Astronomical Data Analysis Software and Systems IX
ASP Conference Series, Vol. 216, 2000
N. Manset, C. Veillet, and D. Crabtree, eds.

Real Time Linux and the WTTM Project

P. N. Daly

National Optical Astronomy Observatories, 950 N. Cherry Avenue, Tucson, AZ 85719, U.S.A.

Abstract. The WIYN Tip-Tilt Module (WTTM) is a low-order, low-cost adaptive optics solution to improve delivered image quality within a 4 × 4 arc minute science field across the B–H bands and optimized for the V–I bands. The system is greatly simplified by the use of a quad-cell APD-based error sensor and there is no deformable mirror. The software to handle this novel module and instrument will be written under real time Linux with *LabVIEW for Linux* acting as a supervisory GUI and post-analysis suite. This combination is proving highly effective in other projects and should enable us to concentrate on the correction algorithms: fixed frequency, fixed signal to noise, predictive and 'maximum bandwidth'. This paper examines the suitability of real time Linux for serious scientific instrumentation in hard real time with frequencies >8kHz.

1. Introduction

The adaptive correction of atmospherically induced image distortion is a relatively new discipline in astronomy founded in the work of Babcock (1953). An extensive review of the state of the art has been written by Tyson (1998). The WIYN telescope consortium has recently agreed to the development of a tip-tilt corrector and imager for the Nasmyth focus. The conceptual design of the WIYN Tip-Tilt Module (WTTM) is shown in Figure 1 and the PDR held in March 1999 found the design capable of improving delivered image quality and better science. First light on the telescope is scheduled for the last quarter of 2001.

This design uses 4 avalanche photo diode detectors (APD) arranged as a simple quad-cell—with each detector counting C_1, C_2, C_3, C_4 counts clockwise from top left say—as an error sensor. Since the beam splitter component has a deliberately introduced astigmatism, the automated detection of focus (ϵ_z) as well as tip-tilt demands (ϵ_x, ϵ_y) can be computed from:

$$\epsilon_x = [C_2 + C_3 - C_1 - C_4]/\Sigma C_i \quad (1)$$
$$\epsilon_y = [C_1 + C_2 - C_3 - C_4]/\Sigma C_i \quad (2)$$
$$\epsilon_z = [C_1 + C_3 - C_2 - C_4]/\Sigma C_i \quad (3)$$

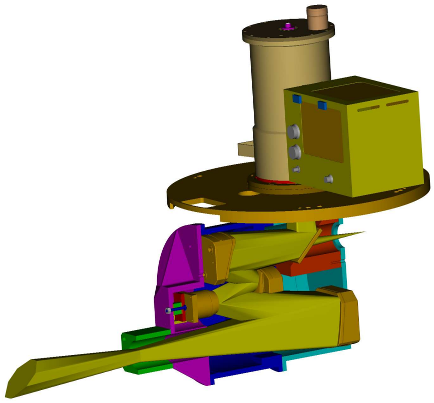

Figure 1. The WIYN Tip-Tilt Module.

2. Software Issues Overview

Analysis of the physical properties of the tip-tilt mirror have resulted in a maximum desirable oscillation frequency of ~1.2kHz. To achieve this we shall sample the APD modules at a rate up to an order of magnitude higher and apply the appropriate corrections via a closed loop servomechanism. The data acquisition, error calculations and communications with the *Physik Instrumente* tip-tilt module will be performed in real time using real time Linux—a powerful new set of extensions to the already phenomenal capabilities within the open source Linux operating system.

We have also chosen *LabVIEW for Linux* as a supervisory GUI for this project as the well engineered, rapid prototyping environment offers a rich feature set—particularly in post detection analysis—at modest cost (when compared to writing an in-house system under Java or Qt, for example). We were concerned, however, that LabVIEW is a memory intensive application and that real time Linux running at too high a rate would freeze the system and hang the interface. For these reasons we plan to utilize a dual Pentium III architecture. One CPU will be allocated for real time tasks and the other for the standard Linux kernel thus keeping both systems responsive. The 256MB of main memory will be partitioned with one half reserved for real time Linux to communicate bulk data to LabVIEW for engineering analysis (Fourier or Lomb transforms and so on). The question is: can real time Linux perform at such frequencies?

3. Real Time Linux

Real time Linux was developed at New Mexico Institute of Technology by Barabanov (1997) and Yodaiken. The standard Linux kernel uses disabling interrupts as a means of synchronization and such promiscuous use inflicts a non-deterministic interrupt despatch latency making hard real time programming problematic at best. Yodaiken and co-workers have created a small, tightly coded layer of emulation software with a real time kernel that sits between the Linux kernel and the interrupt hardware controller. Thus all occurrences of the x86 instructions `cli`, `sti` and `iret` are replaced with emulating macros S_CLI, S_STI and S_IRET and, in effect, convert hard interrupts into soft interrupts. The real time executive treats the Linux kernel as the lowest priority task thus ensuring that real time tasks retain access to the CPU first i.e., the standard Linux kernel is fully pre-emptive.

Tests of periodic time generation and interrupt response have been carried out by physicists at UNLP[1] using the 2.0.29 Linux kernel and an early New Mexico patch (RTL). Their data is reproduced in Table 1—see their web site for details of the experiments—and clearly show that RTL is capable of hard real time performance.

Table 1. UNLP Linux Experiments

	Standard Linux				Real Time Linux			
Load	$\overline{x}$	σ_x	x_{min}	x_{max}	$\overline{x}$	σ_x	x_{min}	x_{max}
Time Interval Generation (μs)								
None	19996	1.45	19989	20054	19999	0.89	19993	20013
Background	19999	45.61	18186	20676	19999	1.31	19993	20016
Full	20009	538.84	16804	29908	19999	2.45	19990	20016
Hard Disk	19991	391.14	16594	25054	20000	1.92	19985	20014
Network	19999	114.35	19653	22007	20001	2.53	19981	20019
Calculation	19999	16.79	19643	20405	19999	1.39	19988	20016
Interrupt Response Time (μs)								
None	21.62	1.55	20.80	48.7	2.36	0.55	2.1	15.5
Background	28.16	312.70	21.30	22103.0	2.34	0.35	2.1	10.8
Full	369.31	528.20	29.10	27546.0	3.84	1.04	2.3	19.0
Hard Disk	440.61	420.82	26.60	8292.0	3.20	0.65	2.1	19.4
Network	114.33	91.59	27.50	2593.8	4.38	1.71	2.5	25.0
Calculation	4790.10	2684.80	26.20	9391.1	2.56	0.43	2.2	14.5

4. The WTTM Software Simulator

To assess the suitability of real time Linux for the WTTM, and become familiar with the feature set on offer, a software simulator was written as shown schematically in Figure 2. As can be seen, the code has been divided into real time and timesharing systems linked by a 1MB segment of shared memory and a couple of FIFO buffers. On the real time side, the rfCode task handles incoming user commands on the wFifo that allows the other tasks to be stopped, re-configured and re-started as the user directs.

The programmable 8254 (APIC) is used to generate realistic interrupts at frequencies of 2^n where $1 < n < 15$. Upon receipt of the interrupt, the rqCode

[1] `http://www.fisica.unlp.edu.ar/rt/`

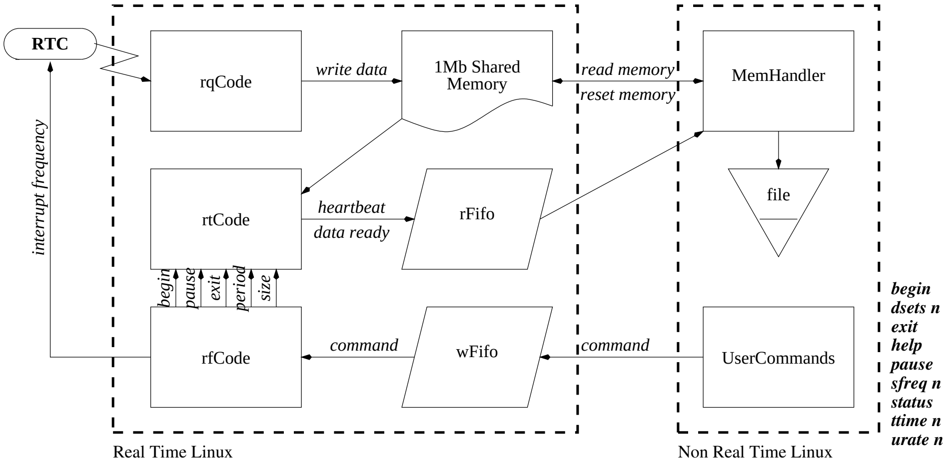

Figure 2. The WTTM Software Simulator.

interrupt service routine is invoked and random APD data is generated, centroid and focus error demands are calculated and these values and a time stamp are written to shared memory. Timing tests show that this module executes within 35 μs. This task has been run as high as 8192Hz but at such high frequencies the shared memory is rapidly depleted.

The rtCode module is a periodic task that awakens when a specified number of data points have been collected. Once invoked it sends a heartbeat to the timesharing code and checks the integrity of the memory resident data. If there is no discrepancy, a further message is put on the rFifo to let the user space memory handler collect and display the data.

On the timesharing side, the code consists of an extensible command line parser (UserCommands) and a memory management module (MemHandler). This latter module listens on the rFifo for heartbeats and 'data ready' messages from the rtCode task. Upon receipt of a 'data ready' command, the shared memory is read and reset, and a fast Fourier transform is performed on the centroid errors. The raw data and the real and imaginary parts of the Fourier components are then displayed using PGPLOT. All memory data is written to an ASCII text file. In this way, many engineering aspects of the project are simulated.

The WTTM software simulator—although not a robust or complete coding of the project—clearly shows that real time Linux is capable, on a quite modest machine (486DX/66 with 32MB), of meeting the critical restraints of the system.

References

Babcock, H. W. 1953, PASP, 65, 229

Barabanov M. 1997, A Linux-based Real-Time Operating System, M.Sc. Thesis, New Mexico Institute of Mining and Technology

Tyson, R. K. 1998, Principles of Adaptive Optics, Academic Press