RCOS.JAVA: A SIMULATED OPERATING SYSTEM WITH ANIMATIONS

David Jones, Andrew Newman

ABSTRACT
RCOS.java (Ron Chernich's Operating System) is a Java-based, simulated operating system designed to address student difficulties in understanding operating systems concepts. This paper describes the rationale, design, features and planned use of RCOS.java. The intent with RCOS.java is to emphasise active, student-based learning and the development of the higher level learning skills of analysis, synthesis, evaluation or problem solving.

KEYWORDS
Simulation, Animations, Operating Systems

INTRODUCTION

A course covering the algorithms, concepts and theories behind the design, construction and operation of operating systems is a recommended core component for courses in computer science [7]. For a number of reasons students find operating systems related topics, like many computer science topics, difficult to relate to and learn [17].

RCOS.java is a Java-based, simulated operating system that attempts to address some of these problems by providing students the opportunity to observe animations and to actively experiment with the algorithms, data structures and services of an operating system. RCOS.java builds on lessons gathered during the use and development of previous computer-aided learning systems at Central Queensland University (CQU) [9,10].

A unique characteristic of RCOS.java is that it combines the features of previous operating systems courseware such as animation [18], concurrency simulation [5,22] and toy operating systems [9,10] into a single system with the features of Java and the World-Wide Web. Importantly RCOS.java includes features that can be combined with appropriate pedagogy to offer support for active and discovery-based learning.

CONTEXT

The current shape of RCOS.java has been influenced by a number of contextual factors including the characteristics of CQU students, the delivery modes used at CQU and previous experience with teaching operating systems at CQU. This section sets the scene for the development of RCOS.java by describing these contextual factors.

Students and Delivery Mode

The primary purpose of RCOS.java is to address problems experienced during the teaching of the course 85349, Operating Systems at Central Queensland University (CQU). CQU is an multi-campus institution based in Australia with four branch campuses based in Central Queensland, five campuses (referred to as the International Campuses) managed by a commercial partner in Sydney, Brisbane, Melbourne, the Gold Coast and Fiji, international partners offering supported distance education to students in Singapore, Malaysia and Hong Kong and a large cohort studying via print-based distance education. In many courses less than 20% of students are on the same campus as the instructor in charge of the course.

85349 Operating Systems is CQU's traditional operating systems course and was a core course in the second year of CQU's Bachelor of Information Technology degree until 1999. 85349 averages around 200 students per offering with approximately half those students studying via print-based distance education. In recent years it is common for less than 40 students to be located on the same campus as the course coordinator for 85349.

Distance education students receive a study guide, textbook and other resources as their primary learning material. Most distance students never talk face-to-face with a course coordinator or visit a CQU campus. An increasing number of CQU courses, including 85349 (http://www.infocom.cqu.edu.au/85349/), make some use of the Websites and mailing lists. All CQU, distance, computing students are expected to have access to an Intel-based computer running a version of the Windows operating system. The software used in all CQU computing courses is designed to run on that platform.

On-campus students learn via traditional on-campus delivery approaches including lectures and tutorials. Lectures to the students at the Central Queensland based campuses of CQU are given by a course coordinator at one campus and distributed via video-conferencing to the other three campuses. Each of the International Campuses has a lecturer who is responsible for organising lectures and tutorials for students at that campus. As a result this means that in some courses there can be up to five or six essentially independent staff at different campuses giving the same lecture.

Previous Approaches

Over the last ten years a combination of four basic approaches have been used at CQU to teach operating systems. The following discusses briefly these approaches, how they were implemented and the CQU experience. The four approaches are: purely theoretical, separate unrelated projects, simulated operating systems, and the use of a complete operating system.

The purely theoretical model introduces concepts and explanations with no practical application or demonstration. Many students never fully understand the relevance or meaning of the concepts and theory without the reinforcement and demonstration practical application provides [10]. Many of the older operating systems textbooks [24,32] follow this method with the assumption that course coordinators will provide the practical activities. More recent operating systems textbooks have started to mention and incorporate some of the existing operating system courseware such as Nachos [10], OSP [21], and BACI [5].

To provide the practical demonstration of the theory and the ability for students to test their understanding it is possible to introduce a collection of unrelated practical projects and

activities. Examples of individual projects from the literature include physical demonstrations [36], animations [18] concurrency simulators [5,22], and quizzes [16].

Often these resources do not meld together in any way and generally don't deal directly with an operating system. Consequently students often have difficulty relating the activity with the operating systems principles that the activity is meant to explain. There is also a tendency to produce students with an understanding of each separate concept but without a grasp on how these concepts fit together to produce an operating system [37]. Additionally, the observation that students work hardest at assessable activities leads to students who understand only the principles associated with those activities that were assessable.

To address the lack of integration provided by separate projects some classes use a simulated operating system that provides an environment in which most operating system concepts can be shown. The intent of using a simulated, rather than a real operating system, is to reduce the learning curve associated with becoming familiar with the system. However, experience using PRMS (Process and Resources Management System)[18] and RCOS [9,10] has shown that the effort required by students to understand a simulated operating system can still be considerable [10].

Additionally the simulation may not exhibit exactly the behaviour of a typical operating system [14, 31, 37]. As a result the student must perform the conceptual leap from the behaviour of the simulation to real operating system behaviour.

The 1992 offering of 85349 made use of PRMS [18]. PRMS was used to demonstrate the operation of operating systems data structures via animation and allow students to modify the source code and investigate the outcomes. While the animation provided by PRMS was well received by students it only covered two of the areas studied in operating systems and had numerous design and implementation problems which made it difficult for students to modify. In particular the combination of animation code with operating system code confused students and made it difficult for them to form appropriate mental models of the operating system.

Drawing on the lessons learnt during the use of PRMS and a wish list of desired features [10] RCOS was built during 1993 with further development occurring during 1994. RCOS is a portable, multi-tasking operating system (OS) designed to demonstrate general operating system principles through controlled animation, and to allow simple modification and experimentation with OS data structures and algorithms.

The use of RCOS during 1995 and 1996 was a success with 74% of students agreeing that helped them understand operating system concepts with 28% believing it was too difficult to use [9]. Use of RCOS in 85349 ended in 1996 due to the transition of CQU students away from MS-DOS to the Windows platform.

Using a real operating system means that students are able to experiment with a real system and its implementation details. However the students must climb an even steeper learning curve than that involved with a simulated operating system [19]. The sheer speed and complexity of modern computer systems make them difficult for the average student to understand [26].

An attempt in the early 1990s to use the Minix [35] operating system at CQU confirmed these difficulties. Many distance students were not able to install Minix let alone observe it in action

[10]. However, in an appropriate context a real operating system can be useful in teaching operating system concepts.

An example of this is the responses to a question posted on Slashdot (http://www.slashdot.org/) in late 2000. It was a request about what systems Universities were using to teach operating systems. Responses from students throughout the world mentioned the following operating systems (both complete and toy): Mach, Yalnix, Linux, Minix, RoadRunner, Nachos, OSP, Xinu, Lego, uCOS, DLXOS, L4. (http://slashdot.org/askslashdot/00/12/08/2339254.shtml)

WHY IS LEARNING ABOUT OPERATING SYSTEMS DIFFICULT?

The study of operating systems involves gaining an understanding of both the responsibilities of an operating system, the algorithms and data structures used in implementing these responsibilities, and how an operating system interacts with the other components of a computer system. Teaching operating systems involves balancing coverage of a significant amount of theory with the desire to provide some practical relevance of the material and has always been a challenge [30].

The economic pressures and the tendency to open access in today's Universities results in courses having large enrollments including students from all walks of life. The variety in the background, culture and capabilities of students makes it exceedingly hard to introduce the difficult, abstract concepts of operating systems. A further problem at CQU is that very few of the students entering 85349 have the prerequisite knowledge expected by most of the common operating system textbooks. Further adding to the difficulty is the inconsistent use of terminology throughout the field. Two different textbooks may have subtly different definitions for important concepts. These differences become very confusing for students attempting to grasp the concepts.

A large number of the concepts introduced in operating systems courses (e.g. processes and concurrency) are abstractions and are not easily understood [36]. Additionally many of these concepts are dynamic concepts that cannot be easily explained in a static medium such as print. While operating systems texts use images to illustrate concepts they do not portray the dynamic nature of operating systems abstractions [36]. Students find it easier to understand these concepts when they are grounded to visual objects.

Studies have shown that relatively few students reach an acceptable level of achievement in science and mathematics [8]. Ben-Ari [2] suggests that students often have no effective model of a computer as an accessible ontological reality and that this is a serious obstacle to learning computer science. Many of the traditional approaches used to teach operating systems rely on behaviourist teaching practices and do not explicitly address the construction of a model of the operating system by students. Without a good theoretical base students revert to rote learning.

It is now widely held that objectivist approaches suffer a number of weaknesses that decrease their educational effectiveness. For example, Confrey [13] suggests that objectivist approaches require that the instructor, rather than students, is responsible for simplifying the contents of a knowledge domain. As a result the objectivist approaches present students with a simplified model of complex knowledge and deprive the students of the benefits gained by simplifying

the knowledge domain. It can be said that modern day operating systems textbooks contain little more than a simplification of the complex knowledge domain that is operating systems.

Ben-Ari [2] believes that many of the phenomena observed in the computer science literature, such as the haphazard construction of computer science concepts and the perception that computer science is hard, can be avoided by using a constructivist approach. For example, contemporary theories of learning place emphasis on gaining knowledge through appropriation of information. This is central to a constructivist tradition of learning that emphasizes the role of the learner in knowledge creation. Implications for teaching involve encouraging students to plan and define their goals for learning based on their existing knowledge; allowing students opportunity to examine their processes of thinking and problem solving, that is to become meta-cognitive processors of information; and providing learning experiences that engage the learner.

The importance of student activity, especially in the construction of mental models, can be seen in the literature on the use of animation in computer science education. While students report feeling that animations aid their understanding of algorithms [33] there have been a number of reports showing limited or negative results (e.g. [6]. In the majority of cases the positive result from the use of animation was small and not statistically significant [34]]. Pane et al [29] suggests that the challenge is to engage the student in the activity to encourage students to actively experiment with animations and simulations. Designing and implementing an animation forces the student to identify the fundamental operations of the algorithm. It encourages the student to become the teacher [34]. Bergin et al [3] attribute higher motivation and better integration and retention of content to the active engagement on the part of students.

RATIONALE AND AIMS OF RCOS.JAVA

The design and construction of RCOS.java was influenced by the CQU context, previous experience in teaching operating systems and the literature on educational technology. However, the initial spark for the RCOS.java project was provided by a combination of the release of Windows'95, a move at CQU from Pascal to C++ as the first year programming language, the arrival of Java, the Java Cup International and the presence of three advanced-level students ready to work.

During 1995 it become obvious that CQU would adopt C++ as the first year programming language. RCOS supported the ability to execute programs written by students using the RCOS Pascal compiler. Rather than expect new students to learn Pascal simply to use RCOS it became necessary to modify the RCOS compiler to compile C/C++ programs.

Implemented as an MS-DOS application RCOS relies on a number of low-level BIOS calls. The use of these calls meant that the operation of RCOS under Windows'95 was problematic and would become increasingly so with the development of Windows NT. While RCOS was designed to be portable the effort involved to port RCOS to the Windows32 platform was not trivial. However, with increasing use of the newer Windows operating systems amongst CQU students it became necessary to identify an alternative.

In late 1995 the hype surrounding the Java programming language was just commencing. One

of the main features of Java being promoted was portability. The ability to write code once and then run it on any system. This feature promised to protect the investment required to write a system as complex as RCOS.java. To encourage interest Sun Microsystems organised a Java contest with attractive prizes and a category for educational applications (http://java.sun.com/applets/archive/javacontest/).

The long terms goals in developing RCOS.java were to provide a system that would

- reproduce all the benefits and capabilities of RCOS, RCOS.java had to provide a simulation of an entire operating system and supporting hardware with code that was simple for students to understand and modify. This should include the ability to execute student programs and provide animated representations of the internal state of the operating system and hardware.
- address some of the problems experience with RCOS, and The major problems was that portability could only be achieved by rewriting low-level support functions.
- harness improving technology to provide new possibilities. For example, use of the platform independence, distribution, and multimedia capabilities.

The pedagogical emphasis in the development of RCOS.java is to provide a system to make it easier for students to construct and test mental models of operating systems. Animation of the operating system operation is intended to aid in the construction of mental models. While the ability for students to interact with the animation, write their own programs, and to modify the internals of RCOS.java are provided to enable students to test their mental models.

THE FEATURES OF RCOS.JAVA

RCOS.java is currently a Java application consisting of 250 classes, with over 100,000 lines of source code. The classes can be broken into five major categories: hardware, operating system, animation, message system, p-code compiler. RCOS.java is an open source project and the code and associated resources are available from http://rcosjava.sourceforge.net/

Hardware

RCOS.java implements a collection of simple hardware devices including CPU, Disk Drive, RAM and terminals. The CPU is a stack-based, p-machine CPU [12] and is based on the CPU used in RCOS. The disk is based on an IBM 3740 8 inch diskette. RAM has 20 elements each of 1024 bytes. User input/output is via 8 text-only (80x15) "green screen" terminals.

The simplicity of the hardware devices is intended to ease the initial student learning curve. The flexibility inherent in the design of RCOS.java means that at a later stage more complex simulated hardware could be introduced.

Operating System

The RCOS.java operating system is a micro-kernel, message passing operating system. The micro-kernel handles interrupts, context switching and generates messages to other operating system components to perform traditional operating system services such as memory

management, process and disk scheduling.

Each of the operating system components provides a layer of abstraction and implements various operating system specific algorithms. For example the process scheduler allows students to choose from round robin, FIFO (First In, First Our) or priority based scheduling algorithms. Operating system components that have been implemented include a disk drive scheduler, a memory manager, inter-process communication handler that supports semaphores and shared memory, a process scheduler and terminal manager.

Animation

RCOS.java has a number of classes, referred to as animators that are responsible for providing a graphical representation of what is happening with the simulated operating system and hardware. Some animators also provide methods by which the user can modify the operation of the operating system and hardware in real time. For example, the user can see the lifecycle of a program and modify its execution behaviour while it is running and also modify the algorithms used for process and disk scheduling.

Current animators include the:

- Process scheduler animator,
  The process scheduler displays the Zombie, Blocked, Ready and running queues. It allows modification of the scheduling algorithm, the quantum, and the speed of the animation display. Figure 1 is a screen dump of the animator for the process scheduler.
- IPC manager animator;
  Provides a view of currently allocated memory, semaphore values and of shared memory. It shows in real time memory the allocation, reading and writing of memory.
- CPU animator.
  The CPU Animator shows an in-depth view of the P-Code CPU including the current stack, register values and assembly language instructions.

While the current interface is based upon a multi-window format (one window for each animator) the flexibility of the design enables the interface to be modified without need of changing the underlying operating system or messaging system. The animation system has also been designed to enable the messaging system to interact with allowing text descriptions to be displayed against a currently displayed animator.

Message system

The components of the RCOS.java operating system communicate via the use of RCOS messages. The messaging system uses a number of software patterns to increase flexibility. Each message is placed into three broad categories based on its destination: animator only, operating system only, and both (animator and operating system bound).

Since all activities in RCOS.java achieve through the distribution of messages it is possible to add a recording feature. This feature is designed to allow a session with RCOS.java to be saved and played back at a later time. The applications of the recording feature include students or teachers creating RCOS.java sequences to demonstrate and explain particular operating system concepts. The use of an XML based recording format allows the saved data to be modified and used by other XML aware applications.

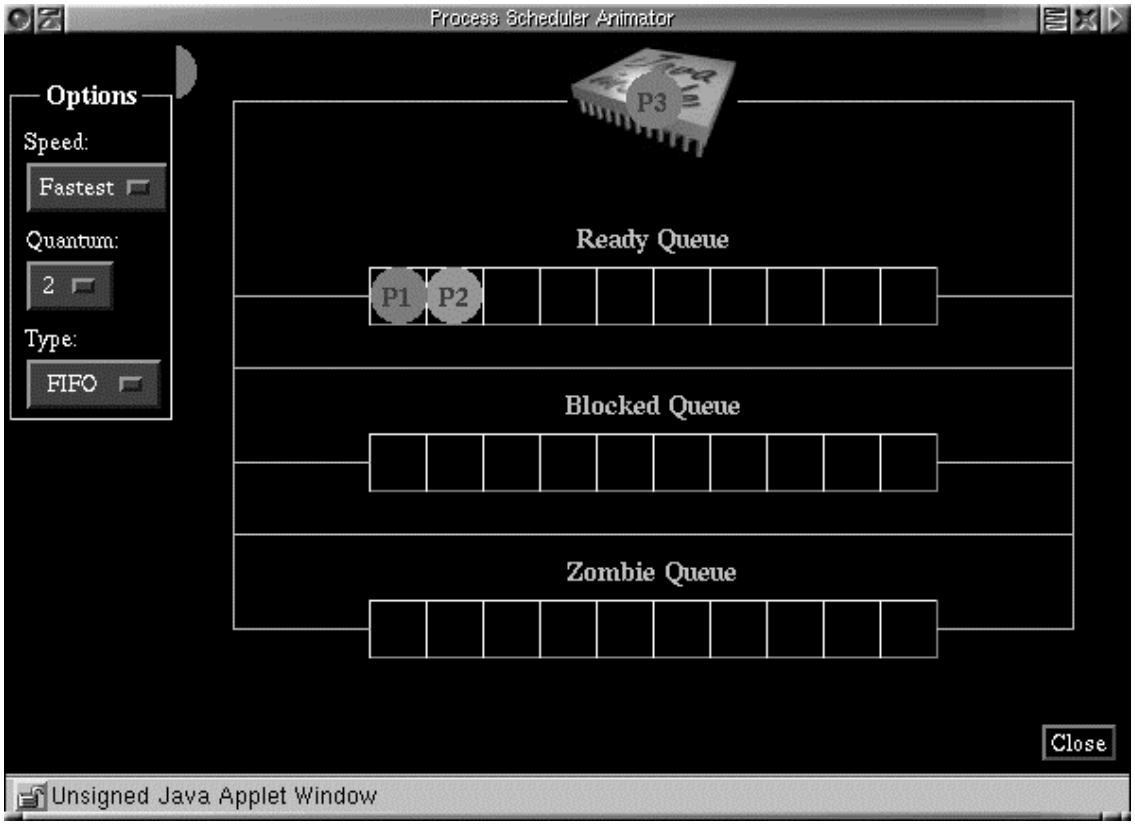
Fig. 1: RCOS.java, Animator for process scheduler

P-code compiler

The RCOS.java CPU executes p-code [12]. The P-Code compiler distributed with RCOS.java is implement in Java and compiles a simplified C/C++ syntax. The compiler allows students to create their own applications to be run in RCOS.java. The compiler was implemented in Java using ANTLR (ANother Tool for Language Recognition) tool. ANTLR provided a C/C++ grammar, which was modified it to include various specific RCOS, commands for disk drive, interprocess commands and memory management. Only simple compilation is allowed with libraries and other advanced C/C++ features not implemented.

FUTURE PLANS

Since the initial implementation RCOS.java has been redesigned, recoded and converted to Java 1.2. The choice of implementation in the Java language, and Java's subsequent "success" as a language, has ensured that any time and effort has not been lost over the long coding time. Future plans for RCOS.java revolve around two major tasks: developing appropriate educational uses of RCOS.java and further improving the features and capabilities of RCOS.java.



Due to a number of factors RCOS.java has not yet been used to teach operating systems. Computer-based learning (CBL) has been widely criticised for using poor teaching strategies. Van der Brande [38] claims that 'only 3% of educational software has been written in the

context of an articulated pedagogic rationale' (p23). The intent with RCOS.java is to emphasise active, student-based learning and the development of the higher level learning skills of analysis, synthesis, evaluation or problem solving. Achieving this lofty goal is no easy task and will take a significant effort over the next year.

Future planned extensions to RCOS.java include upgrades to the current RCOS.java hardware including more complex CPU, disk drive, terminals and the addition of new hardware devices such as network cards. Further expansion of the recording mechanism to handle different forms of multimedia, such as audio, is also planned.

CONCLUSION

RCOS.java is a simulated operating system implemented in Java and provides a number of features intended to aid students in gaining an understanding of the algorithms, concepts and theories behind the design, construction and operation of operating systems. These features include

- animation of the internal operation of the operating system and hardware
- the ability to write, compile and execute "RCOS.java programs"
- the ability to read and modify the code which implements the RCOS.java "operating system"
- the ability to create "scripted" RCOS.java sessions to demonstrate particular topics.

Development work on RCOS.java is continuing as an open source project. The code and related resources for RCOS.java can be found at http://sourceforge.net/projects/rcosjava/

REFERENCES

1.W. (Tony) Bates, Technology, Open Learning and Distance Education, Routeledge, 1995

2.Mordechai Ben-Ari, Constructivism in Computer Science Education, SIGCSE Bulletin, 30(1), 1998, pp 257-261

3.Bergin J., Brodlie, K., Goldweber, M., Jimenez-Peris, R., Khuri S., Patiiio-Martmez, M., McNally, M., Naps, T., Rodger S., and Wilson, J. An Overview of Visualization: its Use and Design. Proceedings of ITiCSE'96, pp 192-200

4.Boroni, C., Goosey, F., Grinder, M., Rockford, R., and Wissenbach, P. WebLab! A Universal and Interactive Teaching, Learning, and Laboratory Environment for the World Wide Web. Proc of the SIGCSE Technical Symposium 28 (1:199-203, March 1997).

5.Bill Bynum, Tracy Camp, After you, Alfonse: A Mutual Exclusion Toolkit, Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education, SIGCSE Bulletin, 28(1), March, 1996, pp 170-174

6.Michael D. Byrne, Richard Catrambone, John Stasko, Do algorithm animations aid learning? Technical Report GIT-GVU-96-18: Graphics. Visualization and Usability Centre, Georgia Institute of Technology, Atlanta, GA, August 1996

7.Duit, R. (1991), Students' conceptual frameworks: consequences for learning science, in The

Psychology of Learning Science, S. M. Glyn, R. H., Yeany and B. K. Britton (eds), Lawerence Erlbaum Associates, 1991

8.Ron Chernich, Bruce Jamieson, David Jones, RCOS: Yet another teaching operating system, Proceedings of the First Australasian Conference on Computer Science Education.

9.Chernich, R., Jones, D.. (1994). The Design and Construction of a Simulated Operating System. Proceedings of Asia Pacific Information Technology In Teaching and Education '94, Brisbane. pp 1033-1038,

10.W A Christopher et a. (1993), The Nachos Instructional Operating System. Proceedings of the Winter 1993 Usenix Technical Conference, pp 481-489.

11.Chung Kin-Man and Yuen H (1978): A "Tiny" Pascal Compiler, Byte Magazine, Byte Publications Inc., Petersborough N.H., USA, Volume 3, Numbers 9, 10, 11, September, October, November 1978.

12.Confrey, J. (1990). What constructivism implies for teaching. In Constructivist Views on the Teaching and Learning of Mathematics. National Council of Teachers of Mathematics (pp.107-122), Reston (Virg).

13.Peter Denning, Douglas E Comer, David Gries, Michael Mulder, Allen Tucker, A Joe Turner, Paul R Young (1989), Computing as a Discipline. Communications of the ACM, 32(1), pp 9-23

14.Goh, A. (1992). An Operating Systems Project. ACM SIGCSE Bulletin, 24(3), pp 29-34

15.Tony Greening, WWW support of student learning: A case study, Australian Journal of Educational Technology, 1998, 14(1), pp 49-59

16.Murray W Goldberg (1996), Calos: An experiment with computer-aided learning for operating systems. In Proceedings of the ACM's 27th SIGCSE Technical Symposium on Computer Science Education, pp???

17.D.G. Hannay, (1992), Hypercard automata simulation: finite-state pushdown and Turing machines, SIGCSE Bulletin, 24(2), p55-58

18.James H Hayes, Leland Miller, Bobbie Othmer and Mohammad Saeed (1990), Simulation of Process and Resource Management in a Multiprogramming Operating System, Proceedings of the 21st ACM Technical Symposium on Computer Science Education, February, 1990, p125-128.

19.Kavka, C. et al. (1991). Experiencing Minix as a Didactical Aid for Operating System Courses. ACM Operating Systems Review, 25(3),

20.Kehoe, Colleen and Stasko, John T., Using Animations to Learn about Algorithms: An Ethnographic Case Study, Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA, Technical Report GIT-GVU-96-20, September 1996.

21.Kifer, M. Smolka, S. (1992). OSP An Environment for Operating System Projects. Operating Systems Review, 26(4), pp 98-99.

22.Barry Kurtz, Hong Cai, Chris Plock, Xijia Chen, A Concurrency Simulator Designed for Sophomore-level instruction, SIGCSE Bulletin, 30(1), 1998, pp 237-241.

23.Andrea W. Lawrence, Albert Badre, John Stasko, Empirically evaluating the useof animations to teach algorithms. Proceedings of the 1994 IEEE Symposium on Visual Languages, pp 48-54, St Louis, October 1994

24.Lister, A.M. Eager, R.D. (1988). Fundamentals of Operating Systems (fourth edition). London: Macmillan Education Ltd.

25.Colin McCormack, David Jones (1997), Building a Web-based Education System, John Wiley & Sons, New York.

26.Thomas L. Naps, Algorithm visualization on the World Wide Web---the difference Java makes! Proceedings of ITiCSE'97, Uppsala Sweeden Pages 59-61

27.Urban Nulden, The ExCon Project: Advocating Continuous Examination, SIGCSE Bulletin 30(1), 1998, pp 126-130

28.John Pane, Albert Corbet, Bonnie John, Assessing Dynamics in Computer-Based Instruction, Common Ground: CHI 96 Conference Proceedings (1996), New York: ACM Press.

29.Alfredo Perez-Davila, OS Bridge Between Academia and Reality, Proceedings of the 26th SIGCSE Technical Symposium on Computer Science Education, SIGCSE Bulletin, 27(1), March 1995, pp 146-1481

30.Ramakrishnan, S. Lancaster, A. (1993) Operating System Projects: Linking Theory, Practice and Use. Proceedings of the 24th ACM Technical Symposium on Computer Science Education, 256-260.

31.Stallings, W. (1992). Operating Systems. New York: Macmillan Publishing.

32.John Stasko, Albert Badre, Clayton Lewis, Do algorithm animations assist learning? an empirical study and analysis, Proceedings of the INTERCHI'93 Conference on Human Factors in Computing Systems, pp 61-66, Amsterdam, April, 1993

33.John Stasko, Using Student Built Algorithm Animations as Learning Aids, Proceedings of the 28th Technical Symposium on Computer Science Education, 1997, pp 25-29

34.Tanenbaum, A. S. (1987) Operating Systems: Design and Implementation. Englewood Cliffs, NJ: Prentice Hall

35.Rodney S. Tosten, Using a Model Railroad system in an Artificial Intelligence and Operating Systems Course, ACM SIGCSE Bulletin, 25(1), March 1993

36.Withers, J.M. Bilodeau, M.B. (1992). An Examination of Operating Systems Laboratory Techniques. ACM SIGCSE Bulletin, 24(3), 60-64

37.van der Brande, L. (1993), Flexible and Distance Learning, New York, John Wiley.

David Jones
Faculty of Informatics and Communication
Central Queensland University
Rockhampton, 4701
Australia
Email: d.jones@cqu.edu.au

Andrew Newman
Bluedog Technologies
Brisbane, 4064
Australia
Email: andrew@bluedog.com.au