Technologie-Sprint ABI compliance

Titel: *ABI Checking for essential Libraries*

Start: *10/2021*

Ende: *10/2021*

Verantwortlicher Projektpartner: ES

Beteiligte Standorte: ES, UL

Beteiligte Nutzer:innen/Arbeitsgruppen:

Status: *finished*

Abstract: Software packages including their libraries on HPC Systems are in part provided by the OS' distribution (such as the Linux kernel or `libc`), in part by the HPC suppliers (such as network-stack libraries like `libmxm` and `libslurm`) or are compiled by the HPC centers themselves (such as Open MPI). For stability of the system as a whole, it is of utmost importance, that SW libraries' Application Binary Interface (ABI) do not change unnoticed. This TS evaluates the basics of ABI changes, semantic versioning as a means to signal breaking changes, and the tool ABI compliance checker.

Vorarbeiten:

Prior Art: Various libraries adhere to semantic versioning, requiring trust in its correctness

Relevanz: Security advisories may require urgent replacement of important libraries (like `libslurm`), leading to follow-up issues e.g. due to ABI changes.

Ergebnis: *Successful*

Empfehlungen: *ABI compliance checker should be used*

Zusätzliche Dokumentationen: *Wiki entry on ABI compliance checker*

Aufwand in Personentagen: 10

Long version: See below

Introduction

Compiled software packages provide binaries in the form of executables and libraries, which other software may rely on. The foundation programmers use is the Application Programmer Interface (API), which the compiler and linker bind to – and which are encoded in the functional relation caller, caller's version, callee and callee version, compiler-settings. The software packages on HPC systems is provided by various entities: 1. the OS' distribution, e.g. RedHat Enterprise Linux (RHEL), providing the bulk of the software such as the Linux kernel or

the GNU `libc` library. 2. the HPC vendor and suppliers, e.g. Mellanox/NVIDIA providing network stack libraries like `libmxm` or SchedMD providing the scheduler library `libslurm`. 3. the HPC centers providing specific implementations and newer versions of Software adapted to the requirements of their HPC users. Examples are Open MPI adapted to provide communication with CUDA-enabled applications computing on NVIDIA GPUs.

This TS will cover the underlying problem with ABIs changing over time, the issue exemplified based on Open MPI and SLURM and will propose a solution based on the package *ABI compliance checker*.

ABI issues

Together with the underlying processor architecture (e.g. AMD64, now known as x86-64) and operating system's conventions plus the compiler's definitions and settings, the software defines an Application Binary Interface (ABI). While the ABI of the underlying hardware and libraries provided by the OS' distribution must not change incompatibly, the ABI of other compiled software may change over time, i.e. from one software's version to another installed one. Consider a library `libx` which provides two visible functions in version 1.0.0 of the library:

```
// from the project's configuration header
#ifdef HAVE_ATTRIBUTE_VISIBILITY
#  define x_visible __attribute__(__visibility__("default"))
#else
#  define x_visible  // empty, not visibility attribute,
#endif
// From the project's header file x.h
enum enum1 {
  ENUM_VAL1 = 0,
  ENUM_VAL2,
}
int func1(int param1, enum1 e) x_visible;

struct visible {
  char   c_val;  // on x86-64, compiler must include 7 Bytes padding
  double d_val;
}
struct opaque;
int func2(struct visible * s1, struct opaque * s2) x_visibile;
```

Examples of incompatible ABI changes are: 1. Remove a function previously visible and available, e.g. eliminating above `func2` or removing a visibility (see above `__attribute__(__visibility__("default"))` for GNU compiler) on this functions declaration – which may happen inadvertantly upon installation if the configure's/cmake's magic fails to detect `HAVE_ATTRIBUTE_VISIBIILITY` (albeit this should then have all symbols exported), 2. Changing an existing

user-exposed enum by deleting entries or changing values, e.g. setting `ENUM_VAL1` to start at 1, 3. Changing return types or the set return values of functions, 4. Changing the types or the order of fields of a user-visible struct, e.g. reducing the width of `double d_val` to a `float` type or adding another entry. 5. Compiling using non-standard parameter passing (e.g. GNU's `-mregparam`, `-msseregparam` or `-freg-struct-return`) or for another ABI (GNU/Linux -x32)

It is good practice to denote changes using semantic versioning (see below) – and to first mark functions and globally visible variables to be deleted in future using e.g. `__attribute__((__deprecated__("to be deleted in v2.0")))`.

Adding new functions, new types or adding new entries to the end of existing enums, adding new fields to existing opaque structs will **not** change the ABI. Furthermore any changes in invisible, un-exported parts of the compiled library will also not change the ABI to the caller if default settings of the software package are not affected[1].

Examples of incompatible ABI changes with MPI

The MPI standard and it's main implementations based on Open MPI and MPIch are good examples to show the intricacies. Both implementations use Semantic Versioning as described below. The MPI standard mostly defines opaque types, however there's one user-visible datatype called `MPI_Status` as structure, which at least must provide the fields `status.MPI_STATUS`, `status.MPI_TAG` and `status.MPI_ERROR`. Both MPI implementations however define further entries, which the user should not rely on, as they may change over time. With regard to opaque types like `MPI_Communicator` MPIch defines them as an integer type, while Open MPI defines opaque pointers to opaque structs. The MPI API and their implementation's ABI are rather stable towards the calling MPI-application. Problems however may arise in the interaction of the MPI implementation's runtime (in case of Open MPI the ORTE/prrte component and PMIx) with the scheduler on this Cluster: if `libslurm.so` changes incompatibly, starting MPI-jobs may fail. To a lesser extent other components (in case of Open MPI the so-called modular component architecture, mca) may be affected, such as libraries used by MPI to call into vendor's binary libraries such as Melanox' `libmxm.so` or `libhcoll.so` for hierarchical collectives being part of HPC-X.

Semantic Versioning

Semantic versioning is a concept to define version numbers in a 3-level scheme (MAJOR.MINOR.PATCH declaring a public API for this version), several rules on how changes are reflected and the relevant numbers are chosen. A project must declare, which level will denote incompatible ABI change of the public API, i.e. whether MAJOR or MINOR level will denote an incompatible change. Then PATCH denotes versions with only bug-fixes and backward compatible changes.

[1] The KDE project provides best-practices for C++ programmers: KDE Binary Compatibility Examples

The library `libx` of version 1.0.2 may savely replace the version 1.0.0, while version 2.0.0 would include binary incompatible changes, any user (be it application or library) would need to be re-compiled (and possibly amended in case of missing or deprecated functions).

On Unix systems this version is then encoded in the libraries file name and must represent the above-mentioned semantic version. Applications linked against a specific version of any library would then not be able to run against another incompatible version. The following output of `ldd` shows six of the 43 dependencies of an exemplary MPI application compiled and linked against Open MPI-4.1.1:

```
linux-vdso.so.1 (0x00007ffc64bc1000)
libmpi.so.40 => /opt/bwhpc/common/mpi/openmpi/ *newline added*
              4.1.1-gnu-11.1/lib/libmpi.so.40 (0x00001515c38e3000)
libpthread.so.0 => /usr/lib64/libpthread.so.0 (0x00001515c36c3000)
libc.so.6 => /usr/lib64/libc.so.6 (0x00001515c3301000)
liblustreapi.so.1 => /lib64/liblustreapi.so.1 (0x00001515c30cf000)
libslurm_pmi.so => /usr/lib64/slurm/libslurm_pmi.so
```

As may be seen, the libraries are from software packages provided by all above mentioned entities, many but *not all* of them use semantic versions encoded into the library-name. For example `libc.so.6` points to the implementation `libc-2.24.so`. Another entry `libslurm_pmi.so` is unversioned as such and rather uses `/usr/lib64/libslurm.so` which currently points to implementation `libslurm.so.36.0.0`.

Upon updating Slurm it is important to ensure, that libraries will not have incompatibly changed it's ABI.

ABI compliance checker

As the name suggests ABI compliance checker tests for incompatible ABI changes based on dumps generated by a sister project ABI dumper, resulting in a concise HTML-based report.

The tool may work on the actual underlying source or on the compiled binary, based on the debugging information. Therefore compilation with `CFLAGS="-Og -g"` is recommended.

Requirements are `abi-dumper.pl` and `abi-compliance-checker.pl`[2].

Generating a dump of a library such as Open MPI, run: `V=4.1.1 abi-dumper.pl -vnum ${V} -o openmpi-${V}.dump PATH_TO_OMPI-${V}/lib/libmpi.so`

Comparing an old version, such as `OLD_VERSION=4.0.6`, run: `abi-compliance-checker.pl -l openmpi -old openmpi-${OLD_VERSION}.dump -new openmpi-${V}.dump`

[2] To install run: `git clone https://github.com/lvc/abi-dumper git clone https://github.com/lvc/abi-compliance-checker.git`

Example of report

An example report between Open MPI-3.1.6 and Open MPI-4.0.6 is shown in the following figure:

API compatibility report

Binary Compatibility | Source Compatibility

Test Info

Module Name	openmpi
Version #1	3.1.6
Version #2	4.0.6
Arch	x86_64
GCC Version	11.2.0
Subject	Binary Compatibility

Test Results

Total Header Files	107
Total Source Files	581
Total Objects	1
Total Symbols / Types	875 / 956
Compatibility	74.8%

Problem Summary

	Severity	Count
Added Symbols	-	41
Removed Symbols	High	2
Problems with Data Types	High	0
	Medium	63
	Low	131
Problems with Symbols	High	15
	Medium	20
	Low	5
Problems with Constants	Low	0
Other Changes in Data Types	-	5

This shows several changes of sizes of internal data structures and 2 removed symbols.

Checking for compliance of Open MPI-4.1.2rc2, aka the second release candidate compared to Open MPI-4.1.1 is provided below – and shows only one high problem with a symbol. This turns out to be an internally used function `ompi_coll_base_bcast_intra_basic_linear`.

A set of three SLURM versions was compiled: `for V in 20.02.7 20.11.8 21.08.2 ; do cd slurm-${V} ; mkdir -p COMPILE; cd COMPILE ; rm -fr * ; ../configure --prefix=$PWD/usr CFLAGS="-g -Og" | tee configure.out && make -j72 | tee make.out && make install ; cd ../../ ; done`

and then analyzed: `for V in 20.02.7 20.11.8 21.08.2 ; do PATH_TO/abi-dumper.pl -vnum ${V} -o slurm-${V}.dump slurm-${V}/COMPILE/usr/lib/libslurm.so ; done` showing several symbol additions and changes. An example of a user-visible change between slurm-20.02.7 to slurm-20.11.8 might be

slurm_mpi.h

[–] struct **mpi_plugin_client_info_t** 1

	Change	Effect
1	Field **jobid** has been removed from the middle position of this structural type.	1) Previous accesses of applications to the removed field will be incorrect. 2) Layout of structure fields has been changed and therefore fields at higher positions of the structure definition may be incorrectly accessed by applications.

Caveats

The generated difference reports are exhaustive, as may be seen above. Whether actual damage arises from e.g. high-severity change in Slurm is not really tractable to the common installer as a daily task. In the case of the above high-severity symbol change, it's a matter of usage in the caller of this function. Nevertheless, the output is very helpful to the developer using an application, e.g. to determine changes in the semantic versioning of one's own sofware.

Best practices

A proposed best practice is to generate a dump with every installed library exported to users (e.g. `libmpi.so`). Upon installing a newer version of the same library, generate a ABI compliance report and check for serious errors.

For starters, the page ABI Laboratory provides generated reports, e.g. the ABI report on Open MPI. However versions are not kept up-to-date since about a year.

Alternative ABI checkers

icheck abicheck libabigail and the ABIgail tutorial

API compatibility report

Binary Compatibility | Source Compatibility

Test Info

Module Name	openmpi
Version #1	4.1.1
Version #2	4.1.2rc2
Arch	x86_64
GCC Version	11.2.0
Subject	Binary Compatibility

Test Results

Total Header Files	111
Total Source Files	613
Total Objects	1
Total Symbols / Types	927 / 1036
Compatibility	83.3%

Problem Summary

	Severity	Count
Added Symbols	-	0
Removed Symbols	High	0
Problems with Data Types	High	0
	Medium	4
	Low	10
Problems with Symbols	High	1
	Medium	7
	Low	5
Problems with Constants	Low	0

Figure 1: ABI compliance checker for Open MPI, 2