|  |  |
| --- | --- |
|  | **2014** |
|  | The Company  Martien Huijsmans |

|  |
| --- |
| **C++ Logbook** |
| This document captures the outcome of exploring C++. |

Inhoud

[TodoList 4](#_Toc420872226)

[Architecture guidelines 4](#_Toc420872227)

[Deployment 4](#_Toc420872228)

[Implementation preparation 4](#_Toc420872229)

[Tooling 4](#_Toc420872230)

[Code::Blocks IDE 4](#_Toc420872231)

[Eclipse IDE 4](#_Toc420872232)

[Installation Eclipse C++ on debian 4](#_Toc420872233)

[Reference other projects 5](#_Toc420872234)

[MinGW for windows (with Chocolatey) 5](#_Toc420872235)

[GCC for windows 7 64 bit 5](#_Toc420872236)

[Google Test 5](#_Toc420872237)

[Electric Fench 5](#_Toc420872238)

[Valgrind: memcheck 6](#_Toc420872239)

[Codecoverage 6](#_Toc420872240)

[Use-after-free, buffer-overflow 7](#_Toc420872241)

[Profiling-cpu: Valgrind 7](#_Toc420872242)

[Profiling-cpu: OProfile 7](#_Toc420872243)

[Profiling-cpu: Zoom 8](#_Toc420872244)

[Cpu-Profiling: perf-timechart 9](#_Toc420872245)

[CPU-Profiling + HeapChecker + Heap-profiler: Google Perf tools 9](#_Toc420872246)

[CPU /GPU Peformance tuning 9](#_Toc420872247)

[Heap-Profiling: Massif 9](#_Toc420872248)

[Debugging: GDB / KDBG / DDD / Nevimer 9](#_Toc420872249)

[Debugging: strace 10](#_Toc420872250)

[WireShark + GUI 10](#_Toc420872251)

[Testing within Eclipse 10](#_Toc420872252)

[Compiler flags for debugging; 10](#_Toc420872253)

[Trouble shooting 10](#_Toc420872254)

[Loading core dump 10](#_Toc420872255)

[GCC 10](#_Toc420872256)

[C++ version 10](#_Toc420872257)

[Frameworks & Libraries 11](#_Toc420872258)

[Overview 11](#_Toc420872259)

[Installation of logging framework 11](#_Toc420872260)

[Google Test Framework 12](#_Toc420872261)

[Rest SDK 12](#_Toc420872262)

[C++ 12](#_Toc420872263)

[Resource\_Acquisition\_Is\_Initialization (RAII) 12](#_Toc420872264)

[Exception handling 13](#_Toc420872265)

[C++ exception stack-trace 13](#_Toc420872266)

[Format of main 14](#_Toc420872267)

[Memory Management Model 14](#_Toc420872268)

[Memory Management 14](#_Toc420872269)

[Time & Date 14](#_Toc420872270)

[Threading 15](#_Toc420872271)

[Threading support in C++11 15](#_Toc420872272)

[Async 15](#_Toc420872273)

[String 15](#_Toc420872274)

[Smart Pointers 15](#_Toc420872275)

[Marco & Namesspace 15](#_Toc420872276)

[TypeId (including online run) 15](#_Toc420872277)

[Forking own and other process 16](#_Toc420872278)

[DSO / ABI 16](#_Toc420872279)

[Cross platform development 16](#_Toc420872280)

[Build Options 16](#_Toc420872281)

[GNU Make 16](#_Toc420872282)

[Automake 16](#_Toc420872283)

[Cmake 16](#_Toc420872284)

[Maven 16](#_Toc420872285)

[Scons 16](#_Toc420872286)

[Code coverage (with scons) 16](#_Toc420872287)

[Networking 17](#_Toc420872288)

[Packaging 17](#_Toc420872289)

[Dynamic libraries & versioning (major/minor) 17](#_Toc420872290)

[Unix commands for debugging 17](#_Toc420872291)

[References 19](#_Toc420872292)

# TodoList

Todo

|  |  |
| --- | --- |
| Virtual destructors |  |
| Compliance of logging output with specific standards |  |
| CMake |  |
| In project test Threading, class MainApp, there is code duplication for getting current time as string. |  |

# Architecture guidelines

## Deployment

Install software programs directly compiled from source into "/usr/local" or "/opt" to avoid collision with system programs  
Ref: <https://www.debian.org/doc/manuals/debian-reference/ch12.en.html>

## Implementation preparation

* Target platforms, e.g. Linux (+flavours), windows, etc.
* Which C++ compliance level
* Which C++ frameworks?
* Which tooling, e.g. Eclipse, make
* Which environment, e.g. GIT, ..

# Tooling

## Code::Blocks IDE

Cross platform C/C++ IDE with many plugins.  
<http://www.codeblocks.org/>

## Eclipse IDE

## Installation Eclipse C++ on debian

Installing Eclipse 3.8 (Juno)  
Ref: <http://www.cyberciti.biz/faq/debina-linux-installing-eclipse-jdt-eclipse-cdt-packages/>   
Supported from debian, but outdated.

Installing Kepler (4.x)  
Ref: <http://installeclipsedebian.blogspot.nl/2013/06/how-to-install-eclipse-42-juno-java-jee.html>

Scons Plugin: <http://sconsolidator.com/>

* Install Eclipse plugin from:   
  http://www.sconsolidator.com/update
* Import existing scons project:   
  File -> Import -> New Scons project from existing source

### Reference other projects

When a include/library is added, refresh doesn’t work in eclipse.  
But Project -> Properties -> C/C++Builds -> Settings -> CrossG++ Compiler -> Includes  
(no reason to change anything) -> Ok.  
Eclipse will now refresh the /user/local/include.

In eclipse an error can also be causes by a missing library.

## MinGW for windows (with Chocolatey)

Cmd.exe shall be opened as administrator.  
First install Choco as described here: <https://chocolatey.org/>   
Install mingw64: <https://chocolatey.org/packages?q=mingw>   
Cmd.exe shall be opened as administrator.

Next open cmd.exe and check compilers.  
$ gcc  
$ g++

Install python 2.7  
 <https://www.python.org/downloads/>  
Update path to point to: c:\python27;C:\Python27\Scripts

Installation scons 2.3.3  
<http://www.scons.org/>   
check  
$ scons

## GCC for windows 7 64 bit

Ref: <http://mingw-w64.sourceforge.net/download.php#mingw-builds>   
Used this link for installation: <http://win-builds.org/download.html>

## Google Test

Google's framework for writing C++ tests on a variety of platforms  
<https://code.google.com/p/googletest/>

## Electric Fench

<http://en.wikipedia.org/wiki/Electric_Fence>

## Valgrind: memcheck

Memcheck can detect many memory-related errors that are common in C and C++ programs  
<http://valgrind.org/docs/manual/quick-start.html>   
Note: it put’s requirements on how source is compiled, e.g. compiled with –g and –oo. Read the URL.

Usage:  
// run for memory leak  
$ valgrind --leak-check=full --log-file=valgrind.out target/test/tests  
valgrind.out now contains the places with errors

Valgrind last line can look like: ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)  
Supressed errors are related to libraries.  
You can view / modify the suppressed errors in the file /usr/lib64/valgrind/default.supp.  
The suppressed errors are just errors that are system specific, known problems with your libraries. They can be false positives.  
Ref: <http://stackoverflow.com/questions/7579619/how-would-i-discover-the-suppressed-error-in-valgrind>  
  
There is also a Jenkins plugin:  
<https://wiki.jenkins-ci.org/display/JENKINS/Valgrind+Plugin>   
  
Note: Valgrind is both a tool to create tools and provides tools, such as Memcheck.

## Codecoverage

<http://meekrosoft.wordpress.com/2010/06/02/continuous-code-coverage-with-gcc-googletest-and-hudson/>   
Note: also put’s requirements on compilation flags and linking, e.g. -fprofile-arcs -ftest-coverage and link the final executable with -lgcov.  
Ref: <http://stackoverflow.com/questions/566472/where-are-the-gcov-symbols>

Todo: At least 2 flavors

* Lines touched
* Paths leading to lines touched

To view/edit reports:  
<http://gcovr.com/guide.html>   
gcovr is installed via the python packet manager: pip.  
Install via command below. Next follow link above -> installation (tried once. It failed).  
$ yum -y install python-pip  
Note: pip may need a proxy.

## Use-after-free, buffer-overflow

[AddressSanitizer](https://code.google.com/p/address-sanitizer/wiki/AddressSanitizer) (ASan) is a fast memory error detector.   
It finds use-after-free and {heap,stack,global}-buffer overflow bugs in C/C++ programs.

## Profiling-cpu: Valgrind

Valgrind works on VM

Install valgrind if needed  
$ sudo yum install valgrind  
Install KCacheGrind if needed  
$ sudo yum install kcachegrind  
Install graphviz, which is needed by kcachegrind  
$ sudo yum -y install graphviz  
Note: install valgrind was quick and easy.  
But kcachegrind resulted in about 170 packages being installed.  
  
Next run the target program under control of valgrind.  
$ valgrind --tool=callgrind ./TileConsumer   
On completion a file with the valgrind data is generated, e.g. callgrind.out.13465  
Next run [KCacheGrind](http://baptiste-wicht.com/posts/2011/09/profile-c-application-with-callgrind-kcachegrind.html) to display collected data.  
$ kcachegrind

## Profiling-cpu: OProfile

<http://oprofile.sourceforge.net/about/>   
oprofile requires that Linux runs has guest OS.

Installation on fedora:  
$ sudo yum install oprofile

To run  
run: operf ./target/test-app  
ref: <http://oprofile.sourceforge.net/doc/overview.html#getting-started-with-operf>  
make sure that the app is compiled with debug info:  
make sure code is compiled with debug info. Option –g  
Ref: <https://gcc.gnu.org/onlinedocs/gcc/Debugging-Options.html>

Quick impression:  
$ opreport  
Prints:  
Using /home/martien/profiling/oprofile\_data/samples/ for samples directory.  
CPU: Intel Westmere microarchitecture, speed 3.325e+06 MHz (estimated)  
Counted CPU\_CLK\_UNHALTED events (Clock cycles when not halted) with a unit mask of 0x00 (No unit mask) count 100000  
CPU\_CLK\_UNHALT...|  
   samples|      %|------------------  
   239725 100.000 app-test          
CPU\_CLK\_UNHALT...|  
          samples|      %|        ------------------  
            91713 38.2576 libabc-print.so  
            65444 27.2996 libabc-label.so  
            37292 15.5562 libabc-commons.so  
             8990  3.7501 libabc-engine.so  
             6984  2.9133 libiomp5.so  
             6650  2.7740 no-vmlinux  
             5629  2.3481 libstdc++.so.6.0.19  
             4564  1.9038 libmwmorphop\_packed.so  
             3806  1.5877 libabc-lollipol.so  
             2988  1.2464 [libc-2.18.so](http://libc-2.18.so)  
             1374  0.5732 libimfg.so  
             1307  0.5452 libmwbw.so  
             1278  0.5331 libmwbt.so  
             1015  0.4234 libippm.so  
              505  0.2107 libtkh.so.2  
              117  0.0488 [ld-2.18.so](http://ld-2.18.so)  
               55  0.0229 libtfsmalloc.so.2  
                9  0.0038 [libpthread-2.18.so](http://libpthread-2.18.so)  
                5  0.0021 test-app

Textual callgraph:  
$ opreport –callgraph: gove soverview  
ref: <http://oprofile.sourceforge.net/doc/opreport.html#opreport-callgraph>

Covert oprofile to format understood by kcachegrind  
$ opreport -gdf | op2calltree

## Profiling-cpu: Zoom

<http://www.rotateright.com/>   
Includes info on L1/l2/l3 cache misses.

Profiling: perf

Found via, see bottom:  
<http://stackoverflow.com/questions/2486840/linux-c-how-to-profile-time-wasted-due-to-cache-misses>   
I tried commands on VirtualBox-Linux.  
$ perf record -e LLC-loads,LLC-load-misses yourExecutable  
next  
$ perf report  
I could use “-e LLC-loads,LLC-load-misses”, but command executes and provides a report. Report includes a data on virtualbox. Interesting.

Perf wiki: <https://perf.wiki.kernel.org/index.php/Tutorial>   
Show many more option than what I showed above.

MANY examples on howto use perf (nice): <http://www.brendangregg.com/perf.html>

## Cpu-Profiling: perf-timechart

Initial blog: <http://web.archive.org/web/20090922171904/http://blog.fenrus.org/?p=5> which includes diagrams.  
  
Manpage: <http://linux.die.net/man/1/perf-timechart>

## CPU-Profiling + HeapChecker + Heap-profiler: Google Perf tools

<https://code.google.com/p/gperftools/?redir=1>

## CPU /GPU Peformance tuning

<https://software.intel.com/en-us/intel-vtune-amplifier-xe> (commercial; >$800)

## Heap-Profiling: Massif

<http://valgrind.org/docs/manual/ms-manual.html>

## Debugging: GDB / KDBG / DDD / Nevimer

Make sure that program is compiled with –g, so that debug information is included.

Start   
$ gdb  
next: help

Run to run program  
$ gdb <name executable>  
 run (to start the executable)  
  thread apply all bt  
 to print stack trace; useful after program crashes

Start GDB with executable and core dump  
Make sure that debuginfo RPM is installed.  
gdb path/to/the/binary path/to/the/core

Running GDB from Python:  
<http://stackoverflow.com/questions/22370928/is-it-possible-to-automaticly-restart-program-in-gdb-after-exception-segfault?rq=1>

KDBG is a GUI to gdb  
home: <http://www.kdbg.org/>   
It requires KDE.

Alternatives

* Ddd
* Nevimer: <http://xmodulo.com/debug-program-nemiver-debugger.html>

## Debugging: strace

Function: print system calls  
Usage:  
$ strace <name of executable>

## WireShark + GUI

$ sudo yum install wireshark wireshark-gnome  
ref: <https://ask.wireshark.org/questions/13243/wireshark-command-not-found>   
After this wireshak is available.   
Localhost works on Linux, provided that you run as sudo.

# Testing within Eclipse

<https://github.com/xgsa/cdt-tests-runner/wiki/Tutorial>

## Compiler flags for debugging;

Pass -g to your compiler and linker to have this information in the executable.   
You may also want to add -O0.

## Trouble shooting

--verbose flag  
<http://stackoverflow.com/questions/16710047/usr-bin-ld-cannot-find-lnameofthelibrary>   
Can be set for G++ and LD (Xlinker) in Eclipse

## Loading core dump

Ref: <http://stackoverflow.com/questions/16413123/eclipse-cdt-gdb-open-core-dump>

## GCC

### C++ version

Ref: <http://stackoverflow.com/questions/266168/simple-example-of-threading-in-c>

* *On GCC compile with -std=c++11.*
  + <http://en.wikipedia.org/wiki/C++11>
* *On GCC versions below 4.7, use -std=c++0x*

*Support in GCC*: <http://gcc.gnu.org/projects/cxx0x.html>

# Frameworks & Libraries

## Overview

|  |  |
| --- | --- |
| Approved | |
| Boost | THE C++ Library project: <http://www.boost.org/> |
| Logging | <https://github.com/log4cplus/log4cplus>  Install headers (into /user/local/include) and libs (into /user/local/lib) |
| Test Framework | <https://code.google.com/p/googletest/>  Testing doc: <http://feelings-erased.blogspot.de/2012/07/eclipse-juno-has-landed-with-unit.html> |
|  | Modelled after JUnit  <http://sourceforge.net/projects/cppunit/files/cppunit/>  Installl is described here. <http://linuxtortures.blogspot.nl/2012/04/unit-testing-with-cppunit-and-eclipse.html>  It has a .configure / make / sudo make install. |
| STL | <http://www.cplusplus.com/reference/stl/> |

|  |  |
| --- | --- |
| Tried. Failed | |
| Template library | <http://www.stlsoft.org/> Didn’t like. |
| FastFormat | String formatting library analysis: <http://stackoverflow.com/questions/446276/is-there-a-catch-with-fastformat>  e.g. no wchar\_t support. But no installation support of headers / libraries. |

## Installation of logging framework

Steps

* Downloaded source code from source forge as described on:  
  <http://sourceforge.net/p/log4cplus/wiki/Home/>
* Then I found   
  <http://gisfromscratch.wordpress.com/2012/01/08/log4cplus-on-ubuntu/>   
  which described the next steps
  + Unzip into the /home/martien/packages directory.
  + Opened INSTALL, which described the 4 steps
  + sudo .configure (although I didn’t use sudo)
  + make (not clear for me where the output was).
  + sudo make install (because access needed to certain libraries).
  + Sudo ldconfig (to refresh the library cache)
* The link also described where the header files and binaries are located.
  + The /user/local/include is already present in the Eclipse C++ project
  + In /usr/local/lib lib’s are stored.  
    BUT python has different directories for different releases.
* Examples on how to use  
  <http://sourceforge.net/p/log4cplus/wiki/CodeExamples/>
* Update the eclipse configuration
  + With library path
  + With libraries, but for library liblog4cplus.a, the name is: log4cplus.  
    So leave out lib and suffix.

Formatting the log output:  
<http://stackoverflow.com/questions/15388923/log4cplus-not-printing-file-names-and-line-numbers>   
But there is also support in the macro’s.

## Google Test Framework

After installation according to provide build instruction do the following.

* Copy gtest/include to /usr/local/include  
  This will create a gtest directory.
* Copy libraries to /usr/local/lib
* Add above directories to the test project
* Add libraries
  + gtest
  + gtest\_main
  + pthread

When testing with a shared library, run\_configuration needs to contains the path to the shared libraries.  
Here it is described how to configure.  
<http://linuxtortures.blogspot.nl/2012/04/unit-testing-with-cppunit-and-eclipse.html>

## Rest SDK

<http://casablanca.codeplex.com/>

<https://code.google.com/p/ffead-cpp/>  
Test framework for REST / SOAP: <https://github.com/sumeetchhetri/gatf>

# C++

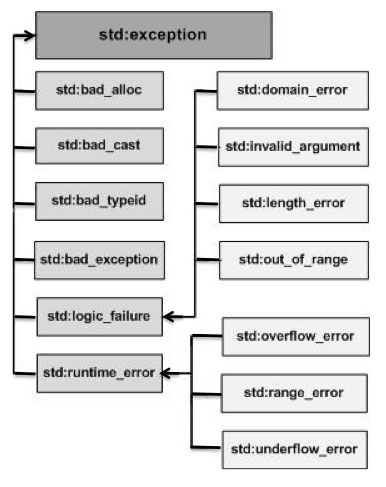
## Resource\_Acquisition\_Is\_Initialization (RAII)

<http://en.m.wikipedia.org/wiki/Resource_Acquisition_Is_Initialization>

## Exception handling

Guidelines on how to apply (and not) exception handling:  
<http://en.wikibooks.org/wiki/C%2B%2B_Programming/Exception_Handling>

<http://www.tutorialspoint.com/cplusplus/cpp_exceptions_handling.htm>   
C++ Standard Exceptions:

C++ provides a list of standard exceptions defined in <exception> which we can use in our programs. These are arranged in a parent-child class hierarchy shown below:  


Exception Propagation  
<http://www.boost.org/doc/libs/1_55_0/libs/exception/doc/boost-exception.html>   
<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2179.html>

JNI Exception handling:  
Augment the Java stacktrace from JNI. Looks interesting.  
<http://monochrome.sutic.nu/2013/09/01/nice-jni-exceptions.html>

Found this quote (in Poco doc).  
In good old C++ tradition, you should always throw by value and catch by (const) reference.  
  
Interesting POCO document with exception ,assertion and NDC (c++ stacktrace).  
<http://pocoproject.org/slides/020-ErrorHandlingAndDebugging.pdf>

## C++ exception stack-trace

This feels like a interesting library for debug build.  
<http://libcsdbg.sourceforge.net/>

## Format of main

Ref: <https://publib.boulder.ibm.com/infocenter/comphelp/v8v101/index.jsp?topic=%2Fcom.ibm.xlcpp8a.doc%2Flanguage%2Fref%2Fmainf.htm>

int main (void)

int main ( )

int main(int argc, char \*argv[])

int main (int argc, char \*\* argv)

## Memory Management Model

This is tightly coupled to threading.

#include <mutex>  
// Mutex can be used for shared resource access.  
// On release of the lock ALL data modified by the thread will be flushed to main memory.  
std::mutex storeMutex;

Cache Coherence: <http://en.wikipedia.org/wiki/Cache_coherence>   
  
Avoiding and Identifying False Sharing Among Threads / Cache Lines  
<https://software.intel.com/en-us/articles/avoiding-and-identifying-false-sharing-among-threads/>

## Memory Management

Following combinations:

* new, delete used for classes. Constructor/destructor of class get’s called.
* calloc (init to 0), malloc, free for raw memory

Ref: <http://stackoverflow.com/questions/4061514/c-object-created-with-new-destroyed-with-free-how-bad-is-this>

## Time & Date

Std has time support (from specific GCC version).  
<http://stackoverflow.com/questions/997946/how-to-get-current-time-and-date-in-c>

std:chrono  
<http://solarianprogrammer.com/2012/10/17/cpp-11-async-tutorial/>

## Threading

### Threading support in C++11

Prior to C++11, C++ had no thread concept and no sleep capability, so your solution was necessarily platform dependent. Here's a snippet that defines a sleep function for Windows or Unix:  
Ref: <http://stackoverflow.com/questions/1658386/sleep-function-in-c>

But there is a precompile option: \_GLIBCXX\_USE\_NANOSLEEP, which as discussed in following links  
<http://stackoverflow.com/questions/12523122/what-is-glibcxx-use-nanosleep-all-about>   
during installation (.configure) controls what functionality can be provided on the platform.  
Strange, because the development and runtime platform may be different.  
Note: I have gcc version 4.7.2 ($ gcc –v).

There was a complaint from Eclipse (std::thread and <varName>.join were not found).  
I followed instruction “88” at  
<http://stackoverflow.com/qustions/9131763/eclipse-cdt-c11-c0x-support>   
and the Eclipse problem was solved (for a few seconds ☹).  
Note: I could run the application with this error.

Presentation   
Hurb Stutter  
<https://www.youtube.com/watch?v=c1gO9aB9nbs>  
18:05

compare\_exchange\_weak is for loops  
compare\_exchange\_strong is for non-loops  
ABA problem  
linearizebility: <http://www.cs.tau.ac.il/~afek/Mila.Linearizability.ppt>

### Async

There is also the concept of async execution, where the actual mapping to threads is left to the OS.  
Described in this tutorial

<http://solarianprogrammer.com/2012/10/17/cpp-11-async-tutorial/>

## String

Sources  
<http://www.cplusplus.com/reference/string/string/>

Nice String example:  
<http://www.mochima.com/tutorials/strings.html>

Combining string and int  
<http://stackoverflow.com/questions/191757/c-concatenate-string-and-int> .

## Smart Pointers

Underlying theory

<http://en.wikipedia.org/wiki/Resource_Acquisition_Is_Initialization>

<http://en.cppreference.com/w/cpp/memory> : unique\_ptr, shared\_ptr, weak\_ptr  
Each type is well explained.

## Marco & Namesspace

In C++, macros don't obey namespaces,

## TypeId (including online run)

<http://en.cppreference.com/w/cpp/language/typeid>

## Forking own and other process

<http://www.thegeekstuff.com/2012/05/c-fork-function/>

## DSO / ABI

<http://www.akkadia.org/drepper/dsohowto.pdf>  
<http://mentorembedded.github.io/cxx-abi/abi.html>

# Cross platform development

To run scons on windows  
- install chocolatey: <https://chocolatey.org/>  
 as described on main page  
in cmd.exe:   
$ cinst mingw  
This will install mingw 64 bit.

# Build Options

## GNU Make

<http://www.gnu.org/software/automake/manual/automake.html#GNU-Build-System>  
Used. Ok for simple cases. Native.

## Automake

Intro: <http://en.wikipedia.org/wiki/Automake>   
Not tried.

## Cmake

Intro: <http://en.wikipedia.org/wiki/Cmake>   
Not tried.

## Maven

<https://github.com/maven-nar/nar-maven-plugin>

## Scons

<http://www.scons.org/>   
Install scons on fedora is easy:  
$ sudo yum install scons

## Code coverage (with scons)

Ref: <http://o440.wordpress.com/2011/02/25/scons-and-code-coverage/>

# Networking

Example of HTTP with Boost  
<http://stackoverflow.com/questions/7046370/https-request-with-boost-asio-and-openssl>

# Packaging

Debian packing tutorial  
<http://www.debian.org/doc/manuals/packaging-tutorial/packaging-tutorial.en.pdf>

RPM: see rpm-yum doc.

# Dynamic libraries & versioning (major/minor)

Discussed in this link:  
<http://stackoverflow.com/questions/3839756/how-do-applications-resolve-to-different-versions-of-shared-libraries-at-run-tim>

# Unix commands for debugging

List imported and exported functions.  
$ nm -D libtools-1.4.0-SNAPSHOT.so

list libraries where things are resolved.  
$ ldd libtools-1.4.0-SNAPSHOT.so  
$ ldd <name executable>

Examining object files (libraries / executables):

* nm: Lists symbols from object files.
* objdump: Displays detailed information from object files.  
  e.g. objdump -x binary-or-library |grep RPATH
* readelf: Displays information about ELF object files, e.g. rpath  
  $ readelf -d binary-or-library | head -20

ref: <http://www.ibm.com/developerworks/aix/library/au-unixtools.html>

find will tell you if the file exists.  
$ find /usr -iname "\*libXp.so.6\*"

will tell you if it has been installed using rpm.  
$ rpm -ql /usr/liblibXp.so.6

tell linux  
$ uname –a  
Linux xyz-buildserver 3.10.0-123.1.2.el7.x86\_64 #1 SMP Wed Jun 4 15:22:01 EDT 2014 x86\_64 x86\_64 x86\_64 GNU/Linux

Ask yum to tell what rpm provides a library  
$ yum whatprovides '\*/libXp.so.6'

to list all the dependancies of a package and see what needs to be installed without downloading or installing anything.  
$ yum deplist <package>

Display content rpm  
$ rpm -qlp <name>.rpm  
  
Install RPM, no dependency check  
$ rpm -i --nodeps <name>.rpm

RPM list scriplets  
$ rpm -qp --scripts filename.rpm

RPM list scriplets for installed package  
$ rpm -q --scripts packageName  
  
Start file manager from shell  
$ xdg-open .

Find the .so that contains function dilate\_flat\_uint16\_tbb.  
for s in $(find . -name '\*.so'); do echo $s; nm -D $s | grep dilate\_flat\_uint16\_tbb ; done | less;  
for s in $(find /opt/visionengineapp/lib -name '\*.so'); do echo $s; nm -D $s | grep queue ; done | less;

To check 32/64 bit version linux  
$ uname –m

Check if lib is 32/64 bit  
$ file native/libabc.so   
native/libsbc.so: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, stripped  
  
cpu info (different flavors)  
<http://www.binarytides.com/linux-cpu-information/>

List open files on the system  
$ lsof

Read stdin to stdout & file  
$ <first command> | tee file.txt | <next command>

Setup UDP / TCP connections  
<http://linux.die.net/man/1/nc>

Find and delete files/folders  
$ find . -name .project | xargs rm -r

Disk info (blkid provides information on file system; fdisk not not).  
Useful when mounting a disk.  
$ sudo fdisk –l  
$ sudo blkid

Fedora, start terminal from commandline.  
$ gnome-terminal&

# References

Programming on debian:  
<https://www.debian.org/doc/manuals/debian-reference/ch12.en.html>

C++ tutorials  
<http://www.mochima.com/tutorials/> (strings, vectors)  
<http://www.cplusplus.com/doc/tutorial/>