Niksula Watchdog
S-88.150 Tietokonetekniikan Työt

Hannu Aronsson <haa@cs.hut.fi>
Heikki Suonsivu <hsu@cs.hut.fi>

December 8, 1993

Contents

Abstract

This paper documents the Niksula Watchdog, a device which reboots a computer when no disk activity has not been detected for a certain time.

1 The Niksula Watchdog

Helsinki University of Technology has a number of Sony workstations, for various uses, but mostly for computer science students' work. They are also used as X terminals.

We have had increasing amount of problems with the stability of these systems. The nasty thing is that the servers die and will not boot automatically. As they are located in the Computing Center's computer room, with limited access, they won't get manually booted during normal hacker working hours (7pm till 7am), and thus lots of people can't do their work.

As we haven't been able to get the souce code for these systems from Sony, we had to implement a kludge to work around this problem. A software fix would have involved replacing the operating system with a working one.

Thus, we implemented hardware, which will detect a lockup and reset the system, if necessary.

The idea of the Niksula Watchdog is simple. We generate a clock, which drives a decimal counter, which in turn drives the reset unless the counter is reset by a disk access. Basically this is simple, though some compilications arose from connecting the Niksula Watchdog into the computer. The front panel LED outputs could not be used directly, as the voltages were not suitable.

These machines crash so that the disk light stays off (possible SCSI bus jam?), so we can use that as an indicator for our watchdog.

A simple piece of software is used to access the disk often enough to avoid unintentional reset.

This device is not Sony hardware specific, though it requires finding out correct connection points for inputs, and possibly somewhat different reset logic.

2 The Niksula Watchdog Hardware Manual

This section briefly describes the hardware implementation.

2.1 The Sony 18xx System

The Sony NWS 18xx server has a somewhat unusual reset arrangement: The reset button will shut the power down completely, and another switch is used to restart the system. Simultaneously pressing both switches produces a real reset.

There are two boards in the Sony 18xx server which need some soldering (figure 1). Not exactly knowing their names, we named them **the bus logic board** and **the switch board**, the first contaning various bus logic, including reset and power logic, and the latter containing mostly passive components like switches, the reset button, a DIP switch and similar components.

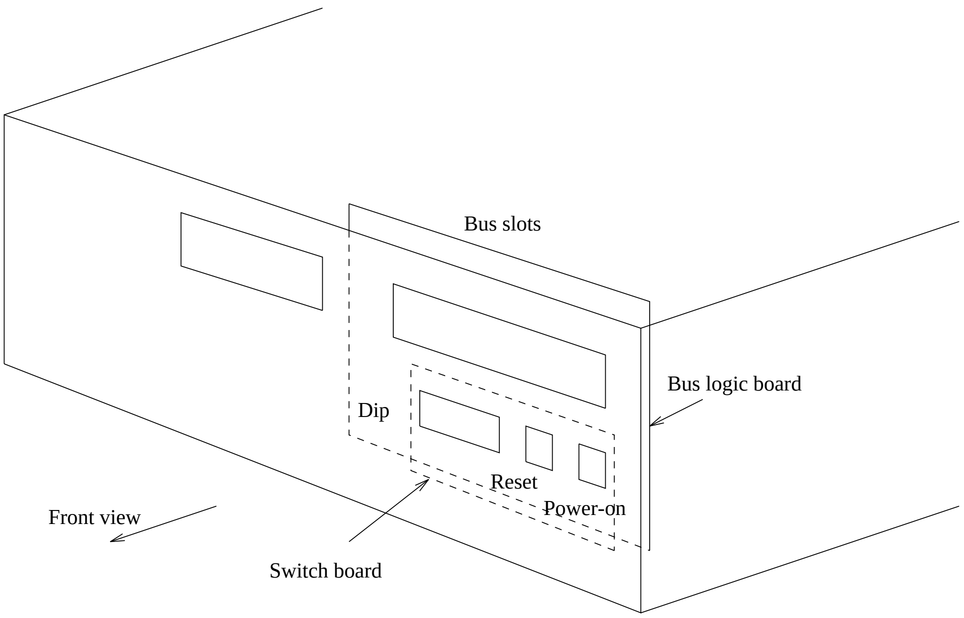


Figure 1: The sony server, front view.

The watchdog was first tested on a Sony NWS 15xx machine, on which all of this functionality is on the switch board with easy access.

2.2 The Watchdog

2.2.1 The Power Supply and The Ground

The power and ground were picked up from suitable points on the bus logic board.

2.2.2 The Clock Generator

The clock generator is an RC circuit with signal cleanup done with schmitt-trigger inverters. With 3.8MΩ resistor and 1μF condensator. This results a clock tick of approximately 1/3 Hz (3 second clock pulses).

2.2.3 The Disk Activity Signal

The TTL disk activity signal was picked up from bus logic board and routed directly to the reset pin of the 74HC4017 counter.

2.2.4 The Counter

The counter used is 74HC4017 10-output decimal counter. Enable is directly connected to power/ground, clock to the clock generator, and outputs to the status LEDs and controls. Carry is unused, because it is active for longer than one clock cycle (states 5-9).

2.2.5 The Connections to The Reset and Power-On

Normally, the reset switch on a 18xx grounds the reset signal, which has a pull-up resistor connection to 5V power.

The reset signal we used is an inverted bit 9 output from the counter. This will drive the reset signal down if the counter ever gets that far, thus turning the machine off. Reset line is routed via an unused configuration DIP switch on the front panel.

To make the system turn itself up the Power-On button is routed around directly, thus always connecting the line. There is no special power-on delay mechanism, as this is already provided by the system. The signal is routed through an another unused configuration DIP switch in the switch board to allow watchdog power off.

The Niksula Watchdog configuration is described in table 1.

Reset DIP	Power-on DIP	
off	off	The Watchdog completely disconnected.
off	on	Automatic power-on after a power failure
on	off	Watchdog power-off (fail-stop)
on	on	Watchdog reset (default mode)

Table 1: Watchdog configuration options.

2.2.6 The Status Display

To make the Niksula Watchdog more appealing, it is equipped with lots of *blinkenlights*. Clock and reset have their own LEDs, and to make the Niksula Watchdog more exciting, each output of the counter, otherwise unused, is connected to a LED. The first stages are green, then yellow and finally the red reset indicator. [1]

3 The Software

The software required is simple. It opens the raw disk device, and then keeps reading a block from it with a certain interval, short enough to avoid unintentional reset.

There are some special requirements for the software:

- It has to be started as early as possible to avoid anything started up before it from locking up and thus causing the system reboot again. It is started in `/etc/rc` before anything else, thus right after `init(8)` and before the root disk checks etc (figure 2).
- It must not use anything which could cause it to die when everything is fine, thus it must not use the network, nfs disks, or blocking system calls other than the read from the disk device.
- The software must be installed on the root filesystem. If it were on an nfs-mounted disk, it could hang up in page-in, and all other disks are mounted afterwards.

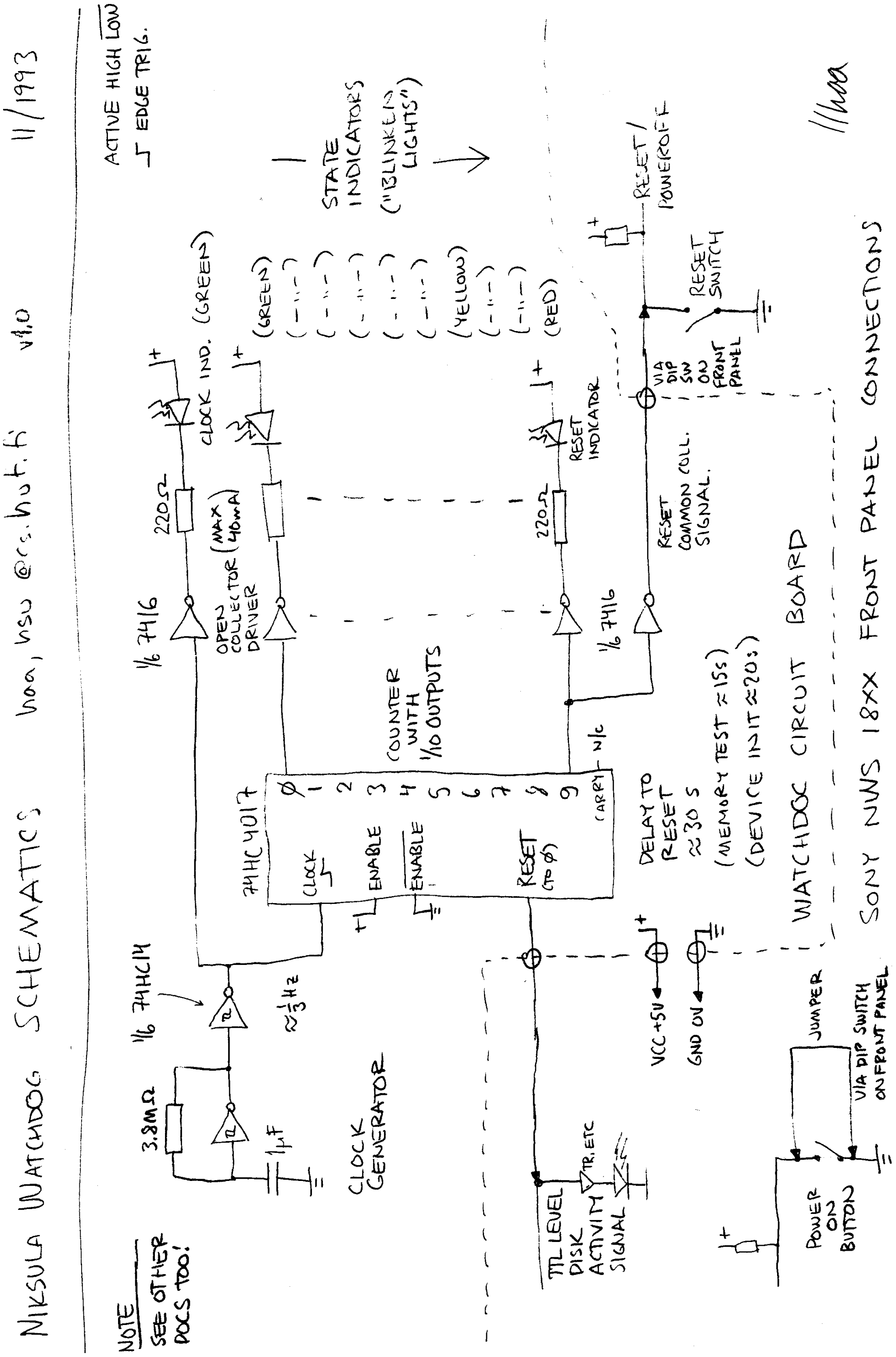
The listing of the program is presented in appendix B.

```
# Start watchdog stuff
if [ -r /etc/feedwatchdog ]
then
        echo 'Starting /etc/feedwatchdog...' >/dev/console
        /etc/feedwatchdog
fi
```

Figure 2: An excerpt from the beginning of the `/etc/rc`.

[1] This makes the Niksula Watchdog particularly suitable as a gambling device: "I bet it will boot if I run this program".

A The Schematics



B The Program Listing

```
/*
 * feedwatchdog -- Feed the watchdog timer, keep it happy so we won't
 *                 be bitten...  Unless we fall asleep ;-)
 *
 * Exercise for the computer technology laboratory course
 *
 * Ti N   33849M   Hannu.Aronsson@cs.hut.fi
 * S  N   30852T   Heikki.Suonsivu@cs.hut.fi
 *
 * The watchdog timer we built is reset from the DISK I/O light on the
 * Sony NEWS workstation.
 *
 * The delay until RESET is asserted is approximately 30 seconds
 * of inactivity. This is long enough to pass memory tests and other
 * boot-time delays.
 *
 * This program should be started in rc around the fsck disk checks,
 * so it's running always, even in single user or fsck-failure mode.
 *
 * This program is written for maximal reliability and minimal depen-
 * dencies, not clarity, unfortunately. It should not stop/jam in any
 * non-fatal case.
 *
 * Something like this should be added to the head of /etc/rc (or
 * whatever is *first* run by init. Make sure feedwatchdog is
 * physically on the root filesystem.
 *
 *      # Start watchdog stuff on modesty-blaise.cs.hut.fi
 *      if [ -r /etc/feedwatchdog ]
 *      then
 *              echo 'Starting /etc/feedwatchdog...' >/dev/console
 *              /etc/feedwatchdog
 *      fi
 *
 */

#include <sys/fcntl.h>          /*   These are quite safe to include */
#include <signal.h>             /*   Unlike stdio etc. */

#define DISK "/dev/rsd00a"      /* We read from this disk device */
                                /*   Raw device recommended */
                                /*   Can't have any kind of buffering */

#define DELAY 19                /* Seconds to delay between reads */
                                /*   Not too long, not too short... */
                                /*   18 seconds is enough to make the lights */
```

```
                                /*   reach the first yellow LED, which makes */
                                /*   for lots of panic and amusement for */
                                /*   people watching the device ;-) */
main()
{                               /* Variables used: */
  int i, max;                   /*   Loop counters */
  int disk = -1;                /*   "Feeding" disk file descriptor */
  char buffer;                  /*   I/O buffer for Feeding */

  max = 30;                     /* Ignore all signals */
  for (i=0; i<max; i++)         /*   This guessed from signal.h */
    signal(i, SIG_IGN);         /*   Don't care if it succeeds or not */

                                /* Close all files */
  max = getdtablesize();        /*   How many files can we have? */
  for (i=0; i<max; i++)
    close(i);                   /*   Don't care about "errors" */

  switch(fork()) {              /* Switch to background */
  case -1:                      /*   Oops, fork failed */
    break;                      /*   Gotta do the work myself ;-( */

  case 0:                       /*   This is the child */
    break;                      /*   Who will do all the hard work */

  default:                      /*   This is the parent */
    exit(0);                    /*   Bred successfully, OK to exit */
    /*NOTREACHED*/
    break;
  }

  disk = open(DISK, O_RDONLY);  /* Open disk device */
  if (disk < 0)                 /*   This must succeed, of course */
    exit(1);                    /*   May fail e.g. if we are not root ;-) */

  nice(-39);                    /* We want high priority */
                                /*   Make it to 0 from anything 0..39 */
                                /*   Probably this does not matter so much */

  while(1) {                    /* Main loop */
    read(disk, &buffer, 1);     /*   Read one byte from there */
    sleep(DELAY);               /*   Wait until next read */
  }
  /*NOTREACHED*/
  exit(1);                      /* This should never be reached */
}

/* End */
```