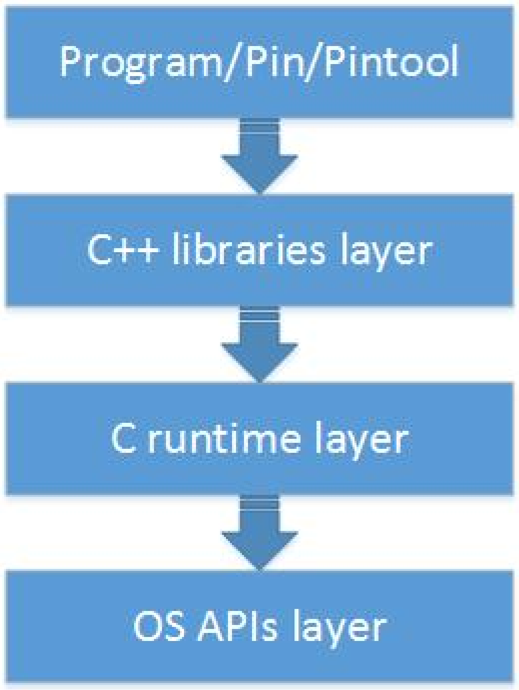
PinCRT overview

PinCRT architecture

PinCRT composed of 3 individual software layers, each of them depends on the previous layers:



- OS-APIs
    - Implements a common set of functions to interact with the operating systems.
    - Every supported operating system has its own OS-APIs implementation.

- C Runtime
    - Implements basic C functions like printf(), strcpy(), etc.
    - Provides process initialization/termination (e.g. calling C++ constructors).
    - Implements “Compiler runtime” – functions that explicitly instrumented by the compiler (e.g. 64 bit integer support on 32 bit architecture).
    - Provides a way to interact with the loader (on Unix-like OS).
    - Based on the implementation of libc for Android (bionic).

- C++ libraries
    - Implements C++ standard library functionality (e.g. STL).
    - Provides limited C++ runtime support:
        - Doesn't support exception handling.

        - Doesn't support operations that requires runtime type information (RTTI) - For example dynamic cast.
    - Based on STLPort implementation.

Technically, what is PinCRT?

- A set of headers that replace (mostly) the C runtime headers (e.g. stdio.h).
- A set of libraries (dynamic and/or static) that Pin tool can link with.

So, this means that in order to build a Pin tool with PinCRT we must:

- Compile the tool sources with PinCRT's headers (by replacing the include path).
- Link the tool object files with PinCRT's libraries (dynamically or statically).

Important note:

- Pin tool linked against PinCRT cannot be linked against any native system library (e.g. libc.so, msvcrt*.dll, etc.).
- There are several native system headers that can be included from a PinCRT enabled source file (see a list of them in "migrating from native CRT to PinCRT" section), but as a general rule you must not include any native system header when building for PinCRT (just use the PinCRT's replacements).

PinCRT features

- OS agnostic:

Provides the same interface (API) and the same behavior for C runtime on all platform.

- Compiler agnostic:

You can build your code with any compiler, then link it to the same binaries of PinCRT.

- Implements its own thread local storage

Doesn't share the thread local storage with glibc, MS-CRT, etc. This is a special requirement for binary instrumentation environment.

Supported operating systems

Currently PinCRT supports these operating system:

- Linux
- OS X
- Android
- Windows

Build with PinCRT for Linux, Android, or OS X

Building with GCC or clang

Compiler flags that needs to be used:

Macros:

- For all platforms:

```
-D__PIN__=1 -DPIN_CRT=1
```

- For IA32:

```
-DTARGET_IA32
```

- For Intel64:

```
-DTARGET_IA32E
```

- For Linux:

```
-DTARGET_LINUX
```

- For Android:

```
-DTARGET_ANDROID
```

- For OS X:

```
-DTARGET_MAC
```

Compiler flags:

```
-fno-exceptions -funwind-tables -fno-rtti –fno-stack-protector
```

- Note that –fno-exceptions, and -fno-rtti are C++ specific (they're only supported in g++, not gcc).
- Note that some compilers might not support some of these switches

Include path:

```
-isystem $(PIN_ROOT)/extras/stlport/include \
-isystem $(PIN_ROOT)/extras/libstdc++/include \
-isystem $(PIN_ROOT)/extras/crt/include \
-isystem $(PIN_ROOT)/extras/crt/include/arch-$(BIONIC_ARCH) \
-isystem $(PIN_ROOT)/extras/crt/include/kernel/uapi \
-isystem $(PIN_ROOT)/extras/crt/include/kernel/uapi/asm-x86
```

Whereas:

```
BIONIC_ARCH is 'x86' for IA32, and 'x86_64' for Intel64
PIN_ROOT is the root directory of the Pin kit
```

Linker flags:

First of all, all dependent libraries from the linker command line, denoted by –l (small l) should be removed. These need to be replaced with their PinCRT counterparts.

Add these libraries to the linkage command line:

```
-nostdlib -lc-dynamic -lm-dynamic -lstlport-dynamic -
L$(PIN_ROOT)/$(TARGET)/runtime/pincrt
```

Whereas:

```
TARGET is 'ia32' for IA32, and 'intel64' for Intel64
PIN_ROOT is the root directory of the Pin kit
```

If you're building a PinTool that was used to link with pindwarf, you should link with pin3dwarf instead. I.e. replace -lpindwarf to -lpin3dwarf.

- Linker flags only for Mac:

```
-Wl,-no_new_main
```

Build with PinCRT for Windows

Building with Visual Studio

Compilation:

Macros:

- For all platforms:

```
/D__PIN__=1 /DPIN_CRT=1 /DTARGET_WINDOWS
```

- For IA32:

```
/DTARGET_IA32 /D__i386__
```

- For Intel64:

```
/DTARGET_IA32E /D__LP64__
```

Compiler flags:

```
/GR- /GS- /EHs- /EHa- /FP:strict /Oi-
```

Include Path:

```
-I$(PIN_ROOT)/extras/stlport/include
-I$(PIN_ROOT)/extras
-I$(PIN_ROOT)/extras/libstdc++/include
-I$(PIN_ROOT)/extras/crt/include
-I$(PIN_ROOT)/extras/crt
-I$(PIN_ROOT)/extras/crt/include/arch-$(BIONIC_ARCH)
-I$(PIN_ROOT)/extras/crt/include/kernel/uapi
-I$(PIN_ROOT)/extras/crt/include/kernel/uapi/asm-x86
```

Whereas:

```
BIONIC_ARCH is 'x86' for IA32, and 'x86_64' for Intel64
PIN_ROOT is the root directory of the Pin kit
```

Additional include file:

You should add this switch to the compilation command line to include this file in every compilation:

```
/FIinclude/msvc_compat.h
```

__Important__:

If you're including Windows.h from one of you source files:

We provide our own replacement for this header file (since including Windows.h usually includes all MS-CRT stuff that we don't want to get). Our Windows.h replacement is actually a "wrapper" header for the original Windows.h that forces it to declare only the thing we wanted. In order to do so we must know where to find the original Windows.h file (located in Windows SDK). In that case, you'll need to add this to the compilation flags:

```
/D_WINDOWS_H_PATH_="$(ORIGINAL_WINDOWS_H_PATH)"
```

__Linker flags__:

First of all, Microsoft's libc libraries (e.g.: libcpmt.lib or libcmt.lib) should should be removed from the linker command line. These need to be replaced with their PinCRT counterparts.

Add these libraries to the linkage command line:

```
/NODEFAULTLIB stlport-static.lib m-static.lib c-static.lib os-apis.lib
ntdll-$(BITNESS).lib
```

Whereas:

```
BITNESS is '32' for IA32, and '64' for Intel64
```

__Libraries search path__:

```
/LIBPATH:$(PIN_ROOT)/$(TARGET)/runtime/pincrt
/LIBPATH:$(PIN_ROOT)/$(TARGET)/lib-ext
```

Whereas:

```
TARGET is 'ia32' for IA32, and 'intel64' for Intel64
PIN_ROOT is the root directory of the Pin kit
```

__Ignore linker warnings__:

```
It is safe to ignore linker warnings #4210, and #4049.
Add these switches to the linker command line to suppress these
warnings:
```

```
/IGNORE:4210 /IGNORE:4049
```

Additional objects to link

For every libc, there is a static part that must be statically linked with all executables/shared objects. The exact object files to link depends on whether you build an executable or shared object.

For building a shared library or PinTool:

- This object file must be first in the linkage command:

```
crtbeginS.o
```

For building an executable or stand-alone tool:

- This object file must be first in the linkage command:

```
crtbegin.o
```

The above object files are located at

```
$(PIN_ROOT)/$(TARGET)/runtime/pincrt
```

Whereas:

```
TARGET is 'ia32' for IA32, and 'intel64' for Intel64
PIN_ROOT is the root directory of the Pin kit
```

Migrating from native CRT to PinCRT

Get rid of OS-specific functions calls

As a general rule, PinCRT supports only ISO C functions.
So, in general, no OS-specific functions are implemented in PinCRT (although there are some implementation of OS-specific function for some functions we chose).

Make sure you're using the correct header files

When building a Pin tool with PinCRT, all compilation units (I.e. source files that compiled to object files) must include only the two categories of header files list below.
If one of the header files that was being include does not fall into one of those two categories than there is a good chance that that your Pin tool won't be able to compile, link, or run.
The two categories of header files supported by PinCRT:

1. PinCRT and Pin header files (that reside on one of the Pin kit subdirectories).
2. System headers which are PinCRT-enabled – see the list of PinCRT enabled system headers.

How to make sure my source includes only PinCRT compatible headers:

We recommend using the GCC/Clang "-E" flag,or Visual Studio's "/E" flag to dump the raw output of the C/C++ preprocessor output.
From this output you can see all the files that are being included in the compilation of the source file.
Make sure each of the files included (either directly or indirectly by you source file) falls into one of the two categories above.

PinCRT-enabled system headers

The header files below belongs to native CRTs, but may be included from PinCRT enabled Pin tool:

Linux, and OS X* (headers from glibc or Apple libc)

- stddef.h
- stdarg.h
- float.h

Windows (Visual Studio's libc)

- intrin.h
- xmmintrin.h
- windows.h (with some restrictions).

Make sure you’re linking your Pin tool with PinCRT libraries

When linking a Pin tool with PinCRT, all the libraries which you tool depends on must be linked with PinCRT as well.
Also, linking with a native CRT (e.g. glibc, or Visual Studio CRT) is not possible.
This means that you might need to rebuild some libraries that your tool is using so they’ll be linked with PinCRT.
If one of the libraries your tool is linked against (either directly, or indirectly) is linked with other C-Runtime than PinCRT this would lead to crashes when running it with Pin.
How to check the libraries your tool depends on

On Linux and Android

From a Linux terminal, run:
ldd <your tool shared object file>
These are the valid PinCRT libraries that you should see:

- libc-dynamic.so
- libdl-dynamic.so
- libm-dynamic.so
- libstlport-dynamic.so
- libpin3dwarf.so
- libxed.so
- linux-vdso.so.1 (Not part of PinCRT but valid as well).

Below are the libraries that should *not* be seen in the output.
Note that if you spot a shared object that depends on one of the libraries listed below, then this shared object must be rebuilt with PinCRT so it won’t depend on any of them libraries:

- libc.so.6
- libm.so.6
- libgcc_s.so.1
- libstdc++.so.6
- Any library under the directories: /lib, /lib32, and /lib64

Any other library in the output of “ldd” must be inspected again for dependent libraries using “ldd”.

On OS X*

From an OS X* terminal, run:
otool -l <your tool dylib file>
And look at the LC_LOAD* commands for depedent dylibs.
These are the valid PinCRT libraries that you should see:

- libc-dynamic.dylib
- libm-dynamic.dylib
- libstlport-dynamic.dylib
- libpin3dwarf.dylib
- libxed.dylib

Libraries located in any of the /usr/lib subdirectories should *not* be seen in the output.
If you spot a library that depends on one of the libraries under /usr/lib, then this library must be rebuilt with PinCRT so it won't depend on any of them libraries:

Any other library in the output of "otool" must be inspected again for dependent libraries using "otool".

On Windows

From Visual Studio command line prompt, run:
dumpbin /DEPENDENTS <your tool DLL file>
And look at all of the dependencies of your DLL.
These are the valid PinCRT libraries that you should see:

- pinvm.dll
- ntdll.dll
- kernel32.dll – works for now but might not be working in future version of Pin. The reason is that when using it Pin tool might undesirably interfere with the application.
- Non-system DLLs that you explicitly specified in link time, and you already checked it with using this procedure with "dumpbin".

If you spot any other library in the dependents list then the library we investigated must be rebuilt with PinCRT so it won't depend on any of them libraries:

Re-build any third party library with PinCRT

If your Pin tool is using a third party library then this library most probably needs to be re-built again from source with PinCRT.
Actually, the only case when a third party library may not be rebuilt is when dealing with a library which doesn't require any libc services – which is quite rare.

Common issues that may happen during the migration

Problem:

You encounter one of the error messages below when trying to run Pin:

```
E: Unable to load XXX: dlopen failed: cannot locate symbol "stdout" referenced by
"XXX"...
E: Unable to load XXX: dlopen failed: cannot locate symbol "stderr" referenced by
"XXX"...
```

Solution:

You most probably built your tool (or at least one of the object files that were linked to your tool) with the native CRT headers instead of PinCRT headers.

1. Locate the object(s) file that was/were built with the wrong header files.
2. Make sure you added the required compilation flags to the compilation command line of the object files you found.
3. If you sure you used the right compilation command line then see the section: “How to make sure my source includes only PinCRT compatible headers”.

Problem:

You encounter one of the error messages below when trying to build your Pin tool:

Compiler error:

```
error: exception handling disabled, use -fexceptions to enable
```

Linker error:

```
undefined reference to `__cxa_allocate_exception'...
undefined reference to `__cxa_throw'
unresolved external symbol __CxxThrowException@8 referenced in function …
```

Solution:

C++ exceptions is currently not supported in PinCRT. In the meantime, you’ll have to rewrite your program not to use exceptions.
If you’re getting one of the linker errors, make sure you added all of the PinCRT compilation flags specified in this document – in particular `–fno-exceptions` on clang and gcc, and `/EHs- /EHa-` in Visual Studio.
This will make the compiler notify you about the exact locations in your source where you used C++ exceptions.

Problem:

You encounter one of the error messages below when trying to build your Pin tool:

Compiler warning:

```
warning C4541: 'dynamic_cast' used on polymorphic type 'XXX' with /GR-;
unpredictable behavior may result
```

Linker error:

```
error LNK2019: unresolved external symbol ___RTDynamicCast referenced in function
…
error LNK2001: unresolved external symbol "const type_info::`vftable'"
undefined reference to `__dynamic_cast'
undefined reference to `__cxa_bad_cast'
undefined reference to `vtable for __cxxabiv1::__vmi_class_type_info'
undefined reference to `vtable for __cxxabiv1::__class_type_info'
undefined reference to `__gxx_personality_v0'
```

Solution:

C++ Runtime type information (RTTI) and dynamic cast are currently not supported in PinCRT. In the meantime, you'll have to rewrite your program not to use RTTI or dynamic casts.
If you're getting one of the linker errors, make sure you added all of the PinCRT compilation flags specified in this document – in particular `–fno-rtti` on clang and gcc, and `/GR-` in Visual Studio.
This will make the compiler notify you about the exact locations in your source where you used C++ RTTI or dynamic cast.

Problem:

You encounter one of the error messages below when trying to build your Pin tool:

Linker error:

```
error LNK2019: unresolved external symbol @__security_check_cookie@4 referenced
in function …
error LNK2019: unresolved external symbol ___security_cookie referenced in function
…
undefined reference to `__stack_chk_fail'
```

Solution:

One or more of the object files you're trying to link was compiled with the stack protector feature (or security check in Visual Studio).
Locate the faulty object files (by locating object files with undefined reference to the symbols that the linker complains about), and, make sure you added all of the required compilation flags specified in this document – in particular `–fno-stack-protector` on clang and gcc, and `/GS-` in Visual Studio.

Problem:

You encounter one of the error messages below when trying to build your Pin tool:

Linker error:

undefined reference to `__dso_handle'
error LNK2001: unresolved external symbol _fltused
error LNK2001: unresolved external symbol atexit
error LNK2019: unresolved external symbol _CRT_INIT referenced in function Ptrace_DllMainCRTStartup

Solution:

You probably didn't link your Pin tool with one (or two) of the object files crtbeginS.o, crtbegin.o, crtendS.o, crtend.o, crtbeginS.obj, crtbegin.obj.
The exact files set that you need depends on the target OS of your Pin tool and whether you're trying to build shared library (Pin tool) or executable (stand alone tool).
Please refer to the specific linkage instructions for more details.

Problem:

After migrating a Pin tool from using MS-CRT to PinCRT all *wprintf() function started to act weird.
In particular, the "%s" format doesn't format a string correctly.

Solution:

There is a difference between MS-CRT's *wprintf() functions and almost every other CRT's (including PinCRT's) *wprintf():

- In MS-CRT: the default strings in *wprintf() are wide characters (UTF16) so specifying "%s" in *wprintf() function actually means "%ls".
- In PinCRT (and glibc): the default strings in *wprintf() are multi-bytes characters (UTF8).

The code you ported from MS-CRT that calls *wprintf() probably expects a wide characters argument wherever a "%s" was specified in the format string.
So the best solution would be to replace all the "%s" in the format string to *wprintf() to "%ls".

Problem:

On Visual Studio, you encounter one of the error messages below when trying to build your Pin tool:

Linker error:

error LNK2019: unresolved external symbol __CIlog
error LNK2019: unresolved external symbol __libm_sse2_log_precise
or any error in this form:
error LNK2019: unresolved external symbol __CI*
error LNK2019: unresolved external symbol __libm_*

Solution:
You forgot to add the /Oi- compiler flag when you compiled the object file with the missing reference.