A Window System

Mitch Bradley

ABSTRACT

This simple window system can be used with most ASCII terminals. Arbitrary rectangular regions of the screen may be defined as windows. Output directed to a window will be confined to that region of the screen. By assigning a different window to each task, tasks may conveniently share the screen without worrying about each other. The system also manages access to the keyboard so that multiple tasks may simultaneously request input without confusion.

Motivation

Sometimes it is convenient to divide a terminal screen into several independent regions. For example, an editing program may want to display the file being edited in one area and echo commands in another area. In a multitasking system, different tasks may want to display output on the same screen without interfering with one another. The "bookkeeping" necessary to prevent the interference can be cumbersome if each task must take all the others into account. This program takes care of the bookkeeping so that each task may write to its own part of the screen without concern for the other tasks. It is also possible for single task to have more than one window, so that it can, for instance, accept commands in one place and display status in another.

Design Decisions

The system is designed to use a conventional character-oriented ASCII terminal; it does not require a bit-mapped display. It does not support sophisticated graphics, pop-up/pull-down menus, or mouse input. Such features are useful in many applications; however, they are currently extremely machine-dependent. Many conventional applications can still benefit from the screen-sharing capabilities of this simple window system without bit-mapped graphics capability. Most programs do not have to be changed at all in order to be used within a window.

A window does not retain a memory copy of the characters it displays. Consequently it is not possible to implement true scrolling in an arbitrary window. For the case where a window is the full width of the screen, it is possible to scroll by deleting the top line in the window and inserting a blank line at the bottom of the window. The window system does this. However, the visual effect is unpleasant if the window is not at the bottom of the screen, so scrolling is only done for full width windows at the bottom of the screen. For other windows, a linefeed at the bottom of the window causes the output to wrap around to the top of the window. This behavior is a compromise between esthetics, speed, and simplicity of implementation.

Windows

A window is a rectangular region of the terminal screen. Each window has a screen position, a size, and a cursor postion within the window.

A window is created with:

window: (line0 col0 #lines #cols --) (Input stream: <name>)

Creates a new window whose name is taken from the input stream. The position and size of the window is taken from the stack, and the cursor position within the new window is set to 0,0 (the upper left-hand corner of the window).

<name> (-- addr)

When the new window <name> is later executed, an address is left on the stack. This address is the address of the data storage area where some window variables is stored. This address is used to refer to the window.

Each task has a current window selected by the user variable:

my-window (-- addr)

addr is the address of a (per-task) user variable which contains the address of a window. The window whose address is stored in **my-window** will receive the output from the task.

Each window is described by six variables. When a task references one of these variable names, the address left on the stack depends on the address stored in **my-window**. In other words, only the variables for the currently-selected window are directly accessible.

line0 The topmost screen line in the window. If **line0** contains 0, the window starts at the physical top of the screen.

col0 The leftmost screen column in the window. If **col0** contains 0, the window starts at the left-hand edge of the screen.

#lines The number of lines in the window. 1 <= **#lines** <= number of lines on the screen.

#cols The number of columns in the window. 1 <= **#cols** <= number of columns on the screen.

#line The line number of the cursor position within the window. 0 <= **#line** < **#lines**.

#out The column number of the cursor position within the window. 0 <= **#out** < **#cols**.

Window Functions

The primary interface between a program and the window system is **wemit**. **wemit** behaves like **emit**, except that it displays the character only within the selected window, wrapping around if the program attempts to write past the edge of the window. In my system, **emit** is a "**defer**" word, so **wemit** may be installed as the action performed by **emit**.

wemit (char --)

char is displayed on the currently-selected window at the cursor position. The cursor is advanced to the next character position.

If the cursor was already at the end of the line, a new line is selected before displaying the character. If the cursor is already at the bottom of the window, the method of selecting a new line depends on the position and size of the window. If the window is not the full width of the screen, or if the window is not at the bottom of the screen, the new line is selected by moving to the top of the window and clearing that line. If the window is full width and at the bottom of the screen, the new line is selected by scrolling lines in the window up to make room for the new one.

If the character to display is not a printable ascii character, it is either interpreted as a control character or displayed with a caret, e.g. ^X. The following control characters are interpreted:

linefeed (control J)

Advances the cursor to column 0 of the next line, and clears that line.

carriage return (control M)

Moves the cursor to column 0 of the current line.

backspace (control H)

Moves the cursor backwards one character position, unless it was already at column 0 in the window, in which case nothing happens.

tab (control I)

Inserts enough spaces to move the cursor to the next tab stop within the window. The variable **tabstops** contains a number (initially 8) which is the distance between successive tabstops.

formfeed (control L)

Erases the window and positions the cursor at the upper left-hand corner.

The window system also provides some functions for cursor control within the window, such as are commonly used by screen-oriented editors, etc.

right (--)
The cursor is moved one position to the right, unless it is already at the right edge of the window.

left (--)
The cursor is moved one position to the left, unless it is already at the left edge of the window.

+chars (n --)
The cursor is moved n positions to the right if n is positive, or -n positions to the left if n is negative. The cursor will not move past either edge of the window; it will stop at the edge if n is too large. **+chars** is generally faster than repeated calls to **left** or **right** for large n (where "large" is dependent on the terminal and the baud rate). A general guideline is: If you know you only want to move 1 or 2 positions, use **left** or **right**, otherwise use **+chars**.

kill-line (--)
The line containing the cursor within the current window is erased (filled with blanks) from the cursor position to the right edge of the window. The cursor position is left unchanged.

erase-screen (--)
The entire window is erased, and the cursor is left at the upper left-hand corner of the window (line 0, column 0).

insert-char (char --)
char is displayed on the screen at the cursor position, and the rest of the line within the window is moved over to accomodate it. The character that was previously at the rightmost edge of the screen is lost. The cursor is moved one position to the right, so that repeated calls to insert-char will insert characters from left to right.

delete-char (--)
The character at the cursor position is deleted from the window, and the rest of the line is moved to the left to fill in the deleted position. A blank is placed in the vacated position at the right edge of the window. The cursor position is left unchanged.

border (--)
A border is drawn around the window. The characters that make up the border appear in the lines and columns just outside the extent of the window. If an edge of the window is at the edge of the screen, the border is omitted along that edge.

Input

In a multitasking environment, if several tasks each want input from the same keyboard, care must be taken that only one task at a time actually gets the keys typed. The window system manages this by assigning the keyboard input stream to one of the tasks. All keys typed will go to that task, unless the user types control X. When control X is typed, the keyboard is assigned to another task that is requesting input. Repetitive control X's will cycle through all the tasks that want input.

Terminal Interface

To be used with a particular terminal, a number of functions must be defined appropriately for the terminal.

(#lines (-- n)
n is the number of lines on the terminal's screen.

(#columns (-- n)
n is the number of columns on the terminal's screen.

(left (--)
Moves the terminal cursor one position to the left. If the cursor is already in the leftmost column, the behavior is undefined.

(right (--)
Moves the terminal cursor one position to the right. If the cursor is already in the rightmost column,

the behavior is undefined.

(up (--)
Moves the terminal cursor up one line. If the cursor is already on the topmost line, the behavior is undefined.

(down (--)
Moves the terminal cursor down one line. If the cursor is already on the bottom line, the behavior is undefined.

(at (line column --)
Moves the cursor to the indicated line and column on the screen. If line is outside the range 0 <= line < (#lines, or if column is outside the range 0 <= column < (#columns, the behavior is undefined.

(insert-char (char --)
char is displayed on the screen at the cursor position. The rest of the line is moved over to the right to make room for the inserted character. The righmost character on the line is lost.

(delete-char (--)
The character at the cursor position is deleted from the screen, and the rest of the line is moved over to fill in the vacated position.

(kill-line (--)
The current line is erased from the cursor position to the edge of the screen. The cursor position is left unchanged.

(kill-screen (--)
The screen is erased from the cursor position to the end of the screen. The cursor position is left unchanged.

(insert-line (--)
The current line and all subsequent lines are moved down to make room for a blank line that is inserted at the cursor position. The last line on the screen is lost.

(delete-line (--)
The current line is deleted from the screen, and subsequent lines are moved up to fill in the space. The last line on the screen is filled with blanks.

(erase-screen (--)
The entire screen is erased and the cursor is left in the upper right-hand corner.

beep (--)
The bell (or whatever noisemaker the terminal has) is rung.

dark (--)
Subsequent characters will be displayed in some sort of standout mode, if the terminal has one. The preferred standout mode is inverse video.

light (--)
Subsequent characters will be displayed in the normal (not standout) mode.

Source Code

The code for the window system is written in Forth 83. It will be published later.