Stubborn Processes: How to kill ColdFusion 5?

Posted At : July 18, 2003 11:12 AM | Posted By : Steven Erat
Related Categories: Linux, ColdFusion, New England

I'm working on a very interesting problem where ColdFusion 5 on RedHat Linux 7.2 won't stop, won't restart, and can't be killed by root. First time I've ever come across this... Only way to stop the processes is to reboot the machine.
I never knew that a situation could arise where root could not **kill a running process**. Thanks to the help of the folks from the Boston Linux and Unix User Group, **BLU**, for giving me some possible explanations and troubleshooting steps. Here's the start of the **thread archive**.
BLU suggests that a buggy device driver (not to be confused with a software ODBC driver) that interacts with the hardware could be in a hung state, and if a process is making a call to that driver when it gets hung up, then the process might not die. A hardware failure might be to blame here.

Troubleshooting steps to gather more details include:

- run strace against the parent cfexec and parent cfserver pids
- run "ps -elf | grep cf"
- run "lsof -p [cfexec | cfserver] pid" to see what they are holding onto
- get dmseg to look for any hardware failure messages

Well, I'll follow up more here on the results when I get this information.Update 7/22: The mailing list thread continued over the weekend, with some suspecting that CF's processes were zombies. One member replied:

> A zombie process can not be killed because it is already dead (hence the name)... Most of its resources have already been released, but its entry in the process table hangs around, waiting for its parent to collect its exit status. Zombies are generally the result of careless programming, where a process forks a child, and does neither of the following:
> - call one of the wait() functions to collect the child's exit status
> - tell the child it is not interested in the exit status
> Usually, what will happen in such cases is the parent will eventually die, and init will become the zombie's parent process. The init process periodically collects the exit status of orphaned child processes. When this happens, the zombie will disappear. If the parent process was a daemon which does not die until the system goes down, then the zombie process will remain until the system is shut down.

and then this, regarding how a process can be immune to death by root:

> If you look at the output of ps, you will see that the process is in state 'D', which is usually called "I/O wait state" (though the 'D' stands for disk, and another common name for this is "Disk wait state"). The process is blocked in a system call which is waiting for I/O from the kernel. The process can

> not be killed, because while it is in that state it is executing in the context of the kernel. As has been pointed out, the most common causes of this condition are: - waiting for I/O from an NFS server which has crashed or otherwise stopped responding to requests - waiting for I/O from a device which has a hardware fault, or whose driver is buggy, and as a result is not returning the expected data Another possibility could be a resource deadlock, but that would generally result in two co-dependent processes sleeping, waiting for eachother to release some resource... It generally would not result in a process in 'D' state.

Update 7/29: Just found another mailing list that discusses how processes are immune to being killed, and I found **this particular thread** to be helpful:

> The `D' tells you that the process is in an uninterruptible disk wait state. This can only be caused by a kernel problem with the i/o routines, a filesystem problem, or a device driver problem. There really isn't anything you can do but restart the machine to get rid of that process unless you're lucky enough to have it return from it's wait (not likely). For some explanation, a problem like this is caused when an application tries to read/write from disk, but the i/o call never returns, and it doesn't result in a timeout either. `kill -9', as most of you know, is used to send SIGKILL to a process. The SIGKILL signal can not be blocked, ignored, or handled by an application in any way, which means that once your application receives a SIGKILL, it's done for. The reason it doesn't work in this situation, however, is that the program needs to accept the signal that's trying to be sent first. It can't do this while it's waiting to return from it's disk activity. If you'd like something to read, check out "Advanced Programming in the UNIX Environment" (Stevens 1992). It's got some very useful information regarding signals and how they're handled. Think of it this way: the process called some kernel function and went (eg. was swapped) away until that function returns. There's no process to kill, only the kernel function => kernel itself.