Subject: RAM resident utilities, TRAPs from TRAPs
Posted by Anonymous on Sat, 08 Nov 1986 18:49:49 GMT
View Forum Message <> Reply to Message

Originally posted by: braner&#64batcomputer.tn.cornell.edu (braner)

Article-I.D.: batcompu.1457
Posted: Sat Nov 8 13:49:49 1986
Date-Received: Mon, 10-Nov-86 00:48:35 EST
Reply-To: braner@batcomputer.UUCP (braner)
Organization: Theory Center, Cornell University, Ithaca NY
Lines: 96
Summary: LONG, of interest to programmers only

[]

I'd like to share with y'all some thoughts and experiences concerning
the topic of writing memory-resident utilities for the ST.

By a "memory-resident utility" I mean a program that is installed in
RAM when you "run" (double-click) it, and is later called upon to
do something under certain conditions. "Desk accessories" fit that
definition, but I am NOT referring to those here, but rather to
programs that are intended to be invisible, such as printer drivers,
my recent "killer.prg", keyboard translation tables, etc. Such
utilities are invoked via an altered exception vector, system variable,
and the like.

Since I wrote a few of these recently, I got to think about some
protocol for their installation. Of course, there might be an official
protocol already in existence, but I have not heard of any. Here is
my suggested protocol:

The part of the program that is to be resident is compiled/assembled
as near the beginning of the program as possible, to save memory.
For most compilers/assemblers this is done simply by putting that
part of the code at the beginning of the source file.

When "run", the program installs the needed vectors, then does a
"terminate and stay resident" GEMDOS call (Ptermres()) to exit,
asking GEMDOS to save only the parts that are really necessary
(i.e. the installation part is discarded).

Let's define the "utility address" (UA) to mean the address that is pointed
to by the vector/variable/etc - usually the address called to execute the
utility. This address is easy to find later by examining the vector.
I use the 8 bytes PRECEDING the UA to help manage multiple utilities:

During installation, the program saves the former contents of the vector
(i.e. the former UA) in the 4 bytes preceding the (new) UA. In the 4 bytes
before that (i.e. at -8(UA) in AL jargon) a magic number resides,
a number that lets you identify the utility. (I use a mnemonic string
of 4 ASCII codes.)

The magic number and the old vector are useful for many purposes:

The installation procedure may check the vector before
altering it, and warn the user that this (or another) utility
has already been installed.

Programs that want to bypass a utility can do so by looking-
up the former vector and using it instead.

It makes it easy to write programs that deactivate utilities.
Also, in theory one could write a program to unload such
utilities, freeing up the RAM.

Now some important classified information:

If you want to use OS calls (traps) inside an interrupt handler
(something that is necessary but officially verboten), make sure
the interrupted routine was not in supervisor mode. This will
ensure that, in particular, you are not calling a TRAP within a TRAP.
(Thanks to whoever posted "WATCHER" here recently - that is where I
took this idea from.) The way to check that is to examine the CPU
status which was saved on the stack as part of the exception processing.
In AL simply do:

```
UA:
 BTST #5,(A7)  * zero if user mode
 BNE ...  * do NOT call any traps
 ...   * action with traps here
```

That works when the code is at the same "stack level" as the RTE,
i.e. may safely end with an RTE. That is the case when it is called
directly by the exception vector or is JMPed to. But look out for the
cases where UA is called as a subroutine, so more stuff has been put
on the stack since the exception happened. For example:

WARNING: The following info is unofficial and will probably change
in future versions of the TOS ROMs. Do NOT use this
info in a program you expect others to use after such
revisions.

In the current ROM TOS version of the Alt-Help screen dump, the UA stored at $502 is called to actually do the screen dump. You may install your own UA there. But the exception handler saves all registers except A7 (15 registers) on the stack, then calls a subroutine which in turn calls ($502) as a subroutine. What it all adds up to is that the needed check is now:

```
 BTST #5,68(A7)
```

(Note that you do NOT need to save any registers you use inside your screen dump code.)

Any comments/suggestions would be appreciated.

- Moshe Braner