GDB Pretty-Printers, ELF and GDB again

ELF symbols versioning, gdbserver usage

Red Hat

Author Jan Kratochvíl <jan.kratochvil@redhat.com>

February 18, 2012

Agenda

1. GDB Pretty-Printers
2. ELF symbols versioning
3. gdbserver usage

Section 1

GDB Pretty-Printers

redhat.

Demo of GDB Pretty-Printers - std::string

```
(gdb) print/r stdstringvar
$1 = {static npos = <optimized out>, _M_dataplus =
      {<std::allocator<char>> =
       {<__gnu_cxx::new_allocator<char>> =
        {<No data fields>}, <No data fields>},
      _M_p = 0x601028 "abcd"}}

(gdb) print stdstringvar
$1 = "abcd"
```

redhat.

Demo of GDB Pretty-Printers - std::list

```
std::list<int> stdlistvar = { 10, 20, 30, 40 };

(gdb) print/r stdlistvar
$1 = {<std::_List_base<int, std::allocator<int> >> =
      {_M_impl = {<std::allocator<std::_List_node<int> >> =
       {<__gnu_cxx::new_allocator<std::_List_node<int> >> =
        {<No data fields>}, <No data fields>},
       _M_node = {_M_next = 0x602010, _M_prev = 0x602070}}},
      <No data fields>}
(gdb) print stdlistvar.front()
$2 = (int &) @0x602020: 10

(gdb) print stdlistvar
$3 = std::list = {[0] = 10, [1] = 20, [2] = 30, [3] = 40}
```

redhat.

Existing GDB Pretty-Printers in Fedora

GDB Pretty-Printers:

- **libstdc++**: std::string, other std::*
- **glib2-devel**: GList etc., Alexander Larsson blog
- **mono-core**: C#
- **libreoffice-gdb-debug-support**:
  rtl::Reference auto-dereferencing etc.
- **gdb-debuginfo**: type, main_type etc.

New commands:

- **gdb-heap**: new commands like "heap"

Use **base**, **base-devel** or **base-debuginfo** rpm?

redhat.

Demo of GDB Pretty-Printers - libreoffice

```
(gdb) print/r pSVData.mxDisplayConnection
$1 = {m_pBody = 0x7f3c05e3e6c0}

(gdb) print pSVData.mxDisplayConnection
$2 = RtlReferencePrinter:to_string
rtl::Reference to {[...],
m_aErrorHandlers = empty std::list, m_aAny = uno::Any ":0"}

(gdb) print *pSVData.mxDisplayConnection.m_pBody
$3 = {[...],
m_aErrorHandlers = empty std::list, m_aAny = uno::Any ":0"}
```

redhat.

Writing new GDB Pretty-Printers in Python

- auto-loaded scripts: **FILE-gdb.py**
- for multilib:
  - printers code location: **/usr/share/libXYZ/gdb/*.py**
  - via: **/usr/share/gdb/auto-load/lib/libglib-*.so-gdb.py**
  - via: **/usr/share/gdb/auto-load/lib64/libglib-*.so-gdb.py**

redhat.

Simple GDB Pretty-Printer for std::string

```
import re
class StdStringPrinter:
    def __init__(self, val):
        self.val = val
    def to_string(self):
        return self.val['_M_dataplus']['_M_p']
def str_lookup_function(val):
    tag = val.type.unqualified().strip_typedefs().tag
    regex = re.compile("^std::basic_string<char,.*>$")
    if regex.match(tag):
        return StdStringPrinter(val)
gdb.pretty_printers.append(str_lookup_function)
```

Section 2

ELF symbols versioning

redhat.

ELF symbols versioning

- from Solaris
- library: exported symbols by name
- application: imported symbols by name

$\Longrightarrow$ How to match them?

sample: **memcpy** vs. **memmove** compatibility

redhat.

Basic library and application

library:

```
void mymemcpy (char *d, char *s, long l) {
  while (l--)
    *d++ = *s++; }
```

application:

```
void mymemcpy (char *d, char *s, long l);
int main (void) {
  char ab[] = "abc";
  mymemcpy (&ab[0], &ab[1], 3);
  puts (ab); return 0; }
```

expected output: **bc**

redhat.

Application broken by new library

original library:

```
void mymemcpy (char *d, char *s, long l) {
  while (l--)
    *d++ = *s++; }
```

new library:

```
void mymemcpy (char *d, char *s, long l) {
  while (l--)
    d[l] = s[l]; }
```

expected output: **bc**
actual output: –nothing–

redhat.

New library using #define

```
extern void mymemcpy_new (char *d, char *s, long l);
#define mymemcpy(d, s, l) mymemcpy_new (d, s, l)
```

redhat.

Versioned library -Wl,–default-symver

```
void f_old (char *d, char *s, long l) {
  memmove (d, s, l); }
void f_new (char *d, char *s, long l) {
  while (l--)
    d[l] = s[l]; }
__asm__(".symver f_old,mymemcpy@");
__asm__(".symver f_new,mymemcpy@@libcpy.soV2");
```

file libmymemcpy.ver:

```
/* use: -Wl,-soname,libcpy.so
        -Wl,--version-script=libcpy.ver
        -Wl,--default-symver */
libcpy.soV2 {};
```

redhat.

Versioned library

```
void f_old (char *d, char *s, long l) {
  memmove (d, s, l); }
void f_new (char *d, char *s, long l) {
  while (l--)
    d[l] = s[l]; }
__asm__(".symver f_old,mymemcpy@libcpy.so");
__asm__(".symver f_new,mymemcpy@@libcpy.soV2");
```

file libmymemcpy.ver:

```
/* use: -Wl,-soname,libcpy.so
        -Wl,--version-script=libcpy.ver */
libcpy.so {};
libcpy.soV2 {};
```

redhat.

Versioned library symbols

```
readelf --wide --dyn-syms libcpy.so

Num: Value Type Bind   Name
  7: 0068c FUNC GLOBAL mymemcpyold
 11: 0068c FUNC GLOBAL mymemcpy@libcpy.so
 12: 006b9 FUNC GLOBAL mymemcpynew
 16: 006b9 FUNC GLOBAL mymemcpy@@libcpy.soV2
```

redhat.

Versioned library functionality

The same application source:

built with / using	old	new plain	new versioned
old	OK	broken	compatible
new plain	OK	fix app	fix app
new versioned	OK + warning	fix app	fix app

warning: **no version information available**

Section 3

gdbserver usage

redhat.

gdbserver usage

- pro: no symbol files needed by gdbserver
- con: some advanced features are missing
- con: core files not supported
- more matching the "cloud" deployments
- future of GDB debugging

redhat.

gdbserver usage in extended mode

```
operatorhost$ gdb
(gdb) set sysroot remote:
(gdb) target extended-remote | ssh
          gdbserverhost gdbserver --multi -
(gdb) set remote exec-file /path/to/program
(gdb) file /path/to/program - local copy
(gdb) start
```

Execution via pipe is there only since Fedora 17.

redhat.

gdbserver usage with pipe

```
operatorhost$ gdb
(gdb) set sysroot remote:
(gdb) target remote | ssh
          gdbserverhost gdbserver - /path/to/program
(gdb) symbol-file remote:/path/to/program
(gdb) tbreak main
(gdb) continue
(gdb) sharedlibrary
```

Execution via pipe is there only since Fedora 17.

redhat.

gdbserver usage

```
gdbserverhost$ gdbserver :1234 /path/to/program
operatorhost$ gdb
(gdb) set sysroot remote:
(gdb) target remote gdbserverhost:1234
(gdb) symbol-file remote:/path/to/program
(gdb) tbreak main
(gdb) continue
(gdb) sharedlibrary
```

The end.

Thanks for listening.