ACM Februrary 7, 2003

HURD

Pankaj Jangid
pankaj@ncst.ernet.in

Introduction

- Free software license, patents...
- Users freedom
- GNU/Linux is not enough
- We need GNU/HURD

Roadmap

- GNU Mach the micro-kernel
- HURD Architecture
- Extending/Developing HURD
- Where to get it ? (Debian and GNU)

History

- 1983: Richard Stallman founds the GNU project.
- 1988: Decision is made to use Mach 3.0 as the kernel.
- 1991: Mach 3.0 is released under GPL compatible license.
- 1991: Thomas Bushnell, BSG, founds the Hurd project.
- 1994: The Hurd boots the first time.
- 1997: Version 0.2 of the Hurd is released.
- 1998: Debian hurd-i386 archive is created.
- 2001: Debian GNU/Hurd snapshot fills three CD images.
- 2003: Debian GNU/Hurd J2 has four CDs now.

OS Responsibilities.

From the paper “Towards a New Strategy of OS Design”

> “The fundamental purpose of an operating system (OS) is to enable a variety of programs to share a single computer efficiently and productively. This demands memory protection, preemptively scheduled timesharing, coordinated access to I/O peripherals, and other services. In addition, an OS can allow several users to share a computer. In this case, efficiency demands services that protect users from harming each other, enable them to share without prior arrangement, and mediate access to physical devices”.

Hurd provides all this with more flexibily.

Kernel Architectures

Micro kernel

- Low level memory management
- Task management
- IPC
- Basic hardware support

Monolithic kernel

- Message passing not required
- Filesys, authentication, network sockets, POSIX interfaces, etc. everything in kernel

Micro Vs Monolithic

Micro kernel

- Clear cut responsibility
- Flexible design, easier debugging
- More stability (less code to break)
- New features not added to the kernel

Monolithic kernel

- Difficult to understand
- Buggy module may cause a crash
- Small changes can have far reaching side effects

Single server Vs Multi server

Single Server

- A single task implements the functionality of the operating system.

Multi Server

- Many tasks cooperate to provide the system's functionality.
- One server provides only a small but well-defined part of the whole system.
- The responsibilities are distributed logically among the servers.

A single-server system is comparable to a monolithic kernel system. It has similar advantages and disadvantages.

Multi server is superior

- Clear cut responsibilities
- More stability: If one server dies, all others remain
- Easier development cycle: Testing without reboot (or replacing running servers), debugging with gdb
- Easier to make changes and add new features

The Hurd goes beyond this..

The Hurd goes beyond all this, and allows users to write and run their servers, too!

- Users can replace system servers dynamically with their own implementations.
- Users can decide what parts of the remainder of the system they want to use.
- Users can extend the functionality of the system.
- No mutual trust necessary to make use of other users services.
- Security of the system is not harmed by trusting users services.

Servers and Translators

- Communication between users (programs) and servers via Mach ports
- Port is created by opening a file in a server owned directory.
- A file can also have a translator associated with it.
- Example – ftp, tar, gzip, etc.

Translators in Hurd

Are used

- To access services that are not structured like hierarchical filesystem.
- Password Servers
- Network protocol server

Can be used

- Transparent CVS
- Conceptual union of other directories
- Dynamic signature “/home/pankaj/signature”

Mach Inter Process Communication (IPC)

Ports are message queues which can be used as one-way communication channels.

- Port rights are receive, send or send-once
- Exactly one receiver
- Potentially many senders

How to get a port ?

Traditional Mach

- Nameserver provides ports to all registered servers.
- The nameserver port itself is provided by Mach.
- Like a phone book: One list.

The Hurd

- The filesystem is used as the server namespace.
- Root directory port is inserted into each task.
- The C library finds other ports with “hurd_file_name_lookup”, performing a pathname resolution.
- Like a tree of phone books.

Example of `hurd_file_name_lookup`

```
mach_port_t identity;
mach_port_t pwserver;
kern_return_t err;

pwserver = hurd_file_name_lookup
                ("/servers/password");

err = password_check_user (pwserver,
                           0 /* root */, "supass",
                           &identity);
```

Pathname resolution example

Task: Lookup /mnt/readme.txt where /mnt has a mounted filesystem.

- The C library asks the root filesystem server about /mnt/readme.txt.
- The root filesystem returns a port to the mnt filesystem server (matching /mnt) and the retry name /readme.txt.
- The C library asks the mnt filesystem server about /readme.txt.
- The mnt filesystem server returns a port to itself and records that this port refers to the regular file /readme.txt.

File System Servers

- Provide file and directory services for ports (and more).
- These ports are returned by a directory lookup.
- Translate filesystem accesses through their root path (hence the name translator).
- The C library maps the POSIX file and directory interface (and more) to RPCs to the filesystem servers ports, but also does work on its own.
- Any user can install file system servers on inodes they own.

Active Translators

- “settrans -a /cdrom /hurd/isofs /dev/hd2”
- Are running filesystem servers.
- Are attached to the root node they translate.
- Run as a normal process.
- Go away with every reboot, or even time out.

Passive Translators:

- “settrans /mnt /hurd/ext2fs /dev/hd1s1”
- Are stored as command strings into an inode.
- Are used to start a new active translator if there isn’t one.
- Startup is transparent to the user.
- Startup happens the first time the server is needed.
- Are permanent across reboots (like file data).

Authentication

A user identity is just a port to an authserver. The auth server stores four set of ids for it:

- effective user ids
- effective group ids
- available user ids
- available group ids

Basic properties:

- Any of these can be empty.
- A 0 among the user ids identifies the superuser.
- Effective ids are used to check if the user has the permission.
- Available ids can be turned into effective ids on user request.

Operations on authentication ports

The auth server provides the following operations on ports:

- Merge the ids of two ports into a new one.
- Return a new port containing a subset of the ids in a port.
- Create a new port with arbitrary ids (superuser only).
- Establish a trusted connection between users and servers.

Establishing trusted connections

- User provides a rendezvous port to the server (with io_reauthenticate).
- User calls auth_user_authenticate on the authentication port (his identity), passing the rendezvous port.
- Server calls auth_server_authenticate on its authentication port (to a trusted auth server), passing the rendezvous port and the server port.
- If both authentication servers are the same, it can match the rendezvous ports and return the server port to the user and the user ids to the server.

Password Server

- The password server “/servers/password” runs as root and returns a new authentication port in exchange for a unix password.
- The ids corresponding to the authentication port match the unix user and group ids.
- Support for shadow passwords is implemented here.

Process Server

The superuser must remain control over user tasks, so:

- All mach tasks are associated with a PID in the system default proc server.

Optionally, user tasks can store:

- Their environment variables.
- Their argument vector.
- A port, which others can request based on the PID (like a nameserver).

Also implemented in the proc server:

- Sessions and process groups.
- Global configuration not in Mach, like hostname, hostid, system version.

User tasks not registering themselve with proc only have a PID assigned.
Users can run their own proc server in addition to the system default, at least for those parts of the interface that don't require superuser privileges.

Filesystems

Store based filesystems

- ext2fs
- ufs
- isofs (iso9660, RockRidge, GNU extensions)
- fatfs (under development)

Network file systems

- nfs
- ftpfs

Miscellaneous

- tmpfs (under development)

Developing the Hurd

Over a dozen libraries support the development of new servers.
For special server types highly specialized libraries require only the implementation of a number of callback functions.

- Use libdiskfs for store based filesystems.
- Use libnetfs for network filesystems, also for virtual filesystems.
- Use libtrivfs for simple filesystems providing only a single file or directory.

Debian GNU/Hurd

Goal:

- Provide a binary distribution of the Hurd that is easy to install.

Constraints:

- Use the same source packages as Debian GNU/Linux.
- Use the same infrastructure:
  - Policy
  - Archive
  - Bug tracking system
  - Release process

Side Goal:

- Prepare Debian for the future:
  - More flexibility in the base system
  - Identify dependencies on the Linux kernel

Debian GNU/Hurd: Good idea, bad idea?

Upstream benefits:

- Software packages become more portable.

Debian benefits:

- Debian becomes more portable.
- Maintainers learn about portability and other systems.
- Debian gets a lot of public recognition.

GNU/Hurd benefits:

- Large software base.
- Great infrastructure.
- Nice community to partner with.

References

- Hurd home page
  http://www.gnu.org/software/hurd/
- Towards a New Strategy of OS Design
  http://www.gnu.org/software/hurd/hurd-paper.html
- The Hurd Talk by Marcus Brinkmann
  http://www.gnu.org/software/hurd/hurd-talk.html
- The Hurd Hacking Guide, an introduction to GNU Hurd and Mach programming by Wolfgang Jahrling
  http://www.gnu.org/software/hurd/hacking-guide/hhg.html
- Debian GNU/Hurd
  http://www.debian.org/ports/hurd