USING ROLE PLAY FOR AN UPPER LEVEL CS COURSE[*]

Michael Leverington and Murat Yüksel
Computer Science and Engineering Department
University of Nevada, Reno
Reno, Nevada 89557
michael@edtech-teched.com
775 784 - 1414

Michael Robinson
Curriculum, Teaching, and Learning Department
University of Nevada, Reno
Reno, Nevada 89557
robinson@unr.edu
775 682 - 7531

ABSTRACT

This paper reports on initial experiences with using role play interaction in an upper level operating systems (OS) course. Three role play scenarios were implemented related to multi-programming, concurrency and synchronization, and a culminating experience that included several OS transactions with emphasis on input and output (I/O) operations. Students reported better understanding of the concepts, and stated appreciation for the chance to see some of the abstracted components made more concrete. The activities are discussed, the initial student feedback is provided, and future plans for improvement and further implementation of these and other scenarios in the future are briefly presented.

INTRODUCTION

The research [2,10,11] shows what experienced teachers already know related to teaching difficult concepts in any course. It takes student involvement and interaction to support learning and to elicit evidence that this has happened. In addition, a specific kind of student involvement or interaction called role play has become popular among Computer Science (CS) educators in recent years. Role play can be an effective educational tool because it allows student interaction and involvement by having them "act out" physical and/or conceptual components of a given system. While role play can be used in almost any environment, CS educators use role play for teaching concepts such as software engineering [9], introductory CS (CS1) [6], and more specifically for teaching Object Oriented Programming (OOP) concepts [1].

[*] Copyright © 2009 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

A variety of research related to the examples provided above was easily found, but there was little research or reporting on attempts to teach upper level technically-oriented courses using role-play activities, such as the Principles of Operating Systems course. There are some good reasons to consider role-play in this particular course, and at least a few interesting ways to implement it as will be presented in this report. The paper begins by discussing the problem and the need for the teaching process, and continues with a brief discussion of the implementation of three different role-play activities. In the concluding remarks, future plans for extension and adaptation of role-play activities in the course will be briefly presented.

THE LEARNING PROBLEM

Operating Systems (OS) on general purpose computers are developed and implemented to manage everything that the computer is able to do for a given user, which includes interacting through the keyboard, mouse, and monitor, managing data storage systems at various levels, scheduling and managing the processor, and correctly and appropriately interacting with hardware and software systems that may be added or removed at any time. In order to evaluate, analyze, or learn about a system with this level of complexity, it is unreasonable to try to understand all of its parts. Indeed, it is impossible to do so in a reasonable period of time. For this reason, the entire OS software must be considered as a package consisting of several abstracted layers and subsystems. This creates difficulty for educators, even in the upper levels of a Computer Science program.

Teaching and Learning Abstraction

Students in this course are at least in their third year, but more likely in the fourth year of their degree program, or they are graduate students. Given their previous coursework and experiences, it should be expected that they have worked with abstracted components. However, while they may have had experience with abstracting data and other functionality, it may be difficult to learn concepts relating to unfamiliar systems unless the students are allowed to interact with and discuss the components inside [4]. In any event, teachers report problems unless a balance can be struck between teaching the abstract and the application [3]. In this case, the balance must be found between fully abstracting a part of the OS or fully exposing the concrete components of that system.

Concept Difficulties

One of the primary realities is that the OS course, like many others in Computer Science, contains a significant amount of topical content, as mentioned previously. It would be difficult to teach this or many other CS courses without using lecture for much of the content. To its credit, lecture can be quite effective when well developed and well implemented. In this particular case, there was evidence of student learning to both depth and breadth in this course. However, both informally through verbal interactions, and formally through quizzes and examinations, there was evidence that the students were struggling with some of the course concepts. The course Instructor, who has extensive teaching experience with several courses including OS, was looking for ways to improve his students' learning.

A Possible Solution

Schoenfeld [12] argued that metacognition and appropriate belief systems are critical to problem solving. Metacognition is considered to be the set of cognitive tools that manage and organize the students' learning strategies, among other things. Belief systems are fundamental to the learning process, and have been constructed by students as a result of their previous experiences, sometimes in concert with, and sometimes in spite of, their formal learning. It is these belief systems that are relied upon when students attempt a problem-solving activity.

Others package and apply cognition, metacognition, and affective components into an acronym called BACEIS (Behavior, Affect, Cognition, Environment, Interacting, Systems) [7,8] and argue that the affective experience, in addition to the cognitive activities, is a major contributor toward solving problems. In order to activate the necessary student cognitive and metacognitive involvement in their learning, and to either inculcate new knowledge and/or overcome improperly constructed prior learning, students must appreciate the value and personally incorporate the reality of the concept(s) being taught.

Role play is a way to respond to these identified needs. The affective element is supported when the students have the opportunity to break away from the normal classroom activities and try out new things, and when they experience the exhilaration of deeper understanding of a concept with which they previously had been struggling. In addition, cognitive elements are challenged significantly when the students must tactically figure out what their next step or steps should be for the given OS circumstance.

It also requires significant metacognitive analysis and reflection to observe, interpret, and organize their knowledge as they watch and/or participate in the activities needed to manage a working operating system. In addition, out of much of the research related to beliefs and student ideas, it is known that direct contact with concepts or concept representations that present clear, unambiguous results will strongly support new concept learning, and/or diminish previously held misconceptions [5,13].

Finally, when reviewing the concepts taught in the OS course, it seemed that there were a few that offered excellent opportunities for role play, comparable, at the upper level, to the concepts taught with role play at lower levels [8]. Among those were some of the concepts with which students were struggling. It was decided to give role play a try in this course.

THE ROLE PLAYS

For reasons of development time, and again because class and lecture time is precious, it was decided to limit the pilot study to three for the first time, and some simple preliminary analysis and evaluation activities of the role play were developed to assist with deciding whether to continue with this study. The three role plays are described below, and some student feedback and analysis is provided thereafter.

Role Play 1: Multiprogramming

The first activity involved studying the creation and implementation of processes using the *fork* function, and sometimes the *exec* family of system calls. The *fork* operation is a little complicated to begin with since it returns two different values to the two different processes that exist after the call. Students need to recognize which process gets which response and by itself, this is not terribly difficult. However, once the two processes start implementing further actions, including possibly creating more processes, tracing the code and the results of the code activities becomes more complicated.

For the activity, two research staff established themselves as the system processor and the operating system software working together while the students played the role of processes. The course Instructor did not play a role so that he could be free to observe, interact with, and query the students. The role play itself involved running three sample sets of program code in a step-wise fashion so that students could see the actions and consequences of each segment of code. The first set of code was very simple, and is shown in Figure 1 below.

```
1  printf( "Process begins\n" );
2  pid1 = fork();
3  printf( "One fork completed\n" );
4  pid2 = fork();
5  printf( "Second fork completed\n" );
```

Figure 1: Simple Code Segment

The code in Figure 1 was given to the students to let them practice the forking action in advance. Using randomly drawn numbers, a student was first called to a section of the chalkboard where she placed her Process ID (PID) value (i.e., her randomly assigned number) at the top of a section of the chalkboard, and she then played the role of the process. The operating system (OS) and processor would direct her to take one step at a time which was made easy to follow by placing line numbers beside each step of code. The process called the output action (i.e., "printf"), and then implemented the first fork operation.

At this point, another student was randomly called to another section of the chalkboard. Each student was then directed to take one or two steps at a time by the OS/processor. All the students could see that the new person's return value "pid1" would be zero while the original person's "pid1" was assigned the PID of the new process. Later, both processes would call a fork operation again, adding two more students to the play.

```
printf( "Process begins" );
pid1 = fork();

if( pid1 > 0 )
  {
   printf( "First Parent ID: %i\n", pid1 );
   waitpid( pid1, NULL, 0 );
  }

if( pid1 == 0 )
  {
   printf( "First Child ID: %i\n", pid1 );
  }
```

Figure 2: Decision-Making Code Segment

Although it is not complete, the code in Figure 2 provides an example of the increase in complexity of decision-making and action for the students to follow. A second fork operation was implemented after this code with second "parent" and "child" output displays. The students were stepped through this code one process and one line at a time, observing the "wait" conditions, but not having to deal with which process stopped first. The third scenario placed the forking operations inside some other tests and a loop, and in this case, the students had to figure out where and how the processes -- that now had exit code in them -- were stopped.

Role-Play 2: Concurrency

The second activity involved concurrency and synchronization issues. Like the first activity, concepts and actions that are not easy to visualize and/or comprehend were targeted. In a format comparable to the previous activities, concurrency issues appear to happen in parallel, and it is sometimes difficult for students to follow all the action that is occurring. When race conditions become part of the problem, and common data quantities are being manipulated and possibly corrupted by more than one code segment, the overall operation can become confusing to students who are new to learning about operating systems.

The concurrency and synchronization activities were comparable to the multiprogramming activities in the sense that the activity was started with small code segments, and the students became the active threads. In this code, three threads were spawned and joined, and it was different in its actions in that "reader/writer" activities were implemented that would have parallel threads managing some incoming and outgoing data..

Initially, students were provided code that implemented given functions as threads and rejoined the threads upon completion. As before, the students would go to the chalkboard acting as threads, but this time, they had to manage both local and global data in addition to implementing their other specified actions. After running a few steps into the threaded code to see the process work, the OS/processor then stepped the threads through in such a way as to corrupt the outgoing data.

The second scenario included using the same code but implementing functions that disabled the interrupts before and after the critical code sections. Subsequent scenarios followed, including test-and-set (TSL) operations and the use of semaphores to protect the operations.

Role Play 3: I/O Operations

The third activity started out with a focus on identifying response- and reaction-time differences related to I/O operations and devices. Examples would be: 1) system bus memory access to the processor compared to loading hard drive data to memory through the I/O bus with or without Direct Memory Access (DMA) support; 2) access or response time of the hard drive or a CD-ROM drive related to the other system components such as the buses, the memory or the processor; and 3) interrupt actions from a variety of sources, including I/O devices and the DMA controller. In the course of developing these things, it was found that most of the internal hardware and operating system interactions would be presented. This was acceptable because it was near the end of the semester and the overall OS actions could be reviewed.

Several student desks and chairs were set up at the front of the classroom organized in generally the same way a high-level computer structure might be set up. Once the desks and chairs were placed, the students were invited to fill them. A "loading the kernel" script was prepared in advance, but the students were informed that they would have to learn their tasks and become much more autonomous in their actions once the "computer" was booted.

3 x 5 index cards were used as data with one color representing interrupts, one color representing data quantities for the OS, and two other colors for programs that would be loaded. The students worked through "loading" the kernel and some supporting OS components so they would become comfortable with their roles, and they were then informed that the user had requested to load and run a program. A time-step display was provided on the classroom screen, and the index cards were passed between students over the data buses with one time step allowing one bus location movement. The scenarios included one data-intensive program and one I/O intensive program; the operational differences between these two programs became clear as the activity progressed.

DISCUSSION AND CONCLUSIONS

The feedback from the students was virtually all positive. More importantly, some of the student comments directly responded to the issues upon which these activities were designed. One comment, "I didn't understand how processes worked before this activity. I thought that all processes executed simultaneously but couldn't see how that was possible. Now I understand the stepping is crucial to how they are executed", provided evidence that the student really needed to see the process in action, and at length, to understand it. Another one, "Sometimes it feels like we aren't even talking about a real thing since I can't take it apart or put it together. Seeing how an OS works was tremendously helpful", speaks directly to the challenge stated at the beginning of this paper that is related to making some of the high-level abstractions concrete. Still another comment states the same thing: "Seeing people as processes made more sense than some abstract "process", and when the other students made a mistake, I was usually thinking the same thing, so it helped me see what I was doing wrong or assuming".

Those comments that were not completely positive were made by individual students who felt that they had already come to understand the material: "I understood most of the details prior", and "I already realized it, but if I hadn't, this would definitely have helped". All comments were collected anonymously along with short questionnaires that also demonstrated positive student feedback. This data will be added to future feedback quantities as triangulation components as the research on these activities evolves.

This was a first attempt at using role play to teach critical components of this upper-level course. However, there is enough positive feedback to support further pursuit of this activity-based approach. From the experience, it has been recognized that more initial training should be given to the students so they become more comfortable and more fluent with their roles. It was also found that even though the scripting must be kept pretty tight, some students suggested that they be challenged to come up with the next steps.

There were some students who commented that the operation appeared disorganized. Video recordings had been made and were reviewed for two of the activities, and some on-the-fly adjustments were observed. However, it did not appear that there were serious management flaws in the operation. However, it is theorized that these role-play activities might be the very first interactive kinds of activities that some of these students have experienced, and they simply might not be familiar with how a classroom alternative to lecturing might look.

The other improvements that can be made are mostly logistical adjustments made in response to student feedback. Outside of direct student feedback, future plans include pre/post quizzes and exam topic comparisons to previous (i.e., non-interactive) student groups will be studied. This was not implemented for the first pass so that the process could be refined and fluent before it was evaluated. Future plans also include memory management and processor scheduling activities. The student feedback and evidence of improved understanding seems to support further implementation of this strategy.

REFERENCES

[1] Bennedsen, J., Caspersen, M.E., Programming in context - a model-first approach to cs1, *ACM SIGCSE Bulletin*, *36*, (1), 477-481, 2004.

[2] Black, L., Interactive whole class teaching and pupil learning: theoretical and practical implications, *Language and Education*. *21*, (4) 271-283, 2007.

[3] Carbone, A., Mannila, L., Fitzgerald, S., Computer science and it teachers' conceptions of successful and unsuccessful teaching: a phenomenographic study, *Computer Science Education*. *17*, (4), 275-299, 2007.

[4] Darling-Hammond, L., Bransford, J., *Preparing Teachers for a Changing World*, San Francisco, CA: Jossey-Bass, 2005.

[5] Donovan, M.S., Bransford, J.D., *How Students Learn: Mathematics in the Classroom*, Washington, D.C.: The National Academies Press, 2004.

[6] Goode, J., Increasing diversity in k-12 computer science: strategies from the field, *ACM SIGCSE Bulletin*, *40*, (1), 362-366, 2008.

[7] Hartman, H.J., *Metacognition in Learning and Instruction*, Norwel, MA: Kluwer Academic Publishers, 2001.

[8] Hartman, H.J., Sternberg, R.J., A broad baceis for improving learning, *Instructional Science*, *21*, (5), 401-425, 1992.

[9] Henry, T.R., LaFrance, J., Integrating role-play into software engineering courses, *J. of Computing Sciences in Colleges*, *22*, (2), 32-38, 2006.

[10] Holt, L.C., Kysilka, M.L., *Instructional Patterns: Strategies for Maximizing Student Learning*, Thousand Oaks, CA: SAGE Publications, 2005.

[11] Kane, L., Educators, learners and active learning methodologies, *Int J. of Lifelong Education*, *23*, (3), 275-286, 2004.

[12] Schoenfeld, A., *Mathematical Problem Solving*, Orlando, FL: Academic Press, 1985.

[13] Smith, J.P. III, diSessa, A.A., Roschelle, J., Misconceptions reconceived: a constructivist analysis of knowledge in transition, *The J. of the Learning Sciences*, *3*, (2), 115-163, 1993.