Forth and Unix Working Group

Mitch Bradley

ABSTRACT

The Forth and Unix working group included a number of people who are currently using Forth with the UNIX† operating system, and a few interested observers. The group agreed on a set of guidelines for the interface between Forth and UNIX, based on the experience of the participants. Adoption of these guidelines should increase the ability of Forth users under UNIX to share code.

System Call and C Language Interface

It is frequently desireable to use UNIX system calls from within Forth. Also, since UNIX has an extensive set of library routines that are written in or callable from the C language, Forth can benefit from being able to execute C subroutines. The following wordset defines an interface between Forth, C, and the operating system. The scheme is quite general; it should serve equally well to integrate Forth into another operating system (other than UNIX), or another language envoronment (other than C).

SYSCALL: (--) (Input Stream: system-call-name <parameter-list>)
A defining word used in the form: SYSCALL: <name> <parameter-list>
Defines <name> so that when <name> is later executed, the UNIX system call of the same name will be invoked. This should only be used to define Forth interfaces to system calls (as opposed to C language subroutines). <name> should be the same as the name of a UNIX system call, but with an underscore (_) as the first character. For example, the *read()* system call, which reads from a file, would be interfaced to Forth with:

SYSCALL _read <parameter list>

<parameter list> will be described later.

SUBROUTINE: (ext-name --) (Input Stream: <name> <parameter-list>)
Defines <name> so that when <name> is later executed, the external subroutine ext-name is invoked. ext-name is passed to SUBROUTINE: as the address of a packed string. ext-name is usually the external name of a C language subroutine. <parameter list> is described later.

ENTRY: (ext-name –) (Input Stream: <name> <parameter list>)
Builds an entry point so that the already-existing Forth word <name> may be called from outside of Forth. ext-name is the external name by which the Forth word will be known to the outside world.

This is useful, for example, when Forth calls a C routine which performs output, but the programmer wants the C output to go through the Forth I/O system. This example might result in the following:

" _putchar" ENTRY: EMIT <parameter list>

<parameter list> is described later.

† UNIX is a trademark of Bell Laboratories.

Note that ENTRY: is not a defining word, in that it does not cause a new name to be created in the Forth dictionary.

DATA: (ext-name --) (Input Stream: <name>)

Defines <name> so that when <name> is later invoked, the address of the data storage area associated with the external symbol ext-name is left on the stack. For example, if a C subroutine defines an external array:

```
int primes = { 2, 3, 5, 7, 11, 13, 17, 19 };
```

that array could be accessed from Forth by declaring:

```
" _primes" DATA: primes
```

(The external name of C objects in the UNIX world is the name with an underscore prepended, hence *_primes*).

Since the details of how arguments are passed to and from subroutines is usually different between Forth and the rest of UNIX, it is necessary to provide a means for moving arguments between the Forth stack(s) and wherever the UNIX and C language routines expect the arguments to be. Rather than requiring the Forth programmer to deal with this, the interface wordset provides a way to describe the arguments in such a way than appropriate conversions may be made automatically. A suggested implementation is to compile an appropriate bit of assembly code for each SYSCALL: , SUBROUTINE: , or ENTRY:, which would perform the argument conversions/movements. The argument specification is done with a <parameter list>. A <parameter list> specifies the type and order of the input and output arguments. The <parameter list> is a list of the types of the input arguments, followed by "--", followed by the type of the output argument, followed by "END".

The possible types are from this table:

void_ty	null type
addr_ty	address (a pointer to something)
int_ty	"standard" or "normal" integer (1 stack cell)
float_ty	floating point
dfloat_ty	double precision float
string_ty	string
char_ty	1 byte
uchar_ty	1 byte unsigned
short_ty	2 bytes signed
ushort_ty	2 bytes unsigned
long_ty	4 bytes signed
ulong_ty	4 bytes unsigned

The order of the input arguments is **opposite** from that of the C specification; i.e. the **rightmost** C argument is mentioned **first** in the Forth <parameter list>. This is due to the fact that most C compilers actually process arguments from right to left, so this scheme is likely to cause fewer potentional problems.

Example

The UNIX system call to create a new file is called *creat*. It's C language description is:

```
int creat(name,mode)
char *name;
int mode;
```

This means that it takes 2 arguments: a string (char *) which is the name of the file to create, and an integer "mode" which controls which users have various access permissions on the new file. The return value is an integer which is a UNIX file descriptor useful for subsequnetly accessing the file, or -1 if an error occurred. The Forth interface to *creat* is specified as follows:

```
(                mode     name          fd  )
SYSCALL: _creat  int_ty  string_ty  --  int_ty END
```

Errors

In UNIX there is a global variable *errno* which generally contains an extra error status code if the last system call failed for some reason. The Forth interface to this is the Forth word:

ERRNO(-- error-code)

After each UNIX system call, the value left on the stack by ERRNO will be 0 if the system call succeeded, or the contents of the UNIX global variable *errno* if the call failed. Any data storage required by ERRNO should be in the USER area, so that different Forth tasks may independently perform system calls without conflict.

Case Sensitivity?

The group had mixed feelings about this issue. The following (incomplete) set of guidelines were agreed-upon:

1 The Forth system should be able to accept either upper case or lower case input.

2 At the users option, upper case and lower case input should be treated as either distinct or indistinct.

3 Programmers are strongly encouraged to avoid the use of names that differ only in the case of the letters used; e.g., don't name one variable "blockno" and another different variable "BLOCKNO".

Input Delimiters

The Forth phrase BL WORD should treat all control characters, as well as the ascii blank character, as delimiters, both when skipping initial delimiters, and when scanning for the delimiter which terminates a word. This greatly simplifies the interpretation of ordinary text files, which may contain tabs, linefeeds, carriage returns, and formfeeds as separator characters in addition to ordinary blanks. This may be efficiently implementing by testing for "(char) BL <=" instead of "(char) BL =" when skipping or scanning for delimiters.

WORD with any character other than BL as the delimiter should treat only that character as the delimiter. In this case, leading delimiters should NOT be skipped. Not skipping leading delimiters prevents a common Forth bug whereby a zero-length string is not processed correctly. For example, some systems will not do the obvious thing when confronted with ( ) or ." " The author has NEVER seen a case where WORD with a non-blank delimiter should have skipped leading delimiters.

The actual delimiter encountered which terminates the scanning of WORD should be stored in the USER variable:

DELIMITER (– addr)

addr is the address of a USER variable which contains the actual delimiter encountered when executing the previous invocation of WORD . If the delimiter encountered was the end of the input stream, the value contained in the USER variable is -1.

This makes it easy to check for a number of end conditions.

Environment

A Forth program can access the UNIX shell Environment Variables with:

GETENV (str1 -- [str] flag)

str1 is the address of a counted string which is the name of the desired environment variable. flag is true if that environment variable is set, and str is the address of a counted string which contains the value of that environment variable. flag is false if that environment variable is not set, and str is not present.

The user may set the environment variable FPATH in his shell environment. If set, Forth may use the value of this variable as a list of directory names in which to search for files. Example (csh syntax):

setenv FPATH .:/usr/wmb/lib/forth/:/usr/local/lib/forth

If this envoronment variable is not set, Forth may use a system-dependent default list of directories in which to search for files. The default list contains the current directory as its first component, but the rest of the list is system-dependent.

Filename Extensions

Ordinary UNIX text files containing Forth source code (not in block format) should have names ending with the extension ".fth". (".f" would be nice but Fortran got it first!). Files containing Forth blocks should have names which end with ".blk".

Subprocesses

The following words provide the capability of executing UNIX subprocesses from within Forth:

SH (--) (Rest of Line: string arguments to process)
: A subshell is spawned to execute the UNIX command line which is the remainder of the Forth input line. If the user's SHELL environment variable is set, it's value controls which shell to use (Bourne shell or C-shell or Korn shell). Otherwise the Bourne shell (/bin/sh) is used. As a possible optimization, the implementation of SH is allowed to directly execute the command line in a subprocess rather than spawn a subshell, if it can determine that no special shell metacharacter expansions (like wildcards, for instance) are required.

SH[(--) (Input Stream: characters up to next])
: Similar to SH , but only those characters between the brackets [ ] are included in the command line.

-SH (command-string --)
: Similar to SH, but the command line is taken as the address of a packed string from the stack.

CHILD-STATUS (-- status)
: status is the return status returned by the most-recently executed subprocess. The implementation should keep any data associated with CHILD-STATUS in the USER area so that different tasks may execute subprocesses without conflict.

Open Issues

Many issues remain to be addressed, to wit: Terminal independent display control - TERMCAP vs Termio vs something else? Object file formats and Forth words for controlling the dynamic loading of them (as opposed to the specification of the interface points, which is covered here). Signal handling. Multitasking and the interface with (blocking) UNIX I/O system calls.

Participants

Mitch Bradley
Bill Sebok
Peter Blake
Tom Almy
Dave Hooley
Harry Arnold