Chapter 6

Thoughts on the future development of cirt

6.1. Dividing cirt into two processes

6.1.1. The problem of size

At present **cirt** is about as large as it can get without running out of core space, so before more facilities can be added some way round this problem will have to be found. There are three obvious approaches to the problem:

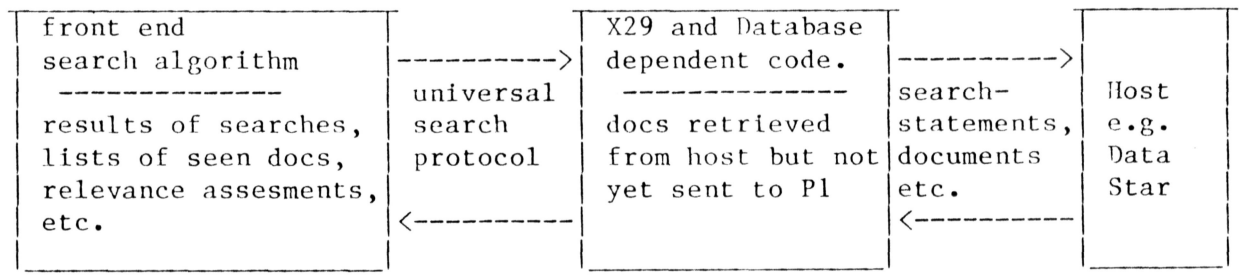
1 Upgrade the hardware to a processor which supports separate I/D spaces (11-24?).

2 Install a version of UNIX which supports overlaying. There are a number of these available: e.g. the DEC release of UNIX (which they have in the Computer Science Department at TCU), the Berkeley release and probably others. Whatever version was used would have to be compatible with the network software. This question has not been investigated fully.

3 Split the program into two processes. This is quite an attractive option for a number of reasons and is discussed in detail below.

6.1.2. Splitting the process

A natural way to divide the program is as follows



Essentialy Process 1 would contain the code now in cirt.c and search.c along with any improved front-end code, additional search algorithms etc. Process 2 would contain the code in x29.c and lex.yy.c. The code in print.c would have to be rewritten and the new code would be split between the two processes.

The idea is that Process 2 should contain all the Host and Database dependent code and that the two processes should talk to each other using some sort of universal search protocol (USP). Process 2 then converts between the USP and the Search statements and document listings

required by and sent from the Host. How universal the USP can really be, is debatable. Probably the best thing to do is to make it as universal as possible consistent with easy compatability with Data-Star.

This setup has a number of advantages:

(i) It should solve the space problem, at least for a while.

(ii) The viewing of documents will be speeded up because it will be possible for Process 2 to be retrieving and storing documents while Process 1 is waiting to be told the verdict on the previous document.

(iii) The data base dependent code is all in one process and it should (if the USP is adequate) be possible to convert to another Host by just rewriting Process 2.

To set against that there may be a slight slowing down of the search process.

6.1.3. Communication between the processes

This should be done with pipes and possibly intercepted kills. Briefly the way it is done is as follows.

Process 1 is executed in the usual way from the **shell**. Process 1 then calls **pipe** twice to set up a two way communication channel. It then calls **fork** and if **fork** returns zero it executes Process 2 using **execl** and passes the file descriptors returned by **pipe** (one read and one write) as arguments. The integer file descriptors should be converted to ascii strings before passing them and then converted back to integers by Process 2.

It will probably prove to be necessary for Process 1 to be able to interrupt Process 2 while Process 2 is waiting for material from the Host. This would have to be done using **signal** and **kill**. A description of **pipe**, **fork**, **execl**, **signal** and **kill** can be found in chapter 2 of the UNIX manual.

6.1.4. Flow of Control in P2

Normaly P2 would run through a cycle as follows

1. Get next USP command from P1
2. Hold dialog with Host.
3. Send reply to P1
4. Goto 1.

Sometimes there would be no stage 2 but probably every command should get a reply.

Occasionally it will be necessary to force P2 to stop waiting for input from the Host and to reply to a command. This could be done by using an intercepted **kill** but this sort of interrupt would be kept for emergencies.

6.1.5. The Universal Search Protocol

A suggested outline for the USP is described in Chapter 7.

6.1.6. Notes

1 In addition to providing different versions of P2 for different Hosts it would be possible to provide a P2 which actualy did searches on its own database.

6.2. Other Additions

There are a number of facilities we would have liked to have included in **cirt** but either we didn't have time or there wasn't room. These are listed below.

6.2.1. Save and Restore

It would be useful to be able to save a search so that it can be restored and completed at a later date. This could be used in conjunction with the Data-Star **..Off Continue** command. There should be no great difficulties but there are a number of different data structures which would have to be saved. These include the search tree, the query, the list of seen documents and a number of single parameters.

6.2.2. Term deletion and improved term addition

It might be useful to be able to delete query terms which have been used in a search but this would involve extensive modification of the search tree. This would be made simpler if instead of storing the search as a number it could be stored as a string consisting of a boolean combination of search numbers.

Another possibility which should be looked at but might prove unworkable is the addition of new terms within the search tree. This might prove more efficient for terms with low frequencies and high weights.

A third possibility, mentioned in Chapter 3, is the addition of new terms by ORing with existing terms in the search tree.

6.2.3. Other search algorithms

At some time the Jamieson algorithm ought to be implemented on a proper data base, if only for comparison. We think that it will remain impossible to use the Morrissey algorithm, at least in the near future, but people may well think of other promising algorithms.

6.2.4. Automatic query expansion

With the current search algorithm the possibilities for this are rather limited but something should be done.

6.2.5. Other Databases on Data-Star

The main problem at the moment is that **cirt** expects a numerical field in the AN paragraph. This applies to Medline but not to some other databases (e.g. Inspec). To get round this either the document identifier could be stored as a string or some sort of hash coding could be used. There may also be other problems with as yet untried data bases.

6.2.6. Checking the X25 connection is up

The PAD program checks the state of the X25 connection before it issues its first prompt and **cirt** should probably do the same. There is a routine called **l2trace** in the X25 library which does it. Its source code can be inspected in /x25/rel2/usr/src/netio.c. The source of the PAD program would also help.

6.2.7. A logout command

This would send a ..o [cont] to Data-Star and reset the data structures.

6.2.8. A Help Command

This clearly needs doing, as part of a more 'user-friendly' front-end (see below).

6.2.9. A Limit Command

This is a command to enter a query which will be AND'd with every query sent to the Host. This effectively restricts the data-base to those documents which are retrieved by the limit query. Unfortunately it is intrinsic to our search algorithm that the limiting has to be done before the search is started, whereas the Data-Star Limit command which would be used by our limit command is called after the query is done. This makes the implementation of a limit command a bit fiddly.

6.3. Making cirt more user-friendly

Although fairly straightforward to use and having a simple command language, **cirt** could certainly be made more user-friendly in a number of ways. **Cirt**'s own messages to the searcher tend to be terse, and could perhaps do with some expansion. Certainly a Help command should be implemented. The searcher should perhaps have more control over the way the documents are displayed on the screen - e.g. an automatic title display, with the abstract available on request.

The extent to which **cirt** is adapted in this direction must depend on the kind of user envisaged, with a considerable difference between

trained intermediaries and end-users. Since the immediate plan is for use by intermediaries (see Chapter 8), it is considered that the changes required are not very extensive at this stage. Furthermore, the best way to determine an appropriate set of changes to **cirt** would be to allow an experienced intermediary to use the system over a period of time.

6.4. Bugs

The treatment of truncated search terms which match more than 100 actual search terms could be considered to be a bug. It is certainly unsatisfactory.

If a query gets garbled on the way out (this should never happen but it seems to sometimes), then cirt doesn't notice.

The BREAK key doesn't always work properly when in TALK mode.