caused you, but I will follow through this time to ensure that it doesn't happen again. If you could just send me mail with the serial number, it will help expedite things.

Thanks, and again, I humbly apologize for the mistake.

--Brian

>From alexn Tue Apr 30 07:57:12 1991
To: v-ibmbe
Cc: bradsi v-ibmbe
Subject: RE: P70 replacement
Date: Tue Apr 30 06:44:33 PDT 1991

Do you remember getting my summary of problems done some months ago? Lorisi and I put together an email describing the specifics (our PC repair person helped make it VERY specific) of the problems associated with both p70s we have had over the past several months, and sent it to you. Let me know if you still have it, and I can add to it any recent happenings.

If you don't have it, let me know and I'll work with Lori and PC Repair to put together another one.

Thanks for following up on this - we're anxious to get it resolved as well.

Alex

>From v-ibmbe Mon Apr 29 18:08:41 1991
To: alexn
Cc: bradsi v-ibmbe
Subject: P70 replacement

Date: Mon Apr 29 18:06:17 1991

Hi Alex - Brad said you'd be sending me a summary of the problems you've experienced with your P70. Could you please include the serial number of the unit as well? I'm anxious to get going on this.

Thanks --Brian

4
From philba Wed May 1 08:33:10 1991
To: bradsi
Subject: resend: Win31 Robustness: TODAY, 4pm, 3/2131
Date: Wed May 01 08:25:44 1991

>From bens Tue Apr 30 14:55:06 1991
To: philba
Subject: resend: Win31 Robustness: TODAY, 4pm, 3/2131
Date: Tue Apr 30 14:51:13 1991

Let's meet this afternoon to discuss what our research has uncovered so far, see where we are on making concrete work estimates, and set some milestones.

Here is my first pass at the Robustness Memo:

Overview

In recent weeks, IBM has been trumpeting OS/2 2.0 as a much more robust workstation environment than Windows 3.0. They demonstrate a "bad" Windows app causing an Unexpected Application Error (UAE), and claim that Windows is now corrupted. While IBM is overstating the difference in robustness between Windows and OS/2, there is no question that Windows has room for improvement.

The key goal is to make Windows 3.1 (Win31) robust enough that:

1) An app that crashes will not harm Windows
2) ISVs can write applications that rarely crash

This paper describes how we will add robustness to Win31 while maintaining our performance edge over OS/2:

1) Validate parameters to API calls -- especially those where an invalid value would cause Win31 to corrupt its internal state.
2) When an app crashes, clean up any global state in Win31.

Having done this work, we will honestly be able to say that Win31 and OS/2 2.0 have comparable levels of robustness:

1) An app cannot corrupt Win31 or other apps by making a bad API call.
2) An app cannot corrupt Win31 or other apps by crashing itself.

This still leaves the following differences between Win31 and OS/2 2.0

1) An app can corrupt another app in Win31, but not (directly) in OS/2.
2) An app can corrupt more of Win31 than it can of OS/2 2.0.

We claim that (1) and (2) above are rare, and so the added nominal protection of OS/2 provides little actual improvement in robustness.

What is Robustness?

I will define "robustness" as the ability for Windows to survive the crash (usually by GP-Fault) of a Windows or DOS application. Windows is very robust if there is no possible way for a Windows or DOS application -- even one written with the malicious intent of crashing Windows -- to corrupt another application or the system as a whole. Windows is very unrobust if

even a casual error in a program can affect another Windows application or the system as a whole.

In the rest of this paper, I will address the robustness of Windows as it is affected by Windows applications. I will address DOS applications in a future paper, since they do not appear to crash very often.

Why is Windows 3.0 Less Robust than OS/2?

In a fully robust operating system, there is no way for an app to corrupt other apps or the system itself. NT OS/2 is an example of such a system. The address space of each app is separated both from other apps and from the system, and the parameters to every system call are fully validated. Hence, the system has total control over what the application can and cannot do.

Windows 3.0 is architecturally different from NT: There is a single address space, shared by all apps and by the system. The benefit of this design is that Windows can be faster. There is no address space switching overhead when switching between apps, and there are no ring transitions when executing Windows KERNEL, GDI, or USER functions. NT, by contrast, must switch address spaces (by going to Ring 0) when switching between apps *and* when executing Windows functions – since these functions are implemented in a server process.

Since Windows apps and the system are all in the same address space, apps have unrestricted access to system code and data. For example, an app could scribble all over USER's data segment (assuming the app could find it). [The app cannot write directly to USER's code segment (due to the read-only bit in the segment descriptor), but that only helps a little.]

How are we going to make Win31 More Robust?

As bad as a single address space sounds, in practice it is probably a very rare source of UAEs. Ask yourself, how would an application get the selector of a system data segment? A *malicious* app could definitely use TOOLHELP.DLL, or go spelunking around and find it. But a *normal* app is almost always using selectors it has received from a system call. If not, then the app would not work very well.

In the rare case where an application uses a selector that does not "belong" to the application, that selector is likely a random value.

If the app is not using one of its selectors, then it has (via programming error) used some random value as a selector. In this case, it is most likely that the random value is an invalid selector.

So, while a bad app *could* scribble on Windows, in practice we assume this to be a very rare case.

The two areas where Windows *is* noticeably weak are: 1) parameter validation, and 2) cleanup after an app crashes.

Here is what a Win31 DLL must do to be robust:

1) "Trust" user pointers

It is expensive to verify that a pointer passed in by a user is valid. You would have to determine: 1) the selector is valid, has the right access bits, and far ptr + length are within the limit of the selector; 2) The segment is "owned" by the calling task (as opposed to another task); and 3) the segment does not belong to Windows.

So, a Windows DLL must take the following stance when handling user pointers:

USE OF A USER POINTER MAY CAUSE A FAULT.

Be prepared to clean up any global state (via your GP-Fault handler) if you fault on the pointer.

Implementing this stance may require work in either the mainline code, the GP-Fault handler code, or both. If cleaning up after a GP-Fault is really difficult, consider using VERR/VERW, LAR, LSL (all with appropriate, errata-proof macros!).

2) Validate all non-pointer parameters

This is especially important if a parameter is a handle, or an index into a table. In these cases, the parameter is logically a pointer into private DLL data. If the handle is bad, or the index is too large, the DLL will probably access its own memory in an incorrect way, and likely corrupt itself.

3) Minimize global state

If an application causes a Windows DLL to GP-Fault -- which would only occur to a DLL that has followed rules 1 and 2 -- then the cause is a bad user pointer parameter. The DLL's GP-Fault routine has to be able to clean up the API call so that subsequent calls into the DLL (from other tasks) will succeed.

4) Add a GP-Fault routine to the DLL

This routine is called when a GP-Fault occurs. USER, GDI, and KERNEL each have such a routine. If a code segment of the DLL was executing when the fault occurred, a flag in the parameters passed to the routine will be set.

The task of this routine is to return the DLL to a consistent state from a possibly inconsistent state. For example, if GDI is going to clean up DCs for the faulting task, it may need to validate all the fields of the DC -- not assuming, for example, that the

pointers are valid.

5) Make Debug version more picky

a. Call DebugFillBuffer(LPSTR lpstr, WORD cb) on all user buffers. This routine fills the user buffer to the specified size, which will help catch the case where the actual buffer size is smaller than what the app claimed.

We cannot turn this on in a retail system, since it is slow, and may crash an app that would not otherwise crash (since in practice it may only get return values that fit in the actual buffer size).

6) Be nice in failure path of API

If an API call fails when an app does not expect it to, we can be clever in the error path to try and prevent the app from dying.

a. If an API was supposed to return an ASCIIZ string, put a 0 byte in the first position of the user buffer.

b. [any other ideas like this?]

It is up to the programmer to decide between validating a pointer in an API as opposed to handling the possibility of a fault in the GP-Fault routine.

If cleaning up after a GP-Fault (on a pointer parameter) in the middle of a particular GDI call is difficult, then validating the pointer parameter in the API is the right choice. Usually (we hope), cleanup is easy, and so we can leave it to the GP-Fault routine. For example, most Get/Query API calls do not affect any DLL state, so faulting on a write to a user pointer would require no work (at least for that API) in the GP-Fault routine -- the GP-Fault routine may still want to clean up global resources used by the faulting task - hdc, hwnd, etc.

Address Space Separation for Applications and the System

If we implement the following guidelines in this memo, Win31 will be as robust as OS/2 *except* for separate application address spaces and running GDI/drivers at Ring 2.

1) Should we face the incompatibilities (and performance hit) of address space switching? Not to mention the work!?

2) Should we run GDI at Ring 2, to prevent access to GDI data?

Note that PMWIN (OS/2's USER) runs at Ring 3, so its data *is* accessible to OS/2 PM applications.

A *malicious* application can crash OS/2 as easily as it can crash Windows -- it only needs to find the PMWIN (USER) data segment and scribble all over it.

A "normal" application would only "accidentally" modify a segment that did not belong to it. What is the probability that a Windows app would accidentally load the selector of a system data object, and then write to that object?

We should "consider" the following work items:

1) Run parts of GDI, Drivers, and/or Kernel at a lower ring

   Moving GDI and/or drivers to Ring 1 would protect GDI from apps, and allow us to get a performance benefit with IOPL=1 by avoiding the IOPM slowdown.

2) Make more selectors read-only

   The following selectors "types" are reported by HeapWalk:

Now	Proposed	Type
R/E		Code
R/W		Data
R/W		DGroup
R/W	Variable	Module Database
R/W		Private
R/W		Private Bitmap
R/W	R/O	Resource Accelerator
R/W		Resource Cursor
R/W	R/O	Resource Dialog
R/W	R/O	Resource Font
R/W	R/O	Resource Group_Cursor
R/W	R/O	Resource Group_Icon
R/W		Resource Icon
R/W	R/O	Resource Menu
R/W	R/O	Resource String
R/W		Resource UserDefined
R/W		Task

   R/W - Read/Write
   R/E - Read/Execute
   R/O - Read-Only
   Variable - Usually Read-Only, code that needs to write the object temporarily makes the selector R/W, then reverts to R/O.

   What compatibility implications are there to changing some resources to be Read-Only?

KERNEL Notes (ralphl owns this)

- Make a list of all global cleanup work items (if any)

- Make list of all APIs that need parameter validation: <lpstr,cb> pairs, handles, enumerated types, and estimate work effort.

o Estimate work to add Local* handle checking. Right now, DEBUG version RIPs on bad handle, and goes on to use it as if it were valid (to be compatible with retail version).

o Add routine, DebugFillBuffer(LPSTR lpstr, WORD cb), to KERNEL. It fills the specified buffer with 0xFE bytes. In DEBUG builds, all API which take <pbuf,cb> argument pairs will call this routine, and then fill in the buffer with the data that was actually requested.

This will help app authors catch bugs where the allocated size of a buffer is smaller than what they told Windows -- memory outside the buffer will get a string of 0xFE values. If used as either a selector or offset, these values are likely to cause a GP Fault!

The GP-Fault handler will look to see if the fault was caused by one of these values, and if so the RIP will suggested the nature of the problem "Probable bad buffer passed to Windows. Allocated size of buffer was smaller than size Windows was told."

o Stay on known system stack when dispatching to USER to handle a GP-Fault. Make sure bottom of stack is configured correctly so that the display driver stack checking code (in BitBLT) does not panic. See viroont or raypat for details.

This prevents a recursive fault in the case where an application faults on its stack. GP-Fault cleanup must be run on a valid stack.

? How does DLL GP-Fault clean-up interact with letting apps hook GP-Faults?

o Code review all (?) of kernel with eye on how KERNEL responds in out-of-memory situations.

USER Notes (davidds owns this)

o Make a list of all global cleanup work items (est = 3 days research)

o Make list of all APIs and Messages which take <pointer,cb> pairs, and identify which ones already call DebugFillBuffer (see KERNEL section), and which ones do not.

o Make a list of all APIs with enumerated parameters, and identify which ones are already being range checked and which ones are not.

o Estimate work for making SetWindowHook robust.

o Make a list of all other robustness work that was added since Win3.0.

o Validate Menu Handles (est = 2 days of coding/testing)

GDI Notes (viroont owns this)

o Make a list of all global cleanup work items

- Make list of all APIs that need parameter validation: <lpstr,cb> pairs, handles, enumerated types, and estimate work effort.

- Tag DCs with hTask, and in GP-Fault handler call device driver for each DC owned by dying task, so that device driver (especially printer drivers -- may not be necessary for display driver) can clean up any resources pointed to by the DC (lpdev, actually).

Display Driver Notes (raypat owns this)

A GP fault handler entry point will be added to the display driver. When called by the system, this routine will examine a status word in the driver's data segment which will indicate whether the system was executing driver code at the time of the GP fault. If it was, this routine will reset the driver global state (if any) and the default h/w state. In particular, it will dump any cached fonts (in the 8514), and any thing else that might be appropriate.

The display driver contain little, if any, global state data. The most crucial aspect of robustness in the display driver is the state of the video hardware. For the vga/ega, this is a very straight forward problem of simply resetting the video h/w to the expected default state. (est = 1 day coding/testing)

The 8514, however, is another matter. It's h/w is finicky about reading/writing too many, or too few words to/from its ports. Therefore, we would have to write the GP fault handler carefully. It would have to inspect the Update Controller's status bits and, in the case of a write, write a (junk) word at a time into the input port and stop when the command operation completion bit is set. A similar technique would be used for read operations. This involves tracking down all places in the driver where we hit the hardware and possibly setting some status variable to indicate a read or write operation. At GP fault time, this variable would be examined to determine which kind of h/w reset is appropriate (read or write reset).
(est = 5 days coding/testing)

Printer Driver Notes (lins owns this)

There are three groups of things we can do:

1. Review algorithms of drivers under extreme conditions such as low memory. This is so that drivers do not GP fault by itself.
   a) Unidrv: What happens if driver cannot get any memory in GetBandFormat?
   b) Unidrv: Check for mini-driver integrity:
      1) Check LockResource return values.
      2) Check magic word in GPC data.
   c) Unidrv/Pscript: Also look at TrueType interactions:
      Check return value from EngineGetGlyphBmp.

   d) Unidrv: check the total memory needed by local variables in

each routine, especially nested routines. Since they reside on app's stack, unidrv should be careful not to overflow the app's stack. Unfortunately, there is no absolute guarantee here.

e) Review the header information for font summary. Make sure the validation process is complete and robust.

2. Parameter checking.
Most DDI calls are coming from GDI and we assume that they are valid.
There are 3 calls coming directly from apps:
ExtDeviceMode / DeviceMode
DeviceCapabilities
AdvancedDeviceMode.
Verify:
PortName, ModelName, lpOutput buffer size.

3. Fault handling during a GP Fault:
Driver must be able to get its LPDV during GP fault in order to clean up since one driver may support multiple DC's at the same time. We want to clean up for the app which faulted but do not take down other apps. With LPDV of the DC of the faulting app we can:
a) Reset all global data: Most global data in unidrv/Pscript driver are read-only except for
BOOL gDlgBusy = FALSE;
b) Free all global memory allocated - essentially execute disable(). Call CloseJob if one is currently being spooled.

Appendix A. OS/2 PMGRE (GDI-equivalent) Notes

a. All objects (hdc, hbrush, etc.) are tagged with pid, object type. Objects cannot be passed among processes without knowledge of GRE.
b. All parameters are validated; especially, handle is verified to make sure it is for correct object type (i.e., hdc == DC). This validation is done *inline*, since GRE does not reenter itself.
c. All objects are locked while in use to avoid close/use races.
d. Code in critical sections (e.g., linked list manipulation) is carefully coded to avoid windows. CLI/STI are used to prevent death in key areas.
e. ExitList routine checks and cleans up critical sections (identified by FastSafeRAM semaphores), frees objects owned by dying process, and cleans up any global variables.
f. All GRE data is at Ring 2.

Appendix B. OS/2 PMWIN (USER-equivalent) Notes

a. All objects tagged with pid, ??? (tid, hqueue)
b. All parameters are validated. Since PMWIN often calls itself, this is done in a layer. PMWIN calls itself at internal entry points, to avoid redundant parameter validation.
c. Only one thread in PMWIN at a time, so no object locking.
d. Code in critical sections does not receive special treatment.

e. ExitList routine frees objects owned by dying proces, resets FastSafeRamSems, global variables
f. Most PMWIN data is at Ring 3, accessible to apps.

-- cut here ----------

5
From jonro Wed May 1 08:35:16 1991
To: bradsi richab richt
Subject: Re: response
Date: Wed May 01 08:21:52 1991

Fortunately, the LAN Man issue has been dealt with.
I've talked to Mike and Steveb personally. You're right
Rich, we had a bit of a process breakdown.

6
From danfr Wed May 1 08:39:41 1991
To: dosbug hildegak
Cc: dos5beta hildegak wolfm
Subject: RE: To Bug # 6726... is now Bug # 1281.
Date: Wed May 01 08:37:18 PDT 1991

Reverances to bug #6726 now need to be changed to Bug #1287.

Thank you,
Bugmaster,
Danfr

>From hildegak Tue Apr 30 05:46:29 1991
To: dosbug
Cc: dos5beta hildegak wolfm
Subject: To Bug # 6726...

Date: Tue Apr 30 13:35:32 PDT 1991

The tester has now updated his Phoenix bios and has now version:
80386 advanced Rom Bios 1.02.16
The problem is still the same...
Any ideas about what I can tell this unlucky tester ?
thnx Hildegard

7
From danfr Wed May 1 08:46:02 1991
To: dosbug hildegak
Cc: dos5beta hildegak wolfm
Subject: RE: To Bug # 6726... to 1281
Date: Wed May 01 08:40:48 PDT 1991

(ACTION ITEM) GeorgeM to create border font with negative descent and report findings next week

10) Underline/StrikeOut
Gunter has added some support. No progress this week.
(ACTION ITEM) Gunter to add simulation to GDI.
LinS to investigate adding support to PCL4 driver.

11)Clipping of Accents on UpperCase Characters
GeorgeM reports that updated Ingredients font tool is "in the mail"
Once we get updated tool, we will update the fonts with the new metrics to see if clipping of accents is still occurring.

12) Do we need DDI Support For GetCharABCWidths?
No progress this week (ACTION ITEM) DavidW, RonG investigating

13) Bolting T2 into GDI for Beta I
The work to do this has just begun.

14) Identifying Win Apps that ATM doesn't work correctly with
(ACTION ITEM) TimMcC to peruse Adobe forum on CompuServe.

14
From adriank Wed May 1 09:48:00 1991
To: bradsi
Subject: RE: win4
Date: Wed May 01 10:45:38 PDT 1991

I still don't have a handle on how much work people really think win4 is. This is my first task I think. My major questions at the moment are:

- the 3.1 vs OS/2 battle will take its toll on the available resource for starting work on 4.0. Not sure what this means longer term.

- the networking stuff is cool, but why would the net companies (principally Novell) simply surrender the workstation business to MS? I think that I would look at it that way if I was asked simply to "plug into" MS's view of the net interface. Where's my value add? What distinguishes my product?

- there still seems to be a filesystem debate going on. This has a fundamental impact on everything (notably the shell.) Seems late to be (still) having this debate.

- I worry that Win32 is OS/2 revisited (it has lots of the same faces). After talkign with Muglia I am less worried about it. BUT I think we have to be very, very serious about the development tools for it. The old SDK ragbag is just hopeless. I think we need to spend lots of money and people on tools, training, docs etc etc immediately. Maybe we are, I just don't know that part of the plan yet.

Its a pity that 3.1 didn't already go out. The .1 says its no big

deal. I wonder if we'll be forced to call it 4 0 and say its a big deal? Getting Win32 out 'early' should be done via development tools I think (like Apple with System 7.0). I don't think that feature alone is worthy of its own major release.

15
From jonl Wed May 1 09:53:52 1991
To: johncon richab
Subject: Win Count
Cc: bradsi
Date: Wed May 1 09:53:41 1991

what should our 'sold copies' of Win3 be for the WinWorld (May20) keynote be?

16
From dennisad Wed May 1 09:55:45 1991
To: bradsi davidcol philba
Subject: FW: Phone Calls from "executive staff" members
Date: Wed May 01 09:50:34 1991

In case you haven't seen this yet...

>From path Wed May 1 09:46:56 1991
To: a-donae spag
Subject: FW: Phone Calls from "executive staff" members
Date: Wed May 01 09:44:40 PDT 1991

FYI below.....

>From tomsu Wed May 1 08:56:16 1991
To: admina opr recep
Cc: buckf exadmin pattih susanr tomsu
Subject: Phone Calls from "executive staff" members

Date: Wed May 1 08:50:30 1991

A series of phone calls have been received from a male caller claiming to be various members of the executive staff/board of directors. THe caller is asking for confidential information such as org charts, reporting structures, internal phone extensions, as well as information on unreleased product.

If you should receive a call like this do not give out any information. Tell the caller you will forward them to the Admin that reports to them (in past calls this has caused the caller to hang up). If the caller stays on the line and the Exec admin isn't in her office, tell the caller you will gather the info and get a number where they can be called. In past calls the caller has refused to give this information and has said he would call back for the information a few minutes