AN UNDERGRADUATE SYSTEMS PROGRAMMING LABORATORY

John Lees
Systems Programming Lab
Computer Science Department
University of Michigan-Flint
Flint, MI 48503

It is becoming recognized that computer science is a laboratory discipline, as much as is physics or chemistry. Yet while many people will readily admit the need for laboratories with digital electronics or microprocessor courses, they do not see the need for a computer science department to have a laboratory for courses such as systems programming or operating systems, saying that the facilities of the computer center exist to satisfy the needs of programming courses, particularly on small campuses. I find this attitude impossible to support, especially as equipment costs plummet.

The facilities provided by most campus computer centers are black box in nature--a student shoves a Fortran deck in one side and receives a program listing and a run out the other side. The current trend (to be encouraged!) to substitute terminals for keypunches and card readers still leaves a black box in the middle. But it is exactly this black box, the computer system, which is of interest to systems programming and operating systems students.

If the computer system used by the computer center is one of the popular and well documented ones, e.g., IBM, used as an example in textbooks, then you can give your students a glimpse of what goes on inside the system. The odds are, however, that all the students will be able to do is look without touching. The same holds true of other computer center equipment, such as data communications equipment, which is generally not documented or explained to users in any way.

This is all well and good if the computer center is being used as a production shop, which is indeed the way student programmers use such facilities in beginning courses. It is not good at the systems programming level. Theory is not practice, and simulated systems environments are not real. As is well known in more traditional laboratory sciences, there is undeniable merit in having students perform their own experiments, on their own equipment, making their own mistakes, and experiencing their own successes and failures.

At the University of Michigan-Flint, the computer center is a remote batch station on a system running MTS [1], which is a non-standard operating system with almost no public documentation on internals. This is a severe handicap in teaching systems courses. Through the establishment of our systems programming lab, we are trying to provide the facilities and establish the atmosphere which will encourage our students to take an experimental attitude to solving systems programming problems. Perhaps at this point I should elaborate on what I mean by "experimental attitude" in this context.

The experimental process basically consists of formulating a hypothesis about how things really are and then constructing suitable experiments to prove or disprove the hypothesis (or show that the experiment is worthless, which leads to a new hypothesis). Thus, one builds up a theory of how the world, or a particular computer system, works. An example might go like this: From my reading of the manuals (probably out of date) and the source listings (probably uncommented) of other system programs, I believe that the data communications subsystem works in such and such a manner, and that I may safely modify the terminal handling routine to implement a new function. Constructing an experiment consists of actually modifying the terminal handling routine and using the modified routine in place of the standard one. This sort of thing may crash the system. In fact, it is likely to crash the system. It is this type of programming experimentation which a computer center, by its mission of providing dependable service to an entire university, cannot provide to the computer science department.

I believe that this type of experience is vital to a student who intends to become a systems programmer. There is currently a large gap between textbook presentations of memory management, file systems, data communications, job and process scheduling, etc., and their actual implementations. Many areas, such as device drivers and system utilities, are mentioned only briefly in texts because they are system specific and can be generalized only at a superficial level. All the more reason for students to be able to dig into at least one specific instance and see what is going on. Employers prefer students who have actually done such things over students who have read about doing such things; a point not to be taken lightly considering the current job market. Many more of our graduates go into industry than go on to graduate school.

We are still in the process of building up our systems programming lab, but let me describe our present (in some instances I am actually talking about equipment which is ordered but will not be delivered until summer) set up and how we use it. (See Figure 1)

The basis of the lab is a Burroughs B1800 system with disk, mag tape, line printer, and a string of terminals. Before this system was installed, we had students who had never seen a tape or disk drive, much less used one themselves. Also in the lab is a Heathkit H-11 system with floppy disk and a couple of terminals. That system was assembled by a group of our students. Soon to be included is another H-11 which we intend to turn into a data concentrator on the Burroughs.

The H-11 systems (which are based on the DEC LSI-11 processor) have no commercial software on them. The data concentrator we intend to download from the Burroughs, the other H-11 system is used as a machine language lab for our mini and microcomputer architecture course and as a development system. We are accumulating software for the H-11 systems as students do course projects and directed studies using the H-11. I firmly believe that all students should have some idea of what goes on in all those black boxes their terminals are attached to. A few students even become enthusiastic about learning how to talk to a disk drive at the machine language level. And let's be realistic--structured programming in a high level language is best for 99.9% of programming, but that 0.1% has to be done or the rest of the system will never get off the ground. There have to be a few people who understand how things actually work at the most basic level.

On the Burroughs system, we run a full complement of standard commercial software; a full-sized operating system (MCP), data communications handler (NDL), text editor (CANDE), and all the usual system utilities. Because we have an educational software license, we also have complete source listings for everything, and the compilers for Burroughs' System Development Language (SDL) and Micro Implementation Language (MIL). In other words, in the B1800 we have a complete commercial system which we may poke around in and modify to our hearts' content.

Our systems programming, operating systems, and computer architecture courses are still largely theoretical and still use texts that tend to be overbalanced toward IBM systems, but we now also cover some details of the Burroughs system to compensate for that. The B1800 has a very interesting architecture, based on the concept of emulating various ideal host machines and switching on the fly from the SDL host to the Fortran host to the Pascal host to the Basic host, etc. [2, 3, 4]. Looked at at different levels, the B1800 provides material for examples in several different courses.

Although the systems programming lab is used for minor projects during a regular course, its real use is in directed studies and in semester-long special topics courses such as systems programming projects or data communications, during which students take on major projects of a systems nature; designing, implementing, and documenting them. Projects presently under way include tape utilities, new terminal handlers, the downloader to the H-11, modifications to existing programs, and installation of software from other universities [5]. We have a new requirement in our computer science curriculum of a senior programming project and expect some of them to be done in the systems programming lab.

Setting up a systems programming laboratory in the first place takes a good deal of effort. Funding for a major machine purchase must be found, although prices continue to decrease and most companies have educational discounts and can sometimes be talked into additional grants of equipment. There is often the problem of convincing administrators that the computer science department's facility will not be an unnecessary duplication of the computer center's facilities. A suitable room or rooms, with adequate power and air conditioning, must be found. All of this can easily take a year or more.

Once the laboratory is set up, there are the continuing problems of support and

continuity. There must be an ongoing budget for supplies and maintenance. The problem of continuity, caused by the continual change of the student "staff" of the lab, is best met by giving a faculty member overall responsibility for the general direction and maintenance of the lab and the guiding of individual projects into some kind of usable whole. It is not enough simply to buy a couple of minicomputers and put them in a room with a sign on the door. Of course, this is going to take an appreciable amount of that faculty member's time and so will be another of the costs of maintaining the lab.

Since the student population using the lab will be constantly changing, it is imperative that they waste as little time as possible becoming familiar enough with the facility to do useful work. Documentation is the key to this. Student projects must be well documented and the vendor's documentation of standard programs will certainly have to be supplemented. Even if the vendor's documentation is suitable for its usual customers, it will most certainly not be detailed enough for the purposes of a systems programming laboratory. In our case, students must also be able to quickly learn SDL and MIL, which languages they do not encounter elsewhere in our curriculum.

Our answer to the documentation problem is to put it on-line. We have two usercodes on the Burroughs system for this purpose. EXPLAIN is used for files containing documentation on any and everything. EXAMPLE is used for a library of programming examples illustrating how to use a feature of a particular language (the very thing which everyone moans about being left out of language manuals). Right now we are using the text editor to create and access these files. In the future, we may involve a database course in creating the documentation files in formats which will enable searches by topic. The possibilities are endless...

References

1. Michigan Terminal System. MTS is a system developed on the Ann Arbor campus which runs on a very large Amdahl 470 system. Although the user documentation is good, there is next to no information available on how MTS internals work, and even less of a chance to poke around and play than with a standard IBM operating system.

2. W. T. Wilner, "Burroughs B1700 Memory Utilization," Proceedings of the Fall Joint Computer Conference, 1972, Vol 41, AFIPS.

3. W. T. Wilner, "Design of the Burroughs B1700," FJCC72, AFIPS.

4. E. I. Organick and J. A. Hinds, Interpreting Machines: Architecture and Programming of the B1700/B1800 Series, North-Holland, N.Y., N.Y., 1978.

5. The Computer Science Department, University of Utah, Salt Lake City, Utah, 84112, makes available some software for B1700 and B1800 machines. Programs available include: Pascal, a text editor, more efficient file structure, text, formatting program, etc. As the Utah software was written on different hardware under an earlier MCP, from minor to major revision is needed to install them on our system. Very good experience for our students! Soon we hope to be making some of the software developed at UM-F available to other university B1800 users.

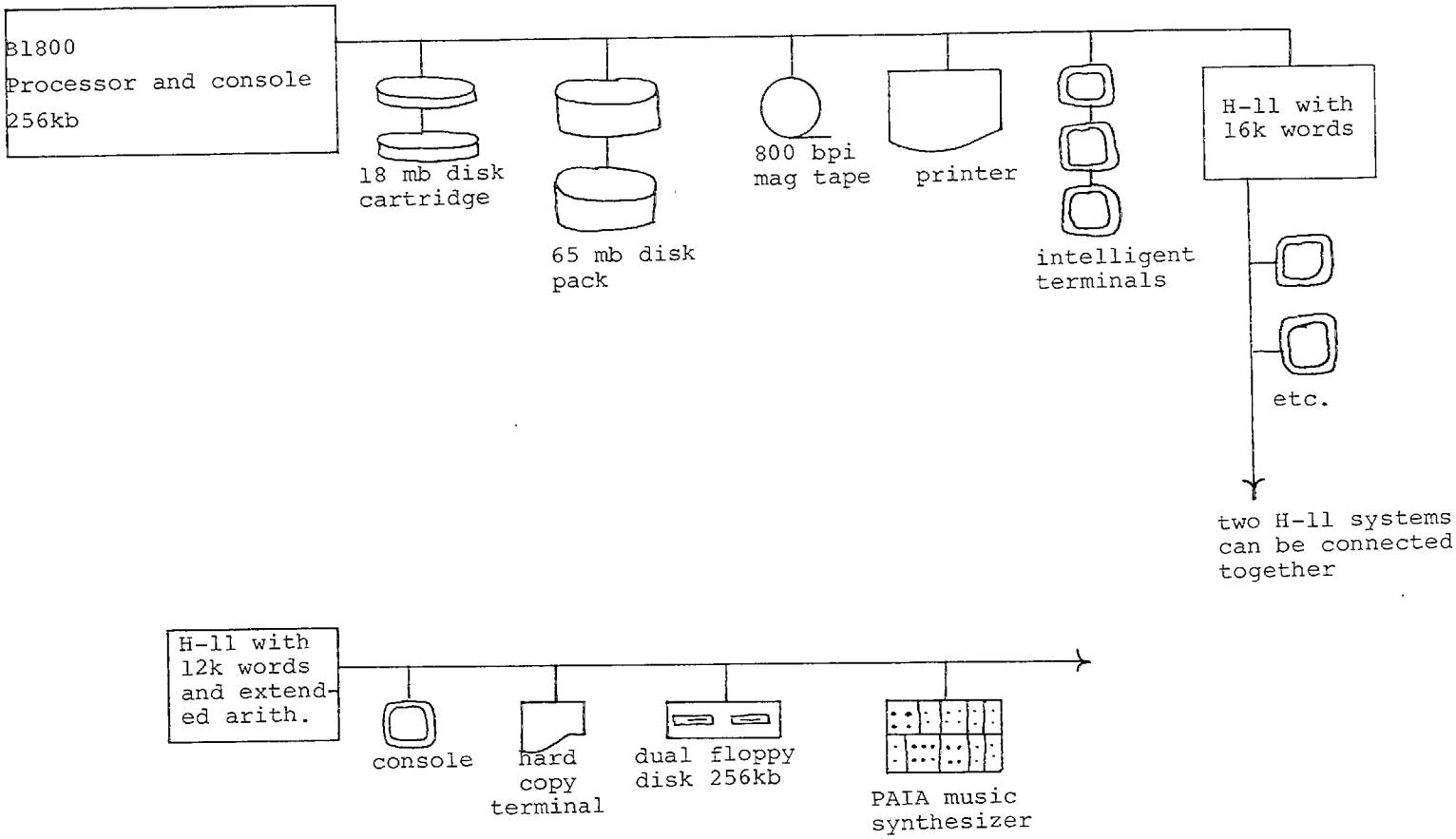
B1800
Processor and console
256kb
18 mb disk cartridge
65 mb disk pack
800 bpi mag tape
printer
intelligent terminals
H-11 with 16k words
etc.
two H-11 systems can be connected together
H-11 with 12k words and extended arith.
console
hard copy terminal
dual floppy disk 256kb
PAIA music synthesizer

FIGURE 1