CASE STUDY

Performing remote debugging at user sites

Project: A major contractor created a widely-distributed DoD (U.S. Department of Defense) system designed to assess military readiness for a variety of emergency situations.

Problem: To remain effective, this system needs close to 100 percent uptime. But the budget would not cover flying senior support engineers to multiple sites to track down every bug that appeared during operational testing.

Solution: The contractor was already using Aprobe technology to find bugs in its integration testing lab. They soon realized its power could be extended to remote debugging.

Probes were defined by the contractor in the contractor's test lab, then sent by e-mail or ftp to user sites. A technician at each site loaded the probes into the Aprobe directory and re-started the system. As the application ran, trace data was logged. The trace captured code-level, system-level and hardware/software configurationdetails, minimizing the data each site had to supply manually.

At any point, the trace data could be e-mailed back to support staff, who would step through the trace. This helped them zero in on bugs quickly.

They would also use Aprobe to create a temporary patch to test a fix. When the fix worked, it could be left in place until the next build was ready.

Remarks: OCS was able to debug the problem in the customer's environment without burdening the customer with extra tasks. Remote debugging avoided the high cost and delay of sending senior support staff to do on-site troubleshooting. The probes had virtually no impact on system performance.

These application-specific probes will be used throughout the life of the military system to ensure rapid time-to-resolution of any issues.

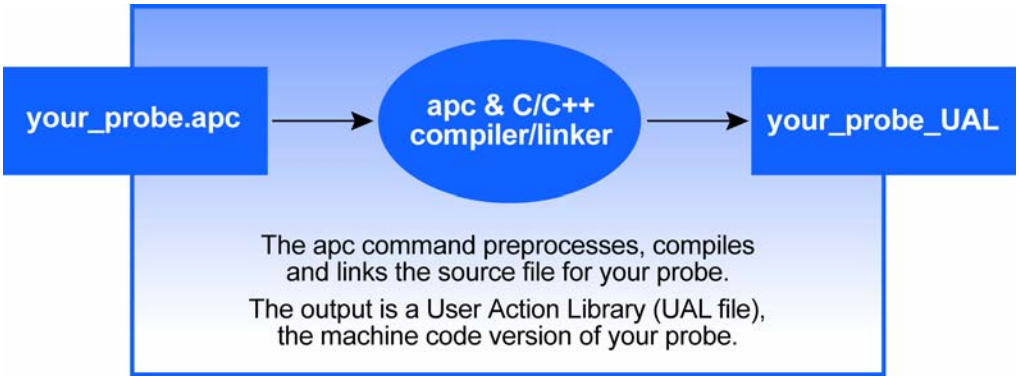
Compiling and linking probes

The C code for a probe contains several non-ANSI directives, such as:

- `probe`, `on_entry`, `on_exit`, `on_line` and `on_offset`, which specify where the probe will be inserted in the application.
- `log`, which logs data to buffers and/or log files
- `$return`, `$myParm1`, `$myParm2`, `$$EAX`, and others which refer to return values, positional parameters, registers, and so forth. (The prepended "$" is followed by the identifier for functions, variables, and parameters in the target application.)

These directives are processed by **`apc`**, the Aprobe preprocessor for C. **`apc`** automatically generates pure ANSI C functions, and translates the directives to calls to the Aprobe API from your probes.

Your probes (now in the form of ANSI C functions) are then compiled by a standard C compiler, linked, and stored in a library called the User Action Library (UAL file).



The UAL file is implemented as a DLL on Windows and as a shared library on Unix/Linux.

In effect, the one or more probes (patches) that you write in C are translated into a shared library. That shared library contains not only the probes, but code that specifies where in the application to insert each probe.