Trampoline Labs (Posix Target on Virtual Machine)

Linux host preparation

The Linux distribution is a Lubuntu, a lightweight Ubuntu version (Debian based).

- login : prets
- password: kywy

You can download the provided Virtual Machine (VirtualBox required), or execute the foolowing commands in your Linux. However, if you intend to use your own Linux, configure and check it prior to Prets labs.

Note: the **'$'** before each command is the user prompt.

install classical development tools

```
$ sudo apt install build-essential python3
```

You can update the region (language/keyboard) of the Linux distribution in:

```
menu=>settings=>LXQt settings=>keyboard and mouse=>keyboard layout
menu=>settings=>LXQt settings=>locale
```

Text editor

Install a basic text editor

```
$ sudo apt install gedit
```

Atom and VSCode are also installed on the Virtual machine:

```
sudo snap install code --classic
sudo snap install atom --classic
```

Communication

We will use *tmate* to share your terminal through a network and to ease the debug. You can use the one provided by Ubuntu directly (`sudo apt install tmate`), or get a newer version from the GitHub web site:

```
$ wget https://github.com/tmate-io/tmate/releases/download/2.4.0/tmate-2.4.0-static-linux-amd64.tar.xz
$ tar xJf tmate-2.4.0-static-linux-amd64.tar.xz
$ sudo cp tmate-2.4.0-static-linux-amd64/tmate /usr/local/bin/
$ rm -rf tmate-2.4.0-static-linux-amd64*
```

We will also use zoom:

```
$ sudo apt install ibus libxcb-xtest0 libgl1-mesa-glx
$ wget https://zoom.us/client/latest/zoom_amd64.deb
```

```
$ sudo dpkg -i zoom_amd64.deb
$ rm zoom_amd64.deb
```

Slack is also installed on the VM: Debian paquet on the web site.

Trampoline installation

Get Trampoline from GitHub, we will use a specific branch to get traces (functionnality to capture events of the kernel):

```
$ git clone https://github.com/TrampolineRTOS/trampoline.git
$ git checkout -b trace origin/trace
```

Goil

We now compile and install the goil OIL compiler:

```
$ cd trampoline/goil/makefile-unix/
$ ./build+release.py
$ sudo cp goil /usr/local/bin
```

Viper

Viper (Virtual Processor Emulator) is a tool that runs concurently to Trampoline in a separate Linux process. The communication between the process that runs Trampoline and the one of Viper is done through POSIX signals. In our application, Viper will send periodically signals to Trampoline, to mimics a periodic timer. As a consequence, Viper will be the tool that models the environment.

```
$ cd ../../viper
$ make
```

We make Viper available for this session (first line) and new ones.

```
$ export VIPER_PATH=/opt/trampoline/viper
$ echo 'export VIPER_PATH=/opt/trampoline/viper' >> ~/.bashrc
```

You can (should!) check the last command (only the last line of the following command is important):

```
$ tail ~/.bashrc
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
PATH=$PATH:/opt/trampoline/viper
```

install Trampoline

Then, we install Trampoline in `/opt`. This is a read only directory for custom apps.

```
$ cd
$ sudo mv trampoline /opt
```

labs

We now get the labs, and test the installation with the compilation and execution of the lab1.

```
$ git clone https://github.com/TrampolineRTOS/Labs.git
$ cd Labs/posix/labs/lab1
```

There are 2 files:

- `lab1.oil` is the *description file* of the application. There is only 1 task defined, with the AUTOSTART attribute.
- `lab1.c` is the *application file*. Here is the `main` and the implementation of the task.

We first compile the `lab1.oil` file using the goil compiler:

```
$ goil --target=posix --templates=/opt/trampoline/goil/templates/ lab1.oil
Created 'lab1/tpl_os.c'.
Created 'lab1/tpl_os.h'.
Created 'build.py'.
Created 'make.py'.
Created 'lab1/tpl_app_custom_types.h'.
Created 'lab1/tpl_app_config.c'.
Created 'lab1/tpl_app_config.h'.
Created 'lab1/tpl_app_define.h'.
Created 'lab1/tpl_static_info.json'.
Created 'readTrace.py'.
Created 'lab1/stm_structure.c'.
Created '/home/prets/Labs/posix/labs/lab1/build/lab1.oil.dep'.
No warning, no error.
```

The application dependant files are generated in `lab1/` directory, and a python based build script is provided (`make.py`). So, we make the application:

```
$ ./make.py
Nothing to make.
Making "build/os" directory
[  5%] Compiling /opt/trampoline/os/tpl_os_kernel.c
[ 10%] Compiling /opt/trampoline/os/tpl_os_timeobj_kernel.c
[ 15%] Compiling /opt/trampoline/os/tpl_os_action.c
[ 21%] Compiling /opt/trampoline/os/tpl_os_error.c
[ 26%] Compiling /opt/trampoline/os/tpl_os_os_kernel.c
[ 31%] Compiling /opt/trampoline/os/tpl_os_os.c
[ 36%] Compiling /opt/trampoline/os/tpl_os_interrupt_kernel.c
[ 42%] Compiling /opt/trampoline/os/tpl_os_task_kernel.c
[ 47%] Compiling /opt/trampoline/os/tpl_os_resource_kernel.c
[ 52%] Compiling lab1.c
```

```
Making "build/lab1" directory
[ 57%] Compiling lab1/tpl_app_config.c
[ 63%] Compiling lab1/tpl_os.c
Making "build/machines/posix" directory
[ 68%] Compiling /opt/trampoline/machines/posix/tpl_machine_posix.c
[ 73%] Compiling /opt/trampoline/machines/posix/tpl_viper_interface.c
[ 78%] Compiling /opt/trampoline/machines/posix/tpl_posix_autosar.c
[ 84%] Compiling /opt/trampoline/machines/posix/tpl_posix_irq.c
[ 89%] Compiling /opt/trampoline/machines/posix/tpl_posix_context.c
[ 94%] Compiling /opt/trampoline/machines/posix/tpl_trace.c
[100%] Linking lab1_exe
```

The binary `lab1_exe` is built. We can run:

```
$ ./lab1_exe
Hello world
```

We have to quit using `Ctrl+C`, because it is an endless application

We can now read the kernel events of this simple application:

```
$ ./readTrace.py
[         0] proc idle                change to state READY_AND_NEW
[         0] proc task1               change to state READY_AND_NEW
[         0] proc task1               change to state RUNNING
[         0] proc task1               change to state SUSPENDED
[         0] proc idle                change to state RUNNING
```

The trace is quite short and show the state changes of the tasks. All is done at startup (date is 0) and `idle` task will run forever.

If you get these outputs, your Trampoline installation is Ok!