Desarrollo de Aplicaciones en Red

José Rafael Rojano Cáceres
http://www.uv.mx/rrojano

What we saw

- **Distributed system: Collection of independent computers that for the user works like if it where one CPU.**
- **Distributed operating system: Is the actor in charge to achieve the past.**
- **Network operating systems: is the actual result, wait to to became in a true DOS. Each machine have their kernel.**

1. OS Design

- **The design of OS can be saw from three points of view:**
  - ***What services are offered***
  - ***Interface available for users and programmers***
  - ***Component and interconnections***

1.1 Operating System Service (1)

- **The number and type of service differs between each OS. The basic interface offered are:**
  - ***User interface***
    - *CLI, GUI, and batch interface*
  - ***Program execution***
  - ***I/O operations***
    - *ports or device*
  - ***File-system manipulation***
  - ***Communications***
    - *Shared memory or message passing*
  - ***Error detection***

1.1 Operating System Service (2)

- **Secondary services can be found**
  - ***Resource allocation***
    - *CPU scheduling routines according to the complexity of the job*
  - ***Accounting***
  - ***Protection and security***

2. Service description

2.1 User interface (1)

- As we saw, CLI a GUI approach exists
- CLI are operated by a command interpreter
- The command interpreter is called shell
- Two approach for building a interpreter
  - *by itself execute programs (size of interpreter)*
  - *Implements command as a system programs (to load and execute programs)*

7

2.1 User interface (2)

- GUI started in the early of 70s at Xerox
- Widespread by the Macintosh in the 80s
- Several GUIs does exist
  - *Mac OS*
  - *Windows*
  - *CDE and X-window*
  - *KDE, GNome*

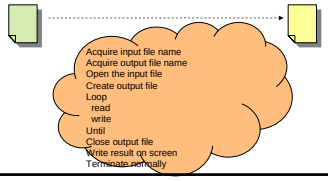
8

3. System calls

9

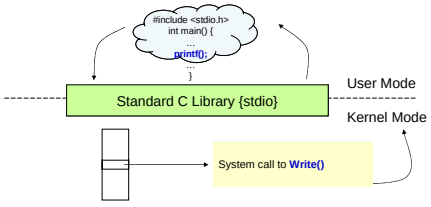
3.1 System calls

- It provides an interface for calling services in an OS
- *Imagine how many system calls does exist in order to copy from one file to another*
- System programmers usually consider this aspect to provide an API (*application programming interface*) for the developer



10

- For APIs use does exist different ones for example:
  - *Win32*
  - *Java*
  - *Posix API for posix systems includes Linux, Mac, Unix*
- Also it's important to say that many times the programmer use API to do system calls in stead of invoke system calls, this help to the portability



11

3.2 Type of System Calls

- Five general group are found:
  - *Process control*
  - *File manipulation*
  - *Device manipulation*
  - *Information maintenance*
  - *Communications*

12

3.2.1 Process control

- **Process control refers to the mechanism that OS use for handling the termination of a program.**
- **It can be normal (end) or abnormal (abort).**
- **In such case of error, a trap is issue causing dump file.**
- **In the next slide you will see a resume of type of system call**

- **Process control**
  - ***End, abort***
  - ***Load, execute***
  - ***Create or terminate process***
  - ***Get or set process attribute***
  - ***Wait***
  - ***Allocate memory***
- **File management**
  - ***Create, delete, Open, close, read, write***
- **Device management**
  - ***Request, release a device. Read or write***
- **Information maintenance**
  - ***Get or set, time, system data or process properties***
- **Communications**
  - ***Create, delete connection, send or receive message***

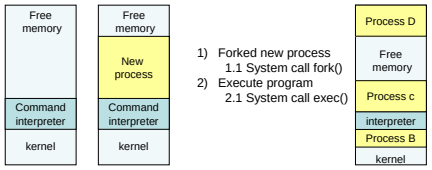
Example of getting process properties

- **/** Identifica el id del proceso hijo y padre */**

```
1.  #include <sys/types.h>
2.  #include <stdio.h>
3.  #include <unistd.h>
4.  void main(void)
5.  {
6.     pid_t id_proceso;
7.     pid_t id_padre;
8.     id_proceso = getpid();
9.     id_padre = getppid();
10.    printf("Identificador de proceso: %d\n", id_proceso);
11.    printf("Identificador del proceso padre: %d\n",
    id_padre);
12. }
```


3.2.1.1 Multitasking environment

- **In multitasking environment in stead of single-tasking child process can be executed. Mechanism are different, Ex.:**



a) b)
DOS environment Unix environment

Example of create process

```
1.  #include <sys/types.h>
2.  #include <stdio.h>
3.  #include <unistd.h>
4.  void main(void)
5.  {
6.     pid_t pid;
7.     int status;
8.     pid = fork();
9.     switch(pid)
10.    {
11.       case -1:   /* error del fork() */
12.          perror("Error el el proceso fork");
13.          break;
14.       case 0:    /* proceso hijo */
15.          execlp("ls","ls","-l",NULL);
16.          perror("Error en el proceso de creación exec");
17.          break;
18.       default:   /* padre */
19.          printf("Yo soy el proceso padre\n");
20.    }
21. }
```


3.2.2 File management

- **The design of File System has many aspect, however we can identify some common system calls in this aspect:**
  - ***Basic operations over files (O,C, R, W, S)***
  - ***Get or set attributes to files and directories***

3.2.3 Device management

- One a process is executed in need access to different resource, the resource controlled by the system can be seen as device
- Device can be hardware or software
  - *Disk, printers*
  - *Virtual devices*
- So, in this category is necessary to consider the administration for accessing and releasing devices

19

3.2.4 Information maintenance

- In this aspect many systems provide functionality for date and time.
- Also the system provides mechanism to identify process, as we saw in the example on page 15.

20

3.2.5 Communication

- As we said there are two ways for
  - *Shared memory*
  - *Message passing*

21

3.2.5.1 Shared memory

- *To use this model the process employ system calls to access shared memory owned by other process*
- *Shared memory requires that process removes or grant access level*
- *It is necessary for this model that programs provide mechanism for concurrency*
- *There is a variant to this model, thread, where shared memory is the default mechanism*

22

3.2.5.2 Message passing

- In this model through message are send between process
- Communication must be establish in advanced
- Name of the other process and communicator (PC) must be known
- Identify this properties through system calls are sent to the correspondent system calls for processing

23

3.3 System Programs

- *It's another aspect to consider in the design of an OS*
- *Many system programs are just interface of the system calls*
- *They are divided in the next categories*
  - File management
  - Status information
  - File modification
  - Programming language support
  - Programming loading and execution
  - communications

Programación de sistemas

24

4. Design & implementation in OS

25

General aspects

- Define goals and specification at highest level: considering hardware
- But, beyond this consideration basically we have two approach
  - *User goals*
  - *System goals*
- In general we found the rules of design in software engineering

26

5. Operating System Structure

27

5.1 General aspects

- The approach to design the structure of an OS is in general based in the partition task this is because of complexity

28

5.2 Simple structure

- *Many commercial OS does not have defined structures, because are small like Msdos*
- *In Msdos interfaces and levels are not well separated for that applications can access basic functionalities of I/O*
- *Another example is the early Unix which was separated in kernel and system programs, later the kernel was layered in services (see fig 5.2)*

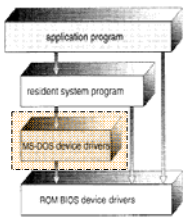


Fig. 5.1 MS-DOS layer structure.

29

5.3 Layered structure (1)

- *With proper hardware support, OS can be broken into smaller and efficient pieces.*
- *In the figure 5.2 and 5.3 you can see an approach where OS is broken in layers or levels. The bottom (0) is hardware and upper level (N) are applications*
- *The main advantage in this approach is simplicity and debugging*
- *Each layer is implemented with those functions provides for the lower layer*
- *A layer does not need to know how are operations implemented, only need to know what it does*

30

5.3 Layered structure (2)

- *Also layered approach can be less efficient that others types*
- *For example a system call can take so much time going from upper to lower considering that it may needs execute I/O operations, the system call can be longer that in a non layered system.*

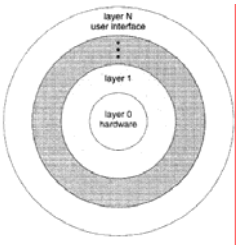


Fig. 5.2 A layered operating system.

Unix systems structure

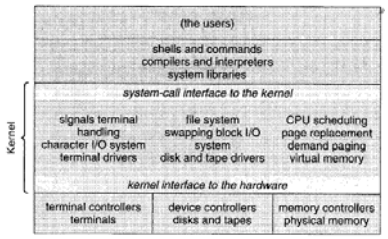


Fig. 5.3 UNIX system structure.

5.4 Microkernel

- *As was said the expansion of kernel in UNIX become complex, in 80s, Carnegie Mellon University develop Mach system with an approach in microkernel.*
- *This method consist in remove all nonessential components from the kernel implementing them as system or user-level programs.*
- *The main function is to provide an interface between user and services through message passing.*
- *Modularity and easily modifications without affect the kernel*
- *There is also more security because programs are executed with user privileges*
- *Ex.: Tru64 unix, QNX*

5.5 Modules

- *Another approach is the construction of OS as modules*
- *This approach is based in the object oriented paradigm*
- *Here the kernel has some core components and dynamically links to services*
- *Examples of this are Solaris, Linux and MacOS*

A core kernel structure

Fig. 5.4 Solaris loadable modules.

- *Message passing does not exist*
- *MacOs knows as Darwin as hybrid: layers and microkernel*

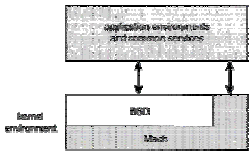
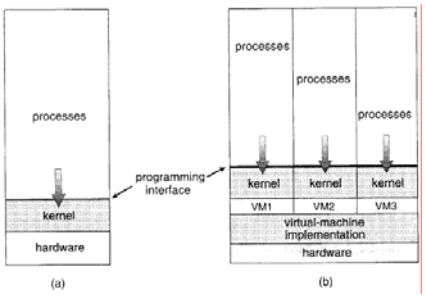


Fig. 5.5 The Mac OS X structure.

Virtual Machines

- *A virtual machine is the conclusion of a layered system.*
- *The idea in a VM is the abstraction of hardware into several different execution environments, creating the idea of a own computer.*
  - Virtual memory
  - Scheduling

Model of VM



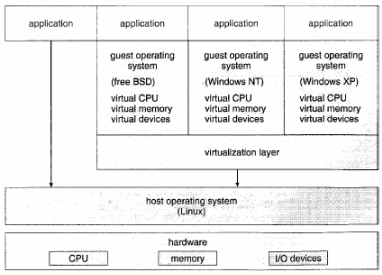
37

VMware example

- ***Vmware abstract Intel x86 hardware into isolated VM.***
- ***It let run different guest OS***

38

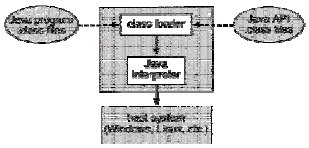
VMware architecture



39

Java VM example

- ***Java defines an architectural-neutral <u>byte code</u>***
- ***Two components for java architecture***



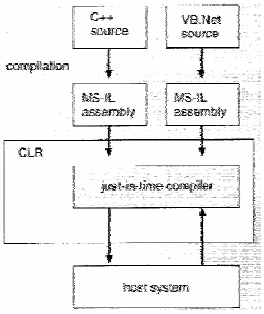
40

.Net platform example

- ***Also program are targeted to run on platform instead of specific architecture***
- ***The core of the architecture is the CLR (Common Language Runtime)***
  - **Which run intermediate code representation call MS-IL (Microsoft Intermediate Language)**

41

.Net architecture



42

Reference

- **Silberschatz et Al, Operating Systems concepts 7th. Wiley.**

43