Daemon threads

Your operating system has various threads that provide services to user threads. That's their sole purpose: to serve user threads. The term originated with the Unix operating system, but most operating systems have them in one form or another, with slightly differing properties. Examples are:

- In Unix, *crond* is job scheduler, running jobs in the background. Line printer daemon *lpd* manages printer spooling.
- In Windows systems, daemons are called *Windows services*. One such is Prefetch-and-Superfetch, which caches frequently used files and other components to RAM. The *DNS Client* resolves domain names to IP addresses.
- In Java, the *garbage collector* is a daemon, continuously finding objects that cannot be reached from the Java program and putting the memory they occupy on a "free list" so the memory can be used for some other purpose.

We won't be writing Java programs that use daemon threads. (Look at the next page for a realistic use of a daemon thread.) Nevertheless, a basic understanding of daemons, including where the word came from, is useful.

The basic property of a Java daemon thread

In Java, suppose your main-program thread is named t. Suppose t creates and starts threads t1, t2, and t3. Suppose also that it has made t3 a daemon (we show how to do this later). Here is the basic difference between t3 and the other threads:

When thread t, t1, and t2 die (i.e. complete method run()), daemon thread t3 is killed.

Thus, as long as one normal thread is alive, daemon threads stay alive. But when *all* normal threads die, all daemon threads are killed. Why? Their job was to work in the background, supporting the normal threads. When there are no more normal threads, there is no reason for them to live.

Methods dealing with daemons

Two methods in class Thread deal with daemons:

t.isDaemon()	Return true iff t is a daemon.
t.setDaemon(b)	Make t a daemon if b is true and a non-daemon if b is false. This method must be called before t is started; otherwise an exception is thrown.

Why was the name daemon chosen?

I thought a demon, or daemon since you spelled it that strange way, was an evil spirit, a devil, a fiend, a ghost, an imp. Why is it used for a good thing —for a background thread that serves?

The term *daemon* was coined by people in MIT's project MAC, in 1963, which did groundbreaking work in operating systems. At the website www.takeourword.com/TOW146/page4.html, Prof. Fernando J. Corbato of MIT talks about the use of the word *daemon*:

> Our use of the word daemon was inspired by the Maxwell's daemon of physics and thermodynamics. (My background is Physics.) Maxwell's daemon was an imaginary agent, which helped sort molecules of different speeds and worked tirelessly in the background. We fancifully began to use the word daemon to describe background processes that worked tirelessly to perform system chores.

Example of a daemon thread

I, Gries, with 57 years experience programming, have no idea how to write a useful daemon. But these up-and-coming youngsters like Sampson do. I asked Adrian whether he had a good example, and this is what he replied:

> Here an example inspired by a real-life situation I had to deal with recently. Say you have a program that has a bunch of time-consuming work to do and you'd like to offload some of it to a server. It can be useful to have a separate thread manage all the logistics of queuing up the work for the remote server, sending it along, and keeping track of its completion. This auxiliary thread isn't the main thrust of your program, but it has to wait around forever to manage communication with the main thread and from the server. It should die automatically when the main thread finishes. That's what a daemon does.

Well, that didn't tell me what the daemon was, how big it was, what the code looked like, just what it did. So I asked for more information. It turns out the daemon he wrote was in Python. Here's what he said:

> This particular daemon happened to be in Python. In fact, the code's online—here's the line where I set the client thread in question to be a daemon thread (same semantics as in Java).
>
> https://github.com/sampsyo/cluster-workers/blob/master/cw/client.py#L54
>
> Basically, the worker client thread needs to go into an event loop, waiting for socket events from the server and for work enqueuing from the main client code. It's usually just sitting around in an OS `select` call waiting for one of these two things to happen. And it's still waiting in `select` when the main thread finishes all its work and wants to shut down. Rather than having a complicated messaging system to tell the client thread to shut down, it just dies silently because it's marked as a daemon. Neat & clean!

Well, I, Gries, still don't know that much about this daemon thread. But I *do* know that if I want to learn more about what's going on in computing and programming languages, I should join Adrian Sampson's research team —if he'll let me.