EUUG

papers presented at the

European UNIX® Systems User Group Spring Meeting

1 - 3 April 1985
Paris, France

E U U G

European UNIX+ Systems User Group

Proceedings EUUG Conference

Paris Spring 1985

+UNIX is a trademark of AT&T Bell Laboratories
*Ada is a trademark of the Ada Joint Programming Office
=TeX is a trademark of the American Mathematical Society

Computer Aides to Rewriting and Copy Editing of English Text

Lorinda Cherry

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

Writing is generally thought of as a three stage process; planning, producing a first draft, and revising and editing that draft. The UNIX† WRITER'S WORKBENCH†† software is a set of programs to help writers of English with the last stage, revising and editing. It includes programs that analyze the writing style of the text, provide the writer with other views of the text that may identify the parts that need revision, and help in the copy editing process. In this paper I will describe the programs, discuss how they have been used, and how they might be extended to help writers who are writing in English as a second language.

February 12, 1985

† UNIX is a Trademark of AT&T Bell Laboratories.
††WRITER'S WORKBENCH is a Trademark of AT&T Technologies.

Computer Aides to Rewriting and Copy Editing of English Text

Lorinda Cherry

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

Introduction

One of the earliest uses of the UNIX† operating system was for document preparation. The document preparation tools have evolved into a very sophisticated suite of programs to typeset complicated equations[1], tables[2], and graphics [3, 4]. Along with the programs for making the page pretty came other tools for checking the content of the document. Among the first was `typo` [5] for finding typographical errors, which was quickly followed by `spell` [6] for finding spelling errors. The WRITER'S WORKBENCH†† software[7, 8, 9] represents the next step in this evolution. It includes programs to analyze the text at the word and sentence level as well as programs to do some of the routine work of copy editing. Section 1 of this paper will describe the higher level programs, `wwb`, `proofr` and `prose`. Section 2 will discuss the `parts` and `style` programs on which many of the other programs are built. Section 3 discusses the use of WRITER'S WORKBENCH in freshman composition courses and the results of a study on `style`'s and `parts`' performance on student text. Section 4 speculates on how these programs might help writers who are writing in English as their second language.

`wwb = proofr + prose`

The WRITER'S WORKBENCH software is organized hierarchically as shown in Appendix 1. At the top level the `wwb` program runs `proofr`, which is the copy editing part of the system, and `prose`, which is the stylistic analysis part. From a technical standpoint `proofr` is probably the least interesting but the most useful part of the WRITER'S WORKBENCH software. A sample of `proofr` output is shown in Appendix 2. `proofr` is a shell script that runs five separate programs. The first, `spellwwb`, is an augmented version of the UNIX system `spell` program that, in addition to looking in the usual `spell` dictionary, also looks in the user's private dictionary. Words found in the user's dictionary, usually proper names and acronyms, are not reported as errors.

Next `proofr` runs the punctuation checker, `punct`. `punct` makes sure that all sentences in the text begin with a capital letter, that quote marks and parenthesis balance, and that consecutive punctuation characters occur in the correct order. This last check refers to rules like periods and commas always go inside quote marks unless the quoted string is a single character. This rule, incidentally, is an American English standard, not a British one. These rules are difficult to remember and even harder to find by eye.

Perhaps the most useful of `proofr`'s programs is `diction`. `diction` prints all sentences that contain frequently misused or wordy phrases found in its dictionary of approximately 450 phrases. Many of the phrases in the dictionary are correct in some context but indicate wordiness in others. When run as part of `proofr`, the program also runs the `suggest` program, which proposes substitutions for the phrases. Like `spellwwb`, the user may also have a private dictionary of phrases for the program to find in addition to the default dictionary or phrases to suppress from the default dictionary. For example, *terminate* is often a long or pompous word for *end*. However, in writing about software *terminate* is an accepted and specific word. Users writing

† UNIX is a Trademark of AT&T Bell Laboratories.
††WRITER'S WORKBENCH is a Trademark of AT&T Technologies.

program documentation may want to add *terminate* to their private dictionary as a phrase from the default dictionary not to be flagged.

The final program run by `proofr` is `gram`, a rudimentary grammar checker. Currently `gram` only checks for two kinds or grammatical errors; split infinitives and misused indefinite articles as in "an man" or "a honor."

After running `proofr`, `wwb` runs `prose`, which compares the values from a table of statistics computed about the text with a set of standards and produces a two or three page description of the text in terms of several stylistic features. The stylistic features on which `prose` reports are those that experts on writing agree lead to better prose and many are supported by psychological research. They include readability indices, variation in sentence type, sentence length, and sentence beginning, the use of active rather than passive verbs, the use of verbs rather than nominalizations and limiting the use of expletives. `prose` composes the report from stored text, selecting sentences and explanations based on how the user's text statistics compare to the standards. An example of one page of `prose` output is in Appendix 3. In addition to reporting on the style of the text `prose` also suggests other programs that the user might run to look at it. Experienced users can run `prose` with a flag to get a short form of the output rather than the long report.

`parts` and `style`

The program that is the basis of much of the WRITER'S WORKBENCH software is the `parts` program for finding parts of speech or word classes in running English text. `parts` began as the front end to a synthetic speech system. In such a system the parts of speech of the words are needed to add inflection to the words in the sentences as well as to decide the stressed syllable in words like con-trast' (the verb) and con'trast (the noun). Several design requirements of a speech synthesis program made `parts` an ideal tool for text analysis. They include:

speed – the system couldn't spend all of its time parsing
size – the part of speech program couldn't be too big
vocabulary – the vocabulary it would see was unknown; it had to work on any text
performance – it didn't have to be 100% correct but it had to produce some answer

The speed requirement makes it feasible to run `parts` on long papers. The size, vocabulary and performance requirement dictated that the program not be dictionary based, which led to a system that could be run on text on any subject. Although a real parser would be more powerful for text analysis, parsers in today's technology have vocabulary as well as speed limitations. `parts` runs at about 400 words/second on a VAX 750 with an accuracy of about 95%.

`parts` is really a four program pipeline. The first program, `deroff`, extracts the running text from the document. To do this it removes the formatting commands, headings, tables, equations, graphics and other non-sentence features. The second program splits the text into words, punctuation and sentences and looks each word up in a small dictionary of about 450 function words and irregular verbs. Function words are prepositions, conjunctions, articles and auxiliary verbs. The third program checks all words not yet assigned a word class for certain suffixes. `parts` uses 51 suffixes, some of which denote a unique word class; others denote a partial or dual word class. Of importance here is that all English suffixes that end in the letter "s" are checked. Any word ending in an "s" that does not match one of the suffixes must either be a plural noun or a singular verb and subject-verb agreement can be used by the last program to determine which. Each suffix also has an exception dictionary that is checked before a class is assigned.

The last program does most of the work. It uses heuristics of English word order and the assignments or partial assignments made by the other programs to assign parts of speech to all of the words in the text. A detailed description of these heuristics can be found in [10]. Here I'll

simply step through an example sentence, found in Figure 1.

Figure 1
Example of `parts` Stages

The	UNIX	operating	system	exists	for	many	different	kinds	of	hardware.
art					prep_adv	pron_adj			prep	
art	unk	ing	unk	nv_pl	prep_adv	pron_adj	unk	nv_pl	prep	unk.
art	adj	adj	noun	verb	prep	adj	adj	noun	prep	noun.

The first line is the input. The second line is the output from the dictionary lookup program. Only *the*, *for*, *many*, and *of* were in the dictionary. The third line is the output of the suffix checking program. It found the *ing* on *operating* and the *s* on *exists* and *kinds*. The class *unk* was assigned to all of the other words with no suffix match, implying that they do **not** end in *s*. The third line is the input to the last program and its output in the last line of the figure. This program first checks the sentence or clause for a verb or auxiliary verb. Then it starts at the beginning of the sentence to assign parts of speech. Here the article introduces a noun phrase and subject-verb agreement is used to label *system* as a noun and *exists* as a verb. Other rules of word order are used to assign parts of speech to the rest of the words.

The `style` program[11] uses the parts of speech found by `parts` to produce a table of stylistic measures of the text like that shown in Figure 2.

Draft

readability grades:

(Kincaid) 9.6 (auto) 10.8 (Coleman-Liau) 10.8 (Flesch) 8.9 (61.3)

sentence info:

no. sent 110 no. wds 2143
av sent leng 19.5 av word leng 4.78
no. questions 2 no. imperatives 0
no. content wds 1268 59.2% av leng 6.18
short sent (<14) 28% (31) long sent (>29) 11% (12)
longest sent 51 wds at sent 81; shortest sent 2 wds at sent 48

sentence types:

simple 55% (60) complex 27% (30)
compound 8% (9) compound-complex 10% (11)

word usage:

verb types as % of total verbs
tobe 37% (77) aux 15% (32) inf 17% (35)
passives as % of non-inf verbs 15% (27)
types as % of total
prep 13.0% (279) conj 3.9% (83) adv 4.0% (85)
noun 29.7% (636) adj 17.7% (380) pron 3.1% (67)
nominalizations 2 % (50)

sentence beginnings:

subject opener: noun (21) pron (13) pos (0) adj (20) art (26) tot 73%
prep 13% (14) adv 10% (11)
verb 1% (1) sub_conj 3% (3) conj 0% (0)
expletives 1% (1)

Figure 2

This is the table used by `prose` to compare the text with the standards. Other things can be

learned about the text besides the measures used by `prose` already discussed in the first section of this paper. If, for example, the number of dependent clauses, roughly the sum of the number of complex and compound-complex sentences from the "sentence types" section of the table, is close to the number of sentences that begin with subordinate conjunctions (sub_conj), then the author writes almost exclusively with leading dependent clauses. Since variety is so important in writing, any strong pattern of this type may indicate monotonous or tedious prose. Another common problem, particularly in technical writing, is the overuse of adjective qualifiers. A noun-adjective ratio near unity often indicates this problem. Other metrics from the table may also be important; their identification and importance is an open research question.

WRITER'S WORKBENCH Software in Composition Instruction

In the school year 1981-1982 Bell Labs and Colorado State University began a research exchange to test the WRITER'S WORKBENCH system for teaching composition[12, 13, 14, 15]. During the first semester randomly selected students in freshman composition were given access to a computer lab and the `spell`, `style`, `diction`, and `suggest` programs to write their essays. Pre- and post-tests of students in the experimental group showed significant improvements in editing skills over the control group. Encouraged by these results the faculty at Colorado State expanded their program to use the other WRITER'S WORKBENCH programs. They packaged several of the non-standard WRITER'S WORKBENCH programs to run automatically on the student text and added a data base of commonly confused word pairs, like affect-effect, as another file for `diction` to use. Students use their lab time to type their papers and run one program to get 7 or 8 pages of comments on their papers. After reviewing the comments, they come back to the lab, make corrections and get a final draft of their papers to turn in. Currently, more than 4000 students in courses including freshman composition, advanced writing, business writing, and composition analysis are using the computer lab and the WRITER'S WORKBENCH programs. Professors are delighted at being able to spend more time teaching writing and less time correcting surface errors. They also report that the students' papers are better.

One question remained open about the use of the WRITER'S WORKBENCH software on student text: did the performance of `parts` and therefore, `style` degrade when run on student text? `parts` was developed experimentally on grammatically correct English text. Much of student text is far from grammatically correct and often is much less formal in style than the text used to develop `parts`. An experiment[16] using 47 student essays that were assigned parts of speech by two graduate students and the program revealed a 4% degradation in `parts`' performance from the original study on grammatically correct text. This 4%, however, only led to a 1–2% difference in the metrics used by `style` and did not change the suggestions and comments made by `prose` about the text.

How might WRITER'S WORKBENCH help writers' of English as a second language?

Writers' of English as a second language have many of the same problems of wordy and awkward phrasing and lack of variation as native English writers. To the extent that the problems are the same, the programs should prove useful. However, further research on the writing habits of speakers of different languages needs to be done. No language translates directly, word-for-word into English. Phrase patterns differ and idioms exist that may be detectable with `parts` and `style`. At the very least phrase dictionaries for `diction` could be developed that could make a big difference in the naturalness of writing by non-native English writers.

Conclusion

This paper has been a brief introduction to the WRITER'S WORKBENCH software and the programs on which it is based. It is a natural extension of the UNIX document preparation tools and has proved to be useful as a teaching tool. With further research it should also be useful for writers of English as a second language.

References

1. Kernighan, B. W., Cherry, L. L., "A System for Typesetting Mathematics," *Commun. of the ACM*, **18** (3), pp.151-157, 1975.
2. Lesk, M.E., "TBL – A Program to Format Tables," *UNIX Programmer's Manual*, Bell Laboratories, Murray Hill, N.J. Section 10 (January, 1979).
3. Van Wyk, C.J., "A High-Level Language for Specifying Pictures," *ACM Trans. on Graphics* **1** (2)**pp.**
4. Kernighan, B. W., "Pic – A Language for Typesetting Graphics," *Software Practice & Experience* **12,** pp. 1-20, January, 1982.
5. Morris, R. and Cherry, L.L., "Computer Detection of Typographical Errors," *IEEE Trans. on Professional Comm.*, **PC-18,** pp. 54-64, March 1975.
6. McIlroy, M.D., "Development of a Spelling List," *IEEE Trans. on Comm.* **30,** pp. 91-99, 1982.
7. Macdonald, N. H., Keenan, S. A., Gingrich, P. S., Fox, M. L., Frase, L. T., and Collymore, J. C., "Writer's Workbench: Computer Aids for Writing, Release 2.0," Bell Laboratories, 1981.
8. Cherry, L. L., "Writing Tools," *IEEE Trans. on Comm.* **30,** pp. 100-105, 1982.
9. Macdonald, N. H., Frase, L. T., Gingrich, P. S., and Keenan, S. A., "Writer's Workbench: Computer Aids Text Analysis," *IEEE Trans. on Comm.* **30,** pp. 105-110, 1982.
10. Cherry, L. L., "PARTS – A System for Assigning Word Classes to English Text," *Computing Science Technical Report #81*, 1980, Bell Laboratories, Murray Hill, NJ 07974.
11. Cherry, L. L. and Vesterman, W., "Writing tools – The STYLE and Diction Programs," *Computing Science Technical Report #91*, 1981, Bell Laboratories, Murray Hill, NJ 07974.
12. Kiefer, Kathleen E. and Smith, Charles R., "Textual Analysis with Computers: Test of Bell Laboratories' Computer Software," *Research in the Teaching of English*, Vol. **17**, 1983, pp201-14.
13. Smith, Charles R. and Kiefer, Kathleen E., "Using the Writer's Workbench Programs at Colorada State University," *6th International Conference on Computers and the Humanities*, Editors, Sarah K. Burton and Douglass B. Shorts, Rockville, Md., Computing Science Press, 1983, pp 672-84.
14. Smith, C. R. and Kiefer, K. E., "Writer's Workbench – Computers and Writing Instruction," *Proceedings of the Future of Literacy Conference*, April 1–3, 1982, Baltimore, Md.
15. Kiefer, K. E. and Smith, C. R, "Improving Students' Revising and Editing: The Writer's Workbench System," *The Computer in Composition Instruction*, ed. W. Wresch, National Council of Teachers of English, (1984).
16. Cherry, L. L., "A Study of Parts' and Style's Performance on Real Student Text," Internal memorandum, Bell Laboratories (November 1984).

Appendix 1

COMMAND-FUNCTION TABLE

Commands

```
acro file.................finds acronyms
chunk file................segments at phrase boundaries
match stylefile1 2  N.....collates statistics from different texts
org file..................shows text structure
rewrite file..............first draft of document with probs. highlighted
sexist file...............finds sexist phrases and suggests changes
spelltell pattern.........prints commonly misspelled words containing pattern
style file file1..........summarizes stylistic features
syl -n file...............prints words of n syllables or longer
topic file................summarizes content
   parts file.............assigns grammatical parts of speech
wwb file..................runs proofreading and sytlistic analysis
   proofr file............proofreading comments
      dictplus file.......finds awkward phrases, and suggests changes
       diction file......finds awkward phrases
       suggest phrase....suggests substitutions for awkward phrases
      double file.........detects repeated typings of words
      punct file..........corrects punctuation
      spellwwb file.......checks spelling, using spelldict
      gram file........finds split infinitives and incorrect indefinite articles
       parts file.......assigns grammatical parts of speech
   prose file............extended editorial comments
      style file..........summarizes stylistic features
       parts file.......assigns grammatical parts of speech
wwbmail...................sends mail to WWB Development group
```

Explanations

```
punctrules................explains punctuation rules
splitrules................explains split infinitives
worduse word..............a glossary of words often confused in writing
wwbhelp word..............gets information about word (e.g. spell)
wwbinfo...................prints a copy of this table
wwbstand..................prints standards used by prose to evaluate documents
```

Environmental Tailoring

```
dictadd...................adds phrases to ddict dictionary
spelladd..................adds words to spelldict dictionary
mkstand...................builds standards for prose from user documents
```

User Specified Dictionaries

```
ddict.....................personal list of awkward phrases
spelldict.................personal list of spellings
```

Note. Indented commands are automatically run by the less indented commands that immediately precede them.

Appendix 2
Example Text and Proofr Output

Example Text

```
The large number of probrams that comprise the UNIX
operating system make it it very useful.
utilizing ``sed'', we are able to always edit very large files.
A other powerful program is ``awk''.
```

Proofr Output

```
Example

****************************  SPELLING  ****************************

Possible spelling errors in Example are:

probrams                sed

If any of these words are spelled correctly, later type
                spelladd word1 word2 ... wordn
to have them added to your spelldict file.


****************************  PUNCTUATION  ****************************

For file Example:

The program next prints any sentence that it thinks is
incorrectly punctuated and follows it by its correction.

lines 3,4
OLD: utilizing ``sed'', we are able to always edit very large files.
OLD: A other powerful program is ``awk''.
NEW: Utilizing ``sed,'' we are able to always edit very large files.
NEW: A other powerful program is ``awk.''

For more information about punctuation rules, type:
                            punctrules


****************************  DOUBLE WORDS  ****************************

For file Example:

it twice line 2 file Example
operating system make it it very useful.

****************************  WORD CHOICE  ****************************

Sentences with possibly wordy or misused phrases are listed next,
followed by suggested revisions.

beginning line 1 Example
The large *[ number of ]* probrams that *[ comprise ]* the UNIX
operating system make it it *[ very ]* useful.
```

```
beginning line 3 Example
*[ utilizing]* ``sed'', we are able to always edit
*[ very ]* large files.

file Example: number of lines 4 number of phrases found 5

Please wait for the substitution phrases.


------------------   Table of Substitutions   --------------------

PHRASE                        SUBSTITUTION

comprise: use "include" for " comprise"
number of: use "enough" for " sufficient number of"
number of: use "many" for " a large number of"
number of: use "often" for " in a considerable number of cases"
number of: use "several, many, some" for " a number of"
number of: use "some" for " in a number of cases"
number of: use "usually" for " except in a small number of cases"
utilizing: use "using" for " utilizing"
very: use " " for " very"
very: use "complete" for " very complete"
very: use "doubtless, no doubt" for " there is very little doubt that"
very: use "in a sense or OMIT" for " in a very real sense"
very: use "unimportant" for " of very minor importance"
very: use "unique, uncommon" for " very unique"

-----------------------------------------------------------------

    * Not all the revisions will be appropriate for your document.
    * When there is more than one suggestion for just one bracketed
      word, you will have to choose the case that fits your use.
    * Capitalized words are instructions, not suggestions.

NOTE: If you want this program to look for additional phrases
      or to stop looking for some, for instance to stop
      flagging "impact," type the command dictadd.

*************************  GRAMMATICAL ERRORS  ************************

For file Example:

Possible grammatical errors:

      split infinitive: "to always edit " after line 0 file Example
      "a other" should be "an other" after line 0 file Example

For information on split infinitives type:
                           splitrules
```

Appendix 3
Page of Prose Output

Jan 22 12:38 1985 PROSE OUTPUT FOR Paper Page 1

NOTE: Your document is being compared against standards derived from 30 technical memoranda, classified as good by managers in the research area of Bell Laboratories.

READABILITY

The Kincaid readability formula predicts that your text can be read by someone with 10 or more years of schooling, which is a good score for documents like this.

VARIATION

Variation in sentence length, type, and openings prevents monotony. More importantly, a lack of such variation suggests that every topic and every sentence has equal weight, which makes it difficult for the reader to pick out the important points.

In this text 55% of the sentences are simple, 27% are complex, giving a difference of 28. This difference should range from -6 to 5 for good documents of this type.

Although the simple sentence is the most direct and comprehensible form for an individual sentence, overusing such sentences may make a document seem disjointed. Writing instructors say that a document is better when less important ideas are grammatically subordinated to more important ones so that the grammatical structure emphasizes the logical structure.

This document could be improved by combining some of the sentences to subordinate minor ideas to major ones. To do this, join two simple sentences by using a "that" clause or an adverb, such as "although." Put the less important sentence in the subordinate clause after the "that" clause or adverb. For example, the following sentences

a. The simple sentence is the most comprehensible form for an individual sentence.

b. Overusing such sentences may make a document seem disjointed.

were combined in the paragraph above. The combined sentence subordinates sentence "a" to sentence "b," thus emphasizing that the information in sentence b is more important than that in sentence "a."

Additionally, the longest sentence is 51 words long. Sentences this long are frequently lists, which will be easier to follow if you convert them into a list format. To find all your sentences over 50 words, type the following command after this program is done.

style -l 50 filename [l=lowercase L]

Concurrent processing in Ada+ and Unix++

H. W. Hesselink

Delft University of Technology, the Netherlands

ABSTRACT

Over the past years interest in programming languages supporting concurrent processing has grown. Some of these languages are extensions of existing ones, e.g. Concurrent Pascal, others were designed to support concurrent processing from the start. An interesting (if only because of its backing ...) example of the second group is Ada which is likely to become a major language in the future.

At Delft University of Technology a group is working on DAS, the Delft Ada Subset, which implements most of Ada except for its concurrent processing features. It therefore seemed an interesting exercise to compare the support for concurrent processing in Unix and Ada and to see to what extent this part of Ada could be mapped onto Unix. It may seem a little odd to compare a programming language with an operating system, however Unix in combination with a programming language (especially C obviously) offers a similar kind of programming environment to Ada. Ada has simply integrated what in Unix are separate system calls into itself (in the process ensuring that such facilities are standardised). The comparison is therefore between Ada on the one hand and Unix plus a programming language on the other. Because the programming language in this case was C the combination will henceforth be referred to as cunix. The version of Unix discussed in this paper is Version 7 so as not to cloud the issue with features present in some of the more current versions but not in others. Mention is made of the effect of some of these features on the cunix view of concurrent processing.

+Ada is a trademark of the Ada Joint Programming Office
++Unix is a trademark of AT&T Bell Laboratories

March 17, 1985

1. Concurrency in Ada

In Ada a concurrent process is called a task and is essentially just another data object. As such, tasks follow the Ada scoping rules*: they are declared at block entry and all tasks and possibly subtasks declared in a block must have terminated before that block can be exited. Tasks can be declared "anonymously", that is, the definition is also the declaration, or task types can be defined for use in later declarations. Arrays of tasks can be declared and one can have pointers to tasks. This last point is extremely important as this is the only way to dynamically allocate tasks.

Tasks are intended to be cheap, that is, creating a task should not involve much overhead. It is quite reasonable to declare an array of thousands of tasks to implement e.g. a finite element algorithm.

The language reference manual[1] places no restrictions on the way tasks are mapped onto hardware: all tasks on a single processor, each task a separate processor or anywhere in between.

Communication between tasks is via shared data and/or via the rendezvous. Data is shared according to the normal scoping rules, for instance a task has access to the variables (if not locally hidden) of the task that created it. A rendezvous can be compared to a call to a procedure in a different task. A rendezvous occurs when one task calls an entry in another task, which then accepts it. One of the two tasks will reach the rendezvous point first, at which time it will block. When the second task reaches the rendezvous point parameter values may be passed from the entering to the accepting task, this task will execute any statements in the body of the accept statement, parameter values may possibly be passed back to the entering task and then the tasks will go their respective ways. A rendezvous is therefore also a synchronisation point between tasks. A rendezvous is not symmetrical: the entering task must specify which task it wishes to connect to while the accepting task simply serves all entries on a first-come first-served basis. Multiple entries to one accept will be queued.

A task may listen to several entry queues simultaneously using the select statement. A select statement contains accepts for several types of entry, the first entry called is accepted. Each accept statement in a select may have a guard statement which will be evaluated at the time

---

* which are the "normal" rules of block structured languages such as algol and Pascal.

that the select statement is begun. Figure 1 shows a stylised piece of Ada code which implements a bounded buffer.

```
loop
   select
      when buffer not full
      accept a writer
          store data in buffer
      end accept
   or
      when buffer not empty
      accept a reader
          retrieve data from buffer
      end accept
   end select
end loop
```

Figure 1

Entry, accept and select statements can all limit the time they block waiting for a rendezvous. They can choose to execute only if a rendezvous is immediately possible or else to terminate if a rendezvous does not occur within a specified time.

Tasks terminate either by being aborted or by executing the terminate statement. A terminate statement may be an alternative in a select statement, it will be chosen if the block that created the task is at its end and all sibling and dependent tasks have either terminated or can also choose a terminate alternative. If this is the case all these tasks are dormant and may be painlessly removed.

2. Concurrency in cunix

A cunix process can create a new process by executing the fork systemcall. This results in a new process (the child) which is an exact copy of the old process (the parent) except for the return value from the call to fork. This return value may be used to distinguish between the two processes. The copy includes status information such as signal settings and the list of open files so that, for instance, files open in the parent at the time of the fork are also open in the child. Each user is allowed some maximum number, typically 25, of simultaneous processes. Processes cannot share data.

A cunix program can start a new program by executing the exec systemcall. This causes the old program to be overlayed with the new program, it does not result in a new process. Some status information is kept across the exec so that for instance files open in the old program will be open in the new program (but special signal settings will not be

kept).

A cunix program can therefore create a new program by first forking off a copy of itself and then execing the new program.

Communication between processes is via pipes which are essentially anonymous FIFO buffers. By executing the pipe systemcall a process obtains two file-descriptors to a new pipe, one to write to the pipe and one to read from it. If the process now forks off a child, because files remain open across a fork, the parent and the child can communicate via the pipe. Obviously they must be agreed on which will be the reader and which the writer.

A pipe is simply an ordered byte stream, no format is imposed on data passing through it. A process writing to a pipe that is full or reading from a pipe that is empty will block. A process reading from an empty pipe with no-one to write to it will receive a signal.

Several processes can write to the same pipe although a reader cannot determine where data it reads originated or even where boundaries between data from different processes lie. Several processes can read from the same pipe but which process actually gets the data depends on which process is running at the time the data becomes available. Because a process can only read from one file-descriptor at a time it is not possible for a process to "listen" to several pipes and read from whichever pipe has data available first.

A process terminates by being aborted or by executing the exit systemcall, it can do this whenever it wishes. A parent process can wait for one of its children to terminate by executing the wait systemcall. This call will return immediately if a child has already terminated (and has not yet been waited for), else the parent is suspended until a child terminates. The wait call returns the identity of the child that terminated as well as its exit status.

3. Two views of concurrent processing

Lauer and Needham[2] argue that most operating systems (the successful ones anyway ...) fall naturally into one of two classes. These they call procedure-oriented systems and message-oriented systems. According to Lauer and Needham a procedure-oriented system is "characterised by a large, rapidly changing number of processes and a process synchronisation mechanism based on shared data" and a message-oriented system is "characterised by a relatively small number of processes with an explicit message system for communicating between them".

A procedure-oriented system allows easy creation and deletion of processes. These processes typically communicate via shared data and cooperate using locks, semaphores or some other similar mechanism. Each process is usually associated with a single goal or task, processes contending for a resource are queued on a lock associated with that resource. Procedure-oriented systems tend to have elaborate support for access to shared data.

A message-oriented system allows easy passing of messages between processes. The number of processes is fairly static, process creation and deletion is usually expensive. In these kind of systems processes are usually associated with a particular resource. Communication between processes is via specific paths which tend to exist for relatively long periods and processes can choose to receive a particular kind of message or a message of any kind over these paths. Cooperation between processes (synchronisation, resource sharing etc.) is achieved via message passing, messages that have not yet been acted upon are queued at the destination. Processes rarely share data.

Lauer and Needham also argue that these two classes are duals of each other. For each model they define a small set of primitives necessary to implement a system supporting that model. They show that the performance of the two types of system is the same and then show how a program written for one type of system can be mapped into a program for the other type of system without changing the logic of that program. They assume the same underlying hardware, not optimised for one or the other model, or else different hardware with a similar level of support for each model.

They conclude that the choice of model for a particular system* will not influence the structure and performance of applications programs that are to run on the system, nor will it influence the complexity of the system itself (what they term "zeroth-order" and "first-order" considerations). Only "higher-order" considerations such as the facilities offered by the underlying hardware (and personal preferences of the implementors ...) are relevant to the choice.

It seems reasonable to extend this classification to cover concurrent systems in general. What conclusions can be drawn about the support for concurrent processing in Ada and in cunix?

If we look at Ada we see that tasks are cheap and that

---

* or subsystem, some systems can be viewed as a collection of subsystems that behave according to one or the other model and that are coupled by explicit interface mechanisms.

they can communicate via shared data. Furthermore, a locking mechanism such as semaphores is easily implemented. This means that Ada supports the procedure-oriented model. However, a flexible message passing mechanism is also provided that supports all the primitives Lauer and Needham suggest are necessary in a message-oriented system so that the message-oriented model is supported too.

Processes in cunix do not have shared memory and while processes can be easily created and destroyed this is still a fairly expensive business, so that cunix does not support the procedure-oriented model. While pipes support message passing their state cannot be examined and it is not possible to listen to several pipes simultaneously. This means that cunix also cannot be said to support the message-oriented model.

Ada supports both models. It was argued that the models are duals of each other so that this seems rather a case of overkill. Cunix originally supported neither model. Newer releases of Unix have added features which change this situation, some mapping quite closely onto the primitives suggested by Lauer and Needham. It is interesting to see that Berkeley inclines to the message-oriented model (e.g. the select call in 4.2Bsd) while AT&T inclines to the procedure-oriented model in the form of e.g. shared memory.

4. Mapping Ada tasking onto cunix

Without modifying the Unix kernel there are essentially two ways to implement tasking in cunix. One method is to combine all tasks into a single cunix process together with some form of scheduler, the other method is to map each task onto a separate cunix process.

The first method effectively involves building one's own little procedure-oriented operating system into cunix. Some advantages of this approach are:

- shared data is available for free
- because the system has to be built anyway it can be designed to provide optimal support for tasking.

A disadvantage is that no real concurrency can be achieved because, underneath, there is just the one cunix process. This can lead to other problems, for instance a task blocked on an I/O operation causes all other tasks to be blocked also.

The second method has the following problems:

- lack of control over processes (e.g. how to tell when a process dies)

- pipes form a limited inter-process communication mechanism

- processes cannot share data

- the limited number of processes that may be created

- mapping a task onto a process may be overkill

and is usually not even considered. It seemed interesting however to see exactly what the limits of this approach are, so a model of part of Ada tasking was devised and implemented in cunix*.

The model supports:

- tasks as "access types" (Ada terminology), that is, as objects that may (only) be referenced by pointers

- the rendezvous mechanism with entry, accept and select statements

- parameter passing at rendezvous

- guards on accept statements

- the ability to limit the time a task will wait for a rendezvous

- the "count attribute" (Ada terminology) which allows a task to determine how many tasks are queued on one of its entries

Not supported are:

- shared data

- scope rules (neither the visibility of shared data nor the lifetime of tasks)

- the terminate statement as an alternative in a select statement

According to Lauer and Needham shared data can be mapped onto a message passing mechanism. One of the tests of the cunix tasking implementation was to port Habermann's[3] Ada version of the dining philosophers problem to it. That version uses a separate task for each philosopher and for "philosopher manager" processes associated with each philosopher. Shared data is used to allow philosopher processes

---

* purely as an academic exercise, the code was in no way intended to be useable in the real world.

access to the philosopher manager processes of other philosophers. This last aspect was simulated in the cunix implementation using a separate task with no other function than to accept requests to process the "shared" data.

There is no shared data so the visibility of data in other tasks is a moot point. Because of the lack of control over processes it is very difficult to control the lifetime of tasks. Because tasks are access types a task can call an entry in any task it knows the address of so that it is impossible to control which tasks may communicate.

The terminate alternative was not implemented because it was felt enough code had already been generated for a mere academic exercise... It could be added provided that blocks that create tasks are identified and indicate when they are about to exit. This could be done as shown in figure 2.

```
{
    int block_id = new_block();

    .
    .
    creation of one or more tasks
    .
    .
    exit_block(block_id);
}
```

Figure 2

This would also allow more control over the lifetime of tasks and over which tasks may communicate with each other. The implementation could be made to look much cleaner (for instance the identification of blocks could be made transparent to the user) by using a preprocessor along the lines of Stroustrup's[4] classes program.

The implementation uses a separate task manager process of which all tasks are direct children, a new task is created by forking off a copy of the task manager. This means that the semantics of fork are lost: a new task does not inherit the environment of the creating task (e.g. the signal settings, the open files) but that of the task manager process. The implementation could be changed so that a new task is actually a child of the creating task, but the problems involved in letting the task manager talk to an indirect child make this very inefficient.

The major problem in implementing Ada tasking in cunix is the lack of shared data. Several versions of Unix now provide shared data but not in a form that will support

tasking. In terms of scope a task is simply a block and it therefore has access to the data of surrounding blocks. For instance a new task will have access to the local variables of the task that created it. This means that part of the stack must be shared. The stack of a program that has created several tasks might look like figure 3, in which each node of the tree represents the creation of a new task and each leaf is a task.

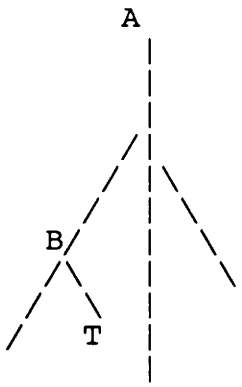


Figure 3

The line from T to B represents the private stack of task T, the path from B to A represents parts of the stack that are shared with other tasks. This is a so-called cactus stack, it cannot be implemented with a shared data mechanism that is intended to provide support for shared access to global data.

A mechanism developed at ACE[5] allows a process to consist of many logical segments which in turn consist of contiguous chunks of physical memory. These segments* may be arbitrarily shared, with independent attributes (e.g. read/write permission, the mapping into the process' address space) for each process. This mechanism leads to a very natural implementation of e.g. shared memory and COFF[6] , it could also be used, however, to create a cactus stack.

5. Conclusions

A summary is given of the facilities for concurrent processing in Ada and cunix. Two models of concurrent processing are discussed and the support for these models in Ada and cunix is considered. It is shown that Ada supports both models while cunix supports neither.

Two methods of mapping Ada tasking onto cunix are

* not to be confused with the usual Unix segments text, data and bss!

discussed: putting all tasks into a single cunix process; giving each task a separate process. Although the first method has several shortcomings it is usually chosen. To investigate the limits of the second method a part of Ada tasking was implemented in cunix. The implementation uses a separate process for each task, communication between tasks is via a message system implemented on top of pipes.

It was found that a large part of Ada tasking can be supported in this way though not with any great efficiency. Features added to newer versions of the Unix kernel, such as shared data and named pipes, do not solve the shortcomings of the first method but allow a much better implementation of the second method.

Due to the structure of Unix certain aspects of Ada tasking cannot be mapped using the second method. Problems are: the limited number of Unix processes available; the lack of an adequate shared data mechanism.

6. Acknowledgements

This work was done as a fourth-year project at the computer architecture group, Delft University of Technology, my thanks to them for their support and for the use of their facilities. I am indebted to Hans van Someren for many educational discussions (and for assigning me the project in the first place :-).

References

1. Reference Manual for the Ada Programming Language, United States Department of Defense (Jul. 1982).

2. Lauer, H. C. and Needham, R. M., "On the Duality of Operating Systems Structures," Proc. Second International Symposium on Operating Systems, IRIA, (Oct. 1978).

3. Habermann, A. N. and Perry, D. E., Ada for Experienced Programmers, Addison-Wesley (1983).

4. Stroustrup, B., Classes: An Abstract Data Type Facility for the C Language. May 20, 1981.

5. Someren, H. v., COFF Loader User Reference Manual, ACE Associated Computer Experts (Sep 13, 1984).

6. "Common Object File Format," in Unix System V Documentation, AT&T Bell Laboratories ().

Greek Characters on UNIX*

Stephen Hull

Research Centre of Crete†

ABSTRACT

A case study is presented of the provision of Greek capabilities on a 4.2 BSD UNIX system. The problem is broken down, and the many complications are presented, both those specific to Greek and those generally applicable to any additional character set. By evaluating the tradeoffs associated with the various issues, we arrive at a set of "best" suggestions for the task. Several commercially available solutions are briefly surveyed, and the issue of official standardisation is looked at. Finally, suggestions are made for future similar projects.

This research was partially supported by a NATO Science for Stability grant.

1. Introduction

In order to efficiently deal with the task of providing full Greek capabilities on a UNIX system, we must first consider what we mean by the term "full capabilities". A helpful way of approaching the definition of this term is to divide the user's interaction with the system into smaller pieces. Just as we can consider the interaction of a user as separate input and output streams connected to a tty driver process which in turn passes data on to the processor, utilities and storage, so can we break down our problem (figure 1).

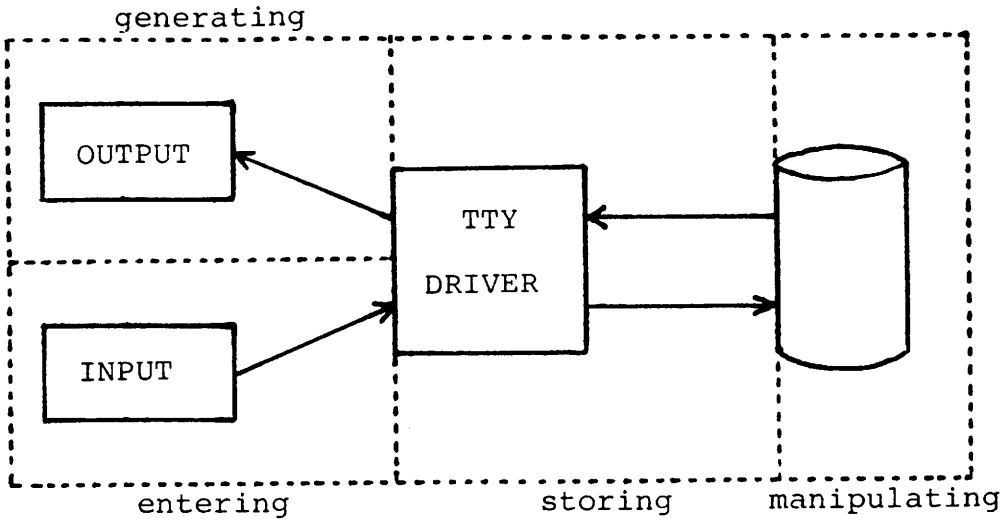


Figure 1 - Breakdown of Task

Specifically, we can identify the subtasks of

* entering,
* storing,
* manipulating and
* generating

Greek text.

Entering text involves passing data from the user to the tty driver. Storing text concerns the passage of data between the tty driver and the processor, utilities and storage. Manipulating text involves using the data we have entered. Finally, generating text concerns the passing of data from the tty driver to the output device.

There is a further division which can help us. When regarding the subtask of generating text, it is reasonable to consider regular and high-quality text. The idea here is not to limit the capabilities of regular text. Rather, both types would allow complete Greek interaction. The difference is that we can simplify our task by taking certain desirable but non-essential peculiarities of Greek text and providing them only in high-quality text. They can then be either approximated or ignored in regular text.

2. The Problem Domain

2.1. General Considerations

The desired capabilities should be provided in addition to standard ASCII communication and all the facilities

normally provided by UNIX. It is therefore necessary that our design still permit the use of UNIX and the I/O devices in the conventional manner.

There are several ISO standards pertaining to aspects of this task, such as encoding of 7- and 8-bit character sets. We would like to conform to these standards if possible.

The provision of the ASCII character set on UNIX is what we might term "homogeneous", that is, the bit strings generated at the input device, those passed between the tty driver and storage and sent from the driver to output are all identical. This need not be the case. As we shall see, there are a number of cases in our task where a difference may be desirable (if not necessary) for an effective solution.

2.2. Significant Characteristics of Greek

Unlike other European languages, where it may be necessary to supplement ASCII with several extra keys for additional letters or diacritical marks, Greek requires a whole new alphabet. As a result, our design must include a complete keyboard layout.

One unique problem of Greek concerns the letter sigma. While there is nothing unusual about upper case sigma, lower case sigma can be one of two symbols. "Teliko sigma" is used when the sigma occurs at the end of a word; the regular sigma is used in all other cases. Although teliko sigma is displayed differently and may be entered differently, it should be treated internally the same as a regular sigma.

Greek is perhaps unique in being a language with two different and not wholly compatible systems of diacritical marks. Greece in 1982 officially adopted the "monotonic" system of diacritical marks, in which there is a single accent as well as a diaeresis (called dhialytica). In addition, however, if we wish to provide full capabilities, we must also support the old "polytonic" system, with three accents, two breathing marks, a diaeresis and an additional mark, called ipogegrammeni or "iota subscript." A user may wish to combine the two systems or ignore one of them. It is also worth noting that mapping from polytonic to monotonic is a trivial operation, as all accents become the single monotonic accent and other marks are simply discarded. The reverse transformation, however is not simple.

Greek has several punctuation marks not found in an ASCII character set. These must be provided in some manner.

There are several characters in Greek not commonly or currently used. Classical Greek has four extra letters, the

dhigamma, koppa, stigma and sampi. There is also a letter-based numbering system similar to Roman numerals, called Byzantine numerals, which requires three of these classical letters and raised and lowered primes. In addition to all this, there is an alternative way of writing beta. There is an "initial beta" is used at the beginning of words in the same way that the teliko sigma is used at the end.

Finally there are a number of differences in the way upper and lower cases are dealt with. As mentioned before, there is no teliko sigma (or, for that matter, initial beta) in upper case: the standard capital sigma is always used. Diacritics are only displayed in upper case in the polytonic system, and then only if they fall on the initial letter of a word. All other marks are ignored. Accents and breathing marks are placed before the letter: this leads to special treatment of upper case alphas due to the proximity of the diacritics. The exception to all this is the diaeresis, which is displayed regardless of the position of the letter. However, it appears differently with upper case letters than it does with lower case. The ipogegrammeni, or iota subscript, appears under a lower case letter but comes after an upper case letter. Finally, when the letters omicron and upsilon appear together, they can be replaced by a diphthong denoting the two.

2.3. Hardware Characteristics

Our main interest here is terminals and the way in which they may deal with input and output. They may be classified as either 7-bit or 8-bit devices.

In addition to the standard ASCII table, 8-bit devices have the capability to accept up to another 128 characters. In practice, however, it is common to follow the guideline of International Standard ISO 2022, where we split the ASCII table into two sections: C0, a set of 32 control characters, G0, a set of 94 graphic characters, and the two special characters, SPACE and DEL. The eighth bit enables us to specify two alternative tables, C1 and G1. Since we want no extra control characters, this leaves us with G1, and we have 94 extra positions to assign characters to (see figure 2).

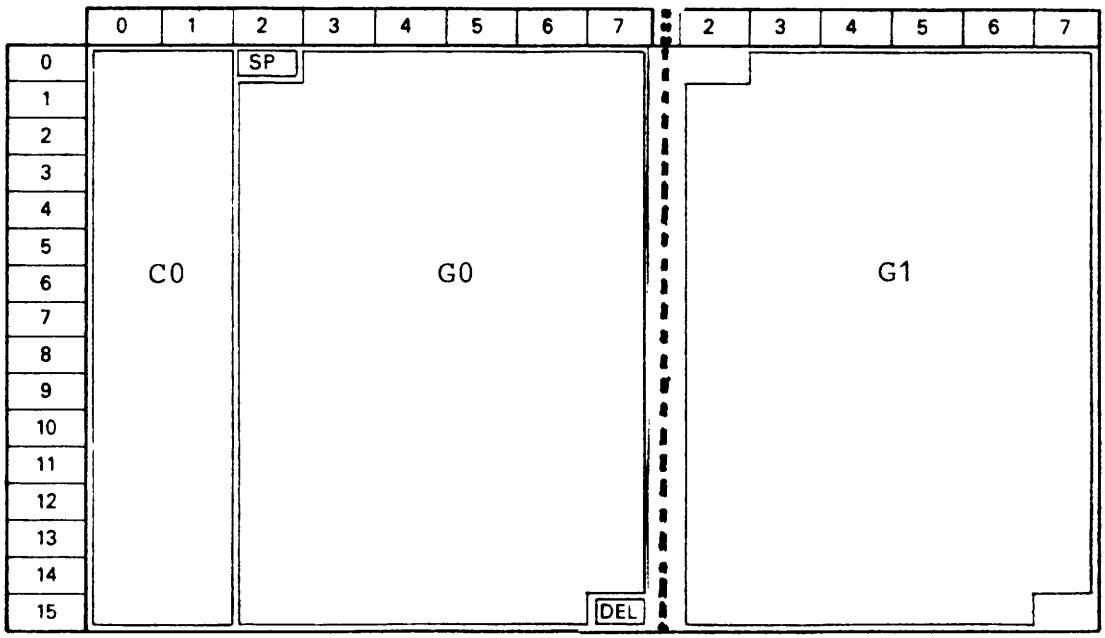


Figure 2 - Alternate Character Set (adapted from [ISO 2022]). In 8-bit devices, since the alternative table is indicated by the eighth bit of the character byte, we need no special escape sequences.

In the case of 7-bit devices, we need such escape sequences. Not only must the input device send sequences to indicate if G0 or G1 is to be indexed by the 7-bit data, but similar sequences must be sent to the output device to indicate what is to be displayed. Note that a 7-bit terminal which cannot switch to displaying an alternative character set cannot display Greek in addition to ASCII. We also note that in order not to have to switch character sets, it may be desirable to duplicate commonly used ASCII characters in the alternative set.

We should also consider the use of hardcopy output devices. Line printers would need a special print band and so can be safely ignored. Dot matrix printers and laser printers are the most promising types of device. Both can take 8-bit data, and it is simply a matter of providing software or ROMs to specify the characters corresponding to these data. It is possible that some of our requirements might be handled at the level of the hardcopy device, thus relieving us of them elsewhere.

2.4. Entering Text

We must decide on the full character set we wish to be able to enter. This set must be mapped to a standard keyboard. Finally, we must decide upon the bit sequences with which we will encode them and send them to the tty driver. We note that, in the case of 8-bit terminals, it may be

possible to do all necessary encoding within the terminal and simply have the driver pass the data unaltered into storage. This is not possible with a 7-bit terminal, and the encoding for entering will differ somehow from the storage and probably the generating encodings.

We should bear in mind that we may wish to provide standard ASCII characters in the Greek keyboard layout, especially punctuation marks common to ASCII and Greek and the numerals.

We should also take into account the possibility that there might be situations where the user does not want the full Greek capabilities. Particularly, there may only be a need for the monotonic accent system; perhaps in this case the user should not be burdened with the polytonic as well.

2.5. Storing Text

As the data passes through the tty driver to a buffer or file, we may wish to alter the encoding for several reasons. We may have an ASCII character which was sent from the Greek keyboard with a non-ASCII encoding which we should change to ASCII. We may have a sequence of key hits which should be stored as a single character or, conversely, a character on the Greek keyboard which signifies more than one character. Of particular interest here is the way in which diacritical marks are handled. When we re-encode the data, we should take into account the use we may make of the text. Different applications may suggest different encodings of characters; nevertheless, we should find one suitable for all.

There are also several issues which must be considered which are independent of such things as applications.

Should information about which character set is being used be stored in each character, or should we store special characters to indicate switching from one set to the other? Should we keep characters one byte long, or expand them to two bytes?

Tied in with this last point is the issue of future expansion. We may wish to consider future polyalphabetic systems which support a wide variety of languages. Should we design our system to facilitate the eventual incorporation of such capabilities?

2.6. Manipulating Text

There are two areas we must consider here: existing utilities on UNIX and the need for additional utilities.

Existing utilities, both standard UNIX ones and UNIX

based products, may or may not be able to work with full Greek capabilities. Those that cannot may be altered if possible, or we may wish to provide alternative utilities.

There may be a need for new utilities, either to deal with some of the problems presented by Greek (such as facilities to map between upper case and lower case or between polytonic and monotonic systems of diacritics) or to provide facilities for operations not encountered before, such as transliteration of Greek text into the Latin alphabet and back.

2.7. Generating Text

The primary decisions made here concern what encodings must pass from the tty driver to the output device and what they should signify. Decisions made here should reflect the choices made with regard to entering text since, in the case of a terminal, the user should see some sort of causal connection between what is entered and what is displayed.

Of particular interest here are the use of dead keys, that is, keys which result in the display of a character but do not advance the cursor position. Dead keys are used for characters which are intended to appear in the same location as the following character. We are also interested in the combination of characters within the output device in order to relieve the tty driver of such concerns.

We must also decide which facilities for Greek display, if any, should be considered high-quality text, and how they should be provided.

3. Evaluations of Issues and Choices

3.1. The Greek Language

1. ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩαβγδεζηθικλμνξοπρστυφχψως

2. 0123456789

3. ' ¨ ᾽

4. ᾽ ῾ ~ ᾽ ῾ ¨ ῀ ᾽ ῾ ῀ ῾ ῾ ¨ ¨

5. . , : ; ! - ( )

6. — « » ·

7. ϙ ϡ ϛ ϝ ϐ , ' ϻ

8. Ἀ Ἁ Ἂ Ἃ Ἄ Ἅ Ἆ Ἇ Ἰ Ἱ

9. ά έ ή ί ϊ ΐ ό ύ ϋ ΰ ώ

10. ά ὰ ᾶ ά ὰ ᾶ ᾰ ᾱ ᾲ ᾳ ᾴ ᾷ ᾰ ᾱ ᾲ ᾳ ᾴ ᾷ
έ ὲ έ ὲ ἕ ἔ Ἓ Ἔ
ή ὴ ῆ ή ὴ ῆ ῐ ῑ ῒ ΐ ῗ ῃ ῄ ῂ ῆ ῇ ῐ ῑ ῒ ΐ ῗ
ί ὶ ί ὶ ῖ ϊ ῖ ι ῖ ῖ ῖ ῖ ί ὶ
ό ὸ ό ὸ ὅ ὄ Ὅ Ὄ
ῤ ῥ
ύ ὺ ύ ὺ ῦ ϋ ῦ ῠ ῡ ῢ ΰ ῧ ῠ ῡ ῢ ΰ ῧ
ώ ὼ ῶ ώ ὼ ῶ ῶ ῶ ῶ ῶ ῶ ῳ ῴ ῲ ῶ ῴ ῲ ῶ ῳ ῴ ῲ ῶ ῷ
Ą
Ę
Ǫ

Figure 3 - Greek Characters

Figure 3 lists all of the characters we might want for our full Greek solution. As can be seen they break down into ten categories.

Category one consists of the 49 upper and lower case letters, including the teliko sigma.

Category two is the ten numerals, already provided in ASCII.

Category three is the set of monotonic diacritical marks: the accent, the diaeresis and the combination of the two.

Category four comprises the polytonic diacritics: the accents oxeia (acute), vareia (grave) and perispomeni

(circumflex), the breathing marks psili (smooth) and dhaseia (rough), the diaeresis, the iota subscript and the valid combinations (excluding the iota subscript).

Category five is ASCII punctuation which is used in Greek.

Category six is non-ASCII Greek punctuation: the pavla (dash), eisagogika (quotation marks) and ano teleia (raised dot).

Category seven is "extra" symbols: those rarely used, or used only in the ancient language. Here we have the koppa, sampi, stigma, dhigamma, initial beta, raised and lowered prime and the drachma symbol.

Category eight is special upper case combinations of letters and diacritics.

Category nine is the set of all possible combinations of letters and monotonic diacritics.

Finally, category ten is the set of all possible combinations of letters and polytonic diacritics.

3.2. Hardware Characteristics

It has already been noted that for 7-bit devices it is necessary to define escape characters to indicate which character set is being used. ISO 2022 does this for us: the "shift out" (SO - octal 016) character indicates switching to the G1 set and the "shift in" (SI - octal 017) character indicates switching to the G0 set.

There is also the issue of putting duplicates of common ASCII characters in our alternative set to lessen the amount of switching between character sets on 7-bit terminals. It should be noted that these characters would still be stored as their correct ASCII encodings, but could then be differently encoded for entry and display. This scheme sounds nice until we consider that the input and display facilities are only needed to deal with speeds a human user can operate at. Taking this into account, it seems unnecessary to complicate input and output in an attempt to shave a few minute fractions of a second from the response time. As a result, we can eliminate categories two and five, the ASCII numerals and punctuation, from the set of characters we need to consider.

Some of the characters described in the previous section may be considered as being only necessary for "high quality" text. Taking this approach, we can relegate their production to the output device. For example, some could be implemented in the software that drives or formats text for

a laser printer. The special positioning of diacritics with an initial upper case alpha and the omicron-upsilon diphthong might be handled this way, since the diacritic position is not critical (no pun intended) and the diphthong is by no means mandatory. Similarly, both the initial beta and the drachma symbol can be considered optional and the dash can be faked with two hyphens as it often is in Latin characters. The eisagogika or Greek quotation marks are often replaced by ordinary quotation marks in Greek texts when the correct ones are not available. Finally, the upper prime can be replaced by the ASCII apostrophe.

3.3. Storing Text

Perhaps the main question surrounding the provision of Greek characters is the actual encoding of the characters within the system. Because of this, and because the decisions made here will affect all other decisions we make, we will deal with this topic now. Note that this encoding may be done in the tty driver or in the input device. What is important is that this encoding is the internal representation of the characters.

We can identify seven issues which must be considered here:

* how to store information about which character set is being used,
* allowing for future expansion,
* contiguousness of the alphabet,
* transliteration of characters to the Latin alphabet,
* fall-back capabilities,
* the diacritic problem and
* the teliko sigma problem.

We have two locations for the character set information to choose between: we can place it with each character, either by setting the eighth bit or making characters two bytes long and placing information in the top byte, or we can store a special byte which indicates that the following data is from a specified set until another special byte appears, changing the set. Both this latter approach and the two-byte character approach have the advantage that we can expand to further character sets by simply defining new byte-long codes to indicate each new set. However, there are notable drawbacks as well. The two-byte character approach would require rewriting virtually every text handling utility in UNIX and would mean that all text files

were suddenly doubled in length and most of this length would be waste space. The "special byte" approach would effectively render the random accessing of Greek files impossible: it would be necessary to read a file serially from the front or back up to the nearest special byte to determine what sort of text was being accessed.

The eight-bit character approach doesn't suffer from these drawbacks, but it is accompanied by problems of its own. We are limited to two character sets, though standards exist (ISO 2022 and ISO 2375) for the establishment of escape sequences to specify additional character sets in the two available "positions". As a result, we are not too concerned about expansion possibilities. A more significant problem is the fact that many UNIX utilities cannot cope with the idea of eight-bit bytes. We shall save this topic for the section on manipulating data. Despite these problems, it seems clear to us that this approach is the least traumatic one to take, and will therefore continue this discussion using the eight-bit character as a model.

Perhaps the thorniest issue to consider here is contiguousness. The argument for contiguous lower and upper case alphabets is simply that one of the commonest operations on a computer is sorting, and the speed of this task is affected by whether or not a contiguous alphabet is used. Mapping an "unsortable" encoding to one that can be sorted and back will not affect the order of the time or space requirements, since it adds only a linear time factor, but the operations will definitely be slower and take more space.

Another desirable feature which ASCII employs is that the relative positioning of upper and lower case letters is such that corresponding forms of a letter differ by only one bit, simplifying operations where case is ignored or converted.

Some approaches to encoding a Greek character set attempt to facilitate the transliteration of Greek into Latin characters. This is done by locating the characters in places corresponding to similar looking or sounding characters in the ASCII table. Its effectiveness is limited by the fact that a one-to-one transliteration is crude at best (see figure 4) and that there are several currently

Figure 4 - A well-known English sentence
transliterated into Greek characters

used transliteration schemes, some of which map Greek characters to combinations of Latin letters or letters and accents.

When the output device is incapable of generating the desired character, we may wish to supply a reasonable approximation or replacement. This technique is termed fall-back. One reason to adopt a transliteration scheme is that it allows us to implement fall-back more easily. However, fall-back is something which can be easily be handled within the tty driver or the device itself. It seems unreasonable that such concerns should affect the way our data is stored.

The diacritic problem can be simply stated as what to do about characters which we wish to ignore some of the time. Diacritics are effectively meaningless in sorting, and we wish to be able sometimes to ignore them for searching operations (e.g. they may not be provided for upper case words). One glance at the set of characters we wish to support makes it clear that it would be unreasonable to store letter-diacritic combinations as single characters, even if it were not unreasonable to have, for example, twenty-four different ways of representing a lower case alpha, each of which could be encoded in a different way complete with diacritic. The question then becomes where or how to store the diacritics.

Two possibilities are to either place them where they occur in the word (before the letter they appear with, most likely) or to add them to the end of the word along with information about their placement. In the first case we have an approach which is simple to implement and takes only the minimum necessary storage. Sorting and searching, however, may require some sort of filter to remove or transform the diacritics. The second approach can simplify sorting and searching operations, particularly word-based ones. The major drawback is that this method requires more space, due to the need to store location information for each diacritic. As well, since the method storage is significantly different from the appearance of the entered or displayed text, the data must be processed between entry and storage and between storage and display. As a result,

storing diacritics with the letter is the preferred approach.

There are frequent instances where a letter will have more than one diacritic. Should all diacritics be stored separately, or should we have special characters corresponding to all combinations of diacritics? It is common for Greek typewriters to have "combination" keys. These keys allow the typing of combinations of polytonic accents, breathing marks and diaeresis. The keys are a convenience, but they also allow the separate design of the diacritic combinations for improved legibility and appearance. Combinations such as this can save us some storage (although an almost negligible amount) and don't set us back noticeably in terms of complexity or processing. On the negative side, as soon as we allow combination characters in storage we have more than one way of encoding the same information.

A solution is to allow combination keys for entering, but change the data to single diacritics for storage. There is only one monotonic combination, and there are eight possible combinations of polytonic diacritics less the iota subscript. We leave out the iota subscript because, not only do we not have to worry about legibility since it appears at the bottom of characters, but the complexity of a keyboard with 8+11=19 diacritical keys probably loses more than it gains in terms of efficiency and ease of use.

With the teliko sigma we have a case where we wish the character to be treated exactly like an ordinary sigma up to the point where it is displayed. Ideally our method of representing it will allow it to be treated as such while not decreasing the efficiency of our manipulations. The two concerns here are how we choose to represent the difference between the two sigmas and how we fit them into our alphabet.

We can simply have two encodings and deal with data manipulation problems as they arise. However, there are other approaches we can take as well. We can represent all lower case sigmas with the same encoding and leave it to the logic of the output device to display the proper character, depending on whether or not the following character is alphabetic. This is a pleasing approach, as the situation is handled in a manner reflecting the situation the characters are identical except for their appearance. Problems arise, however, when we wish to display either sigma out of context. We could have three sigmas, the regular "smart" sigma and two non-varying ones, or we could select two symbols never used with sigma, such as the acute and grave accents, and use them to indicate one or the other sigma. This seems rather unnatural, but we can be reasonably sure that any sigma specified in that manner will be an unlikely candidate for sorting or searching.

March 17, 1985

Our preferred approach to this problem is to select a single character such as an accent to indicate that the following sigma should be treated as a teliko sigma, and use this whenever one is called for. This effectively reduces the problem to the diacritic problem, the only difference being that the "diacritic" is not displayed.

In all cases, we benefit from the fact that the special character comes at the end of words. This means that its importance in sorting will be almost negligible. For this reason as well it seems very unnecessary to disturb the contiguousness of the alphabet by insertion of the teliko sigma when it can follow omega with little effect.

There are several documents about encoding Greek characters which are of particular interest. The first is ISO 5428, "Greek alphabet coded character set for bibliographic information interchange." It specifies an encoding for the basic alphabet plus the polytonic diacritics, a number of Greek punctuation marks and the four classical Greek letters in both cases (curious, since they are caseless), as well as the initial beta. The classical letters occur in their original positions and the special beta and teliko sigma are located after the regular beta and sigma. As a result, the modern Greek alphabet is far from contiguous. In addition, there is no monotonic accent system and no diacritic combinations are provided. We feel that there are too many shortcomings in this system for it to be adopted as is, but it is nonetheless interesting to see how some of the issues were addressed. We note that diacritics, that is, characters used in conjunction with another character, are kept distinct from symbols: non-alphanumeric characters which stand on their own.

The other document of particular interest to us is the CEC Liaison Report to ISO/TC97/SC2. It is a brief look at possible data exchange standards for Greek. ISO 5428 is mentioned and considered "not obviously compatible." Reference is also made to two transliteration schemes. The first is an informal one-to-one matching of Greek and Latin characters, evolved from use of Latin-based Telex machines. It corresponds to the standard layout of characters on a Greek keyboard. The second is ISO 843, which differs from the informal scheme in that two Greek characters are mapped to pairs of Latin letters and two further ones are mapped to accented Latin letters. The report goes on to suggest that contiguity of the alphabet be sacrificed in favour of the informal transliteration scheme. The advantage is seen to be the ability to enter and display a legible approximation to Greek on terminals having no Greek capability. At the same time, the paper notes that such capabilities offer no advantage within Greece. The report notes that a proper transliteration scheme, such as that of ISO 843, could be implemented on most systems quite easily, but that it is

unlikely that this would be found outside of Greece. It also suggests that contiguity is probably not as necessary a quality as in times past, simply because operations which benefit from it are so much faster nowadays that the saving in time is negligible.

The sentences of figure 4 were transliterated using this scheme. Given the crudeness of this scheme and the simplicity with which a proper transliteration could be provided, we suggest that more is to be gained by preserving the contiguous alphabet.

3.4. Entering Text

Having looked at ways the information might be stored in the computer, we can now consider the other aspects of the task and how they connect with storage.

Since the encoding sent from the terminal to the tty driver does not necessarily have to correspond to the stored encoding, we have considerable freedom to design the data entry. Nevertheless, if our terminal equipment is smart enough to process the information, we may be able to minimise changes to UNIX. The exception to this is when we use 7-bit terminals. Here the driver must keep track of which set is being used and translate the data properly.

The physical layout of the keyboard is quite straightforward. There is a generally accepted layout for Greek letters, corresponding to the crude transliteration scheme mentioned in the last section. This includes a position for the teliko sigma. Regardless of how we represent this character in storage, it seems best to follow conventional Greek typing practice and have the user explicitly choose which sigma to employ. As regards punctuation, we note that there is no standard layout for punctuation on Latin keyboards, although the ISO has published guidelines (ISO 3243). Although this standard is intended to guide the extension of Latin keyboards, we can base a Greek layout on it as well (see figure 5).

Figure 5 - Towards a Greek keyboard (after ISO 3243)

Using this standard, we have a minimum of 21 free positions on a standard 44-key keyboard. As well, there are five optional keys and four others which, though required for a Latin keyboard, are of little or no use in Greek, for a total possible of 30 key positions. Note that with this method we are keeping characters common to ASCII and Greek on the keyboard.

We have a possible 25 characters or character combinations we may wish to assign (see figure 6), but this number can be reduced.

Figure 6 - Characters to Assign

While the combination diacritic keys are useful ones to have, the three grave accent combinations are extremely rare, and we can simply require the user to enter the separate marks one at a time.

As far as the positioning of the specific characters, we can give a few reasonable guidelines. The commonest characters should occur in unshifted positions. These are without a doubt the single and combination monotonic diacritics. The quotation marks could be placed in the same shift

position on adjacent keys. The four seldom used classical letters might be relegated to a distant corner of the keyboard. However we might bear in mind the ISO guideline which suggests that the top right key be used for diacritical marks.

Accepted practice for entering diacritics on typewriters is that they be implemented as "dead" keys, that is, keys which, when depressed, produce a character but do not advance the cursor. It makes sense to keep this feature.

There are restrictions in Greek on which marks can be used with which letters, and it is simple to incorporate these into our system. Intentionally incorrect diacritics can still be faked using backspace characters if they are needed. For example, the diaeresis only occurs with iota and upsilon. When a disallowed key follows the diaeresis, one of four things could happen. We could have the character placed under the mark. We could have no response no change would occur until either an allowed letter or the erase key was typed. Typing the key could result in the erasure of the diacritic. Finally, the result could be the erasure of the diacritic and the entry of the chosen character.

The first two methods are probably the least desirable, the first because it allows the user to enter incorrect combinations, and the second because it involves extra keystrokes and would probably cause high user frustration. The remaining two approaches both have things to recommend them. Erasure and character entry means that if the user enters a faulty diacritic, no correction is necessary provided the letter needs no other diacritic. If the character is faulty, on the other hand, not only must it be erased, but the diacritic must be re-entered. Just erasing the diacritic would place the user in "starting position" for that letter again. This means that recovery is equally simple if the mistake was in the letter or in the diacritic. As well, simple erasure acts as a more noticeable sign that an error has occurred, so the less observant typist has a better chance of seeing the error. For these reasons we suggest erasure only as the preferred method of dealing with incorrect diacritic-letter combinations.

3.5. Manipulating Text

We can divide UNIX utilities into three groups: those which cannot deal with 8-bit data, those which can deal with it but cannot distinguish it from 7-bit (and thus operate less effectively than possible) and those which can deal with both types equally well.

The first type include utilities such as cc, the C compiler, which do not need the capability anyway (except for

commenting, and this can be accomplished relatively easily), but there are others, most notably vi, the screen editor. Vi uses the top bit to store information about the text being edited, so it is unlikely that it could ever be modified. Fortunately, we know of at least one commercially available UNIX visual editor which seems capable of doing the task, Human Computing Resources's HCR/EDIT.

The second group of utilities include such facilities as diff and file, the tools used respectively to find differences between two files and to determine what a file is data, text, binary, etc. These run fine on 8-bit files. The problem is that they don't correctly interpret them. File, for example, thinks that all such files are data, and diff successfully detects differences, but assumes that the file is a binary, and so does not specify what the differences are. We feel that this group of utilities can be fixed without too much problem.

The last group of utilities handles 8-bit files with no problem, generally because they do not need to interpret what the file's contents are. A good example here is tar, which archives to files, tape or standard input and output with no ill effects.

We should also consider here new utilities which Greek characters may require to be used effectively. Transliteration has already been mentioned. It is a fairly simple process to transliterate from Greek to Latin using any of a number of schemes, but the reverse is not necessarily the case. Similarly, we have no problem switching text from the polytonic to the monotonic system of diacritics, but the reverse is a tremendous task. The ease with which we can change case depends on whether or not we store accents with upper case words. Since the accents are only displayed with the first letter of a word, there is no need to specify them. But unless we do, we have insufficient information to produce a lower case version of the text. The opposite is, of course, not true. Furthermore, if a program converting lower to upper case preserves even those accents which are unseen, the reverse conversion can take place quite easily.

3.6. Generating Text

An initial note should be made here about the current state of Greek fonts on computers. Many facilities provide some sort of Greek capability for mathematical purposes. Often this takes the form of the set of letters which cannot be duplicated from an ASCII font. In such a scheme, letters such as upper case alpha, beta and eta must be replaced by A, B, H and so on. Such facilities never provide accents, but there is a more important shortcoming. This is that these fonts are not designed for text, but for symbols. As a result, there are many small details about the relative

weight of the characters, the way they fill the space (especially in a fixed-space font such as is found on a CRT) and the subjective position they have in a line of text that add up to make Greek text in these fonts at best crude-looking and at worst a headache to read. Although not a technical necessity, it is nonetheless important that good fonts be developed for the various output devices that may be used.

We can divide the remaining tasks here into two major areas: providing expected response to the user's input and high-quality text details.

All of the complications in providing expected response concern diacritical marks and involve the use of dead keys, overstriking and character replacement.

The dead key is a simple thing to provide. Displaying a diacritical mark and either not advancing the cursor or immediately backspacing is a straightforward operation. However, the next step, the placing of the letter under the diacritic can be accomplished in two very different manners. The first is to allow overstriking. This has the advantage of performing the exact actions the user enters, but can have problems, most notably on a CRT display. When the characters are formed by means of some sort of low-resolution matrix, such as in a dot-matrix printer or CRT, designing legible letters and accents which can be combined is a difficult task. In particular, a letter which looks good on its own may be awful with an accent, and a letter which can accommodate an accent may look out of place on its own. For this reason it may be advisable to use character replacement instead of overstriking.

In this situation the user may type in a diacritic which is again a dead key. However, when the letter to accompany the diacritic is typed, we do not have an over-strike. Instead, the diacritic is erased and a combination character put in its place. The advantage is obvious, but there are two possible drawbacks. First, the fact that the display does something far different from the user action could be disorienting to the user, particularly in situations where system response is slow. Second, each combination character is an additional character in our output character set, and the total number of possible combinations is quite large. In the monotonic system there are eleven possible combinations of accent, diacritic and lower case letter, and far more for the polytonic system 79 not counting the iota subscript and 118 with it. We are fortunate with the Greek alphabet in that, of all the letters which can take diacritical marks, only the rho has a descending part or descender and the rho is never used with the iota subscript and only the upper case alpha, eta and omega have ascenders and for these letters we must worry only about

the iota subscript. As a result, it is quite likely that we may not suffer the problems we have just described. If we do, there is a possible compromise solution which may combine the best of both approaches. This is to have different versions of the letters in question which are designed to take any diacritic. This would mean only eleven extra characters, but the result is that the regular characters would not have to be compromised to make room for extra marks.

High quality text issues have been briefly covered before. These include such things as correct location of diacritics with upper case letters and the omicron-upsilon diphthong. The idea here is to take various special cases or rarely used parts of Greek text and incorporate them into formatting facilities for a high quality output device such as a laser printer.

4. The Real World

Several manufacturers have developed Greek solutions of varying completeness. We will give a brief survey of them along with a short evaluation of each.

IBM has developed a ROM for the PC which contains a monotonic Greek character set. They leave implementation of the actual facilities to the OEM or ambitious user. Their specification makes teliko sigma a separate character; as well, the accent and diaeresis must be entered separately when they occur together. The layout of characters is frankly odd, and we suggest that it would create problems for applications using Greek.

Pronet Micro Systems of Toronto have developed a Greek system for the PC. Their implementation is very thorough, and is distinguished by their choice of two-byte characters. As a result, their possibilities for expansion are large, and they are in fact working on an Arabic character set at this time.

The University of Toronto has also developed a simple system to support the ATHENIANS project, a census of ancient Athens. It supports classical Greek, but does not allow mixing of Greek and ASCII characters.

Logoi Systems of New Hampshire(?) does Greek typesetting on microcomputer. Their system has full polytonic capabilities and produces quite attractive output. Of particular interest is their approach to sorting. Although diacritics are stored where they occur in words, a filter encodes them into a suffix for each word prior to a sort operation. The words are then converted back afterwards.

Perhaps the most impressive system we have seen is that

by Datapac SA of Thessaloniki, Greece. Datapac has provided a polytonic and monotonic solution integrated into System III UNIX. The system can support both 7- and 8-bit terminals and includes utilities redesigned to handle Greek. They cannot perform any high quality text functions, and do not provide any of the rare characters we have mentioned. Nevertheless, the facilities provided are done so in a clear, intuitive manner. Polytonic or monotonic can be selected, and the keyboard behaves appropriately for each system, including forbidding incorrect letter-diacritic combinations.

5. Footnote: Standardisation

One goal in the development of a character set is standardisation. This is desirable not only to establish the design and increase its credibility, but to promote adoption of similar schemes in the hopes of eliminating unnecessary future complications. The task of standardisation is itself complex, however. We now give a brief look at what standards exist and what bodies exist to manage these standards, in the hopes of clarifying some of the procedures called for.

There are two types of standards that concern us: those regarding character sets in general and those concerned specifically with Greek. Several of these we have already mentioned. ISO 3243 sets guidelines for Latin alphabet keyboards, but can serve as a useful guide for a Greek keyboard. ISO 646 and ISO 4873 are the standards for 7- and 8-bit character sets, respectively, and ISO 6429 concerns additional graphical control sequences. An organised procedure for extending these character sets through the use of escape sequences is presented in ISO 2022, and the procedure for registration of these sequences is to be found in ISO 2375. ISO 6937 is a general standard for character sets bringing together the ideas of most of these other standards.

Several documents exist which touch upon Greek characters specifically. We have mentioned ISO 5428, which specifies a Greek character set for bibliographic use, and the CEC Liaison Report. The European Community has published a document outlining what is known as the "EC-REPERTOIRE." This is a character repertoire proposed for exchanges between the member states of the EC. It includes EC-32, a Greek-oriented sub-repertoire which contains a full monotonic Greek alphabet, including combination characters. Finally there is a document produced by the Greek standards association with, we believe, the intention of being registered as a character set as per ISO 2375. It is called simply "Proposal 88," and consists of a set of 94 graphic characters. The authors seem to have attempted a compromise between a full transliteration scheme and a contiguous

alphabet, with the result that neither goal has been achieved.

In addition, any attempt to develop a standard must deal with several organisations. The most obvious is the International Standards Organisation. There is also a European standards organisation as well, known as SEN. The European Computer Manufacturers' Association should also be contacted. Any country will have its own national standards organisation, such as Greece's ELOT. Finally, there may be additional national organisations or government bodies concerned with science and technology. All of these organisations will be concerned with any attempt to develop a standard.

6. Conclusions and Recommendations

There are several points made in this paper which we feel may be usefully applied in future projects of this nature. The first is our approach of breaking down the requirements into entry, storage, manipulation and display. This recognises that, although entry, storage and display are linked in the user's perceptions, they are all, in fact, separate operations. We have seen that this separateness can be exploited to our benefit, yet we can retain the atomicity of the three operations which the user perceives.

Something which results directly from this approach is the realisation that the entry, storage and display encodings of the characters need not be the same. In an unaccented language it is possible to define an encoding which pays attention to storage issues and disregard requirements of the other operations with little problem: this is the genesis of ASCII. However, as Greek reveals particularly well, the different tasks have different needs, and there is little reason why we cannot develop encodings to satisfy each of these needs.

The ordering of diacritics in storage which we have suggested exemplifies this attitude. Keeping the marks integrated in the word makes sense from the user's point of view, but is a decided disadvantage in storage. So why store them in that manner? There is no reason. Our suggested method recognises this.

One last facet of our approach bears restating, and that is the discrimination between regular and high-quality text. We recognise the limitations of conventional 24x80 displays and simplify our task considerably. At the same time we place only minor limitations on the user, and pass the task of generating rare characters onto the devices which have the ability to do so.

Addressing in MMDF II

Steve Kille

<Steve@CS.UCL.AC.UK>

Department of Computer Science
University College London

ABSTRACT

The Multi-channel Memorandum Distribution Facility (MMDF) is a powerful and flexible Message Handling System for the UNIX* operating system. It is a message transport system designed to handle large numbers of messages in a robust and efficient manner. MMDF's modular design allows flexible choice of User Interfaces, and direct connection to several different message transfer protocols.

This paper is intended as a sequel to the paper presented by Doug Kingston at the 1984 Usenix meeting in Salt Lake City [Kingston84]. A brief technical overview of MMDF is given, and then two aspects are considered in more detail. First, the table-driven approach taken by MMDF to handling a structured address space is described, and then the extension of this approach to use with distributed nameservers is considered. Several distributed message systems using similar, but distinct, message and address formats have emerged. The message reformatting approach taken by MMDF to allow interworking is described. Finally, a comparison with other systems is made.

Introduction

The Multi-channel Memorandum Distribution Facility (MMDF) is a powerful and flexible Message Handling System for the UNIX operating system [Crocker79,Kingston84]. It was originally developed for use on Csnet [Comer83], to provide message relaying services. A version of MMDF stabilised in 1982 is the production system used by most Csnet

*UNIX is a Trademark of Bell Laboratories.

sites. MMDF II, which is described here, has many changes relative to the earlier system. MMDF II is on beta test at several sites both in Europe and the US, and is expected to be fully available before this paper is presented. Although it is not currently a production system, it has been used to provide message relaying services for about two years at the Ballistic Research Laboratories, Maryland and at University College London, and more recently on the Csnet hub machine. Each of these sites has a variety of particularly demanding requirements, and are regarded as good testbeds.

The main requirements and features of MMDF II are as follows:

(1) Robust, reliable, and efficient message relaying. Particular emphasis is placed on reducing message loss to an absolute minimum. A two phase warning is used to handle message transfer delays.

(2) Support for multiple Message Transfer Protocols, and general interworking between them. Currently supported are: Phonenet [Crocker79], the Arpanet Simple Mail Transfer Protocol (SMTP) [Postel82], the UK JNT Mail Protocol (Grey Book) [Kille84], UUCP Mail [Nowitz78], and indirectly CCITT X.400 protocols by use of the EAN system developed at University of British Columbia [CCITT83,Neufeld83].

(3) Support for a wide range of User Interfaces. At present, v6mail, Shoens Mail, msg/send [Vittal81], and MH [Borden79], can all be used with MMDF II.

(4) Authentication that is submission time verification that a message conforms to RFC 822 (the standard on which the generic message format for all MMDF II messages is based) [Crocker82], and that the source is correctly specified.

(5) Authorisation that is the restriction of message flow for policy reasons, or assigning responsibility for a given message transfer (possibly for billing purposes) on the basis of the networks, users, and hosts involved. This is discussed more fully in [Brink85].

(6) Submission time address checking. This allows for maximum address checking within the User Interface, and for more efficient network use by protocols such as SMTP and Phonenet.

(7) Flexible handling of local addresses, including aliasing, delivery to pipes and files, and enhanced support for distribution lists by use of a list channel. This is described in [Kingston84].

(8) User configurable delivery time options. This allows messages to be sorted according to destination and message header, and then processed accordingly. Processing can include: filing; redistribution; standard delivery; delivery to a pipe; user notification; destruction. This is described in more detail in [Kingston84].

(9) Straightforward runtime configuration (by use of a tailor file).

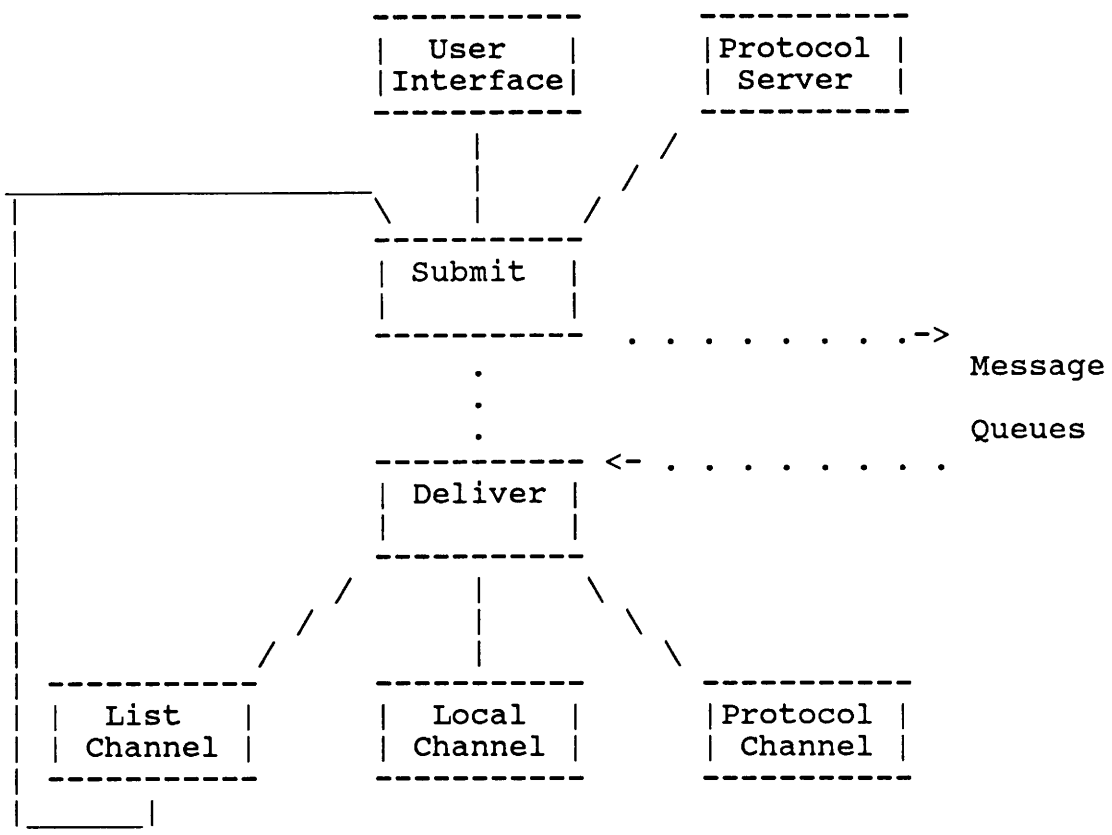
(10) Flexible handling of remote addresses.

(11) Message reformatting to support several environments using protocols similar to, but different from, RFC 822.

These last two points are the main subject of this paper.

MMDF System Structure

To describe the address handling, it is first necessary to understand the basic system structure.



MMDF Process Structure

The key to the operation of MMDF II is the process submit. Submit accepts messages in one of two modes:

(i) 'Message' mode, where a message is accepted on the standard input and addresses extracted from specified message header fields.

(ii) 'Protocol' mode, where addresses are first checked in an SMTP like negotiation [Postel82], and then the message is accepted. This mode is usually invoked by another process interacting with submit by use of a pair of pipes.[1]

Submit is invoked both by User Interface processes to send local messages, and by Message Transfer Protocol servers (e.g. an SMTP server, rmail,[2] or a JNT Mail Protocol server). Submit verifies the addresses, and then

[1] See pipe(2) in any UNIX reference manual.
[2] rmail is the process invoked by UUCP to transfer mail.

checks conformance to authorisation policies. It also authenticates the source and format of the message. The most important part of this authentication is that locally generated messages have a correct originator specification (i.e. that of the invoker of submit), and that messages originated remotely are only submitted by a process with the appropriate privileges. Finally, the message is queued, with the text of the message (both body and header) in one file, and the associated addresses and control information in another file. Each address is associated with a channel, and each channel has an associated queue, managed independently as a directory containing files which are links to the appropriate address file. The address to channel binding is discussed later.

Each channel has a process for delivering messages associated with it. More than one channel may use the same channel process (and thus protocol), as channels may have: different (sets of) hosts associated with them; different routes to the same hosts; or a different quality of service to the same hosts.[3] .ps +2 There are two special channels:

(1) Local channel. This delivers messages to local mailboxes, including any user specified processing.

(2) List channel. This allows a list to be expanded in two phases, by passing the message back to submit with a modified return address. This approach gives several advantages, particularly when handling large lists.

Both of these channels are discussed in more detail in [Kingston84]. Deliver is a process which reads the queue associated with one or more channels, and invokes a channel process with which it interacts through a pair of pipes. Deliver will read addresses from the queue, and pass them in turn to the channel process. When an address is fully processed (which may not be immediately if multiple addresses are being sent to a given host before any text is transferred) this information is passed back to deliver. Deliver then sends any necessary error messages, and the queue is adjusted accordingly. Caching and time control parameters allow for a variety of backoff strategies to handle hosts which do not respond. Deliver may also be invoked in pickup mode to allow messages to be pulled as well as pushed, which is particularly important for dialup links. This is discussed in [Crocker79].

---

[3] This flexible mapping becomes particularly important when applying authorisation on the basis of multiple criteria. A fuller discussion is given in [Brink85].

Address handling

The mechanism for verifying addresses is now considered. First, some terminology is defined:

AddressAddress is used to mean the text that the user supplies to a typical message sending interface in order to direct a message to a desired destination. This loose usage has been selected, because it is familiar to most message system users.

Domain
The DARPA domain scheme [Mockapetris84], and the UK Name Registration Scheme (NRS) [Larmouth83], allow an untyped global namespace to be allocated in an hierarchical fashion. Examples of domains are: UK, Salford.AC.UK, CompSci.Salford.AC.UK, USC-ISID.ARPA[4], ARPA, UUCP. If a domain is sufficiently specified .ps +2 it is possible to identify a direct connection to a Host, which may be associated with the domain or be a relay to the domain.

MailboxA mailbox (User Agent) is the source or destination of a message, often associated with a human user. A mailbox can be globally specified in RFC 822 as local-part@domain.

Host A host is a computer connected to directly by the local computer. A domain associated with a host not directly connected to by the local computer should be regarded only as a domain: the domain to host binding is immaterial if there is no direct connection.

Route
Routes are considered to be implicitly at the message level. A Route is an explicit sequence of domains over which a message should traverse. Two types of route are considered:

(i) A Formal Route consists of a sequence of globally valid domains, recognised as a source route by the originating system. In RFC 822, this is specified by a syntax of the style <@domain1:local-part@domain2>. A Formal Route is used to reach domains which cannot be accessed directly, or where indirect access is preferred.

---

[4] To identify a specific service in the context of domain UK is unlikely to make sense, whereas it might for Salford.AC.UK.

(ii) An Informal Route is one that is not recognised as a route by the sending system, but is interpreted as such by a receiving system. In the RFC 822 world, a common Informal Route syntax, making use of a special form of local-part is: user%domain2@domain1. Interestingly, this syntax is a Formal Route in the JNT Mail world. An Informal Route is used when different domains have a different interpretation of the global namespace (e.g. user%domain2@domain1 is used where the message originator's local system does not interpet domain2 as intended, but domain1 does).

A number of domain specifications have been adopted informally by different groups (e.g. .ARPA .AC.UK .MAILNET .UUCP .CSNET .CDN .EDU), and it seems likely that many networks will be able to agree on a global namespace. All Message Routing in MMDF II is currently controlled by tables. This is likely to continue for hosts where all external communication is by use of dialup links, and the NRS (at least) will distribute information as tables. The approach taken by MMDF II to table based domain handling is now described.

The MMDF II process submit performs the address checking function. There are two orthogonal operations. The first is to parse the address, to extract a Formal Route. The second phase is to interpret the 'end' domain of the route. If the domain being verified is identified as the local domain, the next domain component is considered. If the local-part is reached, it is then interpreted as an Informal Route. MMDF II supports a number of Informal Route specifications, including the UUCP "!" syntax. If the local-part under consideration is not an Informal Route, it is looked up as an alias. If it is identified as an alias, the alias value is then parsed as a sequence of addresses, and the whole procedure recurses. Finally, if it is not an alias, it will be checked as a local account and, if valid, queued for delivery by the local channel.

Each channel is associated with a set of hosts[5], which are directly connected to the local system. .ps +2 A host may be associated with more than one channel. Each channel has a channel table[6] .ps +2 which maps its associated hosts onto the necessary connection information (e.g. transport

---

[5] The directness is conceptual rather than physical. This is illustrated later.

[6] Tables are considered here as if they were physical files. In practice, they are implemented by hashed database lookup. Many of the operations described are optimised further by taking advantage of the structure of this database.

addresses). The host namespace is arbitrary, but given that all hostnames must be unique (to allow for host to channel binding), the convention that the hostname is that of the associated domain is convenient. A reason for sometimes violating this convention is discussed later.

Domains are handled in a two level manner. The top part of the domain is specified in the tailor file, and indicates an associated table. This top part is referred to as the domain table specification The rest of the domain is specified as the LHS of the table, with the RHS of the table specifying the host associated with the domain. The split can occur in more than one way. For example, CS.SALFORD.AC.UK could be split as:

domain table specification	table entry
SALFORD.AC.UK	CS
AC.UK	SALFORD.CS
UK	CS.SALFORD.AC
""	CS.SALFORD.AC.UK

The last case has a null domain table specification, known informally as the 'top' table.

The method of domain interpretation is now considered. When a potential domain is examined by submit, it is first assumed to be a full domain specification. .ps +2 Components are repeatedly stripped from the LHS and matched against the list of domain table specifications. Thus, if TREES.BIO.STANFORD.EDU is considered, domain table specifications are searched for in the order: BIO.STANFORD.EDU, STANFORD.EDU, EDU, "". For each match, the remainder is looked up in the associated domain table. If a domain is identified, it is then mapped into its official value by looking for the first component in the table with the same RHS. This allows for domain aliases. The RHS (host name) is then looked up in the channel tables, and the address is bound to the first channel satisfying management requirements. Some extensions to this basic procedure are now described.

Source RoutesAn alternative specification for the RHS of a domain table is as a series of components; the first being the host directly connected to, and the rest being a sequence of domains traversed in order. These domains will be added to the source route passed to the channel (message transport protocol). This source

---

The procedure is substantially optimised for domains in this form, as this is the most common case.

route information can be added in an ad hoc manner, or can be systematically obtained (e.g. from the NRS).

SubdomainsIf CS.SALFORD.AC.UK is specified, and has no corresponding domain table entries, but there is one for SALFORD.AC.UK, CS.SALFORD.AC.UK is accessed as a source route through SALFORD.AC.UK. This approach means that not all leaves of the tree need be stored. In particular, the 'top' table (i.e. the one with null domain table specification) may have entries such as:

```
CSNET:CSNET-RELAY.ARPA
CA.UUCP:UCB-VAX.ARPA
```

This would route all subdomains of CSNET (e.g. UPENN.CSNET) through CSNET-RELAY.ARPA, and all subdomains of CA.UUCP through UCB-VAX.ARPA. This is useful in (the currently common) cases where the logical domain structure has a physical mapping.

Partially specified domainsWhen all these checks have been made on the assumption of fully specified domains, these checks are repeated in each domain table on the assumption that the domain table specification was omitted. For example, if there is a table ARPA, and BRL is being tested, BRL will be looked up in table ARPA (and matched) on the assumption that BRL.ARPA was intended.

NRS domain orderingIt has been assumed until now that the UK NRS (Name Registration Scheme) and DARPA Domain scheme specify domains in the same manner. An unfortunate difference is, that although the semantics are similar and syntax the same, the ordering of the hierarchy is reversed. Thus, SALFORD.AC.UK in the DARPA form is UK.AC.SALFORD in NRS form. This difference has caused much confusion to (UK) users communicating with systems using both of these schemes, and so it is not very satisfactory to assume consistent specification (within the UK). Therefore, MMDF II may be configured to check for domains in either order. It does this in an interleaved fashion (i.e. for each check done performing it first in one order, and then the other). This minimises the weight given to the order a domain happens to be specified in, and maximises weight on the degree of specification (e.g. preferring to interpret "UK..AC.AVON.MULTICS" as MULTICS.AVON.AC.UK rather than UK.AC.AVON.MIT-MULTICS.ARPA). In practice, the few problems which have occurred, have been caused by palindromic partial domain specifications. Except where explicitly noted otherwise, this paper assumes DARPA ordering, as NRS ordering is mostly confined to

the UK Academic Community.

Routing by the channelThere are currently two cases of routing by channel. In the first case, which can be used for any channel, the channel connects only to one host acting as relay for all the hosts in the channel table. The second is the UUCP channel. Here, hosts are considered to be any UUCP host reachable through UUCP, and the RHS of the channel table is the UUCP route (expressed in the form host!host!host!%s). Use of this style of channel routing should be minimised, as it is preferable both to route incrementally and to bind addresses to channels as late as possible.

Support for multiple machinesIt is common for UNIX sites to have many machines, and to have an allocation of users between machines with no clear meaning external to the site. If the machine name must be specified to send mail to a user, this leads to names which are hard to memorise/guess, and makes the movement of mailboxes between machines visible external to the site. MMDF II allows for multiple machines to share a common namespace, common binaries and common tables, and for each user to have a default machine for mail delivery. Further, many machines can appear externally to be the same domain. This is achieved by treating each machine as a (different) subdomain of the visible domain (e.g. a message apparently from Steve@CS.UCL.AC.UK might be originated on machine VAX2.CS.UCL.AC.UK). This approach is clearly more user-friendly. In some circumstances it will be more efficient, and in others, less.

It is interesting to note how the MMDF II addressing approach fits into the transition of various networks to a domain based addressing scheme. There is currently a draft proposal for the UUCP transition [Horton85]. The author's interpretation of this suggests, assuming that this proposal is followed, that MMDF II will provides all of the needed functionality for users to get the full advantage of a domain based scheme. In the UK, the information supplied by the NRS can be used in a straightforward fashion.

Perhaps the most interesting case is the DARPA Domain Nameserver scheme [Mockapetris84]. In general, domain information will not be available in table form, but is obtained dynamically by querying a sequence of nameservers. MMDF II will be extended to support this within the timescale of the DARPA transition. A brief description of a possible approach is described, the aim being to illustrate that the DARPA scheme can be integrated, rather than to describe how it will be integrated. In general, a final solution will have a mixture of table and nameserver determined bindings. It seems likely that the domain namespaces

controlled by tables and nameservers will overlap substantially in many cases. A channel will be either table driven (as at present), or nameserver driven. A nameserver channel will be given a domain, which will be mapped into a connection (and possibly a route) when the domain in question is processed. At message submission time, the domain namespace would first be checked for fully specified domains contained in domain tables, which would be dealt with as before. .ps +2 A table (most likely the 'top' table) could specify a domain as being associated with one or more (nameserver) channels (e.g. ARPA maps to the SMTP channel). This domain would then be bound to one of those channels on the basis of management considerations. This gives submission time checking of only the higher level domains, with the full nameserver checking occurring in background when the channel is invoked. Alternatively, the domain could simply indicate use of a nameserver. In this case, the domain being checked would be passed to the nameserver. .ps +2 If matched, any specification of a 'Mail Forwarder' could be used to specify a source route. The 'class' of address returned would be looked up (in a table) to determine a set of channels. This fuller submission time checking is desirable, but would only be acceptable if the mean nameserver response time was not too large. Then, table checks for partial domains would be made. Assuming that completion services are provided by a local nameserver, the potential partial domain could be checked by the nameserver, if desired. This scheme would extend naturally if there was a need to use either different types of nameserver, or disjoint systems of the same type of nameserver. This is however, only one of several potential approaches.

Message Reformatting

The need to reformat messages is an unfortunate reality. It is caused by the requirement of interworking between functionally similar, but differently encoded, end to end message protocols. MMDF II provides two basic types of message reformatting: the first, applicable to all message transfer protocols; and the second protocol specific. MMDF II messages are expected (by submit) to be in a generic format, and are queued directly. This format is flexible, so that User Interface Processes do not need to have overly stringent requirements in order to generate legal format

---

Note that a domain determined from a table could be bound to a nameserver channel. In general, this would be undesirable.

Detailed interaction with a DARPA nameserver will be provided by a resolver interface, which is likely to be implemented as a library of functions on UNIX. This is certainly the case for two implementations currently in progress [Karp84,Terry84].

messages. In particular, the following aspects are flexible:

- Formal Routes may be specified either in RFC 822 format or as 'local-part@domain@domain'.

- Domains do not have to be normalised, that is to say they may be partially specified or fully specified aliases (e.g. ISID, USC-ISID, ISID.ARPA, USC-ISID.ARPA are all acceptable).

- Local domain specifications may be omitted.

In a system configured for the UK Academic Community, the following are also flexible:

- Formal routes may also be specified in 'local-part%domain@domain' form, as used by the JNT Mail Protocol.

- Domains may be specified in either NRS or DARPA order.

The general reformatting provided by MMDF II may be selected optionally on a per channel basis. This reformatting should be regarded conceptually, as being provided by deliver. It is really performed within the standard routines for interaction with deliver, called by each channel. If reformatting is selected, before each message is processed, a reformatted header will be generated into a temporary file. This is then used (transparently) by the channel instead of the real message header. The reformatting functions provided are now described. It is interesting that they all relate to address format, which in practice is the only interesting message transfer protocol independent form of reformatting. If reformatting is selected for a given channel, an alternative for each of the functions must be specified.

(1) Domain normalisation (the conversion of all domain specifications to fully qualified preferred forms) is a basic function of message reformatting. An alternative form of normalisation makes use of the host namespace maintained by MMDF II and, in cases where a domain has an associated host, normalises to the hot name. This is useful when connected to two worlds with differing views of the global namespace, or in transition situations. A common host specification is the leftmost component of the domain (e.g. UPENN is the host associated with UPENN.CSNET).

(2) Formal Route specifications may be reformatted to either RFC 822 form (@domain1:local-part@domain2) or to percent form (user%domain2@domain1). This is useful for mapping between JNT Mail formal source routes and

RFC 822 formal source routes. It is also useful where domains are used internally, and are not known globally. This mechanism allows formal routes to be mapped into informal routes, thus making the domain interpretation private.

(3) Domain/Host specifications may be output in either DARPA or NRS ordering. This is protocol independent, as the NRS ordering is associated with the UK Academic Community and not with any specific message transfer protocol.

(4) The local domain may be mapped into another specified form. This is used when one system has a different name in different environments.

(5) A treatment known as 'exorcising' against a table of domains. It is assumed that only the domains specified are known by domains accessed through the channel, and addresses relative to all other domains will be mapped to an informal source route behind the local system.

Message reformatting is also applied on a protocol specific basis. This occurs in two places.

Channel ReformattingChannels may perform reformatting in addition to that provided as standard. The JNT Mail channel performs some simple reformatting, to ensure that the return path, calculated according to the protocol, is correct. MMDF II currently supports two UUCP channels. The first assumes that the remote site is 'modern' and does no reformatting on the basis that the remote site expects a standard RFC 822 message. The second UUCP channel assumes that the remote site is 'old fashioned', and maps all addresses into UUCP route syntax (e.g. user@CS.UCL.AC.UK is changed to ucl-cs!user). Both channels may be used simultaneously by one system.

Server ReformattingWhere an incoming message does not conform to the generic MMDF II format, the protocol server must perform reformatting. In practice, this is only done for UUCP and EAN X.400. The protocol server for UUCP (the rmail process), will map incoming addresses, which (by their syntax) appear to be from 'old fashioned' systems into RFC 822 format addresses. Where possible, UUCP routes are shortened. This reformatting will not change the message format if the remote UUCP site is 'modern'.

Comparison with other Systems

The only system considered in comparison is Sendmail [Allman83]. A comparison with various non-UNIX systems

would be interesting, but not appropriate here. No other UNIX systems with comparable functionality are known to the author. Only addressing and message reformatting aspects are considered here.

Sendmail's address parsing and message reformatting is controlled by a configuration file. Sendmail mailers are analogous to MMDF II channels, but are less tightly coupled. Sendmail's address parsing and domain interpretation are controlled by the configuration file, and are not specifically decoupled. Whilst allowing for an amazingly general syntax, it can lead to cases where domain interpretation is syntax dependent. [10] .ps +2 The decoupling of address parsing and domain interpretation, as provided by MMDF II, is seen as both correct and desirable.

Current domain handling by Sendmail uses explicit recognition of domains listed in the configuration file to bind to a given mailer. After this partial address checking, the message is then queued for a specific mailer, where further checking is done. If configuration files are kept to a reasonable size, this approach must rely on the fact that the higher level domains can be used to determine the message transfer protocol in most cases. The MMDF II approach of maximising address checking at submission time is seen as desirable in view of the trend towards logical domain specifications which do not imply specific message transfer protocols. A domain handling package is expected to be added to Sendmail in the near future, and this may well allow for a more flexible approach.

Sendmail reformatting, is highly general and flexible, and has been used to deal with a variety of unusual formats. Some difficulty has been encountered when trying to handle NRS domain ordering, but this is probably not insuperable. MMDF II's more basic facilities cover a wide range of commonly used formats and, where appropriate, are more straightforward to configure. It seems likely that more systems will be moving to standard formats. The simpler approach also tends towards greater robustness, and protects against protocol violations.

Afterword

MMDF II is expected to be available as a production system by the time this paper is presented. It is available in the US through University of Delaware, and in the UK through University College London. There is a (currently

---

Careful design of Sendmail configuration files should minimise this dependency, although generation of configuration files to handle complex cases is not straightforward.

nominal) licence fee for commercial sites, and a tape handling charge.

References

Allman83. E. Allman, "SENDMAIL - An Internetwork Mail Router," Paper (1983).

Borden79. B.S. Borden and others, "The MH Message Handling System," Project Airforce document, obtainable from RAND, Santa Monica, Ca 90406, (November 1979).

Brink85. D. Brink and S.E. Kille, "Authorisation and Accounting in Store and Forward Messaging systems," Submitted to Networks 85, Wembley, (June 1985).

CCITT83. , "Recommendations X.400," Message Handling Systems: System Model - Service Elements (November 1983).

Comer83. D.A. Comer, "A Computer Science Research Network: CSNET," Communications of the ACM Vol. 26(10) pp. 747-753 (October 1983).

Crocker79. D. Crocker, E. Szwkowski, and D. Farber, "An Internetwork Memo Distribution Capability - MMDF," Proc. 6th. Data Comm Symp, IEEE/ACM, (November 1979).

Crocker82. D.H. Crocker, "Standard of the Format of ARPA Internet Text Messages," RFC 822 (August 1982).

Horton85. M.R. Horton, "UUCP Mail Transmission Format Standard," Draft received as message (January 1985).

Karp84. P.D. Karp, "DRUID: A Distributed Name Server," Stanford Internal Report (June 1984).

Kille84. S.E. Kille, (editor), JNT Mail Protocol (revision 1.0), Joint Network Team, Rutherford Appleton Laboratory (March 1984).

Kingston84. D.P. Kingston, "MMDFII: A Technical Review," Usenix Conference, (August 1984).

Larmouth83. J. Larmouth, "JNT Name Registration Technical Guide," Salford University Computer Centre, (April 1983).

Mockapetris84. P.V. Mockapetris, "A Domain Nameserver Scheme," IFIP WG 6.5 Conference, Nottingham, (May 1984).

Neufeld83. G.W. Neufeld, "EAN: A distributed message system," Proceedings CIPS National Meeting, Ottowa, pp. 144-149 (May 1983).

Nowitz78. D.A. Nowitz and M.E. Lesk, "A Dial-up Network of UNIX systems," UNIX Programmer's Manual, 7th edition, Bell Labs. (August 1978).

Postel82. J.B. Postel, "SIMPLE MAIL TRANSFER PROTOCOL," RFC 821 (August 1982).

Terry84. D.B. Terry and others, "The Berkeley Internet Name Domain Server," Usenix, Salt Lake City, (August 1984).

Vittal81. J. Vittal, "MSG: A simple message system," Proc. Int. Symp. Computer Message Systems, Ottowa, North Holland, (April 1981).

RFC indicates a DARPA Request For Comments, which may be obtained from: USC Information Sciences Institute, Marina del Rey, Ca. USA.

Acknowledgements

Many people have worked on MMDF II, including Phil Cockroft, Bernie Cossell, Dave Farber, Dan Long, Lee McLoughlin, Mike Muus, Julian Onions, Brendan Reilly, Denis Rockwell, and Marshall Rose. Particular credit to Dave Crocker who designed the bulk of MMDF II, and to Doug Kingston who bore the brunt of making it work as a production system.

ACSNET - The Australian Alternative to UUCP

Piers Dick-Lauder

Basser Dept of Computer Science
University of Sydney

R.J. Kummerfeld

Basser Dept of Computer Science
University of Sydney

Robert Elz

Dept of Computer Science
University of Melbourne

ABSTRACT

ACSNET is a network with goals to serve a function similar to that currently served by the UUCP network. Routing is implicit, and addressing absolute, with domains. The network daemons attempt to make use of full available bandwidth on whatever communication medium is used for the connection. Messages consist merely of binary information to be transmitted to a handler at the remote site. That handler then treats the message as mail, news, files, or anything else. Intermediate nodes need not consider the type of the message, nor its contents.

1. Introduction

ACSNET is a loosely coupled network of heterogeneous machines, and has a purpose and function similar to that provided by the UUCP network in wide use in most of the UNIX* world.

Before continuing we must make two points. First the matter of naming. Perhaps ACSNET's biggest problem is the lack of a suitable name. The developers (PD-L and RJK) call the software 'The Sydney Unix Network' or SUN for short, while the network built using SUN is called ACSNET. Unfortunately, 'SUN' may be confused with a Unix based workstation of the same name and 'ACSNET' suggests some relationship, or at least similarity with the U.S. CSNET network. No other label has gained sufficient approval to catch hold, so ACSNET serves for the time being.

Second, a note on our use of UUCP for comparisons with ACSNET in this paper. The authors have no intention to discredit UUCP, or to belittle its achievement in linking the world's UNIX systems in a manner never before attempted. However, its presence as a current de-facto standard for UNIX to UNIX communications places it in a position where we cannot avoid making comparisons in order to illustrate certain points with far greater economy of words than would otherwise be possible.

2. Overview

ACSNET provides a message passing service, from one host to another, possibly utilising intermediate hosts in a store and forward manner. Messages may be mail, files, printjobs, news, or almost anything that can be transferred in a string of bytes.

*UNIX is a Trademark of Bell Laboratories.

Routing in ACSNET is implicit, users need only be concerned about the name of the host at which the message is to be delivered. They need not be concerned about which hosts the message might visit on its journey to its final destination. If, for some reason, a message is undeliverable, the network will make every attempt to return the message to its original sender.

Messages can be transported over any medium capable of supporting a connection between two hosts. This may be phone lines, ethernet, X.25, twisted pairs, etc. Preferably, the link should provide a transparent 8 bit data link, but links with only 7 useful data bits can be accommodated (at a slight loss in throughput). The network daemons make good use of full duplex communication channels, transferring messages in both directions simultaneously, providing, in ideal conditions, effective throughputs up to twice that attainable with UUCP.

3. Messages and Handlers

A message is a string of bytes addressed to a handler at one or more hosts. A handler is a process that will receive the messages at the final destination. Typically the handler will impose some further protocol, often recognising a user name (in some form of representation) that the message is directed to. A message, of itself, is addressed merely to a host and a handler.

The notion of messages addressed to handlers is one of the primary differences between ACSNET and UUCP. UUCP functions as a remote command execution system built upon a file transfer protocol. Mail, news, etc. are transmitted by sending the content of the item to the remote system as a file, then sending a request to execute the remote mail, or news, receiver with the file as standard input. ACSNET simply

transfers a message addressed to the mail handler on the remote system.

ACSNET uses trailer protocols, where the header follows the message. This allows file copying to be avoided on intermediate hosts when routing statistics are updated. ACSNET only ever performs disk to disk copying of a message when it is being copied to its final destination, and optionally, when queueing the message in the first instance.

Currently handlers exist for mail, news, file transfer, and remote printing. A remote command execution handler could be added if the security issues could be adequately solved. Any other handler could be created just as easily, for any purpose that the sending and receiving hosts agree upon.

Unlike UUCP, it is not necessary for intermediate hosts to know of the new handler for correct functioning.

4. Addressing

All messages carry a destination host address. This is the ASCII name of the destination host. Messages also contain a source host address, and may contain a user address. This last item may be anything that the handler requires for its functioning.

Messages may be addressed to more than one host. A copy of the message will be sent to each host addressed, and that will be done with the minimum possible message traffic (or something approaching it). A message may also be broadcast to all hosts. This is most often used for network management messages, such as new hosts connecting, and similar events.

Users also have the option to guide their message through a specific set of hosts. Primarily this is used for

network testing, loopback messages would otherwise be impossible to create.

Such a route is expressed in a notation borrowed from UUCP as

host-1!host-2!host-3 ...

However, note that there is a fundamental difference between this form of addressing and UUCP addressing. Each host-N is the absolute address of some site. The ! does not imply a link between two adjacent hosts. In the above example, the message will visit, in order, host-1, host-2, and host-3. But the route taken to travel from host-2 to host-3 is not specified, and in fact, given a suitable topology, the message might travel that route via host-1!

Strictly, in all the above, the term 'host name' should be replaced by 'domain specification', but that is harder to type, read, say, and think about. Any of the places where a host name was specified, ACSNET would really expect a domain specification. Domains might simply be host names, or they might specify local sub-domains, or perhaps a domain that is not within the ACSNET network, in which case the message will be sent to an appropriate gateway.

Note, nowhere here has it been specified what syntax should be used by users in communicating with user agent programs. It is to be expected that

user@domain

will be the most common format, though the older Australian net syntax of

user:host

will be supported into the distant future. Almost none of the ACSNET code either knows or cares what syntax users will use to send messages.

5. Routing

Each host maintains two tables† to contain network information. The first of these is the network state file, and contains for each known host, a list of the domains to which it belongs, a list of the hosts to which it is directly linked, and the cost and current status of each of those links. One of the domains is special, and is considered to be the primary domain of the host.

The table can also contain various other information, such as a human understandable description of each host, and statistics on messages and bytes sent, received, and passed through. This information is optional, and probably would be deleted on a small host.

The state table is considered public information, and is sent to any host that requests it. It is broadcast to all hosts whenever a new link is added.

The second table is the routing table. This table indicates which link should be used to transmit a message bound for any host or domain on the net. It is built from the state table, usually whenever that is altered. As each message arrives on a link it is passed to a routing process. That process passes it to its appropriate handler if this is (one of) its final destination(s), and queues it to be transmitted on the next link if the message has further to go. The routing table is used to make this decision.

Information on which links to transmit a broadcast packet that originated at any particular host is also retained here. This is arranged so that broadcast packets travel over a minimum traversal of the graph, and implements Dalal and MetCalfe's extended reverse path forwarding algorithm.

---

† For 'table' read 'file'.
Y. K. Dalal and R. M. MetCalfe, CACM, Dec. 1978.

This table also contains miscellaneous information, such as local aliases for hosts on the network. It is private information, and is not exported to other hosts.

Routing messages are broadcast to all hosts in the sender's primary domain whenever a link changes status (goes up or down). These are brief messages, indicating the nature of the change, and carry a timeout age, after which they are deleted wherever found. This works well, as typically, distant parts of the net are not interested in local changes, which often might have become outdated before the message reached the host. Rerouting to avoid links that are down can usually be handled by nodes relatively close to the broken link.

6. Gateways

The routing table for any local link can indicate that a non-standard spooling program should be used to send a message over a designated link, or to deliver a message to a particular domain. This can be used to build interfaces to newer, or older, versions of the network software, or to implement a gateway to a foreign network. The spooling program is responsible for performing any transformations required of the message to meet the standards required by messages entering the new network.

This performs admirably when interfacing to a network with similar capabilities, but is less of a success connecting to UUCP for anything but mail, as there is no standard way of performing possibly multi-hop file or news transfers.

7. Calls and Daemons

ACSNET uses node to node daemons to transfer messages from one host to another. Unlike the UUCP uucico process, ACSNET daemons have no knowledge of how, or when, to connect to a remote host, except in the trivial case where that is

accomplished by opening a tty compatible special file. To handle other cases, the routing table may contain the name of a process to run to establish a connection to another host. That process is expected to make the connection, then exec the daemon with standard output open (read-write) to the remote host. This permits easy expansion to a wide variety of possible connection types.

A pair of daemons transfer data between themselves over three channels in each direction simultaneously. This allows up to 6 messages to be in flight between any pair of nodes at any one instant. Messages are assigned to one of the three channels based upon their size. This allows small messages to overtake larger ones on another channel, and prevents those extraordinary delays that can occur when a particularly huge message is being transferred.

The daemons also keep track of the current position in each message in transit. This allows messages to be restarted without transmitting data that had been received correctly at its destination, should a link die prematurely, or a system crash.

Messages on each channel are sent via a windowed packet scheme, similar to that used by HDLC (and UUCP, and X.25), and are checksummed using the standard CCITT CRC-16 algorithm. This checksumming can be disabled for a link if it is known to be reliable, typically if ACSNET is used over another protocol. Messages can also contain end to end checksums, to guard against corruption while waiting at an intermediate node.

8. Status

ACSNET is currently being used on VAX 11/780 and 11/750 processors, PDP-11/34's, Sun Workstations, Perkin-Elmer, Plexus (P60), ELXSI 6400, Gould, IBM PC, and other less widely known machines. These processors are variously operating

under V7, 4.*BSD, Systems III and V, and Venix/86. A VMS version has been suggested, but at this stage, not attempted.

There are about 200 hosts on the Australian network, and though a few of these currently use the previous software, the remainder will probably convert in the near future.

9. Todo

There will be a message disassembly, reassembly facility, to permit huge messages to be transmitted without overloading intermediate nodes.

A UUCP gateway is needed. This is hard because of problems with multi-hop file transfers.

Some tuning remains to be done. One possible improvement on System III and V systems, and 4.2BSD, would be to use the available interprocess communication mechanisms (named pipes, sockets) to allow the routing process and handlers to become daemons, and be created just once, rather than once per message.

10. Availability

The source code is available under license. Anyone interested should apply to Bob Kummerfeld at the address above.

11. Conclusions

ACSNET is a suitable network system for connecting comparatively large networks, of a size comparable to the UUCP network, that may operate in a relatively unmanaged environment. Its implicit routing makes it considerably easier to use than standard UUCP, and more accurate and adaptable than

the heuristic UUCP routing algorithms now becoming available.

We feel that ACSNET would be suitable as a general replacement for UUCP.

What is an international UNIX system?

Judy Guist - ucbvax!hpda!jg
Duncan Missimer - ucbvax!hpda!duncan
Hewlett-Packard
11000 Wolfe Road
Cupertino, CA 95014

ABSTRACT - The authors believe that it is possible to create a fully international, fully functional UNIX system which can assume more than one linguistic identity at run time. This paper presents a model for language- and custom-independent software, which we use to outline some of the ways in which vanilla UNIX is unsuitable for use outside of the English-speaking research environment. We define two terms - native language support (NLS), and localization - and show how these concepts can be used to design an international UNIX by moving language- or custom-dependent information out of the source code and into the file system where it belongs.

Some evolution is inevitable

As UNIX(tm) use in Europe and elsewhere moves out of the English-tolerant environment of research and academic institutions, the day of reckoning comes closer and closer. Eventually the crypto-American usage built into so much of the user-level software will have to be ripped out to provide the sort of processing that users need. Already Japanese vendors are offering versions of UNIX adapted for processing of Japanese characters[3], and Tintel's paper [2] describes a UNIX environment adapted to support the need to talk to the user in his own language.

Specialization vs. generality

When one finds a single country with very specialized processing needs and a large internal market, it is a great temptation to take some of the most important parts of a system and modify them for local needs, but for an organization which intends to support the whole of this rich development environment across a variety of languages, specialized adaptations for language X or Y represent an unacceptable cost in terms of software development and maintenance. The challenge, then, is to create a single system adapted to the linguistic requirements of a wide variety of people. Multilingual word processing is crucial to any worldwide offering, but is not enough. The ultimate goal should be to provide for any speaker of any language a system which suggests in all its parts no national identity other than that of the user, while offering him the full level of facilities available to the original user in English. This linguistic identity should be switchable at will, so that a researcher should be able to sort his mailing list according to Swedish custom and send it to a

colleague in Sweden, and then sort it according to Spanish usage for transmission to Madrid. Meanwhile, on another terminal on the same system, another user runs a spelling checker for yet a third language.

What will have to change?

At the risk of sounding presumptuous (or tedious) we review below some of the major parameters which for our purposes will characterize a language.

- character size - Since 7-bit ASCII doesn't contain the characters needed to support most of the world's languages, it will be necessary support a variety of 8- and 16-bit character sets. Software that edits strings, for example, must know whether to delete one or two bytes per character, and must not assume that the 8th bit is available for use as a flag. It is well known that some of the most important programs (sh, csh, the editors) assume 7-bit objects and use the 8th bit freely.

- shifting - Some languages discard accents on upshifting while others do not. Some languages do not even need the notion of "case" at all. Current library routines support only USASCII.

- collation - Sorting is provided by the routines strcmp(3), sort(1), qsort(3). This sorting is based on machine collation of ASCII characters and is not even adequate for American "dictionary" usage, much less a language like Spanish which treats "ch" and "ll" as single characters. Collation needs to vary according to the language used, as well as the character set. For East Asian languages with large character sets several collation sequences are typically needed, one based on sound, one based on number of strokes, one based on radicals, and so forth.

- directionality - The assumption that displayed text goes from left to right does not hold for languages such as Arabic or Hebrew. In some cases the East Asian languages may be written vertically. Needless to say, none of the standard software addresses this problem.

- classification - Classification of characters on the basis of the coded value will change from character set to character set, and can no longer be hard coded. Although most of the 8-bit character sets we support preserve the ASCII codes for the bytes 0-127, 16-bit character sets may use these byte values in 2-byte characters. Software which assigns special meaning to bytes ("metacharacters") in this range will have to take care to distinguish 1-byte from 2-byte characters.

- escape sequences - In multilingual environments standard escape sequences will most likely be used to indicate a change of character set. Most of these are based on ISO 2022. An example is the Japanese Industrial Standard sequences used to switch back and forth between an 8-bit character set based on ASCII and a full 16-bit ideographic character set. Software that processes characters may have to change its assumptions about the current state of any of the aspects discussed above, e.g. directionality, character size, when such sequences are encountered. Because of the amount of work involved, Japanese manufacturers have typically modified software to provide reduced functionality in some commands (vi, nroff) in order to support a mixture of two known character sets. The problem is not trivial.

- hyphenation and spelling (current "spell" and nroff hyphenation support English only)

- computation and display of date and time (some of the stickier problems here will be solar time in Saudi Arabia and emperor dates in Japan)

- names of days of the week and months

- names of currency units, ways of subdividing currency units

- representation of numbers - variations in the symbol used for the radix character - variation in the symbol used for grouping digits, as well as the number of digits grouped.

- Error messages, prompts and the responses to prompts, and mnemonic command names should all be based on the user's language. (Discussed by Tintel)

- If messages are built up in chunks (e.g. by printf()) it should be kept in mind that the syntax of another language may force a change in the order of the sentence fragments.

- Languages with complicated writing systems will have special input and display requirements of the sort discussed by Becker.

Given this tremendous list of points of divergence and our goal of supporting equitably a variety of languages over the range of standard UNIX software, where do we begin and how do we do it? First, how do we identify to the program what sort of language will be governing its behavior? Fortunately, we have here a good precedent, an area where a single program has been expected to support a wide range of

inconsistent user interfaces - terminals. Just as the environment variable "TERM" is passed in to allow a program to access the "termcap" data base, a language name can be passed in to identify the messages to be used for communicating with the user, the character set he will be using, his preferred rules for collation, etc. Such a mechanism would already allow a single user to change his linguistic identity at will.

Since the problems of local user messages and multilingual word processing have been discussed elsewhere [2, 1], this paper will focus on the remaining issues. Although some of them may seem trivial, these issues do stand in the way of using UNIX for any sort of business processing.

Several routines in the standard UNIX product represent an encapsulation of functions which are language sensitive. Examples are those described in the manual pages for ctime(3c), conv(3c), ctype(3c), and the sorting routines mentioned earlier. It is possible to parameterize these routines, either explicitly or implicitly, so that they no longer depend on ASCII-based tables compiled into the code, but rather combine a sufficiently general algorithm with data retrieved from the file system. If enough study is done ahead of time, addition of support for a new language will just involve the addition of data. Just as the invention of a new terminal model does not require recompilation of every program that uses the terminal data base, adding the capability to sort Arabic or Korean should not require algorithm changes to any existing software.

About time for an example

The following excerpt is ctime(3), taken from the Berkeley 4.2 distribution.

```
static struct dstab usdaytab[] = {
1974,5,333,/* 1974: Jan 6 - last Sun. in Nov */
1975,58,303,/* 1975: Last Sun. in Feb - last Sun in Oct */
0,119,303,/* all other years: end Apr - end Oct */
};
static struct dstab ausdaytab[] = {
1970,400,0,/* 1970: no daylight saving at all */
1971,303,0,/* 1971: daylight saving from Oct 31 */
1972,303,58,/* 1972: Jan 1 -> Feb 27 & Oct 31 -> dec 31 */
0,303,65,/* others: -> Mar 7, Oct 31 -> */
};

/*
 * The European tables ... based on hearsay
(omitted)
static struct dayrules {
intdst_type;/* number obtained from system */
intdst_hrs;/* hours to add when dst on */
structdstab *dst_rules;/* one of the above */
enum {STH,NTH}dst_hemi;/* southern, northern hemisphere */
} dayrules [] = {
(omitted)
struct tm *
localtime(tim)
unsigned long *tim;
{
(omitted)
for (dr = dayrules; dr->dst_type >= 0; dr++)
if (dr->dst_type == zone.tz_dsttime)
break;
if (dr->dst_type >= 0) {
year = ct->tm_year + 1900;
for (ds = dr->dst_rules; ds->dayyr; ds++)
if (ds->dayyr == year)
break;
(omitted)
switch (dr->dst_hemi) {
case NTH:
(omitted)
```

This represents the "quick hack" approach to internationalizing UNIX software and it suffers from the following limitations

It is brittle - one political change and the code is invalid.

It is closed - anybody who wants more or less than what is offered here is out of luck.

Somebody has to maintain a list of timezone numbers and make sure they match what comes from the kernel.

What is really needed is a "TIMECAP" environment variable which contains a string describing the relation between GMT and local time - certainly nothing as difficult as describing the behavior of a terminal. With a little legwork ahead of time it would be possible map solar and lunar calendars and any sort of scheme for starting and ending summer time. The result would be a scheme that needs no code changes to "boldly go where no software has gone before" and have a good change of working correctly when it gets there.

Some terminology

In our project we have used two terms to draw a distinction between generalizing the code and the actual creation and installation of data to support a particular language. We refer to the modification of code (which is really the removal of language dependencies) as "native language support". If we have done this work right all that is necessary to actually support a new language, set of customs, or geographic location is to add files characterizing that environment, a process we call "localization", and which should be thought of as being analogous to writing a new "termcap" entry.

Character set support

Extending this idea to the areas of character handling, we can read in dynamically tables to be indexed for classifying or shifting characters. Collation involves a lot more work to establish ahead of time a scheme powerful enough to meet the needs of all languages to be supported, but in our case we were fortunate enough to be able to import directly the work done for a business-oriented system at another division of our company. We have found, by the way, that the mapping algorithms used to accommodate German and Spanish have shown themselves adequate to support Arabic without any modifications to the code. Again we emphasize that by pushing language-specific knowledge out of the code we managed to save ourselves a good amount of engineering and support work.

Summary

Our conclusion has been that the problem of supporting processing needs for a variety of languages is best solved by borrowing the approach used to support terminals. The software itself is extended to support a more general model of processing ("language support") but the actual information used to support a particular language is moved out of the code and into the file system where it can be added ("localization") or removed at will. We would like to add a special plea for standards. The area we are talking about is uncharted, but the sooner we can agree on a solution, the sooner this world can be enjoyed by a much wider audience.

Bibliography

[1] Becker, J. "Multilingual Word Processing" Scientific American July 1984 pp. 96-107

[2] Tintel, P. "EURIX - a UNIX based system using European natural languages" EUUG 1984

[3] (no author listed) "Nihongo, realtime, network nado kino'o kakucho'o ga susumu UNIX" Nikkei Electronics 22 October 1984 pp. 171-208

System Aspects of Low-Cost Bitmapped Displays

David S. H. Rosenthal
James A. Gosling

Information Technology Center*
Carnegie-Mellon University
Schenley Park
Pittsburgh PA 15213

ABSTRACT

The design of low-cost bitmapped displays is reviewed from the perspective of the implementors of a window manager for UNIX.* The interactions between RasterOp hardware, the multi-process structure of UNIX, and the functions of the window manager are discussed in the form of a check-list of features for hardware designers.

1. Introduction

Chip manufacturers are announcing products designed to improve the performance and reduce the cost of bitmapped displays. Workstation manufacturers are marketing systems featuring UNIX and a window manager on such displays. The window manager has an overwhelming effect on the perceived performance of the workstation it is running on.

We have recently designed and implemented a window manager for 4.2BSD UNIX. It is intended to be easy to port between displays. It runs on the Sun workstation, and throughout its development we have been reviewing designs for other displays that are to be used in the future. In attempting to obtain the best performance from the Sun displays and remain portable to these others, we have encountered many interactions between display hardware features and window manager software; what follows is a

---

*UNIX is a Trademark of Bell Laboratories.

*The Information Technology Center is funded by IBM.

James Gosling's present address is Sun Microsystems Inc., 2550 Garcia Ave., Mountain View, CA 94043

March 17, 1985

compilation of our experiences.

2. Models

To provide a context for the discussion of these interactions, we set out the range of hardware and software we are concerned with.

2.1. Hardware

Workstation hardware typically consists of a processor, memory management hardware, memory, and I/O devices including a mouse, keyboard, and monochrome display. The display may have either:

* pixels visible in the CPU's address space, with RasterOps performed by the CPU (possibly with special hardware assistance).

* pixels not addressable by the CPU, but manipulated by an autonomous RasterOp processor. Communication with this processor is via registers or command queues in the CPU's address space.

2.2. Software

This hardware typically runs a form of the UNIX operating system, whose importance for this discussion is that it supports multi-process interaction. When multiple interactive processes compete for a finite real screen resource, arbitration is undertaken by some form of window manager, either:

* a part of the UNIX kernel, accessed via special system calls, or

* a special user-level process, accessed via interprocess communication channels.

The RasterOps affecting the parts of the screen resource allocated to each client may be performed by:

* The kernel, when the client requests them using system calls.

* The user-level window manager process, when the client requests them using remote procedure calls.

* The client directly, if it has the pixels or RasterOp processor control registers mapped into its address space.

The goal in all cases is to provide clients with a *protected* RasterOp, one that can affect only those pixels

allocated to the client.

3. Check-List

After providing memory to store the pixels, and a mechanism to generate the video output, the designer of a low-cost bitmap display is faced with the question of how much RasterOp support is required. With careful design, at least for the MC68000 (which can exploit auto-increment addressing), many cases of monochrome RasterOp are close to the limit set by display memory bandwidth even if they are done entirely in software. Next generation microprocessors typically have barrel shifters, making the alignment shifts of software RasterOps much faster. If special RasterOp hardware is to be cost-effective compared to a software solution, the following points should be considered:

1. Can user processes operate on the bitmap without system call overhead?

2. If special RasterOp hardware is provided, can a client access the bitmap without using it?

3. Can a client given access to the bitmap be prevented from accessing other IO devices?

4. If special RasterOp hardware is provided, can multiple user processes access it?

5. If the RasterOp hardware is an autonomous processor, does it support one or many command queues?

6. Can the RasterOp hardware be used off-screen?

7. Does the RasterOp processor implement clipping?

8. If the display has a color map, can it be shared between multiple windows?

9. Does the display support a cursor?

10. Can the display track the mouse autonomously?

11. Does the hardware allow non-rectangular RasterOps?

12. Can the display draw characters fast?

4. Discussion

Can user processes operate on the bitmap without system call overhead?

* Either the bitmap itself, or the registers and command queues controlling the RasterOp processor, or both must be capable of appearing in the address space of one or more user processes. In this way the process can do RasterOps directly; the cost of a system call (perhaps 0.3ms) per RasterOp is prohibitive.

If special RasterOp hardware is provided, can a client access the bitmap without using it?

* The presence of RasterOp hardware does not eliminate the need for the CPU to access the pixels directly. Unless the function set of the RasterOp processor matches the application requirements exactly, some display operations will need to be implemented in software. Inappropriate support that cannot be programmed around is worse than none.

Can a client given access to the bitmap be prevented from accessing other IO devices?

* A corollary of the need for user processes to address the bitmap is that the system's memory management and protection unit must be capable of controlling access to the I/O space at a relatively fine grain. It should not be necessary to trust a graphics process with access to the disk controller hardware.

If special RasterOp hardware is provided, can multiple user processes access it?

* If user processes can access the RasterOp hardware directly, its internal state such as source and destination coordinates, function codes, mask bits, and so on must be regarded as part of a process' state. They must in general be saved and restored across context switches; the performance impact of doing so can be severe (even if the hardware permits it).

* The overall impact of saving and restoring the RasterOp processor's state can be reduced if processes using it can be identified. The analogous problem for floating-point processors has traditionally been solved by initializing them to a state in which an attempt by the processor to use them will cause an interrupt. At that point the process is known to use the processor, and can be marked as needing the extra context. Thus, if the processor has any context to save, either the memory management unit or the processor itself should be capable of generating an interrupt on all access

attempts.

If the RasterOp hardware is an autonomous processor, does it support one or many command queues?

* Designs with autonomous RasterOp processors can reduce the need to save and restore RasterOp context by implementing several independent command queues and multiplexing these queues together. The window manager can then assign a queue to each window and treat them as if they had independent processors. A means to drain the queues before changing the shape or position of a window will be needed.

Can the RasterOp hardware be used off-screen?

* Many window managers require RasterOps that operate uniformly on rasters both on and off the screen. Off-screen rasters are typically in process virtual address space. If the RasterOp support cannot be applied to these rasters, the window manager will have to implement a software RasterOp even if it is not used on-screen.

Does the RasterOp processor implement clipping?

* A major role of a window manager is providing client processes with a protected RasterOp, that is in ensuring that a client can draw only within its assigned area of the screen. Thus, much of the window manager's processing is devoted to clipping. The window manager must apply a clipping rectangle to all client output, though within these limits the client may wish to impose a smaller clipping rectangle. Hardware support for clipping is useful, but it would be much more useful if it provided both a ''system'' clip rectangle that could not be changed from user mode, and also a ''user'' clip rectangle. Output would be clipped to the intersection of the two. The window manager would set the system rectangle to the window, and the client would set the user rectangle as desired. Even better would be clipping to the union of a set of rectangles for each mode, to permit clipping to partially overlapped windows.

If the display has a color map, can it be shared between multiple windows?

* Just as the pixels are a real resource to be shared among competing clients, so also are the entries in the color map. Clients should be able to use a number of different pixel values and corresponding color map entries without being aware that other windows are also using the color map. For example, all windows should

be able to use pixel values from 0 to some limit.

Hardware assistance for this sharing would be useful. The window manager should be able to specify a number of rectangles, in each of which the relationship between the pixel value and the color map entry it selected would be different. These rectangles might select from a number of independent full-size color maps, or provide a base register to be added to the pixel value before the color lookup (and perhaps a limit register to truncate the range).

Does the display support a cursor?

* An essential feature of a window system is a cursor, tracking a pointing device around the screen. The cursor can be displayed either:

  a) by temporarily changing some pixels in the bitmap from which the screen is refreshed, or

  b) by mixing the video from two separate bitmaps, one for the screen image and one for the cursor.

  The first is often preferred for low-cost systems; it requires little or no extra hardware but can impose significant performance loss. The problem is that if the cursor affects pixels in the screen bitmap, it must be absent during any RasterOp that affects those pixels. There are two approaches to ensuring that the cursor gets removed:

  * The process performing the RasterOp can handshake with the process maintaining the cursor before every RasterOp†. The cost of the necessary system call on every RasterOp, or at least on every RasterOp when the cursor is displayed, is very significant, and the cursor will flicker badly.

    This problem recurs in a milder form even if the process performing the RasterOps is also the process maintaining the cursor. The synchronization cost is less, but the cursor flicker is still present. It is particularly offensive because the cursor is typically the focus of the user's attention.

- ----------

† Ideally, it would do so only when the source or destination rectangles overlapped the cursor, but this is normally impracticable.

* If the video refresh controller is capable of interrupting when a specified scan line is reached, the interrupt routine can arrange for this to happen shortly before the first scan line containing the cursor. It can then put the cursor into the bitmap, wait at interrupt level until the refresh has passed the cursor, and then remove it. No user RasterOps can occur while the cursor is in the bitmap, and the cursor will not flicker, but the cost is the fraction of the CPU corresponding to the ratio of the cursor height to the screen height.

Video-mixed cursors are normally regarded as too expensive for low-cost displays, because they require either a complete second bitmap, or a smaller bitmap plus extra logic to position the cursor. But their effect on overall performance is so great that this may be a mistake. They should be considered in the design of advanced video-generator and controller chips.

Can the display track the mouse autonomously?

* Another major load on the system is tracking the mouse. Autonomous display hardware sufficiently intelligent to monitor locations in main memory for the mouse coordinates, and to position the cursor to correspond, would off-load significant processing. The off-load would be greater if it supported a ''clip'' rectangle for the cursor, and could interrupt if the cursor tracked across the boundary. Many window systems wish to change the cursor shape as it tracks across windows, or even regions within windows.

Does the hardware allow non-rectangular RasterOps?

* Non-rectangular RasterOps such as ''fill trapezoid'' can significantly improve the performance of applications using polygonal graphics, but need careful implementation. In particular, if they are to abut correctly it must be possible to save and restore the error parameters of the Bresenham or other algorithms tracing their edges. This is an example of the need for sub-pixel addressing, which also occurs in grey-scale and other anti-aliased applications.

Can the display draw characters fast?

* The overwhelming majority of all RasterOps will paint a character. The cost of these will be dominated by setup time unless the font contains very large characters, or the client sends long strings of contiguous characters. Note that single character RasterOps are important as echos of user type-ins. What-you-see-is-

what-you-get editors are major applications for this class of display, and their performance is dominated by re-painting characters as the user types.

Thus, the design of RasterOp engines should consider making character-drawing a special case, so that the hardware understands the font tables, knows the amount of shim space to add between printing and space characters, and so on. This spreads the setup time across as many characters as possible.

* Typical windows may contain characters from a large number of fonts; twelve per window is not uncommon. The stored fonts from which characters are drawn are frequently stored in the 224 by 1024 pixel off-screen area of an 800 by 1024 pixel display. Thus, specialized RasterOp hardware can be used to paint characters even if it can access only the bits in the hardware bitmap.

  Unfortunately, Parkinson's Law shows that this space is insufficient to store all the required fonts, so that the off-screen space can at best be a cache for active fonts, and cache misses will occur. The code to manage free space in the font cache, to gather usage information, and to perform cache reloads on misses is difficult to write, and imposes disturbing performance irregularities. Font definitions, and the individual glyphs, are variable size, adding to the normal problems of writing a pager.

  If the RasterOp hardware is not restricted to accessing the pixels in the hardware bitmap, but can access the whole of physical memory (even at reduced bandwidth), the problem is easier. The font cache can now be larger, stored in wired-down pages of system physical memory. But it is still only a cache.

  If, however, the hardware support for RasterOp is applicable even in process virtual address space the fonts can be stored in virtual memory, and the system's pager can deal with the problem of ensuring that the fonts in use are readily accessible. Virtual memory RasterOps are normally available only if the RasterOp is implemented by the CPU, either in software, microcode, or as a processor extension chip.

5. Conclusion

We have set out a number of points worthy of consideration in the design of low-cost bitmap displays intended to support multi-process interaction. Although RasterOp hardware may appear attractive, it needs careful design if its potential is to be fully realised, particularly for

drawing characters. Assistance with cursor drawing and sharing of the color map may be more cost-effective uses for limited hardware resources.

Acknowledgements

This work originates from insightful critiques of some proposed hardware designs given by Bob Sproull. Bob Sidebotham, Andy Palay, Fred Hansen and Bruce Lucas helped implement the ITC's window manager.

Anyone attempting to design bitmap displays should read Hardware/Software Tradeoffs for Bitmap Graphics on the Blit by Pike, Locanthi, and Reiser in Software Practice & Experience January, 1985.

A CONTRACTUAL MODEL OF SOFTWARE DEVELOPMENT

Vic Stenning

Imperial Software Technology,
London.

ABSTRACT

The technical process of software development - and, indeed, development of any complex man-made system - can be viewed as repeated application of a single step. Each individual step takes some existing representation (or mathematical model) of the desired system and transforms it into some more 'concrete' representation. This results in a sequence of representations leading from some very abstract model of the desired system to an actual implementation (i.e. a representation that is usable for the real-world application).

Any practical approach to system development must address not only the technical issues, but also issues of project management and data management. Thus the technical process outlined above must be incorporated into some broader project organisation. One possible approach is to employ a so-called 'contractual' model of development, whereby contracts are let internally within the project for the performance of the individual transformation steps and any other work that needs to be done.

Individual contracts can be fulfilled by the use of appropriate technical development and project management methods. Data management methods must be employed both within individual contracts and at the level of the contract hierarchy for a complete project. An integrated project support environment can support the chosen methods within the framework of the contractual model.

Introduction

Concern here is with systems that typically have the following characteristics:

- they are large

- the requirements are externally imposed

- the development timescale is externally constrained

- development therefore involves a team rather than a single individual

This should be contrasted with the situation of an individual programmer producing a small program to meet some personal need. The scenario outlined above is not just quantitatively different, it is also qualitatively different. Not only are the problems larger, but also there are entirely new problems. These problems include:

- the determination, from external sources, of exactly what the system has to do. (Typically these sources will provide information that is inconsistent, ambiguous and incomplete)

- the achievement of effective communication and cooperation between the various members of the project team

- the achievement of effective interaction and cooperation between distinct components of the complete system

Further, it should be recognised that any complex system must evolve if it is to continue to operate effectively in a world that is constantly changing, and that provision must be made for supporting this evolution. This need to be able to change existing systems leads to a number of well-known problems that are often referred to euphemistically (and incorrectly) as 'maintenance' problems.

In addressing these problems there are fundamental principles. These principles are just common sense, but unfortunately they are applied all too rarely in practical projects - partly because "there's nowt as uncommon as common sense", and partly because the proper application of these principles raises difficult technical and logistical problems. The common sense principles are:

(i) to take small manageable steps, checking carefully at each step, rather than attempting some great leap into the unknown;

(ii) to be explicit and precise on every aspect of development, rather than relying on implicit or ambiguous assumptions;

(iii) to maintain full records of all aspects of the development, to be consulted (and of course supplemented) when making subsequent changes.

These principles underlie the process and contractual models that are described in subsequent sections.

The Process Model

The process model reflects a particular view of the nature of the software development process and the needs of that process. (The term 'development' is used consistently to encompass not only initial development prior to product release but also subsequent 'maintenance' during the operational life of the product).

The model recognises that the distance between the original concept of some desired system and the eventual implementation of that system is too large to be bridged in a single step. Therefore a sequence of intermediate representations is employed. Each of these representations is the result of a single step from its predecessor representation towards the final implementation; the individual steps are sufficiently small to be performed with some confidence. Each representation is expressed using notations and abstractions that are appropriate to its position in the sequence. Thus the early representations will use the terms of the application domain and will be expressed in a language suitable for modelling of that domain. By contrast, the later representations will use the terms and notations of computer systems implementation languages.

Each representation is derived from its predecessor by means of transformation. As each new representation is produced, it must first be shown that the representation is internally consistent. Following this, there are two quite distinct concerns. First, is the new representation correct with respect to its predecessor(s)? Second, has real progress been made towards the goal of an implementation that meets the needs of its users? These two concerns are addressed by verification and validation respectively. By 'verify' we simply mean 'increase confidence in the correctness of'; any appropriate technique may be employed, with formal (mathematical) verification neither being explicitly required nor explicitly excluded. By 'validate' we mean 'increase confidence in the appropriateness of'; relevant techniques include inspection, reviews, animation and prototyping. Note that verification must always be with respect to an earlier representation - it cannot be performed in isolation - and is therefore essentially 'backward looking'. By contrast, validation can be performed in isolation, and is essentially forward looking.

Despite the verification at every step, errors can still be made. Similarly, it is possible to follow an ill-advised design path, even though every representation is validated before proceeding. Whenever such problems are discovered, while working on some later representation, it

is not sufficient just to make appropriate changes to that later representation in order to correct the error. Rather, the designer must backtrack to the representation at which the error was introduced (which may be the very first one in the sequence) and make appropriate changes to that representation in order to correct the error. These changes must then be propagated through all subsequent representations, again with verification and validation at every step. Thus the entire process is iterative.

Of course, changes in the system requirements are inevitable, both during initial development and during the operational life of the product. Such changes in requirement are handled in the same way as errors, that is by backtracking to the appropriate representation followed by change propagation.

Scope of the Process Model

The process model provides a framework that directly addresses some of the concerns of the introductory section. In particular, it provides a basis for systematic initial development, with small steps and incremental checking, and it suggests that 'maintenance' activities during the operational life of the system are no different from initial development activities. Indeed, with this model the activities of backtracking, introducing a change and propagating that change are in a sense the norm, with original creation of some new representation being a rather special case.

Besides providing a useful framework for technical development activities, the process model also gives some indication of the data management needs of a project. Consider, for example, the relationship between successive representations, and the impact of backtracking and change propagation (which of course creates new 'versions' of the affected representations). However, the simple process model by no means addresses all aspects of data management - for example, there is no basis for considering the internal structure of an individual representation - and in no way addresses project management. Thus there is a need for a broader model, compatible with the process model, that allows the three key issues of technical development, project management and data management to be addressed in a consistent fashion. This is the purpose of the contractual model of development.

The Contractual Model of Development

As the name suggests, the contractual model is based upon the notion of identifiable contracts. (It should be noted that the term 'contract' is not used here in any legal sense; it simply means a documented agreement between two parties, typically within the same project team). The

essence of any contract is a precise specification. Within the model this specification is recognised as having three major components: a technical specification, a set of management constraints, and acceptance criteria. The technical specification serves to precisely define the required deliverable from the contract. The management constraints define the bounds within which the deliverable is to be produced - costs, timescales, standards, and so on - and also define the obligations for reporting from the contractor to the client. Finally, the acceptance criteria define objective measures which can be used to judge successful completion of the contract.

Upon completion (i.e. when the acceptance criteria have been satisfied) the contract returns a deliverable that meets the specification. However, during the course of the contract a variety of reports - progress, problems, queries, and so on - will flow between contractor and client. Thus the full contractual interface recognises three main elements: a specification, a deliverable, and a set of reports.

The power of the contractual model lies in the fact that it is recursive. Suppose that the task to be performed on a contract is dependent upon various major sub-tasks, and that each of these sub-tasks can be precisely defined. It is then possible to let sub-contracts for the completion of these sub-tasks, the sub-contracts may themselves let further sub-contracts, and so on. Thus, within a single project, which is itself treated as a single contract, there may be an extensive contract hierarchy formed on the relationship 'is a sub-contract of'.

[It should perhaps be emphasised that the contract hierarchy within a project does not simply reflect some hierarchical decomposition of the final product. Recalling the process model, contracts may be let for the production of a statement of requirements or a system specification. These may lead to sub-contracts for feasibility studies, production of a prototype, independent review, or whatever. Only in the later stages of a project would contracts call for the production of components of the final product, and even then there need be no direct correspondence between the contract hierarchy and any product decomposition hierarchy. Thus, while the contract hierarchy can readily accommodate product decomposition, it has a far wider scope and purpose].

Use of the Contractual Model

Properly employed, the contractual model provides a basis for addressing the three key concerns of technical development, project management and data management.

For technical development purposes, the contractual model is entirely compatible with the process model. Individual contracts can be let for the production of each representation in a sequence, with the relationships between these representations being maintained within the client contract. Sub-contracts may be let as appropriate for carrying out feasibility investigations, verification, validation and so on. Within the contractual structure a wide variety of specific technical methods could be employed, ranging from the formal to the structured (Of course, the nature and notations of the specification and verification criteria for a contract must be compatible with the methods employed).

The contractual hierarchy provides the coarse structure for management of the project, and the contractual interface shows the nature of management at this coarse level. A sub-contract is specified precisely both in terms of the technical task to be performed and the management constraints and reporting obligations that apply. Within the limits imposed by the specification, the sub-contractor is free to organise and partition the work in any manner that seems appropriate. Again, within an individual contract a variety of specific management methods could be employed. Progress, and any problems that may jeopardise fulfillment of the contract, must be reported back to the client. Responsibility for determining the strategy for dealing with such problems rests with the client, not the contractor.

Similarly, the model provides a basis for overall data management, in that the main data units to be managed are precisely those that form the specifications and deliverables for individual contracts. There may also be finer-grained data management concerns, but with the contractual model the control of specifications and deliverables is of primary importance.

An Integrated Project Support Environment

In conjunction with British Telecom, Imperial Software Technology is currently developing an integrated project support environment based directly upon the contractual model. This environment is designed to operate on a distributed network of computers of various sizes, including personal workstations.

The environment consists of a 'framework' that provides a database system, a user interface and tool integration facilities, plus an extensible set of tools. The initial toolset is sub-divided into four classes, namely office automation, project management, technical development and data management. The office automation tools provide the normal facilities - text processing, mail, bulletin boards, diaries and so on. The project management tools provide a

conventional set of co-ordination, planning and monitoring tools directly related to the contract model. The technical development tools support certain established methods - CORE for requirements analysis, an SDL-based method for specification of concurrent systems, VDM for the specification and refinement of abstract data types - within the contractual structure. Finally, the data management tools support integration and build, library control and version management.

It is the contractual model that integrates this environment, thus justifying the 'I' in 'IPSE'. Different users of the environment could easily adopt their own preferred methods for project management, technical development or data management, and introduce new tools to support these methods, without invalidating existing tools or compromising the overall structure for the IPSE. Further, new technical development tools can be employed with existing management tools, and vice versa. Thus the use of the contractual model as the basis for the IPSE would seem to provide a desirable combination of stability and flexibility.

TeX+ must eventually replace nroff/troff

Timothy Murphy

School of Mathematics
Trinity College Dublin

as the standard UNIX* text-formatter. For the output of TeX is an order of magnitude superior to that of troff. Thus

(1) TeX reads a whole paragraph before deciding where to break the lines.

(2) In TeX the space between 2 letters depends on both, in troff only on the first;

(3) The spacing around mathematical symbols in TeX depends on the "function" of that symbol: binary operator, relation, left parenthesis, etc;

(4) The size of matching parentheses in a mathematical formula is automatically adjusted in TeX to the height of the expression they enclose.

But if TeX is to be integrated into UNIX, it must undergo major surgery:

(A) TeX must be re-written in C, to allow reasonable interaction with the environment.

(B) TeX should be "modularised", so that eg the hyphenation module can easily be replaced for non-English input.

(C) There are 5 phases in TeX: syntactic analysis, semantic analysis, paragraph building, page building, and output driver. These could run as different processes, or else be overlaid.

(D) An option should allow TeX to use temporary files whenever possible, eg for storing rare macros.

(E) An interactive option could show input and output on different screens, or windows. As soon as a paragraph (or displayed formula) had been typed in, it would be processed and output. But the input would be held in a buffer, so that if modification were desired the paragraph could be edited and re-processed; and only when the user was satisfied would the input be entered, and the next paragraph begun.

+TeX is a Trademark of the American Mathematical Society.
*UNIX is a Trademark of Bell Laboratories.

March 17, 1985

UNIX at IRCAM

Robert Gross, Michele Dell-Prane, Dan Timis, and David Wessel

Since 1976 IRCAM*, has supported research on fast real-time digital signal processing, room and instrument acoustics, psycho-acoustics, compositional algorithms, as well as different methods of sound synthesis including the singing voice and physical modeling. The results of this research have been applied by invited composers in many contemporary music pieces with real-time digital electronics and computer generated tapes.

IRCAM has developed a series of high speed digital signal processors culminating with the 4X machine recently used by Pierre Boulez in his work 'Repons'. General purpose computing and program development is done on a group of VAX, SUN, Plessey, and Valid computers networked together. Work with artificial intelligence and expert systems has affected a majority of the current research projects.

Two years ago UNIX became the underlying foundation for all the research and musical production at IRCAM. UNIX must not only drive the 4X and support standard program development including numerical computation, and compiler design but also must do musical sample computation, storage, and real-time playback or record operations. This latter problem is not easy to deal with in UNIX and has been solved by using the advantages of UNIX files to simulate a hierarchical sound-file system based on the UNIX model. We are currently using the CARL** sound-file system developed at UCSD.

This talk will explore some of the current development projects at IRCAM including the 4X software design, Acoustical research, Chant/Formes project, applications on the array processor, and the development of music oriented software for the Apple Macintosh using the SUMACC's*** environment. We will also discuss our experience with the CARL software and our attempts to utilize the 4.2 BSD file system to support the I/O throughput required by the DA/AD conversion.

*Institut de Recherche et Coordination Acoustique/Musique
**Computer Audio Research Laboratory
***Stanford University

Published by the European UNIX® Systems User Group,
Owles Hall, Buntingford, Herts SG9 9PL.
Tel: Royston (0763) 73039.