INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE: CENTRATOM TRIESTE

UNITED NATIONS INDUSTRIAL DEVELOPMENT ORGANIZATION

INTERNATIONAL CENTRE FOR SCIENCE AND HIGH TECHNOLOGY
c/o INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS 34100 TRIESTE (ITALY) VIA GRIGNANO, 9 (ADRIATICO PALACE) P.O. BOX 586 TELEPHONE (040) 224172 TELEFAX (040) 224163 TELEX 460449 APH I

SMR/474 - 14

COLLEGE ON
"THE DESIGN OF REAL-TIME CONTROL SYSTEMS"
1 - 26 October

MINIX

M. REIS
Laboratorio de Instrumentacao e Fisica
Experimental de Particulas (L.I.P.)
Av. Elias Garcia 14-1
Lisbon 1000
Portugal

What is Minix ?

- Multiuser Multiprogramming Operating System .
- Unix Version 7 compatible.
  (IPC mechanisms are restricted to pipes and signals)
- Designed to be readable and used for teaching operating systems

ADVANTAGES:

- Sources may be bought from Prentice Hall for a moderate fee.
- The system may be modified by the user to fit his needs

DISADVANTAGE:

- Minix as provided is not as efficient as an optimized operating system (e.g. Unix)

Minix:General Information

Supported Hardware

IBM: PC, XT, AT, PS/2

clones:

AT&T 6300,	AMIGA 100/Sidecar
AMSTRAD	AMSTRAD Portable
COMMODORE PC	COMPAQ Deskpro
COMPAQ Portable	DEC VAXmate
HP Vectra	OLIVETTI M24
TANDY 1000	TOSHIBA
UNISYS micro IT	ZENITH

Software Support

- Andrew Tanenbaum controls official releases.
- Usenet Newsgroup: comp.os.minix publishes:
  - bugs and code patches.
  - new tools and library routines.
  - general problem discussion.
    (on Bitnet distribution list is INFO-MINIX)

Minix:Related References

The Book

Tanenbaum, A., *Operating Systems: Design and Implementation,* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987 **(Contains chapters on Minix and Minix source code listings).**

The First Article

Tanenbaum, A., *"A Unix Clone with Source Code for Operating Systems Courses,"* Operating Systems Review, Vol. 21, No. 1, January 1987, pp. 20-29.

A Useful Reference on Unix

Bach, M., *The Design of the Unix® Operating System* , Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987 **(Contains an extensive description of the internals of Unix which are in some parts of the system similar to Minix).**

Minix versus Unix V.7
(general design concepts)

Common design features

- **User Shell Interface**
- **Similar Multiprogramming Environment**
- **Similar Library Procedures**
- **Unix Commands and Utilities**
- **Device Independent I/O**
- **Similar File System Internal organization**
- **C compiler and combined assembler/loader**
- **Memory Management designed with hardware MemoryManagement in mind**

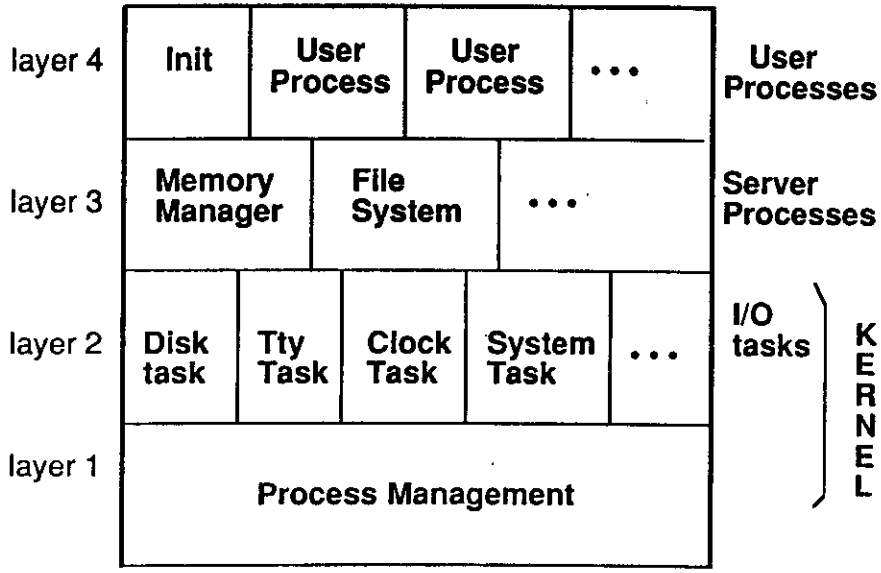
Minix versus Unix V.7
(general design concepts (suite))

Where Minix is different

- **Modular structure**
- **File system runs as a user program (in user mode).**
- **RPC based networking (using the AMOEBA protocol)**
- **Runnable on PC clones and other microcomputers.**
- **Memory manager runs in user mode**
- **Memory management is simple and segmented. (PC compatibility).**
  (while Unix v. 7 runs on machines with Virtual Memory, Paging and Swapping)
- **Special commands to overcome implementation differences (e.g.** chmem, dosread, doswrite, ...).
- **Internally all communication uses message passing mechanism (** " rendez-vous ").

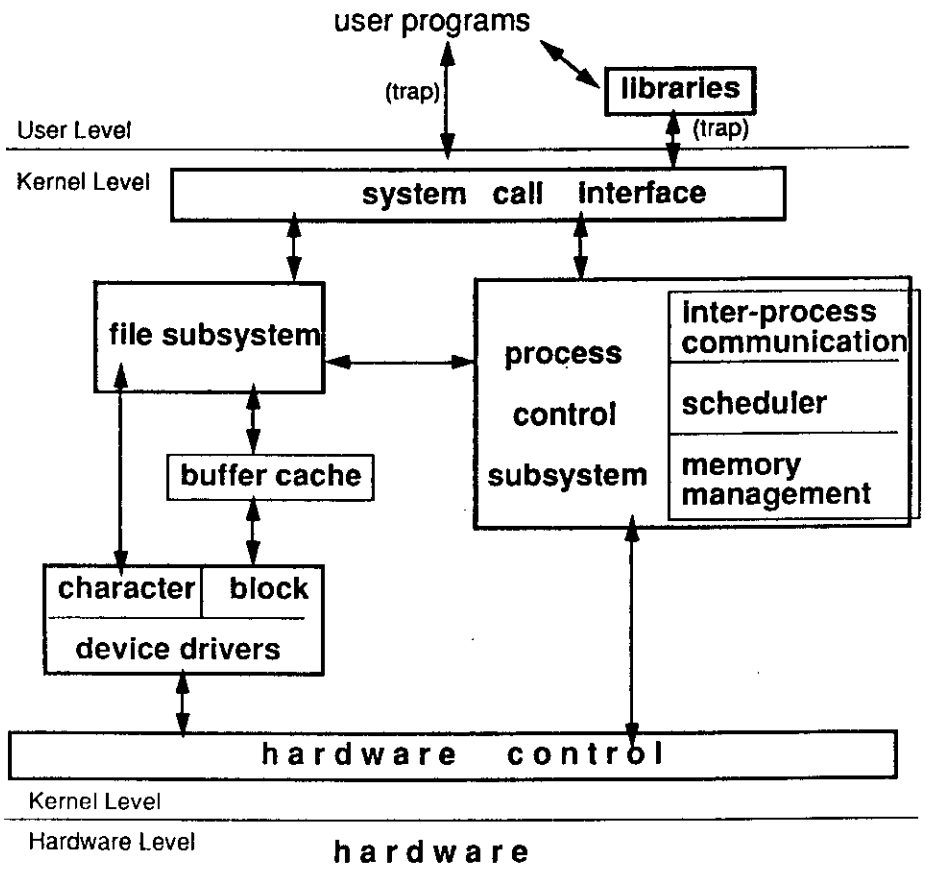
Inside Minix

Internal Structure of Minix : the 4 layers



Tasks in layer 1 and 2 are linked together forming the kernel. They share a common address space but each task has its own context stack and memory map and they all communicate by message passing.

UNIX Kernel Block Diagram



Process Management
(Process Creation)

- **PROCESS CREATION** (UNIX like):
  - **fork** - Create another process with a similar memory image to this one.

  The fork system call stages:

  1. Check to see if process table is full.
  2. Try to allocate memory for the child
  3. Copy the parent's image to the child's memory.
  4. Find a free process slot and copy parent's slot to it
  5. Enter child's memory map in process table.
  6. Choose a pid for the child.
  7. Tell kernel and file system about child.
  8. Report child's memory map to kernel.
  9. Send reply messages to parent and child.

In OS9 forking implies only duplication of the process' data space.

- **exec** - Replace the current process' memory image with a new one, including setting up a new stack.

Process Management
(Process Creation(suite))

The exec system call stages:

1. Check permissions. Is the file executable ?
2. Read the header to get the segment and total sizes
3. Fetch the arguments and environment from the caller
4. Release the old memory and allocate a new one.
5. Copy stack to new memory image.
6. Copy text and data segments to new memory image.
7. Check for and handle setuid, and setgid bits.
8. Fix up process table entry.
9. Tell kernel that process is now runnable.

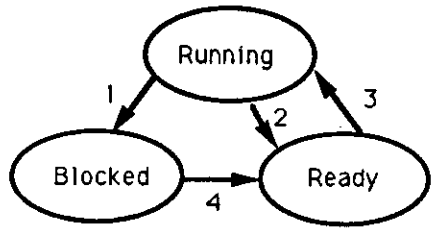
Usage of fork and exec:

```
if ( fork() != 0 ) {                /* fork off child process */
        wait( &status );            /* parent code */
} else {
        execve( command, parameters, 0 ); /* child code */
}
```


Process Scheduling

Process Status Diagram:

In Minix we may distinguish 3 states: **Running, Ready** or **Blocked**. A process gets blocked depending on it having to wait for input. The transitions between Ready and Runnable are caused by the process scheduler.



1. Process blocks (for input or due to system call).
2. Scheduler picks a different process from the runnable queue and preempts current process
3. Scheduler selects a process to run.
4. Process becomes runnable.

Process Scheduling
(suite)

Multilevel queueing system with 3 levels of priority:

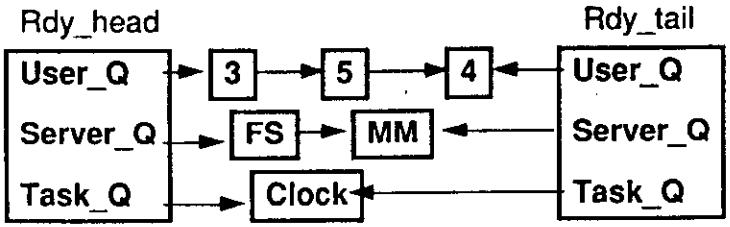
Highest priority: layers 1,2 ---> kernel

Middle priority: layer 3 ---> servers

Lowest priority: layer 4 --->user processes

Round-robin in each level

User processes have 100 msec time slices



Why Minix is NOT real-time

<u>Minix uses Non-preemptive scheduling</u>

- **Tasks, the memory manager and the file system are never pre-empted no matter how long they have been running. They should block after finishing their work.**

- **a Task (I/O task) running at the time of the interrupt will continue to run after processing is finished.**

Notice the different strategy of OS9:

It IS real-time, because it relies on preemptive scheduling, whereby a running process will be preempted as soon as a higher priority process becomes ready. The previously running process looses therefore the rest of its timeslice

Minix may be used in applications where time constraints are not very tight, provided these programs take into account these limitations.

Interprocess Communication

Inside Minix:

The "rendez-vous" principle.

A process doing a RECEIVE from a particular source or ANY indicates that it is ready to accept a message. If there is no message available for this process from that source, the message is delivered, else the process is blocked until the message is available.

Likewise a SEND results in a delivered message if the destination has previously issued a RECEIVE, else the sending process is blocked until it is delivered. SENDREC is merely a SEND followed by a RECEIVE from the same process.

These calls may only be used by the tasks or servers.

For the user: (similar to Unix and OS9).

- **pipes**
- **signals**

These are the only IPC mechanisms provided to the user.

If using Minix Networking (AMOEBA RPC):

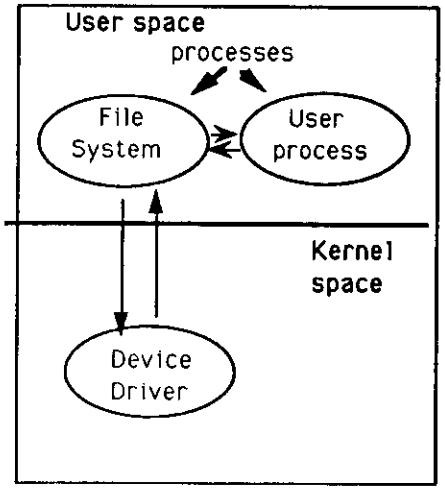
client/server model: getreq, putrep, trans

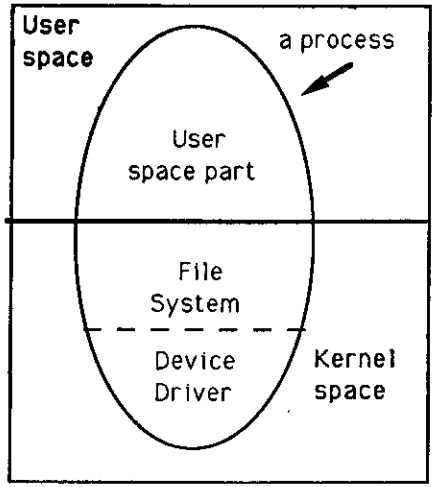
Minix versus Unix User-System Communication

Minix is highly modular and moderately efficient.

All the interactions between the servers and the parts of the Minix kernel are via the message mechanism.

Whereas in UNIX all processes have a user-space and a kernel-space part which communicate via a trap followed by state saving. Inside the Unix kernel communication is via procedure calls.





The Minix approach is more modular, has cleaner interfaces, and extends more easily to distributed systems. Some efficiency is lost because of number of messages exchanged in each system call. The Unix approach is more efficient.

I/O in Minix
(Main Concepts)

The 4 layers of I/O software in Minix:

1 - **Interrupt Handlers:**
Interrupt procedures running in kernel, that save the context of the running process and builld a message to be sent to the corresponding device driver.

2 - **Device Drivers:**
I/O tasks in the kernel with their own execution context.

Outline of the main procedure of a driver:

```
io_task()
{
    initialize();
    while(TRUE) {
        receive(ANY, &message);  /* wait request */
        decode_message();        /* who & what  */
        do_the_work();           /* do operation */
        send( caller, &message); /* reply to caller */
    }
}
```


I/O in Minix
(Main Concepts(suite))

The 4 layers of I/O software in Minix(suite):

3 - **Device Independent I/O Software:**
Contained in the Minix file system. Comprises: interfacing with device drivers, buffering, block allocation, protection and management of all the UNIX-like file structure.

4 - **User-level I/O Software:**
Unix-like libraries and utilities.
No daemons are provided in the standard configuration, but as e.g. spooler daemons are just user processes, so it is easy to add them as needed.

However. . .

• Adding a device-driver requires recompilation of the kernel..(Major difference to OS9).

• Devices are described by special files under /

I/O in Minix
(General Remarks)

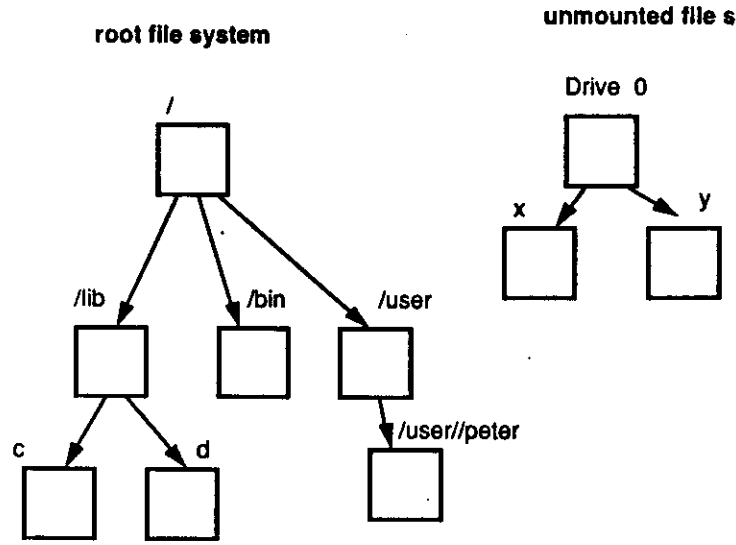
Deadlock avoidance:

- No algorithm is being used. However care is taken between the Memory Manager and the File System.
- Problem solved by having the File System only *replying* to the Memory Manager's *requests*, except when at start up it reports its size to MM.

RAM DISK:

- Like in OS9 is associated with a block device.
- Contains the root file system., so that removable media may be used, e.g. a floppy may be mounted and unmounted at will.
- Minix mounted file system support provides an integrated file system.

The Mount command



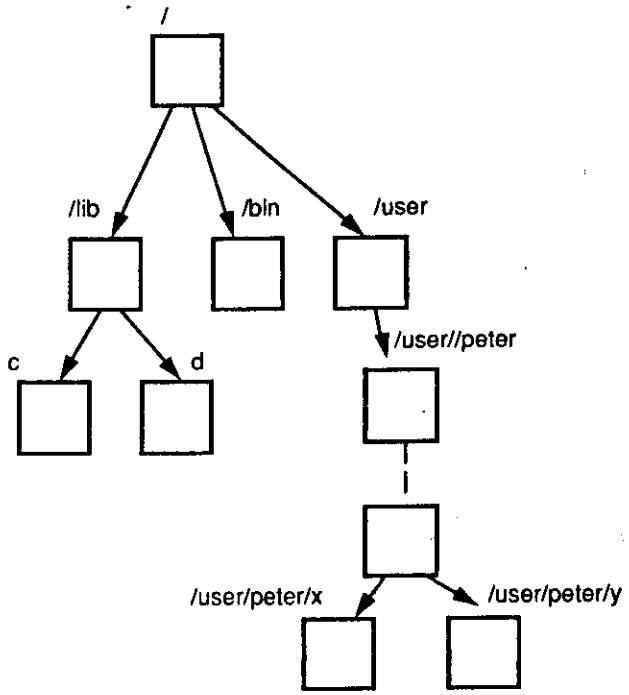
before mounting: We have 2 different file systems completely independent :

- the root file system
- another file system e.g. on a floppy disk

The mount system call provides a mean to merge both file system trees. This way the files on floppy may be accessed in a device independent way.

The Mount command (suite)

root file system with Drive 0 mounted



Now the files on floppy may be accessed in a device independent way without the need to specify the drive

MOUNT/UMOUNT (Warnings)

To mount a floppy with a user file system:

/etc/mount /dev/fd0 /user

Then work on the files you want to change and when you are finished, **before removing the floppy disk:**

/etc/umount /dev/fd1

- **The user may only remove the floppy after "ok".**
- **If a floppy disk is removed while it is mounted the system may hang, but it will continue to run once the floppy is re-inserted.**
- **If a floppy is removed while it is mounted and another one is inserted into its place both file systems will be seriously damaged**

CREATING A FILE SYSTEM ON A FLOPPY DISK

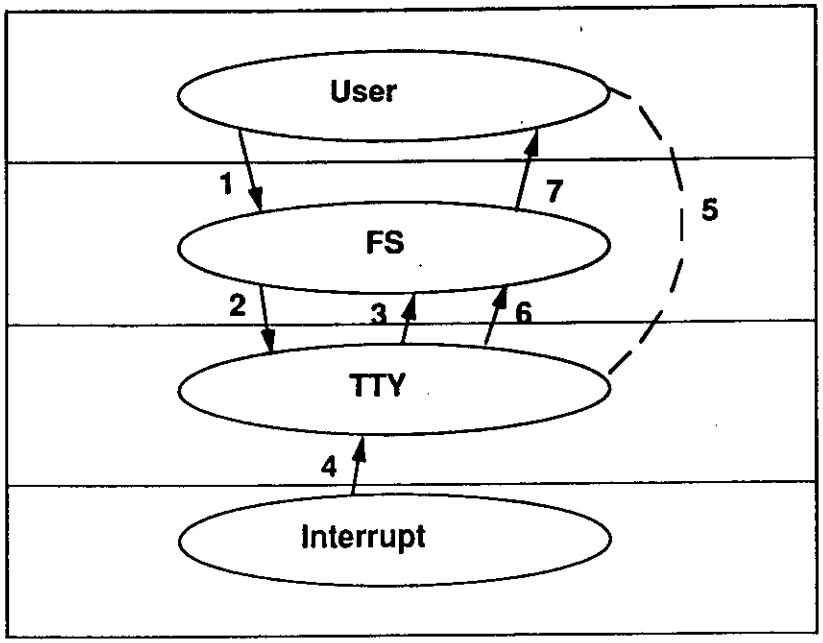
On a 360K floppy disk: mkfs /dev/fd0 360

Only after mkfs a disk may the user execute the mount of this file system on a certain e.g. empty directory.

Any changes to this directory will be as if done on the device. Before exiting umount must be executed.

Terminal Input
(A simplified I/O call example)

Read request from terminal when no characters are pending.
FS - File System, TTY - Terminal Task,
Interrupt - Interrupt routine.



The SYSTEM TASK

- **Being the File System (FS) and the Memory Manager user processes outside the Kernel they cannot access directly the Kernel data structures to supply necessary information.**
- **Solution: There is a Kernel task (the System Task) that communicates with the File System and the Memory Manager via the standard mechanism of message passing and which also has access to all the kernel tables.**

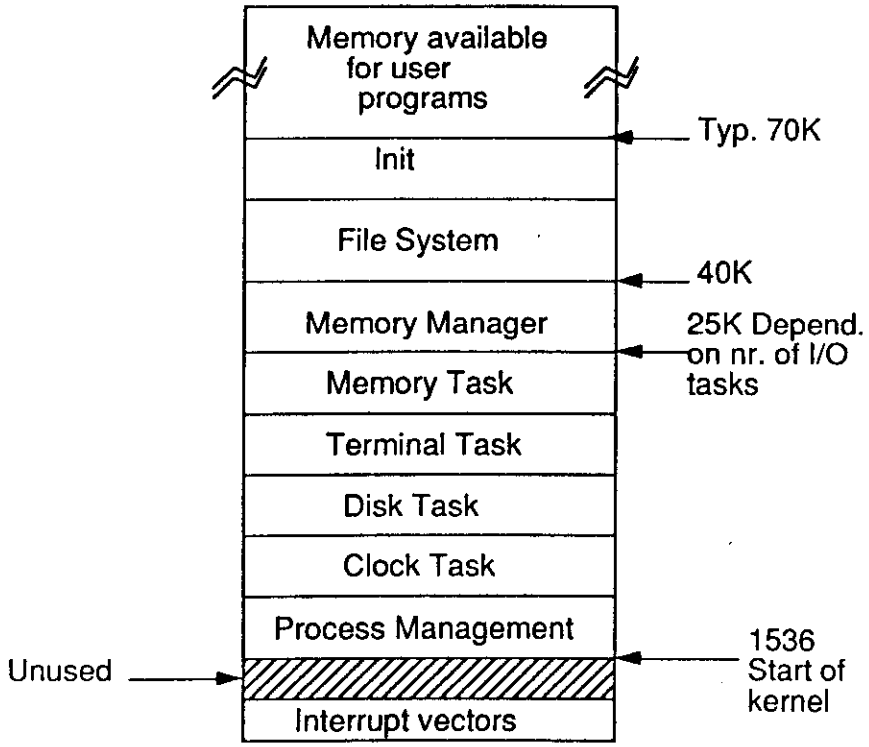
MESSAGE TYPES ACCEPTED BY THE SYSTEM TASK

Message type	From	Meaning
SYS_FORK	MM	A process has forked
SYS_NEWMAP	MM	Install memory map for new process
SYS_EXEC	MM	Set stack ptr. after EXEC call
SYS_XIT	MM	A process has exited
SYS_GETSP	MM	MM wants a process stack ptr.
SYS_TIMES	FS	FS wants a process execution times
SYS_ABORT	Both	Panic: Minix unable to continue
SYS_SIG	MM	Interrupt a process with a signal
SYS_COPY	Both	Copy data between processes

MEMORY MANAGEMENT (Introduction)

MINIX MEMORY MAP FOR THE IBM PC

(After Minix has been loaded from the disk into memory).



MEMORY MANAGEMENT (Main Concepts)

- **Memory Management**

  Influenced by the IBM PC Hardware Memory Management and 8088 lack of virtual memory support and support for stack overflow.

  Simple so that is easily portable.

- **Memory Management Function divided between:**

  Memory Manager Server (MM)

  Handles Minix System Calls involving memory management deciding which processes to load.

  System Task in Kernel

  Setting up of Memory Maps.

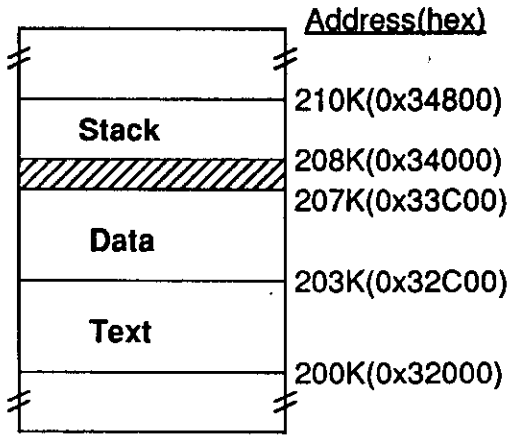
- **Due to the 8088 architecture each segment effectively must begin at an address that is multiple of 16 bytes. We define a click as a group of 16 bytes.**
- **MM manages the addresses in clicks.**
- **MM records memory allocation by having a copy of a part of the kernel's memory allocation tables.**

MEMORY MANAGEMENT
(Main Concepts(suite))

- **A process can be organized in memory in 2 ways:**

  Separated Instruction and Data Spaces

  Non-separated Instruction and Data Spaces



Non-Separate I & D

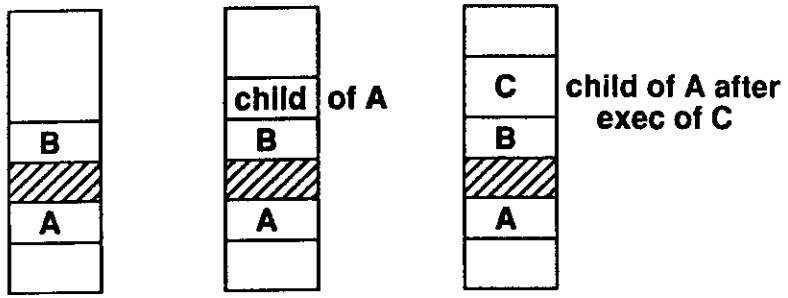
	Virtual	Physical	Length
Text	0	0x3200	0
Data	0	0x3200	0x1C0
Stack	0x200	0x3400	0x80

Separate I & D

Virtual	Physical	Length
0	0x3200	0xC0
0	0x32C0	0x100
0x140	0x3400	0x80

MEMORY MANAGEMENT
(Memory Allocation)

FORK x EXEC memory allocation implications



A and B are user procs. **A forks** **A's child execs**

- **Memory Allocation is done in click boundary from a pool of free memory using the first fit algorithm.**

  This algorithm leads to memory fragmentation.

  OS9 version on Rosy uses a similar scheme.

- **To overcome this implementation default and to allow for sizing of the dynamic allocation part of an executable program Minix provides chmem.**

- **chmem: is a command used to modify the excutable file header to provide for a larger or short dynamic allocation area (gap between data and stack components.)**

The Minix File System
(Main Concepts)

- Managed by a File Server (FS) outside the Kernel.
- Except for the interaction with the library procedures, and the kernel done in Minix via message passing, the rest of the internals of the Minix File System are almost identical to Unix.
- **Unix-like device independent interface:**
- **File System handles I/O device independent interface:**

  interfacing to drivers, block allocation, buffering.
- **Unix-like File System specifics: (*)**

  File types: data files, pipes, device files. (*)

  Unix-like File System structure:

  Boot block, Superblock, i-node structure, block cache

  User's view of a directory tree.

  File descriptors and corresponding indexation to i-nodes

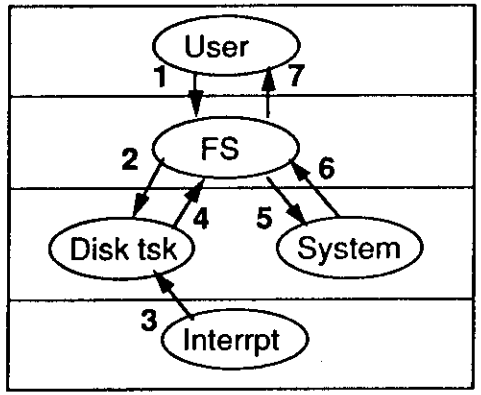
  File protection modes

  Creating links to files: soft and hard links.

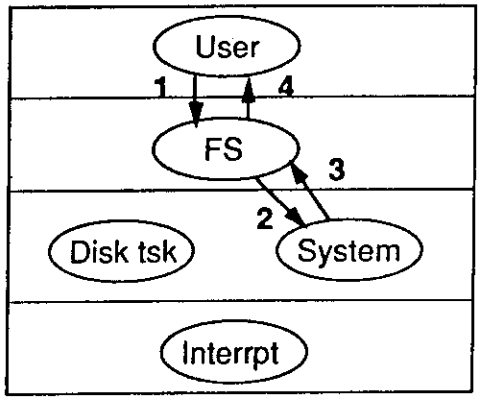
  Mounting and Unmounting a File System

(*) All these topics were extensively covered by the lectures on Unix by U. Reich.

Minix Message Passing
(Reading a block from disk)



- The WORST CASE: The block is on the disk.



- The BEST CASE: The block is in the cache.

Minix Start-Up Overview

- Computer turned on
- Hardware reads first sector of first track into memory and jumps to it.
- Bootstrap program executes loading the operating system into memory and starts it running.
- Kernel, MM, FS initialize themselves one after the other.
- Init is started
- Init reads /etc/ttys to get number of terminals installed.
- Init forks off one process per each terminal.
- Each of Init's children executes /bin/login and waits until someone logs in.
- After successful login, /bin/login executes the user's shell.
- Shell prompts user for command.
- Forks off new process per command and waits for child to terminate.
- Then the shell will prompt again for a command.
- Now the user is in a Unix Programming Environment with a Shell.
- It is his turn to do some work.
- Hope you enjoy and learn o.s.s with Minix.