Operating System Structure

1. Monolithic systems
2. Layered systems
3. Microkernels
4. Client- server systems
5. Virtual machines
6. Exokernels

Monolithic Systems

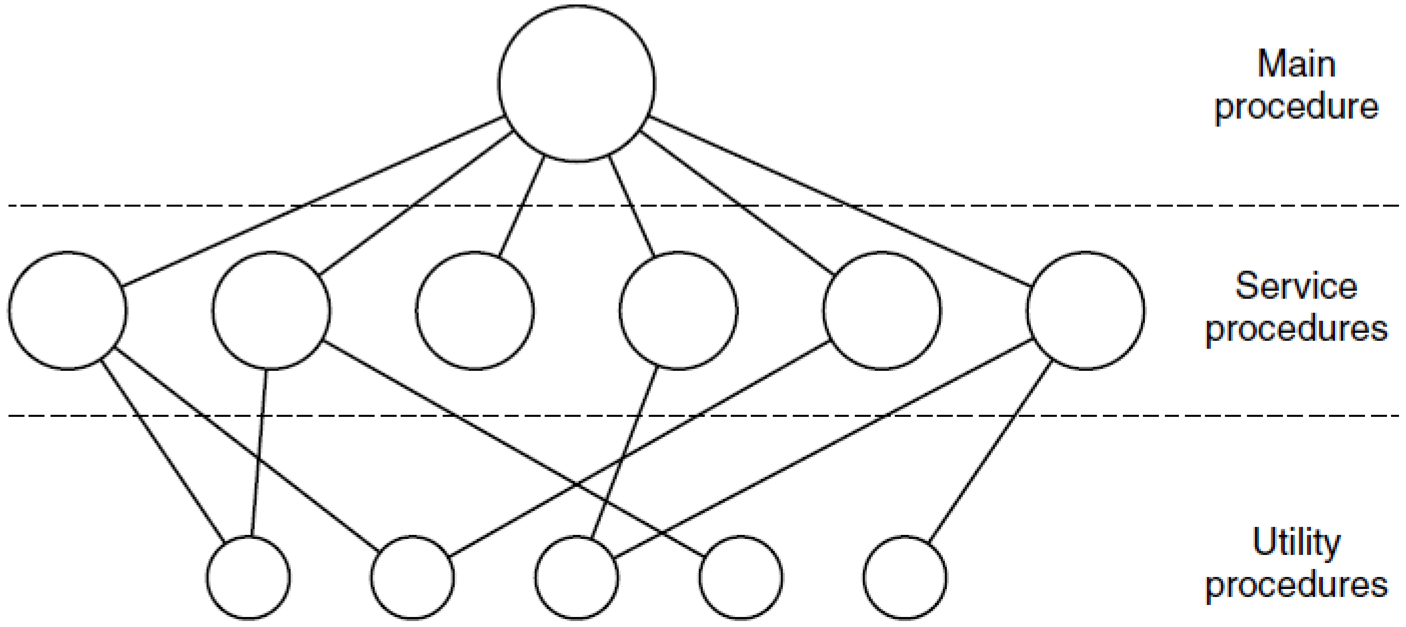
The operating system operates as a single program in kernel mode, organized as a collection of procedures combined into a single executable.

Besides the core OS, many systems support loadable extensions (like device drivers or file systems) that are loaded on demand, such as shared libraries in Linux and DLLs in Windows.

The basic structure of an OS consists of:

1 - A main program that invokes service procedures.
2 - A set of service procedures handling system calls.
3 - Utility procedures that support the service procedures.

Monolithic Systems



A simple structuring model for a monolithic system.

Monolithic Systems

1. Main Program

- This is like the "boss" of the system.
- It decides when and how to use different tools (services).
- For example, when you open a file, the main program figures out what needs to be done.

2. Service Procedures

- These are like tools in the toolbox.
- Each one does a specific job, like reading a file, writing to disk, or managing memory.
- When a program asks the OS to do something (like a system call), these service procedures handle it.

3. Utility Procedures

- These are helper tools.
- They don't interact with users directly but support the service procedures.
- For example, they might help with copying data or managing low-level details.

Monolithic Systems

In a monolithic system, all OS components (process management, file system, drivers, memory management) operate within a single address space and share the same privilege level.

This design offers high efficiency due to minimal communication overhead but comes with complexity, making debugging and maintenance difficult.

Stability is a concern, as an error in one component can crash the entire system.

Additionally, the system lacks modularity, complicating the process of modifying or replacing individual components.

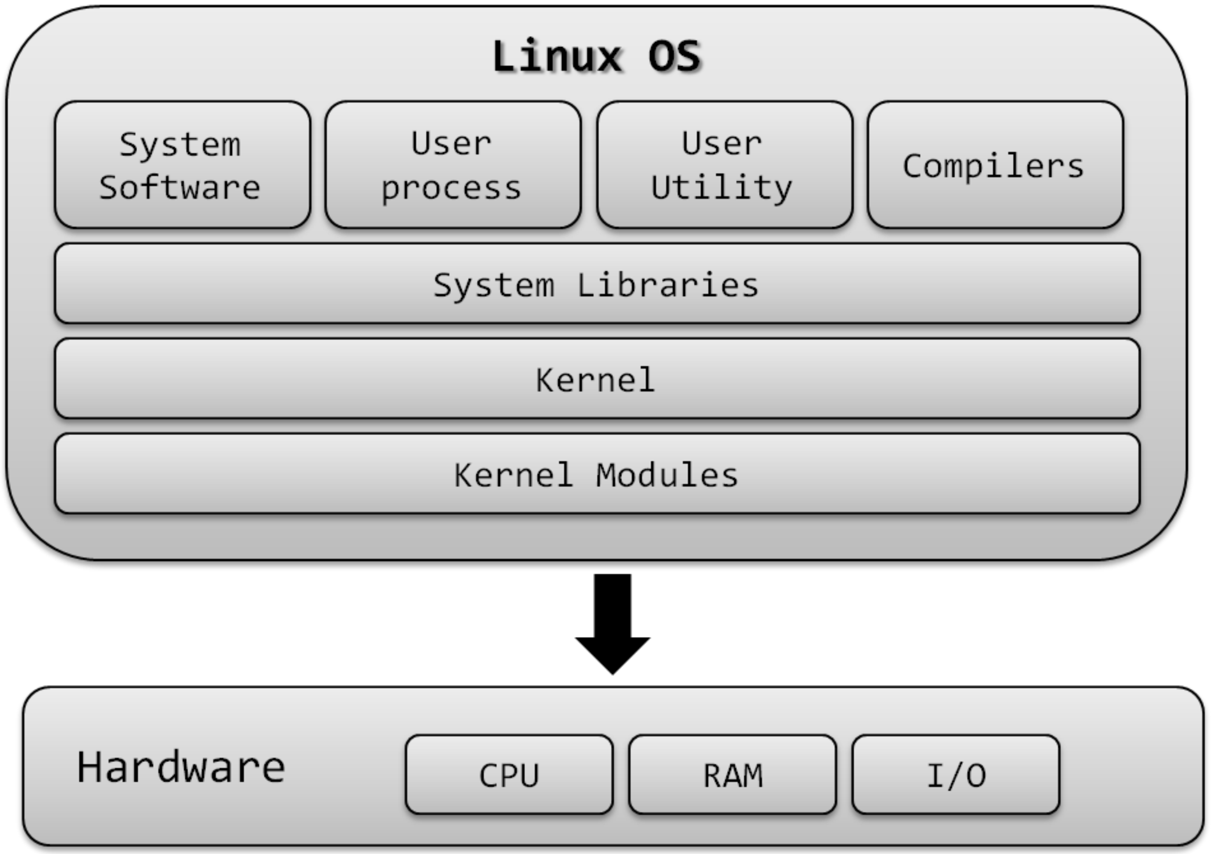
Monolithic Systems

Examples :

Linux, Microsoft Windows, etc.

Monolithic Systems

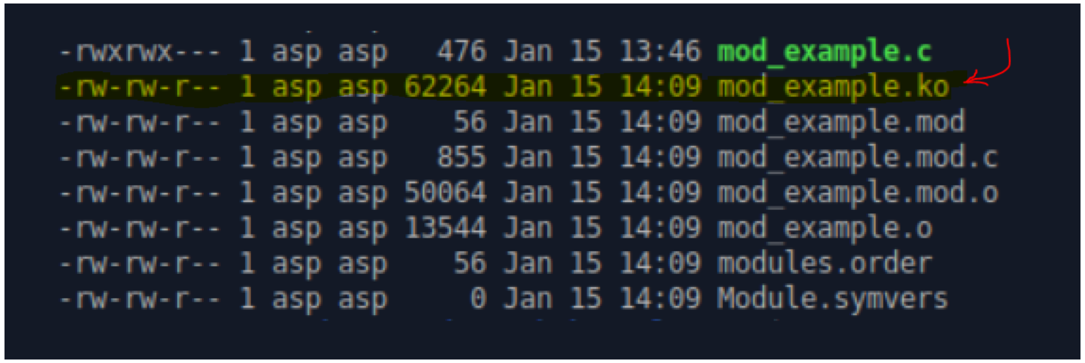
Linux Operating System



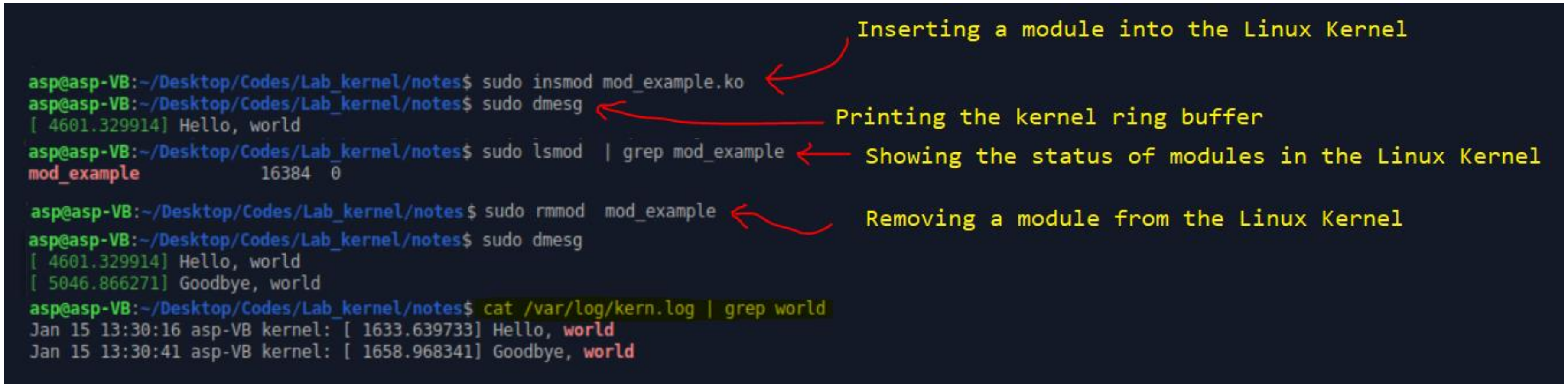
Monolithic Systems

```c
#include <linux/module.h>    /* Needed by all modules */
#include <linux/kernel.h>    /* Needed for KERN_INFO */
#include <linux/init.h>      /* Needed for the macros */

MODULE_LICENSE("GPL");
MODULE_AUTHOR("COMP-4735");
MODULE_DESCRIPTION("HELLO WORLD");


static int __init hello_init(void){
    printk(KERN_INFO "Hello, world \n");
    return 0;
}

static void __exit hello_exit(void) {
    printk(KERN_INFO "Goodbye, world\n");
}



module_init(hello_init);
module_exit(hello_exit);
```

Monolithic Systems



```
-rwxrwx--- 1 asp asp   476 Jan 15 13:46 mod_example.c
-rw-rw-r-- 1 asp asp 62264 Jan 15 14:09 mod_example.ko
-rw-rw-r-- 1 asp asp    56 Jan 15 14:09 mod_example.mod
-rw-rw-r-- 1 asp asp   855 Jan 15 14:09 mod_example.mod.c
-rw-rw-r-- 1 asp asp 50064 Jan 15 14:09 mod_example.mod.o
-rw-rw-r-- 1 asp asp 13544 Jan 15 14:09 mod_example.o
-rw-rw-r-- 1 asp asp    56 Jan 15 14:09 modules.order
-rw-rw-r-- 1 asp asp     0 Jan 15 14:09 Module.symvers
```



```
asp@asp-VB:~/Desktop/Codes/Lab_kernel/notes$ sudo insmod mod_example.ko
asp@asp-VB:~/Desktop/Codes/Lab_kernel/notes$ sudo dmesg
[ 4601.329914] Hello, world
asp@asp-VB:~/Desktop/Codes/Lab_kernel/notes$ sudo lsmod  | grep mod_example
mod_example            16384  0

asp@asp-VB:~/Desktop/Codes/Lab_kernel/notes$ sudo rmmod  mod_example
asp@asp-VB:~/Desktop/Codes/Lab_kernel/notes$ sudo dmesg
[ 4601.329914] Hello, world
[ 5046.866271] Goodbye, world
asp@asp-VB:~/Desktop/Codes/Lab_kernel/notes$ cat /var/log/kern.log | grep world
Jan 15 13:30:16 asp-VB kernel: [ 1633.639733] Hello, world
Jan 15 13:30:41 asp-VB kernel: [ 1658.968341] Goodbye, world
```

Inserting a module into the Linux Kernel
Printing the kernel ring buffer
Showing the status of modules in the Linux Kernel
Removing a module from the Linux Kernel

Layered Systems

The operating system is organized as a hierarchy of layers, each one constructed upon the one below it, e.g. the THE system built at the Technische Hogeschool Eindhoven in the Netherlands by E. W. Dijkstra (1968) and his students.

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Structure of the THE operating system.

Layered Systems

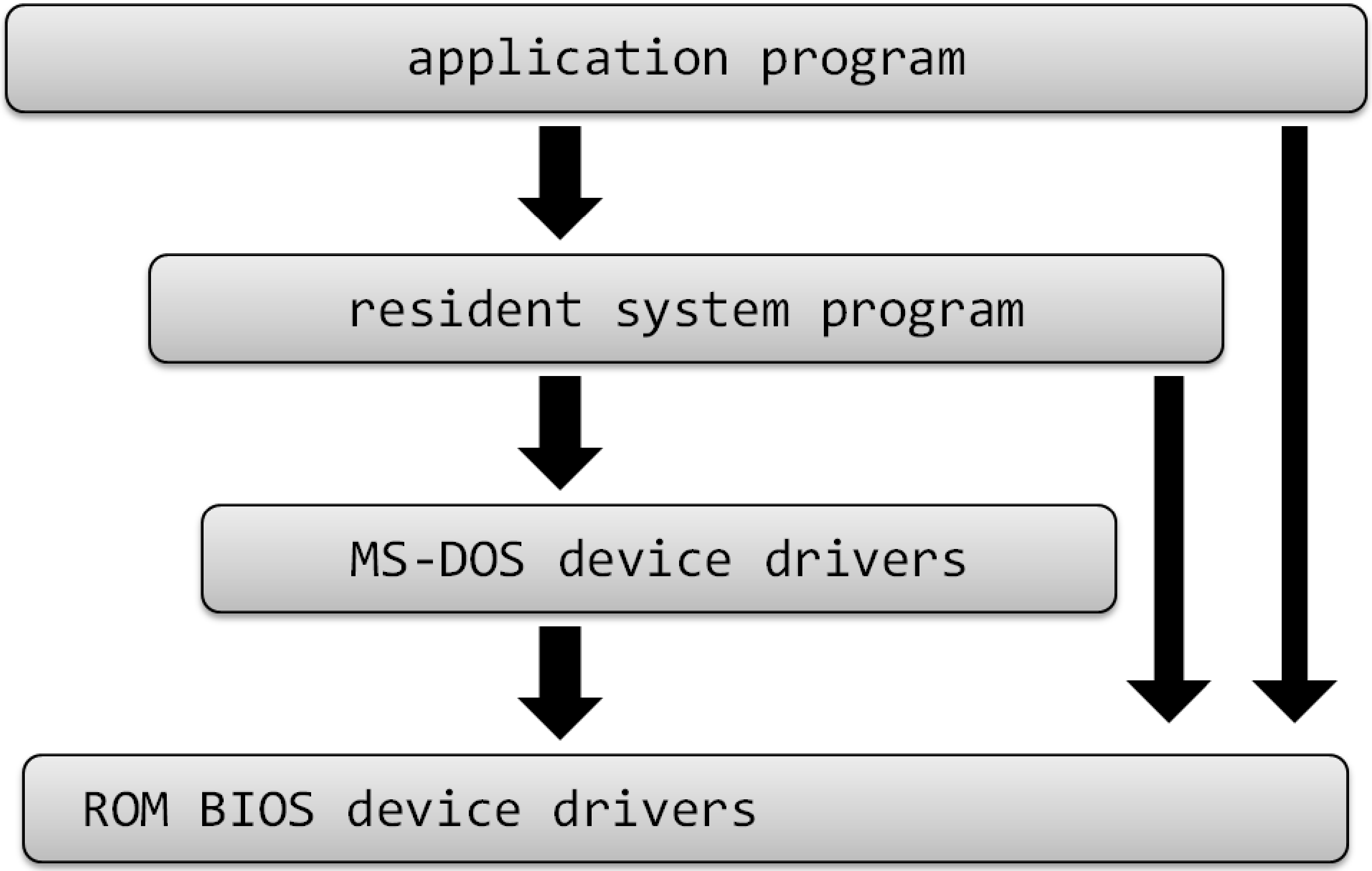
A layered system OS organizes functions into hierarchical layers, where each layer relies only on the one below it.

Advantages:
- Simplifies design and debugging by isolating functions.
- Enhances modularity and maintainability.

Disadvantages:
- Can introduce overhead due to strict layer interactions.
- Less efficient than monolithic systems, as lower layers may slow down upper ones.

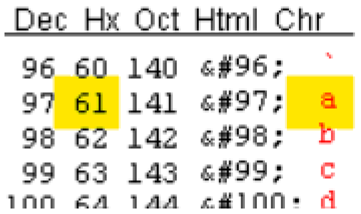
Layered Systems

MS - DOS

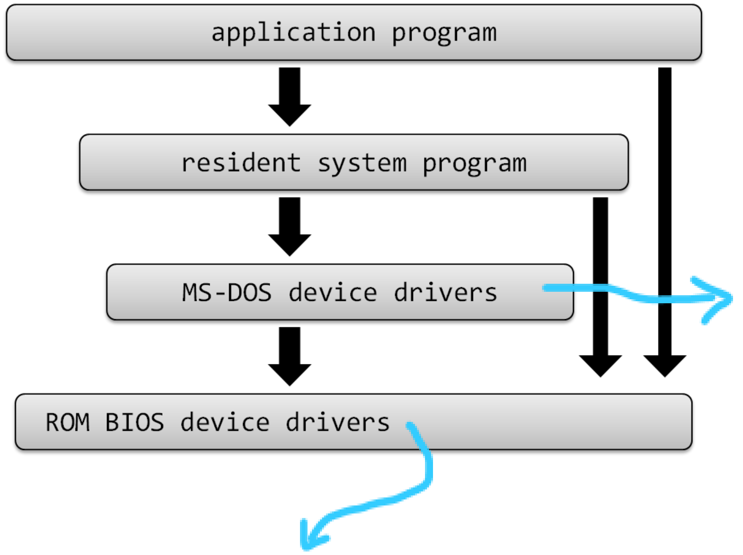


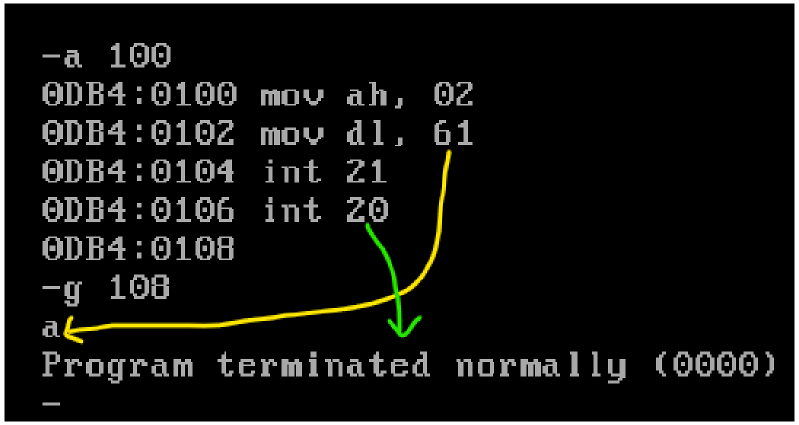
Layered Systems

MS - DOS

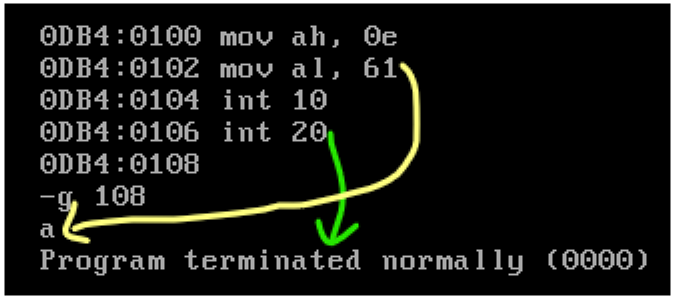


Dec	Hx	Oct	Html	Chr
96	60	140	&#96;	`
97	61	141	&#97;	a
98	62	142	&#98;	b
99	63	143	&#99;	c
100	64	144	&#100;	d





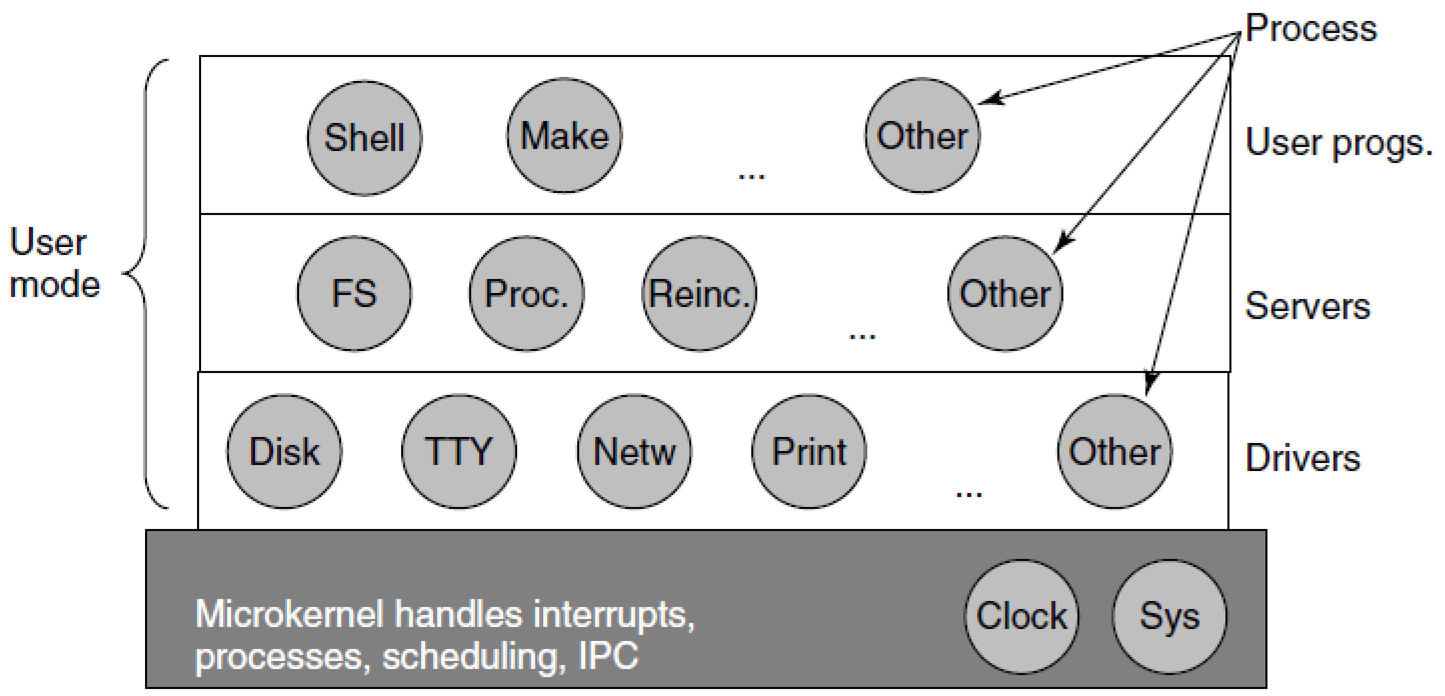
```
-a 100
0DB4:0100 mov ah, 02
0DB4:0102 mov dl, 61
0DB4:0104 int 21
0DB4:0106 int 20
0DB4:0108
-g 108
a
Program terminated normally (0000)
_
```



```
0DB4:0100 mov ah, 0e
0DB4:0102 mov al, 61
0DB4:0104 int 10
0DB4:0106 int 20
0DB4:0108
-g 108
a
Program terminated normally (0000)
```

Microkernels

The basic idea is to achieve high reliability by splitting the operating system up into small, well-defined modules, only one of which—the microkernel—runs in kernel mode and the rest run as relatively powerless ordinary user processes.



Simplified structure of the MINIX 3 system.

Microkernels

A microkernel OS minimizes the core kernel's responsibilities, running essential services (like inter-process communication and basic scheduling) within the kernel, while other services (like drivers and file systems) run in user space.

Advantages:

- **Modularity:** Easier to add, modify, or replace components without affecting the core kernel.
- **Stability:** A failure in a user-space service is less likely to crash the entire system.

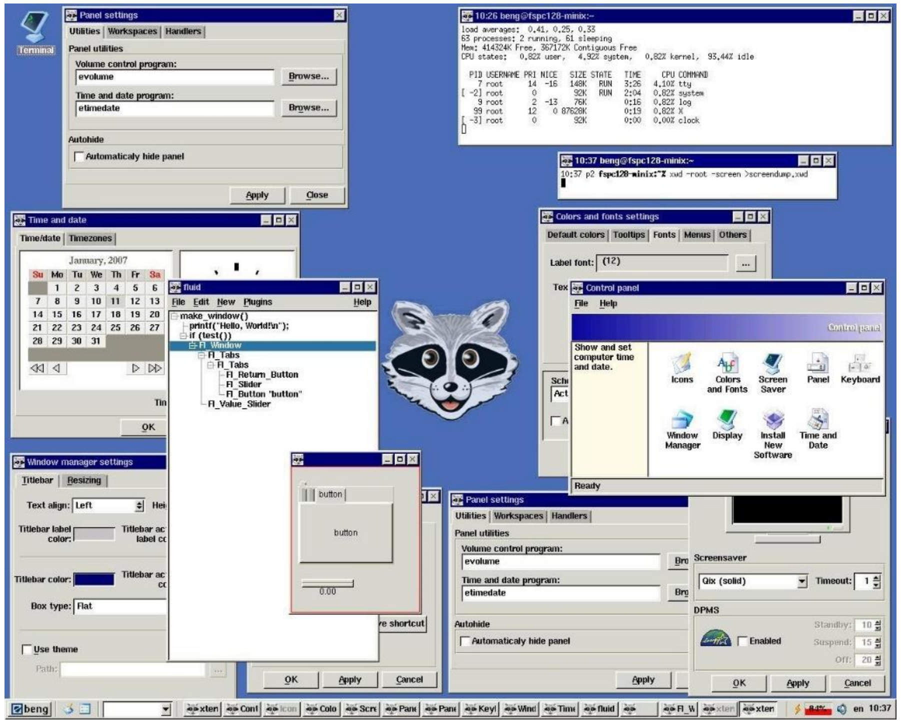
Disadvantages:

- **Performance:** Potentially higher overhead due to increased communication between user-space services and the microkernel.
- **Complexity:** More complex design and implementation compared to monolithic kernels.

Microkernels

Examples:

Minix, HelenOS, Horizon, The L4 microkernel family, Zircon (Fuchsia OS).



```
10:26 beng@fspc128-minix:~
load averages:  0.41, 0.25, 0.33
63 processes: 2 running, 61 sleeping
Mem: 414324K Free, 367172K Contiguous Free
CPU states:   0.82% user,   4.92% system,   0.82% kernel,  93.44% idle

  PID USERNAME PRI NICE   SIZE STATE   TIME     CPU COMMAND
    7 root      14  -16   148K   RUN   3:26   4.10% tty
[ -2] root       0         92K   RUN   2:04   0.82% system
    9 root       2  -13    76K         0:16   0.82% log
   99 root      12    0 87628K         0:19   0.82% X
[ -3] root       0         92K         0:00   0.00% clock
```

```
10:37 beng@fspc128-minix:~
10:37 p2 fspc128-minix:~% xwd -root -screen >screendump.xwd
```

```
make_window()
  printf("Hello, World!\n");
  if (test())
    Fl_Window
      Fl_Tabs
        Fl_Tabs
          Fl_Return_Button
          Fl_Slider
          Fl_Button "button"
      Fl_Value_Slider
```

Panel settings
Volume control program: evolume
Time and date program: etimedate

Time and date: January, 2007

Colors and fonts settings
Label font: (12)

Window manager settings
Text align: Left
Box type: Flat

Control panel
Show and set computer time and date.

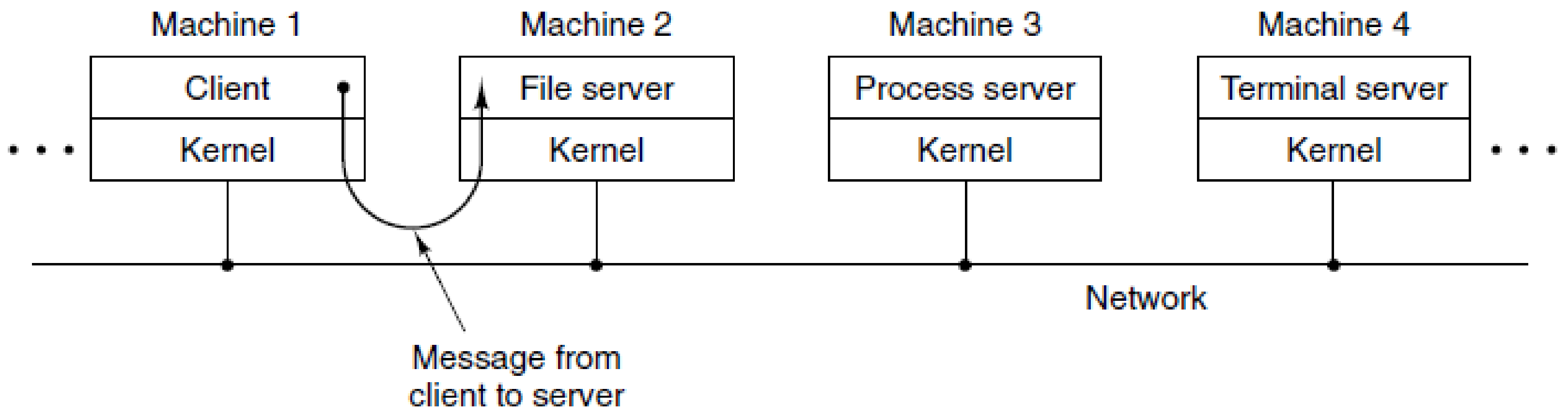
Screensaver: Qix (solid)
Timeout: 1
Standby: 10
Suspend: 15
Off: 20

MINIX 3

Client-Server Model

- Servers provide some service.
- Clients use these services.
- Communication between clients and servers is often by message passing.

- Client-server model is an abstraction that can be used for a single machine or for a network of machines.



The client-server model over a network.

Virtual Machines

Virtualization allows you to run two or more operating systems using only one machine.

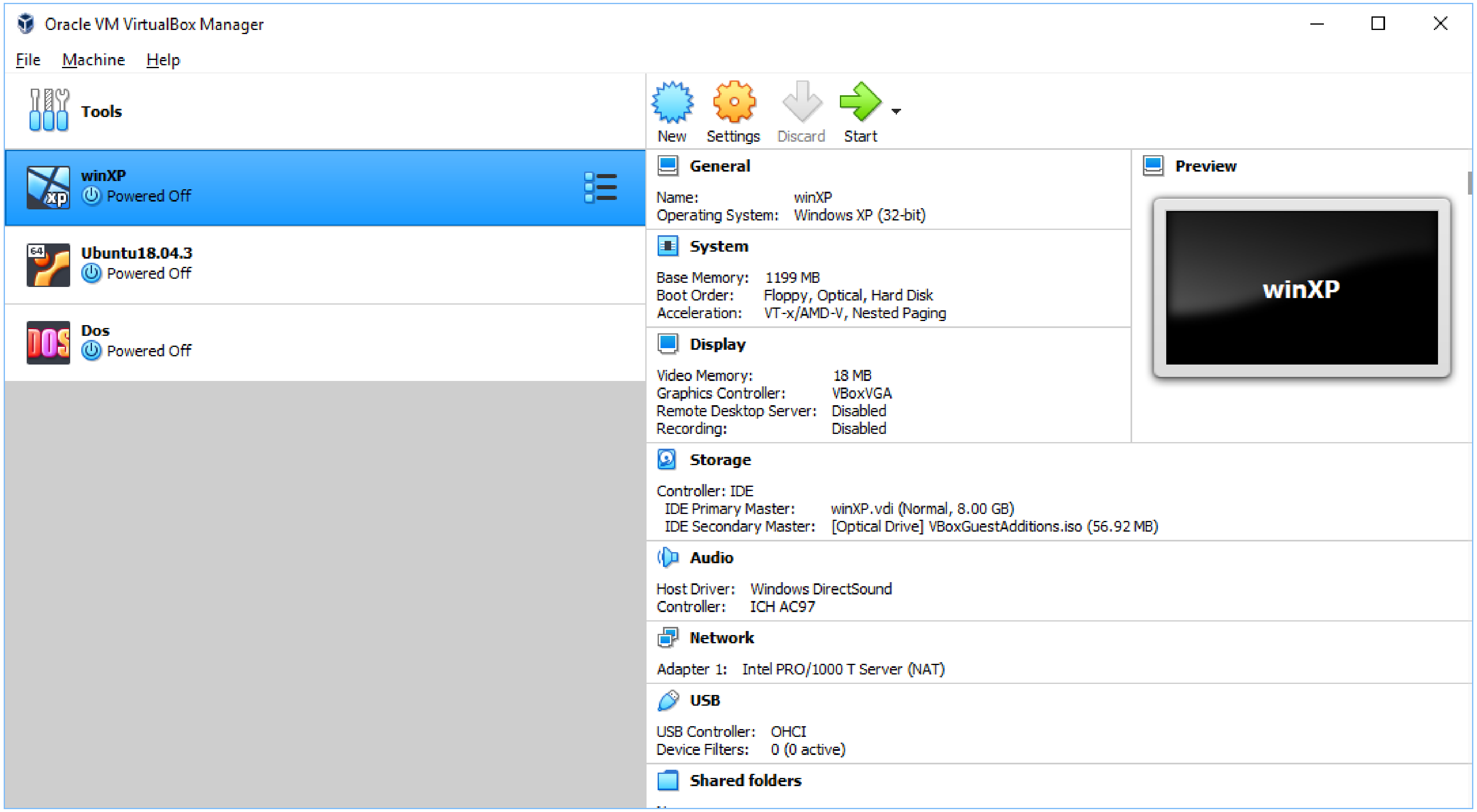
Before Virtualization: Many companies have traditionally run their mail servers, Web servers, FTP servers, and other servers on separate computers.

After Virtualization: They can run many servers on the same machine without having a crash of one server bring down the rest.

Web hosting world:

- **NO Virtualization**
  - Shared hosting, a login account on a Web server, but no control over the server software
  - Dedicated hosting, you have your own machine, very flexible but not cost effective.
- **With Virtualization**
  - Each customer can have different fully controllable operating systems on the same machine.

Virtual Machines


Oracle VM VirtualBox Manager

winXP
Powered Off

Ubuntu18.04.3
Powered Off

Dos
Powered Off

General

Name: winXP
Operating System: Windows XP (32-bit)

System

Base Memory: 1199 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging

Display

Video Memory: 18 MB
Graphics Controller: VBoxVGA
Remote Desktop Server: Disabled
Recording: Disabled

Preview

winXP

Storage

Controller: IDE
IDE Primary Master: winXP.vdi (Normal, 8.00 GB)
IDE Secondary Master: [Optical Drive] VBoxGuestAdditions.iso (56.92 MB)

Audio

Host Driver: Windows DirectSound
Controller: ICH AC97

Network

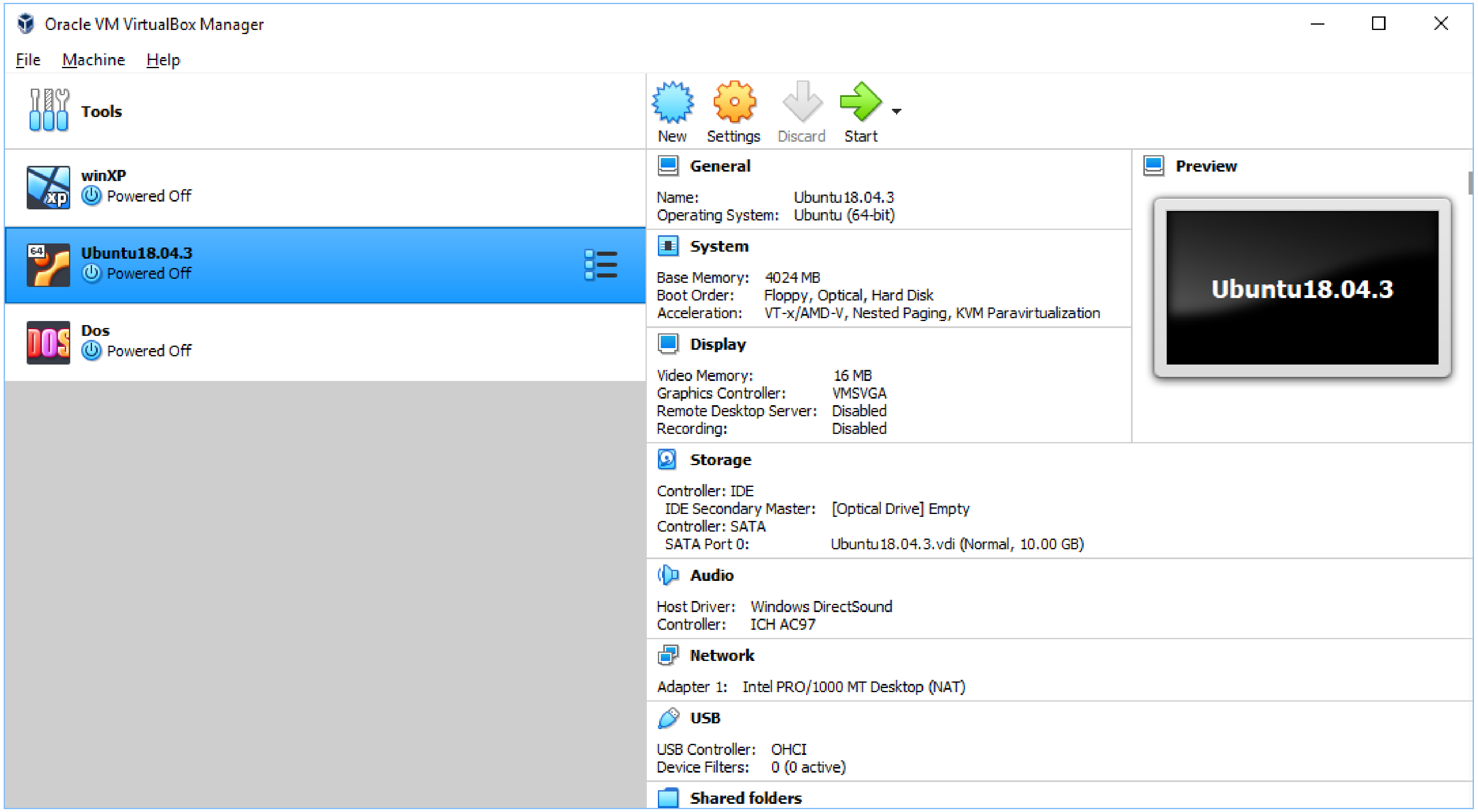
Adapter 1: Intel PRO/1000 T Server (NAT)

USB

USB Controller: OHCI
Device Filters: 0 (0 active)

Shared folders

Virtual Machines



Oracle VM VirtualBox Manager

File Machine Help

Tools

winXP
Powered Off

Ubuntu18.04.3
Powered Off

Dos
Powered Off

New Settings Discard Start

General

Name: Ubuntu18.04.3
Operating System: Ubuntu (64-bit)

System

Base Memory: 4024 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

Display

Video Memory: 16 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage

Controller: IDE
 IDE Secondary Master: [Optical Drive] Empty
Controller: SATA
 SATA Port 0: Ubuntu18.04.3.vdi (Normal, 10.00 GB)

Audio

Host Driver: Windows DirectSound
Controller: ICH AC97

Network

Adapter 1: Intel PRO/1000 MT Desktop (NAT)

USB

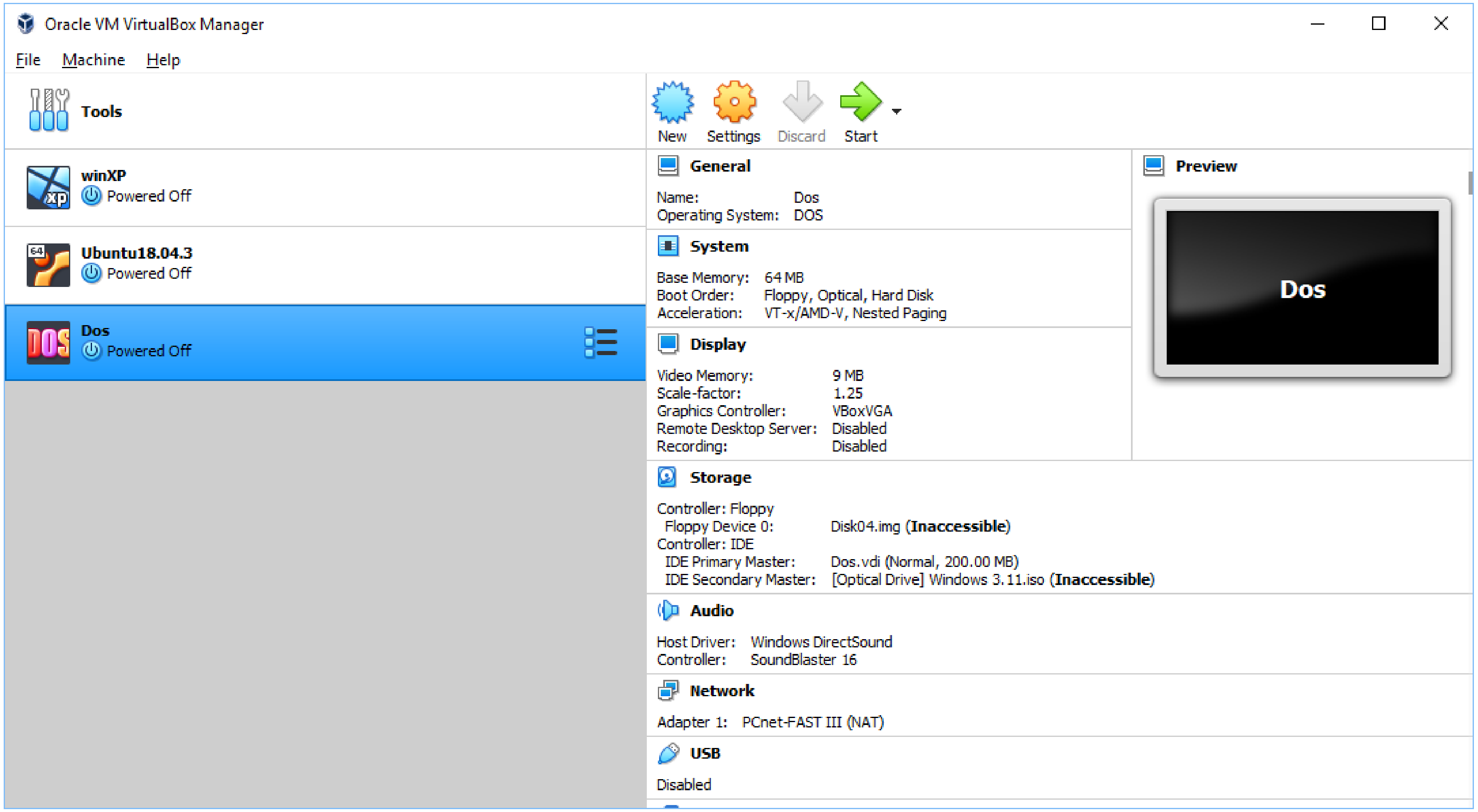
USB Controller: OHCI
Device Filters: 0 (0 active)

Shared folders

Preview

Ubuntu18.04.3

Virtual Machines



winXP
Powered Off

Ubuntu18.04.3
Powered Off

Dos
Powered Off

General

Name: Dos
Operating System: DOS

System

Base Memory: 64 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging

Display

Video Memory: 9 MB
Scale-factor: 1.25
Graphics Controller: VBoxVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage

Controller: Floppy
 Floppy Device 0: Disk04.img (**Inaccessible**)
Controller: IDE
 IDE Primary Master: Dos.vdi (Normal, 200.00 MB)
 IDE Secondary Master: [Optical Drive] Windows 3.11.iso (**Inaccessible**)

Audio

Host Driver: Windows DirectSound
Controller: SoundBlaster 16

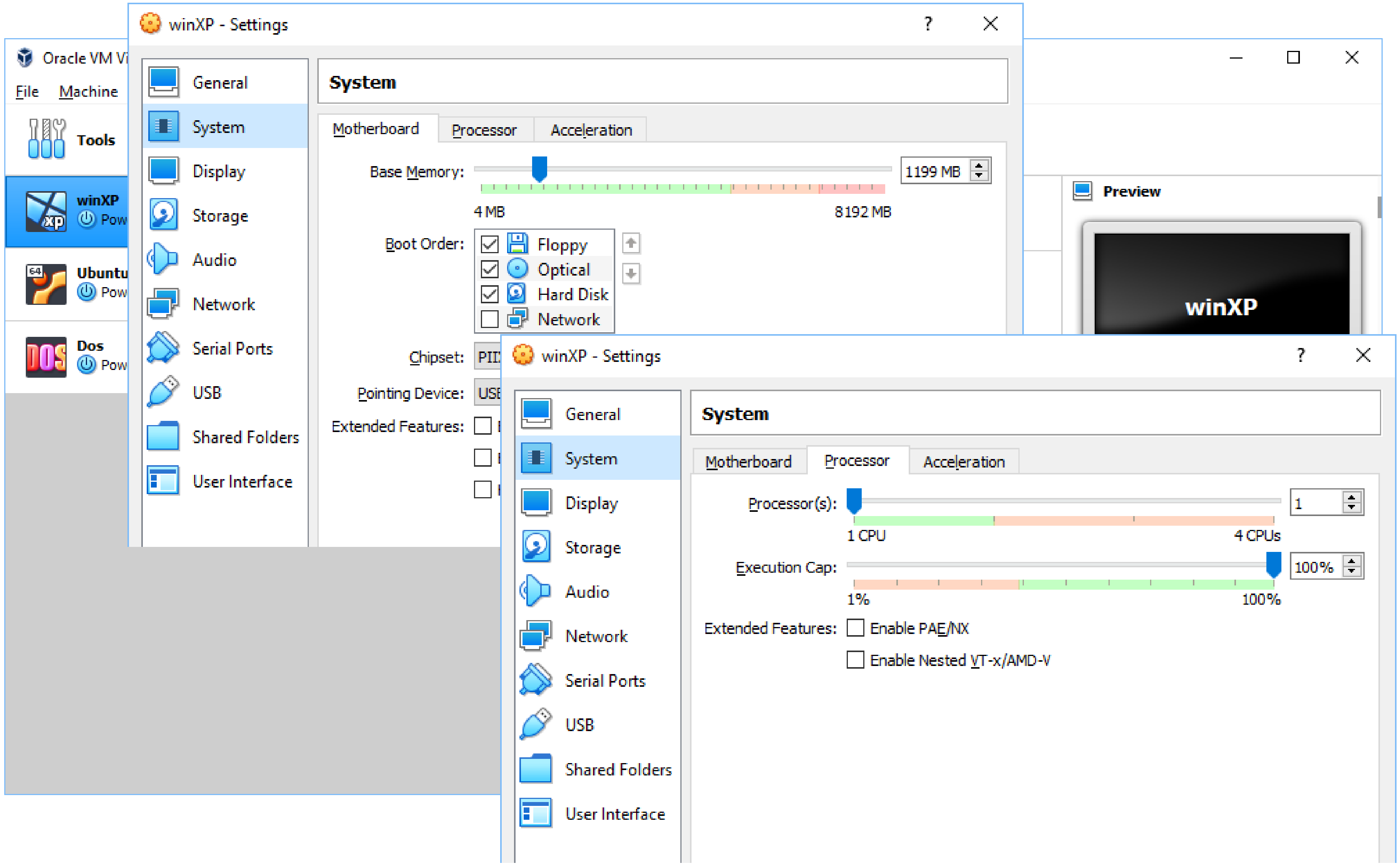
Network

Adapter 1: PCnet-FAST III (NAT)

USB

Disabled

Virtual Machines



winXP - Settings

General
System
Display
Storage
Audio
Network
Serial Ports
USB
Shared Folders
User Interface

System

Motherboard | Processor | Acceleration

Base Memory: 1199 MB (4 MB – 8192 MB)

Boot Order:
- [✓] Floppy
- [✓] Optical
- [✓] Hard Disk
- [ ] Network

Chipset: PII

Pointing Device: USB

Extended Features:

winXP - Settings

General
System
Display
Storage
Audio
Network
Serial Ports
USB
Shared Folders
User Interface

System

Motherboard | Processor | Acceleration

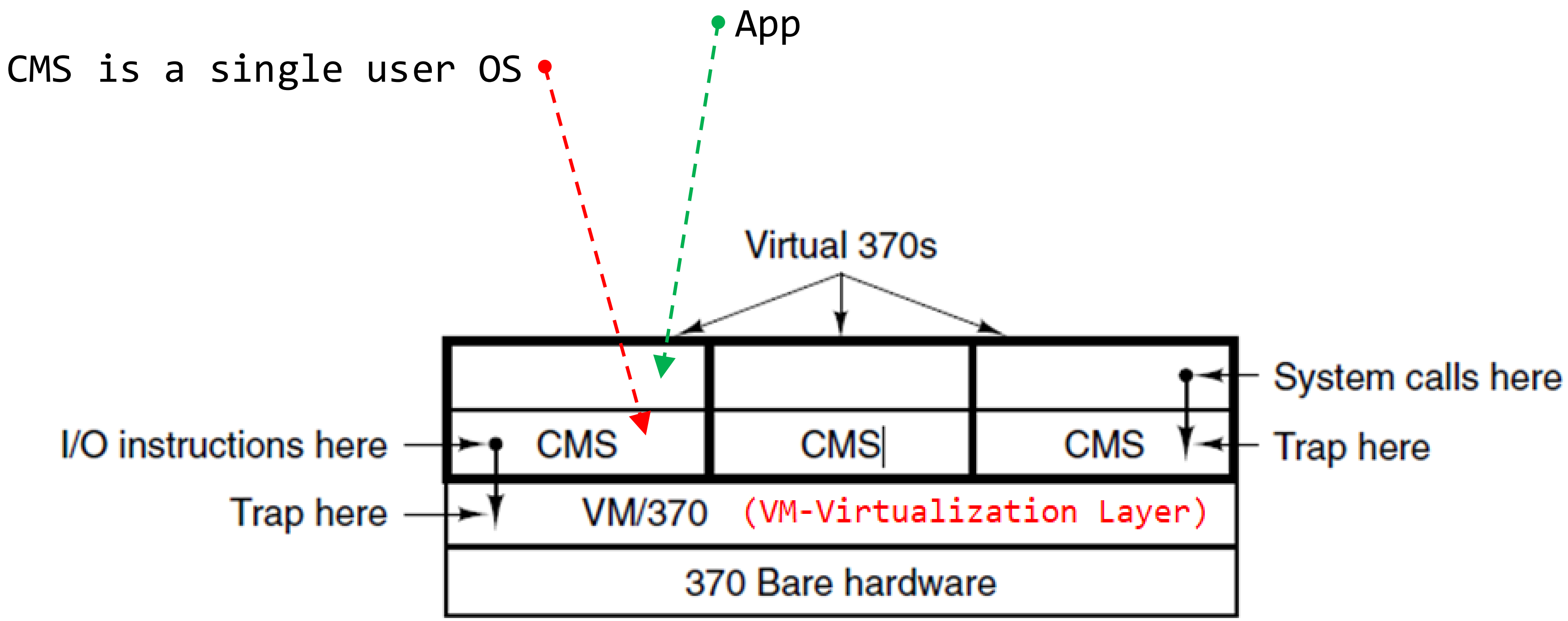
Processor(s): 1 (1 CPU – 4 CPUs)

Execution Cap: 100% (1% – 100%)

Extended Features:
- [ ] Enable PAE/NX
- [ ] Enable Nested VT-x/AMD-V

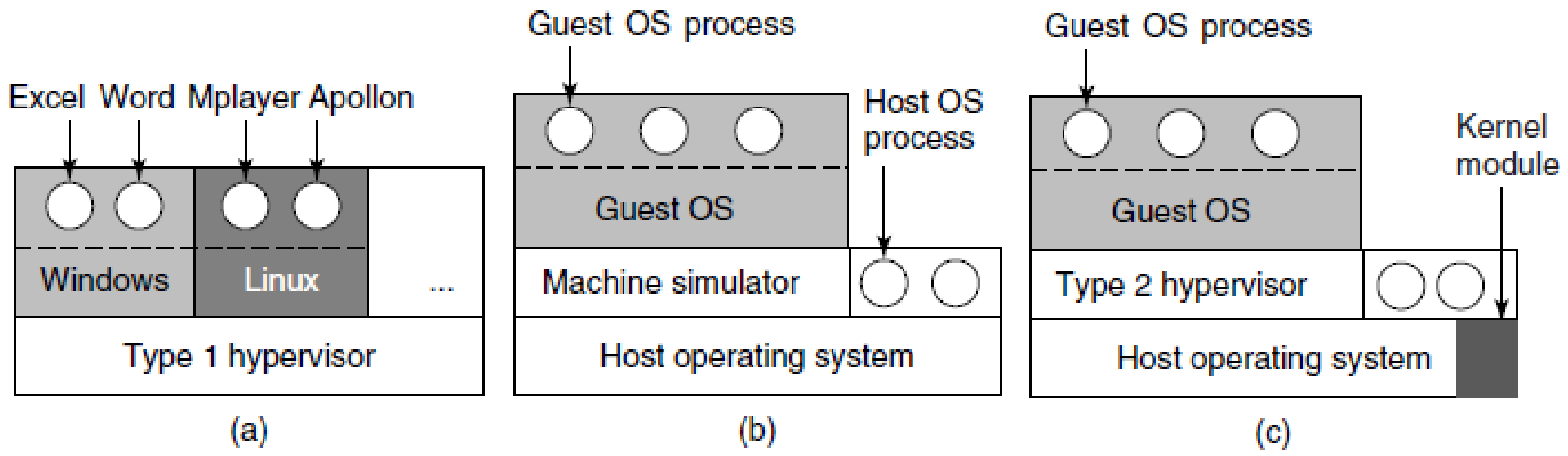
Virtual Machines

Control Program/Cambridge Monitor System (CP/CMS)



The structure of VM/370 with CMS.

Virtual Machines Rediscovered



(a) A type 1 hypervisor. (b) A pure type 2 hypervisor. (c) A practical type 2 hypervisor.

`Hypervisor (virtual machine monitor)`

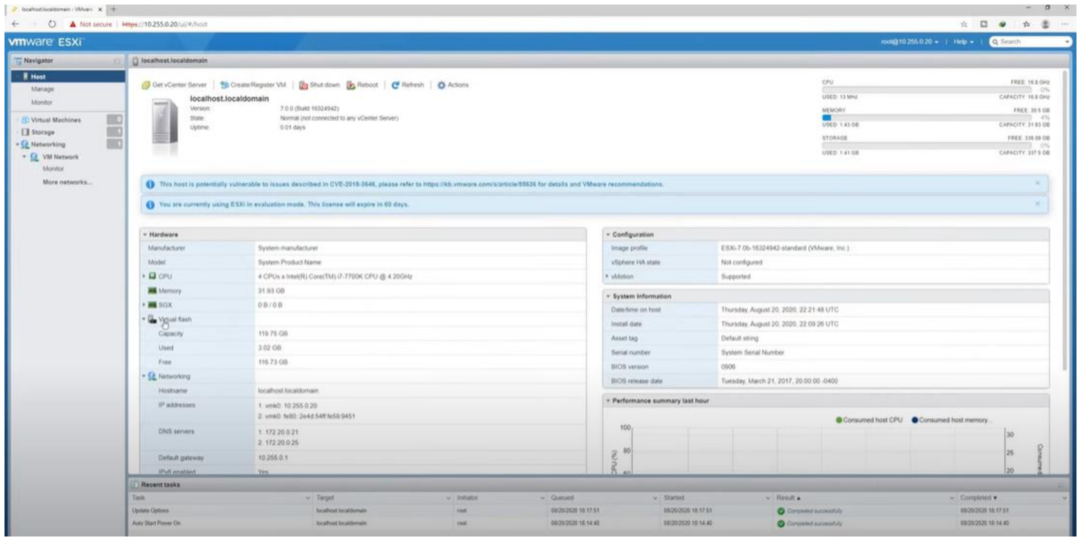
Virtual Machines Rediscovered

Type 1 hypervisor

Example:

ESXi (Elastic Sky X Integrated):

1. Type-1 hypervisor
2. Runs directly on system hardware
3. No need for an operating system (OS)



localhost.localdomain

localhost.localdomain

Version:	7.0.0 (Build 16324942)
State:	Normal (not connected to any vCenter Server)
Uptime:	0.01 days

CPU FREE: 16.8 GHz 0% USED: 13 MHz CAPACITY: 16.8 GHz
MEMORY FREE: 30.5 GB 4% USED: 1.43 GB CAPACITY: 31.93 GB
STORAGE FREE: 336.09 GB 0% USED: 1.41 GB CAPACITY: 337.5 GB

This host is potentially vulnerable to issues described in CVE-2018-3646, please refer to https://kb.vmware.com/s/article/55636 for details and VMware recommendations.

You are currently using ESXi in evaluation mode. This license will expire in 60 days.

Hardware

Manufacturer	System manufacturer
Model	System Product Name
CPU	4 CPUs x Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz
Memory	31.93 GB
SGX	0 B / 0 B
Virtual flash	
Capacity	119.75 GB
Used	3.02 GB
Free	116.73 GB
Networking	
Hostname	localhost.localdomain
IP addresses	1. vmk0: 10.255.0.20 2. vmk0: fe80::2e4d:54ff:fe59:9451
DNS servers	1. 172.20.0.21 2. 172.20.0.25
Default gateway	10.255.0.1
IPv6 enabled	Yes

Configuration

Image profile	ESXi-7.0b-16324942-standard (VMware, Inc.)
vSphere HA state	Not configured
vMotion	Supported

System Information

Date/time on host	Thursday, August 20, 2020, 22:21:48 UTC
Install date	Thursday, August 20, 2020, 22:09:26 UTC
Asset tag	Default string
Serial number	System Serial Number
BIOS version	0906
BIOS release date	Tuesday, March 21, 2017, 20:00:00 -0400

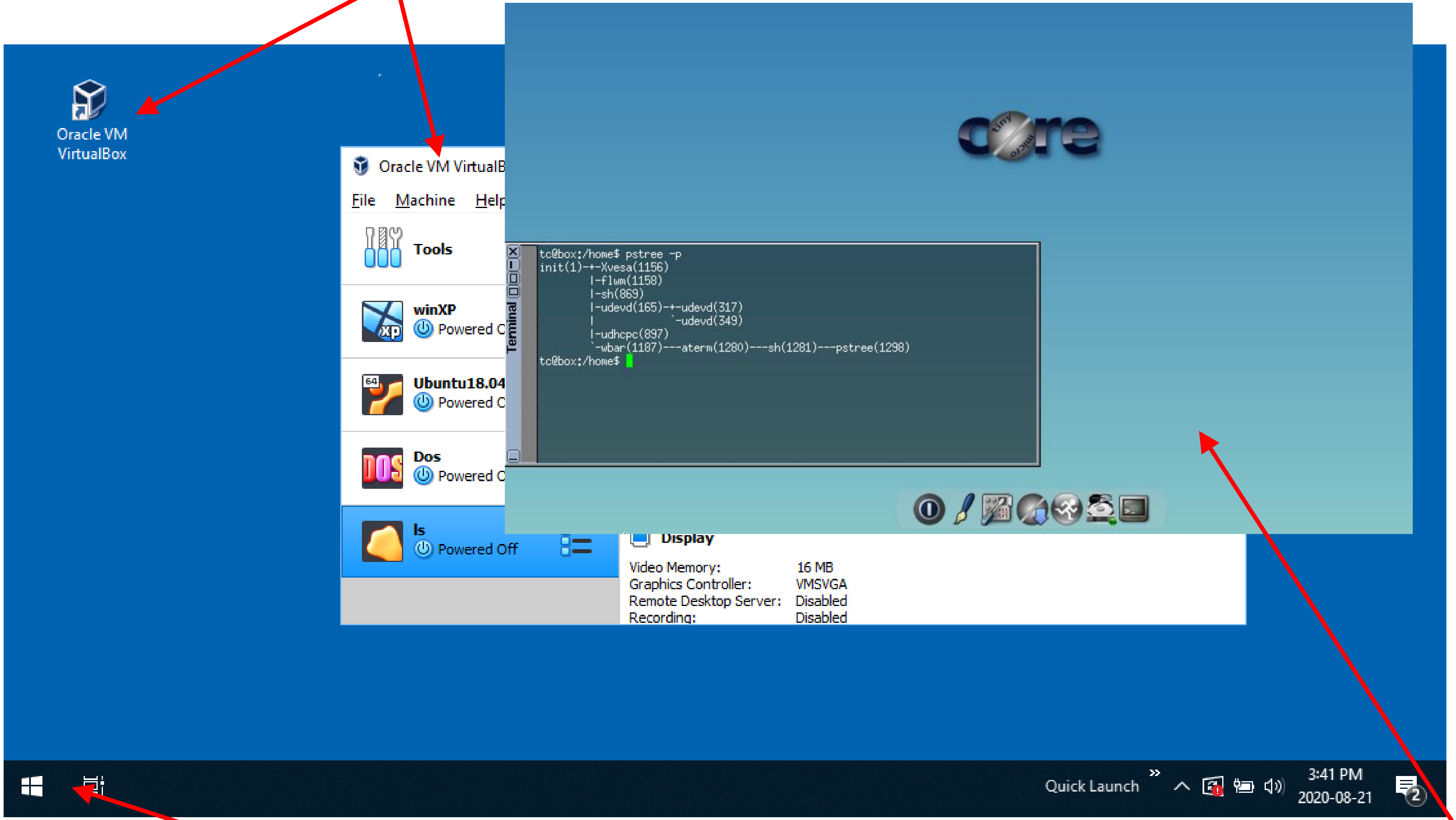
Performance summary last hour

Recent tasks

Task	Target	Initiator	Queued	Started	Result ▲	Completed ▼
Update Options	localhost.localdomain	root	08/20/2020 18:17:51	08/20/2020 18:17:51	✓ Completed successfully	08/20/2020 18:17:51
Auto Start Power On	localhost.localdomain	root	08/20/2020 18:14:40	08/20/2020 18:14:40	✓ Completed successfully	08/20/2020 18:14:40

Virtual Machines Rediscovered

Type 2 hypervisor(VirtualBox)



Video Memory: 16 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

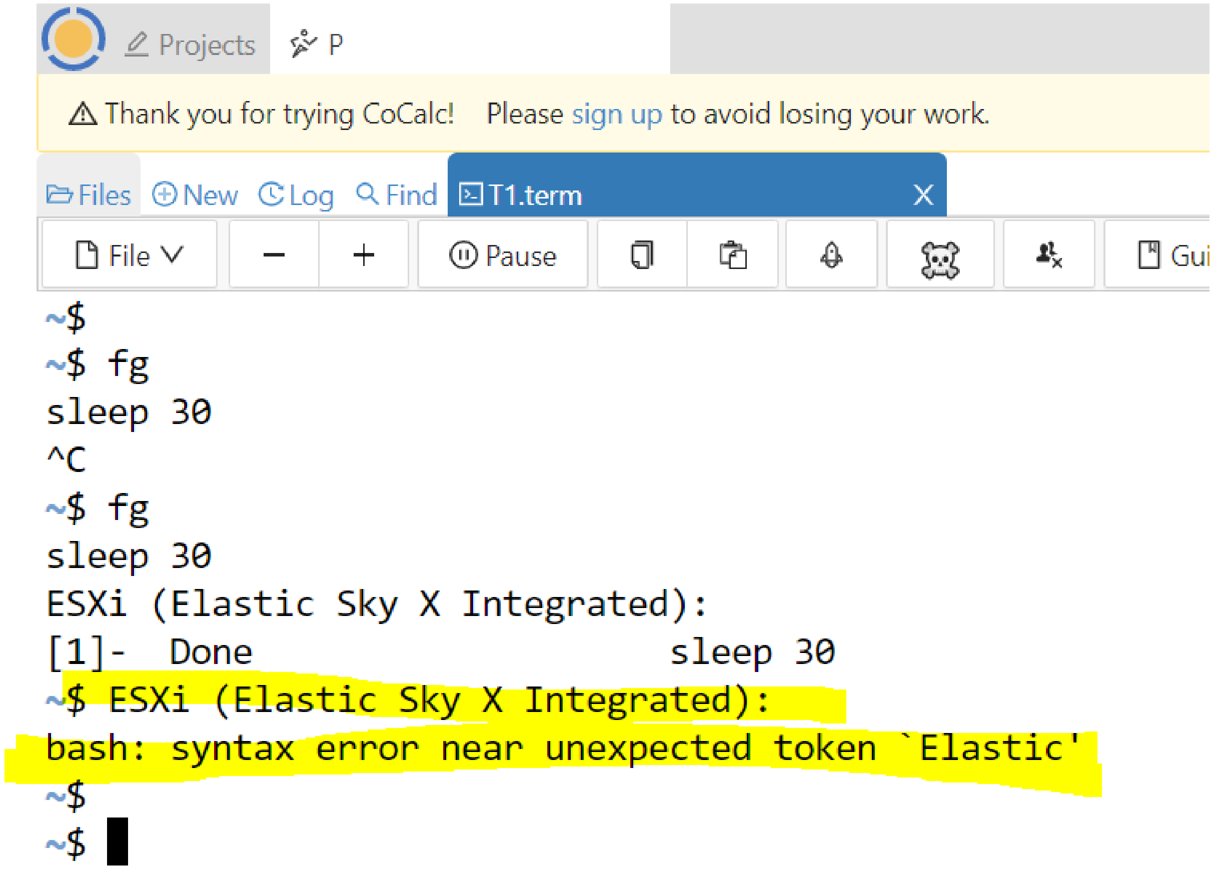
```
tc@box:/home$ pstree -p
init(1)-+-Xvesa(1156)
        |-flwm(1158)
        |-sh(869)
        |-udevd(165)-+-udevd(317)
        |            `-udevd(349)
        |-udhcpc(897)
        `-wbar(1187)---aterm(1280)---sh(1281)---pstree(1298)
tc@box:/home$
```

Host OS (Windows 10)

Guest OS (Tiny Core Linux)

Virtual Machines Rediscovered

```
Real or Virtual machine(type 1 or 2)?
```



⚠ Thank you for trying CoCalc!   Please sign up to avoid losing your work.

```
~$
~$ fg
sleep 30
^C
~$ fg
sleep 30
ESXi (Elastic Sky X Integrated):
[1]-  Done                    sleep 30
~$ ESXi (Elastic Sky X Integrated):
bash: syntax error near unexpected token `Elastic'
~$
~$ 
```

Clocks (Timers)

Clock Hardware

Clocks are built out of three components:

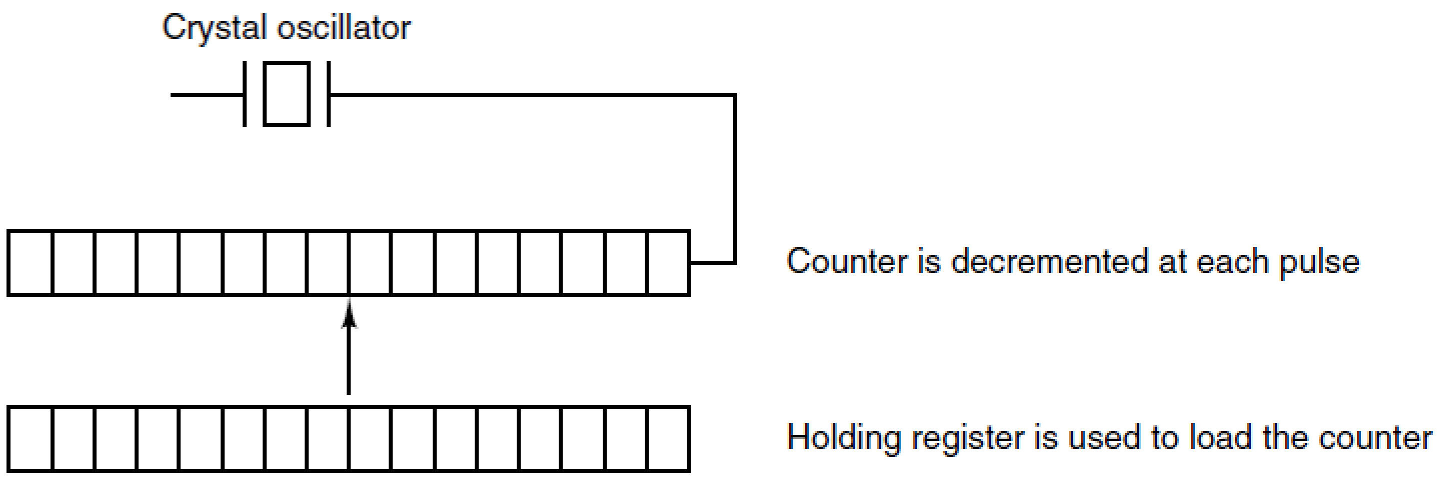
1) Crystal oscillator
Can generate a periodic signal, typically in the range of several hundred megahertz to a few gigahertz, depending on the crystal chosen.

2) Counter
This signal is fed into the counter to make it count down to zero. When the counter gets to zero, it causes a CPU interrupt.

3) Holding register
The value of the holding register is copied into the counter.

<u>All the clock hardware does is generate interrupts at known intervals.</u>

Clock Hardware


Crystal oscillator
Counter is decremented at each pulse
Holding register is used to load the counter

A programmable clock.

Clock Hardware

One-shot mode

When the clock is started, it copies the value of the holding register into the counter and then decrements the counter at each pulse from the crystal. When the counter gets to zero, it causes an interrupt and stops until it is explicitly started again by the software.

Square-wave mode

After getting to zero it causes an interrupt, the value of the holding register is copied into the counter and whole process is repeated again indefinitely. These periodic interrupts are called <u>clock ticks</u>.

Clock Hardware

The advantage of the programmable clock is that its interrupt frequency can be controlled by software.

If a 500-MHz crystal is used, then the counter is pulsed every 2 nsec. With (unsigned) 32-bit registers, interrupts can be programmed to occur at intervals from 2 nsec to 8.6 sec.

$T = 1/f = 1/(500 \times 10^6)$ sec $= 10^9 /(500 \times 10^6)$ nsec $= 2$ nsec

32-bit register $(2^{32}-1) \times 2$ nsec $= 8{,}589{,}934{,}590$ nsec ≈ 8.6 sec

Clock Software

Typical duties of a clock driver:

1. Maintaining the time of day.
2. Preventing processes from running longer than allowed.
3. Accounting for CPU usage.
4. Handling alarm system call from user processes.
5. Providing watchdog timers for parts of system itself.
6. Profiling, monitoring, statistics gathering.

Clock Software

Maintaining the time of day

Time of day in ticks

Clock rate f = 60 Hz and a 32-bit counter

T = 1/f = 1/60 ≈ 0.0167 second

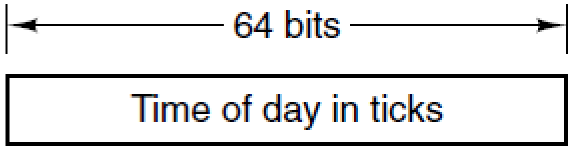
The largest unsigned 32-bit binary number = $2^{32} - 1$ = 4294967295

Total time = 4294967295×0.0167 = 71725953.8265 seconds = 2.274 years

Will overflow in just over 2 years.

The largest unsigned 64-bit binary number = $2^{64} - 1$

Total time = $(2^{64} - 1)$×0.0167 = 9768538369.8297023773 years



Makes maintaining the counter more expensive since it has to be done many times a second.

Clock Software

Maintaining the time of day

Time of day in seconds

The time of day is maintained in seconds, rather than in ticks.

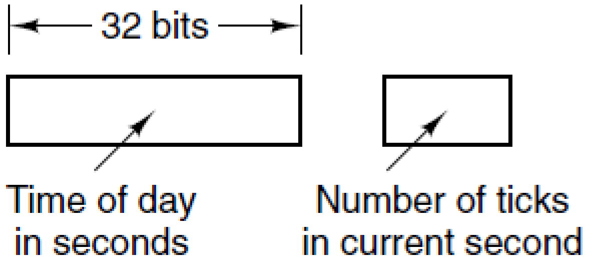
Clock rate f = 60 Hz and a 32-bit counter

1 second = 60 ticks

The largest unsigned 32-bit binary number = $2^{32} - 1 = 4294967295$

Total time = $4294967295 \times 1 = 4294967295$ seconds $\approx$ 136 years

Will work until the twenty-second century.

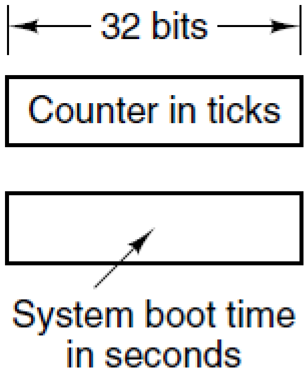


Clock Software

Maintaining the time of day

Count in ticks, but to do that relative to the time the system was booted, rather than relative to a fixed external moment.

Current time = the stored time of day in the memory + counter



Clock Software

Preventing processes from running longer than allowed

Whenever a process is started, the scheduler initializes a counter to the value of that process' quantum in clock ticks. At every clock interrupt, the clock driver decrements the quantum counter by 1. When it gets to zero, the clock driver calls the scheduler to set up another process.

Accounting for CPU usage

Start a second timer, distinct from the main system timer, whenever a process is started up. When that process is stopped, the timer can be read out to tell how long the process has run. To do things right, the second timer should be saved when an interrupt occurs and restored afterward.

Clock Software

Handling alarm system call from user processes

In many systems, a process can request that the operating system give it a warning (a signal, interrupt, message,…) after a certain interval. If many signals are expected, it is more efficient to simulate multiple clocks by chaining all the pending clock requests together, sorted on time, in a linked list.

Each entry on the list tells how many clock ticks following the previous one to wait before causing a signal.

Example:

Signals are pending for 4203, 4207, 4213, 4215, and 4216.

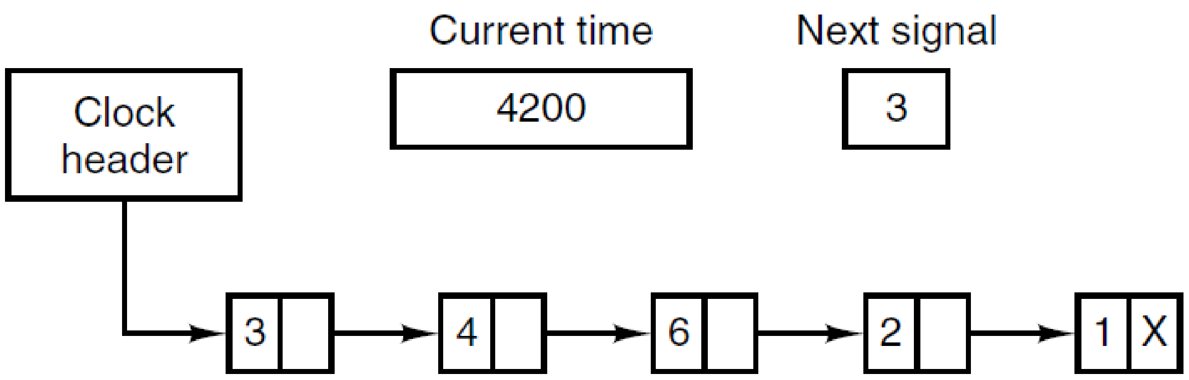
Clock Software

Handling alarm system call from user processes

Example:

Signals are pending for 4203, 4207, 4213, 4215, and 4216.

the next interrupt occurs in 3 ticks. On each tick, *Next signal* is decremented. When *Next signal* = 0, the signal corresponding to the first item on the list is caused, and that item is removed from the list. Then *Next signal = 4*.

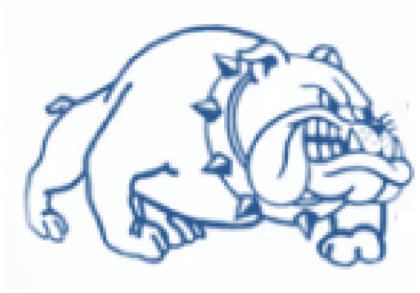


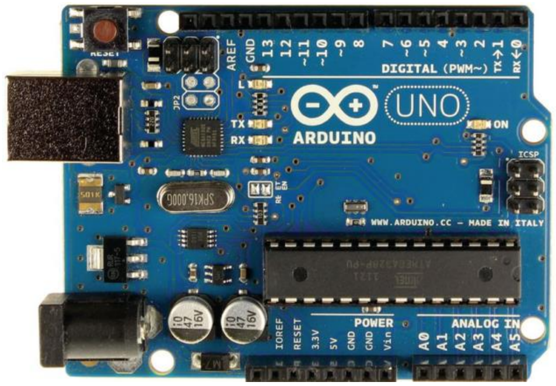
Simulating multiple timers with a single clock.

Clock Software

Providing watchdog timers for parts of system itself

Detect and recover from crashes, infinite loops Example: For instance, a watchdog timer may reset a system that stops running, especially used in embedded system.





https://www.youtube.com/watch?v=GC6dGypGctQ