**SwitchDoc Labs**
*Raspberry Pi, Arduino tutorials, scripts and projects*

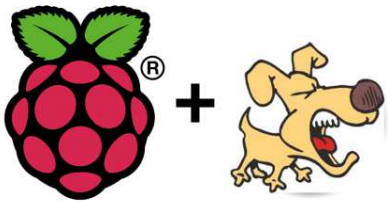


### Reliable Projects 2: Using the Internal WatchDog Timer for the Raspberry Pi

Posted on November 20, 2014 by John Shovic

# Reliable Projects 2: Using the Internal WatchDog Timer for the Raspberry Pi

**Summary:** In Part 2 of this series I look at how to set up the Raspberry Pi *internal* watchdog timer. I also talk about the issues with the Raspberry Pi internal WatchDog and explain why an *external* WatchDog Timer, such as the SwitchDoc Labs Dual WatchDog Timer is a better choice in many, *but not all*, systems.



Part 1 – Introduction

Part 3 – The Arduino Internal WatchDog Timer

Part 4 – Internal Versus External WatchDog Timers

Part 5 – Setting up an External WatchDog Timer for Raspberry Pi/Arduino Systems

## Setting up the Raspberry Pi Internal WatchDog Timer

First of all a definition. *Wto* is defined as the maximum amount of time the WatchDog timer can count before it needs to be reset (in other words, when it will reboot the computer if the computer goes away.

The BCM2835 System on a Chip that powers the Raspberry Pi has a WDT on board. It has 20 bits and counts down every 16us for a *Wto* of 16 seconds. This means you have to write to the internal WTD earlier than every 16 seconds, or the WDT will fire.

Run the following command to load the internal WatchDog kernel module:

```
$ sudo modprobe bcm2708_wdog
```

For Raspbian, to load the module the next time the system boots, add a line to your /etc/modules file with "bcm2708_wdog".

```
$ echo "bcm2708_wdog" | sudo tee -a /etc/modules
```

Now run “lsmod” and look for the line in below:

```
bcm2708_wdog 3537 0
```

This verifies that the WatchDog module was loaded successfully. Now modify /etc/modules and add bcm2708_wdog to load the module on boot by running the following command:

```
sudo echo bcm2708_wdog >> /etc/modules
```

Then we use the watchdog(8) daemon to pat the dog:

```
sudo apt-get install watchdog chkconfig
sudo chkconfig watchdog on
sudo /etc/init.d/watchdog start
```

The watchdog(8) daemon requires some simple configuration on the Raspberry Pi. Modify */etc/watchdog.conf* to contain only:

```
watchdog-device = /dev/watchdog
watchdog-timeout = 14
realtime = yes
priority = 1
```

To set the interval to pat the dog every four seconds:

```
interval = 4
```

Finally:

```
sudo /etc/init.d/watchdog restart
```

This sets up the internal Raspberry Pi WatchDog.

## Testing the Internal Raspberry Pi WatchDog

To test the Internal WatchDog, set it up as above.

Next, edit a file called forkbomb.sh and put the following commands in the file:

```
#!/bin/bash

swapoff -a
:(){ :|:& };:
```

Execute the forkbomb.sh file:

```
sudo sh -x forkbomb.sh
```

Your Raspberry Pi will eventually reboot.

A fork bomb works like this: The function is invoked twice and the pipeline is backgrounded; each successive new call on the processes spawns even more calls to ":" (the function). This leads rapidly to an explosive use of system resources, slowing response to a halt and killing the ability of the Raspberry Pi to pat the watchdog timer. If you don't turn the swap drive off then the fork bomb has to fill that also, which makes the bomb much, much slower.

## Problems with the Internal Raspberry Pi WatchDog


However, there are a number of problems with the internal WatchDog.

1. The internal WatchDog does NOT power cycle the system. It reboots the Raspberry Pi. This means that it does not restart in all conditions. Especially in low power / brownout conditions often experienced with Solar Powered Systems.
2. If the Raspberry Pi takes longer to bootup than 14 seconds (or to whatever value you set *Two*), the WatchDog can fire which puts the Raspberry Pi in an infinite bootup sequence. This can happen. I have done it.
3. If you halt the Raspberry Pi (sudo shutdown -h now), the Raspberry Pi will never reboot. If your program does this by accident, you are finished.
4. I have found the internal WatchDog to be unreliable. I never could track it down, but it feels like some kind of conflict between user space and kernel space.
5. There are some situations where the Pi will be unresponsive, but the heartbeat may still occur. High load situations for example.
6. The internal WatchDog is not completely independent of the Raspberry Pi. Theoretically, this should not matter, but the Raspberry Pi running Linux is a complex system.

These are a few of the issues that can be resolved with an external WatchDog. However, it doesn't mean the

internal WatchDog is useless, just limited.

## Next up:

Part 3 – The Arduino Internal WatchDog Timer



**About John Shovic**

Dr. John C. Shovic is currently Managing Partner of SwitchDoc Labs, LLC and Chief Technical Officer of InstiComm, LLC, a company specializing in mobile medical software solutions for health practitioners. He has worked in industry for over thirty years and has founded multiple companies: Advance Hardware Architectures, TriGeo Network Security, Blue Water Technologies, InstiComm, LLC, and bankCDA. He has also served as a Professor of Computer Science at Eastern Washington University, Washington State University and the University of Idaho. Dr. Shovic has given over 55 invited talks and has published over 35 papers on a variety of topics on HIPAA, GLB, computer security, computer forensics, embedded systems and others.

View all posts by John Shovic →

This entry was posted in Project Curacao, Raspberry Pi, Software, SwitchDoc Dual WatchDog Timer. Bookmark the permalink.

## One Response to *Reliable Projects 2: Using the Internal WatchDog Timer for the Raspberry Pi*

Pingback: *Reliable Projects 1: WatchDog Timers for Raspberry Pi and Arduinos - SwitchDoc Labs*