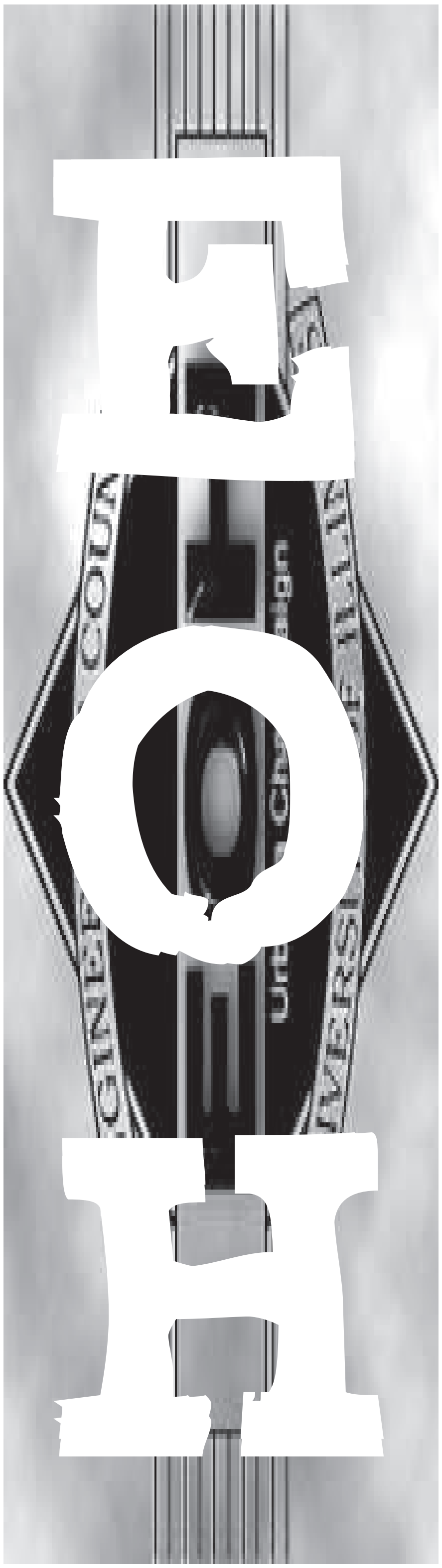
The Journal of the Association for Computing Machinery at UIUC: The

Banks of the Boneyard



ACM & EOH

by Mark Ashton

ACM always has a huge presence at the Engineering Open House, and this year will be no exception. EOH 98 will be featuring some of the most impressive and varied projects ACM has ever delivered.

In the offering is everything from robotics to fighting games. In between are a whole slew of efforts covering some of the coolest aspects of computer hardware and software. We'll also have running events and displays for your enjoyment and edification. Be sure to attend as much as you can; you won't want to miss any of it! Here's a quick guide to the projects being presented. You can get more information from each SIG's article in this issue.

Projects (alphabetical by SIG):

Be Users Group:

SoundMangler - a real time audio filtering application for BeOS.

MacWarriors:

QTVR Walk-through - a virtual reality walk-through of DCL.

SIGArch:

The Return of the Big Blue Board - a huge 4 bit adder designed to teach people about digital logic.

Power Glove Tic Tac Toe - play tic tac toe with a Nintendo Power Glove.

SIGArt:

RedBeard - a robot that builds a cognitive map of its environment and uses this map to navigate intelligently to a destination.

continued on page 2...

SIGBio:
Virtual Anatomy Textbook - a web-based anatomy textbook intended for college and high school students.

SIGDave:
OozeFighter - a 3D polygon-based fighting game for Win32, using DirectX.

SIGNet:
Voodoo City - a web-based gaming and chat server.

SigOPS:
Fishnet: A Distributed Fish Environment - a multiplatform distributed fish tank running on homebuilt and commercial OSes.

SIGSoft:
VIDEX - VIDEX is an integrated development environment for X. VIDEX wraps popular existing tools into a unified development experience.

SIGVR:
VRMLMazeHunt - a multiplayer maze game based on the Virtual Reality Modeling Language.

For more information on ACM and/or EOH, check out:

http://www.acm.uiuc.edu

http://eoh.cen.uiuc.edu

From the Chair

by Paul Swanson, ACM Chair

By the time you read this, ACM may already be showing its exhibits at the annual Engineering Open House. This time of year is always hectic for ACM Special Interest Group members, who invariably have procrastinated their big projects until it is finally necessary to spend 96 hours straight coding or soldering or painting. Descriptions of these projects can be found in the rest of this issue of the Banks.

Aside from that, next month are ACM@UIUC's annual elections for the top four positions of Chair, Vice Chair, Secretary, and Treasurer. All four of these positions are required to attend 3 weekly office hours, the weekly top four meeting, and the weekly executive meetings. Specific responsibilities for each position are listed below:

Chair: The chair is ultimately responsible for everything that ACM@UIUC does. More specifically, though, the chair runs the weekly top four and executive board meetings, sets the agenda for and runs monthly general meetings, and is the main point-of-contact for the group at large.

Vice Chair: The vice chair runs ACM in the chair's absence. As a general fix-it guy, the vice chair needs to know everything that's going on in ACM, and his role as advisor is essential to making sure events go smoothly.

Secretary: The secretary is responsible for keeping ACM records, including minutes taken at ACM meetings.

Treasurer: The treasurer maintains ACM's financial integrity and provides regular information about ACM's financial status.

If you are interested in any of these positions, email me at acm@uiuc.edu. Any ACM member is eligible to run, and if you think you can provide good service to ACM over the course of the next year, I encourage you to do so. All ACM members who are interested in securing the future of ACM should attend the next General Meeting on April 2nd, 1998, at 6 P.M., to cast your vote for these top four positions.

SIGNet

by Jason Govig

Welcome to SigNet, the special interest group for networking and security! I am your new co-chair, Jason Govig. Together, Chris Sims and I will bring you to networking heaven.

In the beginning of the semester, we formed a new web site, Voodoo City. (http://www.acm.uiuc.edu/voodoo)

This new, hip, web-based, chat-enabled, Java-powered gaming site is in need of talented individuals with all levels of experience to help continue this project after EOH. From novices to network gurus, from designing simple graphics and web pages to writing networked client-server games, SigNet wants you!

For all of your networking needs, join us every Monday at 7:30 P.M. in 1102 DCL. Want more information? Email signet@uiuc.edu or check out our web page at http://www.acm.uiuc.edu/signet/

The Banks of the Boneyard is a student publication of the UIUC Chapter of the ACM, the First Society in Computing, and does not necessarily represent, in whole or in part, the views of the University of Illinois, the Department of Computer Science, or the International ACM. Partially paid for by SORF and the Department of Computer Science.

Editor and Layout & Design: Joe Doyle

The Association for Computing Machinery at UIUC
1225 Digital Computer Lab, MC-258
1304 W. Springfield Avenue
Urbana, IL 61801
(217) 333-5828

For more information, email acm@uiuc.edu
or visit our homepage at www.acm.uiuc.edu

The newsletter staff encourages ACM members to submit letters to the editor, articles, photographs and artwork for future issues of *The Banks of the Boneyard.* If you would like to contribute or participate in its creation, please email boneyard@acm.uiuc.edu.

In This Issue

From the editor: EOH is an exciting time for everyone, and especially for ACM. In this issue, one of the largest *Banks* I've ever seen, we tell you a bit about what we're doing for this year's EOH, along with some special feature articles ranging from MP3 encoding to making your own QuickTime VR movie.

The issue is roughly divided into two halves. The first half is the usual slew of SIG updates, along with a special FAQ about ACM@UIUC on page 8. The second half is comprised of EOH project descriptions and special feature articles written by some of our members. Enjoy!

SIG Updates:

EOH and Other Nifty Stuff:

SIG-BIO

by Robert Hyslop

This semester, SIGBio has been working hard to develop its latest addition to the Virtual Anatomy Textbook, our web-based interactive anatomy lesson, for exhibition in EOH '98. This year, we have decided to take a brief look at the digestive system (to complement the Lymphatic and Nervous systems done in the two previous years). Digestion is one of the most often thought about organ systems in the human body mainly because of all the problems it causes people on a daily basis. In previous years, we have sought to bring to light anatomical systems few people know or thing about, such as the Lymphatic (part of the body's extensive defense system). With digestion, we will cover all the basics and we hope to shed light on how and why things go wrong so often.

This year's exhibit will include several video segments along with an interactive Java based quiz to test your knowledge of the digestive system. As with the nervous system, navigation through the pages will be via a great Javascript ***menuing*** program that will allow instant access to any page in the site.

Don't miss the SIGBio exhibit. We will be located in the DCL atrium during the entire EOH.

http://www.acm.uiuc.edu/sigbio

SigArch

by Brian Burgner

Here at SIGArch, we truly believe bigger is better. Hailing from a long line of large hardware projects, such as the ACM soda machine, SIGArch has decided to go against the flow of the computer market. While industry is making .25 and .18 micron chips, SIGArch is making a 4 foot by 8 foot 4-bit adder. Although our adder is not a 600 MHz processor, and only has one function, it is very educational, and it just looks cool. This expanse of blue will be powered by TTL logic similar to that used in ECE 110 and ECE 249. Currently, we are in the process of soldering the 400 LED's that make up the mind boggling display of this Big Blue Board.

On a more advanced note, we are using a DEC StrongARM and a Xilinx FPGA to make Tic-Tac-Toe for the 90's. In remembering our roots however, we are using a PowerGlove for the user input. The StrongARM, which Digital Semiconductor generously donated to ACM, is a 200 MIPS RISC processor which can be powered by two AA batteries. The FPGA is one of the newest from Xilinx and was donated as well. It is a programmable chip that we can use to design nearly any piece of hardware we want to. We can't thank DEC or Xilinx enough for their generosity.

SIGArch meets at 7:00 P.M. every Tuesday and 9:00 P.M. every Thursday in L510 DCL. If you are interested in getting more information on any of our projects, want to be added to our mailing list, or interested in computer architecture, mail sigarch@uiuc.edu and we'll set

http://www.acm.uiuc.edu/sigarch

Sig Dave

by Michael Kolb

Hey again, ACM people. For the last few weeks SIGDave, the special interest group for short term silliness, has been hard at work on its EOH project -- the 3D fighting game engine. We've been cranking out the code to have something pretty to show off for EOH this week. Be sure to look for our exhibit!

Also, on Feb 28th, SIGDave and SigVR sponsored a one day code-a-thon to make a VRML 1.0 browser. And They did write much Code and feast on great quantities of Papa Del's Pizza, and it was Good, and there was Much Rejoicing. Look for more events like this in the coming weeks, and let me know if you have any other ideas for projects.

If you have any questions about SIGDave, feel free to mail me, Michael Kolb, at sigdave@uiuc.edu Or just show up to one of our meetings, Wednesdays at 8 PM at or near the ACM office. We'll see you there!

http://www.acm.uiuc.edu/sigdave

RedBeard-related injuries incurred to date:

Feb 26: Misha puts 14V across his tongue trying to strip a live wire with his teeth

Feb 28: Joe stabs himself with a ruler

Mar 1: Ben nearly saws off a finger

Mar 3: Misha sets his hair on fire with a hot-air gun

Mar 8: Misha epoxies his fingers together

...And more to come!

RedBeard Rolls:

Navigating DCL's Seven Seas

by Misha Voloshin

For the past month, strange noises might have been heard coming from L510. The whining of hacksaws, the whirring of motors, the screams of CS majors getting their fingers severed, or perhaps merely the bassline of Iron Maiden's "No Prayer For The Dying."

Well, SIGArt has something to show for all our blood, sweat, and tears. RedBeard, the robot designed to chart and navigate the halls of DCL, is rapidly nearing completion. Armed with a rotating ultrasound ranging turret, a set of super-high-torque motors, and a set of batteries that weigh more than the rest of the machine combined, RedBeard is one powerful autonomous unit. Built from the collective mechanical engineering knowledge of SIGArt's team of student computer programmers, RedBeard is miraculously able to move and maneuver with surprising firmness and agility.

The true beauty of RedBeard lies not in its brawn, however, but in its brain. Equipped with a software package that SIGArt has had completed for a couple months now, RedBeard's claim to fame is its ability to maneuver accurately through an environment despite the fact that it does not have absolute control over its motors. Wheels, after all, are mechanical devices; they slip, they idle, they get bumped against rocks and people's feet. The software that runs RedBeard is able to take into account the unpredictable imperfectness of the real world and still have the robot know where it's going and, most importantly, how far it's gone once it's moved.

The process of building this bot has not been an easy one, given the fact that most of the SIGArt team, comprised almost entirely of programmers, has had very little experience even with electronics, much less with such alien concepts as inertia, torque, and structural integrity. Nonetheless, the robot is, at the time of this writing, very near completion. We will proudly present our work at EOH, and the scars and burn-marks will have nice stories behind them which we can tell our grandchildren. If anyone behind SIGArt's desk at EOH happens to have a peg-leg, an eyepatch, or a hook-hand, you'll know that the body part was sacrificed for a good cause!

http://www.acm.uiuc.edu/sigart

SigGraph

by Hans Van Slooten

SigGraph is the Special Interest Group for Computer Graphics. What that means is that we take our creativity and visualize it on the computer. Our group does everything from actual digital art and animations, to computer games and graphical concerts.

With EOH upon us, SIGGraph has been working hard to get its major project done: CTFOTEC (or Capture the Flag on the Engineering Quad). Come and see a virtual Beckman in all of its glory while someone else charges and tackles you. With fully modeled and texture-mapped buildings and 3D character models, we're sure you'll be spending a lot of time near our table.

Also, don't forget to experience some of the amazing graphics from Russel Simmons on various PCs around the SigGraph area. You'll have to see them to believe them.

Finally, Sounds and Visions! Come see it for the first time or if you seen it before, come see it again! In 1320 DCL you'll see excerpts from our biannual concert Sounds and Visions. This is only to prepare you for the concert coming later this Spring. With music produced by members of SigMusic, it promises to be the best show yet.

http://www.acm.uiuc.edu/siggraph

BUG

by Jason Luther and Matt Wronkiewicz

The Be Users Group is a special interest group in the ACM devoted to development on the BeOS platform from Be, Inc. BeOS is a modern multiprocessing, multithreaded, protected memory, high performance, "buzzword enabled," Media OS that runs on PowerPC and Intel machines.

For EOH, the BUG created a real-time sound manipulation application. The application provides a digital audio stream to add-ons (plug-ins) that actually filter the data. The framework allows for anyone to develop an add-on that can do anything it wants with the sound data - SoundMangler has done most of the work already. The application was designed and an alpha version created last semester during an all-day Code-A-Thon. Look for us in the atrium of DCL during EOH.

After EOH, the Be Users Group will begin a series of application development workshops targeted at developers unfamiliar with the BeOS. We invite you to attend this series to learn to write useful applications and unleash a new level of performance from your machine.

We meet Wednesday of each week in the ACM office (and move to 1102) at 9:00 P.M.

`http://www.acm.uiuc.edu/bug/`

by Mike Magin

The UIUC Linux Users Group (LUG) meets every Tuesday night at 9 PM in 1102 DCL. Users of any experience level are welcome to come as we discuss various topics of interests to Linux users.

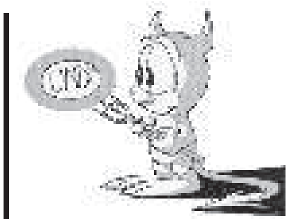
For the uninitiated, Linux is a free work-alike clone of the UNIX operating system. (It borrows features from both SYSV and BSD variants.) Linus Torvalds originally began its development in 1991 on Intel x86 based PCs, but it has been ported to the DEC Alpha, the Sun SPARC, the Motorola/IBM PowerPC, the Motorola 68000 series and MIPS architectures.

At meetings, we always discuss new or changing things that relate to Linux, and then we have a question and answer session that consumes the majority of the meeting time. We have found that this works especially well for solving peoples' difficulties with Linux, since almost always, for a given question, someone in the group is able to answer it.

Our primary current project is to create our own Linux distribution. We are doing this for two reasons. The first is to fix many of the flaws we see in other major distributions such as Slackware and RedHat. The second is to create a distribution which is more suited to local users by providing packages such as ph, Kerberos, bootp scripts, and PPP scripts. The first public release of the
distribution will occur in just a few days. For more information see:
`http://www.acm.uiuc.edu/lug/distrib/`

In general, for more information email `lug@uiuc.edu`.

`http://www.acm.uiuc.edu/lug/`



by Tony Sintes

This is an important semester for SigUNIX.

SigUNIX will be hosting 3 Tcl/Tk workshops on (tentatively):
April 3 Tcl/Tk: The Basics
April 10 Tcl/Tk: The Extensions
April 17 Tcl/Tk: Writing new Tk widgets in C

We also have some cool projects lined up and are setting the ground-work for a EOH '99 project that will showcase UNIX and its strengths.

Everyone and anyone regardless of experience is welcome to attend. Just be sure to bring an interest in UNIX and doing cool things with UNIX. SigUNIX meets on Thursday nights at 8 P.M. in the ACM office. Weather permitting, we move to a computer lab or lounge.

`http://www.acm.uiuc.edu/sigunix/`

by Ray Kaplan

With the advent of VRML (Virtual Reality Modeling Language) 1.0 in 1995, a text based file format was created that describes a three dimensional virtual world. Like HTML, this file was text to allow for quick transmission over the internet, where a client would view it with a compliant browser. With subsequent revisions in 1996 and 1997, the VRML97 international standard was created which now allows for fully interactive virtual worlds as well as more multimedia capabilities like sound. An External Authoring Interface using Java applets was developed to allow for VRML generation on the fly as well as the creation of more dynamic worlds. VRML browsers are now included in Internet Explorer and Netscape Navigator as an ActiveX control and a plug-in respectively.

The current VRML standard, however, does not have any networking defined. There are a few browsers that have networking capabilities, but none of them have been standardized. This leads into SigVR's current project, VRMLMazeHunt. VRMLMazeHunt is a game that is a continuation of SigVR's EOH project from last year which was a single player maze game. This year's project is a multiple player game where any two (or more) people on the internet that have a VRML browser that supports the External Authoring Interface. VRMLMazeHunt uses Java applets which use the External Authoring Interface and a CGI script for the networking allowing any webserver to be used. Due to the limited amount of information transmitted during game play, VRMLMazeHunt is not affected by the internet connection speed.

VRMLMazeHunt was developed in part with VRCreator 2.0, a VRML generation program donated to SigVR by Platinum Technology. More information about VRMLMazeHunt can be found at its web page on SigVR's website at,
`http://www.acm.uiuc.edu/sigvr/projects/eoh98/`

`http://www.acm.uiuc.edu/sigvr/`

by Mark Ashton

VIDEX is a Visually Integrated Development Environment for X/Windows. Presented last year at EOH as the early stages of an extended development effort, VIDEX met with much interest. The idea behind VIDEX is that Windows and Mac programmers shouldn't be the only ones to get a nice environment to develop applications in. We provide that nice environment, and so much more.

While traditional IDEs (Integrated Development Environments) are tied to vendor-specific compilers, editors, and debuggers, VIDEX is designed to be able to use whatever tools the developer is most comfortable with. It is also designed to be very extensible, with a flexible event system that allows easy addition of new modules.

VIDEX presents a fairly standard interface to the user. There is a project manager with a file viewer, there are editor windows, debugger windows, and all the usual rigamarole. This interface is familiar to most developers who have worked outside of the UNIX world. It provides an efficient and centralized way to use all of your favorite development tools.

The internals are what make VIDEX truly interesting. It was originally written in Python, a rapid-development interpreted language. This allowed us to quickly prototype and test code for the finished product. We are currently moving it into C++, which will result in more efficient and faster-running executables. The event system is designed to be extended. By writing some simple wrappers, the developer can make any tool a part of VIDEX. Therein lies its greatest potential.

Stop by the SIGSoft booth at EOH and see why we are so excited about this project. VIDEX isn't just another EOH code-fest, it represents a fundamentally different way of developing software. Stop by once you've played all the games and see something you could use to get real work done. ;)
`http://www.acm.uiuc.edu/sigsoft/`

Everything You Ever Wanted to Know About ACM@UIUC, But Were Afraid to Ask.

by Paul Watts

What is ACM@UIUC?

ACM@UIUC is a student chapter of the Association for Computing Machinery, an international professional society founded in 1947, a year after the unveiling of ENIAC. ACM@UIUC has been active since 1965, and we have records back when former CS department head Duncan Lawrie was an undergrad.

At its heart, ACM@UIUC is a place for people to get together and do cool stuff. This means not only software or hardware projects, but all the different social and professional events ACM has each year.

How do I join ACM?

All you need to join ACM is $12 and a pulse. Just show up to our office (1225 DCL), fill out a membership form, and you're in. Once you're in, you're entitled to all the benefits of being an ACM member, such as joining a SIG and getting your resume in our Resume Book.

What is a SIG?

SIG stands for "Special Interest Group." A subsection of ACM, a SIG specializes in one aspect of computers, as in Artificial Intelligence (SigArt), Computer Architecture (SigArch) or Biology (SigBio). There are also a number of user groups, such as the Linux User's Group (LUG) and the Be User's Group (BUG). ACM@UIUC is the first ACM student chapter to have SIGs as a part of its general structure.

How do I join a SIG?

Show up to one of their meetings! The meeting times and places are available online (`http://www.acm.uiuc.edu/sigs/`), and you're welcome to show up there for a few meetings to see how you like it. If you decide it's great and you want to be a part of it, we ask that you join ACM.

What are the General Meetings?

ACM General Meetings are monthly talks that are open to the public. They generally occur on the first Thursday of every month, at 6:00 P.M. in 1320 DCL. They feature a variety of technical speakers: in the past we've had such names as Michael Abrash of Quake fame and Richard Stallman of the Free Software Foundation. This month we have Mike Muhney of CelebritySoft, talking about entrepreneurship in computing.

Do you need to be a member to go to General Meetings?

No! The General Meetings and ALL our social events (such as our movie nights and SigGraph/SigMusic's Sounds & Visions) are open to the general public.

What is the Midwestern ACM Chapter Conference?

The Conference is an annual event which gathers together ACM members from around the Midwest, from U of I to Kansas to Minnesota. In the three days of the conference we have technical talks on a variety of subjects, a programming contest, and a job fair.

What is ACM@UIUC Chair Paul Swanson's Real Identity?

His real identity is Jeremy Miller, star of TV's *Growing Pains.*

Who is Jeff Spicoli?

Jeff Spicoli wears Vans, dreams about surfing, and smokes a lot of weed. Spokesperson and role model to the early MTV generation, Jeff Spicoli preaches his gospel from his high school, Ridgemont High.

For more information:
ACM: `http://www.acm.uiuc.edu/history/`
SIGs: `http://www.acm.uiuc.edu/sigs/`
Events: `http://www.acm.uiuc.edu/events/`
Conference: `http://www.acm.uiuc.edu/conference/`
Jeff Spicoli: `http://alanb.com/spicoli/`

VRML98 SYMPOSIUM

by Ray Kaplan

Last month, the VRML (Virtual Reality Modeling Language) Consortium gathered in California for their yearly meeting. The VRML Consortium is an organization dedicated to the purpose of promoting three dimensional content on the web, specifically VRML. The Consortium is made of several companies including Microsoft, Silicon Graphics, Evans and Sutherland, and Platinum Technology.

The most important news from the symposium was that Sun Microsystems joined the Consortium. Sun has been working on the Java 3D API for the purpose of creating three dimensional graphics in Java. With the addition of Sun, it is hoped that there will be more interoperability between VRML and Java 3D.

There were also several sessions on working to define new elements to the current VRML standard. This included the meetings of several of the Consortium's Working Groups like the MPEG4 group which is working on integrating the MPEG4 standard into VRML, and the Distributed Interactive Simulation to allow for more networking support in VRML.

Many of the attendants also demonstrated new applications that they developed with VRML. This included everything from business applications such as Platinum's Forrest and Trees to three dimensional interactive games. On the application end, there were several panel discussions on subjects such as data visualization with VRML and circumventing the 28.8 modem barrier.

The VRML Consortium has its web page at `http://www.vrml.org` The Consortium meets once year to bring together people from all aspects of VRML development, and continue to develop the future of VRML and three dimensional content on the web.

ATTENTION ATTENTION ATTENTION ATTENTION ATTENTION

ACM GENERAL MEETING THURSDAY, April 2nd at 6:00 in Room 1320 DCL

ATTENTION ATTENTION ATTENTION ATTENTION ATTENTION

On Thursday, April 2nd, 1998, ACM will be rocked to its very core, and you'll be responsible. Because on April 2nd, ACM@UIUC will be holding its annual student officer elections. This is **your** big chance to make a difference in **your** society. Whether you want to run for an office, or you simply want to exercise your right of free franchise, you *need* to be at this meeting.

If you are interested in running for a Top Four position, check out this issue's *From the Chair* on page two. If you are a member of ACM@UIUC (you've paid your dues), you are heartily encouraged to run.

If you can attend only one General Meeting all year, make it this one. It is without exaggeration the most important Meeting for ACM that can be. We'll see you there.

"Hey, Where Can I Get an MP3 of That???"

by Paul Swanson

The Planet radio station (WPGU 107.1) is a top 40 radio station, meaning that at any given time, only about 40 songs are on its playlist (I see 35 at their homepage right now). From talking to their program manager, I discovered that they rotate in about 3 new songs a week. That works out to around 200 songs a year.

200 songs on 200 singles for a year's worth of radio play at the Planet. With home CD burners, that could be condensed to 10 CDs (20 songs a CD is 3:45 per song).

I can fit all 200 songs on a single CD. I am not making this up.

How is this possible? With a compression algorithm designed specifically for audio files: MPEG Audio. The Motion Pictures Experts Group formed in 1990 to develop a standard for video and audio compression at restricted bit-rates. Four years in the making, the ISO standard finally arrived, but it would be another year or two before personal
computers would be powerful enough to decompress audio on the fly.

MPEG audio layer 3 files, or MP3s (so named because the file extension is almost always .mp3), are unlike most other compressed files in two ways. First off, the compression algorithm does not take advantage of any self-similarity in the original sample, which is the basis for most traditional compression algorithms (running gzip on the resulting file typically results in an additional 10% compression). This results in a stateless output file which is divided up into frames of 8 or 24 ms
chunks, and a decompressor can start playing at any of these frames. But most importantly, an MP3 cannot be decompressed exactly into the original signal, which makes MPEG audio a "lossy" compression format. However, most people tout MPEG audio as "CD quality"; how can this be if the audio sample is not the original digital data at 44.1 kHz 16-bit stereo?

Psychoacoustic research has shown that the human ear does not respond to certain elements of an audio sample, particularly when a loud signal "masks" a softer signal, either in another frequency band or later in time. The encoder takes advantage of this knowledge and ignores data that will be masked at any particular point in time, thereby only keeping the most important sound elements. Experienced listeners with expensive sound equipment generally rate 128 kbps (the bit-rate)
MPEG layer 3 as a 4 out of 5, which means that there is a slightly noticeable difference (a 5 sounds exactly like the digital sample), but it is not annoying or distracting. For the average home stereo, this difference is undetectable.

So what are the implications for you, the music aficionado? At present, if you have a Pentium-class machine, you can burn a CD of your favorite 200 songs, and play any one of them at will. Soon there will be stand-alone MP3 players that you can install in your car, for 10 hours of non-repeated music with no advertisements. I believe that's better than any radio station in existence.

The only problem with MP3s, of course, is that they bring about a myriad of copyright infringement policies. It would be extremely impractical to trade a complete CD audio track with someone else, and music publishers don't care about trading reduced quality audio. However, MP3s are a great archival format, since they are "near CD quality" and typically have a compression ratio of 10:1 (1 megabyte of data per minute of stereo sound).

Of course, a very few music publishers are already taking advantage
of this as a method of distribution, by making available an MP3 of an artist's popular work to lure you into buying the entire CD. Hopefully the MPEG audio format will quickly die out as the next intellectual property legal debate, and just as quickly become a great sampling format.

So You Wanna Make a VR Movie...

by Vikram Adukia

For our EOH project this year, we have chosen to do QuicktimeVR. With QuicktimeVR, you can explore an interactive, 360 degree environment right from your computer. Inside each scene, you can zoom in on various objects, or click on predefined "hotspots" to go another location.

However, how do you make such a movie? Well, it's pretty easy once you understand how.

Here's what you're going to need:
1) a digital camera
2) a tripod
3) some "incrementor" to move the camera in a set number of degrees
4) some sort of VR authoring program. For our project, we used QuicktimeVR Authoring Studio

First off, you are going to need to decide on a scene to do a picture of. Through our experimentation, we found that outdoor scenes work the best. In good daylight, you have all the light that you would need, and the scenery integrates well into the final movie.

Once you decide on a location, you're going to need to setup your picture taking equipment. You attach the camera to the incrementor, and the incrementor to the tripod. Now, all you do is take a picture, and then move the camera using the incrementor. Repeat this picture taking process until you make it a full 360 degrees around.

Go back to where you have your computer setup and download the pictures to the computer. Use your VR authoring program to "stitch" the pictures together. By aligning the images, you can put together all the pictures together to make a single large picture that spans all the pictures.

Finally, the authoring program will convert the image into a VR scene. What it essentially does is take the 360 degree flat picture just created, and wrap it around.

Tada! You have a VR scene. If you want to get into more fancy aspects, you can also create hotspots to allow the viewer to go to other areas and go more in depth into your virtual world.



Three photos used to construct part of a QTVR movie

A 360 degree panorama of the Engineering Quad, used in a QTVR movie

Detecting Network Errors Using Java

by Jason Govig

The heart of many interactive web sites lies in chat. A chat room can serve as a center of communication, allowing many people to interact while exploring and playing with whatever that site has to offer. Naturally, SigNet decided to create a chat server to be the heart of Voodoo City. It has developed through many stages and two languages into the server that is being used today.

The original version of the chat server that was written in C++ was very simple, allowing only two clients to connect and talk alternatively. Using file descriptor sets, the clients were able to talk simultaneously and clients could be added and removed at will. We soon discovered problems detecting clients that would drop unexpectedly. This resulted in rewriting the chat server in Java.

The first steps in detecting client errors resulted in classifying dropped clients into two categories: dead and stale. We described dead clients as users who would quit the application used to connect to the server (telnet or an applet) without sending a 'QUIT' command to the server first. Stale clients occurred when the line between the client and the server would unexpectedly drop, and the server would think that the client was still connected. This condition was harder to fix.

Dead clients were detected in the input stream. We used a `java.io.DataInputStream` to read messages from the clients. If, when reading a line with `DataInputStream.readLine`, there was a `NULL` detected, either by a `nullPointerException` or a test, we would know that the client disconnected from the server. After detecting the `NULL`, we could then tell our MasterControl to remove the client and cleanup all streams and sockets.

To detect stale clients, we had to test the output stream. To output messages to the clients, we used `java.io.PrintStream`. After sending a message using `PrintStream.println`, an error flag is set if there are any errors writing to the stream. After checking this using `PrintStream.checkError`, we could then call the routines to remove and cleanup the client. Due to the blocking nature of the sockets for reading lines, there were problems cleaning up the client. We then switched to Java 1.1 and used `java.net.Socket.setSoTimeout` to let the socket timeout after a few seconds. Then after the timeout, it would check the state of the client and complete the cleanup if the client was set to be removed.

Switching to Java was a great thing: multi-threading the control of the clients is much easier, networking code is simplified, and most of the error detection is handled very easily. Another bonus was the availability of documentation for Java network programming as opposed to C++. All of this made Java a prime language to simplify and help detect errors in our chat server.

TCL/Tk: A Scripting Language Can Do THAT?!?

by Tony Sintes

Imagine a language that you can learn in an afternoon and that allows you to build complex Graphical User Interfaces (GUIs) with little effort and without knowing C, MOTIF or MFC. As a bonus, imagine that it also allows for easy extension, integration with existing applications, rapid development, and cross platform support.

Imagine that none of it needs to be compiled and that changes can be seen on the fly.

Imagine that it exists now, it is mature, and it is FREE.

Is all this a dream? Hardly! Such a beast actually exists: Tcl/Tk, the "Tool Command Language."

Don't let its simple nature, or name (Tcl is pronounced "tickle"), fool you.

continued on page 13...

...continued from page 12

By providing a simple but powerful scripting language, Tcl/Tk has found use in everything from simple time saving scripts, to telecommunication applications that help service millions of telephone calls per hour. Since a Tcl/Tk application looks and responds just like a natively compiled program, many people have used Tcl/Tk applications without knowing it. For example, anyone who has used the administrative GUIs provided with RedHat Linux has seen and used Tcl/Tk.

All of this sounds pretty good, but what exactly makes Tcl/Tk so great?

Tcl/Tk offers many benefits, but the 2 main strengths are its:
- ease of use
- extendibility

Let's start with ease of use.

Tcl is a scripting language. The Tcl interpreter provides the core commands and rules needed by the Tcl language. This interpreter executes all Tcl scripts. The Tk in Tcl/Tk is actually a package built on top of Tcl that implements a set of new Tcl GUI commands. When a Tcl interpreter includes the Tk Tcl commands, it is called a "wish." With a wish, GUI scripting is possible.

Like other programming languages, Tcl has all the constructs that you would expect. There are control structures, string manipulation commands, file access commands, arrays, and lists. But unlike other languages, there are no rigid typing rules. Instead, only the interpreter and its substitution rules are fixed. All commands appear as strings to the Tcl interpreter. The interpreter has no idea about the commands or their arguments. The only thing that the interpreter knows is that the first word of the command string is the command; the rest of the string is the command's arguments. To execute the command, the interpreter simply passes the other words in the string to the first word, the command.

So what?

Well, this means that nothing about the Tcl language is defined before runtime. There is no need to allocate and cast a variable before it is used. Instead, when a variable is needed, it is just used directly. Furthermore, this allows old commands and new commands to be defined and redefined at anytime.

For example, the following code sequence:

```
int k=0;
k = 1 + 2;
```

can be replaced by:

```
set k [expr 1 + 2]
```

So how does the interpreter see this command sequence?

The line is parsed. The interpreter sees "expr 1 + 2." To evaluate "expr 1 + 2" the substring "1 + 2" is passed to "expr." "1 + 2" are the arguments to "expr." The result is treated like any other Tcl script. So now, the string looks like this to the interpreter:

```
set k result
```

Now, "k result" is passed to set. As a result, the argument k is set to the argument result.

What about Tk? Since Tk is built up of a set of new Tcl command, Tk commands work just like any other core Tcl command. The following Tcl command creates and displays a button that says "Hello World." The button is gray and the text is black. When the button is pressed the window closes.

```
Button .b -text "Hello, world!" -
command exit -background gray -
foreground black Pack .b
```

We see two commands: "Button" and "Pack." "Button" creates a new button widget. It accepts a variety of arguments. Here we specify the text, window path, colors, and command to execute when it is pressed. When the command completes, the window path to the new button is returned (it is the same as the one we specified ".b"). However, the "Button" command doesn't actually display the button. Instead, the window path must be passed to a geometry manager. Here, we pass it as an argument to the "Pack" geometry manager.

continued on page 14...

...continued from page 13

Once the path has been passed to a manager, the button is displayed.

While arguments are not cast, the commands still expect certain kinds of arguments. If a command is passed a argument that doesn't make sense, an exception will be thrown. So we still need to pass the right types of arguments to a command.

While it may seem confusing at first, this approach allows for an intuitive syntax that is flexible as well as close to natural language. It also allows the language to be easily extended with new Tcl commands and Tk widgets.

Because everything is a string, it is possible to pass Tcl commands between Tcl programs. After the string is passed, it will be interpreted with the rest of the script. This is a powerful feature that allows inter-process communication between applications or clients and servers. Because it is just a string, the commands sent can be any valid Tcl command. As a result, special cases do not normally need to be taken into consideration.

Because of an intuitive syntax, lack of type casting, and GUI scripting features, Tcl/Tk lends itself to rapid application development. Because the language is extensible, these applications can be quite complex.

What about extensibility?

Tcl's ability to be extended is its most powerful feature. Tk is nothing more than an extension to the core Tcl commands. Users can easily write new Tcl commands in C. These new procedures can be compiled directly into the interpreter. Or, if you'd rather keep the interpreter small, the new commands can be compiled into a shared library and loaded into the interpreter at runtime. Since widgets are just new Tcl commands these can be written into shared libraries as well.

One of the nice things about Tcl/Tk is the fact that you don't have to compile source code. So far, I've made it seem like you have to write new Tcl commands in C. Well, you don't. There are two extensions to Tcl/Tk that allow you to write new commands and widgets in Tcl. These extensions are [incr Tcl] and [incr Tk]. These extensions allow you to create new commands or widgets by gluing together preexisting Tcl/Tk commands in an object oriented framework. These new commands and widgets can be used just like any other command.

Complete applications can be written using [incr Tcl] and [incr Tk]. Sometimes it can be hard to keep a large application written in straight Tcl/Tk maintainable. But if written with these extensions, the application will be much more maintainable. In fact, I wouldn't write a large application without the extensions.

Because of its component nature, Tcl/Tk can be used to integrate existing applications. It can also be used to provide a scripting level interface into a new or preexisting application or system. Simply add the Tcl commands needed to interface with the system and you're set. Once the commands are added you can write scripts to interface with the system. Using [incr Tcl], [incr Tk], and the new commands, you could even create a
new widget that interfaces with your system. Now, you can use this widget to interact with your system in any script.

That all sounds great, but how about an example?

Okay! Say you have a contact database somewhere on your network. You want a GUI screen that accepts 5 pieces of information (name, address, phone #, age, sex). When you have the contact info typed in you just want to hit a button and have the information sent to the database. For kicks, let's say your legacy database talks to the network with CORBA.

So we need:
- new Tcl commands that will send info to the database using CORBA
- a widget that provides entry fields for the data and a submission button

Here's what we do:

First we write 2 new Tcl commands
- one command that binds to our CORBA object
- the second command sends the data to the database object

Next we write a "Mega-Widget" with [incr Tk] for the entry box
- the widget is built up of 5 entry fields, 1 button, and

continued on page 16...

Programmable Logic - Taking Hardware Design to the Next Level

by Brian Burgner

Programmable Logic Acronyms

CPLD - Complex Programmable Logic Device

EEPROM - Electrically Erasable Programmable Read-Only Memory

FPGA - Field Programmable Gate Array

I/O - Input/Output

PAL - Programmable Array of Logic

PLA - Programmable Logic Array

PLD - Programmable Logic Device

PROM - Programmable Read-Only Memory

ROM - Read-Only Memory

SRAM - Static Random Access Memory

Programmable logic is an old concept. Perhaps the earliest design was simply a multiplexor with fixed inputs used to implement a simple Boolean function. Logically following from this was storing function values in read-only memory. ROM. The function call would look at an address in ROM and a result would be produced. Both of these methods make it easy to implement a truth table, provided the truth table doesn't change too much. To accommodate this, EEPROM's could be used instead of normal ROM's so that the function can be updated.

Another step up in complexity is the programmable logic device (PLD). This is an array of gates connected by fuses. There are three popular forms of PLD's. These are programmable read-only memory (PROM), programmable array logic (PAL), and programmable logic array (PLA). All three of these designs consist of an AND array and an OR array; the differences lie in where the fuses are placed.

In a PROM, there is a series of fuses between a fixed AND array, which is basically just a decoder, and a fused programmable OR array. This setup implements the sums of minterms in a Boolean function. It is similar to using a normal ROM, but is more efficient in terms of gates per function.

Next is the PAL. A PAL has a fused programmable AND array with the outputs of the AND array going into a fixed OR array. Having only the AND gates programmable makes a PAL very easy to program, but takes away some of its flexibility. Occasionally, the outputs of the OR array will feed back into the AND array so that larger functions can be implemented. Like the PROM, PAL's implement Boolean functions as sums of minterms.

The most flexible PLD used to be the PLA. The PLA is very flexible because it has a fused programmable AND array as well as a fused programmable OR array. PLA's are more difficult to program since the output doesn't always correspond to minterms, but the larger number of fuses allows designs to be designed with a smaller PLD when a PLA is used.

All of these forms of programmable logic are still in use today, but due to their permanent nature, they are not optimal for prototyping. As a result, reprogrammable logic was developed. Reprogrammable logic is basically programmable logic that can be easily reprogrammed multiple times. Two varieties of reprogrammable logic exist today; complex programmable logic devices (CPLD's) and field programmable gate arrays (FPGA's). Each

continued on page 16...

..."TCL/TK" continued from page 14

an etched border
- when the button is pressed, it calls our new Tcl command that sends data to the database and passes it the new info.

Now that we have the new "Mega-Widget," we only need to wrap everything together with a Tcl script. The script simply binds to the CORBA object and creates an instance of the new "Mega-Widget."

That's it. We now have a GUI that interfaces to our database. As an added bonus we also have two new Tcl commands and a new widget that can be used in any future Tcl program that interfaces with the database.

Of course, instead of writing a script, we could have wrapped everything together into a reusable object. This object would do everything that the script would do, but now we could embed this self contained object into any other Tcl program. We could embed multiple objects or mix it together with other objects, such as one that allows us to retrieve information from the database.

In other examples, we may wish to provide GUI interfaces to a variety of command line utilities. By using Tcl/Tk as glue, we can reuse the existing applications by just gluing a new interface onto existing applications.

Because it is extensible, Tcl also makes a good candidate for the basis of a new language or parser. Simply add the commands needed by your language. For example, this could be used as a basis for a web browser or configuration language. All configurations for an application could be written to a file. When the application is restarted an embedded Tcl interpreter could parse the configuration file and set up the application, or applications, or network, etc. to the predefined values.

For more information about Tcl/Tk see:

```
http://sunscript.sun.com/
http://www.scriptics.com/
comp.lang.tcl
```

or check out the following books:

Tcl and the Tk Toolkit, by John K. Ousterhout, and *Tcl/Tk Tools*, by Mark Harrison.

Of course you could always come to a SigUNIX meeting! They are held every Thursday night at 8 P.M. Just come to the ACM office.

..."Programmable Logic" from page 15

has its own advantages and its own applications.

CPLD's are generally combinations of PAL's and PLA's that have been wired together with reprogrammable fuses similar to those found in electrically erasable programmable read-only memories (EEPROM's). These fuses are set with approximately 15 volts DC so that normal 5 volt input/output (I/O) signals can't reset them. As a result, your function remains implemented until it is reprogrammed with a device that applies 15 volts to the fuses. CPLD's tend to be fairly small (< 1000 gates), and the number of I/O pins is very close to the number of minterms in the functions they imitate. This makes CPLD's good companions to microcontrollers since they are easily reprogrammable, but keep their current function even when they are powered down. They are also useful for microcontroller applications since they are usually small chips with 8-50 pins.

FPGA's tend to be much more complex than CPLD's. Many FPGA's implement functions that require over 100,000 logic gates. As a result, FPGA's have a high ratio of functions per I/O pins. This makes them very useful for designing application specific integrated circuits (ASIC's). A common use for FPGA's today is prototyping ASIC's since a new chip doesn't have to be fabricated for each design. If a design doesn't work, a new one can simply be loaded into the FPGA. Most FPGA's use static random access memory (SRAM) to store their current program so they lose their program when they are shut down. To counter this, many FPGA's are wired to EEPROM's so that the design can be downloaded to the EEPROM and when the circuit is powered up, the design is reloaded into the FPGA.

The programmable logic market keeps expanding to include new fields such as memory management and digital signal processing, two applications that FPGA's have performed very well in. Companies like Xilinx, Altera and Cypress Semiconductor continue to improve the performance of the chips in terms of clock speed as well as the number of gates that can be

continued on page 18...

SIGOps and Fishnet: The Mother of All CS EOH Projects

by Douglas Armstrong

During the spring semester of '97 SigOps first gave its workshop on how to, "Roll Your Own OS," by Paul Swanson. The goals were to provide an environment in which it would be easy to start up quickly with a simple kernel and then gradually add pieces to it; as opposed to writing code for three months before even seeing a message on the screen. The lessons provide the general theory and what to do, but keep it general and don't get too specific as to bias the learner. This has led to an incredible amount of ingenuity and creativity in the operating systems written. To facilitate this development environment `boot.com` was pioneered. This was a simple bootloader that would boot the machine from DOS into protected mode as well as load files passed to it into memory, then transfer execution. The idea was for an operating system author to be able to compile and test a "null" kernel in about 5 minutes. This allows just that. When the complete development environment was finished it had an NFS connection, which allowed an extremely fast edit-compile-download-test-reboot cycle. Restructuring the workshop into its current form of "How to Write an Operating System," it can be seen at `http://www.acm.uiuc.edu/sigops/roll_your_own/`. We then gave the workshop again at the beginning of this school year to a larger audience. From this experience quite a few people have discovered that it is possible to write their own OS. Each person has worked independently on their own OS with a great variety in design philosophy. With five operating systems already multitasking, it's been a success.

Striving to get rid of DOS entirely, however, we created a new boot system. In this new system the bootloader creates a "filesystem" of the files into a boot image. The options for booting this image are even better. Brian Swetland developed the Netboot system. This loads a server onto the test bed that waits for a client to remotely send a boot image over the network. Later the secrets of the boot sector on a floppy disk were cracked. This allowed us to create bootable floppies that can either have an OS boot image on them or the netboot system. In addition, you can still load the image or netboot from DOS using `boot.com v2.0`. In all, you can:

* load from the net using boot.com & netboot
* load from the net using a netboot floppy and netboot
* load from floppy with a bootmaker generated disk image
* load from DOS w/ boot.com v2.0

Check out the preliminary specifications at
`http://www.acm.uiuc.edu/sigops/rsrc/boot.html`

Searching for a way to present our work for EOH, we came up with the FishNet idea. Every machine in the display can each run a different independent homebrew OS. For this we created the specification for the DFP, or Distributed Fish Protocol. Each client is a fish tank which is part of a larger virtual fish tank. The clients can pass fish from one machine to the next as they swim off screen. This shows how all these different systems can work together. As a further part of the demonstration Paul Watts will have a Windows based client that they can interact with. This helps demonstrate how our systems can also work with other, more "traditional," operating systems.

For this project I created an NE2000 network card driver core that can be ported to any OS regardless of the driver methodology. Some preliminary work for the VGA card has also been done that looks very promising. Take a look at the OS descriptions elsewhere in this newsletter and check out the FishNet page at
`http://www.acm.uiuc.edu/eoh98/`

Here are the OS descriptions. I have put them in order of inception.

G.O.O.S.E.

Game Object-oriented Operating System Environment
Author- Paul Swanson

A single address space operating system designed to maximize data flow and performance, and minimize memory footprint. GOOSE will only load the appropriate drivers for the host computer and for the target game, yet is flexible enough to provide real-time multithreading, efficient paging, and debugging facilities to the game developer.

continued on page 18...

...continued from page 17

OpenBLT

Author- Brian Swetland

OpenBLT is a microkernel operating system for PC's based on the Intel 80386 or later CPUs. Some aspects of OpenBLT were inspired by Andy Valencia's VSTa OS, Mach, assorted UNIX variants, and the author's deranged imagination. The acronym BLT expands to "Brian's Lightweight Tasker" or "Bacon, Lettuce, and Threads". The word Open was tacked on for no particular reason.

The OpenBLT kernel is responsible for managing system memory, providing IPC facilities, and creation, destruction, and scheduling threads of control.

Below, we'll examine Tasks, Aspaces, and Ports (the basic objects managed by the OpenBLT kernel) in terms of their use in and appearance to userland programs.

Tasks

Tasks in OpenBLT are single threads of control. Each task is associated with an address space, within which it runs. Tasks may terminate themselves using the

```
void os_terminate(int status);
```

system call. A task may start another task (sharing the same address space) using the

```
int os_thread(void *addr);
```

system call. The new task begins execution at the specified address and a 4K stack is created for it. *[os_thread() will probably take a stack pointer address in the next revision]*]

Address Spaces

Address spaces are currently not readily manipulated from userland. In the near future, shared memory system calls should allow the creation of new address spaces and sharing memory between address spaces. A thread may request that its address space be expanded using the os_brk() system call:

```
int os_brk(int addr);
```

Ports

IPC (interprocess communication) in OpenBLT occurs when one task sends a message to another task via a port. Ports are 'owned' by one task (the task that created them). Messages are send from one port to another port (thus a port must exist for the sending task to send from). A port may be restricted so that it can only receive messages from one specified other port *[future versions will likely allow a list of allowed senders .]*

All messages sent or received via an OpenBLT port are described with a message header. The header indicates the source port, destination port, size of message, pointer to message data (or receive data buffer), and flags (to allow for future expansion -- non-blocking IO, scatter/gather IO, etc.)

continued on page 19...

..."Programmable Logic" from page 17

implemented on one chip. In the future, FPGA's could very easily be an integral part of personal computers, with each chip optimizing itself to accomplish the task at hand.

Information for this article came from *Computer Engineering: Hardware Design*, 1st edition by M. Morris Mano. This used to be the book for ECE 290. A similar text is used for both ECE 290 and CS 231. ECE 311 is actually a class on hardware design and uses Xilinx FPGA's for it's machine problems. More information on chip specifications can be found at Xilinx's webpage, www.xilinx.com, or Altera's webpage, www.altera.com, or Cypress Semiconductor's webpage, www.cypress.com. If you're interested in actually using FPGA's, come visit SIGArch in L510 DCL or email `sigarch@uiuc.edu` for more information.

...continued from page 18

```
typedef struct {
    int flags;
    int src;
    int dst;
    int size;
    void *data;
} msg_hdr_t;
```

`int` **`port_create`**`(uint32 restrict);`
Create a new port owned by the running thread. Only allow messages from port 'restrict' to be received. If restrict is 0, messages from any source will be received. Will return ERR_MEMORY if the system lacks the resources to create the port, otherwise the port number will be returned.

`int` **`port_destroy`**`(uint32 port);`
Destroy an existing port. Returns ERR_RESOURCE if port is not a valid port or if there are other ports slaved to this port. Returns ERR_PERMISSION if the running thread is not the owner of this port. Returns ERR_NONE upon success.

`int` **`port_send`**`(msg_hdr_t *mh);`
Send the message in mh->data, of length mh->size from port mh->src to port mh->dst. Returns mh->size upon success, ERR_MEMORY if there is a bounds error, ERR_RESOURCE if either the source or destination ports do not exist, ERR_PERMISSION if the source is not owned by the running thread or the destination is not sendable from the source port.

`int` **`port_recv`**`(msg_hdr_t *mh);`
Receive a message (in buffer mh->data, max length mh->size) from port mh->dst. Upon success, mh->src will be set to the sender, mh->dst will be set to the destination if it was a slaved port, and the number of received bytes will be returned. If the data or header are out of bounds, ERR_MEMORY is returned. If the destination port does not exist ERR_RESOURCE is returned. If the running thread does not own the destination port, ERR_PERMISSION is returned. This call will block if no messages are available, unless the port is set to NOWAIT, in which case ERR_WOULDBLOCK is returned.

`int` **`port_option`**`(uint32 port, int opt, int arg);`
Modify port options. The two functions below are wrappers around port_option()

`int` **`port_slave`**`(uint32 master, uint32 slave);`
Cause the master port to receive all messages sent to the slave port. If master is 0, the slave is released from bondage. Returns ERR_NONE upon success, ERR_PERMISSION if the master and slave are not both owned by the running thread, or ERR_RESOURCE if the master or slave are not valid ports.

`int` **`port_set_restrict`**`(uint32 port, uint32 restrict);`
Change the restriction on a port. Returns ERR_NONE on success, ERR_RESOURCE if the port does not exits, or ERR_PERMISSION if the running thread does not own the port. A restriction of 0 allows any port to send to this port.

Finding other tasks to talk to

Since port numbers are generated dynamically by the kernel and tend to be different depending on the order tasks are started, etc., there needs to be a mechanism for locating device drivers or just other tasks to talk to. The kernel automatically creates port 1 (the uberport) and gives ownership of this port to the first task created. This task is the namer, a service that allows drivers to register ports under specific names and allows tasks that wish to find these drivers to look up the port that is associated with a specific name.

The utility functions listed below are part of the OpenBLT libc and provide convenient ways to access the namer.

```
int namer_newhandle(void);
int namer_delhandle(int nh);
int namer_register(int nh, int port, char *name);
int namer_find(int nh, char *name);
```

Connections between Connectionless ports

Ports in OpenBLT are connectionless. Each port_send() specifies a source and destination port. Tasks may send from any port they own to any port they are allowed to write to. Ports may be writable to be any other port or restricted to just one other port. *[multiple port restrict lists will be in a later version]*

The general procedure for providing a service in OpenBLT is

continued on page 20...

...continued from page 19

 * Create an unrestricted port
 * Register this port with the namer under a descriptive name (e.g. ne2000, console, etc.)

When one task (a client) wishes to connect to a service (the server), the following occurs:
 * The client looks up the servers unrestricted port using the namer
 * The client creates a port that is restricted to the server's unrestricted port
 * The client sends a message to the servers unrestricted port requesting a connection
 * If the server accepts the connection, it will send a response back to the client's port notifying it that the connection is accepted and possibly providing a different port than the unrestricted port that the client should send future messages too.

[a toolkit in libc will streamline this process]

Handling Devices

The security model in OpenBLT is still being designed. Most likely there will be a task that is a "security manager" which will be able to give permissions (for accession IO devices, etc.) to tasks that need them. The two syscalls described below are sufficient for simple device drivers in the initial (overly trusting) version of OpenBLT:

```
void os_handle_irq(int irq);
```

Ask the kernel to make this task eligible to process the specified IRQ. The kernel will also allow the task IO access.

```
void os_sleep_irq(void);
```

Suspend this task until the IRQ it is registered to handle is triggered. When that IRQ is triggered, the task will be scheduled (preempting the currently running task). The IRQ will be ignored by the kernel when the task is no sleeping in os_sleep_irq().

Startup

On startup, OpenBLT creates initializes its internal memory and resource management, sets up kernel space at the 2gb line (0x8000000), creates an address space and task for each userland program that it can find. An idle task is created (to give the scheduler something to schedule, should all other tasks be sleeping on something). The kernel schedules the first task in the run queue to execute and the fun starts.

Xtream/OS

Author- Douglas Armstrong

a.k.a. Doug's Wicked OS a.k.a. Vision OS depending on the target audience. Instead of being just another POSIX derivative, the Xtream/OS has gone with a completely different object oriented component model philosophy. This methodology will allow me, in the future through the addition of compatibility modules, to add compatibility with older systems, such as Linux and DOS. In its current stage it functions as a 32-bit preemptive multitasking OS with full support for user applications as well as dynamically loaded kernel modules.

Nomad-
If you want to understand what truly makes this OS different, then you have to know about Nomad. Nomad is an object model somewhat similar to CORBA or COM/DCOM. You can create different objects of many types that conform to one "interface." This means allows flexibility inherent in the system. Since both a terminal or a modem or the console or a TCP/IP connection could be a "stream" interface they could all be used interchangeably by the client. Nomad is even used for IPC (Inter-process communication) between programs by creating a pipe to talk across address spaces. Since pipes can talk between any two components, you could, for example, remotely mount a FAT partition whose driver is an fs (file system) component by simply setting up a network pipe. An even more bizarre situation would be to configure network pipe to use a remote pointer device, such as a mouse on another machine.

Component based system-
If you're not familiar with UNIX, it has a directory /dev/etc for system devices, where they are abstracted as files. Xtream/OS takes this a step forward into a complete and dynamic object system. Everything in

continued on page 21...

...continued from page 20

the OS is an object. The program object will contain a task object for each thread that's running. The task object then contains a stack object. The program object contains a "space" object that represents it's virtual memory space, keeps track of application/dll/pipe/heap memory spans, and has a code object that is inherited from the file interface. Here's where the use of Nomad becomes evident. During run-time you can dynamically load a new code component, such as the ELF file format. The Kernel can then automatically load applications of this type.

API-

Even the API is object orientated. Instead of having an interrupt (or some other mechanism) to handle all system calls, the main kernel API is very limited. It does, however, allow the loading of components. You then talk to these components using whatever interface they have. This means the system's API can change dynamically with time. The shell might load up a turf (short for turface or text surface) to display on. This turf could be the main console, a remote login, a window in a GUI, or whatever. Later on, I plan to extend this concept to graphics. The main graphics card will have a surface interface. The same interface will be used by the windowing system. So an application that previously used the card in full screen mode (for performance) could also act as a window in the windowing system with no change in the application code. Again, you could also setup a network pipe to display on a remote machine with no extra work.

A sampling of current features-
[] Preemptive Multitasking
[] Dynamic linking of ELF files
[] Kernel Debugger (symbolic, breakpoints, stacktrace, memory dumps, system information, single step, heap explorer) Includes Tasker which allows killing and execution of tasks, you can even restart the shell if it dies.
[] Network stack - Ethernet / IP / ICMP / ARP / UDP
[] Nomad
[] VFS - Virtual File System
[] fsboot - Bootstrap file system
[] Exception handlers - gracefully kill misbehaving tasks
[] Very minimal libc implementation
[] DLL support - Not quite finished yet...
[] Ash - Soon to be renamed shell Drivers-
[] NE2000
[] Some standard ISA architecture system devices PIC/PIT/etc...
[] Keyboard
[] Console
[] Soda machine (for the office)
[] COM port (for debugging output only)

This is a quick glance at what's going on behind the scenes in the Xtream/OS.

The Xtream/OS is an independent invention, Copyright 1997, 1998 Douglas Armstrong.

KONK/OS

Author- Erik Gilling

KONK/OS is a completely object based kernel. Each distinct part of the kernel is a C++ object including pager, task, address space, and scheduler. KONK/OS currently in it's infancy will hopefully have network and graphics support by EOH.

The Dave Operating System

Author- David Wentzlaff

The Dave OS is a cooperatively multitasking operating system that was born at the beginning of this school year. When I started to design it I had no deep design goals, but as it evolved I have come up with some. One of the most important is security. Because the Dave OS is 32 Protected mode, it allows a lot of hardware security. However, the Dave OS will not allow direct hardware access, because this can easily violate the system. Another goal of the Dave OS is to have dynamically loadable drivers. This means you can boot the OS without a keyboard driver (for example), load it later and then, when a keyboard is no longer needed, take it out of memory to save space. Also, some day, I would like the OS to have a built in compiler so that users can overcome the limitation of needing to use binaries; rather, they can directly execute source code. Finally, a goal of the Dave OS is to have a built in Java Compiler/Interpreter that is in the kernel and will allow for Drivers to be written in Java.

continued on page 22...

...continued from page 21

Some of its current features are:

* It multitasks!
* It has a unique driver model that allows for drivers to be started and stopped while the OS is running, no need for a reboot.
* It has a keyboard driver.
* It has a NE2000 Network driver. (Special Thanks to Doug Armstrong)
* Started implementation of the Standard "C" Library.
* Each new task has its own address space.
* It has a built in process control so that processes can be stopped, restarted and killed.

Future features to be implemented:

* Make the OS preemptively multitasking.
* Dynamic Linking to DLL's (but implemented correctly)
* Java support.
* Built in compiler and assembler.
* Debugging unit.

acm

student chapter
university of
illinois urbana-
champaign

membership form

Return or mail this form to:
1225 Digital Computer Lab, MC-258
1304 W. Springfield Ave.
Urbana, IL 61801

name:

campus address:

campus phone:

home address:

home phone:

electronic mail:

curriculum:

UNIVERSITY STATUS

- ☐ freshman
- ☐ sophomore
- ☐ junior
- ☐ senior
- ☐ m.a. / m.s.
- ☐ ph.D.
- ☐ faculty / staff
- ☐ postdoc
- ☐ alumni
- ☐ other

special interest groups

lug	linux users group
bug	be users group
sigarch	architecture
sigart	artificial intelligence
sigbio	biocomputing
sigbiz	enterpreneurship
sigcas	computers and society
sigdave	short-term distractions
siggraph	graphics
sigir	information retrieval
sigmicro	microcomputers
sigmusic	music
signet	networking and security
sigops	operating systems
sigsoft	software development
sigunix	unix programming
sigvr	virtual reality

MEMBERSHIP TYPES

return form with check or money order payable to the ACM at UIUC

- ☐ $40 for eight semesters
- ☐ $22 for four semesters
- ☐ $12 for two semesters

ACM NATIONAL MEMBER

- ☐ yes — #
- ☐ no
- ☐ currently applying

for internal use

Received Date: _____ by:_____
Chk #_______ $_______

Enter Date:_______ by: ______