MODIFYING UNIX TO SUPPORT 8-BIT CHARACTERS

George Drettakis, Vassilis Prevelakis

RCC/CSI/TR/1987/005

Computer Science Institute
Research Center of Crete
Modifying Unix to support 8-bit Characters.†

George Dreuxakis,
Vassilis Prevelakis.

Institute of Computer Science,
Research Centre of Crete,
Heraclion, Crete.

ABSTRACT

This article discusses the issues and experiences from the modification of Unix to support mixed Latin/Greek text. The project was carried out at the CS Institute of the Research Centre of Crete (RCC). The Institute, founded several years ago, participates in many international research projects and in programs for the development of Computer Science and its applications in Greece. The environment selected for research and development is the Unix operating system, as it is widely used on a worldwide basis in such institutions. One of the first tasks of the staff at RCC was to adapt the Unix environment to local Greek needs, allowing the use of tools and utilities with mixed Greek/Latin text. We present a detailed description of the attempt, the decisions made, and the solutions adopted.

1. Introduction.

The objective of this effort was simple: to support Greek and Latin text simultaneously in a uniform manner, allowing full use of all Unix programs and utilities. This goal presented problems many of which were totally unforeseen.

The first and most basic issue was the internal representation of Greek characters that was to be adopted. Two main tendencies exist on this matter. The first is to use the same internal representation for both Greek and Latin characters with control characters used to switch between the two, burdening the I/O devices with the task of switching between two states; a Greek mode in which characters typed and displayed appear in the Greek alphabet, and a Latin mode in which the characters appear in their normal Latin character set. This solution was considered inadequate as difficulties appear when handling mixed Greek/Latin text, especially in the case of interactive programs such as text editors.

The second tendency, the one adopted, is to take advantage of the fact that internal representation of the full English alphabet and punctuation marks uses the codes from 0 to 127. This means that only 7 bits are required for their internal representation. As most modern computer systems use 8-bit bytes for character representation, the subsequent 127 free internal codes can be used for the modern Greek alphabet.

This decision to use all 8-bits proved to have one serious repercussion. A large number of Unix programs take in account the fact that all 'known' characters fit in a 7-bit byte and as a consequence use the eighth bit as a cell of information for their own use.

The problem, therefore, was to find a solution that would allow the use of mixed Greek/Latin text with all Unix programs and utilities.

First attempts concentrated on determining the source of the problems described above. The source code of the Vax 4.2 BSD Unix distribution, available with a source license of the Institute, was carefully

† The work reported in this article was partly funded by the CR-offices project of NATO SPS through the Greek Ministry of Research and Technology.
examined to locate the sections that do not support 8-bit characters. It was discovered that this appears both in the Unix kernel and the utilities and tools. Subsequent releases of UNIX (e.g., UNIX 4.3 BSD, System V) alleviate some but not all the problems of 8-bit representation mentioned in this article.

1.1. The Unix kernel.

The Unix kernel in general allows full 8-bit data to flow freely. Files are handled as eight-bit byte streams, the obvious reason for this being that executable files are handled in the same manner as text files on Unix, so a distinction would be impossible. In particular no attempt to compress file storage is attempted. The issues to be dealt with in the kernel are in the terminal I/O code and the directory naming scheme.

1.2. Unix utilities.

Unix utilities are a set of tools that perform a large number of tasks to help the programmer and user. Traditionally most Unix utilities are filters. In this case the standard input is read, a transformation performed upon it and the output appears on the standard output. The standard input and standard output are normally associated with the users terminal, but the I/O redirection facilities of Unix allow them to be connected to files. Greek characters can be used with these programs in a fairly straightforward manner. All that is necessary is to write two filters. One that maps Greek characters to 7-bit data and another that restores 7-bit Greek characters to their 8-bit representation. Data is filtered through the first, passed through the original program, and then recovered from the second filter. This solution impairs efficiency somewhat, as a 7-bit representation of Greek characters requires some form of 'escaping' and data has to be processed by two additional programs, but as most filters are not interactive, response time is usually of minor importance.

Other Unix utilities require some interaction with the user. These include text editors, window systems, data base systems and interactive programs in general. These do not allow the solution of filters and mappings, as mixed Greek/Latin input and output may be required from the program. An example would be a user writing a letter in Greek, who wishes to include an English name in the text. Switching between the two alphabets is required in more than one places. Initially the terminal would have to support the shift, the program would then have to store the character internally in a correct form and echo the character in the correct alphabet. A 7-bit solution to these problems, as in the case of filters, is simply inapplicable. The only way to allow the use of these programs with 8-bit characters is to modify the source code.

In section 2, general considerations concerning the modification process are presented together with issues dependent on the nature of the C programming language. A detailed description of the kernel modifications follows in section 3. Finally the Unix utilities modified are discussed in section 4 and concluding remarks of section 5 end this article.

2. General considerations for modifying Unix to support 8-bits.

Unix is a family of usually small programs that allow complex and occasionally time consuming tasks to be performed with ease. The full set of these programs is large and a product of many man years of software development effort.

Both the kernel and the utilities have been tested and used by a large community of users over a long period of time. Most programs have been fully debugged in this process and have a stable form that has evolved over the years of use and changing needs. Many of these will not support eight-bit characters, for the reasons described above.

There are two possible approaches to overcome this difficulty: rewriting the programs from scratch or modifying the code to support 8-bit characters. The decision depends on many factors that vary depending on time and money constraints and, for an environment such as the one at Crete, on the existence of staff with Unix expertise.
2.1. Rewriting the software from scratch.

The first solution considered was to rewrite the programs that would not support 8-bit characters. The arguments against such an approach are many, some of them obvious. Time constraints make building a large program from scratch prohibitive. Rewriting the full set of Unix utilities is practically impossible, and would require many hundreds of man years.

However, there are cases in which rewriting the software may seem reasonable. Examples would be applications that require embedded knowledge of the Greek language such as spelling checkers or even sorting in which the diacritical marks used in Greek present a problem [Hull].

Another factor that contributed to our decision was local lack of experience. The original authors were either the inventors of Unix themselves or highly talented people in an environment that supports and helps such development. Although expertise is growing rapidly in our community, in-depth knowledge is still limited. Even if such programs where to be written, some time would have to elapse to achieve the quality of existing software. They would have to be tested and it is doubtful if it would ever be possible to reach the robustness that the existing Unix programs have attained over the years.

2.2. Modifying existing code.

The general availability of the Unix source code makes modifying existing software an attractive prospect. Yet, the decision to attempt such modification presents difficulties that surprisingly result from the availability itself.

Although Unix was initially developed in a research environment, many widely accepted software engineering practices have been overlooked during its evolution. Documentation is sparse and in places non-existent. The programming style is sometimes archaic, although this is partly due to the age of the programs, some of which date from the early seventies. One final but important factor that is obvious in many Unix programs, especially the Berkeley releases, is the tendency to develop in an ‘add-on’ manner. Programs designed to do a particular task have been modified to support changing needs and ‘features’ totally different from the intentions of the initial design [Kern].

The above reasons result in code that is often obscure, badly or hastily written and occasionally with little or no documentation. All the issues described make modifying existing Unix software a difficult and in many cases tedious task. This may come as a surprise to many as Unix is known for its careful design and its bias towards elegant solutions vs. efficiency [Bell]. The article does not dispute these qualities of the system without which the whole project described would have been impossible. The points noted here simply show how a task that should have been easy proved to be exceptionally difficult.

A method was required that would allow our community to make full use of all the useful tools of Unix in a reasonable time period, allowing the development of applications and other work with Greek/Latin text.

2.3. Problems due to the C language.

Unix and its utilities are written using the high level systems language C. The difficulties of our task presented in this section result from certain features peculiar to the language and also, in certain cases, from issues of bad programming style.

It is interesting to note here that the power of C is partly due to its lack of type checking and the ease in which low level operations may be performed. Many of the issues described stem from these characteristics and would possibly have been avoided had a more strict language been used. On the other hand one must not forget that it would have been totally impossible to write large parts of the Unix code without these features of C.

As mentioned before, the problems encountered are a consequence of the use of the eighth bit by programmers in Unix. A typical example of this method is the storage of some specific information concerning a character in the same byte, together with the character itself. In C this is implemented using the bitwise operators ‘&’ and ‘|’ (bitwise and, or). It is usual to define a constant that contains a ‘1’ in the bit that is to be used as a flag. For example the eighth bit (using octal for historical reasons) would be defined in C as:
#define QUOTE 0200

When specific information, designated by the eight bit being '1' in a character, is stored in a byte, the following or operation would be performed.

    char c;
    ...
    c |= QUOTE;
    ...

The information stored in this manner can then be retrieved by performing an and operation as follows:

    if ( c & QUOTE ) {
    ...
    }

When an or operation as the one described above is performed the eighth bit in a byte is forced to be '1'. Operations of this kind can have unexpected results. If the variable 'c', in the above example, initially had the value (latin) 'C', 'orring' with the constant QUOTE would convert the value into (greek) 'beta'. Conversely if the operation

    c &= "QUOTE;

is performed the character initially having the value 'beta' will end up having the value (latin) 'C'.

This method of storing information accounts for the largest part of the problems we were forced to deal with. The most general solution proposed and applied was simple. Instead of storing characters in the one byte char type of C, characters are stored in the two-byte type short. The bit containing the information is moved to the 15th position by changing the constant definition. In this manner the code described above would simply become:

    #define QUOTE 0100000
    short c;
    ...
    c |= QUOTE;
    if ( c & QUOTE ) {
    ...
    }
    c &= "QUOTE;

As is obvious from above, this solution allows 8-bit modifications to be performed without changing the code itself. All that is necessary is to modify the declarations of variables and constants. In theory, this should mean that no further modifications would be necessary to the code at all.

In reality a number of other problems appeared. In many places the eighth bit is simply stripped, that is anded with zero for no apparent reason. These operations were results of programming sloppiness as they made use of absolute constants bypassing the symbolic values declared. If a more careful style had been adopted the problem would have been solved by the modifications described above. As an example, the following statement occurs several times in Unix code:

    c &= 0177;

... If a symbolic constant had been used the change would have been made in the declarations only. The cases where such brute-force operations where performed were thankfully limited.

Apart from simple character variables many text manipulation programs use arrays. The use of arrays of short integers in place of characters required the addition of certain standard character manipulation functions. The standard C library string handling routines had to be replicated to handle arrays of short instead of arrays of char as they do now. A set of conversion routines also was necessary to perform copying of strings represented by char to strings represented by short and vice versa. The full collection of standard macros such as isalpha( c ) that returns whether a character is alphanumeric, was replicated providing such macros as isalphaG( c ) that returns whether a character is a Greek alphanumeric character.
Finally a problem due to C language semantics proved to be irritating. In expressions char type variables are promoted to type int during evaluation [K&R]. If, for example, we have a segment of code:

```c
char c;
...
if (c < 0) {
  ...
}
```

the char variable `c` will be promoted to type int that is stored in four bytes. In the process, sign extension will take place if the eighth bit is '1'. This results in all Greek characters being negative (!) producing indiscernible bugs. The solution given was to use the C type unsigned which, by definition is never negative.

The approach of modifying existing software was adopted in most cases, after considering all the problems discussed above. This decision was applied to many programs in the Vax kernel and utilities, and also to software available for the Sun workstations such as the window system and relevant applications.

3. The Unix kernel.

The kernel is one of the most complicated and difficult to understand Unix programs. Modifications to the kernel are also difficult to implement as they involve lengthy recompilations and a system reboot every time a new kernel is to be tested.

Fortunately, making the Unix kernel compatible with 8-bit terminals involved changes to two localized areas, namely the terminal I/O drivers and the directory blocks in the file system. The first change enables Unix to support 8-bit terminals and was crucial to the success of the project. The second (in conjunction with a Greek shell) enables the user to use 8-bit filenames.

3.1. Changes to the BSD 4.2 kernel

The modifications to the I/O drivers were done by brute force: all the queues originally implemented as arrays of char were converted to arrays of short. In this way when the driver uses the most significant bit in each character as a flag, the 16th bit is used leaving the 8th bit to be used as part of the character encoding. The implication of such a policy is that when a new terminal controller is attached to the system (e.g. the DEC dhu11 controller) the driver for that controller has to be modified as well. Although easy to describe, the above changes were quite difficult to implement.

Apart from the general lack of documentation (in the form of accompanying documents or in-line comments in the programs) the code in the terminal drivers themselves is totally opaque. The reasons are numerous and, in some cases, well founded:

- Since I/O takes up a sizable portion of system time, efficiency, rather than readability was the prime design consideration. For example pointers are used to a large extent, to avoid copying data from one buffer to another.
- The same code has to support a variety of low-level device-dependent drivers along with code for the pseudoterminals. This results in many calls being routed through vector tables and is responsible for constructs like:

  ```c
  (*cdevsw[major(tp->l_dev)].d_stop)(tp, 0);
  ```

- Despite the impression given by the separation of code into files grouped in subdirectories, there are many interdependencies in the driver software. Changing the size of one element in the file structure had wide repercussions necessitating changes in files residing in three different directories. Even the machine specific low level kernel source code file, `locore.s` (written in assembly) had to be modified as the direct memory access transfers between the terminal controller and the main memory had to map bytes (received by the controller), to arrays of shorts (the queues used by the driver).
3.2. Changes to the BSD 4.3 kernel

This latest Unix release from Berkeley goes a long way towards supporting 8-bit terminals. Most of the low-level drivers allow 8-bit characters through. However the driver, as delivered by Berkeley, does not solve the problem of supporting 8-bit terminals under BSD Unix. To see why this is so we must first examine the reason for the Berkeley enhancements in the new driver.

It is quite common to use serial lines to connect computers to other computers (what is commonly known as host to host communication). Programs on both hosts carry out some sort of software handshaking which enables the computers to exchange data. These programs use their own error correction schemes and go to great lengths to prevent the computers from generating a parity bit for the data sent, so that the full 8 bits of the transmitted character can be used to carry data useful to the programs.

In earlier BSD versions such programs had to use RAW mode which has always provided 8-bit I/O. In RAW mode the computer not only suspends all input or output processing, but does not perform any buffering either. This forces the communicating programs to do their own buffering at a great cost:
- the operating system must arrange for the user-level communications program to be restarted for every character received. This results in a lot of needless context switching and waste of CPU time.
- the communications programs themselves are made complex by the buffer handling routines that have to be added to them.

The new driver provides a flag (PASS8) that, in conjunction with an existing flag (LITOUT) enables both serial input and output while the driver is in a mode that supports buffering.

The PASS8 flag enables 8-bit input while the LITOUT flag suspends all output processing including parity generation.

In the context of 8-bit terminals the PASS8 flag is extremely significant in that, for the first time in BSD Unix systems, a terminal can supply 8-bit data to an unmodified shell (the standard command shells in BSD Unix cannot work in RAW mode).

Unfortunately the LITOUT mode has many undesirable side-effects. By suspending output processing the driver does not perform many necessary conversions (e.g. tabs to spaces, newlines into carriage-return plus line-feed etc.).

To make matters worse the driver strips the 8th bit when echoing typed in characters. So even with the PASS8 and LITOUT flags set the user cannot employ an 8-bit terminal.

Like 4.2 BSD, kernel changes had to be made, but to a limited extent, as some of the nastier work on the tty driver had already been done at Berkeley.

Our objective was to provide Greek terminal support without changing the behaviour of the documented modes of the tty driver. To do this, we introduced a new flag (called GREEK). The function of the new flag is similar to that of the LITOUT and PASS8 flags in that it enables 8-bit I/O. The main difference is that it allows the tty driver to perform output processing and it preserves the 8th bit during echoing.

The problem with introducing a new mode was that there was no space in the new tty structure for the extra flag. Instead of extending the structure by adding an extra word for flags, we decided to substitute the least useful flag (in our opinion at least) with the GREEK flag. We chose the TILDE flag because we believe that practically all the terminals that are in wide circulation can display the full ASCII character set.

The areas where the TILDE flag was used, were identified and isolated by means of conditional compilation constructs. The GREEK flag was introduced in the the /sys/h/ocvt.h file and the tty driver was modified so that it now recognises the GREEK flag.

From the user level programs we only had to modify the sty utility so that the following command

```
% sty -greek
```

(entered to the command interpreter) enables the GREEK flag, and

```
% sty -greek
```

disables it, making the driver compatible with the standard BSD 4.3 driver.
From the user point of view these modifications provided a reliable 8-bit system where work for converting the Unix utilities could take place.

3.3. Making binary-only distributions 8-bit

When the first SUN workstations were delivered to our Institute we were faced with a very difficult problem. How could the 8-bit changes we had made on the VAX be ported to the Suns when:
- we only had a binary-only distribution
- Sun kept producing new releases for their BSD 4.2 derived operating system

The solution hinges upon the fact that Sun Unix binary releases allow the user to attach his own device drivers. So we produced a logical device driver (based on the BSD 4.2 pseudo-terminal device driver) that can be plugged to any BSD 4.2 derived release of Unix.

Before we can proceed with the description of the new device driver a short discussion of the function and implementation of the BSD 4.2 pseudo-terminals follows.

Pseudo terminals were really constructed to support window managers, where the terminal screen is broken into windows and within each such window, the user runs a separate shell. The problem with this approach is that many programs (such as screen editors) will not work unless their standard input and output is a terminal. Even programs such as Is behave differently depending on whether they talk to a pipe or a terminal. This is where pseudo terminals come in. These new devices are logical devices (i.e. they are simply objects within the kernel and do not drive any hardware) but they retain all the characteristics of the standard terminal drivers. Thus user level programs cannot easily determine whether they are talking to a real terminal or a pseudo-terminal.

Since real terminals and pseudo terminals provide almost the same services, the implementors of the pseudo-terminals retained the upper-half of the terminal device driver and simply wrote a set of routines implementing the lower half of the pseudo-terminal driver. Thus real and pseudo terminals share a large part of the code.

Our approach was to completely separate the two types of drivers and produce a standalone pseudo-terminal driver that was 8-bit (in fact it was the pseudo terminal code from the VAX) and install this on the Sun kernels. The new 8-bit pseudo-terminals replaced the standard pseudo-terminals on the /dev directory, so that programs (such as rlogin) that use pseudo terminals use the 8-bit variety by default.

By modifying the shelltool program (the program that handles the terminal emulation under the Sun window manager) the Sun workstation user can have windows emulating 8-bit Greek terminals on his screen. The modifications to the shelltool are discussed later on.

3.4. Greek on 7-bit terminals

Standard ansi terminals (such as the vt100e) are inherently 7-bit. To be able to support Greek on terminals of this type the character set ROM in the terminal must be changed putting Greek characters in the alternate character set positions (these usually contain mosaic type graphics symbols). A user-level program is then placed between the user shell and the terminal driver.

During output to the terminal, this program maps 8-bit characters into 7-bit characters plus escape sequences. For example if a file contains

```
latin latin ELLHNIKA ELLHNIKA latin latin
```

these would be mapped to

```
latin latin <SI> ELLHNIKA ELLHNIKA <SO> latin latin
```

where <SI> and <SO> are the ASCII shift-in and shift-out characters. In fact, any escape sequence may substitute the <SI> and <SO> codes. The escape sequence appropriate for the particular terminal, can be stored in the /etc/termcap file.

Input from the terminal is slightly more complicated as the front-end program has to convert 7-bit codes into 8-bit. An escape sequence from the terminal (again defined in the /etc/termcap) file can be used by the user to toggle between Greek and Latin modes. This escape sequence can be generated by one of
the function keys on the terminal keyboard.

In Latin mode the characters would be sent to the user program without any processing. In Greek mode, the ASCII characters will have to be mapped to 8-bit Greek characters according to a standard mapping table (e.g. f should be mapped to ø, g to γ etc.). Echoing will be done by the user level program so that Greek characters are echoed when the user is in Greek mode.

A special problem is presented by the dead keys used to produce emphasis and diacritical marks. When a user presses a dead key the appropriate mark should appear on his screen to indicate that the character has been received [Hull]. When the user presses a phoneme however, the mark on the screen should be erased and the composite character (phoneme plus mark should be displayed). This can be done by sending the RUBOUT character for that terminal (again acquired by reading the /etc/termcap file) and then the composite character. This approach could have problems in certain situations where the character is at a margin boundary and the terminal is of the brain washed variety.

Some people would argue that the function of the front end program should be performed by the tty driver. We do not share this view for the following reasons:

- efficiency is not a serious concern as the terminals are attached to the VAX over slow (9600 baud) serial lines.
- these additions would make the tty driver larger and more complex. Considering that the driver is already quite difficult to understand, debugging would be a nightmare.
- the program needs information about the terminal. In the above discussion it was supplied by the termcap file. The kernel of course, would need to have its own (built in) terminal information table. This would force a system reconfiguration when a new type of terminal was connected to the system and even when the existing tty line allocations had to be changed.

4. Unix utilities.

Our team aimed at providing a minimum working environment that would support Greek/Latin text. After completing the necessary kernel changes, this required modifying a small number of Unix utilities to allow people to perform everyday tasks. As the modified system would be used mainly for research and development, the main interest was to provide text editors, some form of formatting, the window environment for the Sun workstations, and the command interpreter csh.

4.1. The editor `ed' and the screen editor `vi'.

The basic Unix editor `ed' was easily modified to support 8-bit characters. This is a line editor and certainly not adequate for the needs of the Institute. As most people used the screen editor `vi' to perform text editing tasks it was decided that modifying the editor was an imperative.

The editor is organised in a "two-level" manner. The first "level" is an extended line editor that is very powerful and supports regular expressions, substitution and searching etc. This section - also called "ex" - uses the eighth bit in the regular expression algorithms and also uses the information in the eighth bit to display certain characters in a special manner depending on the context. An example of this use is the display of control characters. Ex has a special state in which control characters are visible called list mode. In this state control characters are passed to the output routines with the eighth bit set to ‘1’ and are displayed as an escape sequence such as ‘A’ or ‘B’, or special characters such as ‘S’ for newline. This information is not stored for further use in ex.

The second level called visual handles the full screen capabilities of the editor. In this level the screen is stored as a two dimensional array of characters. In this array certain characters are 'quoted', that is they have their eighth bit set to '1', the information being stored to be interpreted later on.

What we required was a method to allow the editor to support 8-bit characters with minimum modifications to the rest of the code. The method selected was the one already applied successfully to the tty driver code. The screen representation was doubled and now consists of a two dimensional array of short integers, as described in the general section above. Unexpected problems from bad programming style were encountered in a few places. The Unix environment and its debugging tools allowed these problems to be solved with comparative ease.
4.2. Other programs: curses, nroff/troff.

The library ‘curses’, that supports device independent cursor motion on CRT terminals was also modified to accept eight bit characters. As the library shares a substantial amount of code with ‘vi’, the modification was comparatively easy. In ‘curses’ the eighth bit is used to distinguish between characters with normal display mode and characters in reverse video. The solution of making the internal representation of characters an array of two byte shorts was again chosen to solve the problem.

Typesetting and text formatting in general is considered one of UNIX favourite features. Somewhat sadly these programs do not support eight-bit characters. The source code of these programs use the eighth bit in many places in a somewhat indiscernible manner. They are old programs, and have been developed with complex typesetting/formatting device characteristics in mind. The result is a large and unmanageable piece of software. For these reasons no attempt was made to modify the source code.

As an alternative the already existing support for mathematical fonts by nroff/troff was chosen to support Greek typesetting/formatting. A mapping between eight-bit Greek characters and the special fonts was constructed. When text is to be formatted it is first filtered through a program that maps 8-bit representation to the troff input specification for the special fonts. For example, the Greek character ‘ο’ is mapped to the sequence \(\alpha\) in the troff input language. Diacritical marks above vowels in the Greek alphabet are also handled by the filter using the typesetting capabilities of troff that allows arbitrary positioning of punctuation marks. The troff output is then sent to the output device. This method has acceptable results with the use of a PostScript laser printer facilities that provide high quality Greek fonts.

For nroff, the low-resolution counterpart of troff, a similar method was used. In this case the mapping had to be reversible. The reason for this being that low resolution devices (terminals, dot-matrix printers) use eight bit representation for Greek characters. All the filters described above were written using lex and in some cases a modified lex that supports 8-bit characters.

Another alternative was to use the driving tables that are provided by nroff and to use the same mapping and filters for troff, using the eight-bit Greek chars in the place of a special font. Both solutions have been applied successfully.

4.3. The command interpreter ‘csh’.

Another program that has been modified to support eight-bit characters is the command interpreter ‘csh’. In this case, arguments to commands are placed in arrays of ‘words’. Special characters used to expand into filenames or variables used by the shell itself are for example ‘*’ that expands to all files in a directory etc. There are cases when a user may wish to suppress the special meaning of these characters allowing them to be used literally. This information is stored together with the character by setting the high-order bit in the byte. This results in inability to use the shell with Greek characters. For this modification a rather different approach was attempted. Instead of changing the representation of characters into two-byte arrays, a parallel bit array is used.

The idea is simple. If for instance we have an array of 3 words (for simplicity each word 5 characters long) we have an array of 15 bits (rounded to byte boundaries). If a character is ‘quoted’ the corresponding bit in the array is set. This alternative could not be used in the previous cases described above as the complexity of such changes would be prohibitive. In csh the array of ‘words’ is a small datastructure with comparatively local usage and small life cycle. To be precise the array exists during the parsing of a command entered to the unix shell up to the moment the command is actually executed. This allowed the modification to take place easily, and more importantly it allowed testing of the modifications to take place thoroughly. This would have been impossible in a program such as ‘vi’ in which the ‘quoting’ of a character may exist for the whole length of an editing session.

4.4. The Sun windows environment.

Our main objective was to produce a tool that would provide satisfactory terminal emulation while providing full 8-bit support.

Since the modified kernel on the Suns supported 8-bit pseudoterminals we felt that the standard terminal emulator (the shelltool) could be used with only minor modifications.
The shelltool is a rather small program (less than 150 lines of code) which makes calls to various Sun libraries. We thus were forced to make modifications to the libraries themselves.

The situation here was similar to that in 'vi' and 'curses'; the contents of the virtual terminal screen were held in an array of characters with the eighth bit specifying whether the character should be displayed in reverse video.

Since the code in the shelltool library heavily depended on interrupts, it is very difficult to locate the thread of control. Using a debugger was also difficult: as the program creates a child process which performs all processing. In the end we had to add a lot of debugging code which helped us isolate the routines that accessed the array.

Once the offending routines were identified, the array could safely be extended to an array of shorts and the resulting shelltool could at least display Greek characters.

Although the Sun keyboard is capable of generating 8-bit codes, the mapping for the Greek characters does not correspond to the standard Greek keyboard. Dead keys were not available either and characters with emphasis marks had to have dedicated keys on the keyboard. In short, this was deemed to be unacceptable and further modifications to the code in the libraries were made.

We decided to have two modes of keyboard entry; one for standard ASCII and another for Greek characters, complete with dead key support. To do this we added an extra entry to the window menu which enables the user to toggle between Greek and Latin data entry modes. A visual indication of the mode is also provided in the window banner. The toggle can also be achieved by pressing a user definable function key on the Sun keyboard.

A new procedure was inserted between the keyboard input routine and the rest of the program, to act as a filter for characters typed on the keyboard. The keyboard generates 7-bit characters that, according to the currently active mode are either left intact (Latin mode) or converted to Greek characters via a mapping table.

Apart from the Shelltool program the modification to the libraries were used in Sam, the screen editor for bitmap displays with mouse support, written by Rob Pike at Bell Labs. The author, on a visit to Crete, used the modified libraries to allow his program to support 8-bit characters. Thanks to the modifications of the library these changes took very little time to implement (a few afternoons).

5. Concluding remarks.

The whole project of modification began with time estimates which proved to be very optimistic. The aims to be accomplished in this time were those described above: support of a Research and Development environment, tools for program development, and full set of utilities for document preparation and formatting. These services would make a very useful addition to existing resources at the Institute, allowing full use of the Unix system and environment in both Greek and English.

The reasons for the optimistic time estimates result from several factors. Source code for all the operating system was available, this being one of the reasons for the selection of Unix as development environment of the Institute in the first place. A second, equally important, reason was the modularity and well designed nature of the system. According to many published reports modifications to Unix to support many diverse tasks from multiprocessors to unique devices had proved comparatively easy in the past (Bell).

The reasoning described above proved to be wrong in several ways. Apparently a modification of this type had been totally unforeseen by the authors of the original programs, and as a consequence no 'hacks' had been left to allow such changes to be easily implemented. Even the latest Berkeley version (4.3) does not fully support 8-bit characters. This can partly be explained by the fact that similar problems have not been encountered in the United States. Conversely, many European countries have had to use extended character sets. Another characteristic that became apparent was that modularity and good design of the system and some of its utilities are not as perfect as expected. Programs written in old and forgotten styles have not been altered in such an ad hoc manner and in so many different places that the initial intention of their author is totally lost. These problems led to a situation that could not have been envisaged at the outset. Modifications that were expected to take minimal time turned out to delay all subsequent development.
It was initially estimated that the project would not require more than one man years. The full modification took approximately three man years spread over a two year period. To present examples of the time spent, the initial kernel changes took four man months to complete, the modifications to "vi" three part time man months, the suntools environment changes near two full man months. The other changes were completed in respective time periods.

The conclusion that our team arrived at is that there should be a different policy in the documentation and support of such research and academic systems. More comprehensive documentation should be provided. One way to improve things would be to have documentation organised on a per program basis instead of per system as it is now. Some programs should be totally rewritten, as has been the case with the terminal driver code in System V (with the use of streams) [SysV].

On the whole, the attempt has been successful since the aims stated have been accomplished. The Institute staff currently has the set of tools referred to in this article at its disposal. The utilities are used in everyday tasks such as program development using Greek characters, Greek correspondence, document preparation etc. In conjunction with a Database commercially available on Unix that supports 8-bit characters the Greek system has been successfully used for the development of Greek office applications.

Porting the utilities to other Unix systems proved trivial and the modified programs now run on Plexus, and Masscomp machines running Unix system V. Some of the tools modified were transferred to the Xelos Perkin-Elmer system at the Ministry of Research and Technology, where they are being used with success. Overall, the results of the project provide a comfortable complete Greek/Latin system that the Institute community enjoys and uses in many diverse activities ranging from programming and development to secretarial tasks. A set of 8-bit Greek terminals are provided that allow text editing using the programs described in this article, and the Sun workstation high resolution bitmap displays are used with Greek/Latin terminal emulator windows. Dot-matrix printers allow low-resolution hard copies to be made, while an Apple laser printer provides high quality documents of the Institute to be printed.

Perhaps the most important achievement of the whole project was the experience gained by the research and technical personnel of the Institute on Unix development, and systems design in general.

Future plans for the project include a high quality desktop publishing application for the Research Centre of Crete, together with the development of other formatting tools such as Tex. Also under consideration is the addition of Greek font support for high quality text previewers on bitmap displays.

In conclusion we believe that the project proved a worthwhile attempt, providing an experience otherwise unattainable. We believe that too much time was spent on needless drawbacks of the existing systems, although without their power the whole attempt would have been hopeless.

Acknowledgments.

Many people have worked on this project. The first thanks go to Dennis Allison, Simon Gibbs and Steve Hull, responsible for the first 4.2 kernel changes, the idea of doubling character representation and the modifications of ed. Professor Manolis Katevenis contributed the design of Greek fonts for the Sun Workstation and the development of the troff filters. Finally Dimitris Nastos participated in the modifications of csh and the design of the nroff filters, suggested and implemented the parallel bit array.

References.


[K&R]