CS 342302 Operating Systems

Fall Semester 2021

Prof. Pai H. Chou

Weekly Review 11

Scope: Chapter 10, Virtual Memory

* + simplevm.py, typescript

## 1. Definitions and Short Answers

1. Of the several benefits of virtual memory, why is partial program loading a good idea?
   1. How does it improve performance (run faster)?
   2. How does it use less physical memory?
   3. How does it allow more processes to run at the same time?

A:

1. Another benefit of virtual memory is that it allows the logical address space to be larger than physical. Why is this a good idea?

A: It is a good idea because of the virtual address space abstraction presented to program. The program does not have to worry of running out of memory (2^64 bytes) and views the address space contiguously from address 0 (usually) until the end of space.

1. What is the difference between **paging** and **swapping**?

A: Paging is a physical memory allocation procedure for a process while swapping swaps an entire process between main memory and backing store or disk for with another process. A specialized kind of swapper is a pager where instead of swapping the entire process, only specific pages are swapped between main memory and disk (page-in and page-out).

1. What does **demand paging** mean? What are its benefits compared to non-demand-paging?

A: In demand paging we bring pages into memory only when needed (lazy evaluation). Page could be program code or data. Benefits: Less I/O needed, less memory needed, faster response and thus more users.

1. When the **valid-invalid bit** in the page table entry is 'v', what does it mean for demand paging?

A: Means page is in memory. Proceed normally.

1. When the valid-invalid bit is 'i' in the page table entry, what are two possibilities? How does the OS handle them?

A: Two possibilities: 1) Page is an invalid reference -> Abort 2) Page is not in memory, call the page handler and page-in into to memory. How do we know which possibility? With additional data structures such as PCB.

1. What happens on a page fault in demand paging? Does the OS or CPU handle. it like a regular interrupt? What is a subtle but important difference in how user code is resumed?

A: Upon page fault a trap to the OS is made. Then the OS looks at another. table (PCB) and decides if it is an invalid reference or a non-resident page. The address of the current instruction is pushed into the stack instead of the next instruction.

1. What does **zero-fill-on-demand** mean? When is it used and why?

A: Zero-fill-on demand means to initialize an entire page to 0 when it is to be allocated. Used due to privacy concerns. We don’t want previous data from previous users of that frame.

1. Of the costs of page fault, which has the highest cost?
   1. servicing the interrupt
   2. restart the process
   3. page-in, page-out
2. What is **copy-on-write** and how does it help make fork() run more efficiently?

A: In copy-on write, pages are copied only when either process (child or parent) modifies or writes a shared page.

It helps make fork() run more efficiently because the OS does not have to make a copy of a page for the child until it is needed (lazy-evaluation).

1. Does vfork() use copy on write? How does it work for a shell to launch a process?

A: No, it doesn’t use copy-on-write. It is used for implementing command-line shells. After creation, child calls exec() and sharing is stopped. Child does not modify parent’s (shell) address space.

1. What does a **frame allocation algorithm** decide? How many frames to allocate to a process?

A: **Determines how many** frames to allocate to a process. Could be fixed or flexible.

1. What does a **page replacement algorithm** decide? Which frame to replace when no frames are available?

A: Finds a victim page in memory to page out and frees the frame.

1. What is the Belady's Anomaly?

A: In Belady’s anomaly you get more page faults when you have more frames. It is common in FIFO-based algorithms.

1. What is the Optimal page replacement algorithm? Can it be implemented exactly?

A: In the Optimal page replacement algorithm, page that will not be used for the longest period. No, it can’t be implemented exactly since we cannot look ahead.

1. What is a good page replacement algorithm in practice?

A: The least recently used algorithm is a good page algorithm in practice.

1. To implement LRU,
   1. What does a Counter store, when is it updated, and how is it used?

A: A counter stores the clock value of the page. It is updated every time the page is referenced through the page entry with the clock value. It is used to determine the least recently used page.

* 1. How does a stack algorithm work when the page being referenced is already on the stack?

A: The algorithm needs to find the page being referenced in the stack unlink it and push it to the top of the stack. (Doubly linked list).

1. Of LRU approximation algorithms,
   1. In **single reference bit algorithm**, what page gets replaced?

A: Pages with any reference bit with value 0 get replaced. May be many of them. Very rough approximation.

* 1. In **additional reference bits algorithm**, how are multiple reference bits for each page maintained?

A: With a sampled update (e.g. sample every 100 ms). The page with the smallest sampled update is chosen as a victim. However, multiple pages may have the same history so other considerations must be made.

* 1. How does **second-chance algorithm** decide what page to replace? What update to reference bit does it do under what condition?

A: It checks the page’s reference bit and if it is 0, it is replaced. Otherwise, the page’s reference bit is updated to 0, this is the second chance, and the page is left in memory.

* 1. in **enhanced second chance** algorithm, what are the four combinations in order of replacement priority? (Reference bit, modify bit)

A: (reference bit, modified bit): (0,0) -> Neither recently used nor modified. Best Choice

(0,1) -> Not recently used but modified. Must page out before replacement.

(1,0) -> Recently used but clean – Probably will be used again soon.

(1,1) -> Recently used but modified – Probably will be used again soon and need to write out before replacement.

1. In Counting-based algorithms,
   1. what is the issue with least-frequently used (LFU) algorithm?

A: Access may be heavy un startup but rarely used after. With large counts, it is harder to replace.

* 1. What is the rationale behind most frequently used (MFU) algorithm?

A: Page with smallest count? -> Probably just brought in and yet hasn’t been used.

1. In Page-buffering,
   1. How does it make sure there are always free frames available by the time a page fault occurs?

A: A victim is evicted upon a free frame being allocated to a page. This isn’t immediate but when it is deemed convenient.

* 1. An extended version keeps a list of modified pages. What is the purpose?

A: To make them clean when backing store idle.

* 1. Another extended version is to retain the frame contents even when the frame is put on the free frame list. What is the purpose?

A: If referenced again before used, there is no need to load contents again from disk.

1. What are two schemes for **fixed allocation** of frames to processes?

A: Fixed allocation and priority allocation.

1. In **priority allocation** of frames, upon page fault, how would OS **select victims** to reflect priority of processes?

A: OS would select victim based on its priority. So, we select the victim with the lowest priority.

1. Between **global replacement** and **local replacement**,
   1. What are the advantages and disadvantages of global replacement?

A: Advantages: Greater throughput, better utilization.

Disadvantage: Less predictable execution time.

* 1. What are the advantages and disadvantages of local replacement?

A: Advantage: More consistent per-process performance.

Disadvantage: Possibly underutilized memory. Page number needs

may change.

1. What is the difference between a **major page fault** and a **minor page fault**?

A: The difference is where the page is. In a major page fault, page is not in memory while in a minor page fault, page is in memory, but it hasn’t been mapped yet.

1. What is the purpose of a **reaper**?

A: The purpose of a reaper (kernel routine) is to reclaim pages when the amount of free memory drops below some threshold and adds frames to the free frame list.

1. What is the meaning of **thrashing**? What causes thrashing?

A: Thrashing means a process is busy swapping pages in and out. It doesn’t get real work done. The main cause is sub-allocation of frames to a process and thus causing a high-page fault.

1. How does the **working set model** approximate locality? And what condition implies thrashing?

A: Locality D is approximated as the sum of the total demand frames WSSi.

If D > m (total available # of frames), thrashing occurs. Processes must be suspended or swapped out (policy example).

1. In the page fault frequency (PFF) strategy, what does it mean when the page fault rate is too high and what does the OS do? Too low?

A: Too high? -> OS needs increase number of frames.

Too low ? -> OS needs to decrease number of frames.

1. What is different between physical memory allocation **for kernel** vs. for user process?

A: Physical memory allocated for the kernel MUST be physically contiguous. Efficiency is required and thus paging is NOT an option.

1. How does the buddy system decide how much memory to allocate to serve a request? What are its advantages and disadvantages?

A: Allocate a chunk that is the smallest power of 2 that is big enough to handle the request.

Advantage: Quickly coalesce unused chunks into larger chunk.

Disadvantage: May cause internal fragmentation.

1. In SLAB allocation,
   1. What is a slab and how big does it have to be?? What does a slab contain?

A: A slab is one or more contiguous physical contiguous pages. It must be big enough to contain one or more instances of a kernel data structure.

* 1. How many **caches** are there? What does a cache contain?

A: One cache per kernel data structure. Initially a cache contains objects marked as free, once a data structure is allocated, the object is marked as not free.

* 1. Why is there **no fragmentation** in slab scheme?

A: Because the granularity is object based, not based on pages or buddy chunks.

## 2. Programming Exercise

In this programming exercise, you are to implement paging algorithms for OPT, FIFO, LRU, and SecondChance in Python. To do this, create a SimpleVM class using the following [template](https://drive.google.com/file/d/13-woRgRBegJUVea5aUULqb8ePhytb1b3/view?usp=sharing) and rename it **simplevm.py**.

class SimpleVM:

\_ReplacementPolicies = ['OPT', 'LRU', 'FIFO', 'SecondChance']

def \_\_init\_\_(self, numPages, numFrames, replacementPolicy):

self.numPages = numPages

self.numFrames = numFrames

if not replacementPolicy in SimpleVM.\_ReplacementPolicies:

raise ValueError('Unknown replacement policy %s' % replacementPolicy)

self.replacementPolicy = replacementPolicy

self.pageTable = [None for i in range(numPages)]

self.valid = ['i' for i in range(numPages)]

self.frames = [None for i in range(numFrames)] # storage

self.dirty = [False for i in range(numFrames)]

# we prefill swapspace content with chars '0','1','2'..  
 self.swapSpace = [chr(ord('0')+i) for i in range(numPages)]  
 # a frameTable maps a frame to the page, if any.

self.frameTable = [None for i in range(numFrames)]

# policy-specific code here

if self.replacementPolicy == 'LRU':

# use a "stack" (really more like a queue) to track age.

self.stack = []

if self.replacementPolicy in ['FIFO', 'SecondChance']:

# both FIFO and SecondChance are somewhat like RR

# so you could either use a circular buffer or perhaps  
 # keep index. Your own code here!

if self.replacementPolicy == 'SecondChance':

self.reference = [False for i in range(numFrames)]

def getFreeFrame(self, pageNum):

# find a free frame if any, or return None if not found.  
 # see comment in template for more info.

for i in range(self.numFrames):

if self.frames[i] is None:

return i

return None

def pickVictim(self, future=None):

# finds a page whose frame is to be evicted to fulfill page fault.

# this is called only if getFreeFrame returns None

if self.replacementPolicy == 'OPT':

# use future knowledge to pick victim

if future is None:

raise ValueError('cannot pick OPT without future')

# Your code here!!!

# find page that won't be used for longest time in future

# Note if future is empty list, then any page is ok!

# in any case, return the victim page's frame number.

if self.replacementPolicy == 'LRU':

# Your code here!! pull the victim from the bottom of the stack

# the assumption is if we have free frame in the first place,

# we would not need to evict anybody.

if self.replacementPolicy == 'FIFO':

# Your code here!  
 # pick victim in FIFO order

if self.replacementPolicy == 'SecondChance':

# Your code here!!

# base on referenceBit

# if we have not returned by then, it is an unknown policy

raise ValueError('unknown poliy %s' % self.replacementPolicy)

def pageIn(self, frameNum, pageNum):

# Your code here!!  
 # called to bring in a page from swap space to the frame.

# pageNum is used to find location in swap space.

# for simplicity, we use pageNum to index into swap space.

# Assume frameNum is free, and thus no page is currently using it.

# Update the page-to-frame table and frame-to-page table,

# set valid bit for the page, and clear dirty bit for the frame.

# in case of SecondChance, also clear reference bit.

def pageOut(self, frameNum):

# Your code here!!

# this flushes a frame (for a given pageNum) to swap space.

# Note that we only mark it as not-dirty, but it does not

# change state of valid bit because that is someone else's decision

# whether they want to reclaim the page or just flush it.

# Similar to pageIn, we assume swap space uses the virtual address

# as we have only one process.

def getFrame(self, pageNum, future=None):

# this is a utility that may be helpful, but not required.

# - see if pageHit, if so, return valid frame # for read/write.

# - if pageFault,

# - see if free frame available; if so, grab it;

# - but if no free frame, pick victim, page out first,

# fall thru to page-in

# - page-in and return the frame number

# - bookkeeping: look up the page# whose frame will be reassigned

# - set its pageTable entry to None, clear that page's valid bit

# finally, return the frame number for caller to use.

def updateAccess(self, frameNum, write=False):

# Your own code!! in different cases below!

if self.replacementPolicy == 'LRU':

# Your code here!! find frame in stack; if found, pop it.

# in either case, push back on stack.

if self.replacementPolicy == 'SecondChance':

# Your own code!! - mark the reference bit

if write: # for future use, if supporting write-access

self.dirty[frameNum] = True

def readPage(self, pageNum, future=None):

# Your code here!!

# get the frame number -- can call the getFrame() method for this.

# use the frame number to get the data so we can return it.

# do some bookkeeping by calling updateAccess

def writePage(self, pageNum, data, future=None):

# Your code here!!

# analogous to the readPage, except

# the frame is written to with data.

# do bookkeeping with write=True

You will find the test cases in the template file.

Here is a sample output of the program:

$ python3 simplevm.py

-------------- policy (read): OPT--------------

readPage(7)='7', pageTable=[None, None, None, None, None, None, None, 0], valid=iiiiiiiv, frames=['7', None, None]

readPage(0)='0', pageTable=[1, None, None, None, None, None, None, 0], valid=viiiiiiv, frames=['7', '0', None]

readPage(1)='1', pageTable=[1, 2, None, None, None, None, None, 0], valid=vviiiiiv, frames=['7', '0', '1']

readPage(2)='2', pageTable=[1, 2, 0, None, None, None, None, None], valid=vvviiiii, frames=['2', '0', '1']

readPage(0)='0', pageTable=[1, 2, 0, None, None, None, None, None], valid=vvviiiii, frames=['2', '0', '1']

readPage(3)='3', pageTable=[1, None, 0, 2, None, None, None, None], valid=vivviiii, frames=['2', '0', '3']

readPage(0)='0', pageTable=[1, None, 0, 2, None, None, None, None], valid=vivviiii, frames=['2', '0', '3']

readPage(4)='4', pageTable=[None, None, 0, 2, 1, None, None, None], valid=iivvviii, frames=['2', '4', '3']

readPage(2)='2', pageTable=[None, None, 0, 2, 1, None, None, None], valid=iivvviii, frames=['2', '4', '3']

readPage(3)='3', pageTable=[None, None, 0, 2, 1, None, None, None], valid=iivvviii, frames=['2', '4', '3']

readPage(0)='0', pageTable=[1, None, 0, 2, None, None, None, None], valid=vivviiii, frames=['2', '0', '3']

readPage(3)='3', pageTable=[1, None, 0, 2, None, None, None, None], valid=vivviiii, frames=['2', '0', '3']

readPage(0)='0', pageTable=[1, None, 0, 2, None, None, None, None], valid=vivviiii, frames=['2', '0', '3']

readPage(3)='3', pageTable=[1, None, 0, 2, None, None, None, None], valid=vivviiii, frames=['2', '0', '3']

readPage(2)='2', pageTable=[1, None, 0, 2, None, None, None, None], valid=vivviiii, frames=['2', '0', '3']

readPage(1)='1', pageTable=[1, 2, 0, None, None, None, None, None], valid=vvviiiii, frames=['2', '0', '1']

readPage(2)='2', pageTable=[1, 2, 0, None, None, None, None, None], valid=vvviiiii, frames=['2', '0', '1']

readPage(0)='0', pageTable=[1, 2, 0, None, None, None, None, None], valid=vvviiiii, frames=['2', '0', '1']

readPage(1)='1', pageTable=[1, 2, 0, None, None, None, None, None], valid=vvviiiii, frames=['2', '0', '1']

readPage(7)='7', pageTable=[1, 2, None, None, None, None, None, 0], valid=vviiiiiv, frames=['7', '0', '1']

readPage(0)='0', pageTable=[1, 2, None, None, None, None, None, 0], valid=vviiiiiv, frames=['7', '0', '1']

readPage(1)='1', pageTable=[1, 2, None, None, None, None, None, 0], valid=vviiiiiv, frames=['7', '0', '1']

page faults = 9, page ins = 9, page outs = 0

-------------- policy (write): OPT--------------

writePage(7, 'A'), frames=['A', None, None], swapSpace=['0', '1', '2', '3', '4', '5', '6', '7']

writePage(0, 'B'), frames=['A', 'B', None], swapSpace=['0', '1', '2', '3', '4', '5', '6', '7']

writePage(1, 'C'), frames=['A', 'B', 'C'], swapSpace=['0', '1', '2', '3', '4', '5', '6', '7']

writePage(2, 'D'), frames=['D', 'B', 'C'], swapSpace=['0', '1', '2', '3', '4', '5', '6', 'A']

writePage(0, 'E'), frames=['D', 'E', 'C'], swapSpace=['0', '1', '2', '3', '4', '5', '6', 'A']

writePage(3, 'F'), frames=['D', 'E', 'F'], swapSpace=['0', 'C', '2', '3', '4', '5', '6', 'A']

writePage(0, 'G'), frames=['D', 'G', 'F'], swapSpace=['0', 'C', '2', '3', '4', '5', '6', 'A']

writePage(4, 'H'), frames=['D', 'H', 'F'], swapSpace=['G', 'C', '2', '3', '4', '5', '6', 'A']

writePage(2, 'I'), frames=['I', 'H', 'F'], swapSpace=['G', 'C', '2', '3', '4', '5', '6', 'A']

writePage(3, 'J'), frames=['I', 'H', 'J'], swapSpace=['G', 'C', '2', '3', '4', '5', '6', 'A']

writePage(0, 'K'), frames=['I', 'K', 'J'], swapSpace=['G', 'C', '2', '3', 'H', '5', '6', 'A']

writePage(3, 'L'), frames=['I', 'K', 'L'], swapSpace=['G', 'C', '2', '3', 'H', '5', '6', 'A']

writePage(0, 'M'), frames=['I', 'M', 'L'], swapSpace=['G', 'C', '2', '3', 'H', '5', '6', 'A']

writePage(3, 'N'), frames=['I', 'M', 'N'], swapSpace=['G', 'C', '2', '3', 'H', '5', '6', 'A']

writePage(2, 'O'), frames=['O', 'M', 'N'], swapSpace=['G', 'C', '2', '3', 'H', '5', '6', 'A']

writePage(1, 'P'), frames=['O', 'M', 'P'], swapSpace=['G', 'C', '2', 'N', 'H', '5', '6', 'A']

writePage(2, 'Q'), frames=['Q', 'M', 'P'], swapSpace=['G', 'C', '2', 'N', 'H', '5', '6', 'A']

writePage(0, 'R'), frames=['Q', 'R', 'P'], swapSpace=['G', 'C', '2', 'N', 'H', '5', '6', 'A']

writePage(1, 'S'), frames=['Q', 'R', 'S'], swapSpace=['G', 'C', '2', 'N', 'H', '5', '6', 'A']

writePage(7, 'T'), frames=['T', 'R', 'S'], swapSpace=['G', 'C', 'Q', 'N', 'H', '5', '6', 'A']

writePage(0, 'U'), frames=['T', 'U', 'S'], swapSpace=['G', 'C', 'Q', 'N', 'H', '5', '6', 'A']

writePage(1, 'V'), frames=['T', 'U', 'V'], swapSpace=['G', 'C', 'Q', 'N', 'H', '5', '6', 'A']

page faults = 9, page ins = 9, page outs = 6

-------------- policy (read): LRU--------------

readPage(7)='7', pageTable=[None, None, None, None, None, None, None, 0], valid=iiiiiiiv, frames=['7', None, None]

readPage(0)='0', pageTable=[1, None, None, None, None, None, None, 0], valid=viiiiiiv, frames=['7', '0', None]

readPage(1)='1', pageTable=[1, 2, None, None, None, None, None, 0], valid=vviiiiiv, frames=['7', '0', '1']

readPage(2)='2', pageTable=[1, 2, 0, None, None, None, None, None], valid=vvviiiii, frames=['2', '0', '1']

readPage(0)='0', pageTable=[1, 2, 0, None, None, None, None, None], valid=vvviiiii, frames=['2', '0', '1']

readPage(3)='3', pageTable=[1, None, 0, 2, None, None, None, None], valid=vivviiii, frames=['2', '0', '3']

readPage(0)='0', pageTable=[1, None, 0, 2, None, None, None, None], valid=vivviiii, frames=['2', '0', '3']

readPage(4)='4', pageTable=[1, None, None, 2, 0, None, None, None], valid=viivviii, frames=['4', '0', '3']

readPage(2)='2', pageTable=[1, None, 2, None, 0, None, None, None], valid=viviviii, frames=['4', '0', '2']

readPage(3)='3', pageTable=[None, None, 2, 1, 0, None, None, None], valid=iivvviii, frames=['4', '3', '2']

readPage(0)='0', pageTable=[0, None, 2, 1, None, None, None, None], valid=vivviiii, frames=['0', '3', '2']

readPage(3)='3', pageTable=[0, None, 2, 1, None, None, None, None], valid=vivviiii, frames=['0', '3', '2']

readPage(0)='0', pageTable=[0, None, 2, 1, None, None, None, None], valid=vivviiii, frames=['0', '3', '2']

readPage(3)='3', pageTable=[0, None, 2, 1, None, None, None, None], valid=vivviiii, frames=['0', '3', '2']

readPage(2)='2', pageTable=[0, None, 2, 1, None, None, None, None], valid=vivviiii, frames=['0', '3', '2']

readPage(1)='1', pageTable=[None, 0, 2, 1, None, None, None, None], valid=ivvviiii, frames=['1', '3', '2']

readPage(2)='2', pageTable=[None, 0, 2, 1, None, None, None, None], valid=ivvviiii, frames=['1', '3', '2']

readPage(0)='0', pageTable=[1, 0, 2, None, None, None, None, None], valid=vvviiiii, frames=['1', '0', '2']

readPage(1)='1', pageTable=[1, 0, 2, None, None, None, None, None], valid=vvviiiii, frames=['1', '0', '2']

readPage(7)='7', pageTable=[1, 0, None, None, None, None, None, 2], valid=vviiiiiv, frames=['1', '0', '7']

readPage(0)='0', pageTable=[1, 0, None, None, None, None, None, 2], valid=vviiiiiv, frames=['1', '0', '7']

readPage(1)='1', pageTable=[1, 0, None, None, None, None, None, 2], valid=vviiiiiv, frames=['1', '0', '7']

page faults = 12, page ins = 12, page outs = 0

-------------- policy (write): LRU--------------

writePage(7, 'A'), frames=['A', None, None], swapSpace=['0', '1', '2', '3', '4', '5', '6', '7']

writePage(0, 'B'), frames=['A', 'B', None], swapSpace=['0', '1', '2', '3', '4', '5', '6', '7']

writePage(1, 'C'), frames=['A', 'B', 'C'], swapSpace=['0', '1', '2', '3', '4', '5', '6', '7']

writePage(2, 'D'), frames=['D', 'B', 'C'], swapSpace=['0', '1', '2', '3', '4', '5', '6', 'A']

writePage(0, 'E'), frames=['D', 'E', 'C'], swapSpace=['0', '1', '2', '3', '4', '5', '6', 'A']

writePage(3, 'F'), frames=['D', 'E', 'F'], swapSpace=['0', 'C', '2', '3', '4', '5', '6', 'A']

writePage(0, 'G'), frames=['D', 'G', 'F'], swapSpace=['0', 'C', '2', '3', '4', '5', '6', 'A']

writePage(4, 'H'), frames=['H', 'G', 'F'], swapSpace=['0', 'C', 'D', '3', '4', '5', '6', 'A']

writePage(2, 'I'), frames=['H', 'G', 'I'], swapSpace=['0', 'C', 'D', 'F', '4', '5', '6', 'A']

writePage(3, 'J'), frames=['H', 'J', 'I'], swapSpace=['G', 'C', 'D', 'F', '4', '5', '6', 'A']

writePage(0, 'K'), frames=['K', 'J', 'I'], swapSpace=['G', 'C', 'D', 'F', 'H', '5', '6', 'A']

writePage(3, 'L'), frames=['K', 'L', 'I'], swapSpace=['G', 'C', 'D', 'F', 'H', '5', '6', 'A']

writePage(0, 'M'), frames=['M', 'L', 'I'], swapSpace=['G', 'C', 'D', 'F', 'H', '5', '6', 'A']

writePage(3, 'N'), frames=['M', 'N', 'I'], swapSpace=['G', 'C', 'D', 'F', 'H', '5', '6', 'A']

writePage(2, 'O'), frames=['M', 'N', 'O'], swapSpace=['G', 'C', 'D', 'F', 'H', '5', '6', 'A']

writePage(1, 'P'), frames=['P', 'N', 'O'], swapSpace=['M', 'C', 'D', 'F', 'H', '5', '6', 'A']

writePage(2, 'Q'), frames=['P', 'N', 'Q'], swapSpace=['M', 'C', 'D', 'F', 'H', '5', '6', 'A']

writePage(0, 'R'), frames=['P', 'R', 'Q'], swapSpace=['M', 'C', 'D', 'N', 'H', '5', '6', 'A']

writePage(1, 'S'), frames=['S', 'R', 'Q'], swapSpace=['M', 'C', 'D', 'N', 'H', '5', '6', 'A']

writePage(7, 'T'), frames=['S', 'R', 'T'], swapSpace=['M', 'C', 'Q', 'N', 'H', '5', '6', 'A']

writePage(0, 'U'), frames=['S', 'U', 'T'], swapSpace=['M', 'C', 'Q', 'N', 'H', '5', '6', 'A']

writePage(1, 'V'), frames=['V', 'U', 'T'], swapSpace=['M', 'C', 'Q', 'N', 'H', '5', '6', 'A']

page faults = 12, page ins = 12, page outs = 9

-------------- policy (read): FIFO--------------

readPage(7)='7', pageTable=[None, None, None, None, None, None, None, 0], valid=iiiiiiiv, frames=['7', None, None]

readPage(0)='0', pageTable=[1, None, None, None, None, None, None, 0], valid=viiiiiiv, frames=['7', '0', None]

readPage(1)='1', pageTable=[1, 2, None, None, None, None, None, 0], valid=vviiiiiv, frames=['7', '0', '1']

readPage(2)='2', pageTable=[1, 2, 0, None, None, None, None, None], valid=vvviiiii, frames=['2', '0', '1']

readPage(0)='0', pageTable=[1, 2, 0, None, None, None, None, None], valid=vvviiiii, frames=['2', '0', '1']

readPage(3)='3', pageTable=[None, 2, 0, 1, None, None, None, None], valid=ivvviiii, frames=['2', '3', '1']

readPage(0)='0', pageTable=[2, None, 0, 1, None, None, None, None], valid=vivviiii, frames=['2', '3', '0']

readPage(4)='4', pageTable=[2, None, None, 1, 0, None, None, None], valid=viivviii, frames=['4', '3', '0']

readPage(2)='2', pageTable=[2, None, 1, None, 0, None, None, None], valid=viviviii, frames=['4', '2', '0']

readPage(3)='3', pageTable=[None, None, 1, 2, 0, None, None, None], valid=iivvviii, frames=['4', '2', '3']

readPage(0)='0', pageTable=[0, None, 1, 2, None, None, None, None], valid=vivviiii, frames=['0', '2', '3']

readPage(3)='3', pageTable=[0, None, 1, 2, None, None, None, None], valid=vivviiii, frames=['0', '2', '3']

readPage(0)='0', pageTable=[0, None, 1, 2, None, None, None, None], valid=vivviiii, frames=['0', '2', '3']

readPage(3)='3', pageTable=[0, None, 1, 2, None, None, None, None], valid=vivviiii, frames=['0', '2', '3']

readPage(2)='2', pageTable=[0, None, 1, 2, None, None, None, None], valid=vivviiii, frames=['0', '2', '3']

readPage(1)='1', pageTable=[0, 1, None, 2, None, None, None, None], valid=vviviiii, frames=['0', '1', '3']

readPage(2)='2', pageTable=[0, 1, 2, None, None, None, None, None], valid=vvviiiii, frames=['0', '1', '2']

readPage(0)='0', pageTable=[0, 1, 2, None, None, None, None, None], valid=vvviiiii, frames=['0', '1', '2']

readPage(1)='1', pageTable=[0, 1, 2, None, None, None, None, None], valid=vvviiiii, frames=['0', '1', '2']

readPage(7)='7', pageTable=[None, 1, 2, None, None, None, None, 0], valid=ivviiiiv, frames=['7', '1', '2']

readPage(0)='0', pageTable=[1, None, 2, None, None, None, None, 0], valid=viviiiiv, frames=['7', '0', '2']

readPage(1)='1', pageTable=[1, 2, None, None, None, None, None, 0], valid=vviiiiiv, frames=['7', '0', '1']

page faults = 15, page ins = 15, page outs = 0

-------------- policy (write): FIFO--------------

writePage(7, 'A'), frames=['A', None, None], swapSpace=['0', '1', '2', '3', '4', '5', '6', '7']

writePage(0, 'B'), frames=['A', 'B', None], swapSpace=['0', '1', '2', '3', '4', '5', '6', '7']

writePage(1, 'C'), frames=['A', 'B', 'C'], swapSpace=['0', '1', '2', '3', '4', '5', '6', '7']

writePage(2, 'D'), frames=['D', 'B', 'C'], swapSpace=['0', '1', '2', '3', '4', '5', '6', 'A']

writePage(0, 'E'), frames=['D', 'E', 'C'], swapSpace=['0', '1', '2', '3', '4', '5', '6', 'A']

writePage(3, 'F'), frames=['D', 'F', 'C'], swapSpace=['E', '1', '2', '3', '4', '5', '6', 'A']

writePage(0, 'G'), frames=['D', 'F', 'G'], swapSpace=['E', 'C', '2', '3', '4', '5', '6', 'A']

writePage(4, 'H'), frames=['H', 'F', 'G'], swapSpace=['E', 'C', 'D', '3', '4', '5', '6', 'A']

writePage(2, 'I'), frames=['H', 'I', 'G'], swapSpace=['E', 'C', 'D', 'F', '4', '5', '6', 'A']

writePage(3, 'J'), frames=['H', 'I', 'J'], swapSpace=['G', 'C', 'D', 'F', '4', '5', '6', 'A']

writePage(0, 'K'), frames=['K', 'I', 'J'], swapSpace=['G', 'C', 'D', 'F', 'H', '5', '6', 'A']

writePage(3, 'L'), frames=['K', 'I', 'L'], swapSpace=['G', 'C', 'D', 'F', 'H', '5', '6', 'A']

writePage(0, 'M'), frames=['M', 'I', 'L'], swapSpace=['G', 'C', 'D', 'F', 'H', '5', '6', 'A']

writePage(3, 'N'), frames=['M', 'I', 'N'], swapSpace=['G', 'C', 'D', 'F', 'H', '5', '6', 'A']

writePage(2, 'O'), frames=['M', 'O', 'N'], swapSpace=['G', 'C', 'D', 'F', 'H', '5', '6', 'A']

writePage(1, 'P'), frames=['M', 'P', 'N'], swapSpace=['G', 'C', 'O', 'F', 'H', '5', '6', 'A']

writePage(2, 'Q'), frames=['M', 'P', 'Q'], swapSpace=['G', 'C', 'O', 'N', 'H', '5', '6', 'A']

writePage(0, 'R'), frames=['R', 'P', 'Q'], swapSpace=['G', 'C', 'O', 'N', 'H', '5', '6', 'A']

writePage(1, 'S'), frames=['R', 'S', 'Q'], swapSpace=['G', 'C', 'O', 'N', 'H', '5', '6', 'A']

writePage(7, 'T'), frames=['T', 'S', 'Q'], swapSpace=['R', 'C', 'O', 'N', 'H', '5', '6', 'A']

writePage(0, 'U'), frames=['T', 'U', 'Q'], swapSpace=['R', 'S', 'O', 'N', 'H', '5', '6', 'A']

writePage(1, 'V'), frames=['T', 'U', 'V'], swapSpace=['R', 'S', 'Q', 'N', 'H', '5', '6', 'A']

page faults = 15, page ins = 15, page outs = 12

-------------- policy (read): SecondChance--------------

readPage(7)='7', pageTable=[None, None, None, None, None, None, None, 0], valid=iiiiiiiv, frames=['7', None, None]

readPage(0)='0', pageTable=[1, None, None, None, None, None, None, 0], valid=viiiiiiv, frames=['7', '0', None]

readPage(1)='1', pageTable=[1, 2, None, None, None, None, None, 0], valid=vviiiiiv, frames=['7', '0', '1']

readPage(2)='2', pageTable=[1, 2, 0, None, None, None, None, None], valid=vvviiiii, frames=['2', '0', '1']

readPage(0)='0', pageTable=[1, 2, 0, None, None, None, None, None], valid=vvviiiii, frames=['2', '0', '1']

readPage(3)='3', pageTable=[1, None, 0, 2, None, None, None, None], valid=vivviiii, frames=['2', '0', '3']

readPage(0)='0', pageTable=[1, None, 0, 2, None, None, None, None], valid=vivviiii, frames=['2', '0', '3']

readPage(4)='4', pageTable=[1, None, None, 2, 0, None, None, None], valid=viivviii, frames=['4', '0', '3']

readPage(2)='2', pageTable=[None, None, 1, 2, 0, None, None, None], valid=iivvviii, frames=['4', '2', '3']

readPage(3)='3', pageTable=[None, None, 1, 2, 0, None, None, None], valid=iivvviii, frames=['4', '2', '3']

readPage(0)='0', pageTable=[2, None, 1, None, 0, None, None, None], valid=viviviii, frames=['4', '2', '0']

readPage(3)='3', pageTable=[2, None, 1, 0, None, None, None, None], valid=vivviiii, frames=['3', '2', '0']

readPage(0)='0', pageTable=[2, None, 1, 0, None, None, None, None], valid=vivviiii, frames=['3', '2', '0']

readPage(3)='3', pageTable=[2, None, 1, 0, None, None, None, None], valid=vivviiii, frames=['3', '2', '0']

readPage(2)='2', pageTable=[2, None, 1, 0, None, None, None, None], valid=vivviiii, frames=['3', '2', '0']

readPage(1)='1', pageTable=[2, 1, None, 0, None, None, None, None], valid=vviviiii, frames=['3', '1', '0']

readPage(2)='2', pageTable=[None, 1, 2, 0, None, None, None, None], valid=ivvviiii, frames=['3', '1', '2']

readPage(0)='0', pageTable=[0, 1, 2, None, None, None, None, None], valid=vvviiiii, frames=['0', '1', '2']

readPage(1)='1', pageTable=[0, 1, 2, None, None, None, None, None], valid=vvviiiii, frames=['0', '1', '2']

readPage(7)='7', pageTable=[0, None, 2, None, None, None, None, 1], valid=viviiiiv, frames=['0', '7', '2']

readPage(0)='0', pageTable=[0, None, 2, None, None, None, None, 1], valid=viviiiiv, frames=['0', '7', '2']

readPage(1)='1', pageTable=[0, 2, None, None, None, None, None, 1], valid=vviiiiiv, frames=['0', '7', '1']

page faults = 14, page ins = 14, page outs = 0

-------------- policy (write): SecondChance--------------

writePage(7, 'A'), frames=['A', None, None], swapSpace=['0', '1', '2', '3', '4', '5', '6', '7']

writePage(0, 'B'), frames=['A', 'B', None], swapSpace=['0', '1', '2', '3', '4', '5', '6', '7']

writePage(1, 'C'), frames=['A', 'B', 'C'], swapSpace=['0', '1', '2', '3', '4', '5', '6', '7']

writePage(2, 'D'), frames=['D', 'B', 'C'], swapSpace=['0', '1', '2', '3', '4', '5', '6', 'A']

writePage(0, 'E'), frames=['D', 'E', 'C'], swapSpace=['0', '1', '2', '3', '4', '5', '6', 'A']

writePage(3, 'F'), frames=['D', 'E', 'F'], swapSpace=['0', 'C', '2', '3', '4', '5', '6', 'A']

writePage(0, 'G'), frames=['D', 'G', 'F'], swapSpace=['0', 'C', '2', '3', '4', '5', '6', 'A']

writePage(4, 'H'), frames=['H', 'G', 'F'], swapSpace=['0', 'C', 'D', '3', '4', '5', '6', 'A']

writePage(2, 'I'), frames=['H', 'I', 'F'], swapSpace=['G', 'C', 'D', '3', '4', '5', '6', 'A']

writePage(3, 'J'), frames=['H', 'I', 'J'], swapSpace=['G', 'C', 'D', '3', '4', '5', '6', 'A']

writePage(0, 'K'), frames=['H', 'I', 'K'], swapSpace=['G', 'C', 'D', 'J', '4', '5', '6', 'A']

writePage(3, 'L'), frames=['L', 'I', 'K'], swapSpace=['G', 'C', 'D', 'J', 'H', '5', '6', 'A']

writePage(0, 'M'), frames=['L', 'I', 'M'], swapSpace=['G', 'C', 'D', 'J', 'H', '5', '6', 'A']

writePage(3, 'N'), frames=['N', 'I', 'M'], swapSpace=['G', 'C', 'D', 'J', 'H', '5', '6', 'A']

writePage(2, 'O'), frames=['N', 'O', 'M'], swapSpace=['G', 'C', 'D', 'J', 'H', '5', '6', 'A']

writePage(1, 'P'), frames=['N', 'P', 'M'], swapSpace=['G', 'C', 'O', 'J', 'H', '5', '6', 'A']

writePage(2, 'Q'), frames=['N', 'P', 'Q'], swapSpace=['M', 'C', 'O', 'J', 'H', '5', '6', 'A']

writePage(0, 'R'), frames=['R', 'P', 'Q'], swapSpace=['M', 'C', 'O', 'N', 'H', '5', '6', 'A']

writePage(1, 'S'), frames=['R', 'S', 'Q'], swapSpace=['M', 'C', 'O', 'N', 'H', '5', '6', 'A']

writePage(7, 'T'), frames=['R', 'T', 'Q'], swapSpace=['M', 'S', 'O', 'N', 'H', '5', '6', 'A']

writePage(0, 'U'), frames=['U', 'T', 'Q'], swapSpace=['M', 'S', 'O', 'N', 'H', '5', '6', 'A']

writePage(1, 'V'), frames=['U', 'T', 'V'], swapSpace=['M', 'S', 'Q', 'N', 'H', '5', '6', 'A']

page faults = 14, page ins = 14, page outs = 11