

Advanced search

Linux Journal Issue #127/November 2004



Features

OSCAR and Bioinformatics by *Bernard Li*

Use the software that the big labs use, and put a decade of Linux cluster management experience to work for you.

Scientific Visualizations with Pov-Ray by *Leigh Orf*

Here's how a much-needed patch turned the popular rendering package into a scientific power tool.

Improving Application Performance on HPC Systems with Process Synchronization by *Paul Terry, Amar Shan and Pentti Huttunen*

It's a simple concept that gives big results. A team from Cray takes a leap forward in the struggle to keep all processors in the cluster occupied efficiently.

Indepth

Readers' Choice 2004 by *Heather Mead*

Evolution or mutt? Vim or Kate? Old school or eye candy? And what's your favorite beverage for coding sessions? Heather has the answers.

MyHDL: a Python-Based Hardware Description Language by *Jan Decaluwe*

Design hardware in Python? Why not? New features of the language are making it a simple, readable choice for new hardware ideas.

Revision Control with Arch: Introduction to Arch by *Nick Moffitt*

Get started with a new, flexible working style that's convenient for far-flung projects and hacking on your laptop.

Embedded

[Linux and RTAI for Building Automation](#) by *Andres Benitez and Vicente Gonzales*

Simple commodity units and Linux do the work of a big expensive system. Sounds familiar, but we're talking about air conditioning.

Toolbox

At the Forge [Aggregating with Atom](#) by *Reuven M. Lerner*
Kernel Korner [AEM: a Scalable and Native Event Mechanism for Linux](#) by *Frédéric Rossi*

Cooking with Linux [Performing at the Speed of Light](#) by *Marcel Gagné*

Paranoid Penguin [Linux Filesystem Security, Part II](#) by *Mick Bauer*

Columns

Linux for Suits [We're Going to Be a 90% Linux Shop](#) by *Doc Searls*
EOF [No 2.7 Kernel?](#) by *Greg Kroah-Hartman*

Reviews

[GumStix WS200X](#) by *Michael Boerner*

[Mastering UNIX Shell Scripting](#) by *Marco Fioretti*

Departments

[From the Editor](#)

[Letters](#)

[upFRONT](#)

[New Products](#)

[Archive Index](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

OSCAR and Bioinformatics

Bernard Li

Issue #127, November 2004

With new cluster deployment and management tools, you can set up a 64-node cluster in an hour, then put it to work on your research.

The OSCAR (Open Source Cluster Application Resources) Project has been around for about four years. The concept initially was proposed in January 2000, and the first organizational meeting was held in April of the same year. The group acknowledged that cluster assembly is time consuming and repetitive. Thus, the project's goal was to create a toolkit to automate this process. In doing so, the group hoped to broaden the usage of clusters and adapt them for the academic and private sector.

The OSCAR Project is overseen by the advisory group OCG (Open Cluster Group), an informal group with open membership. The OCG strives to make cluster computing more practical for high-performance computing (HPC) research and development. The group, like the OSCAR Project, is directed by representatives from research/academia as well as industry. Key players of the group include Bald Guy Software, BC Genome Sciences Centre, Dell, Indiana University, Intel, Louisiana Tech University, Oak Ridge National Laboratory, Revolution Linux and Sherbrooke University.

OSCAR is one of the OCG working groups. Other projects include HA-OSCAR (high-availability), Thin-OSCAR (diskless) and SSS-OSCAR (Scalable Systems Software). To learn more about OCG and its projects, see the on-line Resources for this article.

The first release of OSCAR was in April 2001, and since then we have released two major versions. Our release cycle usually coincides with the SuperComputing Conference, which is held annually in November. The current version as of writing is 3.0, and we are aiming at releasing 4.0 by SuperComputing04.

The goal of OSCAR is to provide users with the best practices for installing, programming and maintaining HPC clusters. Many open-source components individually work well in an HPC environment but require specific setup routines. OSCAR acts as the glue to integrate all these components together to provide a working toolkit. The project targets mid-size clusters (50+ node clusters). Community feedback suggests this size represents the majority of clusters assembled today.

OSCAR has the following components:

- Administration: System Installation Suite (SIS), Cluster Command and Control (C3) and OPIUM (user management).
- HPC tools: the parallel programming libraries: MPICH, LAM/MPI and PVM; batch systems: OpenPBS/MAUI, Torque and SGE; monitoring tools: Ganglia and Clumon; and other third-party OSCAR packages.
- Core infrastructure/management: OSCAR Database (ODA) and OSCAR Package Downloader (OPD).

OSCAR developers are spread out over various geographical locations and weekly teleconferences are set up to discuss ongoing development issues. The group also holds annual general meetings to brainstorm new features to be included in future releases. An annual symposium also is held, usually in conjunction with HPCS (International Symposium on High Performance Computing Systems and Applications) where users are encouraged to present papers on their experiences with OSCAR as well as other development work relating to HPC. The 2nd annual OSCAR Symposium commenced in May 2004 in Winnipeg, Canada, and the proceedings are now available.

Introduction to Bioinformatics

Bioinformatics is the marriage between biology and computer science/IT and is a rapidly growing field with high stakes in the HPC world. In simple terms, bioinformatics is concerned with the use of computer algorithms and systems to analyze biological data such as DNA, RNA, protein and regulatory elements.

Biological data are mainly string sequences. The analyses are usually string manipulations making Perl the programming language of choice for most bioinformaticians. Many open-source Perl programmers contribute to the Bioperl effort, which is a deposit of Perl modules specifically for performing bioinformatics analyses. Java is employed for larger projects and often for projects that involve graphical interfaces. Python is gaining a strong foothold in the field as more programmers learn about the ease of use and high readability of this relatively new but powerful programming language.

Linux clusters are very popular within the bioinformatics community, because a lot of the analyses tend to be long-running and repetitive. Linux clusters are ideal for running such embarrassingly parallel jobs that can be executed independently of one another. These are not considered true parallel programs because they do not need parallel programming libraries such as MPI. Bioinformatics routinely performed on clusters encompass running multiple scripts and algorithms on different inputs and can be executed on different CPUs, each with its own address space.

Walk-Through of a Typical OSCAR Installation

Installing a cluster with the OSCAR Toolkit is a straightforward process. If you have installed Linux before, you should have little trouble.

Currently, the OSCAR Project supports three Linux distributions: Red Hat 8.0, Red Hat 9.0 and Mandrake 9.0. The main Linux installation requirement is that an X windowing environment such as KDE or GNOME is installed; otherwise, a typical workstation install with software development tools should be sufficient.

After Linux is installed and configured on the head node, you can download the OSCAR tarball from the projects page, untar it and do the configure, make, make install routine.

OSCAR, by default, is installed to /opt/oscar. You can change this using the --prefix flag with configure. After OSCAR has been installed, you can start the OSCAR Wizard, which provides step-by-step installation menus for setting up your cluster.

To invoke the wizard, go to /opt/oscar and type `./install_cluster ethX`. Here, ethX refers to the interface that is on the cluster network.

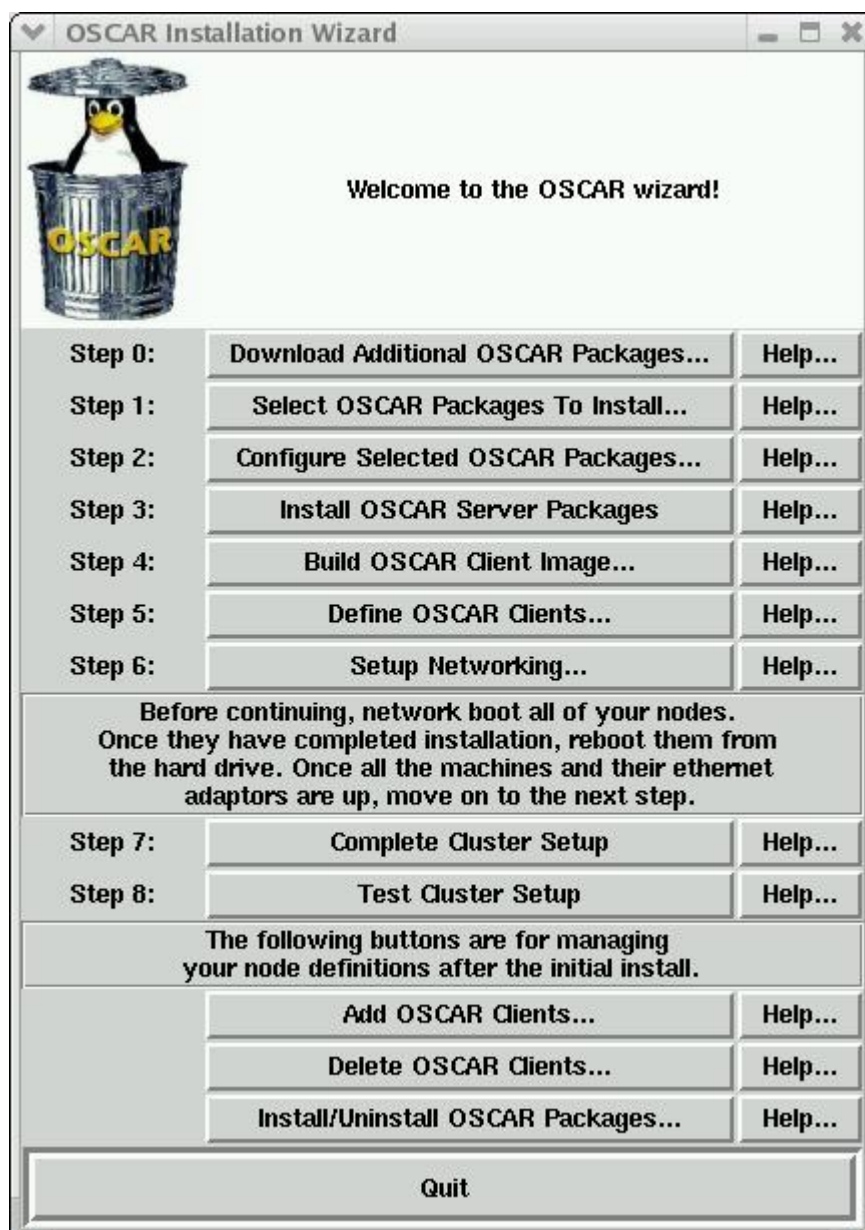


Figure 1. The OSCAR installation main menu. In less than ten steps the cluster is ready to compute!

OSCAR comes with many prebundled packages. Other packages available from the various repositories also may be of interest. To download those, simply click on Download Additional OSCAR Packages in the OSCAR Wizard and choose the package(s).

Next, you can select the packages you want to install. Packages have three main categories: core, provided and third party. Core packages must be installed and cannot be deselected. Provided packages are the ones the OSCAR team recommends you install, and third-party packages are all the remaining packages available from the repositories.

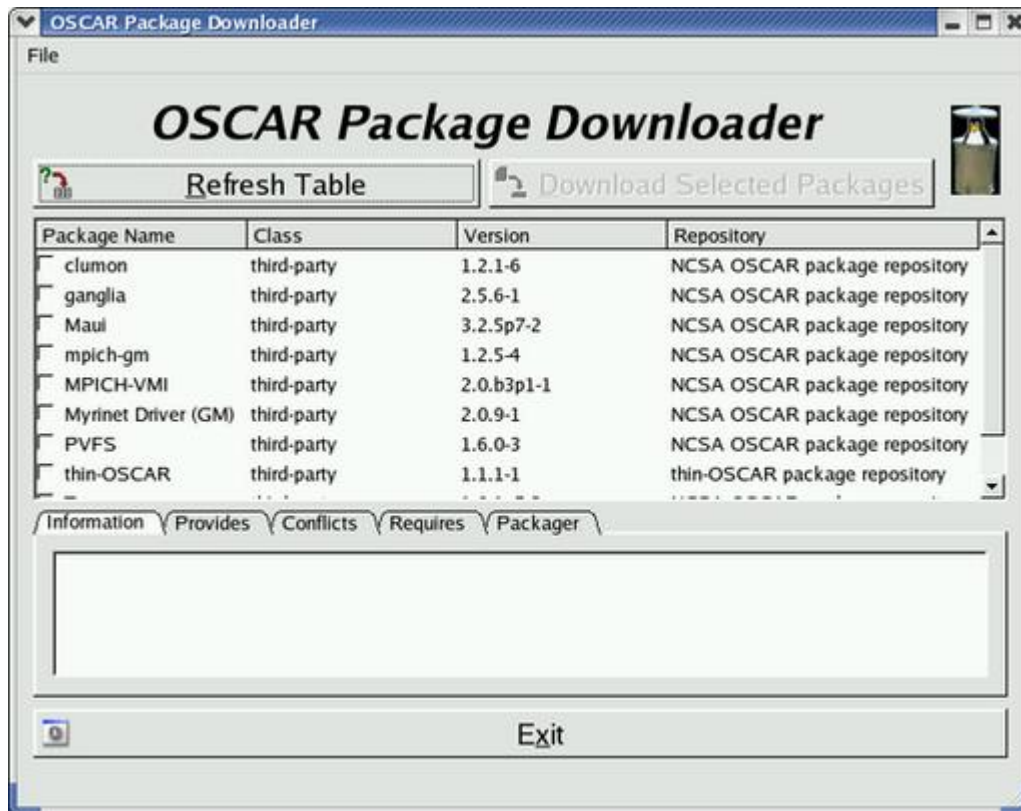


Figure 2. OSCAR Package Downloader—additional packages can be downloaded in this menu.

Configuration changes can be made to packages using the Configure Selected OSCAR Packages menu.

The next stage is to Install OSCAR Server Packages. This is non-interactive and basically sets up packages for use on the server. When it is done, you are alerted by a pop-up window.

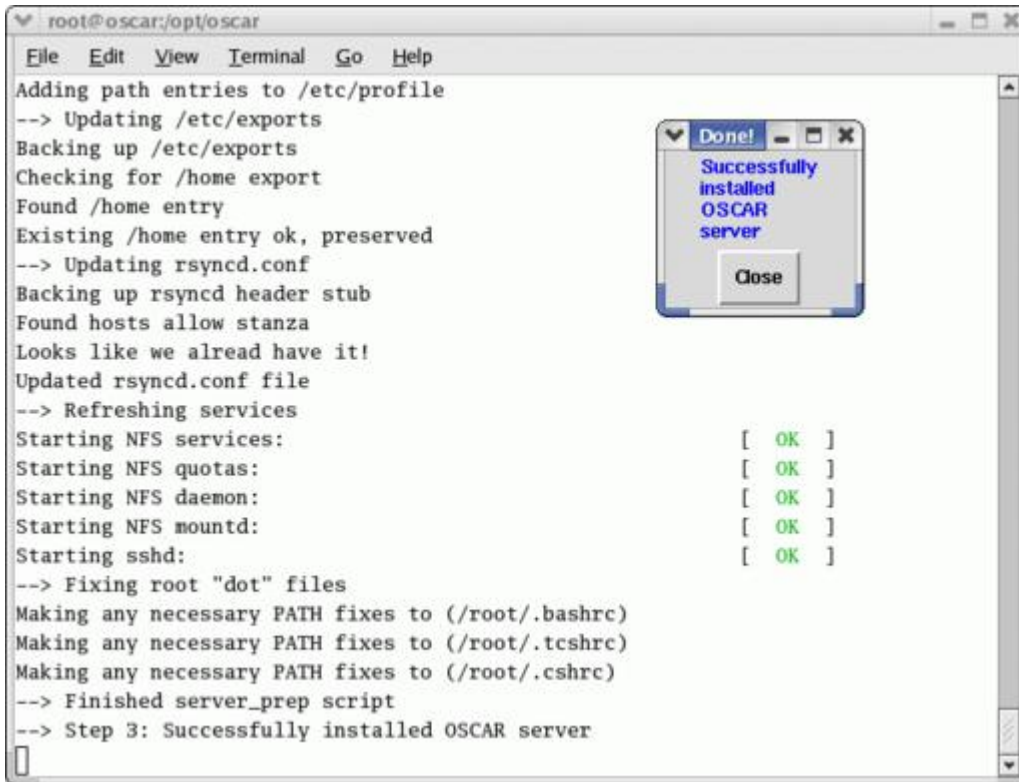


Figure 3. Non-interactive step where server packages are installed.

Now the fun part begins. You can build a client image with the Build OSCAR Client Image step. In this step, you select a few options for the client image you want to build. This image then is pushed to your client nodes. You can provide a list of RPM packages to be installed on the base image, and you also can decide how to partition the hard drive and assign IP addresses. Lastly, you can choose the post-image action, such as rebooting the machine when imaging is done.

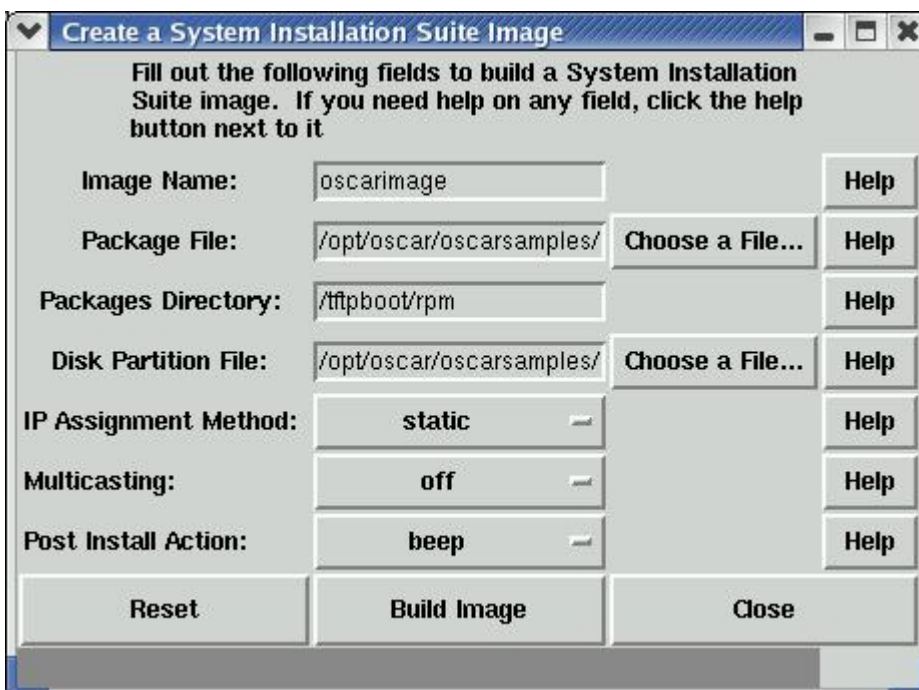


Figure 4. Create a client image based on a user-provided package list and partition table.

In the Define OSCAR Clients step, you can specify the domain name, base name of your clients, the number of nodes you want to bring up for this session and other network settings. After you click the Add clients button, these definitions are configured and the cluster is almost ready to be rolled out.

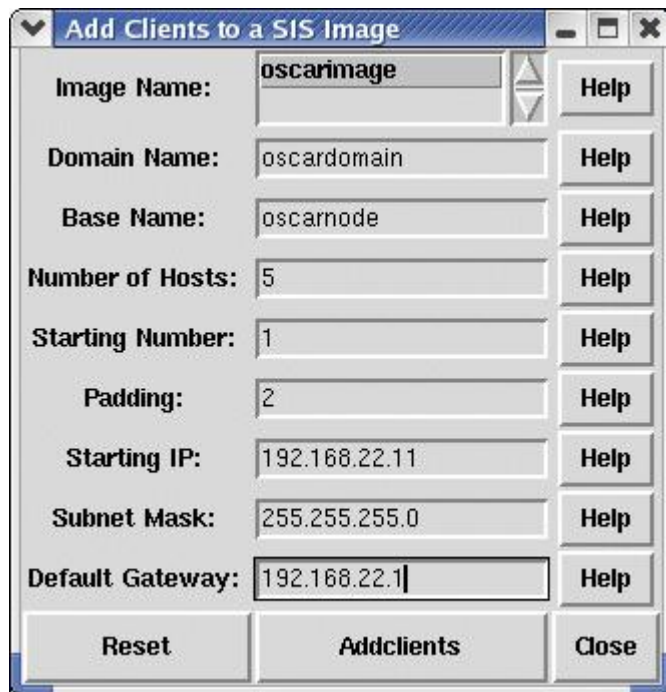


Image Name:	oscarimage	Help
Domain Name:	oscardomain	Help
Base Name:	oscarnode	Help
Number of Hosts:	5	Help
Starting Number:	1	Help
Padding:	2	Help
Starting IP:	192.168.22.11	Help
Subnet Mask:	255.255.255.0	Help
Default Gateway:	192.168.22.1	Help
Reset		Addclients
		Close

Figure 5. Cluster nodes and network settings are defined here.

Next, you will want to set up networking for your cluster. Here, you can boot up your client nodes with PXE or floppy disk, and the OSCAR head node then collects the MAC addresses and you can assign them to particular hosts. Once this is done, the client nodes are imaged immediately. Typically, it takes anywhere from 10–30 minutes to image each node, depending on the speed of your hard drive. When deploying a cluster, multiple nodes can be imaged at the same time. We usually start up ten nodes to be imaged at a time so the head node does not get heavily loaded. With this staggered approach, you should be able to deploy a 64-node cluster within an hour.

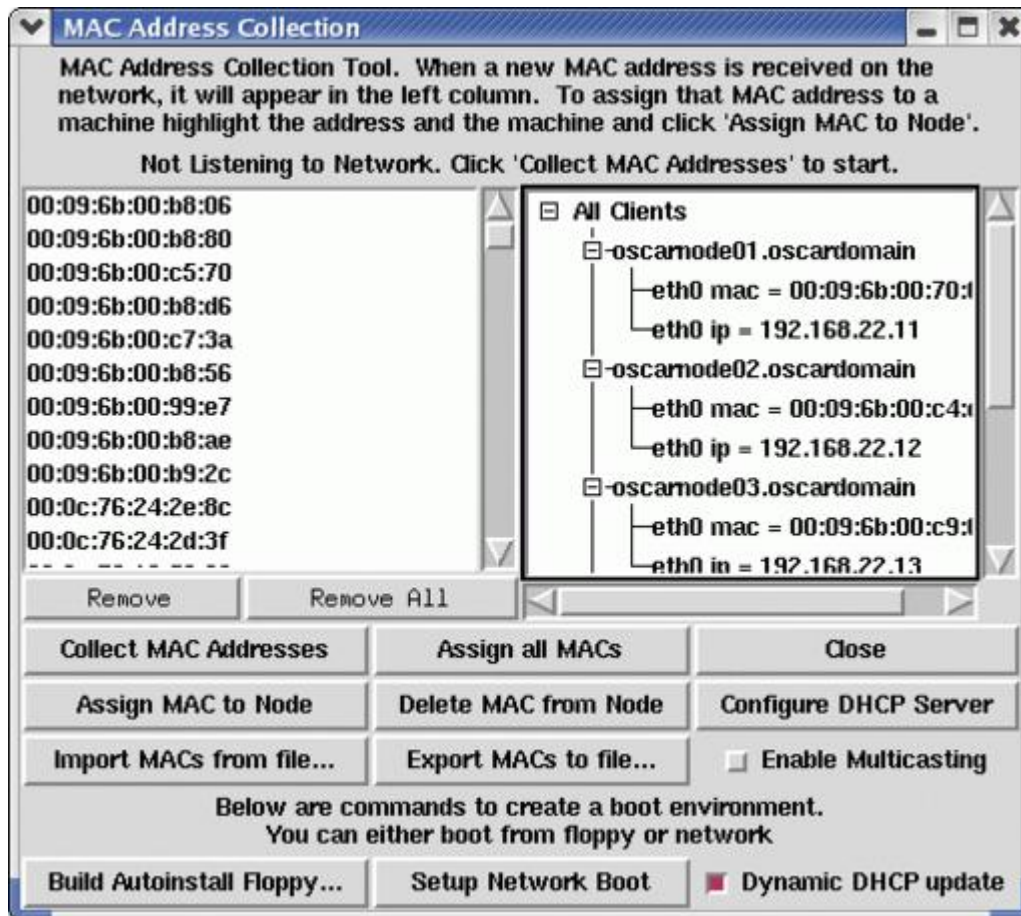


Figure 6. Cluster nodes are network-booted, and their MAC addresses assigned to host entries.

After the nodes are imaged and rebooted, you can continue with the next step, which is to Complete the Cluster Setup. This again is a non-interactive step in which final installation configurations and other clean-up functions are performed.

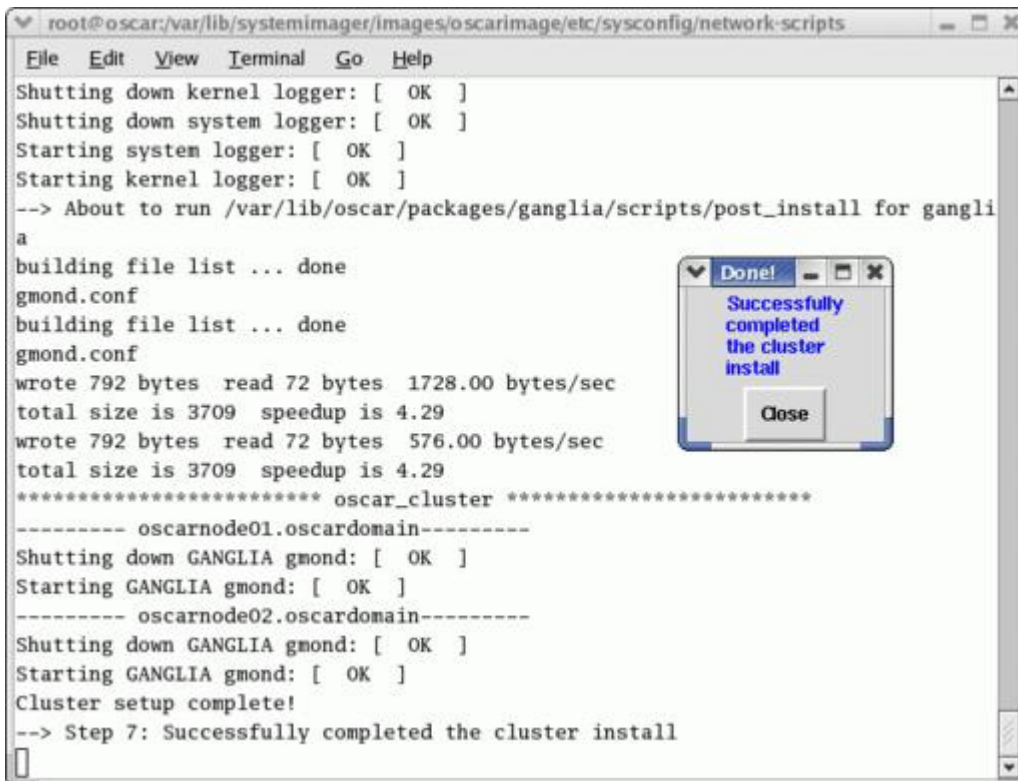


Figure 7. Non-interactive step where final installation configurations and clean-up functions are performed.

Lastly, you may want to Test Cluster Setup. This runs a series of tests for the cluster install as well as for individual packages. If all goes well, you will pass all the tests, thus confirming your cluster setup is complete and ready to run computations.



Figure 8. All systems go—cluster is ready to rock and roll.

The OSCAR Toolkit is easy to install and, in general, works with most hardware. However, if you run into problems, two mailing lists are available for help. The oscar-users list is the first place you should ask questions. Most of the core team frequently reads the list, and other users help out too. However, if you have questions regarding the development of OSCAR, there is the oscar-devel list. Both lists are closed lists; you need to subscribe before you can post to them.

New Features in OSCAR 4.0

We are planning to include various features in this upcoming release. They are partitioned into four main categories: NEST, node groups, Linux distributions and SIS.

NEST (Node Event and Synchronization Tools) is used to ensure that OSCAR packages are in sync with their centrally stored configurations across all the cluster nodes. Currently, when you install a new cluster node, post_install scripts for OSCAR and all packages have to be run on all the cluster nodes regardless of whether they need to. Although this model has worked on a medium scale, clearly a scalability limitation is imposed by it. The biggest change with NEST is that package configuration will be pulled from the server as opposed to being pushed to the clients. Operations will be executed only if they are necessary, which is a more elegant way than the brute-force execution scheme we are employing presently.

Node groups are arbitrary groupings of nodes in the cluster. With this new feature, it is possible to install and manage OSCAR packages selectively for these groups. In the upcoming release, we plan to support only the server and client node groups. In the future, however, users will be allowed to define their own groups.

One of the key features that defines OSCAR is our support for several Linux distributions. With the new release we hope to introduce support for Fedora Core 2 and 3, Red Hat Enterprise Server 3.0 and Mandrake 10. Supporting these distributions will also introduce support for IA-64 and x86-64 architectures.

System Installation Suite (SIS), which includes SystemImager, is the collection of programs that performs the image deployment of an OSCAR system. There are two main SIS-related improvements. First is the disk type autodetection. Traditionally, OSCAR cluster images are created with one type of hard disk in mind (either IDE or SCSI). With this OSCAR-specific patch, you can use the same image to deploy on different machines with different types of hard disks, as long as the base hardware is similar.

Second, a tool is available so users can use specific kernel modules for booting the nodes for imaging. Sometimes it is difficult to get newer hardware to work with OSCAR, because the SIS kernel boot image does not have the supported drivers. With this tool, you can use an existing kernel with known working modules as the SIS boot kernel and use that to boot up your client nodes so they can be imaged. This feature will be included in the next SystemImager development release, which we hope to include in OSCAR 4.0.

Creating Packages for OSCAR

OSCAR supplies packages for commonly used cluster-aware applications. They are simply RPM packages with corresponding metafiles and installation scripts. These packages are created and maintained by the OSCAR core team and package authors. If there is an application you would like to install on your cluster, but did not find it available from OPD (OSCAR Package Downloader), please create a package for it. The OSCAR team is open to contributions and possibly even hosting an OSCAR software package you might create. The team extends this invitation to software developers too.

OSCAR packages reside in package repositories, which are decentralized Web spaces provided by the package authors to host the package files. The URIs of these repositories are stored in a master repository list.

Creating OSCAR packages is generally a straightforward process. If an RPM is readily available, you are already halfway done. What remains is to create some files to store metadata about the package/RPMs and also scripts to propagate configuration files for the entire cluster. This is relatively easy to do because certain assumptions can be made about an OSCAR cluster.

If, however, an RPM package is not currently available, you need to package the application in RPM format before continuing. Creating an RPM may or may not be easy, depending on the complexity of the application. You need to create a spec file and build the RPM(s) and corresponding SRPM with the source tarball.

The first OSCAR package that the Genome Sciences Centre (GSC) has put out is for Ganglia, a cluster monitoring system. We have started working on our second package, which is for Sun Grid Engine, an open-source batch scheduling system sponsored by Sun Microsystems. This should be available from OPD at the GSC OSCAR Repository at a later date.

Bioinformatics Applications and OSCAR Cluster

The most commonly used bioinformatics program is BLAST, a sequence alignment/search tool. Querying genetic databases with multiple gene sequences is highly amenable to parallelization; it is very easy to split this

problem into many sub-jobs, with each sub-job querying the database with one set of input. Solutions exist that perform such database/query splitting for you, most notably the open-source mpiBLAST as well as the commercial version from Paracel. Parallelized versions of BLAST scale quite well but, of course, reach a point where adding more nodes simply does not increase performance due to the overhead of starting separate and smaller jobs on multiple nodes.

FPC is another application we use a great deal and is used for assembling, editing and viewing fingerprint-based physical maps. The original parallelized version of this application was developed at the GSC, but it was not batch system-aware. We recently have integrated parallel FPC with Sun Grid Engine so that users can easily request a specific number of nodes to run this application.

We also are developing a peer-to-peer application called Chinook for sharing bioinformatics services. Currently, it is still under development, and we are working on integrating it with our cluster. This potentially could link up grids in the future and provide an alternative to the Globus toolkit.

Currently, our 200-node cluster is being heavily used to discover and classify regulatory elements in the human genome. The human genome consists of roughly 30,000 genes, and we are employing different algorithms to scan each gene. Genes are grouped into batches of 1,500, and a typical run would take about four days on an Opteron 2.0GHz machine. Without cluster technology, this kind of research would not be possible.

Conclusion

The OSCAR Toolkit has come a long way since its first release. More and more people have found it easy to use and deploy—the key to getting clustering technology more widely adopted. Bioinformatics will continue to grow with high-performance computing. Soon, it is likely that cluster toolkits geared toward the bioinformatics community will become more widely available—a solution that includes all the tools for running parallel bioinformatics applications and is easy to install and deploy.

Acknowledgements

The author would like to thank Mark Mayo, Asim Siddiqui and Steven Jones for giving him the opportunity to work on the Linux cluster at the GSC and for identifying OSCAR as the tool to use. The author also would like to thank the OSCAR core team, developers and users for creating a great community for sharing HPC knowledge and information. Last, but not the least, the folks at NCSA who contributed much time and effort into the OSCAR Project. Personal thanks to Jeremy Enos, Reni Warren and Martin Krzywinski for providing valuable comments and suggestions for this article.

Resources for this article: </article/7760>.

Bernard Li is a High-Performance Computing Specialist at Canada's Michael Smith Genome Sciences Centre. He spends time managing the Linux cluster infrastructure and integrating bioinformatics applications with the cluster. He is a core developer for OSCAR and a fan of Sun Grid Engine. He can be reached at bli@bcgsc.ca.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Scientific Visualizations with POV-Ray

Leigh Orf

Issue #127, November 2004

With a little work, the Persistence of Vision Raytracer (POV-Ray) can be adapted to create stunning three-dimensional imagery from floating-point scientific data files.

I am a meteorologist at Central Michigan University doing research with collaborators at the University of Illinois on the behavior of supercell thunderstorms, the long-lived rotating monsters that wreak havoc across the Great Plains of the United States every spring. My primary tool for studying the behavior of these fearsome storms is a numerical model called NCOMMAS, a computer application written in FORTRAN 90 that uses the equations of physics to emulate the three-dimensional state of the atmosphere over time. This model produces an immense amount of data over the course of a four-hour thunderstorm simulation, on the order of 200GB, even when using a lossy compressed history file format. One of the great challenges I face in my research is finding ways to visualize this data in a way that provides scientific insight into the physical nature of the simulated storm.

One way to visualize 3-D data is to use a raytracer, a computer application that simulates the behavior of light interacting with virtual objects in three dimensions to create a bitmapped image (Figure 1). This bitmapped image can be displayed on a computer screen and/or saved to disk in an image format such as PPM or TIFF. The Persistence of Vision Raytracer, POV-Ray for short, is a popular open-source raytracer that caught my attention while I was working on my doctoral thesis at the University of Wisconsin in the mid-1990s. At the time, I was looking for software to visualize my 3-D model data of microbursts, severe downdrafts that sometimes descend from thunderstorm clouds. Being accustomed to the shared-source nature of the academic world and being a poor grad student, I was looking for free software distributed in source code form that I could download and modify to fit my own specific needs. POV-Ray was the logical choice for me then, and it continues to suit my needs today in creating raytraced representations of my research data.

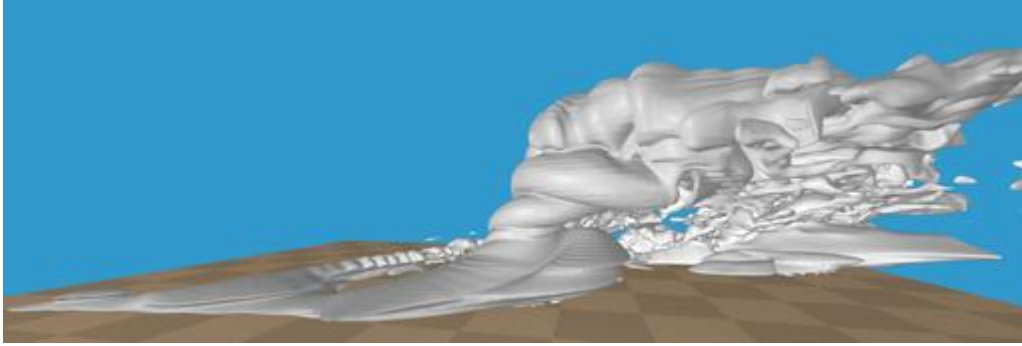


Figure 1. An aerial view of the whole supercell thunderstorm cloud from a distance of about 30 kilometers, rendered with POV-Ray.

Rendering scientific data isn't the task for which POV-Ray was designed, however, and few researchers are using POV-Ray for rendering scientific data. Other raytracing packages geared toward the researcher doing work with numerical models exist, but they are proprietary and expensive. In this article, I outline the process of modifying POV-Ray so that isosurfaces of 3-D scientific data can be rendered natively.

Getting the Source

Although POV-Ray is distributed in binary form for Linux, Mac OS and Microsoft Windows, you need to obtain the source code in order to apply patches and make further customizations. I am using the latest version of POV-Ray available as of this writing, version 3.5. You need to select the Unix/Linux/Generic Source Code option from the POV-Ray download page. In addition, you need to obtain Ryouichi Suzuki's Density File extension patch (see the on-line Resources), which actually is a Zip file containing replacement source code for a handful of the POV-Ray files. The file `pov35dfjs.zip` should be unpacked in the `povray-3.50c/src` directory, where 13 files will be overwritten.

Scenes and Isosurfaces

POV-Ray works by reading a scene description file that contains all of the information necessary to create a bitmapped image. POV-Ray has its own scene description language, which is well documented on the POV-Ray Web site. If you never have used a raytracer before, I recommend familiarizing yourself with raytracer basics and POV-Ray's scene description file before making modifications to the source.

Items rendered in POV-Ray are called objects. Examples of objects include Box, Sphere, Torus and Plane. The user specifies where objects exist in the scene, what parameters to use in creating the objects, what color to make the objects, lighting parameters and so forth. These specifications are made in a scene description file, sometimes called a pov file because of the `.pov` suffix, which is a plain-text file parsed by POV-Ray at runtime.

One versatile object is the isosurface, a 3-D shape whose surface represents points where a function's value is constant. The constant value of the function is chosen by the user. POV-Ray contains many predefined objects that actually could be represented as isosurfaces. For instance, the following section from a scene description file would render a gray sphere with a radius of 0.7 units, centered at the origin, which is Cartesian gridpoint (0,0,0):

```
sphere
{
  <0,0,0>, 0.7
  pigment {rgb .5}
}
```

The same object could be rendered as an isosurface object in the following way:

```
#declare R = 0.7
isosurface
{
  function {x*x + y*y + z*z - R*R}
  pigment {rgb .5}
}
```

This works because the mathematical formula describing a sphere of radius R is:

$$x^2 + y^2 + z^2 - R^2 = 0$$

This versatility of the isosurface object makes it the object of choice for my thunderstorm images.

Density Files

In the sphere example, a mathematical function was used to calculate the isosurface value. My thunderstorm numerical model data cannot be represented as a mathematical function; instead, it is represented as three-dimensional floating-point arrays containing model variables such as temperature, wind speed and cloud concentration at each grid location (Figure 2).

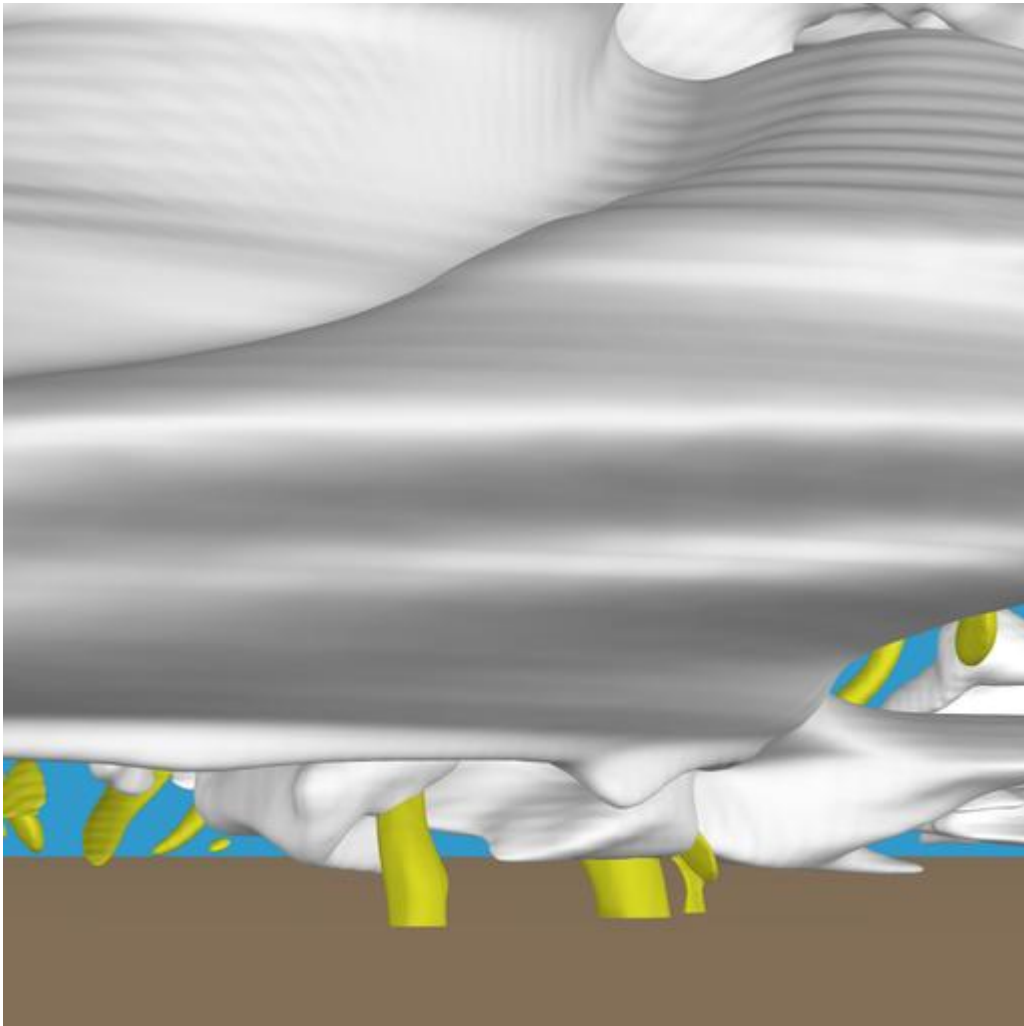


Figure 2. An example of multiple isosurfaces, focusing on a region of the supercell called the wall cloud. The yellow isosurfaces in the foreground, which are located below the wall cloud, represent where tornado-like swirling motion is occurring.

POV-Ray 3.5 has a feature called a density file that allows for the mapping of functions represented as gridpoint values. The POV-Ray documentation describes a density file as follows: "The density_file pattern is a 3-D bitmap pattern that occupies a unit cube from location $\langle 0,0,0 \rangle$ to $\langle 1,1,1 \rangle$. The data file is a raw binary file format created for POV-Ray called df3 format."

Density files can be used as functions passed as an argument to the isosurface object. Here is an example of a density file being used for isosurface rendering:

```
#declare DENSFUNC=function
{
  pattern
  {
    {
      density_file df3 "cloud.df3"
      interpolate 1
    }
  }
  isosurface {function { 0.1 - DENSFUNC(x,y,z) }
```

In the above example, an isosurface with value 0.1 would be created from the cloud.df3 file using a trilinear interpolation scheme (more on interpolation below).

The density file format is strict, and the data values are represented as 8-bit data (unsigned short integers ranging from 0 to 255) scaled internally to range from 0.0 to 1.0. Because my thunderstorm data is 32-bit floating-point data, it is not feasible to use the density file format with the stock POV-Ray 3.5.

Enter Ryouichi Suzuki, who has been providing POV-Ray with unofficial add-on code since 1996. He provided the first patches to POV-Ray 3.0, which introduced the isosurface object, now a standard object in version 3.5. Suzuki's code, contained in the Zip file referenced above, contains routines that expand the functionality of POV-Ray density files, including the option of rendering floating-point density file data.

When using density files as functions one must consider that although a mathematical function is a continuous expression—it is defined for any floating-point value of x , y and z spatial coordinates—a density file is a discrete set of data referenced by integer array indices. Interpolation must be done for referencing space in between gridpoints when rendering a scene. The two interpolation methods available are trilinear and tricubic spline. Trilinear interpolation is fast but usually does not produce as smooth an interpolation as does tricubic spline interpolation.

Getting Model Data into POV-Ray

After patching the POV-Ray 3.5 code with Suzuki's density file code, you can render floating-point isosurfaces if you adhere to the df3 format or Suzuki's extended format. In my case, I had hundreds of gigabytes of HDF (hierarchical data format) model data, which is designed specifically for numerical models. Because I am interested in not only producing a few isosurface images but making animations from hundreds, sometimes thousands, of these images, writing an HDF to df3 converter for each of these files was not a viable option. Hence, I started looking closely at the POV-Ray routines that handle density file data with the hope that I could modify the code to read HDF data.

It was important to me that the modifications I made to POV-Ray would not cause a loss of functionality or break compatibility with the unpatched version. I achieved this goal by adding some new objects, referenced in the scene description file, that could be parsed and rendered by my patched version, while leaving all other objects alone.

The main piece of code I modified is found in pattern.cpp, which contains the Read_Density_File routine. This routine, as you might have guessed, reads

density file data into a three-dimensional array. Using this routine as a template, I created a new routine, Read_Hdf_File, to read my history file data into POV-Ray. This is where most of the modification of the POV-Ray code needs to be made to adhere to your own data format. An abbreviated version of Read_Hdf_file is shown in Listing 1.

Listing 1. Abbreviated Read_Hdf_File Listing from pattern.cpp

```
void Read_Hdf_File (DENSITY_FILE * df)
{
    Locate_Density_File(df->Data->Name);
    df->Data->Type = 1; //floating point data
    Open_HDF_File(df->Data->Name);
    //povray needs array dimensions
    Read_HDF_File_Geometry(nx,ny,nz);
    df->Data->Sx = nx;
    df->Data->Sy = ny;
    df->Data->Sz = nz;
    //this array will contain density file data
    Allocate_Memory(mapF,nx,ny,nz);
    //read variable into mapF array
    Get_HDF_File_Data(Var,mapF,nx,ny,nz);
    //density file pointer now points to model data
    df->Data->DensityF = mapF;
}
```

The function of Read_Hdf_file is to read HDF floating-point data into mapF, a 3-D array of floats, where it now is ready to be manipulated as a density file. I wrote a separate piece of code called history.c, which contains all of the HDF I/O routines referenced in pattern.cpp. Your data file format will require its own format-specific code to read your 3-D data into POV-Ray.

A few more files were modified in order for POV-Ray to recognize the new HDF file format natively and to allow for the rendering of more than one model variable per scene. Table 1 contains a list of the files modified and a brief description of what was done to each file.

Table 1. A listing of modified files to accommodate model data into POV-Ray and a brief description of what was done.

File	Modification
pattern.cp	Get_HDF_File_Data routine added, which reads model data into memory.
pattern.h	Add declaration of Read_Hdf_File.
parstxtr.cpp	Add case statement block for HDF_TOKEN.
tokenize.cpp	Add HDF_TOKEN to Reserved_Words array.
frame.h	Add char *Var1 to Density_file_Data_Struct structure.

File	Modification
parse.h	Add HDF_TOKEN to TOKEN_IDS.

The HDF file format allows for more than one variable to be stored in each file, unlike the density_file format. In my case, each HDF file is a snapshot of the model state at a given time and contains 12 3-D variables per file. It often is illustrative to look at multiple variables, such as cloud, rain, hail and snow, together in one scene. I achieved this by creating a new token representing the HDF file format, called HDF_TOKEN (distinct from DF3_TOKEN representing the original df3 format), and added a new character array called Var to the structure Density_file_Data_Struct. Var is assigned in the scene description file and is passed as an argument to the HDF routines to specify what model variable to select. In order to parse the variable name (represented as a character string), I added an additional case statement to the Parse_PatternFunction routine in parstxtr.cpp (Listing 2). Notice the addition of Parse_Comma and Parse_C_String, which grab the variable to be read.

Listing 2. The HDF_TOKEN case requires extra parsing to allow for the specification of which variable to raytrace. This code snippet is found in the Parse_PatternFunction routine in parstxtr.cpp.

```

EXPECT
CASE (DF3_TOKEN)
    New->Vals.Density_File->Data->Name =
        Parse_C_String(true);
    Read_Density_File(New->Vals.Density_File);
    EXIT
END_CASE
CASE (HDF_TOKEN)
    New->Vals.Density_File->Data->Name =
        Parse_C_String(true);
    Parse_Comma();
    New->Vals.Density_File->Data->Var =
        Parse_C_String(true);
    Read_Hdf_File(New->Vals.Density_File);
    EXIT
END_CASE

```

The Scene Description File

All of the pieces are now in place to construct a scene description file to be interpreted by POV-Ray. I used the example found on Suzuki's density file extension Web page as a template and modified it to fit my needs. Listing 3 contains the full scene description file used to render an isosurface of cloud concentration with a sky-blue background and a tiled surface, shown in Figure 1. Starting at the top, the #version statement is required in order for this unofficial version of POV-Ray to function. The following nine #declare statements specify the Cartesian coordinates that bound the box containing the isosurface, as well as scaling factors.

Listing 3. cloud.pov

```
#version unofficial dfe 3.5;
#include "colors.inc"

#declare x0 = 0.0;
#declare x1 = 700.0;
#declare y0 = 0.0;
#declare y1 = 600.0;
#declare z0 = 0.0;
#declare z1 = 80;

#declare scalex = (x1-x0+1);
#declare scaley = (y1-y0+1);
#declare scalez = (z1-z0+1);

#declare R = 0.7;
#declare G = 0.7;
#declare B = 0.7;

#declare AMBIENT = 0.5;
#declare DIFFUSE = 1.1;
#declare SPECULAR = 0.3;
#declare ROUGHNESS = 0.01;
#declare BRILLIANCE = 1.0;

camera {
    up <0,0,1>
    sky <0,0,1>
    right <3.0,0,0>
    direction <1.0,0,0>
    location <420,70,70>
    look_at <370,300,90>
}

light_source {<100,100,100> color Gray25 shadowless}
light_source {<400,200,30> color Gray20 }
light_source {<1000,-500,150> color Gray25 }
light_source {<-400,-500,150> color Gray25 }

#declare QCFUNC = function { pattern{
    density_file hdf "supercell.ck10990.hdf", "QC"
    interpolate 2 //tricubic spline
    frequency 0
    scale <scalex,scaley,scalez> } }

#macro QCISOSFC(iso,trans)
isosurface{ function{ -QCFUNC(x,y,z) }
threshold -iso
max_gradient 0.0002
contained_by{box{<x0,y0,z0>,<x1,y1,z1>}}
texture{ pigment{color rgbt<R,G,B,trans>}
finish{ambient AMBIENT diffuse DIFFUSE
specular SPECULAR roughness ROUGHNESS
brilliance BRILLIANCE} }
no_shadow
}
}
#end

QCISOSFC(0.0002,0.0) // render cloud

box { <x0,y0,z0> <x1,y1,z0> // tiles 5km square
pigment {checker color NewTan,
color .90*NewTan scale 50}
finish {ambient 0.5 diffuse 0.5} }

background {SkyBlue} // what else?
```

Continuing through the scene description file, the color and finish parameters are declared, and the camera and lighting parameters are set. The lines that follow contain the important bits for creating the isosurface. QCFUNC is

declared as a function that uses the HDF file `supercell.ck10990.hdf` as a source of data; it chooses the variable `QC` (representing cloud concentration) within the file to render. Tricubic spline interpolation is chosen, and the entire domain is scaled so that all spatial indices, such as camera location and light location, coincide with array index values of the data. By default, POV-Ray's domain ranges from 0.0 to 1.0 in all three directions.

I created a macro called `QCISOSFC`, which takes as arguments the value of the isosurface I wanted to render and the level of transparency of the isosurface. Transparency is a useful isosurface property when rendering two isosurfaces where one exists inside another. For example, it is useful to render a transparent cloud that contains an isosurface of hail concentration, because hail often is contained within a thunderstorm cloud. `QCFUNC`, defined above, is selected as the isosurface function to render. The chosen isosurface binds a volume of cloud concentration greater than the chosen isosurface value of 0.0002.

The `max_gradient` parameter basically tells POV-Ray how much work it needs to do to find the isosurface. Technically, it tells POV-Ray what maximum gradient (largest change over distance) the function representing the isosurface data contains in the vicinity of the chosen isosurface. It is a number that must be chosen carefully. Too low a value produces an isosurface with holes or one that does not render at all; too large a value causes POV-Ray to run for a much longer time than is necessary. Some experimentation is required to find an appropriate value for `max_gradient`. I chose a value of 0.0002, which may seem small; however, cloud concentration ranges from 0.0 to about 0.01. POV-Ray warns you after rendering an isosurface with too large or small a value of `max_gradient` and suggests a value it deems appropriate after rendering.

Making Pictures and More

In order to compile with your changes, you may need to make some minor modifications to `src/Makefile`, which is generated once you run `configure` from the top-level POV-Ray directory. This is the case if you are using external libraries for your history file reading routines or if you've written a separate piece of code to handle file I/O.

Once compiled, you can invoke POV-Ray from the command line. The following command would read from `cloud.pov` and create a 600×400 anti-aliased PPM file, displaying to the screen as it rendered:

```
/home/orf/povray-3.50c-orf/src/povray +D \  
Input_File_Name=cloud Width=600 Height=400 \  
Antialias=on Output_File_Type=P
```

Once your data has rendered successfully with POV-Ray, you have POV-Ray's extensive set of configurable options to choose from to render your scene exactly the way you want. If you have data that changes over time, making animations is straightforward and rewarding. I have written Python scripts to invoke an instance of POV-Ray on each of the processors on my small renderfarm, where each processor works concurrently on a different model time. The resulting PPM files then are joined together to make animations, using mjpegtools. See my research page for some animations. I also have created stereo images and animations for display on our department's GeoWall system. Stereo pair generation is trivial with POV-Ray and truly can give you a whole new perspective on your data. Getting POV-Ray to work with my model data has opened the door to many exciting possibilities for me, and I hope that it will for you, too.

Resources for this article: </article/7754>.

Leigh Orf is an Assistant Professor of Atmospheric Science at Central Michigan University and a long-time Linux user. His research interests include making realistic simulations and visualizations of thunderstorms using massively parallel Linux clusters. When not working, he enjoys brewing his own beer, communicating via ham radio, playing the saxophone and going on camping trips with his wife, Annie. He can be reached at leigh.orf@cmich.edu.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Improving Application Performance on HPC Systems with Process Synchronization

Paul Terry

Amar Shan

Pentti Huttunen

Issue #127, November 2004

Hey, cluster node, don't work on that housekeeping task right now—we're all waiting for you to finish your part of the MPI job! Here's a scheduling policy designed for cluster efficiency.

One would expect that doubling the processing power available to an application would double the application performance or cut the run time in half. Unfortunately, HPC users know this is far from true, with actual performance efficiency dropping to as low as only 5% of a system's theoretical peak performance. HPC researchers and application developers have spent and continue to spend much effort trying to find the source of these performance losses and boost sustained application performance. When we set about developing the Cray XD1 system, we joined the ranks of researchers attacking this problem. This article describes how we learned from those who went before us and how we built on that knowledge to develop a new Linux scheduling-based solution that promises to improve real application performance significantly on Linux HPC systems.

The majority of research has focused on the structure of the HPC applications themselves. Various research teams attempted to improve the efficacy of caching, looked for ways to minimize interprocessor communications and explored a variety of similar measures, but each strategy offered performance gains of only a few percent. Another area of research has shown particular promise, however. By understanding the interaction between the HPC application and the system background processes, one can find ways to modify this interaction to improve performance.

Where Does the Performance Go?

In a seminal paper documenting their research into this interaction, researchers Petrini, Kerbyson and Peking of the Los Alamos National Labs (see the on-line Resources) quantified the loss in application performance caused by what they deemed “noise”—the interaction between large multiprocess MPI jobs and background processes. They observed that housekeeping tasks, or noise, delayed individual processors from reaching MPI barriers (synchronization points in the application) and caused all other processors to wait while one processor finished up its housekeeping. This resulted in wasted cycles on all other processors.

The top half of Figure 1 illustrates this interaction and how it results in lost performance. The processes illustrated are part of a parallel job, each running on a separate processor and periodically synchronizing through the use of MPI barriers. In the first part of the computation, Process 1 is delayed because the node's scheduler pauses process execution to run background processes, such as those found on every Linux or UNIX node. Processes 2 and 3 also are delayed. Repetition of this pattern results in substantially reduced sustained application performance. The magnitude of the impact is a function of the frequency with which barriers are encountered and the number of processors.

Petrini and colleagues quantified this loss of performance running SAGE, a compressible Eulerian hydrodynamics code, on their HPC system, named ASCI Q. ASCI Q is a cluster of 2,048 HP ES45 nodes, where each node is a four-way SMP. Petrini, et al., observed that better performance was obtained when they restricted SAGE to run on only three of the four processors in the SMP when more than 256 nodes were utilized. They theorized that this result was caused by background noise, and the theory was verified by eliminating many of the sources of noise and observing the improvement in performance.

This research points to lack of process synchronization and wait time as the culprit that is robbing fine-grained and highly parallel HPC applications of up to 50% (and perhaps more) of their potential performance. Unfortunately, a means to stop this thievery still was not at hand. The method employed by Petrini, et al., to identify the culprit—restricting the system's freedom to run housekeeping tasks—doesn't present a practical solution for most HPC applications. The prospect of relegating one-quarter of the processors on an HPC system to running housekeeping tasks is not palatable to many HPC sites. In addition, many background processes cannot be removed, limiting the performance gain achievable using this approach.

Recovering the Missing Performance

When we set out to build a new high-performance computer, we also set out to find a way to prevent this performance theft. We considered a new approach using the Linux scheduler to synchronize scheduling of MPI jobs and housekeeping tasks. Previous work and our research suggest that this new synchronized scheduling approach can deliver a 50% or greater performance boost to many fine-grained parallel applications running on 32 or more processors.

Implementing a Synchronized Scheduling Policy

Our approach was to create a new Linux scheduling policy. To achieve the desired gains, this policy must synchronize the schedulers on all nodes in a Linux HPC system to ensure that MPI processes run concurrently on all processes and that Linux housekeeping processes execute at the same time. Thus, the scheduler must have a means to achieve global synchronization, as illustrated in Figure 1. To achieve global synchronization, we designed a feature in the communications processor to synchronize the clock in each processing node.

The new Linux scheduler policy defines a scheduling frame of 128 time slots, 120 of them reserved for application execution and eight for housekeeping processes. Schedulers on different processors are able to align their scheduling frames by exploiting a globally synchronized clock, which guarantees sub-microsecond variation in time between nodes in the system. At any moment in time, all processors either are executing the HPC application or running housekeeping processes (bottom half of Figure 1).

This approach to process synchronization is scalable to high processor counts, because scheduling decisions are made locally on each node. This provides a significant boost to sustained application performance by eliminating wasted CPU cycles caused by waiting at barriers.

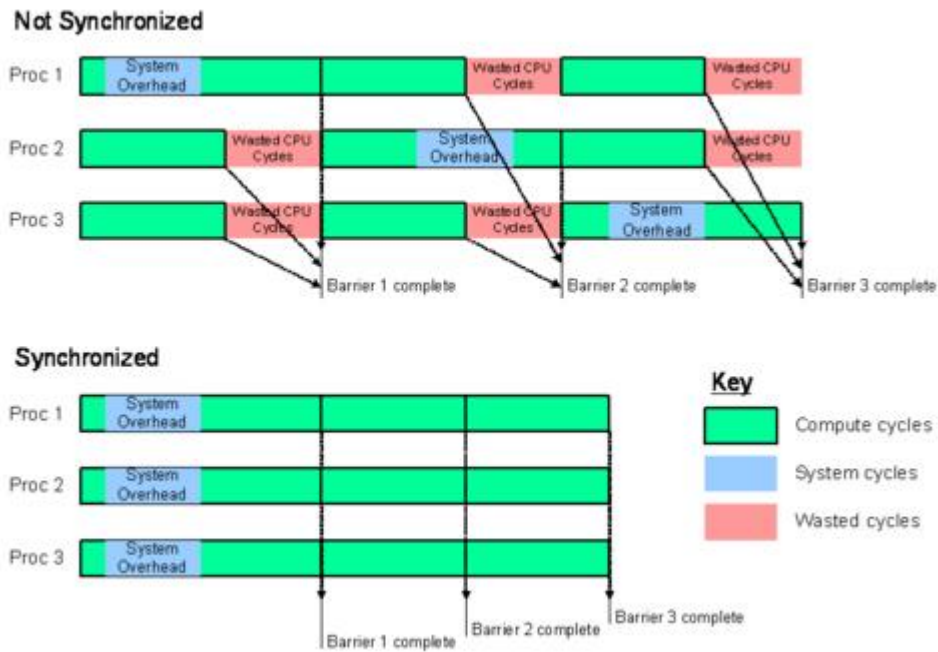


Figure 1. An Example of Asynchronous and Synchronous Execution of Processes

The synchronized scheduler is implemented as a new policy augmenting the three existing policies in the scheduler associated with the Linux kernel. The Linux scheduler is invoked when the process being executed is blocked or voluntarily gives up the CPU, when the processor receives an interrupt or at the end of a 10-millisecond timeslice. The scheduler selects the next process to run based on the scheduling policy applicable to that process and its priority. With the new synchronization policy in place, Linux then selects from one of the following scheduling policies, two for real-time processing and two for conventional time-sharing processes, listed in order of decreasing precedence:

1. FIFO (first in, first out): a process marked FIFO runs until it relinquishes control of the CPU. This priority is used for short duration, real-time system processes. FIFO processes run ahead of others.
2. Round-robin: a process using this policy receives a 10-millisecond timeslice, in turn. It is available for real-time processing.
3. Synchronized: we added the synchronized policy to enable synchronized scheduling of processes in a multiprocessor batch job. The workload management system marks each process as using this policy when it is started. These processes and their offspring gain the benefits of synchronized scheduling.
4. Priority: priority scheduling is the familiar time-sharing mechanism known to Linux users. Processes using this scheduling policy have priorities associated with them and receive time proportional to their priority. All user processes and virtually all system processes run under this policy. The scheduler selects the next process to run from the policy class with the greatest precedence. FIFO and round-robin system processes run

first. Processes marked for synchronized scheduling run before processes using the normal priority scheduler.

The new synchronized scheduling policy creates a scheduling frame that dictates when batch jobs and other user and system processes are executed. The frame includes a predefined number of time slots that are cycled through in sequence. A time slot represents 10 milliseconds (one system timer tick in Linux), during which the process assigned to the time slot is executed. The current implementation has 128 time slots, 120 for the execution of batch jobs and eight for other processes. During the latter time slots, the synchronized scheduling policy indicates there are no runnable batch processes, and the conventional priority scheduling policy takes over for all other housekeeping processes. When no batch jobs exist, the behavior of the Cray scheduler is indistinguishable from the conventional Linux scheduler.

The number of time slots in a scheduling frame is configurable, but it must be a power of two. The ratio of time slots reserved for batch processing versus other processes also may be adjusted. Figure 2 illustrates a typical scheduling frame, with the locations of batch time slots shown in red and housekeeping time slots in grey.

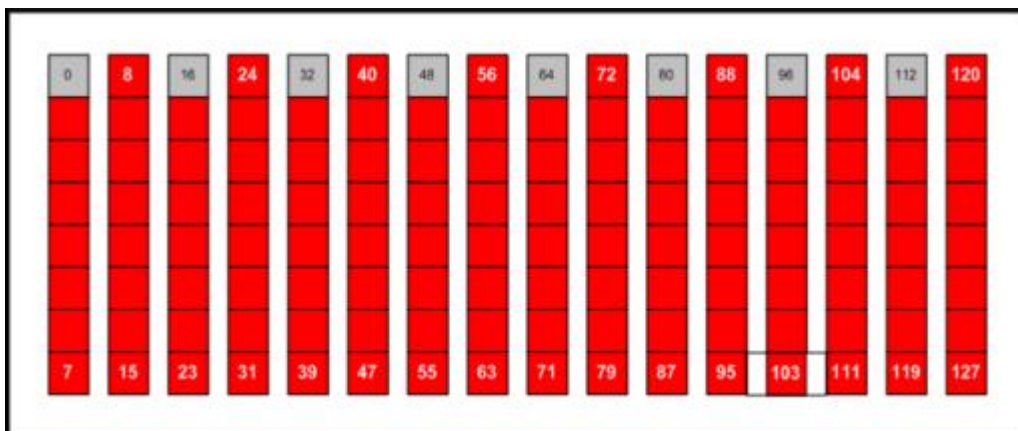


Figure 2. The Time Slots (128) with Eight Reserved Time Slots

A scheduling frame is created when the first batch process is started on a node. All batch time slots are assigned to that process. The creation of additional batch processes results in an even distribution of time slots across processes. If n batch processes are created, the first batch process receives the first $120/n$ time slots, the second receives the next $120/n$ time slots and so forth. The synchronized scheduler thus is able to support batch jobs that require multiple processes on each CPU.

A batch process executes to the end of its allotted time, as long as it makes no blocking or CPU-yielding system calls. If the batch process yields the CPU, perhaps as a result of making a blocking system call, another batch process is scheduled to run. If there are no runnable batch processes, control passes to

the conventional priority scheduler to run housekeeping processes. Of course, batch processes regain the CPU if they are unblocked by the handling of an interrupt.

Alignment of Scheduling Frames between Processors

So far, we have discussed only scheduling of batch jobs and system processes within a single node. However, to stop the performance thievery, this synchronized scheduler must work across all processors. Here, we encounter a critical system design criteria that makes this synchronized scheduler approach possible—the availability of global time synchronization. In our design, global time synchronization is carried out by communications processors designed within the HPC system. These processors offload communications processing from the application processors. Communications processors also run a time synchronization protocol to achieve global clock synchronization. Tight time synchronization can be achieved because the communications processors have control over packet scheduling and jitter—the difference in time between any pair of processors is less than 1 microsecond. A further advantage of delegating time synchronization to the communications processors is this load is removed from the processors carrying the application workload, leaving more time for application processing and further reducing interrupts to the application processors.

The time synchronization protocol includes additional fields for time slot alignment. The protocol uses a master-slave paradigm, where one node acts as the source of the time and time slot information and all other nodes in the system synchronize themselves to the master node's clock. The time synchronization packets received from the master identify the time slot being executed and the time elapsed since the start of the time slot, enabling precise alignment of scheduling frames across the entire HPC system.

Performance Implications

This synchronized scheduler delivers synchronized execution of the processes in a parallel application. How much performance degradation can be avoided or how much potential performance can be gained is a function of how frequently the application uses barriers and/or collective operations, how much time is taken by system housekeeping processes and the number of processors employed by the application.

Our research indicates significant speedup can be achieved. Figures 3 and 4 show the theoretical speedup that can be achieved through the use of the synchronized scheduler, relative to the conventional priority scheduler. Figure 3 assumes that background processing requires 1.5% of the CPU, and Figure 4 assumes that 6.25% of the CPU is consumed by background processing—this is

a realistic metric on most clusters. Curves are shown for applications encountering an average of 100, 200 and 300 barriers per second.

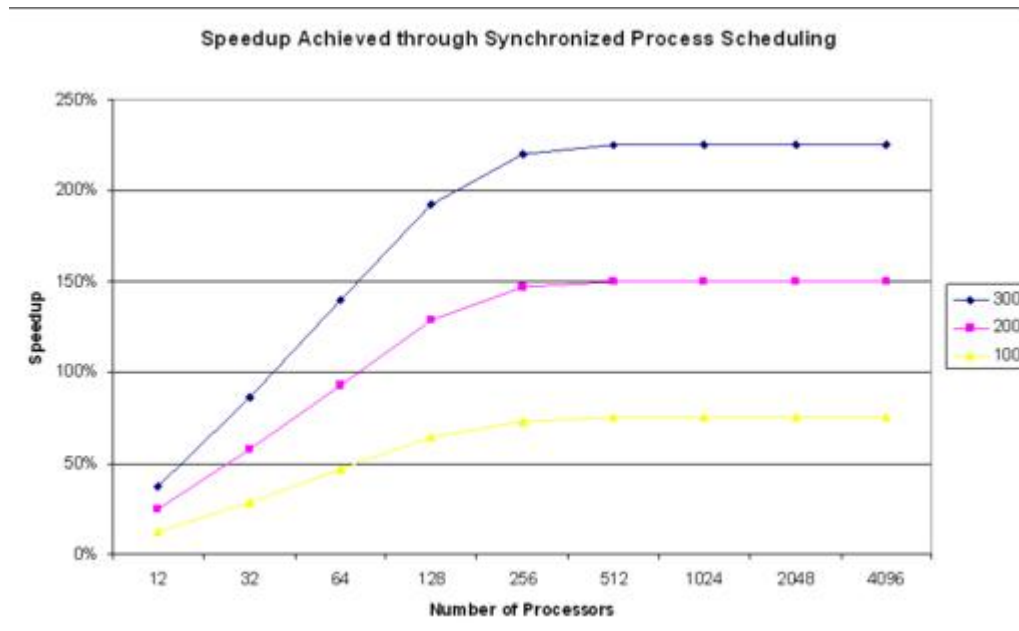


Figure 3. Theoretical Speedups with Process Synchronization with 1.5% Dæmon CPU Utilization

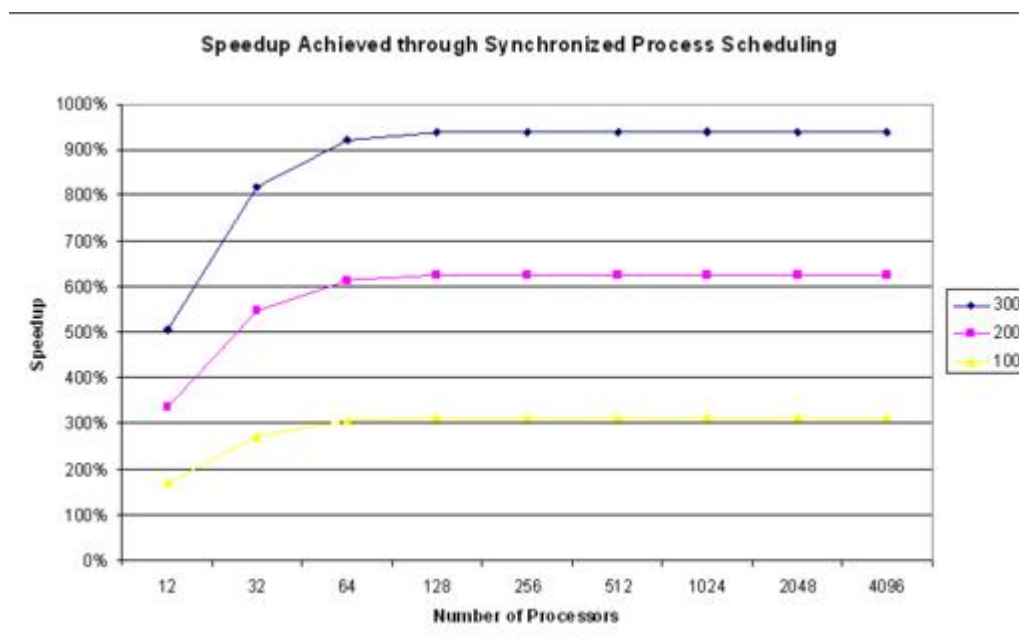


Figure 4. Theoretical Speedups with Process Synchronization with 6.25% Dæmon CPU Utilization

As the number of processors increases, the performance gain from the synchronized scheduler increases and asymptotically approaches a maximum value. This reflects the fact that performance doesn't continue to degrade with the conventional scheduler. After a certain processor count is reached, the probability of at least one processor being delayed by housekeeping increases to 100%. The addition of more processors does not significantly add to the application delay encountered at barriers.

Conclusions

By focusing on the interactions between the HPC application and the system background processes, HPC researchers identified a major culprit for performance losses in parallel applications. Additional research identified ways of preventing this thievery, but none to date have provided successful, real-life implementations. Global process synchronization using the Linux scheduler eliminates wait time due to noise and promises significant performance gains. By looking beyond the application and into the role of the rest of the HPC system, we believe we have found a scalable, real-life implementation. With Linux process synchronization using a global clock synchronization and Linux running on each processing node, the Cray implementation ensures application processes run concurrently on all processors and housekeeping is performed concurrently on all processors and bounded in time. Our process synchronization solution can prevent performance theft and increase application performance for fine-grained highly parallel applications running on 32 processors or more by up to 50%.

Resources for this article: </article/7756>.

Dr Paul Terry is the Chief Technology Officer for Cray Canada, Inc., previously OctigaBay Systems, which was acquired by Cray in April 2004. He is a technology strategist for innovative computing architectures and is responsible for establishing the company's technology vision and leadership.

Amar Shan, Director of Product Management, Cray, Inc., is responsible for introducing Cray's leading-edge technical innovations and creative business solutions into the marketplace. He has more than 20 years' experience in the computing and telecommunications industries in product management, development and architecture roles.

Pentti Huttunen, Benchmarking Specialist at Cray, Inc., is responsible for researching parallel computing technologies and optimizing applications to ensure that they are running efficiently on a variety of platforms at Cray, Inc.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

2004 Readers' Choice Awards

Heather Mead

Issue #127, November 2004

We know the timing of this article puts us at a disadvantage. It's November of a US presidential election year, and it's hard for our Readers' Choice awards to compete. But, we know you've been waiting to find out whether C beat C++ for Favorite Programming Language and where Gentoo fell in the top three in the Favorite Distribution category. It's time for the awards.

Favorite E-Mail Client

1. Mozilla
2. Ximian Evolution
3. KMail

Once again, the top three choices in this category were GUI-based clients. The only difference between this year's top three and last year's is the order: Mozilla jumped to first place from third, Evolution dropped to second and KMail dropped to third. The fifth-place finisher came from the write-in votes—the new Mozilla Thunderbird was only a few votes shy of fourth-place mutt this year.

Favorite Distribution

1. Debian GNU/Linux
2. Mandrakelinux
3. Gentoo

Debian won first place for the second year in a row, picking up almost 300 more votes than second-place Mandrakelinux. Last year's number two, Red Hat, fell to fourth this year, as Gentoo cracked the top three to come in third. The most popular write-in vote was Red Hat's all-free, community-oriented Fedora, coming in at number eight.

Favorite Desktop Workstation

1. Homemade
2. HP xw8200 Linux Workstation
3. Monarch Athlon 64 System Special

Seventy-five percent of voters agreed with the sentiments of the reader who voted for his “bastard child of desire and affordability”, the ever-popular homemade desktop workstation. Like proud parents rattling off the list of talents and skills their offspring possess, voters wrote in the entire list of components in their homemade systems. Other write-in voters echoed the opinion of *LJ* contributing editor Greg Kroah-Hartman, who selected the Apple Power Mac G5, well supported in Linux, as his desktop of choice in this year's Editors' Choice Awards.

Favorite Database

1. MySQL v4.0
2. PostgreSQL
3. Oracle 9i DB

This year's top three favorite databases were a repeat of last year's top three. Although the Editors picked PostgreSQL as their favorite back in August, readers selected MySQL over PostgreSQL by a 2 to 1 ratio. Combined, MySQL and PostgreSQL own 78% of the votes. Add in Oracle 9i and that percentage climbs to 83%. So what else are readers using for their database work? SQLite, GemStone/S and Firebird picked up less than a hundred votes apiece, and Versant was the most popular write-in vote.

Favorite *LJ* Column

1. Cooking with Linux
2. Kernel Korner
3. Paranoid Penguin

Ah, Marcel. Like Susan Sarandon, David Bowie and '02 Bordeaux, Cooking with Linux columnist Marcel Gagné keeps getting better with age. Whether he's showing us a new game, recommending a lovely Australian red or demonstrating a little monitoring GUI, he's always supplying us with useful information in fun ways.

Most Indispensable Linux Book

1. *Linux in a Nutshell, 3rd Edition*, Ellen Siever, et al.

2. *Running Linux, 4th Edition*, Matt Welsh, et al.
3. *Advanced UNIX Programming, 2nd Edition*, Marc Rochkind

I was beginning to think I'd never see a new title in the top three spots in the book category—just advancing edition numbers. But after five-plus years, we finally have a new one. *Advanced UNIX Programming* by Marc J. Rochkind landed in third place this year. Yes, it's a second edition of a book originally published in 1985, but the content is mostly new. The most popular write-in vote continued to be man pages.

Favorite Backup Utility

1. tar
2. Amanda
3. Arkeia Network Backup v5.2

For another year, tar was by far the most popular backup utility, gathering votes from 65% of readers who responded. Amanda came in a distant second, garnering 5% of all votes. On the write-in side, rsync took the top spot. After that, it was a onesy-twosy game of Bacula, personal shell scripts and proprietary offerings. This is weird, though: no one said backups are for wimps.

Favorite Audio Tool

1. xmms
2. mplayer
3. Audacity

A bit of a shake-up in the audio category, as last year's number two and number three tools—noatun and mpg123, respectively—were knocked out by mplayer and Audacity. xmms remains the clear favorite, however, picking up just fewer than 50% of the total votes. On the write-in side, KDE's amaroK came in as the clear favorite, racking up enough votes to tie mpg123 for sixth place overall.

Favorite Desktop Environment

1. KDE
2. GNOME
3. Window Maker

For several years now, KDE and GNOME have finished first and second, respectively, with an ever-increasing distance between the two. This year, KDE received two votes for every one GNOME received. Window Maker holds on to

the number three spot, beating XFce by a single vote. No one said “they all suck” this year, and the only write-in voter who expressed frustration said he might try to write his own environment.

Favorite Linux Web Site

1. LinuxFR
2. Slashdot
3. Freshmeat.net

Last year we noted how close voting was in this category, and this year, for the first time ever, Slashdot isn't the favorite Web site. Granted, only six votes separated LinuxFR from Slashdot, but has Slashdot lost some of its cachet? Or is it simply because even hardware vendor news sounds exciting in French?

Favorite Linux Training

1. Linux Certified, Inc., Linux Systems & Network Administration Class
2. SuSE Linux Training
3. Tie: Novell Certified Linux Engineer and Linux Lunacy Cruise

Judging by the numbers, most of our voters aren't into formal training, preferring instead to use a combination of books, Web resources and, as one voter put it, “hard knocks”. Among those going the more formal route, Linux Certified, which offers classes in San Francisco and Boston, was the favorite.

Favorite Distributed File Sharing System

1. BitTorrent
2. Gnutella
3. Red Hat Global Filesystem

In its first year on our official nominee list, BitTorrent claimed first place with no trouble whatsoever, winning 62% of the votes. Last year's favorite, Gnutella, fell hard to second place, with just less than 11% of the total. eMule and eDonkey grabbed most of the write-in votes, except for the person who believes “communism is wrong”. BitTorrent users can check out legaltorrents.com for many gigabytes of music, books and movies all released under Creative Commons licenses.

Favorite Programming Beverage

1. Coffee
2. Water

3. Tea

Year after year, what people drink while programming is one of the most hotly debated categories. Especially among the write-in responses, our voters are loyal to their beverages beyond belief. Caffeine in all its forms continues its reign, claiming around 85% of all votes. And who drinks Five Alive? I didn't even know they made that anymore!

Favorite Embedded Distribution

1. Qtopia
2. MontaVista Linux
3. BlueCat

In its third year on the ballot, the embedded distribution category continued to gain an increasing amount of total votes. Judging by the write-ins, a lot of embedded Linux work is done using customized or homemade variations. Among the commercial variants, the PDA environment Qtopia remains the favorite with a strong lead, almost twice the number of votes as the second-place MontaVista. The Open Embedded Project also is attracting a number of developers.

Favorite Web-Hosting Service

1. RackSpace Managed Hosting
2. Hurricane Electric Web Hosting
3. ServerBeach

With just less than 20% of all the votes, RackSpace is your favorite Web-hosting service again this year. But most of the votes in this category continue to come in the form of write-ins, almost 64%. So what are you using? OVH, Speakeasy, Amen (a French service) and DreamHost all received several mentions. Overall, Web hosting remains a DIY job among this voting crowd.

Favorite Development Tool

1. GNU Compiler Collection (GCC)
2. Emacs
3. Eclipse

All these vendors are releasing development tools like crazy, but our voters stick with the basics, thanks anyway. GCC and Emacs combine to claim 33% of the total votes. After that, it's Eclipse and KDevelop. Among write-ins, vi, vim and VisualWorks SmallTalk are the most popular.

Favorite Text Editor

1. vim
2. vi and vi clones
3. GNU Emacs

Last year, vim beat vi by almost three times as many votes; this year, it was by only twice as many. Hmm, I wonder what that could mean? Seriously, what's the voting process for if not a chance to develop new conspiracy theories? Coming in at a strong number four is Kate, the KDE Advanced Text Editor. Could readers finally be ready for a modern user interface in an editor instead of Meta-x this and Escape-colon-that? Stay tuned.

Favorite System Administration Tool

1. Webmin
2. YaST
3. KDE Desktop Sharing

Be honest now, you all were most anxious about voting in this category, weren't you? Nothing screams excitement like sysadmin tools. But we're grateful to have them, that's for sure. Webmin collected barely more than 25% of the total votes to take first place. On the write-in side, a collection of text editors and shells claimed most of the votes.

Favorite Server

1. HP Integrity rx4640
2. HP ProLiant DL585
3. SGI Altix 3000

HP claimed first and second place for a combined 35% of all votes in this category. Last year's first-place finisher, the Altix 3000, fell to third this year. The write-in votes featured a lot of Dell and IBM server variations. And, of course, many of you continue to build your own servers.

Favorite Network or Server Appliance

1. Cyclades AlterPath ACS
2. Net Integrator, vMark 1
3. Veritas Storage Foundation, v4.5

Only a few hundred voters expressed a preference in this category. Among those who did vote, the Cyclades AlterPath ACS was the favorite for a second year.

Favorite Portable Workstation

1. Monarch Hornet 64 Custom System
2. Linux Certified LC2430 Linux Laptop
3. EmperorLinux Toucan Laptop, vT42p

Laptops continue to dominate this category, although various Zaurus PDA models made a number of appearances in the write-in section. The big hardware vendors—Dell, IBM, Sony, Toshiba and Apple—ate up most of the write-in votes. HP introduced its Linux laptop too late to catch the voting, so we'll see how they do next time.

Favorite Processor Architecture

1. AMD Athlon
2. Intel Pentium 2, 3 and 4
3. PowerPC

PowerPC and Opteron switched places this year, but only 20 votes separated the two architectures. Meanwhile, Athlon held on to the top spot for another year, having received 40% of the total vote count. Intel Itanium picked up the most write-in votes.

Favorite Office Program

1. OpenOffice.org
2. AbiWord
3. StarOffice

Receiving 72% of the votes, OpenOffice.org is by far this year's favorite office program. In fact, OpenOffice.org received 2,180 more votes than the second-place finisher, AbiWord. It's hard to beat office software that makes more logical sense than its competitors, is compatible with almost everything and, oh yeah, is free.

Favorite Programming Language

1. C
2. Perl
3. C++

Ah, favorite programming language—time for a flame war. A bit of a shake-up this year: after being knocked out of first place last year, C reclaims it this year and C++ drops to third. The P language in the top three is Perl, while PHP slips to fourth place, closely followed by Python. The voting was close this year, too; only 59 votes separated C from C++.

Favorite Instant Messaging Client

1. Gaim
2. Kopete
3. Jabber

Receiving almost 800 more votes than Kopete, Gaim is the clear winner of the favorite instant-messaging client award. Kopete and Jabber switched places since last year, but only seven votes separated them. Quite a few voters wrote that they hate IM. Come on, guys, who has time these days to wait for e-mail?

Favorite Graphics Program

1. The GIMP
2. ImageMagick
3. gqview

It's okay if you want to skip this category; nothing really changes here. The GIMP won almost 70% of the votes...again. Inkscape is the most popular write-in vote this year. A number of you continue to use WINE to access Photoshop and other non-Linux programs.

Favorite Linux Game

1. *Frozen Bubble*
2. *Tux Racer*
3. *Quake 3*

We find it quite interesting that every year one of the categories that receives the most votes is Favorite Linux Game. According to some, games on Linux pale in comparison to what's available for other platforms. But that doesn't stop our voters from wasting hours at the keyboard, does it? *Frozen Bubble* retains its addictive powers for another year and claims first prize.

Favorite Web Browser

1. Mozilla
2. Konqueror

3. Opera

The top two spots are the same this year as they were last year, with Mozilla claiming 50% of all votes. Opera broke the top three this year, sending Galeon down to fifth place. On the write-in side, it's all about the Firefox. To those of you complaining that Firefox is not Mozilla or another Gecko-based browser, your pleas have been duly noted.

Resources for this article: www.linuxjournal.com/article/7757.

Heather Mead is senior editor of *Linux Journal*.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

MyHDL: a Python-Based Hardware Description Language

Jan Decaluwe

Issue #127, November 2004

Hardware design finally enters the 21st century. This new tool brings the readable code of Python and the test discipline of extreme programming to hardware projects.

Digital hardware design typically is done using a specialized language, called a hardware description language (HDL). This approach is based on the idea that hardware design has unique requirements. The mainstream HDLs are Verilog and VHDL.

The MyHDL Project challenges conventional wisdom by making it possible to use Python, a high-level, general-purpose language, for hardware design. This approach lets hardware designers benefit from a well-designed, widely used language and the open-source model behind it.

Concepts

An HDL contains certain concepts that reflect the nature of hardware. The most characteristic concept is a model for massive concurrency. An HDL description consists of a large amount of tiny threads that communicate closely with one another. This design calls for an approach to threading that is as lightweight as possible. HDL descriptions are executed on a dedicated runtime environment called a simulator.

When designing MyHDL, I took a minimalistic approach, which is in line with the Python spirit and a good idea in general. Therefore, an important part of MyHDL is a usage model for Python. The other part consists of a Python package, called `myhdl`, that contains objects that implement HDL concepts. The following Python code imports some MyHDL objects that we are going to use shortly:

```
from myhdl import Signal, Simulation, delay, now
```


MyHDL models concurrency with generator functions, recently introduced in Python (see the on-line Resources). They are similar to classic functions, except they have a nonfatal return capability. When a generator function is called, it returns a generator, which is the object of interest. Generators are resumable and keep state between invocations, making them usable as ultra-lightweight threads.

The following example is a generator function that models a clock generator:

```
def clkgen(clk):
    """ Clock generator.
        clk -- clock signal
    """
    while 1:
        yield delay(10)
        clk.next = not clk
```

This function looks similar to a classic function in Python. Notice that the functional code starts with a while 1 loop; this is the idiomatic way to keep a generator alive forever. The essential difference between a classic and a generator function is the yield statement. It behaves similarly to a return statement, except the generator remains alive after yielding and can be resumed from that point. Moreover, the yield statement returns a delay object. In MyHDL, this mechanism is used to pass control information to the simulator. In this case, the simulator is informed that it should resume the generator after a delay of ten time units.

The parameter clk represents a clock signal. In MyHDL, signals are used for communication among generators. The concept of a signal is inherited from VHDL. A signal is an object with two values: a read-only current value and a next value that can be modified by assigning it to the .next attribute. In the example, the clock signal is toggled by setting its next value to the inverse of its current value.

To simulate the clock generator, we first create a clock signal:

```
clk = Signal(bool(0))
```

The signal clk has a Boolean zero as its initial value. Now, we can create a clock generator by calling the generator function:

```
clkgen_inst = clkgen(clk)
```

To have a minimally useful simulation, let's create another generator that monitors and prints the changes of the clock signal over time:

```
def monitor():
    print "time: clk"
    while 1:
```

```
print "%4d: %s" % (now(), int(clk))
yield clk
```

The `yield clk` statement shows how a generator can wait on a change of the signal value.

In MyHDL, a simulator is created with the `Simulation` object constructor, which takes an arbitrary number of generators as parameters:

```
sim = Simulation(clkGen_inst, monitor())
```

To run the simulator, we call its `run` method:

```
sim.run(50)
```

This runs the simulation for 50 time units. The output is as follows:

```
$ python clkgen.py
time: clk
 0: 0
10: 1
20: 0
30: 1
40: 0
50: 1
```

At this point, we can describe how the simulator works. The simulation algorithm is inspired by VHDL, an HDL slightly less popular than Verilog but a better example to follow. The simulator coordinates the parallel execution of all generators using an event-driven algorithm. The object that a generator yields specifies the event for which it wants to wait before its next invocation.

Suppose that at a given simulation step, some generators become active because the event they were waiting on has occurred. In a first simulation phase, all active generators are run, using current signal values and assigning to next values. In a second phase, the current signal values are updated with the next values. As a result of signal value changes, some generators become active again, and the simulation cycle repeats. This mechanism guarantees determinism, because the order in which the active generators are run is irrelevant for the behavior of the model.

A Real Design Example

Having introduced the concepts, we now are ready to tackle a real design example with MyHDL. I have chosen a serial peripheral interface (SPI) slave hardware module. SPI is a popular synchronous serial control interface originally designed by Motorola.

A single SPI master can control multiple slaves. There are three common I/O ports: `mosi`, the master-out, slave-in serial line; `miso`, the master-in, slave-out serial line; and `sclk`, the serial clock driven by the master. In addition, a slave select line, `ss_n`, exists for each slave. SPI communication always occurs in the two directions simultaneously. In general, the active clock edge that triggers data changes is configurable. In this example, we use the rising edge.

The MyHDL code of the SPI slave is shown in Listing 1. A classic Python function called `SPISlave` is used to model a hardware module. The function has all interface signals as its parameters, and it returns two generators. This code illustrates how hierarchy is modeled in MyHDL: a higher-level function calls lower-level functions and includes the returned generators in its own return value.

Listing 1. MyHDL Model of an SPI Slave

```
from myhdl import Signal, posedge, negedge, intbv

ACTIVE_n, INACTIVE_n = bool(0), bool(1)
IDLE, TRANSFER = bool(0), bool(1)

def toggle(sig):
    sig.next = not sig

def SPISlave(miso, mosi, sclk, ss_n,
             txdata, txrdy, rxdata, rxrdy,
             rst_n, n=8):
    """ SPI Slave model.

    miso -- master in, slave out serial output
    mosi -- master out, slave in serial input
    sclk -- shift clock input
    ss_n -- active low slave select input
    txdata -- n-bit input with data to be transmitted
    txrdy -- toggles when new txdata can be accepted
    rxdata -- n-bit output with data received
    rxrdy -- toggles when new rxdata is available
    rst_n -- active low reset input
    n -- data width parameter

    """

    cnt = Signal(intbv(0, min=0, max=n))

    def RX():
        sreg = intbv(0)[n:]
        while 1:
            yield negedge(sclk)
            if ss_n == ACTIVE_n:
                sreg[n:1] = sreg[n-1:]
                sreg[0] = mosi
                if cnt == n-1:
                    rxdata.next = sreg
                    toggle(rxrdy)

    def TX():
        sreg = intbv(0)[n:]
        state = IDLE
        while 1:
            yield posedge(sclk), negedge(rst_n)
            if rst_n == ACTIVE_n:
                state = IDLE
                cnt.next = 0
            else:
                if state == IDLE:
```

```

        if ss_n == ACTIVE_n:
            sreg[:] = txdata
            toggle(txrdy)
            state = TRANSFER
            cnt.next = 0
        else: # TRANSFER
            sreg[n:1] = sreg[n-1:]
            if cnt == n-2:
                state = IDLE
            cnt.next = (cnt + 1) % n
            miso.next = sreg[n-1]

    return RX(), TX()

```

The module interface contains some additional signals and parameters. `txdata` is the input data word to be transmitted, and `txrdy` toggles when a new word can be accepted. Similarly, `rxdata` contains the received data word, and `rxrdy` toggles when a new word has been received. Finally, there is a reset input, `rst_n`, and a parameter `n` that defines the data word width.

Inside the SPI slave module, we create a signal, `cnt`, to keep track of the serial cycle number. It uses an `intbv` object as its initial value. `intbv` is a hardware-oriented class that works like an integer with bit-level facilities. Python's indexing and slicing interface can be used to access individual bits and slices. Also, an `intbv` object can have a minimum and a maximum value.

The RX generator function describes the receive behavior. Whenever the slave select line `ss_n` is active low, the `mosi` input is shifted to the shift register `sreg`. The `yield negedge(sclk)` statement indicates that the action occurs on the falling clock edge. In the last serial cycle, the shift register is transferred to the `rxdata` output and `rxrdy` toggles.

The TX generator function is slightly more complicated, because it requires a small state machine to control the protocol. The `yield` statement specifies two events in this case, meaning that the generator is resumed on the event that occurs first. When the reset input is active low, `cnt` and `state` are reset. In the other case, the action depends on the state. In the IDLE state, we wait until the select line goes active low before accepting the data word for transmission and going to the TRANSFER state. In the TRANSFER state, the shift register is shifted out serially. The state machine maintains the proper serial cycle count and returns to the IDLE state on the last shift.

Verification

The SPI slave module was modeled at a level that stays close to an actual implementation. This is a good way to introduce MyHDL's concepts. However, using MyHDL for this purpose doesn't provide a lot of advantages over traditional HDLs. Instead, MyHDL's real value is it makes the whole of Python available to hardware designers. Python's expressive power, flexibility and extensive library offer possibilities beyond the scope of traditional HDLs.

One area in which Python-like features are desirable is verification. As with software, in hardware design, verification is the hard part. It generally is acknowledged that traditional HDLs are not up to the task. Consequently, yet another language type has emerged, the hardware verification language (HVL). Once again, MyHDL relies on Python to challenge this trend.

To set up a hardware verification environment, we first create a test bench. This is a hardware module that instantiates the design under test (DUT), together with data generators and checkers. Listing 2 shows a test bench for the SPI slave module. It instantiates the SPI slave module together with an SPI tester module that controls all interface pins. To be able to use multiple SPI tester modules that verify various aspects of the design, the SPI tester module is a parameter of the test bench.

Listing 2. A Test Bench for the SPI Slave Module

```
import unittest
from random import randrange

from myhdl import Signal, intbv, traceSignals

from SPISlave import SPISlave, ACTIVE_n, INACTIVE_n

def TestBench(SPITester, n):

    miso = Signal(bool(0))
    mosi = Signal(bool(0))
    sclk = Signal(bool(0))
    ss_n = Signal(INACTIVE_n)
    txrdy = Signal(bool(0))
    rxrdy = Signal(bool(0))
    rst_n = Signal(INACTIVE_n)
    txdata = Signal(intbv(0)[n:])
    rxdata = Signal(intbv(0)[n:])

    SPISlave_inst = traceSignals(SPISlave,
        miso, mosi, sclk, ss_n,
        txdata, txrdy, rxdata, rxrdy, rst_n, n=n)

    SPITester_inst = SPITester(
        miso, mosi, sclk, ss_n,
        txdata, txrdy, rxdata, rxrdy, rst_n, n=n)

    return SPISlave_inst, SPITester_inst
```

For the tests themselves, we use a unit testing framework. Unit testing is a cornerstone of extreme programming (XP), a modern software development methodology that is an intriguing mixture of common sense and radically new ideas. The genuine XP approach is to develop the test first, before the implementation. XP is a useful methodology, but its lessons virtually are ignored by the hardware design community. With MyHDL, Python's unit testing framework, unittest, can be used for test-driven hardware development.

Listing 3 shows test code for the SPI slave module. Tests are defined in a subclass of the unittest.TestCase class. Each method name with the prefix test corresponds to an actual test, but other methods can be written to support the

tests. A typical test suite consists of multiple tests and test cases, but we describe a single test to demonstrate the idea.

Listing 3. A Test Case for Receiving Data via SPI

```
import unittest
from random import randrange

from myhdl import Simulation, join, delay, \
    intbv, downrange

from SPISlave import SPISlave, ACTIVE_n, INACTIVE_n
from SPISlaveTestBench import TestBench

n = 8
NR_TESTS = 100

class TestSPISlave(unittest.TestCase):

    def RXTester(self, miso, mosi, sclk, ss_n,
                 txdata, txrdy, rxdata, rxrdy,
                 rst_n, n):

        def stimulus(data):
            yield delay(50)
            ss_n.next = ACTIVE_n
            yield delay(10)
            for i in downrange(n):
                sclk.next = 1
                mosi.next = data[i]
                yield delay(10)
                sclk.next = 0
                yield delay(10)
            ss_n.next = INACTIVE_n

        def check(data):
            yield rxrdy
            self.assertEqual(rxdata, data)

        for i in range(NR_TESTS):
            data = intbv(randrange(2**n))
            yield join(stimulus(data), check(data))

    def testRX(self):
        """ Test RX path of SPI Slave """
        sim = Simulation(TestBench(self.RXTester, n))
        sim.run(quiet=1)

if __name__ == '__main__':
    unittest.main()
```

The RXTester method is a generator function designed for a basic test of the SPI slave receive path. It contains a local generator function, stimulus, that transmits a data word on the SPI bus as a master. Another local generator function, check, checks whether the data word is received correctly by the slave. The complete test consists of a number of random data word transfers. For each data word, we create a stimulus and a check generator. To wait for their completion, MyHDL allows us to put them in a yield statement. For proper synchronization, we want to continue only when both generators have completed. This functionality is accomplished by the join function.

When we run the test program, the output indicates which tests fail at what point. When everything works, the output from our small example is as follows:

```
$ python test_SPISlave.py -v
Test RX path of SPI Slave ... ok
-----
Ran 1 test in 0.559s
```

Waveform Viewing

MyHDL supports waveform viewing, a popular way to visualize hardware behavior. In Listing 2, the instantiation of the SPI slave module is wrapped in a call to the function `traceSignals`. As a side effect, signal changes are dumped to a file during simulation, in a standard format. Figure 1 shows a sample of the waveforms rendered by `gtkwave`, an open-source waveform viewer.

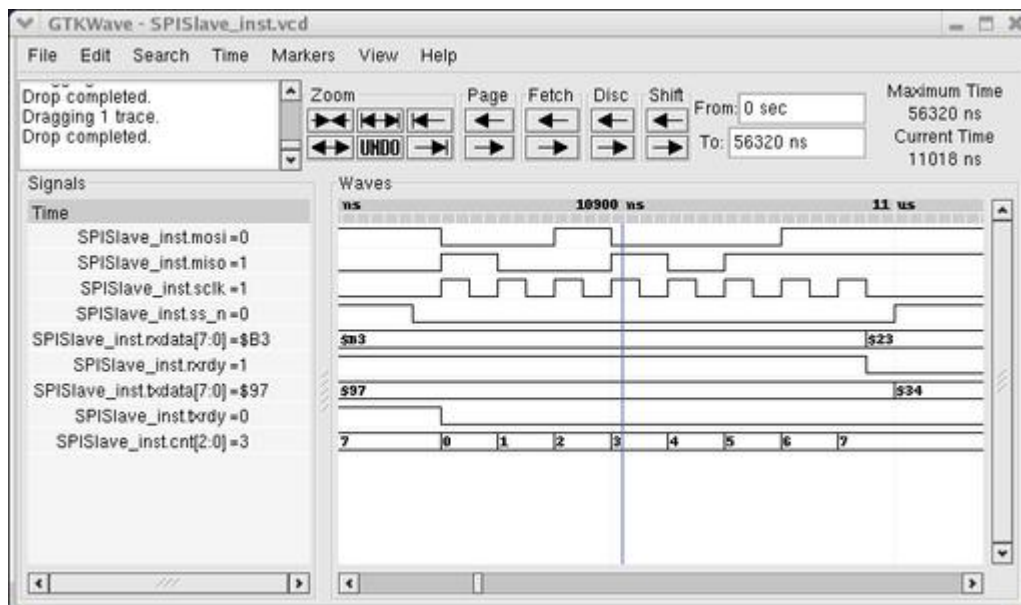


Figure 1. Using `gtkwave`, you can visualize all the signals as the test suite runs.

Links to Other HDLs

MyHDL is a practical solution with links to other HDLs. MyHDL supports co-simulation with other HDL simulators that have an interface to the operating system. A bridge must be implemented for each simulator. This has been done for the open-source Verilog simulators `Icarus` and `cver`.

In addition, an implementation-oriented subset of MyHDL code can be converted automatically into Verilog. This is done with a function called `toVerilog`, which is used in the same way as the `traceSignals` function described earlier. The resulting code can be used in a standard design flow, for example, to synthesize it automatically into an implementation.

Epilogue

Tim Peters, a famous Python guru, explains his love for Python with the paradoxical statement, "Python code is easy to throw away." In the same spirit, MyHDL aims to be the hardware design tool of choice to experiment with new ideas.

Resources for this article: </article/7749>.

Jan Decaluwe has been an ASIC design engineer and entrepreneur for 18 years. Currently, he is an electronic design and automation consultant. He can be reached at jan@jandecaluwe.com.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Revision Control with Arch: Introduction to Arch

Nick Moffitt

Issue #127, November 2004

Whether you're moving up from CVS or just getting serious about a revision control system, here's a powerful tool that will keep records of changes and keep your projects under control.

Arch quickly is becoming one of the most powerful tools in the free software developer's collection. This is the first in a series of three articles that teaches basic use of Arch for distributed development, to manage shared archives and script automated systems around Arch projects.

This article shows you how to get code from a public Arch archive, contribute changesets upstream and make a local branch of a project for disconnected use. In addition, it provides techniques to improve performance of both local and remote archives.

History of Revision Control

Revision control is the business of change management within a project. The ability to examine work done on a project, compare development paths and replicate and undo changes is a fundamental part of free software development. With so many potential contributors and such rapid release of changes, the tools developers use to manipulate these changes have had to evolve quickly.

Early revision control was handled with tape backups. Old versions of a project would be dragged out of backup archives and compared line by line with the new copy. The process of restoring a backup from tape is not quick, so this is not an efficient method by any means.

To work around this lag, many developers kept old copies of files around for comparison, and this was soon integrated into early development tools. File-based revision control, such as that used by the Emacs editor, uses numbered

backup files so you can compare foo.c~7~ with foo.c~8~ to see what changed. Versioned backup files even were integrated into the filesystem on some early proprietary operating systems.

For nearly two decades, the preferred format for third-party contributions to free software projects has been a patch file, sometimes called a diff. Given two files, the diff program generates a listing that highlights the differences between them. To apply the changes specified in the diff output, a user need only run it through the patch program.

In the 1990s, the Concurrent Versions System (CVS) became the default for managing the changes of a core group of developers. CVS stores a list of patches along with attribution information and a changelog. A primitive system of branching and merging allows users to experiment with various lines of development and then fold successful efforts back into the main project.

CVS has its limitations, and they are becoming a burden for many projects. First, it does not store any metadata changes, such as the permissions of a file or the renaming of a file. In addition, check-ins are not grouped together in a set, making it difficult to examine a change that spanned multiple files and directories. Finally, nearly all operations on a remote CVS repository require that a new connection be opened to the server, making it difficult for disconnected use.

Efforts such as the Subversion Project have come a long way toward fixing the flaws found in CVS. Subversion is effectively a CVS++, and it supports file metadata change logging and atomic check-ins. What it still requires is a centralized server on the network that all clients connect to for revision management operations.

Distributed Revision Control Systems

A new generation of revision control systems has sprung up in the past few years, all operating on a distributed model. Distributed revision control systems do away with a single centralized repository in favor of a peer-to-peer architecture. Each developer keeps a repository, and the tools allow easy manipulation of changes between systems over the network.

Projects such as Monotone, DARCS and Arch are finding popularity in a world where free software development happens outside of well-connected universities, and laptops are much more common.

One of the most promising distributed systems today is GNU Arch. Arch handles disconnected use by encouraging users to create archives on their local machines, and it provides powerful tools for manipulating projects

between archives. Arch lacks any sort of dedicated server process and uses a portable subset of filesystem operations to manipulate the archive. Archives are simply directories that can be made available over the network using your preferred remote filesystem protocol. In addition, Arch supports archive access over HTTP, FTP and SFTP.

One advantage to not having a dedicated `dæmon` is that no new code is given privilege on your server machine. Thus, your security concerns are with your SSH `dæmon` or Web server, which most system administrators already are keeping tabs on.

Another advantage is that for most tasks no root privilege is needed to make use of Arch. Developers can begin using it on their own machines and publish archives without even installing Arch on the Web server machine. This affects the pattern of adoption as well. Using CVS or Subversion is a top-down decision made for an entire project team, although Arch can be adopted by one or two developers at a time until everyone in the group is up to speed.

Obtaining tla

Arch was originally a set of shell scripts and wrappers around Tom Lord's hackerlabs libraries. The name of the program in those days was `larch`, and it was more than a little clumsy to use. The client now has been entirely rewritten in C and is called `tla`, which stands for Tom Lord's Arch. The interface is still not perfect, but it is good enough for regular use by a skilled developer. Packages of `tla` are available for most GNU/Linux distributions (see the on-line Resources).

Checking Out a Read-Only Project

Once you have `tla` installed, it's good to test it by checking out some code. Arch stores your data in a directory known as an archive. Within the archive, data is organized into nested categories: projects (the name of the work as a whole), branches (a particular thread of development or other descriptive term) and versions (a simple numerical indicator you can use to indicate how far a specific branch has progressed).

The first step to getting some code is to register a public archive so that Arch associates a name with the archive location:

```
$ tla register-archive http://www.lnx-bbc.org/arch
```

You should now see the `lnx-bbc-devel@zork.net--gar` archive listed when you run `tla archives`. If you're curious about what projects are stored in there, you can use the `tla abrowse` command to get a full list:

```

$ tla abrowse lnx-bbc-devel@zork.net--gar
lnx-bbc-devel@zork.net--gar
lnx-bbc
  lnx-bbc--research
    lnx-bbc--research--0.0
      base-0 .. patch-10

  lnx-bbc--stable
    lnx-bbc--stable--2.1
      base-0 .. patch-29

scripts
  scripts--gargoyle-bin
    scripts--gargoyle-bin--1.0
      base-0 .. patch-7

```

This listing tells us that the `lnx-bbc-devel@zork.net--gar` archive has two projects, `lnx-bbc` and `scripts`. The `lnx-bbc` project has two branches, `research` and `stable`. The `lnx-bbc--research` branch has only one version (0.0) and that version has had ten changes recorded in the archive. The `lnx-bbc--stable` branch has only one version (2.1) with 29 changesets.

Because you now have the LNX-BBC public archive registered in your local listing, you can check out a copy of the LNX-BBC stable branch:

```

$ tla get \
lnx-bbc-devel@zork.net--gar/lnx-bbc--stable lnxbbc

```

Once it finishes downloading and applying patchsets, you should have a directory named `lnxbbc/` that is full of files. To simulate a change in the code, `cd` into `lnxbbc/` and edit `robots.txt` to add a new comment somewhere.

Contributing Changes

Now that you have made a change, running `tla what -changed` should print `M robots.txt` to indicate that `robots.txt` has been modified. To get the details of the change, you can run `tla what -changed --diffs`, which should print out a diff file ready to be sent back to the project's development group:

```

--- orig/robots.txt
+++ mod/robots.txt
@@ -1,3 +1,5 @@
+# Welcome, robots!
+
+ User-agent: *
+ Disallow: /garchive/
+ Disallow: /cgi-bin/

```

The drawback to this is that the diff does not indicate metadata changes. Moved files will not be listed, and new files will not be created when another developer runs this diff through `patch`. In order to submit a more complicated change to the project maintainers, you must generate a changeset.

In Arch, a changeset is represented as a directory tree full of bookkeeping files, patches, new files and removed files. The best contribution technique is to create a changeset directory and then tar it up for delivery:

```
$ tla changes -o ,,new-robot-comment
$ tar czvf my-changes.tar.gz ,,new-robot-comment/
```

Arch ignores files beginning with two commas, an equal sign and a few other special characters. By using a `,,` at the start of our changeset directory name, we avoid the annoyance of Arch complaining that our new directory doesn't exist in the archive. It is probably good practice to use your e-mail address or some other identifier in the tarball filename and changeset directory name.

Keeping Up to Date

Now and then you'll want to download the latest changes to the project. This is as simple as running `tla update` from inside the checked-out copy.

Arch first runs `tla undo` to set aside your local changes before applying new changesets. Once all the patches have been applied, it runs `tla redo` to re-apply your local changes.

All of the `tla` commands introduced above require a functioning network connection to the `lnx-bbc.org` system that hosts the archive. For disconnected use, you need to create a local archive and then make a branch within it.

Setting Up an Archive

Before you can begin working in a read-write archive, you must identify yourself to `tla`:

```
$ tla my-id "J. Random Hacker <jrh@zork.net>"
```

Once you have entered your e-mail address, it is time to create an archive for your projects. Arch lets you make many archives, but you can keep as many projects and branches as you like in the same archive.

Archive names have two parts, separated by two hyphens: the first is your e-mail address, and the second is some identifier. Many people like to use the four-digit year as the identifier and roll over to a new archive each year:

```
$ tla make-archive -l jrh@zork.net--2004 ~/ARCHIVE
$ tla my-default-archive jrh@zork.net--2004
```

The `my-default-archive` command makes certain operations on the local archive easier to type.

Setting Up a Project Branch

Arch encourages developers to fork and merge projects using branches. Branches are the primary mechanism for moving code from one archive to another, even over a network. You can use a branch for a complete code fork to pursue an entirely new line of development, or you can use a branch to cache a copy of a project on your laptop so that you can work for a while in an environment that lacks network access.

Published branches are also the primary development communications mechanism for developers who use Arch. Instead of mailing large changeset tarballs or patch files around, a contributor most likely would set up a branch to make local changes and then invite the upstream developers to merge those changes back into the main project. This is where the decentralized and democratic nature of Arch's design shines. Any developer can join the development effort without needing special privilege in the core team's archive.

Before you can branch the `lnx-bbc` project, you have to set up a space for the project in your archive. The format for a project identifier is similar to that of the archive name: the category (or project name), two dashes, the branch name, two dashes and the version number. It is most likely Tom Lord's experience as a LISP hacker that informed his decision to use these dashes:

```
$ tla archive-setup lnx-bbc--robot-branch--0.0
```

This creates a category called `lnx-bbc`, a branch called `robot-branch` and a version called `0.0`. You did not need to specify `jrj@zork.net--2004/` in front of the project name because that is your default archive.

Tagging Off the Branch

Finally, it is time to tag off the branch from the remote archive. This means the `robot-branch` begins as a tag pointing to a particular revision of a project in the `lnx-bbc-devel@zork.net--gar` archive, and all local changes start from that point:

```
$ tla tag \  
lnx-bbc-devel@zork.net--gar/lnx-bbc--stable--2.1 \  
lnx-bbc--robot-branch--0.0
```

At this point, running `tla abrowse` should show your default archive as follows:


```
jrh@zork.net--2004
lnx-bbc
  lnx-bbc--robot-branch
    lnx-bbc--robot-branch--0.0
      base-0
```

Working with Your New Branch

You are now ready to check out a copy of your new branch:

```
$ tla get lnx-bbc--robot-branch robot-branch
```

At this point, you can go into the robot-branch directory and make some changes:

```
$ chmod 444 index.txt
$ tla mv faq.txt robofaq.txt
$ echo "ROBOT TIME" > robot-time
$ tla add robot-time
$ tla rm ports.txt
```

The `tla mv` command renames a file in such a way that Arch keeps track of the change. It is important to use this command in place of the standard `mv`. The `tla add` command prepares a new file to be inserted into the archive, and `tla rm` schedules removal of a file.

All of these changes can be checked in to your local branch now:

```
$ tla commit
```

Your preferred text editor (as specified in the `$EDITOR` environment variable) will be started up with a template for your check-in log. Once you have filled out the log entry, saving and exiting finalizes the commit.

Now running `tla abrowse` shows that you have two revisions of the robot branch in the archive, `base-0` and `patch-1`:

```
jrh@zork.net--2004
lnx-bbc
  lnx-bbc--robot-branch
    lnx-bbc--robot-branch--0.0
      base-0 .. patch-1
```

Merging Projects from Two Different Archives

Of course, while you work on your branch, development may have continued on the original archive. Running `tla update` fetches changes only from your local branch and not the original project. To fold in changes from upstream, you need to `star -merge`:

```
$ tla star-merge \  
lnx-bbc-devel@zork.net--gar/lnx-bbc--stable--2.1
```

In the event of conflicts (situations where both your branch and the upstream project have changes to the same lines of code), Arch uses the standard patch method of creating .orig and .rej files for each file that has conflicts. It is a good idea to use the find utility to seek out any rejects before committing your star-merge.

Speeding Up Archive Operations

You may have noticed that revisions are named either base-0 or patch-#, where # is the number of patches to base-0 that must be applied. Arch uses a log-structured archive format, so that archive operations only ever add information to a project. This means that for big projects with many revisions, it can take a long time for certain tasks.

To speed up operations, you can make a snapshot of a given revision. Arch snapshots are simply a compressed tarball of a checked-out revision. When a checkout or other operation is performed, Arch looks for the highest-numbered snapshot and applies any necessary patches from there:

```
$ tla cacherev
```

Once this is finished, you can run `tla cachedrevs` to see what revisions have snapshots within your archive:

```
lnx-bbc--robot-branch--0.0--base-0  
lnx-bbc--robot-branch--0.0--patch-1
```

Libraries

Because you do not always have access to create snapshots in an archive, it can be useful to make a local cache to speed up file operations. Arch provides a second kind of cache, called a library, that stores copies of checked-out files from various revisions. This is especially helpful for remote archives, because it means you do not even need to download the base snapshot revision before applying changesets:

```
$ mkdir ~/LIBRARY  
$ tla my-revision-library ~/LIBRARY  
$ tla library-config --greedy ~/LIBRARY  
$ tla library-add \  
lnx-bbc-devel@zork.net--gar/lnx-bbc--stable--2.1
```

This library is not small, with the example above comprising over 78MB as of June 2004. The advantage over a slow link, however, is well worth the trouble. In addition, laptops often have slow ATA hard drives, and involved archive operations can be a drag as the drivers use up plenty of CPU cycles. A greedy (auto-updating) Arch library can make your revision control operations quicker and more responsive, even for local archives.

More to Come

In the next article in this series, you'll learn how to make publicly available mirrors so that upstream developers can star-merge back from your branches. In addition, you'll learn how to cherry-pick changesets from a busy branch and how to use GnuPG to sign your changesets cryptographically for security purposes.

The third and final installment of this series will describe centralized development techniques with Arch. You'll learn how to manage a shared access archive using OpenSSH's SFTP protocol and how to write scripts to perform automated tasks on your archives.

Resources for this article: </article/7752>.

Nick Moffitt is a Linux professional living in the San Francisco Bay Area. He is the build engineer for the LNX-BBC Bootable Business Card distribution of GNU/Linux and the author of the GAR build system. When not hacking, he studies the history of urban public transportation.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Linux and RTAI for Building Automation

Andres Benitez

Vicente Gonzalez

Issue #127, November 2004

This easy-to-deploy Web-based control system uses standard phone wiring and can manage any device that supports an infrared remote control.

This article presents the design and development of a control system for centralized operation of different air-conditioning equipment in a building by using Linux and the real-time Linux application interface (RTAI). Each air conditioner, distributed throughout the building, has its own infrared (IR) remote controller. The goal is to replace them with a centralized computer-based control system to operate the air conditioners, including turning them on or off and setting the desired temperature and fan speed.

The idea for this project came from the need of a local university to have a centralized and flexible way to operate its air conditioners at a cost within its budget. Commercial software and hardware exists for the same purpose, but normally they are too expensive and manufacturer-dependent.

This project's hardware solution consists of a central control computer, running Linux, connected to an RS-485 microcontroller network. The microcontrollers have the capability to send commands to the related air conditioner using infrared signals to operate the nearby equipment.

The software design of the system includes two real-time tasks—a main control task and the RS-485 network control task—as well as two non-real-time tasks—a Web server and a database. The Web server is in charge of the user interface, making it available from any browser in the university's computer network. The PostgreSQL database is used as the main data repository.

The implementation is a low-cost solution with the flexibility to extend it as necessary. Moreover, it is not manufacturer-dependent, and it works with any

air conditioner that supports an IR remote controller. Each air conditioner works independently using its own temperature control system. To supervise the operation of this equipment, each microcontroller in the network is equipped with a temperature sensor to monitor the actual classroom temperature and report it to the central computer.

User Interface

The entire user interface is based on Web pages. The first page displays a summary of the actual state of each air conditioner. This information includes an identification string, the room in the building where it is located, the actual room temperature and whether the equipment has a preprogrammed operation sequence according to which it actually is operating. For each air conditioner, this page has a link to the operation interface for the specific piece of equipment. Before going to this page, the system asks for a user name and password. At this level, it is possible to interact with the system directly controlling the air conditioner or to create or change the actual program for automatic operation (Figure 1).

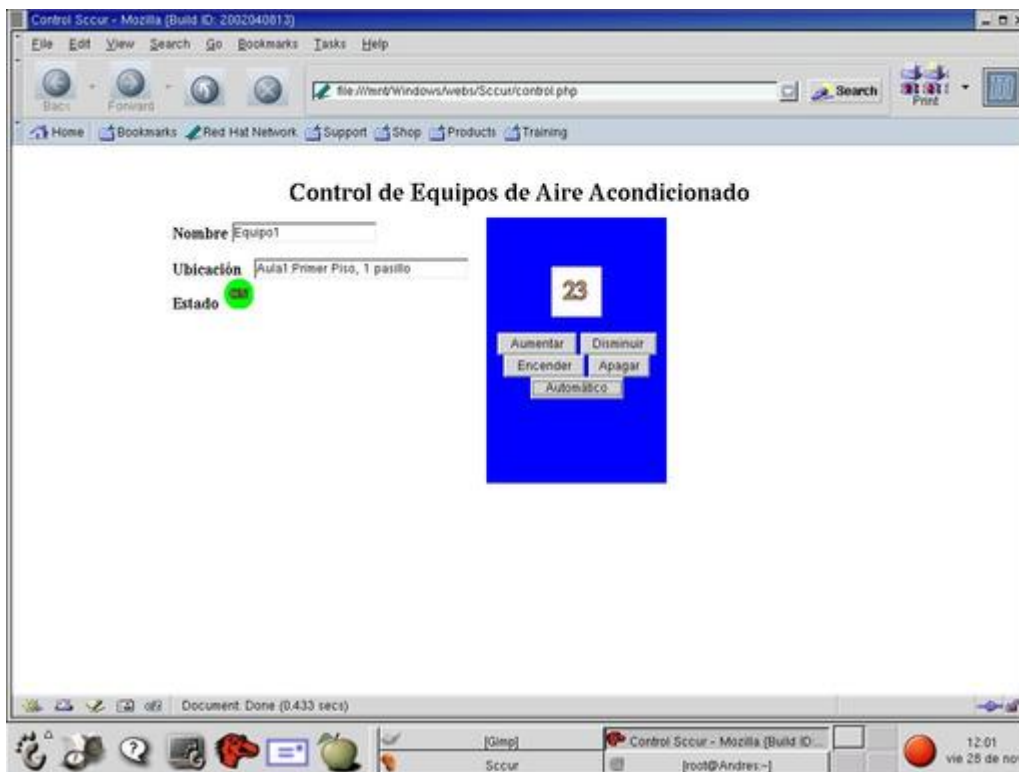


Figure 1. From the operation interface, an authorized user can turn the air conditioner on, set a new temperature or turn it off.

The most interesting part of the system is the programmed operation. For example, every day the system can turn on the air conditioners automatically, with a predefined temperature setting for each room, before classes begin. Then, the system can turn off the air conditioners at a time in the evening when the activities in the building or in a specific classroom end.

Hardware Architecture

The hardware architecture is composed of a central control computer and microcontrollers commanding the air-conditioning equipment. All the microcontrollers are connected to an RS-485 two-wire network. The microcontroller used in this application is the AT89C2051, an Intel 8051 derivative from ATMEL. It is encapsulated in a 20-pin DIP package and is equipped with 128 bytes of data RAM, 2KB of ROM for code, one asynchronous serial port and 14 independent digital I/O ports. Figure 2 shows the microcontroller board and its parts.

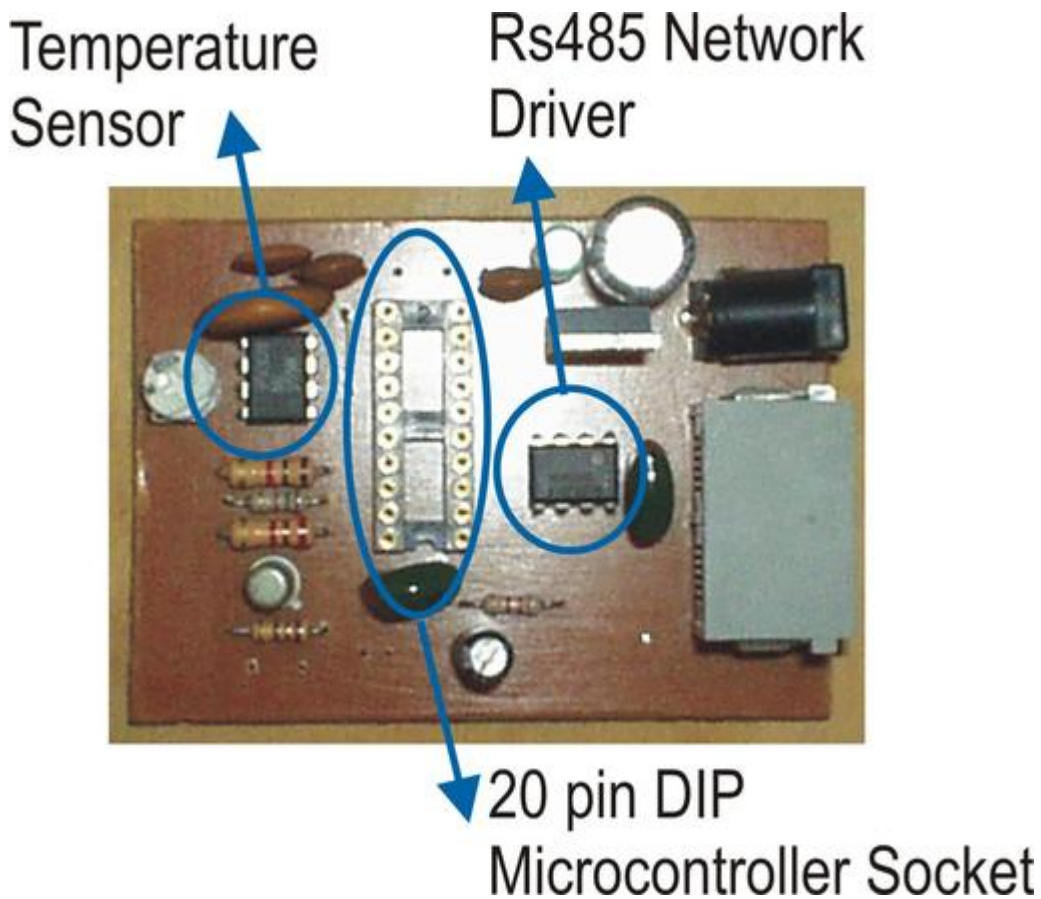


Figure 2. The microcontroller board with the microcontroller removed.

The software in the microcontroller generates the IR signal using one digital output port. The serial port connects the microcontroller to the RS-485 network. Each microcontroller board is equipped with a temperature sensor, the DS1620 from Dallas Semiconductor. The microcontroller communicates with the temperature sensor using a digital three-wire synchronous serial interface. The microcontroller has no hardware support for synchronous serial ports; therefore, it is implemented in software using normal I/O ports.

The RS-485 network is used in this application because it is easy to deploy, cheap to implement and easily can connect a useful number of nodes. Only one pair of Category 3 telephony grade cable connects the nodes. Due to the

hardware driver limitation, the maximum number of allowed nodes is 32, but this number easily can be extended using network repeaters. The maximum cable length between the control computer and the first repeater or between repeaters is 1,200 meters.

A master-slave protocol controls access to the physical cable. The computer running Linux is the master, which polls each node with a predefined rate. On every polling the master can send commands to the node; the polled node answers by sending data to the master or by sending an empty packet to say that the node is active. A drawback of using this access-control protocol occurs if the master goes down—the entire network goes down too.

Considering the limited resources of the microcontrollers, the 9th-bit protocol is used to determine whether the packet sent through the network is for this controller. Each byte transmitted through the network has an additional bit. The packet destination address is the only one transmitted with this additional bit set to high. The microcontroller's UART (universal asynchronous receiver and transmitter) is programmed by default to generate an interrupt only if the 9th bit of the received byte is high. The interrupt service routine then compares the received byte with the node address. If there is a match, the routine programs the UART to receive all the bytes regardless of the 9th-bit state, until the end of the packet. If the destination address does not match this node address, the interrupt service routine returns.

The central control computer UART, which is PC hardware, does not directly support the 9th-bit protocol. To overcome this limitation, the driver simulates it by using the parity bit. Before transmitting a byte, the driver configures the parity to generate a one in the 9th bit of the address byte and a zero in the 9th bit of the other bytes.

Figure 3 shows a diagram of the tasks in the system and the communication links between them.

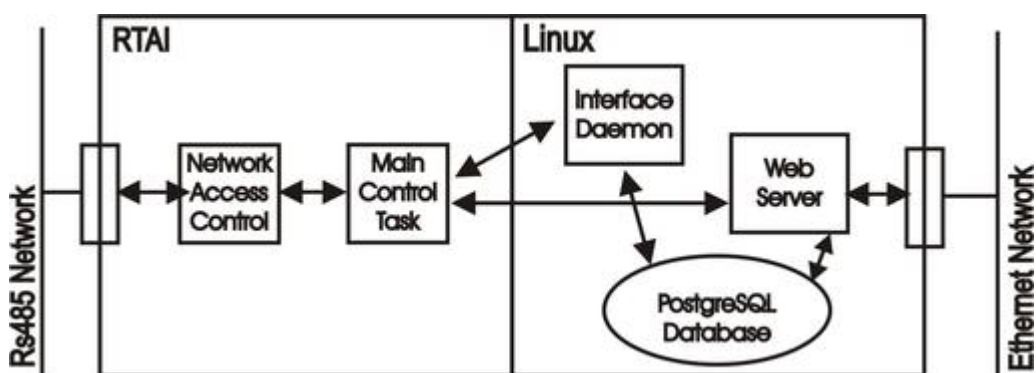


Figure 3. Communication among the Tasks in the System

Real-Time Tasks

The main control task, the network access-control task and the software driver for the physical layer of the RS-485 network are the tasks that run in the real-time executive. An RS-485 driver was developed for RTAI. This driver is similar to any other serial driver, except for the 9th-bit protocol used in this application, as described above.

The other real-time task is the network access-control task, which is in charge of periodically sending packets to each network node. This packet can be a command to generate an IR signal, a poll to see if the node is active or a command to the microcontroller to transmit the actual room temperature. The node answers with an acknowledgement to the first two types of packets and with the actual room temperature to the last one. The information about the actual state of every node is available to the main control task, which in turn informs the user interface if a node fails.

The main control task, using information retrieved from the database, operates the air-conditioning equipment in the building, as programmed. This task also can receive instructions from the user interface that overrides the programmed configuration, using two RT-FIFOs. RT-FIFOs are an interprocess communication routine for communication between real-time tasks and normal Linux tasks. To communicate with the PostgreSQL database, a Linux *dæmon* was developed. This *dæmon* communicates with the main control task using two more RT-FIFOs. An additional important function of this *dæmon* is to send to the main control task the system date and time; no support for reading it exists in RTAI.

The developed system sends commands to the air conditioners, eliminating the need for local remote controllers. We do not interfere with the air-conditioner temperature control system, nor do we touch any internal circuitry. Each air conditioner has its own temperature control system built-in, and the temperature sensor in each microcontroller supervises that the equipment is working fine. Figure 4 shows the microcontroller board installed.

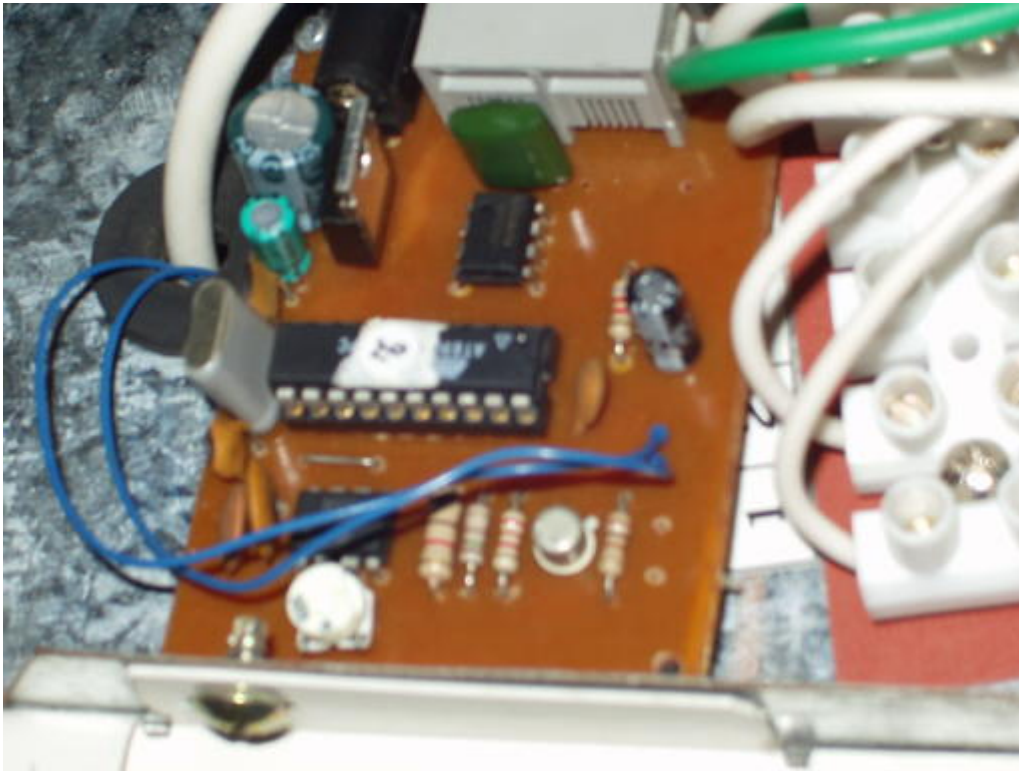


Figure 4. The complete microcontroller board as installed.

Linux Tasks

The Linux tasks are in charge of presenting the user interface through a Web server and running the PostgreSQL database engine, which is the main data repository. As described above, another Linux side task is a daemon used for the RTAI main control task to access the system date/time and the database.

The user interface is simple. The first page presents information about the actual state of each air conditioner. Every type of user can access this page. In order to change the program or send commands to a particular air conditioner, the system asks for a user name and password. PHP is used to generate the Web pages dynamically to present the information retrieved from the database.

In the PostgreSQL database, the system stores general information about the air conditioners, such as BTU, location, brand and microcontroller network node address; the programmed operations; and the IR commands needed to operate each air conditioner.

IRC Command Interface

An important part of the system is the module that reads the air-conditioner remote controller signals and stores the information, associated with the corresponding equipment, in the database to reproduce it using the networked microcontrollers. This module is used only when adding a type of air

conditioner that has a different brand and/or different remote controller commands.

Two tasks are part of this module: the first is a real-time task that reads the IR signal. The LIRC Project as well as the Ripoll and Acosta paper in the on-line Resources, present detailed information about IR remote controllers and sample implementations using normal Linux and RTLinux, another real-time executive for Linux. The other task for this module is the user interface that runs on Linux. The two tasks communicate using an RT-FIFO.

Due to the small amount of RAM available in the microcontroller and the long IR signal duration, an important function of this software is to help the user obtain repetitive patterns within the different IR remote controller signals associated with each button or combination of buttons. These patterns are coded in the firmware of the microcontroller and are used to reconstruct the command to control the equipment. For example, if there are ten different patterns, the information sent to the appropriate microcontroller in the network is something like: repeat pattern one ten times, then pattern two three times and so on, until the complete command is reconstructed. This technique has the advantage of using fewer resources for signal reconstruction. The disadvantage is the software of the microcontroller needs to be changed to introduce the patterns of the newly added equipment whenever a new air conditioner is introduced.

Costs

Currently, nine air conditioners are controlled using the system described here; all of them are located in the same building. Another 15 will be added soon. The hardware cost of each microcontroller node is \$60 US, and the central control computer costs about \$500 US. The other costs are the deployment of the RS-485 network and, of course, the development and implementation of the system.

Conclusion

The implemented system fulfills the specifications of the actual user. Because of this project's success, every new air conditioner acquired by the university must be compatible with the system. The only condition that must be met to comply with the system is every new air conditioner must have an IR remote controller.

Thanks to RTAI, the system main control task is independent of the user interface tasks running in Linux. Even in the improbable situation that Linux goes down, the system would continue on with the programmed operations.

In the future, the system may be extended easily to control, for example, the building lights, alarms, access control to restricted areas and other systems.

Resources for this article: </article/7742>.

Andres Benitez (adorego@conacyt.org.py) is working on his Electronic Engineering degree from the Catholic University in Asuncion, Paraguay. The work described in this article is the final project for that degree.

Vicente Gonzalez (vgonzale@uca.edu.py) is a civil engineer with an MSc degree in Automation from the Polytechnic University of Madrid, Spain. Currently, he is an assistant professor in the department of Electronic and Computer Science Engineering, Catholic University in Asuncion.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

At the Forge

Aggregating with Atom

Reuven M. Lerner

Issue #127, November 2004

Want to give everyone a polite reminder when you have new content on your Web site? Give your site the latest syndication standard and you'll have a new tool to keep visitors coming back.

In the world of organized crime, a syndicate is a collection of gangsters who work together. In the world of newspapers, a syndicate distributes information to subscribers, allowing each publication to tailor the content of information it receives. Comics, news stories and opinion columns often are distributed by syndicates, providing greater exposure for the authors and more content for the readers.

In the past few years, Web developers also have begun to use the term syndicate, as both a verb and a noun. Fortunately for our safety, syndication on the Web has more in common with newspapers than with the mob. But as with organized crime, many people have been hurt in public disputes (albeit with words, not guns), leading to a split and a fair amount of acrimony in the world of Web syndication.

The result of this split is Atom, a new syndication format that has much in common with RSS (rich site summary or RDF site summary, depending on the version and whom you ask). I believe that Atom offers a number of advantages over any version of RSS, and that the simplicity with which Atom feeds can be created makes it an obvious choice over RSS. That said, the fact that most Weblog products provide RSS feeds means that the two camps happily can coexist for now. Understanding how both work also means your organization can decide to adopt one or both standards, depending on your needs.

Some History

As we saw last month, RSS really is two different formats, or more precisely, two different families of formats. RSS 0.9x and RSS 2.0 are from the same family and demonstrate the evolution, over time, of syndication on the Web. RSS 2.0 is maintained mainly by Dave Winer of Userland, scripting.com and (most recently) Harvard University. Winer has given ownership of the standard to Harvard but also has declared that version 2.0 will be the final one. Nevertheless, the combination of RSS 0.9x and RSS 2.0 represents a widespread, stable, well-understood and ambiguous protocol for syndicating Web content.

A separate flavor of RSS, confusingly known as RSS 1.0, uses the resource development framework (RDF) produced by the World Wide Web Consortium (W3C). RDF is designed to make it possible for computers to understand a site's contents, allowing it to make connections between sites, much as people instinctively do all the time. RSS 1.0 produces a summary that is incompatible with all other versions of RSS, using RDF to produce a standardized description of the site's contents.

The fact that RSS 1.0 used the RSS name caused a great deal of friction and animosity, with many people variously blaming Dave Winer, the vagueness of the RSS specification and the proponents of Atom's predecessor. At the end of the day, a number of prominent individuals—led by Tim Bray, Mark Pilgrim and Sam Ruby—were backed by such companies as Six Degrees (which publishes Movable Type software for Weblogs) to produce a specification, initially called PIE and Echo, which attempts to address the shortcomings of RSS.

The development of Atom took some time, because it involved understanding and defining exactly what syndication means on today's World Wide Web. RSS no longer is used only for news sites, its original target, but also for Weblogs and nontextual content. The developers decided to make internationalization a top priority, meaning that it should be possible to produce a syndication feed in any language. Another priority was the development of extensions—that is, it should be possible to add new functionality to the Atom feed without having to redefine the core Atom specification.

As of this writing (mid-August 2004), the Atom specification now exists in version 0.3, along with a standard API for editing content over the network. Atom has begun the process of becoming standardized by the IETF (the Internet Engineering Task Force, which produces and publishes Internet standards), meaning it is on its way to being a universally accepted standard, much like TCP/IP, SMTP or HTTP. This undoubtedly will lead to even greater interest in Atom from organizations that wait for the IETF's stamp of approval.

Atom is still in its initial stages, lacking public specifications for a number of items, such as its extension mechanism. But its authors have, to date, produced a standard whose complexity is fairly close to RSS 0.9x and 2.0, written in as unambiguous a fashion as possible, which includes many members of the Web syndication community and offers a vision of syndication that goes far beyond the Web.

Producing an Atom Feed

Although RSS was designed to summarize a news feed or Weblog, Atom was created with a more general purpose in mind. For example, factory machines could produce status reports in Atom, with an aggregator displaying those that are malfunctioning. Libraries could produce Atom feeds of the latest additions to their collections, with smart aggregators looking for books on certain subjects. Fax machines could be replaced by fax modems, using Atom to distribute fax images to appropriate groups of people.

You even could use Atom feeds to create a newspaper publishing system, where reporters send their stories not as e-mail, but instead publish drafts on an Atom feed. Each editor would aggregate Atom feeds from the reporters under his or her control, moving them onto an outgoing Atom feed when the editing was complete. The final feed would end up in the production department, where the text would be laid out and made ready for actual printing. The newspaper's content flow thus would be a flow of many Atom feeds into a single, final feed representing the newspaper itself.

Producing an Atom feed is fairly simple, if you use Perl or another high-level language for which an Atom library exists. Perl, for example, has the XML::Atom module, available from CPAN (Comprehensive Perl Archive Network). I had a bit of trouble installing XML::Atom on my machine running Fedora Core 2 and Perl 5.8.3, but I was able to work around it by ignoring the optional DateTime module during the installation process. I would not recommend doing so in a production environment.

Although XML::Atom is the overall package name, programs that create Atom feeds actually use XML::Atom::Feed and XML::Atom::Entry. Here is a short Perl program that produces a simple feed, based in part on the sample program in the perldoc on-line documentation for XML::Atom::Feed:

```
#!/usr/bin/perl

use strict;
use diagnostics;
use warnings;

use XML::Atom::Feed;
use XML::Atom::Entry;

# Create a new Atom feed
```

```

my $feed = XML::Atom::Feed->new;
$feed->title('My Weblog');

my $entry;
# Create a first entry for the feed
$entry = XML::Atom::Entry->new;
$entry->title('First Post');
$entry->content('First Post Body');
$feed->add_entry($entry);

# Create a second entry for the feed
$entry = XML::Atom::Entry->new;
$entry->title('Second Post');
$entry->content('Second Post Body');
$feed->add_entry($entry);

# Now produce the XML output
my $atom_feed_xml = $feed->as_xml;

# Display the XML output
print $atom_feed_xml, "\n";

```

The above program produces the following feed, which I have formatted with extra whitespace for easier reading:

```

<?xml version="1.0"?>
<feed xmlns="http://purl.org/atom/ns#">
  <title>
    My Weblog
  </title>
  <entry >
    <title>
      First Post
    </title>
    <content mode="xml">
      <default:div xmlns="http://www.w3.org/1999/xhtml">
        First Post Body
      </default:div>
    </content>
  </entry>
  <entry >
    <title>
      Second Post
    </title>
    <content mode="xml">
      <default:div xmlns="http://www.w3.org/1999/xhtml">
        Second Post Body
      </default:div>
    </content>
  </entry>
</feed>

```

As you can see, we create a single XML::Atom::Feed object, containing one or more instances of XML::Atom::Entry. Each entry object corresponds to a single <entry> tag in the Atom feed, which in turn represents a single entry in our Weblog or a single message from our factory floor.

The Atom specification indicates that the feed may contain a number of attributes and sub-elements, including a language, a description of the Weblog or site, copyright information and other general information about the originating site. Each entry, in turn, has its own set of elements, such as a title, an indication of when it was created and a summary. Each Atom element also

has a MIME type indicating what type of content it contains, much like HTTP responses and e-mail attachments.

Of course, creating a feed, as in the above example, is necessary only if you are writing a new Atom-powered application or if you are adding Atom capabilities to a Weblog product. Most Weblog products now provide Atom feeds, either as part of their standard distribution or through a plugin or other extension mechanism. For example, an Atom feed plugin for the Blossom Weblog product makes it easy to add such a feed from a Weblog; install the plugin (by placing it in the plugins directory), and anyone interested in receiving an Atom feed from the Weblog in question will be able to do so.

It shouldn't come as a surprise that this is so easy to accomplish, given the fact that Blossom is written in Perl, that Perl provides excellent tools for working with XML and that the plugin simply needs to summarize and rewrite content from the most recent entries in the Weblog. Because Blossom makes it so easy for plugins to modify the main page (so as to advertise the Atom feed) and to retrieve content (through the plugin API), it might be slightly easier to work with Atom from that product. Given that most Weblog products are written in a high-level language, such as Perl, Python or PHP, it should be easy to add an Atom feed where none currently exists.

Parsing an Atom Feed

To parse an Atom feed, either because we are writing an aggregator or because we want to create an Atom-powered application, we have several options. The easiest way is to continue to use XML::Atom::Feed to discover and retrieve feeds, for example:

```
#!/usr/bin/perl

use strict;
use diagnostics;
use warnings;

use XML::Atom::Feed;

# Get the Atom feeds for www.diveintomark.org
my @uris =
    XML::Atom::Feed->find_feeds(
        "http://www.diveintomark.org/");

# Print each Atom feed URI
foreach my $uri (@uris)
{
    print "uri = '$uri'\n";
}
```

In the above example, we see a single URI printed. Now that we know where the feed is, we can get a list of links in it, turning those links into XML:

```
#!/usr/bin/perl
```

```

use strict;
use diagnostics;
use warnings;

use XML::Atom::Feed;

# Get an Atom feed
my @uris = XML::Atom::Feed->find_feeds("http://www.diveintomark.org/");

foreach my $uri (@uris)
{
my $feed = XML::Atom::Feed->new(URI->new($uri));

my @links = $feed->link();

foreach my $link (@links)
{
    my $link_xml = $link->as_xml();
    print "link = '$link_xml\n";
}
}

```

Of course, we don't have to produce or display XML; we can parse the link information, sending new links to subscribers by e-mail, adding them to a database or ignoring those that fail to meet certain criteria.

Because Atom feeds are so regular, and because they operate using Internet standards such as XML, Unicode and MIME, we can be confident that the content our feed parses can be handled in straightforward ways. We can farm out different content types to different handlers, parse them in different ways and even (as in the newspaper example above) place them onto new feeds, becoming a super-aggregator.

If you are interested in creating an aggregator or in understanding how to work with the different myriad versions of RSS and Atom, it also is worth looking at Mark Pilgrim's feed aggregator. Written in Python and constantly updated, this is probably the best-documented piece of open-source engine for working with syndication feeds.

RSS or Atom?

So, should your Web site (or Weblog) provide syndication feeds in RSS, in Atom or in both? It is clear to me that Atom is the best of the two (or three) syndication format families produced to date. Dave Winer's RSS formats were groundbreaking when they were released, but they have too many problems to form the basis of full-fledged, enterprise-ready standards. We have seen the agony that results from half-baked standards, such as early versions of HTML and JavaScript, and given that syndication stands a good chance of becoming an important communication mechanism, completeness and unambiguity are important factors to consider.

It is similarly important to consider the growing international use of the Internet and that people want to syndicate media other than text. Atom's lack

of ambiguity regarding special characters is another big step forward, ensuring that we can include < and > in our Weblog entries without having to worry about the implications for syndication. Most important, the planned provisions for extensions will make it possible for Atom to meet the needs of specific groups and applications without opening the entire specification anew.

Although Atom is remarkably complete, it is also straightforward to use. A great deal of time and energy clearly have been put into making Atom as easy to use as possible. Creating a new API is not a simple task, particularly when it is meant to be as general as possible.

Finally, the mess of RSS version numbers that resulted in (and from) petty and political arguments has served no one very well. Because Atom has a different name, although literally an issue of semantics, it reduces the confusion that developers and users alike face when working with RSS.

Conclusion

Atom is an attempt to solve many of the problems associated with RSS and to turn syndication into a building block for new types of high-level communication across Internet applications. Atom is slightly more complicated than Dave Winer's versions of RSS, but it is less complicated (in its initial version) than RSS 1.0, which used RDF to describe and summarize Web sites. The combination of easy-to-use software tools for working with Atom feeds, its extensibility and the authors' commitment to being a part of the Internet standards community, makes it clear that Atom will play a key role in the future of Web communication.

Resources for this article: </article/7751>.

Reuven M. Lerner, a longtime Web/database consultant and developer, now is a graduate student in the Learning Sciences program at Northwestern University. His Weblog is at altneuland.lerner.co.il, and you can reach him at reuven@lerner.co.il.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Kernel Korner

AEM: a Scalable and Native Event Mechanism for Linux

Frédéric Rossi

Issue #127, November 2004

Give your application the ability to register callbacks with the kernel.

In a previous article ["An Event Mechanism for Linux", *LJ*, July 2003], we introduced the necessity for Linux to adopt a native and generic event mechanism in the context of telecom. Many existing solutions attempting to increase the capabilities of Linux have failed thus far. Others did not reach our level of satisfaction, because carrier-grade platforms have different levels of real-time requirements. In order to succeed, such an event mechanism must be bound tightly to the host operating system and take advantage of its capabilities in order to deliver better performance.

At the Open Systems Lab (Ericsson Research) in Montréal, Canada, we started a project in 2001 to develop a generic solution, the Asynchronous Event Mechanism (AEM). AEM allows an application to define and register callback functions for some specific events and lets the operating system execute these routines asynchronously when the events have been activated.

AEM provides an event-driven methodology of development. This is achieved through the definition of a natural user interface in which event handlers contain in their parameter lists all of the data necessary for their execution sent directly by the kernel.

AEM also is motivated by the fact that complex distributed applications based on multithreaded architectures have proven difficult to develop and port from one platform to another because of the management layer. The objective of AEM is not only to reduce the software's development time, but also to simplify source code generation in order to increase portability between different platforms and to increase the software's life cycle.

The biggest challenge of this project was designing and developing a flexible framework such that adding or updating a running system with new event-handling implementations is possible. The constraint was to be able to carry out system maintenance without rebooting the system. The modular architecture of AEM offers such capabilities.

AEM is a complementary solution to other existing notification mechanisms. One of its great benefits is the possibility to mix event-driven code and other sequential codes.

AEM: Architecture Overview

AEM is composed of one core module and a set of loadable kernel modules providing some specific event service to applications, including soft timers and asynchronous socket interfaces for TCP/IP (Figure 1). This flexible architecture permits AEM capabilities to be extended at will.

There is no restriction on what a module can implement, because each exports a range of independent pseudo-system calls to applications. In fact, this allows two different modules to make available the same functionality at the same time. Interestingly, this offers the possibility of loading a new module to implement an improved revision without breaking other applications—they continue to use the older version. This design provides the ability to load the necessary AEM modules depending on the applications' need or to upgrade modules at runtime.

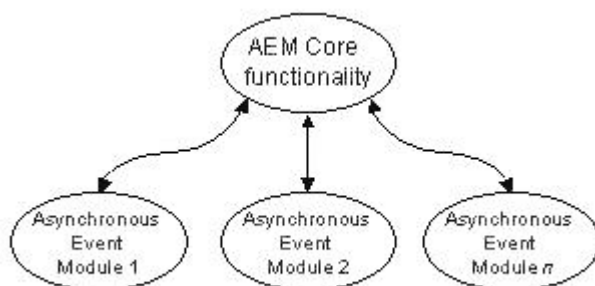


Figure 1. AEM is based on one core kernel module providing the basic event functionalities and a set of independent kernel modules providing asynchronous event services to applications.

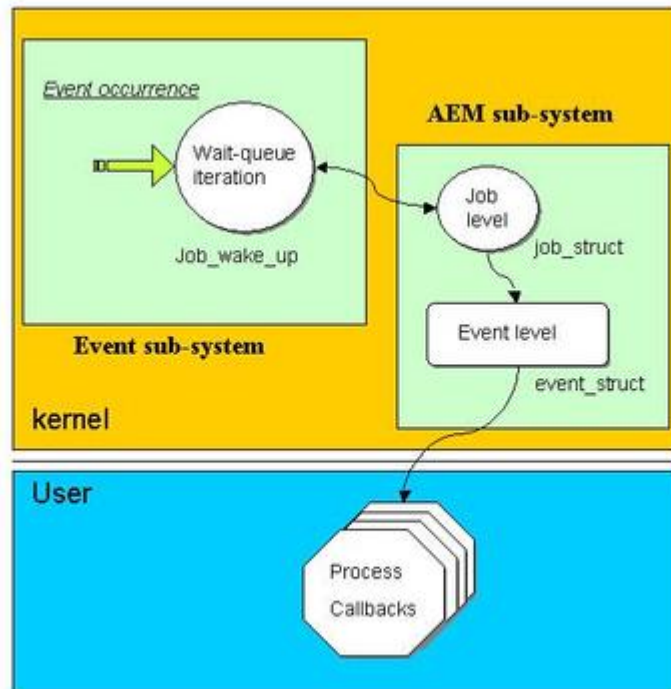


Figure 2. Architecture behind event activation and process notification in AEM. Event wait queues containing sleeping jobs are scanned, and all concerned jobs wake up at the occurrence of an event. It immediately follows the activation of the corresponding event for each related process.

The condition for such flexibility is the presence of event activation points located at strategic places in the kernel (Figure 2). Each activation point is a specific AEM queue used to activate events. In the following sections, we describe the internals of AEM in detail.

AEM: Internals Overview

The concept of asynchrony is a major problem when the main flow of a program's execution is broken without warning in order to execute event handlers. Input requests then are handled without the knowledge of previous input states. These are delivered directly by the core kernel or the interrupt handlers and are received without presuming any kind of order. This situation constitutes a problem for some applications, including those based on TCP/IP, which rely on the transaction state to proceed.

AEM is a three-layer architecture composed of a set of pseudo-system calls for event management, a per-process `event_struct` performing event serialization and storing context information in order to execute user callbacks and a per-event `job_struct` performing event activation.

Events

From the AEM perspective, an event is a system stimulus that initiates the creation of an execution agent, the event handler. Support is provided by the `event_struct`, which is a structure initialized during event registration that contains the context necessary to execute one event handler. Some of the main fields are address of a user-space event handler, constructors and destructors for the event handler and relationship with other events (list of events, child events and active events—see Figure 3).

There can be as many registered events per process as necessary. When an event is detected, we say it is activated, and the user-defined callback function soon is executed. Events are linked internally in the same order as they arrive into the system. It then is up to each handler constructor to manage the data correctly and keep the events serialized for each process without presuming any order of arrival.

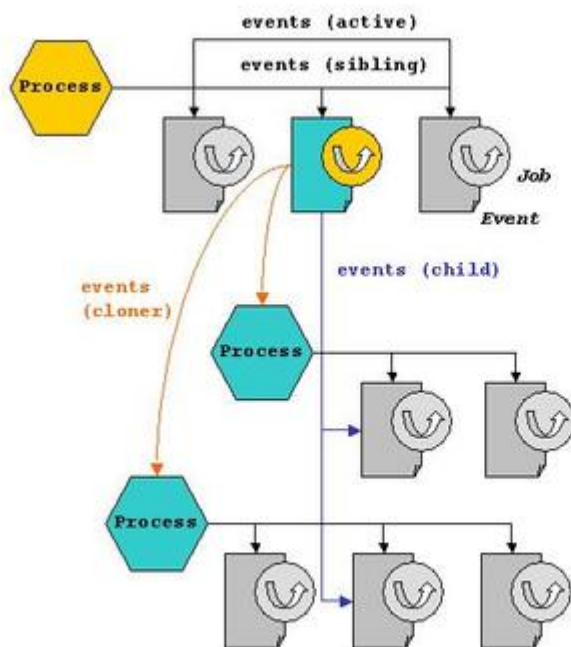


Figure 3. The Relationship between a Process and Its List of Events.

Some process events are active and linked to an active events list. Upon activation, an event can create a process. These events are called cloners, and the relationships between these events and created processes are recorded internally. An event registered by the top process in Figure 3 has created two new processes below it. They remain attached to this event and keep their own list of events.

Event handlers are used during event registrations and must be implemented at the user level. They define their own fixed set of parameters in order to provide event data completion directly to the user-space process. This operation is done by event constructors and destructors executed right before and right after handlers are called. Event handlers are executed in the same context as the calling process. The mechanism is safe and re-entrant; the current flow of execution is saved and then restored to its state prior to the interruption.

A priority is associated with each event during registration that represents the speed at which an application is interested to receive notification. It is possible to register twice for the same event with two different priorities.

Other real-time notification mechanisms, such as the real-time extension of POSIX signals, do not consider priorities during the scheduling decision. This is important, because it allows a process receiving a high-priority event to be scheduled before other processes. In AEM, the occurrence of an event pushes the event handler to be executed depending on its priority. To some extent, an event handler is a process, because it has an execution context. Changing process priorities dynamically is a real issue when the rate of event arrival is high, because priorities are updated quickly at the same rate. We solved this problem by introducing a dynamic soft real-time value calculated using a composition of event priorities. This value influences the scheduling decision without affecting the Linux scheduler and brings soft real-time responsiveness to applications.

Jobs

A job is a new kernel abstraction introduced to serve events before notifying processes. It is not a process, although both share the same conceptual idea of executable entity. One typical action performed by a job is to insert itself into a wait queue and stay there until something wakes it up. At that point, it quickly performs some useful work, such as checking for data validity or availability before activating the user event, and goes back to sleep. A job also guarantees that while it is accessing some resource, no other job can access it. Several jobs can be associated with one process, but there is only one job per event.

This abstraction layer between the kernel and the user process is necessary. Otherwise, it is difficult to ensure consistency in checking for data availability or agglomerating multiple occurrences of the same event when the handler is executed. If something goes wrong, the process wastes time handling the event in user space. Deciding whether to concatenate several notifications is event-specific and should be resolved before event activation.

A generic implementation of jobs would consider software interrupts in order to have a short latency between the time an event occurred and the time the process is notified. The goal is to execute on behalf of processes and provide the same capabilities as both an interrupt handler and a kernel thread, without dragging along a complete execution context.

Two types of jobs are implemented, periodic jobs and reactive jobs. Periodic jobs are executed at regular intervals, and reactive jobs are executed sporadically, upon reception of an event. Jobs are scheduled by their own off-line scheduler. According to the real-time theory of scheduling, both types of jobs could be managed by the same scheduler (see the Jeffay et al. paper in the on-line Resources). In our context, a job is a nonpreemptive task. By definition, jobs have no specific deadlines, although their execution time should be bounded implicitly because of their low-level nature. This assumption simplifies the implementation. The constraint in our case is for reactive jobs to be able to execute with a negligible time interval between two invocations so as to satisfy streaming transfer situations.

Our implementation of periodic and reactive jobs is different in both cases in order to obtain a better throughput in case of sporadic events. A job scheduler and a dispatcher handle periodic jobs, whereas reactive jobs change state themselves for performance reasons. Figure 4 describes the job state evolution and functions used to move from one state to another.

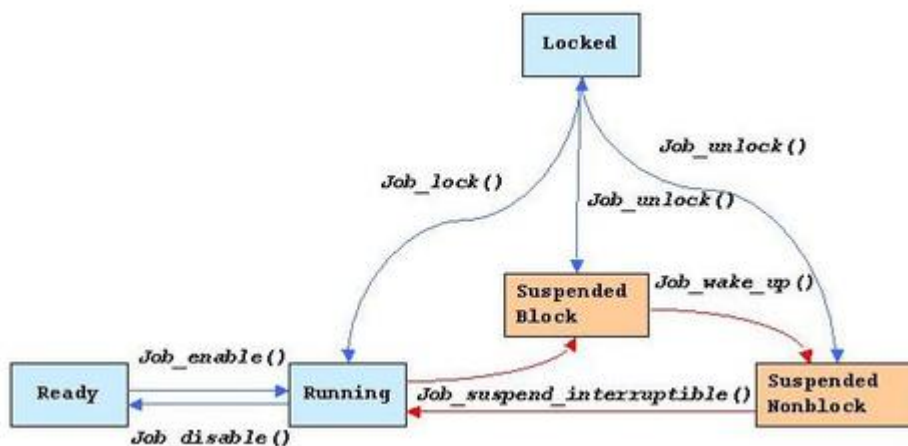


Figure 4. The State Transition Graph for Periodic Jobs and Reactive Jobs

Once a job has activated the corresponding event, either a process is executed asynchronously or the current flow of execution of the user program is redirected somewhere else in the code. The user decides how to handle events at registration time.

Asynchronous Execution of Processes

Event handlers can be executed either by breaking the main thread's flow of execution or by creating a new process to handle the event and execute it in parallel. In either case, the event is managed transparently and asynchronously without explicitly polling for its occurrences. This management has some important implications, because an application does not have to reserve and consume system resources before it is needed. For example, we adapted a simple HTTP server for benchmarking AEM, which is fully single-threaded and capable of quite good performance for that type of server. It is described at the end of this article.

Sometimes situations arise in which it is necessary to create a new process as a response to an event. Unfortunately, creating processes dynamically is resource consuming; during that period, neither the new nor the parent process is able to handle new requests. In some critical situations, there might be no resource left for that purpose, such as a shortage of system memory. This is a problem because emergencies might require creating new processes to shut down the system gracefully or to carry out handover procedures.

For this reason, we introduced a new concept called capsules. A capsule is a task structure already initialized and part of a pool containing other free capsules. When a process wants to create a new execution context, a capsule is linked out from that pool and is initialized with only a few of the current process parameters.

During event registration, specific flags indicate whether a handler is to be executed inside a process. No parameter or a 0 means the handler is to be executed by breaking the current flow of execution. These flags are:

- `EFV_FORK`: to create a process with the same semantic as `fork()`.
- `EVF_CAPSULE`: to create a process from the capsule pool.
- `EVF_NOCLDWAIT`: this flag has the same semantics as the signal `SIG_NOCLDWAIT`. When the child process exists, it is re-parented to the capsule manager thread.
- `EVF_KEEPLIVE`: to prevent the process/capsule from exiting by entering into a kernel loop—like `while(1)`; in user space.

Memory Management

Memory management is a major issue for an event-driven system, because events are handled by executing callback functions located in the application space directly from the kernel. The simplest solution would be to allocate memory when the process registers for an event. Consider, though, either the case of a huge number of events to register or the case of a process that needs

to be restarted, needs new events to be added or needs events to be unregistered. In case of failure in event management, the system integrity becomes inconsistent. It is less critical for the operating system kernel to manage such resources itself and allocate memory on demand on behalf of processes.

Regarding performance, it is necessary to be able to allocate this memory inside a pre-allocated pool to prevent memory fragmentation and maintain soft real-time characteristics. Generic memory allocators, such as glibc's `malloc()`, are not aligned with this requirement even if they provide good performance for general purposes.

Some data types, such as integers, are passed to user space easily, but more complex types, such as character strings, require specific implementations. Process memory allocation is managed by the glibc library. This becomes complicated if we want to allocate memory from the kernel, because we have to take care that this new address is located correctly or mapped into the process space. Allocating memory efficiently and simply on behalf of user processes is something currently missing in the Linux kernel but needed.

AEM's `vmtable` is filling the gap in this area of memory allocation. It implements a variation of the binary buddy allocator—covered in Knuth, see Resources—on top of a user process memory pool. This permits the management of almost all kinds of data of unplanned sizes. In the case of a real shortage of memory, it can fall back on the user decision by using event handlers. This feature offers the possibility of relying on glibc as a last resort if something bad happens.

AEM has been designed to return a valid pointer in constant time when free blocks are available. This often is the case for telecom applications, which are most likely to receive requests of the same size for the same type of application. We also want to prevent memory fragmentation caused by a flood of requests of different sizes in a short time interval.

`vmtable` also has an interesting extension: it can be used to provide user-space initialization procedures for device drivers. This is possible using a pointer, allocated from the kernel by the AEM subsystem and then passed up to the user process in a callback function. It then is given back to the kernel when the function returns. Callback functions are not only usable as a response to an event, but also as a means to communicate safely with the kernel.

In this scenario, each user process is assigned a pool of memory called a `vmtable`. Forked processes and capsules automatically inherit a `vmtable`,

depending on their parents' vmtable. Two different types of strategies are implemented:

1. VMT_UZONE: allocation is done inside process' heap segment. This provides fast access but consumes the user process address space.
2. VMT_VZONE: allocation is done in the kernel address space and mapped inside process' address space. This provides minimal memory consumption, but accesses take more time due to the page faults handling.

Both strategies carry some advantages and disadvantages, depending on the situation. The preferred strategy is chosen at runtime when starting the application. Figure 5 illustrates the architecture layout of vmtable.

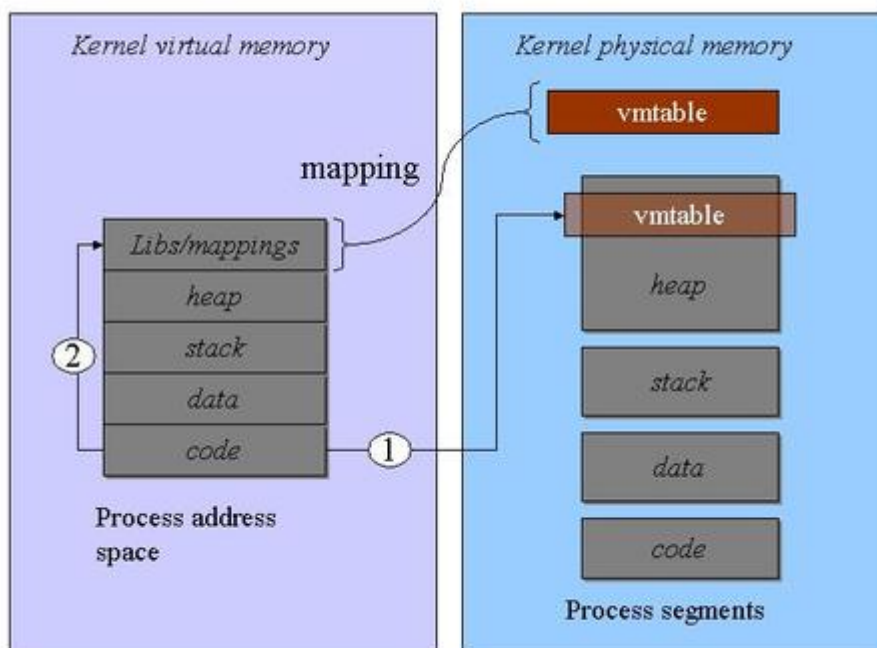


Figure 5. The Architectural Layout of vmtable

In the current implementation, physical pages are allocated explicitly by the vmtable subsystem. This is done to ensure that they really are allocated contiguously, so the memory pool can be used safely for I/O operations. A future extension is to use vmtable for direct I/O in the context of network performance.

vmtable exports a simple and easy-to-use interface to AEM users and module developers, hiding the complexity of memory allocation in kernel space. Simple routines are implemented in the AEM core module to allocate and free memory in a transparent manner. This interface greatly simplifies maintenance of modules and their adaptations between different Linux kernel releases.

Scalability

We performed testing to measure the behavior of AEM during a simple exchange between two remote processes. This test was done to ensure the time needed to context switch an event handler did not cause a performance problem. More recently, we also performed benchmarking to measure the scalability of AEM and to test the internal implementation of jobs and wait queues, which represent the base functionality of AEM.

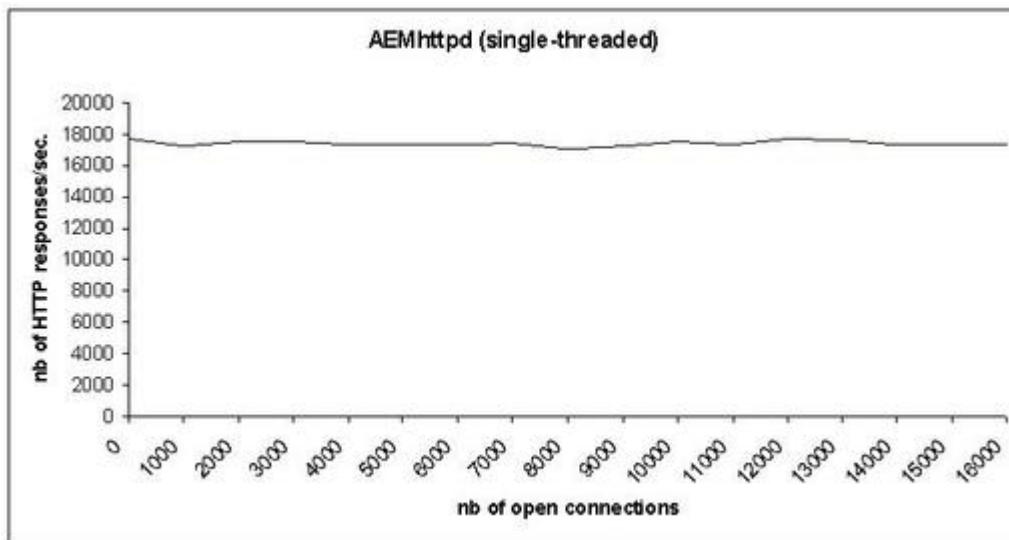


Figure 6. AEMhttpd, a single-threaded HTTP server used to make scalability measurements of the AEM internal implementation. Here we have performed 100 active connections for each sample.

In order to generate figures we could compare easily, we decided to use an existing Web server and adapt it with the AEM interface. AEMhttpd is a simple single-threaded HTTP server (see Resources.) A single-threaded server runs entirely in the main thread of execution. It means that neither kernel threads nor user threads are created to handle HTTP requests. The measurements done with this type of server focus on the implementation capabilities rather than on the performance of the server itself.

In the example illustrated in Figure 6, we have run 100 active transactions. For each transaction, we increased the number of open connections to increase the number of jobs in the sockets' connection wait queues. In a standard server implementation based on `select()`, this would have increased the time to respond to requests, because all descriptors would have been scanned in sequence. With AEM, only active jobs (that is, sockets with data ready) execute their corresponding event handlers. This proves that AEM provides a generic and scalable implementation.

Conclusion

Linux is widely used in the industry, imposing itself as the operating system of choice for enterprise-level solutions. It also is on the edge of becoming the operating system of choice for the next-generation IP-based architectures for telecom services. There already is a wide acceptance of Linux capabilities, but providing enhancements at the kernel level will attract the next-generation service providers that demand scalability, performance and reliability.

AEM is a solid attempt to provide the asynchronous notification of processes together with event data completion. It also brings an event-driven methodology that enables a secure programming paradigm for application developments. In addition, AEM implements a mechanism that focuses on increasing application reliability and portability by exporting a simple user interface.

Acknowledgements

All of the researchers at the Open Systems Lab and Lars Hennert at Ericsson for their useful comments, and Ericsson Research for approving the publication of this article.

Resources for this article: </article/7746>

Frédéric Rossi (Frederic.Rossi@ericsson.ca) is a researcher at the Open Systems Lab at Ericsson Research, in Montréal, Canada. He is the creator of AEM and the main driver behind its development.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Cooking with Linux

Performing at the Speed of Light

Marcel Gagné

Issue #127, November 2004

Now you can come up with a spaceship design that will look good even when the red shift kicks in. Watch relativity, star maps and more.

You are right, François, computers and operating systems have come a long way. Not only do we have the good fortune to be running the operating system of the future today, but we can take advantage of machines that are faster than ever before. I remember with some, well, I hesitate to call it fondness, but I do remember my first x86-based PC. It was a turbo-charged XT with a processor that ran at 10MHz. I also spent a small fortune upgrading its RAM from 640K to 1,024K by plugging a couple of dozen chips in to IC slots on the main board.

Quoi? Of course not, *mon ami*, although that stuff was fun at the time, I would not give up the technology of today. That almost would be like giving up Linux for another operating system. You know, François, it's interesting to think about exactly how far we have come—from megahertz to gigahertz processors in only a few short years! Where will we end up in another ten years? I suppose that faster than the speed of light may be possible, though I fear it may take somewhat longer, *mon ami*. Still, you have given me an idea.

Mon Dieu! Our guests are here already. To the wine cellar, *immédiatement!* Head to the north wing and check behind that new shipment of Bordeaux wines Henri delivered yesterday. You'll find a few cases of 2000 Châteauneuf-du-Pape. Forgive me, *mes amis*. Please sit and make yourselves comfortable. François and I were discussing how far technology has come in the last few years. My faithful waiter brought up the idea of faster-than-light computing, certainly the ultimate in high-performance computing, this issue's theme. Even if we could have computers delivering information at beyond light speed, we still would need to absorb information at our own pace. One thing is for sure,

we wouldn't see the stars zipping by as we read the latest on-line *Linux Journal* column.

In terms of high performance, nothing beats the speed of light, at least not without some strange matter or access to a matter/anti-matter engine and some dilithium crystals. You can get that feeling by firing up your screensaver and selecting rocks if you are using xscreensaver or the OpenGL space in KDE's own list. How objects look as you approach the speed of light is a popular mainstay of science-fiction films, but generally speaking, we never get to see what it actually might look like. That was the inspiration behind Daniel Richard G.'s Light Speed!, a program designed to show precisely what happens to our view of an object as it approaches the speed of light. The program takes into consideration various relativistic effects, such as Lorentz contraction, red/blue Doppler shift, headlight effect and optical aberration. The About page on the Light Speed! Web site describes all these effects (see the on-line Resources).

Mes amis, you are sure to enjoy this wine—truly high-performance strength, dark fruit flavors, a hint of coffee and mocha and a long finish. Enjoy it while we crank up the speed a bit. The build is very easy. You should be aware that you need the OpenGL or Mesa development libraries loaded as well as the gtkglarea libraries. From there, it's a simple extract and build five-step:

```
tar -xzvf lightspeed-1.2a.tar.gz
cd lightspeed-1.2a
./configure
make
su -c "make install"
```

Start the program by running `lightspeed`. You should see a window appear with a three-dimensional lattice cube. In the upper right-hand side, an input box lets you enter a speed in meters per second. Start with something fairly high; you also can use the up and down arrow keys to increase or decrease speed with finer control. When you press Enter, the object is accelerated to that speed with the resulting effects shown in the graphical window.

A cube getting distorted as it approaches the speed of light is only so interesting, although you can create a more complex lattice by clicking File on the menu bar, selecting New Lattice and choosing the number of points in three dimensions. The real question on my mind is what happens to a spaceship as it approaches the speed of light? Luckily, the Light Speed! Web site also has an objects download feature that you should pick up too. It contains three additional objects, including a model of the starship from *Star Trek Voyager* (Figure 1). To use a different model, click File and select Load Object.

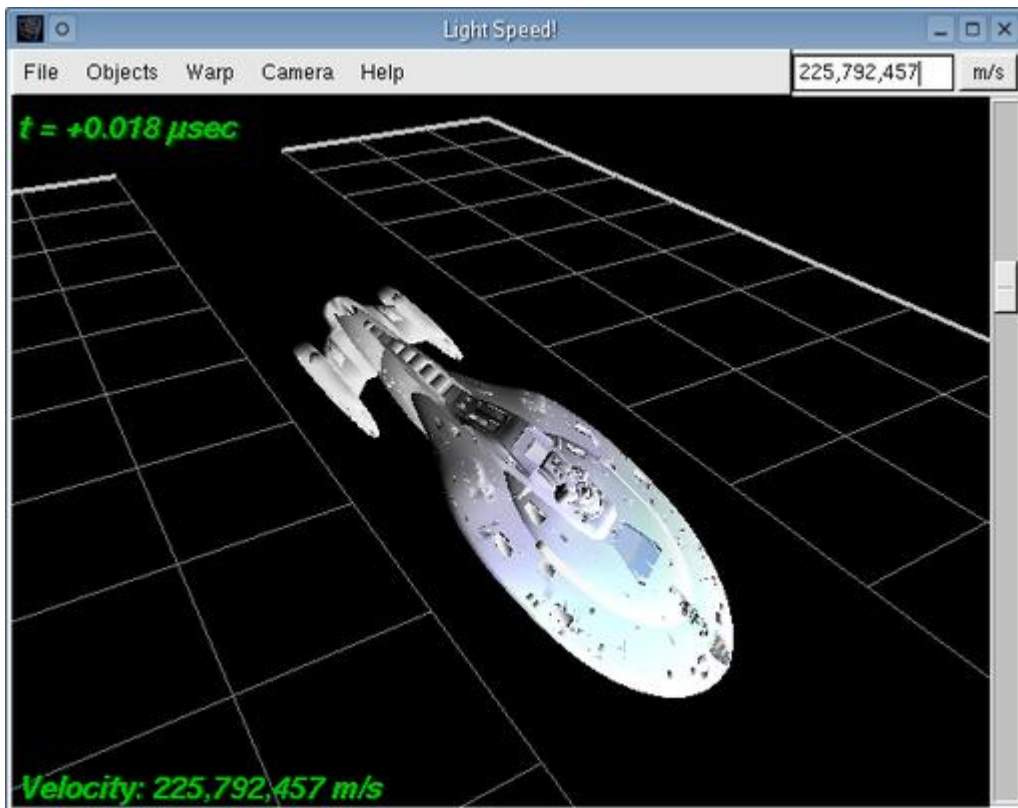


Figure 1. What really happens to a starship as it approaches the speed of light?

One of the more interesting things you can do to extend this bit of educational fun is to head over to the 3-D Cafe (see Resources), where you will find a lot of three-dimensional models and meshes to try. Don't limit yourself to spaceships, though; a race car approaching light speed also is fun to use. Keep in mind that only models with a 3DS (3D Studio) or LWO (LightWave 3D) extension work with this feature.

As much fun as it is to imagine what really happens under these conditions, what we all really want to do is go flying through space at warp ten while the stars zip by, arriving at some distant world before we can empty another bottle of wine. For just such a trip, get your hands on Chris Laurel's Celestia. With Celestia, you can tour around our solar system, visit over 100,000 different stars, check out what's happening with various Earth-launched spacecraft and much more. Source is available on the site, but there are binaries for Mandrake and SuSE and others also are available. If you can't find binaries for your distribution, never fear. Because this is an OpenGL project, you need the 3-D libraries, but the build itself simply is another extract and build five-step:

```
tar -xzvf celestia-1.3.1.tar.gz
cd celestia-1.3.1
./configure
make
su -c "make install"
```

To run the program, call `celestia` from the command line or your command launcher. You need to know about a few keystrokes right now, because they

make the experience that much more fun. Pressing the letter L accelerates time by a factor of ten. Doing so puts your travel through space in motion relative to whatever object you have chosen as your point of reference. Pressing K decelerates time, should you start going a little too fast. Pressing Alt-C brings up the Celestia browser from which you can select objects of many flavors. At the bottom of the screen are four radio buttons. Click the With planets button, and a list of stars with known planets appears. Want to visit the planet orbiting 51 Pegasi? Right-click on the object's name, select Goto and strap yourselves in for a faster-than-light trip to this alien world. Once there, right-click on the star, 51 Pegasi, select Follow and you can watch the planet's orbit as you remain focused on the star.

Keystrokes also let you specify the representation of stars, from tiny pinpricks to fuzzy points to scaled discs. To find out what all the keystrokes do, click Settings on the menu bar and select Configure Shortcuts.

Celestia is a great program to sit back and explore and is well worth the download. Aside from stars and planets, you can visit spacecraft currently orbiting nearby worlds, such as the Mars Global Surveyor. Try heading for the spacecraft, click on Mars and then select follow (Figure 2). Now accelerate time. A number of major asteroids also are in the database if you'd prefer a trip to Eros.

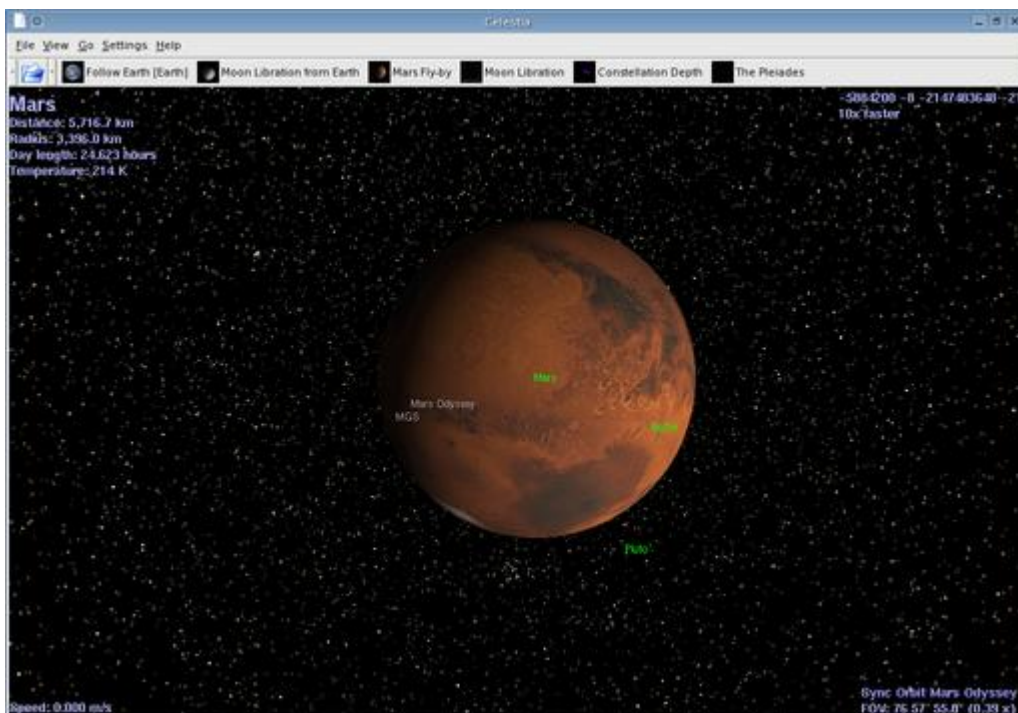


Figure 2. Tracking the Spacecraft Orbiting Mars

If zooming through space at or exceeding the speed of light is enough to turn your stomach not to mention your face a few shades of green, then perhaps a more down-to-earth space-based approach is in order. Why not observe the

stars and planets from the comfort of your non-moving seat? The best way to do this is with a great program called KStars, originally created by Jason Harris.

KStars is a desktop planetarium program that displays the locations of stars and planets on your desktop. Because KStars is a part of the KDE desktop environment—included in the kdeedu package—you don't have to look far to get a copy. You can find the latest on the package by visiting the Web site.

KStars is amazing fun but much more than a toy. With a database of the planets, 130,000 stars, 13,000 deep-sky objects, the planets and many asteroids, KStars is an astronomical treasure. With it, you visually can identify the position of stars, galaxies, nebulae and other glories of the night sky. You can control what is displayed, zoom in on objects and—I love this part—download images from on-line resources, such as the *Hubble* and the Space Telescope Science Institute. Simply right-click on an object of interest, and the pop-up offers you additional information and links to high-resolution images of those objects when appropriate. Figure 3 shows my KStars session pulling up information on the Trifid Nebula.

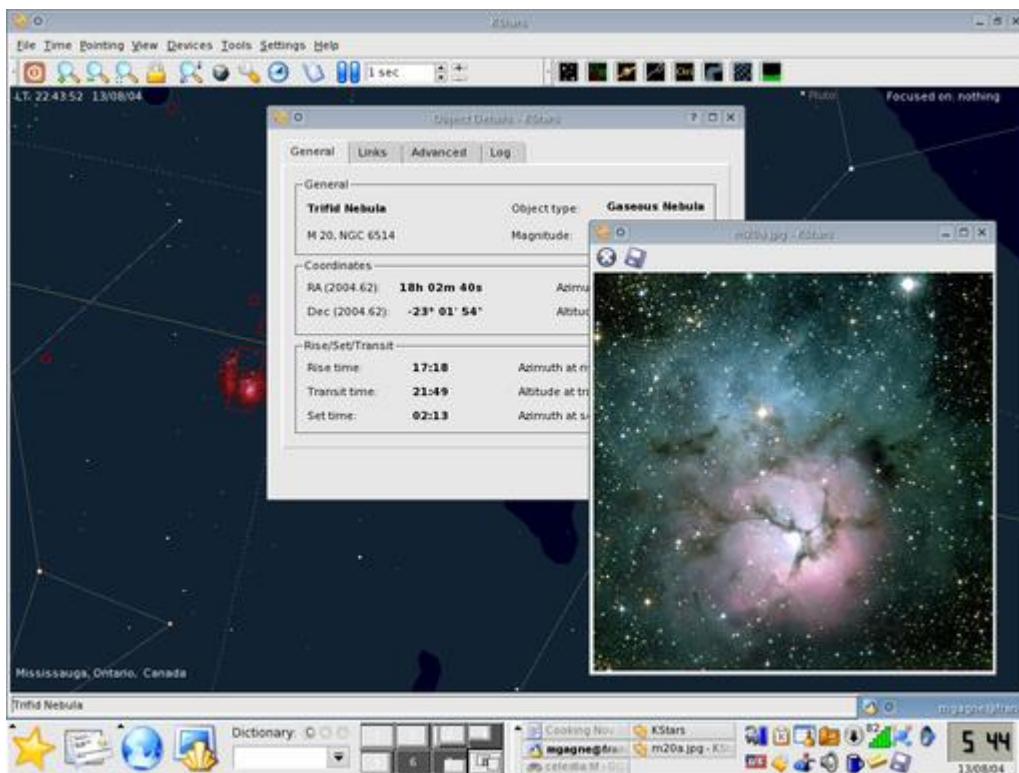


Figure 3. KStars can import images from on-line databases such as *Hubble's*.

When you start KStars, it assumes your location is Greenwich, United Kingdom, which probably is not what you want. Start by clicking Location on the menu bar and selecting Geographic. A dialog box will appear with a world map. Click an area on the map close to where you live. Doing so provides you with a list of geographical points in a list to the right of the map. Make your selection and

click OK. Should you happen to know your latitude and longitude, you can enter that at the bottom of the window instead.

KStars includes much more than what I can cover in this short visit. For instance, KStars can control your telescope, locating and tracking objects. Furthermore, if you are into astro-photography, KStars can control CCDs, currently supporting Finger Lakes Instruments devices with others in development.

Mon Dieu! Although it may not have happened at hyper-light speeds, it certainly has happened fast. Yes, I'm talking about the clock, *mes amis*, which already is telling us it is closing time. With talk of moving so quickly, it is at times like this that we can truly appreciate sitting back under a starlit night, slowly sipping a little more of this excellent Châteauneuf-du-Pape. Until next time, *mes amis*, let us all drink to one another's health. *A votre santé Bon appétit!*

Resources for this article: </article/7753>.

Marcel Gagné (mggagne@salmar.com) lives in Mississauga, Ontario. He is the author of the all-new *Moving to the Linux Business Desktop* (ISBN 0-131-42192-1), his third book from Addison-Wesley. In real life, he is president of Salmar Consulting Inc., a systems integration and network consulting firm. He is also a pilot, writes science fiction and fantasy, and folds a mean origami T-Rex.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Paranoid Penguin

Linux Filesystem Security, Part II

Mick Bauer

Issue #127, November 2004

We covered the fundamentals of permissions last month. Now it's time to learn some useful bits to make cooperation among users convenient and secure.

Last time, we looked at file and directory permissions from the ground up—what users and groups are and how to set and remove read, write and execute permissions on files and directories. In this column, we look at some more advanced types of permissions, explore permission numeric modes and the command `umask` and see how to delegate root's authority with `su` and `sudo`. This article contains more intermediate-level information than last month's, but hopefully it should make sense, even if all you know about permissions is what you read here last time.

The Sticky Bit

Recall last month's long listing of the `extreme_casseroles/` directory:

```
drwxr-x--- 8 biff drummers 288 Mar 25 01:38 extreme_casseroles
```

Recall also that we set the group permissions on this directory to `r-x`, that is, group-readable and group-executable, so that our fellow members of the `drummers` group could enter this directory and enjoy the recipes stored therein.

Suppose that our drummer friend Biff wants to allow his fellow drummers not only to read his recipes but to add their own as well. As we saw last time, all he needs to do is set the group-write bit for this directory, like this:

```
chmod g+w ./extreme_casseroles
```

There's only one problem with doing that, however. Write permissions include both the ability to create new files in this directory and also to delete them. What's to stop one of his drummer pals from deleting other people's recipes? The sticky bit, that's what.

In olden times, the sticky bit was used to write a file (program) to memory so it would load more quickly when invoked. On Linux, however, it serves a different function. When you set the sticky bit on a directory, it limits people's ability to delete things in that directory. That is, to delete a given file in the directory you either must own that file or own the directory, even if you belong to the group that owns the directory and group-write permissions are set on it.

To set the sticky bit, issue the command:

```
chmod +t directory_name
```

In our example, this would be `chmod +t extreme_casseroles`. If we now do a long listing of the directory itself, by using `ls` with the `-ld` option to list the directory's permissions rather than its contents, that is, `ls -ld extreme_casseroles`, we see:

```
drwxrwx--T 8 biff drummers 288 Mar 25 01:38 extreme_casseroles
```

Notice the `T` at the end of the permissions. We'd normally expect to see either `x` or `-` there, depending on whether the directory is other-writable. The `T` denotes that the directory is not other-executable and has the sticky bit set. A lowercase `t` would denote that the directory is other-executable and has the sticky bit set.

To illustrate what effect this restriction has, suppose a listing of the contents of `extreme_casseroles/` looks like Listing 1.

Listing 1. Contents of `extreme_casseroles/`

```
drwxrwxr-T 3 biff drummers 192 2004-08-10 23:39 .
drwxr-xr-x 3 biff drummers 4008 2004-08-10 23:39 ..
-rw-rw-r-- 1 biff drummers 18 2004-07-08 07:40 chocolate_turkey_casserole.txt
-rw-rw-r-- 1 biff drummers 12 2004-08-08 15:10 pineapple_mushroom_surprise.txt
drwxr-xr-x 2 biff drummers 80 2004-08-10 23:28 src
```

Suppose further that the user `crash` tries to delete the file `pineapple_mushroom_surprise.txt`, which `crash` finds offensive. `crash` expects this to work, because he belongs to the group `drummers` and the group-write bit is set on this file. Remember, though, that `biff` set the parent directory's sticky bit. Therefore, `crash`'s attempted deletion fails, as we see in Listing 2.

Listing 2. Attempting Deletion with Sticky Bit Set

```
crash> rm pineapple_mushroom_surprise.txt
rm: cannot remove `pineapple_mushroom_surprise.txt':
Operation not permitted
```

One more note on the sticky bit: it only applies to the directory's first level downward. In Listing 1, you may have noticed that besides the two nasty recipes, `extreme_casseroles/` also contains another directory, `src`. The contents of `src` will not be affected by `extreme_casseroles'` sticky bit, although the directory `src` itself is. If `biff` wants to protect `src's` contents from group deletion, he needs to set `src's` own sticky bit.

setuid and setgid

Now we come to two of the most dangerous permissions bits in the world of UNIX and Linux, `setuid` and `setgid`. If set on an executable binary file, the `setuid` bit causes that program to run as its owner, no matter who executes it. Similarly, when set on an executable, the `setgid` bit causes that program to run as a member of the group that owns it, again regardless of who executes it.

When I say run as, I mean the program runs with the same privileges as. For example, suppose `biff` writes and compiles a C program, `killpms`, that behaves the same as the command `rm /extreme_casseroles/pineapple_mushroom_surprise.txt`. Suppose further that `biff` sets the `setuid` bit on `killpms`, with the command `chmod +s ./killpms` and also makes it group-executable. A long listing of `killpms` might look like this:

```
-rwsr-xr-- 1 biff drummers 22 2004-08-11 23:01 killpms
```

If `crash` runs this program, he finally can succeed in his quest to delete the Pineapple-Mushroom Surprise recipe: `killpms` runs as though `biff` had executed it. When `killpms` attempts to delete `pineapple_mushroom_surprise.txt`, it succeeds because the file has user-write permissions and `killpms` is acting as its user/owner, `biff`.

IMPORTANT WARNING

`setuid` and `setgid` are very dangerous if set on any file owned by root or any other privileged account or group. I'm illustrating `setuid` and `setgid` so you understand what they do, not because I think you actually should use them for anything important. The command `sudo`, described later in this article, is a much better tool for delegating root's authority.

If you want a program to run `setuid`, that program must be group-executable or other-executable for what I hope are obvious reasons. In addition, the Linux kernel ignores the `setuid` and `setgid` bits on shell scripts. These bits work only on binary (compiled) executables.

setgid works the same way but with group permissions. If you set the setgid bit on an executable file with the command `chmod g+s filename`, and if the file also is other-executable (-r-xr-sr-x), when that program is executed it runs with the group ID of the file rather than of the user who executed it.

In the above example, if we change killpms' other permissions to r-x (`chmod o+x killpms`) and make it setgid (`chmod g+s killpms`), no matter who executes killpms, killpms exercises the permissions of the drummers group, because drummers is the group owner of killpms.

setgid and Directories

What about directories? Well, setuid has no effect on directories, but setgid does, and it's a little non-intuitive. Normally, when you create a file, it's automatically owned by your user ID and your (primary) group ID. For example, if biff creates a file, the file has a user owner of biff and a group owner of drummers, assuming that drummers is biff's primary group, as listed in `/etc/passwd`.

Setting a directory's setgid bit, however, causes any file created in that directory to inherit the directory's group owner. This is useful if users on your system tend to belong to secondary groups and routinely create files that need to be shared with other members of those groups. For example, if the user animal is listed in `/etc/group` as being a secondary member of drummers but is listed in `/etc/passwd` as having a primary group of muppets, then animal has no trouble creating files in the `extreme_casseroles/` directory, whose permissions are set to `drwxrwx--T`. However, by default, animal's files belong to the group muppets, not to drummers, so unless animal manually reassigns his files' group ownership (`chgrp drummers newfile`) or resets their other permissions (`chmod o+rw newfile`), other members of drummers cannot read or write animal's recipes.

If, on the other hand, biff or root sets the setgid bit on `extreme_casseroles/` (`chmod g+s extreme_casseroles`), when animal creates a new file therein, the file has a group owner of drummers, exactly like `extreme_casseroles/` itself. All other permissions still apply; if the directory in question isn't group-writable to begin with, the setgid bit has no effect, because group members are not able to create files inside it.

Now we've covered all possible permissions: read, write, execute, sticky bit, setuid and setgid. If you understand all six of these, you're probably in the minority of Linux users. But wait, there's more!

Numeric Modes

So far we've been using mnemonics to represent permissions—r for read, w for write and so on. Needless to say, as with everything else, your system actually uses numbers to represent permissions. The `chmod` command recognizes both mnemonic permission modifiers (`u+rwx,go-w`) and numeric modes.

A numeric mode consists of four digits: as you read left to right, these represent special permissions, user permissions, group permissions and other permissions. Recall that other is short for other users not covered by user permissions or group permissions. For example, `0700` translates to no special permissions set, all user permissions set, no group permissions set and no other permissions set.

Each permission has a numeric value, and the permissions in each digit place are additive: the digit represents the sum of all permission bits you want to set. If, for example, user permissions are set to 7, this represents 4 (the value for read) plus 2 (the value for write) plus 1 (the value for execute).

As I just mentioned, the basic numeric values are 4 for read, 2 for write and 1 for execute. (I remember these by mentally repeating the phrase, read-write-execute, 4-2-1.) Why no 3, you might wonder? Because this way, no two combination of permissions have the same sum.

Special permissions are as follows: 4 stands for `setuid`, 2 stands for `setgid` and 1 stands for sticky bit. For example, the numeric mode `3000` translates to `setgid` set, sticky bit set and no other permissions set, which is, actually, a useless set of permissions.

Here's one more example of a numeric mode. If I issue the command `chmod 0644 mycoolfile`, I am setting the permissions of `mycoolfile`, as shown in Figure 1.

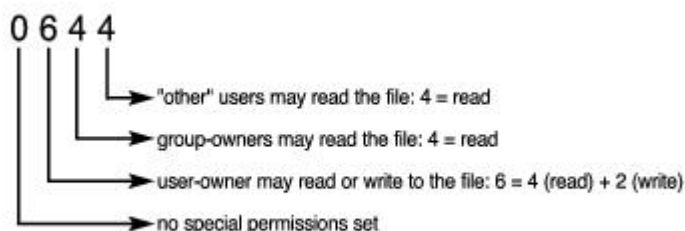


Figure 1. Permissions for `mycoolfile`

For a more complete discussion of numeric modes, see the info page for `coreutils`, node `Numeric Modes`. That is, enter the command `info coreutils numeric`.

umask

I want to cover one last command specific to permissions before closing with a couple of other topics. `umask` is a command built into the bash shell that prints or sets your default permissions mask. To see yours, simply enter the `umask` command without any arguments; it returns a four-digit number. On my system, it looks like Listing 3.

Listing 3. Checking My Default Permissions Mask

```
mick@localhost:/home/mick> umask
0022
```

Mode 0022 means no special permissions, no user-owner permissions, group and other permissions set to write, right? How can that be?

Actually, `umask` deals in masks, not in modes per se. 0022 is what is subtracted from the number 0777 to determine the numeric mode of files you create: $0777 - 0022 = 0755$.

Aha! So, files I create have user-owner permissions set to read-write-execute ($7 = 4 + 2 + 1$) and group and other permissions set to read-execute ($5 = 4 + 1$)? Right? Almost. It also happens that `umask` sets the execute bit automatically only on directories. Even if your permissions mask includes execute permissions, the execute bit does not set automatically on regular files you create. So, if my permissions mask is 0022, resulting in default permissions of 0755, and I create a file named `default_file` and a directory named `default_dir`, long listing output for those two items look like Listing 4.

Listing 4. File and Directory with Mask of 0022

```
-rwxr-xr-x  2 mick users      48 2004-08-13 08:31 default_dir
-rw-r--r--  1 mick users       4 2004-08-13 08:31 default_file
```

To change your default permissions mask, simply issue the command `umask` with the new mask as its argument. For example, if I want all my files to have group-read permissions but no other permissions, this translates to a numeric mode of 0740. If I subtract that from 0777 I get a mask of 0037. Therefore, the `umask` command I enter is `umask 0037`. This new mask, however, applies only to my current session and any new shells I start from it. To make it persistent, I can add the line `umask 0037` to my `.bashrc` file.

su and sudo

I should say a few words about the reality of users, groups and permissions. The whole problem with UNIX security is that far too often, permissions and

authority on a given system boil down to root can do anything, although users can't do much of anything.

Sadly, it's much easier to do a quick `su -` to become root for a while than it is to create a granular system of group memberships and permissions that allows administrators and sub-administrators to have exactly the permissions they need. Sure, you can use the `su` command with the `-c` option, which allows you to specify a single command to run as root rather than an entire shell session (for example, `su -c rm somefile.txt`), but this requires you to enter the root password. It's never good for more than a small number of people to know root's password.

Another approach to solving the root-takes-all problem is to use role-based access control (RBAC) systems, such as SELinux, which enforce access controls that reduce root's effective authority. However, this makes things even more complicated than setting up effective groups and group permissions. This is not to say that SELinux and the rest aren't good things—I love RBAC.

A better middle ground is to use the `sudo` command. `sudo` is short for superuser do, and it allows users to execute single commands as root, without actually needing to know the root password. `sudo` is now a standard package on most Linux distributions.

`sudo` is configured with the file `/etc/sudoers`, but you shouldn't edit this file directly. Rather, use the `visudo` command, which opens a editor on the file; `vi` is the editor by default. You can use a different editor by setting the `EDITOR` environment variable. For example, to use `/usr/bin/gedit`, do this:

```
export EDITOR=/usr/bin/gedit
```

Space doesn't permit me to explain `sudoers`' syntax in detail; see the `sudoers(5)`, `sudo(8)` and `visudo(8)` man pages for complete information. In the space available here, let's run through a quick example.

Remember the user `crash`'s quest to rid the world of Pineapple-Mushroom Surprise? Although in this case it would be overkill—the permissions techniques I've already illustrated are sufficient—you could use `sudo` to allow `crash` to realize his goal, assuming you (`biff`) have root privileges. First, become root (`su -`). Next, execute the command `visudo`. You're now in a `vi` session, editing the file `/etc/sudoers`; see the `vi(1)` man page if you're new to `vi`. Go down to the bottom of the file and add this line:

```
crash localhost=/bin/rm /home/biff/extreme_casseroles/pineapple_mushroom_surprise.txt
```

Save and exit the file.

Now, to do his thing, crash enters the command:

```
sudo rm /home/biff/extreme_casseroles/pineapple_mushroom_surprise.txt
```

whereupon he is prompted to enter his password. After he enters this correctly, the command:

```
/bin/rm /home/biff/extreme_casseroles/pineapple_mushroom_surprise.txt
```

is executed as root, and the offending file is gone.

Alternately, the line in `/etc/sudoers` could look like this:

```
crash localhost=/bin/rm /home/biff/extreme_casseroles/*
```

That way, crash can delete anything in `extreme_casseroles/`, regardless of the sticky bit setting.

As handy as it is, sudo is a powerful tool, so use it wisely; root privileges never should be trifled with. It really is better to use user and group permissions judiciously than to hand out root privileges, even with sudo. Better still, use an RBAC-based system such as SELinux if the stakes are high enough.

That's it for now. I hope you've found this tutorial useful. Until next time, be safe!

Mick Bauer, CISSP, is *Linux Journal's* security editor and an IS security consultant in Minneapolis, Minnesota. He's the author of *Building Secure Servers With Linux* (O'Reilly & Associates, 2002).

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Linux for Suits

We're Going to Be a 90% Linux Shop

Doc Searls

Issue #127, November 2004

The biggest Linux success story is one too few IT workers can tell. Linux and open source are pushing proprietary software out to the edges and taking over the core, where the money is.

At the O'Reilly Open Source Convention (OSCon) last summer, the killer talk wasn't one of the keynotes, although most of those were excellent. It was a breakout session called "Commercial OSS Business". The panel featured an A-list lineup: Matt Asay, Director of Novell's Linux Business Office and founder of the Open Source Business Conference; Brian Behlendorf, founder of the Apache Foundation and CollabNet; Bob Lisbonne, a VC with Matrix Partners; Jason Matusow, Director of Microsoft's Shared Source Initiative; and Zack Urlocker, Marketing VP with MySQL. The moderator was Tim O'Reilly himself.

They all had terrific stuff to say—much of it quotable. Yet the best contributor to the session wasn't a panelist at all, but an audience member who grabbed the microphone, stood in front of the stage and put on a performance worthy of Frank Sinatra fronting Tommy Dorsey's band. It was Phil Moore, Executive Director in the UNIX Engineering team at Morgan Stanley. He began:

I work for the 38th largest company in the world, Morgan Stanley. We have a billion dollar IT budget. And we use a little of everything. Unfortunately. Excuse me, a LOT of everything. The trend I've seen in the last ten years...is the exponential growth in the variety and the depth and breadth of installation of open-source software in our infrastructure....What I'm seeing is that in the infrastructure, the core infrastructure, open source is going to take over, leaps and bounds....I'm predicting, right now, that by 2006 or 2007, we're going to be a 90% Linux shop.

He spoke for several minutes, pacing in front of the stage, addressing both the audience and the panel. When he finished, there was applause.

Phil's speech was so compelling for three reasons beyond the content of his talk. First, he was a customer rather than a vendor. It's customary at tradeshows for vendors to fill session panels. As good as this panel was, nothing any panelist could say carried the authority that comes from real customers. Second, Phil clarified market roles by making it obvious that customers are in charge of adoption—not vendors. Third, he testified to the enormous success of The SCO Group's "Fear, Uncertainty and Doubt" (FUD) campaign by being courageous enough to stand up and speak out about it.

At one point in his soliloquy, Phil said, "We're still mostly a Solaris shop, but we are rapidly moving to Linux, though I'm not supposed to talk about that, for fear of being sued by SCO." Then he turned to Matt Asay and said, "Which is the reason why I couldn't go to your conference, the OSBC. I wasn't allowed to go." That filled a blank for me, because Phil was slated to be on my DIY-IT panel at OSBC and couldn't make it; now I knew why. His cancellation brought the number of AWOL panelists to two out of the original four. The other panelist came to the conference but was told at the last minute by his employer that he couldn't speak, and he ended up sitting in the audience. Significantly, the two absentees were from large companies with buildings full of lawyers. To his and his employer's credit, R0ml Lefkowitz of AT&T Wireless did make the panel, as well as a speech of his own given the same day at the conference.

But at OSCon, Phil got to speak, and the message he carried was one I've been hearing privately from many other IT professionals: Linux is rapidly becoming default infrastructure, and open source is the preferred code condition for all infrastructure. Not that they're abandoning Microsoft—far from it. But Microsoft's primary goods—desktop operating systems and applications—are becoming niched to the verge of quarantine. Phil made this clear when he turned to Jason Matusow of Microsoft and said, "You can have the desktop. It's a pain in the ass. I don't want it. I just want the (core) where all the money's made."

Later he added, "I will bet my career that Microsoft is going to get wiped out on the desktop in the next ten years. Not in this country." Turning again to Jason, he said, "You're going to own it here because America loves you guys. You're set, for at least ten years." Turning back to the audience, he continued:

Look overseas at what's happening [with Linux]. It doesn't matter what distribution. Because [Linux is] economical for people in foreign countries. It lets them invest in their own local software companies without putting money into these guys' pockets [indicates Microsoft] or some other foreign corporation that

doesn't have a vested interest in your own economy and your own culture. That's going to be the number one reason why open source ends up taking over the planet.

As for other vendors serving the IT space, he said, "I think you'll see proprietary companies shifted out to the leaf nodes, coming up with special-purpose applications that are difficult to do on a large scale." But in his current role as an enterprise IT architect, Phil Moore still supports Microsoft desktops in the midst of a growing infrastructure comprised of Linux and other open-source building materials. And, he expects to maintain that relationship for the foreseeable future.

While Phil and the others were talking, I realized that open source and the proprietary software industry are at crossed purposes only where they compete outright. But when Linux and open-source products serve as infrastructural support, Microsoft OSes and apps are supported along with everything else.

In conversations that followed, out in the halls at OSCon and in subsequent meetings at OSCon and LinuxWorld Expo, which followed the next week, I began to visualize the subject, starting with the traditional industrial market model. This model was best described by John Perry Barlow in his 1995 essay "Death From Above", which argued against the asymmetrical bandwidth delivery plants that the cable and phone companies then were beginning to build out. Here's how it begins:

Over the last 30 years, the American CEO Corps has included an astonishingly large percentage of men who piloted bombers during World War II. For some reason not so difficult to guess, dropping explosives on people from commanding heights served as a great place to develop a world view compatible with the management of a large post-war corporation.

It was an experience particularly suited to the style of broadcast media. Aerial bombardment is clearly a one-to-many, half-duplex medium, offering the bomber a commanding position over his "market" and terrific economies of scale.

This industrial tradition has a number of ideals. The most obvious one is to sell unique and proprietary products to the largest number of people. Less obvious, but no less important, is to create and sustain market categories populated with intermediaries and to hold as many dependents as possible—from distributors and OEMs on down to customers—captive on the manufacturer's platform.

The Open Source and Free Software movements are driven by ideals that are roughly orthogonal to the few-to-many model. To describe those ideals, also found in the Internet's original architecture, it helps to start with the qualities of open-source products: nobody owns them, everybody can use them and anybody can improve them. It's tempting to call this Peer to Peer, but to is not the operative preposition here. Hacking may involve the transport of packets, but the collaborative activities involved are with, not to.

Juxtapose any with any on few to many, and you can see the cross-purposed result. It's easy to see how this presents a problem, not only for software giants such as Microsoft but for few-to-many empires including the entertainment industry and consumer electronics. Protecting few-to-many from any with any has become a cause for the whole entertainment industry. The Digital Millennium Copyright Act, lobbied through Congress in 1998, is landmark achievement in paranoia.

Yet now large customers such as Morgan Stanley show us we misconceive the market when we see only conflict between open-source and proprietary software business imperatives. They make this clear when they put any with any in a supportive position beneath few to many. By its relationship-agnostic nature, any with any can include and support peer to peer, many to many, business to business or any other pair of nouns flanking a preposition.

If Linux is infrastructure, where does infrastructure fit? This question matters, because it provides the context within which paranoid few to many forces attempt to control infrastructure and prevent any with any from working.

That context is best described in the "layers of time" diagram from the Long Now Foundation that we first visited in May 2002.

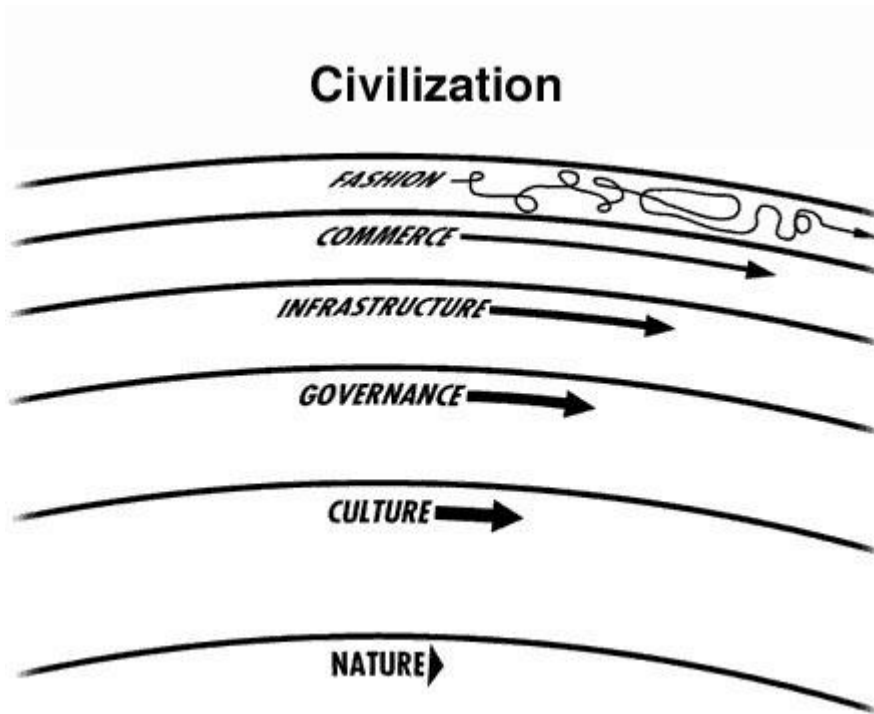


Figure 1. The Long Now Foundation's "Layers of Time" Diagram

This is a model of basic dependencies, as well as a way to sort out differences in rates of change. Each higher level depends on the one below it. That's how the few-to-many system, which operates at the Commerce level, depends on Infrastructure. It's also why commercial interests often work to control infrastructure one layer below, at the Governance level, often with great success. In fact, regulatory systems around the world have served commercial interests for centuries.

Open-source infrastructure, however, was established at fashion-level speeds—faster than industrial establishments could react in most cases. Given the amorphous and ungovernable nature of the Net, regulating it presented a severe challenge. Plus, it brought too many benefits. Today, it's a risk for companies *not* to take advantage of any-with-any infrastructure.

The single unqualified lobbying success against any-with-any was the DMCA, a big, bad, dumb piece of legislation that needs to be repealed. Meanwhile, plenty of badness still is reposing in old copyright and patent law. Without that badness, SCO's FUD campaign would not have been so successful. And that success is much more widespread than it appears, precisely because it's working. With the rare exception of guys like Phil Moore, IT workers at big companies aren't telling Linux success stories.

I've avoided writing about SCO ever since it made news by suing IBM early last year. First, I believed SCO had no case. Second, given this column's three-month lead time, writing about the subject seemed pointless. But paranoia about discussing Linux and open source has become a prevailing condition

inside large companies. Legal departments began putting IT workers and everybody else under gag orders as soon as SCO began suing large customers, such as Daimler-Chrysler. This has caused a news hole of massive dimensions, even though it's not especially visible at vendor-centric conferences or in vendor-driven publications.

So, although everybody continues to handicap the futures market in Linux desktops, the real challenge is talking about how successful Linux really is in the enormous and far more important market for enterprise infrastructure.

Resources for this article: </article/7755>.

Doc Searls (info@linuxjournal.com) is senior editor of *Linux Journal*. His monthly column is Linux for Suits and his biweekly newsletter is SuitWatch. He also presides over Doc Searls' IT Garage (garage.docsearls.com), a sister site to *Linux Journal* on the Web.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

EOF

No 2.7 Kernel?

Greg Kroah-Hartman

Issue #127, November 2004

Forget everything you knew about odd-numbered and even-numbered kernels.

At the 2004 Linux Kernel Summit, the core kernel developers announced they weren't creating a 2.7 development kernel anytime soon. They said they liked the way things were going and didn't want to change things. This caused a lot of confusion, so this article is an attempt to explain.

During the 2.5 kernel development cycle, the top-level maintainers' process changed. Before Linus started using the BitKeeper version control system, kernel maintainers would send Linus 10–20 patches at once, then wait for him to release a snapshot to determine whether the patches had made it in. If not, they would try again. This worked pretty well for more than ten years.

Early in the 2.5 development cycle, a huge flame war over dropped patches ended with Linus deciding to try BitKeeper. After much hacking by the BitKeeper developers to clean up some features that kernel developers needed, Linus released the 2.5.3 kernel on February 2, 2002, and announced he was going to use BitKeeper. This really didn't change the way the majority of kernel developers worked. They still send patches to the upper-level maintainers and wait. But for the small subset of maintainers that decided to use BitKeeper, life changed a lot. They would create a BitKeeper tree, populate it with the changes they wanted to send to Linus and then point Linus to it. He would suck the patches into his tree and merge any minor conflicts with other people's work.

BitKeeper had some unexpected consequences. First, everyone had an up-to-date view of Linus' tree at any moment. A few developers, including Peter Anvin and Jeff Garzik, created the ability to make nightly snapshots appear as patches

at kernel.org. They also, with the help of the BitKeeper developers, created CVS and Subversion repositories for users of those version control systems.

Knowledge of the current state of the tree meant maintainers could start sending patches to Linus faster and see when he accepted them. Instead of waiting two weeks for a new snapshot, they could send in more changes immediately. The rate of kernel development instantly increased.

Second, every patch accepted into Linus' tree started going to a mailing list, which enabled everyone to see changes. Developers could watch what was happening in all parts of the kernel, see the reasoning as explained in changelog entries and point out problems. The list increased the peer review process, allowing new bugs to be noticed sooner, while the area of development was still fresh in the developer's mind.

2.6 Finally Escapes from Development

On October 31, 2002, kernel developers announced the 2.6 feature-freeze. On July 7, 2003, the first 2.6.0-test1 kernel was released, and the maintainer process changed again. Andrew Morton started being the funnel to Linus for almost all patches. However, maintainers that used BitKeeper kept having their trees being pulled directly in by Linus. Finally, on December 17, 2003, the 2.6.0 kernel was released, representing an average of 1.66 changes per hour for the 680 days of 2.5 and 2.6 development.

The next five 2.6.x kernel releases happened about every month, with 538–1,472 changes per release. Then, with the 2.6.5 kernel things started to move much more quickly; 2.6.6 came out with 1,757 changes, and 2.6.7 had 2,306 changes. From 2.6.0 to 2.6.7, the stable kernel, at 2.2 patches per hour, was changing at a rate greater than the “development” kernel had. But, the 2.6.7 kernel was the most stable Linux kernel ever, by a wide range of benchmarks and regression tests.

Have the core kernel developers gone mad and started to add untested code willy-nilly? No. In 2.6, Andrew Morton continued to stage all proposed patches for testing before sending them to Linus. Maintainers using BitKeeper would check the status of their patches in Andrew's tree, and if no problems were found, they would ask Linus to accept them.

So, all changes now are being tested by users in the -mm tree. This is different from how things had been done before. Now, patches are tested, built, used and abused by users in the world before being deemed acceptable. If a specific patch or set of patches is found to have problems, Andrew simply drops them from his tree and forces the original developer to fix the issues.

Because of the ability for a wider range of testing of patches to go into the tree, the development process for 2.6 will consist of the following: 1) Linus releases a 2.6 kernel release. 2) Maintainers flood Linus with patches that have been proven in the -mm tree. 3) After a few weeks, Linus releases a -rc kernel snapshot. 4) Everyone recovers from the barrage of changes and starts to fix any bugs found in the -rc kernel. 5) A few weeks later, the next 2.6 kernel is released and the cycle starts all over again.

However, if a set of kernel changes ever appears that looks to be quite large and intrusive, a 2.7 kernel might be forked to handle it. Linus will do this, putting the new experimental patches into that tree. Then he will continue to pull all of the ongoing 2.6 changes into the 2.7 kernel, as the 2.7 kernel stabilizes. If it turns out that the 2.7 kernel is taking an incorrect direction, the 2.7 kernel will be deleted, and everyone will continue on with the 2.6 tree. If the 2.7 tree becomes stable, it either will be merged back into the 2.6 tree, or it will be declared 2.8.

All of this is being done because kernel developers are working very well together in the current situation. Large changes that are arguably rather intrusive, like the change from 8k to 4k kernel stacks, are being merged into the “stable” kernel series. Users have access to the latest features with a greatly reduced delay time. Distributions can provide a more stable kernel to their customers, as they are not forced to backport features from the “development” kernel into their “stable” kernel, as was the case during the 2.5 development series.

Quicker development ensures that the in-kernel API will change constantly. This always has been the case for Linux, but now it is even more pronounced. Thus, any kernel modules that are not kept in the main kernel.org tree quickly will stop working. It is essential that these modules be added to the main kernel tree. That way, any API changes also are made to the affected modules, and all users benefit from the added features these modules provide.

The process really hasn't changed suddenly, it has evolved slowly into something that has been working quite well—so well in fact, no one outside of the kernel community realized it had changed, only that they were using the best Linux kernel ever.

Greg Kroah-Hartman currently is the Linux kernel maintainer for a variety of different driver subsystems. He works for IBM, doing Linux kernel-related things, and can be reached at greg@kroah.com.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

GumStix WS200X

Michael Boerner

Issue #127, November 2004

I found GumStix to be responsive to my concerns, and the company has shown itself to be responsive to their users and open to challenges in developing novel products.

Product Information.

- Vendor: GumStix
- URL: www.gumstix.com
- Programming: www.gumstix.org/tikiwiki
- Prices: WS-200ax—\$139 US, WS-200ax-bt—\$184 US, Serial Cable—\$12 US, USB cable—\$15 US, 32MB MMC—\$25 US, 128MB MMC—\$65 US and Antenna—\$10

The Good.

- Unbelievably small.
- Easy to work with.
- Inexpensive.

The Bad.

- No easy I/O line access.
- Evolving documentation and product lines.
- Fragile Bluetooth antenna connector.

GumStix, founded in 2004, focuses on single-board computers (SBCs) built around the Intel XScale PXA255 chip with Linux onboard. This review covers two GumStix products in the WaySmall line, the WS200 and WS200-bt, one with and

one without an Infineon ROK104001 Bluetooth module. The case is basic and small (1.5" × .25" × .5") with two mini-DIN8 serial connects and a USB mini-B port connector, a port for an MMC Flash memory module and a 0.65 mm 4.5V power connector. The version equipped with Bluetooth has an antenna connector as well.

It is clear that the GumStix product line is evolving and expanding rapidly. Since this review began, GumStix has added Bluetooth as an option, and the company provided the second evaluation unit well into the review process. In addition to Bluetooth, the newer version of the WS200 has a 60-pin Hirose daughterboard connector rather than the 24-pin MOLEX connector on the original evaluation model. I found GumStix to be responsive to my concerns, and the company has shown itself to be responsive to their users and open to challenges in developing novel products. Hopefully, that attitude will not change.

The GumStix has the potential to be a truly breakaway product. Several other SBCs are available, but none offers the combination of price, functionality, size and low power consumption that GumStix offers. If you're an embedded developer, the speed will make you happy, and the ease of use will make you smile for days. I had the WaySmall running, connected to my Fedora Core 2 notebook, in less than 15 minutes. The WaySmall devices are an excellent place to start learning embedded Linux.

Now, a little bad news: the documentation is a work in progress; however, the company indicates that it understands the documentation has issues and is working hard to improve it. GumStix recently added a Wiki with up-to-date information and is rewriting the user manuals.

Toolchains

The Intel XScale PXA255 CPU with its ARM core has several toolchains available, and the manufacturer recommends both the gcc-3.3.2 and gcc-3.4.0 compilers. By publication time, additional sets of tools will be added to the ones listed here. The variety of tools is a useful aspect of the GumStix, because not all tools provide the same options and utilities. Further, because most companies have preferred toolchains, and many of us have our own preferences, not being tied to a particular toolchain is an excellent feature.

Don't believe the GumStix manual when it states that it takes 30 minutes to download and install the toolchain and to create, install and run the ubiquitous HelloWorld.c. The time it takes to do so depends on many variables, such as, which toolchain you select and how much horsepower the host has. Finally, an MMC adapter on the host is recommended by GumStix, but I found it to be absolutely essential.

The uClibc toolchain already was installed on my machine, so it was not necessary to reinstall it, but I tried to make sure it worked. I was not surprised to find it did not, as it was unable to resolve a server for one of the components. I brought this point up with GumStix in a conversation, and the rep said the company was preparing a new set of tools to resolve this and some other issues. At that point, however, the new tools were not ready for review, which was a relief because I found uClibc temperamental to configure and install.

Embedded

One nice feature of the GumStix is the option not to have the Bluetooth capability. That might sound strange to those of you new to embedded applications, but there are many reasons not to want this feature. First, don't pay for something you don't use. Second, the absence of Bluetooth allows one to reduce the overall complexity of the devices, making them more reliable. Third, the Bluetooth module consumes power and processor time. With Bluetooth being optional, you can develop your application and then drop Bluetooth and use a simpler replacement, without having to worry about compatibility.

One point on the GumStix design: as mentioned earlier, the Molex connector was replaced by the Hirose connector. This is a real improvement, as the Hirose is more solid than the Molex and makes the GumStix-daughterboard connection much more stable. Mechanical stability definitely is an issue with the GumStix. At present, the connector is the only means of physical stability between the GumStix board and any daughterboard. This definitely is not an optimal arrangement. Hopefully, GumStix will add a drill hole or at least call out some locations on the silk screen where holes could be added or glue points might be placed. Mounting the GumStix in high-vibration or impact applications will make this a must. I found that even fairly mild handling could dislodge the daughterboard enough to make the connection fail, unless the case was securely attached.

Web

GumStix comes with BusyBox preloaded. BusyBox is an embedded application package with a large number of tools, one of which is a Web server. To use it, install your site at `/var/www/html`, reboot the WaySmall and off you go. One immediate application is to add a Web interface to your embedded application. Additionally, one could build a dedicated Web application for a WaySmall and essentially have an application in a box. Keep in mind that the Intel PXA255 has no floating-point unit, so number crunching is a stretch. However, several popular lightweight Web applications could run easily on a dedicated WaySmall.

Simple text-based HTML created with a minimal amount of graphics and no scripting was easy to accomplish. If you have an application that would not stress the server, you will be in good shape. The processor speed was more than adequate, but the RAM, storage and bandwidth were limitations on the evaluation unit. The 32MB of RAM is too little storage space for anything significant. The RAM is fixed, too, so you have to work around it. Storage is more flexible, however, with up to 512MB available.

Remarks

I suggest checking your intended host computer to see if it has a serial port, because a lot of newer machines delete them in favor of USB. I also suggest that you purchase the power supply, the 128MB MMC module and an MMC adapter for your host machine. Third, as indicated earlier, mechanical stability is a real issue with the GumStix. During the evaluation process, the antenna of the WS200f-bt became damaged and the Bluetooth failed. This occurred because the serial cable became tangled with the antenna. Additionally, when the host machine was moved, a load was placed on the PCB connector and a solder failed. Thus, the Bluetooth-equipped WaySmall may be too fragile for practical applications. This is a known problem, however, and will be resolved when the integrated antenna is added.

Bluetooth

Bluetooth is an excellent addition. The bandwidth of Bluetooth is substantially better than the serial connection and should be better than the USB 1.1 option. The Bluetooth-enabled models allow you to go wireless. They automatically boot to a configuration with rfcomm, generating a Bluetooth serial port called /dev/rfcomm0, and the startup script starts a getty over it. I was able to establish a serial connection over the Bluetooth, and it was faster than the USB 1.1 connection.

What Next?

GumStix is refining its products rapidly, but I am going to make some predictions:

- First, look for an integrated antenna. As a part of this change, I expect to see the serial ports dropped in favor of serial-over-Bluetooth.
- Second, expect to see Ethernet added to the GumStix, certainly an Ethernet-enabled device with a connector off one end. I would prefer to see wireless Ethernet in lieu of Bluetooth, but that is my preference.

Wrap-Up

The bottom line is the GumStix SBCs are cool. Their ease of use, small sizes, low power consumption and flexibility make them excellent choices for a wide range of applications. GumStix are good alternatives to most of the other SBC form factors presently available and should be given serious consideration for any new embedded development efforts.

Michael Boerner is a consultant based in St. Louis, Missouri. He likes to focus on embedded Linux and device drivers and can be reached at michael@boernerconsulting.com.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

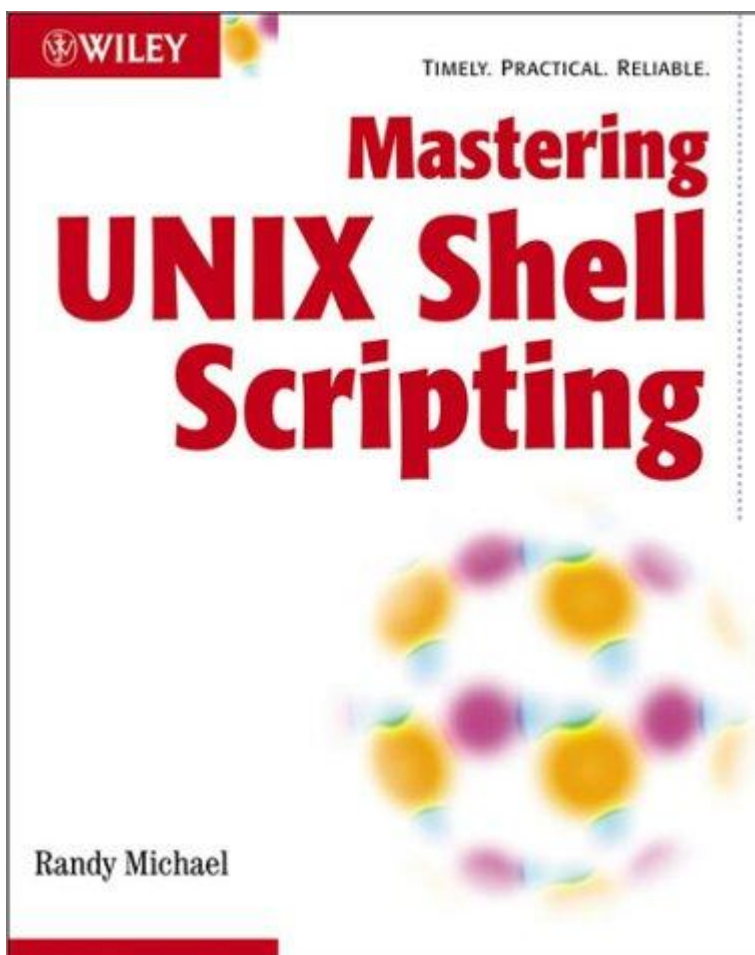
Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

Mastering UNIX Shell Scripting by Randy Michael

Marco Fioretti

Issue #127, November 2004



John Wiley & Sons, 2003

ISBN: 0471218219

\$45 US

This Wiley Publishing tome is big (more than 650 pages), useful and very complete. Be warned, however, that its scope is limited to system administration. The purpose of the book is to solve “real world...problems for those who have to automate these often complex and repetitive tasks”. Little information is directly employable for end-user tasks, such as printing booklets, mirroring Web sites or searching through e-mail.

The author has a lot of experience with shell scripting, and it shows. The scripts to solve each problem are well written and discussed line by line. For each, a preamble gives the big picture or introduces some command that is used in the rest of the chapter.

Anyone interested in scripting for maximum portability or ease of maintenance can learn a lot from this book. Everything needed to use the same script with Linux and all varieties of UNIX is present: Linux, Solaris, AIX and HP-UX have one subsection each.

The first chapter is a quick tutorial of shell scripting and a summary of all the techniques discussed later. The second one goes head first into deep scripting mode, setting the pace for the whole book. It offers 12 different ways to read a file line by line, including benchmarks to find the fastest one.

The most arcane shell commands and options are explained with plenty of examples. “Here” documents, a way to feed input to a script or command within the script itself, are explained thoroughly. Readers learn more than they could imagine about traps, typeset, getopts and other techniques for managing command-line arguments.

System monitoring receives the most coverage: several chapters explain how to detect and report problems in processes, disk space, memory and CPU usage.

Other important administration activities have their own sections. The author moves with ease from system snapshots to print queues, automated FTP and building sudo from source. Several methods to add menus and progress bars to shell scripts are explained. Floating-point math, number conversion and generation of random passwords and numbers also are covered. The volume ends with 45 pages devoted to sending pop-up messages from UNIX to Microsoft Oses. All scripts are available for download at the Wiley Web site area devoted to the book.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

From the Editor

Got a Linux Server? Thank a Beowulf.

Don Marti

Issue #127, November 2004

Ten years ago, Donald Becker and Thomas Sterling built a 16-node cluster, the original Beowulf, and started Linux and commodity hardware on a program of relentless improvement.

In May 1965, IBM Chairman T. J. Watson, Jr., wrote of large scientific computers, "at some point between two and three years ago it became evident that the fallout from the building of such large-scale machines was so great as to justify their continuance at almost any cost" (on Dr Mark Smotherman's site at www.cs.clemson.edu/~mark/acs.html). That doesn't mean that high-performance computing (HPC) startups have an easy time entering the enterprise market. From Control Data to Thinking Machines, history shows that if you concentrate on winning in HPC, you don't get the skills to cross over to regular business customers.

But ten years after the first Beowulf, examples of what Watson called fallout are everywhere on the Linux scene. From smoking out bad power supplies to fixing device drivers to making manageability work for the PC architecture, HPC customers are ruthless in demanding tweaks to turn a rack of off-the-shelf stuff from a maintenance nightmare to an asset. For an IT vendor, an HPC program can work like an automaker's racing program to test cutting-edge ideas and get everyone fired up to win bragging rights.

Early Linux clusters were labor-intensive, with "crash carts" including keyboard and monitor for BIOS access. Today, LinuxBIOS makes the pit crew's work feasible for more and more machines per administrator. See Bernard Li's article on how to take advantage of years of cluster experience from some of the biggest, most innovative Linux supercomputing sites (page 52).

And, forget about manageability for a while—let's talk performance issues. Paul Terry, Amar Shan and Pentti Huttunen might have just sped up many people's work by a whole lot. Check out their scheduler performance numbers on page 68 and prepare for more efficient work on your parallel jobs.

Leigh Orf has some great imagery of thunderstorms, rendered just for this issue, and the software behind them is something you can download and hack yourself. Get some ideas about scientific visualization on page 62, and send us some images.

This issue isn't all clusters—Andres Benitez and Vicente Gonzales show how they turned inexpensive non-networked air conditioners into a money-saving system for a classroom building (page 44). And Nick Moffitt, whose spam-fighting articles have been a hit on the *Linux Journal* Web site, is here with an introduction to a new, flexible revision control system (page 90).

Whether you're putting together a cluster or enjoying the benefits of Beowulf-driven improvements in hardware, Linux and related tools, have a great time experimenting with all the amazing technology and cool projects in this issue.

Don Marti is editor in chief of *Linux Journal*.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search**Real Blogs, Real Examples**

When I saw the letter by Geraint Williams in the August 2004 issue of *Linux Journal*, although I was glad to see a reply defending free speech, I still was surprised that a technical article would have been illustrated by an example containing controversial political content. But, then I found that the example in question was an explanation by your author, Reuven Lerner, of why he had chosen the name Altneuland for his blog—because the United States seemed different to him after eight years of living in Israel. I'm sure the page appeared to many others as it did to me—entirely innocuous rather than dangerously controversial.

—

John Savard

Look Daddy, Tux!

I'm a Linux fan/geek, from Buenos Aires, Argentina, and I have been using Linux for more than eight years. I work as a Network Engineer/Consultant for Ericsson. I've been a subscriber to *Linux Journal* for three years. I'll renew after I go back to Buenos Aires and will wait every month for this amazing magazine that makes my day every time I receive it. Here is a picture of my small Tux fan. Now she's four years old, but at the time of the picture, she was 1 1/2. Every time she sees a penguin she says, "Look daddy, Tux is there! Tux! Tux!" That makes me laugh. Keep it up—doing this great magazine.

—

Francisco Puente



Busy Linux User

I couldn't resist sending a picture of my granddaughter, Savannah. I have no doubt she'll grow up on Linux and love it as much as she does my penguins. Thanks for a superb, top-quality journal.

—

Michal Ludvig



Penguin Fashions

Attached is a photo of my one-week-old daughter Sarka with a penguin slobber-jacket. Enjoy!

—

Michal Ludvig



Public Computer for Penguins

This costume is property of Linuxhelpdesk.net. We do an EU project on ease of use of Linux in Finland. See www.linuxhelpdesk.net, or contact jukka.penttinen@ncp.fi for more information.

—

Eero



Picking Out Hardware with Dad

Attached is a photo of young Robert and me planning baby's first computer before heading home from the hospital after his birth. How much beefier will the Ultimate Linux Box be when he has the motor control to use a keyboard or mouse?

—
David



Flat Is the New Up

I have been noticing an interesting trend in *LJ*: it's getting thicker! Congratulations. As a frequent reader of trade journals and hobby computer magazines, I know what that means to the publisher. To my dismay, some of my other trade journals are getting thinner. Some even have embraced Microsoft in order to provide content!

I agree with reader Robert W. Carter that *LJ* is the new *Byte* [see Letters, September 2004]. It is catering to my hobbyist curiosity and feeding my demand for up-to-date articles and reviews of Linux software and hardware applications. What you need now is to get Steve Ciarcia to write you a monthly column (is he still publishing *Circuit Cellar*?), and get Jerry Pournelle a cameo every once in a while. That will transform *LJ* into the nurturer of the new generation of computer hobbyist in the same way *Byte* did in the 1980s. Keep up the good work!

—

Anibal Morales

LJ hasn't gotten thicker or thinner in a while. *Circuit Cellar* is still around and knocks our socks off with great projects using inexpensive 8-bit processors. Check it out at circuitcellar.com.—Ed.

Bad Web Site, Bad, Bad

Why is horizontal scrolling required to read Don Marti's article "Breaking the Laptop Barrier" [www.linuxjournal.com/article/7698] posted August 3, 2004? I viewed several other *LJ* articles with Mozilla without any horizontal scroll issues, so I'm pretty sure it's the one article.

—

Scott

Someone posted a long URL in the comments, which messed up the layout. Watch our Web site for a redesign that will fix this and other issues.—Ed.

Airplane Painting Tool

I regularly buy *Linux Journal* in the local bookstore with foreign literature. Although I'm not in the IT business, I find a lot of articles interesting (my computers are all running either Linux or some variant of BSD). My father, who is an active radio control flyer, recently bought a new model airplane named X-Free. The decision about the painted decoration was quite easy because of the name, but The GIMP made it even easier. We took a photo of the unpainted airplane and used The GIMP to apply the finish and tweak it until we were satisfied. We then printed out the modified photo and my father copied the design to the real airplane.

—

Tomaz Solc



Penguin Papa's Photo

Here is a photo of my two boys wearing—back-to-front, of course—their GeekStuff Linux Tux baseball caps. I have them convinced that they are the only boys in all of Ireland with caps like these—walking open-source advertisements! This picture was taken last Easter at a small fishing village in the southeastern corner of Ireland. The water looks inviting, but don't be fooled. The air temperature was about 15°C. Brrrrrrr!

—

Paul



Good News on Wireless Router Code

The article "Linux on Linksys Wi-Fi Routers" in the August 2004 issue says:

Many similar wireless routers, such as the Belkin F5D7230-4, the Buffalo tech WBR-G54 and the ASUS WL-300g and WL-500g, all use Linux in their firmware, and the list expands daily. Unfortunately, none of these companies has complied with GPL requirements and released the source code.

Belkin has released its source code: web.belkin.com/support/gpl.asp. Perhaps it was after the article went to press. More information on hacking the firmware can be found at www.seattlewireless.net/index.cgi/Belkin_20F5D7230_2d4.

—

Brian King

Photo of the Month: More Penguin Cake

My husband is a computer programmer and uses Linux a lot. He also enjoys the game *Tux Racer*. When he commented that I make special cakes for the kids' birthdays and not for his, I couldn't resist attempting to create a Tux cake to surprise him. After seeing the Photo of the Month winner for September 2004, he suggested I submit this photo.

—

Margaret Haller



Photo of the Month gets you a one-year extension to your subscription. Photos to info@linuxjournal.com. By the way, *Tux Racer* is now available as an arcade game. Congratulations to the tuxracer.com team.—Ed.

More Radio, Please

I, for one, would love to see more amateur radio-related articles in *LJ*. We are a technically oriented group. There must be scores of others out there like myself.

—

Richard (WB2RAR)

Weather Maps via Ham Radio

Thanks for the September 2004 issue of *Linux Journal*. I truly enjoyed the radio articles. Please keep them coming, especially articles like the one on PSK31. I enjoy using Linux to decode different digital modes on the HF band. With the storms of this last week, I enjoyed using the Linux program HamFax to receive Weather Faxes from NOAA—I could have gotten the images from the Internet, but where is the challenge in that?

—

Richard

GPS Software Suggestions

Your September 2004 article on GPS was interesting, especially some of the science behind the technology. I'd like to point out two additional software packages.

1) nmead—written by Chuck Taylor. This reads GPS from your serial port and makes it available on a network port. His site also has a Java-based sample GUI (home.hiwaay.net/~taylorc/gps/nmea-server).

2) ntpd—Network Time Protocol. I personally submitted patches that add nmead support to ntp. I think it will be in the next stable release. Until then, the patches are available at trainguy.dyn.dhs.org/~jminer/gps.html. Thanks for another month of interesting articles!

—

Jon Miner

Hardware for TV Projects

I noticed in the Ultimate Linux Box article, August 2004, that future issues will have projects based on HDTV cards. Are you planning on covering the Hauppauge HDTV cards as well as the pcHDTV HD-2000 card?

—

John R. Klaus

We'll try to make all our TV projects useful on as much hardware as possible. But, if you live in the USA and think you might want to watch HDTV someday, get an unrestricted card now before the Broadcast Flag regulation goes into effect.—Ed.

Photos of Kids Are Fine

Because of the regular appearance of photos of children in the Letters section of *Linux Journal*, I WILL be renewing my subscription. By the way, I also like the photos of pets very much. Does the www.linuxjournal.com Web site have an "About us" page with photos of the writers and spouses, kids and pets? I think that would be pretty great too.

—

Rick Deschene

A new look for the Web site is coming soon.—Ed.

Gorilla Marketing

Like most Linux users, I always am looking for ways to raise awareness and further the proliferation of Linux in industry and at home. Being an employee of a very large company, I have experienced first-hand what it is like to do battle with an 800-pound Microsoft-Certified gorilla of an IT department. To date, I have lost far more battles than I have won on this front, but I have not given up the cause. As a result of the wins we now run a few Linux servers and have a desktop machine here or there, but it still takes "an act of congress" to get approval from the gorilla to install Linux for general-purpose computing.

My most recent tactic was to purchase a subscription to *Linux Journal* and have it sent to an influential member of our IT department. I cannot think of how I could get more Linux-proliferating bang for my hard-earned \$25. For the next 12 months, management will be presented with a wealth of information about the state of Linux. I honestly believe that if only one out of every ten *LJ* readers did the same thing it would go a long way to growing our very own 800-pound penguin.

—

J. Eric Pipas

More Radio, Please (Part II)

I really appreciate these articles [see the September 2004 issue]. I am new to Linux and was a ham way back when. I was also a "real" engineer (Electrical vs. Software Engineer now) when I started. These articles seem like a good way for me to get back into what I love and also to learn Linux. I have been playing around with Linux and see how the two will fit together well for my education in both. Thanks, and keep these articles coming.

—

Tom Richards

Tracing Tool Tip

I enjoyed the article on Linux hacking tools in the September 2004 issue. `strace` is an extremely useful utility for getting in between applications and the kernel to see what's going on. What's missing from the list is its companion program `ltrace`. `ltrace` allows the developer to monitor library calls made by dynamically linked applications, such as calls made to the C library or to GTK. This capability comes in handy when debugging or analyzing other people's code, so `ltrace` should fit right into any Linux developer's bag of tricks.

—

Ryan Underwood

Erratum

I received an e-mail from Mr Lyndon Tynes stating there was an error in one of the Best of Technical Support questions I answered in the June 2004 issue of *LJ*. The question relates to "Changing Desktop Environments" (page 66), and the correct reply should have been written as follows:

You can change the content of `/etc/sysconfig/desktop` from:
`DESKTOP="GNOME"` to `DESKTOP="KDE"` or `DESKTOP="WINDOWMAKER"`,
and your X Window System will start the corresponding window manager. The file that controls which window manager starts is `/etc/X11/xinit/Xclients`; take a look and study it.

—

Felipe Barousse Boué

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

UpFront

- [diff -u: What's New in Kernel Development](#)
- [Globulation 2:](#)
- [LJ Index—November 2004](#)
- [Mairix:](#)
- [On the Web](#)
- [They Said It](#)

diff -u: What's New in Kernel Development

Zack Brown

Issue #127, November 2004

Red Hat has decided to release their newly purchased **Global Filesystem** (GFS) clustering filesystem under the GPL. This project has had a checkered past. It started out as a GPLed project from Sistina, but the company changed its license in 2001 to the Sistina Public License, which required a licensing fee to be paid to Sistina in the event of source code redistribution. Among various outcries at that time, Alan Cox claimed the license change violated the copyright of his own GFS contributions, and the **OpenGFS** Project sprang up, using the last GPLed version of Sistina's code. During the next few years, Sistina made some effort to market GFS in its new proprietary form, including partnering with CommVault in 2003. In 2004, Red Hat bought the code from Sistina and has now re-released it under the GPL. Back in 2000, GFS was considered a likely candidate for inclusion in the official Linux kernel; now that it is available again, Red Hat is keen to submit it for inclusion once more. This time, it looks as though the code will stay free.

Linus Torvalds has proposed a new patch attribution convention, which looks to be achieving an early success. The goal, as he puts it, is to enable kernel developers to track the history of a patch, in the case of charges of copyright violation. There's no confusion over Linus' inspiration here. He and others have had to spend a lot of time debunking each of The SCO Group's charges of copyright violation. Doing so has, until now, involved much wading through

ancient mailing-list archives. Linus' suggested attribution system, which appears to be being adopted fairly widely after only a brief discussion, simply involves developers including their names on patches to indicate compliance with the kernel's license. Each patch will have one name for each developer who edits it before it is included in the kernel. This way, any future charges of copyright violation can be investigated by examining the patches themselves, rather than trying to trace each patch's history through mailing-list discussions. As of this writing, many developers are using the system, although so far there has been no need to use the newly gathered data.

Randy Dunlap has introduced a new wrinkle into the old idea of saving `.config` information in the kernel itself after compilation. His idea is also to include data on the kernel version, as well as the date compilation took place. The initial struggle to get `.config` information into the kernel was fraught with controversy, due to the argument that such data also could be stored outside the kernel. With that part of the controversy resolved, the suggested addition of Randy's new data is finding a much warmer reception. In fact, a number of developers have remarked that the idea itself should have been obvious long before. It seems a deeply embedded part of human nature that even the obvious often must have its initial discoverer.

Jeff Dike's User-Mode Linux (UML) is having some technical difficulties making its way into the official 2.6 kernel tree. Apparently, **Andrew Morton** is more than happy to accept Jeff's patches, although Jeff appears to be having trouble splitting the patches into acceptable chunks. Jeff himself has said that he has "painted himself into a corner" with regard to his UML work; the problem, he says, is finding the right tools to manage the patch split. The UML patch has become so large that splitting it up has become a major undertaking. This actually is a typical occurrence with large features. Often, insufficient effort is given toward preparing the patch for inclusion, and when the developers finally feel the time is right, they discover they have a ton of additional work to do before the patch even will be considered. This usually engenders much controversy. Jeff is no stranger to the issues involved, but even knowing the requirements, it is still difficult to split patches into atomic pieces that each either fix or implement a single thing. Clearly, UML is destined for inclusion in 2.6, but these difficulties may result in significant delays.

John A. Martin has maintained the CREDITS file for quite a long time and has been acknowledged in the MAINTAINERS file for that reason as well; but the CREDITS file now apparently has become self-maintaining, no longer requiring any specific maintainer. Linus Torvalds, Andrew Morton and the other kernel maintainers have taken over much of that role, and developer patches often include their own updates to the CREDITS file, without requiring anyone else to add them. The CREDITS file has, quite simply, become a fully adopted element

of kernel development. Back in the old days, when that file was first conceived, the task was much more daunting, because there were so many contributors not mentioned in it. Now that it has established itself, its current listing is much more accurate. John graciously stepped down when **Adrian Bunk** pointed out that a maintainer was no longer needed; although John said he would be willing to resume maintenance should the need arise in the future.

Globulation 2: www.ysagoon.com/glob2

Don Marti

Issue #127, November 2004

This real-time strategy game breaks away from the *Warcraft* style, where you select units and give orders. Instead, you create a building site or other task and specify how many of your amoeba-like units you want to work on it. Based on units you have available and their skill levels, the game will assign units and tasks.

There's a beautiful level editor that creates interesting random terrain, but the computer players are still fairly dumb. *Globulation 2* supports network play, and development seems to be happening rapidly.



LJ Index—November 2004

-
- 1. Compound annual growth rate (CAGR) percentage for Linux servers in China for the next five years: 49.3

- 2. Approximate number of developers who contribute changes to Linux on a regular basis: 1,000
- 3. Percentage of the above who are paid to work on Linux by their employers: 10
- 4. Percentage of the latest 38,000 changes to Linux that were made by those paid to work on Linux: 97.4
- 5. Billions of dollars generated by Linux for related products and services in 2003: 2.5
- 6. Approximate thousands of IT jobs listed on Dice.com: 49
- 7. Approximate hundreds of IT jobs listed on Dice.com that require Linux skills: 22
- 8. Percentage increase in above number over the last year: 190
- 9. Number of IT jobs listed on Dice.com that require Linux certification: 10
- 10. Position of Apache in Netcraft Web Server Survey: 1
- 11. Apache share percentage in latest survey (August 2004): 67.37

- 1: CCID Consulting, via Oracle
- 2-4: Andrew Morton
- 5: Morgan Reed, Association for Competitive Technology
- 6-9: Computerworld.com
- 10, 11: Netcraft, Ltd., netcraft.com

Mairix: www.rc0.org.uk/mairix

Don Marti

Issue #127, November 2004

“It's in my old mail somewhere” isn't good enough. Add a powerful search feature to your mail without switching mailers with this simple command-line tool. You can search for any text or for words that appear in certain headers.

To use Mairix, edit a short config file to specify where your mail folders live, then run it without arguments to build the index. Run Mairix with an item to search on, and it puts copies of search results in a match folder that you can browse in your mailer like any other mail folder.

On the Web

If you want to read more from *Linux Journal* authors, add our Web site to your bookmarks or RSS reader and catch their regular Web columns.

- “Linux in Government” by Tom Adelstein—how is Linux progressing on the local, state and federal government levels? What initiatives are underway that feature open-source software? Which government agencies are embracing open source and Linux, and what companies are helping them do it?
- “At the Sounding Edge” by Dave Phillips—the author of *The Book of Linux Music & Sound* explores new audio technology and helps you find the best tools for recording, mixing, editing and playback—even tools for your lessons and practice sessions.
- “OOo Off the Wall” by Bruce Byfield—you've made the office suite leap to OpenOffice.org. Now what? Learn tips and maneuvers for making your documents look better, and save time with tools and options not even offered in other office suites.
- “cat/dev/DiBona/brain” by Chris DiBona—what's on the mind of this man on the Linux scene whose distinguished résumé includes VA Research, Slashdot and Google? Fighting spam, showing how to get the most out of .org pavilions (the best part of tradeshows) and many other things.

They Said It

People's stereotype [of the typical Linux developer] is of a male computer geek working in his basement writing code in his spare time, purely for the love of his craft. Such people were a significant force up until about five years ago.

—Andrew Morton, gcn.com/vol1_no1/daily-updates/26641-1.html

Openness matters in two places. One is in source code. The other is in data.

—R0ml Lefkowitz, Director, Open Source, AT&T Wireless (Talk at the O'Reilly Open Source Convention)

Never before in history have we been able to see incumbent businesses protect business models based on old technology against creative destruction by new technologies. And they're doing it by manipulating the political process. The telegraph didn't prevent the telephone, the railroad didn't prevent the automobile. But now, because of the immense amounts of money that they're spending on lobbying and the need for immense amounts of money for media, the political process is being manipulated by incumbents.

—Howard Rheingold, www.businessweek.com/bwdaily/dnflash/aug2004/nf20040811_1095_db_81.htm

This adoption of free software to resolve incompatibility between the economic need for provider diversity and the engineering need to avoid product diversity is, I think, fairly unique across all industry. I can't think of similar examples.

—Andrew Morton, www.groklaw.net/article.php?story=20040802115731932

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

New Products

FlashDisk OpenRAID, CommuniGate Pro 4.2, Unisys ES7000 Servers, Wyse Winterm and more.

SC-4400 FlashDisk OpenRAID

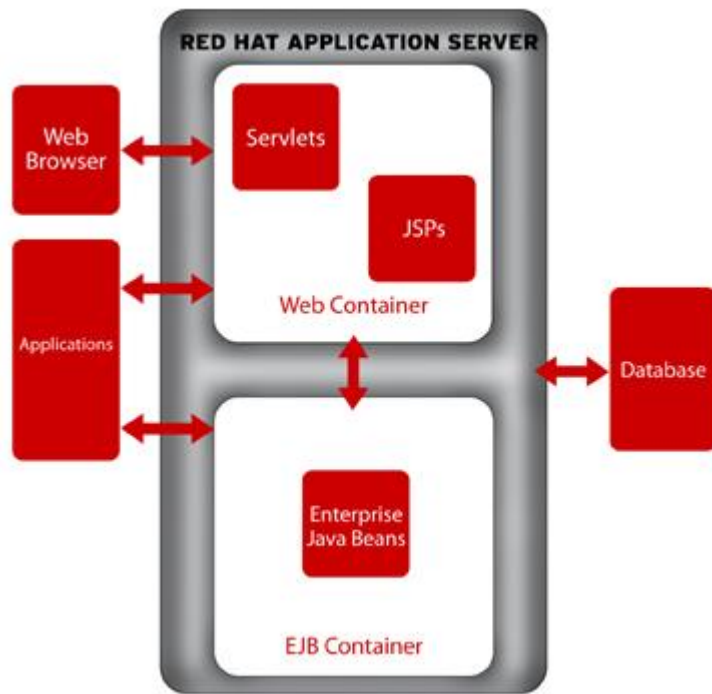
Winchester Systems announced the release of FlashDisk OpenRAID, a data storage array capable of supporting up to five heterogeneous servers. The servers share a storage pool by way of the U160 SCSI protocol, capable of scaling to 2.3TB. With the addition of an expansion chassis, FlashDisk OpenRAID can scale to 4.6TB, supporting up to three hosts. It supports dual redundant 64-bit RAID controllers and can be used for various disk-intensive tasks, such as database, multimedia, e-mail or Web server work. The SC-4400 is compatible with any server or OS and has no host software driver requirements.

Winchester Systems, 149 Middlesex Turnpike, Burlington, Massachusetts 01803, 800-325-3700, www.winsys.com.

Red Hat Application Server

The Red Hat Application Server is an open-source middleware platform for Java 2 Enterprise Edition (J2EE) applications. The server includes a runtime system and development libraries and is tested and supported on all major Java Virtual Machines, including Sun SDK, BEA WebLogic JRockit and IBM JDK. It also is certified with databases, including Oracle Database, IBM DB2, Sybase, PostgreSQL and MySQL. Red Hat Application Server includes JOnAS, Tomcat, Struts, supporting modules for file uploads and tutorials.

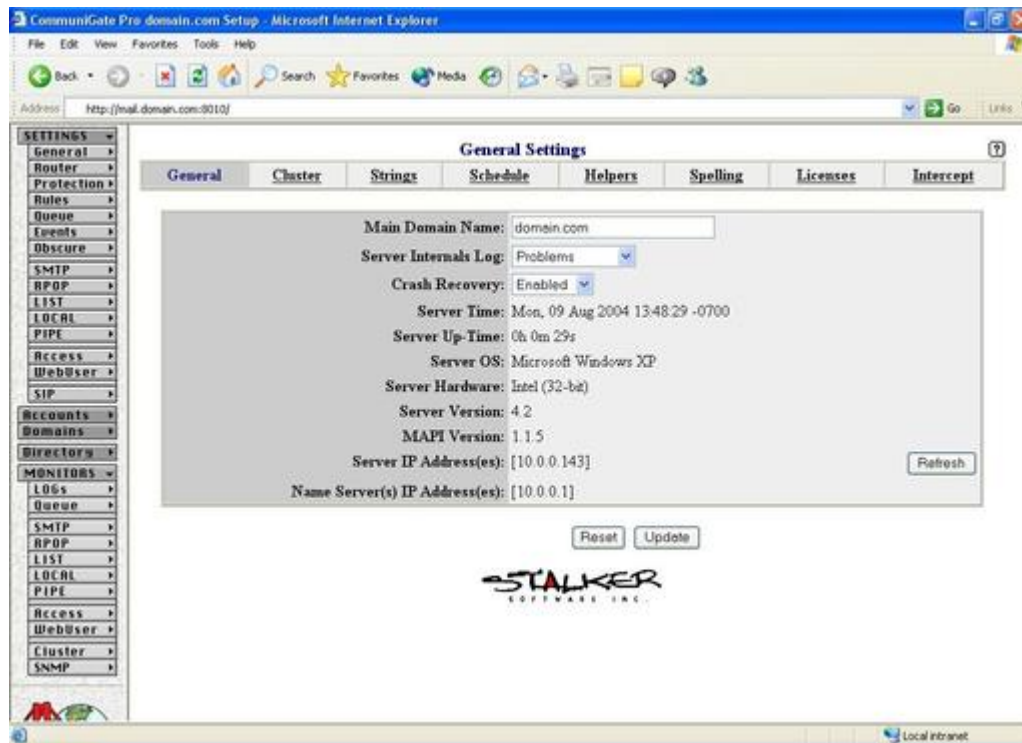
Red Hat, 1801 Varsity Drive, Raleigh, North Carolina 27606, 888-733-4281, www.redhat.com.



CommuniGate Pro 4.2

With the release of version 4.2, CommuniGate Pro Real-Time Communications now offers secure instant messaging, VoIP, video conferencing, whiteboard sharing and desktop and application sharing capabilities. Using a standards-based, cross-platform design enables CommuniGate Pro 4.2 to be utilized by any client, whether in office or remote, using session initiation protocol (SIP) on UNIX, Linux, Mac OS X or Windows. Also new for v4.2 is the choice of a fully customizable or preconfigured Web interface for Web-based mail and calendaring functions. In addition, CommuniGate Pro 4.2 offers remote administrative control features for troubleshooting, maintenance and updating duties.

Stalker Software, Inc., 655 Redwood Highway, Suite 275, Mill Valley, California 94941, 800-262-4722, www.stalker.com.



Unisys ES7000 Servers

Unisys announced a new server line, ES7000 Servers for Linux, available in both 32-bit and 64-bit configurations. The ES7000 servers can include 4 to 32 Intel Xeon MP or Itanium 2 processors, 4 to 512GB of memory, 16 to 256MB of shared cache and 8 to 160 I/O slots, both PCI and PCI-X. Servers can be run as one single-image system, or they can be configured with as many as 12 logical partitions running several different OSes concurrently. ES7000 Servers are certified to run SuSE Linux Enterprise Server, Red Hat Enterprise Linux AS or both.

Unisys Corporation, Unisys Way, Blue Bell, Pennsylvania 19424, 215-986-4011, www.unisys.com.

Wyse Winterm 5150SE

The Wyse Winterm 5150SE is a thin client powered by the Wyse Linux V6 operating system and running on an AMD Geode GX 533 processor. Wyse Linux V6 is based on the 2.6 Linux kernel and offers customization for specific needs across a wide variety of platforms, including Windows, UNIX, Linux, IBM, X-Windows and Java. The Winterm 5150SE offers free seating capabilities so users can log in to different clients without losing their specific configurations. The modular design of the 5150SE allows features to be added and removed as needed. The thin client features a read-only filesystem, no moving parts and a compact chassis with USB and legacy I/O ports.

Wyse Technology, 3471 North First Street, San Jose, California 95134,
408-473-1200, www.wyse.com.

Atigo Wearable Computers

Xybernaut announced that its Atigo line of wireless panel computers now are available with Linux. Atigos can be used as wireless flat-panel display computers or standalone wireless-enabled mobile/wearable computers. Atigos support dual-use functions and are configured with built-in IEEE 802.11b WLAN wireless networking support through standard PC card and/or CompactFlash slots. Atigos with Linux also offer open-source tools for support, standard communication protocols, data management and system configuration. The Atigo T model uses a Crusoe TM5800 processor with a 1GHz CPU and offers 256MB of SDRAM and Flash memory configurations of 128, 256 and 512 MB or 1GB. All Atigos have internal rechargeable Lithium-ion batteries and optional hot-swappable external batteries. They all have 8.4-inch touchscreen-enabled 800 × 600 SVGA displays.

Xybernaut Corporation, 12701 Fair Lakes Circle, Suite 550, Fairfax, Virginia 22033, 703-631-6925, www.xybernaut.com.



[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.