

# Operating Systems: shots

What is an operating system?



# Operating Systems: what1

One of several layers of software between the hardware and the user.

It does no productive work itself but supplies functions to carry out common tasks:

- Starting and stopping programs;
- Managing memory;
- Handling I/O;
- Filesystem access;
- Networking;
- Protection;
- Error handling & recovery.

It should do all of this efficiently!

# Operating Systems: what2

An O/S makes the hardware more user-friendly?

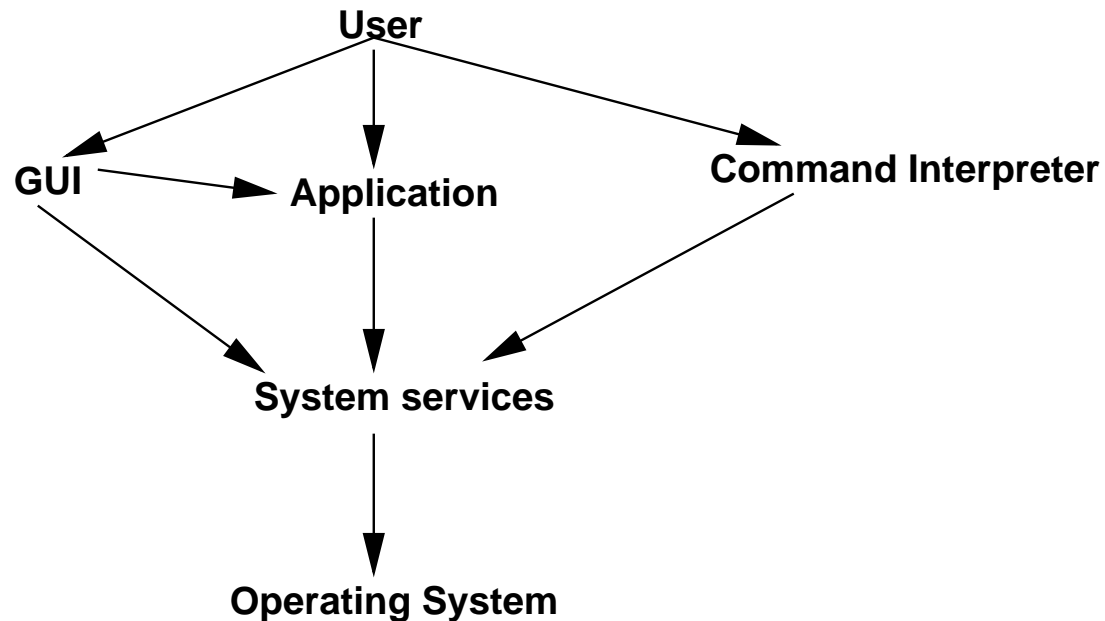
You might think that an O/S is

- icons that you click on

or

- a prompt, at which you enter commands

but the reality is



# Operating Systems: System services (functions)

The real interface to the operating system.

A set of hundreds of functions provided by the O/S

Each O/S has its own set of functions

Makes porting difficult

Hence POSIX

- Portable Operating System interface
- IEEE backed
- provided by most modern O/S

It's an interface definition not an O/S

Tell the compiler to use POSIX mode

# Operating Systems: functions 2

System calls are defined as C functions

They return a value which **MUST** be checked.

The meaning of the value depends on the particular call

- -1 is a common failure value

but doesn't tell us about the cause of failure.

the C function `perror()` will print the cause based on a global system value, `errno`

# Operating Systems: functions 3

Modern CPU's operate in at least 2 modes

- user mode: can execute a subset of instructions
- kernel (supervisor) mode: can execute all instructions

Programmer view	:-	read()	write()
O/S view uses numbers:-		3	4

found in <asm/unistd.h>

The C library function does little!

- puts system call number in register EAX
- puts parameters into registers starting with EBX
- executes instruction INT 80

# Operating Systems: functions 4

The user process triggers kernel activity with the use of INT 0x80

The cpu goes to a lookup table, the Interrupt Descriptor Table (IDT) and looks up the entry at location 0x80.

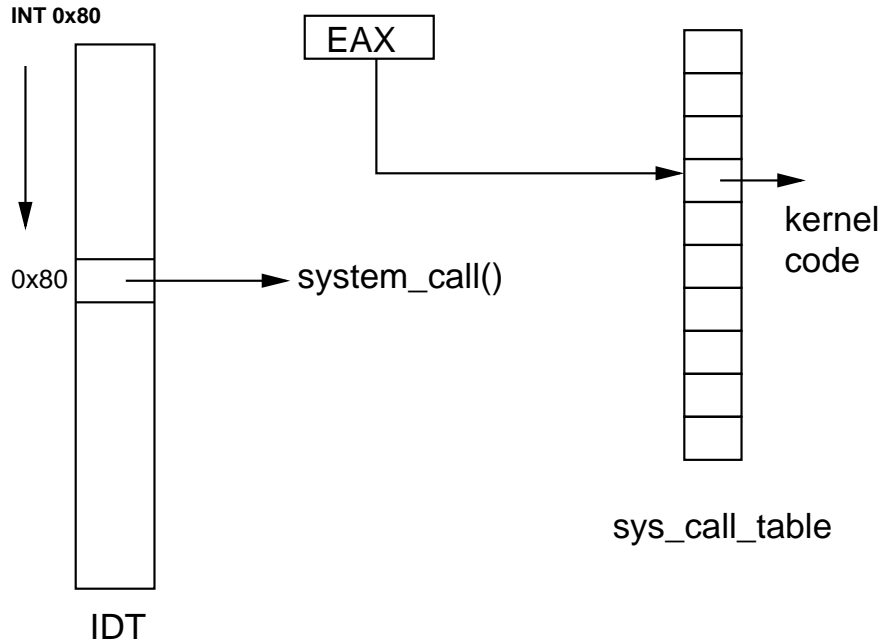
This has 2 effects on the cpu.

- A switch to kernel mode takes place
- A jump to the system call handler, `system_call()`, in the kernel takes place

The kernel does some housekeeping and then looks up the particular call required in another table, `sys_call_table`

The kernel uses the value passed in register EAX as the index into this table and jumps to the appropriate routine

# Operating Systems: functions 5



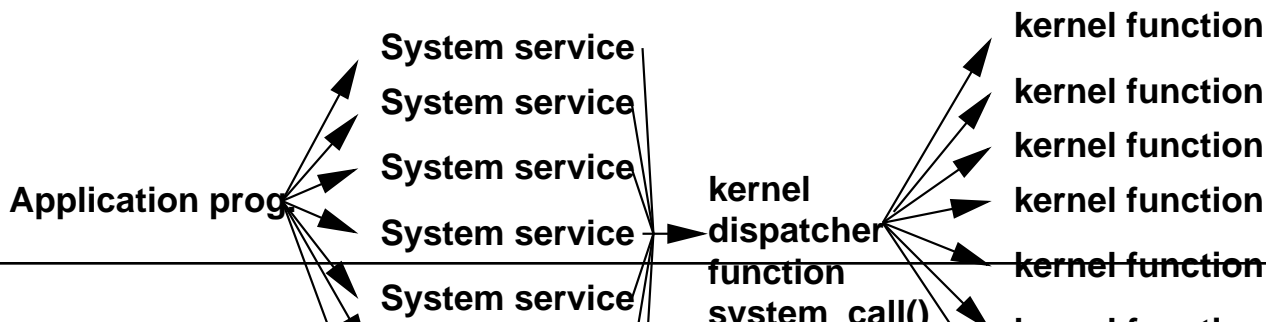
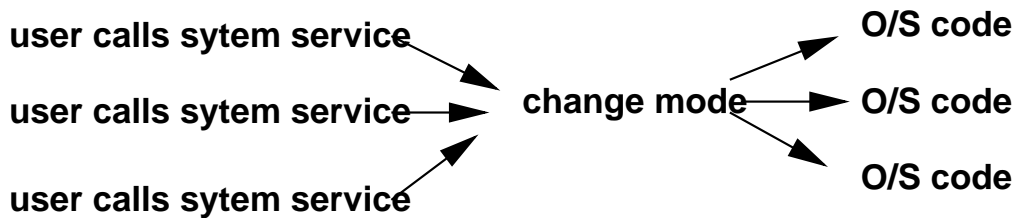
The kernel uses IRET to return from the call, re-entering user mode, This takes us back to system\_call() which restores the user registers and returns an appropriate value to the C function.

The C function then performs a series of tasks ....

# Operating Systems: functions 6

## The C function

- determines whether the system call succeeded
  - if failure then
    - set errno appropriately
    - return -1
  - else
    - return data in correct format
- returns control to the calling program



# Operating Systems: A breather :)

So why study operating systems?

You may rarely program at the system call interface but ...

it will distinguish the professionals from mere users.

Support informed decisions when selecting an O/S

All O/S are tunable, but you need to know how they work in order to tune them.

As a programmer you might need to use system calls

O/S's are amongst the largest pieces of software designed & written.