Chapter 3

MORE ON NAVIGATING THE LISP MACHINE

The last chapter discussed an aspect of programming with the lisp language, as implemented on the lisp machine. This one is about some aspects of using the lisp machine which are more or less independent of programming on it, i.e., what you might call the operating system of the lisp machine.

3.1 The scheduler and processes

Switching back and forth among the different processes can be explicitly controlled by the lisp machine programmer (read the documentation on Stack Groups), but almost never is. A special module called the scheduler generally handles this responsibility. Every 1/60th second the scheduler wakes up and decides whether the current process should be allowed to continue running, and if not, which other process should get a chance.

If the current process has been running continuously for less than a second, and wishes to continue, it is allowed to. (Note that a full second is a long time for this sort of thing, compared to other timesharing arrangements.) Or if it’s been running for a second but no other process wishes to run, it is still allowed to continue. But
if it's been monopolizing the machine for more than a second, and one or more other processes want to run, it's forced to take a rest while the scheduler gives the others a chance. The process chosen by the scheduler is now treated as the previous current process was: it will be allowed to run until some other process(es) wish to run and the current process either volunteers to give the others a chance, or passes the one second mark.

The way a process "volunteers to give the others a chance," or, in less emotionally-laden terms, informs the scheduler that it doesn't need to run, is with the \texttt{process-wait} function. \texttt{process-wait} specifies a condition the process is waiting for. When the condition becomes true, the process is ready to run. When the scheduler decides to resume the process, the call to \texttt{process-wait} returns and the computation continues from there. The first argument to \texttt{process-wait} is a string to appear in the wholine (at the bottom of the screen) while the process is waiting. The second arg is a function and any remaining args are arguments to the function. To see whether the process is ready to continue, the scheduler applies the specified function to the specified arguments. The return value of the function is what the scheduler uses for the "condition" mentioned above. This function is often called the process' \texttt{wait-function}. Here is the \texttt{process-wait} which is responsible for "Tyi" appearing in the wholine most of the time:

\begin{verbatim}
(PROCESS-WAIT "Tyi" SELF ':LISTEN)
\end{verbatim}

This call is buried somewhere in the code windows (or anything with \texttt{tv:stream-mixin}) use for reading from the keyboard. It says that the process will be ready to continue when application of \texttt{SELF} to the argument \texttt{:LISTEN} returns non-nil. Since \texttt{funcall} is equivalent to \texttt{send} when dealing with instances (see the previous chapter), this \texttt{process-wait} will return when \texttt{(send self :listen)} is true. The handler for \texttt{:listen} just checks to see if anything is in the io-buffer, so the process which calls this \texttt{process-wait} will forfeit its turns in the scheduler until it has something in its io-buffer.

Now a question for the bold: what happens if an error occurs in the scheduler? It is, after all, just another piece of lisp code. And even if the scheduler code itself is bug-free, all the \texttt{wait-functions} are called in the scheduler, and any loser* can write a buggy \texttt{wait-function}. It's also the case that blinking of flashing blinkers gets done from the scheduler. (There's a \texttt{clock function list} of things to be done every time the scheduler runs, and by default the only thing on the list is blinking the blinkers.) And any loser can also write a buggy \texttt{:blink} method for his/her blinkers — I've certainly done it. So what happens when the scheduler runs into an

\begin{footnotesize}
\begin{itemize}
\item See hacker's definition at end of chapter.
\end{itemize}
\end{footnotesize}