Abstract This paper describes one well established case-study assignment (RingLAN) used within a first year undergraduate Computer Systems Architecture (CSA) module which is a core component for Computer Science degrees. It requires the students to work in small groups and develop a simple ring LAN for transmitting short text messages. The principal aspects of the protocol (packet size and format, addressing method, packet types, checksum and error handling) are specified, but this still leaves the need for inter-group negotiation to ensure reliable interoperability. Such lab activity introduces students to the basic concepts of end-to-end serial communication, leading into simple packet store-and-forward techniques. While the whole practical exercise is carried out using only PC COM ports it still offers a good challenge for programming skills.

Index Terms - Computer Systems Architecture, student programming case-study, example LAN, TCP/IP

INSTITUTION

The University of the West of England is large by British standards, conferring BSc/BA and postgraduate degrees onto about 5000 students each year. It recruits 60% or more from the locality, the rest arriving from all over Britain. With the cancellation of the national university grants scheme in 1998, the cost of going to university has become a determining factor in the selection of both degree course and university for many British students.

Bristol is a very active educational and research centre with two large universities. The older of the pair, Bristol University, offers the traditional spread of classical and scientific degree courses, while UWE is more commercial and vocational in outlook and mission. This is underscored by the observation that the majority of UWE undergraduate degrees include a full "sandwich" year spent on placement, part training, part work, with either a large employer such as HP, BAE, Orange, Sun Microsystems or a small, specialist organisation. Whichever the student chooses, this intercalated year is a key experience in both education and career development.

COURSE CONTEXT

First year Computer Systems Architecture (CSA) lab sessions give a welcome opportunity to introduce students to a wide set of concepts and practical skills. They need to be fun, relevant to the students’ current experience of computing, and satisfying their precocious expectations. The module also acts to prepare the way for subsequent topics in the second and final years, notably those concerned with networking and systems administration.

The 250 freshman students who take my CSA module have mostly some academic programming experience to their credit. This may derive from one of their three A-level (national exam for 18 year olds) subjects or from a vocational BTEC Diploma obtained at a local college. Before arriving at UWE, computing students, receive a warning letter about the likely problems they will face if they enrol with no prior programming experience. The most common introduction to programming offered to pre-18 year olds used to be Pascal, but has now switched to VB with Access. The deterioration in analytical and language skills following this change has been the cause of many discussions! Our first year computing students also take a full programming module, either based on C or Java.

The CSA module is delivered over two 12 week semesters with one hour lectures and weekly 2 hour lab sessions. The course is based on practical exercises carried out in the computer lab, guided by worksheets, and supported by auxiliary tutors. The course closely follows the author’s textbook [1], making retrospective lecture review quite easy, although the attenuated enthusiasm of students for the printed word is recognised by staff.

CSA is a core subject on four undergraduate degrees: BSc Computer Science, BSc Software Engineering, BSc Computing for Real-time Systems and BSc Computer Systems Engineering. Students taking "softer" IS awards are served by an alternative CSA module which adopts a less technical approach. As may be expected, the students from such diverse programmes have very different expectations, technical prowess and career aspirations. With regard to the latter, some proficiency in HLL programming, systems administration and network management is very commonly sought after by employers.

ROLE OF CSA FOR COMPUTER SCIENCE

The study of computer architecture is best carried out at this academic level by practical exercises. Abstract discussion of
CPU architecture and addressing modes can be a very dry diet, and simulation studies need to be based on concrete experience to be of much enduring value. But by using a good visual debugger, as supplied with MS Visual Studio, with input/output intensive code examples, the students can observe many important features of the hardware actually in use. Revealing s/w dependence on h/w should be a central part of the course, along with the development of an increased personal confidence when dealing with new computer equipment. There are also some side effect spin-offs which can benefit the student. By including suitable practical exercises their analysis and programming skills can be challenged and strengthened. While some basic mathematical principles and practical skills are demanded in order to carry out rough estimating exercises.

Group working does not always come easily to British university students because their school experience has mostly been individual, even competitive, so this has to be explicitly encouraged. Demystify the operating principles of the digital computer might also be an important aim for some students who even today are not technically very experienced. Strangely, modern students can appear less well prepared than earlier generations who were forced to build up their own computers from second-hand parts, and confront many problems on the way.

Another responsibility for all first year modules is to provide a good academic preparation for subsequent modules, and in particular, those involving the exploitation of operating system and networking facilities.

PROBLEMS WITH PROTOCOLS

The advantage of waiting until the second year to introduce all the complexity of TCP/IP and its use for data communications, is that the first year modules offer an opportunity to prepare the way. This can best be done by exposing the students to some of the range of problems that communicating over LANs and WANs can throw up. Thus, discovering problems associated with packet loss, or the policing of unique user IDs, or destination validation, or multiple time-out management and, of course, error control, all stimulate in the student an awareness of complex issues which TCP/IP deals with very effectively. In other words, let them discover the problem for themselves before we show them a solution. Even specialist vocabulary, such as "protocol", "acknowledgement" and "pending", can be a block for the new-comer. The sooner students become acquainted with the multi-tasking demands of communication handling, the sooner they will be gain self-confidence and relevant technical skills.

It is not difficult for first year students to use TCP/IP socket libraries in Java or even C, [2][4]. However such a skill does not equate to a thorough understanding which endures and supports future shifts in technology. Undergraduate teaching must provide more than simply programming recipes. Demystifying the black-box magic of contemporary computing is an important aim which students can appreciate.

RINGLAN ASSIGNMENT

Programming can be seen as dull and difficult by the less imaginative student. But when it involves sending messages, actually communicating with a friend, a real change occurs. Even simple "talk" programs which only send text messages from one PC to a neighbour using the COM ports, can win time and interest from the less committed student. Perhaps this should not surprise us because we have witnessed the same effect on the high street with the explosion of SMS text messaging! The case-study described in this paper builds on this curiosity to know more about data communications, to lay the groundwork for a second year course in systems networking which is based on TCP/IP.
Throughout the first year, students are introduced to analytic and design tools which will underpin their future careers. The ideas and methods of Finite State Machines (State Transition Machines) are used here to help analyse and partition the system functionality. In addition, the students are shown that the complexity of incoming packets is best handled by the use of Decision Tables. In this way, basic structure for their C or Java code is developed through several prototype stages. In several important regards this work prepares the way for the second year courses in Networking, Programming and Software Engineering.

A special cable (FIGURE 2) has been constructed which directs the data round in a closed loop. The hardware handshake lines (CTS/RTS) are also connected in the reverse direction (FIGURE 3). This is important to stop data overrun, although the PC 16550 UARTS are equipped with 16 Byte Tx and Rx buffers which are exploited by the assignment’s demand for the use of only 16 Byte packets.

The fixed length packets (FIGURE 4) contain a Type field to designate six different types of packet, with a block checksum appended to introduce the students to the problems involved if transmission errors occur. This assignment is normally their first attempt to design and implement a concurrent system, albeit with only three tasks. Problems of input/output blocking, critical data and timing mismatches, have to be understood and solved. Simple programming problems involving keyboard buffering have to be recognised and solved in order to obtain single characters i/p without [ENTER]. A common misunderstanding concerns the relative speeds of serial transmission and CPU operations. Thus, although it is safe to write a complete 16Byte data packet to the COM transmit buffer, it is not safe to read incoming packets in 16Byte chunks.

The design method selected for this assignment is Finite State Methods (State Transition Diagrams). Most students are totally unfamiliar with this technique, and certainly do not understand how to implement a FSD in C or Java. A few of them will already have been introduced to Finite State Table dispatching in their I/O Programming module, but the majority prefer to use the Switch/Case style of implementation. To provide the appropriate level of support at the start of a new assignment can be a bit of a dilemma. Students can feel totally overwhelmed by such a complex specification, and inadequate to deal with the new aspects of programming that are clearly demanded. In this case-study, we now give them all the hardware initialisation code which and functioning of the code. This allows tutors to be confident that the work is truly "home grown" and also gives students a welcome opportunity to show-off their achievements. The marking scheme is closely specified and includes bonus marks for good performance, such as getting the code to work correctly with "foreign" versions of the software. Testing the compatibility with others’ programs can reveal interesting errors. Transmitting data packets backwards, or exchanging the open and close braces, are fairly common errors. The possibility of simultaneous logins occurring with the same identifier has been discussed and a solution proposed involving a timestamp.
 prevents blocking on the keyboard or COM port, and complete FSD and code solutions to the keyboard task, leaving the students to complete the remaining 66% of the system. This "teaching by example" seems to achieve better results than spending a lecture describing the problems in abstract language. Seeing an example FSM implementation actually run on your machine is possibly a richer source of information than several OHP slides. It also forces the tutors to polish up their prototype code to avoid too much embarrassment!

An unexpected revelation for some students is to discover that software which imposes a user login may also offer substantial functionality at other times. Logging out does not necessarily mean shutting down the computer! In this case-study, the programs are expected to pass through packets without a local user being logged in, thus maintaining the continuity of the RingLAN for others to use.

```
struct pt {
    DWORD present; //0=!present 1=present -1=maybe
    DWORD retx; //re-tx count
    char packet[16]; //txed pkt waiting ACK
} pendtable[26];
```

**FIGURE 5**

**PENDING TABLE**

The need to implement an ACK/NAK handshake for error control also requires timeout retransmission to deal with the "no reply" situation. This demands storage for transmitted but unacknowledged packets, and so the Pending Table (FIGURE 5) is a natural progression along the path of increasing complexity.

A local directory of currently active users can be included within the Pending Table, helping with this aspect of the requirement.

Time-out handling in software can be a real challenge for the less experienced programmer. When there are several activities in progress, each waiting for a response, the best way to handle their elapsed time counters, and making a decision when a countdown occurs, can be uncertain.

**CODE STRUCTURE FOR CONCURRENCY**

When dealing with such a problem, multi-threading would probably be the method of choice for experienced programmers using Java. Dedicating a thread to the keyboard, a second to the incoming serial data, and a third to the outgoing serial data, offers a nicely partitioned design leading to a clean implementation. Issues of critical data and mutual exclusion can readily be handled by language facilities. But, taking into account that the students have only just started to use Java, and a good deal of their attention is being taken up by the need to come to terms with the Object Oriented approach to system partitioning, it was decided to recommend a simple cyclic loop (FIGURE 6). In this, the 3 concurrent tasks will be represented by 3 separate FSDs with independent State Indices. This plan has the added benefit of providing tutors with the option of presenting students with fully worked exemplars for part of the project, leaving them to complete the rest.

```
initialise

Keyboard handler

In-coming data handler

Out-going packet handler

FIGURE 6

TASK LOOPING
```
Each of the tasks in the above flowchart (FIGURE 6) has a FSD representation given below (FIGURES 7,8,9). The communication between the three tasks is accomplished through the pending table which holds packets waiting to be transmitted and the currently valid destination IDs.

**FIGURE 7**
FSD FOR THE KEYBOARD TASK

**FIGURE 8**
FSD FOR THE INCOMING PACKET TASK

**FIGURE 9**
FSD FOR THE OUTGOING PACKET TASK
The FSD for the receiver task can be quite complex if all the packet decoding is included. However this part is not really a true FSM, it only expresses the various alternatives in a decision tree format. A decision table better helps to identify all the flavours of incoming packets which have to be catered for, and it is possibly better than resorting immediately to a FSD. This also means that the decoding of the incoming packets might more easily be carried out by a single Activity, using an IF/ELSE-SWITCH/CASE tree, rather than through a series of intermediary states with transition options.

**FUTURE EXPANSION**

Any worthwhile programming assignment lives on for many years with refinements and added features updating the schema for each new cohort. If the problem has enough "meat" to inspire curiosity and commitment amongst the students, it is a good resource that should be developed. In this case there are several plans for the future under discussion within the module team.

1. Addition of gateway facilities to provide for the interlinking of several RingLANs.

2. Impose dual addressing, personal/machine, requiring a mapping between the two domains.

3. Require a windowing user interface, through the use of Java or C/Tk. If Linux was adopted as the target operating system, it would also be possible to use the curses library.

4. Segment long messages to fit into a series of small packets.

5. Accept variable length packets.

### TABLE 1

<table>
<thead>
<tr>
<th>Des</th>
<th>Src</th>
<th>Pkt Type</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>you</td>
<td>L</td>
<td>my login id is OK, del pkt</td>
<td></td>
</tr>
<tr>
<td>me</td>
<td>R</td>
<td>illegal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>test msg, del pkt, retn ACK</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>cancel pending entry, del pkt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>re-tx pending table entry</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>logout now, del myaddr &amp; pdng tbl</td>
<td></td>
</tr>
<tr>
<td>you</td>
<td>L</td>
<td>illegal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>reply to my login, del pkt, update pdng tbl</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>message, ret ACK, del pkt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>del pkt from pdng tbl</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>re-tx pkt from pdng tbl, dec count</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>illegal</td>
<td></td>
</tr>
<tr>
<td>you</td>
<td>L</td>
<td>illegal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>where has he gone? del pkt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>re-tx pkt, update pdng tbl</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>del packet, cancel pdng entry</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>del packet, cancel pdng entry</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>illegal</td>
<td></td>
</tr>
<tr>
<td>you</td>
<td>L</td>
<td>re-tx pkt, update pdng tbl, ret R pkt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>illegal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>re-tx packet</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>re-tx packet</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>re-tx packet</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>logout, re-tx pkt, amend pdng tbl</td>
<td></td>
</tr>
</tbody>
</table>

**REFERENCES**