HyperPlatform User Document

Table of Contents

1. About this document

This document describes how to use HyperPlatform to develop your own hypervisor-based tools and general knowledge on hypervisor development. This document is available in multiple formats:

- HTML (http://tandasat.github.io/HyperPlatform/userdocument/)
- PDF (http://tandasat.github.io/HyperPlatform/userdocument/UserDocument.pdf)
- ODT (/Documents/UserDocument.odt)

For more high level information on HyperPlatform, see the project page.

- https://github.com/tandasat/HyperPlatform

2. Get started

2.1. Description

This chapter describes steps to create a new Visual Studio project derived from HyperPlatform and briefly explains where to modify to implement your own logic on the top of HyperPlatform.

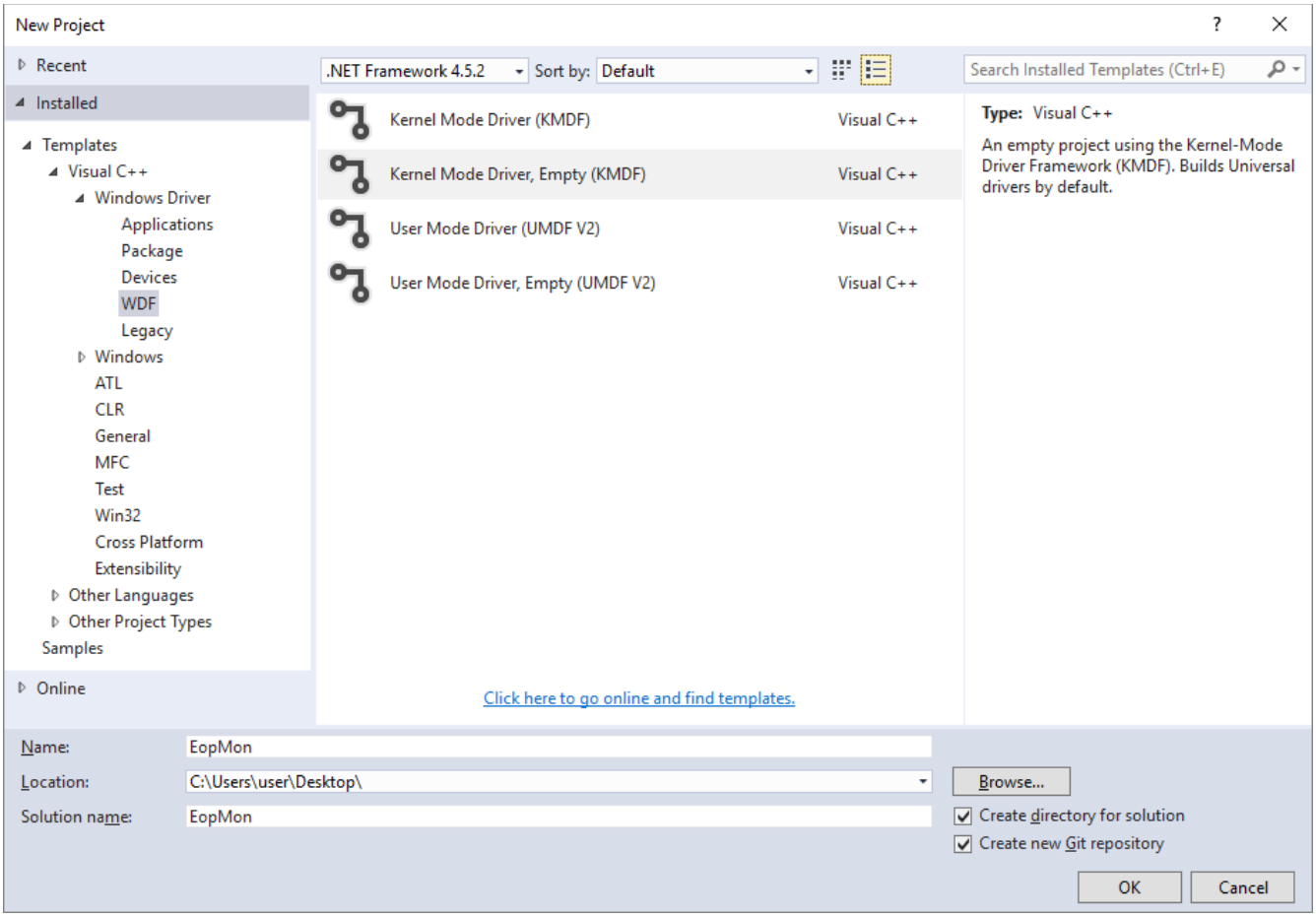
2.2. Prerequisites

You need the following packages to compile HyperPlatform:

- Visual Studio Community 2015 Update 3
  - https://www.visualstudio.com/en-us/news/releasenotes/vs2015-update3-vs
- Windows Software Development Kit (SDK) for Windows 10
  - https://dev.windows.com/en-us/downloads/windows-10-sdk
- Windows Driver Kit (WDK) 10
  - https://msdn.microsoft.com/en-us/windows/hardware/hh852365.aspx

2.3. Creating a new project

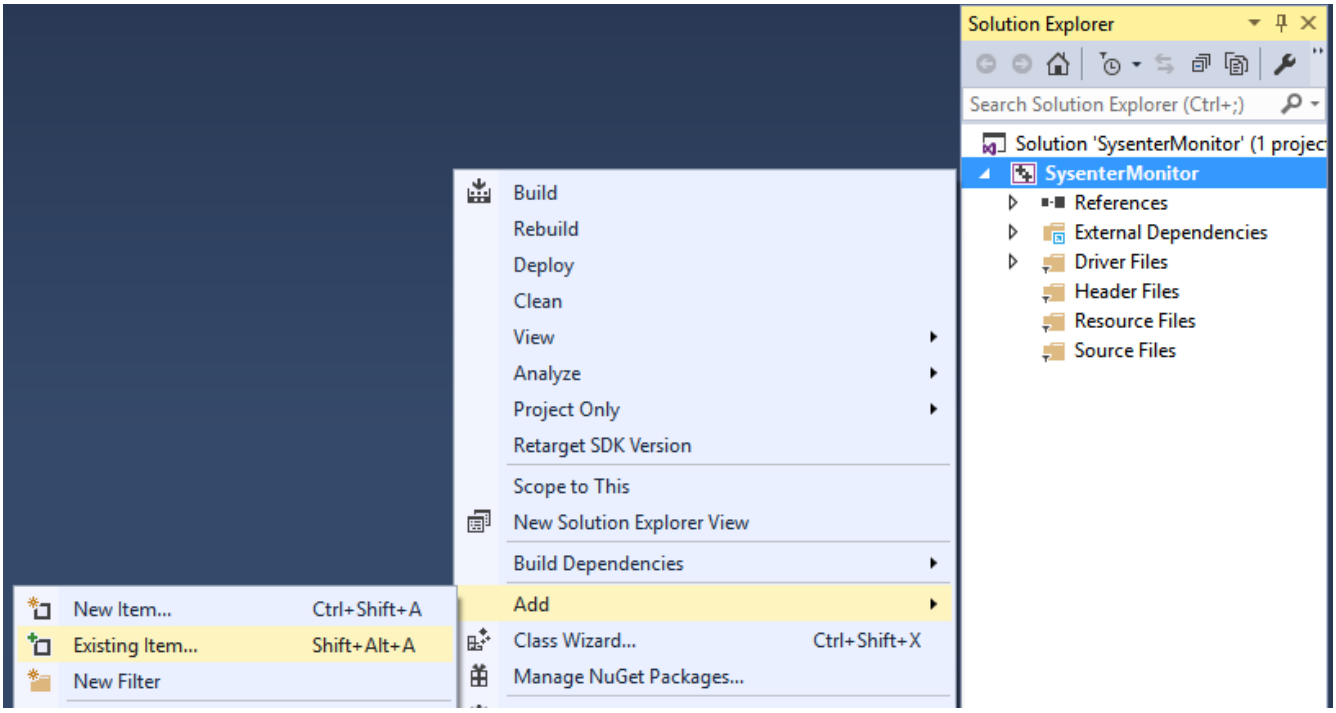
1. Create a new project for 'Kernel Mode Driver, Empty (KMDF)'. In this document, a project is named 'EopMon'.



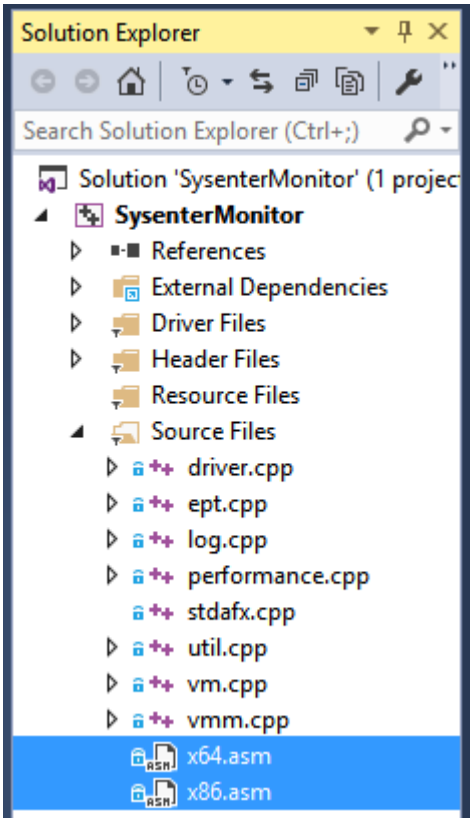
2. Place an entire HyperPlatform folder under the project folder. It can be done by copying files or using HyperPlatform as a git submodule. After placing files, a result of tree command should look like this:

```
> tree .
...
C:\USERS\USER\DESKTOP\EOPMON
├───EopMon
└───HyperPlatform
    ├───Documents
    └───HyperPlatform
        └───Arch
            ├───x64
            └───x86
```

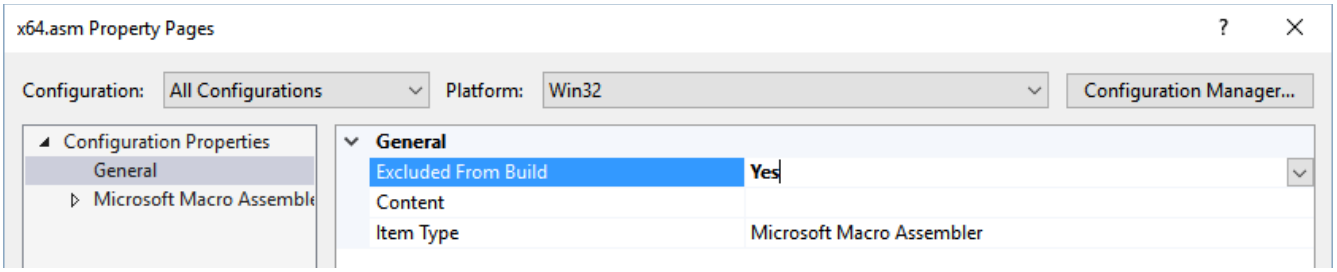
3. Add all cpp and h files under the copied HyperPlatform folder to the project.



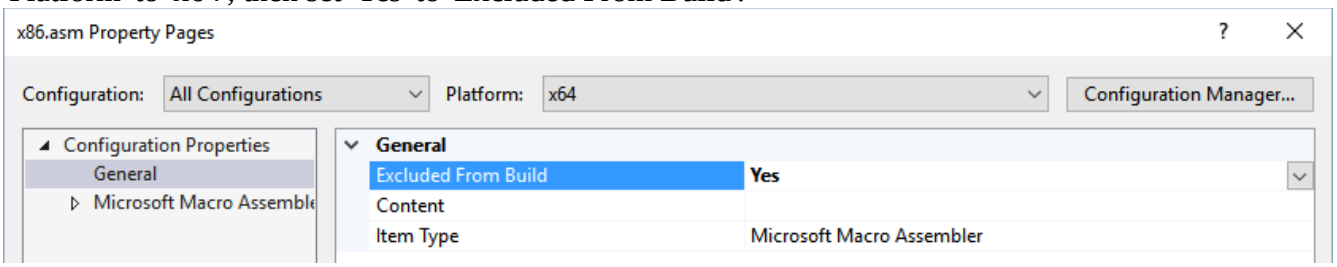
4. Add x86.asm and x64.asm files under the HyperPlatform\Arch folder to the project.



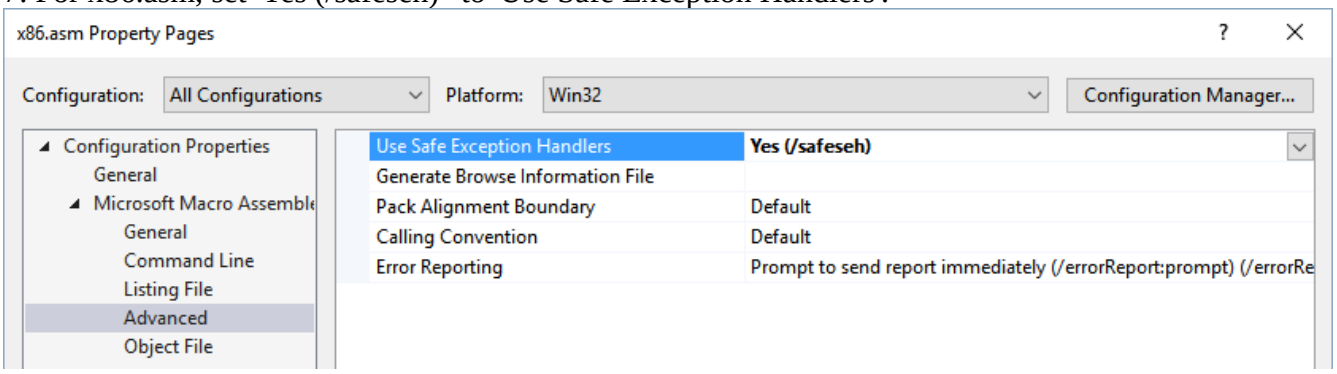
5. Open properties of the x64.asm and switch 'Configuration' to 'All Configuration' and 'Platform' to 'Win32', then set 'Yes' to 'Excluded From Build'.



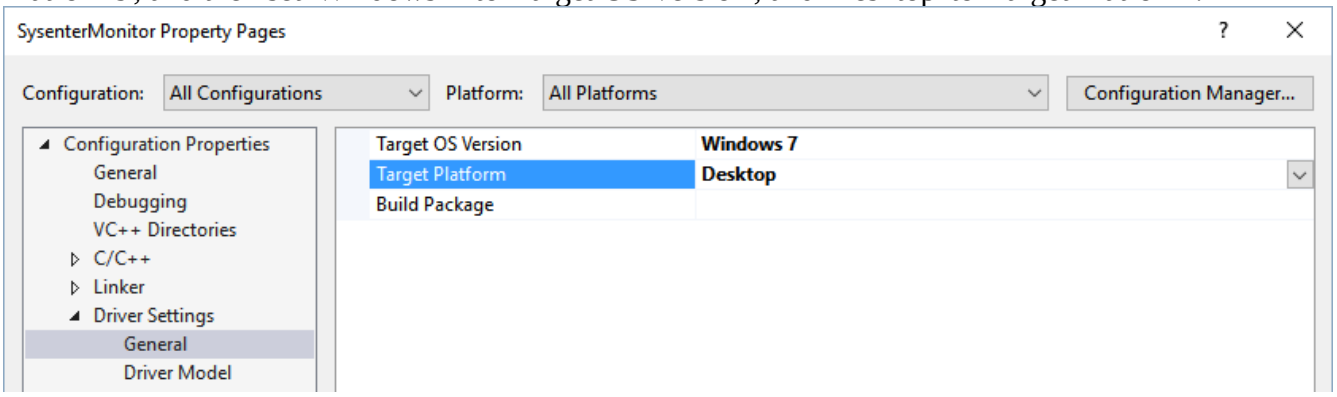
6. Similarly, open properties of the x86.asm and switch 'Configuration' to 'All Configuration' and 'Platform' to 'x64', then set 'Yes' to 'Excluded From Build'.



7. For x86.asm, set 'Yes (/safeseh)' to 'Use Safe Exception Handlers'.



8. Open properties of the project, switch 'Configuration' to 'All Configuration' and 'Platform' to 'All Platforms', and then set 'Windows 7' to 'Target OS Version', and 'Desktop' to 'Target Platform'.



9. Build the project for 'x86' or 'x64' (HyperPlatform does not support ARM architecture).

2.4. What is Next

The rests are left to your ideas, but followings is a list of locations where you may want to take a look at and modify for your own purposes:

- DriverEntry and an unload handler => driver.cpp
- Configurations of when VM-exit occurs => VmpSetupVmcs() in vm.cpp
- An address of a sysenter handler => VmpSetupVmcs() in vm.cpp
- VM-exit handlers => VmmpHandleVmExit() in vmm.cpp
- Any assembler code => x64.asm, x86.asm and asm.h

Also, there are some side projects useful to developers.

- MemoryMon
  - A hypervisor-based tool detecting execution of kernel memory where is not backed by any image files using extended page table (EPT). It is implemented on the top of HyperPlatform.
- GuardMon
  - A hypervisor-based tool monitoring some of PatchGuard activities. It is implemented on the top of HyperPlatform.
- hyperplatform_log_parser
  - A user-mode program parsing logs created by HyperPlatform. Most useful with MemoryMon currently.
- ping_vmm
  - A user-mode program knocking at HyperPlatform's “backdoor”.

Enjoy the ring -1 programming!

3. Development and Debug Tips

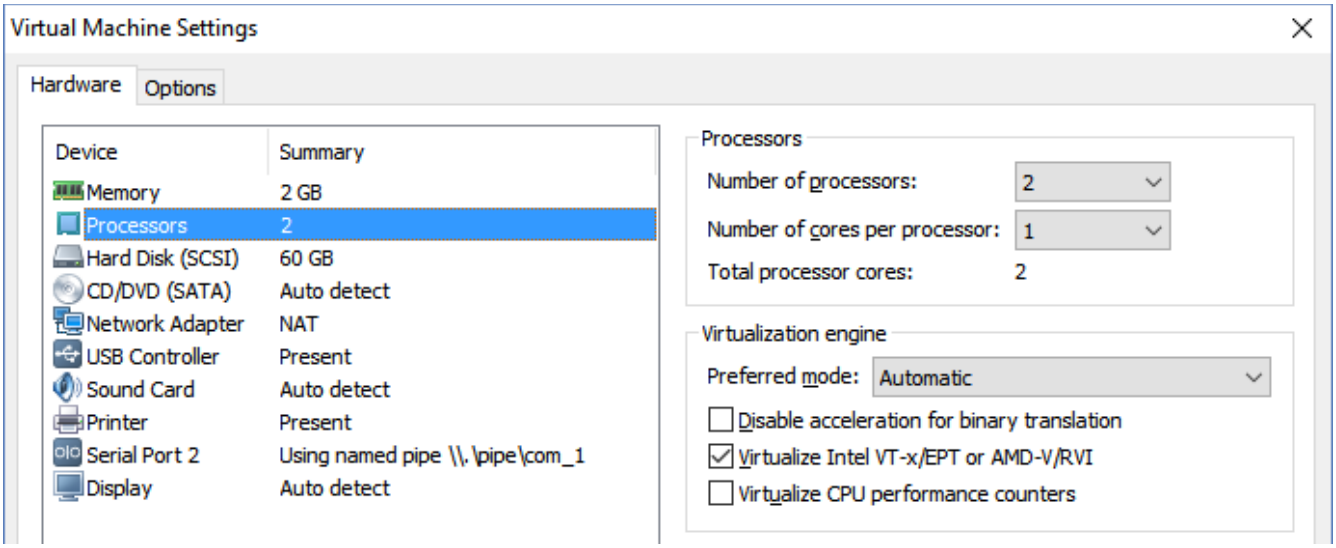
3.1. Description

This chapter explains basic technical know-how of developing and debugging hypervisors.

3.2. Using VMware Workstation

VMware Workstation supports nested hardware virtualization and allows your hypervisor to run inside a VM. In order for running a hypervisor in a VM, you need to change configurations of the VM as followings.

1. Check 'Virtualize Intel VT-x/EPT or AMD-V/RVI'



2. Open a corresponding VMX file and add the following lines.

```
hypervisor.cpuid.v0 = "FALSE"
mce.enable = "TRUE"
```

3.3. Using Bochs

Although VMware works perfect for development majority of times, it can be difficult to chase some bugs through only a kernel debugger. For example, VMware provides no capabilities to trace why a VMX instruction failed or why a guest suddenly received a triple fault VM-exit. By using Bochs, you are able to trace exactly why it happened with Bochs' emulation source code.

Following sections describe steps to run a hypervisor in Bochs with the Visual Studio Debugger.

3.3.1. Compiling Bochs

This section explains how to compile and run Bochs with Visual Studio.

1. Download and install Cygwin. At least, gcc and zip commands are required.

https://www.cygwin.com/

2. Download source code from SVN and extract it.

http://bochs.sourceforge.net/getcurrent.html

3. Open .conf.win32-vcpp and delete below three lines (as they failed to link when I tested).

```
            --enable-usb \
            --enable-usb-ohci \
            --enable-usb-xhci \
```

4. Open the Cygwin console, navigate to the extracted directory, and run the following commands.

```
$ sh .conf.win32-vcpp
$ make win32_snap
```

5. Extract created bochs-20160107-msvc-src.zip, and open bochs-20160107\vs2013\bochs.sln

6. In the Solution Explorer, navigate to the win32res.rc file under the "bochs" project.

7. Right click > View Code, and comment out the below line as it causes a link error.

```
// Manifest for both 32-bit and 64-bit Windows
1 24 build/win32/bochs.manifest
```

8. Build the bochs project. You will see some errors in other projects but can ignore them. It should create boches.exe under bochs-20160107\obj-release or bochs-20160107\obj-debug according with build configuration.

3.3.2. Configuring a VM

This section explains how to install a VM on Bochs.

1. Download a pre-compiled Bochs (one with an exe extension).

https://sourceforge.net/projects/bochs/files/bochs/2.6.8/

2. Through the start menu, run Programs\Bochs 2.6.8\Disk Image Creation Tool (bximage.exe)

3. Create an enough size of a hard disk image. 2048MB should suffice for installing most of modern Windows versions.

```
1. Create new floppy or hard disk image
2. Convert hard disk image to other format (mode)
3. Resize hard disk image
4. Commit 'undoable' redolog to base image
5. Disk image info

0. Quit

Please choose one [0] 1<Enter>

Create image

Do you want to create a floppy disk image or a hard disk image?
Please type hd or fd. [hd] <Enter>
```

```
What kind of image should I create?
Please type flat, sparse, growing, vpc or vmware4. [flat] <Enter>

Enter the hard disk size in megabytes, between 10 and 8257535
[10] 2048<Enter>

What should be the name of the image?
[c.img] <Enter>

Creating hard disk image 'c.img' with CHS=4161/16/63

The following line should appear in your bochsrc:
  ata0-master: type=disk, path="c.img", mode=flat
(The line is stored in your windows clipboard, use CTRL-V to paste)
```

This should create an image file under the below folder.

%USERPROFILE%\AppData\Local\VirtualStore\Program Files (x86)\Bochs-2.6.8

4. In the same folder, create a bxrc (eg. bochsrc.bxrc) with the below contents.

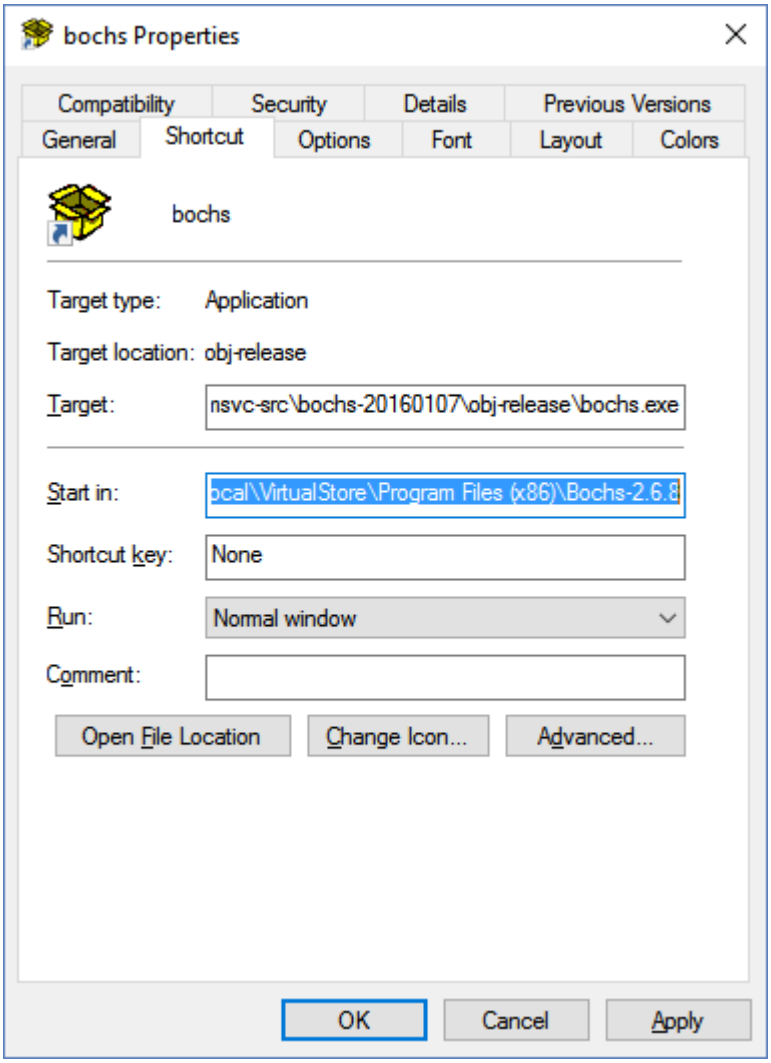
```
# configuration file generated by Bochs
plugin_ctrl: unmapped=1, biosdev=1, speaker=1, extfpuirq=1, parallel=1, serial=1,
gameport=1, ne2k=1, e1000=1
config_interface: win32config
display_library: win32
memory: host=1024, guest=1024
romimage: file="C:\Program Files (x86)\Bochs-2.6.8/BIOS-bochs-latest"
vgaromimage: file="C:\Program Files (x86)\Bochs-2.6.8/VGABIOS-lgpl-latest"
boot: cdrom, disk
floppy_bootsig_check: disabled=0
# no floppya
# no floppyb
ata0: enabled=1, ioaddr1=0x1f0, ioaddr2=0x3f0, irq=14
ata0-master: type=disk, path="c.img", mode=flat, cylinders=20805, heads=16,
spt=63, model="Generic 1234", biosdetect=auto, translation=auto
ata0-slave: type=cdrom, path="Win7x86.iso", status=inserted, model="Generic 1234",
biosdetect=auto
ata1: enabled=1, ioaddr1=0x170, ioaddr2=0x370, irq=15
ata1-master: type=cdrom, path="d.iso", status=inserted, model="Generic 1234",
biosdetect=auto
ata1-slave: type=none
ata2: enabled=0
ata3: enabled=0
pci: enabled=1, chipset=i440fx
vga: extension=vbe, update_freq=5, realtime=1
cpu: count=1, ips=1200000000, model=corei7_sandy_bridge_2600k,
reset_on_triple_fault=1, cpuid_limit_winnt=0, ignore_bad_msrs=1, mwait_is_nop=0
print_timestamps: enabled=0
debugger_log: bochs_debugger.log
magic_break: enabled=0
port_e9_hack: enabled=0
private_colormap: enabled=0
clock: sync=none, time0=local, rtc_sync=0
# no cmosimage
# no loader
log: bochs.log
logprefix: %t%e%d
debug: action=ignore
info: action=report
error: action=report
panic: action=ask
keyboard: type=mf, serial_delay=250, paste_delay=100000, user_shortcut=none
mouse: type=ps2, enabled=0, toggle=ctrl+alt
sound: waveoutdrv=win, waveout=none, waveindrv=win, wavein=none, midioutdrv=win,
midiout=none
speaker: enabled=1, mode=sound
parport1: enabled=1, file=none
parport2: enabled=0
com1: enabled=1, mode=null
com2: enabled=0
com3: enabled=0
com4: enabled=0
ne2k: enabled=0
e1000: enabled=0
```

Here is a list of some notable points of this configuration:

- References to a bios file under C:\Program Files (x86)\Bochs-2.6.8 (You may need to change the path)
- A VM can see contents of Win7x86.iso as a CD in the same folder.
- A Processor is 1.2 GHz, which is acceptably fast to install and run Windows 7

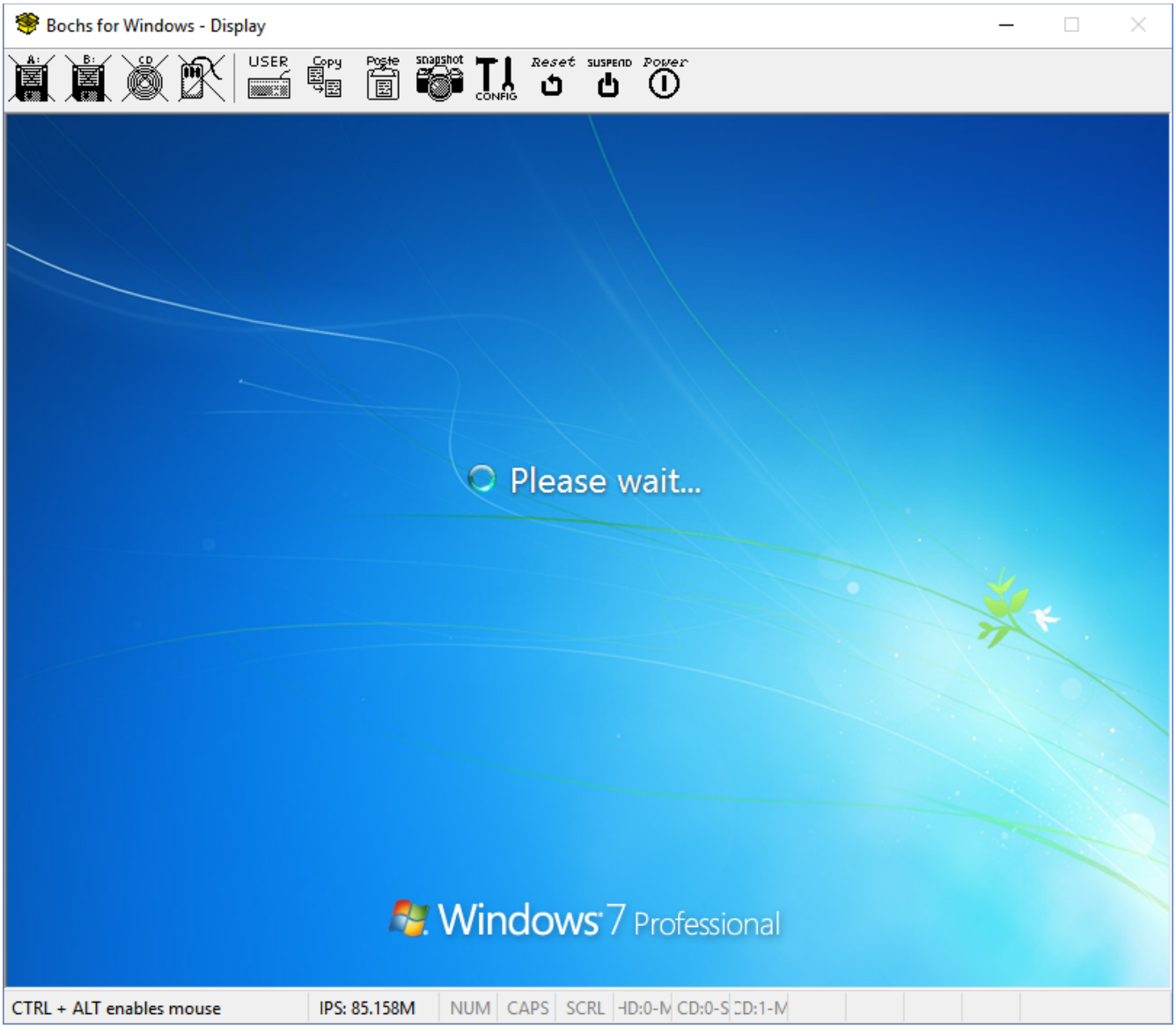
- Logs are stored in to bochs.log in the same folder
- No network connectivity
- Uni-processor system without the Bochs debugger (you can change those if needed, although you have to re-compile Bochs accordingly. See Bochs' document for details).

5. In the same folder, create a shortcut file to the compiled bochs.exe, and change 'Start in' to the current folder.



6. Run the shortcut file, and load the bxrc file via the Load button, then 'Start'.

7. Install Windows OS as usual. Please wait.. Calm down. It is not as fast as VMware.



3.3.3. Debugging code through the Visual Studio Debugger

Once you configured a VM for Bochs, you may want to run your code in Bochs to see what happens. From the point of view of debugging, there are some factors that can affect result of tests (behaviour of your code), however:

1. Bochs may have a bug leading to a different result

2. Bochs' VM has fewer hardware resulting in less frequent VM-exit and a different physical memory layout

3. Bochs' VM is a uni-processor and has no internet connectivity if you use the above configuration

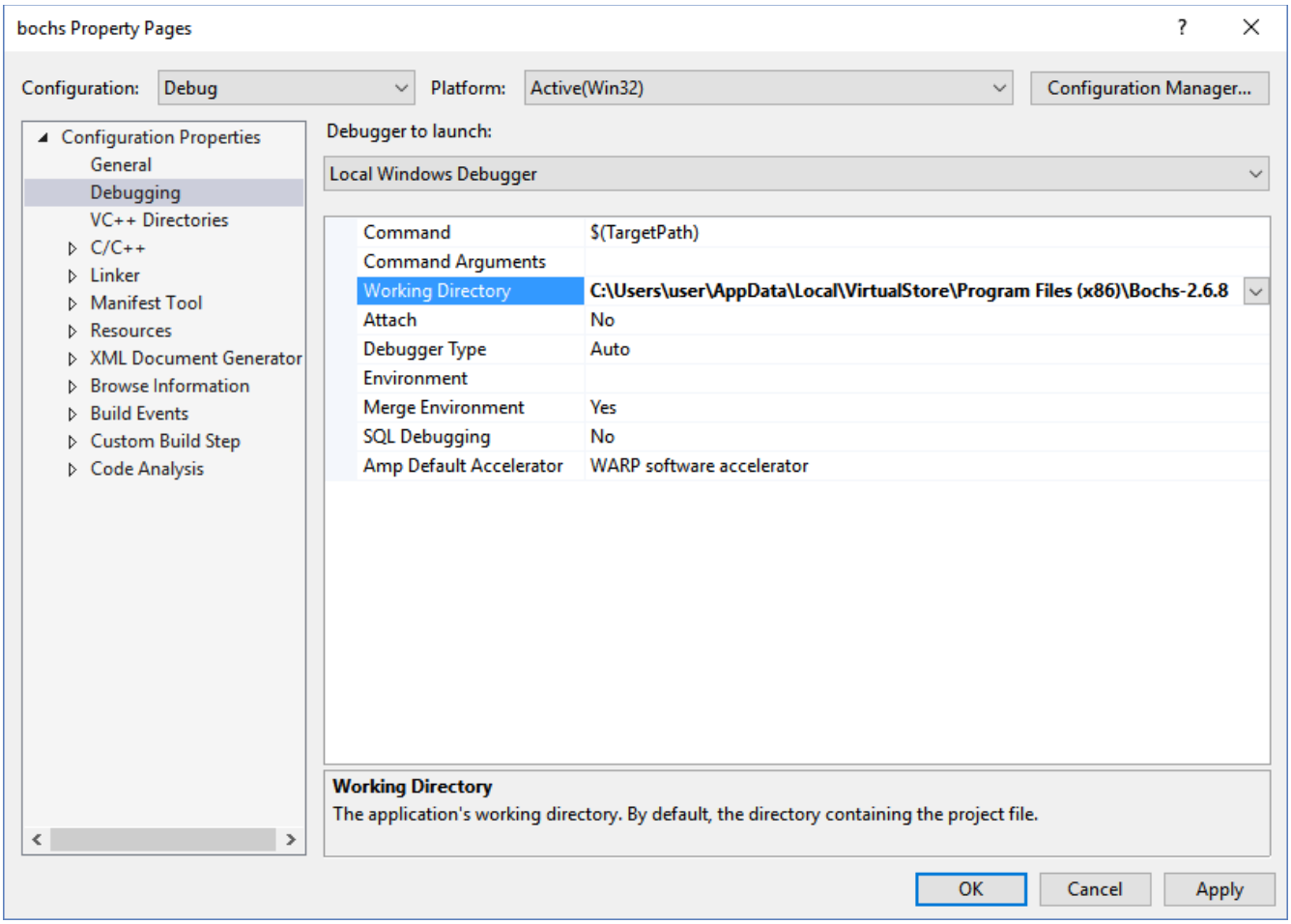
To run your code, you have to copy your files onto the VM. The most straightforward way may be creating an ISO file from a folder and read the ISO file from the VM. To create an ISO file, AnyToISO

might be an option.

> http://www.crystalidea.com/

All logs from Bochs are going to be saved in bochs.log. If you faced any distinguishable error message in it after executing your code, you can search the message from Bochs' source code, and then can set a break point in Visual Studio.

Then, on Visual Studio, open properties of the bochs project, and set the path where the bxrc file is stored to 'Working Directory', and just start Bochs with a debugger and start investigate what is going on inside a processor.



3.4. Coding Tips

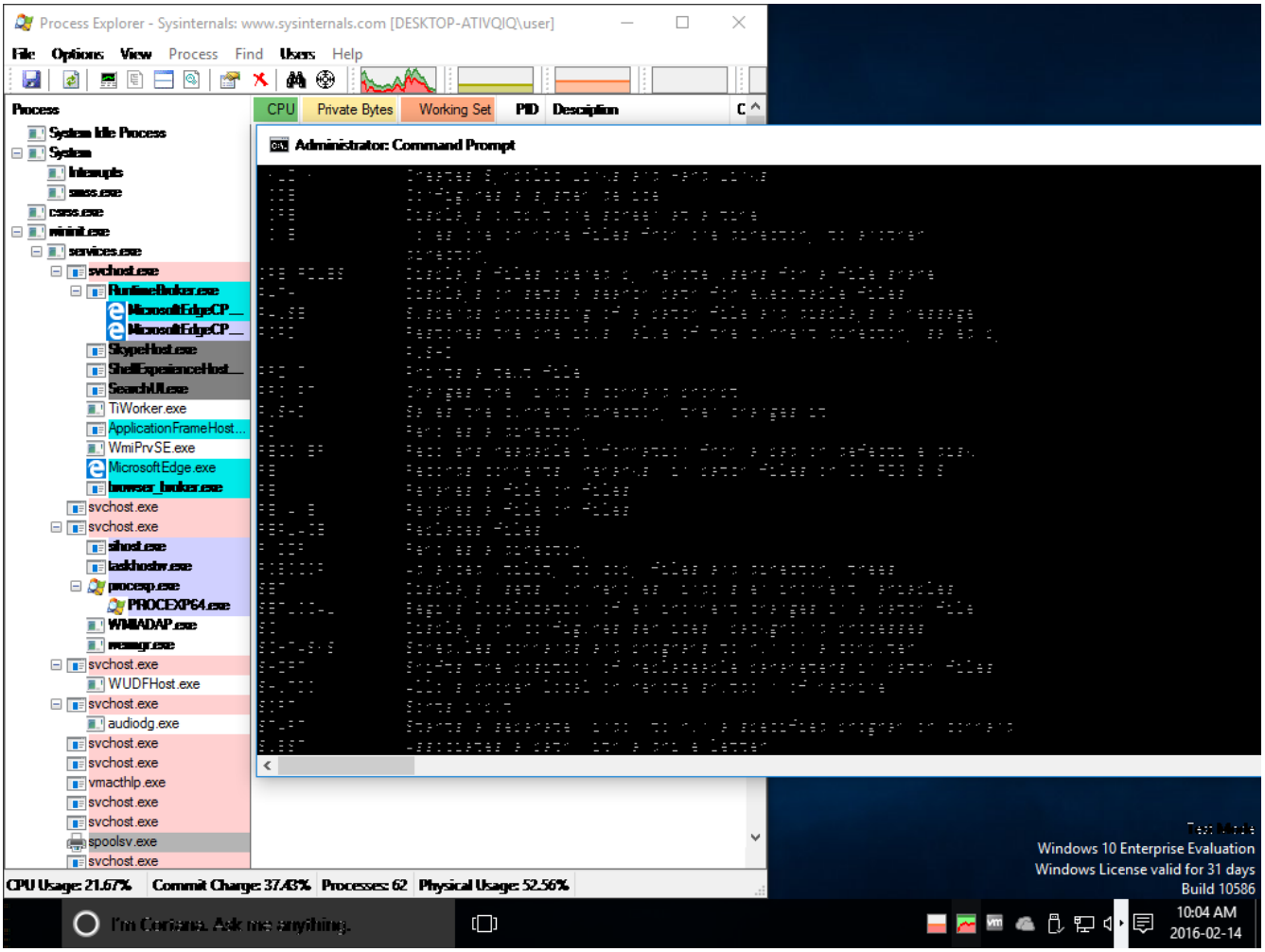
While developing hypervisor shares many aspects with developing an ordinary software drivers, there are some peculiarity all developers should be aware of. This section explains those gotchas as well as some coding facilities HyerPlatform provides.

3.4.1. Gotcha: Avoid using API inside a VM-exit handler

Since a VM-exit handler is executed without receiving interrupts under a high IRQL, it is technically

illegal to use API unless it is allowed to be called on HIGH_LEVEL IRQL. Violation of this rule can lead to unexpected behaviour. For example, using RtlPcToFileHeader in a VM-exit handler can cause this incomprehensible broken font, even though implementation of the RtlPcToFileHeader seems to be safe to use. See the thread on Issue #3 for more details.

- https://github.com/tandasat/HyperPlatform/issues/3#issuecomment-230384767



Instead, use your own implementation, call API beforehand or in a different context. In order to avoid calling API in a VM-exit handler, HyperPlatform per-allocates memory in a safe context and provides log functions that are *mostly* safe to call inside a VM-exit handler (see log.h for details).

Note that some sample projects violate this rule to simplify code instead of being safe, but it is not encouraged, especially when reliability is important.

3.4.2. Gotcha: Do not step-in to VMLAUNCH and VMRESUME

The debugger will never return control to you, and the system will hang. Instead, set breakpoints to where a guest will execute next.

3.4.3. Gotcha: Use breakpoints moderately in a VM-exit handler

Do not put software breakpoints everywhere in a VM-exit handler. Although it seems to be fine in most cases, in some situation, the debugger does not gain control from the target system, and the system just freezes when int 3 is executed. This seem to happen especially in case of an LDT or TR access VM-exit.

Consider using log instead when breakpoints seem to cause the issue.

3.4.4. Gotcha: Use a guest CR3 value for memory access

A VM-exit handler is executed with a hypervisor's CR3 specified through a VMCS field and may unable to see the same memory contents as what a guest sees. Because of this, accessing memory from a VM-exit handler for the sake of a guest may result in referencing completely wrong contents despite the same linear address. Also, the same thing applies when resolving a physical address or a page frame number from guest's linear address because a contents of a page table can be different if a value of CR3 is different too.

In order to avoid this issue, update the current CR3 value with guest's CR3 value before using a guest linear address.

3.4.5. Gotcha: Incompatibility with the Driver Verifier

HyperPlatform is not compatible with the Driver Verifier since it fails the Monitoring Stack Switching test, a part of Automatic Checks which cannot be disabled, due to the fact that a VM-exit handler uses its own thread stack. For more details, see the Issue #4.

3.4.6. Tips: Using STL

HyperPlatform provides a header to use the C++ Standard Template Library (STL) in the project for ease of programming. The following configuration allows you to use the STL for user-mode programs by changing include directories and providing missing functions.

1. Open properties of your project and switch 'Configuration' to 'All Configuration' and 'Platform' to 'All Platforms'.

2. Navigate to 'VC++ Directories' and add '$(VC_IncludePath)' to 'Include Directories'. After that it should look like this;

    $(VC_IncludePath);$(IncludePath)

3. Include kernel_stl.h followed by any STL headers in any source or header files.

```
#include "../HyperPlatform/HyperPlatform/kernel_stl.h"
#include <vector>
#include <string>
#include <memory>
#include <set>
#include <algorithm>
```

Now, you should be able to use the STL. In order to use a lambda expression, a function has to have C+

+ linkage. You can use extern C++ for your function, for example,

```
// Uses STL for test
extern "C++" void TestStl() {
  std::vector<std::unique_ptr<std::wstring>> strs;
  for (auto i = 0ul; i < 10; ++i) {
    strs.push_back(
        std::make_unique<std::wstring>(L"i = " + std::to_wstring(i)));
  }
  // Using a lambda expression
  std::for_each(strs.cbegin(), strs.cend(), [](const auto &str) {
    HYPERPLATFORM_LOG_DEBUG("%S", str->c_str());
  });
}
```

Note that the STL in drivers is not supported by Microsoft (Visual Studio) and a use of it may potentially cause unexpected run-time errors. For example, any memory allocation inside STL calls ExAllocatePoolWithTag() and may result in unexpected behaviour, especially in a VM-exit handler.

3.5. General Debugging Tips

This section briefly mentions some of useful techniques for debugging hypervisors.

3.5.1. General Tips

There are some useful tools and tricks freely available for general driver development. See following links for more details:

- For faster communication between a kernel debugger and a target system, VirtualKD
  - http://virtualkd.sysprogs.org/
- For saving your favourite Windbg windows layout, follow instructions explained in this page.
  - https://briolidz.wordpress.com/2013/11/28/windbg-custom-workspace-and-color-scheme/
- For running arbitrary commands at start up of Windbg, use the -c parameter for Windbg. Here is an example of usage.
  - https://github.com/tandasat/windbg_init/blob/master/windbg_init.txt

3.5.2. Multi-processors vs Uni-processor

Although I strongly recommend to configure a test system with multi-processors and design code for it from the beginning, running code in a uni-processor system is the best way to tell if the issue is race condition. The msconfig command lets you change a number of processors to be used.

3.5.3. Debugger vs Non-Debugger

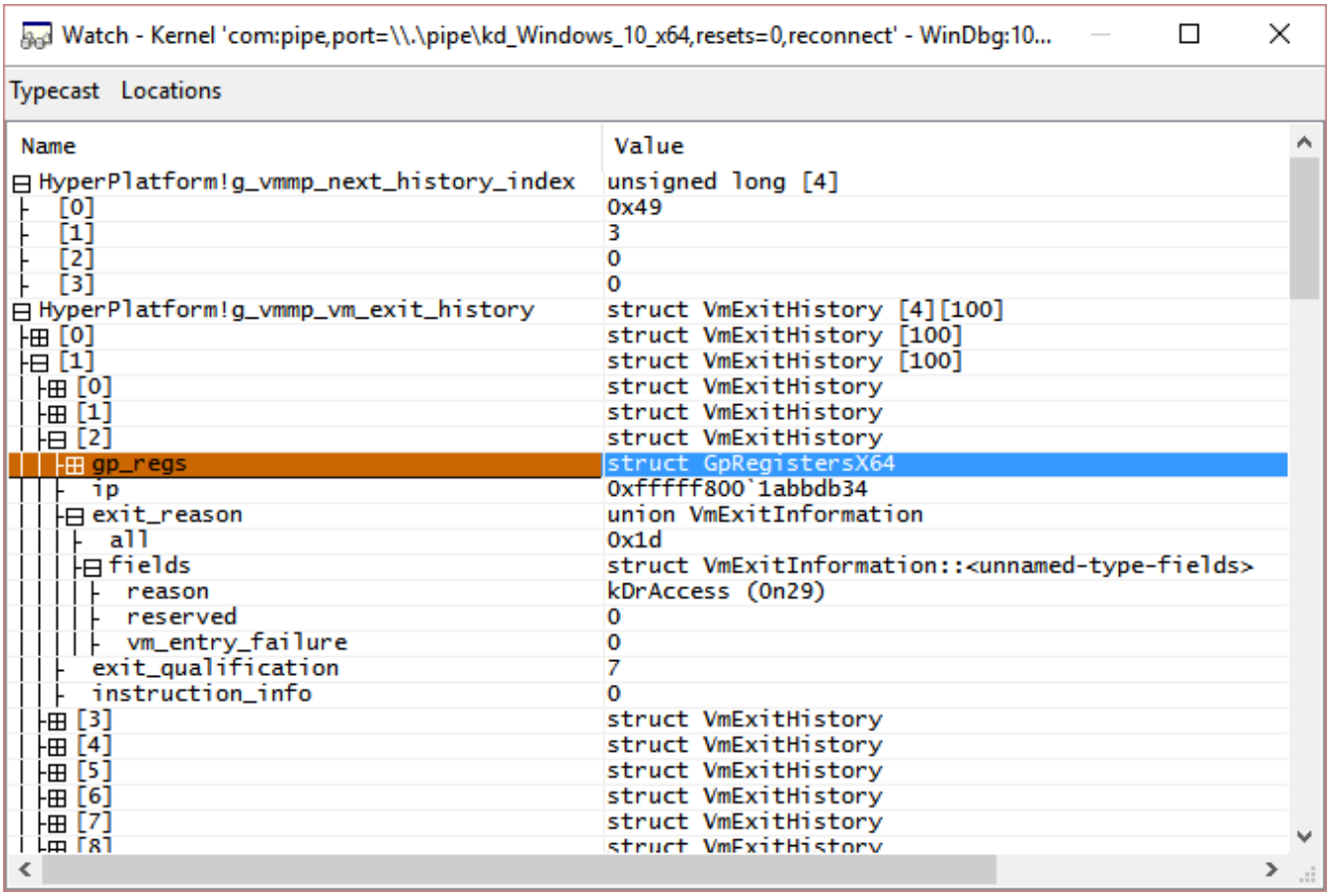
It can be helpful to run code without attaching a kernel debugger and let a system die with a bug check. This is particularly true when a debugger get control with an odd guest status or does not get a control at all. There may be an issue affecting code to pass control to a debugger properly.

The HYPERPLATFORM_COMMON_BUG_CHECK macro can be used to issue a bug check.

3.5.4. Enabling VM-exit history

HyperPlatform provides an option to save last few VM-exit and guest's status at the moment. It can be a powerful diagnostics tool if a debugger does not spot an issue exactly.

This feature is disabled by default. To activate it, set true to kVmmpEnableRecordVmExit in vmm.cpp and modify kVmmpNumberOfRecords and kVmmpNumberOfProcessors if necessary. History is stored in a g_vmmp_vm_exit_history variable. It is circular buffer, and its latest position is obtained by g_vmmp_next_history_index – 1.



3.5.5. Disabling unnecessary VM-exit

By default, HyperPlatform request VM-exit for many actions and executes pass-though style VM-exit handlers. Although authors are working hard to make HyperPlatform stable, there could be bugs in those pass-through code.

To narrow down a scope of potential buggy code, you can disable any of VM-exit by changing set up code in VmpSetupVmcs() if you want.

3.5.6. Using a real device

Do test your code with a real device to spot slight bugs. Since real devices have more devices, tendency of memory access and layouts of physical memory can often be different from those of virtual

machines; this may highlight new, hidden bugs.

3.5.7. Taking a memory dump from a VMware Virtual Machine

Apart from regular kernel debugging for a VM on VMware, you can take a full physical memory dump of the VM and analyze it with Windbg. It comes in to handy when a VM hanged without giving control to a debugger. To take a dump file, follow the below steps.

1. Suspend a VM (Ctrl + J)

2. Navigate to where snapshot files are stored and run the vmss2core command under the VMware Workstation directory with names of the latest vmsn and vmem files. For instance, commands look like this:

```
> cd "C:\Users\user\Documents\Virtual Machines\Windows 8 x64"

> "C:\Program Files (x86)\VMware\VMware Workstation\vmss2core-win.exe" -W8
"Windows 8 x64-Snapshot45.vmsn" "Windows 8 x64-Snapshot45.vmem"

vmss2core version 2452889 Copyright (C) 1998-2015 VMware, Inc. All rights
reserved.
scanning pa=0 len=0x10000000
Cannot translate linear address 7ff7d12b1b00.
Cannot read context LA from PRCB.
...
... 2020 MBs written.
... 2030 MBs written.
... 2040 MBs written.
Finished writing core.
```

Note that vmss2core comes with VMware Workstation by default (version 2780323) does not seem to be functioning (always generates 0 byte of empty files). If that the case for you too, download version 2452889 from VMware's website.

> https://labs.vmware.com/flings/vmss2core

3. Then you can give a created memory.dmp file to the debugger.

```
 > windbg -z memory.dmp
```

4. Contribution

4.1. Description

While HyperPlatform is intended to be a base of your own hypervisor projects, you are also encouraged to contribute to the HyperPlatform project. Free free to report issues, request changes, or ask anything.

4.2. General Coding Style Guide

HyperPlatform follows the Google C++ Style Guide in naming convention. Other conventions are being evaluated.

> https://google.github.io/styleguide/cppguide.html

4.3. Names

All function and global variable names should have prefixes to represent which file belongs to. Abbreviation is allowed for prefixes. For example, LogInitialization.

All macros names should have prefix HYPERPLATFORM_ before a prefix from a file name. For example, HYPERPLATFORM_LOG_DEBUG.

All function, macro and global variables that are not exposed to or not meant to be used from outside a file should have 'p' or 'P' at the end of their prefixes. For example, kLogpPoolTag.

5. Contacts

Any comments, questions and suggestions are welcome.

- Igor Korkin
  - @Igorkorkin or igor.korkin@gmail.com
- Satoshi Tanda
  - @standa_t or tanda.sat@gmail.com