A

Seminar report

On

Hurd

Submitted in partial fulfillment of the requirement for the award of degree
Of CSE

SUBMITTED TO:

www.studymafia.org

SUBMITTED BY:

www.studymafia.org

www.studymafia.org

Preface

I have made this report file on the topic **Hurd**; I have tried my best to elucidate all the relevant detail to the topic to be included in the report. While in the beginning I have tried to give a general view about this topic.

My efforts and wholehearted co-corporation of each and everyone has ended on a successful note. I express my sincere gratitude to ……..who assisting me throughout the preparation of this topic. I thank him for providing me the reinforcement, confidence and most importantly the track for the topic whenever I needed it.

www.studymafia.org

Acknowledgement

I would like to thank respected Mr…….. and Mr. ……..for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a seminar report. It helped me a lot to realize of what we study for.

Secondly, I would like to thank my parents who patiently helped me as i went through my work and helped to modify and eliminate some of the irrelevant or un-necessary stuffs.

Thirdly, I would like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Next, I would thank Microsoft for developing such a wonderful tool like MS Word. It helped my work a lot to remain error-free.

Last but clearly not the least, I would thank The Almighty for giving me strength to complete my report on time.

www.studymafia.org

CONTENTS

www.studymafia.org

INTRODUCTION

When we talk about free software, we usually refer to the free software licenses. We also need relief from software patents, so our freedom is not restricted by them. But there is a third type of freedom we need, and that's user freedom.

Expert users don't take a system as it is. They like to change the configuration, and they want to run the software that works best for them. That includes window managers as well as your favourite text editor. But even on a GNU/Linux system consisting only of free software, you can not easily use the filesystem format, network protocol or binary format you want without special privileges. In traditional Unix systems, user freedom is severly restricted by the system administrator.

The Hurd is built on top of CMU's Mach 3.0 kernel and uses Mach's virtual memory management and message-passing facilities. The GNU C Library will provide the Unix system call interface, and will call the Hurd for needed services it can't provide itself. The design and implementation of the Hurd is being lead by Michael Bushnell, with assistance from Richard Stallman, Roland McGrath, Jan Brittenson, and others.

www.studymafia.org

A MORE USABLE APPROACH TO 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.

On today's computer systems, programmers usually implement these goals through a large program called the kernel. Since this program must be accessible to all user programs, it is the natural place to add functionality to the system. Since the only model for process interaction is that of specific, individual services provided by the kernel, no one creates other places to add functionality. As time goes by, more and more is added to the kernel.

A traditional system allows users to add components to a kernel only if they both understand most of it and have a privileged status within the system. Testing new components requires a much more painful edit-compile-debug cycle than testing other programs. It cannot be done while others are using the system. Bugs usually cause fatal system crashes, further disrupting others' use of the system. The entire kernel is usually non-pageable. (There are systems with pageable kernels, but deciding what can be paged is difficult and error prone. Usually the mechanisms are complex, making them difficult to use even when adding simple extensions.)

Because of these restrictions, functionality which properly belongs behind the wall of a traditional kernel is usually left out of systems unless it is absolutely mandatory. Many good ideas, best done with an open/read/write interface cannot be implemented because of the problems inherent in the monolithic nature of a traditional system. Further, even among those with the endurance to implement new ideas, only those who are privileged users of their computers can do so. The software copyright system darkens the mire by preventing unlicensed people from even reading the kernel source The Hurd removes these restrictions

www.studymafia.org

from the user. It provides an user extensible system framework without giving up POSIX compatibility and the unix security model.

When Richard Stallman founded the GNU project in 1983, he wanted to write an operating system consisting only of free software. Very soon, a lot of the essential tools were implemented, and released under the GPL. However, one critical piece was missing: The kernel. After considering several alternatives, it was decided not to write a new kernel from scratch, but to start with the Mach microkernel. This was in 1988, and it was not before 1991 that Mach was released under a license allowing the GNU project to distribute it as a part of the system.

Kernel Architectures

Microkernels were very popular in the scientific world around that time. They don't implement a full operating system, but only the infrastructure needed to enable other tasks to implement most features. In contrast, monolithical kernels like Linux contain program code of device drivers, network protocols, process management, authentication, file systems, POSIX compatible interfaces and much more.

So what are the basic facilities a microkernel provides? In general, this is resource management and message passing. Resource management, because the kernel task needs to run in a special privileged mode of the processor, to be able to manipulate the memory management unit and perform context switches (also to manage interrupts). Message passing, because without a basic communication facility the other tasks could not interact to provide the system services. Some rudimentary hardware device support is often necessary to bootstrap the system. So the basic jobs of a microkernel are enforcing the paging policy (the actual paging can be done by an external pager task), scheduling, message passing and probably basic hardware device support.

Mach was the obvious choice back then, as it provides a rich set of interfaces to get the job done. Beside a rather brain-dead device interface, it provides tasks and threads, a messaging system allowing synchronous and asynchronous operation and a complex

www.studymafia.org

interface for external pagers. The GNU project maintains its own version of Mach, called GNU Mach, which is based on Mach 4.0. In addition to the features contained in Mach 4.0, the GNU version contains many of the Linux 2.0 block device and network card drivers.

Micro vs Monolithic

Microkernel

- *Clear cut responsibilities*
- *Flexibility in operating system design, easier debugging*
- *More stability (less code to break)*
- *New features are not added to the kernel*

Monolithic kernel

- *Intolerance or creeping featuritis*
- *Danger of spaghetti code*

Small changes can have far reaching side effects Because the system is split up into several components, clean interfaces have to be developed, and the responsibilities of each part of the system must be clear.

Once a microkernel is written, it can be used as the base for several different operating systems. Those can even run in parallel which makes debugging easier. When porting, most of the hardware dependant code is in the kernel.

Much of the code that doesn't need to run in the special kernel mode of the processor is not part of the kernel, so stability increases because there is simply less code to break. New features are not added to the kernel, so there is no need to hold the barrier high for new operating system features. Compare this to a monolithical kernel, where you either suffer from creeping featuritis or you are intolerant of new features (we see both in the Linux kernel).

www.studymafia.org

Because in a monolithical kernel, all parts of the kernel can access all data structures in other parts, it is more likely that short cuts are used to avoid the overhead of a clean interface. This leads to a simple speed up of the kernel, but also makes it less comprehensible and more error prone. A small change in one part of the kernel can break remote other parts.

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.

There exist a couple of operating systems based on Mach, but they all have the same disadvantages as a monolithical kernel, because those operating systems are implemented in one single process running on top of the kernel. This process provides all the services a monolithical kernel would provide. This doesn't make a whole lot of sense (the only advantage is that you can probably run several of such isolated single servers on the same machine). Those systems are also called single-server systems. The Hurd is the only usable multi-server system on top of Mach. In the Hurd, there are many server programs, each one responsible for a unique service provided by the operating system. These servers run as Mach tasks, and communicate using the Mach message passing facilities. One of them does only provide a small part of the functionality of the system, but together they build up a complete and functional POSIX compatible operating system.

www.studymafia.org

MACH MEMORY MANAGEMENT

The Mach virtual memory architecture has some unique features such as the ability to provide much of the functionality through user level tasks. The second is the issue of translation lookaside buffer consistency on multiprocessors. The third is the problem of using virtually addressed caches correctly and efficiently.

Design Goals

Mach provide a rich set of features including the following:

- *Copy-on-write and read-write sharing of memory between related and unrelated tasks*
- *Memory-mapped file access*
- *Large,sparsely populated address space*
- *Memory sharing between processes on different machines*
- *User control over page replacement policies*

Mach separates all machine-dependent code into a small pmap layer.This makes it easy to port Mach to a new hardware related architecture, since only pmap layer needs to be rewritten. The rest of the code is machine-independent and not modeled after any specific MMU architecture.

An important objective in the Mach VM design is to push much of the VM functionality out of the kernel. From its conception, Mach's microkernel architecture allowed the traditional kernel level functionality provided by user-level server tasks. Hence Mach VM relegates functions such as paging to user level tasks.

Finally Mach integrates the memory level management and IPC subsystems to gain two advantages. The location-independence of Mach IPC allows virtual memory facilities to be transparently extended to a distributed environment. Conversely, the copy-on-write sharing supported by the VM subsystem allows the faster tranfer of large messages. There are however some important drawbacks. The VM system is larger, slower,

www.studymafia.org

and more complex than the BSD design. It uses more and larger data structures. Hence it consumes more phyical memory for itself, leaving less available for the processes. Since the design keeps machine dependent code to a minimum, it cannot be properly optimized for any particular MMU architecture.

In addition, the se of message passing adds considerable overhead. The cost is reduced in some cases by optimizing kernel-to kernel message transfers. Overall, though, message passing is still a lot more expensive than simple function calls. Except for the network shared memory manager, external pagers are not used commonly.

Multiprocessor issues for TLB

Maintaining TLB consistency on a shared-memory multiprocessor is a much more complex problem. Although all processors share the main memory, each has its own TLB. Problems arise when one processor changes an entry in a page table that may be active on another processor. The latter may have a copy of that entry in its TLB and hence may continue to use obsolete mapping. It is essential to propagate the change to the TLBs of any processor that may be using the page table.

There are many situations in which a change to one page affects TLBs on several processors:

- *The page is a kernel page*
- *The page is shared by multiple processes, each running on a different processor.*
- *On a multi threaded systems, different threads of the same process may be running concurrently on different processors. If one thread modifies a mapping, all threads must see the change.*

In the absence of hardware support, the kernel must solve the problem in software using a notification mechanism based on cross-processor interrupts.

www.studymafia.org

TLB shootdown in mach

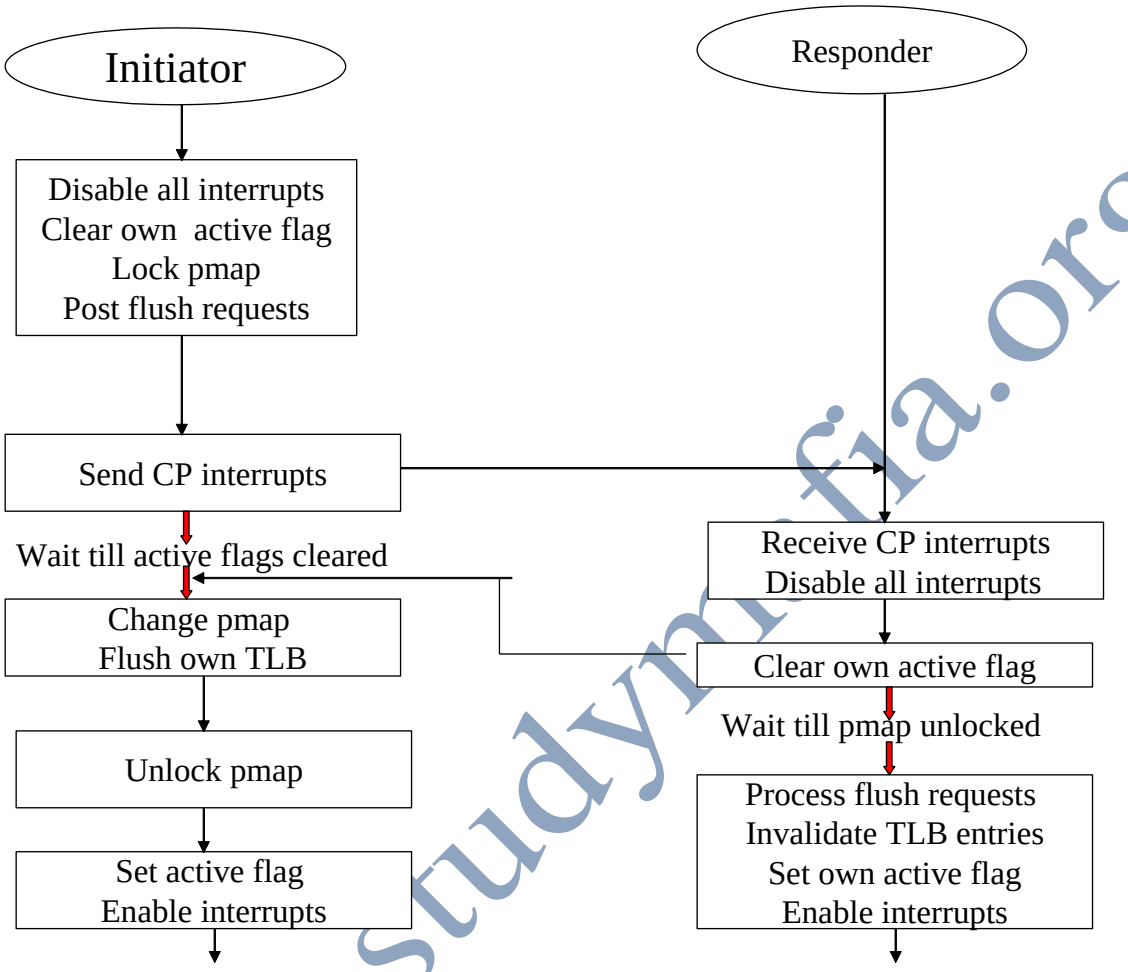

FIG 1 :Mach TLB shootdown algorithm

The Mach

- *An active flag, which shows whether the processor is actively using some page table. If this flag is clear, the processor is participating in shootdown and will not access and modifiable pmap entry.*
- *A queue of invalidation requests. Each request specifies a mapping that must be flushed from the TLB.*
- *A set of currently active pmaps. Each processor usually has two active pmaps-the kernel pmap and that of the current task.*

Each pmap is protected by a spin lock, which serializes operations on it. Each pmap also has a list of processors on which the pmap is currently active.

www.studymafia.org

Synchronization and deadlock avoidance

The shootdown algorithm uses several synchronization mechanisms, and the precise order of the operations is important. It is important to disable all interrupts, otherwise a device interrupt can idle multiple processors for a long time. The lock on the pmap prevents two processors from simultaneously initiating the shootdowns for the same pmap. The interrupts must be disabled before locking the page table, or a processor may deadlock when it receives a cross-processor interrupt while holding a lock.

The initiator clears its own active flag before locking the pmap, to avoid some deadlock conditions. Suppose two processors, P1 and P2, attempt to modify the same pmap. P1 disables interrupts, locks the map, and sends an interrupt to P2. Meanwhile, P2 disables interrupt and blocks on the same lock. Now we have a deadlock, since P1 is waiting for P2 to acknowledge the interrupt, and P2 is waiting for P1 to release the pmap.

Clearing the active flag effectively acknowledges interrupts before they arrive. In the above example, P1 will not block since P2 clears its flag before trying to lock the pmap. When P1 unlocks the pmap, P2 will not resume and process the flush request posted by P1.

The shootdown algorithm has a subtle effect on all resource locking. It requires a consistent policy about whether interrupts are disabled before acquiring a lock. Suppose P1 holds a resource with interrupts enabled, P2 tries to acquire it with interrupts disabled, and P3 initiates a shootdown with P1 and P2 as responders. P3 sends a cross processor image to P1 and P2, and blocks till they are acknowledged. P1 acknowledges its interrupt and blocks until the pmap is released. P2 is blocked on the lock with interrupts disabled and hence does not see or respond to the interrupt As a result we have a three way deadlock. To prevent this, the system must enforce a fixed interrupt state for each lock: Either a lock should always be acquired with interrupts disabled or with interrupts enabled.

Page Replacement

www.studymafia.org

The Mach page replacement uses three FIFO lists-active, inactive, and free Pages migrate from one list to another in the following ways:

1. The first reference to a page results in a page fault. The fault handler removes a page from the head of the free list and initializes it with appropriate data. It then puts the page at the tail of the active list. Eventually, the page migrates to the head of this list, and pages ahead of it become inactive.
2. Whenever free memory falls below a threshold value, the pagedaemon is awakened. It removes some pages from the head of the active list to the tail of the inactive list. It turns off the reference bit in the hardware address translation mappings for these pages.
3. The pagedaemon also examines the number of pages at the head of the inactive list. Pages whose reference bits are set are returned to the tail of the active list.
4. If the pagedaemon finds a page number whose reference bit is still clear, the page has not been referenced while on the inactive queue and can be moved to the tail of the free list. If the page is dirty, it is first written back to its memory object.
5. If the page is referenced while on the free list, it may still be reclaimed. In this case, it goes back to the tail of the active list. Otherwise, it will migrate to the head of the free list and eventually be reused.

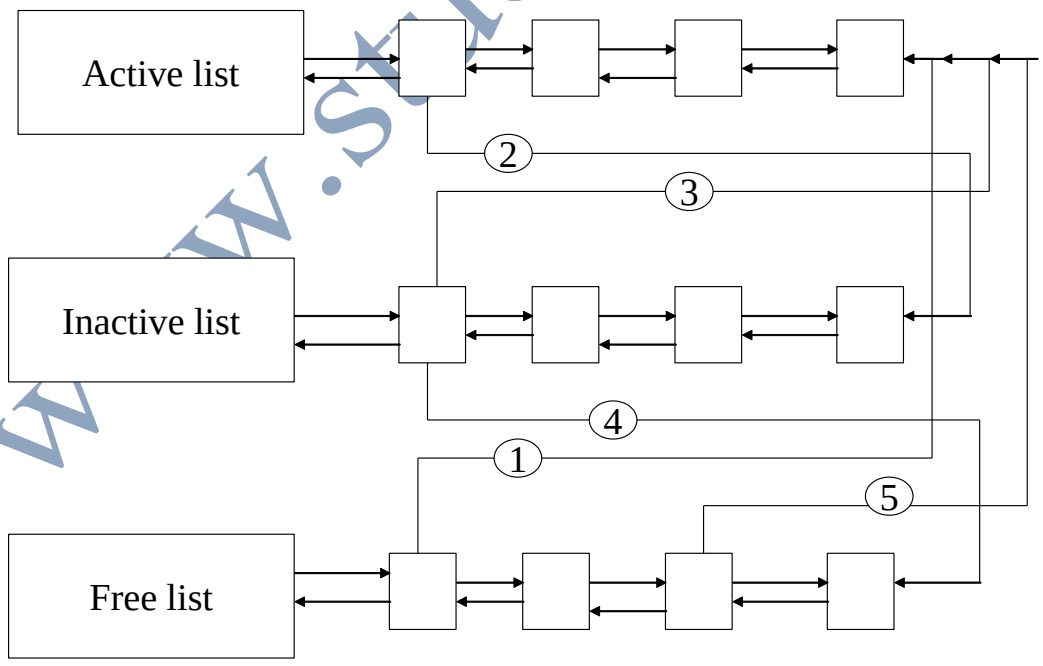

Fig 2 Page replacement

www.studymafia.org

MACH INTER PROCESS COMMUNICATION

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*

MiG provides remote procedure calls on top of Mach IPC. RPCs look like function calls to the user.

Inter-process communication in Mach is based on the ports concept. A port is a message queue, used as a one-way communication channel. In addition to a port, you need a port right, which can be a send right, receive right, or send-once right. Depending on the port right, you are allowed to send messages to the server, receive messages from it, or send just one single message.

For every port, there exists exactly one task holding the receive right, but there can be no or many senders. The send-once right is useful for clients expecting a response message. They can give a send-once right to the reply port along with the message. The kernel guarantees that at some point, a message will be received on the reply port (this can be a notification that the server destroyed the send-once right).

You don't need to know much about the format a message takes to be able to use the Mach IPC. The Mach interface generator mig hides the details of composing and sending a message, as well as receiving the reply message. To the user, it just looks like a function call, but in truth the message could be sent over a network to a server running on a different computer. The set of remote procedure calls a server provides is the public interface of this server.

Ports also represent kernel objects. Hence each object, such as task, thread or process is represented by a port. Rights to these ports represent object references and allow

www.studymafia.org

the holder to perform operations on that object. The kernel holds the receive rights to such ports.

Each port has a finite-size message queue. The size of this queue provides a simple flow-control mechanism. Senders are blocked when the queue is full, and the receivers when the queue is empty.

Each task and thread has a set of default ports. For instance each task has send rights to a task_self_port that represents itself and receive rights to a task_notify_port. Tasks also have send rights to a bootstrap port that provides access to a name server. Each thread has send rights to a thread_self port, and receive rights to a reply port, used to receive replies from system calls and remote procedure calls to other tasks. There is an exception port associated with each task and each thread. The rights to the per-thread ports are owned by the task in which the thread runs; hence these ports can be accessed by all threads within the task.

Tasks also inherit other port rights from their parents. Each task has a list of registered ports. These allow the task to access various system-wide services. These ports are inherited by new tasks during task creation.

Messages

Mach is a message-passing kernel, and most system services are accessed by exchanging messages. Mach IPC provides communication between user tasks, between user and kernel and between different kernel subsystems. A user-level program called the netmsgserver transparently extends Mach IPC across network, so tasks can exchange messages with remote tasks as easily as with local ones. The fundamental abstractions of Mach IPC are the messages and the port.

How to get a port?

- *The filesystem is used as the server namespace.*
- *Root directory port is inserted into each task.*

www.studymafia.org

- *The C library finds other ports with hurd_file_name_lookup, performing a pathname resolution. Like a tree of phone books.*

You need something like a phone book for server ports, or otherwise you can only talk to yourself. In the original Mach system, a special nameserver is dedicated to that job. A task could get a port to the nameserver from the Mach kernel and ask it for a port (with send right) to a server that registered itself with the nameserver at some earlier time.

In the Hurd, there is no nameserver. Instead, the filesystem is used as the server namespace. This works because there is always a root filesystem in the Hurd (remember that the Hurd is a POSIX compatible system); this is an assumption the people who developed Mach couldn't make, so they had to choose a different strategy. You can use the function hurd_file_name_lookup, which is part of the C library, to get a port to the server belonging to a filename. Then you can start to send messages to the server in the usual way.

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.*

So we don't have a single phone book listing all servers, but rather a tree of servers keeping track of each other. That's really like calling your friend and asking for the phone number of the blond girl at the party yesterday. He might refer you to a friend who hopefully knows more about it. Then you have to retry.

This mechanism has huge advantages over a single nameserver. First, note that standard unix permissions on directories can be used to restrict access to a server (this requires that the filesystems providing those directories behave). You just have to set the permissions of a parent directory accordingly and provide no other way to get a server port.

www.studymafia.org

But there are much deeper implications. Most of all, a pathname never directly refers to a file, it refers to a port of a server. That means that providing a regular file with static data is just one of the many options the server has to service requests on the file port. A server can also create the data dynamically.

While a regular filesystem server will just serve the data as stored in a filesystem on disk, there are servers providing purely virtual information, or a mixture of both. It is up to the server to behave and provide consistent and useful data on each remote procedure call. If it does not, the results may not match the expectations of the user and confuse him.

Passive Translators

Many translator settings remain constant for a long time. It would be very lame to always repeat the same couple of dozens settrans calls manually or at boot time. So the Hurd provides a filesystem extension that allows to store translator settings inside the filesystem and let the filesystem servers do the work to start those servers on demand. Such translator settings are called "passive translators". A passive translator is really just a command line string stored in an inode of the filesystem. If during a pathname resolution a server encounters such a passive translator, and no active translator does exist already (for this node), it will use this string to start up a new translator for this inode, and then let the C library continue with the path resolution as described above. Passive translators are installed with settrans using the -p option (which is already the default).

So passive translators also serve as a sort of automounting feature, because no manual interaction is required. The server start up is deferred until the service is need, and it is transparent to the user.

When starting up a passive translator, it will run as a normal process with the same user and group id as those of the underlying inode. Any user is allowed to install passive and active translators on inodes that he owns. This way the user can install new servers into the global namespace (for example, in his home or tmp directory) and thus extend the functionality of the system (recall that servers can implement other remote procedure calls

www.studymafia.org

beside those used for files and directories). A careful design of the trusted system servers makes sure that no permissions leak out.

In addition, users can provide their own implementations of some of the system servers instead the system default. It was already mentioned that only few system servers are mandatory for users. To establish your identity within the Hurd system, you have to communicate with the trusted systems authentication server auth. To put the system administrator into control over the system components, the process server does some global bookkeeping.

But even these servers can be ignored. However, registration with the authentication server is the only way to establish your identity towards other system servers. Likewise, only tasks registered as processes with the process server can make use of its services.

Authentication

A user identity is just a port to an auth server. 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 super user.*
- *Effective ids are used to check if the user has the permission.*
- *Available ids can be turned into effective ids on user request.*

The Hurd auth server is used to establish the identity of a user for a server. Such an identity (which is just a port to the auth server) consists of a set of effective user ids, a set of effective group ids, a set of available user ids and a set of available group ids. Any of these sets can be empty.

www.studymafia.org

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 (super user only).*
- *Establish a trusted connection between users and servers.*

If you have two identities, you can merge them and request an identity consisting of the unions of the sets from the auth server. You can also create a new identity consisting only of subsets of an identity you already have. What you can't do is extending your sets, unless you are the super user, which is denoted by having the user id 0.

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.*

Finally, the auth server can establish the identity of a user for a server. This is done by exchanging a server port and a user identity if both match the same rendezvous port. The server port will be returned to the user, while the server is informed about the id sets of the user. The server can then serve or reject subsequent RPCs by the user on the server port, based on the identity it received from the auth server.

www.studymafia.org

Anyone can write a server conforming to the auth protocol, but of course all system servers use a trusted system auth server to establish the identity of a user. If the user is not using the system auth server, matching the rendezvous port will fail and no server port will be returned to the user. Because this practically requires all programs to use the same auth server, the system auth server is minimal in every respect, and additional functionality is moved elsewhere, so user freedom is not unnecessarily restricted.

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.

The password server sits at /servers/password and runs as root. It can hand out ports to the auth server in exchange for a unix password, matching it against the password or shadow file. Several utilities make use of this server, so they don't need to be setuid root.

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 name server).*

Also implemented in the proc server:

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

www.studymafia.org

The process server is responsible for some global bookkeeping. As such it has to be trusted and is not replaceable by the user. However, a user is not required to use any of its service. In that case the user will not be able to take advantage of the POSIXish appearance of the Hurd.

The Mach Tasks are not as heavy as POSIX processes. For example, there is no concept of process groups or sessions in Mach. The proc server fills in the gap. It provides a PID for all Mach tasks, and also stores the argument line, environment variables and other information about a process. A process can also register a message port with the proc server, which can then be requested by anyone. So the proc server also functions as a name server using the process id as the name.

The proc server also stores some other miscellaneous information not provided by Mach, like the hostname, hostid and system version. Finally, it provides facilities to group processes and their ports together, as well as to convert between pids, process server ports and mach task ports.

User tasks not registering themselves 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 super user privileges.

Although the system default proc server can't be avoided (all Mach tasks spawned by users will get a pid assigned, so the system administrator can control them), users can run their own additional process servers if they want, implementing the features not requiring super user privileges.

File systems

Store based file systems

- *ext2fs*

www.studymafia.org

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

Network file systems

- *nfs*
- *ftpfs*

Miscellaneous

- *hostmux*
- *usermux*
- *tmpfs (under development)*

Currently, we have translators for the ext2, ufs and iso9660 file systems. We also have an nfs client and an ftp file system. Especially the latter is intriguing, as it provides transparent access to ftp servers in the file system. Programs can start to move away from implementing a plethora of network protocols, as the files are directly available in the file system through the standard POSIX file interface.

www.studymafia.org

A Look at Some of the Hurd's Disadvantages

The Authentication Server

One of the Hurd's more central servers is the authentication server. Any user could write a program which implements the authentication protocol; this does not violate the system's security. When a service needs to authenticate a user, it communicates with its trusted authentication server. If that user is using a different authentication server, the transaction will fail and the server can refuse to communicate further. Because, in effect, this forces all programs on the system to use the same authentication server, we have designed its interface to make any safe operation possible, and to include no extraneous operations. (This is why there is a separate password server.)

The Process Server

One the process server maintains a one-to-one mapping between Mach tasks and Hurd processes. Every task is assigned a pid. Processes can register a message port with this server, which can then be given out to any program which requests it. This server makes no attempt to keep these message ports private, so user programs are expected to implement whatever security they need themselves. (The GNU C Library provides convenient functions for all this.) Processes can tell the process server their current `argv' and `envp' values; this server will then provide, on request, these vectors of arguments and environment. This is useful for writing ps-like programs and also makes it easier to hide or change this information. None of these features are mandatory. Programs are free to disregard all of this and never register themselves with the process server at all. They will, however, still have a pid assigned.

It is important to stress that the process server is optional. Because of restrictions in Mach, programs must run as root in order to identify all the tasks in the system. But given that, multiple process servers could co-exist, each with their own clients, giving their own model of the universe. Those process server features which do not require root privileges to be implemented could be done as per-user servers. The user's hands are not tied.

www.studymafia.org

Transparent FTP

Transparent FTP is an intriguing idea whose time has come. The popular ange-ftp package available for GNU Emacs makes access to FTP files virtually transparent to all the Emacs file manipulation functions. Transparent FTP does the same thing, but in a system wide fashion. This server is not yet written; the details remain to be fleshed out, and will doubtless change with experience.

It violates all the layering principles of such systems to place such things in the kernel. The unfortunate side effect, however, is that the design methodology (which is based on preventing users from changing things they don't like) is being used to prevent system designers from making things better.

Terminals

An I/O server will provide the terminal semantics of Posix. The GNU C Library has features for keeping track of the controlling terminal and for arranging to have proper job control signals sent at the proper times, as well as features for obeying keyboard and hang-up signals.

Programs will be able to insert a terminal driver into communications channels in a variety of ways. Servers like rlogind will be able to insert the terminal protocol onto their network communication port. Pseudo-terminals will not be necessary, though they will be provided for backward compatibility with older programs. No programs in GNU will depend on them.

Nothing about a terminal driver is forced upon users. A terminal driver allows a user to get at the underlying communications channel easily, to bypass itself on an as-needed basis or altogether, or to substitute a different terminal driver-like program. In the last case, provided the alternate program implements the necessary interfaces, it will be used by the C Library exactly as if it were the ordinary terminal driver.

www.studymafia.org

Because of this flexibility, the original terminal driver will not provide complex line editing features, restricting itself to the behavior found in Posix and BSD. In time, there will be a readline-based terminal driver, which will provide complex line-editing features for those users who want them.

The terminal driver will probably not provide good support for the high-volume, rapid data transmission required by UUCP or SLIP. Those programs do not need any of its features. Instead they will be use the underlying Mach device ports for terminals, which support moving large amounts of data efficiently.

Executing Programs

The implementation of the execve call is spread across three programs. The library marshals the argument and environment vectors. It then sends a message to the file server that holds the file to be executed. The file server checks execute permissions and makes whatever changes it desires in the exec call. For example, if the file is marked setuid and the fileserver has the ability, it will change the user identification of the new image. The file server also decides if programs which had access to the old task should continue to have access to the new task. If the file server is augmenting permissions, or executing an unreadable image, then the exec needs to take place in a new Mach task to maintain security.

After deciding the policy associated with the new image, the file system calls the exec server to load the task. This server, using the BFD (Binary File Descriptor) library, loads the image. BFD supports a large number of object file formats; almost any supported format will be executable. This server also handles scripts starting with #!, running them through the indicated program.

The standard exec server also looks at the environment of the new image; if it contains a variable EXECSERVERS then it uses the programs specified there as exec servers instead of the system default. The new image starts running in the GNU C Library, which sends a message to the exec server to get the arguments, environment, umask,

www.studymafia.org

current directory, etc. None of this additional state is special to the file or exec servers; if programs wish, they can use it in a different manner than the Library.

Network Protocols

Currently, this includes the CCITT protocols, the TCP/IP protocols, the Xerox NS protocols, and the ISO protocols.

For optimal performance some work would be necessary to take advantage of Hurd features that provide for very high speed I/O. For most protocols this will require some thought, but not too much time. The Hurd will run the TCP/IP protocols as efficiently as possible.

As an interesting example of the flexibility of the Hurd design, consider the case of IP trailers, used extensively in BSD for performance. While the Hurd will be willing to send and receive trailers, it will gain fairly little advantage in doing so because there is no requirement that data be copied and avoiding copies for page-aligned data is irrelevant.

www.studymafia.org

Who Should Use the Hurd?

The GNU/Hurd is not the system you use for web-surfing, email, word processing, and other such tasks now ... it is the system that you will use for these tasks in the future.

Anyone who might be interested in the Hurd: a student studying the system, a programmer helping to develop the Hurd servers, or an end-user finding bugs or writing documentation, will be interested in how GNU/Hurd is similar to, and different from, Unix-like kernels.

For all intents and purposes, the Hurd is a modern Unix-like kernel, like Linux and the BSDs. GNU/Hurd uses the GNU C Library, whose development closely tracks standards such as ANSI/ISO, BSD, POSIX, Single Unix, SVID, and X/Open. Hence, most programs available on GNU/Linux and BSD systems will eventually be ported to run on GNU/Hurd systems.

Although it is similar to other Free Unix-like kernel projects, the Hurd has the potential to be much more. Unlike these other projects, the Hurd has an object-oriented structure that allows it to evolve without compromising its design. This structure will help the Hurd undergo major redesign and modifications without having to be entirely rewritten. This extensibility makes the Hurd an attractive platform for learning how to become a kernel hacker or for implementing new ideas in kernel technology, as every part of the system is designed to be modified and extended. For example, the MS-DOS FAT filesysetem was not supported by GNU/Hurd until a developer wrote a translator that allows us to access this filesystem. In a standard Unix-like environment, such a feature would be put into the kernel. In GNU/Hurd, this is done in a different manner and recompiling the kernel is not necessary, since the filesystem is implemented as a user-space program.

Scalability has traditionally been very difficult to achieve in Unix-like systems. Many computer applications in both science and business require support for symmetric multiprocessing (SMP). At the time of this writing, Linux could scale to a maximum of 8

www.studymafia.org

processors. By contrast, the Hurd implementation is aggressively multi-threaded so that it runs efficiently on both single processors and symmetric multiprocessors. The Hurd interfaces are designed to allow transparent network clusters (collectives), although this feature has not yet been implemented.

Of course, the Hurd (currently) has its limitations. Many of these limitations are due to GNU Mach, the microkernel on which the Hurd runs. For example, although the Hurd has the potential to be a great platform for SMP, no such multiprocessing is currently possible, since GNU Mach has no support for SMP. The Hurd has supports less hardware than current versions of Linux, since GNU Mach uses the hardware drivers from version 2.0 of the Linux kernel. Finally, GNU Mach is a very slow microkernel, and contributes to the overall slowness of GNU/Hurd systems.

The Hurd is still under active development, and no stable release has been made. This means that the Hurd's code base is much less mature than that of Linux or the BSDs. There are bugs in system that are still being found and fixed. Also, many features, such as a DHCP client, and support for several filesystem types, is currently missing.

The deficiencies in the Hurd are constantly being addressed; for example, until recently, pthreads (POSIX threads), were missing. This meant that several major applications, including GNOME, KDE, and Mozilla, could not run on GNU/Hurd. Now that the Hurd has a preliminary pthreads implementation, we may soon see these applications running on GNU/Hurd systems.

The Hurd is a very modern design. It is more modern than Linux or the BSDs, because it uses a microkernel instead of a monolithic kernel. It is also more modern than Apple's Darwin or Microsoft's NT microkernel, since the it has a multi-sever design, as opposed to the single-server design of Darwin and NT. This makes the Hurd an ideal platform for students interested in operating systems, since its design closely matches the recommendations of current operating system theory. In addition, the Hurd's modular nature makes it much easier to understand than many other kernel projects.

GNU/Hurd is also an excellent system for developers interested in kernel hacking. Whereas Linux and the BSDs are quite stable, there is still much work to be done on the

www.studymafia.org

Hurd; for example, among all the filesystem types in use today, the Hurd only supports ext2fs (the Linux filesystem), ufs (the BSD filesystem), and iso9660fs (the CD filesystem). GNU/Hurd offers a developer the opportunity to make a substantial contribution to a system at a relatively-early point in its development.

www.studymafia.org

CONCLUSION

In 1998 Marcus Brinkmann brought the Hurd to the Debian Project. This allowed GNU/Hurd to have a software management system(apt and dpkg), world wide access to the latest free software, and a extreme increase of popularity.

As far as a multi-server microkernel based operating systems are concerned: GNU/Hurd has had it's share of critisim. Many people have criticized the Hurd for its lack of a stable release, its design, which is unfamiliar to those used to monolithic kernels such as Linux and the Hurd an attractive platform for learning how to become a kernel hacker or for implementing new ideas in kernel technology, as every part of the system is designed to be modified and extended.

Scalability has traditionally been very difficult to achieve in Unix-like systems. Many computer applications in both science and business require support for symmetric multiprocessing (SMP). At the time of this writing, Linux could scale to a maximum of 8 processors. By contrast, the Hurd implementation is aggressively multi-threaded so that it runs efficiently on both single processors and symmetric multiprocessors. The Hurd interfaces are designed to allow transparent network clusters (collectives), although this feature has not yet been implemented.

www.studymafia.org

REFERENCES

- www.google.com
- www.wikipedia.com
- www.studymafia.org

www.studymafia.org