Installing and Testing a Server Operating System

Lorentz JÄNTSCHI

Technical University of Cluj-Napoca, Romania, http://lori.academicdirect.ro

Abstract

The paper is based on the experience of the author with the FreeBSD server operating system administration on three servers in use under academicdirect.ro domain.

The paper describes a set of installation, preparation, and administration aspects of a FreeBSD server.

First issue of the paper is the installation procedure of FreeBSD operating system on i386 computer architecture. Discussed problems are boot disks preparation and using, hard disk partitioning and operating system installation using a existent network topology and a internet connection.

Second issue is the optimization procedure of operating system, server services installation, and configuration. Discussed problems are kernel and services configuration, system and services optimization.

The third issue is about client-server applications. Using operating system utilities calls we present an original application, which allows displaying the system information in a friendly web interface. An original program designed for molecular structure analysis was adapted for systems performance comparisons and it serves for a discussion of Pentium, Pentium II and Pentium III processors computation speed.

The last issue of the paper discusses the installation and configuration aspects of dial-in service on a UNIX-based operating system. The discussion includes serial ports, *ppp* and *pppd* services configuration, *ppp* and *tun* devices using.

Keywords

Server operating systems, Operating system configuration, Server services, Client-server applications, Dial-in server, System testing

Introduction

UNIX is an interactive time-sharing operating system invented in 1969 by Ken Thompson after Bell Labs left the Multics project, originally so he could play games on his scavenged PDP-7.

The time-sharing is an operating system feature allowing several users to run several tasks concurrently on one processor, or in parallel on many processors, usually providing each user with his own terminal for input and output; time-sharing is multitasking for multiple users.

Dennis Ritchie, the inventor of {C}, is considered a co-author of the UNIX system. The turning point in UNIX's history came when it was reimplemented almost entirely in C during 1972 - 1974, making it the first source-portable OS. UNIX subsequently underwent mutations and expansions at the hands of many different people, resulting in a uniquely flexible and developer-friendly environment.

By 1991, UNIX had become the most widely used multi-user general-purpose operating system in the world.

Many people consider this the most important victory yet of hackerdom over industry opposition. Another point of view expresses a "UNIX conspiracy". According to a conspiracy theory long popular among ITS and TOPS-20 fans, UNIX's growth is the result of a plot, hatched during the 1970s at Bell Labs, whose intent was to hobble AT&T's competitors by making them dependent upon a system whose future evolution was to be under AT&T's control. This would be accomplished by disseminating an operating system that is apparently inexpensive and easily portable, but also relatively unreliable and insecure (to require continuing upgrades from AT&T). In this view, UNIX was designed to be one of the first computer viruses (see virus) - but a virus spread to computers indirectly by people and market forces, rather than directly through disks and networks.

Adherents of this "UNIX virus" theory like to cite the fact that the well-known quotation "UNIX is snake oil" was uttered by DEC president Kenneth Olsen shortly before

DEC began actively promoting its own family of UNIX workstations. UNIX is now offered by many manufacturers and is the subject of an international standardization effort. Unix-like operating systems include Debian, Linux and LinwowsOS, AIX, GNU, HP-UX, OSF and Solaris, BSD/OS, NetBSD, OpenBSD and FreeBSD (with TrustedBSD and PicoBSD project variations) [1].

FreeBSD (FreeBSD is a registered trademark of Wind River Systems, Inc. and this is expected to change soon) is an advanced operating system for x86 compatible, AMD64, Alpha, IA-64, PC-98 and UltraSPARC architectures. It is derived from BSD/OS (BSD is a registered trademark of Berkeley Software Design, Inc.), the version of UNIX (UNIX is a registered trademark of The Open Group) developed at the University of California, Berkeley. The FreeBSD operating system is developed and maintained by a large team of individuals. While you might expect an operating system with these features to sell for a high price, FreeBSD is available free of charge and comes with full source code.

NetBSD's focus lies in providing a stable, multiplatform, and research oriented operating system. NetBSD's portability leads it to run on 33 platforms as of January 2001. Even more impressive is the list of hardware including traditional modern server equipment like standard Intel-based PCs, Compaq's Alpha, or Sun Microsystem's SPARC architectures. Older server and workstation class hardware like the Digital Equipment Corporation's VAX hardware, Apple's Macintosh computers based on Motorola's 68000 processor series are also support. But what really sets NetBSD apart is its support for more exotic hardware including Sega's Dreamcast, Cobalt Network's server appliances, and George Scolaro's and Dave Rand's PC532 hobbyist computer. NetBSD's dedication to portability has led the way for other operating systems. When the FreeBSD group began porting to the Alpha platform, the initial work from the NetBSD project provided the foundation. With new FreeBSD ports to both the PowerPC and SPARC platforms under way, work from NetBSD is being used again. Linux has benefited from NetBSD's experience as well. The special booter used by NetBSD on the 68000-series Macintosh computers was modified and became the Penguin booter used to launch Linux on these systems. Finally, NetBSD's largest contribution to other systems lies in acting as a springboard for the OpenBSD operating system.

OpenBSD diverged from NetBSD around the release of NetBSD 1.1 in November of 1995. OpenBSD's first release came a year later when OpenBSD 2.0 was released in October of 1996. OpenBSD quickly began focusing on producing the most secure operating system

available. Taking advantage of his Canadian residency, de Raadt realized United States munitions export laws did not hamper him, allowing inclusion of strong cryptography including RSA, Blowfish, and other advanced algorithms. A modified version of the Blowfish algorithm is now in use for encrypting user passwords by default. OpenBSD developers also spear-headed the development of OpenSSH, a multiplatform clone of the wildly popular protocol for secures communications. OpenBSD also advanced the state of code auditing. Beginning in 1996, the OpenBSD team began a line-by-line analysis of the entire operating system searching for security holes and potential bugs. UNIX systems have been plagued for decades by the use of fixed-sized buffers. Besides being inconvenient for the programmer, they have lead to numerous security holes like the fingerd exploit in 4.2BSD. Other security holes relating to mishandling temporary files are easily caught. OpenBSD's groundbreaking audit has also discovered security-related bugs in related operating systems including FreeBSD, NetBSD, and mainstream System V derivatives. However, the nature of this process allows general coding mistakes not relating to security to be caught and corrected, as well. Additionally, a number of bugs in Ports, or third party applications have been discovered through this process.

Referring to FreeBSD, perhaps what sets FreeBSD apart most is its technical simplicity. The FreeBSD installation program is widely regarded as the simplest UNIX installation tool in existence. Further, its third party software system, the Ports Collection, has been modeled by NetBSD and OpenBSD and remains the most powerful application installation tool available. Through simple one-line commands, entire applications are downloaded, integrity checked, built, and installed making system administration amazingly simple.

The most important feature of a server system is system services. Most of the services in a server system are provided through a program or process that sits idly in the background until it is invoked to perform its task, called daemons [2]. The daemon word come from the mythological meaning, later rationalized as the acronym "Disk And Execution MONitor" [3]. A daemon is program that is not invoked explicitly, but lays dormant waiting for some condition(s) to occur. The idea is that the perpetrator of the condition need not be aware that a daemon is lurking (though often a program will commit an action only because it knows that it will implicitly invoke a daemon). For example, under ITS writing a file on the LPT spooler's directory would invoke the spooling daemon, which would then print the file. The

advantage is that programs wanting files printed need neither compete for access to, nor understand any idiosyncrasies of, the LPT. They simply enter their implicit requests and let the daemon decide what to do with them. Daemons are usually spawned automatically by the system, and may either live forever or be regenerated at intervals.

UNIX systems run many daemons, chiefly to handle requests for services from other hosts on a network. Most of these are now started as required by a single real daemon, *inetd*, rather than running continuously. Examples are *cron* (local *timed* command execution), *rshd* (remote command execution), *rlogind* and *telnetd* (remote login), *ftpd*, *nfsd* (file transfer), *lpd* (printing) [4].

The discussed services are Internet domain name server (*named*, [5?query=named]), Internet super-server (*inetd*, [5?query=inetd]), OpenSSH SSH daemon (sshd, [5?query=sshd]), Internet file transfer protocol server (*ftpd*, [5?query=ftpd]), Apache hypertext transfer protocol server (*httpd*, [5?query=httpd]), proxy caching server (*squid*, [5?query=squid), the MySQL server demon (*mysqld*, [5?query=mysqld]) and PHP sub-service (post processed hypertext [6]).

Many system information applications are available. Looking at FreeBSD ports, a good application is phpSysInfo [7]. The problem that appears is that, not always, the system information application makes the updates and a set of problems can appear, especially for CURRENT systems [8].

In order to create a web application for system information, at least our system must have a web server installed. If we propose to test the performances of computer architecture, first we must look at the hardware specifications. Because the application requires using of the system functions, a good idea is to use software with Perl support. Many reasons lead to the PHP language for the applications implementation [9].

Two methods of dialing into a machine to get access to the Internet are widely used. If you dial in and log on as usual (on UNIX you see "login:" and shell prompt or on MPE you type "HELLO" and get a colon prompt), your computer is not directly connected to the Internet, so it cannot send network packets from your PC to the Internet. In this case, you will have to use Lynx to access the WWW. If you dial-in using SLIP (Serial Line IP) or PPP (Point-to-Point Protocol), your computer becomes part of the Internet, which means it can send network packets to and from the Internet. In this case, you can use graphical browsers

like Mosaic or Netscape to access the WWW. The Internet Adapter is supposed to allow users with only shell account access to obtain a SLIP connection [10].

To install and configure a digital modem [11] is quite different from analog modems [12]. Until the digital phone lines take the place of analog ones, the analog modems will stay on base of the most dial-up connections. The present paper is focused on analog modems software configuration to work as a dial-in server.

A FreeBSD 5.2-CURRENT operating system is the support for the dial-in server service. From the start, an appreciation is necessary: the current modems standards are V.30, V.34, V.90 and V.92. Considering the most advanced standard, through an analog phone line we can get at most 4.8 Kb/s as rate of compressed data upload transfer. The dial-in server installation requires, in most of the cases, system configurations and in some cases, kernel recompilations, to support the modem. To know which modem is supported on the system, a good idea is to consult the Release Notes of the system. To avoid any compatibility problems, a good idea is to chouse an external modem.

Operating System Installing Procedure

First step in FreeBSD operating system installation is to create a boot disk set, depending on machine type. If we are using a Personal Computer, based on i386 computer architecture, a disk boot set can be found at:

ftp://ftp.freebsd.org/pub/FreeBSD/releases/i386/5.2-RELEASE/floppies/

For 1.44Mb floppies, all that we have to do is to download at least *kern.flp* and *mfsroot.flp* files. If the planned computer to be a server has exotic or old components, is possible to need also the *drivers.flp* file.

If we use a DOS/Windows operating system type, to create the boot disks is necessary to download and use an image file installation program, which can be found at the address:

ftp://ftp.freebsd.org/pub/FreeBSD/tools/

We can use any of *fdimage.exe* or *rawrite.exe* to create the disks. For *fdimage.exe* the commands (DOS commands) are (assuming that we use a: drive):

fdimage –f 1.44M kern.flp a: (and similarly for *mfsroot.flp* and *drivers.flp* files)

If we use a UNIX operating system type, we can use dd program for disks creation:

dd if=kern.flp of=/dev/floppy (and similarly for *mfsroot.flp* and *drivers.flp* files)

After the boot disks creation, we must boot from "*kern.flp*" floppy and "*mfsroot.flp*" floppy the FreeBSD operating system. Kernel and SysInstall utility are automatically loaded and after that, we have two consoles (alt+F1 and alt+F2 respectively).

The second console is for *DEBUG* messages. In the *DEBUG* console we can watch how modules are loaded. In this moment, a good idea is to look at the *DEBUG* console to assure that our network card is proper identified and used. In our installation procedure, we use a 3COM network card (3c905B) and the DEBUG messages are:

DEBUG: loading module if_xl.ko

xl0: <3Com 3c905B-TX Fast Etherlink XL> port 0x1000-0x107f mem 0x40400000-0x4040007f irq 17 at device 9.0 on pci2

xl0: Ethernet address: 00:04:76:9d:2e:02

miibus0: <MII bus> on xl0

xlphy0: <3Com internal media interface> on miibus0

xlphy0: 10baseT, 10baseT-FDX, 100baseTX, 100baseTX-FDX, auto

Until this step, only CD file system, SYSV messages queue, shared memory and semaphores, serial and network modules are loaded. If is necessary, at this point we have the option to load a specific module from *drivers.flp* floppy.

At this point, *SysInstall* utility load *FDISK partion editor* and we must create a FreeBSD partition using a set of commands:

A – Use entirely disk

C – Create slice

S – set bootable

Q – finish

F – Dangerously dedicated mode (purposely undocumented)

If we chouse to use entirely disk, the choice is simple (A, S, Q). After *FDISK partion editor*, we can chouse to select a boot manager utility (from three possibilities):

Install FreeBSD Boot Manager – FreeBSD system will select the booted operating system;

Install Standard Boot Manager (MBR) – disable other operating system boot managers;

Leave Master Boot untouched – if we want that other existing boot manager to manage booting.

An observation is useful: for PC-DOS users the last option allow to exist both operating systems on same machine.

After *Boot Manager on disk* selection, *FreeBSD Disklabel editor* are invoked and we must specify at least two partitions, one for root mount point ("/") and one for swap ("swap"). A good idea is to specify the swap size at least 2*memory.

The *SysInstall* utility let us now to select the installation options such as preinstalled binaries, services and documentation.

If we plan to update and/or upgrade our system after the installation, a good idea is to select a minimal configuration.

The installation of operating system continues with installation media selection. If our computer is connected to the *Internet* via a high-speed communication line, a good idea is to install the system directly from internet. Anyway, now we have the options to install from a CD, from a DOS partition, over Network File Server, existing file system, floppy disk set, SCSI or QIC tape, and from a FTP server. If we chouse the installation from a FTP server, we must know the network topology to do this. Three possibilities are there: *FTP* (install from an FTP server), *FTP Passive* (Install from an FTP server through a firewall) or *HTTP* (Install from an FTP server through an http proxy).

Anyway, the host configuration is necessary and the configuration will be made for used communication device (in our case xl0). A selection from three protocols it must be done (*IPv6*, *IPv4* or *DHCP*) depending on network topology. In our case, *IPv4* is the proper choice. The *host* (*vl*), the *domain* (*academicdirect.ro*), the *IPv4 gateway* (*193.226.7.211*), *name server* (*193.226.7.211*), *IPv4 address* (*193.226.7.200*) and net mask (255.255.255.0) must be specified. If we chouse to install via a proxy server, we also must specify the proxy address (*193.226.7.211*) and port (*3128*).

After the communication interface configuration, the SysInstall utility load all required modules (see *DEBUG* console) and make the internet connection for installation. At the end of system installation, we have the option to preinstall a set of packages in our system. Anyway, the option can be ignored, since the *SysInstall* utility is also preinstalled in our system and can be invoked anytime after reboot. Some final configurations can be specified at the end of the installation (such as *boot services*, *root password*, *group* and *user* management). Only *root password* is obligatory (root are the *superuser* in FreeBSD system). After reboot, we have a FreeBSD system on our machine.

Operating System Configuration

After the system installation, we can configure it. Many configurations can be done. We can start to download now all system sources. A utility called *cvsup* can be used for this task. *CVSup* is a software package for distributing and updating source trees from a master CVS repository on a remote server host. The FreeBSD sources are maintained in a CVS repository on a central development machine in California. With CVSup, FreeBSD users can easily keep their own source trees up to date. Using *SysInstall* utility, we can fetch the *cvsup* program in the same way as we installed the system, from internet via FTP protocol (*sysinstall/Configure/Packages/…logging… /devel/cvsup-without-gui-16.1h*). After the *cvsup* installation, a *configuration file* (let us call it *configuration_file)* must be created (or edited from /usr/share/examples/cvsup/) and must contain the *host* (this specifies the server host which will supply the file updates), the *base* (this specifies the root where CVSup will store information about the collections you have transferred to our system), the prefix (this specifies where to place the requested files), and the desired release (version). Other options are also benefit:

*default host=cvsup.FreeBSD.org

*default base=/usr

*default prefix=/usr

*default release=cvs

*default delete use-rel-suffix

*default compress

src-all tag=.

ports-all tag=.

doc-all tag=.

cvsroot-all tag=.

Sources can be fetched separately (such as *src-base*) or entirely (such as *src-all*). Tag option is used to fetch one specific version of the sources (when "." means CURRENT versions). In addition, the date option can be used (as example: src-all tag=RELENG_4 date=2000.08.27.10.00.00). Fetching procedure can be done now from a text console, using a simple command: *cvsup -g -L 2 configuration_file* or from a graphical console (X-based) using the command: *cvsup configuration_file.*

Recompilation and System Optimization

The kernel is the core of the FreeBSD operating system. It is responsible for managing memory, enforcing security controls, networking, disk access, and much more. While more and more of FreeBSD become dynamically configurable, it is still occasionally necessary to reconfigure and recompile the kernel. Building a custom kernel is one of the most important rites of passage nearly every UNIX user must endure. This process, while time consuming, will provide many benefits to your FreeBSD system. Unlike the *GENERIC* kernel, preinstalled in our system, which must support a wide range of hardware, a custom kernel only contains support for your PC's hardware. This has a number of benefits, such as:

- faster boot time (since the kernel will only probe the hardware you have on our system, the time it takes your system to boot will decrease dramatically);

- less memory usage (a custom kernel often uses less memory than the *GENERIC* kernel, which is important because the kernel must always be present in real memory; for this reason, a custom kernel is especially useful on a system with a small amount of *RAM*);

- additional hardware support; a custom kernel allows you to add in support for devices such as sound cards, which are not present in the *GENERIC* kernel.

If we follow the acquiring procedure of the sources exactly, we can found for the kernel configuration a set of predefined configuration files at the location: */usr/src/sys/i386/conf/*

If the sources version fit with our system then the *GENERIC* file using must produce same kernel and modules with the existent ones. The idea is to optimize the kernel at compilation time. The kernel can be configured in a configuration file using the prescriptions that can be found in following files: *GENERIC*, *Makefile*, *NOTES* (/usr/src/sys/i386/conf/), *NOTES* from /usr/src/sys/conf/ and *README* and *UPDATING* from /usr/src/. Additionally, we can create the *LINT* file which contain additional kernel configuration options from *NOTES* files with make utility (*cd /usr/src/sys/i386/conf/ && make LINT*). In the optimizing process of the kernel, a good idea is to look at the system characteristics detected by the *GENERIC* kernel using the *dmesg* utility.

After the reading of these files and understanding of the nature of the process, first step is to create our own kernel configuration file (*cd /usr/src/sys/i386/conf/ && cp GENERIC VL*). We can now edit this file (using as example *ee* utility) and set the CPU type (see *dmesg | grep CPU*), ident (same with file name, *VL*), debug and SYSV options (enable or disable), console

behavior (as example: *options SC_DISABLE_REBOOT; options SC_HISTORY_SIZE=2000; options SC_MOUSE_CHAR=0x3; options MAXCONS=5; options SC_TWOBUTTON_MOUSE*).

Most of the essential options are well documented and we cannot miss. Anyway, a large set of network devices can be excluded from the kernel. To find which device driver is using in the system for network adapter management we can look again at the boot messages (dmesg | grep Ethernet). Supposing that we have finished our kernel configuration, the next step is to configure-it according with the new configuration file: cd /usr/src/sys/i386/conf/ && config VL

The next three steps can be emerged in one composed command:

cd ../compile/VL && make depend && make && make install

The "&&" operator has advantage that if we miss something and a error is detected, the following commands are aborted.

Anyway, supposing that we configured, compiled and installed the kernel, but the system do not boot. This is not necessary a problem. *BootLoader* utility allows us to repair this damage. At the boot, hit any key except for the *Enter* key; the system lead us in a shell; following commands solve the problem:

unload kernel

load /boot/kernel.old/kernel

boot

If we want to restore the old configuration:

rm –fr /boot/kernel

cp –fr /boot/kernel.old/kernel /boot/kernel

To prevent that also kernel.old to be loosed in recompilation process, a good idea is to save the *GENERIC* kernel:

cp –fr /boot/kernel /boot/kernel.GENERIC

The optimization process of kernel in generally reduces the kernel size (ls –al):

-r-xr-xr-x 1 root wheel 5934584 Feb 8 02:55 /boot/kernel.GENERIC/kernel

-r-xr-xr-x 1 root wheel 3349856 Feb 14 00:10 /boot/kernel/kernel

The System Services

The kernel configuration process allowed us to define console behavior (to disable cltr+alt+del reboot sequence), to increase the amount of free memory available for processes and increase the system speed. Now can begin to install and configure the server services.

The named service. Name servers usually come in two forms: an *authoritative name server*, and a *caching name server*. An *authoritative name server* is needed when:

- one wants to serve DNS information to the world, replying authoritatively to queries;
- a domain, such as academicdirect.ro, is registered (to RNC, [13]) and IP addresses need to be assigned to hostnames under it;
- an IP address block requires reverse DNS entries (IP to hostname).
- a backup name server, called a slave, must reply to queries when the primary is down or inaccessible.

A *caching name server* is needed when:

- a local DNS server may cache and respond more quickly than querying an outside name server;
- a reduction in overall network traffic is desired (DNS traffic has been measured to account for 5% or more of total Internet traffic).

A named configuration file resides in /etc/namedb/ directory, and to start automatically at boot, the /etc/rc.conf file must contain named_enable="YES".

For a real name server, at least following lines (from /etc/namedb/named.conf file) must fit with our system (academicdirect.ro):

```
zone "academicdirect.ro" {
        type master;
        file "academicdirect.ro";
};
```

So, in *academicdirect.ro* file we must specify the zone. At least following lines must fit (see also [13]):

```
$TTL 3600
academicdirect.ro. IN SOA ns.academicdirect.ro. root.academicdirect.ro. (
        2004020902;             Serial
        3600;                   Refresh
        1800;                   Retry
        604800;                 Expire
```

```
                 86400);                      Minimum TTL
@        IN NS   ns.academicdirect.ro. ;      DNS Server
@        IN NS   hercule.utcluj.ro. ;         DNS Server
localhost IN A   127.0.0.1;                   Machine Name
ns       IN A    193.226.7.211;               Machine Name
mail     IN A    193.226.7.211;               Machine Name
@        IN A    193.226.7.211;               Machine Name
```

To properly create the local reverse DNS zone file, following command are necessary:

cd /etc/namedb && sh make-localhost.

The *inetd* service manages (start, restart, and stop) a set of services (according with Internet server configuration database /etc/inetd.conf), for both IPv4 and IPv6 protocols, such as:

ftp stream tcp4 nowait root /usr/libexec/ftpd ftpd –l # ftp IPv4 service

ftp stream tcp6 nowait root /usr/libexec/ftpd ftpd –l # ftp IPv6 service

ssh stream tcp4 nowait root /usr/sbin/sshd sshd -i -4 # ssh IPv4 service

ssh stream tcp6 nowait root /usr/sbin/sshd sshd -i -6 # ssh IPv6 service

finger stream tcp4 nowait/3/10 nobody /usr/libexec/fingerd fingerd –s # finger IPv4

finger stream tcp6 nowait/3/10 nobody /usr/libexec/fingerd fingerd –s # finger IPv6

ntalk dgram udp wait tty:tty /usr/libexec/ntalkd ntalkd # talk

pop3 stream tcp4 nowait root /usr/local/libexec/popper popper # pop3 IPv4 service

pop3 stream tcp6 nowait root /usr/local/libexec/popper popper # pop3 IPv6 service

In some cases, is possible that *ined* service do not start. A solution is manual starting of a specific service (/usr/libexec/ftpd -46Dh) or creating of an executable shell script and place-it in an rc.d directory:

-r-xr-xr-x 1 root wheel 60 Feb 12 12:34 /usr/local/etc/rc.d/ftpd.sh (ls –al)

/usr/libexec/ftpd -46Dh && echo -n 'ftpd' (ftpd.sh file content)

In other cases, may be we want to use another daemon for a specific service (such as /etc/rc.d/sshd shell script for sshd service).

The *Hypertext Transfer Protocol Server* can be provided also by many applications such as *httpd* (apache@apache.org), *bozohttpd* (Janos.Mohacsi@bsd.hu), *dhttpd* (gslin@ccca.nctu.edu.tw), *fhttpd* (ports@FreeBSD.org), *micro_httpd* (user@unknown.nu), *mini_httpd* (se@FreeBSD.org), *tclhttpd* (mi@aldan.algebra.com), *thttpd* (anders@FreeBSD.org), *w3c-httpd* (ports@FreeBSD.org), but full featured and multiplatform capable remains *httpd* from Apache

[14]. The most important feature of Apache web server is PHP language modules support, which transform our web server into a real client-server interactive application.

The FreeBSD operating system offers a strong database support with MySQL database server (very fast, multi-threaded, multi-user and robust SQL, [15], *mysqld* daemon).

Some configurations are very important for services behavior. Let us to exemplify some of the services configuration options.

For *httpd* service (/usr/local/etc/apache2/httpd.conf):

Listen 80 # httpd port

<IfModule mod_php5.c>

    AddType application/x-httpd-php .php

    AddType application/x-httpd-php-source .phps

</IfModule> # not included by the default but required to work

ServerName vl.academicdirect.ro:80

For PHP module (/usr/local/etc/php.ini):

precision = 14 ; Number of significant digits displayed in floating point numbers

expose_php = On ; PHP may expose the fact that it is installed on the server

max_execution_time = 3000 ; Maximum execution time of each script, in seconds

max_input_time = 600 ; Maximum amount of time for parsing request data

memory_limit = 128M ; Maximum amount of memory a script may consume

post_max_size = 8M ; Maximum size of POST data that PHP will accept

file_uploads = On ; Whether to allow HTTP file uploads

upload_max_filesize = 8M ; Maximum allowed size for uploaded files

display_errors = On ; For production web sites, turn this feature Off

For *squid* service (/usr/local/etc/squid/squid.conf):

http_port 3128 # The socket addresses where Squid will listen for client requests

auth_param basic children 5

auth_param basic realm Squid proxy-caching web server

auth_param basic credentialsttl 2 hours

read_timeout 60 minutes # The read_timeout is applied on server-side connections

acl all src 0.0.0.0/0.0.0.0

acl localhost src 127.0.0.1/255.255.255.255

acl to_localhost dst 127.0.0.0/8

acl SSL_ports port 443 563

acl Safe_ports port 80 # http

acl Safe_ports port 21 # ftp

acl Safe_ports port 1025-65535 # unregistered ports

acl Safe_ports port 591 # filemaker

acl Safe_ports port 777 # multiling http

acl CONNECT method CONNECT

acl network src 172.27.211.1 172.27.211.2 193.226.7.200 192.168.211.2

http_access allow network

acl ppp src 192.168.211.0/24

http_access allow ppp

http_access deny all

PHP Language Capabilities

The PHP language has a rich strong functions library, which can significantly shorten the algorithm design and implementation. In the following, some of them (already tested ones) are presented, using sequences of our first program for system information:

$b = preg_split("/[\n]/",$a,-1,PREG_SPLIT_NO_EMPTY); // split string into an array using a perl-style regular expression as a delimiter

$c = explode(" ",$b[$i]); // splits a string on string separator and return array of components

$c = str_replace("<","(",$c); //replaces all occurrences of first from last with second

$a=`uptime`; //PHP supports one execution operator: backticks (``)

A set of shell execution applications are used with execution operator:

$a=`cat /etc/fstab | grep swap`; // swap information

$a=`uptime`; // show how long system has been running

$a =`top -d2 -u -t -I -b`; // display information about the top CPU processes

$a =`dmesg`; // display the system message buffer

$a=`netstat -m`; // show network status

$aa =`df -ghi`; // display free disk space

$s=`netstat -i`; // show network status

```
$s=`pkg_info`; // a utility for displaying information on software packages
```

The second application (hin.php file) is of client-server architecture and uses a class structure to define a chemical molecule:

```
define("l_max_cycle",16);
class m_c{
        var $a;//number of atoms
        var $b;//number of bonds
        var $c;//molecule structure
        var $e;//seed
        var $f;//forcefield
        var $m;//molecule number
        var $n;//file name
        var $s;//sys
        var $t;//file type
        var $v;//view
        var $y;//cycles structure
        var $z;//file size
        function m_c(){
                $this->a=0;
                $this->b=0;
                for($i=0;$i<l_max_cycle+1;$i++)
                        $this->y[$i][0]=0;
        }
}
$m = new m_c;
$m->n = $_FILES['file']['name'];
$m->z = $_FILES['file']['size'];
$m->t = $_FILES['file']['type'];
$file = "";
$fp = fopen($_FILES['file']['tmp_name'], "rb");
while(!feof($fp)) $file .= fread($fp, 1024);
fclose($fp);
```

```
unset($fp);
```

The molecule is uploaded from a file to the server and processed; the program computes all the cycles with maximum length defined by l_max_cycle constant.

The procedure of cycles finding is recursive one:

```
function recv(&$tvv,&$t_v,$pz,&$mol){
        $ciclu=0;
        for($i=0;$i<$t_v[$tvv[$pz]][1];$i++){
                if(este_v($t_v[$tvv[$pz]][$i+2],$tvv,$pz)==1){
                        $tvv[$pz+1]=$t_v[$tvv[$pz]][$i+2];
                        if($pz<l_max_cycle-1)recv($tvv,$t_v,$pz+1,$mol);
                }
                if($t_v[$tvv[$pz]][$i+2]==$tvv[0])
                        $ciclu=1;
        }
        if(($ciclu==1)&&($pz>1))
                af($tvv,$pz,$mol);
}
```

Other useful PHP functions are used:

```
echo(getenv("HTTP_HOST")."\r\n"); // get the HTTP_HOST environment variable
echo(date("F j, Y, g:i a")."\r\n"); // format a local time/date
echo(microtime()."\r\n"); // the current time in seconds and microseconds
$m->f = implode(" ", $filerc[$i]); // joins array elements and return one string
```

The output data download procedure to the client is achieved via a header function:

```
header("Content-type: application/octet-stream");
header('Content-Disposition: attachment; filename="'.l_max_cycle.'".txt");
```

The program counts the time necessary to compute the all cycles in a molecule using microtime function at the beginning and at the end of the program.

The Web Based System Information Application

Both applications was putted and used on three FreeBSD 5.2-CURRENT servers (j, ns and vl under academicdirect.ro domain). For the first application, links are

http://j.academicdirect.ro/SysInfo, http://ns.academicdirect.ro/SysInfo, and http://vl.academicdirect.ro/SysInfo.

The program display information about the system type (see fig. 1), memory and CPU usage (see fig. 2), file system (see fig. 3), and network status (see fig. 4).
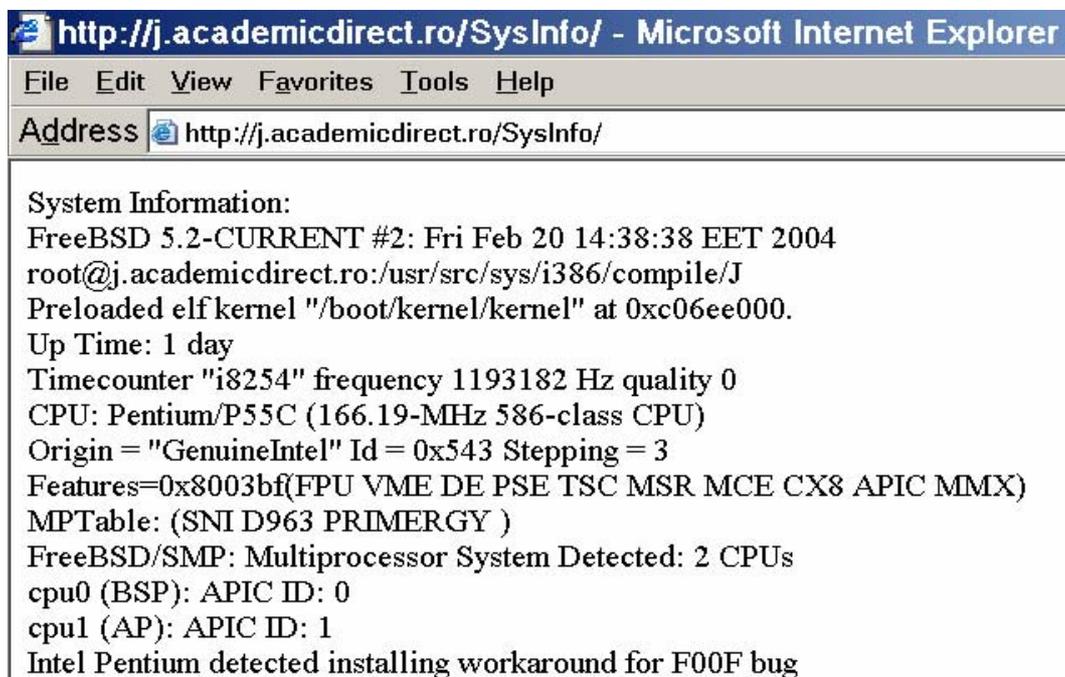
Fig. 1. System type information – picture from j.academicdirect.ro server
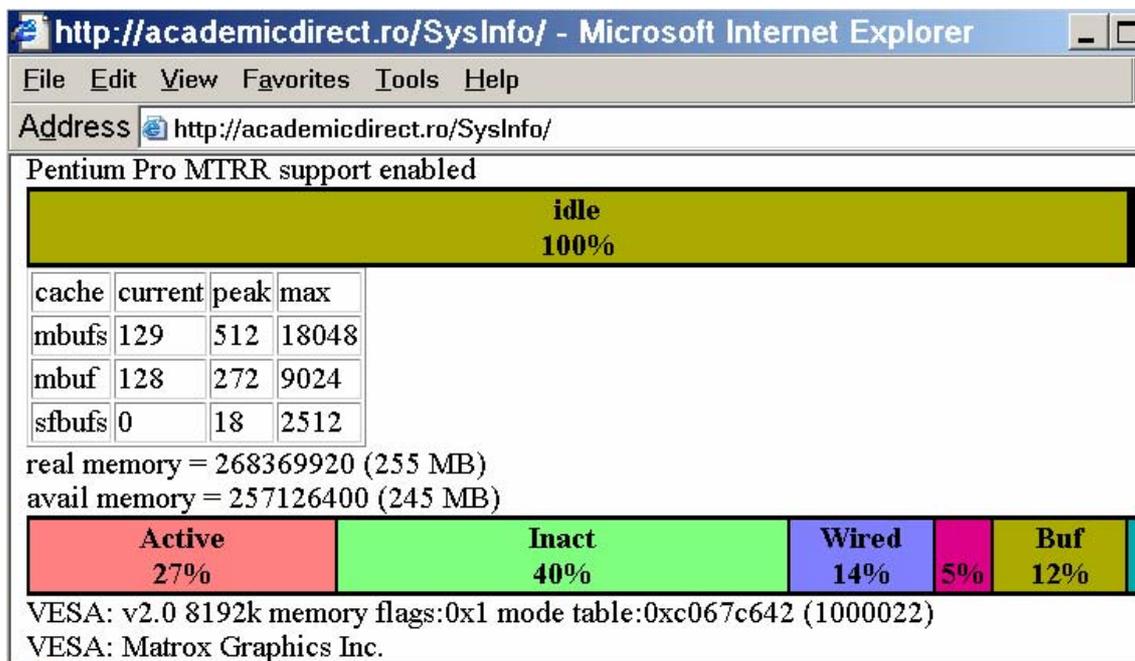
Fig. 2. Memory and CPU usage information – picture from ns.academicdirect.ro server

Filesystem	Size	Used	Avail	Capacity	iused	ifree	%iused	Mounted
/dev/ad0s1a	35G	2.5G	30G	8%	194.2K	4.4M	4%	/
devfs	1.0K	1.0K	0B	100%	0	0	100%	/dev
procfs	4.0K	4.0K	0B	100%	1	0	100%	/proc
/dev/ad0s1b	1024M	0B	1024M	0%	0	0	100%	none

Fig. 3. File system information – table from vl.academicdirect.ro server

Name	Mtu	Network	Address	Ipkts	Ierrs	Opkts	Oerrs	Coll
dc0	1500	(Link#1)	00:02:e3:08:68:69	13612	0	13883	0	0
dc0	1500	172.27.211/24	172.27.211.1	10716	-	15639	-	-
dc0	1500	192.168.211	192.168.211.1	2310	-	2424	-	-
fxp0	1500	(Link#2)	00:90:27:a5:61:dd	1017065	0	729087	0	0
fxp0	1500	193.226.7.128	ns	78368	-	48776	-	-
lo0	16384	(Link#3)		11182	0	11182	0	0
lo0	16384	your-net	localhost	120	-	120	-	-
ppp0*	1500	(Link#4)		10056	1	12317	0	0

Fig. 4. Network status information – table from ns.academicdirect.ro server

A Performance Counter Application

The second application was used as performance counter. It has also a web interface:


```
<form    method='post'    action='hin.php'
enctype='multipart/form-data'>
<input type='file' name='file'>
<input type='submit'>
</form>
```

Fig. 5. Submit form for the hin.php application

About application exploiting experience: there is a bug in Microsoft Internet Explorer 4.01 that does not allow header incomings for downloading of the output file. There is no paper devoted to this subject, in our best knowledge. There is also a bug in Microsoft Internet

Explorer 5.5 that interferes with this, which can be solved by upgrading to Service Pack 2 or later. Anyway, this problem does not affect our program.

The output data of execution for all three servers is presented in table 1:

Table 1. Statistics of the hin.php Program Execution

ip	name	CPU	RAM	mics (start)	s. (start)	mics (stop)	s. (stop)	time (s)
140	j	2P166MMX	128	0.565873	1077348916	0.213418	1077349252	335.6475
211	ns	P2/400	256	0.634248	1077349090	0.432039	1077349225	134.7978
200	vl	P3/800	512	0.623252	1077305562	0.632815	1077305616	54.00956

The data from table 1 allow us to put on a chart the compared results.

An observation is immediate: the time-time dependence from one generation to another one of Pentiums is almost linear, considering only the usage of pointer, string, and integer instructions (without any floating point instructions). In terms of performance, it means just a speed meter.

Also, note that: the usage of dual processor system is not different from the single processor one. A possible explanation comes from the algorithm design, which is classical, not a parallel one.
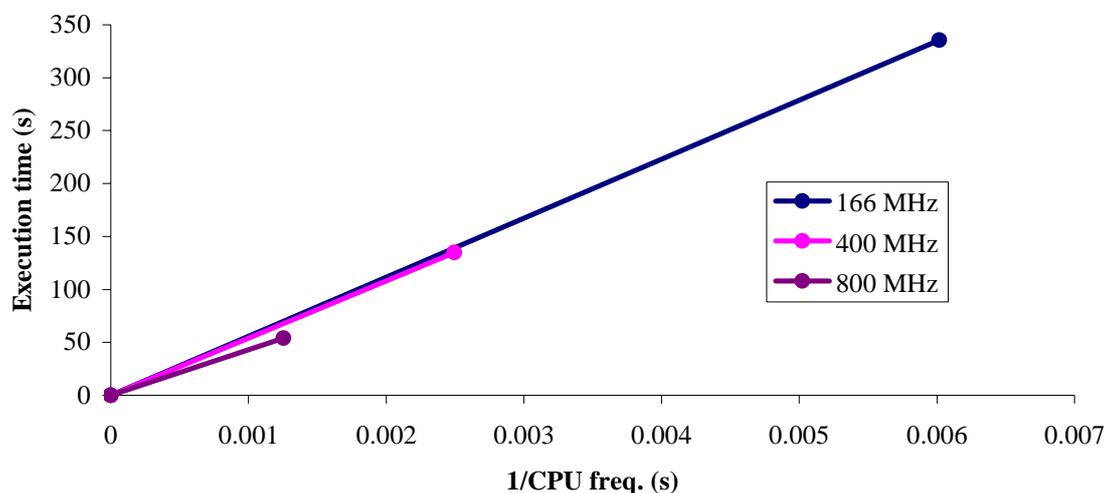
Fig. 6. Execution time vs. 1/CPU frequency

About deviation from linear dependence (Fig. 6 - all lines are draws from 0): it appears that the same real time for the processor is used more efficiently for instructions

processing in P II processors in comparison to the P processors and the jump is more obviously at P III architectures.

Dial-In Service

The system preparation can start from kernel configuration. Putting a line like:

options   CONSPEED=115200

the kernel will use the serial port as default at 115200 bps (instead of 9600 bps) and of course, the kernel must be recompiled. Anyway, looking at kernel boot messages, we must check if the sio device is installed on the system and is working (dmesg | grep sio).

The /etc/ttys file specifies various information about terminals on the system, including about sio ports.

Usually a program gets the control of sio port at the boot time. We chouse do not use the default program for sio console (getty), because this do not control correctly the modem, and we use the mgetty program [16]. Therefore, our entry lines in /etc/ttys file for sio port (called ttyd on FreeBSD system) are like: ttyd0 "/usr/local/sbin/mgetty -s 115200"   dialup on secure

The mgetty program has advantage to control also fax incomings (if the modem support this feature, at least class 2.0 fax). At the end of the mgetty program, we must create the configuration file (/usr/local/etc/mgetty+sendfax/mgetty.config):

direct NO

blocking NO

port-owner uucp

port-group uucp

port-mode 0660

toggle-dtr YES

toggle-dtr-waittime 500

data-only NO

fax-only NO

modem-type auto

init-chat "" ATS0=0Q0&D3&C1 OK

modem-check-time 3600

rings 3

```
answer-chat "" ATA CONNECT \c \r
```

```
answer-chat-timeout 30
```

```
autobauding NO
```

```
ringback NO
```

```
ringback-time 30
```

```
ignore-carrier false
```

```
issue-file /etc/issue
```

```
prompt-waittime 500
```

```
login-prompt @!login:
```

```
login-time 3600
```

```
diskspace 102400
```

```
notify lori
```

```
fax-owner uucp
```

```
fax-group modem
```

```
fax-mode 0660
```

For incoming calls receiving, the /usr/local/etc/mgetty+sendfax/dialin.config file must contain allowed incoming calls (all).

The squid service must be installed in the system and must be proper configured:

```
acl ppp src 192.168.211.0/24
```

assuming that our dial-in intranet network will use 192.168.211.XXX address class.

To transfer the packets to a network card, the /etc/rc.conf file must contain an alias: (ifconfig_dc0_alias0="inet 192.168.211.1 netmask 255.255.255.0") or the gateway service to be enabled (gateway_enable="YES").

If the ppp and tun devices are compiled into kernel, a good idea is to disable the module loadings to avoid over configurations (/boot/loader.conf file):

```
if_tun_load="NO"
```

```
if_ppp_load="NO"
```

For ppp service two utilities are available on a FreeBSD system: ppp and pppd. The ppp service uses the tun device and the pppd service use the ppp device. The used service it can be specified into /usr/local/etc/mgetty+sendfax/login.config file.

Using of ppp Service (tun Device)

The /usr/local/etc/mgetty+sendfax/login.config file it must contain a line like:

/AutoPPP/   -   -   /etc/ppp/ppp-dial

which will start automatically the ppp service using /etc/ppp/ppp-dial shell script.

Not that the /etc/ppp/ppp-dial file must have execution bit set:

-rwxr-xr-x 1 root wheel 45 Feb 3 15:19 /etc/ppp/ppp-dial (ls –al command)

therefore, a chmod +x /etc/ppp/ppp-dial command will solve the problem.

The /etc/ppp/ppp-dial shell script must launch the ppp daemon in direct mode:

#!/bin/sh

exec /usr/sbin/ppp –direct server

The ppp daemon will start using the /etc/ppp/ppp.conf configuration file. Note that the spaces from the beginning of rows are relevant here:

server:

 set dial "ABORT BUSY ABORT NO\\sCARRIER TIMEOUT 5 \"\" AT \

 OK-AT-OK ATE1Q0 OK \\dATDT\\T TIMEOUT 40 CONNECT"

 set ifaddr 192.168.211.1 192.168.211.2-192.168.211.3

 enable pap proxy passwdauth

 accept dns

Normally, the receiver of a connection requires that the peer authenticate itself. This may be done using login, but alternatively, you can use PAP (Password Authentication Protocol) or CHAP (Challenge Handshake Authentication Protocol). CHAP is the more secure of the two, but some clients may not support it. Our script (see above) use pap method. Anyway, ppp daemon looks for /etc/ppp/ppp.secret file in order to authenticate the client. This file contains one line per possible client, each line containing up to five fields. As example, our configuration file it contains:

lori lori * lori *

All must be ok for the ppp daemon with these configurations. Note that a good idea is to protect our /etc/ppp/ppp.secret file:

chmod 0400 /etc/ppp/ppp.secret

Using of pppd Service (ppp Device)

The /usr/local/etc/mgetty+sendfax/login.config file it must contain a line like:

/AutoPPP/   -   -   /etc/ppp/pppd-dial

which will start automatically the pppd service using /etc/ppp/pppd-dial shell script.

Not that the /etc/ppp/ppp-dial file must have execution bit set:

-rwxr-xr-x 1 root wheel 171 Feb 3 18:12 /etc/ppp/pppd-dial (ls –al command)

therefore, a chmod +x /etc/ppp/pppd-dial command will solve the problem.

The /etc/ppp/pppd-dial shell script must launch the pppd daemon. Note that the pppd daemon does not use the configuration from ppp.conf file, and therefore the configuration must be gives here:

#!/bin/sh

exec /usr/sbin/pppd auth 192.168.211.1:192.168.211.2 192.168.211.1:192.168.211.3 nodefaultroute ms-dns 193.226.7.211 ms-wins 193.226.7.211 ms-wins 172.27.211.2

At present, pppd supports two authentication protocols: PAP and CHAP. PAP involves the client sending its name and a clear text password to the server to authenticate it. In contrast, the server initiates the CHAP authentication exchange by sending a challenge to the client (the challenge packet includes the server's name). The client must respond with a response which includes its name and a hash value derived from the shared secret and the challenge, in order to prove that it knows the secret.

The PPP device, being symmetrical, allows both peers to require the other to authenticate itself. In that case, two separate and independent authentication exchanges will occur. The two exchanges could use different authentication protocols, and in principle, different names could be used in the two exchanges. The default behavior of pppd is to agree to authenticate if requested, and to not require authentication from the peer. However, pppd will not agree to authenticate itself with a particular protocol if it has no secrets, which could be used to do so. Pppd stores secrets for use in authentication in secrets files (/etc/ppp/pap-secrets for PAP, /etc/ppp/chap-secrets for CHAP). Both secrets files have the same format. The secrets files can contain secrets for pppd to use in authenticating itself to other systems, as well as secrets for pppd to use when authenticating other systems to it. Each line in a secrets file contains one secret. A given secret is specific to a particular combination of client and server - that client to authenticate it to that server can only use it. Thus, each line in a secrets file has at least three

fields: the name of the client, the name of the server, and the secret. These fields may be followed by a list of the IP addresses that the specified client may use when connecting to the specified server. Therefore, our PAP/CHAP configuration files contain same secrets:

lori academicdirect.ro lori *

The pppd secret files can have also read protection, like for ppp service:

chmod 0400 /etc/ppp/chap-secrets

chmod 0400 /etc/ppp/pap-secrets

Testing of ppp and tun Devices

The debugging mode of the service allows us to look at the communication history for a connection, to identify configuration mistakes and so on.

The default logs file for ppp service is /var/log/ppp.log. After the finishing of configuration process, a good idea is to disable the loggings (set log Phase tun command). Starting with ppp service discussion, another observation is important there: if we are using a windows system for connect to dial-in server, a option must be disabled in windows ppp service configuration:

Start/Settings/Dial-Up Networking/lori(connection)/Properties/Security(Advanced security options:)/Require

encrypted password - must be UNSET

The server message for a debugging mode connection using ppp service is listed there:

Feb 3 17:54:34 ns ppp[647]: Phase: Using interface: tun0

Feb 3 17:54:34 ns ppp[647]: Phase: deflink: Created in closed state

Feb 3 17:54:34 ns ppp[647]: tun0: Command: server: set dial ABORT BUSY ABORT NO\sCARRIER

TIMEOUT 5 "" AT OK-AT-OK ATE1Q0 OK \dATDT\T TIMEOUT 40 CONNECT

Feb 3 17:54:34 ns ppp[647]: tun0: Command: server: set ifaddr 192.168.211.1 192.168.211.2-192.168.211.3

Feb 3 17:54:34 ns ppp[647]: tun0: Command: server: enable pap proxy passwdauth

Feb 3 17:54:34 ns ppp[647]: tun0: Command: server: accept dns

Feb 3 17:54:34 ns ppp[647]: tun0: Phase: PPP Started (direct mode).

Feb 3 17:54:34 ns ppp[647]: tun0: Phase: bundle: Establish

Feb 3 17:54:34 ns ppp[647]: tun0: Phase: deflink: closed -> opening

Feb 3 17:54:34 ns ppp[647]: tun0: Phase: deflink: Connected!

Feb 3 17:54:34 ns ppp[647]: tun0: Phase: deflink: opening -> carrier

Feb 3 17:54:35 ns ppp[647]: tun0: Phase: deflink: /dev/ttyd0: CD detected

Feb 3 17:54:35 ns ppp[647]: tun0: Phase: deflink: carrier -> lcp

Feb 3 17:54:39 ns ppp[647]: tun0: Phase: bundle: Authenticate

Feb 3 17:54:39 ns ppp[647]: tun0: Phase: deflink: his = none, mine = PAP

Feb 3 17:54:39 ns ppp[647]: tun0: Phase: Pap Input: REQUEST (lori)

Feb 3 17:54:39 ns ppp[647]: tun0: Phase: Pap Output: SUCCESS

Feb 3 17:54:39 ns ppp[647]: tun0: Phase: deflink: lcp -> open

Feb 3 17:54:39 ns ppp[647]: tun0: Phase: bundle: Network

Feb 3 17:54:50 ns ppp[647]: tun0: Phase: deflink: open -> lcp

Feb 3 17:54:50 ns ppp[647]: tun0: Phase: bundle: Terminate

Feb 3 17:54:52 ns ppp[647]: tun0: Phase: deflink: Carrier lost

Feb 3 17:54:52 ns ppp[647]: tun0: Phase: deflink: Disconnected!

Feb 3 17:54:52 ns ppp[647]: tun0: Phase: deflink: Connect time: 18 secs: 2177 octets in, 14535 octets out

Feb 3 17:54:52 ns ppp[647]: tun0: Phase: deflink: 69 packets in, 47 packets out

Feb 3 17:54:52 ns ppp[647]: tun0: Phase: total 928 bytes/sec, peak 3171 bytes/sec on Tue Feb 3 17:54:48 2004

Feb 3 17:54:52 ns ppp[655]: tun0: Phase: deflink: lcp -> closed

Feb 3 17:54:52 ns ppp[655]: tun0: Phase: bundle: Dead

Feb 3 17:54:52 ns ppp[655]: tun0: Phase: PPP Terminated (normal).

The pppd service doesn't have a default file for logging, so we must create it and specify the debugging level in our /etc/ppp/pppd-dial shell script. After the finishing of configuration process, a good idea is to disable the loggings.

Feb 3 17:56:47 ns pppd[656]: pppd 2.3.5 started by root, uid 0

Feb 3 17:56:47 ns pppd[656]: Using interface ppp0

Feb 3 17:56:47 ns pppd[656]: Connect: ppp0 <--> /dev/ttyd0

Feb 3 17:56:50 ns pppd[656]: sent [CHAP Challenge id=0x1 <1e9195c5294cb50b80401cd9e34f15a89b2b823509 aef6fead7549f8b0ae5695a7710a966996>, name = "ns.academicdirect.ro"]

Feb 3 17:56:50 ns pppd[656]: rcvd [CHAP Response id=0x1 <8126c29e65530d6114658931eaa57390>, name = "lori"]

Feb 3 17:56:50 ns pppd[656]: sent [CHAP Success id=0x1 "Welcome to ns.academicdirect.ro."]

Feb 3 17:56:50 ns pppd[656]: sent [IPCP ConfReq id=0x1 <addr 192.168.211.1> <compress VJ 0f 01>]

Feb 3 17:56:50 ns pppd[656]: CHAP peer authentication succeeded for lori

Feb 3 17:56:50 ns pppd[656]: rcvd [IPCP ConfReq id=0x1 <compress VJ 0f 01> <addr 192.168.211.3> <ms-dns 0.0.0.0> <ms-wins 0.0.0.0> <ms-dns 0.0.0.0> <ms-wins 0.0.0.0>]

Feb 3 17:56:50 ns pppd[656]: sent [IPCP ConfNak id=0x1 <ms-dns 193.226.7.211> <ms-wins 193.226.7.211> <ms-dns 193.226.7.211> <ms-wins 172.27.211.2>]

Feb 3 17:56:50 ns pppd[656]: rcvd [IPCP ConfAck id=0x1 <addr 192.168.211.1> <compress VJ 0f 01>]

Feb 3 17:56:51 ns pppd[656]: rcvd [IPCP ConfReq id=0x2 <compress VJ 0f 01> <addr 192.168.211.3> <ms-dns 193.226.7.211> <ms-wins 193.226.7.211> <ms-dns 193.226.7.211> <ms-wins 172.27.211.2>]

Feb 3 17:56:51 ns pppd[656]: sent [IPCP ConfAck id=0x2 <compress VJ 0f 01> <addr 192.168.211.3> <ms-dns 193.226.7.211> <ms-wins 193.226.7.211> <ms-dns 193.226.7.211> <ms-wins 172.27.211.2>]

Feb 3 17:56:51 ns pppd[656]: local IP address 192.168.211.1

Feb 3 17:56:51 ns pppd[656]: remote IP address 192.168.211.3

Feb 3 17:56:51 ns pppd[656]: Compression disabled by peer.

Feb 3 17:57:04 ns pppd[656]: Hangup (SIGHUP)

Feb 3 17:57:04 ns pppd[656]: Modem hangup, connected for 1 minutes

Feb 3 17:57:04 ns pppd[656]: Connection terminated, connected for 1 minutes

Feb 3 17:57:05 ns pppd[656]: Exit.

The ppp and tun devices can be monitored via a web interface. Our results are depicted in table 1:

Table 1. Network status table from ns.academicdirect.ro server

Name	Mtu	Network	Address	Ipkts	Ierrs	Opkts	Oerrs	Coll
…	…	…	…	…	…	…	…	…
ppp0*	1500	(Link#4)		18922	1	23447	0	0
tun0*	1500	(Link#5)		9322	34	12425	38	0

Conclusions

Because the FreeBSD is available free of charge (for individuals and for organizations) to use and comes with full source code, anyone which want a featured server operating system (opposing to NetBSD, a very conservative legal copyright ported software system, or OpenBSD, a very conservative ported software security system) can install it.

FreeBSD operating system procedure is easy to follow even if the administrator does not have experience with BSD-like systems.

The feature of boot floppies allows us to install FreeBSD even if we do not have a FreeBSD CD or CD-drive in the system.

CVSup mechanism offers an efficient way to maintain and update the system.

The kernel recompilation allows us to improve the performance of the system, in terms of speed and memory management.

Looking at hardware characteristics and including corresponding options in kernel are obtained a better exploiting of the hardware resources. More, some specific hardware are then detected and configured for use.

Creating a backup copy of the kernel, we can undo any action of kernel reinstalling.

If some service does not start automatically, from unknown reasons (such as *ftpd* on j.academicdirect.ro server), we can try to start manually and after that we can create a script for auto start.

Another exemplified situation show that not always the installation scripts puts all required data in configuration files (*AddType application/x-httpd-php .php*) and if a module does not start, a good idea is to look carefully at service configuration file.

The *SysInfo* application for system information allows one to inspect the system state via web. The application is useful for system administrators and presumes identification of system failures.

The use of the system utilities to obtain the displayed information makes the application portable onto different systems and architectures.

The PHP language offers a very good interface with system utilities and an efficient way to develop client-server applications.

The second application which tests both PHP capabilities and system performance, proves that, even if the constant controlling the number of consecutive calls of *recv()* recursive function has big values (like 40 or 50), the program does not crash. The comparative study on the three Intel-based systems showed the qualitative difference among various Pentium processor architectures.

Surprisingly (or not), using a dual processor system within an interactive time-sharing operating system does not mean that the system makes parallel processing.

The CHAP81 variant of Microsoft Windows operating system creates some misunderstandings in authentication using ppp service (tun device) and only PAP authentication protocol are agreed by the server. The pppd works fine with CHAP81. Same observation is noted for password encryption (see text).

The default program for sio device (getty) doesn't offer the full support for modem control, and mgetty program is a good replacement.

Kernel recompilation allows us to modify the default communication speed for sio ports.

The debugging mode log files is the best starting point if something goes wrong and the communication fails.

References

1. UNIX-like operating system official http sites: debian.org (Debian), www.linux.org (Linux), lindows.com (LinwowsOS), ibm.com/servers/aix/os (AIX), www.gnu.org (GNU), hp.com/products1/unix/operating (HP-UX), opengroup.org (OSF), sun.com/software/solaris (Solaris), www.bsd.org (BSD/OS), netbsd.org (NetBSD), openbsd.org (OpenBSD), freebsd.org (FreeBSD), www.trustedbsd.org (TrustedBSD), picobsd.org (PicoBSD)

2. http://www.bartleby.com/61, *The American Heritage Dictionary of the English Language*, Fourth Edition.

3. http://www.delorie.com/gnu/docs/vera, *Virtual Entity of Relevant Acronyms*, The Free Software Foundation.

4. http://www.instantweb.com/d/dictionary, *Free Online Dictionary of Computing*.

5. www.freebsd.org/cgi/man.cgi, *FreeBSD Hypertext Man Pages*.

6. www.php.net, *The PHP Group* (Arntzen T. C., Bakken S., Caraveo S., Gutmans A., Lerdorf R., Ruby S., Schumann S., Suraski Z., Winstead J., Zmievski A.).

7. http://phpsysinfo.sourceforge.net, *Open Source Development Network*.

8. John D. Polstra, http://www.cvsup.org/faq.html, *CVSup Frequently Asked Questions*.

9. Jäntschi L., Zaharieva-Stoyanova E., *Upload a File to a Server. Case Study*, UNITECH'03 International Scientific Conference, November 21-22 2003, Gabrovo, Bulgaria, volume I "ISC UNITECH'03 GABROVO Proceedings", p. 274-276.

10. Greer D. J., http://www.robelle.com/library/papers/www-paper/clients.html, *Dial-in Access*, Robelle Solutions Technology Inc.

11. http://www.speedtouchdsl.com/supfaq.htm, *SpeedTouch DSL modems*, Thomson.

12. http://www.greatware.net/modems, *Modems*, Great Internet Software.

13. http://server.rotld.ro/cgi-bin/whois?whois=academicdirect.ro, *Romanian National R&D Computer Network*.

14. http://www.apache.org, *The Apache Software Foundation*.

15. http://www.mysql.org, *MySQL AB*.

16. http://www.webforum.de/mgetty-faq.html, *Smart Modem Getty*, Gert Doering.