An Integrated Approach to Teaching Computer Systems Architecture

Umakishore Ramachandran   **William D. Leahy Jr.**

College of Computing
Georgia Institute of Technology
e-mail: **rama@cc.gatech.edu**, **bleahy@cc.gatech.edu**

Abstract

At Georgia Tech, since the Fall of 1999, we have been teaching a first course in systems that represents a radical departure from the usual stovepipe model of teaching computer architecture and operating systems. By making this course a required one for CS majors in their sophomore year, we have accomplished several goals the most important of which is the opportunity for students to pursue deeper exposure to systems in their junior and senior years, through additional courses and research, if they so choose. The pedagogical style embodied in this course fosters a good understanding of the symbiotic relationship between hardware and software for the students early on in their undergraduate experience.

Index Terms: Computer Systems, Operating Systems, Computer Architecture, Computer Systems Organization

1. Rationale for the new approach

Most undergraduate institutions teach Computer Architecture and Operating Systems as two separate courses. However, it is well known among academicians and practitioners that there is a symbiotic connection between the systems software and the hardware.

The design of instruction sets is influenced by high level language. The OS abstractions (such as process, threads, and page tables) are influenced by the details of the processor and memory hardware. The design of the network protocol stack is influenced by the characteristics of the network interface and the vagaries of the physical network. The list goes on… Unfortunately, most students never see this connection when these two courses are offered as distinct stovepipes.

At Georgia Institute of Technology, we followed a similar pattern of offering these two courses separately for a long time at the junior level of the UG program. Several factors came together that forced us to reconsider this format. First of all, the discipline of Computer Science has been expanding and includes topics such as graphics, vision, embedded systems, visualization, and human computer interaction. To allow undergraduate students ample opportunity to explore such emerging topics warranted a rethinking of the "required" or "core" part of the UG CS curriculum. Second. Georgia Tech switched from a quarter to a semester system. This forced us to take a hard look at the curriculum from the point of view of fitting all the required and elective courses within the total credit hours available for the degree. Third, at Georgia Tech, we have a long-standing tradition of involving undergraduates in research. The entry level for systems research was too high for most undergraduates since by the time they took the architecture and OS courses they were ready to graduate.

Given all these factors, we undertook a bold new experiment to offer an integrated architecture-OS semester course "Introduction to Systems and Networks" [1] at the **sophomore level** starting from **Fall 1999**. This course has been a great success since it makes pedagogical sense to the students seeing the system software and hardware issues presented side by side. Further, introducing the students to systems in the sophomore year allows students to get a deeper exposure to systems through additional elective

courses in the junior and senior years, and opens the door for research experience as undergraduates. It is creative thinking like this and other curricular changes that paved the way for innovations in our UG curriculum and the recent birth of the Threads**TM** concept for organizing the UG program at Georgia Tech [2].

2. Overview of the pedagogical style

There is an excitement when you talk to high school students about computers. There is a sense of mystery as to what is "inside the box" that makes the computer do such cool things as play video games with cool graphics, play music be it rap or symphony, sending instant messages to friends, and so on. The purpose behind this course is to take a journey together to unravel the mystery of what is "inside the box." The course takes the viewpoint that what makes the box interesting is not just the hardware but also how the hardware and software work in tandem to make it all happen. Therefore, the path we take in this course is to look at hardware and software together to see how one helps the other to make the box interesting and useful. We call this approach, "unraveling the box": basically look inside the box and understand how to design the key hardware elements (processor, memory, and peripheral controllers) and the OS abstractions needed to manage all the hardware resources inside a box including processor, memory, I/O and disk, multiple processors, and network.

To get a good understanding of what is going on inside the box we have to get a good handle on both the system software and the hardware architecture.

This is the intent of our integrated approach to teaching computer systems.. Correspondingly, the course is divided into five modules:

1. **Processor and software concepts related to processor**
    This unit consists of three sub-parts. The first part deals with HLL constructs and their influence on instruction-set design of the processor. The second part starts out with a simple implementation of the processor. Next, the focus is on processor performance and an efficient implementation of the instruction-set using pipelining techniques. The third part deals with operating systems issues relating to scheduling programs on the processor.

2. **Memory systems and software concepts related to memory systems**

This unit deals with memory management in the operating system and the architectural assists for memory management including memory hierarchies.

3. **I/O subsystems and software concepts related to devices and device controllers**
    This unit deals with input/output issues; program discontinuities due to I/O and other sources are introduced first; the mechanism for interfacing the processor to I/O devices and the corresponding low-level software issues such as device drivers are discussed next, with a special emphasis on disk subsystem. This is followed with a treatment of higher-level storage abstractions such as file systems that may be built on I/O devices such as the disk.

4. **Parallel processors and software issues related to concurrent programming**
    This unit deals with operating systems issues in supporting parallel programming, and the architectural features in multiprocessors for supporting parallel programming.

5. **Network connectivity and software issues related to network protocols**
    This unit deals with the evolution of networking hardware, and the features of the network protocol stack (which is part of the operating system) for dealing with the vagaries of the network.

The hardware and software issues for each of the above five modules are treated concomitantly in the delivery of the course.

The pedagogical style taken in the course is one of "discovery" as opposed to "instruction" or "indoctrination." Further, the presentation of a topic is "top down" in the sense that the student is **first** exposed to the problem we are trying to solve and then initiated into the solution approach. Take for example memory management. We first start with the question "what is memory management?" Once the need for memory management is understood, then we start identifying software techniques for memory management and the corresponding hardware support needed. Thus the discourse almost takes a "story telling" approach to presenting concepts that helps to keep the student interest alive. The discourse includes (through the online material made available to the students) several worked out examples of problems in each of the five modules to elucidate key points.

Further, we maintain a connection between the different modules throughout the course. For example, in the processor module, we use high-level language constructs to illustrate a simple instruction-set design for a processor dubbed LC-2200. Subsequently, we use LC-2200 as the vehicle to develop ideas in processor implementation including pipelining techniques. Similarly, when we introduce architectural assists for memory management or interrupt structures for I/O, we illustrate the enhancements to LC-2200 to facilitate these additional functionalities.

2.1 Why parallelism concepts in a first course on systems?

At the time we designed this course (circa 1998), the answer to this question was not obvious. To this day, most curricula, introduce system level parallelism only in graduate level or at best senior level courses. However, since the early stages of computing, exploiting parallelism has been a quest of computer scientists. As early as the 70's, languages such as *Concurrent Pascal* and *Ada* have proposed features for expressing program-level concurrency. Humans think and do things in parallel all the time. For example, we may be reading a book while listening to some favorite music in the background. Often, we may be having an intense conversation with someone on some important topic, while working on something with our hands, may be fixing a car, or folding our laundry. Given that computers extend the human capacity to compute, it is only natural to provide the opportunity for the human to express concurrency in the tasks that they want the computer to do on their behalf. Sequential programming forces us to express our computing needs in a sequential manner. This is unfortunate since humans think in parallel but end up coding up their thoughts sequentially! Just as research in programming languages dabbled with concurrency from the early days of computing, the operating research community has been concerned with the importance of supporting threads in the operating systems for over a decade.

In the mid 90's, there was a confluence of several trends that helped break the parallelism sound barrier: (1) the popularity of Java programming language with its support for threads; (2) the adoption of *multithreading* in commercial operating systems such as Digital Unix 4.0 and Microsoft Windows NT; and (3) the prevalence of multiple CPUs in desktop computers, not just servers. All of these trends foretold the necessity of introducing parallelism in a fundamental way to the students, which is what we did in our sophomore level integrated systems course.

The intent of the parallelism module is to introduce concepts in developing multithreaded programs, the operating system support needed for these concepts, and the architectural support needed to realize the operating system mechanisms. The important point we want to convey in this module is that the threading concept and the system support for threading are simple and straightforward.

In hindsight, it seems obvious that students should be exposed to system-level parallelism early on since even single-chip processors are starting to have multiple CPU cores in them, and multithreading as a programming concept is being introduced in freshmen programming courses in many institutions.

2.2 Why networking concepts in a first course on systems?

A computer box today does not provide the full functionality to a user without a connection to the Internet. While we take the Internet and network connectivity for granted, it is a revelation to review how we got to this point in the first place. This course module starts with a journey through the evolution of networking from the early days of computing. We then explore network protocols that allow computers to communicate with another. Today, the network protocol stack is an important and integral part of any operating system. This course module gives a glimpse of the functionalities of the protocol stack and how it enables a box to be connected to the outside world and avail of services that we take for granted today (such as e-mail and web browsing).

2.3 Implementing the proposed pedagogical style in a CS curriculum

This course is intended as a first course in systems for students, preferably in the sophomore year of the undergraduate program. This is the way we have used it at Georgia Tech for the past eight years as a required course for all CS majors, where students coming into this course have had a pre-requisite course that deals with logic design and C programming (currently taught using the pedagogical style of Patt and Patel [3]).

Where does such a course fit into the continuum of CS curriculum? Students coming into this course should have a good understanding of data structures, structured programming, and basic logic design.

Most CS programs around the country give this exposure to students in the first two or three semesters of the UG program. Thus this course would ideally fit in the second semester of the sophomore year.

How many credit hours should be devoted to this course? To cover the material in this course it would require about 45 lecture hours (which usually translates to 3 credit hours in a semester system). In addition, since projects make up a significant part of the learning experience, at least 60-90 hours of unsupervised lab time should be dedicated by a student for this course. Thus a course structured around the material being proposed in this paper may account for 4 credit hours in a semester system or 5 credit hours in a quarter system.

What follows this course? This course is intended to give a broad exposure to all the elements of a computer system: architecture, operating system, and networking. Thus this course will serve as an entry point for students interested in seriously pursuing deeper systems topics. For example, in the CS curriculum at Georgia Tech, students specializing in systems go on to take electives including "Advanced Computer Architecture" (using material covered in a textbook such as the one by Hennessy and Patterson [4]), "Advanced Operating Systems" (using material covered in a textbook such as the one by Tanenbaum [5]), and "Computer Networking" (using the material covered in a textbook such as the one by Kurose and Ross [6]). For students not specializing in systems this course also gives the necessary and sufficient exposure to "core" systems issues allowing them to pursue other areas of specialization (such as theory, graphics, and AI).

3. Experience in implementing this pedagogical style

As we mentioned earlier, we have used this style of presenting architecture and OS concepts together to sophomores (CS 2200 [1]) in our UG program from the Fall of 1999. It is a challenging course for students to say the least. We have a significant project component for each of the five modules [7-11]. As is often the case in systems courses, the projects bring home the concepts discussed in the lectures to the students. We allow students to collaborate on the projects but individually interview them for grading purposes to ensure that "learning" has been accomplished. We recruit the top performers in the course as UG teaching assistants for the course in subsequent semesters (which is offered every term) and thus keep the pipeline of knowledgeable students helping their juniors learn the material.

One important advantage of this style is the *reinforcement* of important CS concepts. As is often the case, most students "really get it" the second time. Being exposed to the important systems concepts in the sophomore year help these students to get an in-depth understanding when these concepts are revisited in later advanced courses.

4. Students' reaction to this pedagogical style

As is often the case, when you have a challenging course the reaction will be mixed. But fortunately, there is uniformly positive reaction to the *learning outcome* of this course. The students really appreciate the "demystification" of what is inside a box which is the primary intended learning outcome of this course. In our experience, especially CS majors (as opposed to computer engineering majors) have an aversion to computer hardware. By presenting the hardware and system software concepts together, the students are able to recognize the value of learning computer architecture than when it is taught in a stovepipe format. Our emphasis in the course that designing computer hardware is an algorithmic exercise (for e.g., through project 1 [7]) helps the students overcome the aversion to computer hardware.

Of course, quite a few students complain about the "hardness" of the course. The students need to have a good grounding in C programming to "survive" this course. However, even the ones who have taken the course multiple times to pass it, have told me later on how useful the concepts they learned early on in this course (for e.g., caching) were useful in the workplace (for e.g., web caching as a web designer). I have heard similar comments from students who went as UG interns (to companies such as Intel and Microsoft) that they were instantly useful during their internships because of the systems exposure they obtained as sophomores through this course.

At Georgia Tech, this course (CS 2200) is a required one for all CS majors. I should also add that we have quite a bit of variety in our UG program for students through specialization options in the junior and senior years (e.g., HCI, UI, Graphics, and Robotics). As an instructor, I used to do an informal poll at the beginning of the semester to gauge the student interest for this course. Less than 10% of a class of 100 will say that they are taking this course because they are genuinely interested in systems or that they believe this course will be useful to them in

their chosen area of specialization. However, when I do the same poll at the end of the semester a majority of the students respond that this course has been useful to them despite the fact that they may choose a specialization other than systems. Further, a significantly higher percentage of students respond that they seem motivated to pursue systems as a specialization as a result of this course.

5. Need for courseware, tools, and projects

Teaching such an integrated course is a challenge for the instructors as well. There are several textbooks that are excellent for the stovepipe model of the curriculum (such as Patterson and Hennessy for architecture [12], Silberschatz *et al.* [13] and Tanenbaum [14] for OS), but there was none for such an integrated approach. Therefore, we developed a comprehensive set of notes and slides for the course and used two standard textbooks (Patterson and Hennessy [12], and Silberschatz *et al.* [13]) as background reference material for the students to supplement the course material. All of our courseware (lecture slides, projects, homeworks, example tests) were made are available online for the whole community (for e.g., please see [15] for slides that covers the course material) from the inception of this course.

We taught our course using our courseware with two textbooks serving as background reference material since Fall of 1999 to Fall of 2004. In the Spring of 2005, we turned our courseware into an online textbook [16] since the students continually communicated to us a need for a textbook that matched the style and contents of our course. This online textbook has also been available to the community for the last two years.

To support the pedagogical style of this course, in addition to the online book, we make available a set of online resources. Due to the fact that we have been teaching this course for the last eight years as a required one for all CS majors (3 offerings in each calendar year), there is a significant collection of online resources.

1. We have PowerPoint slides for all the topics covered in the course making preparation and transition (from the stovepipe model) easy.
2. There is a significant project component that dovetails each of the five modules that we enumerated above. We have detailed project descriptions [7-11] of several iterations of these projects along with software modules (such as simulators) for specific aspects of the projects.

3. We have problem sets and solution keys for the different modules of the course.

6. Comparison to other pedagogical styles

In recent times, there are number of proposals to offering complementary material in an integrated fashion. Patt and Patel [3] advocate teaching low level computer hardware (logic design, datapath, and control) concomitantly with C programming. We have found this approach very appealing as a preparation for our first course in systems.

Bryant and O'Hallaron [17] advocate introducing architectural concepts from the point of view of an application programmer. Their approach is intended to help programmers understand the salient features of the "box" from the point of view of developing correct and performance conscious software. Thus the focus of their approach is intentionally on application software and the pitfalls in software development that does not account for system effects. In their textbook embodying this pedagogical style, the authors (Bryant and O'Hallaron) cover a number of esoteric topics not usually found in a single textbook (divided into three parts: program structure and execution; running programs on a system; and interaction and communication between programs). As the authors state in the preface, "If you study and learn the concepts in this book, you will be on your way to becoming the rare 'power programmer' who knows how things work and how to fix them when they break."

There are some aspects of their approach that are complementary to the intended learning outcome of our proposed course. There are some aspects of their approach that are similar to ours. Our course introduces the fundamental principles of computing systems focusing on the inter-relationship between machine hardware and system software. Starry-eyed sophomores are the intended audience, who want to learn how the computer works and not yet ready to create sophisticated application software. Thus we do not deal with issues relating to creating efficient application software. Overall, we believe that the approach proposed by Bryant and O'Hallaron is best applied to senior level undergraduates and thus can be a follow on to our introductory course.

Saltzer and Kaashoek presented MIT's approach to teaching a course in systems at last year's WCAE workshop [18]. Their approach focuses on system building blocks (such as concurrency, communication, fault tolerance, and atomicity) for constructing modular software systems, including operating systems, client/server systems, and databases. This approach is similar in spirit to ours

of presenting related topics in one course rather than in disparate courses, albeit at a higher level, namely, at the level of system software. In our view, such a course would be a nice follow-on to ours.

7. Concluding remarks

At Georgia Tech for the last eight years, we have adopted an integrated approach to teaching a first course in systems that presents concepts in architecture and operating systems in a concomitant fashion. This is a pedagogically sound model that has been very successful and well received by the students at Georgia Tech. We have number of online tools and resources for teaching such a course for use by the community including an online textbook, power-point slides, exercises, and detailed project ideas.

Appendix

This appendix gives a brief description of the projects that could be used in such an integrated course. We will be happy to provide pointers to detailed project descriptions as well as support tools such as simulators.

Processor Design: Students are supplied a data path design that is 90% complete. Students complete the data path to help them become familiar with the design. Then they design the microcode-based control logic (using LogicWorks [19]) for implementing the LC-2200 instruction-set using the data path. This allows the students to get a good understanding of how a data path functions and to appreciate some of the design tradeoffs. The students get actual circuit design experience and functionally test their design using the built-in functional simulator of LogicWorks.

Interrupts & Input/Output: Students take the design from the first project and add circuitry to implement an interrupt system. Then they write (in assembly language) an interrupt handler. The circuit design part of the project is once again implemented and functionally simulated using LogicWorks. In addition, the students are supplied with a processor simulator that they enhance with the interrupt support and use it in concert with the interrupt handler that they write in assembly language. This project not only makes operation of the interrupt system clear but also illustrates fundamental concepts of low-level device input/output.

Virtual Memory Subsystem: Students implement a virtual memory subsystem that operates with a supplied processor simulator. The students get the feel for developing the memory management part of an operating system through this project. The project is implemented in the C programming language.

Multi-Threaded Operating System: Students implement the basic modules of a multi-threaded operating system including CPU and I/O queues on top of a simulator that we supply. They experiment with different processor scheduling policies. The modules are implemented in C using pthreads. The students get experience with parallel programming as well as exposure to different CPU scheduling algorithms.

Reliable Transport Layer: Students implement a simple reliable transport layer on top of a simulated network layer provided to them. Issues that must be dealt with in the transport layer include corrupt packets, missing packets, and out-of-order delivery. This project is also implemented in C using pthreads.

References

[1] CS2200: Introduction to Systems and Networks, http://www.cc.gatech.edu/classes/AY2007/cs2200_spring/
[2] Threads™ An Undergraduate Educational Program of The College of Computing at Georgia Tech, http://www.cc.gatech.edu/content/view/692/144/
[3] Y. N. Patt and S. J. Patel, "Introduction to Computing Systems: from bits & gates to C & beyond," McGraw-Hill.
[4] J. L. Hennessy and D. A. Patterson, "Computer Architecture: A Quantitative Approach," Morgan Kaufmann Publishers.
[5] A. S. Tanenbaum, "Modern Operating Systems," Prentice-Hall.
[6] Kurose and Ross, "Computer Networking: A top down approach featuring the Internet," Addison-Wesley.
[7] CS 2200 Project 1: Processor Design. http://www.cc.gatech.edu/classes/AY2007/cs2200_spring/projects/p1/prj1.html
[8] CS 2200 Project 2: Interrupt & Input/Output. http://www.cc.gatech.edu/classes/AY2007/cs2200_spring/projects/p2/prj2.html
[9] CS 2200 Project 3: Virtual Memory Subsystem. http://www.cc.gatech.edu/classes/AY2007/cs2200_spring/projects/p3/prj3.html
[10] CS 2200 Project 4: Multithreaded Operating System. http://www.cc.gatech.edu/classes/AY2007/cs2200_spring/projects/p4/prj4.html
[11] CS 2200 Project 5: Reliable Transport Protocol. http://www.cc.gatech.edu/classes/AY2007/cs2200_spring/projects/p5/prj5.html
[12] D. A. Patterson and J. L. Hennessy, "Computer Organization & Design: The Hardware/Software Interface," Morgan Kaufmann Publishers.

[13] A. Silberschatz, P. B. Galvin, and G. Gagne, "Operating Systems Concepts," John Wiley & Sons.

[14] A. S. Tanenbaum and A. S. Woodhull, "Operating Systems: Design and Implementation," Prentice-Hall.

[15] On-line courseware for CS 2200, http://www.cc.gatech.edu/classes/AY2007/cs2200_spring/Slides/index.html

[16] Online textbook for CS 2200 http://www.cc.gatech.edu/classes/AY2007/cs2200_spring/textbook/index.html

[17] R. E. Bryant and David O'Hallaron, "Computer Systems: A Programmer's Perspective," Prentice-Hall, 2003

[18] Jerry Saltzer and Frans Kaashoek, "A systems approach to teaching computer systems," WCAE 2006 (held in conjunction with ISCA 2006).

[19] LogicWorks 5 Interactive Circuit Design Software, Pearson Prentice Hall.