Category: Engineering

Kill -9 does not work

From Noah.org

Contents

Why doesn't `kill -9` always work?

You are supposed to be able to kill any process with `kill -9 [PID]`, but you may come across a process that can't be killed. Usually this happens when you are trying to kill a <defunct> process. These are processes that are dead and have exited, but they remain as **zombies** in the process list. The kernel keeps them in the process list until the parent proces retrieves the exit status code by calling the wait() system call. This does not usually happen with daemon processes because they detach themselves from their parent process and are adopted by the init process (PID=1) which will automatically call wait() to clear them out of the process list. You may sometimes see the daemon defunct PID in the process list for a brief moment before it gets cleaned up by the init process. You don't have to worry about these. You can also end up with an unkillable process if a process is stuck waiting for the kernel to finish something. This usually happens when the kernel is waiting for I/O. Where you see this most often is with network filesystems such as NFS and SaMBa that have disconnected uncleanly. This also happens when a drive fails or if someone unplugs a cable to a mounted drive. If the device had a memmapped file or was used for swap then you may be really screwed. Any kernel calls that flush IO may hang forever waiting for the device to respond.

Which processes are wedged?

Look for process in the state 'D' (uninterruptible sleep) or in the state 'Z' (defunct zombie). The following command will list processes in state 'D' or 'Z'. Note that if no processes are in state 'D' or 'Z' then this will still print the `ps` header, but nothing else.

```
ps Haxwwo stat,pid,ppid,user,wchan:25,command | grep -e "^STAT" -e "^D" -e "^Z"
```

For testing you might want to add normally suspended/sleeping processes:

```
ps Haxwwo stat,pid,ppid,user,wchan:25,command | grep -e "^STAT" -e "^D" -e "^Z" -e "^S"
```

kill stuck process

After you send a kill signal to a stuck process you must also send a kill signal to the `rpciod` kernel thread (it will restart when needed).

```
ps Haxwwo pid,command | grep "rpciod" | grep -v grep
```

unmount stuck filesystems

You can sometimes kill a process by unmounting filesystems that it is stuck waiting for. If that doesn't cause the process to generate an IO error or a segfault then go back and try killing the process again.

Use both `mount` and `cat /proc/mounts` to see what filesystems are mounted. Sometimes `mount` will not show NFS mounts where a previous `umount` is still pending -- yet another headache when dealing with NFS.

You can use `fuser` to show which processes have filedescriptors open to a given filesystem. In the command below DEV must be the device name such as '/dev/sda1' or an NFS network name such as 'some_nfs:/home/user'. **Do not** use the mount point directory name for NFS mounts because this will cause `fuser` to hang. Again, for NFS, use only the **nfs_server:/path** name.

```
fuser -v -m [DEV]
```

Note that this does not work if the process has chrooted into a directory in the mounted filesystem. Neither **lsof** nor **fuser** will display the chrooted directory name or the name of the mounted filesystem. You can search for the chroom mount point through the symlinks found at **/proc/*/root**. See Disk_mounting#chrooted_processes_cause_.22device_is_busy.22_error_during_umount.

```
for procpid in /proc/*/root; do
    linktarget=$(readlink ${procpid})
    if [ "${linktarget}" != "/" ]; then
        echo "${procpid} chrooted to ${linktarget}"
    fi
done
```

You can force an NFS share to unmount by using the **lazy** option with `umount`. This may cause the stuck process as well as other processes to segfault as mem-mapped files and the like suddenly disappear. Other weird things can happen as this does not actually force the connection to close for any processes that were connected. For example, shells may still work, but if you `cd` into other directories you may end up with a meaningless working directory yet still able to `ls` files.

```
umount -l [MOUNT_POINT_OR_DEV]
```

The system can't reboot

Sometimes the only thing to do is reboot, but even `reboot` and `halt` will first try to sync filesystems by default and they will end up stuck. This sounds like a Catch-22, but the fix is simple by specifying the

options '-n' to not sync any mounted filesystems and '-f' for force a reboot without calling `shutdown`.

```
reboot -n -f
```

Why is a process wedged?

```
cat /dev/random >/dev/null &
PID=$!
CMDLINE="!-2"
CMD=${CMDLINE%% *}
WCHAN=$(cat /proc/${PID}/wchan)
echo "command: ${CMD}, pid: ${PID}, wchan: ${WCHAN}"
strace -p ${PID}
gdb ${CMD} ${PID}
(gdb) disassemble
Dump of assembler code for function __kernel_vsyscall:
0xb7f6b420 <__kernel_vsyscall+0>:       push   %ecx
0xb7f6b421 <__kernel_vsyscall+1>:       push   %edx
0xb7f6b422 <__kernel_vsyscall+2>:       push   %ebp
0xb7f6b423 <__kernel_vsyscall+3>:       mov    %esp,%ebp
0xb7f6b425 <__kernel_vsyscall+5>:       sysenter
0xb7f6b427 <__kernel_vsyscall+7>:       nop
0xb7f6b428 <__kernel_vsyscall+8>:       nop
0xb7f6b429 <__kernel_vsyscall+9>:       nop
0xb7f6b42a <__kernel_vsyscall+10>:      nop
0xb7f6b42b <__kernel_vsyscall+11>:      nop
0xb7f6b42c <__kernel_vsyscall+12>:      nop
0xb7f6b42d <__kernel_vsyscall+13>:      nop
0xb7f6b42e <__kernel_vsyscall+14>:      jmp    0xb7f6b423 <__kernel_vsyscall+3>
0xb7f6b430 <__kernel_vsyscall+16>:      pop    %ebp
0xb7f6b431 <__kernel_vsyscall+17>:      pop    %edx
0xb7f6b432 <__kernel_vsyscall+18>:      pop    %ecx
0xb7f6b433 <__kernel_vsyscall+19>:      ret
End of assembler dump.
```

Preventing NFS stupidity

Mounting an NFS filesystem with the **soft** option will help prevent stuck processes when a network connection is lost.

```
showmount -e remote_nfs_server
mount remote_ls
mount -o soft nfs_server:/path /media/mount_point
```

Retrieved from "http://www.noah.org/wiki/Kill_-9_does_not_work"
