Chapter

52

Remote Target Simulation

This chapter describes how to run a simulator on a target and connect it to the SDL simulator interface (SimUI) on the host via TCP/IP communication.

Target Simulation

Overview

A simulator executable (called *simulator*) communicates with the SDL suite tools, e.g. the SDL Simulator UI (SimUI), via the SDL suite communication mechanism (the PostMaster). By using some special features it is possible to run the simulator on a target system (called *target*), while the SDL suite tools and Postmaster run on a host system (called *host*) separate from the target. This set-up is known as *target simulation*. This gives the possibility to test a simulator in its correct environment and still get the trace and debug possibilities offered by the SDL suite.

For instance, the target could be a VME147 bus system running VxWorks, and the host is a Sun workstation running SunOS 5. The simulator running on the target is built using a special library, instead of the ordinary simulator library containing the simulator kernel. The special library contains the kernel for a simulator on VxWorks, as well as a communication interface to the host. On the host, the simulator is replaced by a gateway executable (called *gateway*) that communicates with the simulator on the target over the TCP sockets connection. Figure 471 shows the set-up for target simulation.

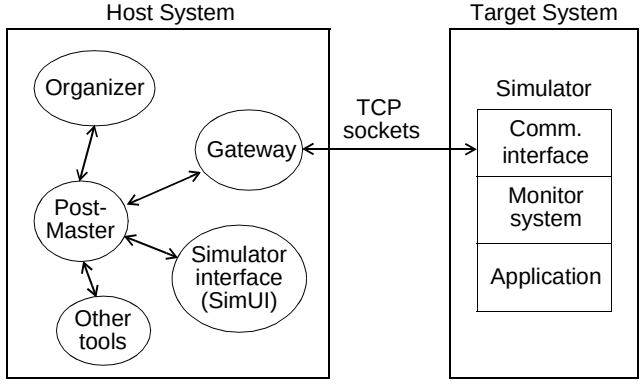

Figure 471: Target simulation set-up

Architecture

Simulators, as well as other executables communicating via the Postmaster, has a set of layers as shown in Figure 472, where the layers are represented by the files that implement them. All units (the Simulator, Postmaster and SimUI) are located on the same host machine.

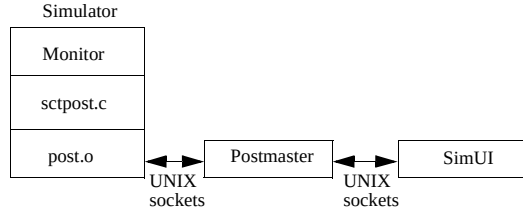

Figure 472: Architecture of an ordinary simulator

`sctpost.c`, `post.o` and a few other files are then linked into the simulator kernel used for making the simulator.

The basic principle of this target simulation is to direct the communication between the simulator and the Postmaster over a TCP sockets connection. This re-direction must be transparent to the simulator and to the Postmaster.

To direct the messages between the Simulator Monitor and the Postmaster through the TCP sockets, a new layer, implemented in the file `tlayer.c`, is introduced between `sctpost.c` and `post.o` according to Figure 473.

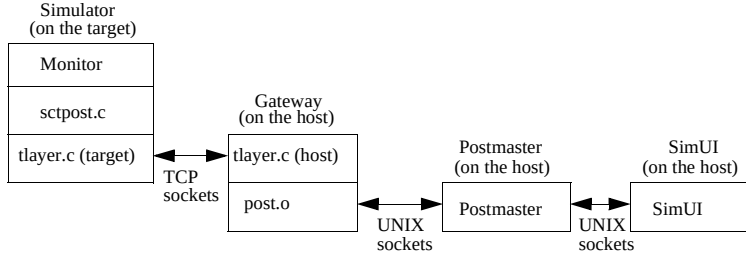

Figure 473: tlayer.c is introduced

For the target, `tlayer.c` emulates `post.o` (the same interface), and for the host, `tlayer.c` emulates `sctpost.c` (the same interface). (**In Windows NT** there are other communication mechanisms on host than UNIX sockets.)

The Configuration File and Simulator UI Command Extension

All configuration is made on host side in a file called `tarsim.cfg`. This file is searched by `host_smr.sct/host_smr.exe` in a directory specified by the environment variables (searched in this order):
Unix: `$TARSIMPATH, $PWD, $HOME, $sdtdir`.
Windows NT: the only searched variables are `%TARSIMPATH%`. You set the environment variable `%TARSIMPATH%` as described below.

1. Open the System Control Panel and select the *Environment* tab.
2. In the *Variable* field enter **`TARSIMPATH`**, and in the *Value* field enter the directory (including logical disk drive) where `tarsim.cfg` is located.
3. Press *Set* and close down the System Control Panel.

The syntax of `tarsim.cfg` is described below. In the delivery, a sample `tarsim.cfg` is supplied.

- All lines beginning with '#' (hash) are considered as comments and left unread.
- All lines after the optional command END are left unread, regardless of contents.
- All commands must be at the first position in a line.
- All commands and arguments are case insensitive, except arguments specifying files, etc.
- Separation between commands and arguments are any whitespace (i.e. space or tab).

Example 323: A sample `tarsim.cfg` ----------

```
# targsim.cfg, config file for simulating an SDT executable simulator
# on remote target with SDT SIMUI on a host.
#
# forces use of invalid IP addresses. gethostbyaddr on some networks
# returns an invalid hostent entry even if the address is usable.
# FORCEIP must come before any used IP address in TARGET or HOST
# default OFF
FORCEIP         ON
#
#SERVICE        IP-ADDRESS              PORT    NAME(optional)
TARGET          146.75.124.23           18000   targetcomputer
#HOST           111.22.33.55            17000   SDT-host
# Target must be specified, host not. Port number is default, target
# has 18000 defined in tlayer.h
#
# setting USETIMEOUT to OFF disables timeouts and makes it possible
# to suspend the target or halt it at a C debugger breakpoint
# default ON
USETIMEOUT      OFF
#
# communication between host and service, only TCP (default) or TTYA/B (n/a)
#           NOTE, only TCP in this version
#COM            TARGET                  TCP
#
# in debug mode signals can be logged on file (default = stderr)
# Note! stderr will not work on Windows NT.
#
#LOG            dummylog.tmp
#
# debug on/off must be set to 'on' to activate logging.
#
#debug          on      #default = OFF
#
# If SDL diagram files are in a different directory than is hardcoded
# into the simulator (which assumes they are in the source directory
# which was specified in SDT when it was generated), you can explicitly
# give the path to the SDL source files here.
# Note! absolute path from root, i.e. starting with /dir1/dir2/... etc.
# trailing directory (a / in the end) is optional.
# If you can not get symbol trace on SDL editor, the SDLpath need to be
# defined to where you have your SDL source files
# Note! this only works on a flat directory structure.
#
#SDLpath        /home/kod/sdl/demongame
#
end
#
after an "end" statement host stops reading, so you can write anything!
```

Tarsim Shell (UNIX only)

On UNIX, there is a very basic shell included in `host.sct`.

If you start `host.sct` without the target running, you will enter a shell where only the commands **INIT** (try contacting target again) and **EXIT** (quit session) are applicable. If you know your target is running and still

do not get contact, please check that the specified IP address and PORT of the target in `tarsim.cfg` are correct. PORT is for the target hard-coded in `tlayer.h`, the header file for `tlayer.c`.

During simulation you can enter the tarsim shell by entering a "**:**" (colon). You can only leave the simulation if the simulator is not running with **GO** or **FOREVER**. If so, first stop the simulation with the *Break* button.

After entering the "**:**" you will see a welcome text to the shell. You can enter the command **HELP** to see the full command set, or **STATUS** to see the current simulation setup. You exit the shell by entering **EXIT**.

Note:

Be sure you are still in the tarsim shell when entering EXIT, since it is the same command for exiting the simulation. You can see if you are in the shell by the fact that all text begin with a colon as the first character in the text window. There is a time-out in the shell, after one minute of inactivity you get back to the simulator again.

LOG and **DEBUG** has the same syntax and meaning as in `tarsim.cfg`.

Addresses like TARGET and HOST, as well as the COM and SDLPATH statements, cannot be changed from the shell.

An additional facility for the logging is **GREP**, which gives a possibility to filter out up to 5 strings that you want to log and skip the rest.

To clear all GREP's and log everything:

```
GREP    *
```

To add a string (this example looks for MSCE, i.e. signals to the MSC Editor):

```
GREP    MSCE
```

Strings are not case sensitive by default. To search for the **exact** string `MyPath/demo.sdt`, prefix it with '=':

```
GREP    =MyPath/demo.sdt
```

If you want to log everything **except**, for instance, signals to the SDL Editor you can use the prefix '!':

```
GREP     !SDLE
```

Note:

If you have GREP MSC active and then enter GREP !MSC, status will be changed of the previously entered argument, so you cannot enter two contradictions and have both active.

You can also combine the inverse and exact functions:

```
GREP     !=MyPath/demo.sdt
```

Up to five GREP arguments can be active simultaneously and each can be up to 25 characters long.

If the GREP buffer is full, empty the old with GREP *.

You can empty the buffer and enter new argument in one statement:

```
GREP     *       MSCE
```

Executing and Terminating the Tools

When the simulator and gateway have been started, the simulator can be controlled from the Simulator UI on the host in the same way as an ordinary simulator. Differences may be seen in execution performance and in the printing of log messages. Using *Open* in the *File* menu in the SimUI, you specify the simulator file `host.sct`. After you have chosen this file, `host.sct` will try to establish contact with the target, according to settings in `tarsim.cfg`. This file must be in the same directory as `host.sct`, and the SDL suite should also be started from this directory. (Setting Source Directory in the Organizer does not help SimUI find the target simulation file dependencies.)

The simulator on the target can be stopped by typing `<Ctrl+C>` on the console. It is though recommended only to stop the target by pressing the *Exit* button in the SimUI. The `host.sct` will then, before it finishes, send a stop signal to the target, and give it a chance to close all open file descriptors for TCP sockets. If not, you can get a bind error if you try to restart the target simulator without rebooting the target system.

The gateway on the host is terminated by issuing the command *Exit* or *Quit* in the SimUI.

When the simulator and gateway have been started, the simulator can be controlled from the Simulator UI on the host in the same way as an ordinary simulator. Differences may be seen in execution performance and in the printing of log messages. Using *Open* in the *File* menu in the SimUI, you specify the simulator file `host_smr.sct`. After you have chosen this file, `host_smr.sct` will try to establish contact with target, according to settings in `tarsim.cfg`. This file must be in the same directory as `host_smr.sct`, and the SDL suite should also be started from this directory. (Setting Source Directory in the Organizer does not help SimUI find the target simulation file dependencies).

The simulator on the target can be stopped by typing `<Ctrl+C>` on the console. It is though recommended only to stop the target by pressing the *Exit* button in the SimUI. The `host_smr.sct` will then, before it finishes, send a stop signal to the target, and give it a chance to close all open file descriptors for TCP sockets. If not, you can get a bind error if you try to restart the target simulator without rebooting the target system.

The gateway on the host is terminated by issuing the command Exit or Quit in the SimUI.

On Windows NT the host gateway is named `host_smr.exe`.

Known Problems

- Some targets must be rebooted after a crash, since the OS does not make `fclose` on open file descriptors upon kill. You will see the error message “bind (0x30)”, meaning that bind fails since the target port address is busy. However, since enhanced shutdown and bind methods has been introduced, the risk of this has been reduced.
- Sometimes the SimUI interface hangs with a clock cursor after several reboots. Try the following solutions in the specified order until it works:

  1. Press the *Break* button in the SimUI.
  2. Select *Restart* in the *File* menu and restart the file `host.sct`
  3. If you still get no access to the SimUI, try `<Ctrl+O>` (shortcut for open file) to reopen the file.

4. Find `host.sct` in the process list (`ps -ef` on Solaris) and kill it. Try 2-3 again.

5. Kill `sdtsimui` in the same way and restart it from the Organizer.

- If you get no contact with the target the first time, check the `tarsim.cfg` configuration file. Correct any incorrect IP address or port to the target (port is default 18000 except on VxWorks/VME) and restart `host.sct` like in step 2 above. The target's port is hard-coded into the file `tlayer.h`.

- If you cannot establish connection on an IP address and you still can ping/telnet the same address, try to set the command `FORCEIP ON` before any `TARGET` statement in `tarsim.cfg`. This forces use of an IP address even if it is reported invalid by `gethostbyaddr()`.

- On a slow network connection you might lose connection with the MSC editor. It can be impossible to start it by pressing the MSC button in the Simulator User Interface, or you can get the message "Error writing MSC event". Other problems that can occur is that it is impossible to `BREAK` when you are running the simulation with `GO` or `GO-FOREVER`.
  The solution to all of these problems is to set the environment variable `sdt_rpctimeout` to `100`. You need to restart the SDL suite after doing this.

- If simulation starts but crashes after starting the MSC trace you are running out of stack space in the target. There is two solutions. Add the compiler flags MAX_READ_LENGTH=500 and MAX_WRI_LENGTH=500 in the makefile when you build the target. The flags reduces the default buffer size used by the simulation kernel. You may need to adjust the value after size and complexity of your system.
  The second fix is to set the stack size for the SDL task when you start it. Most RTOS:s has stack size as a parameter to taskSpawn, t_create or corresponding. Suitable stack size for remote simulation is 30-50 kB. Default stack in many RTOS kernels is 20kB, which will only work for very small systems.

Source Code Files

Source code for the target contains the complete standard Master Library. The only difference is that `post.o` does not contain any Postmaster, but a TCP socket communication program.

Both host and target use the same `tlayer.c` file, with the compile flags `-DTARGET` and `-DHOST`, respectively. Also the flag `-DTCP` need to be specified in both ends to get TCP functionality.

All other files in `sctworld.o` needs the flags `-DTARGETSIM`, `-D/compiler/`, `-D/rtos/`, `-DSCTDEBCLCOM` and `-DXMAIN_NAME=sdlmain`.

Also recommended: `-DMAX_READ_LENGTH=500` `-DMAX_WRI_LENGTH=500`

See the specific integrations for the RTOS and compiler specific flags.

There is also special flags for the TCP connection. Supported flags are currently VXWORKS_SDT, PSPS_SDT and WINDOWS. Default is standard Unix/Posix.

The source file structure in `tlayer.c` is also documented in the top of `tlayer.c` in a comment.

To make the target you will need `tlayer.c` and `tcp.c`, along with the `.h` files which are supplied. A makefile is also provided.

Building a remote simulation project

In the remsim directory there is a subdirectory called SampleProject
Into the `INCLUDE` directory copy *.`c` and *.`h` from the Telelogic Tau installation under `sdt/sdtdir/{ARCH}sdtdir/INCLUDE` and `sdt/sdtdir/remsim/source`. The makefiles are referring to this directory. You could of course change the makefiles to refer to the original installation if you do not need any changes.
In the directory `REMSIMKERNEL` you have the file `Makefile.m` to build a new kernel and `make.opt` for compiler settings used by both final build and making the kernel.
comp.opt defines the syntax for the generated makefile.
In `make.opt` you could by setting `sctLD, sctLDFLAGS` and `sctEXTENSION` decide if you want to build the final executable or a library to link into the final target build.

You build the kernel in the REMSIMKERNEL directory. The sample makefiles will work under Sun/Solaris with a GCC compiler. You only need to change for your cross compiler environment. Make the kernel by typing:

```
make -f makefile.m
```

The result will be a prelinked kernel called sctworld.o.

When you perform a make, you need to choose the option "Generate Makefile and use template" and select env.tpm in the REMSIMKERNEL directory. This links in the template sctenv.c (which in the delivery does nothing). You also need to select "Use kernel in directory" and select REMSIMKERNEL.
Please note that this only sample code to give you ideas how you can build your own cross compilation environment.

Special Information about Windows NT

Compiling

Both target and host has been tested on Windows NT 4.0 and compiled with Borland C++ v5.02 and Microsoft Visual C++ 5.0 and 6.0.

In the directory sdt/sdtdir/remsim/wini386, makefiles for Borland and Microsoft compilers are stored. Normally you will probably not run the target under Windows NT; this is just for demo purposes. Edit the top of the makefile, variables ccbindir and sdtsource to the absolute paths of your compiler binaries and the SDL suite installation. Make from the prompt like this:
Borland: make -f bcc_target.mak
Microsoft: nmake /f ms_target.mak

The host part is called host_smr.exe and is located into the SimUI the same way as on UNIX.

Both parts must be compiled as 32 bit console applications.

Non-supported Features

Due to the fact that there are no stdout or stderr when you run a Windows GUI application, together with the difficulties to determine the right directory to write to a file in, all debugging and logging facilities

has been excluded. That means that the entries DEBUG, GREP and LOG in `tarsim.cfg` will be ignored. There is no extra Target Simulation Shell as on UNIX (which is not needed when you cannot do any message logging). None of this affects the target simulation as such.

Other Known Limitations

It is considerably slower to do a remote simulation in Windows NT than on UNIX, even with the same system, same options, and running target and host on the same machine. There are ways to speed up the simulation. Recommended is to give **`set-tr 0`** as the first command in SimUI. Then all textual tracing is stopped. Instead you can ask for the value of certain variables of interest. MSC trace and SDL trace works OK; MSC is the quicker of the two.

You can of course do a normal host simulation on the NT machine and then you have no limitations. Target simulation is not recommended to be used more than for the final fine tuning into the target environment.

It is especially important in Windows to use the option `SDLPATH` in `tarsim.cfg` if the target has been compiled on another system. The parser in `host.exe` can read NFS syntax "/path/" and will convert it to Windows/DOS syntax "\path\". The thing that cannot be fixed automatically is the correct logical drive letter. Hence you can specify where your SDL source files are with:

```
SDLPATH D:\path\sdlsource
```

in `tarsim.cfg`. Of course the source files must be the same as the ones the target was compiled from.