76.2.5 Compaq's/Digital's/HP's Third Degree

Third Degree is a tool for memory leak detection and memory access checks. It is one of the many tools in the ATOM toolkit. The toolkit is only available on Tru64 (formerly known as Digital UNIX formerly known as DEC OSF/1).

When building Perl, you must first run Configure with -Doptimize=-g and -Uusemymalloc flags, after that you can use the make targets "perl.third" and "test.third". (What is required is that Perl must be compiled using the `-g` flag, you may need to re-Configure.)

The short story is that with "atom" you can instrument the Perl executable to create a new executable called *perl.third*. When the instrumented executable is run, it creates a log of dubious memory traffic in file called *perl.3log*. See the manual pages of atom and third for more information. The most extensive Third Degree documentation is available in the Compaq "Tru64 UNIX Programmer's Guide", chapter "Debugging Programs with Third Degree".

The "test.third" leaves a lot of files named *foo_bar.3log* in the t/ subdirectory. There is a problem with these files: Third Degree is so effective that it finds problems also in the system libraries. Therefore you should used the Porting/thirdclean script to cleanup the **.3log* files.

There are also leaks that for given certain definition of a leak, aren't. See PERL_DESTRUCT_LEVEL for more information.

76.2.6 PERL_DESTRUCT_LEVEL

If you want to run any of the tests yourself manually using e.g. valgrind, or the pureperl or perl.third executables, please note that by default perl **does not** explicitly cleanup all the memory it has allocated (such as global memory arenas) but instead lets the exit() of the whole program "take care" of such allocations, also known as "global destruction of objects".

There is a way to tell perl to do complete cleanup: set the environment variable PERL_DESTRUCT_LEVEL to a non-zero value. The t/TEST wrapper does set this to 2, and this is what you need to do too, if you don't want to see the "global leaks": For example, for "third-degreed" Perl:

```
    env PERL_DESTRUCT_LEVEL=2 ./perl.third -Ilib t/foo/bar.t
```

(Note: the mod_perl apache module uses also this environment variable for its own purposes and extended its semantics. Refer to the mod_perl documentation for more information. Also, spawned threads do the equivalent of setting this variable to the value 1.)

If, at the end of a run you get the message *N scalars leaked*, you can recompile with `-DDEBUG_LEAKING_SCALARS`, which will cause the addresses of all those leaked SVs to be dumped; it also converts `new_SV()` from a macro into a real function, so you can use your favourite debugger to discover where those pesky SVs were allocated.

76.2.7 Profiling

Depending on your platform there are various of profiling Perl.

There are two commonly used techniques of profiling executables: *statistical time-sampling* and *basic-block counting*.

The first method takes periodically samples of the CPU program counter, and since the program counter can be correlated with the code generated for functions, we get a statistical view of in which functions the program is spending its time. The caveats are that very small/fast functions have lower probability of showing up in the profile, and that periodically interrupting the program (this is usually done rather frequently, in the scale of milliseconds) imposes an additional overhead that may skew the results. The first problem can be alleviated by running the code for longer (in general this is a good idea for profiling), the second problem is usually kept in guard by the profiling tools themselves.

The second method divides up the generated code into *basic blocks*. Basic blocks are sections of code that are entered only in the beginning and exited only at the end. For example, a conditional jump starts a basic block. Basic block profiling usually works by *instrumenting* the code by adding *enter basic block #nnnn* book-keeping code to the generated code. During the execution of the code the basic block counters are then updated appropriately. The caveat is that the added extra code can skew the results: again, the profiling tools usually try to factor their own effects out of the results.