NYLXS Journal

June 2002

Vol 1, Num 2

"When talented people gather together for the purposes of developing opportunity for the comminuity and members great things can happen.

Ruben I Safir

Table of Contents

INTRODUCTION TO NYLXS

By Kevin Mark

NYLXS is an all volunteer organization dedicated to the continuation of free software, free speech protection and GNU/Linux. This is being accomplished through the development of leadership and low cost technical proficiency. We aim to convert schools, businesses, and non-profits by showing the cost, security, and organizational benefits of the free software revolution.

This will be accomplished with various initiatives: a 'free software' institute which includes low cost cradle-to-grave GNU/Linux training, an in-services lecture series, a public school Free Software initiative, monthly installfests and online mailing lists. We have many meetings during the month were members help shape the development of our initiatives. We also have online mailing lists which include lively discussion of linux

HW/SW questions, free speech issues, open source politics.

Installfests are free invitations for people new to linux to BYOB (bring your own box) and have gurus from our membership help you get you on

your way to the freedom of linux , maybe have a beer (virtual or otherwise) and mingle with others in the free software community.

The New York Linux Scene Journal

by Michael L. Richardson

The New York Linux Scene Journal's main goal is to Record, Educate, Inform, and Promote Linux in the New York Area (for those of you not from Brooklyn that means the rest of the World) The Journal will be published quarterly. It will be on-line and at least one issue will be printed.

We are calling for papers in the NYLXS Journal. The Journal's looking for serious papers on Free Software research or implementation, programming practices and desktop usage's. If you work with free software, Linux, Perl, Lisp, PHP, Database design, Firewall Network. Get yourself published and increase your profile.

We need the history of Linux (as seen from those who were there), Stories from those who found

Linux later, how the Linux Users Group you belong to got started and the group's history. This includes Sigs, Lugs, High Schools, Colleges and Universities and business that use Linux or Free Software.

Visions of where you see Linux, Free Software, and GNU going. How has Linux made a difference to you, your school, your job, your business and your life? In short we need to explore the world of Linux and Free Software (were it's been, were it's at now, and were it seems to be going).

Be a part of the newest and hottest group to hit the Linux and Free Software world. To those who wish to help (or just see their name in print) please e-mail Michael L Richardson at miker@nylxs.com

NYLXS Journal Staff

- Michael L Richardson Journal Chair/Exective Editor
- Jon Babor Head Editor
- David Katz Senior Editor
- Jontathan Jocobs Editor
- Paul Winkler Editor
- Marcia A Baczynski Editor
- Joe Villari – Web Interface

The following have submitted articles or reports.

Ruben I Safir, Davis Sugar, Brendan W. McAdams, Joe Grastara, Marco Scoffier, Kevin Mark,

Paul Rodríguez, Brian Sietz, Marcia A Baczynski, Dennis Bella, Jon Babor, Michael L Richardson

Our thanks to all those who gave suggestions for this journal.This Journal and its contents are

THE NYLXS JOURNAL, publishies quarterly with at least one issue in print.

To contact NYLXS e-Mail miker@nylxs.com

NYLXS Political Action Report: The Fair Use Bill

by Ruben I. Safir

Following the recent successes of NYLXS and NY Fairuse in pursuing our contacts and lobbying efforts in Washington DC, NYFairuse has created an outline for a new bill which we hope to get written and to push through Congress. Over the last quarter, Free Software has faced many political hurdles, many of them threatening the legality Free Software on privately owned computers. Questions have been raised on the hill about the need to protect digital television broadcasts from copying, and new initiatives have been presented to both the Commerce subcommittee and the Intellectual Property Subcommittee in the House of Representatives. Because of the pressure being put on the Hill by NYLXS, the EFF, and other organizations, Congressman are no longer giving this a free pass on the Hill. They are now starting to challenge media companies on their never ending push to lock down computers and to spy on the public.

NYFairuse is ready to take this fight to the next level, and we are sponsoring a Fair Use Bill designed to empower the lawyers in court against unjustified attacks on civil liberties. The Fair Use Bill, in rough outline, will explicitly state that Fair Use is an extension of the 1st and 4th amendments of the Bill of Rights. In addition, it denies the right of copyright holders to interfere with the property rights of consumers through digital means without a written and signed contract between the parties. Most importantly, it defines legally the term "stake holders" in copyright law as the general public.

We hope to have this Bill written in the next few months, and can use help in this matter from a lawyer. NYLXS hopes to fund this process. The basic outline is as follows:

Consider it a Digital Bill of Rights for the New Millennium:

The legislation to be drafted will accomplish the following main stream objectives which all reasonable people can expect:

- -All copyrights to individual scores, writings, and recordings will be returned to the original artist after a period of 10 years.
- -No technology can be deployed which spies on, wiretaps or discloses privately owned information which is stored on digital devices by any government agency or private 3rd party without the issuance of a publicly pronounced and disclosed warrant limited to a specific criminal investigation.
- -All copyright cases must prove, prior to a judgment of guilt, proof that the actions in question did not infringe on Fair Use, and the individuals rights under the 4th and 1st amendment of the Bill of rights US Constitution.
- -Ownership of all physical media, and devices to read such media, is the sole property of the purchaser of the media, without an expressly negotiated and signed contract between both the copyright holder and the purchaser.
- -No technological software or hardware method can be deployed in a digital product and made available for normal retail sale which inhibits, in any way, the full enjoyment of the property by the purchasers, regardless of any agreement between the designer of the hardware or software products. Such agreements are null, and not contractible.
- -Copyright is an exception to Fair Use as it limits the ability for individuals to enjoy their private property and express themselves with the use of such copyrighted materials. Fair Use is a doctrine to be based on the 4th and 1st amendments of the US Constitution.
- -Individuals have the right to express themselves to others about the means, mechanism and workings of all digital devices, including but not limited to, discussion on how to make fair use of media, how to improve such devices, or to reverse engineer all such devices and the algorithms which are used to help them display, copy or run media.

We need to get as many big guns on this as possible and then relentlessly campaign, actively working to elect supporters and vote out incumbent opposition. In fact, we should look to defeat, not just the proposed spyware legislation, but also defeat Senator Hollings

Remember that all politics is LOCAL.

Ten Reasons Why Fair Use is Important to Geek Linux Users

by Ruben I Safir

In the summer of 1991, in a story now known to almost anyone familiar with the computing industry, a young Linus Torvalds was finishing up the prototype of his revolutionary new operating system kernel in Helsinki, Finland. In what is very much an American-like tale, the young Mr. Torvalds, with little more than his talent, his determination, and a vision, began a quest that fateful summer which would succeed beyond the anyone's wildest expectations. With the help of his friends on the internet, Linus let loose upon the world a software design which would build into an industry, and his vision has been responsible for the generation of thousands of jobs world-wide, feeding families and empowering individuals to produce sustaining wealth through freedom of the human spirit. Disney could not write fables as perfect as this.

However, the benefactors and leaders of Linus's vision, need to come to grips with the essential truth that the conditions upon which Helsinki Miracle was dependent are threatened today. If any of these conditions had not existed the development of Linux would not have happened. The three essential conditions did exist in the summer of 1991: political freedom, technological maturity, and economic opportunity. In 1991, all three of these things nicely converged to produce Linus and Linux.

Before 1991, independent breakthroughs such as Linux could not have taken place in the voluntary atmosphere of free software because the networking technology needed for worldwide remote cooperation simply hadn't been matured enough to facilitate the social process that Linux embodies. The community couldn't form because the information infrastructure had not yet existed for it to do so. Once this technological barrier was overcome, a technologically based hackers community was inevitable.

The second condition that existed in 1991 was that hardware in the personal computing space was largely compliant with opens standards. Many regard this as an accident, but when we examine the development of the PC computer, especially the governments insistence of a competitive industry in PC hardware without monopoly, the standardization was a natural outcome of the political expectations of the day. Compaq's cleanroom reverse engineering of the IBM PC was legally protected by law. This did not have to be the case. Compaq's reverse engineering could have been viewed as a patent or copyright infringement. Had it been so, it is unlikely that Linus would have had enough information about the specifics of PC hardware to create Linux.

Nor would he have the right to access the needed device drivers to make the affair worth pursuing. In addition, even if he had gotten past this barrier, use of closed protocols could have been viewed as a violation of the copyrighted material and exploitation of trade secrets. The only thing that prevented this obstruction of Linux, aside from government anti-trust action, was 'Fair Use'. Fair Use assured the political freedom necessary for Linus to produce Linux.

The final thing that made Linux possible was a proper economic environment to stimulate innovation. That Linus was able to afford the necessary education, which gave him the background to be able to investigate the construction of an operating system, is often overlooked. Linus was not only a well fed graduate student when he created Linux, he was also able to afford a PC to hack on, and money for books, and money for information on computer sciences.

He was in an environment that could afford to freely offer copyrighted information.

He was able to access others' works in his production of the kernel. Journals and Libraries promoted his efforts both through publication of the resultant works and through a steady feed of reliable information. The economic conditions that Linus experienced were designed to encourage Linus to exploit his potential as a world citizen and in so doing, added to our own heritage.

The title of this essay is "Ten Reason Why Fair Use is Important to Geek Linux Users". Nothing has been said that appears to address the title.

What needs to be understood is that Linus, in today's environment, would not be able to produce Linux. The Linuses are facing an escalating hostility to future Linuses. Free enterprise in the computing field today is coming to an end. We are destroying the right to innovate.

When the internet erupted onto the public consciousness, the potential for acquiring information became immediately apparent to any of its users. Internet access has changed the way people today think about acquiring information. Back in the old days, in the dark ages of the middle 1980's, we toiled in a limited world, limited to our block, our families, our town and our friends. Reliable information was scarce and expensive. If I was discussing with someone the historical background behind the painting of the Mona Lisa, my options were very limited. I could go down to the local library and look up the Mona Lisa in the card catalog, hoping to find a book on the topic. Failing that, I was stuck. Specialized libraries on Art History were not available, and institutions lucky enough to have such a detailed and specialized library limited public access. Journals on specialized material were expensive when they existed, or unarchived or unavailable. More important, finding a community of people with expertise in the area of interest required nothing sort of a lifetime academic membership in an reputable institution of higher learning.

The internet has turned all that on it's ear for two reasons: the cost of publication on the internet is very low and the freedom to share ideas is protected. Previously, when someone had an expertise on a topic, or a passion for a topic, that person would have years of research and generalized knowledge on the topic locked up in his person, with little or no way to make the information available to an interested public. This individuals full potential is bottled up.

First he needs to generate a manuscript in the hopes that the manuscript fills a particular need in the marketplace of information. It gets submitted to a gatekeeper who controls the flow of information to the public. This gatekeeper is either a publisher, or a program manager. Censorship of the work is then applied to the work for the benefit of the publisher. Finally, if everything goes right, the censored information is made available to the public unidirectionally. Feedback is neither required or desired. The closest we get to peer review or public conversation is the censored Letter to the Editor section of the press and the heavily censored talk radio phenomena which started in the 1970's.

Obviously the process is fraught with abuse and efforts to correct the problem in the traditional media have been unsuccessful in the face of a demoralizing assault by private economic interests on the public space. Spin doctors belittle the use of public set asides in the media. Most of the efforts are being driven from public view. Public Access television, editorial broadcasts, and even guarantees of children's television programming, and balanced political air time have been removed from our publicly owned broadcasting channels by the private interests which have been entrusted with them.

Over time, failure of the public to protect the public domain, in combination with a quirk in our available technology, has eroded ourparticipatory civic lives. The foundations of Democracy are dissolving. Until now, there have been two means of mass communication: the Telephone and Broadcasting. Broadcasting has always required large capitalization by centralized organizations to successfully publish. Examples of this are Movie producers which need production budgets and distribution channels, Book Publishers which need print houses, advertizing budgets and book outlets, and Television Broadcasters who need radio equipment and studio facilities. All these technological factors prevented individuals from participating in publication.

The telephone as a form of mass media has been more personal than Radio, Television and Book Publishing. The telephone connects two individuals such that each party is a receiver and a publisher of information. This wildly successful invention encourages people to express themselves to others. For example, "telephoneitis" is a common disease that inflicts young people between the ages of 9 and 17. Its onset is a painful experience every parent comes to dread. And it is closely related to the new malady "chatroom addictionitis" which is sweeping bedrooms and study halls across America.

Before the internet the 20th Century was a century of decreasing literacy. We were increasingly reliant on receiving information through the mass media, and decreasingly reliant on the written word for personal communication, our generation is just not as literate as that of our grandparents.

The internet changed this and today we are experiencing the greatest increase in general literacy in a hundred years.

The internet is a convergence of traditional mass media and the telephone Who among us today is not a publisher? Today our appetite for information is enormous and our quest to provide information has reached an unprecedented level. The growth of the internet has turned the global village into the global street corner. Today, everyone lives on the same block. We all share the same stoop. And we have discovered that we all have a need to be heard and to share. The ability to cooperate, share and trust to the degree humans do is without parallel elsewhere in nature. Co-operation and communication are a genetically tattooed traits that have enabled and defined us as a species. It is what make us human.

But we have reached a critical stage in the history of Mankind. A new dawn for mankind is on the horizon. And whether our new technology will continue to be a tool in enabling us or become a tool to shackle us is very much in question. GNU/Linux users and hackers are at the very core of the debates and choices we will make for our future. Economic interests that are built around unidirectional mass media, which have abused the public trust in the past, perceive the internet as a threat. They see that the new technology is capable of ending their monopoly on publishing information.

The publishing industry is willing its ability to influence lawmakers to strip all privacy and every aspect of the control of information from the public. And it wants to use technology to do it. Companies are demanding and receiving the privatization of human civilization itself by attempting to gain complete control of the new venue of human culture: the modern computer. In doing so, they seek to make it a criminal act to communicate about computer infrastructure, or to explore the foundations of the technology that we use each day to interface with others. Future Linuses will have to contend with laws designed to prevent the free exchange of information. Exploration and reverse engineering is today a felony.

Many of us in the computing field are well aware of the issues involved. We have vented our frustration on slashdot and other venues at the growing threat to the free flow of information. When the group 2600 was taken to trial for the publication of the DeCSS source code in New York in the Summer of 2000, many hackers showed up to protest the infringement of fair use and free speech by the Digital Millennium Copyright Act of 1998. But the community has not yet been able to organize effectively, and it needs to be effective politically to protect itself and our free society.

Even worse, there is a large segment of the GNU/Linux User community that considers discussion of copyright and fair use outside the scope of the discussion of GNU/Linux. When the topic is opened for discussion on various Linux User Group mailing lists, some list members complain that the issue is noise. Even the much exalted Silicon Valley Linux Users Group suffers from this. NYLUG, the New York Linux Users Group, and LXNY, her sister free software group, are active in this area, and it reflects on their mailing lists, but even they suffer from this same malady. So, we must enumerate the reasons that political action on Fair Use issues is essential. All GNU/Linux users need to understand that the freedom they experience using Free Software - whether it be software like GNU/Linux, or Software of their own design is something the modern mass media oligopoly is trying to extinguish with a frightening lack of regard for our constitutional rights.

Here are the 10 reasons every Linux User must join in political action to protect the right to develop and use free software:

1: Computer Engineers and Programmers must have the freedom to practice their trades. This right is being jeopardized as access to software and hardware is diminishing. Software intelligence systems are on the drawing boards designed to circumscribe and to diminish the information and access needed to touch or interact with software and hardware systems.

[http://www.theregister.co.uk/content/2/15684.html]

[http://www2.mrbrklyn.com/solutions200.html#two]

This is as if we were to say that a Carpenter was allowed to use a hammer only with size 6 IBM brand nails, but was not allowed to use his own hammer on IBM size 6 nails, even if he built his hammer himself.

2: Computer Programmers must have the right to freely use their property as they see fit. This is the basic question of who owns your computer, software and media: you or the software publishers and media companies? You do, obviously. If someone steals your computer, Time Warner will not call to police to report the crime because you had an mp3 of a performance they owned the copyright for, nor would IBM press charges against the thief because their hard drives were installed on your system. That you have an mp3 of a copyrighted performance of a song or a certain brand of hardware does not, and should not, give the copyright holder or hardware manufacturer ownership of your computer. But IBM and AOL/Time Warner want exactly that. They want the legal right to prevent you from reading the information stored on your hard drive and prevent your access to its data.

If they get all they want from congress, they will soon be able to completely prevent the installation of any software on the hard drive, including the OS, without complete prior approval of the manufacturer of the system. There are already systems in place that force you to buy software that you don't want. In the future, it will be illegal to even remove that software from your system to install different software, and the software will turn itself off if a subscription-like fee isn't paid. Essentially, they want to own and control your computer after you buy it.

3: Computer Programmers must have the ability to innovate new software design. Innovation is never unrelated to previous works. Isaac Newton said it well, "If I've seen so far, it is because I have stood on the shoulders of giants". Newton was clearly one of the most original and creative thinkers in world history, but this might have been Newton's greatest observation. Even great thinkers like Newton could not have made innovative contributions to humanity without being derivative (no pun intended). The ability to improve existing software through study and experimentation is the fuel of software innovation. For programmers to innovate, they must must have fair use access to derive new works from previously published works.

4: Computer Programmers and Users must have free accesses to their hardware. Imagine the following Star Trek episode. The Starship Enterprise is under attack from an unknown and relentless alien spaceship. They manage to defeat the shielding and are boarding the ship. As all looks hopeless, the Captain barks out orders and commands the ships computer, "Computer lock out all command functions

- Pickard 185Alpha". The computer responds,

> **"Failure to comply. Copyrighted material discovered on your IBM hard drives. You are not permitted to block access to this area of your computer.**
>
> **You are not the copyright holder. This attempt to block access to copyrighted material by the copyright holder is not permitted. This incident is being reported to IBM and the content owners. User interface to communications is being locked out and IBM is being notified through subspace."**

Please see current licensing agreement for further information" The damage of having unregulated, private control of such a vital communications tool is so obvious that it amazes me that anyone in a responsible government position can even consider it. Yet congress is seriously considering legislation that would prevent you, and even high-end computer experts from protecting yourselves against eavesdropping and other abuses.

5: Computer users must have uninhibited access to media they own. Linux users are hurt in two ways when they don't have uninhibited access to their media: First, if the media they own is inhibited in it's use, then they can not develop needed software for systems they create. Second, restricted access locks them into purchasing software that they don't need or want in order to be able to use other software they do need. For an example of the second problem, let us look at the "vital books project" at the NYU College of Dentistry. A Linux User who is a Dental Student at NYU College of Dentistry will find that books are no longer available for his education in dentistry; instead, all dental textbooks are engraved on an encrypted dvd viewable only on proprietary hardware and software. He MUST buy these proprietary items for his courses and he is forced to both purchase more than he needs or wants and relinquish his right sell the materials if he no longer wants them. This is not an accident, but a stated goal of the project. (see http://www.vitalviewer.com/Harcourt.html below)[1] Our Linux user must by an entire second system, at considerable added expense, just to participate in his own, already very expensive education. This is a break down of the principles of personal property and of fair use. It prevents the use of anything other than officially approved computers.

6: Computer users must have the right to participate in the economy. Media outlets have been fortunate until recently. They have had the ability to push material on the public in a one way direction. Data goes from producer to consumer. Computers break that very profitable model for distributors of mass media. The public, in the old model, were consumer widgets. No thought was given to consumers as producers of media (or of anything else for that matter). Computers have unshackled people from the chains that forced them to endure the intrusive push of pre-censored media and advertizing. Experienced computer users quickly exchanged that prior passive role for a model of two-way interaction. No longer is it anecessary by-product of the economy that we suffer disenfranchised pockets of communities which interact with the large community only as consumers. GNU/Linux gives its users the opportunity to be producers in our economy and to become wealth builders. It allows users to produce software, images, networks, advertizement, animation, recordings, video, and more, without having to face cartel efforts to pigeon hole them into a consumers role, or crush their efforts with forced upgrades, unchecked license fees, and limited access to tools.

The ability of this growing subset of the Linux user community to join in our economy as producers is significant. It means people can hurdle the restrictions to fair use and, thus, gain full access to the cultural artifacts stored on their computers. Their consciousness no longer need be numbed, as it has been for decades, by the current pop culture with which they have been bombarded. But a collision between those who would have full access to their cultural artitacts and those of would seek to limit that access is inevitable. Those who seek to control their own use and enjoyment of the information, science and the arts, won't find that control. Under the existing DMCA, they'll find felony charges instead. DJ's using Linux to create music mixes will be felons. Programmers altering sound and images - or creating the tools to do so - will, likewise, be criminals. These groups, and many others, are crippled without access to materials for use in derivative works.

---

1 Vital Books Qoute: "The agreement between Harcourt and VSTi will allow Vital Source to bundle digital versions of Harcourt's Mosby, W. B.Saunders, and Churchill Livingstone dental school textbooks along with each participating dental school's internal curriculum course manuals on a single DVD. Using the VSTi VitalviewerTM XML browser, students and faculty will be able to quickly and easily access content across all four years of their curriculum enabling them to find specific subject matter in a highly structured, context-sensitive search environment. In addition to enhancing the student/faculty experience with reference and resource aids, the VSTi service creates a copyright compliant environment on campus, giving Harcourt Health Sciences the ability to maximize student and faculty use of their products in a cost-effective manner."

7: Computer Engineers and Programers must have educational freedom. The future of education looks bleak at the moment. And the future of computer engineering and programming looks bleakest. Unlike other students, who have general academic freedom, computer science students are faced with the prospect of having all knowledge on the topic of computer sciences sealed within encrypted DVD's accessible only on a pay as you go basis. In addition, all the materials and platforms they need to for legitamite research and discovery will be restricted. Today's legal and technical infrastructure is destroying our capability to produce tomorrow discoveries. It's even possible that in the near future there won't be enough trained people to understand even today's technology, or to access materials and information blocked by congresses' new found love affair with access control. Capable engineers and cryptographers don't grow on trees. They need freedom to be nourished developed.

8: Computer users must have a freedom of choice of software and hardware products. Computer users have different needs and abilities. Free choicebetween computer systems and client programs is essential. It is the only way to assure that it will be possible for users to discover the tools that will be best for them to use in applying solutions to their problems. Common standards for media formats are essential. Freedom to choose between products is threatened by hardware and software lockouts which prevent the free movement of information from platform to platform. These lockouts are designed to prevent fair use of copyrighted materials in the digital age. They are designed to prevent use of "unsanctioned" systems, programs and hardware. They are designed not only to control what you buy, but to control how you use what you buy after you buy it. They are designed to eliminate your ability to choose. To choose what programs you run, how you listen to music, how you watch video. We are facing a future where if you want to read a newspaper, you'll be required to buy special, regulated equipment to do it. And if you try to read a news story without reading each and every ad, the equipment will shut down.

9: Computer programers must have the freedom to innovate. Microsoft has been claiming this for months. Individual programers have needed this for years. Fair Use is the only thing protecting programers from being locked out of their systems by overreaching corporations and private concerns.

Fair Use provides the constitutional guarantee that users and programmers have full usage of their legally purchased machines and software.

Innovation demands uses of systems in ways that might not be "approved of" (or thought of) by manufacturers. Leaving the control of permissions the equivalent of Gallilo asking the Church for permission to point his telescope at Jupiter before being permitted to legally doing so. Like your computer? Like to use Linux? Like to write new interfaces for your hand held device? Your right to do so is called....Fair Use..

10: Computer Users Must have the right to "reverse engineer". This is the ultimate test of fair use; the ability to "reverse engineer" any system is fundamental to all of engineering and product development. And it is implicit that one doesn't need permission from manufacturer to do it. If you buy it, you own it, and you are free to take it apart and look at it. This isn't stealing. You can't build and sell something just like it.

That IS a right the original manufacturer has. But he can't prevent you from seeing how it is put together. He can make it difficult, if he wants to, but he can't stop you from looking. When systems are wrapped in encrypted firmware that limits access, then there is no chance of reverse engineering. And, that, in itself isn't a problem. The problem is that these companies are trying to get the congress to REQUIRE that ALL devices have such encrypted firmware and to make it a crime to have a device that doesn't. Even the current DMCA, as restrictive as it is, isn't that restrictive. There are provisions in the 1998 DMCA that allow for reverse engineering, but there are no means to do it under that law that are both legal and practical. Now, these companies, threatened by their own lack of innovation, want to restrict access even further. Access control has nothing to do with copyright or patents. No copyright law or patent law was ever intended to control the use of a product after purchase.

Free Software institute Class

How to break Mach one with SSL: A Cheap, Portable, Robust SSL Accelerator

by Brendan W. McAdams <rit@jacked-in.org>

About the time my firm was ready to go live with our new system (the aforementioned Java one), we ran into a small problem with our hosting facility. Namely, they declared Chapter 11. Not wanting to risk the various woes of being caught in a Chapter 11ed facility (like the possibility the investors would lock the doors to take inventory of assets, effectively locking our equipment in with it), we did what any sane person would do; we moved.

The problem we ran into though is we had an aggressive go-live date locked in with our move execution. It turns out that at our old facility we were renting SSL Accelerators which provided SSL capabilities to our application web servers (BEA Weblogic 5.1); a technical oversight left us without SSL capabilities as we found out the new facility did not have SSL Accelerators available for rent. We were left with four options:

Go live without SSL.

Delay our go live date until we worked out a better solution.

Buy an SSL Accelerator (basically an expensive hardware machine which speaks SSL to a web browserand passes normal HTTP out as SSL from an internal web server).

Add SSL to our Weblogic web servers by buying an SSL License.

It was essentially a no-brainer that numbers one and two were unacceptable. It just wasn't something we wanted to consider. Brief research showed us that numbers three and four were extremely expensive: most vendors wanted ridiculous amounts for their hardware SSL boxes, and BEA wanted our first born children to add SSL licenses to our web servers.

So after some thought (and a lot of alcohol), it occurred to me that all an SSL Accelerator is doing is proxying requests to the internal web server. I came up with a free solution involving Apache + mod_ssl+ mod_proxy.

Essentially what was built was an SSL capable proxy server. The client machines out in the wild request the document via SSL. The proxy inside the web server turns around and over the internal network requests the actual document from Weblogic. Using Apache internal magic, any references to the Weblogic server are translated to the outside proxy box, and the resulting document is passed back over SSL. The transaction is completely secure, and gets the added benefit of having Apache in front of Weblogic, Apache being a much more robust server better able to handle high volume traffic. We saw our overall performance on our Weblogic servers improve greatly as Apache handled the loads much more gracefully. We went live on time and have had great results from the ssl_proxy.

Now let's step through how to build this puppy. You'll need three packages:

OpenSSL <http://www.openssl.org>

ModSSL <http://www.modssl.org> (get the version corresponding to the
version of Apache you download)

Apache <http://httpd.apache.org>

First unpack and build OpenSSL. This is simple and just typing ./config should detect your system and configure it. Run make install and this part is done.

Next you'll need to unpack Apache. Do this, then proceed to unpacking mod_ssl; this is a patch that will be applied to the Apache source to add SSL capabilities.

In the mod_ssl directory, you need to run the configure script:

```
bmcadams@mycroft:~/ssl_proxy/src/mod_ssl-2.8.5-1.3.22$ ./configure \
```

> "--with-apache=../apache_1.3.22" \

> "--with-ssl=../openssl-0.9.6b"

(You will need to substitute ../apache_1.3.22 with the location of your unpacked apache source tree, and ../openssl-0.9.6b with the directory you built the openssl source in) This will patch Apache; there is no need to run make. Now change directories to the apache source directory and run the following command:entral Command, which directs the U.S

bmcadams@mycroft:~/ssl_proxy/src/apache_1.3.22$ ./configure \

"--with-layout=Apache" \

"--enable-module=ssl" \

"--enable-module=proxy" \

"--enable-module=rewrite" \

"--prefix=/usr/local/ssl_proxy/apache"

Substitute /usr/local/ssl_proxy/apache for whatever directory you wish to put Apache into. Run make and make install ; assuming no errors were reported your proxy server should be all built and ready to go; we just need to configure it. Change directories to /usr/local/ssl_proxy/apache (or whatever directory you set as your Apache prefix), then open conf/httpd.conf.

I am not going to go into the details of generating an SSL certificate and fine tuning SSL; I feel that is a topic vast enough to encompass it's own discussion and will, for the time being, be left as an exercise for the reader (have you gotten tired of me saying that yet?).

We will however, delve into the proxy config (this is assuming your SSL server is all setup; the default config installed by Apache-mod_ssl should work for the time being.)

In your config, go inside the SSL virtualhost and add the following:

ProxyRequests Off

ProxyPass / http://10.0.0.167:7001/

ProxyPassReverse / http://10.0.0.167:7001/

ProxyRequests Off is an extremely important directive; I cannot stress this enough. If you do not make sure this is turned off, you will basically make your webserver an open proxy; anyone can configure their web browser to use you as their proxy server and browsing requests to remote sites will appear to come from your machine. This is a bad thing and potential security breach.

What ProxyPass and ProxyPassReverse do are map the website as being reverse proxied. Essentially, any request to the webserver for / and any URIs below it (such as /index.html) is mapped by ProxyPass to point at the webserver at http://10.0.0.167:7001/. What does this mean exactly?

It means that, although the outside user doesn't see it, a request for http://outsideaddress.com/ooga/booga/foo/bar/baz is passed to http://10.0.0.167:7001/ooga/booga/foo/bar/baz for handling. The response the user sees is served up by 10.0.0.167:7001 ; however, on the way back through ProxyPassReverse does some internal magic to make sure things like Location: redirects in headers are rewritten to point the user back at the Proxy server. It would be awful embarrassing to have a user call up and say your site is broken because they can't reach http://10.0.0.167:7001/bah.html.

At this point you are done. You can look at the Apache docs for more info on the ProxyPass and ProxyPassReverse directives. You could even separate parts of your site onto separate back-end sites (/images/ could go to a graphics only server) or setup caching for your static documents.

One last thing to note:

If you are doing port mapping on inside firewalls (public requests port 443 on your webserver, but your webserver [the proxy] listens on 8443 and your firewall maps it), you need to set the following in your Apache config:

UseCanonicalName Off

If you don't do this, what you'll see is anytime ProxyPassReverse has to fix a Location: redirect or anything else, it fixes it to point at https://mybox.com:8443/. Since your firewall doesn't pass 8443 through, your users won't be able to reach the document. Turning off Canonical names will tell Apache to respond back (and let ProxyPassReverse rewrite) as whatever server the user requested (namely, https://mybox.com:443/).

NYLXS - Writing a backup program in GTK and C

by Ruben I. Safir

A client recently asked me to implement a tape backup program which is easier to user you by office staff. As a result of this request, I chose to implement a solution with the existing free software tool available, and use this as an excuse for finally learning a little GTK and brushing off my C programming skills.

The reason for writing this program in C and GTK is two fold. GTK is the most common free X based widget set available today. Over the past few years, I've written a lot of web based applications, but I wanted this program to run only from the server, and not depend on TCP/IP for it's implementation. In addition to this, backups can include thousands of files, and trying to get them process and display that many rows of information can be very difficult.

The current backup is being done with tar CRON to backup the files every day at 1AM. The program mails a list of backuped files to the the administrator and makes a list on the hard drive of all the files.

The old system will continue, but the new one will allow for simple click and go restoration from the tape.

The first thing I did when tackling this project is to construct a decent make file. GTK has standard configuration scripts which usually make compiling easy, but I'm opposed to using them for a number of reasons. First, the configuration scripts often are wrong. And secondly, there good reasons to create a decent make file in terms of interaction with the editor and compiling speeds through dependency control.

My make file for this project looks like this:

```
CC=gcc

LDLIBS= -L/usr/local/lib -lglib -L/usr/lib -L/usr/X11R6/lib -lgtk -lgdk -rdynamic -lgmodule -lglib
-ldl -lXi -lXext -lX11 -lm

CFLAGS= -Wall -g -O3 -I/usr/include/glib-1.2 -I/usr/lib/glib/include -I/usr/X11R6/include
-I/usr/include -I/usr/include/gtk-1.2

INCLUDE= .

rmbackup: rmbackup.o backprog.o tarlib.o

rmback.o: rmbackup.c

backprog: backprog.o

backprog: backprog.o

backprog.o:backprog.c

tarlib: tarlib.o
```

```
tarlib.o:tarlib.c
tarlib.c:
$(CC) -I$(INCLUDE) $(LDLIBS) $(CFLAGS) -c tree.c
backprog.c:
$(CC) -I$(INCLUDE) $(LDLIBS) $(CFLAGS) -c backprog.c
rmbackup.c:
$(CC) -I$(INCLUDE) $(LDLIBS) $(CFLAGS) -c backprog.c
```

From here you can see I broke my program into three main sections:

- tarlib – This is mostly a tar library section of code dealing with the ins and outs of the actual backups.
- backprog – This mostly deals with display of the program in X
- rmback – This runs main and launches the application.

Proper header have been created for this program, and I found that I didn't make a clear a distinction as I wanted to between display and function. But the program is generally designed cleanly.

When using GTK, I found that some of the documentation and the actual resulting outcomes was not as advertized. Most annoying was problems with the GtkList * widget. In theory, the widget creates a clickable list of files, similarly to mail client. Glib has a linked list structure called a Glist which has a series of functions to handle a linked list. You should be able to append a GList directly into a GtkList, but this fails. As a result, this functionality has to be done by hand.

In Gtk, the use of C structs are used extensively to give the feel of object orientation to a procedural language. Everything is constructed as boxes in boxes in boxes. First you make a GtkWidget *. Then you tern the widget into a specific kind of widget through a Gtk library function for that widget. Then you add widgets into your widget as needed. And finally you show all the widgets. For example, this program might create a top level window, and then a packing box in the window, then 3 buttons in the packing box. It is then up to you to create callback functions for the mouse and keyboard signals of each event.

Signal callbacks run your program. The are attached to specific events, and then call programs. In writing this program, I learned a number of lessons. First, I learned that if I create a data object in a function, even if I pass it as a pointer to the signal callback, the data structure does not follow into the stack of the main Gtk loop. I had to globalize through static some of my data type declarations. Secondly, I learned you can't fork once a X widget is running in your program, because it throws your program into a threading condition and X complains, even though the code compiles.

Let's look at the basic construction of the program.

```
#include <stdio.h>
#include "tarlib.h"
#include "backprog.h"

int main(int argc, char **argv){
        struct s1 window1;
        gtk_init(&argc, &argv);
```

```
        window1 = screen1();
        gtk_main();
        exit(0);
}
```

backup.h

```
#ifndef BACKPROG_H
#define BACKPROG_H


struct s1{
    GtkWidget *box;
    GtkWidget *lab;
    GtkWidget *win;
 };

int makescreenone();
struct s1 maketoggles();
void startbackup(GtkWidget *widget, GtkWidget *data);
void listfiles(GtkWidget *widget, gchar *data);
gint closeapp(GtkWidget *widget, gchar *data);
struct s1 screen1();

#endif
```

backup.c

```
#include <stdio.h>
#include <gtk/gtk.h>
#include "backprog.h"
#include "tarlib.h"


struct s1 maketoggles(){
        static struct s1 handle;
        GtkWidget *button;
        GtkWidget *vbox;
        GtkWidget *label;

        vbox = gtk_vbox_new(TRUE, 0);

        label = gtk_label_new("Choose an Option");

        gtk_widget_show(label);

        gtk_box_pack_start(GTK_BOX(vbox), label, FALSE, FALSE,3);

        button = gtk_toggle_button_new_with_label("Create Backup");
        gtk_box_pack_start(GTK_BOX(vbox), button, TRUE, FALSE, 0);
        gtk_widget_show(button);
        gtk_signal_connect(GTK_OBJECT(button), "clicked",
                GTK_SIGNAL_FUNC(startbackup),
```

```
                    label );

        button = gtk_toggle_button_new_with_label("List Files to Restore");
        gtk_box_pack_start(GTK_BOX(vbox), button, TRUE, FALSE, 0);
        gtk_widget_show(button);
        gtk_signal_connect(GTK_OBJECT(button), "clicked",
                    GTK_SIGNAL_FUNC(listfiles),
                    "looking up files to resotre");

        button = gtk_toggle_button_new_with_label("Exit");
        gtk_box_pack_start(GTK_BOX(vbox), button, TRUE, FALSE, 0);
        gtk_widget_show(button);
        gtk_signal_connect(GTK_OBJECT(button), "clicked",
                    GTK_SIGNAL_FUNC(closeapp), NULL);

        handle.box = vbox;
        handle.lab = label;
        handle.win = NULL;

        return handle;
}

void startbackup( GtkWidget *widget, GtkWidget *lab){
        gint ret;
/*      gtk_label_set_text(GTK_LABEL(lab), "CREATING BACKUP"); */
        ret = createbackup();
        g_print("%d",ret);
        g_print("\n");
/*      gtk_label_set_text(GTK_LABEL(lab), "Choose an Option");*/

}

void listfiles( GtkWidget *widget, gchar *data){
        createlist();
        g_print("%s\n", data);

}

gint closeapp( GtkWidget *widget, gchar *data){
            gtk_main_quit();
            return(FALSE);

}

struct s1 screen1(){
        struct s1 handle;
        GtkWidget * win = gtk_window_new(GTK_WINDOW_TOPLEVEL);
        gtk_window_set_title(GTK_WINDOW(win), "Rosenzweig and Maffia Backup Program");
        gtk_container_border_width(GTK_CONTAINER(win), 25);
        handle = maketoggles();

        gtk_container_add(GTK_CONTAINER(win), handle.box);
```

```
	gtk_widget_show(handle.box);
	gtk_widget_show(win);
	handle.win = win;
	return(handle);
}
#include <stdio.h>
#include <stdlib.h>
#include <gtk/gtk.h>
#include "tarlib.h"

gint createbackup(){
	gint ret;
//	gint pid;
	gchar buffer[512] = {'\0'};
	FILE *tarcommand;
	GtkWidget *window_proc;
	GtkWidget *text;
	GtkWidget *box;
	GtkWidget *vscroll;
/*	const char *command = "/bin/tar -cvpPf /tmp/apache.tar /home/ruben/ 2>/dev/null"; */
	const char *command = "nice -10 /bin/tar -cvpP --exclude-from /root/noback  --file
/dev/st0 /  2>/dev/null";
	const char *command2 = "nice -10 /bin/tar -tf /dev/st0 1> /home/jam/backup_files.txt";
	GdkColormap *cmap;
	GdkColor color;

/*	pid = fork();
	if(pid != 0){
	   perror("We are in the child");   */
	   window_proc = gtk_window_new(GTK_WINDOW_TOPLEVEL);
	   gtk_widget_set_usize(window_proc, 600, 400);
	   gtk_signal_connect(GTK_OBJECT(window_proc), "delete_event",
			GTK_SIGNAL_FUNC(delete_win),NULL);

	   text = gtk_text_new(NULL,NULL);
	   gtk_text_set_editable(GTK_TEXT(text), FALSE);
	   box = gtk_hbox_new(FALSE,0);
	   vscroll=gtk_vscrollbar_new(GTK_TEXT(text)->vadj);
	   gtk_container_add(GTK_CONTAINER(window_proc), box);
	   gtk_box_pack_start(GTK_BOX(box), text, TRUE, TRUE,0);
	   gtk_box_pack_start(GTK_BOX(box), vscroll, FALSE, FALSE,0);
	   gtk_widget_show(vscroll);
	   gtk_widget_show(text);
	   gtk_widget_show(box);
	   gtk_widget_show(window_proc);
	   cmap = gdk_colormap_get_system();
	   color.red = 0xffff;
	   color.green = 0;
	   color.blue = 0;
	   if (!gdk_color_alloc(cmap, &color)) {
	       g_error("couldn't allocate color");
```

```
        }




        system("mt -f /dev/st0 rewind");
        system("mt -f /dev/st0 retension");
        tarcommand  = popen(command, "r");
        while(fgets(buffer, 512, tarcommand)){
//                gtk_text_freeze(GTK_TEXT(text));
                gtk_text_insert(GTK_TEXT(text),NULL,NULL,NULL, buffer, strlen(buffer));
//                gtk_text_thaw(GTK_TEXT(text));
                }
                gtk_text_insert(GTK_TEXT(text),NULL,&color,NULL,
                                "DONE", -1);


        ret = pclose(tarcommand);
        ret = system(command2);


         if( ret  == 127 ){
                g_print("Can not open shell\n");
                exit(0);

          }
         return ret;

/*
        }else{ */
         /*ret = fgetc(tar);*/
/*          ret = 0;
         return(ret);
        } */
}

gint delete_win(GtkWidget *widget, gpointer *data){
        return FALSE;
}

GtkWidget *addrow(char *entry){
        GtkWidget *item;
        item = gtk_list_item_new_with_label(entry);
        gtk_widget_show(item);
        return(item);
}

int createlist(void){
        GtkWidget *item;
        GtkWidget *listbox;
        GtkWidget *sw;
        GtkWidget * window_select;
```

```
GtkWidget * vbox;
GtkWidget * button;
GtkAdjustment * vert;
GtkAdjustment * horz;
FILE *fp;
char file[512] = {'\0'};
if(NULL == (fp = fopen("/home/jam/backup_files.txt", "r"))){
        g_error("Can't Open Files");
        exit(1);
}
vert = GTK_ADJUSTMENT( gtk_adjustment_new(0,0,0,0,0,0));
horz = GTK_ADJUSTMENT( gtk_adjustment_new(0,0,0,0,0,0));

window_select = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_container_border_width(GTK_CONTAINER(window_select), 10);
gtk_window_set_title(GTK_WINDOW(window_select), "Files to Restore");
gtk_window_set_default_size(GTK_WINDOW(window_select), 600,400);

vbox = gtk_vbox_new(FALSE,5);

sw = gtk_scrolled_window_new(horz, vert);

listbox = gtk_list_new();
gtk_list_set_selection_mode(GTK_LIST(listbox), GTK_SELECTION_EXTENDED);

gtk_container_add(GTK_CONTAINER(window_select), vbox);

gtk_box_pack_start(GTK_BOX(vbox), sw, TRUE, TRUE,5);

button = gtk_button_new_with_label("Restore Selected Files");
gtk_box_pack_end(GTK_BOX(vbox), button, TRUE, TRUE,5);

gtk_scrolled_window_add_with_viewport(GTK_SCROLLED_WINDOW(sw),listbox);

while(fgets(file, 512, fp)){
         /* g_print("%s",file);
        g_print("%d\n", strlen(file));
        g_print("LAST CHAR %c \n", file[strlen(file) -1]);*/
        file[strlen(file) -1] = '\0';
        item = gtk_list_item_new_with_label(file);
        gtk_container_add(GTK_CONTAINER(listbox), item);
        gtk_widget_show(item);
}
fclose(fp);
gtk_signal_connect(GTK_OBJECT(listbox), "select_child", GTK_SIGNAL_FUNC(file_selected),
NULL);
gtk_signal_connect(GTK_OBJECT(listbox), "unselect_child",
GTK_SIGNAL_FUNC(file_unselected), NULL);
gtk_signal_connect(GTK_OBJECT(button), "clicked", GTK_SIGNAL_FUNC(restore), NULL);

gtk_widget_show(vbox);
gtk_widget_show(listbox);
```

```
	gtk_widget_show(button);
	gtk_widget_show(sw);
	gtk_widget_show(window_select);

	return(1);
}

//Signal Handles for list selection
void file_selected(GtkList * listbox, GtkWidget *widget, gpointer data){
	links = g_list_append(links, widget);
}


void file_unselected(GtkList * listbox, GtkWidget *widget, gpointer data){
	links = g_list_remove(links, widget);
}

//Signal Handle for button

int restore(GtkButton * button, gpointer data){
	GtkWidget *window_proc;
	GtkWidget *text;
	GtkWidget *box;
	GtkWidget *vscroll;
	GdkColormap *cmap;
	GdkColor color;
	   window_proc = gtk_window_new(GTK_WINDOW_TOPLEVEL);
	   gtk_widget_set_usize(window_proc, 600, 400);
	   gtk_signal_connect(GTK_OBJECT(window_proc), "delete_event",
			GTK_SIGNAL_FUNC(delete_win),NULL);

	   text = gtk_text_new(NULL,NULL);
	   gtk_text_set_editable(GTK_TEXT(text), FALSE);
	   box = gtk_hbox_new(FALSE,0);
	   vscroll=gtk_vscrollbar_new(GTK_TEXT(text)->vadj);
	   gtk_container_add(GTK_CONTAINER(window_proc), box);
	   gtk_box_pack_start(GTK_BOX(box), text, TRUE, TRUE,0);
	   gtk_box_pack_start(GTK_BOX(box), vscroll, FALSE, FALSE,0);
	   gtk_widget_show(vscroll);
	   gtk_widget_show(text);
	   gtk_widget_show(box);
	   gtk_widget_show(window_proc);
	   cmap = gdk_colormap_get_system();
	   color.red = 0xffff;
	   color.green = 0;
	   color.blue = 0;
	   if (!gdk_color_alloc(cmap, &color)) {
	       g_error("couldn't allocate color");
	   }


	if(links == NULL)
```

```
            return(0);
        g_list_foreach(links, untar_file, text);
        gtk_text_insert(GTK_TEXT(text),NULL,&color,NULL,
                        "DONE", -1);
        return(1);
}

void untar_file( gpointer data, gpointer text){
        gchar * filename;
        gchar buffer[512] = {'\0'};
        FILE *untar;
        const char *command = "nice -10 /bin/tar -xvpPf /dev/st0 ";
        char command2[512] = {'\0'};
        gtk_label_get(GTK_LABEL(GTK_BIN(data)->child), &filename);
//      g_print("FILES %s\n", filename);
        strcat( command2, command );
        strcat( command2, filename );

        strcat( command2, " 2>/dev/null" );
//      g_print("COMMAND %s\n", command2);
        untar  = popen(command2, "r");
//      g_print("RESOTRING FILES");
        system("mt -f /dev/st0 rewind");
        system("mt -f /dev/st0 retension");
        while(fgets(buffer, 512, untar)){
          gtk_text_insert(GTK_TEXT(text),NULL,NULL,NULL, buffer, strlen(buffer));
            //g_print("RESTORED %s\n", buffer);
        }
        pclose(untar);


}
```

tarlib.h

```
#ifndef TABLIB_H
#define TABLIB_H
#include<gtk/gtk.h>
#include<unistd.h>
#include<string.h>

gint createbackup(void);
gint delete_win(GtkWidget * ,gpointer *);
int createlist(void);
GtkWidget *addrow(char *);
void file_selected(GtkList *, GtkWidget *, gpointer);
void file_unselected(GtkList *, GtkWidget *, gpointer);
int restore(GtkButton *, gpointer);
void untar_file( gpointer data, gpointer dat);
GList *links;
#endif
```

tarlib.c

```
#include <stdio.h>
#include <stdlib.h>
#include <gtk/gtk.h>
#include "tarlib.h"

gint createbackup(){
	gint ret;
//	gint pid;
	gchar buffer[512] = {'\0'};
	FILE *tarcommand;
	GtkWidget *window_proc;
	GtkWidget *text;
	GtkWidget *box;
	GtkWidget *vscroll;
/*	const char *command = "/bin/tar -cvpPf /tmp/apache.tar /home/ruben/ 2>/dev/null"; */
	const char *command = "nice -10 /bin/tar -cvpP --exclude-from /root/noback  --file
/dev/st0 /  2>/dev/null";
	const char *command2 = "nice -10 /bin/tar -tf /dev/st0 1> /home/jam/backup_files.txt";
	GdkColormap *cmap;
	GdkColor color;

/*	pid = fork();
	if(pid != 0){
	   perror("We are in the child");   */
	   window_proc = gtk_window_new(GTK_WINDOW_TOPLEVEL);
	   gtk_widget_set_usize(window_proc, 600, 400);
	   gtk_signal_connect(GTK_OBJECT(window_proc), "delete_event",
			GTK_SIGNAL_FUNC(delete_win),NULL);

	   text = gtk_text_new(NULL,NULL);
	   gtk_text_set_editable(GTK_TEXT(text), FALSE);
	   box = gtk_hbox_new(FALSE,0);
	   vscroll=gtk_vscrollbar_new(GTK_TEXT(text)->vadj);
	   gtk_container_add(GTK_CONTAINER(window_proc), box);
	   gtk_box_pack_start(GTK_BOX(box), text, TRUE, TRUE,0);
	   gtk_box_pack_start(GTK_BOX(box), vscroll, FALSE, FALSE,0);
	   gtk_widget_show(vscroll);
	   gtk_widget_show(text);
	   gtk_widget_show(box);
	   gtk_widget_show(window_proc);
	   cmap = gdk_colormap_get_system();
	   color.red = 0xffff;
	   color.green = 0;
	   color.blue = 0;
	   if (!gdk_color_alloc(cmap, &color)) {
	       g_error("couldn't allocate color");
	  }
```

```
        system("mt -f /dev/st0 rewind");
        system("mt -f /dev/st0 retension");
        tarcommand  = popen(command, "r");
        while(fgets(buffer, 512, tarcommand)){
//              gtk_text_freeze(GTK_TEXT(text));
                gtk_text_insert(GTK_TEXT(text),NULL,NULL,NULL, buffer, strlen(buffer));
//              gtk_text_thaw(GTK_TEXT(text));
                }
                gtk_text_insert(GTK_TEXT(text),NULL,&color,NULL,
                        "DONE", -1);


        ret = pclose(tarcommand);
        ret = system(command2);


         if( ret  == 127 ){
                g_print("Can not open shell\n");
                exit(0);

          }
         return ret;

/*
        }else{ */
         /*ret = fgetc(tar);*/
/*          ret = 0;
         return(ret);
        } */
}

gint delete_win(GtkWidget *widget, gpointer *data){
        return FALSE;
}

GtkWidget *addrow(char *entry){
        GtkWidget *item;
        item = gtk_list_item_new_with_label(entry);
        gtk_widget_show(item);
        return(item);
}

int createlist(void){
        GtkWidget *item;
        GtkWidget *listbox;
        GtkWidget *sw;
        GtkWidget * window_select;
        GtkWidget * vbox;
        GtkWidget * button;
        GtkAdjustment * vert;
        GtkAdjustment * horz;
```

```
FILE *fp;
char file[512] = {'\0'};
if(NULL == (fp = fopen("/home/jam/backup_files.txt", "r"))){
        g_error("Can't Open Files");
        exit(1);
}
vert = GTK_ADJUSTMENT( gtk_adjustment_new(0,0,0,0,0,0));
horz = GTK_ADJUSTMENT( gtk_adjustment_new(0,0,0,0,0,0));

window_select = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_container_border_width(GTK_CONTAINER(window_select), 10);
gtk_window_set_title(GTK_WINDOW(window_select), "Files to Restore");
gtk_window_set_default_size(GTK_WINDOW(window_select), 600,400);

vbox = gtk_vbox_new(FALSE,5);

sw = gtk_scrolled_window_new(horz, vert);

listbox = gtk_list_new();
gtk_list_set_selection_mode(GTK_LIST(listbox), GTK_SELECTION_EXTENDED);

gtk_container_add(GTK_CONTAINER(window_select), vbox);

gtk_box_pack_start(GTK_BOX(vbox), sw, TRUE, TRUE,5);

button = gtk_button_new_with_label("Restore Selected Files");
gtk_box_pack_end(GTK_BOX(vbox), button, TRUE, TRUE,5);

gtk_scrolled_window_add_with_viewport(GTK_SCROLLED_WINDOW(sw),listbox);

while(fgets(file, 512, fp)){
         /* g_print("%s",file);
        g_print("%d\n", strlen(file));
        g_print("LAST CHAR %c \n", file[strlen(file) -1]);*/
        file[strlen(file) -1] = '\0';
        item = gtk_list_item_new_with_label(file);
        gtk_container_add(GTK_CONTAINER(listbox), item);
        gtk_widget_show(item);
}
fclose(fp);
gtk_signal_connect(GTK_OBJECT(listbox), "select_child", GTK_SIGNAL_FUNC(file_selected),
NULL);
gtk_signal_connect(GTK_OBJECT(listbox), "unselect_child",
GTK_SIGNAL_FUNC(file_unselected), NULL);
gtk_signal_connect(GTK_OBJECT(button), "clicked", GTK_SIGNAL_FUNC(restore), NULL);

gtk_widget_show(vbox);
gtk_widget_show(listbox);
gtk_widget_show(button);
gtk_widget_show(sw);
gtk_widget_show(window_select);
```

```
	return(1);
}

//Signal Handles for list selection
void file_selected(GtkList * listbox, GtkWidget *widget, gpointer data){
	links = g_list_append(links, widget);
}


void file_unselected(GtkList * listbox, GtkWidget *widget, gpointer data){
	links = g_list_remove(links, widget);
}

//Signal Handle for button

int restore(GtkButton * button, gpointer data){
	GtkWidget *window_proc;
	GtkWidget *text;
	GtkWidget *box;
	GtkWidget *vscroll;
	GdkColormap *cmap;
	GdkColor color;
	   window_proc = gtk_window_new(GTK_WINDOW_TOPLEVEL);
	   gtk_widget_set_usize(window_proc, 600, 400);
	   gtk_signal_connect(GTK_OBJECT(window_proc), "delete_event",
			GTK_SIGNAL_FUNC(delete_win),NULL);

	   text = gtk_text_new(NULL,NULL);
	   gtk_text_set_editable(GTK_TEXT(text), FALSE);
	   box = gtk_hbox_new(FALSE,0);
	   vscroll=gtk_vscrollbar_new(GTK_TEXT(text)->vadj);
	   gtk_container_add(GTK_CONTAINER(window_proc), box);
	   gtk_box_pack_start(GTK_BOX(box), text, TRUE, TRUE,0);
	   gtk_box_pack_start(GTK_BOX(box), vscroll, FALSE, FALSE,0);
	   gtk_widget_show(vscroll);
	   gtk_widget_show(text);
	   gtk_widget_show(box);
	   gtk_widget_show(window_proc);
	   cmap = gdk_colormap_get_system();
	   color.red = 0xffff;
	   color.green = 0;
	   color.blue = 0;
	   if (!gdk_color_alloc(cmap, &color)) {
	        g_error("couldn't allocate color");
	   }


	if(links == NULL)
		return(0);
	g_list_foreach(links, untar_file, text);
	gtk_text_insert(GTK_TEXT(text),NULL,&color,NULL,
				"DONE", -1);
```

```
	return(1);
}

void untar_file( gpointer data, gpointer text){
	gchar * filename;
	gchar buffer[512] = {'\0'};
	FILE *untar;
	const char *command = "nice -10 /bin/tar -xvpPf /dev/st0 ";
	char command2[512] = {'\0'};
	gtk_label_get(GTK_LABEL(GTK_BIN(data)->child), &filename);
//	 g_print("FILES %s\n", filename);
	strcat( command2, command );
	strcat( command2, filename );

	strcat( command2, " 2>/dev/null" );
//	 g_print("COMMAND %s\n", command2);
	untar  = popen(command2, "r");
//	 g_print("RESOTRING FILES");
	system("mt -f /dev/st0 rewind");
	system("mt -f /dev/st0 retension");
	while(fgets(buffer, 512, untar)){
	  gtk_text_insert(GTK_TEXT(text),NULL,NULL,NULL, buffer, strlen(buffer));
	      //g_print("RESTORED %s\n", buffer);
	}
	pclose(untar);



}
```

Some of the interesting features of this program is the use of a global struct defined in backup, which is being used to pass screens from function to function. This was needed to keep the struct in the memory stack.

The program uses popen to talk to tar, which performs the actual backups, and creates a list of backuped files. popen forks and opens a command to a shell for reading or writing (not both). It gets a file handle to process from the command line of the shell.

The clickable list is generated from a text file which is read in. This file is produced by the tar backup process. In addition, some minor string handling needed to be done. Ideally, a fork of the tar could be done so that control of the panel is not lost while the tape is accessed, and this is an imporvement to be built in on the next version. A straight for is not possible because it causes a threading condition for X, as mentioned before. Finally, the resulting backups generate a list of over 170,000 files. This breaks the scrollable window as currently written. Another method will be needed to deal with this problem. On solution might be to read in 1,000 entries at a time and display them only with a new forward and back button added. This can be done either by reading the entry list file into an array of gchar pointers, or using fseek and tell to do random reads and writes to the file. But then we will need to standardize the size of the line inputs. A search window also needs to be added.

A Brief Introduction to the GNU awk Utility

By ????

Gawk is the GNU version of awk, an interpreted text manipulation language. The name is derived from initials of the last names of the original authors, Alfred Aho, Peter Weinberger, and Brian Kernighan. The syntax of awk is similar in many ways to C syntax, which is not surprising since Kernighan is also one of the original authors of the C programming language. Some claim that awk inspired, in part, the development of Perl. People familiar with C, Perl, and/or shell scripting should easily find a way to sink their teeth into awk or its GNU implementation gawk.

This article will be dealing with gawk, although the concepts and examples should work with other implementations of awk. It is assumed for this article that you are using a Free Software Operating System (this means GNU/Linux and several BSD OSes, among others) or some UNIX-like OS. gawk can be described more fully as a pattern-matching program for processing text files containing data in fields separated by a delineator. An example of such a file is a text file that lists people's names, phone numbers, and email addresses separated by tabs. Another example is a file some_file resulting from the shell command ls -l > some_file. You can use gawk to select lines from the file by matching values in the fields against patterns you define. The patterns used for matching can be regular expressions, relational expressions, or simple pattern-matching expressions. This ability is similar to the grep program. You can also use gawk to print selected fields from matched lines. The combination of these two capabilities allows you to use delimited text files as a no-frills database for situations in which using a real database is not warranted.

Now let's get into some gawk examples. We will be reading and analyzing data from the file that lists users on your system, which is usually called passwd and located in the /etc directory. Here is a simple example that lists the first field of every line in your /etc/passwd file. Type the the following at a command prompt:

```
gawk -F ":" '{print $1}' /etc/passwd
```

Try it. This will show you all the users on the system. The result should look something like this:

```
root

daemon

bin

sys

...
```

Now let's go through the example step by step. The gawk portion invokes the interpreter, and the -F ":" tells gawk that a colon (":") is the character used in between the fields in the file, called a field separator or delineator. The field separator could also be specified as -F: or as -F :. These versions without quotes are equivalent to the version used in the previous example. Also note that the default field separator is any "white space", i.e. a tab or space character, so if no separator is specified, gawk will split lines using spaces and/or tabs if any are present. gawk uses ' ' characters, or single quotes, around curly braces, {}, to enclose the script it interprets, as in: '{script...}'. In the example, the script does not do any pattern-matching and therefore all lines are returned. The print $1 tells gawk to print the first field from each returned line. Finally, /etc/passwd is the file which gawk will read as data. Note that if one wanted to print the second and third fields as well, the script would look like this:

```
gawk -F: '{print $1; print $2; print $3}' /etc/passw
```

Notice the semicolons (;) between print commands. These are used to separate commands in gawk. If you tried the last command, you probably noticed that each field is printed on a new line. If you prefer all the fields on one line separated by a space, use this instead:

{print $1, $2, $3}

Or if you prefer them separated by another character, maybe a comma, try this script instead:

{print $1 "," $2 "," $3}

The double quotes tell gawk that you want the characters inside to be literally printed to the output. The characters within quotes are thus called string literals or just literals. You can see how longer scripts can become cumbersome on the command line. This is why gawk allows you to call scripts from files. Open a new file in your favorite editor and enter the following text:

BEGIN { FS=":" }

{print $1, $2, $3}

The BEGIN { FS=":" } line tells gawk that the field separator is a colon. This is equivalent to using -F: on the command line. The rest of the script is contained in a set of curly braces with commands separated by semicolons. Now save the file and execute it using the following command, replacing "your_script" with the name of your script:

gawk -f your_script /etc/passwd

Let's build a script that will use some of gawk's pattern-matching capabilities. First, a brief explanation of simple pattern-matching expressions. In pattern matching, ~ (the tilde character) means "matches", as in:

$1 ~ "root"{print}

The above line will print any lines in which the first field or part of the first field contains the string of characters in between the quotation marks, in this case 'root'. Note that if the field was 'treeroot' or 'roots', the line would be printed as well. To specify that the field should match exactly, use the special characters ^ and $ to specify the beginning and end of the field, like this:

$1 ~ "^root$"{print}

There is also a way to specify that you want lines that don't match a certain string of characters: use !~ instead of ~ . Where quotes are used in the above examples, slashes (/) can be used to specify a regular expression pattern match, but regular expressions are not covered in any detail here.

So, let's build a script that will print the user names of all users on your system that use a particular shell on your system. In the /etc/passwd file there is a field that specifies the user's default shell. In most cases it is the seventh field, but you make have to look at the file to find out which field it is on your system. For most users it will look something like /bin/sh or /bin/csh . Let's see how many users have bash (Bourne Again Shell) as the default shell. Try the following script:

Script to see who uses bash

BEGIN{ FS=":" }

$7 ~ "/bash$"{print $1}

The first line is a comment and ignored by gawk. Comments start with the pound sign (#), and everything on the rest of the line is disregarded. Another thing to note is that we used the dollar sign ($) in the search pattern again so that we are certain that the pattern matches against the end of the field. Otherwise, if a system kept shells in a directory that began with 'bash', it would be printed out, though that is not our intention. This way, it does not matter in which directory shell programs are located; we are only concerned with what the shell is called.

In most cases, if bash is available, the shell name 'sh' is just another name for bash. Those are not counted by the preceding script, so let's rewrite the script to account for both names. And while we are at it, let's provide a count of the users that use bash, and counts of how many specify each of the names. For this task we will need to use flow control commands (if, else, for, etc.), which are very similar to Perl and C. We will also need to group statements together using curly braces ({ }), just as in Perl and C. Here is the script, with brief comments included:

```
#(1) use BEGIN to set the field seperator and set
# the total number of bash users to zero
BEGIN{
FS=":"; #(2) semicolon seperates statements
TOTAL = 0 # return or linebreak seperates statements also
NUM_BASH = 0 # either works, it is up to you
}
#(3) start the main processing loop that is called for each line of
input
#(4) first check if the user's shell is some form of bash
($7 ~ "/bash$") || ($7 ~ "/sh$"){
#(5) if it is bash, add 1 to the total
TOTAL = TOTAL + 1
#(6) the above line could be written as TOTAL++ (like C and Perl)
#(7) see if the default shell is named bash
if ($7 ~ "/bash$"){
#(8) add 1 to the number who call it bash
NUM_BASH++
}
#(9) see if the shell is named sh
else if ($7 ~ "/sh$"){
#(10) add 1 to the number who know it by sh
NUM_SH++
}
#(11) print the user name since they use bash (by some name)
print $1
}
#(12) use END to print final results at the end of processing
END{
#(13) print a few *'s to end the user list
print "********"
#(14) print the total and explain what it means
print TOTAL " users have bash as the default shell."
print NUM_SH " call it \"sh\". " NUM_BASH " call it \"bash\"."
```

}

Run it and note the output. Now for the explanation. The first part is similar to the previous example, but then we see the first variable declaration. Near comment (2), TOTAL is set to zero. In gawk, variables are not "typed" as they are in many other languages, so you simply write the name of the variable, an equal sign (=), and the value it will hold. Note that gawk does not require variable declaraction; you can just start using a variable and gawk will assign a default value to the variable.

Next the main loop begins, looking at each line in turn. The line after comment (4) checks whether the seventh field matches 'bash' or 'sh'. See how the first half of the line looks like the pattern-matching in the previous example. Notice that the two checks are each grouped in parentheses seperated by two pipes, || . Just like in C and Perl, || means 'or' in gawk. So if a line matches either of the two expressions, the commands between the curly braces ({ }) will be executed.

The first statement adds 1 to the total. Next we check whether the line matched 'bash' or 'sh' and add 1 to the appropriate tally. Notice that the variable NUM_SH is never declared, it is first used when it is incremented (has 1 added to it). gawk assigns it to zero automatically; if it were being used as a string it would be initialized to an empty string to begin with. So as long as you are satisfied with the auto-initialized values, you can just start using variables without declaration.

After all the right counters are incremented, the user name (field 1) is printed for each qualifying line. Finally, the END section is used to report the totals. The END section is useful for this job because it is executed once, after all lines in the data file have been inspected and processed apropriately.

Note that gawk takes input from Standard In (STDIN) if no data file is specified. So it can be manipulated on the command line using input/output redirection (e.g. >>), making it a powerful tool for the command line. gawk is especially useful when combined with other

tools, in particular sed.

Hopefully, you now have enough information to make use of gawk or even add it to your *nix utility belt. Happy gawk-ing!

Open Source Software in the Financial Industry

by Brendan W. McAdams <rit@jacked-in.org>

There are actually two 'cases' I want to focus on: one involves some basic bits of mod perl which can be used to create clean, dynamic database driven content; the other involves building a software based 'ssl accelerator' to add SSL to a non-SSL capable web server and save yourself and your firm fists full of dollars. All of the examples I will discuss can and have been created with free, open source software (with the exception of the database work, which in my case was done in Sybase. However as I don't really go into the SQL and don't touch upon the stored procedures, this shouldn't be an issue; MySQL or Postgres

should suffice nicely.)

A Simple Mod_Perl System

The first section we will discuss is building dynamic content using mod_perl. Acquiring and building a mod_perl capable apache server, as well as a back-end database is left as an exercise to the reader; someone in NYLXS (or myself) should be able to help you with this task if you find you need help.We will assume a good working knowledge of Perl, and some knowledge of Apache for this discussion; mod_perl knowledge will be introduced as best I can. You should be able to approach this with little or no mod_perl knowledge. Further study can be found in the excellent "Writing Apache Modules with Perl and C" by Lincoln Stein &Doug MacEachern (copies of which will NOT be available after this lecture...).

The techniques I seek to introduce you to are the basics of those which I have used to create the most successful electronic (all web based) Municipal Bond Market currently available. We do a large number of trades and see an enormous amount of volume on a daily basis; the large number of traffic is scaled well within Apache, and mod_perl has helped alleviate much of our old 'cgi related stress'.

When I first joined my current firm in February of 2001, they were struggling to stabilize their system. Much of the front-end of the site was running on perl-based CGIs built by a contract programmer. While the contractor did a fairly good job programming (although most of his success is due to our Software Architect designing an amazing front-end design which relies on some of the best Sybase stored procedure work I've ever seen), our traffic (which was light at the time compared to what we see now, but still pretty decent) was stressing the limits of CGI. The core problem with the CGI model is that every time a request is made for an application, the server must spawn a shell process and run the script. This in itself is costly; add to that the fact that in our case we were opening a connection to Sybase and returning anywhere from 100 - 3000 rows of data (our most popular category returned an average of 1100 rows of data), you can imagine what it was doing to our web server.

Our contractor left the firm and I was asked to come up with a way to increase throughput temporarily until our next generation product could go live within the next 2 months. What I ended up developing was a ground-up rewrite of the original CGI code, maintaining the back-end Sybase architecture. Within a month of my starting we were live with the new code and saw throughput capabilities beyond our expectations; we tripled our trades within 2 months. It took another 9 months for our next generation system to go live (due to changing market conditions changing feature requirements, etc); in the meantime our perl system continued to service our customers in an outstanding manner, while many of our competitors (all using closed source, proprietary software) closed their doors for good or bought one another up (or bought each other and then closed). Despite our new system being the centerpoint of our technology (it's a Java based system that handles large transactions, et al in ways that my system couldn't [limited more by architectural design than software capabilities]) the perl system still serves a very active role in our organization, serving as a light, fast, robust and scalable 'prototyping' stage to demonstrate new technologies and features that our firm is pursuing. Rapid prototyping is just something our firm has been unable to do with Java. Enough ranting and self indulgence however; let's focus on building some basic mod_perl.

The first step in setting up your mod_perl application is creating a 'startup.pl' file. This file will allow you to control your generic perl environment that every mod_perl process will use. One of the *most important* things is to set the library path for your mod_perl code (which are actually perl modules). Those of you who know perl know that perl has several internal paths that it searches for code. These are usually something like /usr/local/lib/perl5/site_perl/5.6.0001/blahblah.

You may also know that you can add 'extra' paths to this (this is good for users without root privilege to install their own modules, or people who want to manage homegrown modules without installing them in the system path). You can do this by either adding the directory to the environment variable PERL5LIB, or by a 'use lib $directory;' statement in your actual perl file. We are going to use the latter method to set our perl directory.

In your Apache config directory (standardly ~apache/conf), create the file startup.pl, and start it out as a standard perl script. It will look something like this:

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

Now, start a BEGIN block and lay it out as follows:

```
BEGIN {

use Apache();

use lib Apache->server_root_relative('lib/perl');

}

1; # this is a perlism; all perl modules should return true.
```

Once we finish configuring it, startup.pl is preloaded into every mod_perl process; this is why we do things like preloading commonly used modules in it, and setting the library path. What we have done here is two things:

We've preloaded the Apache module. This is the base module for all mod_perl functions.

We've set the root library path for our project to ~apache/lib/perl.

apache->server_root_relative queries the root server object for the configuration and returns the base path of the apache installation (the server root) relative to the directories we passed it. If my server root is /usr/local/apache, apache->server_root_relative('lib/perl') will return the string "/usr/local/apache/lib/perl".

There are some other things we can do in startup.pl like preloading modules we use frequently (eliminating the need for the app to load them the first time it's run; Apache loads it on startup). We can do this by simply "use"ing the module in startup.pl (outside of BEGIN);

this is left as an exercise for the reader.

Now that we've created startup.pl, we need to tell Apache about it. Open up your Apache config and add two directives at the top of it:

PerlRequire conf/startup.pl

PerlFreshRestart on

We have done two things here. The first is what we just talked about: we've told Apache about our startup file. Now when Apache starts up, it will 'Require' the startup file. This gives us one more added benefit through startup.pl aside from preloads; we can now check if code compiles. Basically, any modules you preload in startup.pl MUST COMPILE, or Apache will not start.

This is more or less a good thing; I try to preload all my modules if possible because I want to know the code doesn't work because it broke at Apache startup; not because a client got a 500 error page. The second line we've added to the config file is the PerlFreshRestart. What this does is tells Apache that if it restarts (usually by an apachectl restart), it needs to dump all modules in memory and reload them from disk. This is a good thing - it makes sure your new code gets loaded if you changed anything. And again, one of the side effects of having preloaded modules in startup.pl will be if you have an issue with any of the new code, the server will simply refuse to restart until the code loads cleanly.

Now let's work on building a simple application. This will be comprised of one database table and two parts: one is a login module that interfaces your browser's "Login/Password" popup box and uses a database for authentication instead of those .htpasswd flat files we all hate so much; the other is a simple mod_perl app that uses a light template system to dump out your user info from the database. This is a simple application but it will show you the ins and outs of basic mod_perl coding. You should leave here with a basic understanding of how to build something, and hopefully convert your CGIs over to mod_perl.

Let's start with our Database table. I'm going to setup a hypothetical mysql table here, and be as generic as possible.

We want a table that contains the following information about a user:

First Name

Last Name

Username

Password

Email Address

User Type (Admin, User)

A simple sql script for this would be something like this:

CREATE TABLE users (

first_name varchar(65) NOT NULL,

last_name varchar(65) NOT NULL,

username varchar(65) NOT NULL UNIQUE,

password varchar(65) NOT NULL,

email varchar(100) NOT NULL,

user_type enum('Admin','User') NOT NULL,

PRIMARY KEY (username)

);

This works in MySQL; It should work in PostGres and other ANSI SQL compliant databases. Create this table in a database. For my examples I've used the database "nylxs".

Now we're going to create our authentication layer. This simple script is going to interface Apache's Authentication phase (you can read more about Apache's phases in the mod_perl book mentioned above; it's well beyond the scope of this discussion), read the username and password from the web browser, and try to authenticate.

Let's start our script. Change into your apache base directory. If you haven't already, create the lib/perl directory (mkdir lib; mkdir perl). Now change into lib/perl and make an apache directory (mkdir apache). Apache is always the base package name for a mod_perl module; this defines to anyone looking at your module that it requires mod_perl. Change into apache, and create an NYLXS directory (mkdir NYLXS). We are creating a package hierarchy. After the Apache, we always want either our company or project name to easily define our stuff. In this case, if

someone requests apache::NYLXS::Foo, Perl will end up looking in ~apache/lib/perl/apache::NYXLS for Foo.pm.

Lets change into the NYLXS directory and open a new file named

DBAuth.pm.

Type the following:

package apache::NYLXS::DBAuth;

What we've done is declared explicitly our namespace for this file: the namespace is apache::NYLXS::DBAuth. It is important in mod_perl to clarify your namespaces. I'll explain why later.

Now add this:

use strict;

use apache::Constants qw(:common);

We've done two things here. One of which is use strict; which I hope most of you perl programmers recognise. We're going to digress a bit here so bear with me...

One of the features of Apache as a server is it's forked model. Basically, a parent process (sometimes with root privileges, so you can listen on privileged ports) listens for incoming requests; when it starts up it spawns off a set number of child processes (usually running as nobody or some other unprivileged user) who are slated to handle requests. When a request comes in, the parent process handles it by handing it off to a child process who does all the work. However, the child does not go away when it's done, and any global variables set within it, memory allocated (that was never garbage collected) and all sorts of other fun things bad programmers do will remain. It is important therefore to make sure that you run in use strict; so that you always declare your namespace and avoid global variables. Keeping proper namespaces should minimize on memory leaks.

One of the first mistakes I made when programming mod_perl was this very thing; I couldn't figure out why data from previous requests kept showing up in later ones. It turned out I wasn't properly scoping my variables and the data was persisting from process to process. This can sometimes be a good thing, but we won't go into that today. You may find in your own explorations, certain times when it's useful.

The Apache::Constants package includes a bunch of variables and constants which may be used, for example the response "OK".

Here's the rest of the script:

```
use DBI;

sub handler {

my $r = shift;

# Declare our variables

my $sql;

my $sth;

my $dbh;

my $remote_user;

$dbh =
DBI-
>connect("DBI:mysql:database=nylxs;host=localhost","roo
t","") or
die $DBI::errstr;

my ($res, $password) = $r->get_basic_auth_pw;

# what user are they authed as? (the Basic Auth
HTTPAuthUser their
Browser sent)

$remote_user = $r->connection->user;

unless ($remote_user &&$password) {

$r->note_basic_auth_failure;

$r->log_reason("Username or Password returned
empty",$r->filename);

return AUTH_REQUIRED;

}

$sql = qq|

SELECT

username,

password,

user_type

FROM

users

WHERE

username = '$remote_user'

AND

password = '$password'

|;

$sth = $dbh->prepare($sql);

$sth->execute;

my $cursor = $sth->fetchrow_hashref;
```

```
unless ($cursor->{'web_user'} eq $remote_user) {

$r->note_basic_auth_failure;

$r->log_reason("Authentication Failure Detected.
Username:
'$remote_user' Password:

'$password'",$r->filename);

return AUTH_REQUIRED;

}

$ENV{'USER_TYPE'} = $cursor->{'user_type'}

$ENV{'REMOTE_USER'} = $remote_user;

$sth->finish;

OK;

}

1; # this is a perlism; all perl modules should return true.
```

You may recognize the DBI stuff for the database; if you don't I recommend reading the docs for DBI on CPAN to get up to speed. There are two new concepts here which you should note:

sub handler.

mod_perl always looks in a module for a subroutine called "handler". This is the default subroutine that mod_perl calls for execution. It passes in an Apache object $r. $r is the commonly used variable name for the Apache object which is always passed into handler subroutines. You can pass this around to other variables as well. This contains methods for accessing Apache internals, sending back information, getting information and many other things.

You'll note that we call several methods from Apache, such as $r->get"_basic_auth_pw, which returns an array containing two strings: the internal Apache reference to the status code it sent back to the client, and the password the client sent. $r->connection->user steps into Apache's subobject for the client connection and returns the username they sent; we use all of this information to determine if this user is authorised to user the site.

We select the data out of the database and verify it; if at any point anything goes wrong we return back the "Authorization Required" message to the browser. The first few times a browser gets this it will ask the user for the username/password combo again. After a few failures it may give up.

When we're done, we pass some data into environment variables. Since we are in the Authentication Phase of Apache, we can use the Environment stack to pass some stateful information down to later phases (Such as composing and returning the document, which we'll cover later). This is one of the best ways to keep state in mod_perl.

Your script is now ready to test. For a simple test, let's do three quick things: First, make sure you add a test user with full info to

the database.

Example:

```
insert into users
(first_name,last_name,username,password,email,user_type
) values

('Brendan','McAdams','bmcadams','nylxs','rit@jacked-
in.org','Admin');
```

Then, in the htdocs directory, create a quick testauth.html file that contains some text, like "Hi, NYLXS".

Finally, we need to add a quick section to Apache:

```
<Files "testauth.html">

Options All

AuthName "NYLXS mod_perl Auth Test"

AuthType Basic

PerlAuthenHandler Apache::NYLXS::DBAuth

require valid-user

</Files>
```

This tells Apache that it needs to require a valid authenticated user for the file testauth.html, and that it should let our Apache::NYLXS::DBAuth module handle the authentication.

Start Apache and try accessing your document; everything should work!

You can email me at **rit@jacked-in.org** if you have any problems/questions when you try this at home.

Privacy in the Aftermath of September 11

Paul Rodriguez <sangretoro@yahoo.com>

Privacy is the most comprehensive of all rights and the one most cherished by a free people. So wrote Justice Louis Brandeis in 1928(1). Defending this right becomes all the more important in an era of constant technological advancement, where the means of monitoring individuals are proliferating rapidly in new and varied ways. As our lives become increasingly enmeshed with this technology, we must be aware of the ways in which our privacy is effected. It is no surprise, therefore, that issues of privacy have taken on a greater profile since the dawn of the Internet. As a community centered around the rights of individuals in relation to technology, the Free Software community is uniquely concerned with the issue of privacy.

This August, NYLXS had the pleasure of attending a conference at Princeton University, which focused on Privacy in the Aftermath of 9/11. Held by the ACLU-NJ and the NJ Bar Association,the conference brought together 20 professionals, who work in the field of privacy, for a panel discussion on the current political environment, recent legislation, and ramifications for privacy and liberty today.

The topic for the early part of the day was recent legislation. In particular, the discussion revolved around two of the most important recent laws, The Patriot Act, and the NJ Open Public Records Act.

The Patriot Act (H. R. 3162)(2) was signed into law on the 25th of October, 2001, as a response to the terrorist attacks in New York and Washington. The law relaxes regulations, expanding investigative powers of the state. The act has been among the most controversial pieces of legislation to follow the September 11 attacks and certainly has widespread political consequences. Critics of the act have maintained that by reducing judicial supervision, it significantly reduces protection from government abuse while falling short of providing increased security from future attacks. Speakers expressed concern at the conference for the deleterious effects the Patriot Act could have on our system of checks and balances. It was felt that by reducing judicial oversight, the law undermines the Judiciary in its traditional role of policing the Executive Branch. This has the potential to create a dangerous imbalance of powers that could have far reaching effects for civil liberties. The new law, therefore, significantly expands the powers of

the executive branch and allows the government more freedom to conduct its affairs in secret. Overall, the Patriot Act was characterized as hastily written and not fully considered, especially in regard to providing adequate protection for privacy and civil rights.

The new law eases privacy restrictions for various forms of investigations, particularly Internet surveillance, and affords the government reductions in liability from harm. Under the new law, for example, one does not need to be the subject of a criminal investigation to be the target of surveillance. In order to conduct Internet surveillance on an individual, a warrant must be obtained from a judge with special training in Internet matters. Such warrants, however, are in practice, fairly trivial to obtain. More leeway is now given on notification of wiretapping, and the new law also eases restrictions on information sharing between the government and Internet service providers.

Scott Christie, of the NJ Division of Criminal Justice, informed us of the current state of police Internet surveillance. He described law enforcement as being essentially self-policing and without oversight in these matters. Law enforcement officers, said Christie, are motivated and kept in check by a sense of personal decency and public obligation.

J.C. Sayler, of the ACLU, compared the current state of Internet privacy to high speed car chases. The analogy refers to the danger to the public interest when potentially criminal information is pursued recklessly. We are in a very similar situation now with Internet privacy, he says, as we were in the early days of automobile pursuits, before the overall dangers had been fully recognized. Even today, he reminds us, understanding the dangers involved with such chases, and a great deal of regulation exists regarding them, law enforcement officers remain immune from liability for personal or property damage.

NJ Open Public Records Act

On January 8, 2002, NJ passed a law reforming access to its public records.(3) New Jersey formerly had one of the most restrictive policies regarding government records. The new law is designed to significantly facilitate public access and civil involvement in government. The question was posed to the panel whether open access to government records would help maintain privacy, or endanger it.

William Kearns, of Vasallo, Guest & Kearns, took the possition that commercial interests were a greater threat to privacy than government. He felt that there was potential for corporate misuse of the information, particularly if available in digital form. While recognizing his concerns, a member of the ACLU from the audience responded that the greatest threat to privacy is government in secret. She asserted that maintaining open public records [was] our best safeguard of individual privacy. Joe Tyrell, from the NJ Foundation for Open Government, agreed with this sentiment, and warned of the danger of distorting ideas when discussing privacy. He cited as an example, that as 1200 Arab-Americans were secretly detained for several months in the aftermath of September 11th, government officials claimed to be protecting privacy, by refusing to release the names and countries of origin of these individuals to lawyers wishing to represent them.

Bruce Solomon, Custodian of Records for the Attorney General, concurred with Mr. Kearns, that the government has an obligation to ensure the privacy of its citizens. This is particularly the case when individuals, with a reasonable expectation of privacy, have entrusted personal information to the government. He expressed appreciation for the delicate issues involved, and understood that there is a fine line between protecting private information and government secrecy. He likened personal information to a scrambled egg. Just as an egg cannot be unscrambled, private information cannot be resealed after being released. Mr. Solomon assured those present that any limitation to right to information shall be construed in favor of the public.

The latter part of the afternoon was dedicated exclusively to Internet

privacy and technology, including Biometrics.

Marc Rotenberg, Executive Director of the Electronic Privacy Information Center (www.epic.org), began by posing the question as to whether surveillance cameras truly offered a security blanket. He cited statistics which called this assumption into question. In England, for example, where video surveillance has been in use much more extensively than in the U.S., with nearly two million cameras currently in operation, crime has actually increased since the implementation of the surveillance. While he does not attempt to make a correlation between increased crime and video surveillance, he does feel that this is a clear indication that it is not an effective deterrent against crime.

Mr. Rotenberg urged us to consider the matter of public surveillance with a measure of caution. We need to maintain a matter of perspective. With regard to the recent terrorist attacks, for example, we need to first assess what happened and then determine what measures can be implemented prevent another attack. He found it very hard to justify a surveillance network (that is currently being proposed and tested nationwide), to watch and surveille individuals who have not been accussed of or are under suspicion of anything. If, under further analasys, the surveillance network, and a national ID system, cannot be justified as a reasonable deterrent against terror, what then is it's purpose? Mr. Rotenberg feels that it can only be described as a form of social and political control.

Central to the issue of the legality of electronic surveillance is whether or not the individual has a reasonable expectation of privacy in any given situation. This was originally made explicit in Katz v. U.S (1967). However, the issue becomes increasingly more complex the more forms of electronic surveillance exist. Mr. Rotenberg poses the question to us: if technology makes it possible to capture everything that is said, what then can be the expectation of privacy in those circumstances? If a man walks down the street with his family, for example, and leans over to whisper to his wife out of earshot of his children, does he have a reasonable expectation of privacy? And if officers on an rooftop opposite him capture the conversation? Would the man's legal standing change if there was a public statement made explaining that it was possible for what they said, even in a whisper to be captured? Mr. Rotenberg doubted whether such a statement would change the man's legal standing. After all, a person's reasonable expectation of privacy really comes down to community standards. It is through precedent that we establish what our community standards of privacy are. In such a situation, what is the basis of expectation of privacy? Is it what is technically feasable? What has been disclosed to the public? Or is it the constitution?

Mr. Rotenberg as Executive Director of EPIC, runs a program entitled Observing Surveillance (http://www.observingsurveillance.org/) . This program seeks to increase public awareness of video surveillance in America, particularly in our nation's capitol. He feels that it is improper for police to collect images sereptitiously of individuals who are engaged in peaceful activity. Observing Surveillance calls attention to the surveillance cameras in Washington, D.C. In the hopes of animate the public to defend their rights. EPIC is proposing that we construct a new statute which regulates video surveillance as a type

of technologically advanced surveillance, and that it be regulated as such. He also believes that additional legal obligations should follow from hightened surveillance. He again posed the question as to whether such cameras can hope to prevent terrorist attacks. The answer he feels is obvious. No. Do cameras make us feel safer? Perhaps. He sited statistics in the UK,

where there are currently nearly two million cameras, and crime has increased. What then is the motivation for such cameras? Perhaps to prevent petty crime and as a form of social control. Sites the case of the Super Bowl. Originally for terrorists, matched 19 ticket scalpers (how did they get in the system) all of which were mismatches and officers could not reach them before they disappeared into the crowd. This was later tried once again in Tampa, FL with disastrous results. It was supposed to be a 9 month test program. After a series of continuous false matches, the program was quickly discontinued after less than 2 months with not one accurate match.

Andrew Appel, Professor of Computer Science, spoke about the privacy from the perspective of an individual using the Internet. The first thing he discussed was the use of encryption software to communicate online. He drew connections between the system of paper mail we have in this country and e-mail. He likened unencryoted email to sending mail without an envelope. That envelope, while not being fool-proof, provided an expectation of privacy. The only problem so far with putting envelopes on your email has been the inconvenience and public laziness. He made sure to point out that the only system of encryption which was at all valid would be using open source software. He says that we can basically take this is a given and starting point for discussion. He also brought up the issue of whether Source Code could be considered speech or not, but drew very little conclusions as he felt that it was outside of his role to be able to judge. He also brought up a recent News report in which the FBI considered using viruses as a form of surveillance. He felt that is was reasonable for Interent users to expect the same level of anonymity on the internet as they enjoy in real life. Namely, that they can read anonymously over the Internet as they would in a library, and that they could purchase a book anonymously as they could with cash in a bookstore. Also, he felt it was reasonable for us to be able to expect to speak anonymously over the Internet, and engage in commercial activity. He did discuss that much Internet monitoring also is done by private companies. For example, he mentioned that it was not the government that monitors transactions under $10,000 on the Internet, but rather commercial corporations. They are pressuring the transfer of commercial information in order to discriminate prices.

Abby Notterman from the Internet Crimes Group (an Internet investigative, consulting, and forensic services organization) discussed how much more information is collected for any one thing than is actually required for the stated purpose. She said that records should be held stating why, what, and by whom information was recorded, with strict enforcement and real penalties for those who violate them. Such systems also need to be used overtly. She had a diagram of better and worse uses of biometric data:

better worse

overt use covert use

opt-in mandatory

fixed duration indefinite duration

Personal Storage Verification Template Databse Identification

Barry Steinhardt, Associate Director of the ACLU clarified the ACLU's possition on the use of biometric. He pointed out that the ACLU and civil libertarians in general are not necessarily opposed to the use of biometric technology, but to recognize that because of the potential for misuse, it must be used responsibly under certain conditions.

ACLU's fair information Principles

1.Notice

2.Consent

3.Limitations on Secondary Use

4.Access And Right to Correct

5.Secure

Breeder documents: not reliable, contain too much information and a very lucrative black market. Story of 2 birth certificates

1)Your personal information should never be collected or given out without your knowledge and permission.

2) Organizations must let you know why they're collecting your info; and they can't use it for reasons other than the one you gave permission for (unless they get a new permission from you.)

3)They must ensure the privacy of the personal info they collect or maintain on you, retaining only what is necessary info, and only for as long as it's needed.

4)You should have the right to examine, copy, and correct your own personal information.

5)There must be no national ID system -- either in law or in practice.

6)Unrelated data bases must be kept strictly separate so info can't be cross-referenced.

7)Personal "biometric" data -- your fingerprints, DNA, retina/iris scans, etc. -- must not be involuntarily captured or used (except for

fingerprinting criminals.)

8)The government must not prohibit or interfere with the development of technologies that preserve anonymity (such as encryption).

9)These principles should be enforceable by law. And no service, benefit or transaction should be conditioned on your waiving your privacy rights. Produced by the American Civil Liberties Union.

Mr. Steinhardt went on to state that when considering the use of any new for of technology, there are some basic questions that need to be

asked first:

1)Does the technology work? (in the case of current biometric technologies, the answer would be no.)

2)Will use of the technology have the desired stated results? Only after the first two criteria are met can we begin to deal with the subsequent issues, such as:

3)What are the legal, ethical, moral ramifications of using such technology?

A proposed a set of criteria we should use when considering questions of new forms of surveillance and security measues.

References:

(1)Justice Brandeis wrote a vigorous dissent in the case of Olmstead v. U.S. 277 U.S. 438 (1928) which upheld the right of Elliot Ness and his untouchables to wiretap the telephone lines of bootleggers as long as it was done at a point between the defendant's homes and their offices. Let's take a look at some of the passages (paraphrased) in this famous dissent:

"The makers of our Constitution understood the need to secure conditions favorable to the pursuit of happiness, and the protections guaranteed by this are much broader in scope, and include the right to life and an inviolate personality -- the right to be left alone -- the most comprehensive of rights and the right most valued by civilized men. The principle underlying the Fourth and Fifth Amendments is protection against invasions of the sanctities of a man's home and privacies of life. This is a recognition of the significance of man's spiritual nature, his feelings, and his intellect. Every violation of the right to privacy must be deemed a violation of the Fourth Amendment. Now, as time works, subtler and more far-reaching means of invading privacy will become available to the government. The progress of science in furnishing the government with the means of espionage is not likely to stop with wiretapping. Advances in the psychic and related sciences may bring means of exploring beliefs, thoughts and emotions. It does not matter if the target of government intrusion is a confirmed criminal. If the government becomes a lawbreaker, it breeds contempt for law. It is also immaterial where the physical connection of the wiretap takes place. No federal official is authorized to commit a crime on behalf of the government." (Justices HOLMES and STONE also dissenting, agreeing with Justice BRANDEIS)

O'Connor, T. 2002. The Right of Privacy http://faculty.ncwc.edu/toconnor/325/325lect04.htm Last Updated: 02/12/01

Full text of Olmstead v. U.S. 277 U.S. 438 available at:

http://laws.findlaw.com/us/277/438.html

(2)Full Text of the Patriot Act:

http://www.eff.org/Privacy/Surveillance/Terrorism_militias/20011025_hr3162_usa_patriot_bill.html and http://www.epic.org/privacy/terrorism/hr3162.html

EFF Analysis of Patriot Act: http://www.eff.org/Privacy/Surveillance/Terrorism_militias/20011031_eff_usa_patriot_analysis.html

ACLU analysis of Patriot Act: http://www.aclu.org/congress/l110101a.html

It may be worth noting that I was unable to find any careful analysis of the Patriot Act that was at all positive.

(3)NJ Open Public Records Act P.L. 2001, c. 404 http://www.njleg.state.nj.us/2000/Bills/PL01/404_.HTM

(4)Digital availability of data, felt Kearns, facilitates aggregating personal information held in separate documents into one database, creating a profile of individuals from the data. To illustrate his point, Mr. Kearns offered as an example the hypothetical case of a burglar able to conduct a cross-referenced search of electrical permits, senior citizen tax exemption requests, dog-licenses, and a list of city residents, to come up with a list of households likely to contain elderly individuals, who have neither a dog nor a burglar alarm.

(5)Joe Tyrell, from the NJ Foundation for Open Government, agreed with this sentiment. He cited what he reffered to as a common rationalle law enforcement agencies government information on individuals (most often that acquired by law enforcement agencies) are often kept secret with the excuse that it may contain incomplete, misleading, and/or patently false information. Mr. Tyrell described the importance of having access to those records if they do, in fact, contain incorrect information. It is an individual's right to be able to review and correct those records as necessary. EPIC http://www.epic.org

Observing Surveillance http://www.observingsurveillance.org/

http://www.library.ca.gov/CRB/97/05/

The URL of the conference itself:
http://www.wws.princeton.edu/~privacy/

The Road From Here Lessons of the recession for Free Software

By Ruben Safir President - NYXLS

"When talented people gather together for the purposes of developing opportunity for members great things can happen."

With the current recession and dot-com crash of late 2001, the rosy optimism of the GNU/Linux based IPO's, and the many of the accomplishments of the late 1990's have been replaced by high unemployment among free software people, and ever more virulent attacks by the marketers of proprietary software and hardware products. Many of the organizational models within the Free Software community, "Linux User Groups" and the efforts of groups like 2600 have served very well in bringing Free Software into the public eye. Rick Moen's FAQ on how to setup a traditional LUG and its defining of core activities, has helped developed GNU/Linux advocacy and has been responsible for for the broad placement of Free Software in homes, schools and businesses. Installfests, monthly technical meetings, stammtisch drinking sessions, and the spirit of volunteerism have greatly aided in the development of software for practical use. But that model is reaching it's maximum ability to effect change. We need new methodologies to ensure the freedom to innovate, compete, and to educate in the digital age.

One of the important lessons in the Jewish tradition comes from the collective work called "Teachings of Our Fathers", which says, "Without bread there can be no Torah". The fundamental concept here is that as individuals strive in freedom for their advancement and improvement, an economic foundation has to exist for the basic necessities of living. Economic freedom and political freedom are linked in a duet. Economic freedom can not be achieved without political freedom, nor can political freedom be achieved without economic freedom. We must have both.

The importance of political policy, and digital freedom is directly tied to the economic fortunes of businesses and individuals. This truth is known by every businessman in the world. Global Corporations have poured millions of dollars into political campaigns, flushed the Halls of Congress with money, and donated huge sums of money to the major political parties of the United States in the hope of effecting favorable legislative change for their industries and individual companies. Open Secrets (http://www.opensecrets.org) lists the 100 largest contributers to political campaigns. The list is sprinkled with Fortune 500 companies including Microsoft, Philip Morris, AT&T AFG, BellSouth, Lockheed Martin, Bristol-Myers Squibb, AIG, UPS, Pfizer, Time-Warner and more, and this is in addition to the donations by individuals who represent corporate interests. Here is a breakdown by industry:

Finance/Insurance/RealEstate $79,661,902

Misc Business $47,447,874

Communications/Electronics $35,928,843

Lawyers & Lobbyists $35,416,790

Labor $33,865,904

Other $33,216,163

Health $29,034,066

Energy/Nat Resource $21,733,893

Agribusiness $19,489,089

Transportation $16,835,110

Ideology/Single-Issue $16,582,843

Construction $13,564,759

Defense $6,145,345

All of these businesses are funded by profits, and can put money and staff into the pursuit of their legislative agendas. They use their full time legal teams to effectively use litigation as a political and business tool, and the result has been a series of serious blows to the Free Software movement which is now facing political annihilation. This has caused a negative economic impact to Free Software developers and end users, as well as trouble in the general economy.

It's time for Free Software people to go on the offensive. People with a technical inclination, by their nature, tend to stick with technical experimentation, and to leave marking and political strategy to others. If they expect to maintain the freedom to do development they're going to have to change.

Late last year, we started a new experimental organization which we hoped will develop new modalities for Free Software advocacy. Many of the things we have been doing have been very successful, so we are encouraged by the initial results. The organization is called NYLXS, and when we started it, we started with a number of premises that helped us succeed.

The first premise was that NYLXS would not be a typical LUG. Instead, we would concentrate our activities in three major areas: business development, organizational funding, and politics. We wanted to address current issues in an effective manner and conduct ourselves in a businesslike fashion. These activities need to be targeted by other organizations, as well, so let's s now examine the NYLXS plan of action so that other organizations can emulate it.

For our tactical model, we looked to one of the most successful grass roots organizations in history: Tammany Hall. Tammany Hall was very effective at at three core political tasks: organizing volunteers, delivering votes, and acting as a provider of jobs and contracts. Ignoring, the graft and corruption, which we don't intend to emulate, we looked at some of the other things that made Tammany Hall so successful. Among those, we saw that one of the keys success of Tammany Hall was the bond built by the organization between it's members and the leadership.

This bonds was built by the tireless advocacy of it's membership, and by the doling out the financial rewards for political success. Tammany Hall carefully chose it's battles and functioned top down when putting people on the street and getting people elected. Nothing is as effective as targeting political candidates in their bedrooms.

Currently, Free Software organizations are short on cash, but do have great potential to mobilize volunteers. If they want to maintain their freedom, one of the primary missions that Free Software organizations must undertake is political involvement. Targeting key members of Congress to become fast adopters of pro-Free Software issues can be effectively organized by Free Software groups throughout the country. We must focus attention on elections and constituencies on the grass roots level. All politics are local.

The problem is that this requires real work, and true volunteerism. It's not fun. Going from door to door, and pamphleting supermarkets requires persistent effort and can be a humbling experience. But it can be done, and doing it will demonstrate to elected officials that we can be effective in getting people elected and unelected. We can even run candidates and put individuals into office.

The week after the destruction of the World Trade Center a local NYC advocacy group, New Yorkers for Fair Use, had members in the street. Its members were going door to door with the "Save the Libraries" campaign. That campaign targeted members the House Judiciary Committee and was very effective at getting pro-digital freedom issues heard. That NY Fairuse was able to organize this effort straight through that disaster, your organization can do the same.

A key component to being effective in this level of organization is the need to develop leadership. Leadership development should be a core component of all Free Software organizations, and the leadership of these organization absolutely must commit themselves to the identification of future leaders. It should have a system for mentoring new members and training those members for leadership. Leadership training needs to be built into all LUGs and Free Software organizations in order to propel the groups forward into the future. It's a good idea to establish a committee within your organization to direct this activity.

Another aspect of running the political arm of a Free Software group is the development of a social services for your organization. Your group is dependent on the loyalty of its members. This loyalty is best earned through a program of membership services. Often, providing valuable help to membership requires nothing more than than a schedule and a kitchen table. When talented people gather together for the purposes of developing opportunity great things can happen.

What services should your organization provide? Here are some suggestions. Mentoring, internships, job banks and resource networking, are all areas that can be formalized as a functions of your membership. People who work for your organization need to feel part of the family and the leadership must actually care about the individual well being of group members. In this time of high unemployment among talented free software people, your organization has a huge pool of untapped assets waiting to be exploited for the benefit of the larger group. Workshops, educational programs, business planning, and Job searching can all be implemented at a minimum cost to the organization.

People often need help with surprisingly simple tasks like resume writing and a url to put resumes on line or to advertise services.

State and local governments will often fund educational programs and job placement efforts. Leaders need to organize their efforts in getting state funding for such programs.

Clients of Free Software consultants are often huge resources when they are happy with the work of Free Software people. They can provide evening space for classes, projectors and white boards and such services as accounting for your organization, or helping to develop publicity channels. These clients should be sought out and we should take advantage of the services they're willing to provide.

It's absolutely key that your political arm provide at least some of these services, so members will develop the loyalty necessary for the hard work of door knocking. Members must come to see themselves as real stake holders in the outcome of digital rights legislation. Both members and client enterprises, need to feel empowered through an association with your organization and Free Software.

Your job as a Free Software leader is to build bridges through education and member services. As more people become indebted to your organization for work, business and personal development, its reach will become broader and it will become more influential in political campaigns and IT buying decisions.

One of the key services that a Free Software organization can and should provide is targeting local industry for complete Free Software desktop and server solutions. It is very important for local Free Software groups to join together, identify local industries, and to build complete functional business environments. Let's look at the accounting industry for example.

A group of dedicated leaders from your organization would meet, and call accounting firms to make appointments with the partners or the employees in those firms responsible for IT purchasing decisions. Then, they'd sit down with those individuals to identify the current software being used by the firm and to determine what additional software needs they may have which are not currently being met. After obtaining this information, your job is now defined and the work cut out for you. You now need to produce a plan for filling ALL of the needs for the firms desktop and server use. (And I mean ALL of their needs. No company can covert its desktop platform to Free Software, or GNU Linux, unless every need that they have is met. Your job is to assemble a desktop and server solution which can fill all the needs.)

For this example, an accounting firm, it's not good enough to just have accounting software. Such firms also need programs that track employee activity, invoice clients, track document movement, handle faxing needs, and provide word-processing and other standard office applications. If any one of its needs isn't satisfied, then the company can not move to a free software platform or entrust its business too you.

Once your organization has produced a complete solution, it can demonstrate that product for the whole industry, and share the development with fellow members. It's likely that members will need to be matchmakers between free software developers and the targeted industry. They may also need to develop some pieces of the system themselves. And they will have to be prepared to develop a comprehensive training program for the new platform. Training and support can be distributed to working members of your organization on a paid basis as a social service of your organization. Use your organizations Public Relations engine to promote your efforts, your members and your solutions. Be prepared to sell the concept of Free Software to businesses and unchain them from the licensing fee structures and restrictive agreements that impair the organization's ability to adapt to the changing needs of the industry.

Another area to explore, is federal, and local Free Software government opportunities. The federal government's biz ops site needs to be carefully combed everyday for potential bids. The process of forming a bid and identifying possible opportunities is daunting to individual consultants without experience in handling this process. Businesses have full-time staff people who handle this process and fill government contracts. Local organizations need to develop similar in house expertise, again, offering this service to working members and building loyalties. Government contracts are especially important in recessionary times like today's. Such contracts can feed a lot of Free Software families. This would be a good thing(TM).

This paper has outlined many new avenues of activity for Free Software Organization and GNU/Linux user groups, which are very different from the sort of activities which LUG's normally perform. Key to producing this level of organized activity is funding. Free Software groups can fund many of their activities from within. Various means are available for accomplishing this task. NYLXS, for example, has had considerable success with its educational program. While charging cut-rate prices for classes ($200 - $300 per course), NYLXS has developed a comprehensive FreeSoftware education program that successfully enlarges membership. This also serves the role of organizational fund raising. It's success depends on the volunteer efforts of it's membership. Our groups members have also started to: build videotape assets for sale, provide ISP networking services, and to provide on-line curriculae. Teaching accomplishes two goals for Free Software organizations: the foundation for future membership and revenue to fund activities. Such a program can also be funded by local and state governments as part of their unemployment and training programs. If targeted well, a teaching program can draw membership from the untapped pool of minorities and women into the Free Software field.

Funding for necessary political and business activities can also be generated by annual dinners, sponsored workshops, journal publication, and public speaking. Membership dues, while not a great fund raiser itself, should be used to determine the membership of Free Software groups. Since so much work is being done by the organization, membership efforts must focus on committed members. A nominal charge for membership sets a bar for entry into the organization, and shows a commitment by the individual to the organization. It's your goal to build the organization one committed member at a time. Each individual brings to the organization a potential. It is your job, as a Free Software leader to develop that potential for the benefit of the individual and the organization.

With regard to funding lobbying and business endeavors for Free Software, one of the long term goals of your group should be to acquire a permanent location to hang your hat. Having a permanent location gives to you a powerful tool in attracting students, new members, and additional income generating opportunities. It provides a permanent teaching space as well as a place for demonstrations and meetings, space for conferences with local politicians and a rental possibility for other groups. In short, you need a clubhouse for your club.

GNU/Linux user groups and Free Software organizations just have to do more to protect our freedom and to survive the economic downturn. We simply are not doing enough to protect our interests. We must do more if we expect to help protect the the freedom of a larger society which is growing increasingly dependent on digital information. Proprietary software vendors are demanding the control of all digital information, and the prevention of free competitive markets. The road is laid out before each and every one of us, and it is up to us to choose the proper path.

GNU Bayonne: telephony and web services for free software

by David Sugar sugar@gnu.org, http://www.gnu.org/software/bayonne

Introduction

Even without considering all the various reasons of why we must have Free Software as part of the telecommunications infrastructure, it is important to consider what the goals and platform needs are for a telephony platform. Historically, telephony services platforms had been the domain of realtime operating systems. Recent advances in CTI hardware has made it possible to offload much of this requirement to hardware making it practical for even low performance systems running efficient kernels to provide such services for many concurrent users.

Telephony services are usually housed in phone closets or other closed and isolated areas. As such, remote maintainability, and high reliability are both important platform requirements as well. The ability to integrate with and use standard networking protocols is also becoming very important in traditional telephony, and certainly is a key requirement for next generation platforms.

So we can summarize; low latency/high performance kernels, remote manageability without the need for a desktop environment, high reliability, and open networking protocols. This sounds like an ideal match for a GNU/Linux system. For these reason, and with the abundance of computer telephony drivers available for it, we choose to build telephony services such as GNU Bayonne primarily under GNU/Linux.

Our initial goal for GNU Bayonne 1.0 is primarily to make telephony services as easy to program and deploy as a web server is today. We choose to make this server easily programmable thru server scripting. We also desired to have it highly portable, and allow it to integrate with existing application scripting tools so that one could leverage not just the core server but the entire platform to deliver telephony functionality and integrate with other resources like databases.

GNU Bayonne, as a telephony server, also imposes some very real and unique design constraints. For example, we must provide interactive voice response in real-time. ``realtime'' in this case may mean what a person might tolerate, or delay of 1/10th of a second, rather than what one might measure in milliseconds in other kinds of real-time applications. However, this still means that the service cannot block, for, after all, you cannot flow control people speaking.

Since each vendor of telephony hardware has chosen to create their own unique and substantial application library interface, we needed GNU Bayonne to sit above these and be able to abstract them. Ultimately we choose to create a driver plugin architecture to do this. What this means is that you can get a card and api from Aculab, for example, write your application in GNU Bayonne using it, and later choose, say, to use Intel telephony hardware, and still have your application run, unmodified. This has never been done in the industry widely because many of these same telephony hardware manufacturers like to produce their own middleware solutions that lock users into their products.

TGI support and plugins

To create GNU Bayonne we needed a portable foundation written in C++. I wanted to use C++ for several reasons. First, the highly abstract nature of the driver interfaces seemed very natural to use class encapsulation. Second, I found I personally could write C++ code faster and more bug free than I could write C code. Why we choose not to use an existing class framework is also simple to explain. We knew we needed threading, and socket support, and a few other things. There were only a couple of existing framework that did most of these things and were available in any form of free software license, but were very large and complex which did far more than we needed. We wanted a small footprint for Bayonne, and the most adaptable framework that we found at the time, ACE, typically added several megs of core image just for their runtime environment.

GNU Common C++ (originally APE) was created to provide a very easy to comprehend and portable class abstraction for threads, sockets, semaphores, exceptions, etc. This has since grown into it's own and is now used as a foundation of a number of projects as well as being a part of GNU. In addition to having portable C++ threading, we needed a scripting engine. This scripting system had to operate in conjunction with a non-blocking state-transition call processing system. It also had to offer immediate call response, and support several hundred to a thousand instances running concurrently in one server image.

Many extension languages assume a separate execution instance (thread or process) for each interpreter instance. These were unsuitable. Many extension languages assume expression parsing with non-deterministic run time. An expression could invoke recursive functions or entire subprograms for example. Again, since we wanted not to have a separate execution instance for each interpreter instance, and have each instance respond to the leading

edge of an event callback from the telephony driver as it steps thru a state machine, none of the existing common solutions like tcl, perl, guile, etc, would immediately work for us. Instead, we created a non-blocking and deterministic scripting engine,
GNU ccScript.

GNU ccScript is unique in several ways. It is step executed, and is non-blocking. Statements either execute and return immediately, or they schedule their completion for a later time with the executive. A given "step" is executed, rather than linearly. This allows a single thread to invoke and manage multiple interpreter instances. While GNU Bayonne can support interacting with hundreds of simultaneous telephone callers on high density carrier scale hardware, we do not require hundreds of native "thread" instances running in the server, and we have a very modest cpu load. \\

Another way GNU ccScript is unique is in support for memory loaded scripts. To avoid delay or blocking while loading scripts, all scripts are loaded and parsed into a virtual machine structure in memory. When we wish to change scripts, a brand new virtual machine instance is created to contain these scripts. Calls currently in progress continue under the old vm and new callers are offered the new vm. When the last old call terminates, the entire old vm is then disposed of. This allows for 100% uptime even while services are modified.

Finally, GNU ccScript allows direct class extension of the script interpreter. This allows one to easily create a derived dialect specific to a given application, or even specific to a given GNU Bayonne driver, simply by deriving it from the core language thru standard C++ class extension.

To be able to create useful applications, it is necessary to have more than just a scripting language. It requires a means to be extended so that it can incorporate things like database access libraries or other functions that fall outside of the scope of the scripting language itself. These extensions should be loaded on demand only when used, and should be specified at runtime so that new ones can easily be added without the need to recompile the entire server.

To support scripting extensions we have the ability to create direct command extensions to the native GNU Bayonne scripting languages. These command extensions can be processed thru plugin modules which can be loaded at runtime, and offer both scripting language visible interface extensions, and, within the plugin, the logic necessary to support the operation being represented to the scripting system. These are much more tightly coupled to the internal virtual machine environment and a well written plugin could make use of thread pools or other resources in a very efficient manner for high port capacity applications.

When writing command extensions, it is necessary to consider the need for non-blocking operations. GNU Bayonne uses ccScript principally to assure non-blocking scripting, and so any plugin must be written so that if it must block, it does so by scheduling a state operation such as "sleep" and performs potentially blocking operations in separate threads. This makes it both hard and complex to correctly create script extensions in this manner. While GNU Bayonne's server scripting can support the creation of complete telephony applications, it was not designed to be a general purpose programming language or to integrate with external libraries the way traditional languages do. The requirement for non-blocking requires any module extensions created for GNU Bayonne are written highly custom. We wanted a more general purpose way to create script extensions that could interact with databases or other system resources, and we choose a model essentially similar to how a web server does this.

The TGI model for GNU Bayonne is very similar to how CGI works for a web server. In TGI, a separate process is started, and it is passed information on the phone caller thru environment variables. Environment variables are used rather than command line arguments to prevent snooping of transactions that might include things like credit card information and which might be visible to a simple ``ps'' command. The TGI process is tethered to GNU Bayonne thru stdout and any output the TGI application generates is used to invoke server commands. These commands can do things like set return values, such as the result of a database lookup, or they can do things like invoke new sessions to perform outbound dialing. A ``pool'' of available processes are maintained for TGI gateways so that it can be treated as a restricted resource, rather than creating a gateway for each concurrent call session. It is assumed gateway execution time represents a small percentage of total call time, so it is efficient to maintain a small process pool always available for quick TGI startup and desirable to prevent stampeding if say all the callers hit a TGI at the exact same moment.

Bayonne Architecture

As can be seen, we bring all these elements we described earlier together into a GNU Bayonne server, which then executes as a single core image. The server itself exports a series of base classes which are then derived in plugins. In this way, the core server itself acts as a ``library'' as well as a system image. One advantage of this scheme is that, unlike a true library, the loaded modules and core server do not need to be relocatable, since only one instance is instantiated in a specific form that is not shared over arbitrary processes. When the server comes up, it creates gateways and loads plugins. Theplugins themselves use base classes found in the server and derived objects that are defined for static storage. This means when the plugin object is mapped thru dload, it's constructor is immediately executed, and the object's base class found in the server image registers the object with the rest of GNU Bayonne. Using this method, plugins in effect automatically register themselves thru the server as they are loaded, rather than thru a separate runtime operation. The server itself also instantiates some objects at startup even before main() runs. These are typically objects related to plugin registration or parsing of the config file.

Since GNU Bayonne has to interact with telephone users over the public telephone network or private branch exchange, there must be hardware used to interconnect GNU Bayonne to the telephone network. There are many vendors that supply this kind of hardware and often as PC add-on cards. Some of these cards are single line telephony devices such as the Quicknet LineJack card, and others might support multiple T1 spans. Some of these cards have extensive on-board DSP resources and TDM busses to allow interconnection and switching. GNU Bayonne tries to abstract the hardware as much as possible and supports a very broad range of hardware already. GNU Bayonne offers support for /dev/phone Linux kernel telephony cards such as the Quicknet LineJack, for multiport analog DSP cards from VoiceTronix and Dialogic, and digital telephony cards including CAPI 2.0 (CAPI4Linux) compliant cards, and digital span cards from Intel/Dialogic and Aculab. We are always looking to broaden this range of card support. At present both voice modem and OpenH323 support is being worked on. Voice modem support will allow one to use generic low cost voice modems as a GNU Bayonne telephony resource. The openh323 driver will actually require no hardware but will enable GNU Bayonne to be used as an application server for telephone networks and softswitch equipment built around the h323 protocol family. At the time of this writing I am not sure if either or both of these will be completed in time for the 1.0 release or will wait until 1.1.

GNU Bayonne and XML Scripting

Some people have chosen to create telephony services thru web scripting, which is an admirable ambition. To do this, several XML dialects have been created, but the idea is essentially the same. A query is made, typically to a web server, which then does some local processing and spits back a well formed XML document, which can then be used as a script to interact with the telephone user. These make use of XML to generate application logic and control much like a scripting language, and, perhaps, is an inappropriate use of XML, which really is designed for document presentation and inter- exchange rather than as a scripting tool. However, given the popularity of creating services in this manner, we do support them in GNU Bayonne.

GNU Bayonne did not choose to be designed with a single or specific XML dialect in mind, and as such it uses a plugin. The design is implemented by dynamically transcoding an XML document that has been fetched into the internal ccScript virtual machine instructions, and we then execute the transcoded script as if it were a native ccScript application. This allows us to transcode different XML dialects and run them on GNU Bayonne, or even support multiple dialects at once.

Since we now learn that several companies are trying to force thru XML voice browsing standards which they have patent claims in, it seems fortunate that we neither depend on XML scripting nor are restricted to a specific dialect at this time. My main concern is if the W3C will standardize voice browsing itself only to later find out that the very process of presenting a document in XML encoded scripting to a telephone user may turn out to have a submarine patent, rather than just the specific attempts to patent parts of the existing W3C voice browsing standard efforts. \\

Going forward

Current plans are to distribute the 1.0 "rc1" release of GNU Bayonne during the opening of this year's Libre Software Meeting in Bordeaux, France, July 9. This release represents several years of active development. We are doing some final work on standardizing how Bayonne operates and how it is deployed. GNU Bayonne does not exist alone but is part of a larger meta-project, ``GNUCOMM". The goals of GNUCOMM is to provide telephony services for both current and next generation telephone networks using freely licensed software. These services could be defined as services that interact with desktop users such as address books that can dial phones and softphone applications, services for telephone switching such as the IPSwitch GNU softswitch project and GNU oSIP proxy registrar, services for gateways between current and next generation telephone networks such as troll and proxies between firewalled telephone networks such as Ogre, realtime database transaction systems like preViking Infotel and BayonneDB, and voice application services such as those delivered thru GNU Bayonne. While much work is being done to finalize the current implementation of GNU Bayonne, work is also moving forward in defining what GNU Bayonne will become. We envision several key areas for future development; native switching, web services, IP voice services, and carrier grade linux support. \\

In native switching, I do mean where Bayonne alone acts as the primary office telephone switch on it's own. One would then plug office telephones directly into a Bayonne server for internal dialtone, and outside lines for interconnection with the public telephone network. The functionality of a Bayonne switch, as well as the features offered to station users and the operation and behavior of the switch as a whole would be completely programmable thru scripting. To support the Bayonne small office phone switch (PABX), we need to support hardware that offers both POTS and PSTN termination. We are initially looking at the new 12 port Voicetronix card, as they will offer such capabilities, and do so on a card with a driver under the GPL. We are open to working with other products as well for this purpose.

For web services, we already support the ability to have Bayonne parse XML content from foreign servers. We are looking to make it possible for other web services to also invoke Bayonne services and functionality directly, such as thru XMLRPC.

In IP voice, conversations are carried as packets directly over ethernet rather than thru circuit switched telephone lines. For IP voice, we are looking to using both the new openh323 driver development and, in particular the openh323 related OPAL library. This library would allow us to use either H323 or SIP for end-point control and allow GNU Bayonne to integrate into any IP voice environment today. When used for IP voice, Bayonne requires no physical hardware as it operates purely over a network stack, and hence, some of the more obscure ports, such as the recent s/360 port of GNU Bayonne, start actually having some value.

The carrier grade linux project offers the ability to support hotswap hardware dynamically on a cPCI bus. This would allow telephony cards to be swapped in and out on a live machine. It would also allow high speed switchovers where one can have active and standby cpu's each running a linux kernel instance. Finally, it looks to address start and stop time of the kernel to make it very quick. All of these have the goal of achieving carrier grade systems that offer 6'9's reliability, or, a maximum of 2-5 seconds of downtime per year.

Since GNU Bayonne is already being deployed by commercial telephone carriers, we are interested in this. There have also been some experimenting with GNU Bayonne as a tandum switch in a public telephone network setting. These applications typically expect and require such level of reliability. Support for carrier grade linux from the application server is actually easy to achieve. Assuming we can receive notification of cards being inserted and removed, all we need to do is activate or deactivate running card resources. This type of operation is very easy to resolve in the Bayonne driver abstraction environment. Our next immediate goal after the 1.0 release is development in this area, and we hope some if not all of these things can be achieved by the 1.1 release.

Acknowledgments

There are a number of contributors to GNU Bayonne. These include:

Matthias Ivers who has provided a lot of good bug fixes and new scheduler code.

Matt Benjamin has provided a new and improved tgi tokeniser and worked on Pika outbound dialing code. Wilane Ousmane helped with the French phrasebook rulesets and French language audio prompts.

Henry Molina helped with the Spanish phrasebook rulesets and Spanish language audio prompts.

Kai Germanschewski wrote the CAPI 2.0 driver for GNU Bayonne.

David Kerry contributed the entire Aculab driver tree. Mark Lipscombe worked extensivily on the Dialogic driver tree.

There have been many additional people who have contributed to and participated in related projects like GNU Common C++ or who have helped in other ways.

WHAT'S IN THE NAME GNU/LINUX?

by Michael L. Richardson

I am aware that those of you who have been around the computer community for a while get tired of repeating and hearing the same thing again and again. But take the time to listen and the explain again and again to those who don't know (do not just say RTFM).

Knowledge is the only thing that we can pass on. It is lack of knowledge that got the DMCA passed. It is lack of knowledge of something better that gets people to buy Microsoft. It is lack of knowledge that gets me to ask or say things that annoy you. (When I ask a question let me phase the whole question, then answer the question).

Passing on what one has learned is one of the scared purposes of life.

You ask what has this got to do with a name? EVERYTHING. We give names power, and what we call a thing gives it it's power (not what the thing really is). We use names as symbols of what a thing is even if it no longer is that thing (eg. Brooklyn Dodgers, it's a Doozy, Bonhomme Richard, they no longer exist but the name invokes something in each who hears it).

And you have the Brand names used instead of the product (Xerox, Kleenex). How many know the name of the first person to transmit and receive radio signals? Most who read this will but, most of those who I went though the NYC School System with would not normally know, the Marconi device (this none of my teachers or the text books I read mention. I found out about it by watching WW2 movies and looking it up). Or any of the other products not named for their creator, founder or writer?

Unfortunately the symbol we know, see, hear, say and write becomes more important than the reality. Without the proper name and background being passed on the why, what and the struggle that went before is forgotten, so too it is with those who should be remembered. So what is the name GNU/LINUX?.

It is a name of history, of dreams, of choice and openmindedness.

During the years I have heard of the GNU Project and the Free Software Foundation. But I really had no idea of what they had to do with Linux. It was constantly stated that Linux was the Kernel of the operating system. I gave no thought to were all the programs that came with Linux came from. Even when GNU or Free SoftWare was talked about I still did not put 2+2 together.

I became aware of GNU/LINUX as a name when I saw the movie Revolution OS (everyone interested in GNU/LINUX would do well to see it).

Latter at an NYLXS Inservice I heard Richard Stallmen n give a brief expiation as to why the name should be GNU/LINUX. Someone told me it was one of his "rants". Rant or not it was something I had not heard before. But I heard enough to go look at the GNU website. As I read I began to put 2 + 2 together. I realized where all those programs came from.

When the GNU Project and the LINUX Kernel merged both brought to the table what the other needed. I was not there and do not know how or why the result was called Linux. The GNU Project needed a kernel (as I understand it they were working on one) Linus Trovalds had a kernel and was looking for programs to run on it. The two met and both saw the possibly of the dream coming to fruition. Both were openminded enough to say lets put these together. The community benefited by the collaboration. It is in the nature of the collaboration that the name GNU/LINUX should have used one for all the work that went before, the other for the practical working of the dream. GNU/LINUX tells us the history of FreeSoftware, the dreams of GNU and LINUS, the choice of GNU to be openminded and live the free software philosophy by collaboration with Linus [they could have said no we are working on our own kernel]. It tells of Linus openmindeness by his choice to collaborate with the GNU Project [he could have said to those that helped him, we now need programs for our kernel to run, or gone to a proprietary system]. Instead both chosen to fulfill the dream.

Is it not fitting that the name of a thing reflect what it truly is?

MEMBERSHIP COMMITTEE REPORT

by Michael L. Richardson

Welcome, my name is Michael :L. Richardson and I am the new membership committee chairperson. For those of you who don't know NYLXS is in the middle of a membership drive. We are asking in the next 3 three months that all members bring at lease one person to a NYLXS event that has not been to one before.

The membership committee is in the process of organizing. Our main goals are to submit membership requirements to the board. Keep tract of members hours. Inform members who need hours which committees need them most. Promote NYLXS membership. I can be reached at miker@nylxs.com or hangout@nylxs.com

NYLXS Committee Heads

Free Software for Schools: (Marcus Conti)
http://www2.mrbrklyn.com/appl/nylinux/mail/bysubject.html?idno=1534

Journal Committee : (Michael Richardson)
http://www2.mrbrklyn.com/appl/nylinux/mail/bysubject.html?idno=1381

Public Relations : (Joe Grastrata)
http://www2.mrbrklyn.com/appl/nylinux/mail/bysubject.html?idno=1416

FreeSoftware Institute: (Paul Rodriguez)
http://www2.mrbrklyn.com/appl/nylinux/mail/bysubject.html?idno=1429

Membership Committee: (Michael L. Richardson) e-mail miker@nylxs.com

Finance Committee: (Joe Maffia)
http://www2.mrbrklyn.com/appl/nylinux/mail/bysubject.html?idno=1429

And our fearless leader Ruben Safir who needs no

introduction as you can just:
http://www.google.com/search?hl=en&q=Ruben+I+Safir&btnG=

NYC Free Software Computer Groups

by Jon Bober

The New York Linux Scene is a new organization in New York and it comes into a crowded city. NYLXS is tied to no one Free Software group in New York City, but it has ties to all of them. At many Free Software events members of many different groups appear, and many member of NYLXS are members of other Free Software groups as well.

Thus, it does not matter for example, whether a NYLUG or DeLUGE is sponsoring an installfest, or NYLXS is throwing an instalfeast. Members of any organization may be found at any event.

More importantly, there are oftentimes when no one group is big enough to handle an issue, and more are always better. In August of 2001, when Dmitry Sklyarov was imprisoned, members of NYLUG, LXNY, and NYFairuse were quickly mobilized. Personally, I became involved through subscribing to the Free Dmitry mailing list, and soon moved over to NYLUG and NYFairuse for New York specific information. The size of NYLUG made it easy to find and an obvious starting point for many who did not know where to turn, and it allowed a smaller, more local organization such as NYFairuse to mobilize better than it otherwise would have. Now, much DMCA-specific (along with other political issues, of course) discussion and action takes place as a result of collaborative efforts among LXNY, NYLXS, NYFairuse, and others.

Some of the Free Software groups in New York City include:

NYLUG is New York's traditional LUG. It holds monthly meetings, always followed by Stammtisch. From www.nylug.org:

?Linuxstammtische - are social gatherings of local Linux, FREEBSD, and other open-source software users at a local brewpub, founded in connection with the first annual Linuxbierwanderung (Linux Beer Hike, August 1999 in Bavaria, August 2000 in England's Lake District). What Is It: Beer (naturlich), good fellowship, and an occasional lesson in the German language. Translated to English, Stammtisch means 'table for regulars.' Consequently, you can generally find nyluggers near the back of The Old Stand (914 3rd Ave. on the corner of 55th Street) after every monthly meeting.

NYLUG meetings generally place at 6:30 PM on the 3rd Wednesday of every month at the IBM headquarters in Midtown and feature guest speakers focusing on Linux. Recent topics/speakers have included Terry Schmit on Wireless Public Networking, Simon Lok on email privacy and key signing (NYLUG also holds GPG/PGP keysignings quite often,) and, in the wake of last year's terrorist attacks, a talk on ?Decentralism Against Terrorism?, by Eric S. Raymond. Also, NYLUG recently held their annual holiday party sponsored by and with a special presentation by AMD.

LXNY is ?New York's Free Software Organization.? Less traditional than NYLUG, and less Linux-centric, LXNY is for all who have an interest in Free Software, regardless of interest in Linux. Formal meetings often feature speakers, but not necessarily. Recent important topics of interest at meetings have been Dmitry Sklyarov and the DMCA and LXNY's planned upcoming Microsoft Refund Day. Also, LXNY supports GNUbies: The LXNY GNU/Linux/Free OS Beginners Group.

GNUbies is an group for people who are new to Free Software. Mostly Linux based, monthly meetings are address topics in Free Software. Unlike NYLUG or some other groups, which often assume a certain amount of technical knowledge, GNUbies assumes none. Topics are generally geared towards the use and installation of Free Software, and are presented for the newbie who knows nothing. Additionally, meetings can be of interest to more experienced users as well. Meetings are generally on the second Wednesday of every month at the IBM building in midtown.

DeLUGE is the Desktop Linux Users Group for Endusers. The group is meant to focus on the use of desktop Linux and to help people to use Linux for all of the same desktop uses of other Operating Systems. Focus is on applications and enduser software ? from office suites to games. The last meeting was on March 6th and featured a presentation on various email clients running under Linux, including Kmail, Pine, Mutt, and Evolution. DeLUGE works for Linux not for the technical guru but for Everyman.

There are many other groups in New York and a good description of them all cannot be contained here. Future journal articles will feature more descriptions of other Free Software groups and will let the reader know what is going on in the Free Software world in New York City. f you wish to have your group listed here send a outline of your group to miker@nylxs.com