HAL

Contents

1 Hardware abstraction **1**

1.1 Overview . . . 1

1.2 In Operating Systems . . . 2

1.2.1 Microsoft Windows . . . 2

1.2.2 AS/400 . . . 2

1.3 References . . . 2

2 LinuxCNC **3**

2.1 Purpose . . . 3

2.2 History . . . 3

2.3 Platforms . . . 4

2.4 Design . . . 4

2.5 Configuration . . . 4

2.6 References . . . 4

2.7 External links . . . 5

2.8 Text and image sources, contributors, and licenses . . . 6

2.8.1 Text . . . 6

2.8.2 Images . . . 6

2.8.3 Content license . . . 6

Chapter 1

Hardware abstraction

"Hardware Abstraction Layer" redirects here. For the UNIX-like operating system subsystem, see HAL (software).

Hardware abstractions are sets of routines in software that emulate some platform-specific details, giving programs direct access to the hardware resources.

They often allow programmers to write device-independent, high performance applications by providing standard Operating System (OS) calls to hardware. The process of abstracting pieces of hardware is often done from the perspective of a CPU. Each type of CPU has a specific instruction set architecture or ISA. The ISA represents the primitive operations of the machine that are available for use by assembly programmers and compiler writers. One of the main functions of a compiler is to allow a programmer to write an algorithm in a high-level language without having to care about CPU-specific instructions. Then it is the job of the compiler to generate a CPU-specific executable. The same type of abstraction is made in operating systems, but OS APIs now represent the primitive operations of the machine, rather than an ISA. This allows a programmer to use OS-level operations (i.e. task creation/deletion) in their programs while still remaining portable over a variety of different platforms.

1.1 Overview

Many early computer systems did not have any form of hardware abstraction. This meant that anyone writing a program for such a system would have to know how each hardware device communicated with the rest of the system. This was a significant challenge to software developers since they then had to know how every hardware device in a system worked to ensure the software's compatibility. With hardware abstraction, rather than the program communicating directly with the hardware device, it communicates to the operating system what the device should do, which then generates a hardware-dependent instruction to the device. This meant programmers didn't need to know how specific devices worked, making their programs compatible with any device.

An example of this might be a "Joystick" abstraction. The joystick device, there are many physical implementations, is readable / writable through an API which many joystick-like devices might share. Most joystick-devices might report movement directions. Many joystick-devices might have sensitivity-settings that can be configured by an outside application. A Joystick abstraction hides details (e.g., register formats, I2C address) of the hardware so that a programmer using the abstracted API needn't understand the details of the device's physical interface. This also allows code reuse since the same code can process standardized messages from any kind of implementation which supplies the "joystick" abstraction. A "nudge forward" can be from a potentiometer or from a capacitive touch sensor that recognises "swipe" gestures, as long as they both provide a signal related to "movement".

As physical limitations (e.g. resolution of sensor, temporal update frequency) may vary with hardware, an API can do little to hide that, other than by assuming a "least common denominator" model. Thus, certain deep architectural decisions from the implementation may become relevant to users of a particular instantiation of an abstraction.

A good metaphor is the abstraction of transportation. Both bicycling and driving a car are transportation. They both have commonalities (e.g., you must steer) and physical differences (e.g., use of feet). One can always specify the abstraction "drive to" and let the implementor decide whether bicycling or driving a car is best. The "wheeled terrestrial transport" function is abstracted and the details of "how to drive" are encapsulated.

Examples of "abstractions" on a PC include video input, printers, audio input and output, block devices (e.g. hard disk drives or USB flash drive), etc.

In certain computer science domains, such as Operating Systems or Embedded Systems, the abstractions have slightly different appearances (for instance, OSes tend to have more standardized interfaces), but the concept of abstraction and encapsulation of complexity are common, and deep.

Hardware abstraction layers are of an even lower level in computer languages than application programming inter-

faces (API) because they interact directly with hardware instead of a system kernel, therefore HALs require less processing time than APIs. Higher level languages often use HALs and APIs to communicate with lower level components.

1.2 In Operating Systems

A **hardware abstraction layer** (**HAL**) is an abstraction layer, implemented in software, between the physical hardware of a computer and the software that runs on that computer. Its function is to hide differences in hardware from most of the operating system kernel, so that most of the kernel-mode code does not need to be changed to run on systems with different hardware. On a PC, HAL can basically be considered to be the driver for the motherboard and allows instructions from higher level computer languages to communicate with lower level components, but prevents direct access to the hardware.

BSD, Mac OS X, Linux, CP/M, DOS, Solaris, and some other portable operating systems also have a HAL, even if it is not explicitly designated as such. Some operating systems, such as Linux, have the ability to insert one while running, like Adeos. The NetBSD operating system is widely known as having a clean hardware abstraction layer which allows it to be highly portable. As part of this system are uvm(9)/pmap(9), bus_space(9), bus_dma(9) and other subsystems. Popular buses which are used on more than one architecture are also abstracted, such as ISA, EISA, PCI, PCI-E, etc., allowing drivers to also be highly portable with a minimum of code modification.

Operating systems having a defined HAL are easily portable across different hardware. This is especially important for embedded systems that run on dozens of different platforms.

1.2.1 Microsoft Windows

The Windows NT operating system has a HAL in the kernel space between hardware and the Windows NT executive services that are contained in the file NTOSKRNL.EXE.[1][2] This allows portability of the Windows NT kernel-mode code to a variety of processors, with different memory management unit architectures, and a variety of systems with different I/O bus architectures; most of that code runs without change on those systems, when compiled for the instruction set applicable to those systems. For example, the SGI Intel x86-based workstations were not IBM PC compatible workstations, but due to the HAL, Windows NT was able to run on them.

Windows Vista and later (Windows Server 2008 and later for servers) automatically detect which hardware abstraction layer (HAL) should be used at boot time.[3]

1.2.2 AS/400

An “extreme” example of a HAL can be found in the System/38 and AS/400 architecture. Most compilers for those systems generate an abstract machine code; the Licensed Internal Code, or LIC, translates this virtual machine code into native code for the processor on which it is running and executes the resulting native code.[4] (The exceptions are compilers that generate the LIC itself; those compilers are not available outside IBM.) This was so successful that application software and operating system software above the LIC layer that were compiled on the original S/38 run without modification and without recompilation on the latest AS/400 systems, despite the fact that the underlying hardware has been changed dramatically; at least three different types of processors have been in use.[4]

1.3 References

[1] “Windows NT Hardware Abstraction Layer (HAL)". Microsoft. October 31, 2006. Retrieved 2007-08-25.

[2] Helen Custer (1993), *Inside Windows NT*, Microsoft Press

[3] Russinovich, Mark. E.; Solomon, David A.; Ionescu, Alex (2008). *Windows Internals: Including Windows Server 2008 and Windows Vista.* (5th ed.). Redmond, Wash.: Microsoft Press. p. 65. ISBN 978-0-7356-2530-3.

[4] Soltis, Frank G. (1997). *Inside the AS/400: Featuring the AS/400e Series* (2nd ed.). Loveland, Colo.: Duke Press. ISBN 978-1-882419-66-1.

Chapter 2

LinuxCNC

LinuxCNC (formerly "Enhanced Machine Controller" or "EMC2") is a free, open-source GNU/Linux software system that implements numerical control capability using general purpose computers to control CNC machines. Designed by various volunteer developers at linuxcnc.org, it is typically bundled as an ISO file with a modified version of 32-bit Ubuntu Linux which provides the required real-time kernel.

Due to the tight real-time operating system integration, a standard Ubuntu Linux desktop PC without the real-time kernel will only run the package in demo mode.

2.1 Purpose

LinuxCNC is a software system for numerical control of machines such as milling machines, lathes, plasma cutters, routers, cutting machines, robots and hexapods. It can control up to 9 axes or joints of a CNC machine using G-code (RS-274NGC) as input. It has several GUIs suited to specific kinds of usage (touch screen, interactive development).

Currently it is almost exclusively used on x86 PC platforms, but has been ported to other architectures.. It makes extensive use of a real time-modified kernel, and supports both stepper- and servo-type drives.

It does not provide drawing (CAD - Computer Aided Design) or G-code generation from the drawing (CAM - Computer Automated Manufacturing) functions.

2.2 History

The EMC Public Domain software system was originally developed by NIST, as the next step beyond the National Center for Manufacturing Sciences / Air Force sponsored Next Generation Controller Program[NGC 1989] /Specification for an Open Systems Architecture[SOSAS]. It was called the EMC [Enhanced Machine Controller Architecture 1993]. Government sponsored Public Domain software systems for the control of milling machines were among the very first projects developed with the digital computer in the 1950s. It was to be a "vendor-neutral" reference implementation of the industry standard language for numerical control of machining operations, RS-274D (G-code).

The software included the RS274 interpreter driving the motion trajectory planner, real-time motor/actuator drivers and a user interface. It demonstrated the feasibility of an advanced numerical control system using off the shelf PC hardware running FreeBSD or Linux, interfacing to various hardware motion control systems. Additional development continues using current and additional architectures (i.e. ARM architecture devices).

The demonstration project was very successful and created a community of users and volunteer contributors. Around June 2000, NIST relocated the source code to sourceforge.net under the Public Domain license in order to allow external contributors to make changes. In 2003, the community rewrote some parts of it, reorganized and simplified other parts, then gave it the new name, EMC2. EMC2 is still being actively developed. Licensing is now under the GNU General Public License.

The adoption of the new name EMC2 was prompted by several major changes. Primarily, a new layer known as HAL (Hardware Abstraction layer) was introduced to interconnect functions easily without altering C code or recompiling. This split trajectory and motion planning from motion hardware, making it easier to generate control programs to support gantry machine, lathe threading and rigid tapping, SCARA robot arms and a variety of other adaptations. HAL comes with some interactive tools to examine signals and connect and remove links. It also includes a virtual oscilloscope to examine signals in real time. Another change with EMC2 is Classic Ladder, (an open-source ladder logic implementation) adapted for the real time environment to configure complex auxiliary devices like automatic tool changers.

Around 2011, the name was changed officially from EMC2 to LinuxCNC. This was done at the insistence of EMC Corporation and the agreement of the project leadership. Internally some refer to LinuxCNC by EMC or EMC2 as it was historically known. EMC Corporation proposed that the LinuxCNC project, as previously named, would be confusing for customers or potential customers with their (mainly) storage related products.

2.3 Platforms

Due to the need of fine grained, precise real time control of machines in motion, EMC requires a platform with real-time computing capabilities. It uses Linux kernel with real time extensions (RTAI) or with RT-PREEMPT kernel using linuxcnc's 'uspace' flavour of RTAPI. Installing EMC2 (and the underlying real time extension) is a daunting task, therefore prebuilt binary packages have been built and are being distributed. The policy for EMC2 is to build packages and offer support on Ubuntu LTS (long-term support) releases.[1]

2.4 Design

LinuxCNC uses the model of 'sense, plan, act' in its interactions with hardware.[2] For instance, it reads the current axis position, calculates a new target position/voltage, and then writes that to the hardware. There is no buffering of commands nor are externally initiated reads or writes allowed. This no-buffering approach gives the most freedom to adding or changing capabilities of LinuxCNC. By using relatively "dumb" external hardware and programming the capabilities in the host computer, LinuxCNC is not locked to any one piece of hardware. It also allows an interested user to easily change behaviour/capabilities/hardware.

This model tends to lend itself to specific types of external interfaces---PCI, PCIE, Parallel port (in SPP or EPP mode), ISA, and Ethernet have been used for motor control. USB and RS232 serial are not good candidates; USB having bad realtime capabilities and RS232 being too slow for motor control.

LinuxCNC has basic "realtime" requirements because of this model. The interval between reading and writing must be consistent and reasonably fast. A typical machine does realtime calculations in a 1 millisecond repeating thread. The reading and writing to hardware must be a small part of this time, e.g. 200 microseconds, otherwise the phase shift makes tuning more difficult and there is less time available for the non-realtime programs, which may make the screen controls less responsive.

LinuxCNC "employs a trapezoidal velocity profile generator."[3]

2.5 Configuration

LinuxCNC uses a software layer called HAL (Hardware Abstraction Layer).[4]

HAL allows a multitude of configurations to be built [5] while being flexible: one can mix & match various hardware control boards, output control signals through the parallel port or serial port - while driving stepper or servo motors, solenoids and other actuators.

LinuxCNC also includes a software programmable logic controller (PLC) which is usually used in extensive configurations (such as complex machining centres). The software PLC is based on the open source project Classicladder,[6] and runs within the real-time environment.

2.6 References

Notes

[1] "Installing EMC2 ... and supported platforms". Linuxcnc Board of Directors. September 18, 2010. Retrieved 2010-09-29.

[2] "Linuxcnc hardware design requirements".

[3] "Simple Tp Notes".

[4] "EMC2's Hardware Abstraction Layer". Linuxcnc Board of Directors. Retrieved 2010-09-30.

[5] "A couple case studies". Retrieved 2010-09-30.

[6] "ClassicLadder". sites.google.com. Retrieved 2014-03-06.

Bibliography

- Proctor, F. M., and Michaloski, J., "Enhanced Machine Controller Architecture Overview," NIST Internal Report 5331, December 1993. Available online at ftp://129.6.13.104/pub/NISTIR_5331.pdf
- Albus, J.S., Lumia, R., "The Enhanced Machine Controller (EMC): An Open Architecture Controller for Machine Tools," Journal of Manufacturing Review, Vol. 7, No. 3, pp. 278–280, September 1994.
- Lumia, "The Enhanced Machine Controller Architecture", 5th International Symposium on Robotics and Manufacturing, Maui, HI, August 14–18, 1994, http://www.nist.gov/customcf/get_pdf.cfm?pub_id=820483
- Fred Proctor et al., "Simulation and Implementation of an Open Architecture Controller", Simulation, and Control Technologies for Manufacturing, Volume 2596, Proceedings of the SPIE, October 1995, http://www.isd.mel.nist.gov/documents/proctor/sim/sim.html
- Fred Proctor, John Michaloski, Will Shackleford, and Sandor Szabo, "Validation of Standard Interfaces for Machine Control", Intelligent Automation and Soft Computing: Trends in Research, Development, and Applications, Volume 2, TSI Press, Albuquerque, NM, 1996, http://www.isd.mel.nist.gov/documents/proctor/isram96/isram96.html

- Shackleford and Proctor, "Use of open source distribution for a Machine tool Controller", Sensors and controls for intelligent manufacturing. Conference, Boston MA, 2001, vol. 4191, pp. 19–30, http://www.isd.mel.nist.gov/documents/shackleford/4191_05.pdf or http://dx.doi.org/10.1117/12.417244

- Morar et al., "ON THE POSSIBILITY OF IMPROVING THE WIND GENERATORS", International Conference on Economic Engineering and Manufacturing Systems, Brasov, 25–26 October 2007, http://www.recentonline.ro/021/Morar_L_01a.pdf

- Zhang et al., "Development of EMC2 CNC Based on Qt", Manufacturing Technology & Machine Tool, 2008, http://en.cnki.com.cn/Article_en/CJFDTOTAL-ZJYC200802046.htm

- Leto et al., "CAD/CAM INTEGRATION FOR NURBS PATH INTERPOLATION ON PC BASED REAL-TIME NUMERICAL CONTROL", 8th INTERNATIONAL CONFERENCE ON ADVANCED MANUFACTURING SYSTEMS AND TECHNOLOGY JUNE 12–13, 2008 UNIVERSITY OF UDINE - ITALY, http://158.110.28.100/amst08/papers/art837759.pdf

- Xu et al., "Mechanism and Application of HAL in the EMC2", Modern Manufacturing Technology and Equipment 2009-05, http://en.cnki.com.cn/Article_en/CJFDTOTAL-SDJI200905037.htm

- Zivanovic et al., "Methodology for Configuring Desktop 3-axis Parallel Kinematic Machine", FME Transactions (2009) 37, 107-115,

- Glavonjic et al., "Desktop 3-axis parallel kinematic milling machine", The International Journal of Advanced Manufacturing Technology Volume 46, Numbers 1-4, 51-60 (2009), http://dx.doi.org/10.1007/s00170-009-2070-3

- Staroveski et al., "IMPLEMENTATION OF A LINUX-BASED CNC OPEN CONTROL SYSTEM", 12th INTERNATIONAL SCIENTIFIC CONFERENCE ON PRODUCTION ENGINEERING –CIM2009, Croatian Association of Production Engineering, Zagreb 2009,

- Li et al., "Control system design and simulation of parallel kinematic machine based on EMC2", Machinery Design & Manufacture 2010-08, http://en.cnki.com.cn/Article_en/CJFDTOTAL-JSYZ201008074.htm

- Li et al., "Kinematics Analysis and Control System Design of 6-DOF Parallel Kinematic Machine with Matlab and EMC2", Advanced Materials Research (Volumes 102 - 104): Digital Design and Manufacturing Technology, 2010, http://dx.doi.org/10.4028/www.scientific.net/AMR.102-104.363

- Klancnik et al., "Computer-Based Workpiece Detection on CNC Milling Machine Tools Using Optical Camera and Neural Networks", Advances in Production Engineering & Management 5 (2010) 1, 59-68, http://maja.uni-mb.si/files/apem/APEM5-1-view.pdf

- Milutinovic et al., "Reconfigurable robotic machining system controlled and programmed in a machine tool manner", The International Journal of Advanced Manufacturing Technology, 2010, http://dx.doi.org/10.1007/s00170-010-2888-8

2.7 External links

- EMC2 project homepage at www.linuxcnc.org
- EMC2 project wiki
- The NIST RS274NGC Standard - Version 3 Aug 2000 also available as a PDF
- The Enhanced Machine Controller homepage at NIST

2.8 Text and image sources, contributors, and licenses

2.8.1 Text

- **Hardware abstraction** *Source:* http://en.wikipedia.org/wiki/Hardware%20abstraction?oldid=623653409 *Contributors:* Kku, Marius, Tagishsimon, Rich Farmbrough, Neg, Danhash, Chobot, DaGizza, NTBot, SmackBot, Brianski, Bluebot, Phatom87, Alaibot, Widefox, JAnDbot, Metrax, Mr. Stradivarius, Alexbot, Arjayay, DanielPharos, Callmejosh, Addbot, Dawynn, AnomieBOT, Erik9bot, Matzi4, WikitanvirBot, ZéroBot, MajorVariola, Ego White Tray, Bomazi, Helpful Pixie Bot, Wbm1058, Tech77, Greenstruck, Monkbot and Anonymous: 13
- **LinuxCNC** *Source:* http://en.wikipedia.org/wiki/LinuxCNC?oldid=646656844 *Contributors:* Archivist, Bearcat, Vadmium, Ringbang, GraemeLeggett, Ichudov, Chris the speller, Chendy, Wizard191, Servant74, Hebrides, Electron9, Arch dude, 7severn7, Cirt, Sun Creator, Onomou, Addbot, Grandscribe, Team4Technologies, AnomieBOT, SubtlySnide, I dream of horses, EmausBot, JonathanMElson, SWPadnos, BG19bot, Game-Guru999, Autodidaktos, K7L, Mogism, Oroszegy, Petebachant, Chester8888 and Anonymous: 13

2.8.2 Images

2.8.3 Content license

- Creative Commons Attribution-Share Alike 3.0