Subject: Re: Limiting the number of VM instances
Posted by robert.dimeo on Thu, 22 Jan 2004 16:02:51 GMT
View Forum Message <> Reply to Message

robert.dimeo@nist.gov (Rob Dimeo) wrote in message
news:<cb539436.0401131159.7bff3179@posting.google.com>...
> Hi,
>
> Is it possible to limit the number of VM instances running at once?
> Let's say that I have APP.SAV running from the VM. I can launch
> another VM session with another version of APP.SAV running. Can I
> restrict this so that only one application runs?
>
> This is probably an OS issue so using XREGISTERED won't work. I would
> like to do this in WIN2K.
>
> Thanks,
>
> Rob

Hi again. Once again the helpful folks at RSI suggested a solution
and I wrote a function encapsulating the logic called
LIMIT_VM_INSTANCE. The main idea is to use a shared memory segment.
In addition to a listing of the function I also include here a simple
widget program that uses this function. Note of course that you must
go through the usual step of creating a SAVE file to use it in the VM.

Hope that this is useful.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;
function limit_vm_instance,   INIT = init,   $
                        EXIT = exit,   $
                        PROCESS_NAME
; Keywords:
;
;  INIT: set prior to defining the widgets
;  EXIT: set in the cleanup
;
; Parameters:
;
;  PROCESS_NAME:  required string variable (that might be set
;               to the same string as the name with which the
;               widget is registered with XMANAGER for instance).
;
if n_params() eq 0 then begin
  !error_state.msg = 'You must pass in a PROCESS_NAME variable'
  return,0
endif
```

```
if keyword_set(init) then begin
  if lmgr(/vm) then begin
    os_handle = 'mysegment'+'_'+process_name
    shmmap, process_name, template = [0B],    $
      os_handle = os_handle
    v = shmvar(process_name)
    if (v[0] eq 1) then begin
      msg = 'An instance of this code is already running.'
      v = dialog_message(msg, /error)
      shmunmap,process_name
      return,0
    endif
    v[0] = 1
  endif
  return,1
endif
if keyword_set(exit) then begin
  if lmgr(/vm) then shmunmap,process_name
  return,1
endif
return,1
end
 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;

; Simple widget application that uses this follows

 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;
pro shared_example_cleanup,tlb
widget_control,tlb,get_uvalue = pstate
; Call the function that limits the number of VM instances
; with the EXIT keyword set.  Note that we must pass in
; the "register_name" for the application which is stored
; in the state structure.
ret = limit_vm_instance(/exit,(*pstate).register_name)
ptr_free,pstate
end
 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;
pro shared_example_event,event
uname = widget_info(event.id,/uname)
case strupcase(uname) of
'QUIT':  widget_control,event.top,/destroy
else:
endcase
end
 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;
pro shared_example
; Call the function that limits the number of VM instances
```

```
; with the INIT keyword set.  Note that we also will need to
; call this same function with the EXIT keyword set in a
; cleanup routine...this is necessary to unmap the variable that
; is in shared memory.  We need a unique process name for
; the memory segment so we can just use the name with which
; we'll register the application with the XMANAGER.
register_name = 'shared_example'
ret = limit_vm_instance(/init,register_name)
if not ret then return

; Limit the instance in an IDL session using the usual
; XREGISTERED function.
if xregistered(register_name) then begin
   msg = 'An instance of this code is already running.'
   v = dialog_message(msg, /error)
   return
endif

; The following simple widget code just puts up a QUIT button.
tlb = widget_base(title = 'shared memory example', $
   /tlb_frame_attr,/col)
void = widget_button(tlb,value = 'quit',xsize = 200,  $
   uname = 'quit')
widget_control,tlb,/realize

; Store the "register_name" variable in a state structure and
; set the uvalue of the TLB to a pointer to the state structure.
state = {register_name:register_name}
pstate = ptr_new(state)
widget_control,tlb,set_uvalue = pstate

; We must use a cleanup routine in order to unmap the variable
; from shared memory so specify it here.  Remember to use the
; register_name variable here.
xmanager,register_name,tlb,/no_block,  $
   cleanup = 'shared_example_cleanup', $
   event_handler = 'shared_example_event'
end
```