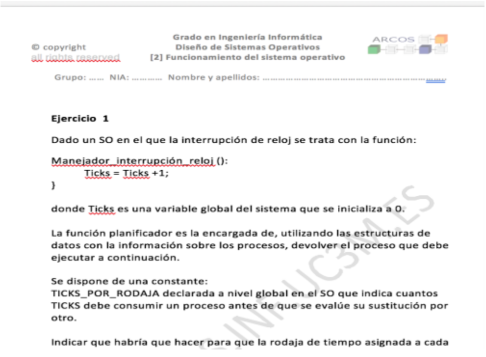
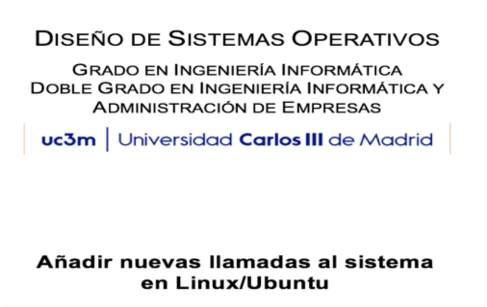
ARCOS Group
Universidad Carlos III de Madrid

Lesson 2
How an operating system works

Operating System Design
Degree in Computer Science and Engineering

Exercises, guided labs and laboratories

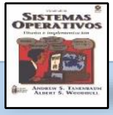
Exercises ✓	Guided Labs. ✓	Laboratories X
 © copyright all rights reserved Grado en Ingeniería Informática Diseño de Sistemas Operativos [2] Funcionamiento del sistema operativo ARCOS Grupo: ...... NIA: .... Nombre y apellidos: .... **Ejercicio 1** Dado un SO en el que la interrupción de reloj se trata con la función: Manejador_interrupción_reloj (): Ticks = Ticks +1; } donde Ticks es una variable global del sistema que se inicializa a 0. La función planificador es la encargada de, utilizando las estructuras de datos con la información sobre los procesos, devolver el proceso que debe ejecutar a continuación. Se dispone de una constante: TICKS_POR_RODAJA declarada a nivel global en el SO que indica cuantos TICKS debe consumir un proceso antes de que se evalúe su sustitución por otro. Indicar que habría que hacer para que la rodaja de tiempo asignada a cada	 DISEÑO DE SISTEMAS OPERATIVOS GRADO EN INGENIERÍA INFORMÁTICA DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y ADMINISTRACIÓN DE EMPRESAS uc3m | Universidad Carlos III de Madrid **Añadir nuevas llamadas al sistema en Linux/Ubuntu**	

Recommended readings

Base

1. Carretero 2007:
   1. Cap.2

Recommended



1. Tanenbaum 2006(en):
   1. Cap.1
2. Stallings 2005:
   1. Parte uno (transfondo)
3. Silberschatz 2006:
   1. Cap.2

To remember…

1. To prepare and review the class explanations.
   - Study the bibliography material: only slides are not enough.
   - Ask your doubts.

1. To exercise skills and abilities.
   - Solve as much exercises as possible.
   - Perform the guided laboratories progressively.
   - Build laboratories progressively.

Overview

- Introduction
- How an operating system works
  - System boot
  - Characteristics and event handling
  - Kernel process
- Other aspects
  - Events concurrency
  - Add new system functionalities

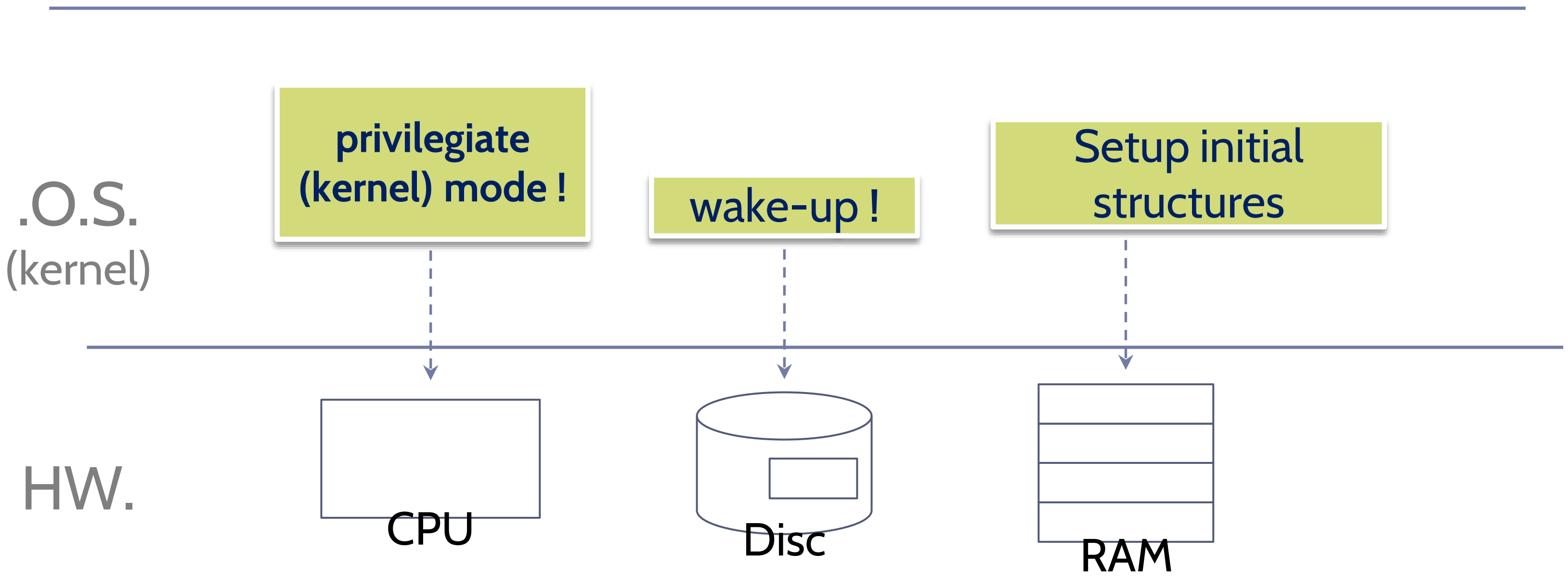
Overview

- **Introduction**
- How an operating system works
  - System boot
  - Characteristics and event handling
  - Kernel process
- Other aspects
  - Events concurrency
  - Add new system functionalities

Scenarios where the O.S. is present (1/3)

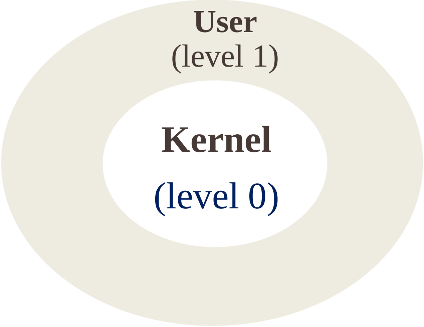
- System boot
  - It initialize the hardware and the kernel process, system and users in the proper order.
  - Behavior as executable application.

Simplified example



kernel and user mode
review

- The operating system needs, at least, two execution modes:

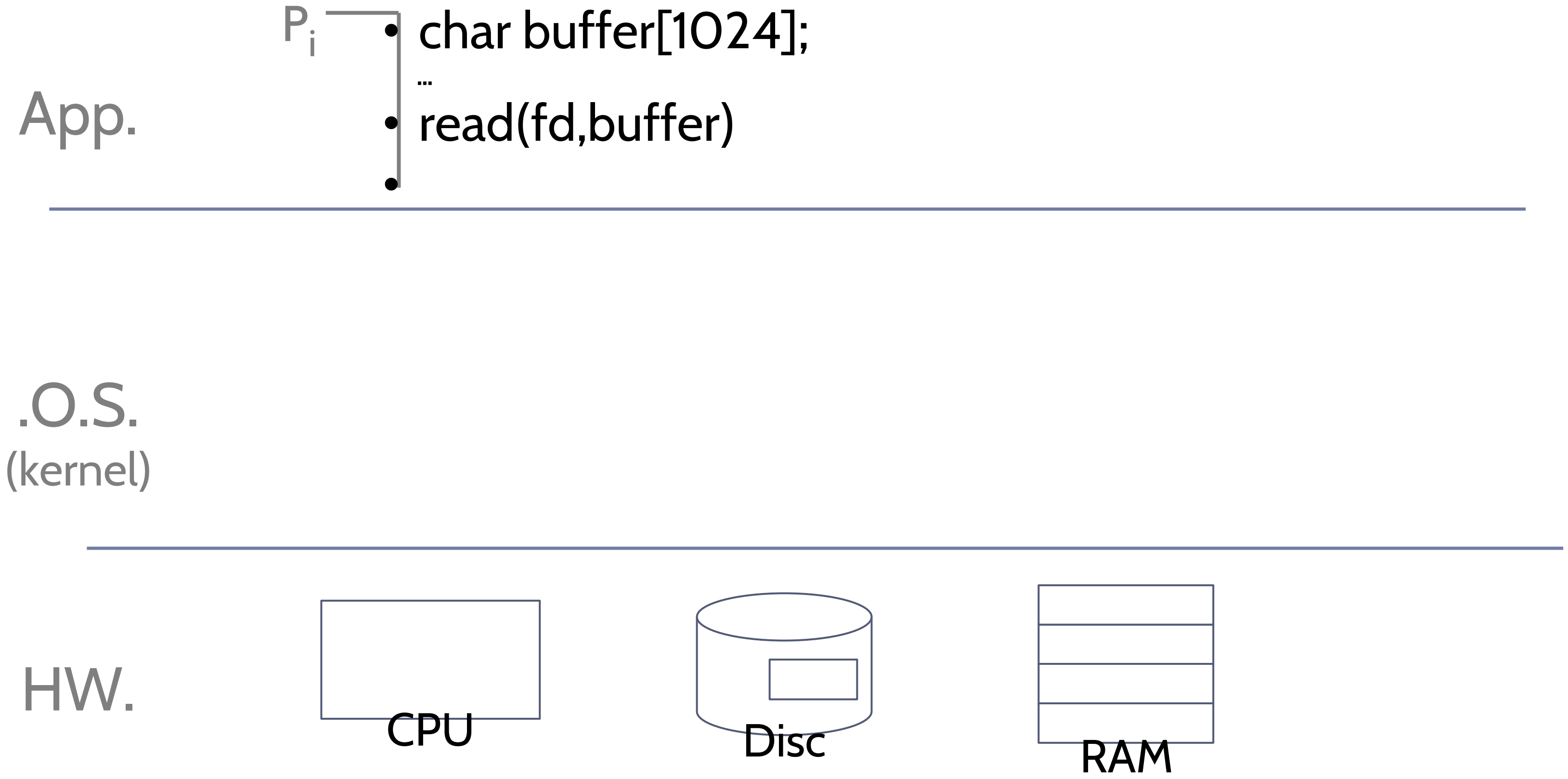


- Privileged mode (**kernel mode**)
  - Able to access to all memory space
  - Able to use all CPU resources
- Ordinary mode (**user mode**)
  - Restricted memory space
  - Some registers or instructions are limited

Scenarios where the O.S. is present (2/3)

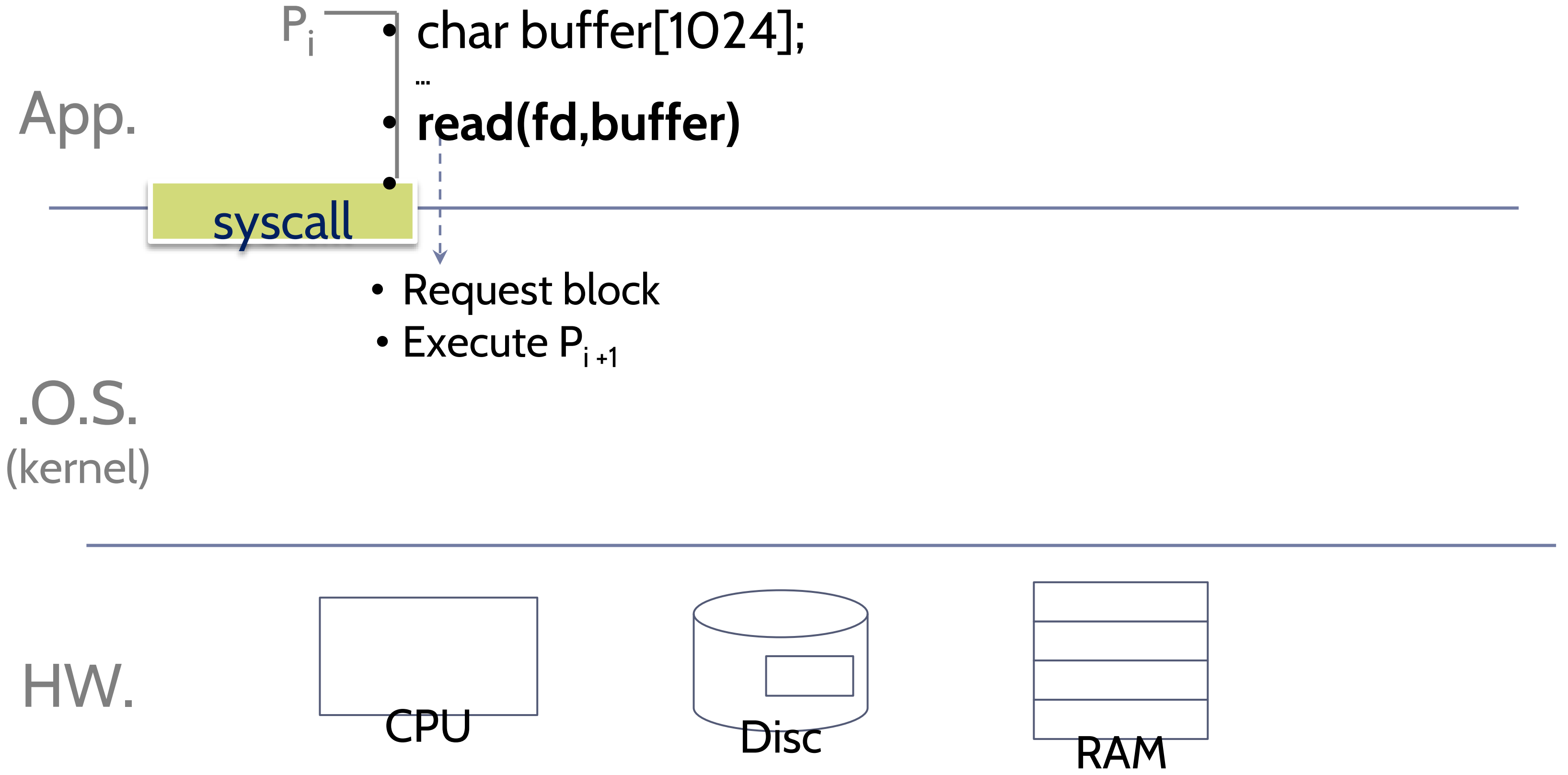
- Event handling (Event treatments)
  - Once booted, the operating system is a passive entity
    - Process and hardware are the active entities (and they use the kernel)
    - Except at boot-time, always there is a process executing (e.g.: *idle*)
  - Access to O.S. services through event handling
    - Hardware interrupts
    - Software interrupts
    - Exceptions
    - System calls
  - Behavior as library.

Simplified example



```
• char buffer[1024];
…
• read(fd,buffer)
•
```

Simplified example



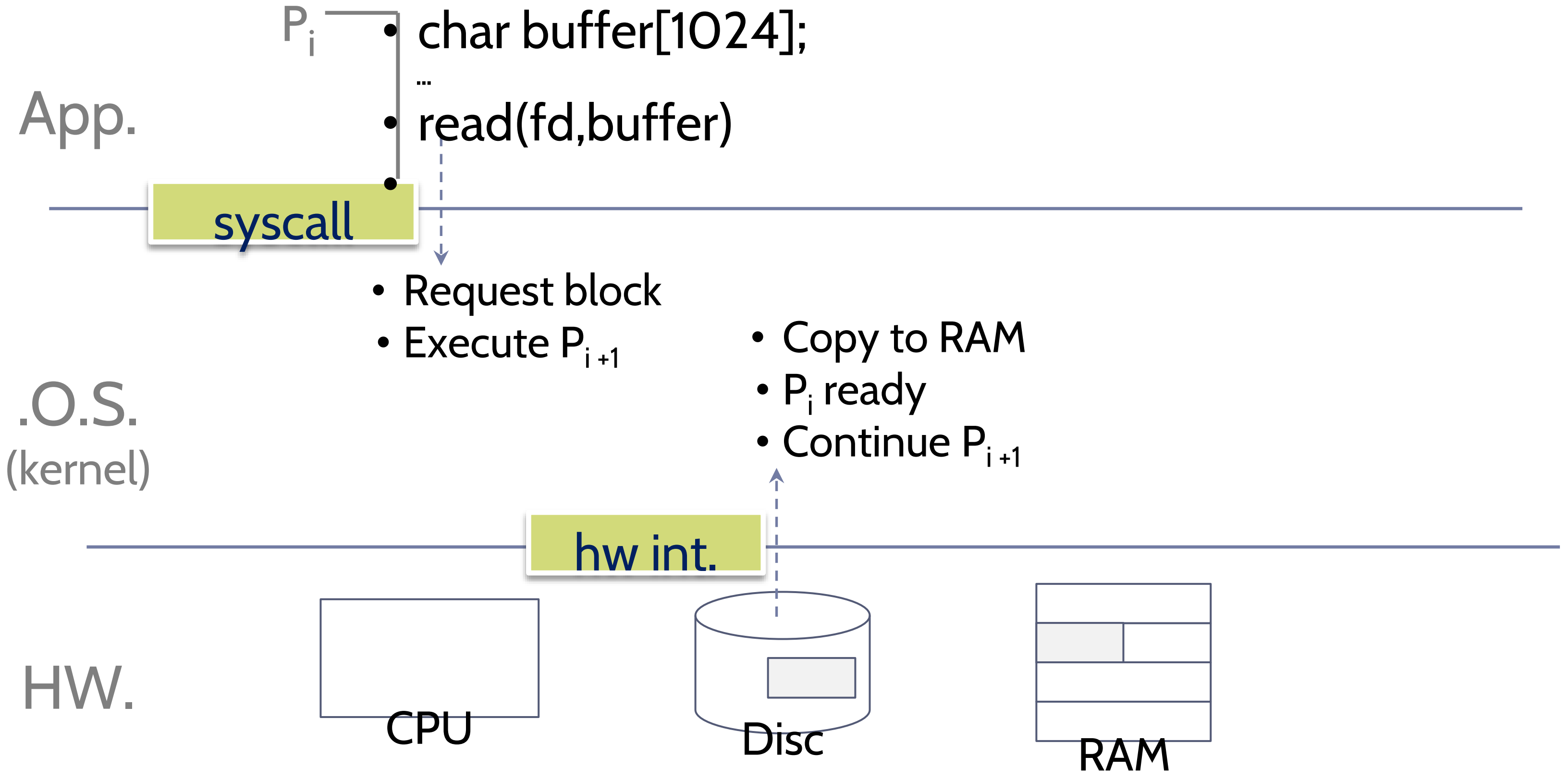
- char buffer[1024];

...

- **read(fd,buffer)**

- Request block
- Execute P_{i+1}

Simplified example



- char buffer[1024];

...

- read(fd,buffer)
-

- Request block
- Execute P_{i+1}

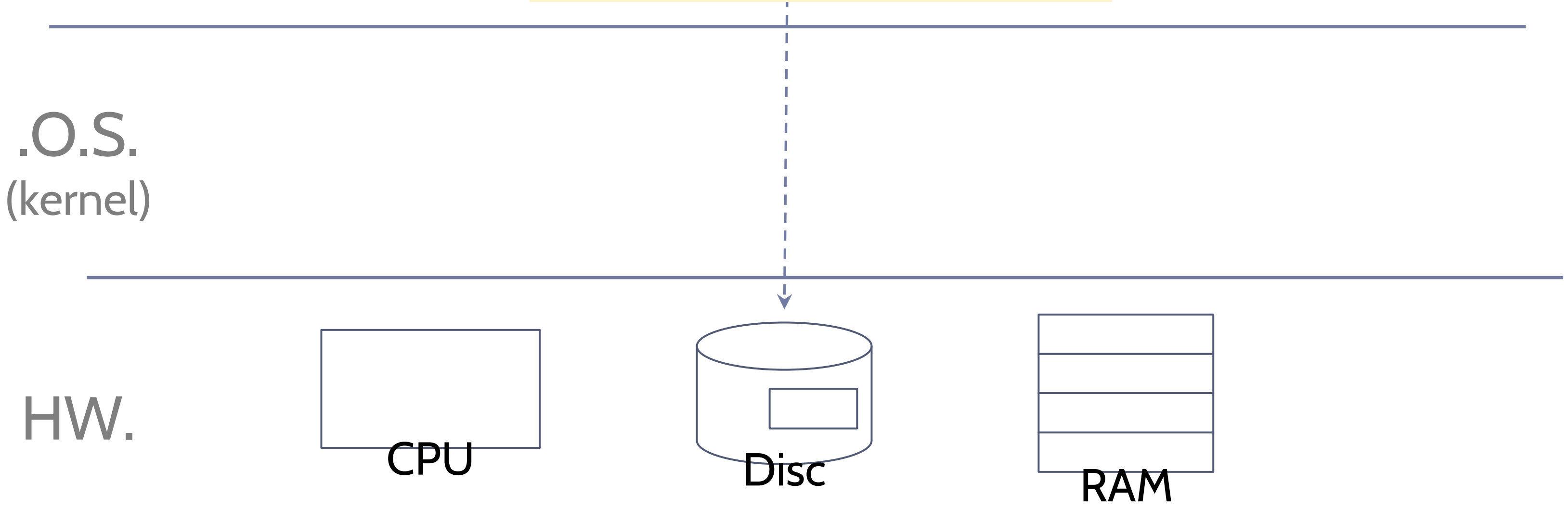
- Copy to RAM
- P_i ready
- Continue P_{i+1}

Scenarios where the O.S. is present (3/3)

- Kernel process
  - It performs tasks related to the operating system that are better developed in the context of a independent process.
  - Behavior as proprietary process, for special tasks.

Simplified example

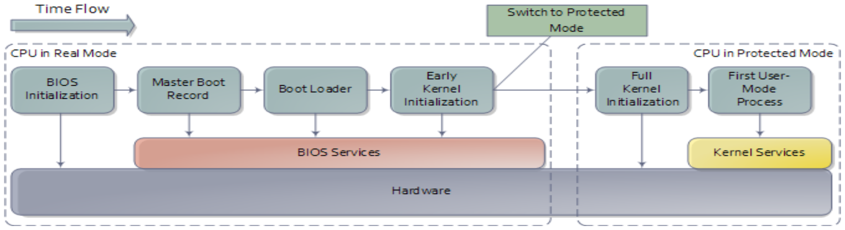
```
while (true) {
        • sleep(1);
        • If (idle > 20m)
            issue sleep to disk
}
```



Scenarios where the O.S. is present
summary

- System boot
  - Perform **initialization tasks for hardware, kernel, and processes in the proper order**.
  - Run as executable program.
- Event handling (treatment)
  - After booting, the **operating system is a passive entity**.
    - Processes and hardware are active entities (they use the kernel)
    - Except at the beginning, there is always a process running (idle)
  - Access to the services of the .O.S.
    - Hardware Int, Software Int, Exceptions, and System calls
  - As library.
- Kernel process
  - Performs operating system **tasks that are best done in the context of an independent process**
  - As priority processes, for special tasks.

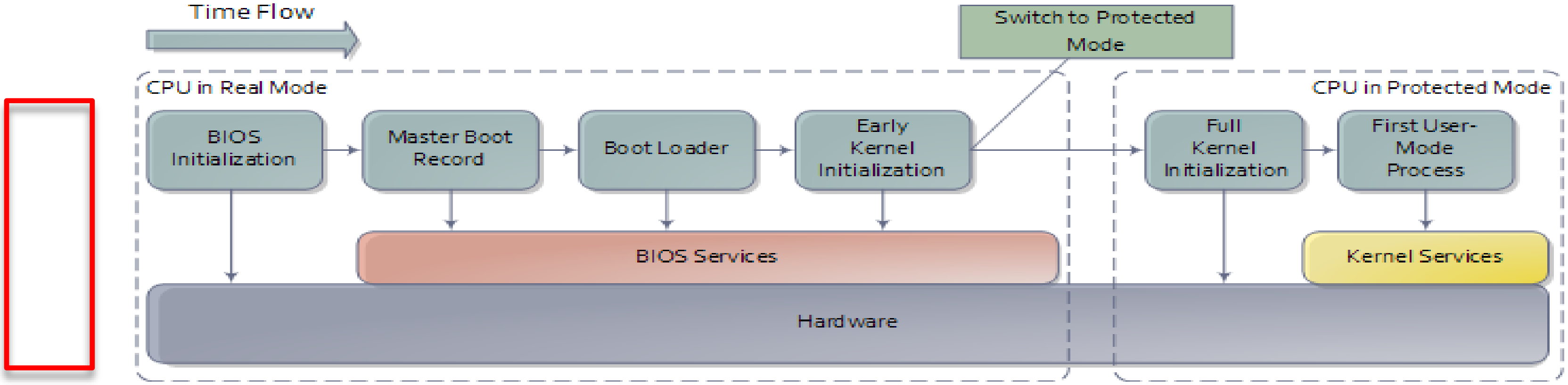
Overview

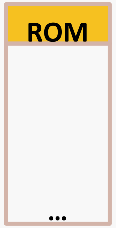


- Introduction
- How an operating system works
  - **System boot**
  - Characteristics and event handling
  - Kernel process
- Other aspects
  - Events concurrency
  - Add new system functionalities

Boot process

PC



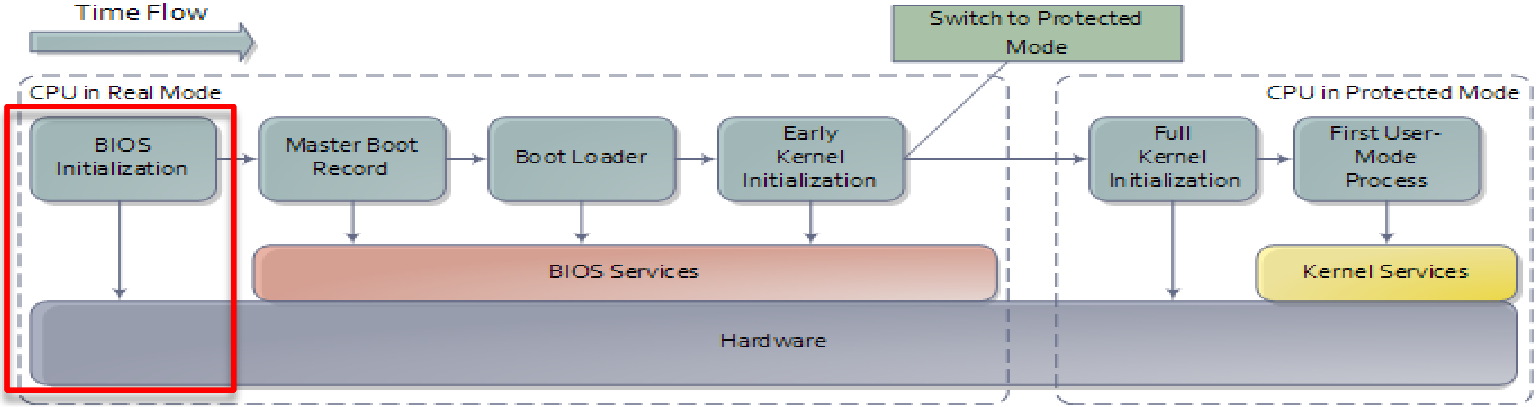


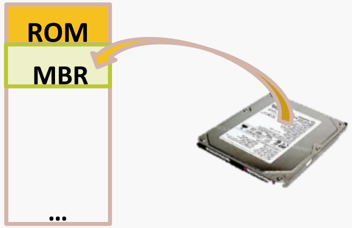
- The *Reset* loads the initial values in the CPU registers
  - PC ← Boot address of the ROM loader (FFFF:0000)

http://duartes.org/gustavo/blog/post/how-computers-boot-up

Boot process

PC



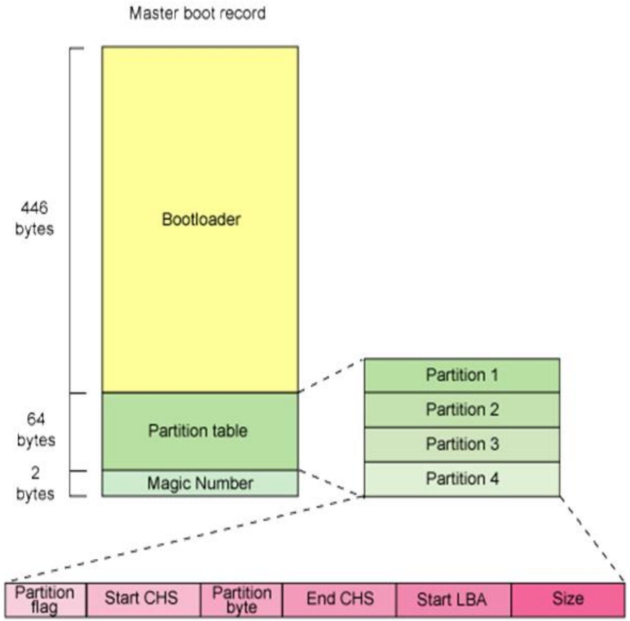


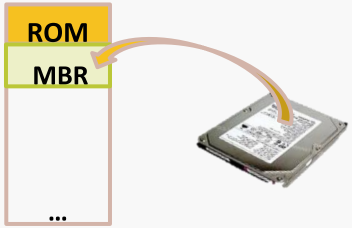
- The boot loader ROM is executed
  - *Power-On Self Test* (POST)
  - *Master Boot Record* is loaded into memory (0000:7C00)

http://duartes.org/gustavo/blog/post/how-computers-boot-up

Boot process

PC



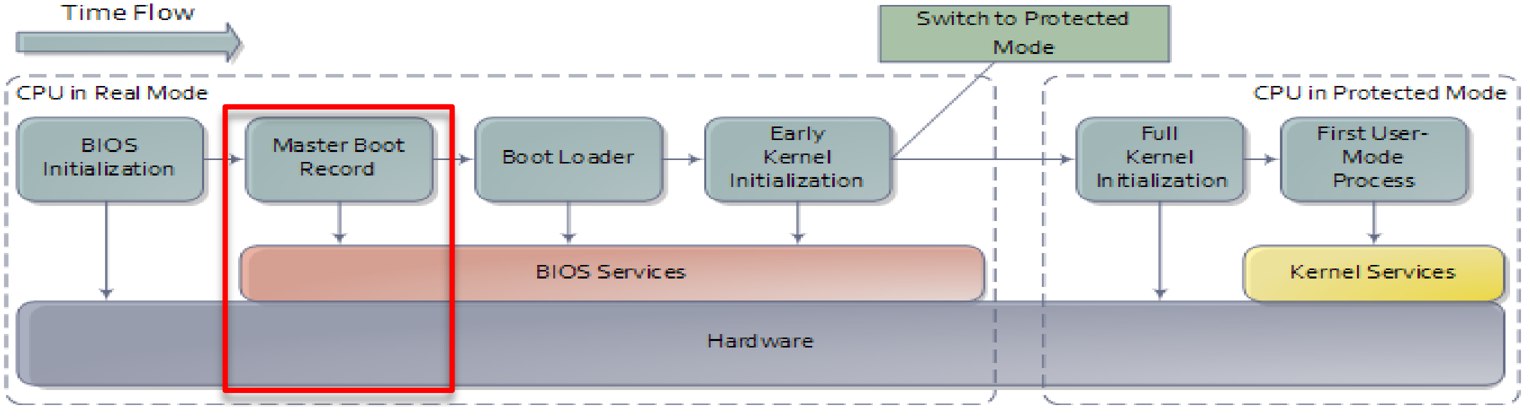


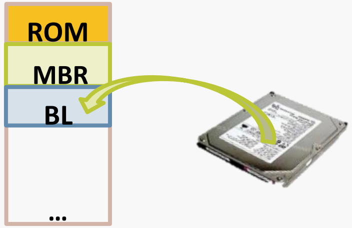
- The boot loader ROM is executed
  - *Power-On Self Test* (POST)
  - *Master Boot Record* is loaded into memory (0000:7C00)

http://www.ibm.com/developerworks/linux/library/l-linuxboot/

Boot process

PC



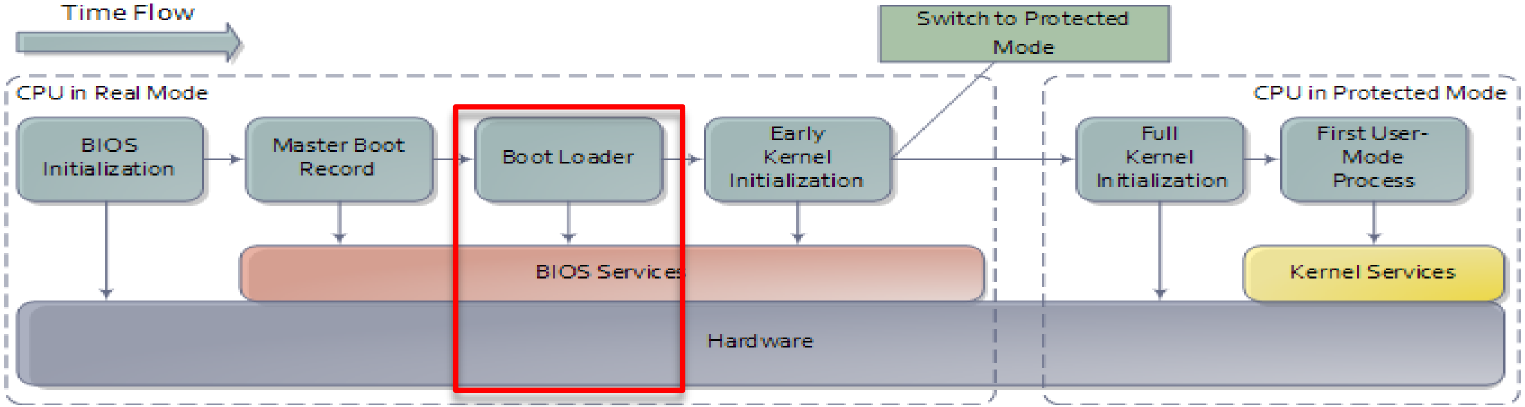


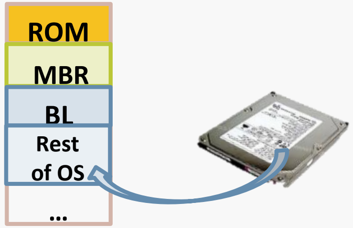
- The *Master Boot Record* is executed
  - (It is the first part of the O.S. loader)
  - It searches for an active partition in the partition table
  - It loads the *Boot Record* into memory from this partition

http://duartes.org/gustavo/blog/post/how-computers-boot-up

Boot process

PC

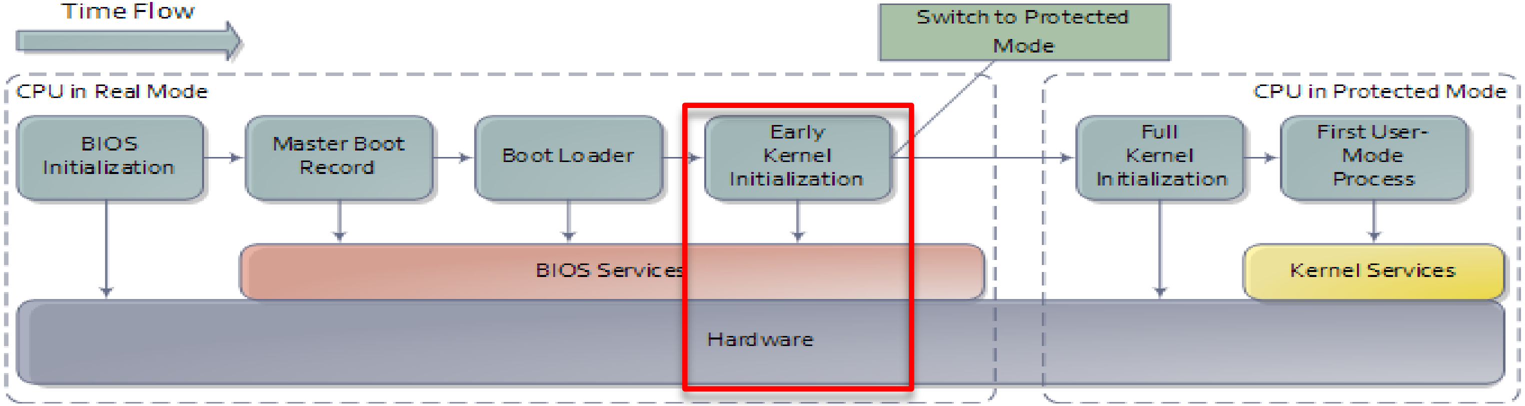


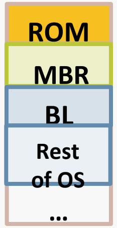


- The *Boot Loader* is executed
  - (It is the second part of the O.S. loader)
  - It might show some boot option list…
  - The boot loader loads into memory the resident part of the operating system (kernel and modules)

http://duartes.org/gustavo/blog/post/how-computers-boot-up

Boot process PC



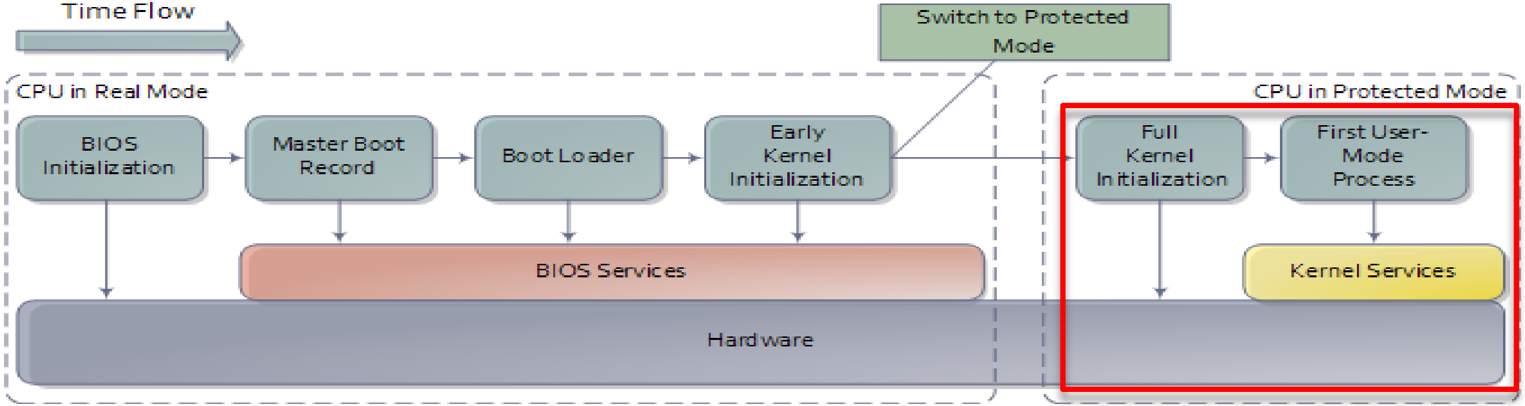


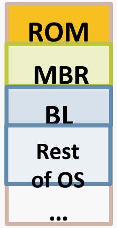
- The kernel initialization is performed (1/2)
  - Hardware initialization
  - Check errors in file systems
  - Establishes the initial internal structures of the O.S.
  - Switch to protected mode

http://duartes.org/gustavo/blog/post/how-computers-boot-up

Boot process

PC




ROM
MBR
BL
Rest of OS
…

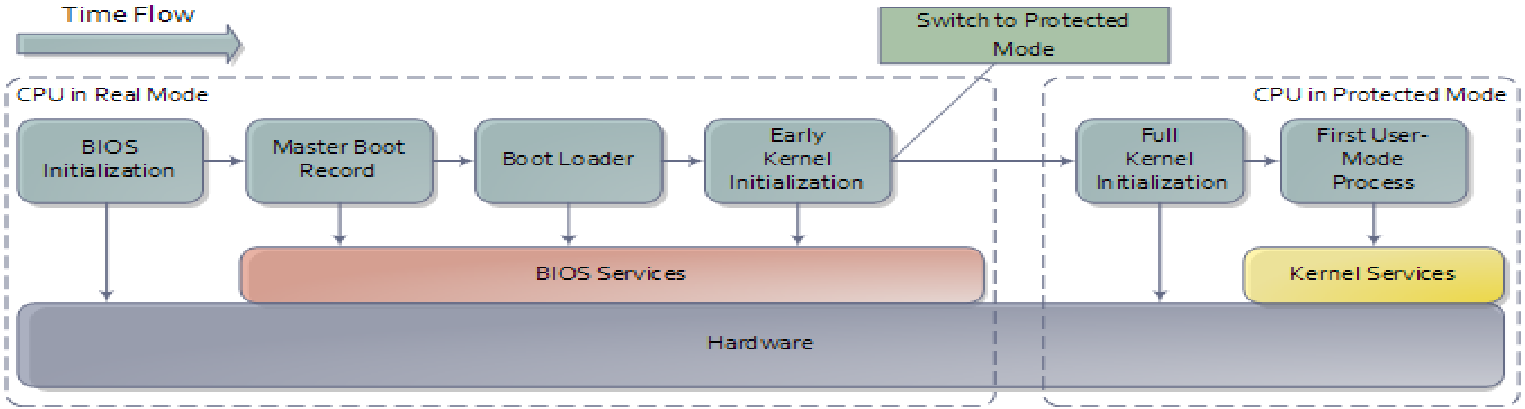
- The kernel initialization is performed (2/2)
  - The rest of the .O.S is set in protected mode
  - The initial processes are built
    - Kernel process, system services and terminals (login)

http://duartes.org/gustavo/blog/post/how-computers-boot-up

Boot process

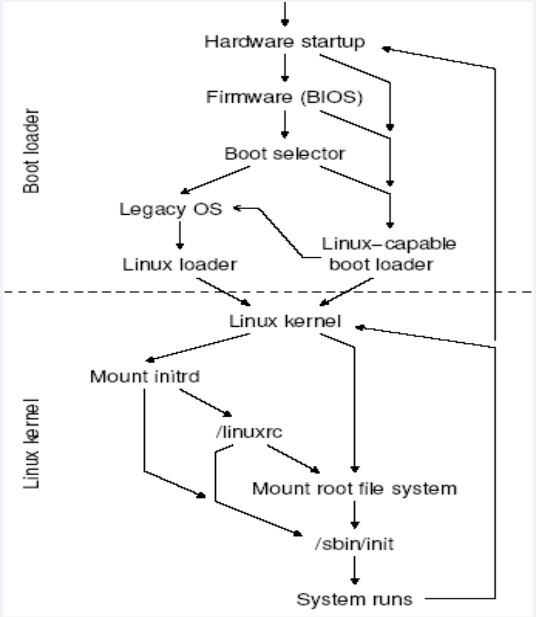
summary

PC



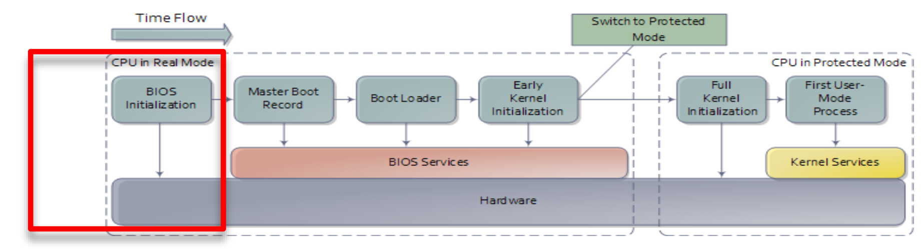
http://duartes.org/gustavo/blog/post/how-computers-boot-up

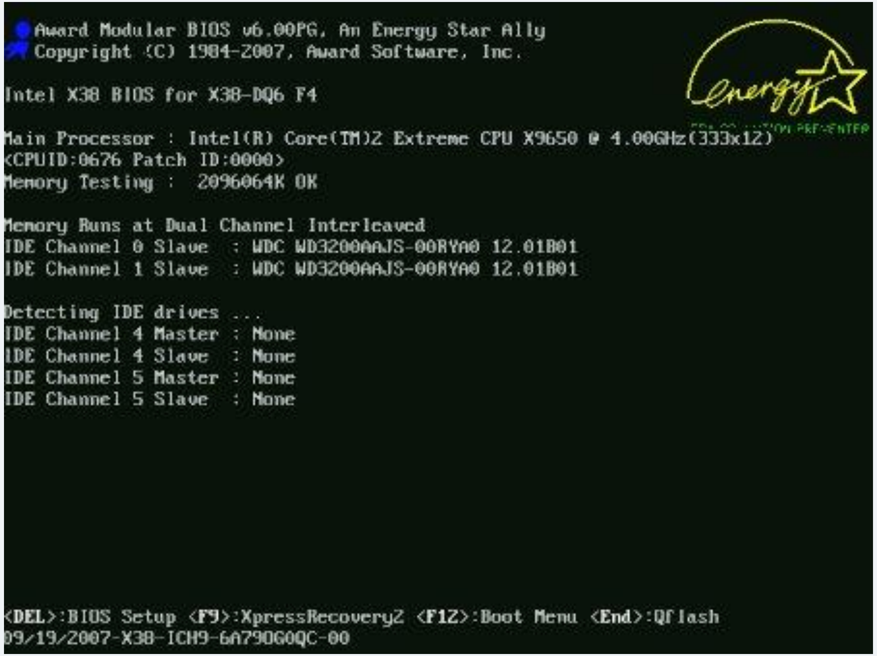
Example of boot sequence



- GNU-Linux

GNU-Linux $_{\text{PC}}$

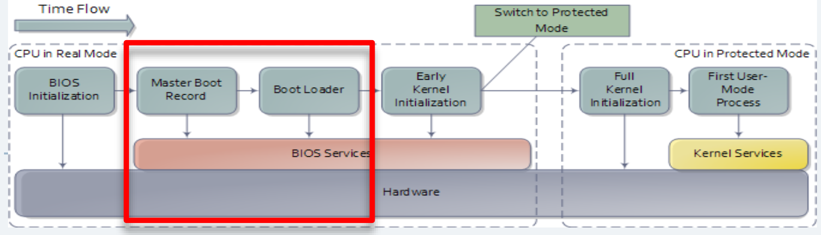


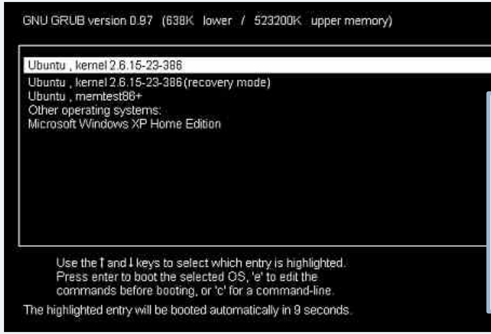


```
Award Modular BIOS v6.00PG, An Energy Star Ally
Copyright (C) 1984-2007, Award Software, Inc.

Intel X38 BIOS for X38-DQ6 F4

Main Processor : Intel(R) Core(TM)2 Extreme CPU X9650 @ 4.00GHz(333x12)
<CPUID:0676 Patch ID:0000>
Memory Testing :  2096064K OK

Memory Runs at Dual Channel Interleaved
IDE Channel 0 Slave   : WDC WD3200AAJS-00RYA0 12.01B01
IDE Channel 1 Slave   : WDC WD3200AAJS-00RYA0 12.01B01

Detecting IDE drives ...
IDE Channel 4 Master  : None
IDE Channel 4 Slave   : None
IDE Channel 5 Master  : None
IDE Channel 5 Slave   : None







<DEL>:BIOS Setup <F9>:XpressRecovery2 <F12>:Boot Menu <End>:Qflash
09/19/2007-X38-ICH9-6A79DG0QC-00
```

http://www.ibm.com/developerworks/linux/library/l-linuxboot/

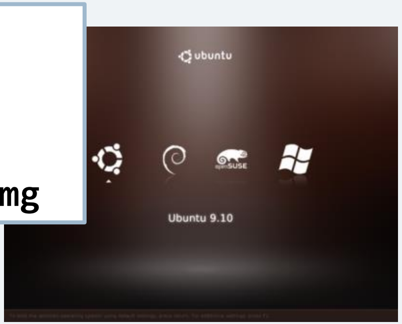
GNU-Linux PC





```
GNU GRUB version 0.97  (638K  lower  /  523200K  upper memory)

 Ubuntu , kernel 2.6.15-23-386
 Ubuntu , kernel 2.6.15-23-386 (recovery mode)
 Ubuntu , memtest86+
 Other operating systems:
 Microsoft Windows XP Home Edition

    Use the ↑ and ↓ keys to select which entry is highlighted.
    Press enter to boot the selected OS, 'e' to edit the
    commands before booting, or 'c' for a command-line.
 The highlighted entry will be booted automatically in 9 seconds.
```

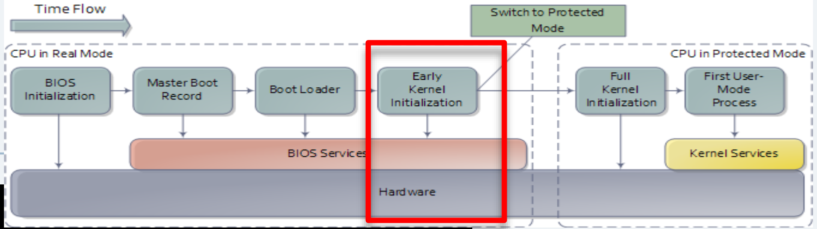
```
grub> set root=(hd0)/boot
grub> insmod linux
grub> linux  /bzImage-2.6.14.2
grub> initrd /initrd-2.6.14.2.img
grub> boot
```

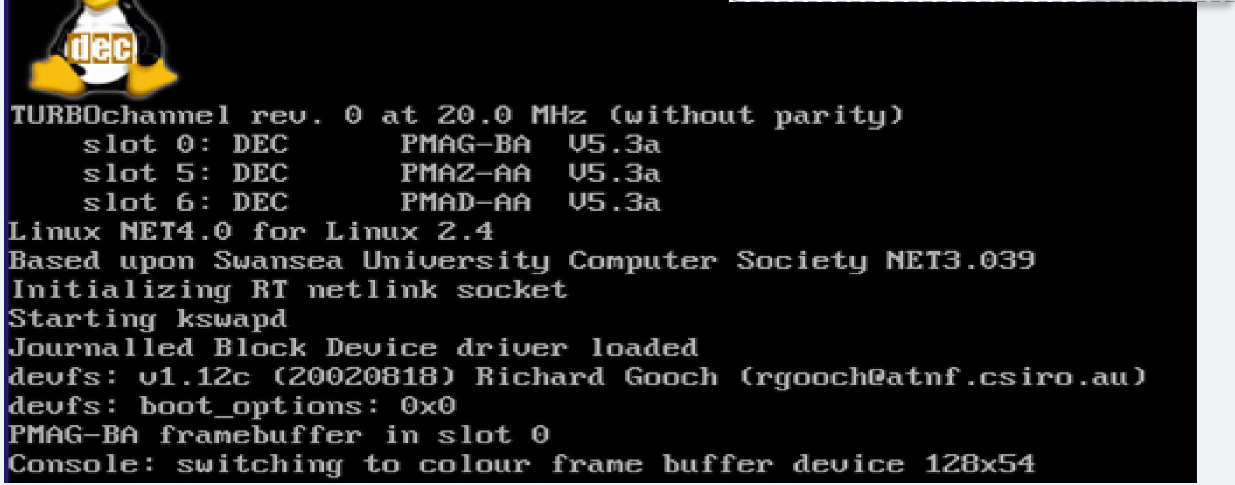


- LILO (*Linux Loader*) or GRUB (*Grand Unified Bootloader*).
  - It shows an option menu (/etc/grub.conf)
  - The kernel image is loaded into memory (vmlinuz) and it is executed with the parameters of the selected menu option.
  - It is also possible to "chain" the bootloader (with other one).

http://funnix.net/wp-content/uploads/2012/07/grub.jpg

GNU-Linux $_{\text{PC}}$



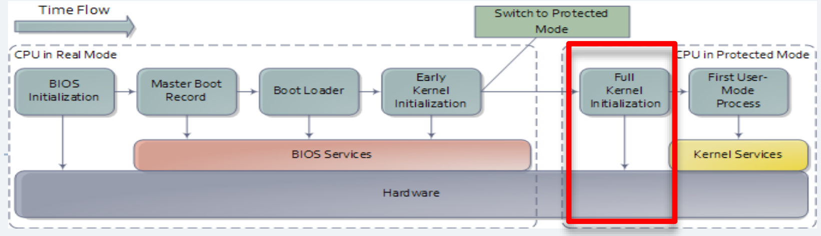


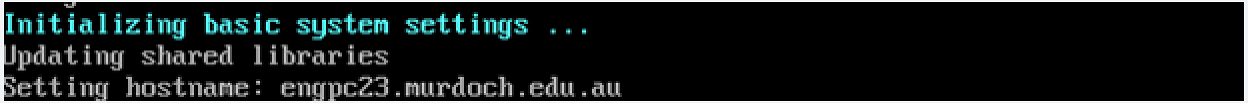
```
TURBOchannel rev. 0 at 20.0 MHz (without parity)
    slot 0: DEC       PMAG-BA  V5.3a
    slot 5: DEC       PMAZ-AA  V5.3a
    slot 6: DEC       PMAD-AA  V5.3a
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
Starting kswapd
Journalled Block Device driver loaded
devfs: v1.12c (20020818) Richard Gooch (rgooch@atnf.csiro.au)
devfs: boot_options: 0x0
PMAG-BA framebuffer in slot 0
Console: switching to colour frame buffer device 128x54
```

- The kernel is executed (vmlinuz): base
  - If needed, the kernel is uncompressed
  - The hardware plug-and-play is done (and the associated kernel drivers are initialized)

http://gxemul.sourceforge.net/gxemul-stable/doc/debian-1.png

GNU-Linux $_{\text{PC}}$



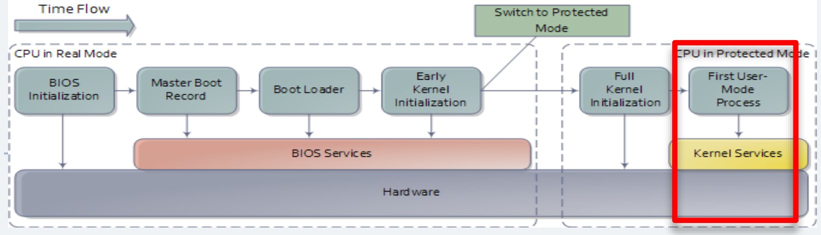


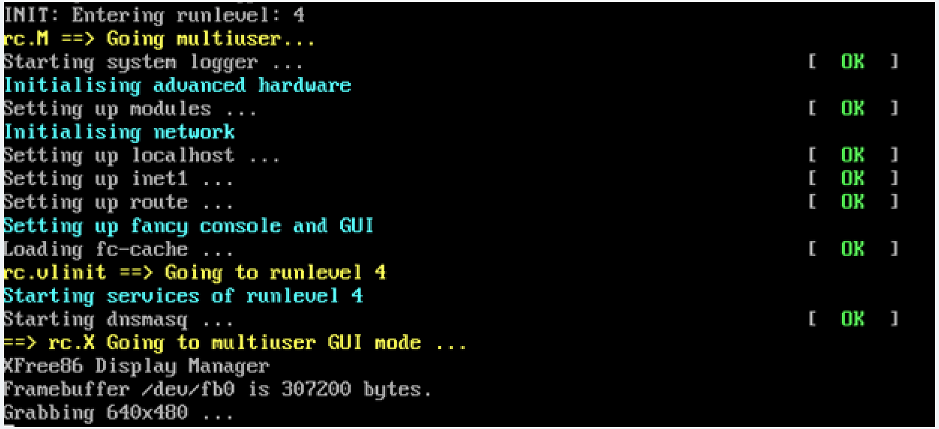
```
Initializing basic system settings ...
Updating shared libraries
Setting hostname: engpc23.murdoch.edu.au
```

- The kernel is executed (initrd): modules
  - initrd is the initial system with the necessary drivers to fully boot.
  - The shell-script /linuxrc is executed
    - It initializes the drivers with the associated configuration.
  - The initrd use to ‘pivot’ to the planned root system:
    - Itself (embedded systems), partition in the hard disk, NFS, etc.

http://milindchoudhary.wordpress.com/2009/03/30/linux-boot-process/

GNU-Linux PC





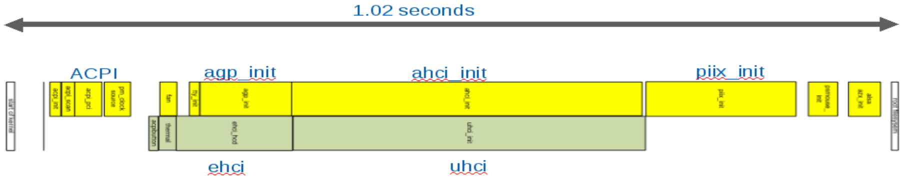
```
INIT: Entering runlevel: 4
rc.M ==> Going multiuser...
Starting system logger ...                                  [  OK  ]
Initialising advanced hardware
Setting up modules ...                                      [  OK  ]
Initialising network
Setting up localhost ...                                    [  OK  ]
Setting up inet1 ...                                        [  OK  ]
Setting up route ...                                        [  OK  ]
Setting up fancy console and GUI
Loading fc-cache ...                                        [  OK  ]
rc.vlinit ==> Going to runlevel 4
Starting services of runlevel 4
Starting dnsmasq ...                                        [  OK  ]
==> rc.X Going to multiuser GUI mode ...
XFree86 Display Manager
Framebuffer /dev/fb0 is 307200 bytes.
Grabbing 640x480 ...
```

- The init process is executed
  - The init process (pid 1) boots all system process…
  - … and the terminal process (*login* o *xlogin*) in order user could authenticate.
  - It goes sleep waiting for the arrival of events (cpu_idle)

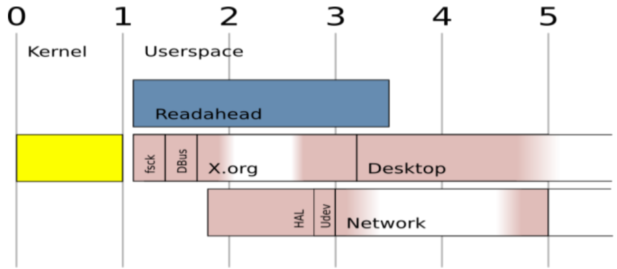
http://www.ibm.com/developerworks/linux/library/l-linuxboot/

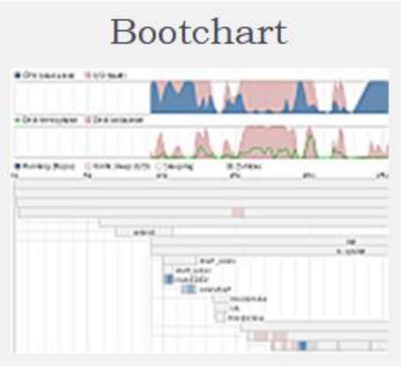
Speed-up the Linux boot

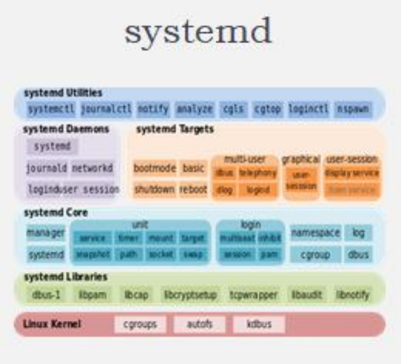
- Asynchronous hardware initialization



- Asynchronous initialization of services

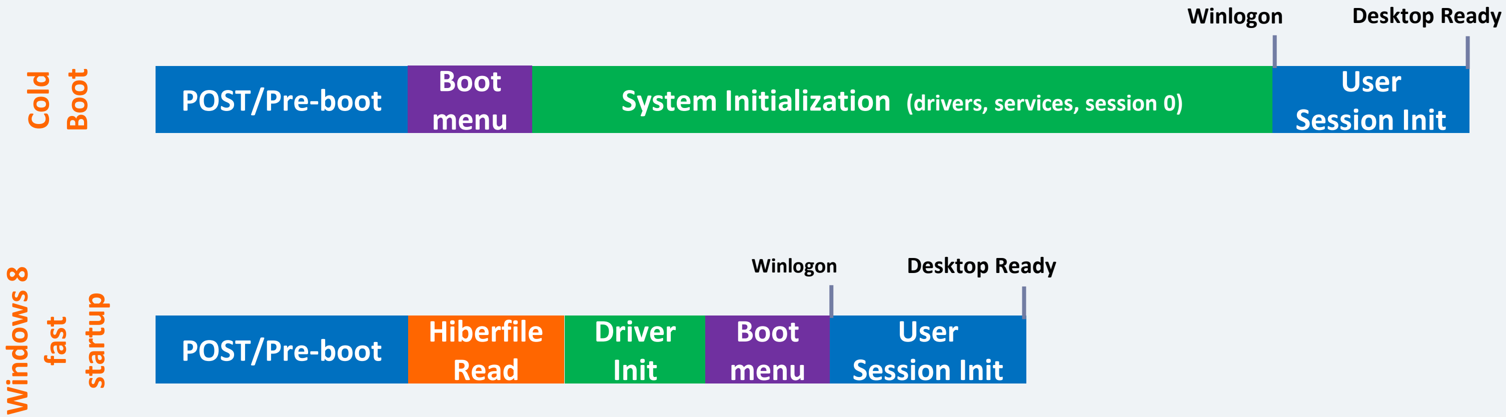






http://www.digitaltrends, http://lwn.net/Articles/299483, https://es.wikipedia.org/wiki/Systemd

Speed-up the “Windows 8" boot

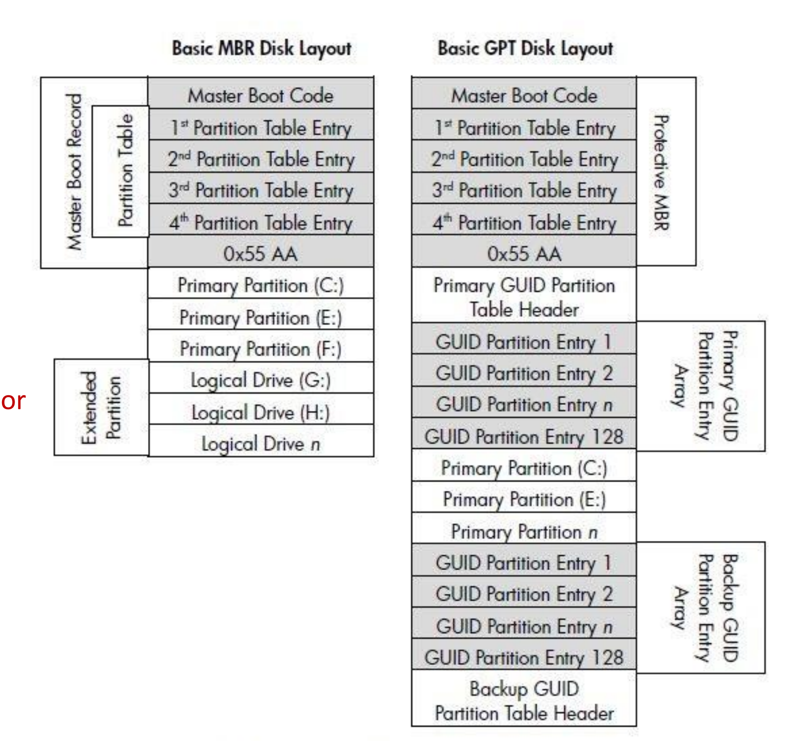


http://www.digitaltrends.com/computing/windows-8-boot-time-scaled-down-to-eight-seconds/

MBR → GPT

Master Boot Record

- 4 primary part.
  3P. + 1E. (+n U.L.)
- 32 bits
- 2 TB/part.
  2^{32}*512 bytes/sector
- BIOS
- Old O.S.
- 1 MBR +
  no CRC32



Basic MBR Disk Layout

Master Boot Record
Partition Table
Master Boot Code
1st Partition Table Entry
2nd Partition Table Entry
3rd Partition Table Entry
4th Partition Table Entry
0x55 AA
Primary Partition (C:)
Primary Partition (E:)
Primary Partition (F:)
Extended Partition
Logical Drive (G:)
Logical Drive (H:)
Logical Drive *n*

Basic GPT Disk Layout

Protective MBR
Master Boot Code
1st Partition Table Entry
2nd Partition Table Entry
3rd Partition Table Entry
4th Partition Table Entry
0x55 AA
Primary GUID Partition Table Header
Primary GUID Partition Entry Array
GUID Partition Entry 1
GUID Partition Entry 2
GUID Partition Entry *n*
GUID Partition Entry 128
Primary Partition (C:)
Primary Partition (E:)
Primary Partition *n*
Backup GUID Partition Entry Array
GUID Partition Entry 1
GUID Partition Entry 2
GUID Partition Entry *n*
GUID Partition Entry 128
Backup GUID Partition Table Header

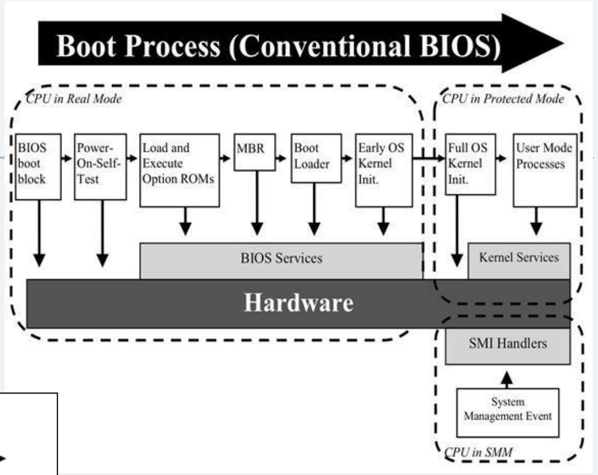
Comparison of MBR and GPT disk layouts

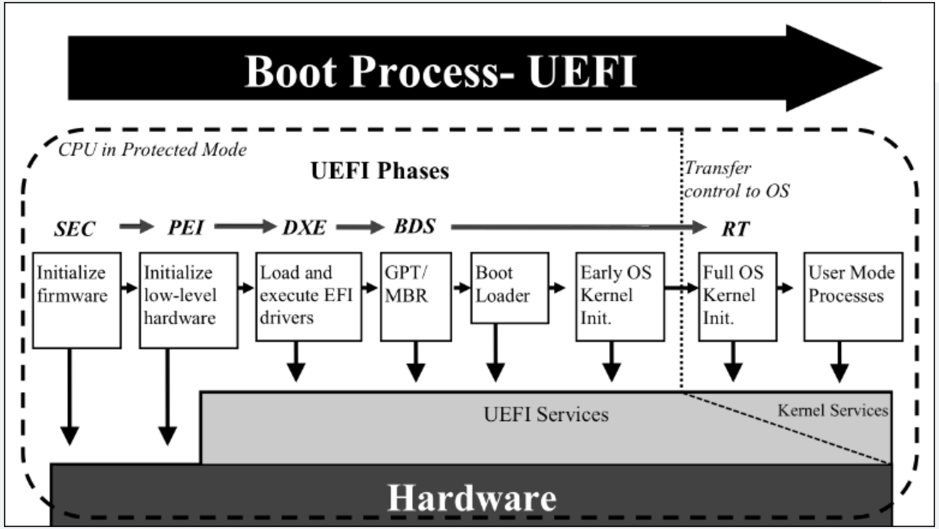
GUID Partition Table

- 128 part.
  128 in several O.S.
- 64 bits
- 9 ZB/part.
  2^{64}*512 bytes/sector
- UEFI
- New S.O.
- 2 GPT +
  CRC32
  more secure

http://www.it-support-singapore.com/wp-content/uploads/2012/08/differenace-MBR-GPT1.jpg

BIOS → UEFI



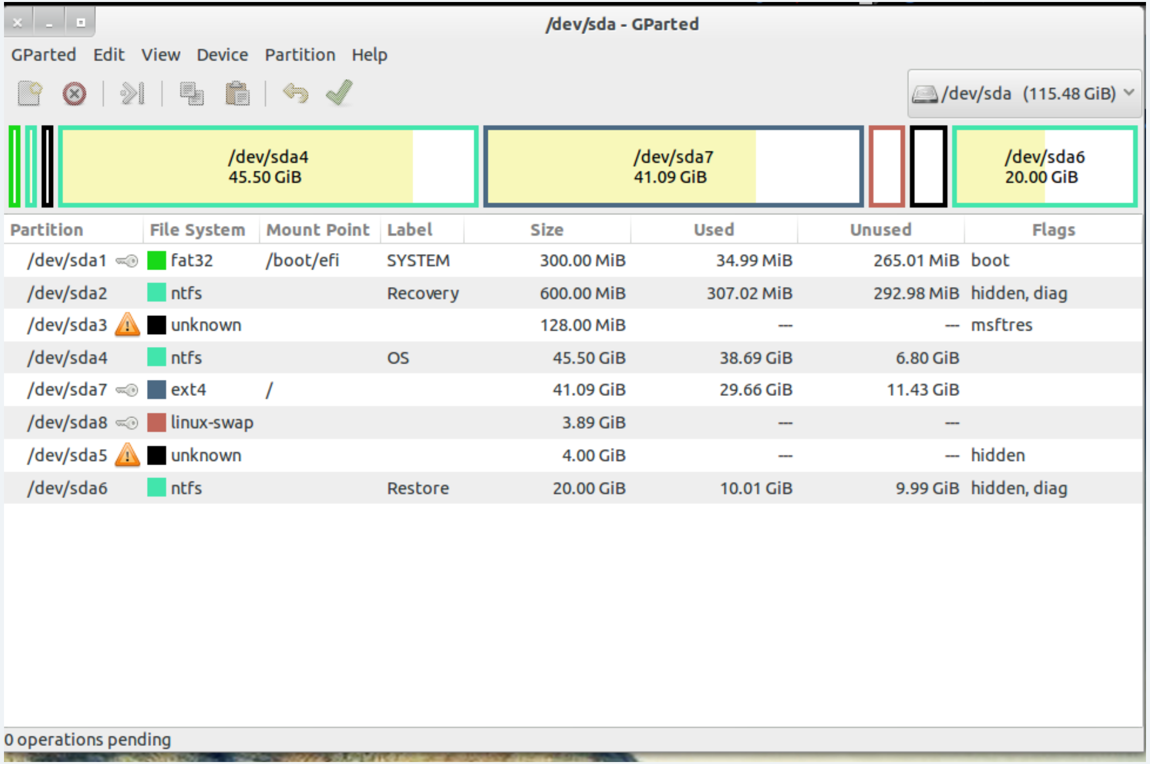


http://answers.microsoft.com/en-us/windows/forum/windows8_1-security/uefi-secure-boot-in-windows-81/65d74e19-9572-4a91-85aa-57fa783f0759

GPT + UEFI

Example of mandatory partitions with dual-boot

EFI
W-recovery
W-MSR
W-system
L-system

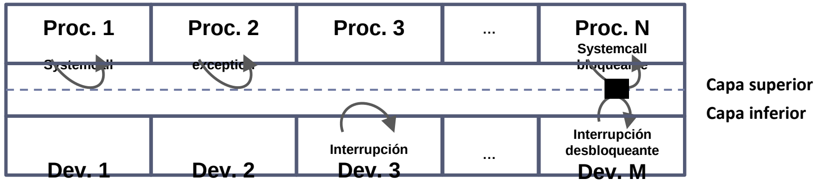


Partition	File System	Mount Point	Label	Size	Used	Unused	Flags
/dev/sda1 🔑	🟩 fat32	/boot/efi	SYSTEM	300.00 MiB	34.99 MiB	265.01 MiB	boot
/dev/sda2	🟩 ntfs		Recovery	600.00 MiB	307.02 MiB	292.98 MiB	hidden, diag
/dev/sda3 ⚠	⬛ unknown			128.00 MiB	—	—	msftres
/dev/sda4	🟩 ntfs		OS	45.50 GiB	38.69 GiB	6.80 GiB	
/dev/sda7 🔑	🟦 ext4	/		41.09 GiB	29.66 GiB	11.43 GiB	
/dev/sda8 🔑	🟥 linux-swap			3.89 GiB	—	—	
/dev/sda5 ⚠	⬛ unknown			4.00 GiB	—	—	hidden
/dev/sda6	🟩 ntfs		Restore	20.00 GiB	10.01 GiB	9.99 GiB	hidden, diag

0 operations pending

http://askubuntu.com/questions/350352/making-windows-8-partition-larger

Overview



- Introduction
- **How an operating system works**
  - System boot
  - **Characteristics and event handling**
  - Kernel process
- Other aspects
  - Events concurrency
  - Add new system functionalities

Event types

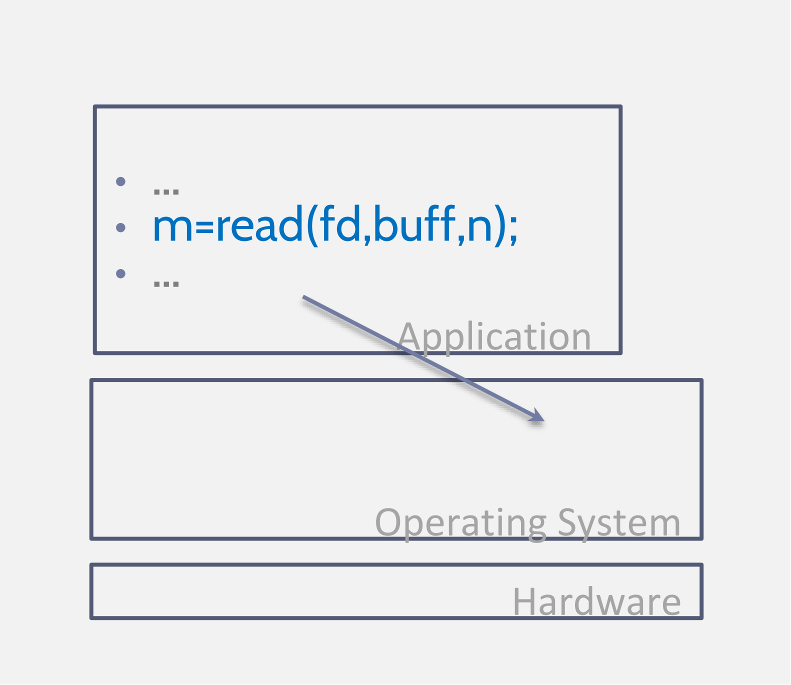
User

- System calls
  - Event for requesting an operating system service
- Exceptions
  - Exceptional events while executing an instruction
- Software interrupts
  - Deferred event as part of a pending event treatment
- Hardware interrupts
  - Events that come from hardware.

Hardware

Event types

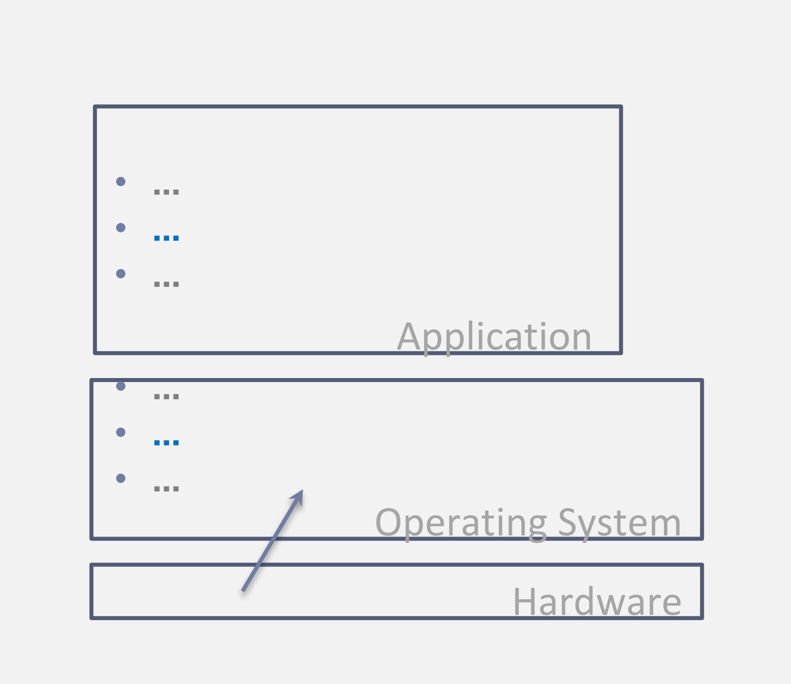
System calls



- Event for requesting an O.S. service.
- User programs access to O.S. services through system calls.
- They are seen by programmers as function calls.

Event types

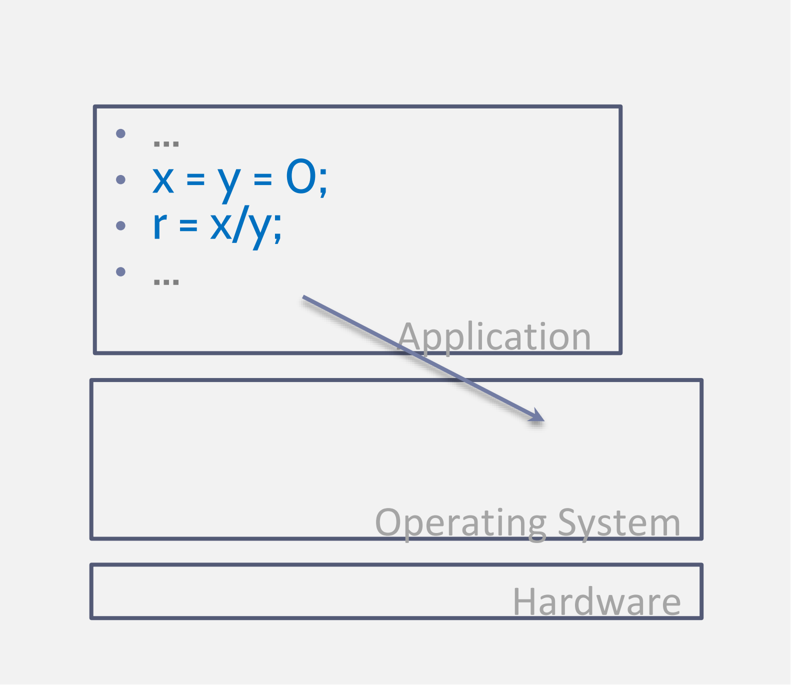
Hardware interrupts



- Events that come from hardware.
- The O.S. has to attend to something that the hardware needs (data arrival, exceptional situation, etc.)
- It requires a set of subroutines associated with each event that the hardware can request.

Event types

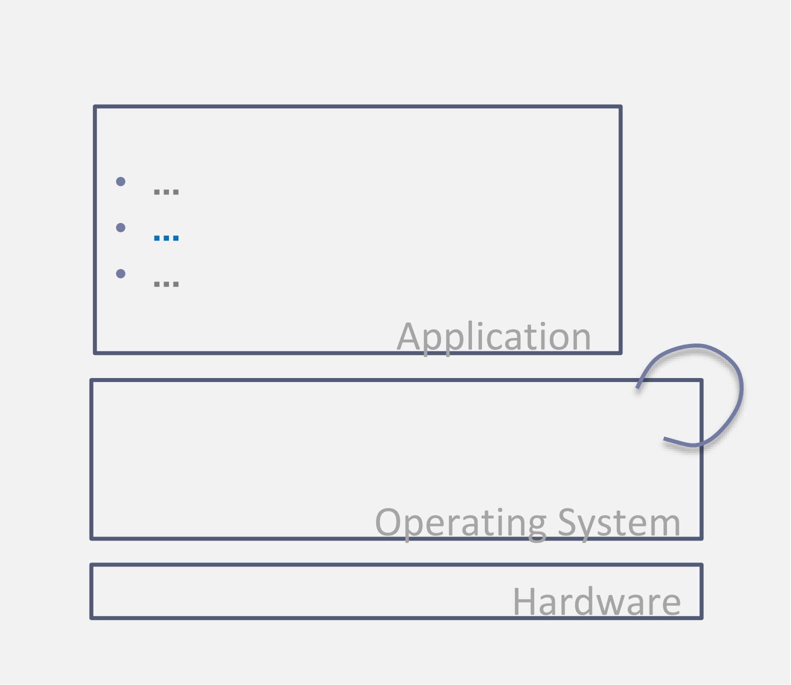
Exceptions



- Exceptional events while executing an instruction.
- They can be problems (division by zero, illegal instruction, segment violation, etc.) or warnings (page failure, etc.)
  - ~ Hardware interruption generated by the CPU itself.
- It requires a set of subroutines associated with each exception that may occur.

Event types

Software interrupts



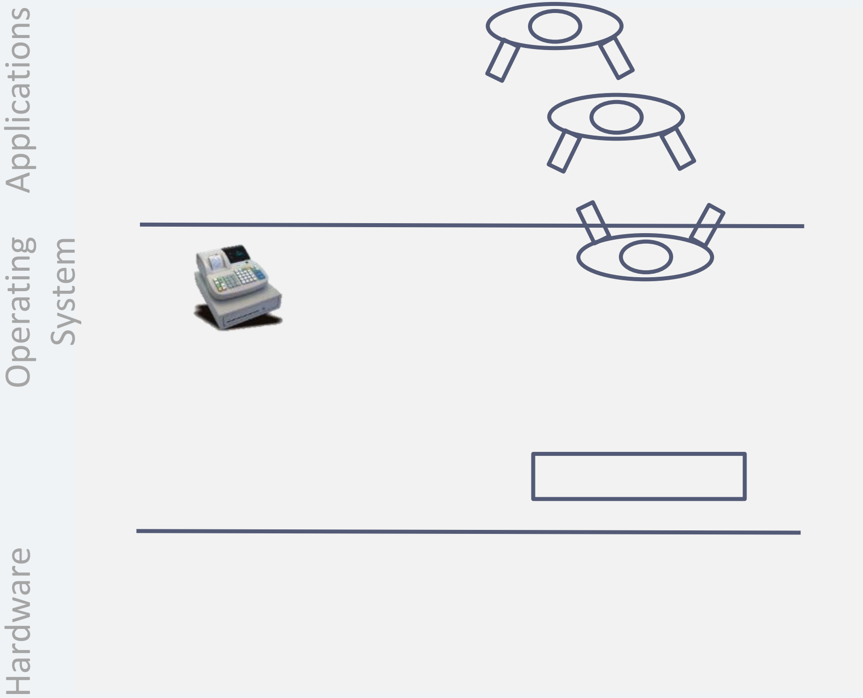
- Event to deferre the non-critical part of the event treatment.
- Part of the event treatment is deferred:
  - To wait better opportunity.
  - Treated most urgent events first.

Metaphor: the book store…

- Hardware devices
  providers.
  - Computer
    the book store.
  - CPU and RAM
    seller and shelves.
- Operating System
  Instruction book the seller follows.
- Process
  Buyers.

Metaphor: the book store…

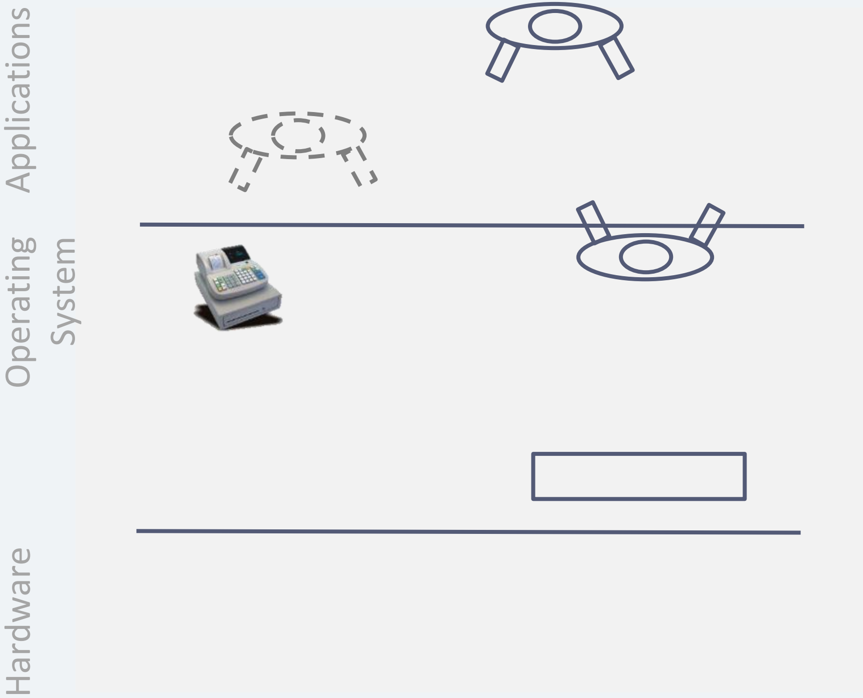
System call



- Buyer want to buy a book
- The process issues a system call
- Seller request the book to the associated provider (because out of stock)
- The O.S. issues a disk request for a data block
- Seller puts the buyer on hold until he has the book to attend to other situations
- The O.S. block the process and execute another process or pending tasks

Metaphor: the book store…

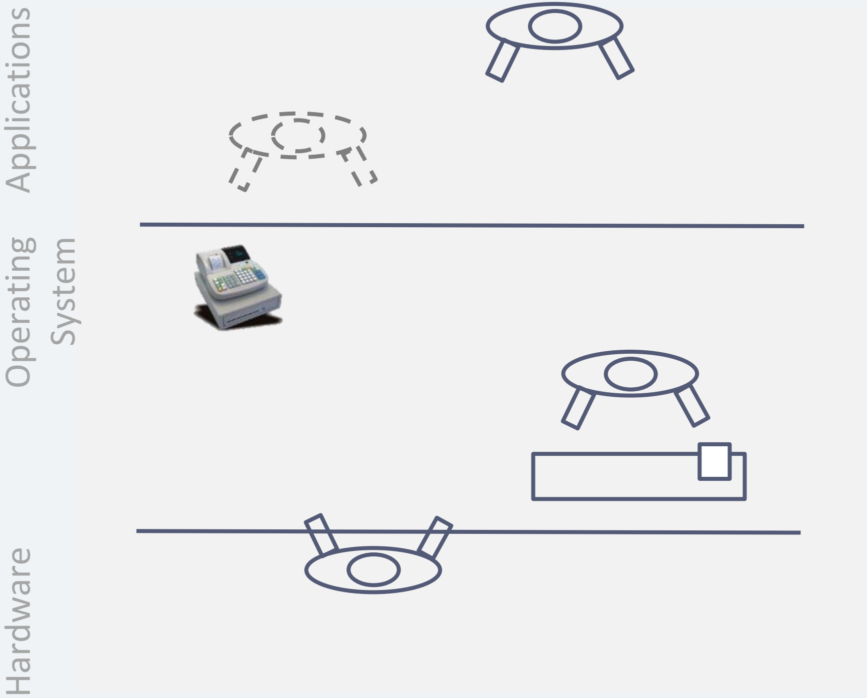
System call



- Buyer want to buy a book
- The process issues a system call
- Seller request the book to the associated provider (because out of stock)
- The O.S. issues a disk request for a data block
- Seller puts the buyer on hold until he has the book to attend to other situations
- The O.S. block the process and execute another process or pending tasks

Metaphor: the book store…

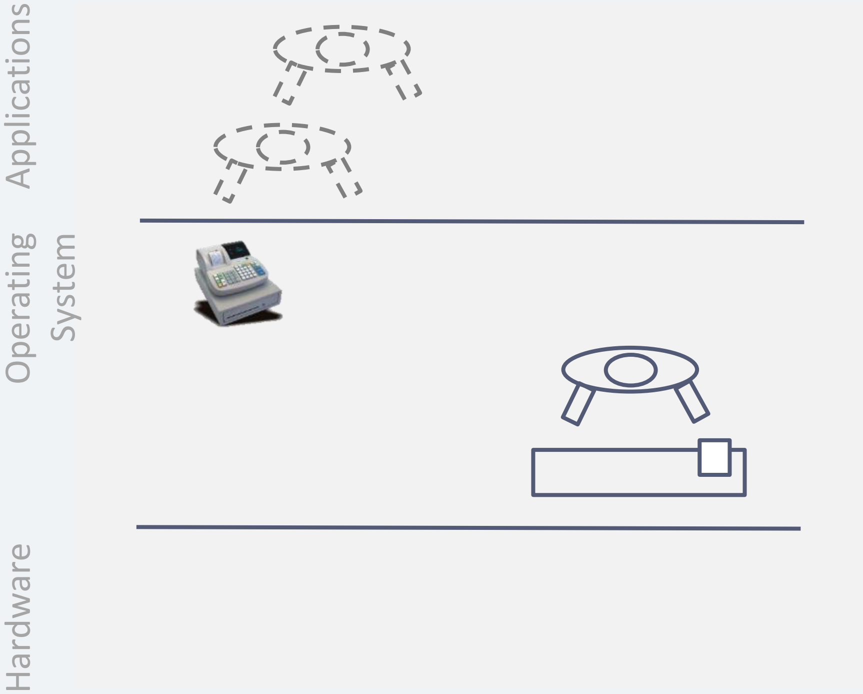
Hardware interrupt



- The provider notifies by phone that he/she is at the door and he/she needs urgent attention (because he/she double parked)
- Hard disk fire a hardware interrupt
- Seller put the book boxes into a temporary shelf, along with a post-it that labels it as 'todo: to deliver'
- The O.S. copies the disk block into memory and activates a software interrupt

Metaphor: the book store…

Software interrupt

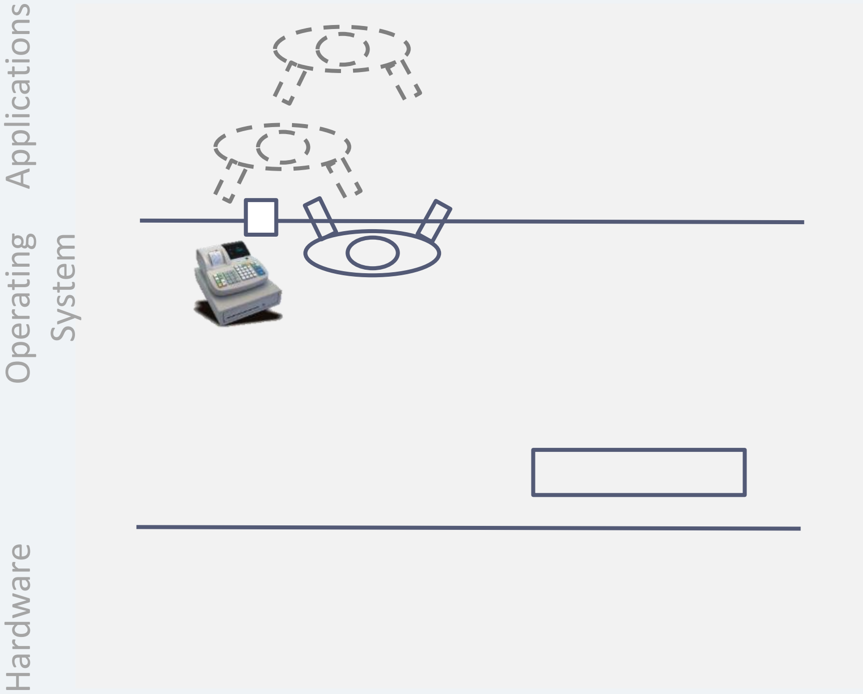


- When no other priority task is pending, the "todo" tasks is done
- If there is no any priority event pending, software interrupts are attended

- For each pending item to be delivered, buyer is notified that can pick it up
- O.S. changes the process state to “ready”, and when it is executed it will copy the data

Metaphor: the book store…

Software interrupt

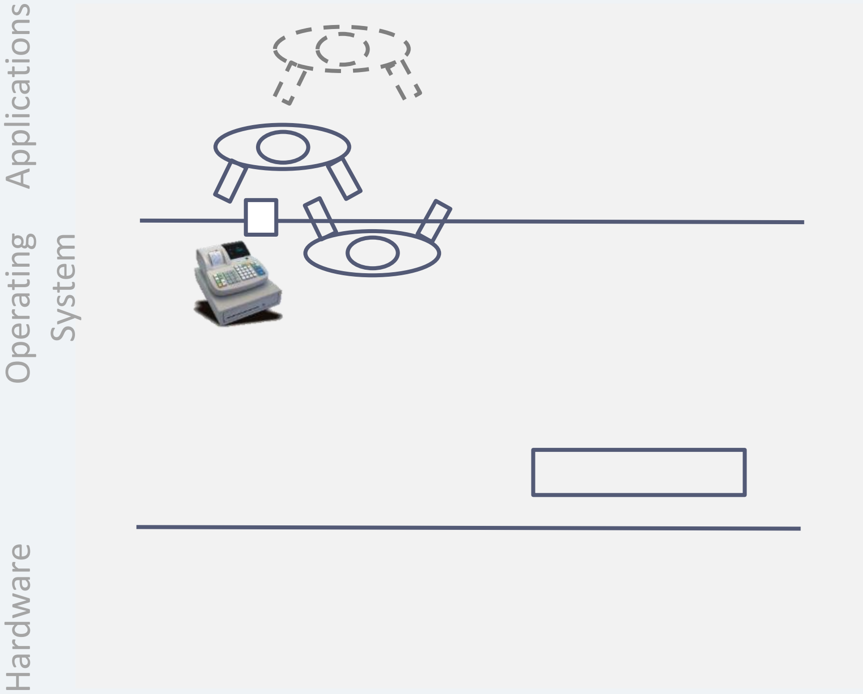


- When no other priority task is pending, the "todo" tasks is done
- If there is no any priority event pending, software interrupts are attended

- For each pending item to be delivered, buyer is notified that can pick it up
- O.S. changes the process state to “ready”, and when it is executed it will copy the data

Metaphor: the book store…

Software interrupt

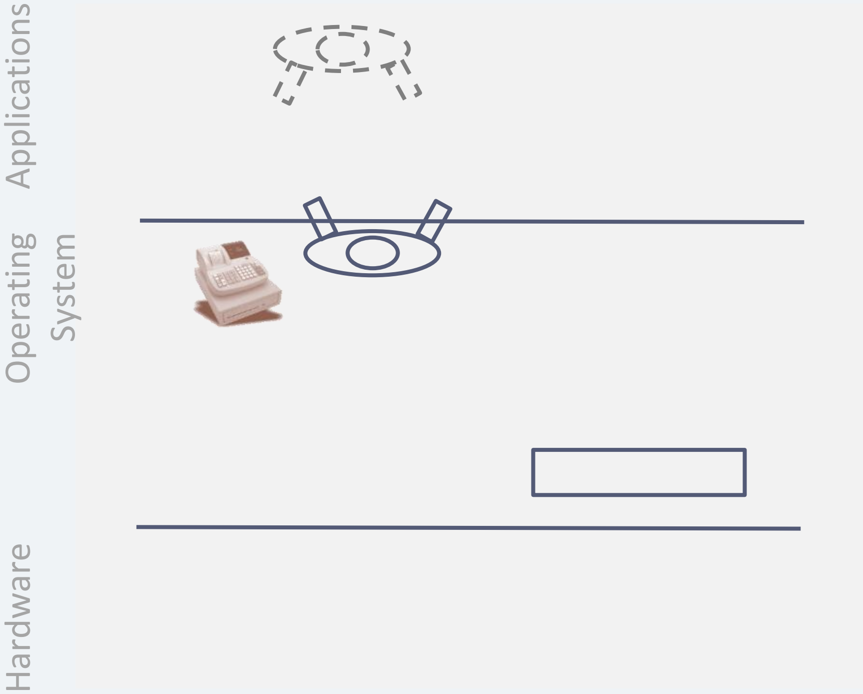


- When no other priority task is pending, the "todo" tasks is done
- If there is no any priority event pending, software interrupts are attended

- For each pending item to be delivered, buyer is notified that can pick it up
- O.S. changes the process state to “ready”, and when it is executed it will copy the data

Metaphor: the book store…

Exception



- If a buyer ask for a coffee, is invited to leave the bookstore (and go to a cafeteria). Then, seller continues serving clients.
- An exception occurs while a process is running, the process is killed
- If the cash register is broken, then the bookstore must be closed
- A serious exception occurs while running the operating system, kernel-panic and stops

Simplified example

App.

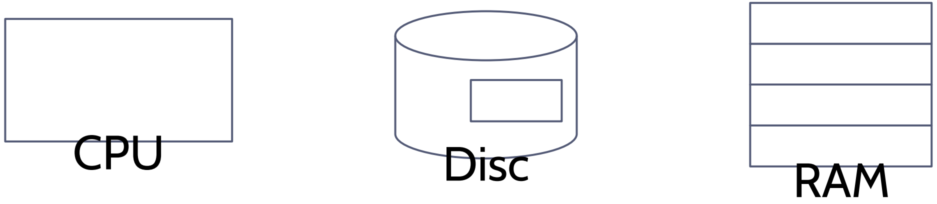
- char buffer[1024];

…

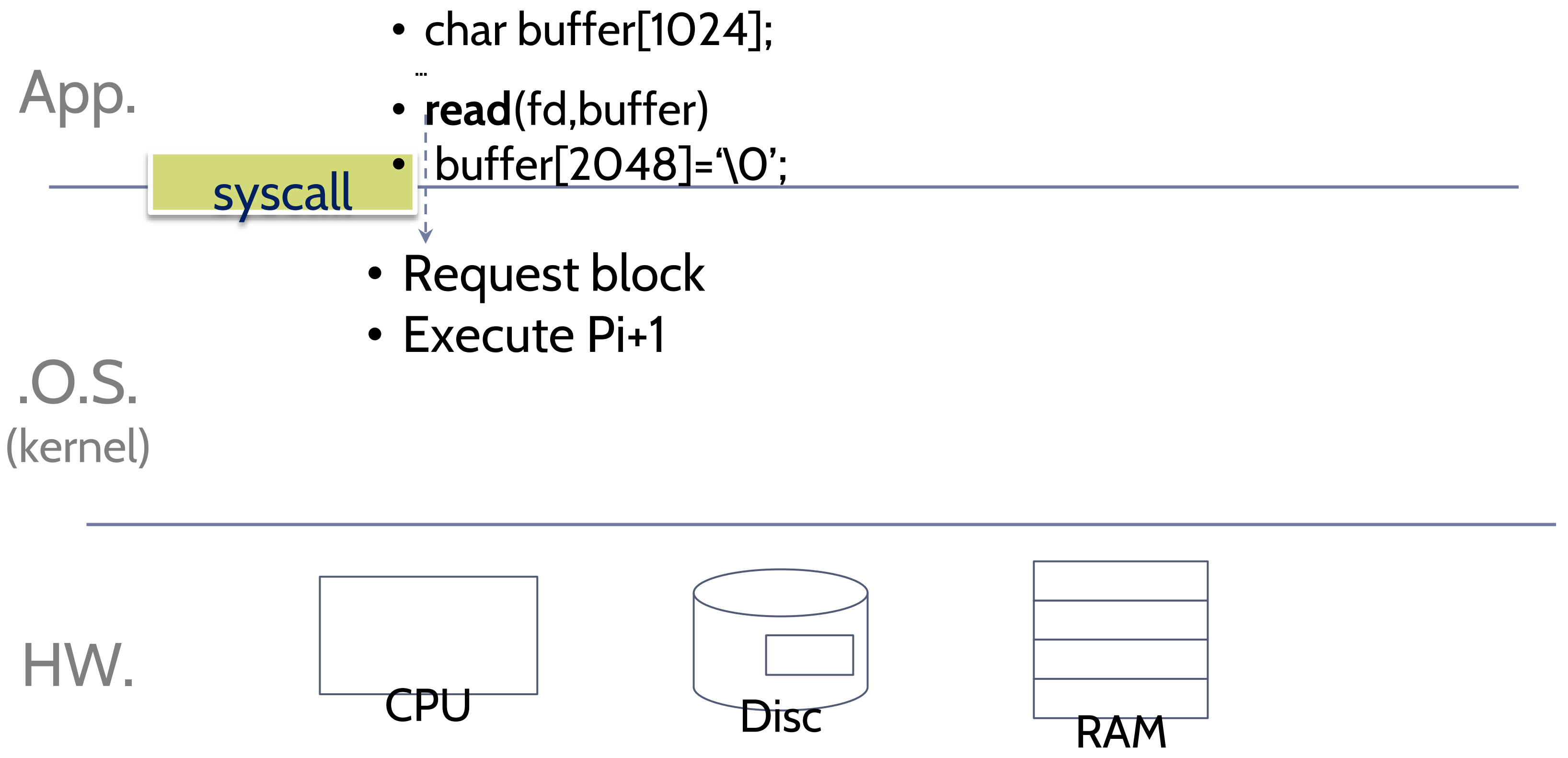
- read(fd,buffer)
- buffer[2048]='\0';

.O.S.
(kernel)

HW.



Simplified example



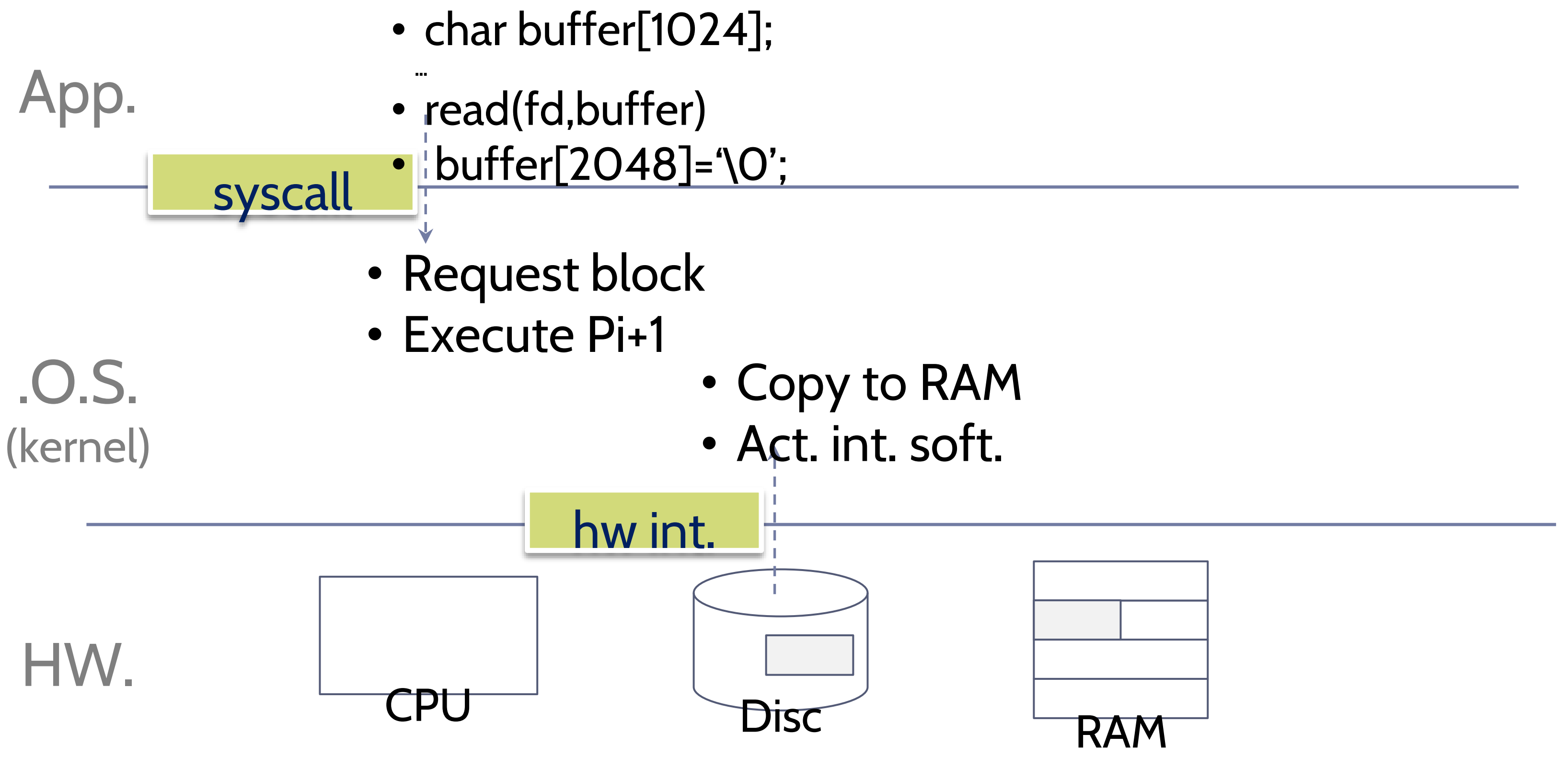
- char buffer[1024];

…

- **read**(fd,buffer)
- buffer[2048]='\0';

- Request block
- Execute Pi+1

Simplified example



App.

- char buffer[1024];

…

- read(fd,buffer)
- buffer[2048]='\0';

syscall

- Request block
- Execute Pi+1

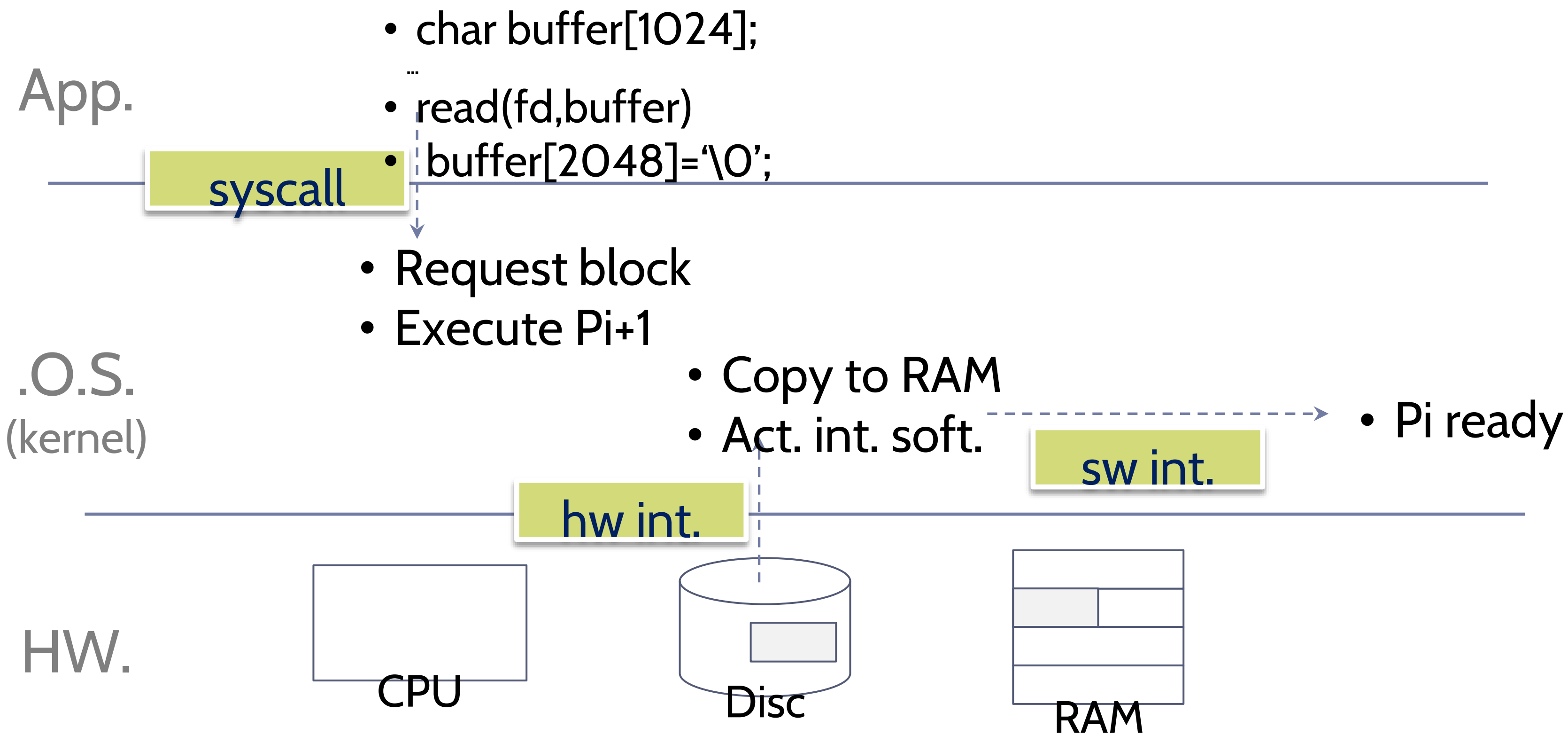
.O.S.
(kernel)

- Copy to RAM
- Act. int. soft.

hw int.

HW.

Simplified example



- char buffer[1024];

…

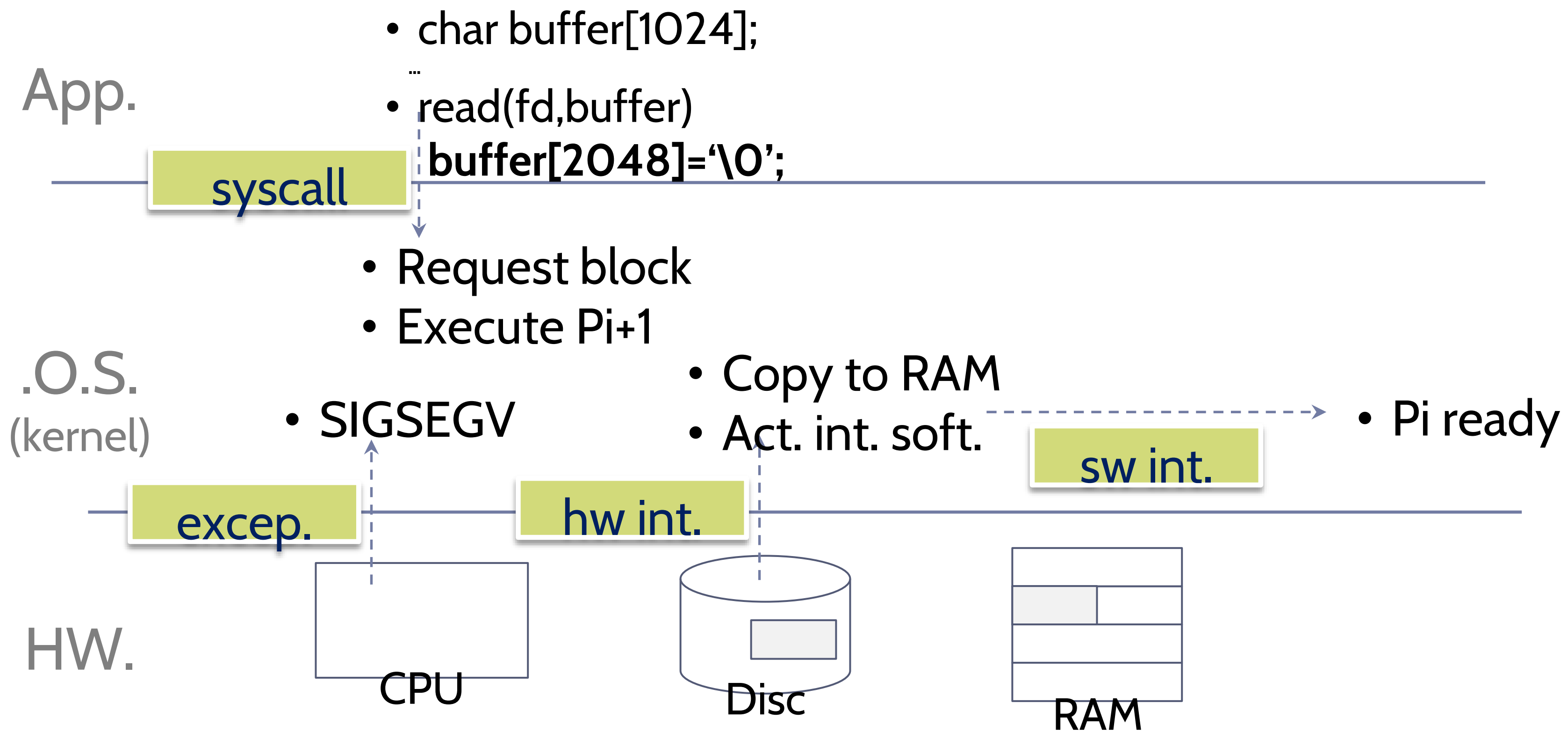
- read(fd,buffer)
- buffer[2048]=‘\0’;

- Request block
- Execute Pi+1

- Copy to RAM
- Act. int. soft.

- Pi ready

Simplified example



App.

- char buffer[1024];

...

- read(fd,buffer)

buffer[2048]='\0';

syscall

.O.S.
(kernel)

- Request block
- Execute Pi+1

- SIGSEGV

- Copy to RAM
- Act. int. soft.

- Pi ready

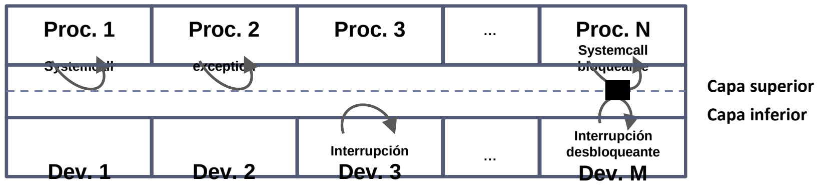
excep.

hw int.

sw int.

HW.

Overview



- Introduction
- How an operating system works
  - System boot
  - **Characteristics and event handling**
  - Kernel process
- Other aspects
  - Events concurrency
  - Add new system functionalities

Classification of events

Hardware interrupts

System calls

Software interrupts

Exceptions

	Synchronous	Asynchronous
Hardware		
Software		

Classification of events

	Synchronous	Asynchronous
Hardware	Exceptions	Hardware interrupts
Software	System calls	Software interrupts

- Generated by software o hardware:
  - Generated by hardware
    - Hardware provides the request and the associated vector
  - Generated by software
    - An assembly instruction provides the request and the associated vector

Classification of events

	Synchronous	Asynchronous
Hardware	Exceptions	Hardware interrupts
Software	System calls	Software interrupts

- Synchronous and asynchronous events:
  - Synchronous events
    - It activation is predictable, and related to the actual process' code
    - Executed in the context of the "requested" process
  - Asynchronous events
    - It activation is unpredictable, and related to any (or none) process
    - Executed in the context of of a process not related with the interrupt

Basic characteristics…

User System Device .O.S. C.P.U. Applications

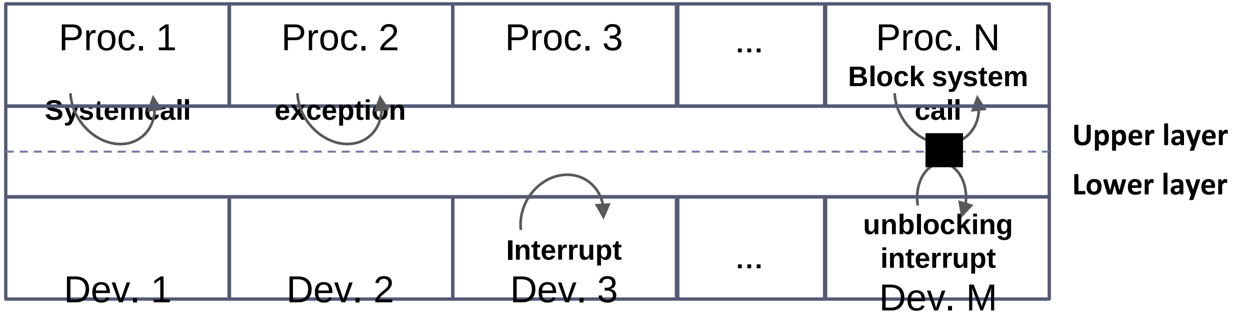
	Previous execution mode	Generated by
Hardware interrupts		
Exceptions		
System calls		
Software interrupts		

Basic characteristics…

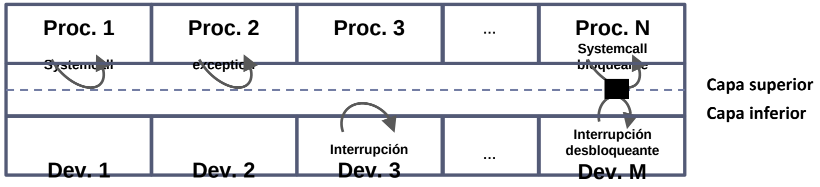
	Previous execution mode	Generated by
Hardware interrupts	• It can be **User or System** ◦ **NO,** it doesn't influences in treatment	• I/O Devices • Interrupts among CPUs (IPI)
Exceptions	• It can be **User or System** ◦ **YES**. it influences in the treatment	• CPU itself (~hw int.. from CPU) ◦ Usually programming errors, NO always (page faults, debugging, etc.)
System calls	• Always **User**	• Applications
Software interrupts	• Always **System**	• Because the treatment of all other events: used by the non-critical parts

Relationship between events

- Components that treats synchronous events
  - More related with process
- Components that treats asynchronous events
  - More related with Devices
- There are tasks that involves both event types.
  - E.g.: access to a disk (system call + disk interrupt)



Overview



- Introduction
- **How an operating system works**
  - System boot
  - Characteristics and event handling
  - Kernel process
- Other aspects
  - Events concurrency
  - Add new system functionalities

Event management

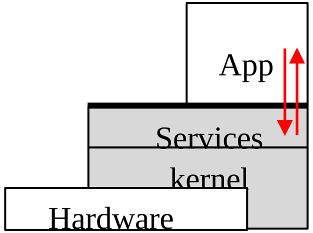
- O.S. event mgm. use to be generic and hardware-architecture agnostic
  - Linux without priority (SPARC has support) and Windows with priority (Intel doesn’t has support)

Event management

- O.S. event mgm. use to be generic and hardware-architecture agnostic
  - Linux without priority (SPARC has support) and Windows with priority (Intel doesn’t has support)
- All events are treated in a similar way (~hw int..)
  - It has been introduced its event management

Event management

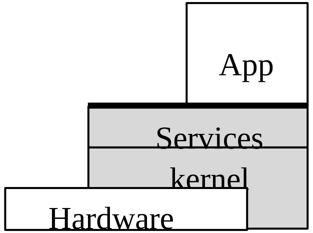
- O.S. event mgm. use to be generic and hardware-architecture agnostic
  - Linux without priority (SPARC has support) and Windows with priority (Intel doesn’t has support)
- All events are treated in a similar way (~hw int..)
  - It has been introduced its event management



- It is saved the state in the system stack
  - Usually the PC and SR (state) registers
- CPU switch into privilegiate mode and jump into the assoc. treatment subroutine
  - Save extra registers if necessary
  - The event handler subroutine treats the event
  - Restore extra registers saved if necessary
- The event handler subroutine ends: RETI
  - Restore the saved state and PC and restore the previous mode

Event management

- Detail 1 > During the boot sequence, no event is handled
  - System mode, disabled interrupts, and inactive MMU
- Detail 2 > Cuando ocurre un evento, entra el S.O para tratarlo:
  - There is a mode switching (into privilegiate mode)
  - but is not mandatory to perform a context switching



- The event is handled in the context of the active process.
- Current active process memory map is used, even though is not related with the event handled.
- The system uses to independent stacks:
  - User stack (user mode) or System Stack (system)

- Detail 3 > An event could be 'fired' while treating other event
  - prioritary event -> push current in a stack and treat the new one;
    otherwise -> wait to end the current treatment to perform the new event's treatment

Event management

- Hardware interrupts:
  - General treatment
  - Examples: W & L
- Exception:
  - General treatment
- System calls:
  - General treatment
  - Examples: W & L
- Software interrupts:
  - General treatment
  - Examples: W & L

Hardware interrupts

characteristics

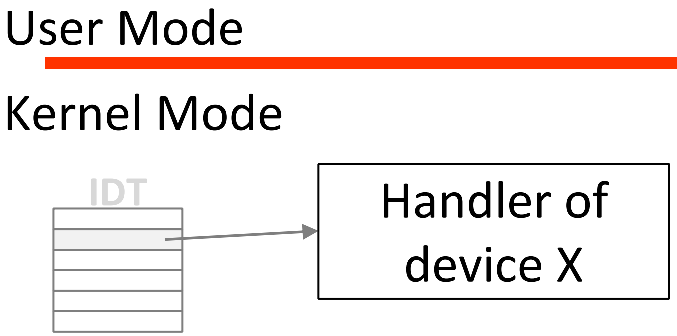
- Asynchronous events that comes from the hardware to notify C.P.U. to handle it

- Previous execution mode:
  - It could be user or system (it does not influences the treatment)

- Generated by:
  - I/O devices
  - System critical conditions (e.g.: power shortage)
  - Inter-processor Interrupts (IPI)

Hardware interrupts

treatment (1/5)



User Mode

Kernel Mode

IDT

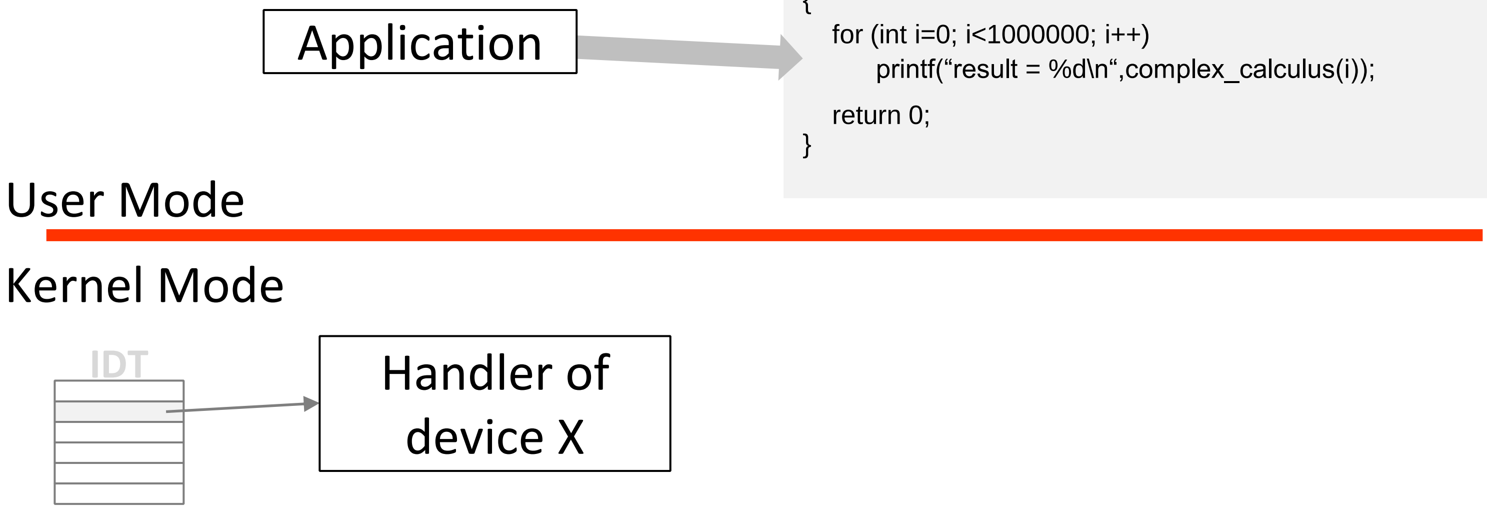
Handler of device X

```
int main (int argc, char **argv)
{
      ...
      /* instalar los manejadores para los vectores de interrupción */
      instal_man_int(EXC_ARITMETICA,   hnd_exceptionAritmetica);
      instal_man_int(EXC_MEMORIA,      hnd_exceptionMemory);
      instal_man_int(INT_RELOJ,        hnd_interruptClock);
      instal_man_int(INT_DeviceS, hnd_interruptDevices);
      instal_man_int(LLAM_SISTEMA,     hnd_SystemCall);
      instal_man_int(INT_SW,           hnd_softwareInterrupt);
      ...
```

Hardware interrupts

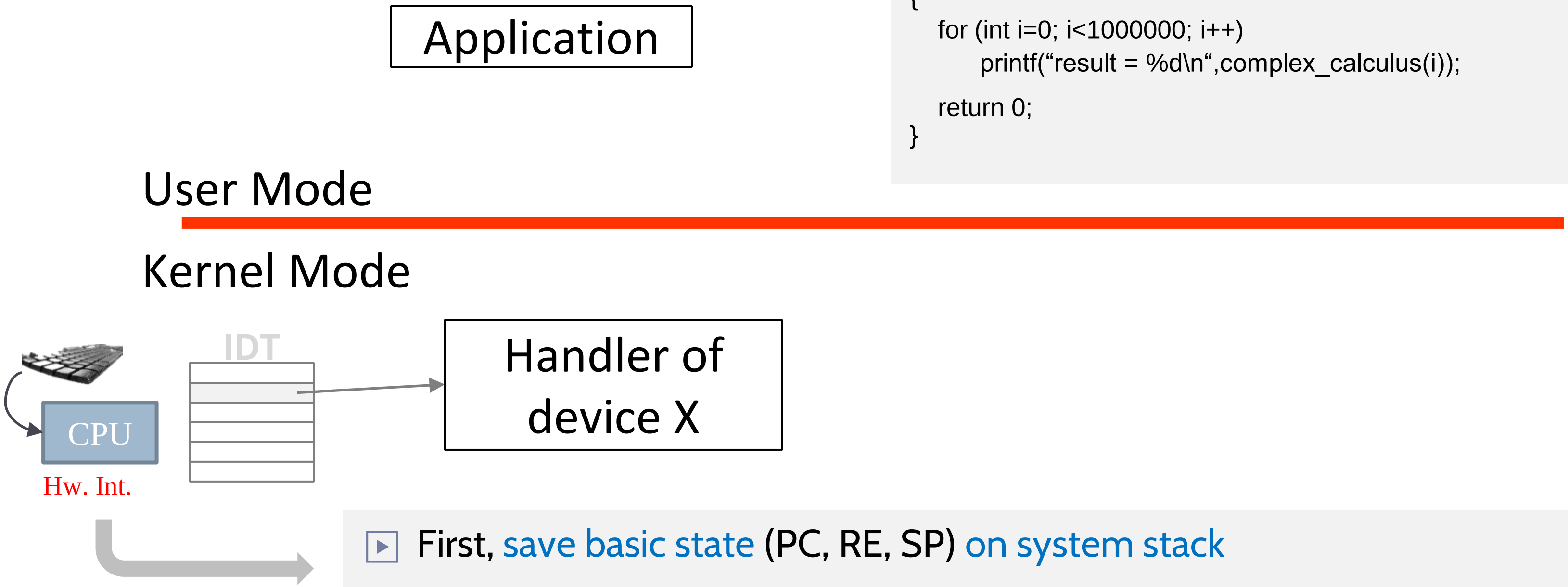
treatment (2/5)

```
#include "services.h"

int main ()
{
    for (int i=0; i<1000000; i++)
          printf(“result = %d\n“,complex_calculus(i));

    return 0;
}
```



Hardware interrupts
treatment (3/5)

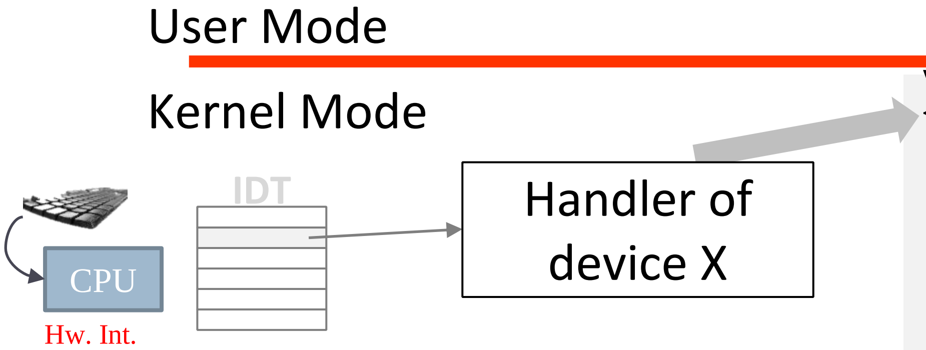
```
#include "services.h"

int main ()
{
    for (int i=0; i<1000000; i++)
          printf(“result = %d\n“,complex_calculus(i));

    return 0;
}
```



- First, save basic state (PC, RE, SP) on system stack
- CPU switch into privilegiate mode and jump to the associated treatment routine

Hardware interrupts

treatment (4/5)

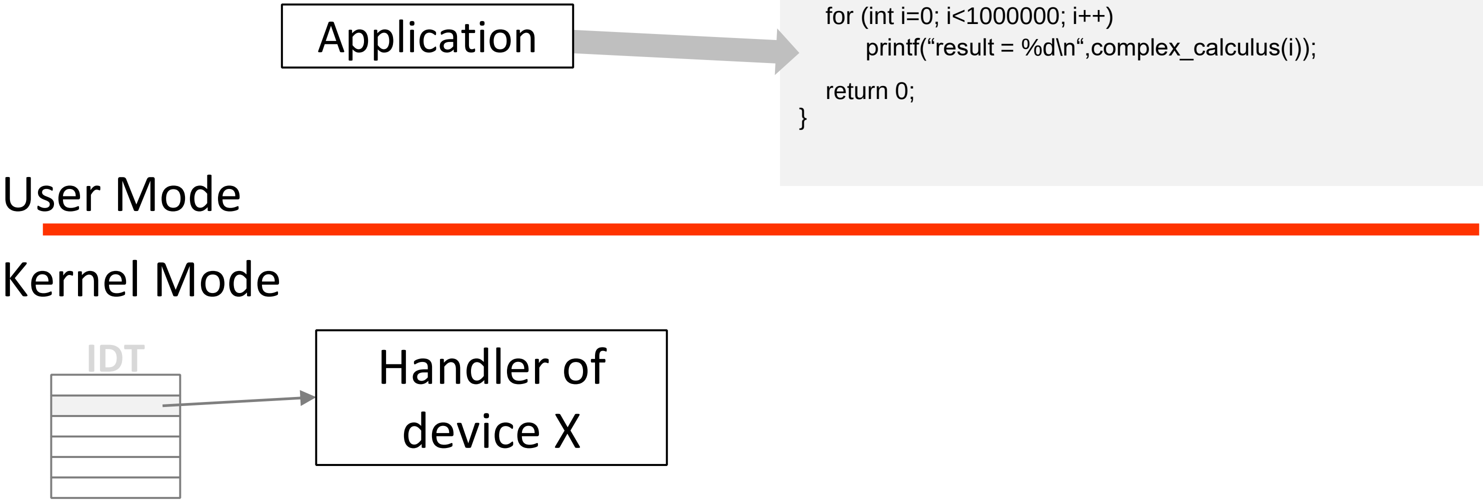


```
void interrupcionDevice ()
{
    Salvar estado (si es necesario)
    La subrutina trata el evento:
        Realiza lo urgente
        Programa una tarea pendiente
        (si necesario)
    Restaura el estado (si necesario)
    Execute instrucción de retorno de interrupción (RETI)
        Restaura estado básico y modo.
}
```

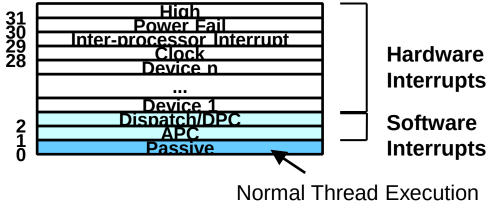
Hardware interrupts

treatment (5/5)

```
#include "services.h"

int main ()
{
    for (int i=0; i<1000000; i++)
          printf(“result = %d\n“,complex_calculus(i));

    return 0;
}
```



Hardware interrupts
treatment in Windows

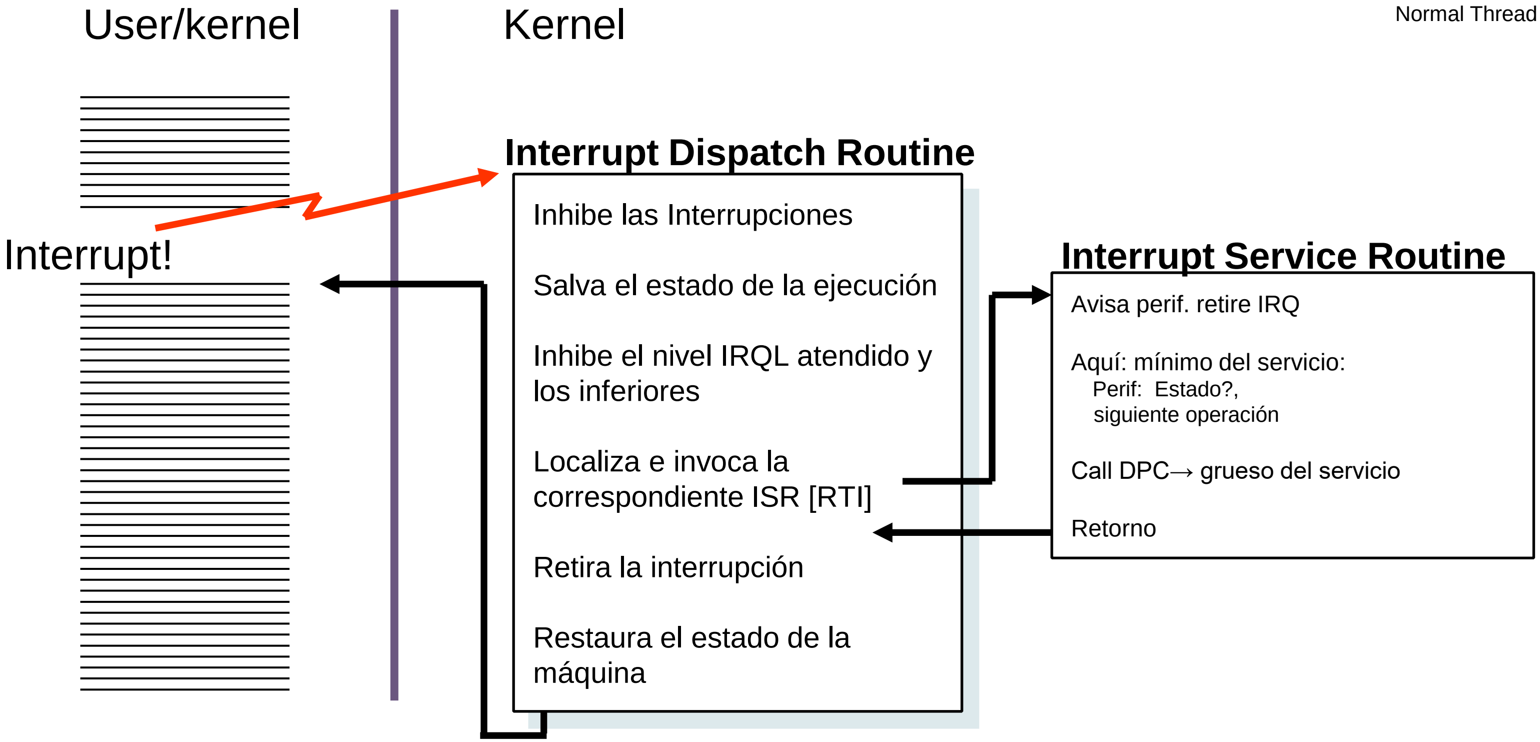


31 High
30 Power Fail
29 Inter-processor Interrupt
28 Clock
Device n
...
Device 1
2 Dispatch/DPC
1 APC
0 Passive

Hardware Interrupts

Software Interrupts

Normal Thread Execution



User/kernel

Kernel

Interrupt!

Interrupt Dispatch Routine

- Inhibe las Interrupciones
- Salva el estado de la ejecución
- Inhibe el nivel IRQL atendido y los inferiores
- Localiza e invoca la correspondiente ISR [RTI]
- Retira la interrupción
- Restaura el estado de la máquina

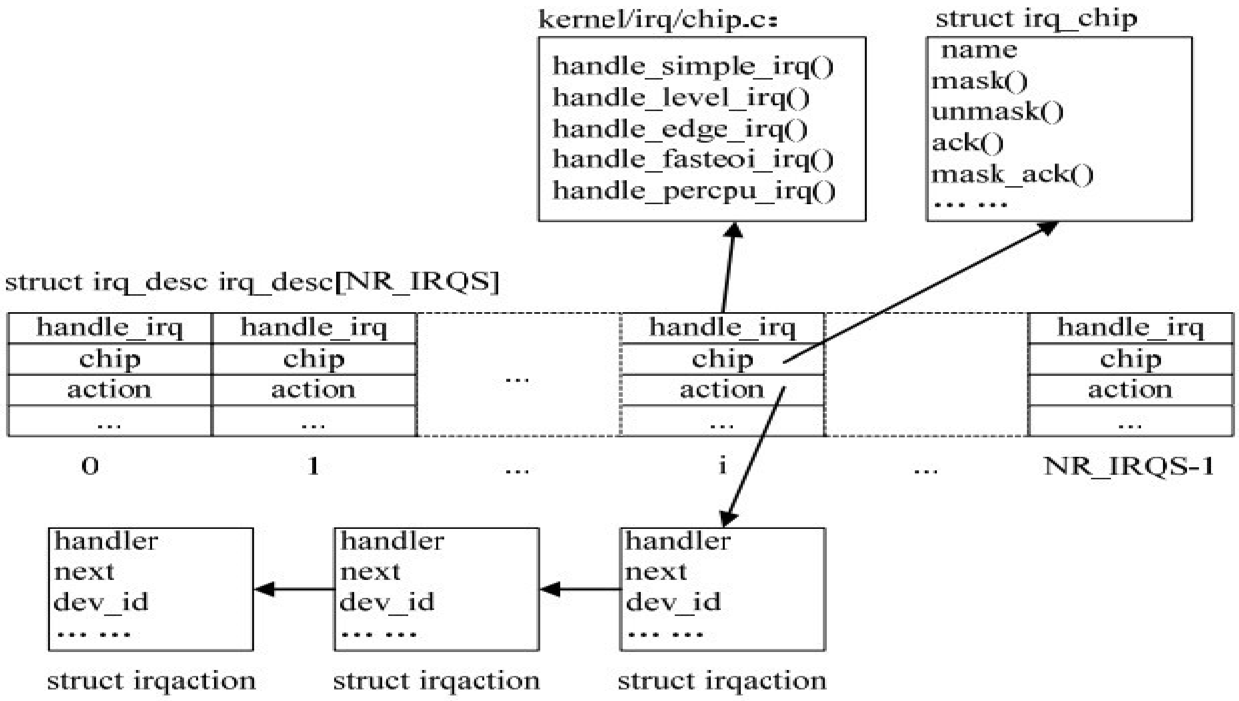
Interrupt Service Routine

- Avisa perif. retire IRQ
- Aquí: mínimo del servicio:
  - Perif: Estado?,
  - siguiente operación
- Call DPC→ grueso del servicio
- Retorno

Inside Windows 2000 (página 104)

Hardware interrupts

treatment in Linux



http://chxxxyg.blog.163.com/blog/static/150281193201062701509 8/

Exceptions

characteristics

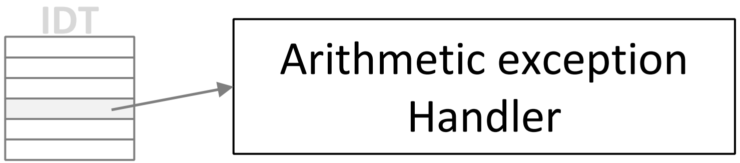
- Synchronous events, exceptional ones while executing an instruction
- Previous execution mode:
  - It could be user or system (YES, it influences the treatment)
- Generated by:
  - Usually by hardware (usually errors)
    - But not always are errors (e.g.: page fault, debugging, etc.)

Exceptions

treatment (1/4)

User Mode

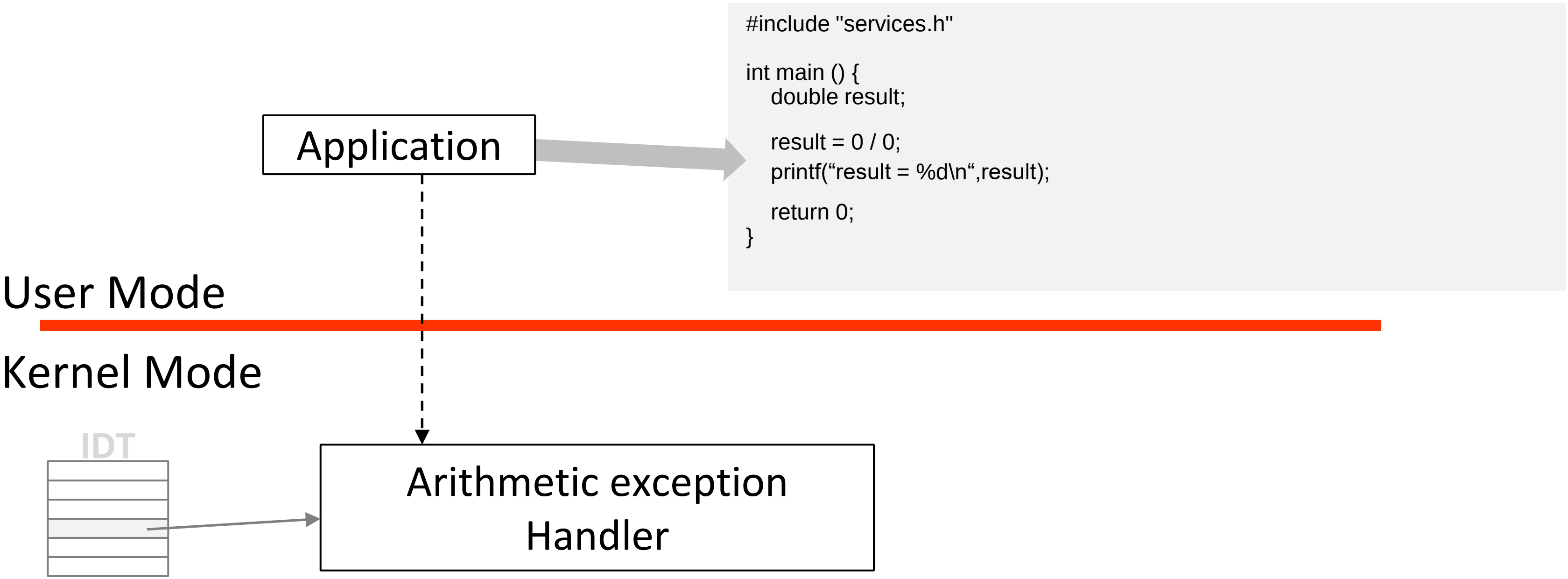
Kernel Mode



```
int main (int argc, char **argv)
{
     ...
     /* instalar los manejadores para los vectores de interrupción */
     instal_man_int(EXC_ARITMETICA,   hnd_exceptionAritmetica);
     instal_man_int(EXC_MEMORIA,      hnd_exceptionMemory);
     instal_man_int(INT_RELOJ,        hnd_interruptClock);
     instal_man_int(INT_DeviceS, hnd_interruptDevices);
     instal_man_int(LLAM_SISTEMA,     hnd_SystemCall);
     instal_man_int(INT_SW,           hnd_softwareInterrupt);
     ...
```

Exceptions

treatment (2/4)

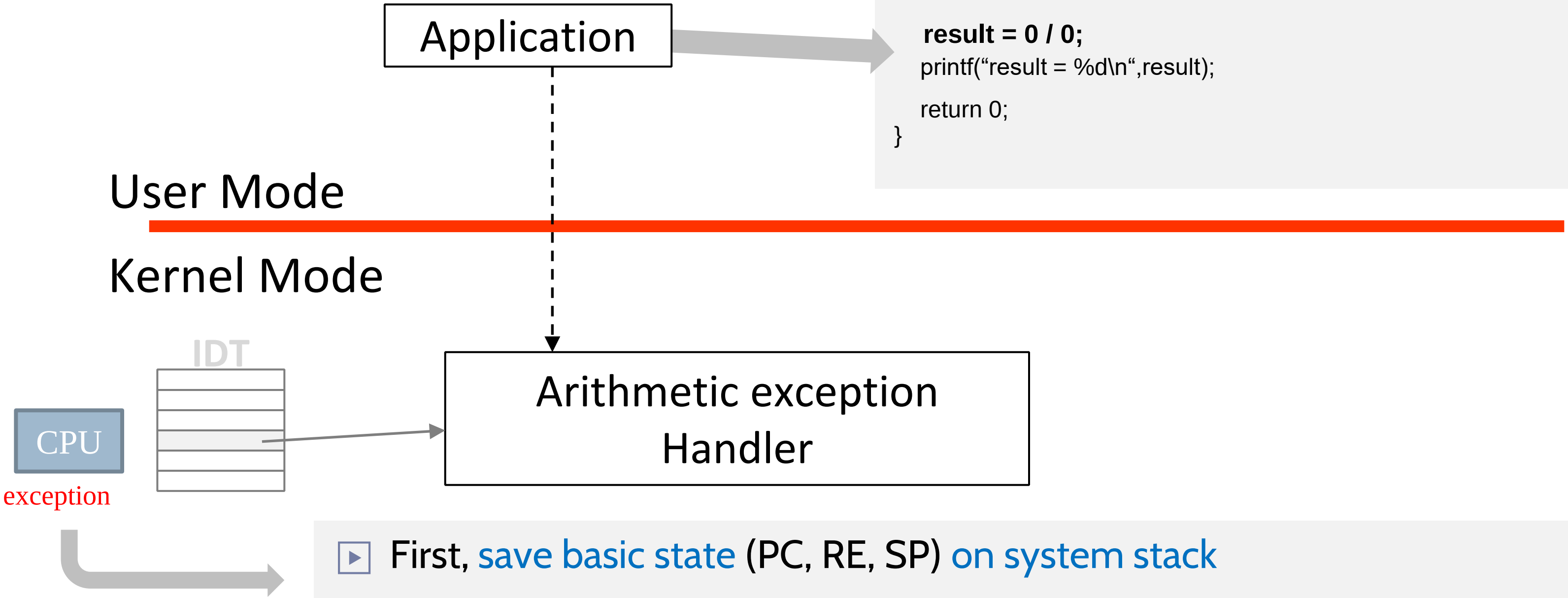


```
#include "services.h"

int main () {
   double result;

   result = 0 / 0;
   printf(“result = %d\n“,result);

   return 0;
}
```

Exceptions

treatment (3/4)

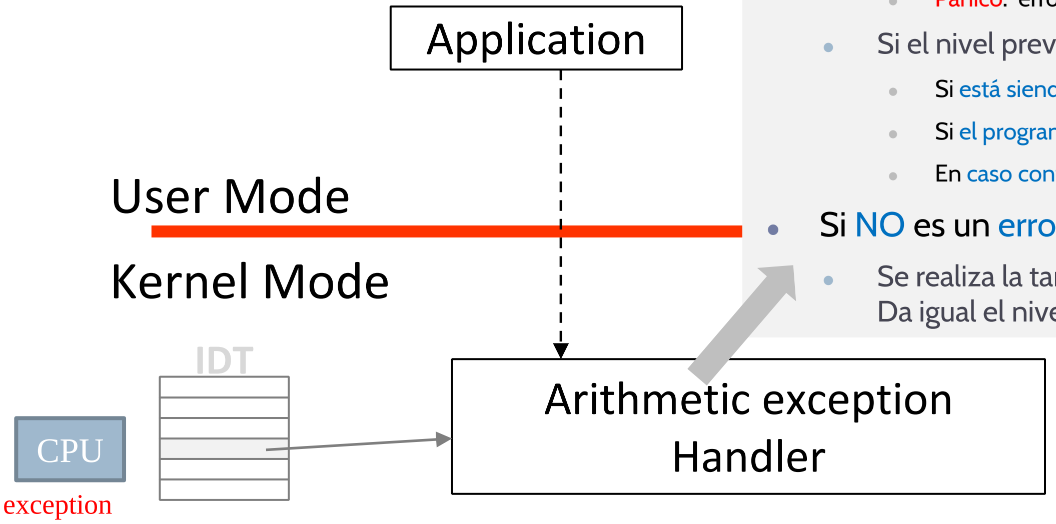
```
#include "services.h"

int main () {
    double result;

    result = 0 / 0;
    printf("result = %d\n",result);

    return 0;
}
```



- First, save basic state (PC, RE, SP) on system stack
- CPU switch into privilegiate mode and jump to the associated treatment routine

Exceptions

treatment (4/4)



- Si es un error:
  - Si el nivel previo de la CPU era de sistema:
    - Pánico: error en el código del .O.S. => mensaje + detener el .O.S.
  - Si el nivel previo de la CPU era de User:
    - Si está siendo depurado, se notifica a depurador
    - Si el programa establece un Handler of la exception, ejecutarlo
    - En caso contrario, se aborta el proceso
- Si NO es un error: (Ej.: fallo de página previsto)
  - Se realiza la tarea prevista (Ej.: asignar una nueva página)
    Da igual el nivel previo de ejecución

System calls
characteristics

- Synchronous events for requesting O.S. services with an unprivileged instruction

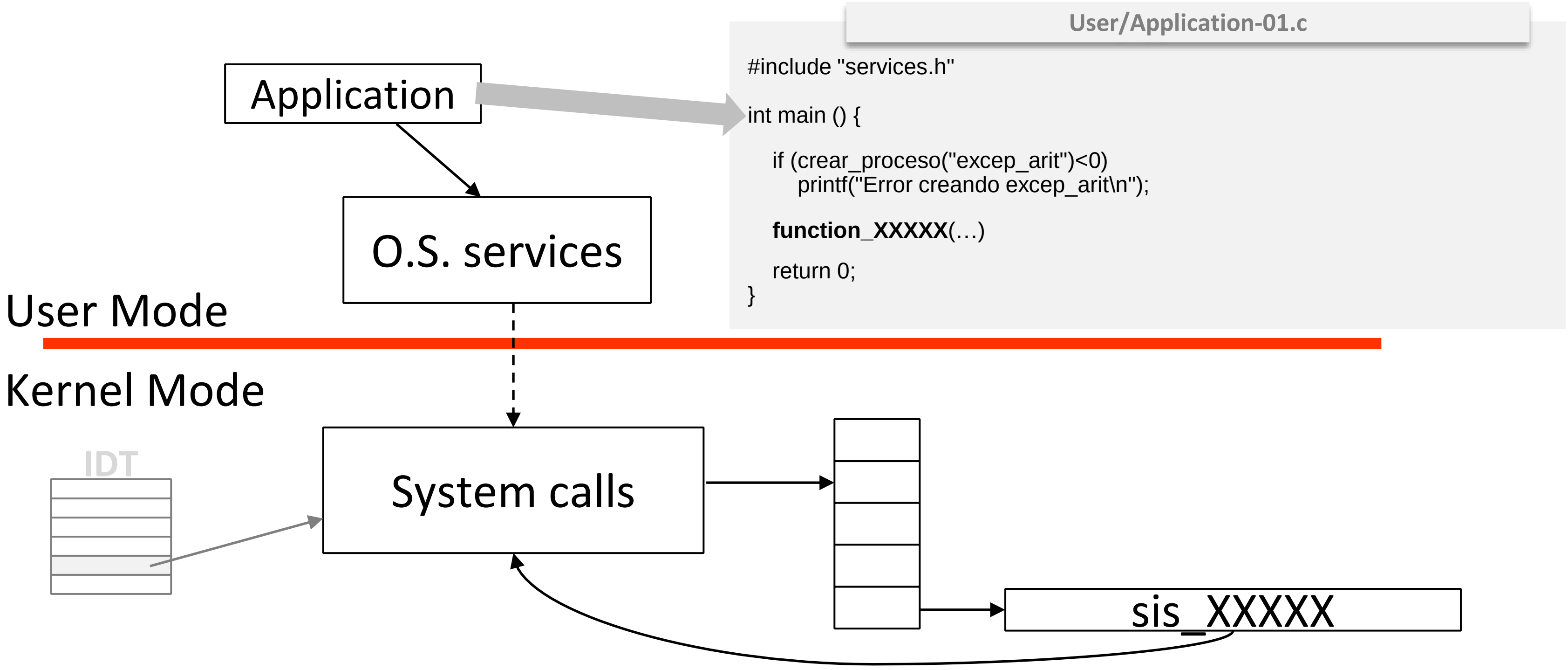
- Previous execution mode:
  - User mode always

- Generated by:
  - By applications

System calls
treatment

```
int main (int argc, char **argv)
{
          ...

          /* instalar los manejadores para los vectores de interrupción */
          instal_man_int(EXC_ARITMETICA,   hnd_exceptionAritmetica);
          instal_man_int(EXC_MEMORIA,      hnd_exceptionMemory);
          instal_man_int(INT_RELOJ,        hnd_interruptClock);
          instal_man_int(INT_DeviceS,      hnd_interruptDevices);
          instal_man_int(LLAM_SISTEMA,     hnd_SystemCall);
          instal_man_int(INT_SW,           hnd_softwareInterrupt);

          ...
```

System calls
treatment (1/9)

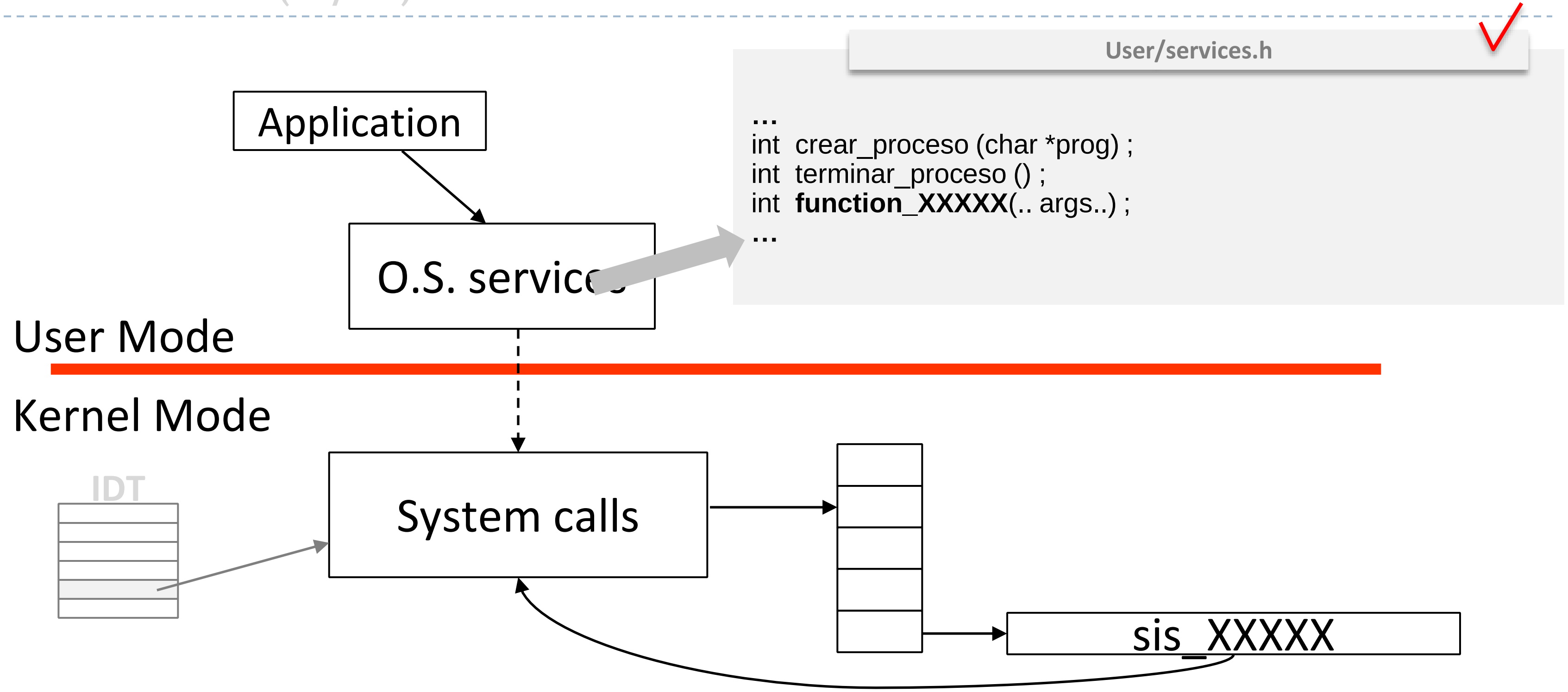


User/Application-01.c

```
#include "services.h"

int main () {

    if (crear_proceso("excep_arit")<0)
        printf("Error creando excep_arit\n");

    function_XXXXX(…)

    return 0;
}
```

System calls

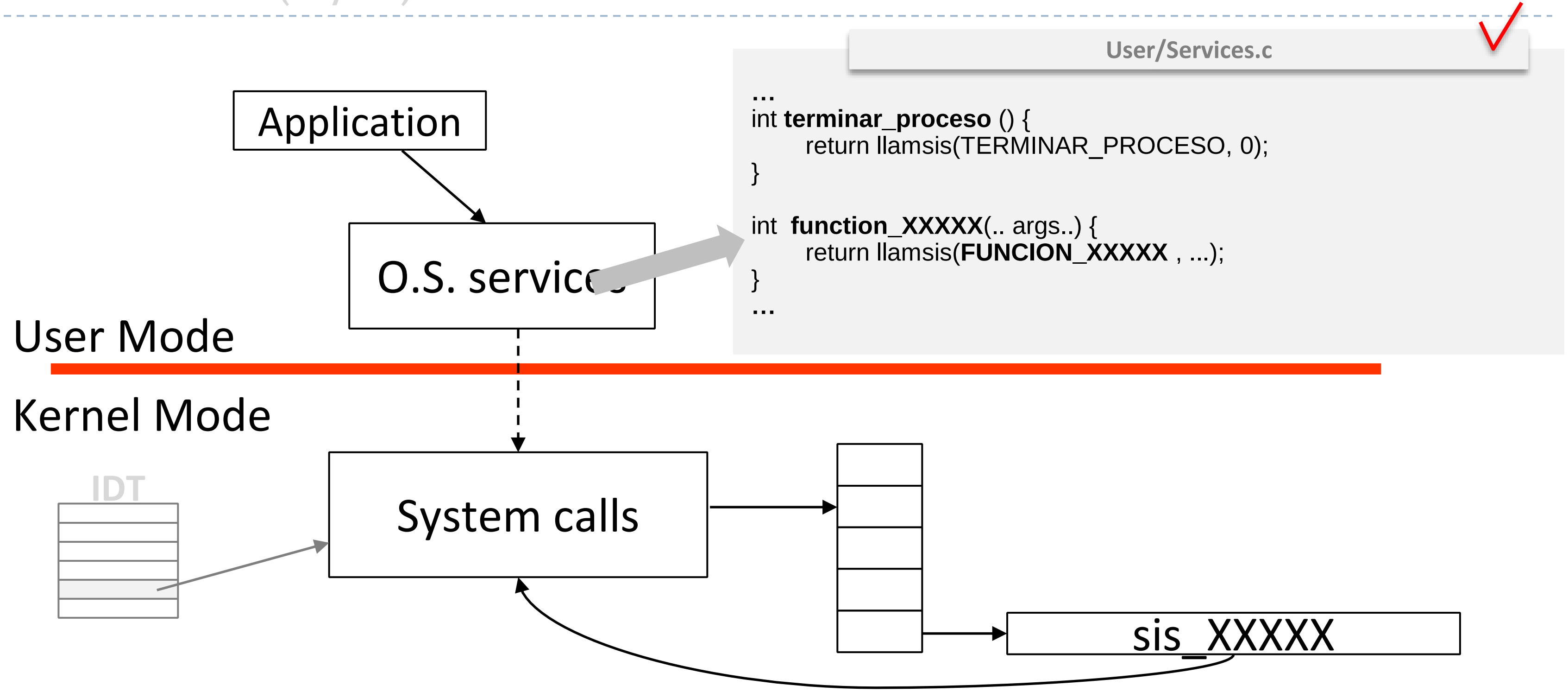
treatment (2/9)



```
...
int  crear_proceso (char *prog) ;
int  terminar_proceso () ;
int  function_XXXXX(.. args..) ;
...
```

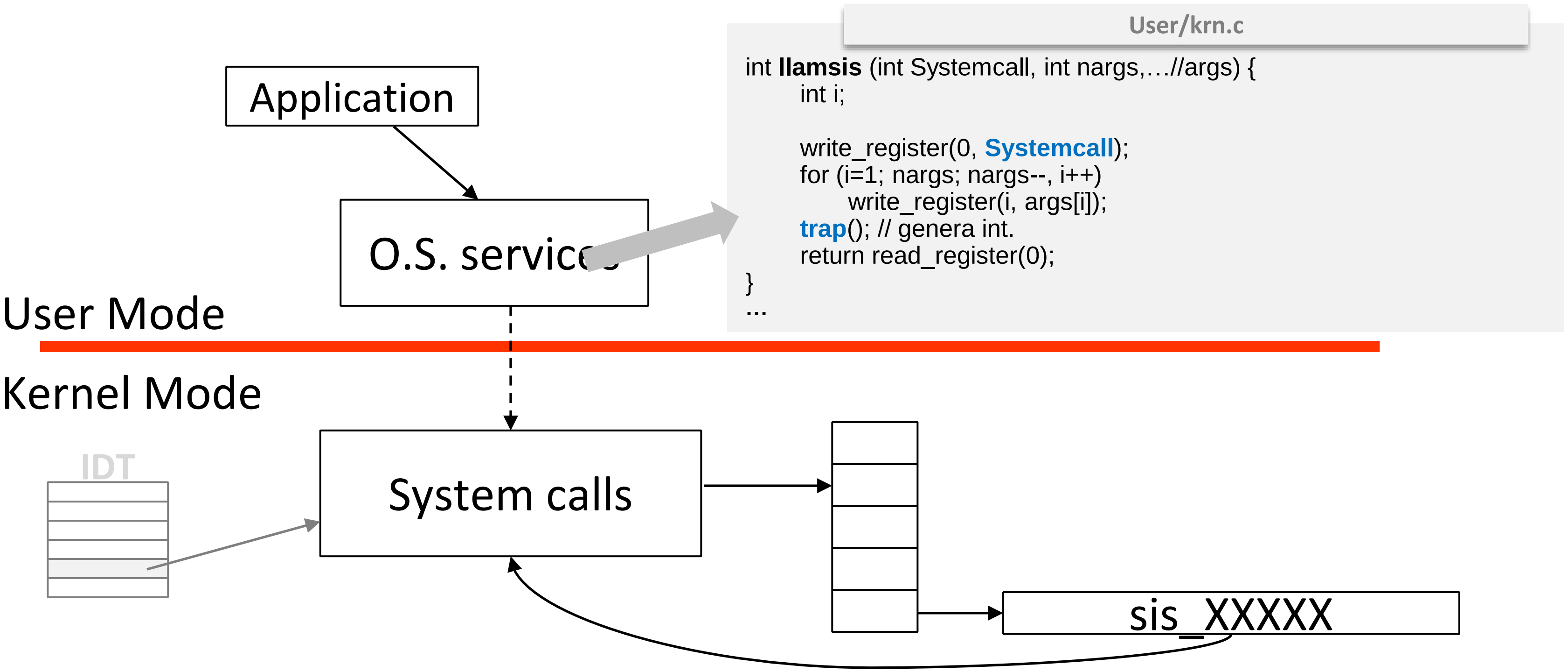
System calls

treatment (3/9)



```
...
int terminar_proceso () {
        return llamsis(TERMINAR_PROCESO, 0);
}

int  function_XXXXX(.. args..) {
        return llamsis(FUNCION_XXXXX , ...);
}
...
```

System calls
treatment (4/9)

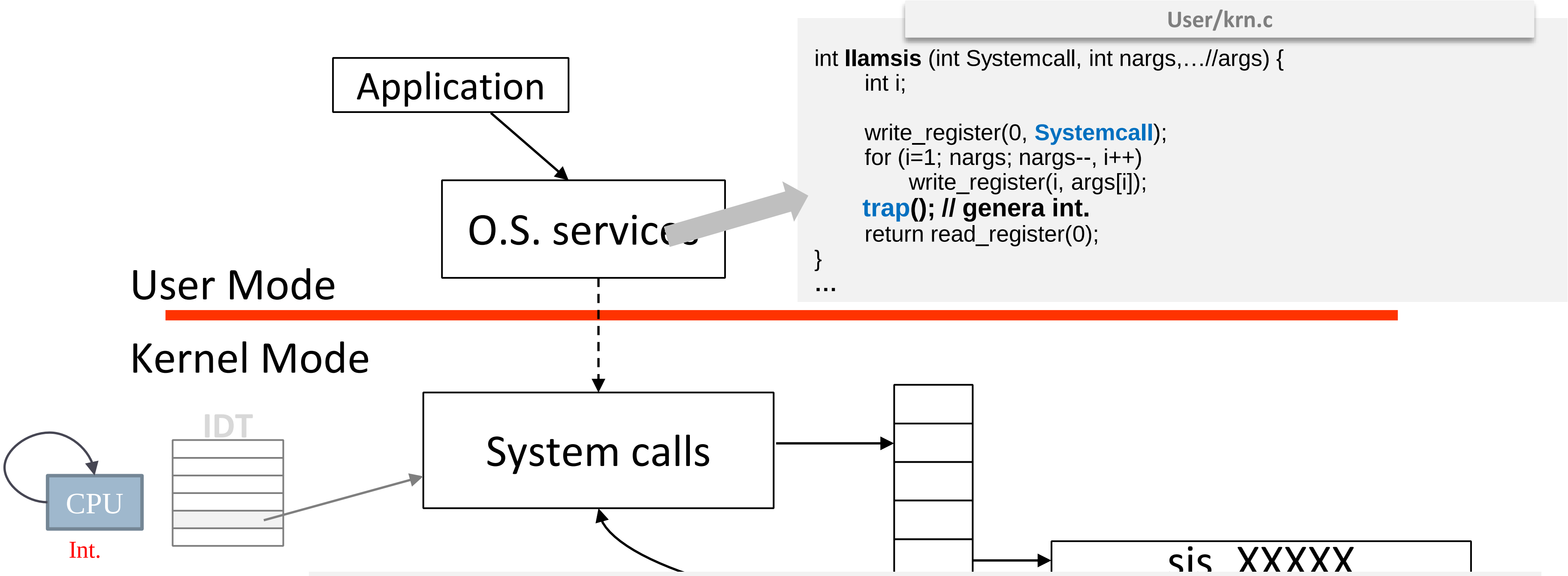


User/krn.c

```
int llamsis (int Systemcall, int nargs,…//args) {
        int i;

        write_register(0, Systemcall);
        for (i=1; nargs; nargs--, i++)
                write_register(i, args[i]);
        trap(); // genera int.
        return read_register(0);
}
...
```

System calls

treatment (5/9)



Application

O.S. service

User Mode

Kernel Mode

IDT

CPU

Int.

System calls

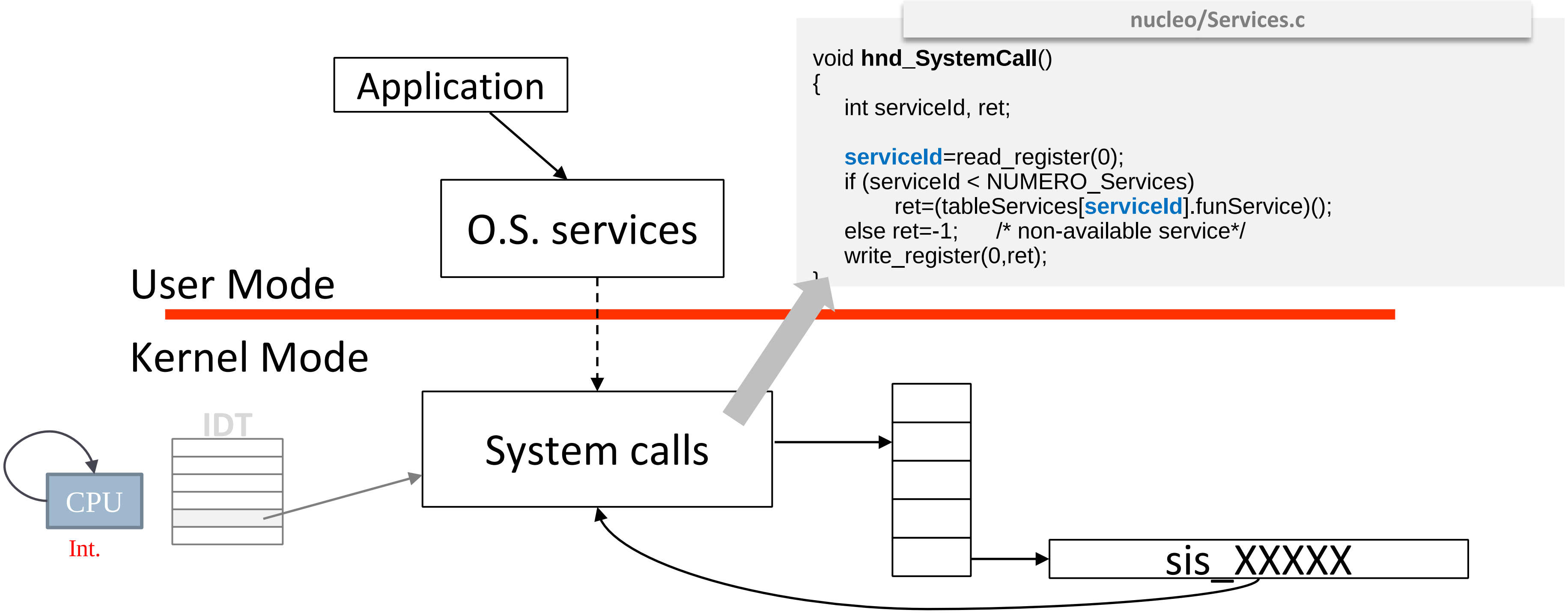
sis_XXXXX

User/krn.c

```
int llamsis (int Systemcall, int nargs,…//args) {
      int i;

      write_register(0, Systemcall);
      for (i=1; nargs; nargs--, i++)
            write_register(i, args[i]);
      trap(); // genera int.
      return read_register(0);
}
...
```

- First, save basic state (PC, RE, SP) on system stack
- CPU switch into privilegiate mode and jump to the associated treatment routine

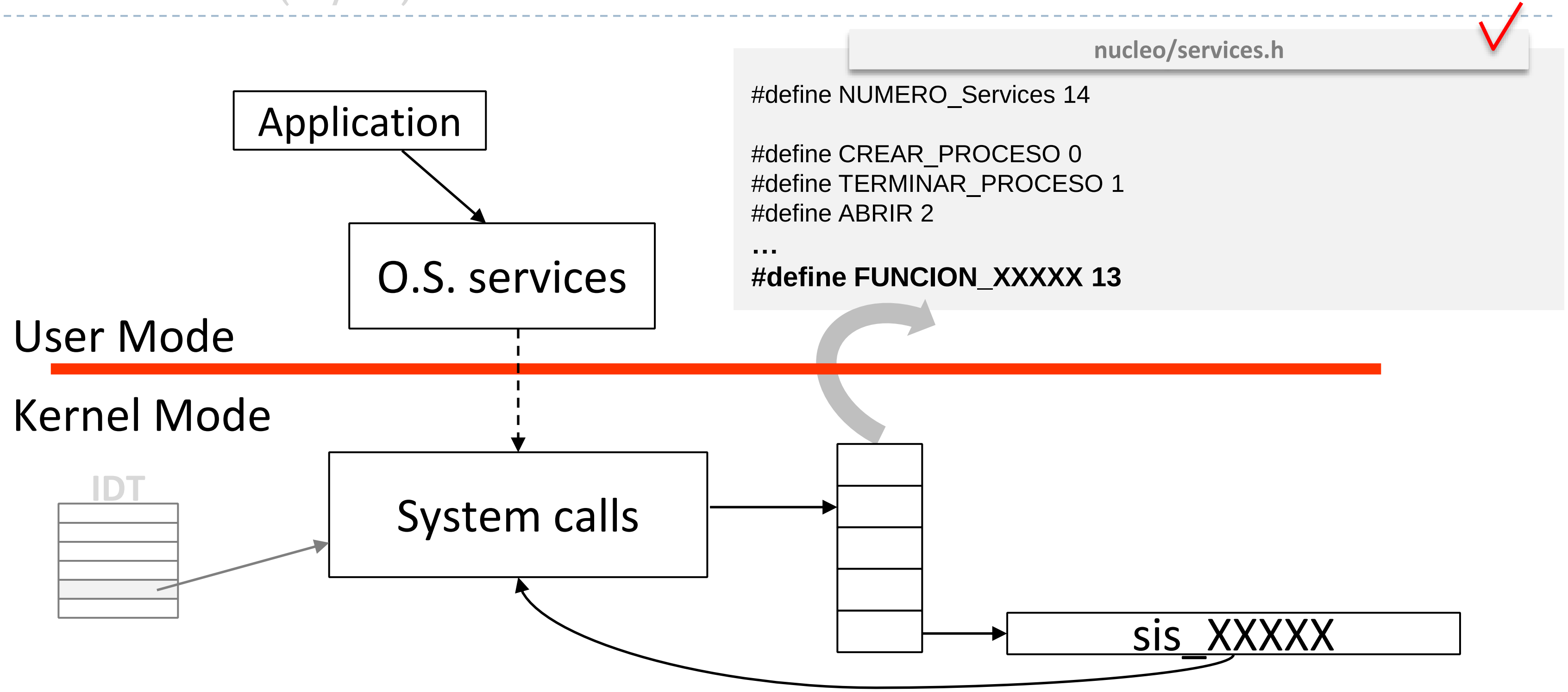
System calls
treatment (6/9)



```
void hnd_SystemCall()
{
    int serviceId, ret;

    serviceId=read_register(0);
    if (serviceId < NUMERO_Services)
          ret=(tableServices[serviceId].funService)();
    else ret=-1;      /* non-available service*/
    write_register(0,ret);
}
```

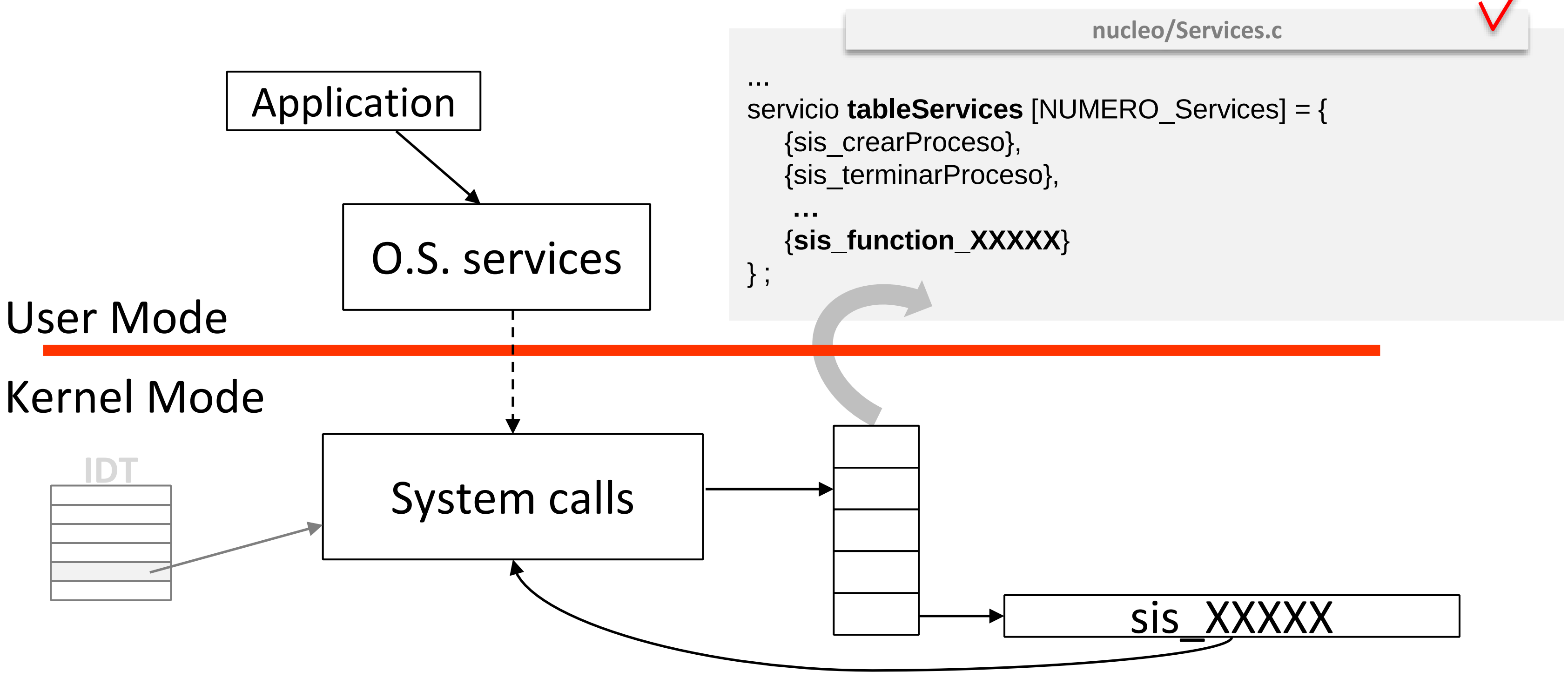
System calls

treatment (7/9)



```
#define NUMERO_Services 14

#define CREAR_PROCESO 0
#define TERMINAR_PROCESO 1
#define ABRIR 2
...
#define FUNCION_XXXXX 13
```

System calls
treatment (8/9)

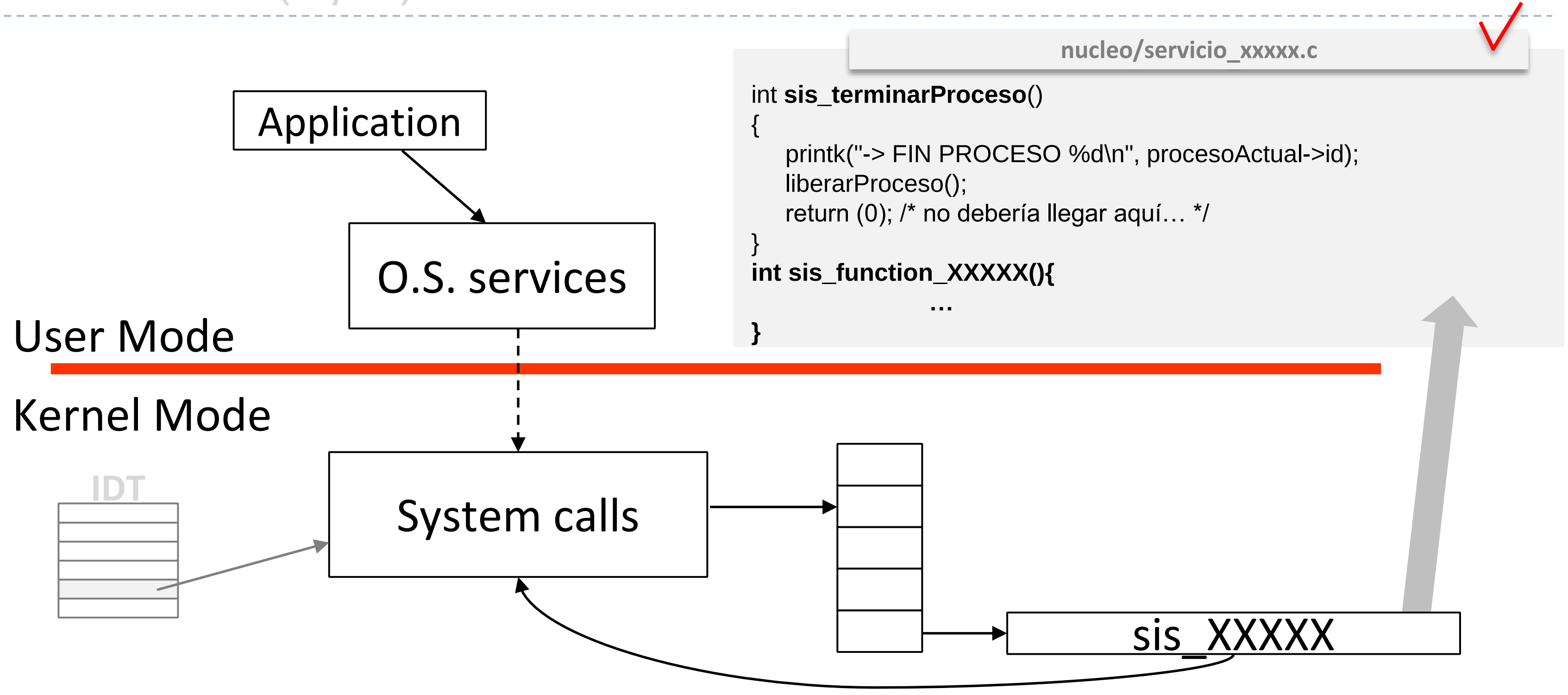


nucleo/Services.c

```
...
servicio **tableServices** [NUMERO_Services] = {
    {sis_crearProceso},
    {sis_terminarProceso},
     ...
    {**sis_function_XXXXX**}
} ;
```

System calls

treatment (9/9)



nucleo/servicio_xxxxx.c ✓

```
int sis_terminarProceso()
{
    printk("-> FIN PROCESO %d\n", procesoActual->id);
    liberarProceso();
    return (0); /* no debería llegar aquí… */
}
int sis_function_XXXXX(){
            ...
}
```

System calls

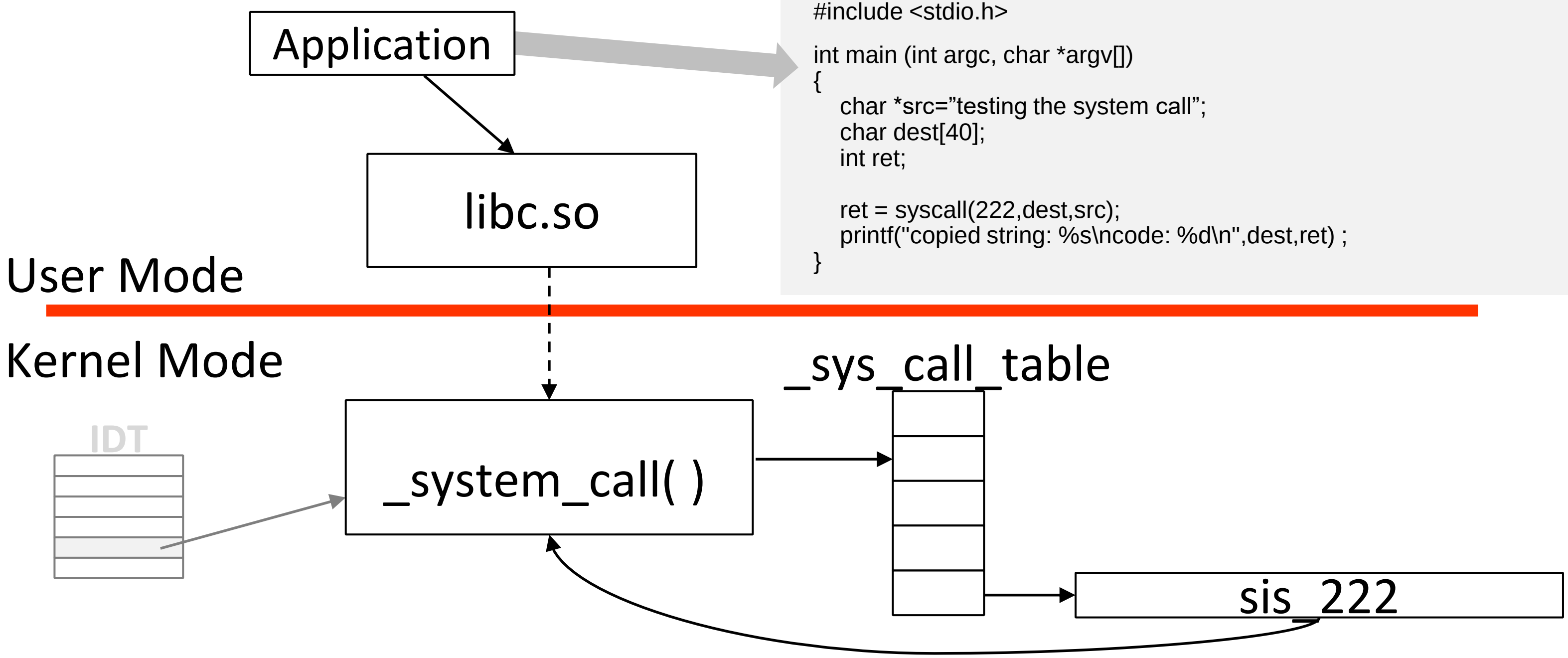
treatment in Linux (1/7)

/usr/src/linux/arch/x86/kernel/traps.c

```
void __init trap_init(void)
{
        ...
        set_intr_gate(X86_TRAP_DE, divide_error);
        set_intr_gate(X86_TRAP_NP, segment_not_present);
        set_intr_gate(X86_TRAP_GP, general_protection);
        set_intr_gate(X86_TRAP_SPURIOUS, spurious_interrupt_bug);
        set_intr_gate(X86_TRAP_MF, coprocessor_error);
        set_intr_gate(X86_TRAP_AC, alignment_check);

#ifdef CONFIG_IA32_EMULATION
        set_system_intr_gate(IA32_SYSCALL_VECTOR, ia32_syscall);
        set_bit(IA32_SYSCALL_VECTOR, used_vectors);
#endif

#ifdef CONFIG_X86_32
        set_system_trap_gate(SYSCALL_VECTOR, &system_call);
        set_bit(SYSCALL_VECTOR, used_vectors);
#endif
...
```

System calls

treatment in Linux (2/7)

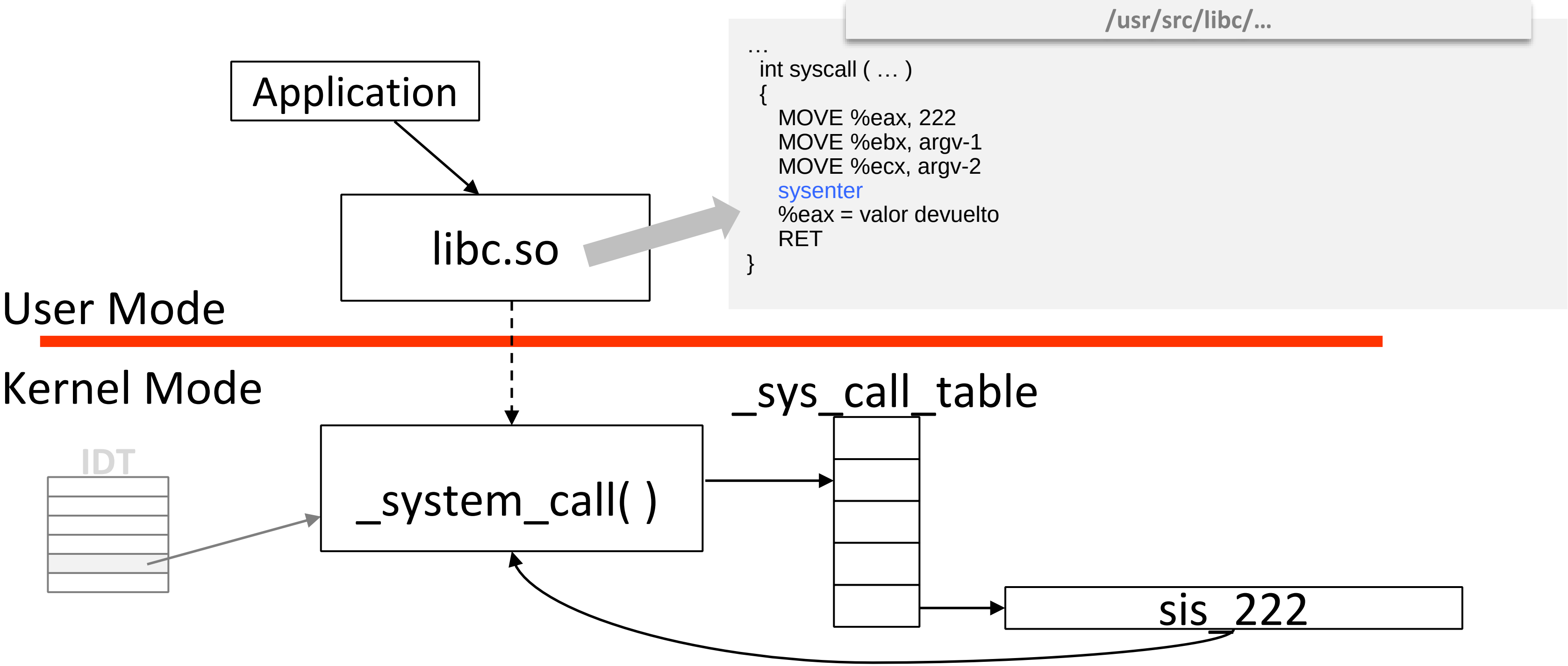


/usr/src/linux/test/test1.c

```
#include <stdio.h>

int main (int argc, char *argv[])
{
    char *src=”testing the system call”;
    char dest[40];
    int ret;

    ret = syscall(222,dest,src);
    printf("copied string: %s\ncode: %d\n",dest,ret) ;
}
```

System calls

treatment in Linux (3/7)

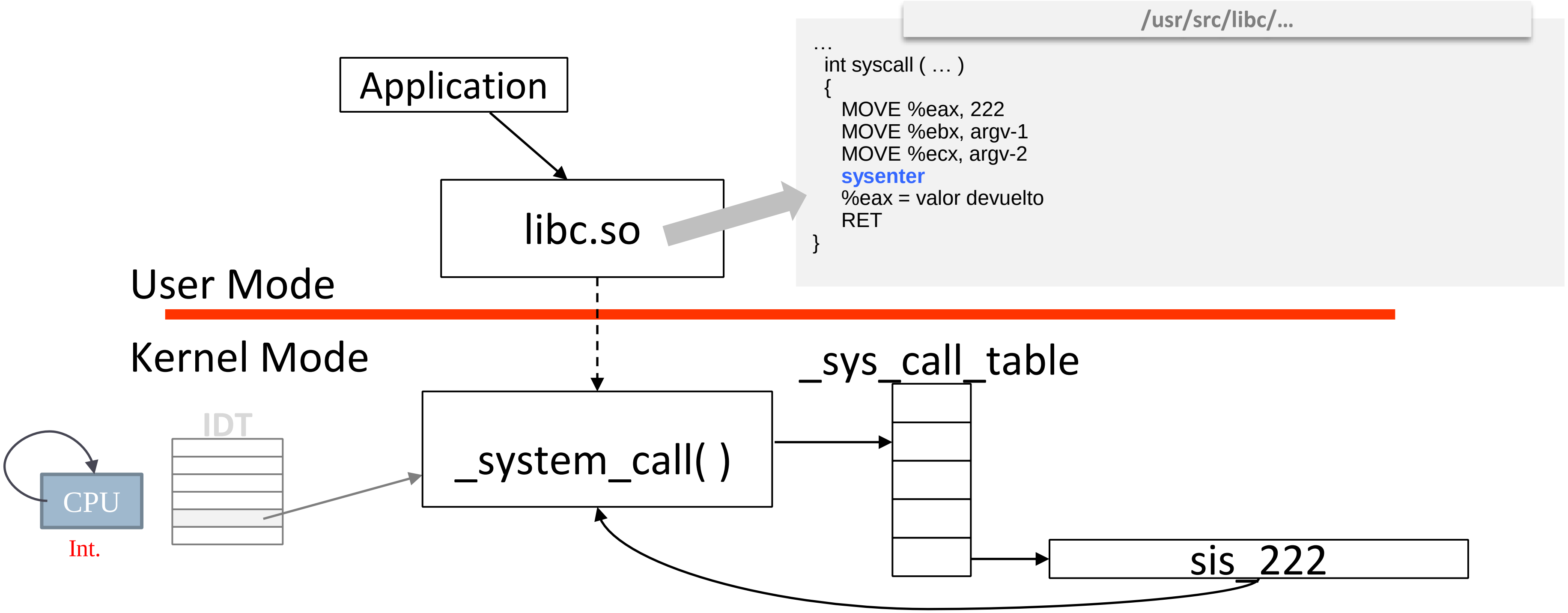


/usr/src/libc/…

```
…
 int syscall ( … )
 {
    MOVE %eax, 222
    MOVE %ebx, argv-1
    MOVE %ecx, argv-2
    sysenter
    %eax = valor devuelto
    RET
}
```

System calls

treatment in Linux (3/7)

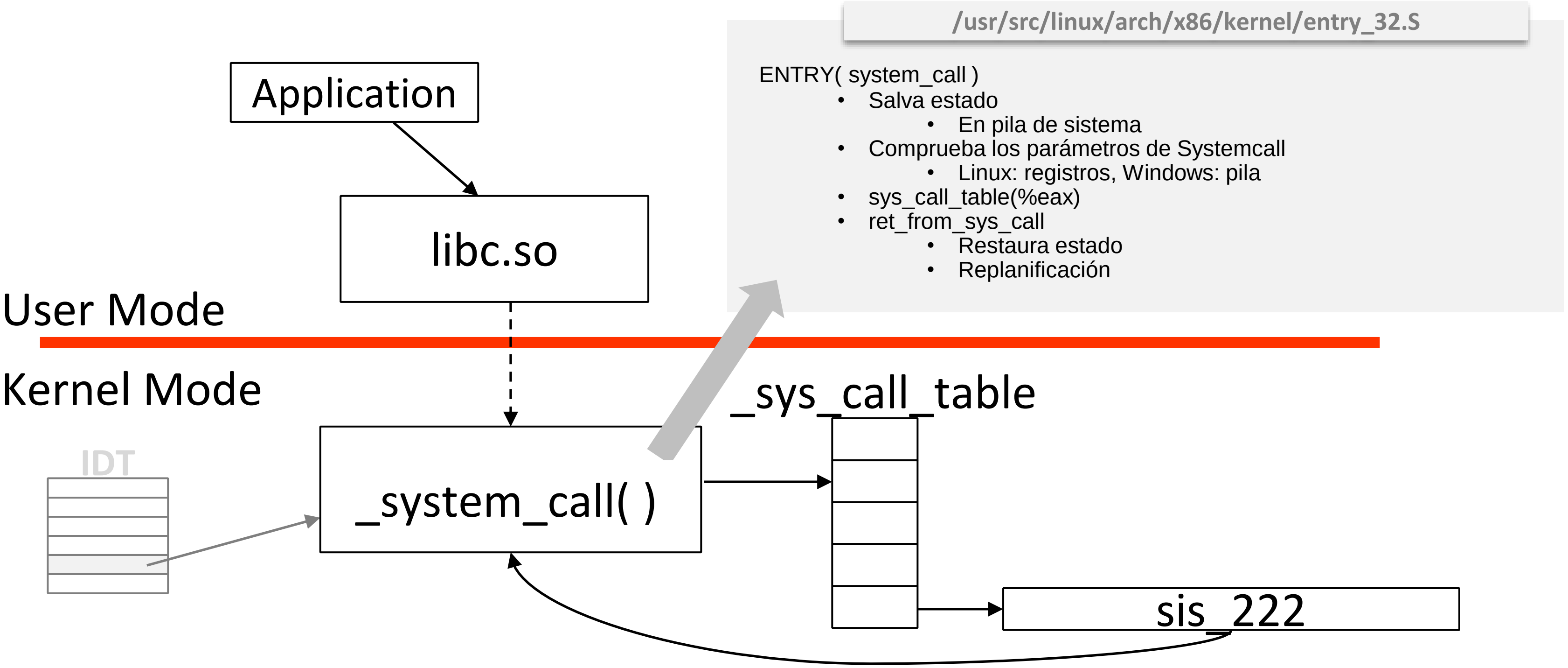


/usr/src/libc/…

```
…
 int syscall ( … )
 {
    MOVE %eax, 222
    MOVE %ebx, argv-1
    MOVE %ecx, argv-2
    sysenter
    %eax = valor devuelto
    RET
}
```

System calls

treatment in Linux (4/7)



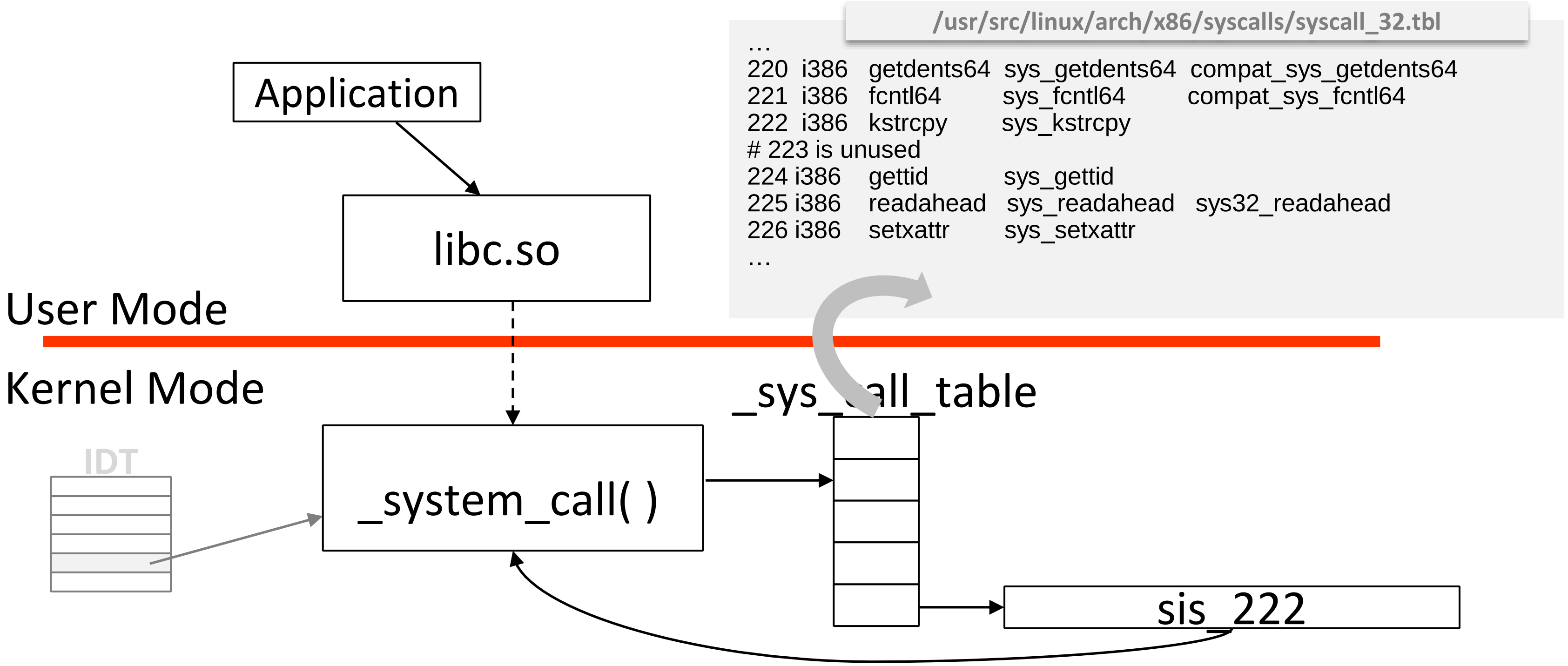
/usr/src/linux/arch/x86/kernel/entry_32.S

ENTRY(system_call)

- Salva estado
  - En pila de sistema
- Comprueba los parámetros de Systemcall
  - Linux: registros, Windows: pila
- sys_call_table(%eax)
- ret_from_sys_call
  - Restaura estado
  - Replanificación

System calls

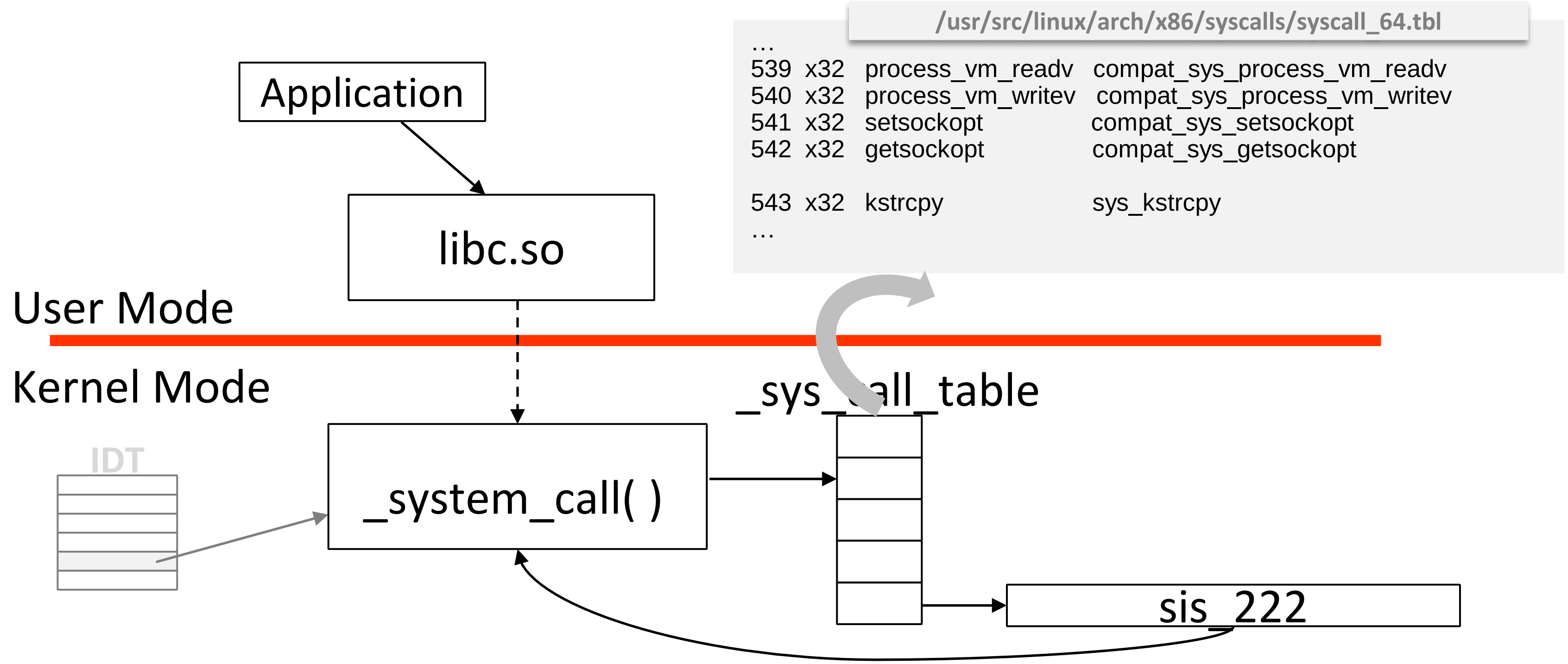
treatment in Linux (5/7)



/usr/src/linux/arch/x86/syscalls/syscall_32.tbl

```
…
220  i386   getdents64  sys_getdents64  compat_sys_getdents64
221  i386   fcntl64      sys_fcntl64      compat_sys_fcntl64
222  i386   kstrcpy      sys_kstrcpy
# 223 is unused
224 i386    gettid        sys_gettid
225 i386    readahead   sys_readahead   sys32_readahead
226 i386    setxattr      sys_setxattr
…
```

System calls
treatment in Linux (6/7)

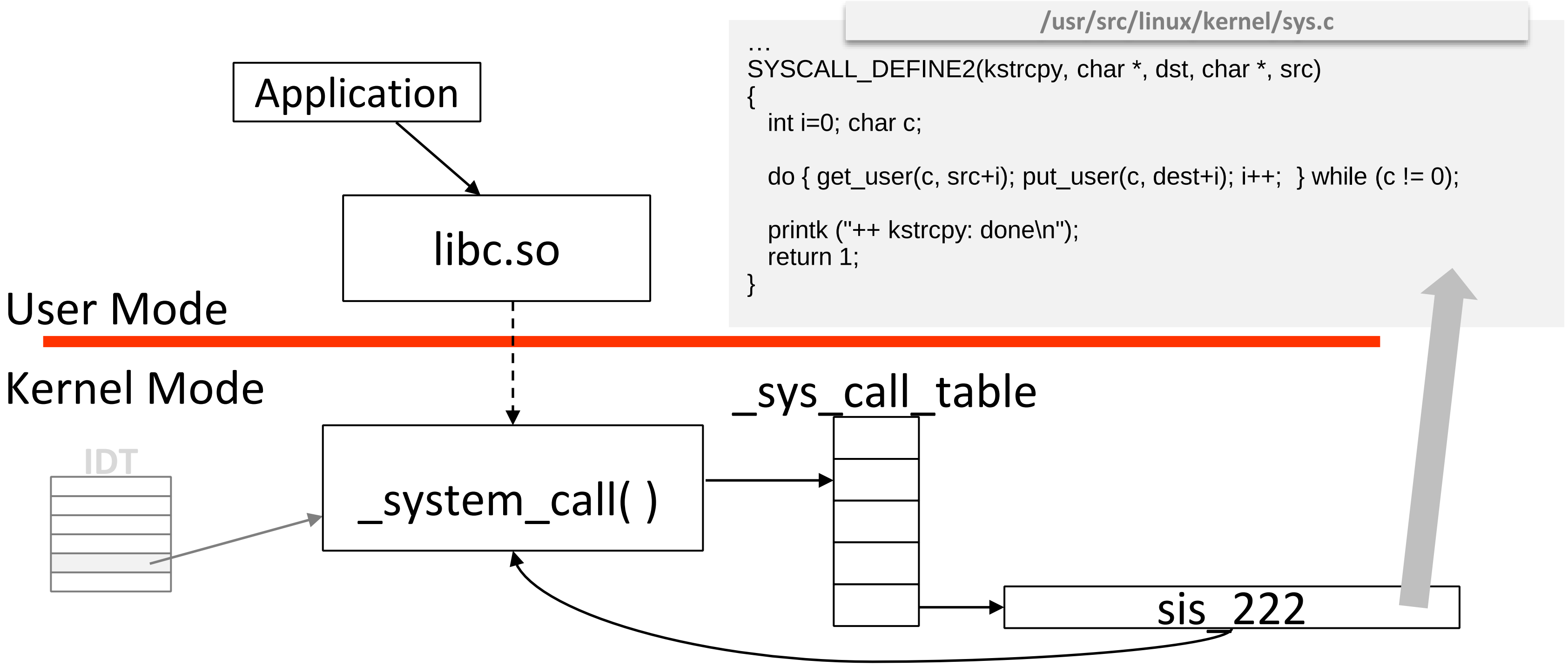


/usr/src/linux/arch/x86/syscalls/syscall_64.tbl

```
…
539  x32   process_vm_readv   compat_sys_process_vm_readv
540  x32   process_vm_writev  compat_sys_process_vm_writev
541  x32   setsockopt         compat_sys_setsockopt
542  x32   getsockopt         compat_sys_getsockopt

543  x32   kstrcpy            sys_kstrcpy
…
```

System calls

treatment in Linux (7/7)

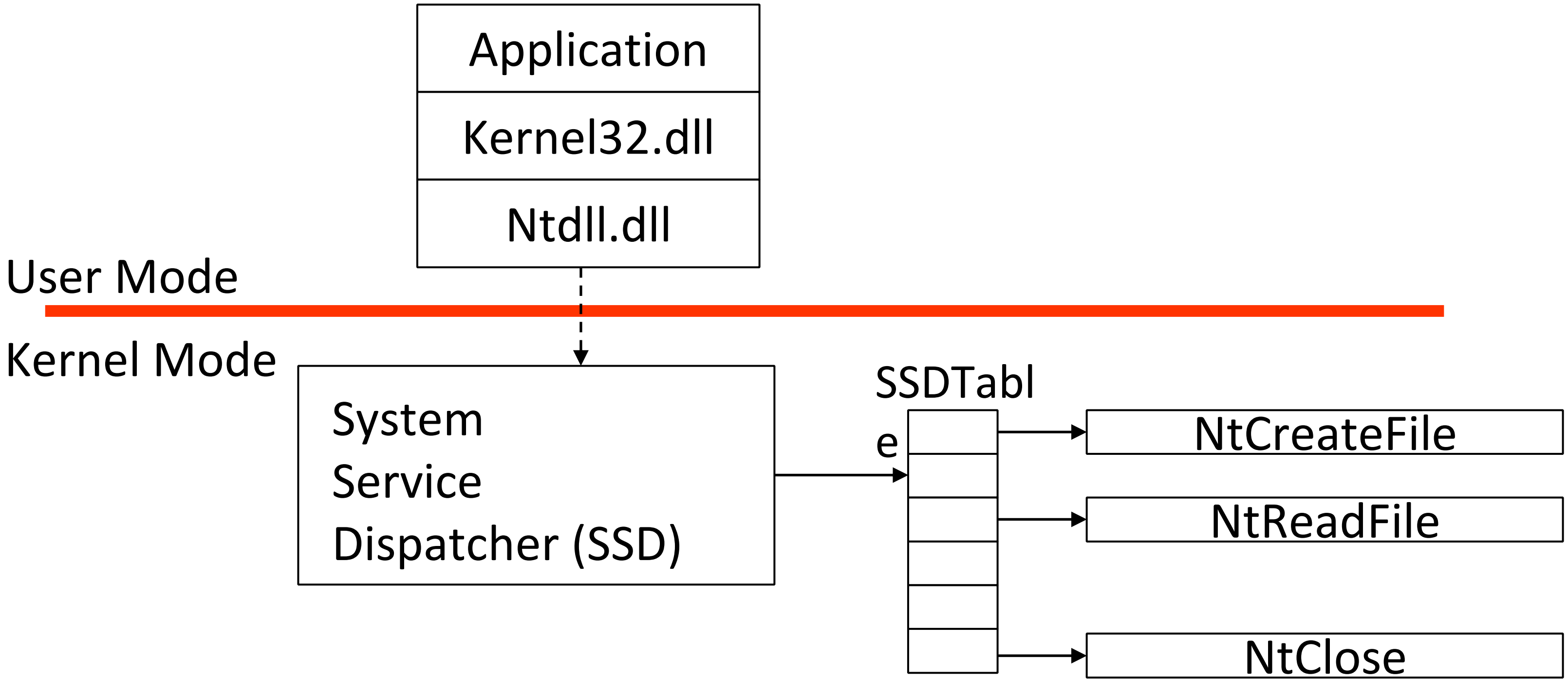


/usr/src/linux/kernel/sys.c

```
…
SYSCALL_DEFINE2(kstrcpy, char *, dst, char *, src)
{
   int i=0; char c;

   do { get_user(c, src+i); put_user(c, dest+i); i++;  } while (c != 0);

   printk ("++ kstrcpy: done\n");
   return 1;
}
```

System calls

treatment in Windows



Application

Kernel32.dll

Ntdll.dll

User Mode

Kernel Mode

System Service Dispatcher (SSD)

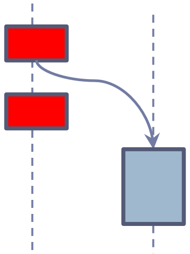
SSDTable

NtCreateFile

NtReadFile

NtClose

Software interrupt
characteristics



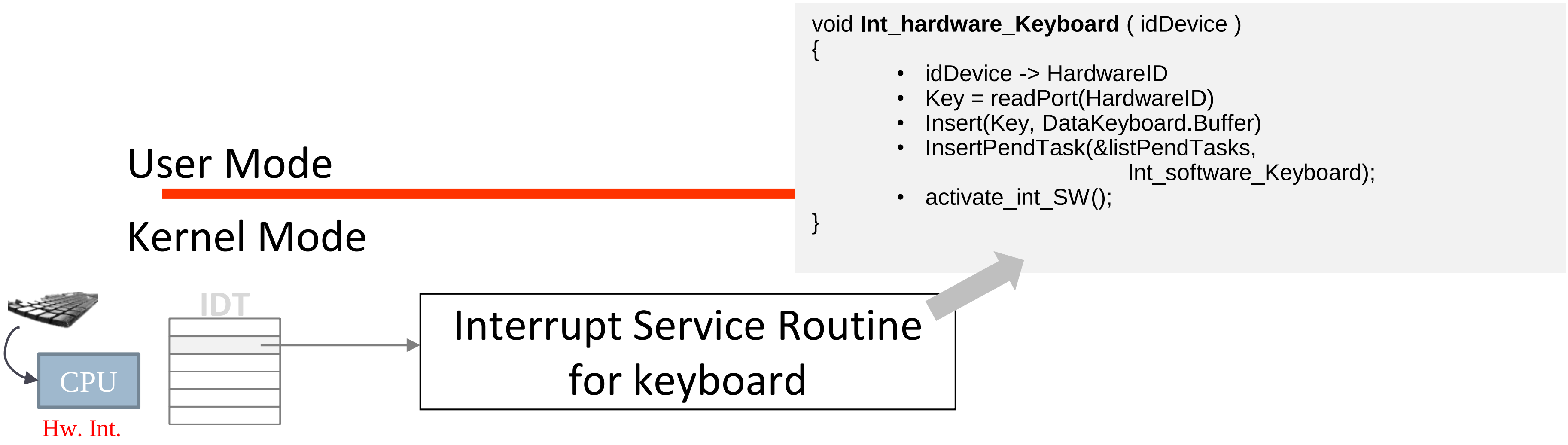
- Asynchronous events to deferre the non-critical part of the event treatment
  - To wait better opportunity.
  - Treated the critical parts first.
- Previous execution mode:
  - Always system mode
- Generated by:
  - In the event treatment of all former events, software interrupts is used for the non-critical parts

Software interrupt
treatment

```
int main (int argc, char **argv)
{
            ...

            /* instalar los manejadores para los vectores de interrupción */
            instal_man_int(EXC_ARITMETICA,   hnd_exceptionAritmetica) ;
            instal_man_int(EXC_MEMORIA,      hnd_exceptionMemory) ;
            instal_man_int(INT_RELOJ,        hnd_interruptClock) ;
            instal_man_int(INT_DeviceS,      hnd_interruptDevices) ;
            instal_man_int(LLAM_SISTEMA,     hnd_SystemCall) ;
            instal_man_int(INT_SW,           hnd_softwareInterrupt) ;

            ...
```

Interrupción hardware

treatment (1/2)



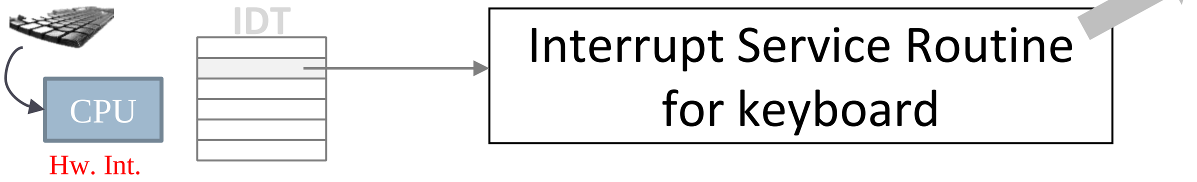
```
void **Int_hardware_Keyboard** ( idDevice )
{
            •  idDevice -> HardwareID
            •  Key = readPort(HardwareID)
            •  Insert(Key, DataKeyboard.Buffer)
            •  InsertPendTask(&listPendTasks,
                                   Int_software_Keyboard);
            •  activate_int_SW();
}
```

Interrupción hardware treatment (1/2)

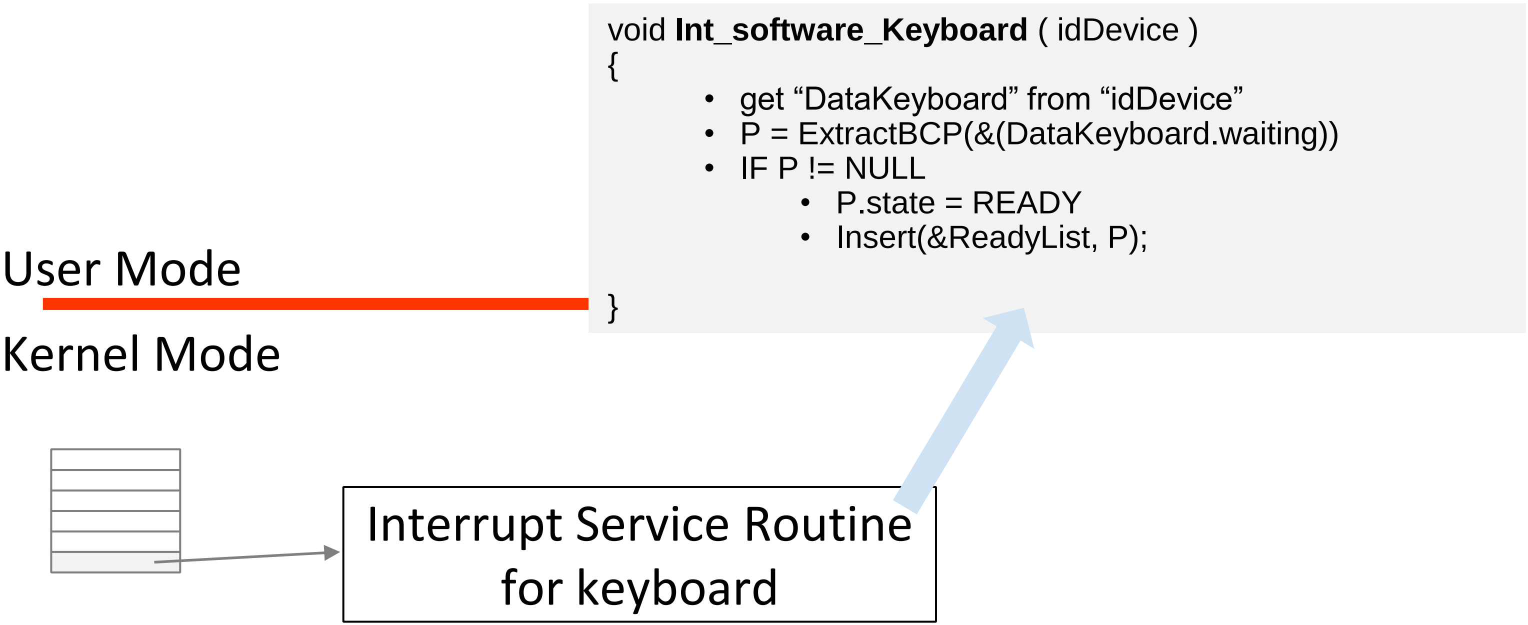
```
void Int_hardware_Keyboard ( idDevice )
{
        •  idDevice -> HardwareID
        •  Key = readPort(HardwareID)
        •  Insert(Key, DataKeyboard.Buffer)
        •  InsertPendTask(&listPendTasks,
                          Int_software_Keyboard);
        •  activate_int_SW();
}
```

User Mode

Kernel Mode



Software interrupt
treatment (1/2)



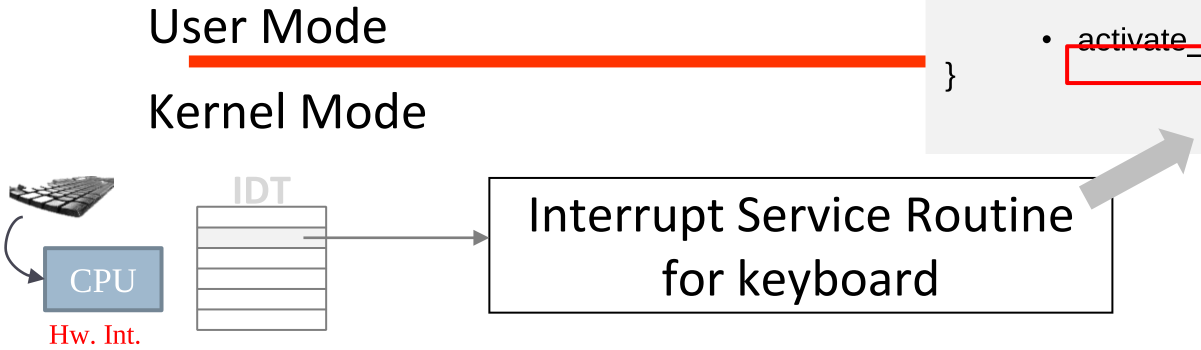
```
void **Int_software_Keyboard** ( idDevice )
{
        • get "DataKeyboard" from "idDevice"
        • P = ExtractBCP(&(DataKeyboard.waiting))
        • IF P != NULL
                • P.state = READY
                • Insert(&ReadyList, P);

}
```

Interrupt with minimal priority: it will be executed when no more critical task are present

Interrupción hardware

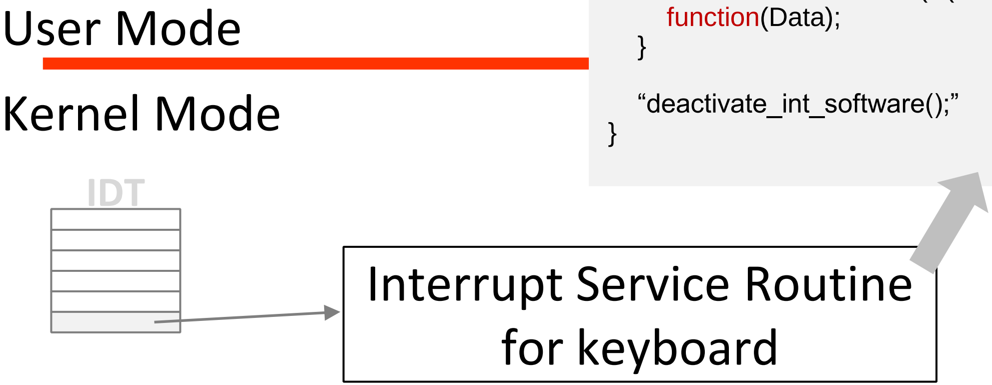
treatment (2/2)

```
void Int_hardware_Keyboard ( idDevice )
{
          •  idDevice -> HardwareID
          •  Key = readPort(HardwareID)
          •  Insert(Key, DataKeyboard.Buffer)
          •  InsertPendTask(&listPendTasks,
                                Int_software_Keyboard);
          •  activate_int_SW();
}
```



Software interrupt
treatment (2/2)

```
void hnd_softwareInterrupt ()    /*  treatment of software interrupts */
{
    void (*function)(void *);
    void *Data = NULL;

    mientras ( thereArePendTasks(listPendTasks) )
    {
        extractFirstPendTask(&(listPendTasks), &(function), &(Data));
        function(Data);
    }

    “deactivate_int_software();”
}
```



Interrupt Service Routine for keyboard

Interrupt with minimal priority: it will be executed when no more critical task are present

Software interrupt

types of treatment in Linux

- **Bottom-Halves (BH):**
  - It was the first implementation of soft.int. in Linux. (removed in k2.6.x)
  - They are always executed in serie (no matters the number of CPUs).
    There are only 32 handlers (previously registered).
- **Softirqs:**
  - Softirq of the same type can be run in parallel on different CPUs.
    There are only 32 handlers (previously registered).
  - For example, system timer uses softirqs.
- **Tasklets**
  - Similar to softirqs except that there is no limit, and easier to use (for programming).
  - All the tasklets are tunneled through a softirq, so same tasklet can not be run at the same time on several CPUs.
- **Work queues**
  - The top-half is said to be executed in the context of an interrupt => it is not associated with a process.
    Without such association the code can not “go sleep” or be blocked.
  - Work queues are executed in the context of a process and have skills of a kernel thread.
    They have a set of useful functions for creation, planning, etc.

http://www.ibm.com/developerworks/linux/library/l-tasklets/index.html

Software interrupt

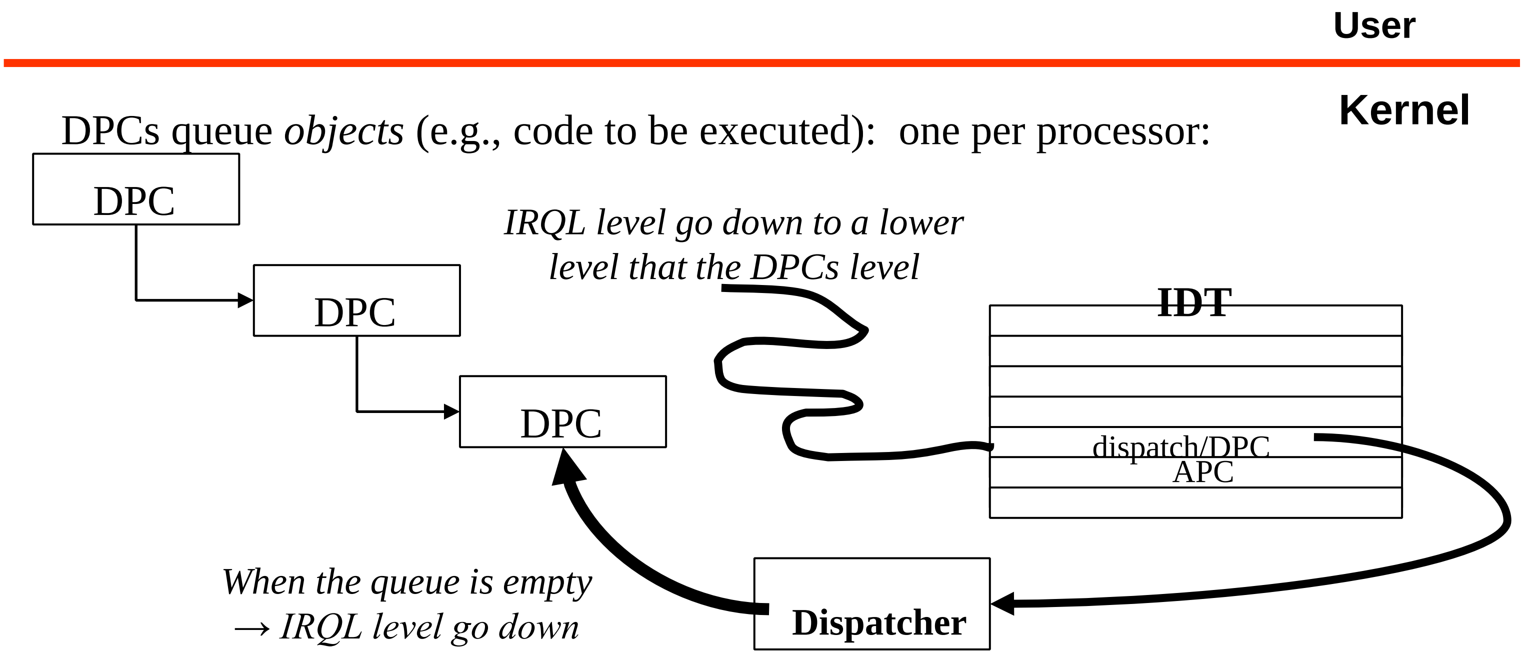
types of treatment in Windows

- Deferral Procedure Calls (DPCs):
  - Common to the entire operating system (a single queue per CPU)
  - They perform deferred tasks that have been enqueued:
    - To complete I/O operations of the controllers.
    - Processing timers expiration.
    - Release of waiting threads.
    - Force re-scheduling when a slice of time expires.

- Asynchronous Procedure Calls (APCs):
  - Individuals to each thread (each thread has its own queue).
    - The thread must give its permission for its APC to run.
  - They can be executed from system mode or user mode.
    - System: allows operating system code to be executed in the context of a thread.
    - User: used by some I/O APIs on Win32

Software interrupt

types of treatment in Windows: DPC



User

Kernel

DPCs queue *objects* (e.g., code to be executed): one per processor:

DPC

DPC

DPC

IRQL level go down to a lower level that the DPCs level

IDT

dispatch/DPC

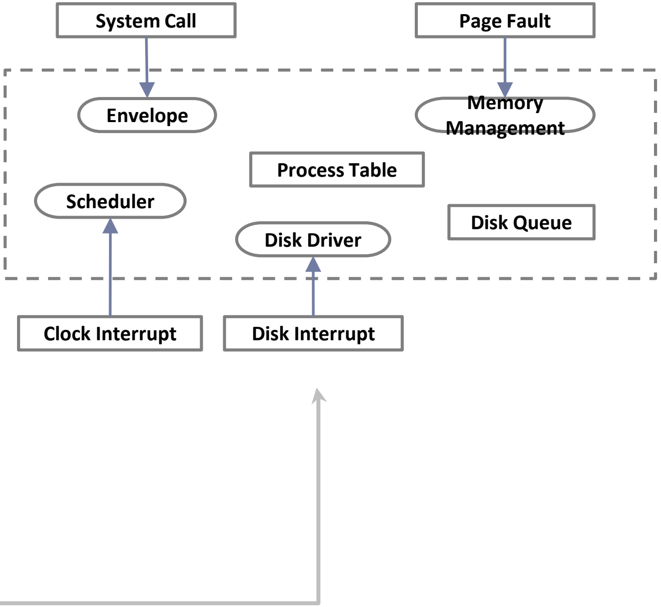
APC

When the queue is empty → IRQL level go down

Dispatcher

Overview

- Introduction
- **How an operating system works**
  - System boot
  - Characteristics and event handling
  - **Kernel process**
- Other aspects
  - Events concurrency
  - Add new system functionalities



Scenarios where the O.S. is present

- System boot
- Events treatment
  - Hardware interrupts
  - Exceptions
  - System calls
  - Software interrupts
- Kernel process
  - It will do Operating System tasks that are better performed within the context of an independent process
    - E.g.: they can perform blocking requests
  - Compiten con el resto de procesos por la CPU
    - The scheduler use to give more priority to them

Different kinds of process

- User process
  - With non-administrator (user) permissions (e.g.: no root user)
  - Only executes in privilege mode if:
    - It needs to resolve a system call it invokes (fork, exit, etc.)
    - It needs to treat an exception that the process itself has fired (0/0, *(p=null), etc.)
    - It needs to treat an interrupt that occurs while this process was executing (TCPpk, ...)
- System process
  - With the administrator (user) permissions (e.g.: root user)
  - It executes in privilege mode as an user process
- Kernel process
  - Belong to the kernel (it does not belong to any user)
  - It always be executed in privilege mode

Kernel process

Example in Linux

- kworker, ksoftirqd, irq, rcuob, rcuos, watchdog, …

```
  PID USUARIO    PR  NI    VIRT    RES    SHR S  %CPU %MEM     HORA+ ORDEN
    1 root       20   0   34100   3484   1500 S   0,0  0,0   0:00.98 init
    2 root       20   0       0      0      0 S   0,0  0,0   0:00.00 kthreadd
    3 root       20   0       0      0      0 S   0,0  0,0   0:00.12 ksoftirqd/0
    5 root        0 -20       0      0      0 S   0,0  0,0   0:00.00 kworker/0:0H
    7 root       20   0       0      0      0 S   0,0  0,0   0:14.27 rcu_sched
    8 root       20   0       0      0      0 S   0,0  0,0   0:08.35 rcuos/0
    9 root       20   0       0      0      0 S   0,0  0,0   0:05.92 rcuos/1
   10 root       20   0       0      0      0 S   0,0  0,0   0:06.10 rcuos/2
   11 root       20   0       0      0      0 S   0,0  0,0   0:06.28 rcuos/3
   12 root       20   0       0      0      0 S   0,0  0,0   0:00.00 rcu_bh
   13 root       20   0       0      0      0 S   0,0  0,0   0:00.00 rcuob/0
   14 root       20   0       0      0      0 S   0,0  0,0   0:00.00 rcuob/1
   15 root       20   0       0      0      0 S   0,0  0,0   0:00.00 rcuob/2
   16 root       20   0       0      0      0 S   0,0  0,0   0:00.00 rcuob/3
   17 root       rt   0       0      0      0 S   0,0  0,0   0:00.29 migration/0
   18 root       rt   0       0      0      0 S   0,0  0,0   0:00.10 watchdog/0
   19 root       rt   0       0      0      0 S   0,0  0,0   0:00.10 watchdog/1
   20 root       rt   0       0      0      0 S   0,0  0,0   0:00.19 migration/1
   21 root       20   0       0      0      0 S   0,0  0,0   0:00.32 ksoftirqd/1
   22 root       20   0       0      0      0 S   0,0  0,0   0:00.00 kworker/1:0
   23 root        0 -20       0      0      0 S   0,0  0,0   0:00.00 kworker/1:0H
   24 root       rt   0       0      0      0 S   0,0  0,0   0:00.09 watchdog/2
   25 root       rt   0       0      0      0 S   0,0  0,0   0:00.25 migration/2
   ...
```

http://www2.comp.ufscar.br/lxr/source/Documentation/kernel-per-CPU-kthreads.txt

Overview

- Introduction
- **How an operating system works**
  - System boot
  - Characteristics and event handling
  - Kernel process
- Other aspects
  - **Events concurrency**
  - Add new system functionalities

Concurrence in multiprocessors

- **UP**: Uni-Processing.
  - Operating System and applications are executed only in one CPU.
  - Simple but worst performance.
- **ASMP**: Asymmetric MultiProcessing.
  - Operating System is executed in one CPU (not all CPU are able to execute the O.S.).
  - Simple but performance could be improved.
- **SMP**: Symmetric MultiProcessing.
  - Operating System can be executed in any CPU.
  - Difficult because synchronization mechanism are needed in order to protect the critical sections.

    E.g.: lock the interruptions is not enough to stop O.S. executing in other CPU.

Example of basic mechanisms…

Linux

Technique	Scope	Skel. example

Disable Interrupts	• One CPU only	`unsigned long flags;`
		`local_irq_save(flags);`
		`/* ... SC: sección crítica ... */`
		`local_irq_restore(flags);`

Spin Locks	• All CPU
	• Busy wait:
	  • NOT sleep, sched., etc. on C.S.
		`#include <linux/spinlock.h>`
		`spinlock_t l1 = SPIN_LOCK_UNLOCKED;`
		`spin_lock(&l1);`
		`/* ... SC: sección crítica ... */`
		`spin_unlock(&l1);`

Mutex	• All CPU
	• Blocking wait:
	  • NOT used on HW. int.
		`#include <linux/mutex.h>`
		`static DEFINE_MUTEX(m1);`
		`mutex_lock(&m1);`
		`/* ... SC: sección crítica ... */`
		`mutex_unlock(&m1);`

Atomic Operations	• All CPU	`atomic_t a1 = ATOMIC_INIT(0);`
		`atomic_inc(&a1);`
		`printk(“%d\n”, atomic_read(&a1));`

Ejemplo de mecanismos compuestos…

Linux

IMPORTANTE

Technique	Scope	Skel. example

RW locks	• All CPU
	• Busy wait:
	  • NOT sleep, sched., etc. on C.S.
		`rwlock_t x1 = RW_LOCK_UNLOCKED;`
		`read_lock(&x1);`
		`/* ... SC: sección crítica ... */`
		`read_unlock(&x1);`
		`write_lock(&x1);`
		`/* ... SC: sección crítica ... */`
		`write_unlock(&x1);`

Spin Locks + irq	• All CPU
	• Busy wait and no interrup.:
	  • NOT sleep, sched., etc. on C.S.
		`spinlock_t l1 = SPIN_LOCK_UNLOCKED;`
		`unsigned long flags;`
		`spin_lock_irqsave(&l1, flags);`
		`/* ... SC: sección crítica ... */`
		`spin_unlock_irqrestore(&l1, flags);`

RW locks + irq	• All CPU
	• Busy wait and no interrup.:
	  • NOT sleep, sched., etc. on Critial Section (C.S.)
		`read_lock_irqsave();`
		`read_lock_irqrestore();`
		
		`write_lock_irqsave();`
		`write_lock_irqrestore();`

Chained execution of event treatment

IMPORTANTE

Event in execution	Event that comes	Usual treatment
Hw. Int. / exception	Hw. Int. / exception	• Always allowed, never or only more priority ones (if C.S. then disabled).
Sys. call / Sw. Int.	Hw. Int. / exception	• Interruptible always (if C.S. then disabled).
Hw. Int. / exception	Sys. call / Sw. Int.	• Can not be interruptible.
Sys. call / Sw. Int.	Sys. call / Sw. Int.	• Non-preemptible Kernel • Non-interruptible (queued). • Old UNIX and Linux some time ago. • Preemptible Kernel. • Critical sections must be protected. • Solaris, Windows 2000, etc.

Chained execution of event treatment

Linux

Kernel Control Path	UP protection	*MP Protection
Exceptions	**Mutex**	**-**
Hw. Int.	**Deshabilitar Int.**	Spin Lock
Sw. Int.	**-**	Spin Lock (SoftIrq, N Tasklets)
Exceptions + Hw. Int.	**Deshabilitar Int.**	Spin Lock
Exceptions + Sw. Int.	**Encolar Sw. Int.**	Spin Lock
Hw. Int. + Sw. Int.	**Deshabilitar Int.**	Spin Lock
Exc. + Int HW. + Sw. Int.	**Deshabilitar Int.**	Spin Lock

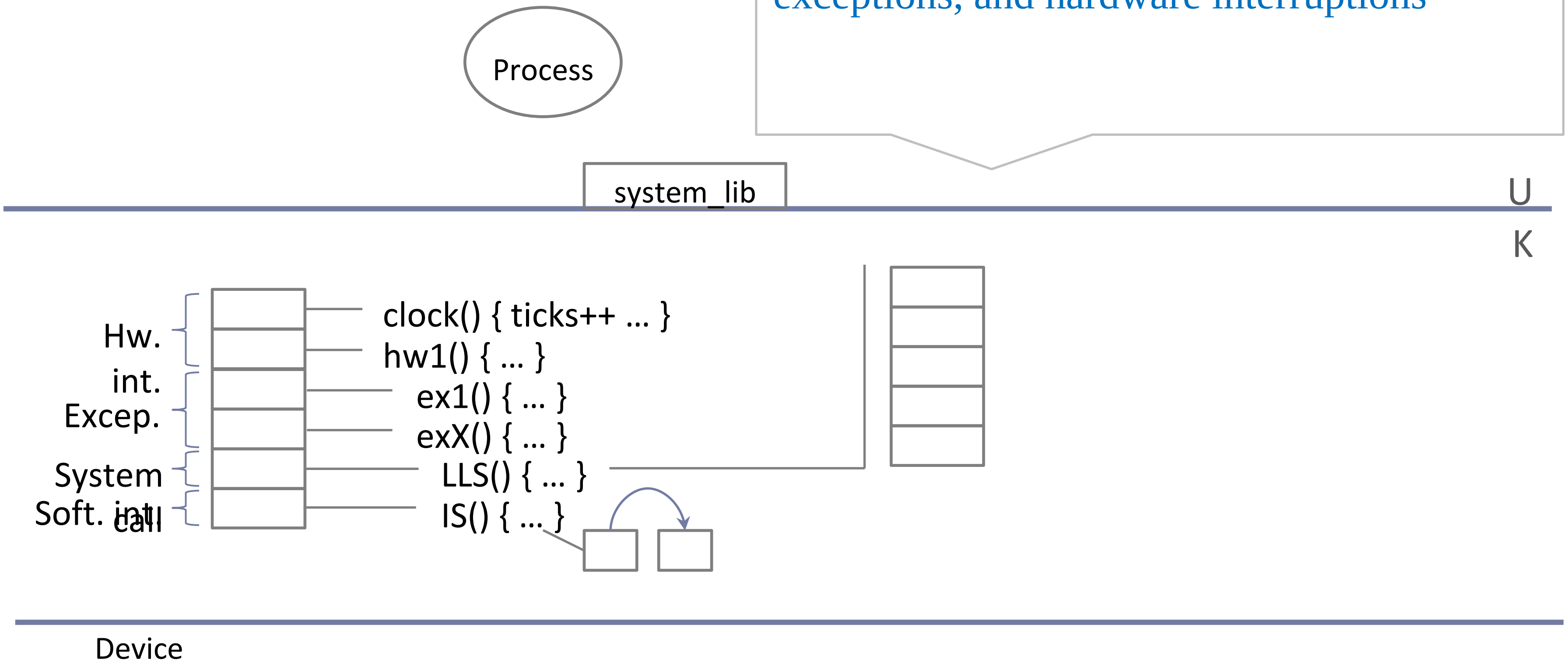
Understanding the Linux Kernel (página 218)

Overview

- Introduction
- **How an operating system works**
  - System boot
  - Characteristics and event handling
  - Kernel process
- Other aspects
  - Events concurrency
  - **Add new system functionalities**

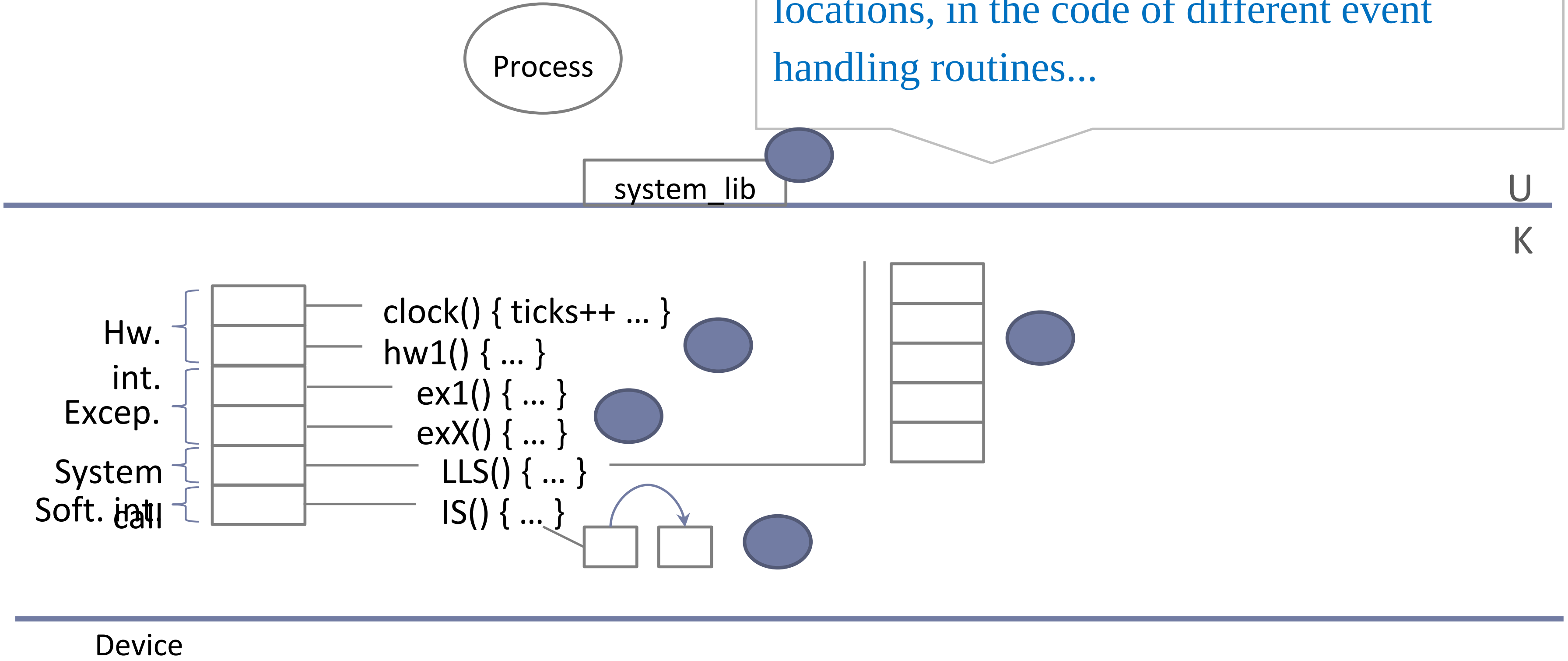
Context…

Internal operation of the kernel divided among: software interrupts, system calls, exceptions, and hardware interruptions



Context…

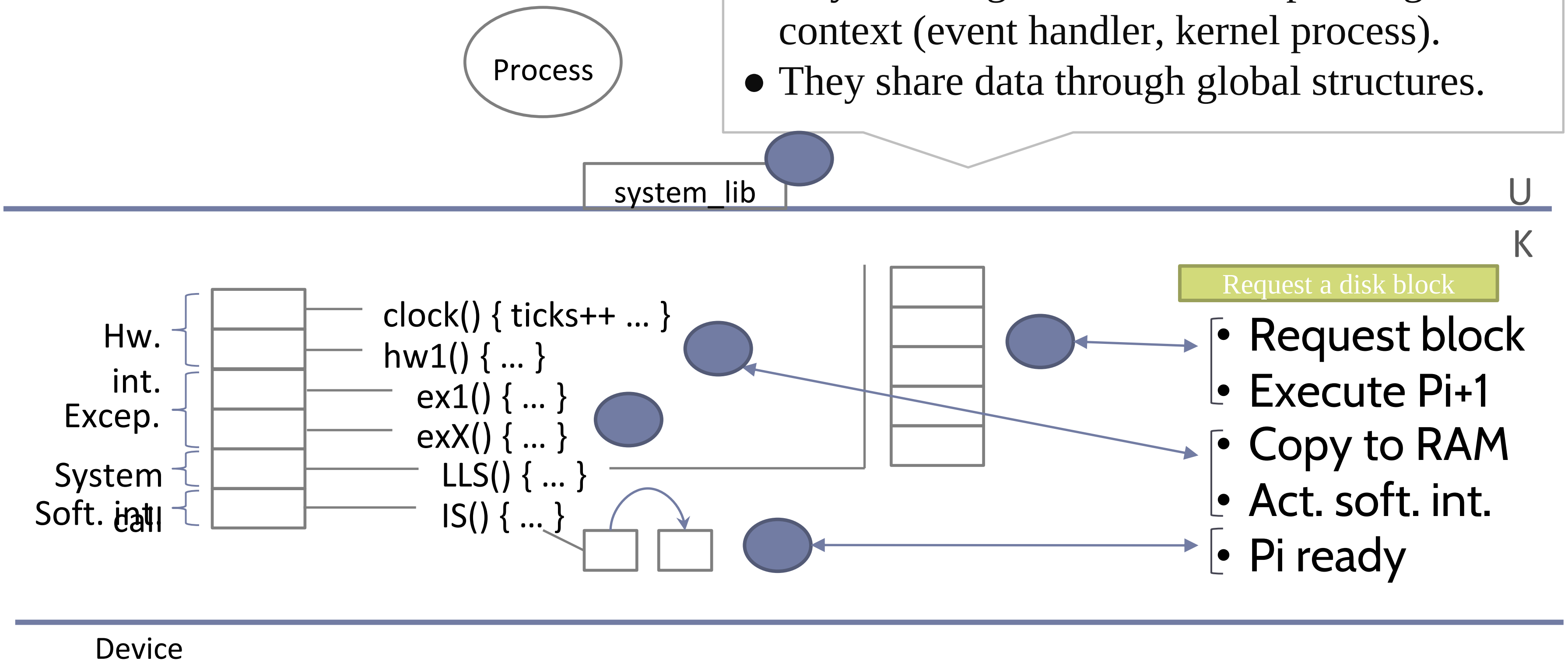
An operating system functionality (existing or to be added) is distributed in different locations, in the code of different event handling routines…



Context…

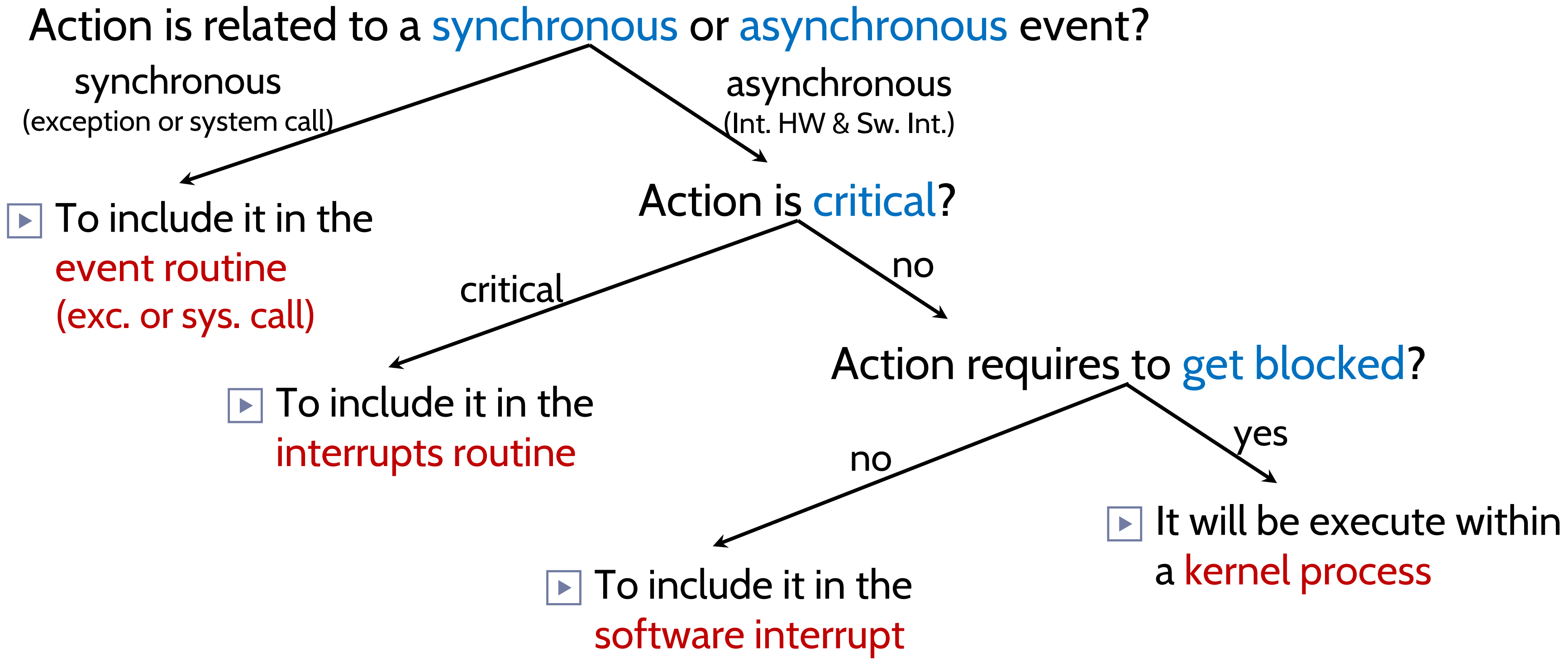
A functionality is a sequence of tasks:

- They can occur at different times, they are assigned to the corresponding context (event handler, kernel process).
- They share data through global structures.



Decision tree for matching the execution context for a new action

IMPORTANTE



Action is related to a synchronous or asynchronous event?

synchronous
(exception or system call)

- To include it in the event routine (exc. or sys. call)

asynchronous
(Int. HW & Sw. Int.)

Action is critical?

critical

- To include it in the interrupts routine

no

Action requires to get blocked?

no

- To include it in the software interrupt

yes

- It will be execute within a kernel process

Lesson 2
How an operating system works

Group ARCOS

Operating System Design

Degree in Computer Science and Engineering

Universidad Carlos III de Madrid