eBPF sample programs

This directory contains a test stubs, verifier test-suite and examples for using eBPF. The examples use libbpf from tools/lib/bpf.

Build dependencies

Compiling requires having installed:

- clang >= version 3.4.0
- llvm >= version 3.7.1

Note that LLVM's tool 'llc' must support target 'bpf', list version and supported targets with command: `llc --version`

Clean and configuration

It can be needed to clean tools, samples or kernel before trying new arch or after some changes (on demand):

```
make -C tools clean
make -C samples/bpf clean
make clean
```

Configure kernel, defconfig for instance:

```
make defconfig
```

Kernel headers

There are usually dependencies to header files of the current kernel. To avoid installing devel kernel headers system wide, as a normal user, simply call:

```
make headers_install
```

This will creates a local "usr/include" directory in the git/build top level directory, that the make system automatically pickup first.

Compiling

For building the BPF samples, issue the below command from the kernel top level directory:

```
make M=samples/bpf
```

It is also possible to call make from this directory. This will just hide the invocation of make as above.

Manually compiling LLVM with 'bpf' support

Since version 3.7.0, LLVM adds a proper LLVM backend target for the BPF bytecode architecture.

By default llvm will build all non-experimental backends including bpf. To generate a smaller llc binary one can use:

```
-DLLVM_TARGETS_TO_BUILD="BPF"
```

We recommend that developers who want the fastest incremental builds use the Ninja build system, you can find it in your system's package manager, usually the package is ninja or ninja-build.

Quick sniplet for manually compiling LLVM and clang (build dependencies are ninja, cmake and gcc-c++):

```
$ git clone https://github.com/llvm/llvm-project.git
$ mkdir -p llvm-project/llvm/build
$ cd llvm-project/llvm/build
$ cmake .. -G "Ninja" -DLLVM_TARGETS_TO_BUILD="BPF;X86" \
           -DLLVM_ENABLE_PROJECTS="clang"    \
           -DCMAKE_BUILD_TYPE=Release        \
           -DLLVM_BUILD_RUNTIME=OFF
$ ninja
```

It is also possible to point make to the newly compiled 'llc' or 'clang' command via redefining LLC or CLANG on the make command line:

```
make M=samples/bpf LLC=~/git/llvm-project/llvm/build/bin/llc CLANG=~/git/llvm-project/llvm/build/bin/clan
```

Cross compiling samples

In order to cross-compile, say for arm64 targets, export CROSS_COMPILE and ARCH environment variables before calling make. But do this before clean, cofiguration and header install steps described above. This will direct make to build samples for the cross target:

```
export ARCH=arm64
export CROSS_COMPILE="aarch64-linux-gnu-"
```

Headers can be also installed on RFS of target board if need to keep them in sync (not necessarily and it creates a local "usr/include"

directory also):

```
make INSTALL_HDR_PATH=~/some_sysroot/usr headers_install
```

Pointing LLC and CLANG is not necessarily if it's installed on HOST and have in its targets appropriate arm64 arch (usually it has several arches). Build samples:

```
make M=samples/bpf
```

Or build samples with SYSROOT if some header or library is absent in toolchain, say libelf, providing address to file system containing headers and libs, can be RFS of target board:

```
make M=samples/bpf SYSROOT=~/some_sysroot
```