ASEE 2022 ANNUAL CONFERENCE
Excellence Through Diversity
MINNEAPOLIS, MINNESOTA, JUNE 26TH-29TH, 2022 ASEE

Paper ID #36640

Work in Progress: A Visualization Aid for Learning Virtual Memory Concepts

John A Nestor (Professor)

John Nestor is a Professor of Electrical Engineering at Lafayette College. He received the Ph. D. and MSEE degrees from Carnegie Mellon and the BEE degree from Georgia Tech. Prior to joining Lafayette, he was a faculty member at Illinois Institute of Technology. His interests include computer engineering, digital design, VLSI, engineering education, and the history of semiconductors and computers.

Zheping Yin

Zheping Yin is a Senior undergraduate student at Lafayette College. His research interests are computer engineering and VLSI design.

Work in Progress: A Visualization Aid for Learning Virtual Memory Concepts

Abstract

Virtual memory is a key feature in modern computer systems. A virtual memory system simulates a memory with a large virtual address space using a smaller physical memory coupled with a backing store such as a disk drive. Virtual memory employs a combination of processor hardware and operating systems software to translate virtual addresses to physical addresses and manage the movement of data between physical memory and disk. Virtual memory is a complex topic which students can find difficult to understand using the static diagrams found in textbooks and lecture notes.

This paper describes an interactive graphical tool called VMV (Virtual Memory Visualization) that is intended to improve student understanding of the underlying concepts and operation of virtual memory. VMV uses animated diagrams to illustrate the organization and operation of a virtual memory system on a step-by-step basis. It supports multiple configurations that can be used emphasize the different roles of hardware and software during different operations. The source code for VMV is available at (https://github.com/jnestor/CADApps).

VMV is being used this semester in a sophomore-level computer organization and architecture course. We developed a set of case studies using lectures and student exercises that focus on basic page translation, page faults, handling memory writes, and using a translation lookaside buffer (TLB). The effectiveness of these case studies will be assessed using a combination of pre/post quizzes, exam problems, and a student survey.

1. Introduction

In its idealized form (Figure 1(a)) [1], a computer system consists of a processor that is connected to a memory containing *instructions* and *data* organized as binary words. The processor operates by fetching instructions from memory and executing the instructions specified by the architecture of the processor, including instructions that read and write data in memory.

It is difficult to build a single memory that is both large enough to support modern applications and fast enough to operate at the speed of modern processors. Instead, memory systems are generally implemented using a *memory hierarchy* [2] that replaces a single memory with multiple levels of memory, as shown in Figure 1(b). In this hierarchy a small and fast *cache memory* is used to store recently used data and instructions, a *main* or *physical memory* is used to store the bulk of program instructions and data, and a larger *backing store* (usually a hard disk or solid-state drive) is used to support programs that require more storage than is available in the main memory.

The process of managing the interaction between the main memory and the backing store is known as *virtual memory*. In a virtual memory system, a processor executing a program accesses memory via idealized *virtual addresses*. In *paged virtual memory*, the virtual address space is divided into fixed-size *pages.* Each page can reside either in main memory or in the backing store. A virtual address is partitioned into a *virtual page number (VPN)* and an *offset*.

During a memory access, processor hardware attempts to map the virtual address into a physical memory address consisting of a *physical page number (PPN)* combined with the offset from the virtual address. During this *address translation* process, the processor reads a data structure in memory called a *page table* that records the location of each virtual page (either in physical memory or in the backing store). When the page is resident in physical memory, the processor can complete the memory access immediately. When the page is not resident in physical memory, a condition called a *page fault* occurs. When this happens, the processor suspends the program requesting the memory access and invokes the operating system. The operating system copies the nonresident page from the backing store into main memory – a process known as *swapping*. If there is no available space in physical memory to store the new page, another resident page must be *evicted* from the main memory using a *page replacement* algorithm. Once swapping has completed, the operating system restarts the suspended program, which completes its memory access to the now-resident page.

A key advantage of virtual memory is that it uses main memory as a cache that stores recently-used pages, ensuring fast execution of the most commonly used instructions and data while at the same time allowing programs that require more storage than is available in the main memory. A second key advantage of virtual memory is the *protection* provided when a computer runs multiple programs concurrently. Each program has its own virtual address space which is distinct from the virtual address spaces of other programs. This enhances the reliability and security of the overall system by preventing programs from interfering with each other.

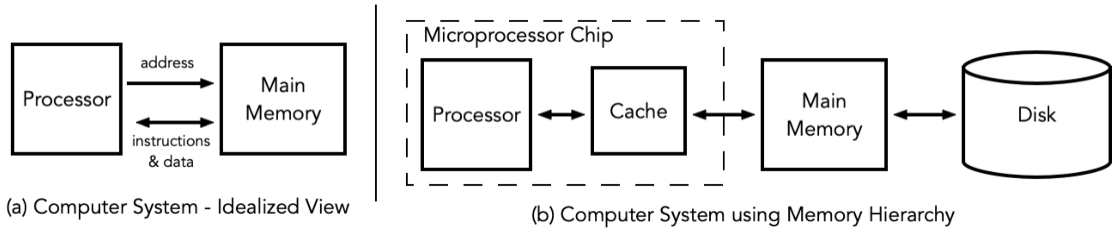


Figure 1 - Memory Systems - Idealized View vs. Memory Hierarchy

Managing virtual memory is a complex process that involves a collaboration between processor hardware and operating system software. The concepts behind virtual memory are explored in textbooks from the perspective of either processor hardware (e.g., [1], [2]) or operating system software (e.g. [3], [4]). From either perspective, students often find these concepts difficult to understand using the static diagrams and written explanations provided in these textbooks. This motivates the development of VMV, a simulation-based *Virtual Memory Visualization* tool that illustrates these concepts graphically using step-by-step animation.

The goal of VMV is to help students build a solid understanding of the following concepts:

- How virtual memory provides an abstraction to a running program in the form of a virtual address space that is broken up into fixed-size pages, while physical memory provides access to a (typically smaller) number of physical pages of the same size.
- How virtual memory pages reside on disk while a subset of these pages are cached in physical memory.
- How memory accesses are handled by processor hardware using page translation.
- What happens when a page fault occurs because a virtual page is not present in physical memory.
- How operating system software handles a page fault by selecting a location in physical memory for the page, evicting the virtual page currently residing there if necessary, and copying the new page from disk to memory.
- How memory writes differ from memory reads and how they are handled during page replacement.

- How a Translation Lookaside Buffer (TLB) is used as a cache of address translations to reduce memory accesses to the page table.
- How a page is chosen for replacement when swapping occurs.

VMV provides a visual simulation of how processor hardware and operating system software collaborate when processing a sequence of memory accesses. It features a graphical representation of the initial virtual address, the page table and translation lookaside buffers used in address translation, the resulting physical address, the physical memory and disk, and the data structure used by the operating system to perform page replacement.

There have been a number of simulation/visualization tools developed for virtual memory in the past ([5]–[13]). These visualizations tend to be written from an operating systems perspective, with a focus on the contents of virtual and physical memory using a fixed configuration and limited features. VMV improves on these visualizations in a number of ways. First, it supports viewing virtual memory from the perspective of both the processor hardware and the operating system software and highlights the role of each. From the hardware perspective, it provides a visualization of the flow of information during address translation via either the page table or translation lookaside buffer (TLB). From the software perspective, it provides a visualization of the flow of data between disk and physical memory during swapping using a page replacement algorithm. VMV can be customized using a configuration file that specifies the size of the virtual address space, the page size, the number of physical pages available, whether to include a TLB, and which page replacement algorithm to use. The configuration file also includes a list of memory references that can be used to initialize the configuration followed by additional memory references that the user can simulate in either single-step or free-running mode. Finally, it allows the user to enter and simulate memory references directly.

VMV complements a similar tool for visualizing the operation of cache memories [14]. Both tools are being used during the Spring 2022 semester in a second-year ECE course covering computer organization and architecture. To support its use, a set of lecture demonstrations and homework assignments have been developed. These tools are also being considered for use in a third-year Computer Science course in operating systems. Both tools are available for download on GitHub at (https://github.com/jnestor/CADApps).

The remainder of this paper is organized as follows: Section 2 describes the goals and overall design of VMV and its implementation. Section 3 describes several case studies that will be used in a sophomore-level computer engineering course to illustrate different concepts of virtual memory. Section 4 concludes the paper and discusses plans for future work.

2. Design and Implementation

2.1 Goals of the Visualization

VMV was created with the following objectives in mind:

- Show the flow of address and data information between the processor, memory, disk, and operating system during the various processes of Virtual Memory including:
    - Memory accesses by the processor starting with a virtual address that is translated to the physical address of a page that is resident in physical memory.
    - Handling page faults, including swapping of pages between disk and physical memory by the operating system.

    - Handling of memory writes, both when the virtual address is resident in physical memory and when a page fault occurs.
    - The operation of different page replacement algorithms.
    - The use of a translation lookaside buffer (TLB) to reduce memory accesses to the page table.
- Clarify the roles of processor hardware and operating system software in handling memory accesses.
- Allow more advanced features (e.g. TLB) to be removed from the display to simplify the visualization when teaching initial concepts.
- Be configurable in terms of page size, memory size, and TLB size.
- Provide both single-step and free-running operation.
- Allow a memory access to be restarted to review its effects.

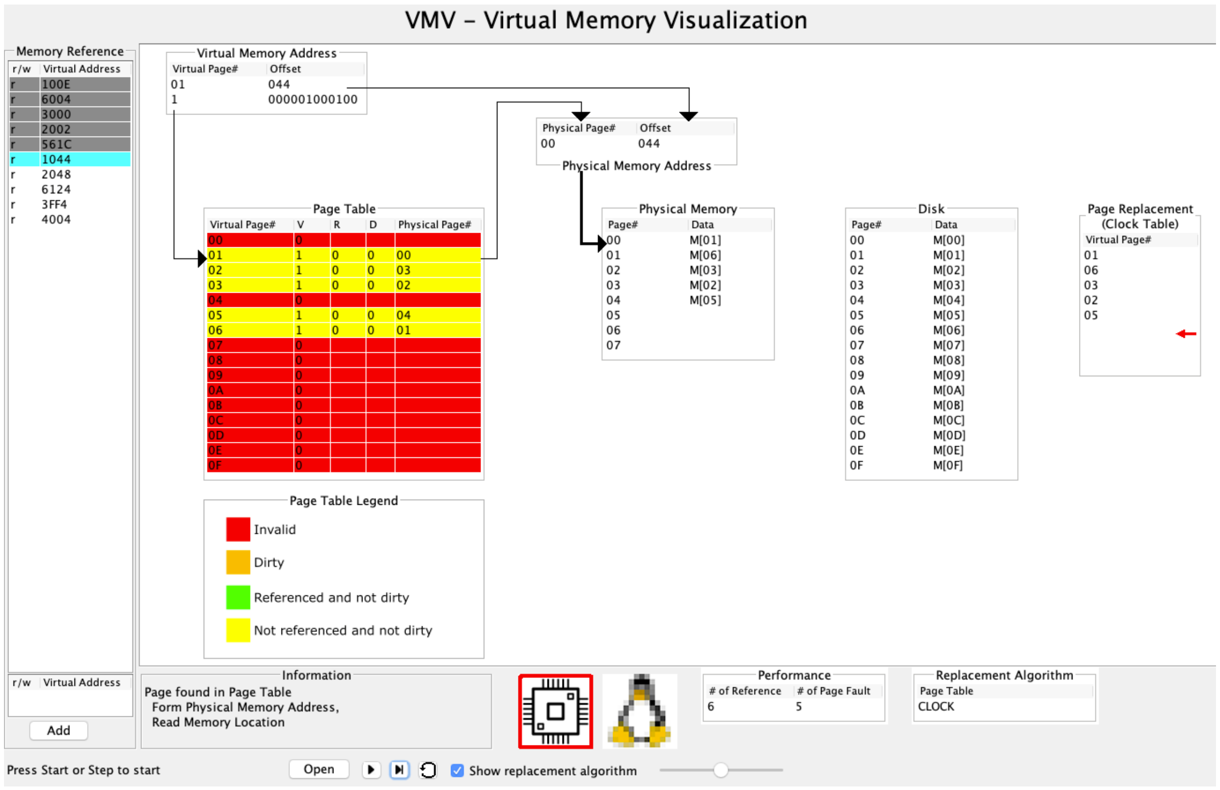
2.2 Visualization Layout

VMV is implemented as a Java application using the Swing GUI Toolkit . Figure 2 shows the layout of the graphical components of the user interface, which is organized as a window divided into several different panes. This window is optimized for viewing on a 1080p monitor, but can also be used with scrolling on smaller monitors. The leftmost pane in the window provides a list of memory references that will be applied during simulation. This sequence can be initialized by the configuration file, but a user can add additional references using the control panel at the bottom of the pane. As simulation proceeds, the current memory reference is highlighted in blue, while previously processed references are highlighted in gray.

The main pane of the window shows the virtual address of the current reference, the physical address that results after page translation, the page table, the physical memory, disk, and a page replacement display that is used when handling page faults.

The page table is central to the address translation process, since it records whether a virtual page resides in physical memory. Each page has a corresponding page table entry (PTE) that contains three status bits (V, D, and R) and a physical page number. The V (valid) bit indicates that the page is currently resident in physical memory in the location specified by the physical page number. The D (dirty) bit indicates that the page has been altered through a memory write but not updated on disk. The R (reference) bit indicates that the page has been “recently” accessed (information that is used during page replacement to decide if a page should be evicted). To allow the user to read this status at a glance, page table entries are color coded depending on their status, as described in a legend placed underneath the page table.

The bottom of the display includes a control panel for loading configuration files and starting, pausing, and single-stepping simulation. The “Information” pane displays updates about the simulation while one of the two images is highlighted to indicate if current actions are being performed by processor hardware or by operating system software. Finally, a performance pane displays statistics about the simulation at its current point in time.



VMV – Virtual Memory Visualization

Memory Reference

r/w	Virtual Address
r	100E
r	6004
r	3000
r	2002
r	561C
r	1044
r	2048
r	6124
r	3FF4
r	4004

Virtual Memory Address

Virtual Page#	Offset
01	044
1	000001000100

Physical Memory Address

Physical Page#	Offset
00	044

Page Table

Virtual Page#	V	R	D	Physical Page#
00	0			
01	1	0	0	00
02	1	0	0	03
03	1	0	0	02
04	0			
05	1	0	0	04
06	1	0	0	01
07	0			
08	0			
09	0			
0A	0			
0B	0			
0C	0			
0D	0			
0E	0			
0F	0			

Physical Memory

Page#	Data
00	M[01]
01	M[06]
02	M[03]
03	M[02]
04	M[05]
05	
06	
07	

Disk

Page#	Data
00	M[00]
01	M[01]
02	M[02]
03	M[03]
04	M[04]
05	M[05]
06	M[06]
07	M[07]
08	M[08]
09	M[09]
0A	M[0A]
0B	M[0B]
0C	M[0C]
0D	M[0D]
0E	M[0E]
0F	M[0F]

Page Replacement (Clock Table)

Virtual Page#
01
06
03
02
05

Page Table Legend

- 🟥 Invalid
- 🟧 Dirty
- 🟩 Referenced and not dirty
- 🟨 Not referenced and not dirty

Information

Page found in Page Table
Form Physical Memory Address,
Read Memory Location

Performance

# of Reference	# of Page Fault
6	5

Replacement Algorithm

Page Table
CLOCK

Press Start or Step to start

✓ Show replacement algorithm

Figure 2 – Basic Layout Showing Address Translation

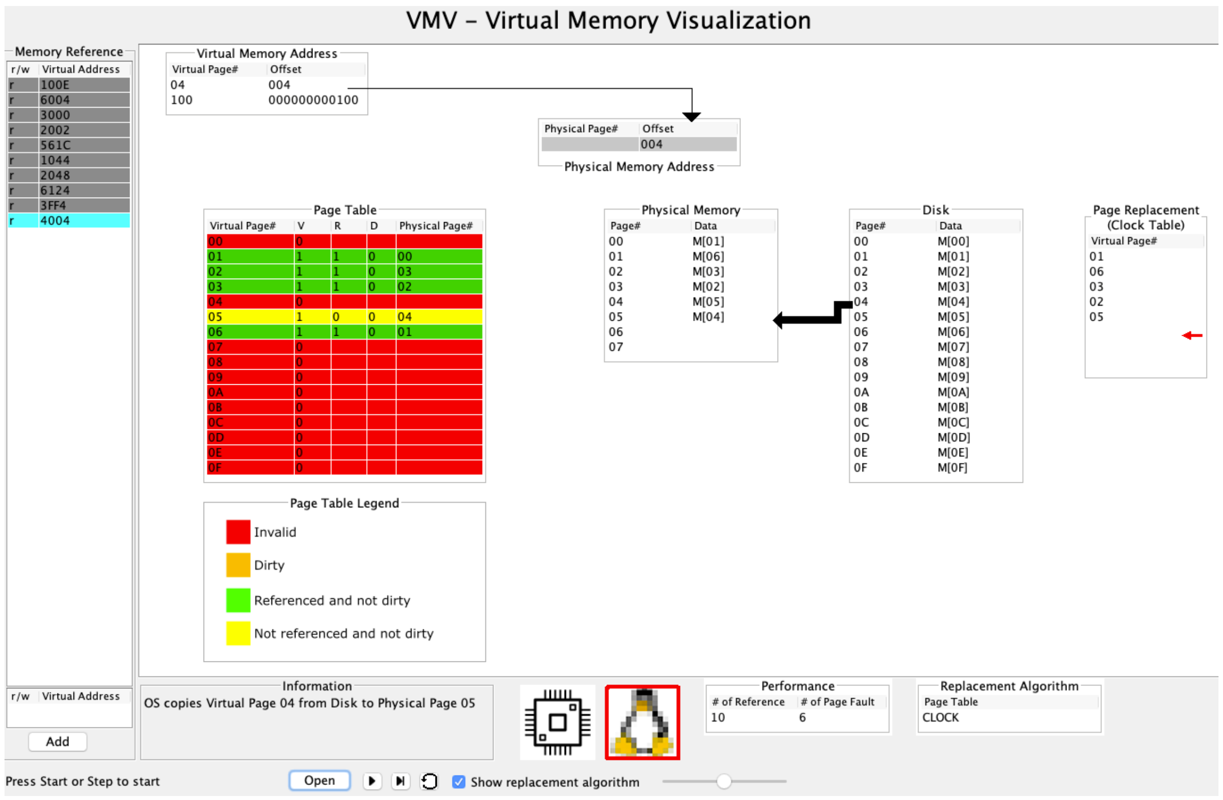
During simulation, the display is updated in steps as a memory reference is processed. For example, Figure 2 shows the display at the moment that the virtual page number is used to index the page table and successfully generate a physical address. As the user steps through the simulation, other arrows and graphical cues indicate what is taking place while the contents of the page table, memory, and disk are altered.

While VMV is configurable, it does make several simplifications over virtual memory in real computer systems. First of all, it uses a single linear page table, while production computer systems use hierarchical page tables. Second, to be of practical use during simulation, the number of pages in the virtual address space and the physical memory must be small enough to conveniently view in the space provided. Third, the system does not currently support accesses from multiple processes. Finally, VMV does not support finer-grained protection of individual page accesses that are common in real computer systems.

2.3 Simulation and Animation

VMV operates by animating the key steps that take place during each memory access. Each step includes visual cues such as arrows indicating the direction of data flow and highlighting of active components. It also updates the contents of different elements such as the page table, TLB, and physical memory. A successful memory access to a resident page is completed in four steps. Accesses to nonresident pages require several additional steps that illustrate the process of handling a page fault and swapping pages in and out of physical memory.

The simulation is implemented using two software finite state machines - one for simulating without a TLB, and one for simulating with a TLB present. Each state represents one of the steps described above; sequencing between states depends on the state of the TLB and/or page table and the actions that must be performed. Both state machines operate by reading memory references from the list on the left side of the display and cycling through the steps required to complete the reference.



VMV – Virtual Memory Visualization

Memory Reference

r/w	Virtual Address
r	100E
r	6004
r	3000
r	2002
r	561C
r	1044
r	2048
r	6124
r	3FF4
r	4004

Virtual Memory Address

Virtual Page#	Offset
04	004
100	000000000100

Physical Memory Address

Physical Page#	Offset
	004

Page Table

Virtual Page#	V	R	D	Physical Page#
00	0			
01	1	1	0	00
02	1	1	0	03
03	1	1	0	02
04	0			
05	1	0	0	04
06	1	1	0	01
07	0			
08	0			
09	0			
0A	0			
0B	0			
0C	0			
0D	0			
0E	0			
0F	0			

Physical Memory

Page#	Data
00	M[01]
01	M[06]
02	M[03]
03	M[02]
04	M[05]
05	M[04]
06	
07	

Disk

Page#	Data
00	M[00]
01	M[01]
02	M[02]
03	M[03]
04	M[04]
05	M[05]
06	M[06]
07	M[07]
08	M[08]
09	M[09]
0A	M[0A]
0B	M[0B]
0C	M[0C]
0D	M[0D]
0E	M[0E]
0F	M[0F]

Page Replacement (Clock Table)

Virtual Page#
01
06
03
02
05

Page Table Legend

🟥	Invalid
🟧	Dirty
🟩	Referenced and not dirty
🟨	Not referenced and not dirty

Information

OS copies Virtual Page 04 from Disk to Physical Page 05

Performance

# of Reference	# of Page Fault
10	6

Replacement Algorithm

Page Table
CLOCK

Press Start or Step to start

Figure 3 – Basic Layout Showing Swap

For simulation without the TLB, the first step is to read the page table entry (PTE) for the virtual page number specified by the current address. If the valid bit of the PTE is set (PTE shaded yellow, green, or orange), then this indicates that the desired page currently resides in physical memory. The physical address is then formed by concatenating the physical page number in the PTE with the page offset from the original address and the memory reference is completed.

If the valid bit is not set (PTE shaded red), a page fault occurs and the next steps show the operating system arranging a swap of pages by selecting a physical page for replacement, copying that page back to disk if the page is dirty (PTE shaded orange), and copying the new page from disk to physical memory while updating the PTEs of the two pages. The next steps show the hardware restarting the memory access and completing it successfully.

When the simulation includes the TLB, the first step is to determine if the TLB contains an entry for the virtual page specified by the current address. If it does, the corresponding TLB entry is used to compute the physical address and complete the memory reference. If not, the next step is to read the page table entry for the virtual page. If the PTE is valid, the TLB is updated using the PTE and the memory reference is restarted. If not, a page fault occurs; the TLB is flushed and the page fault is handled by the operating system using the same process as the non-TLB simulation.

3. Case Studies for Lectures and Student Exercises

VMV is currently being used in a sophomore-level course in computer organization and architecture. To support this activity, we are developing a set of case studies for use in lectures and student exercises. Each case study uses a separate configuration file that sets the size of the virtual and physical memory, selects the page replacement algorithm to be used, and indicates whether a TLB is included. The configuration file contains two sequences of memory references. VMV uses the first sequence to initialize the contents of the virtual memory, physical memory, and page table at startup. The user can then simulate the remaining references on a step-by-step basis.

To assess the effectiveness of the case studies, a brief quiz will be administered to students before and after each case study is used to evaluate its effect on student understanding of basic concepts. Traditional exam problems will be used to evaluate more in-depth understanding. Finally, students will be administered a survey at the end of the semester to gather feedback and suggestions for improvements.

3.1 Virtual Memory Organization and Page Translation

This case study is intended to introduce students to the basic concepts of virtual memory. The learning objectives of this case study are to strengthen student understanding of:

- the organization of a virtual memory system consisting of physical memory, backing store (disk), and page table.
- how a virtual memory address is translated into a physical memory address when a virtual page resides in physical memory.

Because it focuses on basic concepts, the configuration file for this case study excludes the TLB. It specifies an initial sequence of memory read operations that load the physical memory with a number of virtual pages at startup. After startup, a user can sequence through an additional five memory read operations, following how the page table is used to look up the physical location of a virtual page and the physical address is formed by combining the physical page number with an offset. For example, Figure 2 shows the simulation of a page translation from virtual page 01 to physical page 00, which is concatenated with offset 044 to form the physical address 00044.

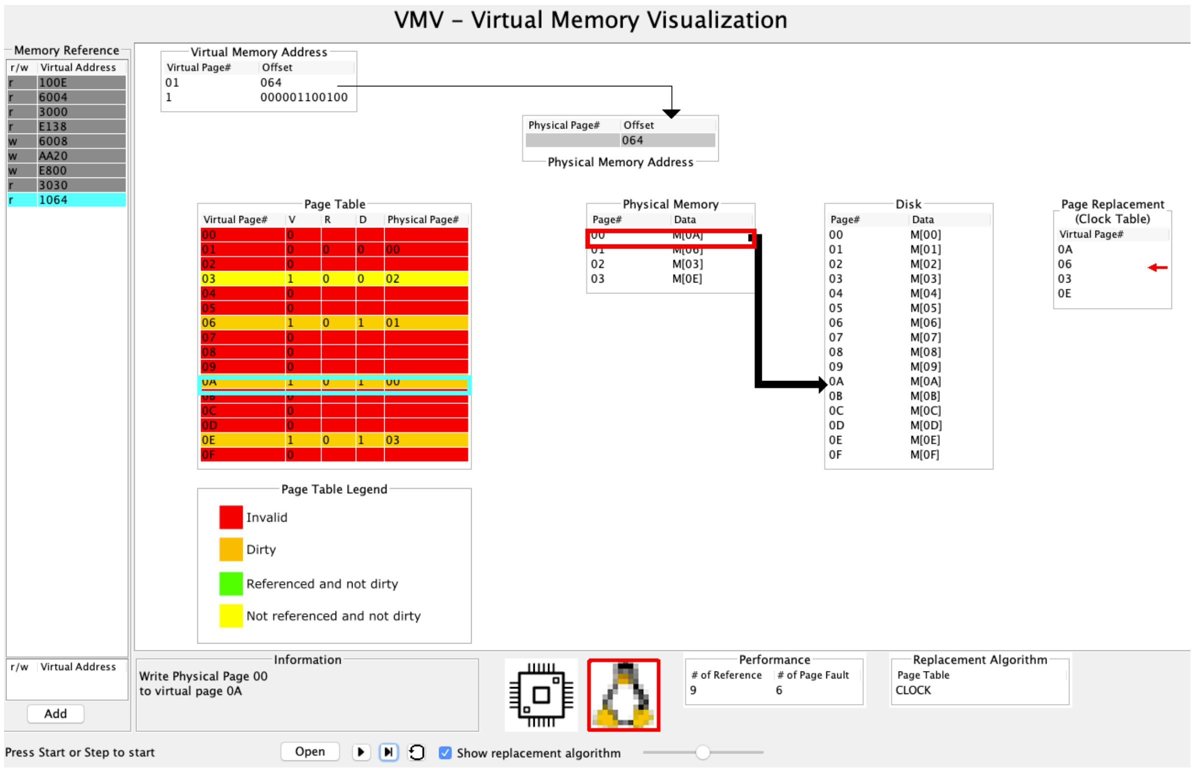
3.2 Page Faults and Page Replacement

This case study focuses on how a virtual memory system handles page faults. The learning objectives of the case study are to reinforce student understanding of:

- how an access to a virtual page that is not resident in physical memory results in a page fault, requiring the suspension of the program requesting the memory access.
- how the operating system handles a page fault by swapping, i.e., selecting a physical memory page in which to place the requested virtual page, evicting the virtual page currently residing in the physical page if necessary, and then copying the requested virtual page into the selected physical page, as shown in Figure 3.
- how the memory access is restarted once swapping has been completed.

This case study uses the same configuration as the previous case study but begins with additional memory references that fill all 8 pages of physical memory before pausing. The user can then observe two additional references to nonresident pages, each of which triggers a page fault and requires a sequence of steps where the page replacement takes place. Figure

3 shows one step in this case study where virtual page 04 is being copied into physical page 05 by the operating system. Operating systems use a variety of algorithms to select a page for replacement when swapping. By default, VMV uses the well-known “clock” algorithm, which uses the “R” (reference) bit in each PTE to determine if a page has been used recently. It also supports the first-in-first-out (FIFO) and least-recently-used (LRU) algorithms.


VMV – Virtual Memory Visualization
Open
Show replacement algorithm
Add

Memory Reference

r/w	Virtual Address
r	100E
r	6004
r	3000
r	E138
w	6008
w	AA20
w	E800
r	3030
r	1064

Virtual Memory Address

Virtual Page#	Offset
01	064
1	000001100100

Physical Memory Address

Physical Page#	Offset
	064

Page Table

Virtual Page#	V	R	D	Physical Page#
00	0			
01	0	0	0	00
02	0			
03	1	0	0	02
04	0			
05	0			
06	1	0	1	01
07	0			
08	0			
09	0			
0A	1	0	1	00
0B	0			
0C	0			
0D	0			
0E	1	0	1	03
0F	0			

Page Table Legend

🟥 Invalid

🟧 Dirty

🟩 Referenced and not dirty

🟨 Not referenced and not dirty

Physical Memory

Page#	Data
00	M[0A]
01	M[06]
02	M[03]
03	M[0E]

Disk

Page#	Data
00	M[00]
01	M[01]
02	M[02]
03	M[03]
04	M[04]
05	M[05]
06	M[06]
07	M[07]
08	M[08]
09	M[09]
0A	M[0A]
0B	M[0B]
0C	M[0C]
0D	M[0D]
0E	M[0E]
0F	M[0F]

Page Replacement (Clock Table)

Virtual Page#
0A
06
03
0E

r/w	Virtual Address

Information

Write Physical Page 00
to virtual page 0A

Performance

# of Reference	# of Page Fault
9	6

Replacement Algorithm

Page Table
CLOCK

Press Start or Step to start

Figure 4 – Saving a modified page during page replacement

3.3 Memory Writes

This case study illustrates how memory write operations are handled in virtual memory. The learning objectives of this case study are to reinforce student understanding of:

- how memory writes to virtual pages are deferred using a “copy-back” strategy when the virtual page resides in physical memory.
- how a page modified by a write operation is marked as modified using the D (Dirty) bit in its page table entry.
- how a modified “dirty” page must be written back to disk when it is selected for eviction.

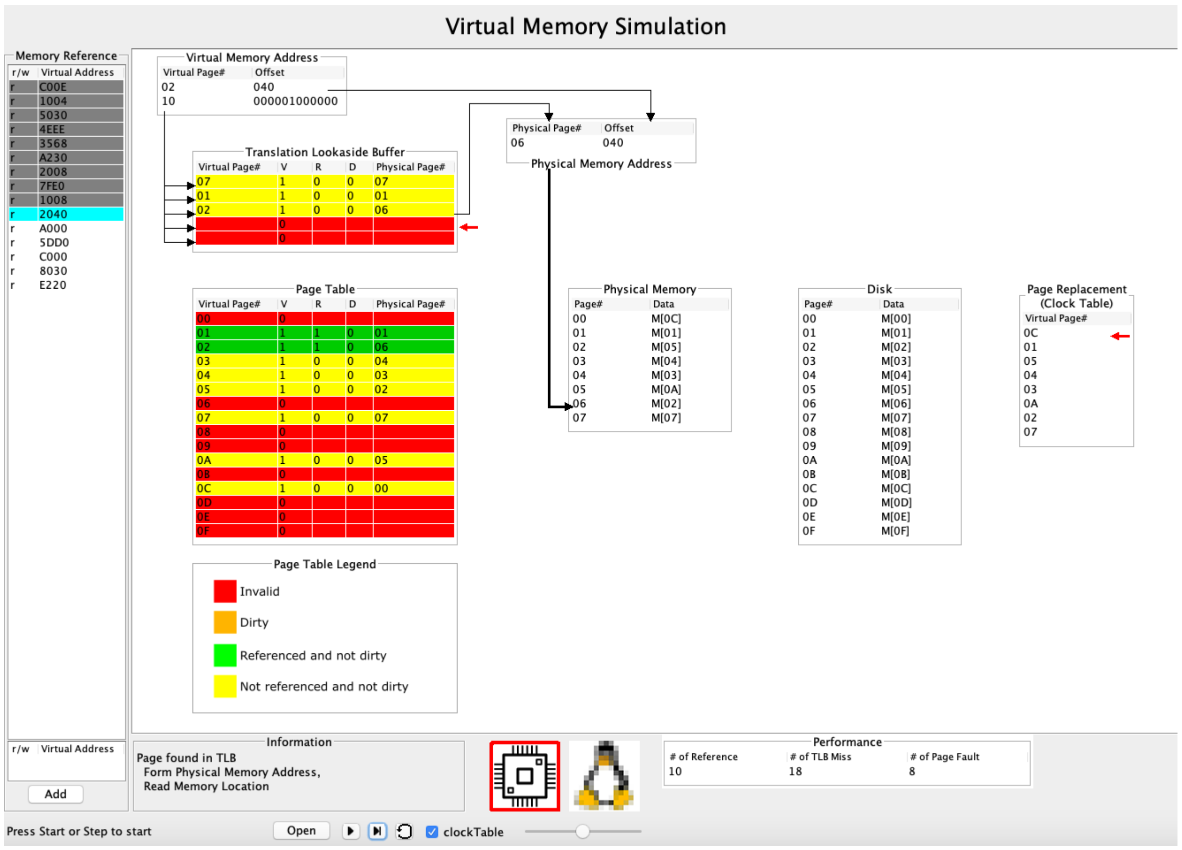
This case study uses a configuration file similar to previous case studies but with a physical memory containing only 4 pages. A sequence of startup read and write operations load all four pages of physical memory. After startup, user can step through a sequence of memory references which illustrate show the effect of memory writes to resident pages and operation of the page replacement algorithm when a page fault occurs, including writing a modified page to memory. Figure 4 shows one step in this case study where modified virtual page 0A is copied back to disk from physical page 00, after which virtual page 01 will be loaded in its place.

3.4 Translation Lookaside Buffer (TLB) Operation

This case study illustrates how a TLB can be added to a virtual memory system to reduce memory access to the page table. The learning objectives of this case study are to build student understanding of:

- how a memory access using virtual address translation actually requires a second memory access to read the page table.
- how a Translation Lookaside Buffer (TLB) is used as a cache for recent address translations to avoid of memory accesses to the page table.
- how TLB misses are handled.
- how page faults affect the TLB.

The configuration file for this case study specifies that the TLB is present in this simulation, resulting in the display shown in Figure 5. It also specifies an initial sequence of memory references that loads five virtual pages into physical memory while the TLB contains three valid address translations. The user can then step through memory reads that include TLB hits, TLB misses that are still resident in physical memory, and page faults. This is followed by illustrating memory write operations that set the dirty bit in the corresponding TLB entry but only update the page table when a TLB entry is evicted due to a TLB miss.



Virtual Memory Simulation

Memory Reference

r/w	Virtual Address
r	C00E
r	1004
r	5030
r	4EEE
r	3568
r	A230
r	2008
r	7FE0
r	1008
r	2040
r	A000
r	5DD0
r	C000
r	8030
r	E220

Virtual Memory Address

Virtual Page#	Offset
02	040
10	000001000000

Physical Memory Address

Physical Page#	Offset
06	040

Translation Lookaside Buffer

Virtual Page#	V	R	D	Physical Page#
07	1	0	0	07
01	1	0	0	01
02	1	0	0	06
	0			
	0			

Page Table

Virtual Page#	V	R	D	Physical Page#
00	0			
01	1	1	0	01
02	1	1	0	06
03	1	0	0	04
04	1	0	0	03
05	1	0	0	02
06	0			
07	1	0	0	07
08	0			
09	0			
0A	1	0	0	05
0B	0			
0C	1	0	0	00
0D	0			
0E	0			
0F	0			

Page Table Legend

🟥	Invalid
🟧	Dirty
🟩	Referenced and not dirty
🟨	Not referenced and not dirty

Physical Memory

Page#	Data
00	M[0C]
01	M[01]
02	M[05]
03	M[04]
04	M[03]
05	M[0A]
06	M[02]
07	M[07]

Disk

Page#	Data
00	M[00]
01	M[01]
02	M[02]
03	M[03]
04	M[04]
05	M[05]
06	M[06]
07	M[07]
08	M[08]
09	M[09]
0A	M[0A]
0B	M[0B]
0C	M[0C]
0D	M[0D]
0E	M[0E]
0F	M[0F]

Page Replacement (Clock Table)

Virtual Page#
0C
01
05
04
03
0A
02
07

r/w	Virtual Address

Information

Page found in TLB
 Form Physical Memory Address,
 Read Memory Location

Performance

# of Reference	# of TLB Miss	# of Page Fault
10	18	8

Press Start or Step to start

Figure 5 – Visualization with Translation Lookaside Buffer (TLB)

4. Conclusion

This paper described the ongoing development of VMV, a visualization tool to help students understand virtual memory. VMV illustrates the underlying concepts of virtual memory including address translation, handling of page faults, and page replacement algorithms. The tool is currently being used in a class on computer architecture and organization with a collection of case studies that illustrate different concepts of virtual memory. VMV's effectiveness will be assessed using pre/post quizzes of concepts, exam problems, and a survey of student response. Several improvements are planned in the future, including support for multiple processes and multiple processor cores, more sophisticated memory protection, and multi-level page tables.

References

[1] D. M. Harris and Harris, Sarah, *Digital Design and Computer Architecture - 2nd Edition*. Elsevier, 2012.

[2] D. Patterson and J. Hennessey, *Computer Organization and Design: The Hardware Software Interface: RISC-V Edition*, 5th ed. Morgan Kaufmann, 2014.

[3] R. H. Arapaci-Dusseau and A. C. Arapaci-Dusseau, "Operating Systems: Three Easy Pieces." https://pages.cs.wisc.edu/~remzi/OSTEP/#book-chapters (accessed Jan. 03, 2022).

[4] A. S. Tanenbaum, *Modern operating systems*, Fourth edition. Upper Saddle River, N.J: Pearson, 2015.

[5] "Virtual Memory Workbench," *Archive of the Hyperlearning Center for the New Engineer*. https://denninginstitute.com/workbenches/vmsim/vmsim.html (accessed Jan. 17, 2022).

[6] S. Khuri and H.-C. Hsu, "Visualizing the CPU scheduler and page replacement algorithms," in *The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, New York, NY, USA, Mar. 1999, pp. 227–231.

[7] L. Null and K. Rao, "CAMERA: introducing memory concepts via visualization," *ACM SIGCSE Bull.*, vol. 37, no. 1, pp. 96–100, Feb. 2005, doi: 10.1145/1047124.1047389.

[8] R. Ontko, "MOSS | Memory Management Simulator | User Guide." http://www.ontko.com/moss/memory/user_guide.html (accessed Jan. 13, 2022).

[9] F. N. Sibai, M. Ma, and D. A. Lill, "Development of a Virtual Memory Simulator to Analyze the Goodness of Page Replacement Algorithms," in *2007 Innovations in Information Technologies (IIT)*, Nov. 2007, pp. 536–540. doi: 10.1109/IIT.2007.4430437.

[10] E. Gopak, "Memory paging visualization." http://ericgopak.github.io/operating-system-concepts/ (accessed Jan. 03, 2022).

[11] A. Paramita and K. G. Smitha, "PARACACHE: Educational Simulator for Cache and Virtual Memory," in *2017 International Symposium on Educational Technology (ISET)*, Jun. 2017, pp. 234–238. doi: 10.1109/ISET.2017.60.

[12] V. K. K. Musunuru, "Virtuo-ITS: An Interactive Tutoring System to Teach Virtual Memory Concepts of an Operating System," MS Thesis, Wright State University, Dayton, OH, 2017.

[13] W. A. Bhat, A. Rashid, F. F. Wani, and F. Altaf, "Virtualization and visualization of virtual memory system for effective teaching–learning," *Comput. Appl. Eng. Educ.*, vol. 27, no. 5, pp. 1286–1294, 2019, doi: 10.1002/cae.22152.

[14] B. Ilbeyi and J. A. Nestor, "VCache: Visualization Applet for Processor Caches," *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education*, June 2010.

[15] "Trail: Creating a GUI With Swing (The Java^{TM} Tutorials)," *The Java^{TM} Tutorials*. https://docs.oracle.com/javase/tutorial/uiswing/ (accessed Jan. 17, 2022).