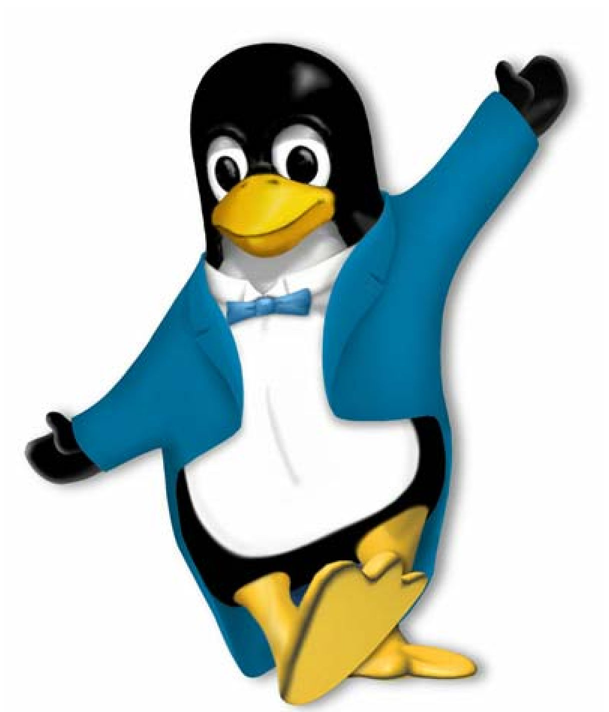
The IBM Corporation
is pleased to Present the

Linux Scholars' Challenge Winning Entries



(presented in alphabetical order by country)

- **Australia**
  Michael Berhanu -- **Page 5**
  University: Australian National University
- **Argentina**
  Sebastian Arena -- **Page 9**
  University: Facultdad de Ingenieria
- **Bulgaria**
  Veselin Kanev -- **Page14**
  University: Varna's Technical University
- **China**
  Feng Gao -- **Page 18**
  University: Nanjing University
- **China**
  Hu Jiangtao -- **Page 29**
  University: Fudan University
- **Finland**
  Mika Pruikkonen -- **Page 35**
  University: Helsinki University of Technology
- **France**
  Tristan Leteutre -- **Page 40**
  University: Ecole Centrale Paris
- **Germany**
  Peter Rost -- **Page 44**
  University: Berufsakademic Dresden
- **Germany**
  Klaus Wehrle -- **Page 48**
  University: University of Karlsruhe

- **Germany**
  Uwe Walter -- **Page 61**
  University: University of Karlsruhe

- **India**
  Muthukumar Shunmugiah -- **Page 66**
  University: P.S.G. College of Technology

- **India**
  Shah Sachin -- **Page 72**
  University: GLS Institute of Computer Technology

- **India**
  Sreeram Jaswanth (Jesse) --- **Page 76**
  University: International Institute of Information Technology

- **Italy**
  Francesco Regazzoni --- **Page 85**
  University: Politecnico di Milano

- **Korea**
  Jongmin Park --- **Page 88**
  University: University of ULSAN Korea

- **New Zealand**
  Frank Kruchio -- **Page 92**
  University: Victoria University of Wellington

- **Portugal**
  Bruno Silva Kruchio -- **Page 95**
  University: Universaidade De Aveiro

- **United Kingdom**
  Helmut Cantzler --- **Page 100**
  University: University of Edinburgh

- **United Kingdom**
  Barnaby Gray -- **Page 106**
  University: **University of Cambridge**

- **United States of America (USA)**
  Dwight Tuinstra -- **Page 109**
  University: **Clarkson University**

- **United States of America**
  Phillip Allen -- **Page 113**
  University: **Clarkson University**

- **United States of America**
  Rimon Barr -- **Page 117**
  University: **Cornell University**

- **United States of America**
  Bryan Clark --- **Page 122**
  University: **Clarkson University**

- **United States of America**
  Mike Schellhase --- **Page 127**
  University: **Mellon University**

- **United States of America (USA)**
  Nick Pattengale -- **Page 130**
  University: **New Mexico Institute of Mining and Technology**

Paper Submitted by:

Michael Berhanu

University:

Australian National University

Country:

Australia

Automated Test Tools that Execute Test Scenarios and Collect Data Resulting from the Test Execution

Part I

Test workbenches and automated test tools are an integral part of any major closed source software development project. This however, is not the case with most (even large) open source projects. One of the major reasons for this is the lack of automated test tools, which work well with the products being developed. This project aims to lay the foundation of such a tool.

For an automated test tool to be effective in the Linux environment it must be well integrated with other tools and techniques used by the open source, and especially Linux community.

The objectives of this project is to build a software product, **BugSeeker** using existing software written under the GNU General Public License, which has the following characteristics:

- automatically executes test scenarios
- collects data resulting from such test executions
- be able to do this for software applications with command line interfaces
- be able to do this for software applications with graphical user interfaces
- compare the expected and output from the software program to check weither the test has passed
- be well integrated with other open source testing software already available
- ability to generate test reports

Part II

The starting point for this project was Sommerville[1], in which automated test tools are discussed and described. After obtaining a broad view of the required system, the following products with similar objectives were examined:

Rational Suite TestStudio (www.rational.com/products/systest.jsp) - commercial product known to be effective tool for testing software applications. This product was a lot more comprehensive than this BugSeeker, but it showed a function that many test tools have, the ability to generate test reports.

Bugzilla (www.mozilla.org/projects/bugzilla/) - Bugzilla is the most widely used bug tracking software system in the open source community. This is why it would be important for a successful automated test tool to be able to directly communicate with Bugzilla.

Mozilla Crash-Data News group (netscape.public.mozilla.crash-data) - Questions asked to this news group were generally based around what features would an automated test tool have, for it to be more effective than a scripting language or manual hand testing. The major feature requested was that the tool should be able to test applications with graphical user interfaces and be able to do regression testing.

Android (www.smith-house.org/open.html) - BugSeeker uses this tool to implement its graphical users interface testing features. It calls the program, gives it input, and records any output that is received.

Part III

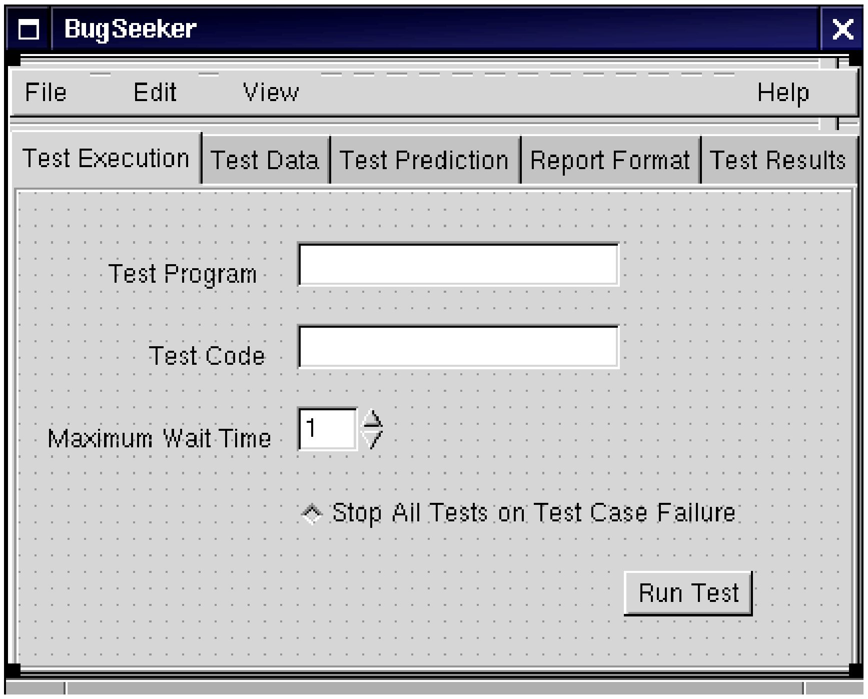
BugSeeker's capabilities are slightly broader than that described in IBM's Linux Scholar Challenge No. 17. Along with the ability to execute test scenarios and collect data from the test execution it also has the following features:

- Graphical user interface: which can be used to test both command line driven applications and applications with graphical user interfaces.
- Test data generator: generates various test data based on regular expressions.
- Oracle pluggin: expected results files or previous version of program (used for regression testing).
- Test results report generation: Currently a table with columns labeled input, outpu
- and test passed (based on weither the output is the same as that described by the Oracle pluggin).
- Integration with Bugzilla: facility to report any bugs found to a Bugzilla server upon the request of the tester.

Snapshot of BugSeeker:

Running under FVWM2. This screen is the first to load when the program starts.

[1] Sommerville, I., 2001, *Software Engineering Sixth Edition*, Addison-Wesley Publishers Ltd. England



Paper Submitted by:

Sebastian Arena

University:

Facultdad de Ingenieria

Country:

Argentina

Figure out the Fastest Way to Boot a Full Linux on a ThinkPad (Boot Speed)

Project Objectives

My objective is to determine the best configuration for Linux to run on an IBM ThinkPad laptop. The key factors to accomplish this are mostly two:

1. Kernel and Modularity
2. Startup Programs (or Daemons)

This task implies to know what are the hardware setups for the ThinkPads, so I can find what should be included on the kernel and what shouldn't. Then I should decide which features will be inside the kernel code and which ones will be on separated modules. The startup programs will be discussed later.

The requirements of the challenge say that boot speed must be improved in a Full Linux box, I believe that a Full Linux includes all the features mentioned before : Video (X Support), Sound (Multimedia), Internet, and Plug and Play , with these any user would have good Linux experience.

Research

First of all I visited IBM Argentina's website to see which were the different models for the ThinkPad. The models I found were: ThinkPad A Series, T Series, X Series, i Series and Transnote.

After reading the hardware information carefully I started with my first objective:Kernel and Modularity. It's known that the smaller the kernel the faster it loads and works, so, using the information from the ThinkPad's models I started finding the common patterns in all of them, to figure out which global things would be included on the kernel code (I used kernel 2.4.9 source code for my research –) and which ones would be good to have on modules. The modules are the best way to guarantee that, since laptops are known to use PCMCIA ports, there are lots of devices that can be used, but not all at the same time, so the modules reduce the kernel code and are loadable any time they are needed. Almost every model uses an Intel Pentium III microprocessor (except the ThinkPad i Series that uses Celeron), despite ThinkPads use a Mobile version of the PIII I checked out and found that this microprocessor is supported by the kernel. Also, the models have 56K Modems and Ethernet cards by default hardware setup, nowadays modems are surely used however people who doesn't work in an office don't take advantage of Ethernet cards, so I decided that Ethernet configuration would be on a separated module. I have extracted information from, about hot-pluggable devices such as the PCMCIA ones, this is a special kernel feature: it enables any device that is plugged into the system to be recognized and ready to use, as it is important for laptops, it will be included in our kernel code. People from helped me to find out what happened to kernel size if certain features were removed, what I did find out is that there is no actual, precise or accurate information on how much the kernel code is increased or reduced by putting or removing features; It seems that when you add any new option the code adds a new structure to support that, but on the same structure, similar features related to the first one can be supported, so concluding, if you remove what you think is a key factor to reduce kernel size, it may not free as much space as another one might.

To solve this problem I have been testing several kernel configurations and I have chosen the one that suits best for our purpose (this configuration is deecribed at the end of the essay). I decided that Parallel Port support , Plug and Play support, TCP/IP networking (but no packet filtering), Enhanced IDE and IDE/ATAPI CDROM support (since most of the models have CD-ROMs included), PPP protocol, DOS, VFAT,ISO 9660 and EXT2 file system support should also be directly included in the kernel code.

SCSI support wasn't needed, despite that Bluetooth and IrDA are nowadays used by laptops, they are not supported by ThinkPads, but they could be easily compiled as modules if devices where design for PCMCIA. Also USB is neither included by default so it wasn't added to the kernel.

The most important things to be compiled as modules are:

1. Sound Support (in this case it would be SounblasterPro compatible (A Series) – supported by kernel 2.4.9 -- or Crystal Semiconductor CS4624/CS4297A (T and X Series)--not supported yet)

2. Video Cards and AGP support (in this case IBM doesn't provide information on cards brand, except from i Series that use an ALI chip that is surely supported)

3. Ethernet cards brand aren't informed either, but as Sound and Video they can be easily compiled as modules. Finally from I found a text about how to make the kernel smaller and optimized that was very helpful.

My second objective was to reduce programs or daemons that are loaded on startup, such as Web-Server, MTA (Mail Transport Agent), X, etc. Most Linux distributions install by default Apache, sendmail, X and others, that are setup to run on startup, but most people don't take any advantage on this. The average user, tends to use a graphical front-end (not all users prefer this, console is fast and easy to use, but I suppose IBM wants to have X in a Full Linux box) to manage their system, to work, to play, etc. , so X should be included at startup (like xdm, kdm, gdm, or plain startx) although it takes some time to load up. X implies a mouse server such as gpm and also server font. This is all that should be loaded, nothing else is required except from the default kernel things such as setserial and other configurations that must be done in order for the kernel to work properly. Some daemons are also needed to log the system status. From I extracted some good tips on how to remove unnecessary programs. At I found out what I could and what couldn't remove from /etc./init.d (here is where startup programs are set). Is good to know also that /etc./rcS** configures other features to be loaded on startup, and are set in different runlevels.

I will now explain what were my results when I started testing some kernel configurations and modifying /etc./init.d to improve boot speed.

Results Achieved

The first thing I did was very simple: I modify /etc./init.d to the minimal requirements as I read on howtos downloaded from and others mentioned before. I don't actually have a ThinkPad or any other laptop, so I tested the configuration on my actual box that is also a PIII (450 MHz) to check if boot speed was improved. I only kept gpm (Mouse Server), Gdm (Graphical Interface) , Log daemons and others needed by the system. I was currently on about 40 seconds to finish booting (of course this also depends on my hard drive and clock speed, on ThinkPads this would be much faster).

The only thing left was to make the kernel as small as possible. To do this I started from a working kernel configuration and removing unnecessary features step by step for a standard installation. I kept in mind what I explained before to set the kernel. My first try made a kernel of 568 kb., it was compiled with no modules at all, everything was inside the kernel code. This kernel did not fulfill any objective because booting speed wasn't considerably reduced.

After several tries I realized that my idea was partly wrong, many more things could be loaded up as modules (the loading of modules also takes time if you load everyones you compile, but not so much time if you just use a few of them and then load them when you need; this task can be set up so the kernel automatically does the loading), such as, a.out programs support (that adds about 10k to the code), APM management, Parallel port, Plug and Play, ISA PNP, Floppy disk, PPP, Mouse and Partition support such as VFAT, DOS, ISO9660 (with Joliet), etc. By making all these as modules I could make a 400 kb. kernel, that is consider as a small Image (because when compiling the Make command doesn't need bzImage (Big Image), it can use zImage if kernel is less than 500 kb.). One remarkable feature that was removed should be mentioned: PCI Name Database administrates names for the PCI devices detected in the system, if removed devices are only seen as numbers for the kernel, the user doen't need to know how the devices are a called (of course they need to know in a higher level, like configuring devices, but since laptops have install the same hardware always, these names can be factory default set). I also removed it because it added 50 kb. in a 400 kb. kernel.

My Final Conclusion is: In order to reduce boot time and increase boot speed you must:

First: Make your kernel as small as possible, so it not only loads faster but works faster afterwards too. A Full Linux should have support for Multimedia (Sound and Video), PNP and Internet, in order to do this many features can be loaded as modules, so the kernel is not overloaded. Everything used once in a while (Ethernet, CD, Floppy, Printer) must be loaded as modules so it can easily be unloaded when not used. This will reduce boot time in 5 to 8 seconds.

Second:Remove from /etc./init.d all programs that aren't needed as essential for the Linux box to run, such as, Web servers, Mail servers, Daemons that await external connections (telnet, ssh). Only log daemons and graphical interface (including mouse server, font server, etc.) should be loaded at startup for the normal user to work, play and administrate the system correctly. This will decrease boot time in between 10 and 15 seconds.

Depending on the setup you choose booting speed can be increased in 15-20 seconds.

Paper Submitted by:

Veselin Kanev

University:

Varna's Technical University

Country:

Bulgaria

The TraffLog Project

Part 1. The Project

The main objective of the TraffLog project is to develop reliable and independent software for IP accounting. That software will be able to log source and destination ip addresses, source and destination MAC addresses, the Type of Service, source and destination port and the time that each IP packet passed thru the server. With all that data logged you can easily create various reports for the IP traffic. For example: you can dump all the packets passed thru the server, with all information for each packet, you can draw various graphics using that data, you can view the most visited web pages, so making decision for building a peer line will be easy.

Part 2. How Did I Build It?

A. The Logging Software - "Accounting Sniffer"

There are several good IP accounting software products based on Linux firewall capabilities. But mixing firewall rules and IP accounting rules on a single Linux server sometimes may cause you a headache. Although you can hardly build IP accounting software capable of logging each packet's header and generating complicated reports using only Linux firewall. So, I decided to build an accounting software based on a packet sniffer. I could use the PCAP library for building my "accounting sniffer", but I decided it's better to build the software using plain Unix sockets. Not using PCAP has one advantage - the software relies on no external libraries for collecting information, and one disadvantage - it can only by run on a Linux server. Storing such amount of data requires a good database system. Without reinventing the wheel and building my own I used MySQL database system and its libmysqlclient library. This gave me a very good opportunity - using MySQL network capabilities my small "accounting sniffer" can work on many different routers of a Wide Area Network and log its data on a centralized database server.

But passing the information from the accounting software to the database system means additional network load. By using a data structure holding the information for the packets and flushing it every N minutes to the database I minimized the network load, because of the mysqlclient library capability to compress the data that is being send to the MySQL server.

Additional minimization of the data amount may be accomplished by searching the data structure for older record that is identical to the current packet (same source, destination address and port) and adding the current packet's size to the older identical packet's size. Using this method I can store information for two or more packets on a single member of my data structure. The disadvantage is the decreasing of the timestamp accuracy with maximum error of N minutes (the time interval between two passing the data to the database system). Placing a centralized "accounting sniffer" on the router with MySQL installed on it will eliminate the problems for time accuracy and network load. The logging software will have a lighting speed internal socket communication with the MySQL server. Building a high speed independed network connection between the centralized gateway Linux router and the database server is also an option.

A major problem accrued with the data storage. Holding information for the all IP packets requires a huge disk drive. And as the months pass the data amount stored in the database will grow, making the SQL queries executed by the server slower. A good decision of the problem is making summaries of the data for the previous months, storing them in the same database and deleting the detailed information.Holding the detailed data for the current and previous month and having summaries for the other months is good for a small cable company.

For testing I ran two of my "accounting sniffers" on two Linux routers of a local cable company and set them to log the data on a MySQL server on the same network. After a month the database was holding 140, 072 entries. The machine with the MySQL server was also used as a game server with Counter Strike server installed on it. When the Counter Strike server was heavy loaded the queries to the MySQL took more than 15 seconds, witch is far away from acceptable. If I try to log the data from all the routers of that company MySQL will ran out of steam. I went to the MySQL web site and looked over the benchmarks. IBM's DB2 is ten times faster than MySQL and is the fastest among Informix, SyBase and Oracle 8.0.3 on retrieving data from the database (according to the benchmarks in the MySQL web site). The "accounting sniffer" should be developed using some commercial DBMS. IBM's DB2 is a great idea.

B. The Front-End Interface

With the data being stored in the SQL server, building the front-end interface is easy. The main development languages have already support for variousdatabase systems, so the front-end interface may be build using some Windows based programming language (Delphi, C++ for Windows), Java or PHP or/and Perl (for Web based interface). I used PHP for building a Web based interface. The Web based interface has one great advantage - it can be accessed thru any terminal with simple web browser. All the calculations needed for the traffic reports are done either by the PHP module or the MySQL server, thus the terminals are not required to be very fast computers with great amount of memory. A simple office computer will do.

Part 3. The Results:

The working "accounting sniffer" software is really small - only 23Kbytes compiled, and it has about 450 lines of ANSI C code. Keeping the size small is essential for the stability of the software and follows one of the great ideas of the UNIX operating systems -"build many small and good working software tools, combine them and get a complex system with great power and durability". And that is what I have done - my small "accounting sniffer" collects the data, then pass it to the stable database server. The reports are made by Apache+PHP web server and passed to the viewing software - the Web browser.

With all the IP packets information logged on a MySQL server you can create various reports for the IP traffic. The front-end software does not limit the reports,as it is build with an open source technology - PHP. Any third-party programmer can expand the front-end capabilities and create very specific reports. There is another one great aspect of this "accounting sniffer" - security. When the network's security officer has all the IP packets information logged she/he can easily determine the source address of a DoS attack, or the IP address of a hacker and what kind of tool she/he used to break in the server.

The only problem that I could forsee is that MySQL DBMS will not manage with huge data - easily created with my "accounting sniffer". MySQL is good for working on small networks, but using this accounting software on vast networks requires a solid DBMS like IBM's DB2.

Paper Submitted by:

Feng Gao

(newmarch@263.nett)

University:

Nanjing University

Country:

China

Linux Device Driver

Introduction

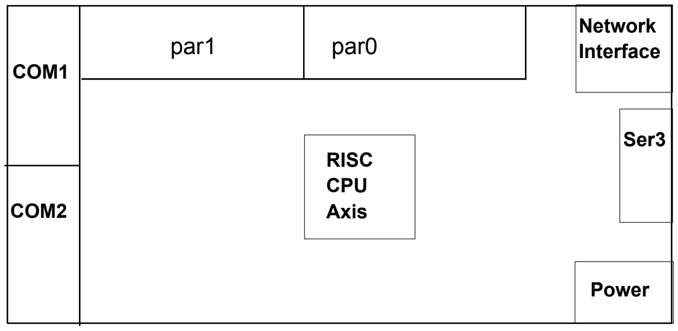
This system is an application of Swedish Axis Company's imbedded system developer board. A minilinux OS called Elinux is solidified inside the Axis Developer Board. It drives standard LCD module and displays what we want, such as roll-screen displaying, files displaying, remote control and so on.

Keywords: Elinux drivers LCD Parallel Printer Ports Socket Programming

Part One: Introduction of Developer Environment

Developer Board ETRAX100 is developed by Axis company, and it is a highly integrated imbedded system development toolkit. For ETRAX development Axis has developed a prototyping board that includes the most commonly used ports on the ETRAX chip. This ThinServer will quickly get you up to speed in development.

This is structure frame of Etrax100:



There are three COM ports£ºCOM1, COM2, Ser3 and two parallel printer ports: par0/LPT1, par1/LPT2. The standard voltage level of par0 is 3.3v, which is compatible with standard environmental devices. Port par0 has a buffer and a converter, but port par1 not . Port par1 can used to be connected with other parallel devices , such as LCD . The kernel of Axis' Developer Board is a special CPU, which integrates 2M flash rom and 8M ram . The maximum entension is 4M flash rom and 8M ram. A minilinux OS called Elinux is solidified inside the flash rom . We can practise some simple tasks , such as telnet (at most 3 users telnet elinux at the same time) and some common commands under linux bash: cd, ls, cat, exit and so on. In this system we use only parallel port A, which is used to connect with LCD.

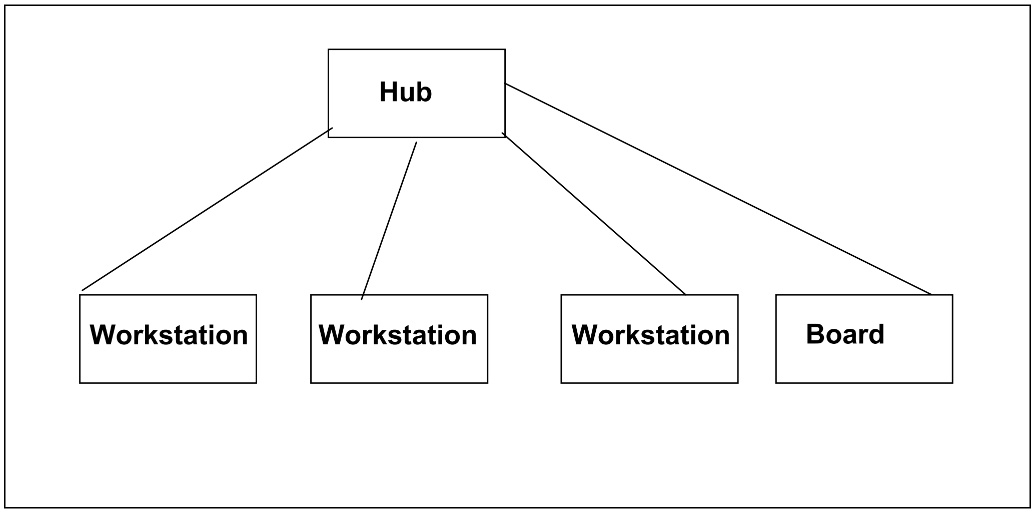
Total 26 pins, as follows:

```
25 23 21 19 17 15 13 11 9   7 5 3 1
26 24 22 20 18 16 14 12 10 8 6 4 2
```

PINS DEFINITIONS:

Header pinout	I/O	Header pinout	I/O
1 ¨CSTROBE, PR_ACK	O	2 -D0	--
3 -D1	--	4 -D2	--
5 -D3	--	6 -D4	--
7 -D5	--	8 -D6	--
9 -D7	--	10 ¨CACK, PR_REQ	I
11 BUSY, RD_WR	I	12 PAPER_E	I
13 SELECT,-INTIO	I	14 ¨CAUTO_FD	O
15 ¨CFAULT,PR_ADR0	I	16 ¨CINIT,PR_INT	O
17 ¨CSEL_IN	O	18 GND	--
19 GND	--	20 GND	--
21 GND	--	22 GND	--
23 GND	--	24 GND	--
25 GND	--	26 OUTPUT_ENABLE	O

Development Frame :



Above we are famaliar with our hardware developping environment.

Then we first install necessary software. We can get all information in the Axis Developer Documentation, which include:

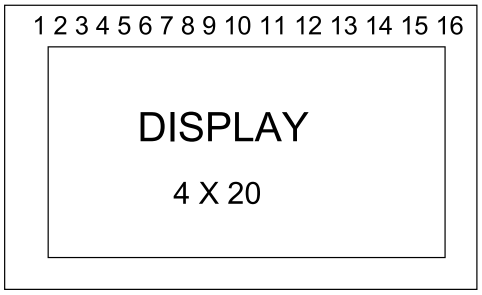
•How to Install the Developer Board Software;
•How to use mkprod;
•How to use boot elinux;
•Network Boot and Images;
•How to Write and Build Applications for ETRAX100;
•How to Write Device Drivers for ETRAX 100;
•How to Add Support for Flash Devices;

According to these we build our software developing environment.

Part Two: Write a LCD HD44780 Device Driver

We use a character lcd whose control chip is HD44780 . HD44780 displays four lines and twenty characters per line . Because the production of LCD has already realized standardization, all LCDs with HD44780 have common pins definition and control signals.

LCD Picture:



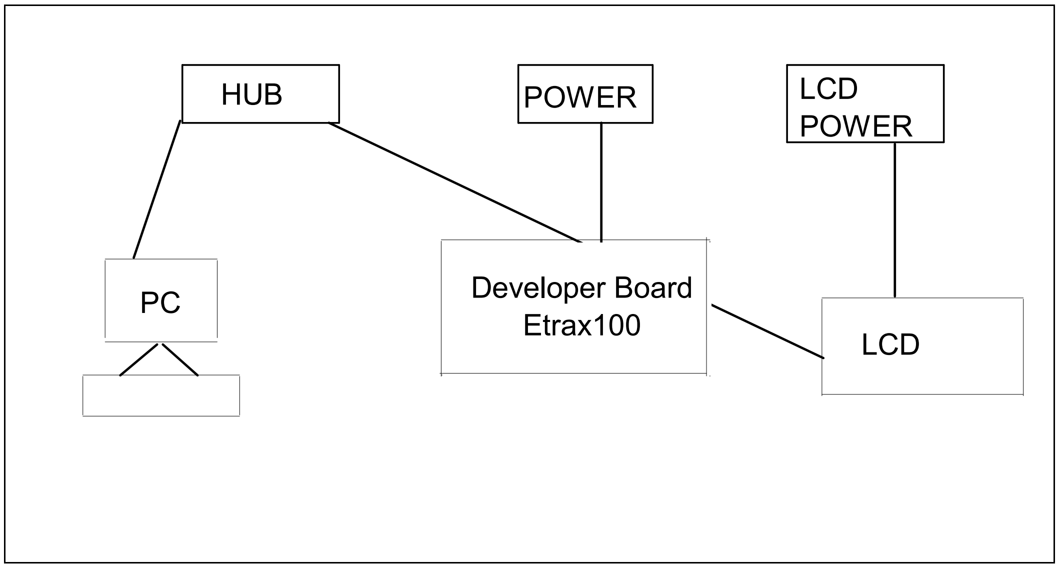
LCD 16 pins definitions:

PIN	FUNCTION	PIN	FUNCTION
1	Vss	2	Vdd
3	Vee	4	RS
5	R/W	6	E
7	DB0	8	DB1
9	DB2	10	DB3
11	DB4	12	DB5
13	DB6	14	DB7
15	NOT DEFINED	16	NOT DEFINED

LCD registers function:

RS	R/W	Operations
0	0	Instruction register (IR) write
0	1	Busy flag and address counter read
1	0	Data register (DR) write
1	1	Date register read

System Development Structure Frame:



Then we write a LCD device driver including a c program header file and *.c file. When we have finished the LCD device driver and all is right , we write an lcd application program . The application realized roll-screen displaying like common big advertising bulletin board. After that, we aslo try a program about remote control, which uses socket programming.

Appendix A : LCD device driver header file; (etrax100parlcd.c)
Appendix B : LCD device driver c program file; (etrax100parlcd.h)
Appendix C : LCD application program file; (lcdtest.c)

Appendix A: LCD device driver header file; (etrax100parlcd.c)

```
/* etrax100parlcd.c */
#include <linux/config.h>
#include <linux/errno.h>
#include <linux/kernel.h>
#include <linux/major.h>
#include <linux/sched.h>
```

```
#include <linux/malloc.h>
#include <linux/interrupt.h>
#include <linux/delay.h>
#include <linux/ioctl.h>
#include <asm/segment.h>
#include <asm/system.h>
#include <asm/irq.h>
#include <asm/svinto.h>
#include "etrax100parlcd.h"
/*   DDRAM (Display Data RAM) addresses  at different display locations.
 *   0              19
 *  64              83
 *  20              39
 *  84             103    */
#define LCD_HD44780_MAJOR 125
/* RegisterSelect (RS) = autofd (bit 18 in R_PAR0_CTRL_DATA)
 * Read/Write (R/W_) = strb (bit 17 in R_PAR0_CTRL_DATA)
 * Enable (E) = init (bit 16 in R_PAR0_CTRL_DATA)   */

#define LCD_E  0x00010000  /* LCD enable */
#define LCD_RW 0x00020000  /* LCD read/write */
#define LCD_RS 0x00040000  /* LCD register select */
#define PAR_OE 0x00100000  /* Output Enable on par port */
  /* Flags used for the flag field in the etrax100parlcd_struct */
#define LCD_BUSY 0x00000001
#define LCD_NL   0x00000002
#define LCD_MAX_ROWS 4
#define LCD_MAX_COLS 20
typedef struct etrax100parlcd_struct {
 unsigned int flags;
 unsigned char row;
 unsigned char col;
} etrax100parlcd_struct;

static etrax100parlcd_struct lcd_status;
static int lcd_write(struct inode *inode, struct file *file, const char *buf, int count);
static int lcd_ioctl(struct inode *inode, struct file *file, unsigned int op, unsigned long arg);
static int lcd_open(struct inode *inode, struct file *file);
static void lcd_close(struct inode *inode, struct file *file);
static struct file_operations lcd_fops = {
 NULL,       /* lseek */
 NULL,       /* read */
 lcd_write,    /* write */
 NULL,       /* readdir */
 NULL,       /* select */
 lcd_ioctl,     /* ioctl */
 NULL,       /* mmap */
 lcd_open,    /* open */
 lcd_close    /* release */
};
static int row_col_to_addr(int row, int col)
{
 if (row == 0)
   return 0 + col;
 else if (row == 1)
   return 64 + col;
 else if (row == 2)
```

```
    return 20 + col;
  else if (row == 3)
    return 84 + col;
}
static void display_write_instr(byte instr)
{  /* We have dual writes to R_PAR0_CTRL_DATA to make sure that the timing requirements are met */
 *R_PAR0_CTRL_DATA = PAR_OE | LCD_E;
 *R_PAR0_CTRL_DATA = PAR_OE | LCD_E | instr;
 *R_PAR0_CTRL_DATA = PAR_OE | LCD_E | instr;
 *R_PAR0_CTRL_DATA = PAR_OE | instr;
 *R_PAR0_CTRL_DATA = 0;
}
static void display_write_text(byte text)
{  /* We have dual writes to R_PAR0_CTRL_DATA to make sure that the timing requirements are met */
  *R_PAR0_CTRL_DATA = PAR_OE | LCD_RS;
  *R_PAR0_CTRL_DATA = PAR_OE | LCD_RS | LCD_E;
  *R_PAR0_CTRL_DATA = PAR_OE | LCD_RS | LCD_E | text;
  *R_PAR0_CTRL_DATA = PAR_OE | LCD_RS | LCD_E | text;
  *R_PAR0_CTRL_DATA = PAR_OE | LCD_RS | text;
  *R_PAR0_CTRL_DATA = 0;
}
static void lcd_init_int(void)
{  /* Initialization by instruction */
  display_write_instr(0x30);
  udelay(4500);
  display_write_instr(0x30);
  udelay(150);
  display_write_instr(0x30);
  udelay(150);
  display_write_instr(0x38);  /* Function set: 8bit, 2 rows and 5x8 dots */
  udelay(50);
  display_write_instr(0x04);  /* Display on/off control: all off */
  udelay(50);
  display_write_instr(0x01);  /* Clear display */
  udelay(1700);
  display_write_instr(0x06);  /* Entry mode set: inc cursor */
  udelay(50);
  display_write_instr(0x0e);  /* Display on/off control: disp on, cursor on */
  udelay(50);
}
static int lcd_write(struct inode *inode, struct file *file, const char *buf, int count)
{
  unsigned int i;
  int retval;
  retval = verify_area(VERIFY_READ, (void *)buf, count);
  if (retval != 0) {
    printk("LCD_HD44780: VERIFY_READ error: %i\n", retval);
    return retval;
  }
  for (i = 0; i < count; i++) {
      if (lcd_status.flags & LCD_NL)
      {
        if (buf[i] == '\n')
        {
          if (++lcd_status.row < LCD_MAX_ROWS)
          {
          lcd_status.col = 0;
          }
        else {
            lcd_status.row = 0;
            lcd_status.col = 0;
            lcd_ioctl(0, 0, LCD_HD44780_CLEAR, 0);
            }
        continue;
              }
            }
        if (lcd_status.col < LCD_MAX_COLS)
        {
            display_write_instr(0x80 | (unsigned char)row_col_to_addr(lcd_status.row, lcd_status.col));
            udelay(50);
            lcd_status.col++;
          }
        else
        {
             if (lcd_status.row < LCD_MAX_ROWS)
             {
               lcd_status.row++;
               lcd_status.col = 0;
             }
             else {
               lcd_status.row = 0;
               lcd_status.col = 0;
               lcd_ioctl(0, 0, LCD_HD44780_CLEAR, 0);
             }
             display_write_instr(0x80 | (unsigned char)
                             row_col_to_addr(lcd_status.row, lcd_status.col++));
             udelay(50);
           }
           display_write_text(buf[i]);
           udelay(50);
         }
        return count;
        }
        static int lcd_ioctl(struct inode *inode, struct file *file,unsigned int op, unsigned long arg)
{
 switch (op)
 {
  case LCD_HD44780_CAH:  /* Cursor at home */
    display_write_instr(0x02);
    lcd_status.row = 0;
    lcd_status.col = 0;
    udelay(1700);
    break;
  case LCD_HD44780_CLEAR:  /* Clear display */
    display_write_instr(0x01);
    lcd_status.row = 0;
    lcd_status.col = 0;
    udelay(1700);
    break;
  case LCD_HD44780_SET_DDRAM:  /* Set DDRAM address */
    display_write_instr(0x80 | (unsigned char)arg);
    udelay(50);
    break;
  case LCD_HD44780_DISP:  /* Display on/off control */
    display_write_instr(0x08 | (unsigned char)arg);
```

```
   udelay(50);
   break;
 case LCD_HD44780_LCD_SHIFT:  /* Cursor/display shift */
   display_write_instr(0x10 | (unsigned char)arg);
   udelay(50);
   break;
  case LCD_HD44780_NEWLINE:  /* Set new line mode */
   if (arg == 1)
   lcd_status.flags |= LCD_NL;
   else if (arg == 0)
   lcd_status.flags &= ~LCD_NL;
   break;
    case LCD_HD44780_GOTO_XY:  /* Move cursor to x, y */
       lcd_status.row = (arg & 0x000000ff);
       lcd_status.col = (arg & 0x0000ff00) >> 8;
       break;
    default:
       return -EINVAL;
       break;
    }
  return 0;
  }
  static int lcd_open(struct inode *inode, struct file *file)
  {
    if (lcd_status.flags & LCD_BUSY) {
      return -EBUSY;
    }
    lcd_status.flags |= LCD_BUSY;
    lcd_status.flags |= LCD_NL;
    lcd_status.row = 0;
    lcd_status.col = 0;
   *R_PAR0_CONFIG = 0x00000060;  /* Manual-mode, enable port, immediate mode change */
    lcd_init_int();
    return 0;
  }
  static void lcd_close(struct inode *inode, struct file *file)
  {
   *R_PAR0_CONFIG = 0x00000000;  /* Reset port */
    lcd_status.flags = 0;
  }
void etrax_par_lcd_hd44780_init()
{
  printk("Etrax/100 parallel LCD HD44780 driver v0.1 (c) 2000 SunWah_NanDa\r\n");
  if (register_chrdev(LCD_HD44780_MAJOR, "LCD_HD44780", &lcd_fops)) {
    printk("unable to get major %d for LCD_HD44780\n", LCD_HD44780_MAJOR);
  }
}
```

Appendix B: LCD Device Driver C Program File; (etrax100parlcd.h)

```
/* etrax100parlcd.h */
/* ioctls for etrax parallel lcd driver */
#define LCD_HD44780_CAH        _IO('l', 0x10)
#define LCD_HD44780_CLEAR       _IO('l', 0x11)
#define LCD_HD44780_DISP       _IOW('l', 0x12, char)
#define LCD_HD44780_SET_DDRAM  _IOW('l', 0x13, char)
```

```
#define LCD_HD44780_LCD_SHIFT  _IOW('l', 0x14, char)
#define LCD_HD44780_NEWLINE    _IOW('l', 0x15, char)
#define LCD_HD44780_GOTO_XY    _IOW('l', 0x16, short) /* 0xff00 = x and 0x00ff = y */
```

Appendix C : LCD Application Program File; (lcdtest.c)

```
/*lcdtest.c*/
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int pos=0;
int length=47;
int ia=0;
int endf=0;
void display(char *buf);
void adjust(char *oldbuf,char *newbuf);

int main()
{
char o_buf[]={"abcderr\nghijklmnopq\nr\nstuvwxyzasd\nasdasdasdasddf"};
char n_buf[length];
printf("long=%d\n",length);
while (!endf)
{
 adjust(o_buf,n_buf);
 printf("ia=%d\n",ia);
 display(n_buf);
}
return 0;
}

void display(char *buf)
{
int lcddev;
int m;
int size=length;
printf("size=%d\n",size);
lcddev=open("/dev/lcd",O_WRONLY);
if (lcddev==-1)
 {
 printf("Cann't open lcd.\n");
 }
write (lcddev,buf,size);
for (m=0;m<=50000000;m++)
 {}
close (lcddev);
}
void adjust (char *bufa,char *bufb)
{
int row=0;
int ib;
int hpos=0;
int num=0;
for(row=0;row<4;row++)
{
```

```
again:  if (row==1)
{
hpos=ia;
printf("hpos=%d\n",hpos);
}
        if (row==4) goto ends;
        for(ib=0;ib<20;ib++)
        {
        bufb[num]=bufa[ia];
        if (bufb[num]=='\0')
                {if (row==0) endf=1;
                else goto ends;
                }
        else if (bufb[num]=='\n')
                {
                row++;
                num++;
                ia++;
                goto again;
                }
        else
                {
                num++;
                ia++;
                }
        }
}
ends:   length=num;
  ia = hpos;
}
```

Paper Submitted by:

Hu Jiangtao

(jthu@fudan.edu.cn)

University:

Fudan University

Country:

China

KDB Debugger Enhancement -

(Add addditional functionality/operations/hardware support to
SGI kernel debugger, i.e., add support for POSIX threads)

Abstract

KDB lacks the keyboard history function, which sometimes may annoy the user, especially in the case of typing long commands. This paper analyzes the problem, designs a solution and delivers the installation guide of my patch file.

Keywords: KDB Keyboard History Function

1. Problem Statement

As we know, KDB (SGI kernel debugger) lacks some convenient functions, such as keyboard history function. It sometimes may annoy the user, especially in the case of typing long commands.

2. Analysis & Design

To add the keyboard history function, we need solve several problems. One is to distinguish the special keys, such as UP and DOWN arrow key, from the common character keys. Another is to record command history and provide a way to traverse the history. The last problem is to issue the history command as if a user types it.

2.1 Special Key Identification

After analyzing the KDB(KDBv1.8-2.4.2) source code, we find out that the implementation of KDB keyboard IO function can be divided into two parts, an architecture dependent part and an architecture independent part, just as Fig.1 shows.

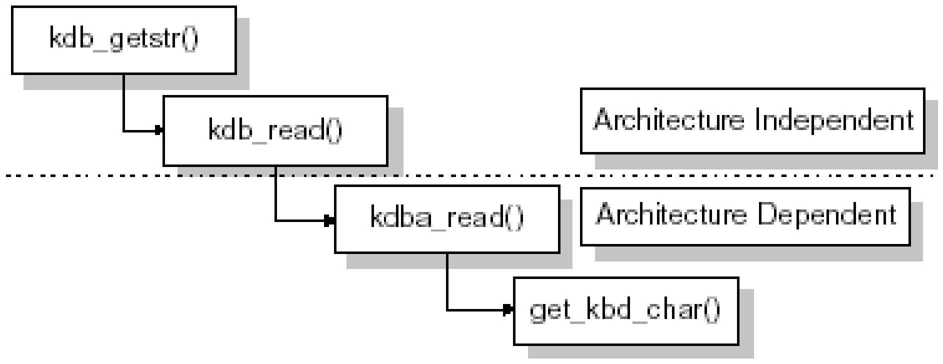


Figure 1. Functions' Call Stack Related to the Keyboard Input

The primary purpose of each function is described in Table.1.

Function Name	*Purpose*
kdb_getstr	Print the prompt string and read a command from the input device.
kdb_read	Just pack parameters to kdba_read() and do no actual work.
kdba_read	Read a string of characters, terminated by a new line, or by reaching the end of the supplied buffer
get_kbd_char	Check if the keyboard controller has a key press.

Table 1. Function Description

From Table 1, we can find out that the identification procedure of special keys happens in get_kbd_char() function. It is rather a simple job, since the scancode does the work. Another problem appears: the special key identification happens in the function of the lowest level, but logical process takes place in upper level. How could we deliver the information to the high levels? There are several ways to solve the problem, such as global variables, function return values, and additional parameters, etc. To minimize source modification and the pollution of global variables, we choose the additional parameters method. For example, the interface transformation of get_kbd_char() is shown as follows:

Function Interface Transformation

- ✍ Origin

  int get_kbd_char();
- ✍ New

  int get_kbd_char(int *key_type);

  where key_type returns the information of the pressed key type.

2.2 Keyboard History Record

Many types of data structure can be chosen to implement the command history record, such as array, list, etc. Considering the running efficiency and implementation simplicity, we choose circular array[2]. Its definition is as follows and Fig.2 shows the relation between the defined variables.

Primary Data Structure

```
#define  KDB_CMD_HISTORY_COUNT     32
#define  CMD_BUFLEN                200
static unsigned int cmd_head, cmd_tail;
static unsigned int cmdptr;
static  char        cmd_hist[KDB_CMD_HISTORY_COUNT][CMD_BUFLEN];
```

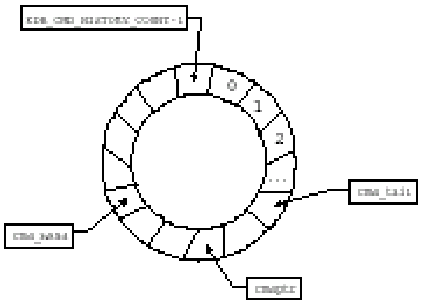


Figure 2. Command History Record

The following code indicates how to traverse the command history.

```
✍ To traverse previous command
  if(cmdptr != cmd_tail)
        cmdptr = (cmdptr-1) % KDB_CMD_HISTORY_COUNT;
  strcpy(cmd,cmd_history[cmdptr]);
✍ To traverse next command:
  if(cmdptr != (cmd_head-1))
        cmdptr = (cmdptr+1) % KDB_CMD_HISTORY_COUNT;
  strcpy(cmd,cmd_history[cmdptr]);
```

2.3 Command Issue Emulation

To emulate a history command, we only need to copy the command from the history buffer to the current command buffer and print it to the screen, as if the user typed it. A little challenge is how to print different commands in the same line as the user traverses the command history using UP or DOWN arrow key. The implementation is rather simple. Just use “\r” to locate the cursor to the head of the current line, print a blank line to erase the original content. Again use “\r” to move the cursor to the head, print the current command buffer as if the user typed it and wait for the user to input.

3. Installation Guide

Installation procedure:

1. Download linux kernel 2.4.2 source code, suppose untar files to /usr/src/linux;
2. Copy my patch file(embedded KDBv1.8-2.4.2) kbh_kdb1.8-2.4.2.patch to /usr/src/linux;
3. Change the current directory to /usr/src/linux;
4. Type "patch –p1 < kbh_kdb1.8-2.4.2.patch" and press enter key;
5. Recompile linux kernel, with configuration CONFIG_KDB;
6. Add the new kernel entry to lilo configure.
7. Restart with the new kernel and press break key to activate KDB.

That's all.

Reference

1. Linux Kernel 2.4.2 source code

2. KDBv1.8 source code

Paper Submitted by:

Mika Pruikkonen

University:

Helsinki University of Technology

Country:

Finland

Investigate Completeness and Usefulness of SCTP Functionality

Project Description and Objectives

The challenge was to investigate the completeness and usefulness of SCTP functionality by developing certain kinds of applications and/or investigating parts of the protocol and different implementations. I decided to concentrate on the first task mentioned in the challenge, i.e., developing a discovery/test tool for SCTP.

Methodology

After reviewing the current sctp implementations, the following observations were of the most significance:

- There exists two quite complete open source userspace implementations, and many more commercial ones. The oss implementations are the reference implementation and an implementation created by the Computer networking technology group of the University of Essen and Siemens. The latter seems more complete and has an active development and user community.

- The linux kernel implementation (called lksctp) is still a work in progress, and doesn't currently provide all the protocol features. There are also kernel implementations being developed for other operating systems.

- lksctp will implement the socket API defined in draft-ietf-tsvwg-sctpsocket-01.txt, which is a proposed socket API for sctp. The Essen's userspace implementation is also developing support For this API, and the development team has announced that it will provide a complete implementation of it in its next release.

The reference implementation's API and the original API of the Essen's implementation follow the application API described in RFC2960. However, since all the real kernel implementations will implement the socket API, it seemed the best choice to implement the tool using the socket API.

Since none of the oss socket API implementations are currently complete nor follow the draft, I decided to only try to develop the application For the lksctp kernel implementation.

The challenge mentions as an example a tool that can be used to investigate the features of the tcp protocol. The example tool is controlled by command-line options, but because sctp is message based, I will use a more interactive approach (the options need to be Controllable on a per-message basis.)

In the early development I had one computer running the kernel implementation, and on another I had the Essen's userspace Implementation. When I got to the level where I had a working application, I tested it by running multiple instances of it on the computer with the kernel implementation. I also tried it with the example applications that came with the userspace implementation. It seemed to be a nice tool to test other applications, and I got some more ideas from that.

Later in development I decided to install the kernel implementation on The other computer too, so that I could test how my program would work when communicating with itself on separate machines. I had to update the lksctp version and recompile the kernels several times during development because I wanted to test and use the new features that were being developed by the lksctp developers.

Results

During development I learned the limitations of the current implementation of lksctp. It doesn't currently support handling of multiple associations with association ids or proper handling of notifications, which means that handling of multiple associations is currently pretty awkward. Also, things like setting the socket options or automatically assigning the local host and port aren't yet possible. The implementation isn't perfectly stable yet either, in heavy use the kernel panics quite frequently.

It should be noted that at this stage of development it is not possible to develop an application that would work as-is in the future, since the socket API interface that lksctp provides doesn't yet comply with the socket API draft, and the draft can also still change.

However, here's what I achieved (from the tool's help):

Usage: sctpdiag [OPTION]...
sctpdiag is a sctp protocol development and analyzing tool. The commands prefixed with a slash are to be used interactively after the tool has been started. All lines not beginning with a slash are sent to the receiver identified by the prompt.

```
-H,     --local=host   | -P,    --local-port=port   specify local and
-h,     --remote=host | -p,     --remote-port=port  remote hosts
/r,     /remote host port       set/change the remote host
/s,     /stream n               switch to stream n
/p,     /ppid n                 set ppid to n
/u,     /unordered              toggle the MSG_UNORDERED flag
/l,     /loop n                 send the message n times
/w,     /wait n                 wait for n ms before each send
/g,     /generate [n] [m]       generate a message with length between
                                n and m, or exactly n, or random  if no arguments are
                                specified. This can be used together with /l, /w, etc.
/q,     /quit                   exit the tool
/?,      -?, --help             display this help
```

This is a general purpose tool, which can be used for debugging and testing other sctp applications. One can also launch multiple instances of the application and utilize them to investigate the protocol and the current implementation. When lksctp developers introduce new features, they should be quite easy to add to the application.

Sctpdiag is, in my knowledge, currently the only application using a open source sctp implementation which provides fully two-directional communication with the possibility of using multiple streams. In lksctp development, there has so far existed only one sctp application, which the developers used for testing and which enabled only either sending or receiving simple messages, and it's user interface was only appropriate for debugging the kernel implementation itself, not for helping in development of other applications or for wider testing between different implementations or inside the lksctp implementation.

An example session with two sctpdiag's running in separate machines:

```
mp@host1$ ./sctpdiag -H host1 -P 1326 -h host2 -p 2118
Welcome to sctpdiag. Type /? for help.
Ready to send...
Listening...
host2:2118 s:0> Testing..
Notification: COMM UP
Received: stream: 1, ssn: 0, body: I prefer this stream.
host2:2118 s:0> /ppid 1320
host2:2118 s:0 p:1320> Oh, that's ok. Got to run, bye.
Received: stream: 1, ssn: 1, body: Ok, bye
```

```
mp:2118 s:0 p:1320> /quit
Mp@host1$
```

```
[mp@host2]$ sctpdiag -H host2 -P 2118 -h host1 -p 1326
Welcome to sctpdiag. Type /? for help.
Ready to send...
Listening...
Notification: COMM UP
Received: stream: 0, ssn: 0, body: Testing..
host1:1326 s:0> /s 1
host1:1326 s:1> I prefer this stream.
Received: stream: 0, ssn: 1, ppid: 1320, body: Oh, that's ok. Got to
run, bye.
host1:1326 s:1> Ok, bye
Notification: SHUTDOWN COMPLETE
host1:1326 s:1>
```

The second sctpdiag in the example is still ready for new connections. It can also have multiple associations open with different peers simultaneously, and it can change the active peer with the /r command. The /l and /g options are nice if you need so send large amounts of data. The /u option isn't yet implemented in lksctp, but it will work as soon as they get it done. Currently it seems as it works but the receiver will see it as a normal, in-band message.

There are several features that could be implemented to sctpdiag, such as more detailed notifications, association handling, automatical localhost configuration (INADDR_ANY), adding and removing network interfaces, and viewing and setting socket options, but we have to first wait for the lksctp developers to get them implemented.

You can get sctpdiag from: http://www.hut.fi/~mpruikko/sctpdiag/. You can find more detailed documentation in the packages. Don't hesitate to contact me by e-mail <mika.pruikkonen@iki.fi> if you have any questions or problems.

Paper Submitted by:

Tristan Leteutre

University:

Ecole Centrale Paris

Country:

France

VideoLAN
A Complete MPEG2 Streaming Solution

I. Description of VideoLAN

The purpose of VideoLAN is to offer a complete MPEG-2 streaming solution, by broadcasting MPEG-2 high-quality streams on a Local Area Network (LAN). By "streaming solution" we mean : a way to broadcast a video stream, to decode and display it on each computer and to manage several streams at the same time.

That is why the project has been split in three parts: the VideoLANclient, the VideoLAN server and the VideoLAN channel server.

I. 1. VideoLAN client

The VideoLAN client is a software MPEG-1 and MPEG-2 decoder, designed to be fast, extensible and portable, written in C.

The client is able to read streams from a hard drive, a DVD or a network input. We wanted it to support differents display modes, such as X11, Linux framebuffer, SDL or GGI. Moreover, it supports video card hardware acceleration.

Installed on spectators computers, the VideoLAN reads the MPEG 2-TS network stream broadcasted by the VideoLAN server, decode it and display it on the screen.

I. 2. The VideoLAN server

The purpose of the VideoLAN server is to broadcast several MPEG 2-TS streams through a network. The inputs can be a hard drive, a DVD, a satellite card or hardware real-time encoded video.

The VideoLAN server is written in C++.

I. 3. The VideoLAN channel server

The VideoLAN channel server allows each spectator to switch between the network streams, by simply selecting a channel like with a television. When a client wants to change channel, it makes a request to the channel server which acts on the network (with SNMP, for example) so that the client should receive the right channel.

II. Methodology

The developpement of the VideoLAN client has included a lot of researches concerning the MPEG decoding and optimisations. The program had been several times re-written, to design a full modulable solution.

The whole network solution, especially concerning our LAN at CentraleParis, has been very much studied to make best profit of our network hardware, and to be able to use VideoLANon it.

The VideoLAN team, composed of students of Ecole Centrale Paris, France, wanted to use classical tools and organisation of OpenSourceproject, to ensure a quality of developpement.

First, we released the VideoLAN client under the GNU/Linux licence in February 2001, the Channel Server in may 2001 and the Server in October 2001.

Then, we use the project tool "Concurrent Version Service" (CVS) and mailing-list tools such as "listar", to improve the communication between all the developpers.

We also created a Web Site <http://www.videolan.org/> where world-wide developpers can be aware of our progresses. They can also download the sources of the project, and get infos such as news, TODO lists and browse the CVS-Web.

The project also gave birth to sub-projects, such as mpeg2dec and libDVDcss. We use external libraries like libDVDread and ucd-SNMP, and drivers form LinuxTV.org for specific cards. Our implication and contribution to the OpenSource world is one of our motivations.

III. Results

Today, the VideoLAN client is one of the best DVD Player and MPEG2 decoder, and has many features in the interface such as chapiter or title selection, fast play or rewind. With all MMX optimisations and video optimisations, are able to play the 25 fps of a DVD on a PIII-450MHz. We also support MPEG 1 streams reading, VideoCD (VCD), MPEG-2 Sytem (PS and TS).

The Channel Server support the VLAN (Virtual LAN) solution to change channels: Streams are broadcasted in several independant VLANs, and the Channel Server changes client's VLAN depending on channel. We are currently studying for multicast support.

The VideoLAN server is able to read from several inputs: files, DVD, a satellite card and a real-time encoding card.

The last tests proved that this solution works: we broadcasted a DVB satellite channel, a DVD and a show recorded in real-time with a video camera on the network of the Centrale Paris Residence, France. Althougth this network is a 10-100MBits Ethernet with an 155MBits ATM backbone, the 1000 students of this residence watch one of this three channels, and change as the wanted.

That is why the whole VideoLAN project is far the most advanced MPEG-2 streaming solution in the world. And with the help of world-wide programmers and financial partners, we hope that this GNU/Linux project will ensure its success.

Paper Submitted by:

Peter Rost

University:

Berufsakademic Dresden

Country:

Germany

Projekt:	**X3DViewer**
Name:	Peter Rost
School:	Berufsakademie Dresden, Germany
Category of Project:	Accessibility Enhancements (#3)

1. Objectives

During the last years of the computer evolution, the PC is more and more used as a plattform for multimedia applications. The development of multimedia-applications for Linux, which are able to compete with Windows app's, began only 2 or 3 years ago. This was supported by several multimedia-projects and the growing 3D-Support (OpenGL, DRI). One part of the 3D-Graphic is the WWW-standard X3D.

The intention of this project is to write a C++ – application to visualize X3D-files. The program should not only work as a standalone component but also as a browser Plug-In (Konqueror, Netscape, etc.). Besides the full support of the X3D vocabulary, also additional functionality like manipulation and scene-interaction will be made possible.

Another intention of this project is to provide the program for multiple platt-forms. The program is based on the free Qt-library from Trolltech. This library is available on several operating systems, like Windows, MacOS and Linux. There is also a Qt/Embedded-Version and an OpenGL/Embedded-version available, so that a porting to mobile devices, for example PocketLinux - handhelds, is easy to realize. But this field of application requires a complex ressource management, which is one part of this project.

2. Methodology

As already mentioned, the program is based on the free Qt-library and uses the QGLWidget-Interface for OpenGL-Calls. Qt is also the basis for the free KDE-Project and the free IDE KDevelop gives a good Qt-Support, why this IDE is used for the project. KDevelop also cooperates well with the tools 'make', ,cvs' and 'g++' , which are available for free also on other platforms. For creating the class structure the free UML-tool Together is used, because this tool provides the biggest functionality in comparison to other UML-tools. By using this tool a clearly structured und easy understandable library was developed, therewith also programmers who are not members of a linux comunity can participate in this project. All implemented classes are derivated from the base class X3DBase,

1

which provides the full functionality to create and manage object trees. So every programmer can work on his part without paying any attention how the object management is solved. The build-up of the object tree respectively the objects in the tree is solved by overloading the SAX-like XML-interface QXMLDefaultHandler/QXMLSimpleReader. This interface definition make the read in of wellformed and valid xml-files possible. A test if it is valid, is easy to implement.

To implement the JavaScript-Interface the SpiderMonkey-Engine of the mozilla-project will be used. The engine is written in C and so it can be integrated in the X3DViewer. An integration is planed and the documentation papers are supplied but it isn't realized yet. One of the most important innovations of VRML97 was the option to define dynamic elements, except of JavaScript. To define dynamic scenes, you need sensors, interpolators and also routes, which link sensors and interpolators. These X3D-items are also derivated from the base class X3DBase. They overwrite the methods *getAttribute()* and *setAttribute()*. If an event occures, it's being handed over between the objects of the event chain by get-/set-calls. Timesensor, all interpolators and route are already realized using this technique.

Another feature of the program is to visualize shadows in real time. This option is already implemented by a self written library. It is built similar to the RT-Shadow library written by Mark Kijlgard, mere it is developed object orientated and similar to X3D. The library uses the Depth-Test of OpenGL to determine shadow regions and mask them into the stencil buffer. By using the stencil test to render the scene, shadowed objects can be excepted from the normal (lightened) render pass. To create core- and half-shadows, the scene will be rendered several times into the accumulation buffer. Also very important is the selection of objects, for example to focus an object. This is realized by a dynamic and unique numeration of the objects and by using the OpenGL selection buffer, where you can give every object a name represented by a number. Because of the unique numeration you can determine the selected objects, which ID's are stored in a stack. Furthermore you can define a sensibility, telling the program how to react on a pick event, by using the *gluPickMatrix()*-function.

X3DViewer should not only be used as a standalone component, but also define several interface definitions. These definitions should enable the program to be used by other applications. One of these interfaces is a database interface, which is based on the ODBC-drivers. These drivers are available on many platforms, like Windows or Linux (unixODBC) and supply a high database independency. But ODBC is not the fastest way to connect a program with a database, why also several databasedependent interface will be implemented, for example to connect a DB2-, Oracle-, Informix- or Sybase-database without using ODBC. Databases have the properties to control parallel processes, to give userrights and to define views. These properties provide an access over distributed networks (for example by mobile computers) and the access management. Databases provide a high functionality, so you can regroup your scene for example. XML, which is the basis of X3D, and databases, both divide structure and content of a document, wherefore an import or export of X3D-scenes from or to databases is easy to realize, because some of the common databases offer several functions for handeling XML-data. Another application area for visualization programs is simulation software. To offer those programs a possibility to communicate with X3DViewer, a clear TCP/IP-communication protocol will be implemented. At this juncture, this protocol will be defined in cooperation with a simulation project of the Fraunhofer IWS[1] and in the near future integrated in the program. The last interface which this program should provide is the posibility to use it in a internet browser. At this moment the integration of X3DViewer into the KDE-browser Konqueror is still in progress and after it is successfully finished the X3DViewer will be restructured as a Netscape plug-in. For the Integration into the Konqueror it is necessary to use the class KParts::BrowserExtension and to derivate your own class from it. To integrate the viewer into Netscape it is necessary to use the QNP* - classes of the QT-library, which provide several classes to define a plug-in interface. Both methods are still being worked out by using the Qt-documentation and the KDE-Developer site (developer.kde.org) and will be implemented in the near future.

3. Results

Currently the program is able to read a main part of geometry data, group definition tags, interpolators and sensors. Other tags will be read in but not evaluated. The problems with reading X3D-files and the build-up of object trees is solved. Also solved are problems with the visualization of appearance tags, like transparency. The options to calculate and visualize shadows in different steps (core and half shadow) and the selection of objects are implemented too. The interface definition to embed the program into other applications is worked out but not implemented, for example a database interface, a TCP/IP-interface definition and the plug-in feature. fOne of the first tests was to read in a 1MB simulation file of the simulation program 'Simulation3D' programed by Mr. Heller at the Fraunhofer IWS. The X3DViewer needs 34 seconds for read-in, parsing and creating the object tree and the OpenGL lists. The scene was rendered with 17 to 20 frames per seconds on a Celeron 400 CPU with 256MB of RAM and a Riva TNT2 graphic card. To increase the number of members of the project, the website people.freenet.de/peter.rost/linux/ was created. The site informs about the intentions, ideas, methods and results of this project.

[1]Fraunhofer Reasearch Center for Beam- and Materialtechnologies in Dresden, Germany

Paper Submitted by:

Klaus Wehrle

(klaus@wehrle.de)

University:

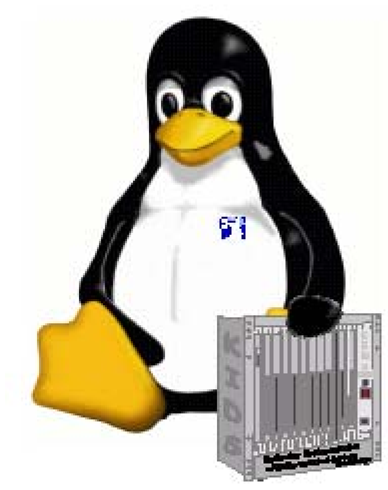
University of Karlsruhe

Country:

Germany

KIDS

KARLSRUHE IMPLEMENTATION
OF DIFFERENTIATED SERVICES



A Flexible and Modular Framework for Individual Traffic Control in the Linux Kernel

Team: Klaus Wehrle, Jerome Freilinger
klaus@wehrle.de, s freili@ira.uka.de

1. Introduction

In the last few years, the Internet Engineering Task Force (IETF) spent a lot of research efforts in investigating several kinds of mechanisms to provide better services than the traditional best effort delivery (the so called *Quality of Service – QoS*). Many applications would benefit from certain guarantees from the network, e.g. video streaming, voice over IP, etc. These guarantees can be assured by extending Internet routers with certain mechanisms, like different scheduling of queues, metering and dropping of network packets. Especially the Differentiated Services architecture developed by the IETF [BBCD+98] is designed of small building blocks (Per Hop Behaviors) composed to offer quality based services. To implement and use this QoS features it is extremely important to have an implementation architecture that offers the possibility to build new QoS models rapidly from elementary entities. The presented Karlsruhe Implementation of Differentiated Services (KIDS) for the Linux OS fulfills these requirements. It can be used in Software Routers based on Linux and in low and middle-end routers based on embedded-Linux. It was also successful used in a pervasive computing environment using a Compaq iPaq.

2. Objective of the KIDS framework

The objective of KIDS is to offer elementary QoS modules for the Linux protocols, which can be combined and linked together to common QoS elements, like traffic shaper, token bucket, classifier, etc. The suite also offers a variety of queues and scheduling mechanisms like priority queueing, weighted fair queueing, round robin, etc. In the following, these QoS elements are called *Behavior Elements*. In the next section a more detailed classification is given. The main principle in building this implementation architecture was the possibility to build new QoS behavior quickly from the existing pool of elementary Behavior Elements. This can mostly be done by varying the elementary modules and connecting them in a special manner. The following example should motivate the architecture of the proposed model: A token bucket is a widely used model to meter a certain network flow and to monitor its conformance to Service Level Agreements (SLAs). Traditionally a token bucket is metering incoming flows to their conformance. If the negotiatedrate is not exceeded, the packets will be forwarded – otherwise they will be discarded. This method of metering a flow is well known, but in some scenarios (i.e. AF PHB in DiffServ networks) the packets should not be discarded. They should either be marked with a lower priority and enqueued in an alternate queue which is served with a lower priority. Or in other scenarios, two token buckets should be combined to control the peak rate and the average rate or the token bucket should be packet-oriented instead of byte-oriented. With existing implementations, new token bucket modules, a marker a priority queueing module, etc. have to be implemented. Secondly, it is always a problem to integrate QoS elements into the right position within the protocol stack. For instance, it is important whether a token bucket is working on the IP layer – before the routing has been done – or at the output queue of a certain interface. In the first case all packets forwarded by IP will be considered in the token bucket meter, whereas in the latter case, only the packets leaving on one interface will be metered. In a third case only the packets leaving a host should be considered. It is obvious that in a protocol like IP, a lot of possible places to integrate QoS behavior can be identified (as shown in section 2.1). As a result of this, a fast development of modules for investigating new network behavior, as i.e. new QoS behaviors, is very time-consuming with existing operating systems, since they have mostly implementations for specific QoS architectures [BSSo00, AlSK99]. Each time, new implementations have to be developed. The reuse of existing implementations is very complex.

The presented modular architecture with its elementary QoS modules, and the individual linking of them, would solve these problems and allow to build immediately any QoS behavior for an Internet router or host – mostly without implementing new models. The existing pool of elementary QoS behaviors, and its smart integration into the Linux implementation of Internet Protocol, offer on the one hand the examination of real IP behavior and on the other hand the possibility to build and evaluate rapidly new QoS behavior in real systems. In the next few sections the basic architecture of the KIDS (Karlsruhe Implementation of Differentiated Services) QoS architecture will be presented. First the principle of `Hooks` is explained, which are strategic points for including QoS elements into protocol stacks. Subsequently the five different kinds of `Behavior Elements` and rules to concatenate them will be introduced. It should be mentioned that the pre-sented KIDS architecture do not need any modification within a standard Linux Kernel. Standard interfaces like traffic control and Netfilter have been chosen to create the concept of KIDS' Hooks. Figure 2 illustrates the KIDS architecture by an example. Three service classes should be distinguished: A Premium class, offering a high priority service with low delay. The flows of the Premium class will be metered by a Leaky Bucket and shaped at the output interface. A second class should offer a better service than Best Effort with a statistical guarantee of bandwidth. This will be achieved by a weighted fair queueing scheduler. The metering will be done by a token bucket. Non conforming packets will not be discarded, but degraded to the Best Effort service, which builds the third service class. The classification to the three service classes is done by a multi-field classifier. This example is a possible implementation of the well-known 'Two bit architecture', which is described in details in [JaNZ99]. To keep the example simple, only the Layer2-Hooks of interface `eth0` and IP-Forward are shown.

2.1 Protocol Hooks

When Qualtiy of Service behavior should be introduced into an existing protocol, one basic problem is the point at which the protocol is extended with the new Behavior Elements. E.g. regarding the Internet Protocol, five strategic points can be identified – to realize these hooks, the Netfilter interface was used and extended to a common interface for KIDS behaviors. But KIDS-Hooks are not limited to Netfilter hooks, they can also be realized in every network protocol by simply inserting a simple function.

Hooks can be distinguished in the set of packets passing the point they have been inserted, e.g. the `IP Post Routing`-Hook represents the set of all packets leaving the IP node on an interface - whether they have been forwarded, or created from the host:

- `IP PRE ROUTING`: All packets arriving on a network interface will pass this hook before routing is processed. Consequently all incoming packets will be processed by the Behaviors attached to this hook.
- `IP Local In`: All packets arriving for the local host will pass this hook after the routing is processed and before they leave IP for the upper protocols.
- `IP Forward`: All forwarded packets will pass this hook after the routing. Consequently it is the right point to perfom QoS mechanisms on routed packets.
- `IP Local Out`: This Hook is suitable for all packets leaving from upper layers, before routing is processed.
- `IP Post Routing`: The last Hook can be used to perfom any action on all packets leaving the host on a network interface card, whether they are forwarded or created from upper protocol layers.

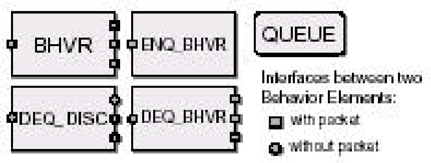


Figure 2: Five different classes of behavior elements

For each network interface, three additional Hooks can be identified: `L2 Input xxL2 Enqueue xx` and `L2 Dequeue xx`, where `xx` is the name of the network interface card in the Linux OS. The first one is located at the entry of a packet in the receive-function and the other two are located around the output queue of a network interface card. These Hooks are the right point to add specific behaviors operating on outgoing queues, like priority queueing, traffic shaping,etc. (refer Fig. 2).

In this description the focus is only on the Internet Protocol Version 4 and the underlying layers. In other protocols like IPv6, TCP, UDP, etc., also Hooks are integrated yet or can be added easily at strategic places to integrate QoS behavior in the network stack.

As described above, a Hook is a place within a protocol where QoS behavior can be added. The Behavior Elements included at such a hook are elementary models offering a certain behavior. They will be described in the following section.

2.2 Behavior Elements

A Behavior Element (BE) is comparable to a black box, which offers a specific basic *behavior*. A BE consists of one in-gate,*n* out-gates and a certain processing behavior inside. At the in-gate a packet enters the box and receives a certain manipulation inside the module. Dependent on the calculation within the box, the packet leaves on a certain out-gate. Behaviors can be concatenated after each other. Consequently, the treatment a packet receives within a BE decides which way it will proceed and which quality it receives.

Two kinds of gates (interfaces) of Behavior Elements and Hooks can be distinguished: *packet-gates* (abbreviated as) and *non-packet-gates* (□). The main difference between them is, that between two packet-gates IP-packets are exchanged, and between non-packet-gates only messages to request (dequeue) packets are exchanged.

The principle rule in the KIDS architecture is, that only gates from the same kind can be connected to each other. The two different types of gates and the interaction between them are described more detailed in section 2.3.

As mentions before five kinds of Behavior Elements can be distinguished (ref. Fig. 2.2). In the following they will be introduced in detail:

- **(conventional) Behaviors (BHVR)** are elementary QoS elements which operate on IP packets. As shown in Fig ure 2.2, a Behavior has only one in-gate and up to *n* out-gates, where *n* depends on the particular Behavior. E.g. a Token Bucket has two out-gates, one for conform packets and one for non-conform packet. Behaviors can be interconnected between one another without fulfilling other requirements.

Example Behaviors are Token Bucket, Marker, Dropper, Classifier, Random Early Detection (RED), etc.

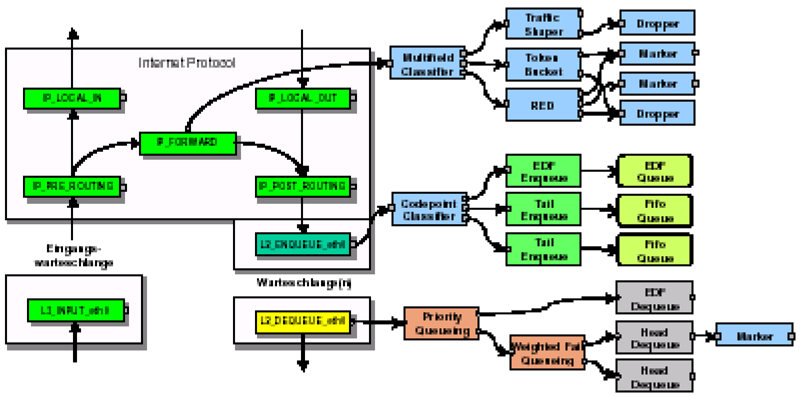


Figure 1: Example configuration of basic QoS Behavior Elements

- **Queue (QUEUE)**: Queues are well known packet queues. Packets can only be enqueued and dequeued with the appropriate kinds of Enqueue and Dequeue Behaviors. Several types of Queues have been implemented in KIDS, e.g. Fifo-Queue, Shaping-Queue, EDF-Queue, etc.

- **Enqueue Behavior (ENQ BHVR)** are specialized Behavior Elements for enqueueing a packet into a queue. The queue is identified by its name and an according Enqueue Behavior should be used. One special characteristic of an Enqueue Behavior is the missing out-gate. Whether the packet is inserted into the queue, or it has to be dropped. Enqueue Behaviors can be connected to out-gates of any Behavior module. The detailed procedure of exchanging messages and packets between Behavior Elements is described in section 2.3.

- **Dequeue Behavior (DEQ BHVR)** can be used to dequeue a packet from a certain queue. E.g. a `Fifo Dequeue` module removes the first packet from the named queue and sends it to its out-gate. Dequeue Behavior modules can only be connected to an □-gate of a L2 Dequeue-Hook or a Dequeue-Discipline. After a Dequeue Behavior all kinds of Behaviors can be connected to the packetgate.

- **Dequeue Discipline (DEQ DISC)**: A Dequeue Discipline is a strategy to choose the next Dequeue Behavior for serving a queue. Dequeue Disciplines are playing a very important role in reaching different service classes within a network. Examples for Dequeue Disciplines are Priority Queueing, Weighted Fair Queueing, Round Robin, etc.

2.3 Interactions between Behavior Elements

With the just presented five types a simple mesh of Behavior Elements can be built and connected to Hooks. But it is important to understand how a packet traverses this mesh and which interactions can occur between the Behavior Elements. The following section describes the sequence of events on two example concatenations of Behavior Elements.

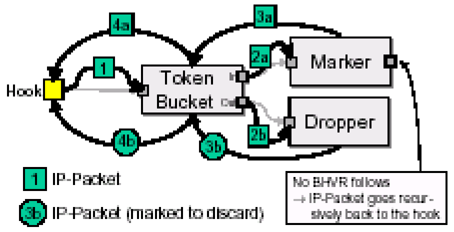

Hook
Token Bucket
Marker
Dropper
1
2a
2b
3a
3b
4a
4b
1 IP-Packet
3b IP-Packet (marked to discard)
No BHVR follows → IP-Packet goes recursively back to the hook

Figure 3: Interactions on a □-junction between Behavior Elements

Interactions

between packet-gates (□ - junctions):

Several Behavior Elements with packet-gates can be arranged into one new model to create a new QoS Behavior. At the connections between the -gates, IP packets are exchanged. Figure 2.3 shows an example. The Hook sends an IP packet to the Token Bucket-Behavior. When the module has completed its operations on the packet, it will be send further, when a module is connected on the dedicated out-gate. That means in the example, that a SLA-conform packet (case *a*) will leave on the `In`-gate (In-Profile) of the Token Bucket; If it is not conform the packet will leave on the `Out`-gate to the Dropper. If no module is connected to a Behavior on the dedicated port, or the Behavior has no port, the packet will be sent back to the previous module where the packet came from.

One can see, that a packet first traverses a chain of Behaviors and then goes recursively back to the Hook, where the normal protocol processing will be continued.

This is the normal procedure, but there are two possible exceptions. The first is when a packet has reached an Enqueue Behavior, which will enqueue it into a Queue. The second exception is a Dropper that marks the packet for discarding. In both cases, the modules will send back an IP packet to the hook, but with a return code like *Packet-*

Enqueued or *Discard-Packet*. The discarding of a packet will be done in the Hook, because all Behaviors between the Hook and the Dropper have to be informed about the loss of the packet. E.g. a token bucket has to put back the tokens of the discarded packet into the bucket, because it has not consumed them.

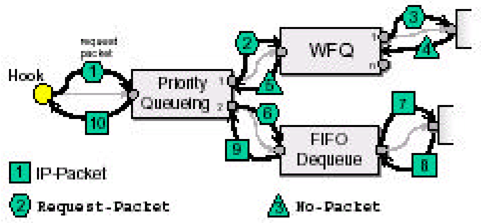


Figure 4: Interactions on ○-junctions between Dequeue Disciplines and Dequeue-Behaviors

Interactions at non-packet-gates (○-junctions):

On ○-junctions (between Dequeue Disciplines or between Dequeue Disciplines and Dequeue Behaviors) no packets will be exchanged. Dequeue Disciplines will first decide which Dequeue Behavior will be asked to dequeue a packet from a queue. This mechanism will be triggered from the Dequeue-Hook sending out a Request-`Packet`-message to the first dequeue discipline or directly to a Dequeue Behavior, if no scheduling algorithm is used.

A Dequeue Discipline decides on which of its out-gates the Packet-Request will proceed. Any combination of Dequeue Disciplines can be built, but finally a Dequeue Behavior has to be called. In Figure 2.3, the Priority Queueing module first calls on the gate with the highest priority. The Dequeue Discipline connected to that gate proceeds with its own scheduling mechanism. In Figure 2.3 theWeighted Fair Queueing module proceeds on gate 1.

Each possible chain of Dequeue Disciplines has to conclude with a Dequeue Discipline which executes the Packet-Request by dequeueing a packet from the Queue. On success and if a Behavior is connected, the packet will proceed on the □-gate of the DequeueBehavior. This follows the same procedure as described previously about □-junctions. When the Dequeue Behavior receives back the packet, it is sent recursively back trough the Dequeue Disciplines to the Dequeue-Hook.

If no packet can be dequeued, the Dequeue Behavior returns a No Packet-ID to the previous Dequeue Discipline, indicating that the dequeue-operation failed. Then, the Dequeue Discipline can choose – according to its algorithm – another -gate to request a packet or it returns the No Packet-ID to its predecessor. On a successful dequeue, the hook starts the transmission of the packet on the network interface.

As mentioned above, the dequeueing is triggered by the Dequeue-Hook. Normally, Packet-Requests will be initiated by the network interface, when it has finished the transmission of the previous packet and is now able to transmit the next packet. But if the interface has been idle for a while, it would not start a new request. Therefore, the Enqueue-Hook can initiate a Packet Request, when he just inserted a packet into one of the output queues of the interface. Such an indication only starts a when the NIC is in idle state.

2.4 Specification language

To manage the creation, destroying and concatenation of Behavior Elements a special language has been developed. With this commands, arbitrary Quality of Service Behavior can be build from existing elementary modules. The configuration can be made manually be using the specification language or by using a graphical user interface wich allows the management of the QoS Behavior Elements by simple drag 'n drop. The GUI is also able to load the current KIDS configuration from a router and able to submit the change that has been done by the administrator. A sample screenshot of kidsconfig is shown in appendix A.

In the following the commands are shortly described:

CREATE ***bhvr class bhvr type bhvr name***
DATA {***private data***}***** **END**

Creates a Behavior (QUEUE, BHVR, ENQ BHVR, DEQ BHVR, DEQ DISC) from a certain type (Dropper, Fifo Queue, Marker, etc.) with the given *bhvr name*.
The first three parameters are equal to all possible Behaviors. This is followed by an optional part, where Behavior specific parameters can be set, as for example `RATE = 64.000kbps`.

CONNECT ***bhvr class1 bhvr name1*** **TO**
{**(*bhvr class2 bhvr name2 gate*)** | **(HOOK**
***hook name*)**}**+ END**
Appends the Behavior `bhvr name1` after Behavior `bhvr name2` or after the given Hook. Because the second Behavior can have more than one output gate, it has to be specified on which gate the second Behavior Element should be connected.

CHANGE ***bhvr class bhvr name*** **DATA (*private data to be changed*) END**

Changes the private parameters of the given Behavior. Only the listed parameters will be changed.

DISCONNECT ***bhvr class bhvr name gate*** **END** Removes the connection, that leaves on the gate of the Behavior.

REMOVE ***bhvr class bhvr name*** **END** Removes a Behavior from the Kernel. This is only possible, if the Behavior has no more connections.

3. Performance Evaluation

This part shows that the overhead of fine granular QoS modules is not reasonable higher than with *monolithic* QoS implementations. It is obvious that this architecture has, due to the simple and elementary QoS models, a certain overhead to switch between consecutive Behavior Elements. As measurements have shown the handover from one Behavior Element to the next takes only 17 CPU cycles at an average. The actual value varies in cause of caching effects and memory access. This shows that it is not very time consuming to build a QoS behavior from many basic elements instead of unsing monolithic and unflexible QoS implementations.

Another performance problem of standard PC hardware is the missing of precise clocks and timers to offer a highresolution traffic shaping in software routers. Normally, standard PC hardware is only able to shape traffic with a precision of 100 Hz to 1000 Hz. At the University of Karlsruhe a new technique have been implemented to realize kernel timers up to a resolution of 1.000.000 Hz. This timer, called *UKA-APICTimer* [WPRM+01] has been used in the KIDS architecture to realize a high precise Earliest Deadline First queue. This queue can be combined with some other basic modules to a Leaky Bucket or a traffic shaper.

And the best proof of the performance and correctness of the KIDS implementation is, that it runs, runs and runs. It was tested over a long time period and also in routers with Gigabit-Interfaces forwarding packets up to 800 Mbps with quality of service support. It was also successfully ported to be used in a Compaq iPaq.

4. Conclusion

In this contribution, an implementation architecture for the simple and rapid creation of Quality of Service mechanisms in the Linux Kernel (v2.4) has been presented. The modules can easily be inserted into the TCP/IP stack of the Linux OS without the need for any kernel modification.

The creation and evaluation of Quality of Service mechanisms can easily be done by using the elementary QoS models and concatenating them in the desired way. Common models for queue scheduling (priority queueing, weighted fair queueing, round robin, etc.), metering (token and leaky bucket), classifying (multi-header-field, DS-codepoint, etc.) and forming of data flows are provided.

References

[AlSK99] W. Almesberger, J. Salim and A. Kuznetsov. Differentiated Services on Linux. Draft-almesberger-wajhakdiffserv-linux-00.txt, February 1999. Internet Draft.

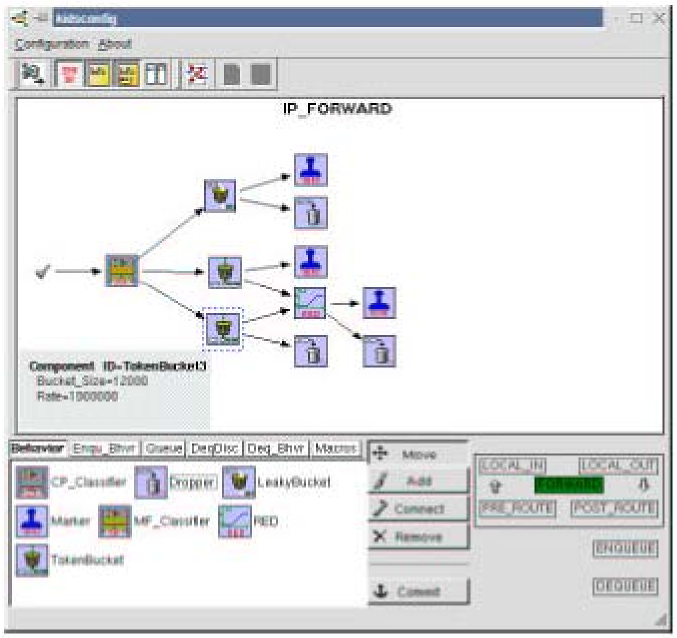
[BBCD+98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang and W. Weiss. An Architecture for Differentiated Services. RFC 2475, December 1998.

[BSSo00] T. Braun, M. Scheidegger, G. Stattenberger and other. A Linux Implementation of a Differentiated Services Router. Proceedings of Networks and Services for Information Society (INTERWORKING'2000), October 2000.

[JaNZ99] V. Jacobson, K. Nichols and L. Zhang. A Two-bit Differentiated Services Architecture for the Internet. RFC 2638, July 1999.

[WPRM+01] K. Wehrle, F. Phlke, H. Ritter, D. Mller and M. Bechler. *Linux Netzwerkarchitektur (Linux Networking Architecture)*.

Addison-Wesley, December 2001



Component ID=TokenBucket3
Bucket_Size=12000
Rate=1000000

Figure 5: Graphical interface for configuration of KIDS Behavior Elements and KIDS Hooks

Paper Submitted by:

UWE Walter

(walter@tm.uka.de)

University:

University of Karlsruhe

Country:

Germany

μ-second precision timer support for the Linux kernel

Uwe Walter, Vincent Oberle
walter@tm.uka.de, vincent@oberle.com

Institute of Telematics
University of Karlsruhe
D–76128 Karlsruhe, Germany

1 Introduction

Nowadays with rising demand on speed and precision in hard- and software technology, a lot of applications need the possibility to execute actions at exactly defined moments of time. Unfortunately the standard timer offered by the Linux kernel for such purposes does not adequately fulfill these requirements because its resolution is far too low. So a new and more precise means for timed execution of programmed functions is desirable.

2 APIC-Timer

Since the first IBM PC all compatible personal computers have been equipped with a programmable interrupt controller (PIC 8259A) which offers the possibility to generate interrupts with a configurable frequency. The Linux kernel uses this periodical moments of time to schedule and execute timer functions. Unfortunately the maximum frequency supported by the standard PIC is 8192 Hz, which would result in interrupts generated each 122 μs. However most of the issued interrupts are wasted if no timer function is attached to them and occupy the system with a tremendous unnecessary workload.

To reduce this inefficiency, the Linux kernel programs the PIC on i386-based architectures with a default value of 100 Hz. Consequently the time interval between two generated interrupts is 10 ms, which automatically results in the same value for the resolution of timer functions execution. By using this standard means it is therefore not possible to issue programmed actions at points of time more accurately than with a worst-case deviation of 10 ms (which results in a mean error of 5 ms).

But help is on its way: With the Pentium processor family, Intel introduced the Advanced Programmable Interrupt Controller (APIC) to replace the old PIC. This APIC is referred to as the local APIC to distinguish it from the I/O-APIC, which is an external controller to manage interrupts on SMP (symmetric multiprocessor) systems. There is no I/O-APIC on UP (uniprocessor) boards but since Intel's P54C all CPUs contain an on-chip local APIC.

Nevertheless, the local APIC is generally disabled on P5 chips and cannot be enabled by software. Only P6 chips (i.e. starting with Pentium Pro) allow to enable the local APIC by software means.

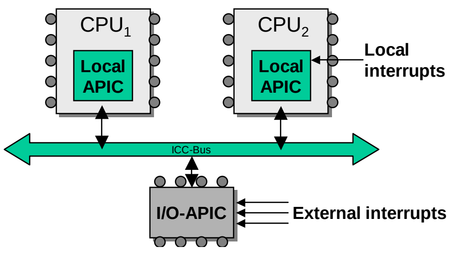


Fig. 1. Local APIC and I/O APIC in P6-PCs

Figure 1 shows the architecture used in i386-compatible PCs. It shows the case of a multiprocessor board, where the I/O APIC is integrated. Nevertheless, in the case of a uniprocessor system the local APIC exists also on board of the CPU.

As described in [1] the local APIC unit contains a 32-bit timer accessible by the CPU. The timer can be configured by a special timer register. The time base can be derived from the processor's bus clock and can optionally be divided. It supports one-shot and periodic modes. For the problem of issuing non-frequent timer functions the one-shot is interesting, because it does not put unnecessary workload on the system. One-shot mode means, that an initial value is copied into the register and counted down to zero with bus speed and an interrupt is generated when the timer reaches value zero.

Based on this APIC timer a kernel module was implemented which exports the basic functionality of the APIC timer to other modules. It is realized as a linked list of events, ordered according to the expiration time. This allows to issue the timers as fast as possible without a search through the complete list. If a timer expires, a user-defined function is invoked, like it would be expected from any normal system timer.

The minimum resolution with this APIC-based timer is in the magnitude of microseconds. As the bus speed of modern x86-based PCs is at least 100 MHz, the minimum resolution should be 0,1 microseconds (1/100 MHz), but due to the calculation time needed for switching to the interrupt service routine (saving context information etc.) the achievable accuracy is about 1 microsecond. Consequently this is a 1,000 to 10,000 times higher precision than the default PC timer.

3 Implementation

The APIC timer consists of two patches and a module for the standard Linux kernel tree. The module can be loaded and unloaded dynamically during runtime and handles all administrative tasks, i.e. adding, modifying and deleting timers. One objective during the development has been to put as much functionality as possible into the module and not into the kernel itself to keep maintenance and overall view easy.

Unfortunately patching the kernel is still necessary. One patch is needed to activate the local APIC on uniprocessor systems and the second patch integrates the necessary interfaces into the kernel. Without these modifications it is not possible to register an interrupt handler for the APIC timer interrupt, because the default handler smp_apic_timer_interrupt is hard-wired. This is the reason why the APIC timer cannot be used in SMP systems, because then its functionality is needed for synchronization between the different CPUs.

The APIC timer module consists of program functions and interfaces for the users of the APIC timer. The difference between the usage of the standard kernel timer and the APIC timer was kept to a minimum.

Registered timer handlers are stored in a double linked list, similar to the storage of the standard timer in the Linux kernel. The list elements are structured like this:

```
struct apic_timer_list {
        struct apic_timer_list  *next, *prev;
        unsigned long long      expires;
        unsigned long           data;
        void                    (*function)(unsigned long long, unsigned long)
};
```

– next und prev are used to link the apic_timer_list-entries.
– The variable expires holds the value of the timestamp-counter-register (TSC) at which the timer function shall be called and executed. (The TSC is a processor register which is incremented by one at each processor cycle. It can be compared to a discrete clock with processor speed precision.) The necessary conversion into bus cycle accuracy is done automatically by the APIC timer module. To improve performance, the linked list is ordered according to execution (expires) time.
– data is simply a pointer which can point to private data of the registered timer handler function. This is extremely useful if multiple instances of a handler function are called and need to keep private data.
– function is the pointer to the timer function which shall be called when the timer is due. After the expires moment of time has been reached function is called with the additional parameter data (see above).

When the local APIC issues an interrupt, the function do_apic_irq is called, which, after a few administrative tasks, itself calls run_apic_timer. That function compares the actual moment of time to the planned expires-value, because that difference will be used for error correction of the following timers to improve accuracy. After that, the expired timer is removed from the list and its handler function is invoked. When their job is done, the local APIC will be programmed for the next due timer in the list.

The APIC timer module exports the following functions (which are quite similar to that of the standard timer, described for example in [2]) as interface for its usage:

– init_apic_timer(struct apic_timer_list *timer) initializes the given structure of type apic_timer_list.
– add_apic_timer(struct apic_timer_list *timer) registers the given structure of type apic_timer_list and inserts it into the double linked list of registered timers, which await their execution. The timer handler function timer->function will be called when the timer->expires time is reached.

– del_apic_timer(struct apic_timer_list *timer) removes a given apic_timer_list-structure from the list of registered (and waiting) timers.
– mod_apic_timer(struct apic_timer_list *timer, unsigned long long expires) modifies the expiration time (expires) of the given registered timer structure apic_timer_list. Probably its position inside the registered timer list has to be updated.

4 Using the APIC-timer for high precision traffic shaping

High precision timers can be used for –and are even needed by– a lot of applications. As an example their benefits are demonstrated by using them for network traffic shaping. Traffic shapers are an important building block for the support of quality of service (QoS) in packet networks, because they reduce packet bursts by delaying packets which arrive too fast for their configured bandwidth. It is obvious that traffic shapers need to be able to transmit waiting data packets at precisely calculated points of time to perform in a satisfying way. The accuracy of these timed actions is crucial to the performance of traffic shapers.

To show the impact of timer precision on the shaping smoothing level, research was done using the following three timers on a standard 700 MHz PC with a 100 Mbps Ethernet connection.

1. The standard PC-timer. Its trigger frequency is left unchanged at the default of 100 Hz which limits its accuracy to 10 ms.
2. A modified PC-timer. By issuing interrupts with 5000 Hz the resolution is improved to 0,2 ms but the system workload increases dramatically because of the amount of unnecessary interrupts.
3. The newly developed APIC-timer.

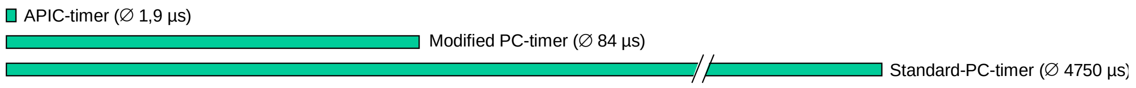


Fig. 2. Mean deviation of the actual to the planned transmission time of data packets for the different timers.

Figure 2 shows the mean deviation of the planned time a packet should be transmitted to the actual value when it is sent. Figure 3 presents histograms of the trigger error for differently delayed packets. It is obvious how the timer resolution affects the deviation.

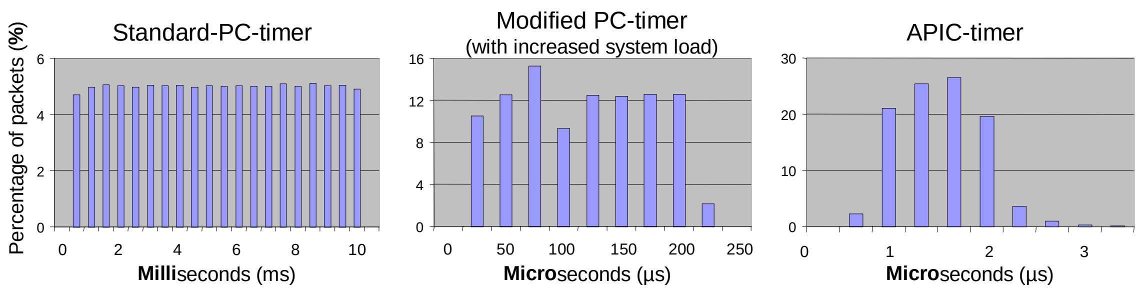


Fig. 3. Deviation of the actual transmission time to the planned sending time for the different timers.

This timer error impacts the smoothness of the shaped network traffic as it is shown in figure 4. In this experiment a constant-rate packet stream of 1292 byte UDP-packets should be shaped to a bandwidth of 1 Mbps. This would result in one data packet transmitted every 10,3 milliseconds. The figure shows the actual interpacket spacing after the shaping process.

When the standard PC-timer is used, the shaper has only the possibility to send a packet either in 10 or 20 ms. There is no way to transmit a packet in between these timer intervals. The shaper was designed to send packets early, so it transmits most of the packets with a spacing of 10 ms. The configured rate is honored by delaying a few packets for two whole timer intervals (20 ms).

The modified PC-timer can handle this job a lot better because of its finer granularity of 0,2 ms. It, too, oscillates between two adjacent timer intervals (which is hardly visible in the figure).

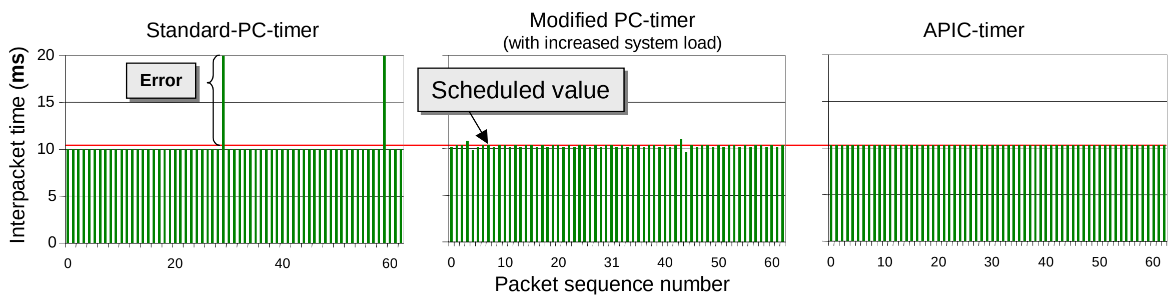


Fig. 4. Interpacket times of a shaped 1 Mbps data stream consisting of 1292 byte packets (nominal spacing 10,3 ms).

By using the APIC-timer it becomes possible to produce a very smooth output data stream. Its very small error of approximately 2 μs is invisible in the figure.

Even at this relatively low bandwidth the standard timer can only produce a jittering output stream. The two other timers perform better and are therefore out through a more challenging test. Figure 5 shows the results of the test shaping a 667 byte packet stream to a bandwidth of 50 Mbps, which equals sending a packet every 107 microseconds.

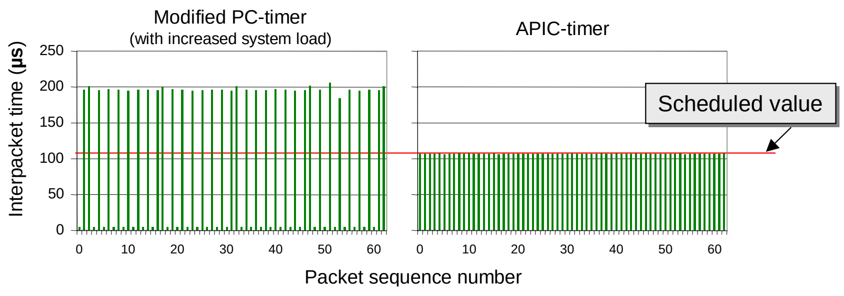


Fig. 5. Interpacket times of a shaped 50 Mbps data stream consisting of 667 byte packets (nominal spacing 107 μs).

In this experiment even the modified PC-timer is operating over its maximum resolution. The scheduled packet spacing cannot be maintained and the shaper has to send packets either immediately or delayed by a whole 0,2 ms timer interval.

Again the APIC-timer shows no signs of decreasing accuracy and produces a smooth packet stream. In following stress tests the APIC-timer was even capable of transmitting packets with a spacing of only 17μs before the test setup reached other limits (packet generation and transmission). This equals a packet frequency of 58,800 packets per second which would –even with small 100 byte packets– result in a bandwidth of more than 47 Mbps.

5 Conclusion

A new and easy-to-use timer module for the Linux kernel has been implemented, which offers the possibility to execute functions at precisely defined moments of time. By making use of the local APIC's functionality an accuracy of about 1 microsecond is achieved for the execution time of programmed actions (although only possible on uniprocessor systems at the moment).

The possible benefits of such an improved timer precision have been shown for traffic shaping as a possible example application.

References

1. Intel architecture software developer's manual, volume 1-3, 2001. http://developer.intel.com/design/pentiumii/manuals/24319{0-2}.htm.
2. Alessandro Rubini and Jonathan Corbet. *Linux Device Drivers*. O'Reilly-Verlag, 2 edition, July 2001. Online version at http://www.oreilly.com/catalog/linuxdrive2/chapter/book.

Paper Submitted by:

Muthukumar Shunmugiah

University:

P.S.G. College of Technology

Country:

India

Remote System Administration (RSA)
Category of Project: System Administration Tool

I have done a project named Remote System Adminitration , and i have written an essay as you mentioned in your mail. The project source code is send as a attachment in this mail. Two RPM files Server program written in java, client program written in C & C++ with a README file showing how to install the software. In case you accept this as a valuable project, I will try to enhance my project so that it accomplishes all the system administration features.

Objective

The General Objective of the project is to provide a tool for the System Administrators to administer the remote(client) systems from being in the server. The following are the main objective of the project Remote System Administration:

- Remote Process Monitoring - Aims to monitor the processes in the client system and provides way to send signals to the processes.
- Remote User management - Aims to provide user management functions such as adding a user, removing a user and modification of user details including the change in password.
- Remote RPM Installation - This is one of the vital goal ofRSA which aims to provide simultaneous installation of RPM packages in the selected client systems.

Description

Necessities for Remote System Administrator:

I myself came across practical problem of installing a rpm package in all the systems in the lab while guiding our system administrator. It's really worthless to waste the golden times of the system administrator in installation of software (repeatedly doing the same) in all the systems. This is the reason why I choose this project. The process monitoring will be useful in the distributed computing environment, networking projects, etc.

The User Management is trivial administrative task. The front end used to develop the Graphical User Interface is done in java so that the server can be any systems (sun Solaris etc). The back end was developed in C & C++ (mostly C).

Methodology

Number of Systems: Two Redhat Linux 7.1 systems were used to do the Project Software Requirement: GNU c++, Java 1.3 Testing : Final testing was done with Five systems.

There are two major processes in the project:

A. Client Side Process

- Will accept the request from the Server process running in the server and will fulfill the request of the server by sending messages.
- Should be executed in each and every client that should be controlled by the server.
- It is a daemon process.
- Interacts more with the system therefore written in C & C++.

B. Server Side Process

- Will provide easily understandable Graphical User Interface to the system administrator.
- Will send request to the Client process according to the user choices.
- Should be executed as a fore ground process in X environment.
- Developed in Java so that the server can be any Operating System (Solaris etc).

Results Obtained

The project was tested with Five systems and all the three modules namely Remote Process Monitor, Remote User Manager and Remote RPM installer are working well and shows expected results. Remote RPM installation is the highlight of the project. Future enhancements are planned to make the RSA system with a full fledgede system administration features.

TO INSTALL THE SOFTWARE FOLLOW THESE INSTRUCTIONS

1. Installation

A. Client Side Installation

rsaclientd daemon must be installed in each client that should be controlled by the server.

The installation procedure is as follows:

1. Get the source file rsaclientd-1.0-1.tar.gz.rpm
2. Execute the following commands

```
rpm -i rsaclientd-1.0-1.tar.gz.rpm

cd /usr/src/redhat/SOURCES

gzip -d rsaclientd-1.0-1.tar.gz

tar -xf rsaclientd-1.0-1.tar

cd rsaclientd-1.0-1

make
```

The client daemon will be installed in /usr/sbin

B. Server Side Installation

The rsaserver must be installed in the server alone. In the server side you might want to edit the configuration file name /etc/rsa.conf

The following steps are to be done to install the rsaserver.

1. Get the source file rsaserver-1.0-1.tar.gz.rpm.

   (Make sure that java is installed in your system and javac_g and java_g are in the PATH variable of the Shell.)

2. Execute the following commands

```
rpm -i rsaserver-1.0-1.tar.gz.rpm

gzip -d rsaserver-1.0-1.tar.gz

tar -xf rsaserver-1.0-1.tar

cd rsaserver-1.0-1/src

make
```

c. /etc/rsa.conf/etc/rsa.conf will contain the IP address of all the clients connected in the rsa network.

Each new entries must be in a new line. It may also include servers own address.

sample entries are

```
192.168.3.1
192.168.3.2
192.168.3.22
```

2. How to start rsaclientd in client side?

There are two ways to start the daemon:

1. Login as root and type the following in the shell

```
/usr/sbin/rsaclientd
```

2. To start the rsaclientd daemon, enter the following line in /etc/rc.local

```
/usr/sbin/rsaclientd
```

and restart the system.

3. How to start rsaserver in server side?

1. Login as root in Kde or gnome or X . open a terminal and type

(make sure that java is installed and path to javac_g and java_g is set in PATH variable)

```
rsaserver
```

A window will appear with all the details of the system in the network of rsa. The clients IP will be read from the file /etc/rsa.conf.

Refer to help for rest

- rsaclientd-1.0-1.src.rpm

- rsaserver-1.0-1.src.rpm

Paper Submitted by:

Shah Sachin

University:

GLS Institute of Computer Technology

Country:

India

NextGen File Management
Category of Project: Device Drivers

Lacking in Current File System:

- **Restriction on file size.**

Administrator cannot have any restriction on file in a directory. Presently he / she car restrict on only directory access or file access. User may create large file or put simple shell to fill whole disk storage. Example:

Create two files: A.TXT contains only 'A'
A contains

```
cat A.TXT >> B.TXT
cat B.TXT >> A.TXT
./A
```

So, Administrator cannot restrict on file size.

- **They are in no extra permission given to files or directory.**

In present system, any user can create a file on other directory's or subdirectories. To restrict this administrator will set rights of user or group. However, these things are at one server but each terminal administrator will set rights on each directory. Therefore, this is cumbersome job to administrator.

Suggested Implementation:

- Administrator give rights / permission to each user accordingly their status like no of file created, number of sub-directory and maximum size of each file. Thus total entry i-nodes will be fixed and accessing time of each file can reduce.

- Each user may set rights such as execution of any file on particular day or time or file can automatically encrypted with some password protected. Therefore, developer can execute certain file or schedule. In present system, AT command is present but It do not maintain execution of successful or unsuccessful entry. Suppose at that time system was down so, at that time AT command will not work. My suggested permission can take history of each file execution. So, such details are provided to the user. This feature is very much useful in Embedded system. User or Developer can schedule that some application or some data transfer will be done on that time. In addition, successful or unsuccessful operation may be checked.

- Administrator will able to mount local hard drives from server and gives rights accordingly. So, from the server, he / she will able to admin whole network drives. User cannot know which computer in the network is containing his file, directory, or data. At present, we combine all computer (around 100 computer having 20 GB HDD) storage capacity then 2,000 GB data can be stored in the network but, As per my calculation around 80 - 100 GB is sufficient for any 100 computer network. At present, other space was used as a second copy of same file or in MP3 songs but this space is utilized efficiently.

- In feature, some embedded system, example in steel plant temperature of plant should be recorded to system after every 10 seconds. Then present system required fixed computer with the plan when user wants then plant computer will give data to user. But in suggested system, Plant will automatically writes data on server's (networks) file system. So, Any authenticated user can see data online and made decision. This is only one example but in feature, much more embedded system.

Problems in the Suggested System:

- Virus: Virus can easily implemented because user may schedule it. But this problem can be solve using permission or authentication from administrator.

- Administrator Jab becomes congested but user may get much benefit and user might restrict in creating extra file and directory.

- Network is the bottleneck of whole file system. But today's networks are much more reliable so it might be not create any problem in features.

- Administrator has new responsibility that whenever any machine was down but user may want data then Administrator has define rules in this manner or stored data in such a manner that when ever user wants data then it was available.

Benefit of Presented File System:

- Feature Expect: Embedded devices are most complex in scheduling. From this, we can run application for the embedded system very easily by configuring a file.

- Costing: Much more effect use of HDD can be achieved.

- Application Cost: We have to purchase only one license for one software. So over all costing of software / application will reduce.

- Managerial aspect: Administrator can manage whole network from one terminal. He may access any file from any ware.

- Parallel processing on multiple computers can be easily configured. Therefore, Processing speed will automatically increased. We can use old / not workable computers for new operating system like Windows Xp.

Requirements for Developing Such File System:

- Some network protocol must be implemented. Those protocols can decide where data was present and accessing methods of that data. Betting indexing is required for better searching data. Protocol, itself decide that which terminals are down and which are coming up in recent.

- Some Important data must be duplicated in two different machines. (For zero fault tolerance.)

- Develop such a hardware devices that they can read system or application or even data from other machine without any error.

- Kernel should read and store all rights on the each files in the system. Kernel examine rights when accessing by the user or some policies like 10% extra growth of file should be allow with some warning message or user program wait until administrator of system allocate more space or remove restriction etc. However, in the present scenario high speed processor and higher memory are available so, Kernel implementation will not restrict with those things.

Note : This file system is not implemented yet.

Paper Submitted by:

Sreeram Jaswanth (Jesse)

University:

International Institute of Information Technology

Country:

India

Category of the Project: Software for Workstation Cluster Management

ClusterPro

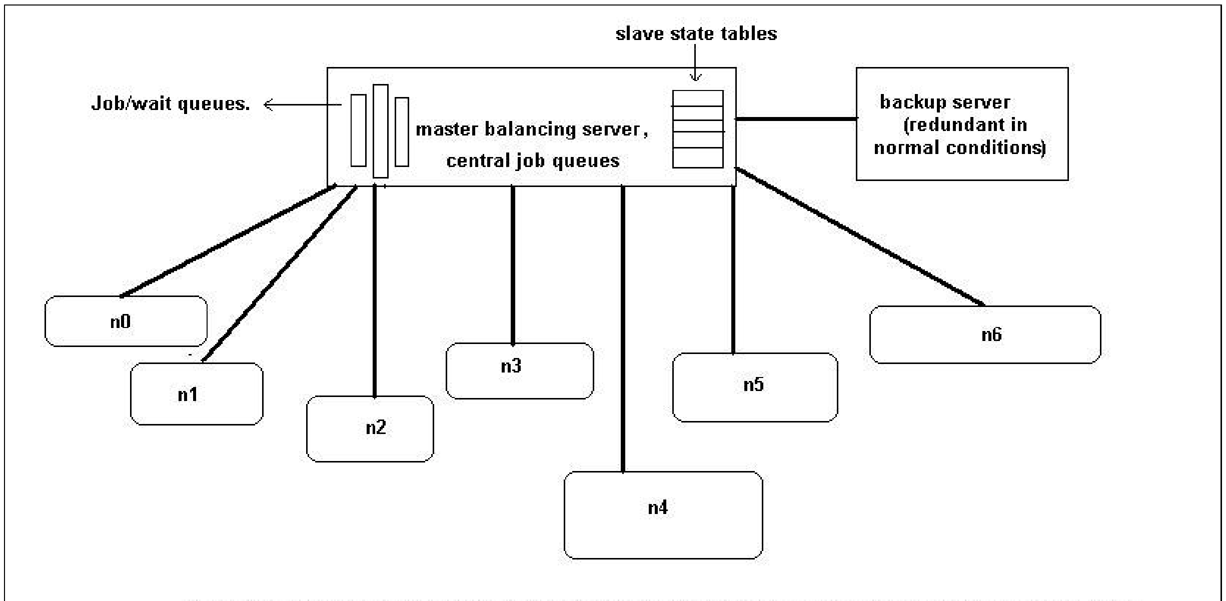
A cluster management package with dynamic load balancing.

Clusters are fast gaining importance and acceptance as an alternative solution to bigger and expensive machines. Usage of clusters for addressing computing needs improves existing resource utilization since it uses only idle CPU-cycles of the nodes. Apart from utilizing unused node resources, clusters also enable a group of nodes to share rare resources, such as specific architectures, special computers dedicated to jobs like vector processing, etc.

A simple monitoring tool devised by the author and used in the campus computer lab showed that on an average, at any given point of time, around 82% of the nodes are very lightly loaded (i.e. are grossly underutilized), 12% are moderately loaded and 5-6% are highly loaded. This goes to show the degree of underutilization in existing computing resources.

The advantages that cluster computing has, over traditional high performance supercomputers etc. is evident from the fact that companies and institutions are switching to cluster computing for fulfilling their large scale computing needs. There are many cluster management software packages available in the commercial domain today. The **"ClusterPro – Cluster manager for workstation clusters"** (here after referred to as 'CLPro') is a package built by the author to efficiently build and operate clusters of in-use workstations.

The basic structure of typical cluster is as shown below:



n0,n1,n2..etc. represent slaves which are part of the cluster (i.e. which are running the cluster package daemon.)

Each workstation has a certain amount of idleness associated with, which is the weighted fraction of resources unused. It is this idleness that the CLPro harnesses.

The master of the cluster maintains a state table which contains information about the slaves that are a part of the cluster. The table typically contains the idleness level (which is a combination of parameters like processor idleness, memory usage, swap usage etc.). When the CLPro service is started on a workstation, the workstation (now a slave in the cluster), reports its idleness parameters to the master balancing server and this information is recorded in the state tables.

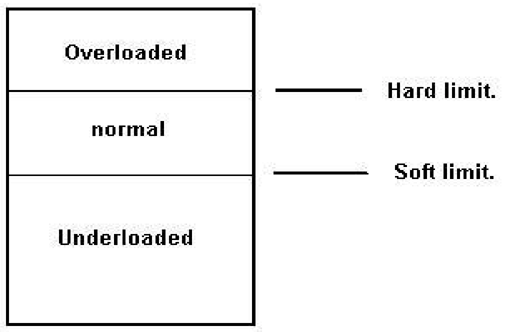
When a slave has a major program to execute, it submits the job to the master. Then the job is checked for suitability and is enqueued on a central job queue. A typical job request consists of (a) a job descriptor, (b) owner descriptor, (c) list of special resources required (if any), (d) argument list. The queues are reconfigurable to support job priorities etc. Once the job has been enqueued, the owner is given a job_log tag, by which he can monitor current status of the job etc. The package supports MPI (Message Passing Interface) libraries hence making it easier to port existing parallel applications onto the CLPro.

Dynamic Load Balancing:

The CLPro has tools for dynamic load balancing among the slaves. This serves to optimize resource utilization by transferring load from overloaded to under loaded slaves. All the resources that are marked by the master_balancer as available for use form a resource pool and jobs are distributed evenly across the pool. A daemon running on the CLPro_master is collecting slave state information and updating state tables in the master server. The tasks are then distributed over the resource pool, according to the "idleness" (defined later in the report) of each slave. Such a distribution is carried out only if the special requirements mentioned in the list of special resources requested in the job submission script. Special resources may include specific architectures, platforms, specific versions of libraries etc.

The process of executing tasks remotely, migrating them and gathering results etc, involves a considerable communication overhead, which may go to unacceptable levels depending upon network load density, peer response time etc. To prevent the slave from making an un-optimal job request, (i.e. submitting a job, which could be executed locally and thus cheaply than by enqueuing it on the master queue) the CLPro uses a set of policies, which govern the decisions regarding the fitness of a job submitted for remote execution. One criterion is the network_load at that instant. The load balancing algorithm uses a pair of threshold network_load values to determine fitness of a job. Beyond the soft_net_load value, the algorithm discourages the submission of remote jobs, since there would be a possibility of loss of critical data during submission or migration, owing to the high network_load.

Another criterion is the slave state (the word 'state' is used to mean a combination of many parameters vital of which are processor load, memory load, swap memory usage etc.). Here, the load balancing algorithms use the classic **ALONSO & COVA (1988)** hypotheses for thresholds. Here too, like in the network_load criterion, two load thresholds are used viz. the soft_load_limit and the hard_load_limit. Below the soft_load_limit all local processes are run locally and all remote jobs requests are accepted. Between the soft_load_limit and the hard _load_limit, all local jobs are run locally and all remote job requests are rejected. Beyond the hard_load_limit, all local jobs are submitted to the master queue and all remote job requests are rejected.



Process Checkpointing and Migration:

The CLPro also features a process checkpointing and migration mechanism for better balancing of loads on slaves. The state of every process not belonging to the local machine is periodically recorded in the process state tables (which are exclusively used for checkpointing purposes only) on the local machine. In the event of a node failure or abnormal overloading of a node, the process can be paused, migrated to another node (this, according to the load balancing policies) and restarted again from the last checkpoint.

However, it has been found out that pre-emptive process migration is, in most cases extraordinarily expensive, without any (in most cases) extraordinary benefits. Process migration which is carried out to as a part of load balancing is very inefficient in its present form. Instantaneous slave loads are too volatile for process migration techniques to be based on them. After a process has been migrated from one slave onto another due to overloading of the first one, the second one might become momentarily overloaded, thus migrating the process again. This may continue for an abnormally long time if many slaves are having widely fluctuating load. Thus migration may not be the most efficient solution here (especially since it involves saving and transmitting large state table, checkpoint files and stacks). Better migration mechanisms are being studied.

Multi-Clusters:

The CLPro can be configured to support multi-clusters too. Multi clusters are used when the number of nodes is very big, and in cases where the nodes are spread over geographically distant locations. In such cases, the nodes are divided into multiple clusters each with a manageable number of nodes. Each cluster in turn has a cluster controller, which acts as an intermediary between a slave and the master of the multicluster. Slave state information is now recorded and maintained both with the cluster controller and master of the multi cluster. The cluster controllers receive the job requests first and decide if the job can be executed cheaply, within the cluster itself or if it has to be executed remotely (i.e. using the resources of the other clusters too.).

Other features of the CLPro include a normally-redundant backup master server on standby for deployment in case the master_balancing server fails. This backup server monitors the state of the balancing server continuously and copies the slave state tables and job queues onto itself periodically. When the backup detects that the master has failed (failure in responding to a test-stream of packets, and failure in acknowledging an express request), the backup_master takes over and broadcasts a message requesting the slaves to report latest load status. It then updates its slave state tables (that it has been copying from the previous master) and assumes the role of the master_server. However, it has been observed in experimental runs that the time taken in -- detecting a master_server failure, collecting latest salve state information, and resuming the job queue – is very large as a result of which, the slave information collected may become stale, and jobs may become redundant. The author is studying better mechanisms to restore normal state of a cluster in case of a master_server failure.

The CLPro was developed as a part of a bigger GRID-computing project for harnessing idle CPU cycles of the PCs in the lab. Before the development of the CLPro, many other cluster management softwares were studied for suitability of use in the project. One major hurdle was the unavailability of open source CMS packages. Proprietary CMS packages like the commercial ones available today, have the biggest disadvantage that they are difficult to configure, reconfigure, and customize. Cluster management needs vary widely from cluster to cluster and even from configuration to configuration. And the unavailability of source code for the packages makes it even more difficult to configure the package to fit to the cluster. The CLPro addresses this issue by providing sysadmins complete flexibility in modifying and fine-tuning the package to suit their clustering needs.

When the author was studying the different CMS packages available today (both commercial as well as research), like CONDOR, NQS (developed by NASA and then taken over by university of Sheffield U.K), CODINE etc., he was surprised to find out that most of the CMS packages didn't support LINUX. This is incredible, considering the fact that LINUX has been emerging as the platform of choice for clustering. So much that the Google web search engine is believed to use a high performance linux cluster for querying on its multi terabyte web-index. The unavailability of CMS packages supporting LINUX as a platform has been one of the motivations for the author, while building CLPro.

MPI (Message Passing Interface) has evolved over time to become the industry standard distributed application programming environment. It has grown, to replace many other environments like PVM, CHIMP etc. But it is surprising that the popular CMS packages today don't support MPI. The product data sheets or information pages of these CMS packages like DQS, CODINE etc, do not even mention any role for the MPI. On the CLPro, programming environments like MPI are supported and there are plans to include support for others like PVM (minimum 3.3.X).

Perhaps the most important feature of the CLPro is its set of load balancing algorithms. Many CMS packages in use today like the GNQS (Generic Network Queue System) etc., feature only static load balancing schemes. This severely restricts the degree of balance that can be achieved, as well as the degree of usage of idle cycles. The CLPro uses dynamic load balancing schemes by means of which it is able to respond to changing slave states. To optimize the distribution of processes, the CLPro uses parameters called “weighted load averages” for every slave. The load status of every slave is reported to periodically to the master server, which in turn updates its state tables. The master averages this information over 5 and 10 minutes, applies an appropriate weight constant and records it as state history. These weighted loads are then used by the master to calculate the final idleness of a slave. A simple polynomial of the form -

$$\mathbf{I = g\acute{a} + (1-\acute{a})g' + T.}$$

Here, I is the final idleness of a slave as to be recorded in the state table of the master, g is the idleness as reported by a slave, g’ is the weighted load average , á is the auxiliary weight factor, and T is a constant which is unique for every slave.

Based on this final idleness value, the state table is sorted in descending order of idleness. And all further allocations of jobs are done basing on the idleness values in this table.

In case there is an abnormal increase in load on a slave, the slave reports this to the master, which responds by dropping that slave’s position down in the sorted list of idle slaves, and all remote jobs on that slave are stopped. If the state of abnormally high load persists, all remote jobs on that slave are checkpointed and are migrated to other slaves (according to the idleness list).

This form of dynamic load balancing has performed much better than static load balancing (used in GNQS etc.,) and even many other dynamic balancing techniques like the ones used in PBS (Portable Batch System), etc.

The Tests and the Results:

The CLPro was tested on a cluster with number of nodes ranging from 2 to 16. The nodes were ordinary PC's in the computer labs.

<u>**Nodes:**</u>

- CPU - Pentium ø 1GHz.
- Main Memory: 128MB.
- Disk – 20 GB.

Each node was using:

- RedHat Linux 7.1 Kernel 2.2.16 with MPICH ver1.2.2.2, glibc ver 2.1.92 and glib ver-1.2.8.

<u>**Network:**</u>

- 10/100 Mbps Local Area Network.
- Network Interface Card: Realtek RTL8139(A)-based PCI Fast Ethernet Adapter.
- High speed hub and switches.

Initial test runs indicated a high communication overhead. And often the results were widely varying owing to the fluctuating LAN traffic and users' local jobs. Improvements were made and the cluster was able to achieve around 3740 MIPS in a test run. This was reasonably good, given the fact that the number of nodes available was very low (ten in number). But still, it has been observed that the network overheads involved were very high. However, the load balancing algorithms showed impressive results. But as in any new package, there remain improvements to be made.

Improvements Planned:

- Incorporate changes to keep the communication overheads and peer- response latencies low.

- Make it more robust. In some tests, slave failures or abnormal local loads resulted in remote jobs dying away.

- Incorporate tools for making the cluster more tightly coupled. In the present version, the CLPro treats slaves as independent identities, ignorant of each other's existence. Improvements in this regard -Tools to incorporate include some kind of shared file services, a directory naming structure for files and similar resources shared by the slaves.

- Porting the CLPro to <u>Java RMI</u>. Also, to provide support for <u>OCCAM</u> or CSP.

- Making it more reliable – the master load balancing server is a ‘single point of failure’. Deploying the backup effectively and restoring normality in the least possible time is critical to ensure high availability of a master server.

- Improving the process migration mechanism.

Conclusions:

The CLPro has made a reasonably good start. And it has a long way to go (in terms of improvements) and the author is already working on them. But nevertheless, the author is confident that it will grow and mature into a robust, efficient and comprehensive clustering solution. The idea was born out of frustration when the author, who was in a research group studying the applications of bioinformatics, was studying DNA sequence alignment algorithms, but could not get access to the enormous processing power required to test them. The CLPro, the author feels is an appropriate answer. And the enormous promise being shown by CLPro is acting as a motivator for the author to make it better and even better.

Paper Submitted by:
Francesco Regazzoni

University:
Politecnico di Milano

Country:
Italy

On-Line Help System to Complement the HowTo's and Man Pages

Part 1: Description of Project

Our project consists in an help files navigator, we called it help_navigatOV.

We tried to make an easy and fast way to provide most useful information for a Linux user or admistrator. The main idea is that you don't have to look for different sources of Information (man pages, HowTo's and so on) but it's all included in the same place.

You don't need to know anything about commands you're searching help for, you only have to know what you want to do. Help pages are about most used commands and main configuration files.Of course, only main features are provided for every item. It would be useless to provide complete information: it would be just a copy of the man pages.

Part 2: Methodology We Used

Developing the project, we tried to maintain a certain consistency with the existing tools. We decided to provide our help_navigatOV with a user-friendly text interface like most utilities (i.e. Linuxconf). So we've implemented our software's interface using an utility called dialog (ver 0.9), that we found at www.redhat.com.

The menu is organized in two parts: the left one contains a list of all commands or file there's some information about. The right one shows a little description about the chosen item. The user can move up or down using arrow keys and can choose a voice pressing Enter. Choosing an item cause all the information about it to be displayed. After reading you can come back to the main menu just pressing Enter, then you can continue selecting and reading another mini-help. You can quit just selecting Cancel from the main screen. For every item considered there's a file in a subdirectory. This file is named like the item it refers and contains two kinds of information: the first one in included between <H1> </H1> tag and it's the main title to be displayed at the top of the window when you select an Item; the second one includes all the information available to be displayed, that's a short brief on the man page about it. Every file is provided with a particular formattation in order to be correctly displayed it in the navigator. You can recognize the file containing information about configuration files because they're in the form file_<name>. All the files are stored in a subdirectory and they've got .hlp extension.

The navigator looks for this subdirectory and it lists all the valid files found in the left side of the screen. After that you can view any of the mini help listed as described above. Now we describe the way we implemented our project. There are two Python scripts: the first one generates and manages the windows on the screen, the second one creates the menu and manages the user choices.

Of course, to make the navigator running you need to have Python interpreter installed. Moreover, you also need the dialog utility mentioned above (ver 0.9 or later) for the user interface. To develop opur project we had to look for specific information about Python Language. We found what we were looking for in a newsgroup (www.deja.com) and on The official Python web site: www.python.org. To learn more about the dialog utility we've consulted the man pages about it. As explained above, to compose the mini help files, we based ourselves on the man pages and HowTo or on our personal experience in using Linux Systems. We decided to provide our system with a network architecture: on server side there are help files and a demon that listen on a port. Clients need only a little script that takes from server the list of help files to create menu and then asks help file needed and shows it.

Part 3: Results

We reached our objectives in developing a fast, easy to use help system.
Moreover, we think our system can be useful both for a beginner and an administrator: the beginner can retrieve main information without going crazy reading the man pages; an expert user can quickly retrieve the few options he needs to keep on working.

An interesting aspect of the system we developed is that you can easily add items just putting a valid help file in the appropriate subdirectory: help_navigatOV will automatically display it the next time it scans that subdirectory.

This feature is very useful in a network. The administrator, in fact, has only to add a new help file to allow at once all client to see it.

Paper Submitted by:

Jongmin Park

University:

University of ULSAN Korea

Country:

Korea

Design and Implementation of a WWW-based C Source Code Documentation Tool Using RE Technologies

1. Introduction

Due to the rapid growth and popularity of a WWW, the existing software engineering technologies are required to adapt to a new environment. That is, software engineering technologies applied to the existing environments should change so as to conform to a new paradigm.

By the virtue of WWW, anyone can access to desired tools at anytime, and from anywhere. WWW-based applications don't require some special installation procedure unlike the general applications. And developers can upgrade their software at anytime, so that they can provide better services for their user.

So the field of software engineering has much to gain from the WWW. Many of the goals of the field can be achieved in a better way by utilizing the facilities provided bye the web.

Software maintenance is a very important field of software engineering because the maintenance of existing software can account for over 60 percent of all effort expended by a development organization, and the percentage continues to rise as more software is produced[1]. And program understanding is an essential part of software maintenance.

In this paper, we discuss the design and implementation of a WWW-based documentation tool(WEBDOC_C) developed by using the reverse engineering and web programming technologies. Our tool automatically extracts the design information from C source code files.

2. WEBDOC_C

We developed a WWW-based C documentation tool called WEBDOC_C. The users connect our web application through web browser and then present the listing of C source code and header files.

Based on these information out tool uploads those files from user's computer into our system. Then analyzer extracts information about call relationships between functions, and information about definitions and references of global variables. Analyzer stores these extracted information into database.

Based in the stored information, Generator creates the call relationships between functions, and the relationships between functions and global variables in the form of structure charts. Users can obtain design information about relationships between functions, global variables by selecting the appropriate function.

2.1 Analyzer

The first work performed by Analyzer is uploading C source code file consisting of project into our system by using the information about the project provided by a user. For the efficiency of the whole development work, we used the existing uploading component in this work. Then, Analyzer performs to analyze these uploaded files. In this work, we used the existing C analysis utilities such as cflow and cxref. These utilities generates information about function calls and the definitions and references of global variables from C source programs as their own formats. That is, C source files uploaded by users is used as input of these utilities, and we developed simple C programs which extract the desired information having our own format from the output of these two utilities and store these information into the database.

Database is consisted of eight tables. Three tables are used for user supports, project information, and information about uploaded files. Two tables are used for maintaining the information about functions, and global variables. And the remaining three tables are used for maintaining the information about call relationship between functions, and the information about the definitions and references of global variables.

2.2 Generator

Generator performs a synthesis work to generate the structure chart format after obtaining the information stored in database and then displays it to users through web-browsers. To show the structure chart on the web browser, we decide to use Java applet and Java Servlet.

Servlets are consisted of four classes to obtain the information about function call relations, relationship between functions and global variables, and relationships between files and functions. Our Java applet is consisted of one class, and displays design information as structure chart format by cooperating Java servlets. Java Applet maintains information about relationships as tree structures and uses Hashtable.

3. Implementation of WEBDOC_C

WEBDOC_C is developed on Red-Hat Linux OS (ver. 6.1), and is based on Apache web server. We developed out servlets on Jakarta-tomcat, and used Mysql database management system.

Our system is consisted of twenty four files including five C programs to performs the work that stores the outputs from cflow and cxref utilities into database as our formats, four Java servlets and one Java applet which obtain information from the database and display structure charts on the web browser. The size of our system is about 4,000 LOCs. Output screen is divided into two frames. The left frame is consisted of work areas where users select as they want and the right frame is consisted of output areas where the output result is shown. Top left corner represents function names as list box, and users can select a function that users want to analyze. Below that box, global variables names are also shown as list box. If users select a global variable, a group of function using that variable.

4. Conclusions

Our tool automatically extracts the design information from C source code files. We have attempted to adapt and existing software engineering tool to a new paradigm and have tried to use and combine the various web programming techniques and the existing technologies. We expect that a study on combining the existing software engineering technologies and web technologies is activated on the basis of our study. In the future, we are planned to study website analysis techniques including web metrics and maintenance of web applications.

Paper Submitted by:

Frank Kruchio

(frank.kruchio@paradise.net.nz)

University:

Victoria University of Wellington

Country:

New Zealand

Linux Server/Workstation Performance and Stability Optimizations

1. Linux Optimizations

The availability of free source code under Linux applications allows the end user to optimize software for a specic processor architecture. In this paper I will describe my objectives, the approach I have taken, the tools and processes I used and the nal results.

1.1 Introduction

Processor architecture specic optimization of software provides faster program execution speed, better efficiency, improved user experience, faster response time and improved usability on a workstation. On the server side it improves throughput which is more important than response time in a workstation environment. One added benet in both the server and only faster but it is also more stable due to being compiled on the processor where the program will eventually be used. There seems to be a small number of websites available on Linux optimization topics and none of them targets the beginner.[1] Linux newsgroups frequently show end user interest in how software optimization is done under Linux, so there seemed to be a genuine interest in the topic.

1.2 Description of the Challenge and Objectives

Linux distributions for the x86 architecture are far from being optimized out of the box, Red Hat is compiled for i386, SuSE i486 and Mandrake is only compiled for i586. My main objective was to provide a simple tutorial with IBM's Toot-O-Matic XML/Java tool that people new to Linux can read and by following the tutorial optimize their own servers/workstations for maximum performance. I wanted to provide a tutorial that is clear, easy to read and understand; so the readers can reproduce the same steps on their own computers at work or home. This way the readers could easily enhance Linux functionality, usability and performance.

[1]These websites focus mostly on kernel optimization.

1.3 Research

Since the focus was on providing an easy to use tutorial, this tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. I had to restrict the Linux distributions to the Red Hat package manager (RPM) based distributions to make sure that the process is simple enough for people new to Linux to follow. For this reason I discarded the idea of using Intel's icc/icpc C/C++ compiler and decided to focus only on the gcc/g++ compiler coupled with RPM's rebuild functionality.

1.4 Benefit and Results

I have tested the tutorial on someone who has never done any optimizations on their Linux computer and found that the person was able to follow the instructions very easily. Some of the questions and feedback I received was used to improve or clarify parts of the tutorial. The benets are two fold: increased stability is one and the performance improvements are immediately noticeable. Applications start faster, respond quicker and the end user experience is much improved. One of the main points that were addressed by the KDE project is that KDE application performance needs to be improved. By optimizing the latest KDE 2.2.2 source RPMs and the QT C++ GUI library this problem is solved. While XFree86 is exible, its memory footprint and performance can be much improved by optimizing it. After optimizations the memory footprint of XFree86 is smaller and the X server is more responsive. Linux is ready for the desktop. Optimizations of the dierent RPM based Linux distributions can be easily done. This tutorial will hopefully make Linux more attractive in the eyes of those who are using Linux now or considering it as a possible alternative in the future.[2]

[2]IBM Linux Scholar Challenge November 30, 2001

Created with LATEX2"

Paper Submitted by:

Bruno Silva

(etbrunos@ua.pt)

University:

Universaidade De Aveiro

Country:

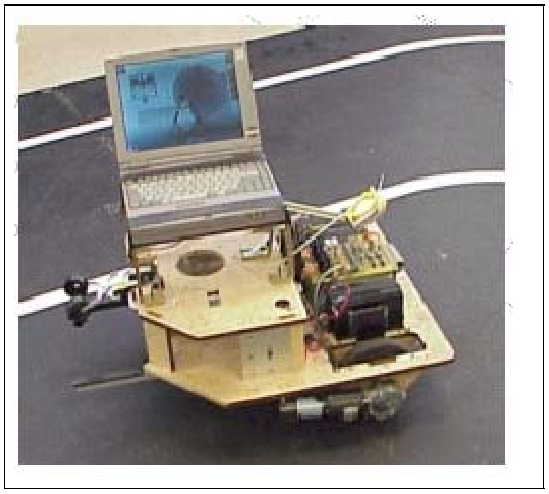
Portugal

Linux-Based Robot withVision System November 2001, by

Team: Bruno Silva, Helder Lemos, Luís Magalhães, Nuno Nunes
etbrunos@ua.pt , hjlemos@alunos.det.ua.pt , lmcrmagalhaes@alunos.det.ua.pt , nmnunes@alunos.det.ua.pt

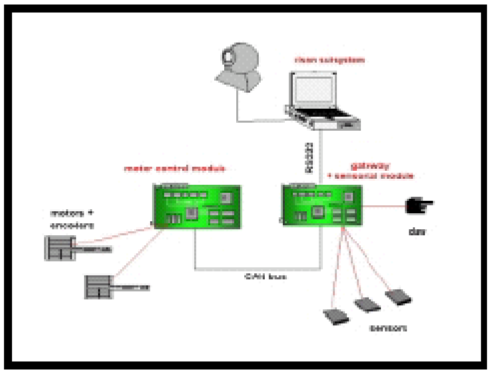
1. Introduction

The **Cyclop** robot was developed in the Department of Electronics and Telecommunications of the Aveiro University, aiming to participate in Robótica 2001 – National Robotics Festival and in FIST 2001 – International Festival of Sciences and Technologies. This project was developed by 4 students during their pre-graduation seminar under the orientation of Profs. J.L. Azevedo, P. Fonseca, M.B. Cunha and L.Almeida.



Cyclop distinguishes itself due to his distributed architecture, built on top of a Controller Area Network (CAN) bus and his WebCam based sight system, whose processing core is a PC running a GNU/Linux OS. The whole system, whose description follows in the next topic, is summarized in figure 1.

Fig. 1 – Cylop robot block diagram



2. Cyclop Robot Overview

The **Cyclop** robot is intended to perform a set of tasks. It has the ability to follow a line drawn on the floor (a white line over a black background or vice-versa) while detecting cylindrical pedestals with 4 cm of diameter holding pool balls, placed at approximately 80 cm of the line. The robot stops following the line, collects the ball and resumes the line following.

The **Cyclop** robot, as mentioned previously, has its operation based on a distributed architecture whose communication (between the several distributed processing modules) is assured by a Controller Area Network (CAN) bus. This bus is used by some major automotive industry manufacturers and allows transmission rates up to 1 Mbit/s. Furthermore the CAN bus is inherently fail-tolerant which makes it a fine choice to this project.

Using a distributed architecture allows the existence of a smaller processing power in each distributed node, hence making possible the usage of simpler and cheaper systems. The Cyclop robot has three processing modules:

- "The *PC*, in the vision subsystem, whose Linux OS is responsible for the more complex processing (as is the case of image processing) and for the global control, behaving as the system “brain”, commanding the information exchange between the different nodes;

- "The *sensorial module* (which comprehends a sensorial information acquisition card and a CANivete) responsible by Cyclop’s sensorial “Perception”.

- "The *motor interface module* (which comprehends a motor drive and a CANivete) responsible by the closed-loop motor control.

The second and third modules processing power resides in the *CANivete* board. CANivete [1] is a platform developed in the Department of Electronics and Telecommunications in the Aveiro University and commercialised by Micro I/O – Serviços de Electrónica, Lda. This board is based on Philips microcontroller P80C592 [2] and enables several functionalities, such as a CAN controller, PWM generators, ADC’s, serial interface, etc, thus making it appropriate to be used as a CAN network node.

The sensorial module has a second functionality, as it performs the role of a *gateway* between the serial port (of the PC) and the CAN bus. It would have been possible (and even desirable) to separate these functionalities but due to structural motives this possibility wasn’t developed.

3. Linux, Control and the Vision Subsystem

The vision subsystem comprises a Philips WebCam (USB ToUcam Pro PCVC740K) and a 100MHz Pentium Laptop PC running RedHat 6.2 in kernel 2.4.3. The option for a GNU/Linux O.S. was made due to its open source philosophy, thus making easier the software development phase. Furthermore Linux is by far the most reliable and robust O.S., especially in embedded-like applications where we face critical time restrictions. During development phase kernel 2.4.3 was the latest version with USB device drivers, hence the option for this kernel. This Web Cam was supported by just a few kernels and the use of this one in particular implied a patch and a recompilation. These operations allowed the Web Cam to work at its maximum frame rate (30 fps).

The communication between the Linux PC and the CAN Network was achieved through a RS232-CAN gateway. This implied the development of routines with the serial port communication using its device drivers.

The video acquisition and processing core was built on top of Video4Linux drivers, allowing to try different cameras/frame grabbers and vision solutions to obtain the best performance.

In order to download the compiled HEX files to the CAN nodes controller cards, it was also developed an application, a LOADER, allowing this programming to be done from the Linux PC console. The original LOADER was developed on a MS-DOS environment which was contrary to the project's philosophy.

Obviously, as in any Linux-based system, all the software was developed in ANSI C, using *gcc* compiler for Linux modules and Linux SDCC for the 80C51 based card (CANivete).

Most of the information used in this project's progress was gathered through the Internet, namely on:

- www.linux-usb\devices.html
- www.smcc.demon.nl (support for Philips USB webcams)
- www.linuxdoc.org
- www.freshmeat.net
- soureforge.net

4. Results

The participation of the **Cyclop** robot in the FIST competition was canceled due to repeated postponements of the event's realization date by its promoters. Nevertheless the **Cyclop** robot accomplished a brilliant performance in the National Robotics Competition, winning the first prize and the Engineering Award. Three small videos with the robot performing its main tasks, during the test phase, are sent in attachment.

5. Further Developments

Further developments on this project include:

- Evolution from the kernel 2.4.3 to a real-time kernel in order to use real-time threads;
- Rewriting interrupt-driven serial port device drivers resulting in a more efficient system;
- Modifications on the processing core allowing the use of two cameras.

6. References

[1] *"CANivete – User's Manual"*
Micro I/O – Serviços de Electrónica, Lda., 1999

[2] *"P8xC592 – 8 bit microcontroller with on-chip CAN Datasheet"*
Philips Semiconductors, June1996

[3] *"80C51 family hardware description"*
Philips Semiconductors, August1996

[4] *"CAN specification version 2.0 – Technical Report"*
Robert Bosch GmbH, 1991

Kernel-HOWTO, *The Linux Kernel HOWTO*
Module-HOWTO, *Linux Loadable Kernel Module HOWTO*
Modules, *Linux Modules Installation mini-HOWTO*
Kerneld, *The Linux kerneld mini-HOWTO*
BogoMips, *BogoMips mini-HOWTO*
BootPrompt-HOWTO, *The Linux BootPrompt HOWTO*
Serial-HOWTO, *Serial HOWTO*
Serial-Programming-HOWTO, Serial Programming HOWTO

Paper Submitted by:

Helmut Cantzler

(helmutc@dai.ed.ac.uk)

University:

University of Edinburgh

Country:

United Kingdom

Application in the Area of 3D Modeling
- Mesh Viewer -
A Lightweight Tool to Display Triangular Meshes

1. Description

The Mesh Viewer [11] is an easy to use lightweight application to display triangular meshes from a variety of file formats. Triangular meshes can be rotated, translated and scaled (all done with the mouse). The model is lighted by multiple light sources. Models can be displayed texture mapped (optional with bilinear filtering), solid or as a skeleton (full or just the front lines). The surface normals of the triangles can be displayed. Features (from a different data file) like edges and points can be displayed into the mesh. Viewpoints can be saved. Screenshots of the model can be taken (as BMP, JPEG, PNG and so on).

In the free software community is a shortage of this kind of software. Now with the recent advantages in 3D acceleration under Linux (DRM and GeForce driver), it is time to advance the 3D software as well. A related software project is Geomview[5]. It can display 3D models as well. The Mesh Viewer differs from Geomview in some ways. Mesh Viewer is a more lightweight application concentrating only on displaying of triangular meshes. It uses a newer graphical user interface library and supports several different file formats. It supports until now files from the Gnu Triangulation Library[8], Geomview file[5] (only polygons consisting of 2 or 3 vertices), pmesh files (used at the University of Edinburgh) and VRML 1.0/2.0 files. Only shapes consisting of triangles are read from the data files. Shapes in VRML 2.0 are rotated, scaled and translated if necessary. The file name for JPEG texture and texture coordinates are read from VRML 2.0 files if existent. VRML 2.0 support was fairly important for the Mesh Viewer development. It started as a development to display reconstructed scenes (computer vision). These scenes are typically saved as VRML 2.0 files[9, 4].

2. Methodology

The project was developed with C++. The first component which was created was the mesh class. It can read and write the different file formats described above. A mesh consists of triangles, edges and vertices. Edges are not necessarily part of a mesh file, but can be created from a set of triangles and vertices easily.

Meshes can be manipulate in a number of ways. The Mesh Viewer typically maps the mesh to the origin and scales it into a normal sphere. As a mesh file is read, properties of the triangles like surface normal,centroid, size, perimeter and distance to the origin are calculated using vector algebra. Similarly, for the edges the length, centroid and orientation is calculated.

OpenGL[10] was used to display the triangular meshes. It is the industry’s most widely used and supported 3D graphics application programming interface (API). [13] provides excellent information on OpenGL and the OpenGL Utility Library. Mesa 3.4.2 with XFree 3.3.6 and the Nvidia GeForce Drivers with XFree 4.1.0 were used as OpenGL libraries. OpenGL is used with enabled depth buffer, double buffering (for flicker-free drawing), the RGB color mode and the stencil buffer. The OpenGL initialization function sets context rendering flags, lighting, texture and a few variables. Texture filtering is used if activated. A function, which is called whenever the OpenGL widget needs to be painted, draws the 3D model depending on the display mode. The model can be drawn as triangles with texture, solid triangles, triangles (wire frame), front lines, edges or points. Surface normals of the triangles are drawn if activated. The model is rotated or translated according to the received mouse move events depending on the last mouse press event (all three mouse buttons are used).

To be able to display textured models it is necessary to read images and convert them into an useful format for the OpenGL library. This is done at two stages. Firstly, texture is expected to be stored in jpeg format. Libjpeg[3] is used to read the image file. The second stage is the scaling of the texture. OpenGL expects that texture is in an array of the size 2^n by 2^n. So, the original image is scaled to a suitable size. So, the original image is scaled to a suitable size.

Two toolkits were used to handle the user interface issue of the project. The Mesh Viewer can use either of them. Though, much more functionality is provided with the QT version. Firstly, the OpenGL Utility Toolkit (GLUT)[2] was used. It is system independent and implements a simple windowing application programming interface (API) for OpenGL. GLUT is well-suited to develop simple OpenGL applications. A simple pop-up menu and a few shortcuts gives access to functionality like changing display mode (“1”...”6”), enabling surface normals and so on. Secondly, a more sophisticated toolkit was used. The Qt[1] toolkit is a cross-platform C++ GUI application framework from Trolltech. The toolkit includes an excellent documentation with a small tutorial. The library provides application developers with all the functionality needed to build state-of-the-art graphical user interfaces. Standard GUI components like a menu, an icon bar, a status bar and progress bars are used. The menu gives access to most of the functionality. The icon bar is a quick way of opening a model, changing the display mode or changing the light intensity. The status bar displays information about the model and the rendering speed (frames per second (FPS)) of the OpenGL widget. The OpenGL widget is in the middle of the main window. Beside the OpenGL widget is a slider responsible for changing the clipping of the displayed 3D Model.

A web page for the Mesh Viewer was created at:
http://www.dai.ed.ac.uk/˜helmutc/mesh_viewer/.

The web page describes the project, the 3D formats, how to use it and the known bugs. It lists the software requirements. Some screenshots of the program are on the page. At the bottom is a link to the source code. The Mesh Viewer (with the web page) is now listed at three major free software indexes in the Internet. It is listed at Freshmeat[6], Apps.kde.com[7] and the free software index of Trolltech[12]. At the first day after listing the project 400 people visited the page. Though, at the following days the number of visitors settled down between 10 and 30 per day.

3. Achievements

This project was started without knowing anything about OpenGL, Libjpeg, GLUT or QT. It turned out that these libraries were fairly easy to use and that the documentations are very good. The online documentation form Libjpeg, GLUT and QT was for the project more than sufficient. Only for OpenGL a special book[13] was needed. Unfortunately, the book does not make it clear that texture must be of the size 2^n by 2^n. Much time was needed to realize that. Scaling the image properly solved the problem.

The mesh class and its support classes rely heavily on the Standard Template Library (STL). Lists, vectors, sets, maps and pairs were used. STL is very easy to use and fairly fast. Though, it seems that there are some memory usage issues.

The OpenGL part is one of the main components of the project. A few details make the rendering visually more interesting. First of all, texture mapping with all the possible options is very important to enhance 3D models. Furthermore, using lighting from the right positions with the right amount of diffuse resp. ambient light is very important to present a good looking model. The Mesh Viewer uses 4 light sources, which is a good compromise between good lighting and rendering speed. Displaying front lines with the stencil buffer gives very good results. Though, rendering speed is slow even with a GeForce graphics card.

Using two different toolkits for the graphical user interface was fairly interesting. Both toolkits provide the same functionality for OpenGL programming. So, most of the source code can be used for both toolkits. “glmesh_common.cc” is basically the same file in both directories (glut and qt). GUI programming with QT as it turned out is very easy and intuitive to use. The documentation with the 14 step tutorial and the example applications were very helpful. Many techniques in QT like the layout managers are similar in Java (AWT/Swing).

Extracting data from more complex file formats like VRML 2.0 is very challenging. Mesh Viewer does not extract all information out of it and probably never will. However, vertices, triangles and the texture coordinates (with the image file) are read fairly reliable. Shapes in VRML 2.0 are rotated, scaled and translated if necessary. This was a major breakthrough for one particular model.

Using the free software indexes is an easy way of getting attention from other members of the community. I got already a few emails asking me mainly how to compile or use it.

References

[1] QT 2.x. *Http://www.trolltech.com/qt/*.

[2] GLUT 3.7. *Http://reality.sgi.com/mjk/glut3/glut3.html*.

[3] Libjpeg 6.x. *Http://www.ijg.org*.

[4] National Research Council (NRC) Canada. *Http://www.vit.iit.nrc.ca/VIT.html*.

[5] Geomview. *Http://www.geomview.org/*.

[6] Freshmeat: Largest index of Unix software. *Http://freshmeat.net/*.

[7] The latest in KDE and Qt Applications. *Http://apps.kde.com/*.

[8] Gnu Triangulation Library. *Http://gts.sourceforge.net/*.

[9] Marc Pollefeys’ models. *Http://www.esat.kuleuven.ac.be/˜pollefey*.

[10] OpenGL. *Http://www.opengl.org*.

[11] Mesh Viewer Web Page. *Http://www.dai.ed.ac.uk/˜helmutc/mesh_viewer*.

[12] Free Qt software index at Trolltech.
Http://www.trolltech.com/developer/freesoftware/.

[13] M.Woo, J. Neider, T. Davis, and D. Shreiner. *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*.

Paper Submitted by:

Barnaby Gray

(bgrg2@cam.ac.uk)

University:

University of Cambridge

Country:

United Kingdom

Category Software - Communication Application

Project and Objectives

The project which I set out to implement was an ICQ Instant messaging client to implement the new ICQ2000/2001 TCP based protocol and bring some of the functionality previously only available to Windows users to Linux users.

My prime objectives were to support logging in to the Authentication Server (the Authorizer) and then with the credentials received logging in to the Basic Oscar Server (BOS) that provides the majority of the service on the ICQ network. To support adding and removing 'buddys' from your contact list and to support user information retrieval (alias, first and last names, email addresses, etc.). Also to support the most important features - sending/receiving messages through the server and sending and receiving SMS messages through the ICQ-SMS gateway. The Graphical User Interface (GUI) should be easy to use and represent a familiar and efficient interface to the user without requiring any detailed programming or protocol knowledge.

Methodology

Initially the biggest and most time consuming step was developing the protocol communication part of the client. I had decided from the start that the most logical way of coding it was to split it in two. The first part is the library (called libicq2000) which handled logging in, maintaining a connection, handling server messages and encoding messages sent to the server. The second part is the User Interface which will link to this library and use it to perform all the low-level protocol and sockets communication. This provided a clean abstraction layer between application and protocol.

The code was all written in C++ since then Object-Orientated methodologies could be practiced. Wrappers round C calls were coded by me for all the underlying socket functionality, and the excellent gtkmm C++ bindings for gtk were used for the graphical toolkit.

The actual details of the Oscar/ICQ protocol will not be entered into here, but it logically splits up into a basic layer (called the FLAP), which encapsulates the most commonly used units of communication (called SNACs). See http://www.aim.aol.com/javadev/terminology.html for an explanation of terminology. These correspond direcly to objects within the library and the result of parsing a packet will be these SNAC objects. The type of SNACs and information contained within is then manipulated and used to change the state of the Client object internally or to signal to the User Interface of an event. All events were implemented with a base Event class and hierarchy of event classes derived from this. The protocol is poorly documented and what documentation there is is thin and often bug-ridden. I had to put a lot of work into figuring out parts of the protocol and research into this to begin with. I wrote a whole system for deconstructing packets into their constituent parts and from this I could more easily decipher streams of communication between the client and the server.

The coding of the GUI was developed partly alongside the library when enough was working, with complex widgets being created (constructed out of gtk widgets) to represent the Message Box, the Contact List and other dialog boxes involved. I challenged the normal instant messaging paradigm and reached new, and what I believe to be, easier and moreefficient ways to communicate. The key one being the move towards an IRC-style, continuous chat window for messaging which is faster and smoother to use.

Results

In the end I released the first release of my library and client on the 1st October and I have been updating it since, adding more features and steadily improving it. The whole project together was 'christened' ickleand the homepage is http://ickle.sourceforge.net/.

I have had a lot of feedback from users for improvements, bug reports and feature requests and have recently had other people joining the project to contribute code. Within the last few weeks there has been a large shift over to the new protocol after ICQ dropped most support for older clients, and consequently all the older protocol based Linux clones. A sure sign of the success of the project has been its steady climb up the Sourceforge rankings to reach a peak of 15th a week ago (out of a total of almost 30,000 Sourceforge projects) and it is now staying within the top 30 projects or so.

Paper Submitted by:

Dwight Tuinstra

(tuinstra@clarkson.edu)

University:

Clarkson University

Country:

USA

An LFS cleaner framework for Linux and other UNIX-like operating systems

30th November 2001

Dwight Tuinstra *(tuinstra@clarkson.edu)*
Clarkson University
Potsdam, New York 13699

Category of Project: File System Improvements

Background

Log-structured filesystems (or "LFSes") [5, 8] preserve the inode semantics of traditional file systems, but take a radical approach to placement of data on the drive. Data and metadata are written sequentially to a log. The log is not a staging area for later insertion into stable store, but rather is the stable store itself. This avoids the high costs of seeks suffered by file systems that attempt to carefully place data onto the disk. Additionally, a log structure enables development of features such as rapid crash recovery, file system cloning, and file "undelete" operations.

LFSes also write updates to the head of log rather than performing an update-in-place. The position of blocks containing the updated data is noted in the log, implicitly invalidating the old blocks. A form of garbage collection is needed to reclaim space taken by invalidated blocks. This is performed by a user-space utility called a "cleaner". Design and implementation of cleaners is one of the primary challenges in creating an efficient and robust LFS.

At present, there appears to be only one complete and working open-source LFS implementation: that of the NetBSD operating system [7]. Linux has the LinLog LFS effort [1, 2], but it lacks a fully operational cleaner. Initial work on a cleaner has been done[3], however it is preliminary and development appears stalled. The Linux-based Swarm/Sting project was described in 1999 [6], but a public release of the code seems not to have happened yet.

Part 1: Project and Objectives

This project seeks to provide a robust, well-documented, fully-functional cleaner for the LinLog LFS; and in the process produce a general framework for constructing LFS cleaners.

Part 2: Methodology and Design

Rather than push forward on a preliminary and unproven native Linux cleaner, the approach is to generalize the NetBSD cleaner into an LFS cleaner framework, then specialize that framework for the Linux LinLog LFS. The general framework will be released under a BSD-style license since it is derived from BSD-licensed code and will be donated back to the NetBSD project. Linux-specific portions will be released under the GPL. The BSD license is an open source license.

Language Any kernel components will be written in C. C++ will be used for the rest, for its support of object-oriented programming, improved error-handling through exceptions, and better memory management through constructors and destructors (cleaners make heavy use of scratch memory). The improved modularity and facilitation of code refactoring should enable implementation of LFS enhancements proposed in the literature [4].

Project Phases (1) Refactor the existing NetBSD cleaner to simplify it and split it into general and NetBSD-specific components. (2) Release/donation of mid-beta-quality refactored code to the NetBSD project. Anticipated date: January 2002. (3) Implement the specialization classes for LinLog. Adjust the framework as necessary to tease out true generalizations, back-porting these to the NetBSD cleaner. (4) Release/donation of mid-beta-quality code to the LinLog project. Anticipated date: February/March 2002. (5) Develop kernel portions of LinLog to support improved performance (such as efficient provision of a "seguse table" as mentioned below). Anticipated start date: March 2002.

Design The design models the high-level objects in an LFS, encapsulating the implementation details of a specific LFS. The general LFS cleaner acts on these objects. Major classes are as follows:

Cleaner_ctl: models all control parameters that affect the behavior of the cleaner, such as timeout periods, amount of cleaning per cleaner wakeup, debug and logging levels, and so on.

LFS_info: models high-level information about a mounted LFS.

Inode_map: Since LFSes write modified inodes to the head of the log, there must be a means of discovering the on-disk location of a given inode. The Inode_map models the LFS implementation structure that contains this mapping.

Segment: Writing and cleaning in an LFS takes place at the level of "segments", which contain data blocks and inodes from (possibly multiple) files. The LFS on-disk partition is modelled as an array of Segments.

Seguse_table: This is a fast-access summary of usage and aging statistics for each segment, updated dynamically by the LFS kernel components. The cleaner reads this information to decide which segments to clean. (The LinLog LFS lacks this table, making any cleaning extremely inefficient. Until this feature can be added to LinLog's kernel components, the LinLog implementation of this class will require accessing the actual on-disk segments.)

Part 3: Results

The project is presently in phase one, abstracting and generalizing theNetBSD cleaner code. This is a non-trivial task due to layers of maintenance and features added to the original 1992 BSD 4.4 code [7]. Current work, including UML diagrams, can be found at [10]. On a previous refactoring attempt (prior to the goal of constructing a general framework), approximately half of each of the two major cleaner code files (*cleanerd.c* and *library.c*) had been reconstructed in an object-oriented fashion and were running in regular use on the author's workstation.

References

[1] Czezatke and Ertl, "LinLogFS - A Log-Structured Filesystem for Linux". *Proceedings of the 2000 USENIX Annual Technical Conference (Freenix Track),* 2000.

[2] Czezatke and Ertl, "LinLogFS - A Log-Structured Filesystem for Linux" (web site). http://www.complang.tuwien.ac.at/czezatke/lfs.html.

[3] Gatwood, "The Linux Log-structured Cleaner Project" (web site). http://www.gatwood.net/projects/linlog.html.

[4] Matthews, Roselli, Costello, Wang and Anderson, "Improving the Performance of Log-Structured Files Systems with Adaptive Methods". *Proceedings of the 16th ACM Symposioum on Operating Systems Principles,* 1997.

[5] McKusick, Bostic, Karels, and Quarterman, Chapter 8 of *The Design and Implementation of the 4.4 BSD Operating System.* Addison-Wesley, 1996.

[6] Murdock and Hartman, "Swarm: A Log-Structured Storage System for Linux". *Proceedings of the 2000 USENIX Annual Technical Conference (Freenix Track),* 2000.

[7] Various contributors, The NetBSD Project's LFS cleaner. (cvsweb site). http://cvsweb.netbsd.org/bsdweb.cgi/basesrc/libexec/lfs_cleanerd.

[8] Rosenblum and Ousterhout, "The Design and Implementation of a Log-Structured File System". *ACM Transactions on Computer Systems,* volume 10 number 1, Feb 1992.

[9] Sargent and Cain, "Implementing Cleaning in a Linux Log-Structured File system". http://www.cs.wisc.edu/~cain/mrclean.ps.

[10] Tuinstra, The Generic LFS Cleaner site (web site). http://www.clarkson.edu/~tuinstra/genclean.

Paper Submitted by:

Phillip Allen

University:

Clarkson University

Country:

USA

Other Challenge' Development Library

Threadpool: A reusable pool of general-purpose threads
Full text: http://www.clarkson.edu/~allenpd/threadpool/paper.html

Part 1

The current trend in computing technology is towards creating robust servers for client server applications, such as scientific computing or the World Wide Web. Many server applications are multithreaded; unfortunately, thread programming is difficult [3]. One way to reduce the complexity of these programs is to use threads as a work crew [2]. Since servers typically handle a lot of traffic, a server could potentially spawn a large number of threads on the fly. This can have a large overhead if it is preformed on short lived operations. A better solution is to create a pool of threads before hand. Each time a new client task enters the system, an idle thread will wakeup and perform the task. This strategy is particularly effective because it amortizes the thread creation/destruction overhead over many client tasks and reduces the instruction cost of starting the client task, after the server initializes, to a mere procedure call. This method of thread Organization is called a threadpool. The goal of our project is to create a generic threadpool library for POSIX threads (pthreads).

We built the threadpool library to satisfy the following requirements for a threadpool are as follows:

1. It must execute all dispatched functions exactly once.
2. When dispatch is called, the work to be done is placed into a queue and dispatch returns. The only time that dispatch blocks is when the queue uses at least 64kB of heap space and is full; in that case, the act of adding information to the queue will block. This prevents the program from crashing due to lack of heap space.
3. There can be no deadlock conditions in the threadpool. Experiencing deadlock due to threads is a potential occurrence but it should not be caused by a threadpool package itself.
4. There should be no busy-wait loops in the threadpool. A busy-wait loop has the tendency to be slow; in some cases, a two-phase wait is faster than blocking [1]; however, programming around busy-wait loops is just as effective.
5. A simple, clean API. To vie the API please visit:
   http://www.clarkson.edu/~allenpd/threadpool/API.html
   The only additional requirement on the threadpool is that the programmer does not attempt to execute threadpool functions through the dispatch of threadpool. Any such function calls will immediately return with no code executed.

Part 2

The threadpool is written in C and uses pthreads (POSIX threads) to handle threading; it ports without change to many flavors of UNIX. There are essentially two parts. First is a queue that is dynamic in size, instead of destroying a node after use with the free command, it stores the unused nodes in a separate queue. This removes the overhead of repeated malloc and free calls. The second part is the actual threadpool functions; with the use of a mutex and three condition variables, there are no deadlock or busy-wait conditions and the code behaves as we designed it to. The total amount of code, without comments or debugging statements, is only 400 lines, half of which is the customized queue.

To add a work crew to a single threaded server only takes three lines, which should greatly ease the task of application developers.

Part 3

The threadpool was tested using a very basic server application; it accepted TCP/IP requests, did an amount of work set at server creation, and then responded to the request. The clients were simple applications that made a request to the server and then waited for a response. If the server dropped a request, it would close down its connection and start a new request. The network connecting the test machines is a switched Ethernet running at a minimum of 100 Mbps; tests were conducted at night in order to minimize the effect of other traffic. The time to send one packet across this connection was 2.230ms. Our tests ranged from I/O intensive to computation intensive applications; in all cases, the addition of threads to the threadpool showed superior performance to the monolithic implementation. For a complete description of the machines used for testing and the results of our tests, please visit:
http://www.clarkson.edu/~allenpd/threadpool/testing.html

Our threadpool provides a simple interface for an application writer to create multithreaded applications. People tend to implement their own threadpools when they need them. It is hard to get a threadpool right without extensive testing and proofs of correctness. Therefore, we believe a solid, fully tested version of a threadpool is a valuable asset to the open source community.

References

1. Arpaci-Dusseau, Andrea C. Scheduling with Implicit Information in Distributed Systems. ACM Transactions on Computer Systems, Vol. 19, No 3, 2001,pp.283-331.

2. Birrell, Andrew D. An Introduction to Programming with Threads. Digital SRC Research Report 35.

3. Ousterhout, John. Why Threads Are A Bad Idea (for most purposes) Invited Talk at the 1996 USENIX Technical Conference (January 25, 1996).

Paper Submitted by:

Rimon Barr
(Rimon Barr <barr@cs.cornell.edu>)

University:

Cornell University

Country:

USA

rImap - Remote IMAP
On the Need for a Generic Open-Source Mail Replication Engine

November 27, 2001

Introduction: Motivation and Objective

The Linux platform and open-source [Perens 97] in general have revolutionized the software industry by lowering many barriers to entry. One key advantage of this movement is that individuals can make incremental contributions to an ever-increasing base of software, rather than being forced to rewrite and reengineer entire systems from scratch. While component technologies have fulfilled some of this promise, open-source is proving to be far more effective. The need for a generic, open-source mail replication engine is motivated in this context.

The appeal and value of the Linux platform increases as a function of the end-user applications and functionality that it supports. Currently, there is a dire lack of Linux desktop productivity applications, and many software development initiatives are working furiously to address this issue. Mail processing is one such critical, end-user, desktop function that has not been sufficiently addressed. Specifically, the majority of mail processing applications does not support mail replication functionality: the ability to synchronize mail replicas at multiple sites.

Mail replication is a necessity in many mail applications for numerous reasons [Demers et al. 94]. Mail replication across servers provides fault-tolerance and availability in the face of node and network failures. Replication can provide increased performance, through load-balancing and latency-aware replica selection. Lastly, replication allows users to process mail while offline and disconnected, which is often the foremost criterion for selecting mail clients in an increasingly mobile world. Many fully featured and popular mail programs, such as Pine, elm, Mutt, mail and various graphical clients, do not support disconnected mail processing.

The purpose of this project is to create a flexible, generic mail replicator that can be easily integrated into existing mail applications. In the remainder of this paper, I will briefly discuss the design of the system, its implementation, and some interesting issues and results I have accumulated from its everyday use for the past few months.

System Design

The system is designed to synchronize messages among replicas. It is a generic, abstract synchronization framework with concrete implementations for specific types of mail stores and message formats. Implementations of the generic *replica interface* have been created for the standard Unix mail file format and for IMAP compliant mail servers, and new replica types can be added to the system with ease.

Synchronization can occur between any two kinds of replicas, as they are all treated uniformly. Replicas *IBM Linux Challenge Contest Entry Page 2 of 3 rImap – Remote IMAP Rimon Barr* are instantiated and parameterized via a URI registered with the system. For example, an IMAP replica instance is specified as:
imap://[username[:password]@][server[:port][/path]

The generic synchronization engine assumes that each replica has a folder structure, with messages in each folder. Replicas support opening and closing of folders, as well listing folder contents, adding and deleting messages, locking, etc. The synchronizer can be made to work with arbitrary objects, although it has been tailored to work with messages. For example, the system emits useful message header information as they are processed, shown in Figure 1. All that is required of an object is a unique object identifier and serializability. Messages are serializable, by definition, and contain a globally unique **MessageID** header [Crocker 82]. If one does not exist, a key is generated by the system.

```
INBOX: local - 12, cucs - 14, 1 xx, 3 <-
xx 27-Nov JavaWorld's Mic (4579) Nokia calls on Java developers deleted.
<- 27-Nov Linux Today (16K) YOUR LINUX TODAY NEWSLETTER FOR No copied.
<- 27-Nov Karen (1691) John Stewart waiting list copied.
<- 27-Nov Melissa (3495) Re: hi copied.
Drafts: local - 1, cucs - 2, 1 ->
-> 27-Nov Rimon Barr (2610) Re: IBM Linux Scholar Challenge copied.
Sent Items: local – 146, cucs - 146, 1 ->, 1 <-
-> 27-Nov Rimon (1411) Re: chat script for logging on to copied.
<- 27-Nov Rimon (2092) Re: by the way... copied.
** 26-Nov Rimon (99K) osr15 flagged.
Bin: local - 667, cucs – 669, 2 <-
<- 27-Nov Red Hat Network (3682) RHN Errata Alert: Updated wu-ftpd copied.
<- 27-Nov Elliott (1398) by the way... copied.
```

Figure 1: Sample run of rImap on personal mail folders

The synchronizer maintains the state of a folder pair to determine whether a difference between two replicas is due to an insertion of a message into one replica or a deletion from the other. The system then proceeds to perform the required message transfers and deletions between the two folders that are out of synchronization, and it also takes care of updating message flags and converting among message formats. The system provides other features, such as unidirectional transfers and limiting message transfers by their size, which are useful over slow modem links.

Lastly, the system can be integrated into an existing mail client or server program, or be run standalone. The integrated version supports a simple API. The synchronizer need only be provided with appropriate replica definitions, which can make up-calls into the client or modify the message store directly. Command-line parameters and a short configuration file drive the stand-alone version. A configuration file is used to define and name replicas and group folders into named sets, which shortens the length of the command-line. A better description of the stand-alone version, with detailed usage instructions, is available in the attached HOWTO document [Barr 01].

System Implementation

The entire system is implemented in Python, and currently consists of 1500 lines of code. This includes the command-line and configuration file processing, message parsing routines, synchronization and locking, output formatting, and replica definitions for interacting with both IMAP compatible servers and *IBM Linux Challenge Contest Entry Page 3 of 3 rImap – Remote IMAP Rimon Barr* standard Unix mail files. Python was chosen for rapid development and robustness. A quick perusal of the Pine source code shows that merely Pine's Unix mail file parsing code is an order of magnitude greater in number of lines of code than the entire system. Automatic memory allocation, string processing and many programmer shortcuts and libraries provided within Python result in a robust end product. Moreover, when written correctly, these benefits do not come at the expense of application performance. For example, reading a large 25Mb mailbox file is performed without a noticeable difference in performance, when compared to analogous Pine and Netscape Messenger code written in C. The entire execution is IO-bound running on a 200Mhz Pentium. In general, the choice of Python has greatly assisted rapid development and increased program robustness, without sacrificing application performance. Integration into existing mail applications is straightforward. Loose integration can be achieved by spawning a process that runs the stand-alone version of the system. A tighter integration is possible through Python's C, C++ and Java language bindings. This only requires writing a specific replica implementation that would interact with the mail client via client-specific up-calls. In addition to the robustness gained through the use of a high-language, the system is written in a failsafe manner. The system has been in daily use on my personal e-mail, and despite the occasional specification incompatibility or coding error uncovered, I have yet to lose a single message. It is now successfully being used by a small group of users in configurations that I have not tested. For example, it was developed against a Microsoft Exchange Server, and recently independently verified against Cyrus IMAPd. The system has been tested primarily on Linux, but is used by some on Windows NT.

Conclusion and Future Work

Many ideas for future development have been included with the program source code. Sub-projects include providing support for POP servers and support for synchronization with popular Web-based e-mail systems via HTTP.

In this paper, I have briefly motivated the need for a generic, open-source mail replicator and described its design and implementation. Based on feedback from users and from personal experience, I believe that it represents a useful, practical contribution. The project source code is fully functioning and is included with this contest entry.

References

[Barr 01] Rimon Barr, HOWTO: Offline Mailing Method using rImap, http://www.cs.cornell.edu/barr/repository/rimap/rimap-offline-mailing.txt

[Crispin 96] Mark Crispin, RFC2060: Internet Message Access Protocol, http://www.faqs.org/rfcs/rfc2060.html

[Crocker 82] David H. Crocker, RFC822: Standard for the Format of ARPA Internet Text Messages, http://www.faqs.org/rfcs/rfc822.html.

[Demers et al. 94] Alan Demers, Karin Petersen, Mike Spreitzer, Doug Terry, Marvin Theimer and Brent Welch, The Bayou architecture: Support for data sharing among mobile users. In Proceedings of the IEEE Workshop on Mobile Computing Systems & Applications. Aug. 1994

[Perens 97] Bruce Perens, The Open Source Definition, Http://opensource.org/docs/definition.html

Paper Submitted by:

Bryan Clark

University:

Clarkson University

Country:

USA

Kernel User Resource Tracking in the Linux Kernel

Part 1 - Describe the project and the objectives you are trying to accomplish.

User Resource Tracking (URT), the monitoring and administering of certain resources such as number of processes and open files, is extremely important to keeping an efficient work environment across a multi-user system.

This system if implemented correctly could provide a valuable resource to companies looking to buy a single large server from where they can run all of their applications and store all of their employee's data, however without limitation on resources a user could bog down the system with their processes when it is server mission critical applications. With a resource limiting system user memory, number of processes and number of open files can be monitored and controlled to allow the server to provide it's main function without interruption. Eventually this can become a group/user tracking system where the company can create groups based on resource needs and users can then share only the resources allocated to them. I think this would be exceptional in an academic or workplace environment where you have different levels of users and different priorities of functions, yet the organization desires centralized data.

Part 2 - Describe the methodology you used in working on you project. (This would include any research you did, resources you used, team or open source community input, etc.)

I first realized the idea of this in March of 2001 when I was hacking my way through the Linux kernel for an operating systems class project and I came across a comment by Linus Torvalds in the <include/sched.h> file were he mentioned the possibility of a complete user tracking system using the available structures that branch off the system task structure. Then when I enrolled in a graduate level Advanced Operating Systems course were we read and discuss many historical and current operating system research papers we were given the assignment for a project of which we could also sign up into the Linux Challenge.

Some of our class time has been spent discussing the high-level project ideas for the challenge and we try to hash out the best method possible, also as a good measure we had to research the concept of our idea everywhere possible to ensure it's originality. I took on a partner for this project because I wanted a set of interaction tools and more input on how to work the idea. My partner and I split up the work from the beginning and then bounced ideas back and forth into the wee hours of the morning. The development environment used a VMWare Workstation for Windows 2000 running Mandrake Linux on which we ran the experimental kernels builds. VMWare gave us the ability to experimentation without the risk of losing everything due to an error like limiting the root's available processes to 3. Of course the VMWare wasn't run on a powerful machine and inherently runs slow, so all our compiling was done on a six-processor Beowulf cluster running Debian Linux.

Part 3 - Results you were able to achieve. (Strikethrough: summarize the following information to help assist our judges.)

Our resource and limitations successfully implemented were for processes, file streams, and sockets. Through the interface program provided by my partner you can change the maximum resources for any user at anytime, also you can read the current resources being used up. If a user were to try and run more processes than were available to him, the call would be cancelled and an error message appears stated there are no more processes left. Similar events happen for the files and sockets.

User Resource Tracking in the Linux Kernel

Abstract

Since the beginning of multi-user and time sharing operating systems like UNIX and it's predecessors there has been a need to make system resources available to every user and to organize this usage in the most efficient manner. User Resource Tracking (URT), the monitoring and administering of certain resources such as number of processes and open files, is extremely important to keeping an efficient work environment across a multi-user system. An operating system kernel must track all of it's resources to ensure that it does not exceed the bounds of the machine on which it is running. The URT uses a similar technique to ensure that a user does not exceed the bounds of system resources that have been set by their system administrator.

Introduction

User Resource Tracking (URT) is the monitoring and administering of certain resources such as number of processes, open files and other critical resources like memory. To see what the URT does first lets examine what has been done previously and how the system builds and improve off of that. The Linux kernel already has resource limiting controls that prevents the kernel from allocating more resources to any user or program than the machine can physically handle. This is a system wide effect for all users, even the root and this does not limit and specific user to a certain type or set of resources.

PAM [1], a pluggable authentication module system, provides various resource limitations for different users on a per login basis. An obvious flaw to this technique is that your users can simply login in to more than one terminal and gain more resources than the desired allocation. The major difference between the URT and PAM is that the checks of the URT are within the kernel, which gives us a system wide tracking of users instead of a shell-based limit.

In order to provide the most robust service for tracking I worked closely with Stephen Evanchik, another IBM Linux Scholar Challenge registrant, who is building the techniques of user tracking into a complete system. Steven's Resource Management System runs on top of the URT and controls the limiting structure and adds a grouping system for users. As a whole the URT aims to provide the most accurate tracking of user resources, with that in place it easily adapts to limiting of these resources being tracked. At present there are no resource tracking systems of this type that we are aware of.

Approach

The author feels that the best approach is the following:

- Maximize accuracy of accounting by following existing kernel methods and not implementing new ones, which are subject to programmer error.
- Achieve the best system performance by not adding new structures that must be maintained in kernel space, only adding new variables on existing structures.
- Provide the best compatibility by using existing structures, thus there are fewer modifications to make to other functions and structures.

Design

Tracking control variables were placed into the user_struct structure for processes, files and sockets. The control variables included were the maximum values allowed, as well as the current value per user for each resource tracked. Limiting controls are then placed at the various functions were allocations and removals occur according to the resource being tracked. To maintain as close a relationship as possible with the current kernel system resource control the same error values are returned upon error in the URT, as are in these system checks. The same error codes were used so that the error messages returned to the user would be the same as if the system resources were completely used up to the point were the kernel could not allocate more resources of the specified type requested.

An important consideration and one discovered very quickly when the computer refused to continue booting was that the root user must not be limited in any way, all of the limitation controls have been wrapped with a user id check before checking for available resource usage. As a statistical concern the tracking variables for the root user increment and decrement and may be viewed to show how many resources the root user is consuming, but there is no limitation because the root user is responsible for many system services that must run without limits.

For default system values the kernel configurator was edited to allow changes to the default values before kernel compile. To change the user limits and to read the current usage during runtime a program can be used accesses the system call that changes the values of the limits for any user as well reads the current resource usage values of any users.

Current Status

Currently, the URT of processes, file streams and sockets is complete with tracking and limiting both finished on a per user basis. Under development right now are two more major resource issues of memory and CPU time. Some future concerns are the default limiter; this should default to an infinite value so as not to limit users who do not require this functionality.

1. Http://www.us.kernel.org/pub/linux/libs/pam/

Paper Submitted by:

Mike Schellhase

(mcs2@andrew.cmu.edu)

University:

Mellon University

Country:

USA

Software Testing
Rtest: Race-Free Software Testing

I. Project Description and Objectives

Complex software regression testing tools allow a developer to test a program against a set of inputs. But what if the developer wants to send the program being tested signals sent at a specific point in the execution of the process or synchronize some other event during the process execution?

In fully automated software testing, a need often arises to do some synchronization with the program you are trying to test to avoid race conditions. The testing program may want to send signals or communicate via pipes with the program being tested and some way of making sure that the program being tested is in a "ready" state may need to be developed. The specific problem that led to this project involved an automated checking of a shell for a computer systems class at Carnegie Mellon University. The grading was done by creating 15 trace files and comparing the output of a student's shell to the output of a reference implementation when fed the same trace file. The trace files included executing basic shell commands, external programs, and also sending signals to the shell. Specifically, the trace files tested the shell's handling of the SIGINT and SIGSTOP Signals. However, a problem arose when testing signals via these trace files of when to deliver the signal. If the shell receives the command for the program to execute but does not yet fork off the new process or create the new job before the signal is sent, the signal will not have its desired effect. A similar undesired effect can occur if the shell has already forked off the new process and exec'd the new program. The new program may already have finished and so a SIGINT signal would effectively be ignored.

Obviously, there are some inconsistencies in the order of events that a successful tester must deal with. But how can this problem be avoided? How can we tell if the parent will be scheduled before the child and so the new program will possibly miss receiving the signal? Or how can we tell if the child will be scheduled first and possibly finish before the signal is sent? We must find a way to control the states of the 2 processes so that we can assure that the child will be in a "ready" state to receive the signal. This problem has a much broader impact than testing student's implementation of a shell. Any program being tested by regression testing is not being sufficiently tested if it does not test the handling of signals and other asynchronous inputs to the program. Rtest is my attempt to solve this problem by synchronizing the tester and the program being tested.

II. Methodology

Rtest deals with the synchronization issues between the tester and the program being tested (hereafter referred to as the client) by stopping the client in known states. To accomplish this, rtest uses ptrace(), a low level system call that allows a parent process to observe and control the execution of a child. First, a child process is forked off to run the client program in. The child then indicates that it wants to be traced by calling ptrace. By design, ptrace will cause the child process to stop when it reaches the exec call to spawn the client. The server process begins by simply waiting for the child to stop with waitpid.When this function returns, we will be in a synchronized state in which the parent can start and stop the client and issue commands at the appropriate time. The parent can now continue the child's execution of the client and supply it with input via pipes and redirecting stdin and stdout.

To synchronize events while submitting inputs or sending signals, ptrace has the ability to stop the client process at entry and exit to system calls. We can then decide what the system call is and respond accordingly. After the inputs and signals have been delivered to the child process, the outputs can be read via a pipe and the outputs compared to some reference output. The project was developed on an Intel PIII 550 MHz machine running Red Hat Linux 6.2 (Linux kernel 2.2.19). It required research of system calls, specifically ptrace parameters and behaviors, and inter-process communication, specifically pipes. A collection of web resources, man pages, and existing header files was used to do this research.

III. Results of the Project

Rtest successfully uses ptrace to synchronize the execution of the client and server processes. Because the client stops execution at a known point in the parent and can be stopped at every subsequent system call (including forking off another child process), the client can be controlled and ensured to be in a "ready" state at a given point in the servers execution. Also, because of the flexible nature of the trace files, which provide input To the client, rtest can test almost any command line program.

In conclusion, the ability to synchronize processes being tested can improve the effectiveness of software testing, which will lead to higher quality software for the Linux environment.

Paper Submitted by:

Nick Pattengale

University:

New Mexico Institute of Mining and Technology

Country:

USA

WindowManagers

1. Main Objectives

Interface improvements could distinguish Linux from all other Operating Systems. While Linux kernel internals feature novel approaches to OS design & implementation, Linux application development has followed a much more traditional path. The elegance of the underlying OS is lost upon the user through application interfaces similar to those running under any other OS. For example,there are as many as five variants of "office" packages that are available for Linux. By playing "interface follow the leader," Linux application developers have repeatedly implemented common interface design errors.

The field of HCI (Human Computer Interaction) has a healthy literature that has been under-utilized by today's Linux community. It is the purpose of this study to show that by implementing superior interface design principles, developers could make Linux stand out from the pack as the premiere computing environment.

2. Methodology

There are a variety of techniques that can be used to improve interface efficiency. It may help for the purposes of this study to break down improvements into two categories: Operational Improvements & Functional Improvements.

Operational improvements, in the context of this study, consist of changes to the visual computing environment. For example, using a technique called "indication" significantly reduces the amount of unnecessary clicking found in traditional interfaces. Indication is realized by keeping track of the location of the GID(Graphical Input Device, e.g. the mouse) and visually distinguishing the element that would be selected if the user were to click the GID.

Functional improvements, once again in the context of this study, are changes to methods of data manipulation. For example, in the environment implemented for this study, a spell-check can be invoked from anywhere that a spell-check command makes sense. This is a substantial improvement over the typical Linux user environment where a invoking a spell-check requires knowing the interface to a particular application. Further illustrating improvement is that instead of having to learn a new spell-check method for each different word-processing package, the means for realizing a spell-check is ALWAYS the same from ANYWHERE.

By developing software that strictly adheres to the improvements outlined in this paper, the author expects to see quantitative gains in interface efficiency.

3. Research

IDEAS (Interface Design Efficiency Assessment System) was written by the author of this paper as a test-bed for interface improvements. IDEAS has as its central concept the ActiveDesktop which, in the context of this study, can be thought of as a prototype X11 WindowManager. The author feels that IDEAS is a demonstration of useful techniques and has therefore licensed the software under the GNU GPL (General Public License) and registered the project at OSDN (Open Source Developer Network) SourceForge (http://ideasys.sourceforge.net). In implementation, IDEAS is a set of Java classes that can be run either as a standalone application or as an applet from within a web browser. Accordingly, as long as you have the Java2 Plug-in installed in your web browser you are urged to view the ActiveDesktop Demo on the IDEAS website. The ActiveDesktop demo will allow the user to try the technique for spell-checking outlined in this paper.

In order to quantitatively measure interface efficiency improvements offered by the ActiveDesktop, the author chose the Keystroke Level Model GOMS method. KLM-GOMS is a scientifically tested method for predicting typical timings for computing tasks requiring user interaction.

The KLM-GOMS analysis will serve the purpose of evaluating one improvement offered by the ActiveDesktop. The ActiveDesktop, however, contains vastly improved functionality that won't be quantitatively analyzed. Such improved functionality consist of:

Operational Improvements:
- (1) Indication, and
- (2) Multiple Selection

Functional Improvements:
- (1) Execution of 0th selection, and
- (2) Transformation

These improvements are best described in context.

Spell-checking a text selection from the ActiveDesktop requires two steps: Selecting the text and Executing a "SpellCheck" command.

- **Selecting the Text**

Selecting the text to spell-check is identical to the typical "drag a selection box around the text" procedure that we're all used to. Seems familiar, doesn't it? However there is an improvement lurking in the wings. A new selection on the ActiveDesktop is treated as the "0th selection." Contrary to current systems in which if you were to make a new selection you would lose the original selection, the ActiveDesktop supports "Multiple Selection." Multiple Selection simply means that when a selection is made, that selection becomes the 0th selection, thereby making the last 0th selection now the1st selection. The scheme can be thought of as a "SelectionStack." Multiple selection allows us to execute the 0th selection as a command that operates on the 1st..nth selections.

- **Executing a SpellCheck Command**

Executing a SpellCheck command consists simply of typing the command "SpellCheck," selecting it, and pressing the EXECUTE key (F1 on the ActiveDesktop). Pressing the EXECUTE button "executes" the 0th selection. Execution is done by IDEAS' ExecutionEngine which looks up "SpellCheck" and recognizes it as a command that equates to performing a spell-check on the 1st selection (the selected text, which was bumped to the 1st selection as a result of selecting the text "SpellCheck"). Was that not easy?

So what would have happened if the 1st selection wasn't text and was, say, a graphic? This is where the functional improvement Transformation rears its convenient head. Transformation occurs when an operand of a command isn't of the correct data type. In the case of performing a "SpellCheck"command on a graphic, the TransformationEngine would look to see if it contained a transformer for graphic->text. If such a transformer existed (OCR), the transformation would be automatically invoked. So spell-checking a graphic is just as easy as spell-checking text!

4. Results

As for a quantitative analysis typical of the numerous improvements offered by the ActiveDesktop, a KLM-GOMS analysis of a spell-check on the ActiveDesktop looks like this:

HPKPKPKPKHK - 5.1 seconds Whereas a typical Linux spell-check technique of boring text is:

HPKPKMPKP - 6.75 seconds

That is a 25% improvement.

Now consider spell-checking a graphic. Using current methods would require a whole new technique.   Namely, it would require invoking a separate OCR application and cutting and pasting the resultant text. Or maybe a spell-check interface is native to the OCR package? The latter scenario would present yet another interface to learn. Remember that by using the ActiveDesktop there is no difference in technique!

The previous paragraph sheds light on the substantial (other than simply "time-saving") improvements offered by the scenario stated in this paper. Consider the execution/ transformation engine as a uniform approach to data manipulation. Such a framework eliminates the need to learn application-specific features. By making the procedure for using computing as habitual as possible, the user focus is shifted from learning how to use a program to actually doing productive work. It is the author's conjecture that such improvements could make the Linux windowed environment stand out as the most productive and least error-prone environment available. It should be noted in closing that the majority of the incredible software base provided to (and by) the Linux community would not be lost in this proposed situation, as effort would only be needed to wrap current functionality with the execution/transformation framework.