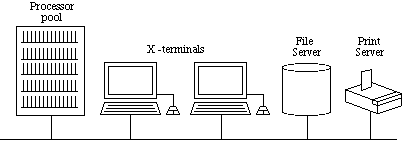
**CASE STUDY:AMOEBA**

#### Introduction to Amoeba

* Originated at a university in Holland, 1981
* Currently used in various EU countries
* Built from the ground up. UNIX emulation added later
* Goal was to build a transparent distributed operating system
* Resources, regardless of their location, are managed by the system, and the user is unaware of where processes are actually run

#### The Amoeba System Architecture

* Assumes that a large number of CPUsare available and that each CPU ha 10s of Mb of memory
* CPUs are organised into processor pools



* CPUs do not need to be of the same architecture (can mix SPARC, Motorola PowerPC, 680x0, Intel, Pentium, etc.)
* When a user types a command, system determines which CPU(s) to execute it on. CPUs can be timeshared.
* Terminals are X-terminals or PCs running X emulators
* The processor pool doesn't have to be composed of CPU boards enclosed in a cabinet, they can be on PCs, etc., in different rooms, countries,...
* Some servers (e.g., file servers) run on dedicated processors, because they need to be available all the time

#### The Amoeba Microkernel

* The Amoeba microkernel is used on all terminals (with an on-board processor), processors, and servers
* The microkernel

manages processes and threads

provides low-level memory management support

supports interprocess communication (point-to-point and group)

handles low-level I/O for the devices attached to the machine

#### The Amoeba Servers: Introduction

* OS functionality not provided by the microkernel is performed by Amoeba servers
* To use a server, the client calls a stub procedure which marshalls parameters, sends the message, and blocks until the result comes back

Server Basics

* Amoeba uses capabilities
* Every OS data structure is an object, managed by a server
* To perform an operation on an object, a client performs an RPC with the appropriate server, specifying the object, the operation to be performed and any parameters needed.
* The operation is transparent (client does not know where server is, nor how the operation is performed)
* Capabilites

To create an object the client performs an RPC with the server

Server creates the object and returns a capability

To use the object in the future, the client must present the correct capability

C:\Users\Dell\Desktop\Operating Systems  Lecture 22  Distributed Operating Systems  Case Study  Amoeba_files\os222.gif

The check field is used to protect the capability against forgery

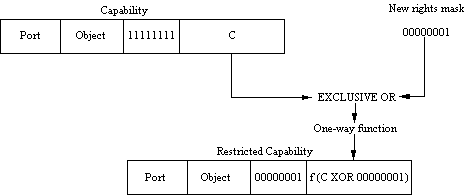
* Object protection

When an object is created, server generates random check field, which it stores both in the capability and in its own tables

The rights bits in the capability are set to on

The server sends the owner capability back to the client

Creating a capability with restricted rights



Client can send this new capability to another process

#### Process Management

* All processes are objects protected by capabilities
* Processes are managed at 3 levels

by process servers, part of the microkernel

by library procedures which act as interfaces

by the run server, which decides where to run the processes

* Process management uses process descriptors

Contains:

platform description

process' owner's capability

etc

#### Memory Management

* Designed with performance, simplicity and economics in mind
* Process occupies contiguous segments in memory
* All of a process is constantly in memory
* Process is never swapped out or paged

#### Communication

* Point-to-point (RPC) and Group

#### The Amoeba Servers

The File System

* Consists of the Bullet (File) Server, the Directory Server, and the Replication Server

The Bullet Server

* Designed to run on machines with large amounts of RAM and huge local disks
* Used for file storage
* Client process creates a file using the *create* call
* Bullet server returns a capability that can be used to *read* the file with
* Files are immutable, and file size is known at file creation time. Contiguous allocation policies used

The Directory Server

* Used for file naming
* Maps from ASCII names to capabilities
* Directories also protected by capabilities
* Directory server can be used to name ANY object, not just files and directories

The Replication Server

* Used for fault tolerence and performance
* Replication server creates copies of files, when it has time

Other Amoeba Servers

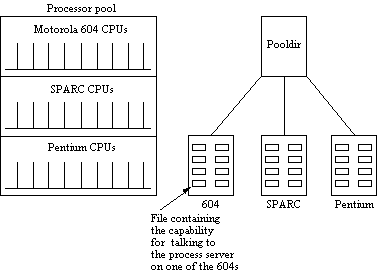
The Run Server

* When user types a command, two decisions have to be made

On which architecture should the process be run?

Which processor should be chosen?

* Run server manages the processor pools



* Uses processes process descriptor to identify appropriate target architecture
* Checks which of the available processors have sufficient memory to run the process
* Estimates which of the remaining processor has the most available compute power

The Boot Server

* Provides a degree of fault tolerance
* Ensures that servers are up and running
* If it discovers that a server has crashed, it attempts to restart it, otherwise selects another processor to provide the service
* Boot server can be replicated to guard against its own failure

**CASE STUDY:DOS Operating System**

**Introduction**

MS-DOS is a single-user operating system. It is designed to operate on machines using the Intel line of 8086 microprocessors. These processors include the 8088, 8086, 80286, 80386, 80486, and the new Pentium (80586). PCs that are called 386s or 486s; are based on their processor names, 80386 and 80486. The 80586 was so different from the previous versions that it was given the name Pentium as a distinction. MS-DOS cannot support a large network of users. Because PCs and MS-DOS became so popular, many network operating systems were designed in the mid-1980s to network MS-DOS machines together; working around the operating system limitations through software. MSDOS is the preferred operating system for most of the Intel processor (currently Pentium) based PCs of the world. MS-DOS does not break out into neat

compartments as easily as some operating systems do. This is partially due to its

simplicity -- no multi-user or multitasking ability. It is also partly due to the way MS-DOS has evolved over the years.

So in many respects DOS was a primitive OS. It was based on previous systems,

of course, and echoes of UNIX and CP/M can be seen in it. But if you read Stephenson's article above, you will realize that it had hidden power, as well, because you could interact more directly with the components of the computer than you can with more modern operating systems. It is this power that makes it valuable to know DOS today. The majority of computer users today use graphical systems, but users of Windows, for instance, have the ability to interact with the computer through DOS (or in the case of NT, with a DOS-analogue) to do things which are difficult or impossible to accomplish through the graphical interface. For this reason, familiarity with DOS is still considered essential for anyone supporting Intel-based machines running Windows.

**Kernel**

MS-DOS uses two hidden files at boot time. These are io.sys and msdos.sys. For all practical purposes, these files, in conjunction with the firmware BIOS (Basic Input Output Services) built into every PC, make up the MS-DOS kernel (basic operating system). They load at the time of startup and allow the command processor to run. These files are not rebuildable or alterable. Software to run printers or CD-ROMs (device drivers) can be installed in MS-DOS but the kernel cannot be changed.

**11.2.2 COMMAND.COM**

This file starts the command processor. The command processor uses the commands you enter at the C:> prompt. When you run application programs and then return to MS-DOS, the system must be able to find COMMAND.COM and reload it back into memory (RAM). The command processor also supports a command language knows as the DOS Batch language. The Batch language files have a .BAT extension and are considered executable by the command processor. The Batch language is not as powerful as some command languages but does support conditional statements and variables (if time = next\_day then...). In the early years of PCs, the Batch language was used for many tasks. Now, low cost utilities often provide many times the functions in addition to direct support. However, batch files are still very common and very useful.

**MS-DOS Services**

Because MS-DOS is not able to execute more than one task at a time, it cannot have

multiple jobs working in the background. However, MS-DOS does use services that allow programs to interact with the computer. These services include basic input/output services (BIOS), print services, and file services. MSDOS services use what is known as an interrupt to the microprocessor. This interrupt concept allows terminate-and-stay-resident (TSR) programs to function. TSRs can sit quietly in the background and appear when the user requests them. TSRs provide MS-DOS with the appearance of running more than one task at a time.

**Startup Files**

MS-DOS uses two changeable startup files. They are CONFIG.SYS and

AUTOEXEC.BAT. The CONFIG.SYS has the task of loading installable device drivers and other system parameters which must run at boot time. The device drivers might be for a CD-ROM drive or a sound board. The AUTOEXEC.BAT starts applications automatically for the user, handles logins for network software, and places information in the PC’s environment. In some cases, there is not a clear-cut line that exists between which commands go into the CONFIG.SYS and which go into the AUTOEXEC.BAT. In general, the CONFIG.SYS holds only device drivers, special commands provided with MS-DOS, and memory managers.

**Managing Input and Output**

MS-DOS normally reads input from the standard input, and normally writes output to

the standard output. The standard input or output may be redirected to a file or a

device. MS-DOS treats devices as files. The output of a command or program can

be piped to another command or program. Following DOS commands are relevant

for I/O manipulation: CLS, CTTY, FIND, GRAFTABL, GRAPHICS, MODE, MORE,

PRINT, SORT, and TYPE.

**Managing Files**

Data is organized into files whether that data is in memory or on disk. Files may be data files or executable files. A file is simply a string of bytes; no record structure is assumed. Applications impose their own record structure on the string of bytes MSDOS requires only a pointer to the data buffer, and a count of the number of bytes to be read or written in order to do I/O. Executable files must be in either the .COM format or the .EXE format. Object modules are maintained in the Intel Corporation object-record format.

Files may be accessed with FCB calls or file handle calls. File handle calls are designed to work with a hierarchical file system. File handle calls are preferable, but FCB calls are provided for compatibility with previous versions of DOS. File handle calls support record locking and sharing. Users interested in writing programs that will be compatible with future versions of MS-DOS should use handle calls rather than the FCB calls.

A file handle is a 16-bit integer (usually 0 to 20) created

by MS-DOS and returned to a program that creates or opens a

file or device. A file may be opened with a handle by using a

pathname and attribute. The program saves the handle and uses

it to specify the file in future operations on the file. An

FCB-like data structure is built by MS-DOS for the file, but

this is strictly controlled by the operating system. The file

handle function calls are as follows:

(a) Create file

(b) Open file

(c) Close file

(d) Read from file or device

(e) Write to file or device

(f) Move file pointer

(g) Duplicate file handle (creates a new handle that refers to the same file as an

existing handle)

(h) Match file handle (cause one handle to refer to the same file as another)

(i) Create temporary file

(j) Create file if name is unique (fails if a file with the same name already exists)

MS-DOS maintains a table that relates handles to files or

devices. Eight handles are normally available to programs, but

this can be increased to 20 via the CONFIG.SYS file. Five of

the handles are pre-assigned to standard devices as follows:

Handle Standard Device Device Name Description

0 Standard input device CON reads char from the keyboard

1 Standard output device CON writes characters to the VDU

2 Standard error device CON writes characters to the VDU

3 Standard auxiliary device AUX controls serial port I/O

4 Standard printer device PRN c ontrols parallel port output

**Memory Management**

Memory is organized as follows in MS-DOS (from low memory to high memory locations):

i. Interrupt vector table

ii. Optional extra space (used by IBM for ROM data area

iii. IO.SYS

iv. MSDOS.SYS

v. Buffers, control areas, and installed device drivers

vi. Resident part of COMMAND.COM

vii. External commands or utilities (.COM and .EXE files are loaded here)

viii. User stack for .COM Files (256 bytes)

ix. Transient part of COMMAND.COM

The interrupt vector table contains the addresses of the interrupt handler routines. IO.SYS is the basic input/output

system; it is the MS-DOS/hardware interface. MSDOS.SYS contains most of the interrupt handlers and function requests. The resident part of COMMAND.COM contains certain interrupt handlers and the code that reloads the transient part of COMMAND.COM as needed; the transient part of COMMAND.COM includes the batch processor, the internal commands, and the command processor.

MS-DOS begins the user's program segment in the lowest address free memory. The program segment prefix (PSP) occupies the first 256 bytes of the program segment area. The PSP points to various memory locations the program requires as it executes. MS-DOS creates a memory control block at the start of each memory area it allocates. This data structure specifies the following:

i. the size of the area

ii. the program name (if a program owns the area)

iii. a pointer to the next allocated area of memory

MS-DOS may allocate a new memory block to a program, free a memory block, or change the size of an allocated memory block. If a program tries to allocate a memory block of a certain size, MS-DOS searches for an appropriate block. If such a block is found, it is modified to belong to the requesting process. If the block is too large, MS-DOS parcels it into an allocated block and a new free block. When a block of memory is released by program, MS-DOS changes the block to indicate that it is available. When a program reduces the amount of memory it needs, MS-DOS creates a new memory control block for the memory being freed. The first memory block of a program always begins with program segment prefix. Normally when a program terminates, its memory is released. The program can retain its memory by issuing function 31, TERMINATE BUT STAY RESIDENT.