**CMPE 180-94 Project**

**Virtual Memory Manager**



**Team members**

Uttara Vishwas Kulkarni (012420811)

Arshiya Hayatkhan Pathan (012431536)

Bing Shi (011505039)

**Instructor**

Dr. Hungwen Li

**Table of Content**

[**Executive Summary**](#_jtqirj7qjzla) **2**

[**Background And Problem Description**](#_pa18as4tl08m) **2**

[**Objectives and deliverables of the project**](#_xtboywjnhq53) **3**

[**Approach/Methodology**](#_4gyplhq440jz) **4**

[**Result and Analysis**](#_jnkcypjg8efk) **6**

[**Summary**](#_701y7k9faxlu) **6**

[**Appendix**](#_ubcjcy62vg39) **6**

[Appendix A. Source Code](#_rus90g78a86y) 6

[Appendix B. Output.txt](#_kzj7fz3kq0rp) 10

# 

# Executive Summary

This project consists of writing a program that translates logical to physical addresses. This program will read from a file containing logical addresses and using a TLB as well as a page table, will translate each logical address to its corresponding physical address and output the value of the byte stored at the translated physical address.

The goal behind this project is to simulate the steps involved in translating logical to physical addresses. It includes translating logical to physical addresses as well as handling page faults. We would like to show a clear picture of this process, for example, dynamically showing the translation process in the terminal. We will also report Page-fault rate and TLB hit rate after completion of the program.

# Background And Problem Description

The virtual memory manager is simulating the process of logical to physical address translation. It consists of the following steps. The visual representation of the address-translation process appears in Figure 2.

1. The page number and offset are extracted from the logical address as shown in Figure 1.

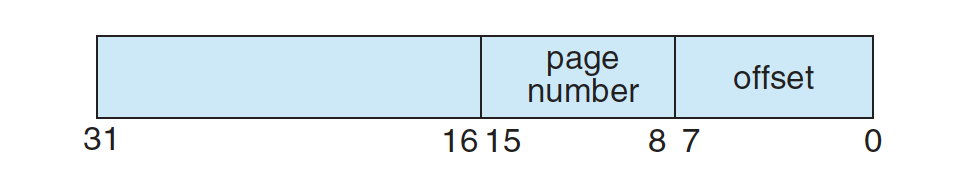


Figure 1. Address Structure

1. Find the corresponding frame number using page number.
   1. Consult TLB. If there is TLB hit, the frame number is obtained from TLB.
   2. If there is a TLB miss, consult page table. If there is a page table hit, the frame number is obtained.
   3. Otherwise, page fault occurs. After handling page fault, restart the process, frame number can be retrieved from TLB.
2. With frame number and offset, data can be extracted from physical memory.

To handle the page fault, a BACKING\_STORE.bin is consulted. It represent a hard drive which can be accessed randomly. When there is a page fault, data will be found in BACKING\_STORE.bin with corresponding page number. The data will be allocated in the physical memory where there is an available frame. After that, TLB and page table will be updated with frame number and page number.

Configuration of our manager is as follows:

• 256 entries in the page table

• Page size of 256 bytes

• 16 entries in the TLB

• Frame size of 256 bytes

• 256 frames

• Physical memory of 65,536 bytes (256 frames \* 256-byte frame size)

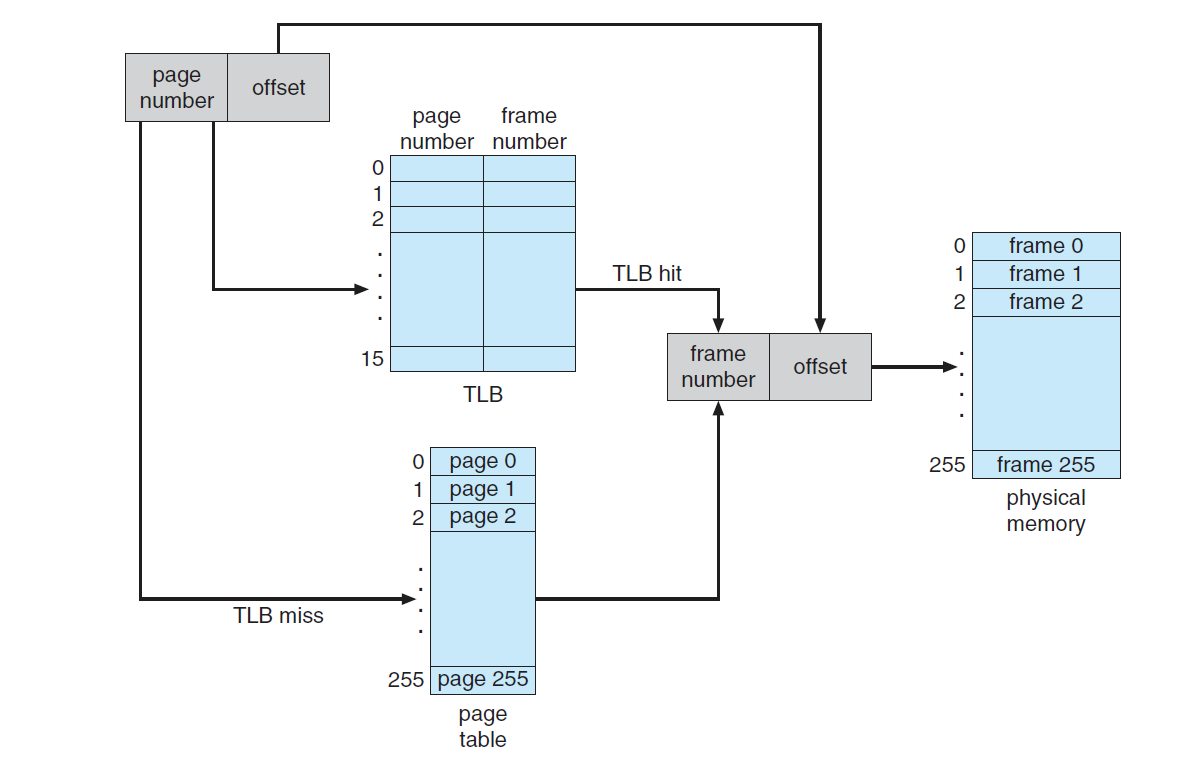


Figure 2. A representation of the address-translation process

# Objectives and deliverables of the project

Objectives:

In order to implement the concept of ‘Virtual Memory Management’, our objective is to simulate translation of a logical address into a physical address that can illustrate the process of mapping logical address to physical address. ‘Virtual address translation’ is implemented using Python 3.

Virtual address translation: Program consists of either virtual addresses or program addresses. For every memory access, either to fetch an instruction or data, the CPU must translate the virtual address to a real physical address. This is virtual address translation.

Deliverables:

* In this project, we will be presenting a file (output.txt) containing all the translated physical addresses. The number of address are 1000.
* The file will have virtual address, mapped physical address and the data.
* Additionally, we will be calculating ‘Page Fault Rate’ and ‘TLB hit rate’.
* In comparison with “correct.txt”, we can conclude the correctness of address translation and rate calculation. (TLB hit rate is slightly different, because we are using LRU to update the table, while the correct file is using FIFO. )

# Approach/Methodology

In this project, we have used “Least Recently Used”(LRU) algorithm in order to handle page fault conditions. As input logical addresses are integer values, we used bit-masking and bit-shifting techniques to convert integer to binary string and obtained corresponding page number and offset.

Program implements Demand Paging for handling page fault. It never brings in a page until that page is referenced. The first reference causes a page fault. Program consults an internal table to determine where the page is located on the backing store. It then finds a free frame and reads the page in from the backing store. The page table is updated to reflect this change, and the instruction that caused the page fault is restarted.

Step 1: Extract Data:

Read logical addresses from a file (address.txt).

Address.txt:

16916

62493

30198

53683

….

The logical address will be split into a page number and an offset. Extract page number from a logical address.

Code:

pageOriginal = logicalAddress & 65280

pageNumber = pageOriginal >> 8

The offset bits are stored in a variable.

Code:

offset = logicalAddress & 255

Step 2: The page number is searched in the TLB and the mapped frame number is

extracted.

Code:

def checkTLB(pageNumber, physicalMemory, offset, logicalAddress, tlb, i, outputFile):

for j in range(len(tlb)):

if pageNumber == tlb[j][0]:

frameNumber = tlb[j][1]

data = part2.readPhysicalMemory(frameNumber, offset, physicalMemory)

Step 3: If the page number is not found in TLB, it is searched in page table and the

mapped frame number is extracted.

Code:

def checkPageTable(pageNumber, logicalAddress, offset, i, pageTable, physicalMemory, outputFile):

for k in range(len(pageTable)):

if pageNumber == pageTable[k][0]:

frameNumber = pageTable[k][1]

data = part2.readPhysicalMemory(frameNumber, offset, physicalMemory)

Step 4: If the page number is neither present in TLB nor in page table, then using

backing store, the corresponding frame number is extracted.

Code:

def pageFaultHandler(pageNumber, tlb, pageTable, physicalMemory):

if int(pageNumber) < 256:

for i in range(256):

if i in physicalMemory.keys():

continue

else:

frameNumber = str(i)

break

backStore = open("BACKING\_STORE.bin", "rb")

physicalMemory[int(frameNumber)] = []

for i in range(256):

backStore.seek(int(pageNumber)\*256+i)

data = str(int.from\_bytes(backStore.read(1), byteorder='big', signed=True))

physicalMemory[int(frameNumber)].insert(i, data)

backStore.close()

Step 5: Update the TLB and page table with the new page number and frame number

entry.

Code:

def updateTLB(pageNumber, frameNumber, tlb):

if len(tlb) < 16:

tlb.append([pageNumber, frameNumber])

else:

tlb.pop(0)

tlb.append([pageNumber, frameNumber])

def updatePageTable(pageNumber, frameNumber, pageTable):

if len(pageTable) < 256:

pageTable.append([pageNumber, frameNumber])

else:

pageTable.pop(0)

pageTable.append([pageNumber, frameNumber])

Step 6: Concatenate the offset with the mapped frame number to get the physical

address.

Code:

physicalAddress = "{0:08b}".format(int(frameNumber)) + "{0:08b}".format(offset)

Step 7: Store translated physical addresses into the file (output.txt).

Code:

outputFile.write(outStr)

Output.txt:

Virtual address: 16916 Physical address: 20 Value: 0

# Result and Analysis

We have successfully translated logical address into physical address. The number of addresses translated are 1000. We have also calculated Page-fault rate which implies the percentage of address references that resulted in page faults. TLB hit rate refers to the percentage of address references that were resolved in

the TLB.

Sample output:

Virtual address: 53683 Physical address: 947 Value: 108

Number of Translated Addresses = 1000  
Page Faults = 244  
Page Fault Rate = 0.244

TLB Hits = 56  
TLB Hit Rate = 0.056

Since the logical addresses in addresses.txt were generated randomly and do not reflect any memory access locality, low TLB hit rate is obtained.

Output.txt(Appendix B)

# Summary

We have designed virtual memory manager which reads the file addresses.txt containing 1000 logical addresses from 0 to 65535. Program translates each logical address to a physical address and also displays contents of the signed byte stored at corresponding physical address.

Program displays and stores the following values to output.txt :

1. The logical address being translated (the integer value being read from addresses.txt).

2. The corresponding physical address (what your program translates the logical address to).

3. The signed byte value stored at the translated physical address.

4. Page-fault rate—The percentage of address references that resulted in page faults.

5. TLB hit rate—The percentage of address references that were resolved in the TLB.

# Appendix

## Appendix A. Source Code

Main.py

import part1

import part2

physicalMemory = {}

tlb = []

pageTable = []

pageFaultCounter = 0

tlbHitCounter = 0

addressReadCounter = 0

if \_\_name\_\_ == '\_\_main\_\_':

outputFile = open('output.txt', 'w')

with open('addresses.txt', 'r') as addressFile:

for line in addressFile:

tlbHit = 0

pageTableTrue = 0

logicalAddress = int(line)

offset = logicalAddress & 255

pageOriginal = logicalAddress & 65280

pageNumber = pageOriginal >> 8

print("Logical address is: " + str(logicalAddress) + "\nPageNumber is: " + str(pageNumber) + "\nOffset: " + str(offset))

addressReadCounter += 1

tlbHit = part1.checkTLB(pageNumber, physicalMemory, offset, logicalAddress, tlb, addressReadCounter, outputFile)

if tlbHit == 1:

tlbHitCounter += 1

if tlbHit != 1:

pageTableTrue = part1.checkPageTable(pageNumber, logicalAddress, offset, addressReadCounter, pageTable, physicalMemory, outputFile)

if pageTableTrue != 1 and tlbHit != 1:

print("This is a page fault!")

part2.pageFaultHandler(pageNumber, tlb, pageTable, physicalMemory)

pageFaultCounter += 1

part1.checkTLB(pageNumber, physicalMemory, offset, logicalAddress, tlb, addressReadCounter, outputFile)

pageFaultRate = pageFaultCounter / addressReadCounter

tlbHitRate = tlbHitCounter / addressReadCounter

outStr = 'Number of translated address: ' + str(addressReadCounter) + '\n' + 'Number of page fault: ' + str(

pageFaultCounter) + '\n' + 'Page fault rate: ' + str(pageFaultRate) + '\n' + 'Number of TLB hits: ' + str(

tlbHitCounter) + '\n' + 'TLB hit rate: ' + str(tlbHitRate) + '\n'

print(outStr)

outputFile.write(outStr)

outputFile.close()

addressFile.close()

Part1.py

import part2

def checkTLB(pageNumber, physicalMemory, offset, logicalAddress, tlb, i, outputFile):

for j in range(len(tlb)):

if pageNumber == tlb[j][0]:

print("Page Number \"" + str(pageNumber) + "\" found in TLB!!")

frameNumber = tlb[j][1]

data = part2.readPhysicalMemory(frameNumber, offset, physicalMemory)

physicalAddress = "{0:08b}".format(int(frameNumber)) + "{0:08b}".format(offset)

physicalAddress = int(physicalAddress, 2)

outStr = str(i) + " Virtual address: " + str(logicalAddress) + " Physical address: " + str(

physicalAddress) + " Value: " + data + "\n"

print(str(i) + " Virtual address: " + str(logicalAddress) + " Physical address: " + str(

physicalAddress) + " Value: " + data)

outputFile.write(outStr)

part2.updateTLBCounter(j, tlb)

return 1

return 0

def checkPageTable(pageNumber, logicalAddress, offset, i, pageTable, physicalMemory, outputFile):

for k in range(len(pageTable)):

if pageNumber == pageTable[k][0]:

print("Page Number \"" + str(pageNumber) + "\" found in page table!!")

frameNumber = pageTable[k][1]

data = part2.readPhysicalMemory(frameNumber, offset, physicalMemory)

physicalAddress = "{0:08b}".format(int(frameNumber)) + "{0:08b}".format(offset)

physicalAddress = int(physicalAddress, 2)

outStr = str(i) + " Virtual address: " + str(logicalAddress) + " Physical address: " + str(

physicalAddress) + " Value: " + data + "\n"

print(str(i) + " Virtual address: " + str(logicalAddress) + " Physical address: " + str(

physicalAddress) + " Value: " + data)

outputFile.write(outStr)

part2.updatepageTableCounter(k, pageTable)

return 1

return 0

Part 2.py

# import struct

def pageFaultHandler(pageNumber, tlb, pageTable, physicalMemory):

if int(pageNumber) < 256:

for i in range(256):

if i in physicalMemory.keys():

continue

else:

frameNumber = str(i)

break

backStore = open("BACKING\_STORE.bin", "rb")

physicalMemory[int(frameNumber)] = []

for i in range(256):

backStore.seek(int(pageNumber)\*256+i)

data = str(int.from\_bytes(backStore.read(1), byteorder='big', signed=True))

# data = struct.unpack("B", backStore.read(1))

# physicalMemory[int(pageNumber)].insert(i, str(data))

physicalMemory[int(frameNumber)].insert(i, data)

backStore.close()

print('Found page \"' + str(pageNumber) + '\" has data: ')

print(physicalMemory[int(frameNumber)])

print('in the backing store!\n')

else:

print('Page \"' + pageNumber + '\" is out of bound!')

return

updateTLB(pageNumber, frameNumber, tlb)

updatePageTable(pageNumber, frameNumber, pageTable)

def updateTLB(pageNumber, frameNumber, tlb):

# remove list[0], append new item at the end

if len(tlb) < 16:

tlb.append([pageNumber, frameNumber])

else:

# FIFO

tlb.pop(0)

tlb.append([pageNumber, frameNumber])

print('Successfully update TLB with pageNumber: ' + str(pageNumber) + ', frameNumber: ' + str(frameNumber) + '!')

def updatePageTable(pageNumber, frameNumber, pageTable):

# remove list[0], append new item at the end

if len(pageTable) < 256:

pageTable.append([pageNumber, frameNumber])

else:

pageTable.pop(0)

pageTable.append([pageNumber, frameNumber])

print('Successfully update pageTable table with pageNumber: ' + str(pageNumber) + ', frameNumber: ' + str(frameNumber) + '!')

def updateTLBCounter(latestEntryIndex, tlb):

# remove list[latestEntryIndex], append new item at the end

latestEntry = tlb[latestEntryIndex]

tlb.pop(latestEntryIndex)

tlb.append(latestEntry)

print('Successfully update TLB with new sequence using LRU!')

def updatepageTableCounter(latestEntryIndex, pageTable):

# remove list[latestEntryIndex], append new item at the end

latestEntry = pageTable[latestEntryIndex]

pageTable.pop(latestEntryIndex)

pageTable.append(latestEntry)

print('Successfully update page table with new sequence using LRU!')

def readPhysicalMemory(frameNumber, offset, physicalMemory):

if (int(frameNumber) < 256) and (int(offset) < 256):

data = physicalMemory[int(frameNumber)][int(offset)]

print('Successfully read frameNumber \"' + str(frameNumber) + '\" offset \"' + str(offset) + '\"\'s data ')

print(data)

print('in the physical memory!\n')

return data

else:

print('Frame number or offset is out of bound')

## Appendix B. Output.txt

1 Virtual address: 16916 Physical address: 20 Value: 0

2 Virtual address: 62493 Physical address: 285 Value: 0

3 Virtual address: 30198 Physical address: 758 Value: 29

4 Virtual address: 53683 Physical address: 947 Value: 108

5 Virtual address: 40185 Physical address: 1273 Value: 0

6 Virtual address: 28781 Physical address: 1389 Value: 0

7 Virtual address: 24462 Physical address: 1678 Value: 23

8 Virtual address: 48399 Physical address: 1807 Value: 67

9 Virtual address: 64815 Physical address: 2095 Value: 75

10 Virtual address: 18295 Physical address: 2423 Value: -35

11 Virtual address: 12218 Physical address: 2746 Value: 11

12 Virtual address: 22760 Physical address: 3048 Value: 0

13 Virtual address: 57982 Physical address: 3198 Value: 56

14 Virtual address: 27966 Physical address: 3390 Value: 27

15 Virtual address: 54894 Physical address: 3694 Valu

……….

………..

991 Virtual address: 48065 Physical address: 25793 Value: 0

992 Virtual address: 6957 Physical address: 26413 Value: 0

993 Virtual address: 2301 Physical address: 35325 Value: 0

994 Virtual address: 7736 Physical address: 57912 Value: 0

995 Virtual address: 31260 Physical address: 23324 Value: 0

996 Virtual address: 17071 Physical address: 175 Value: -85

997 Virtual address: 8940 Physical address: 46572 Value: 0

998 Virtual address: 9929 Physical address: 44745 Value: 0

999 Virtual address: 45563 Physical address: 46075 Value: 126

1000 Virtual address: 12107 Physical address: 2635 Value: -46

Number of translated address: 1000

Number of page fault: 244

Page fault rate: 0.244

Number of TLB hits: 56

TLB hit rate: 0.056

**Reference:**

Operating System Concepts - Abraham Silberschatz, Peter Baer Galvin, Greg Gagne