24 OS Module (os.hhf)

The OS module contains a couple functions that do OS-related tasks.

24.1 The OS Module

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

#include("os.hhf")

or

#include("stdlib.hhf")

24.2 Executing Shell Commands

procedure os.system(cmdStr:string);

The *os.system* function executes a single program and waits for the execution of that command before returning. Here is the syntax for the os.system call:

```
os.system( "system command" );
```

The string you pass as the single parameter roughly corresponds to a command shell command (e.g., the Windows command line prompt or the Linux/FreeBSD/MacOS Shell prompt). This consists of the program name followed by any command line parameters, separated by spaces.

The first thing to note about this function is that the results are system-specific. Although this function is available in all operating systems that the HLA Standard Library supports, the semantics of the commands you pass to this function vary by operating system. Therefore, programs that call this function will not usually be portable between operating systems.

Special notes for Windows users: the *os.system* function does not directly allow the execution of intrinsic (built-in) cmd.exe commands. If you want to execute a command like DIR, CD, MD, etc., that aren't actual programs, but simply commands that cmd.exe executes directly, you have to run an instance of the command interpreter to pull this off, e.g.,

```
os.system( "cmd /C dir" );  // Executes 'DOS' directory command
```

Please see the description of the Windows "cmd.exe" program for more details (type "help cmd" at the command line prompt). Also note that Windows will use the current PATH environment variable to locate the executable program, if it is not in the current subdirectory.

Special notes for Linux/FreeBSD users: If the program name appearing at the beginning of the string does not specify the path to a file that Linux/FreeBSD can find, Linux/FreeBSD will prefix the name with "/bin/" and then "/usr/bin/" in an attempt to locate the file.

The function fails silently if it cannot find or execute the specified program.

```
HLA high-level calling sequence examples:

os.system( "ls" ); // Under Linux or FreeBSD
os.system( "HLA t.hla" );// Runs HLA compiler on the "t.hla" file.

HLA low-level calling sequence examples:

  static
     cmd:string := "HLA t.hla";
         .
         .
         .
     push( cmd );
     call os.system;
```

24.3 Delaying Program Execution

The OS module provides two functions that will suspend (put to sleep) a process for a short period of time. The first function (sleep) lets you specify the suspension time in seconds, the other (mSleep) lets you specify the time in milliseconds. It is important for you to realize that the underlying operating systems do not guarantee that the delay will be exactly equivalent to the duration you specify. Most operating systems only guarantee that they will suspend the program for *at least* as long as you specify – they might actually delay the program even longer.

```
procedure os.sleep( secs:dword );
```

This function suspends the program for at least *secs* seconds. After at least *secs* seconds have transpired, the OS will place the program back into the run queue and the process will begin execution after the call to *os.sleep* on the next regularly-scheduled time quantum.

Specifying an argument value of zero may have no effect (that is, the *os.sleep* call may immediately return), but many operating systems will cause the current process to give up the remainder of it's time slice when you call *os.sleep* in this fashion. You should, however, not count on such semantics in your program.

```
procedure os.mSleep( msecs:dword );
```

This function suspends the program for at least *msecs* milliseconds. After at least *msecs* milliseconds have transpired, the OS will place the program back into the run queue and the process will begin execution after the call to *os.sleep* on the next regularly-scheduled time quantum.

Specifying an argument value of zero may have no effect (that is, the *os.sleep* call may immediately return), but many operating systems will cause the current process to give up the remainder of it's time slice when you call *os.sleep* in this fashion. You should, however, not count on such semantics in your program.