A User Space-based Project for Practicing Core Memory Management Concepts

Sam Silvestro
Department of Computer Science
University of Texas at San Antonio
San Antonio, Texas
sam.silvestro@utsa.edu

Timothy T. Yuen
Department of Interdisciplinary Learning and Teaching
University of Texas at San Antonio
San Antonio, Texas
timothy.yuen@utsa.edu

Corey Crosser
Department of Electrical Engineering & Computer Science
United States Military Academy
West Point, New York
corey.crosser@usma.edu

Dakai Zhu
Department of Computer Science
University of Texas at San Antonio
San Antonio, Texas
dakai.zhu@utsa.edu

Turgay Korkmaz
Department of Computer Science
University of Texas at San Antonio
San Antonio, Texas
turgay.korkmaz@utsa.edu

Tongping Liu
Department of Computer Science
University of Texas at San Antonio
San Antonio, Texas
tongping.liu@utsa.edu

ABSTRACT

This paper presents the design and evaluation of a novel project designed to facilitate the learning of memory management concepts and interactions between different components. This project removes the complexity of a full or specific operating system by implementing memory management inside the user space. Evaluation results show that the mean exam scores improved by about 29% to 34%. On average, the total code size is less than 300 lines and time spent working on this project is under 17 hours. Therefore, this project is beneficial in helping students learn memory management while maintaining a reasonable project workload.

ACM Reference Format:
Sam Silvestro, Timothy T. Yuen, Corey Crosser, Dakai Zhu, Turgay Korkmaz, and Tongping Liu. 2018. A User Space-based Project for Practicing Core Memory Management Concepts. In *Proceedings of The 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3159450.3159581

1 INTRODUCTION

"Operating Systems" is a core course of undergraduate computer science curriculum [1], and memory management is a key component of the OS which determines the efficiency of the entire system, to some extent. Modern operating systems utilize the virtual memory mechanism. Therefore, multiple related concepts (i.e., virtual memory, address space, page tables, address translation, page replacement, page faults, and physical memory management) are required to be covered during lectures as it is important to fully understand the relationship between these concepts. This relationship can be described as follows. In order to support virtual memory, the paging mechanism is typically introduced to overcome the coarse granularity of segments. Physical memory is divided into multiple frames, and the process address space is then divided into virtual pages. The OS is responsible for physical memory management, as well as virtual memory management. Basically, it utilizes per-process page tables to track the relationship between physical frames and virtual pages. An executing program generates a virtual address prior to every memory access, where the page number and page offset can be extracted from this address. The page number of a virtual address is utilized for translation into a physical frame number using the page table, if the page actually exists in main memory. Otherwise, a page fault occurs, and a possible page replacement may be required to find a physical frame. Afterward, a new mapping is created so that the program can access the corresponding page. Each of these concepts themselves are difficult to follow and understand. Further, these concepts should be connected together in order to have a holistic understanding of memory management. So, course projects should be helpful for students to strengthen their understanding of the different concepts and the interactions among them.

SIGCSE '18, Feb. 21–24, 2018, Baltimore, MD, USA

ACM ISBN 978-1-4503-5103-4/18/02.
https://doi.org/10.1145/3159450.3159581

2 MEMORY MANAGEMENT PROJECTS

Although there are many existing projects targeting memory management inside the OS [6–10], none can fully serve this purpose. Existing work typically shares two common shortcomings.

First, projects tend to focus on only one aspect of memory management, such as the SLOB memory allocator [7], buddy manager [8], page replacement [10], and page thrashing [6]. Laadan et al. designed several projects to understand internal OS memory usage, such as the working sets and modified sets of a process, and the page uses of each memory region [9], which is still not sufficient to cover all of the concepts described above. Although these projects are helpful for students to understand one particular concept, they cannot present a holistic view of memory management. Many concepts cannot be practiced through these projects, not to mention the interaction between them.

Second, most of these existing projects are too difficult to implement and debug, since students are typically asked to alter an existing OS [2, 7, 9, 10, 13]. For the implementation, students have to understand data structures and the basic architecture of an existing OS, then determine where to insert their implementation, if

not given hints. Also, changing an existing OS also increases the complexity of compiling and debugging. Students must recompile the entire OS, then verify whether their specific code has been executed. They also must learn how to debug their code inside the OS itself, which is another well-known challenge, even for experts [5]. Therefore, existing projects significantly increase the complexity of implementation and debugging. Rather than practicing these key memory management concepts, students may end up spending a large amount of time and effort just understanding a particular OS, and learning how to perform debugging within it.

While offering an improvement over a real OS, the issue of complexity holds true for an OS framework such as Pintos [12]. The overhead associated with familiarizing oneself with the overall structure and organization could still impose an unnecessary burden for students focused on learning virtual memory concepts. PennOS helps avoids this issue, as it is written from scratch in the user space [3]. However, it lacks a memory management component, instead focusing on process management and scheduling, file system and I/O, and shell integration.

3 DESIGN OF A USER SPACE-BASED MEMORY MANAGEMENT PROJECT

This paper presents the novel design of a user space-based project that exercises memory management fully within the user space. This project's design addresses three critical issues for effectively learning memory management: (1) the project should cover most core concepts, as described above, rather than one particular topic; (2) the project's workload should be reasonably low; (3) students should be able to gain a comprehensive understanding of the interaction between the different concepts. A design built in user space provides several benefits: first, it can significantly reduce the complexity of the project. Since it does not rely on an existing OS, students save significant effort spent understanding its implementation. Second, students do not need to recompile the OS upon making changes. Third, the project is significantly easier to debug when problems arise. Students can instead focus on the project itself, rather than interacting with a complicated OS.

Based on these requirements, we adapted **the idea of simulation** [11]. Null et al. designed an interactive tool based on a randomly generated series of memory references. However, their tool is designed to show graphical representations of the memory system and provide step-by-step descriptions of memory management activities inside the OS based on the given memory references. Although this project is also based on a simulated sequence of memory references, students are required to implement the project by themselves starting from scratch. The similarity is limited only to the utilization of a simulated series of memory references.

The overall objective of the project is for students to practice (1) virtual memory management, (2) address translation, (3) page table design, and (4) page replacement algorithms. This project further designs three sub-projects based on a simulated series of memory references.

3.1 Part I: Translation via Page Table

Students implement address translation based on a given one-level page table. They are further provided with basic machine details, such as the page/frame size, the address space of virtual memory, and the size of physical memory. Students write a program that accepts a filename as its only argument. This input file consists of a sequence of 8-byte `unsigned long` values that simulates a sequence of memory accesses, where each value represents a virtual address. For each virtual address, students must refer to a given page table to translate it into a corresponding physical address. The mapping (or page table) between virtual page numbers and physical page numbers is provided by a directed graph, which students are required to translate into a simple array prior to address translation. For simplicity, the given sequence of memory accesses does not include any page faults, where all corresponding addresses are in the range of the given page table. To verify the correctness of their implementation, students should write the sequence of translated physical addresses into an output file in a format similar to the input file. Students can obtain the `md5sum` of this output file [4] and submit the checksum. For a given sequence of memory accesses and a given page table, correct implementations of address translation should always produce the correct and unique checksum.

3.2 Part II: Page Table and Physical Memory Management

Students implement the design of the page table and will manage physical memory. The details of the hardware are provided to the students, such as the page/frame size, the address space of virtual memory and the size of physical memory. From the size of physical memory and the size of a frame, students may compute the total number of physical pages. The difference for the second part is that the page table is not provided. Students must design the page table and manage the physical memory by themselves. To better mimic a real system, it is assumed that the first few pages are reserved for use by the OS, but other pages are initially freed. For simplicity, we assume there is only one application inside the system.

Students are provided with an input file to simulate the memory accesses of the given application. In this part, these memory accesses can cause page faults. For each logical address, students are required to check whether the corresponding physical frame is marked as valid in the page table. If yes, then the student is required to perform the address translation, as in Part I. Otherwise, they must choose a physical frame and update the mapping in the existing page table. Physical frames are assumed to be allocated in sequential order. When there are no available physical frames, students implement the Least-Recently Used (LRU) page replacement policy to evict the LRU frame from the page table, and then reset the page table afterwards. There is no need to swap the page to disk, which is assumed to have been taken care of automatically. Once a frame is selected for replacement, the student must: (1) invalidate the page table entry that is mapped to the current physical frame, and (2) remap the page table entry of the new logical address to the current frame number. The reverse mapping mechanism is introduced in order to identify the corresponding virtual page mapped to the selected physical frame. Upon each reference, the order of accesses for the corresponding physical frames must be updated in order to preserve the LRU policy. Lastly, the students write the sequence of translated physical addresses into an output file and print the total number of page faults that occurred.

To successfully complete this assignment, students must fully understand and integrate different concepts of virtual memory management, including the management of physical pages, LRU page replacement, page-to-frame mapping performed by the OS, and page table management, in addition to the address translation that was exercised in Part I. Part II also helps students understand the details of a physical page table entry, such as the valid bit, and the concept of reverse mapping. With this knowledge, they must simulate the entire process beginning with a memory reference.

3.3 Part III: General Page Table Design

Part III extends the page table for arbitrary system parameters. The student's main program should now accept three additional parameters, which specify a hypothetical machine's page/frame size, virtual memory size, and physical memory size; other conditions remain the same. Students are again required to output the translated physical addresses for the same sequence of memory accesses, as in Part II. This component prompts students to rethink page table design in an abstract situation. Accordingly, students must determine what should be changed to accommodate an arbitrary environment. This part helps students revisit their implementation from the second part in a generalized context, which requires a better understanding of page table design.

4 EVALUATION DESIGN

This section describes the evaluation for this project.

4.1 Sample

This project was implemented in the OS course in the Spring 2017 semester at University of Texas at San Antonio, a large research university in South Texas. There were two sections of this course, taught by the 5^{th} and 6^{th} authors of this paper. There were 64 students in Section 1, and 32 students in Section 2. Of these 96 combined students, 5 were deemed ineligible to include due to not completing the midterm exam, which serves as a test instrument, and were thus eliminated from the sample. For comparison, data from a previous offering of this course in Spring 2016 were included. This section was taught by the 4^{th} author of this paper, and did not include the project, but the same course materials were used for teaching. There were 22 students in this section (Section 0), where no project was assigned.

4.2 Data Sources

4.2.1 Exam Questions. For comparison, we used three identical questions across all sections.

- Question 1 asked students to translate virtual addresses into physical addresses, and show how page tables are stored in memory. Students were provided basic machine parameters such as virtual and physical address spaces, as well as the page/frame size. Students must demonstrate the ability to calculate the number of bits used for each component of both virtual and physical addresses. They must also know how to compute the number of bytes needed for a page table entry, calculate the number of pages required by the entire virtual address space, and produce how many bytes would be needed for the page table of a process on this machine.
- Question 2 tests the students' ability to both perform memory address translation, as well as compute effective access times when given TLB, memory, and disk access times. The students are again provided with basic machine information, such as the page/frame size and virtual/physical address space sizes. Additionally, they are provided with an initial page table and TLB entries. Students must translate four logical addresses into physical addresses, and estimate the access time required by the machine to do so.
- Question 3 tests the students' knowledge of different page replacement algorithms. The question is framed by providing a theoretical maximum number of pages (3 pages) per process. They are then given a sequence of page references from which they must tabulate, by hand, the specific pages currently held in memory after each page reference. Afterward, they must answer with the total number of page faults that occurred for each of the page replacement algorithms in question: FIFO, LRU, and Optimal.

Questions were graded consistently across exams as there was only one correct response for sub-questions in each question. For the sections in which the project was used, this evaluation only includes exam question scores from those students who also completed the project.

4.2.2 Project Results. Student outcomes were also measured through the scores that students received on their projects and completion rates on individual parts of the project. Students were also surveyed on how much time they spent working on the project.

4.2.3 Surveys. Surveys asked students to rate the value of the project, using a 5-point Likert scale, as it related to the topics of address translation, finding the page index inside a virtual address, the concept of frame numbers, physical memory management, LRU, and page offset inside a virtual address. Surveys were used to evaluate students' perception of the project's value as it related to these memory management concepts. The survey also contained open-ended questions asking students to report what they found to be the most difficult part of the project, and why.

5 EVALUATION RESULTS

This section presents the results organized by each evaluation question. A student's data were only included for analysis if s/he had both turned in the project and completed the midterm exam.

5.1 Project Results

Recall the project includes three parts where Part I and Part II are compulsory, and Part III was optional. Table 1 presents the statistics of students' scores for these different parts of the project. In total, 81 out of 91 students submitted the project and took the exam.

The outcomes show that 89% of students were able to complete Part I while about 74% finished Part II across both sections. In terms of scores, there was a very high average score in Part I (around 95.8%), but only around 79% in Part II and Part III. A median score of 100% for Part I indicates that most students were successful in address translation with a one-level page table. The median score of 87.5% on Part II was lower than Part I, but it still showed that students were moderately successful in managing a page table and

Section	Submitted	Mean	Stdev	Median
Part I				
Section 1	89%	97.7%	9.5%	100.0%
Section 2	89%	91.6%	13.6%	100.0%
Combined	89%	95.8%	11.2%	100.0%
Part II				
Section 1	77%	81.1%	21.0%	90.0%
Section 2	67%	73.8%	22.3%	87.5%
Combined	74%	79.0%	21.5%	87.5%
Part III				
Section 1	17%	79.1%	8.3%	80.0%
Section 2	15%	77.5%	5.0%	80.0%
Combined	16%	78.7%	7.4%	80.0%
Parts I & II				
Section 1	89%	77.0%	25.6%	89.2%
Section 2	89%	67.9%	26.8%	70.0%
Combined	89%	74.2%	26.2%	88.3%
Parts I, II, & III (extra credit)				
Section 1	17%	110.6%	2.2%	111.7%
Section 2	15%	106.7%	3.9%	108.3%
Combined	16%	109.6%	3.0%	110.0%

Table 1: Project results by section and project (both by parts and overall).

	Part I	Part II	Part III	Total
LOC	76.7	220.0	146.7	297.5
Time (h)	5.3	7.4	4.0	16.8

Table 2: Average lines of code for implementation, and average time spent on the project (for each part, and in total).

physical memory. Note, that grades of zero given to students who did not complete a given project component were not factored into the respective scoring statistics for that part. Furthermore, "completion" is defined as simply having submitted some work product in an attempt to fulfill a given part, without regard to its actual completeness or correctness.

Due to the fact that Part III is provided as extra credit, only 15 students actually submitted this part. The mean score for both sections was 78.7%, which shows that students had average success with address translation on a more general page table design. Note, the percentages given for "Parts I, II, & III" in Table 1 reflect values only for the small subset of students who attempted all three project components. Because this group consists of only 15 students combined, we see significantly higher mean scores than those listed for the other components. The distribution of students' scores is shown further in Figure 1. In total, 64% of students received a score higher than 40 points (or 67%), which also reflects the difficulty of the project.

5.2 Implementation Workload

We wrote a script to obtain the lines-of-code for each part. Note that the total is not the strict sum as these three parts share a large portion of their implementation. A survey was administered to obtain the number of hours spent working on each part.

The results are shown in Table 2. Basically, the average LOC for this project was only 297.5, and the average number of hours spent

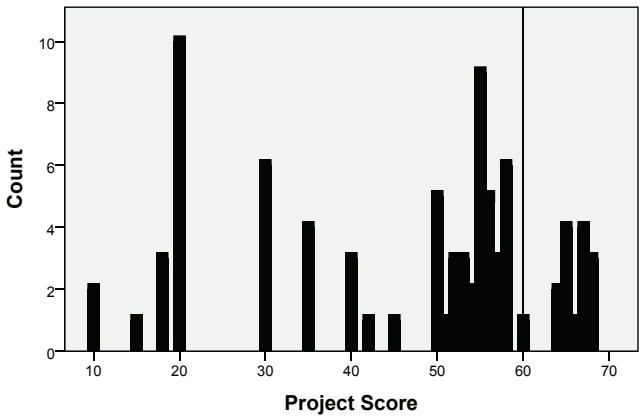

Figure 1: Distribution of student project scores, where the reference line shows maximum combined score for Part I and Part II; Part III is offered as a bonus worth 10 points.

working on this project was only 16.8. As described in Section 1, excluding five outliers, the average hours spent is only 10.8 hours. This indicates that the complexity of this project is totally acceptable, considering the importance of topics covered in an OS course. Also, there is space to extend this project to cover more concepts, if needed, particularly if assigned over a two-week time period.

By manually reviewing the results of 75 surveys, the most commonly reported problems encountered by students were regarding unclear instructions and lack of adequate understanding with respect to the LRU algorithm. As expected, Part II of the project was overwhelmingly cited by students when asked, "What were the most difficult parts of the project, and why?" (with 26 respondents). Specifically, understanding and implementing the LRU algorithm was the most common feedback received (with 10 respondents). Overall, comments that occurred regularly throughout the survey responses were: difficulty in visualizing the concepts; the function and implementation of the LRU algorithm; no means to independently verify correctness of output; understanding what was expected; uneven distribution of work; and too much effort required to complete Part II.

5.3 Exam Results

Table 2 shows the breakdown of scores across both sections, as well as the comparison section (Section 0). Students who did the project (Sections 1 and 2) scored a combined mean of 74% on Question 1, 82.5% on Question 2, and 90.8% on Question 3, which were the exam questions related to memory management. The range of the combined median scores, from 80% to 95% across all three questions, show that most students demonstrated an above average comprehension of memory management. As a comparison, Sections 1 and 2 performed better on the exam questions related to memory management than students in Section 0, who were not assigned the project. An independent samples t-test showed that there was a significant difference in scores between Section 0 ($M = 2.08$, $SD = 0.64$) and Sections 1 and 2 combined ($M = 2.47$, $SD = 0.45$); $t(101) = -3.335$, $p = 0.001$.

The mean exam scores for Section 0 were about 15% lower than the combined mean scores of Section 1 and Section 2 for Question 1 on virtual and physical address translation, and about 19% lower than Section 1 and Section 2 on Question 2 on calculating access time. For Question 1, the median score for Section 0 was about 25% lower than Section 1 and Section 2, which infers that more students were successful in translating virtual and physical addresses on the exam. For Question 2, though Section 0's median score of 80% was reasonably good, the median score for Section 0 was 10% lower than Section 1 and 2's median score on calculating access time. There were negligible differences between Section 0 and the other two sections for Question 3, which covered different page replacement algorithms. This result may imply that students do not have much trouble understanding different page replacement algorithms.

Question 1				
	n	Mean	Stdev	Median
NO PROJECT				
Section 0	22	57.3%	22.7%	55.0%
WITH PROJECT				
Section 1	56	72.5%	26.4%	80.0%
Section 2	25	78.2%	19.6%	85.0%
Combined	81	74.0%	24.6%	80.0%
Question 2				
	n	Mean	Stdev	Median
NO PROJECT				
Section 0	22	61.4%	38.6%	80.0%
WITH PROJECT				
Section 1	56	82.4%	16.9%	87.5%
Section 2	25	82.8%	21.7%	90.0%
Combined	81	82.5%	18.4%	90.0%
Question 3				
	n	Mean	Stdev	Median
NO PROJECT				
Section 0	22	89.1%	18.5%	100.0%
WITH PROJECT				
Section 1	56	91.6%	13.7%	100.0%
Section 2	25	89.0%	13.3%	95.0%
Combined	81	90.8%	13.6%	95.0%

Table 3: Statistics of examination scores, grouped by project participation status, for each question.

5.4 Perception of Project Value

Seventy-five students, across Sections 1 and 2, responded to the surveys. The survey results, shown in Figure 2, indicate that most students found the project valuable to learning five core memory management concepts. On a scale of 1 (Not valuable at all) to 5 (Extremely Valuable), most students across Sections 1 and 2 found the project to be valuable. Sixty-one percent (61%) of students rated the project experience as 4 or 5 (very to extremely valuable) with respect to understanding address translation, finding the page index inside a virtual address, and the frame number concept. Fifty-three percent (53%) of students rated the project experience as 4 or 5 (very to extremely valuable) with respect to physical memory management and the LRU algorithm.

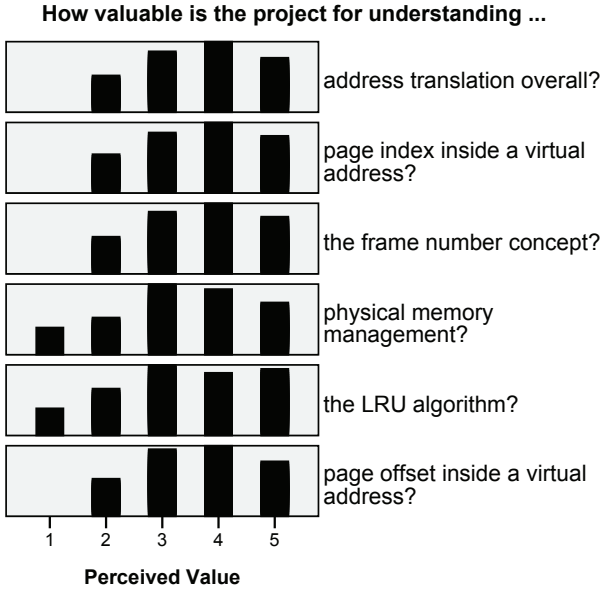

Figure 2: Valuability results on different concepts.

Although most students reported perceiving the project as valuable, they still noted some challenges with it, as well as specific memory management concepts. From the open-ended questions on what students found most difficult in the project, the most commonly reported problems included: unclear instructions and understanding what was expected from them; difficulty in visualizing the concepts; lack of adequate understanding with respect to the LRU algorithm; no means to verify correctness of output, since the answers were not given to students beforehand; uneven distribution of workload between the two required parts; and too much effort required to complete Part II.

6 DISCUSSION

The results showed that this project was beneficial for students in learning about memory management, particularly relating to activities involving address translation and calculating access times. Most students were able to complete the project and received high scores. They also scored higher on exam questions related to memory management as compared to a previous semester, which did not include this project. The only area this project did not necessarily improve upon, when compared to the previous semester, was that of applying different page replacement algorithms, which students appear to have little problem understanding. Student feedback was overwhelming positive in that students found it valuable to learning core memory management concepts. Further, the reported time spent working on the project showed that students spent an appropriate amount of time given the nature of this course. The complexity, as inferred from the grades and time spent on the project, was manageable for the students.

6.1 Potential Project Improvements

The evaluation results, particularly those from student surveys, have been used to improve the project design. One possible revision is to make the overall project a group effort with the hope that collaboration will lead to higher completion rates. Student surveys indicated that Part II took too much effort to complete, which indicates the possibility of dividing that part further. This revision would also make the third part required, which would help students have a more general understanding of page table design. Lastly, this project can teach students the concepts of temporal and spatial locality through utilizing (or constructing an approximation of) real-world memory access patterns for use as project input rather than the random sequences employed in these experiments.

Student surveys suggested that the teaching staff will need to create more resources to help students learn more about the concepts involved before asking to apply them in projects. Also, students recommended more visual resources to help them visualize the concepts. Students also suggested to incorporate inverted page tables, the use of a TLB, and implementing latency calculations for each memory access. The current project could be easily extended to incorporate these concepts. Students regularly expressed their desire to verify the correctness of their programs. Thus, results for a sample set of logical addresses will be provided in order to save students time in verifying the correctness of their programs.

6.2 Novelty

This project differs from other attempts at designing learning environments focusing on memory management. For example, Downey et al. designed a project that allows students to gain experience with paging and thrashing by observing the latency of memory accesses [6]. Hess et al. designed a project for changing the existing SLOB allocator such that the allocator may serve the request in either a best-fit or worst-fit fashion [7]. Laadan et al. designed several projects toward understanding internal OS memory usage, such as the working sets and modified sets of a process, as well as the page uses of each memory region [9]. Singh designed a project that also exercises some key concepts of memory management, such as address translation, updating a two-level page table, and physical memory allocations. However, this project requires the modification of an existing kernel, which may not be easy to control in terms of complexity [13]. These projects presented memory management concepts across several projects whereas our project presented them within the same project to minimize complexity.

Memory management in the user space is not completely new. Nieh and Vaill designed a user-space project for exercising the Buddy algorithm [8]. Arpaci-Dusseau requested students to implement a page fault handler and page replacement policy using LibOS, a library interface to OS services [2]. Null et al. designed their interactive tool based on a randomly generated address reference string [11] by showing students graphical representations of memory activities within the memory system. Thain designed a project that also exercises virtual memory concepts inside the user space [14], which is similar to this project's idea, but their project mainly focused on page replacement policies with page table management functions supplied. Therefore, their project does not provide students with a complete view of how different concepts interact with each other. Further, our project requires students to exercise different memory management concepts by themselves.

7 CONCLUSION

This paper presented a novel project design for an undergraduate Operating Systems course, which focuses on practicing core memory management concepts within the user space. Evaluation results show that this project greatly helps students to understand different concepts related to memory management, with reasonable complexity and effort. The project is now being revised and improved in response to the evaluation results, and will be implemented in subsequent offerings of the OS course. Finally, we have made the project available online at https://github.com/UTSASRG/USMM/.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Award CCF-1566154. The authors would like to thank Shuvra Chakraborty for her help of collecting partial data.

REFERENCES

[1] ACM/IEEE-CS Joint Task Force on Computing Curricula. 2013. *Computer Science Curricula 2013.* Technical Report. ACM Press and IEEE Computer Society Press. https://doi.org/10.1145/2534860

[2] Remzi H. Arpaci-Dusseau. [n. d.]. Project #3: A Real Page Turner. http://pages.cs.wisc.edu/~remzi/Classes/537/Spring2002/p3.html. ([n. d.]).

[3] Adam J Aviv, Vin Mannino, Thanat Owlarn, Seth Shannin, Kevin Xu, and Boon Thau Loo. 2012. Experiences in teaching an educational user-level operating systems implementation project. *ACM SIGOPS Operating Systems Review* 46, 2 (2012), 80–86.

[4] Linux Developers. [n. d.]. md5sum(1) - Linux man page. https://linux.die.net/man/1/md5sum. ([n. d.]).

[5] Jeff Dike. 2000. A user-mode port of the Linux kernel.. In *Annual Linux Showcase & Conference.*

[6] Allen B. Downey. 1999. Teaching Experimental Design in an Operating Systems Class. In *The Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education (SIGCSE '99).* ACM, New York, NY, USA, 316–320. https://doi.org/10.1145/299649.299796

[7] Rob Hess and Paul Paulson. 2010. Linux Kernel Projects for an Undergraduate Operating Systems Course. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE '10).* ACM, New York, NY, USA, 485–489. https://doi.org/10.1145/1734263.1734428

[8] Amit Jain. [n. d.]. Shell Project, part 2 (with Buddy System Memory Manager). http://cs.boisestate.edu/~amit/teaching/552/handouts/p6.pdf. ([n. d.]).

[9] Oren Laadan, Jason Nieh, and Nicolas Viennot. 2011. Structured Linux Kernel Projects for Teaching Operating Systems Concepts. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education (SIGCSE '11).* ACM, New York, NY, USA, 287–292. https://doi.org/10.1145/1953163.1953250

[10] Jason Nieh and Chris Vaill. 2006. Experiences Teaching Operating Systems Using Virtual Platforms and Linux. *SIGOPS Oper. Syst. Rev.* 40, 2 (April 2006), 100–104. https://doi.org/10.1145/1131322.1131323

[11] Linda Null and Karishma Rao. 2005. CAMERA: Introducing Memory Concepts via Visualization. *SIGCSE Bull.* 37, 1 (Feb. 2005), 96–100. https://doi.org/10.1145/1047124.1047389

[12] Ben Pfaff, Anthony Romano, and Godmar Back. 2009. The Pintos Instructional Operating System Kernel. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE '09).* ACM, New York, NY, USA, 453–457. https://doi.org/10.1145/1508865.1509023

[13] Jaswinder P. Singh. [n. d.]. Project V: Virtual Memory. http://www.cs.princeton.edu/courses/archive/fall14/cos318/projects/project5/p5.html. ([n. d.]).

[14] Douglas Thain. [n. d.]. Project V: Virtual Memory. http://www3.nd.edu/~dthain/courses/cse30341/spring2009/project5/project5.html. ([n. d.]).