MIT Record ID: 1537404, Pravina Samaratunga's Maker Project.

Compiling a Real-time Operating System with Controller Area Network, and USB Support by Modifying the Linux Kernel.

1. Motivation

The problem I tackled in this maker project is totally independent of my summer software engineering internship work at Barrett Technology, but was motivated by that experience.

The motivation for the maker project is as follows: Barrett's robotic hand had fine grain movements of less than a millimeter precision. However, one of the problematic issues with the hand is, if the computer that is used to control the hand is also used to print documents or do CPU or disk intensive tasks, the hand movements were not precise enough. This is because the computer system's slices of time dedicated to handling the robot hand is not the same as the physical time in the real world.

In this situation, the robot hand cannot be guaranteed as suitable to do things like delicate surgery, or precision machining of instrument parts, or even balancing a pencil on its pointy end. This is because such activities usually need to know the hand's location and some of its time derivatives (velocity, acceleration, jerk, and snap) at a given time precisely, without that being randomly impacted by other processes in the computer system.

My project rectifies this deficiency by assembling a near real-time operating system using open source software tools. In this system, the system clock for the critical process must try to be synchronous with physical time. The inter-task communication and resource sharing must be well managed, to avoid interrupting the critical process.

I did an Internet search to find out if "real" real time computer systems exist in the world. There are a few companies that sell such products, but they are proprietary, such as Windows CE, or VxWorks, except for the unsupported RTLinux. Therefore I decided to build such an operating system using open source software descriptions available on the Internet via the Linux kernel.

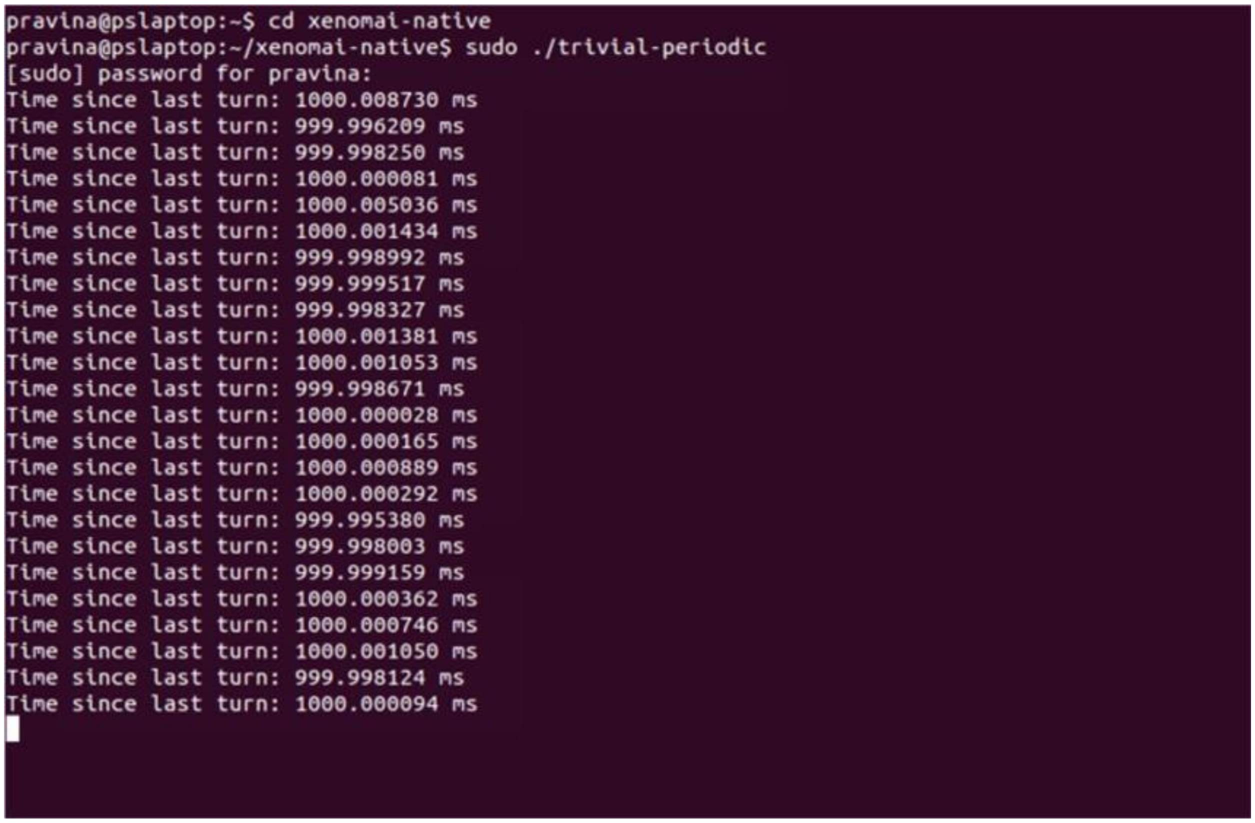
2. Procedure
   a) I downloaded the Linux 3.5.7 kernel to my Ubuntu 12.04 machine from kernel.org.
   b) I looked at the kernel level patches that were carried out by the Xenomai researchers on their website www.xenomai.org, and used the Xenomai software version 2.6.2.1 from gna.org to appropriately patch the C code in the Linux kernel based on a patch description from Georgia Tech.
   c) Using my trusty gcc compiler, I compiled the patched version of the Linux kernel code.
   d) Then I installed the modified Linux kernel in my Ubuntu machine.
   e) I then setup the "grub" boot order manually, in order for my machine to have a choice of dual booting into the compiled real time kernel.

f) I then I compiled a loadable Linux driver module for my Control Area Network (CAN) USB device from Peak Systems. I used the open source code for the driver software obtained from Peak Systems at http://www.peak-system.com/fileadmin/media/linux/ for this purpose. Before this step, I had to recompile the kernel to enable a CAN kernel hook for the Peak Systems CAN device and a virtual CAN (VCAN) kernel hook to test the system.

g) I included a patch for providing near real time USB support from Volkswagon Research (www.wverl.com).

3. Results

The operating system I compiled and assembled from existing source code worked near real time. Figure 1 illustrates a timing diagram I obtained for a looped 1 second task being performed by the system via two VCAN ports.



```
pravina@pslaptop:~$ cd xenomai-native
pravina@pslaptop:~/xenomai-native$ sudo ./trivial-periodic
[sudo] password for pravina:
Time since last turn: 1000.008730 ms
Time since last turn: 999.996209 ms
Time since last turn: 999.998250 ms
Time since last turn: 1000.000081 ms
Time since last turn: 1000.005036 ms
Time since last turn: 1000.001434 ms
Time since last turn: 999.998992 ms
Time since last turn: 999.999517 ms
Time since last turn: 999.998327 ms
Time since last turn: 1000.001381 ms
Time since last turn: 1000.001053 ms
Time since last turn: 999.998671 ms
Time since last turn: 1000.000028 ms
Time since last turn: 1000.000165 ms
Time since last turn: 1000.000889 ms
Time since last turn: 1000.000292 ms
Time since last turn: 999.995380 ms
Time since last turn: 999.998003 ms
Time since last turn: 999.999159 ms
Time since last turn: 1000.000362 ms
Time since last turn: 1000.000746 ms
Time since last turn: 1000.001050 ms
Time since last turn: 999.998124 ms
Time since last turn: 1000.000094 ms
```

Figure 1.

As can be observed from the above data, the timing error of a looped task for CAN messages are less than 0.01% always, and is less than 0.001% most of the time.

4. Problems Encountered

While the operating system I assembled worked near real time, it could still not be used for true real time control of a robot, because I had only a laptop running Ubuntu, and the USB devices do not have their own Interrupt Request (IRQ) number to create interrupts from physical devices. An IRQ is needed to get full use of the Xenomai patch for the Linux kernel. Had I used a CAN PCI device on a desktop computer capable of handling IRQs, this situation could have been avoided.

5. Future Objectives

One of the things I have noticed in the robots I have seen and read about, is that most high end robots are either controlled by an on-board computer consisting of a 104 style motherboard, or an external desktop or a server computer, but never by a standard laptop.

This may be due to USB devices being not designed to have their own IRQs, and thus being unable to operate in a true real time fashion. To date as far as I am aware, real time capabilities of the mainline Linux kernel via the use of the PREEMPT_RT compilation flag and the USB compilation flags without the Xenomai patch, has not been used by any robot vendor. My attempts to use both flags have resulted in a complex sequence of compilation errors.

I believe that if it is possible to make such a modified Linux operating system, there will be a huge increase in the user base for robots, due to ease of use. This way any robot can potentially be used interchangeably just by unplugging the laptop's USB port from a robot and plugging in a new robot while giving the robot CAN Bus access capability via adapters such as Peak Systems' CAN to USB adapter. I intend to investigate the technical and commercial viability of such a product, in the summer of 2014.