Real-Time Linux driving a Spectrometer

Peter Teuben, Andrew Harris, Kate Isaak[1], James Morgan[2]

Astronomy Department, University of Maryland, College Park, MD

Jonas Zmuidzinas

California Institute of Technology, Pasadena, CA

Abstract.
We have been upgrading the software front-end to WASP (Wideband Autocorrelating SPectrometer, operating from 800 to 4000 MHz) from Lynx-OS to Real-Time Linux and will discuss Real-Time software aspects. We are using a standard parallel "PIO" card to receive data from the spectrometer and standard GPIB interfaces that drive a frequency synthesizer for bandpass calibration.

WASP and its front-end computer are fully mobile and can be taken to any suitable radio telescope. The front-end is typically connected via Ethernet to the actual observing computer at the telescope. We will also discuss a typical observing setup in such a loosely coupled environment.

1. Experimental setup

A ruggedized Pentium-133 was acquired (replacing the previous 486-33 Lynx-OS-based setup) to which a standard NI GPIB card and Metrabyte PIO-12 DAQ card were added. RedHat 5.1 Linux was installed and modularized device drivers for the PIO-12 and GPIB cards were obtained from the Linux Lab project.

The WASP (Harris et al. 1998) signal feeds directly into the PIO-12 and is driven at 4.603 kHz by WASP. WASP is also controlled via the PIO-12 card as the i8255 based PIO-12 card contains 3 bytes, configurable in many I/O modes. We use PA to read, PB to write, PC(low nibble) to write and PC(high nibble) to read.

The GPIB card is connected to an HP8648C synthesizer for bandpass calibration, which in turn drives WASP in calibration mode. (see also Figure 1).

---

[1]now at: MRAO, Cambridge, UK

[2]now at: Edge Technologies, Inc., Laurel, MD

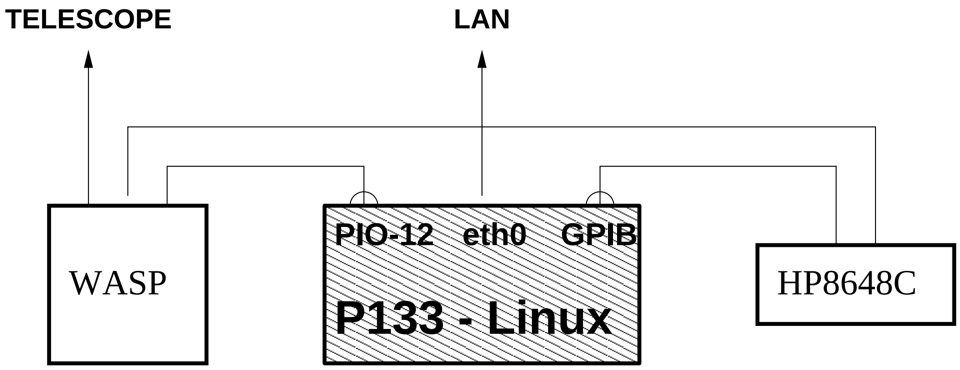

Figure 1. Schematic setup. The Pentium-133 computer is connected to a LAN via Ethernet, through which telescope control will occur. The GPIB card is connected to an HP8648C synthesizer for calibration and the PIO-12 data acquisition card is connected to WASP

2. Real-Time Alternatives (cheating)

Several cheap alternatives to increase performance and/or responsiveness are available on Linux:

`nice` Re-prioritizing of the CPU usage that hooks into the standard scheduling algorithms the kernel uses. An obvious choice, but rarely sufficient, especially for instrument control.

`swapoff` Turning off the swap can sometimes be an effective way to avoid paging in the system and avoid dreaded latencies. Of course this does not solve task switching delays.

`irqtune` This is a simple interrupt optimizer, with the advantage that it works in user-space, without the need to recompile the kernel. For serial lines (e.g. modems) this can make a big performance difference. It is generally available for Linux.

`QNX scheduling` This needs a true patch to the Linux kernel, supplied by Adam McKee, and replaces the scheduling algorithm with one that was freely modeled after the commercially available QNX Real-Time operating system scheduling algorithms. Using a supplied program, `qsched`, users can then prepend commands that re-schedules the commands in QNX style, *e.g.*

```
qsched -t 10000 rmsobs
```

would run `rmsobs` in guaranteed timeslices of 10 seconds. This can of course (like many in these situations) hang the machine if not carefully used. We tried QNX scheduling and found it to be reliable for short observations, but lack of low level scheduling made it impossible to adopt it for complex tasks like the bandpass calibration.

3. Real-Time Linux

Victor Yodaiken and collaborators at New Mexico Tech (Socorro) have recently made patches available to the Linux kernel which turns it into a good Real-

Time operating system. This works by a user supplied kernel module, which takes over control of the computer. Only when this user module has determined there is time left will the kernel get time allotted and can do its usual things. Communications between these modules and user program occurs via special RT FIFO buffers. As usual in this business, programmers need to take special care not to produce situations that can lock up the computer.

We wrote an interrupt driven Real-Time module to guarantee that no data will be lost. Data are then read until a full "lag-readout" has been accumulated (192 bytes at a rate of 4.6kHz) and sent to a user program via a Real-Time FIFO. This turned out to work very reliably. The periodicity and nature of this signal also made it possible to accurately determine if data were missed.

4. Observing

Since WASP was designed to be mobile, a typical observing mode might consist of shipping the WASP box (the current prototype is a 1 cubic meter) together with the HP synthesizer and Linux box to a remote observatory where the IF mixed signal can be spectrum analyzed.

The Linux box will only be controlling the spectrometer itself and is normally part of a local network. The observing telescope computer will drive the telescope and send synchronized commands (in the form of remote shell scripts) to the Linux box to perform integrations, calibrate the signal, set the chopper, etc.

5. Future

WASP2 is being planned, taking data at 50kHz instead of the current 5kHz rate. Our current P133 should be able to handle around 30-50kHz, a Pentium-II-266 should be able to handle a 150kHz interrupt rate. *** need more flesh here ***

References

RT Linux (`http://www.rtlinux.org`) - V. Yodaiken *et al.* (Socorro, NM)

Linux Lab Project (`http://www.llp.fu-berlin.de/`): large repository of device drivers for DAQ cards, as well as the Linux GPIB library.

McKee, Adam: `ftp://sunsite.unc.edu/pub/Linux/kernel/patches/misc/QNXsched-1.21.tar.gz`: "*QNX-style Scheduling for Linux*"

Harris, A. I., Isaak, K. G., and Zmuidzinas, J. Proc. SPIE vol. 3357, pp 384-394 (1998).

WASP homepage: `http://www.astro.umd.edu/~ teuben/wasp/`.

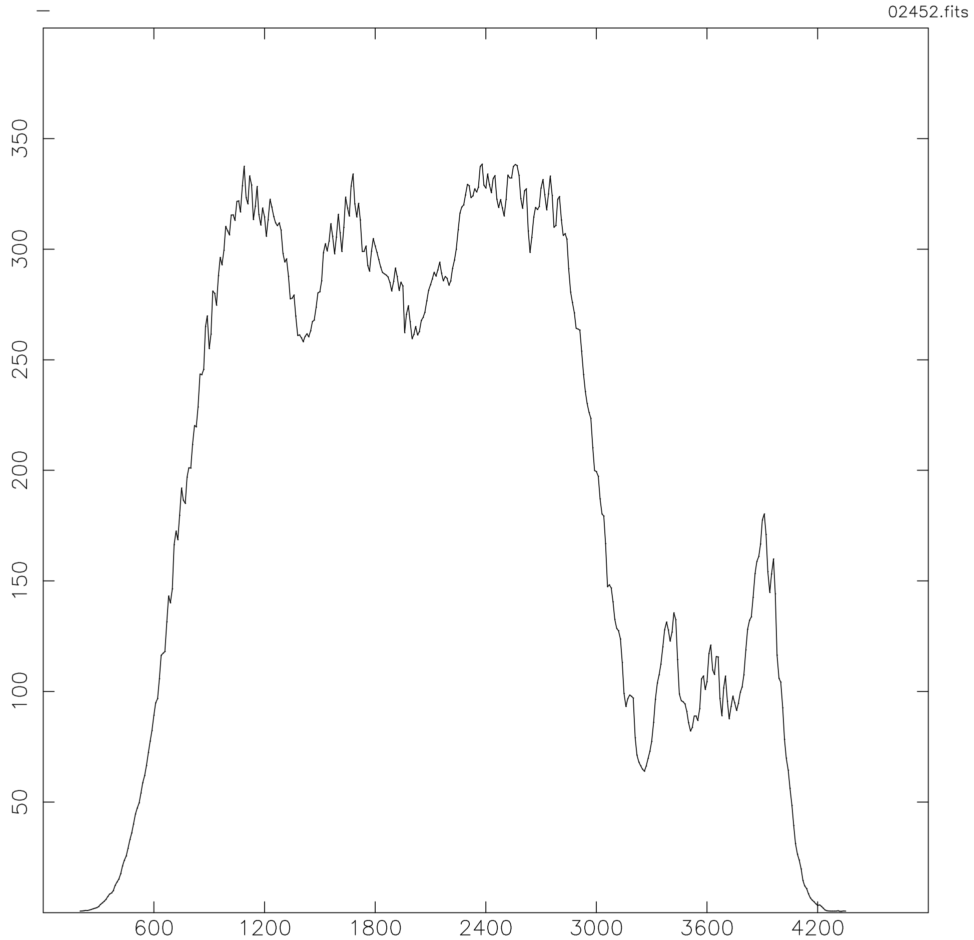

Figure 2. Bandpass filter shape (actually a one-dimensional projection) as determined with the current experimental setup. Horizontal units are MHz, vertical units arbitrary. ** get rid of labeling at top **