TELERAID

TeleRaid is an interactive debugger which can be used either to examine, from one workstation, the state of another workstation's virtual memory, or to look inside a sysout file.

Requirements

REMOTEVMEM, READSYS, RDSYS, VMEM

Either:

Ethernet PUP connection between the two machines. The machine which is to be examined must be in the TeleRaid mode; i.e., the shape of its cursor must be the TeleRaid prompt. It must also have a PUP address.

Or:

A sysout file.

Installation

Load TELERAID.LCOM and the required .LCOM modules from the library.

User Interface

The standard use of TeleRaid is to debug a workstation that has stopped at a maintenance panel halt. Pressing the UNDO key when the machine is in this state transfers control to a small TeleRaid server that responds to simple commands over the network. While the TeleRaid server is running, the cursor changes to TELERAID. Also, on a 1108 workstation, the previous contents of the maintenance panel are restored.

On a 1108 workstation, the maintenance panel halt condition is indicated by a four-digit code that begins with a 9. On an 1186, the four-digit code is displayed at the cursor.

The term "debuggee" is used to denote the sysout file or machine running the TeleRaid server, i.e., the one being debugged, while "debugger" refers to the machine that is viewing the debuggee's virtual memory (usually by running TeleRaid).

Function

(TELERAID *HOST RAIDIX*) [Function]

Enters an interactive debugger viewing the virtual memory of *HOST*, which must denote a machine running a TeleRaid server. *HOST* is either a host name or a PUP address.

RAIDIX is an optional number denoting the radix in which values are printed and numbers are accepted as input; if not specified, it defaults to 8 (octal). The only other accepted value for *RAIDIX* at present is 16, for hexadecimal input and output.

If you don't know a machine's PUP name or address, you can find out by typing control-P on the debuggee: control-P changes the maintenance panel to show the machine's PUP host number in decimal radix. You can also find out your PUP address when Lisp is running (rather than in a maintenance panel halt) by evaluating (PORTSTRING (ETHERHOSTNUMBER)). Users typically do this once and tape a note to the terminal so as to have this information handy.

If the debugger is on the same physical Ethernet as the debuggee, you can use that PUP host number directly as the *HOST* argument. Otherwise, you must convert the PUP host number to octal and use the general form of a PUP address, which is a string of the form *"net#host#"*.

For example, (TELERAID 12) debugs the machine whose PUP address is 12 decimal on the same network. (TELERAID "13#14#") debugs host 14 octal (12 decimal) on network 13 octal.

Note: If the control-P command displays zero in the maintenance panel, it means the machine does not have a PUP host number assigned, or the halt occurred so quickly after booting that the Ethernet has not been fully initialized. In this case, TeleRaid cannot be used. See the description of READSYS (below) for directions on TeleRaiding a sysout file.

TeleRaid Commands

Each TeleRaid command is a single character, followed by arguments appropriate to the command. In the description of the commands that follows, unless otherwise specified, numbers are assumed to be typed in the default radix (octal unless you have specified a different *RAIDIX* in the call to TELERAID).

Displaying a Stack

For casual users, the L command followed by several F commands generally provide the most useful information. Many of the other commands require some knowledge of the internal

representation of Lisp objects and stack frames, something that this document does not attempt to provide.

L shows the stack of the debuggee, as a back trace consisting of a numbered sequence of frame names. The first frame is usually \MP.ERROR if you got here by a maintenance panel halt.

In the case of MP code 9305, the stack shown is the page fault handler's and is uninteresting, except for the argument to the \INVALIDADDR frame.

Use the control-L *P* command to see the stack of the process that took the fault.

control-L *type* Shows the stack of the debuggee starting at some other place. The argument *type* is a single letter denoting which stack to view. The system has a number of special contexts, which are areas of stack space used by certain system routines.

Legal values of *type* are P (page fault), G (garbage collector), K (keyboard handler), H (hard return), S (stack manipulator), R (reset), and M (miscellaneous).

The most interesting of these for most users is P, which for MP code 9305 shows the stack in which the address fault occurred. In addition, *type* F lets you view the stack starting at an arbitrary stack frame; follow F with an octal number denoting the frame (as in the control-X command, below).

K *type* Changes the type of stack link that the L and control-L commands follow to be *type*, which is either A or C. The default is to follow CLinks (control links). ALinks follow the chain of free variable access instead.

Viewing Frames From a Stack

After displaying a particular stack with the L or control-L commands, the following commands view individual frames from that stack:

F *number* Prints the contents of frame *number*, where *number* is the number next to the frame name in the back trace.

Note: Unlike most other commands, *number* is in decimal.

The frame is printed in two parts, a basic frame containing the function's arguments and a frame extension containing control information, the function's local (PROG) variables, and dynamic values. On the left side of the printout are the octal contents of each cell of the frame, with an interpretation, usually as a Lisp value, on the right.

line-feed or control-J Shows the next frame (closer to the root of the stack). Same as F *n* + 1, where *n* is the number of the frame most recently viewed. Immediately after an L or control-L command, *n* is zero, so line-feed views the first frame.

↑ Shows the previous frame. Same as F *n*–1.

D *symbol* Shows the definition cell for *symbol*. A definition cell containing all zeros denotes an undefined function. A definition cell whose left half is less than 400 denotes an interpreted definition; you

can use the V command (below) to have it printed as a Lisp expression.

A *symbol* Shows the top-level value of *symbol*.

P *symbol* Shows the property list of *symbol*.

C *symbol* Prints (using PRINTCODE) the code definition for *symbol*.

V *hi lo* Interprets the virtual address *hi, lo* as a Lisp value and attempts to print it. Virtual addresses appearing in stack frames are already interpreted for you by the F command, as are those in value cells (the A command) and property lists (the P command), but you may want to use the V command if you find a virtual address inside some other structure.

B *hi lo count* Prints *count* words of the raw contents of the virtual memory starting at virtual address *hi, lo*. This is most useful for examining the contents of a datatype, which other commands simply print as its virtual address, i.e., in the form {type}#hi,lo.

_ *hi lo number* Sets the contents of the word at virtual address *hi, lo* to be *number*. This command obviously should be used with care.

control-V *symbol atomicValue* Sets the top-level value of symbol to be *atomicValue*, i.e., this is a remote SETTOPVAL. Only symbols and small integers are acceptable values to set. In addition, if the previous value was not a symbol or small integer, it is not reference counted correctly, so will not be garbage collected.

U Displays the debuggee's screen on your own (just the screen bit map, not the cursor). Typing any character restores your own screen. If the debugee's screen is larger than the debugger's, then you'll see that portion of the screen that fits. You can move the image of the remote screen by pressing the left mouse button and dragging the image, much like an over-size icon.

control-Y Enters the Old Interlisp Executive under TeleRaid, where you can evaluate arbitrary Lisp expressions or call some of the functions listed below to perform TeleRaid operations for which there is no command.

Use the Interlisp Executive's OK command to exit and return to TeleRaid.

Q Quits TeleRaid without affecting the debuggee.

control-N Executes the CONTROL-N TeleRaid command in the debuggee, i.e., causes the debuggee to resume execution, and quits TeleRaid. This command should not be used unless you are sure that the debuggee is resumable.

Viewing the System Stack

The following commands are for use by experts in stack format. A stack address is a number in the default radix denoting where the object of interest starts.

W *address* Walks sequentially through the system stack (i.e., by stack address, not by control or access links) starting at *address*, showing the stack frame type and its name (for frame extensions). If *address* is not given, this command shows the

entire user stack. For the READSYS function (see next section) the walk starts at zero, so it shows the system stack as well.

control-F *address* Prints the basic frame stored at *address*.

control-X *address* Prints the frame extension stored at *address*.

S *address count* Prints raw contents of the stack (as with the B command) starting at *address* for *count* words.

Functions for Saving Work

The following functions do not have corresponding TeleRaid commands, but may be useful to call in the executive obtained from the control-Y command. They can be used to try to patch a broken sysout back into shape, or at least to save some of the work out of a workstation in a maintenance panel halt. Further functions like these can be written using the functions described in the next section.

(VLOADFNS *FN*) [Function]

Reads the EXPR definition of *FN* from the remote environment and stores it locally on *FN*'s EXPR property. *FN* can be a single symbol or a list of symbols.

(VLOADVAR *VAR*) [Function]

Locally sets the variable *VAR* to be the remote top-level value of *VAR*.

(VSAVEWORK) [Function]

Attempts to figure out what has changed and not been saved in the remote environment by looking at CHANGEDFNSLST, CHANGEDVARSLST and the property lists of files on FILELST. For each changed function or variable, it asks you whether to save it, and if so, it uses VLOADFNS or VLOADVAR to fetch it. You can then save the functions or variables from the locally running Lisp.

VSAVEWORK does not know how to save records, properties, etc., although a knowledgeable programmer could use the functions described in the next section to extend VSAVEWORK.

(VUNSAVEDEF *FN*) [Function]

Attempts to do a remote UNSAVEDEF by going down the VGETPROPLIST of *FN*, looking for properties CODE, BROKEN, and ADVISED. If it finds one, it stores the corresponding code object in *FN*'s remote definition cell, and prints a message saying what it has done.

For example, if you've managed to break something that's used by the interpreter, and have thus gotten into a recursive break, you might be able to recover by VUNSAVEDEFing it, then doing a control-D on the remote machine.

(VYANKDEF *NEWSYMBOL OLDSYMBOL*) [Function]

Yanks the definition from function *OLDSYMBOL* and stores it into *NEWSYMBOL*. For example, (VYANKDEF 'PRINTBELLS 'NILL) turns off ringing of the bell in the remote environment.

Note: VUNSAVEDEF and VYANKDEF do not adjust reference counts, or interact correctly with BREAK and ADVISE. They should be thought of as emergency patches designed to get the system running long enough to save state and bail out. In particular, do not call UNBREAK or UNADVISE on a function that you have applied VUNSAVEDEF to, and do not alter or remove its CODE, BROKEN, or ADVISED property. Similarly, do not redefine the function *OLDSYMBOL* that you have yanked a definition from.

Implementation

TeleRaid is implemented in two parts: ReadSys, which reads a remote system's virtual memory, and VRaid, the interactive debugger described above. The remote virtual memory can be either a workstation running a TeleRaid server or a sysout file. The functions inside TeleRaid look like normal Lisp functions, but they are designed to operate on the remote virtual memory, rather than the normal (local) virtual memory. The remote versions of functions normally begin with V.

In general, TeleRaid is not a facility for the casual user. It is mostly used by system implementors performing very low-level debugging. The set of functions described here is a partial list, intended to help the serious programmer who has some interest in doing this kind of debugging.

Reading the Remote Vmem

(READSYS *FILE WRITEABLE*) [Function]

Opens the remote virtual memory *FILE*, which should be the name of a file in sysout format. If *WRITEABLE* is T, then the file is opened for write access, so that commands that alter the virtual memory (e.g., the __ and control-V commands) are permitted. The main use for this is to patch sysouts in simple ways (e.g., by changing a global flag from NIL to T).

FILE can also be a list of one element, the PUP address of a machine running a TeleRaid server, in the same form as the *HOST* argument to the function TELERAID. In this case, *WRITEABLE* is ignored.

If *FILE* is NIL, READSYS closes any open virtual memory file, clears its data structures and reverts to examining no virtual memory.

(VRAID *RAIDIX*) [Function]

Runs the TeleRaid interactive debugger on the virtual memory most recently opened by READSYS.

Manipulating the Remote VMem

The functions and macros described below directly manipulate the remote virtual memory. You can call them directly in the Lisp executive that you get by using the control-Y command under TeleRaid, or at the top level after calling READSYS. You can also, of course, write your own programs to use these functions. In order to use any of the macros below, you must LOADFROM the source file VMEM.

In the following functions, a pointer means a pointer into the remote virtual memory (the argument *PTR*), a 24-bit integer. All other arguments refer to local objects. Functions that fetch out of or store into the remote virtual memory operate on pointers. You can create a local copy of the structure denoted by a pointer by calling V\UNCOPY. You cannot do the inverse, i.e, create remote copies of local structures—the only local objects that you can translate into the remote virtual memory are symbols (assuming the same symbol exists remotely) and small integers.

Note: The functions that store into the remote memory should be used with care. None of these functions perform the proper reference counting. Therefore, if you are storing a value that ought to be reference-counted (roughly speaking, anything other than a symbol or small integer) and/or overwriting such a value, the garbage collector may get confused when the remote memory is resumed.

(VVAG2 *HI LO*) [Function]

Returns a pointer with hi-loc (top 8 bits) *HI* and lo-loc (low 16 bits) *LO*.

(VHILOC *PTR*) [Macro]

Returns the high part of *PTR*, i.e., (LRSH *PTR* 16).

(VLOLOC *PTR*) [Macro]

Returns the low part of *PTR*, i.e., (LOGAND *PTR* 177777Q).

(VADDBASE *PTR D*) [Macro]

Remote \ADDBASE: Returns a pointer that is *D* words beyond *PTR*, i.e., (IPLUS *PTR D*).

(V\UNCOPY *PTR*) [Function]

Returns a local copy of the remote structure pointed to by *PTR*. \UNCOPY only knows how to copy ordinary structures: symbols, integers (not bignums), floating-point numbers, characters, strings and lists. All other pointers, either as the argument to V\UNCOPY or inside structures copied by V\UNCOPY, are converted to local objects of type REMOTEPOINTER that print in the way that datatypes conventionally print—their contents are *not* copied.

(V\COPY *X*) [Function]

Returns a remote pointer to the local object *X*, which must be a symbol or small integer.

(VGETTOPVAL *ATOM*) [Function]

Returns a pointer to *ATOM*'s top-level value.

(VGETVAL *ATOM*) [Function]

Returns a local copy of *ATOM*'s top-level value, i.e., (V\UNCOPY (VGETTOPVAL *ATOM*)).

(VSETTOPVAL *ATOM VAL*) [Function]

Sets *ATOM*'s top-level value to be *VAL*, which must be a symbol or small integer.

(VGETPROPLIST *ATOM*) [Function]

Returns a pointer to ATOM's property list.

(VGETDEFN *ATOM*) [Function]

Returns a pointer to *ATOM*'s function definition.

(VTYPENAME *PTR*) [Function]

Returns the type name of *PTR*.

(VGETBASE0 *PTR*) [Function]

The most primitive fetching function: Returns the 16-bit integer stored in location *PTR*.

(VPUTBASE0 *PTR VAL*) [Function]

The most primitive storing function: Stores the 16-bit integer *VAL* into location *PTR*.

(VFIND.PACKAGE *NAME*) [Function]

Like the CL:FIND-PACKAGE, but returns the remote address of the named package or NIL if not found.

(VFIND.SYMBOL *NAME REMOTE-PACKAGE*) [Function]

Like the CL:FIND-SYMBOL, but returns the remote address of the named symbol.

(VGETBASE *PTR D*) [Macro]
(VPUTBASE *PTR D*) [Macro]
(VGETBASEBYTE *PTR D*) [Macro]
(VGETBASEPTR *PTR D*) [Macro]
(VPUTBASEPTR *PTR D VAL*) [Macro]

These are remote versions of \GETBASE, \PUTBASE, \GETBASEBYTE, \GETBASEPTR and \PUTBASEPTR, respectively. They are implemented in terms of VGETBASE0 and VPUTBASE0.

Limitations

TeleRaid uses PUP, thus the machine being examined must be on the same network or reachable via PUP gateways.

This code has one major shortcoming which will not normally turn up. If the local and remote sysouts conflict in their package setups, it is possible for this code to return symbols interned in what for the Teleraiding machine would be the correct package, but for the remote machine is in fact incorrect. The problem lies in the fact that you cannot uncopy a symbol correctly between two machines with incompatible package setups. An example of such a situation would be where on one machine the package FOO inherits BAR, and on the other BAR is present directly in FOO. BAR's package cell will be different in the two cases.