                                              print("\nIniciando          Thread
{}".format(threading.current_thread().name))
```

```
          a = SimulacaoMapeamentoAssociativo2_512()

          global x2

           x2 = "\nAcertos para a Thread 3 (2 vias - 512): {} ".format(a) + "
--> Porcentagem: " + str((a/enderecos)*100) + "%\nErros para a Thread 3: {}
".format(enderecos - a) + " --> Porcentagem: " + str(((enderecos -
a)/enderecos)*100) + "%\n"



      def task4():

                                          print("\nIniciando          Thread
{}".format(threading.current_thread().name))

          a = SimulacaoMapeamentoAssociativo4_256()

          global x3

           x3 = "\nAcertos para a Thread 4 (4 vias - 256): {} ".format(a) + "
--> Porcentagem: " + str((a/enderecos)*100) + "%\nErros para a Thread 4: {}
".format(enderecos - a) + " --> Porcentagem: " + str(((enderecos -
a)/enderecos)*100) + "%\n"



      def task5():

                                          print("\nIniciando          Thread
{}".format(threading.current_thread().name))

          a = SimulacaoMapeamentoAssociativo2_128()

          global x4

           x4 = "\nAcertos para a Thread 5 (2 vias - 128): {} ".format(a) + "
--> Porcentagem: " + str((a/enderecos)*100) + "%\nErros para a Thread 5: {}
".format(enderecos - a) + " --> Porcentagem: " + str(((enderecos -
a)/enderecos)*100) + "%\n"
```

```
    def task6():
                                        print("\nIniciando         Thread
{}".format(threading.current_thread().name))
        a = SimulacaoMapeamentoAssociativo4_512()
        global x5
         x5 = "\nAcertos para a Thread 6 (4 vias - 512): {} ".format(a) + "
--> Porcentagem: " + str((a/enderecos)*100) + "%\nErros para a Thread 6: {}
".format(enderecos - a) + " --> Porcentagem: " + str(((enderecos -
a)/enderecos)*100) + "%\n"



    def main_task():
        global x, x1, x2, x3, x4, x5
      # print name of main thread
        print("Main thread name: {}".format(threading.main_thread().name))
      # creating threads
        t1 = threading.Thread(target=task1, name='1')
        t2 = threading.Thread(target=task2, name='2')
        t3 = threading.Thread(target=task3, name='3')
        t4 = threading.Thread(target=task4, name='4')
        # extras (tests with 2 ways (128) and 4 ways (512))
        t5 = threading.Thread(target=task5, name='5')
```

```
        t6 = threading.Thread(target=task6, name='6')

        # starting threads

        t1.start()

        t2.start()

        t3.start()

        t4.start()

        t5.start()

        t6.start()

        # wait until all threads finish

        t1.join()

        t2.join()

        t3.join()

        t4.join()

        t5.join()

        t6.join()

        # Results

        print(x + x1 + x2 + x3 + "\n##########  Extras  ##########" + x4 +
x5)



    if __name__ == "__main__":

        main_task()
```

Ao rodarmos o script, encontramos os seguintes resultados, as 4 Threads pedidas no relatório e e as 2 Threads extras que fizemos:

Acertos para a Thread 1 (Direto): 4976 --> Percentagem: 42.53%

Erros para a Thread 1: 6723 --> Percentagem: 57.46%

Acertos para a Thread 2 (Associativo): 8774.0 --> Percentagem: 74.99%

Erros para a Thread 2: 2925.0 --> Percentagem: 25.00%

Acertos para a Thread 3 (2 vias - 512): 5627 --> Percentagem: 48.098%

Erros para a Thread 3: 6072 --> Percentagem: 51.901%

Acertos para a Thread 4 (4 vias - 256): 4702 --> Percentagem: 40.19%

Erros para a Thread 4: 6997 --> Percentagem: 59.80%

########## Extras ##########

Acertos para a Thread 5 (2 vias - 128): 4033 --> Percentagem: 34.47%

Erros para a Thread 5: 7666 --> Percentagem: 65.52%

Acertos para a Thread 6 (4 vias - 512): 5665 --> Percentagem: 48.42%

Erros para a Thread 6: 6034 --> Percentagem: 51.57%

4. CONCLUSÕES

A partir do estudo e desenvolvimento do programa multithread de mapeamento de cache foi possível agregar conhecimento ao adquirido em sala e realizar uma simulação prática de programação com threads, visualizando sua eficiência na execução de tarefas, do mesmo modo que ocorre na CPU, em casos onde a demanda de processamento e threads é bem maior e por isso quanto mais otimização melhor. Ademais, foi possível entender as estruturas de programação que podemos usar para construção das threads e as bibliotecas, funções e classes necessárias para isso.

5. REFERÊNCIAS

[1] SILBERSCHATZ, A., GALVIN, P.B., GAGNE, G. Fundamentos de sistemas operacionais, Ed. LTC, 8ª ed., 2011.

[2] Google Colaboratory. Disponível em: https://colab.research.google.com/.