Debugging DRLVM with GDB on Linux

This document describes the first steps for debugging DRLVM on Linux and provides some useful tips for successful debugging experience.

Enable Debug Build

1. Examine the components to decide which pieces of code you want do debug with GDB:
    - *VM core*
    - *classlib's native part*
    - *JITs*

They all are built in debug mode by default. So you should not care much. **Note**: If you want to start debugging right from the function `main`, you have to rebuild *classlib* in the debug mode, because `main` belongs to the Harmony launcher, which is a part of *classlib*.

2. To set a debug configuration for *VM core* explicitly, use "`sh build.sh -Dbuild.cfg=debug`" when building DRLVM.

3. To build *classlib* in the debug mode, use "`ant -Dhy.cfg=debug`", which is the default ("`ant -Dhy.cfg=release`" is the other option)

Before We Start GDB

Export `LD_LIBRARY_PATH` for GDB:

```
export LD_LIBRARY_PATH=$jre/bin:$jre/bin/default
```

Note: Any slashes at the end of each path and any symbolic links in the paths are prohibited.

Tip: To get the correct `LD_LIBRARY_PATH`, which wouldn't trigger process re-execing in the launcher (this breaks GDB and you won't be able to debug VM), run some java program that does not finish immediately. Then take a look at `/proc/{{pidof java`/environ}} and copy the `LD_LIBRARY_PATH` setting from it. This is the exact string, which launcher uses when it re-exacs the process. If you copy it exactly to `LD_LIBRARY_PATH` setting, launcher will not do exec.

Put some common GDB idioms for DRLVM into the `~/.gdbinit` file that will we sourced at GDB startup:

```
# Just a couple of cool things
#
set print pretty
set print object on

# Needed to run with DRLVM (ignoring SIGUSR2 in debugger that happens during GC, for example)
#
handle SIGUSR2 nostop noprint

# sr: execute all commands from file in current directory
#     it is useful to put commands like 'set args Hello' in .gdbc files
#
define sr
source .gdbc
end

# prnthr <num_threads>
#   print stack traces in a given number of threads
#
# TODO: print with verbose managed frames
#
define prnthr
    set $threadno = $arg0
    while ( $threadno > 0 )
        thread $threadno
        backtrace 20
        set $threadno = $threadno - 1
    end
end

# brst <file>
#     store all breakpoints to <file> to make them easy to reload on future sessions
#     to reload just type 'source <file>'
# undesired effect:
#     changes current logging file
# bugs:
#     assumes logging was off
#     assumes perl is installed
#
define brst
set logging file /tmp/gdb.tmp.breaks
set logging on
info break
set logging off
shell perl -p -i -e 'if ( /([^\s]*\.cpp:[0-9]*)/ ) { print "break $1\n"; } $_="";' /tmp/gdb.tmp.breaks
shell rm -f $arg0
shell cp /tmp/gdb.tmp.breaks $arg0
shell rm -f /tmp/gdb.tmp.breaks
end
```

As soon as you perform actions mentioned above, you can run GDB and, at least, see threads created, etc.

Debugging in GDB (or your favorite GUI, like DDD, kdb, etc.)

DRLVM is quite distributed between shared libraries, so it is not easy to set breakpoints somewhere in libraries' code. The possible approaches are the following:

1. Drop `asm("int3")` in your code, rebuild and catch it via an ordinary run from within GDB
2. Stop at the position when nothing interesting happened, but all libraries are loaded

You can use completion of function names where you set the breakpoints. For example, type:

```
break 'Jitrino::IRBuilder::gen
```

press <TAB>, you can see some variants suggested, type more and select between them. The leading single quote (') is significant!

For the second approach, I use two custom GDB scripts:

- "hstart" making the first stop on my favorite start point
- "hrun" using those breakpoints to stop on the point with further runs in the GDB session

To get these scripts, drop these lines into your `~/.gdbinit`:

```
define hstart
break main
run
break hysl_open_shared_library
continue
finish
disable
break compile_jit_a_method
continue
disable
end

define hrun
en 1
run
en 2
continue
finish
disable
en 3
continue
disable
end
```