18 The Linux Module (linux.hhf)

The hll.hhf library module adds a switch/case/default/endswitch statement that is similiar to the Pascal case statement and the C/C++ switch statement.

18.1 The Linux Module

To use the Linux functions in your application, you will need to include one of the following statements at the beginning of your HLA application:

```
#include( "linux.hhf" )
```

Note that including stdlib.hhf does not automatically include the linux.hhf header file. You must explicitly include the linux.hhf header file to make use of its functionality.

18.2 The Linux Header File

The Linux module contains constants, data types, procedure prototypes, and other declarations needed to make Linux system calls. Obviously, only HLA/Linux users should be making calls to this module. **Warning**: it is perfectly possible to compile the Linux module under Windows and attempt to run the resulting code. However, this will surely crash the system.

Linux systems calls are made via the INT($80) instruction. The HLA Linux module provides wrappers for all these calls so you can invoke them using a high level syntax. Calling the HLA Linux wrappers is a much better idea than embedding INT($80) invocations directly in your code. Sometimes the Linux system calls change (hopefully for the better). Although the Linux developers have done a good job of maintaining older calls in the kernel, your programs that make these older calls may not benefit from additional functionality added to the kernal. On the other hand, if all your programs call the HLA wrapper routines for Linux, then you've only got to change the call in one location (in the wrapper), rather than throughout all your projects, whenever a system call changes.

This section will not attempt to document each of the Linux calls that HLA's Linux module provides. You can read all about these functions in Linux use the "man -S 2 *function_name*" command. The calling sequence is (usually) identical to the "C" interface, so there is no need to regurgitate that information here. Because C and HLA have different sets of reserved words, there were a few conflicts between the standard (C-based) function names and the names that HLA uses, this section will elaborate on those. Also, C's structs and functions use a different namespace and the Linux (UNIX) kernel programmers have employed the dubious style of using the same name for functions and structures. Since HLA doesn't allow this, some type names have been changed, as well. Finally, to prevent namespace pollution in HLA programs, HLA actually uses a NAMESPACE to hold the Linux identifiers (much like other library modules in HLA), so common Linux function names and datatypes will require a "linux." prefix.

It is not good programming style to use all uppercase within identifiers (despite the long-standing C/Unix tradition). Therefore, most all-uppercase constant identifiers you'd normally find in a Linux header file use the HLA convention of lowercase (which is easier to read). Generally, when there is a conflict between a C identifier and an HLA reserved word, the conflict is resolved by prepending an underscore. For example, the Linux system call "exit" becomes "_exit" in HLA (since exit is an HLA reserved word). The two common exceptions to this rule are for the identifiers "name" and "value" (both HLA reserved words) which are usually converted to "theName" or "theValue" in the linux.hhf header file.

Whenever a type name conflicts with a procedure name, the linux.hhf header file appends "_t" to the type name. This is a common Unix practice and one wonders why these structure names didn't have the "_t" suffix to begin with.

Certain Linux functions are overloaded allowing one, two, or possibly three parameters. The linux.hhf header file contains two or three prototypes for each function, each with a fixed number of parameters. However, you can still call the functions using the standard C syntax because HLA provides macros to simulate function overloading (i.e., a variable number of parameters) for these particular functions. Generally, the macro uses the standard C/Linux name (e.g. linux.sysfs) while the actual HLA procedures use a numeric suffix to denote the number of parameters (e.g., linux.sysfs1, linux.sysfs2, and linux.sysfs3).

Please see the "linux.hhf" header file for more details on the spelling of Linux constants, types, and function names.

You should also note that there is a list of constants that begin with "sys_" at the end of the header file. These are the function opcodes that one passes in EAX to the INT($80) system call. If you're going to make the INT($80) calls directly yourself, you should, at least, use these symbol names (e.g., "sys_exit").