1.2 The OS structure

A hierarchical organization

An OS is a complex software system, comprising potentially millions of lines of code. To manage the complexity, designers rely on abstraction and virtualization to subdivide the system into multiple layers, such that the development and subsequent maintenance can be confined to smaller subsystems with well-defined interfaces.

The ***kernel*** of an OS is the minimal set of functions necessary to manage the system resources safely and efficiently. The kernel typically provides the most essential services for memory and device management, for creating and managing units of computation, and for communication among the different concurrent activities within the system.

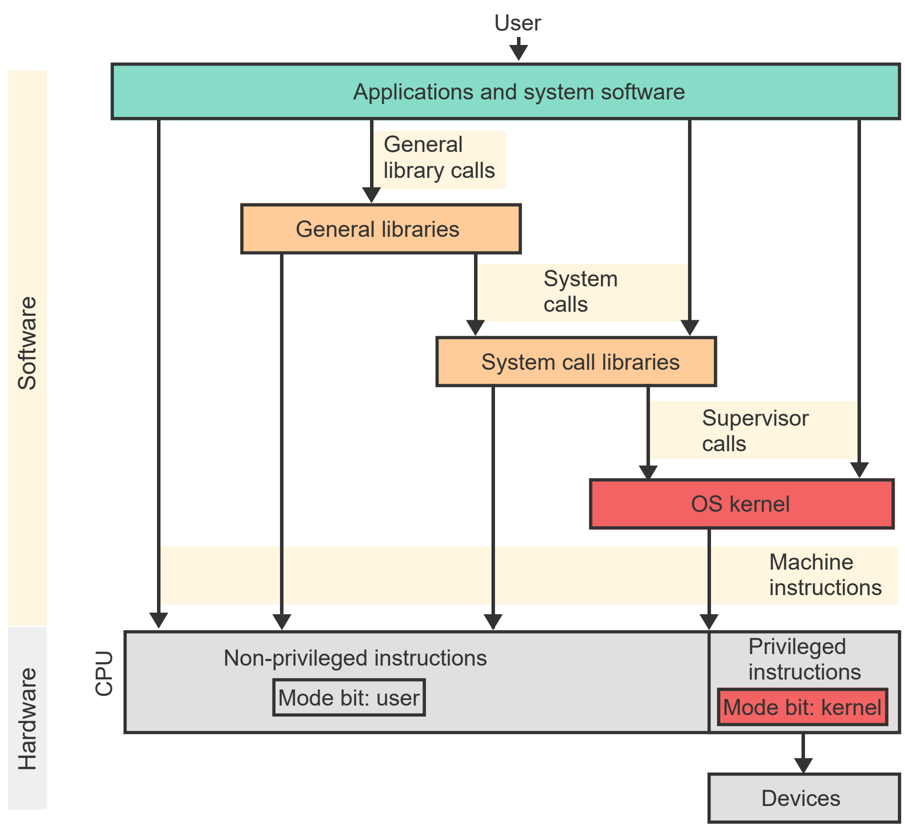
To deal with issues of security, the instruction set provided by the CPU is divided into two sets: privileged and non-privileged. A ***privileged instruction*** performs critical operations that access I/O devices and the CPU's status and control registers. Thus only the OS kernel is allowed to execute privileged instructions.

To guarantee that no programs outside of the trusted kernel can use privileged instructions, the CPU operates in two different modes, indicated by a special mode bit. ***Kernel mode*** is the CPU state where both privileged and non-privileged instructions may be used. ***User mode*** is the CPU state where only non-privileged instructions may be used. Any attempt to execute a privileged instruction in user mode automatically transfers control to the kernel.

The functionality of the kernel is extended by a set of libraries, which utilize kernel functions to implement higher-level system functions. Additional libraries provide a wide range of other services to the users by supporting abstractions above the system calls. Applications and system software, including editors, compilers, and interpreters, form the highest layer of software. Depending on the type of application or service, software at the top level may utilize library services, issue system calls, or invoke kernel services directly.

PARTICIPATION ACTIVITY | 1.2.1: The OS hierarchy.

Start 2x speed



Captions

1. All software comprising a computer system can be divided into several levels of abstraction.

2. The OS kernel is the lowest level of software and provides the basic functionalities necessary for efficiently and safely executing programs.
3. The kernel is extended by library functions, which invoke kernel functions to offer more convenient service functions to higher-level software.
4. Other library functions provide additional abstractions to carry out mathematical functions, string manipulations, and many other support tasks.
5. Applications and system software may invoke kernel functions directly or use the more convenient higher-level abstractions provided by library functions.
6. Software at all levels executes CPU instructions, which are divided into two classes: privileged and non-privileged. A mode bit in the CPU determines which instruction set may be used.
7. Privileged instructions, which access I/O devices and various control and status registers of the CPU, are restricted to the kernel and can only be executed in kernel mode.

Feedback?

PARTICIPATION ACTIVITY | 1.2.2: Principles of the OS hierarchy.

1) Kernel functions may be invoked by _____.

- ○ only system call libraries
- ○ only system call or general libraries
- ◉ any program

Correct

Any program is allowed to issue a supervisor call to request service directly from the kernel.

2) Privileged instructions may be executed by _____.

- ◉ only the kernel
- ○ the kernel and system call libraries
- ○ any program

Correct

No program outside of the kernel is allowed to use privileged instructions.

3) An application that needs to access an I/O device could not be written without _____.

- ○ kernel and library functions
- ◉ kernel functions

Correct

To access an I/O device, privileged instructions are needed, which only the kernel can execute.

4) The kernel can execute _____ instructions.

- ○ only privileged
- ◉ both privileged and non-privileged

Correct

The kernel always executes in kernel mode, which allows the use of privileged instructions in addition to non-privileged instructions.

Feedback?

The shell

When a user logs in, the OS starts a graphical user interface or a shell.

A ***graphical user interface*** (***GUI***) presets various icons on the screen, which the user can click on in different ways to invoke services associated with the icons, or to reveal pull-down menus for additional tasks.

The ***OS shell*** is a command interpreter that accepts and interprets textual commands issued by the user via a keyboard.

Any shell offers dozens of commands, each having various optional parameters, flags, and other modifiers to provide a wide spectrum of operations. The table shows several types of common commands used by Windows operating systems to illustrate the type and form of different commands. The actual format and structure of each command depends on the shell being used. Ex: Some shells require lower case letters while others permit upper case. The same type of command can be expressed by different mnemonics. Ex: "copy" vs "cp", "rm" for remove vs "del" for delete.

Table 1.2.1: Examples of generic shell commands.

Command	Description	Example
DIR	Displays a list of files and subdirectories in a directory (folder).	DIR /A:-D /O:S /S d1 Display files in the directory d1. The minus sign in the flag /A:-D asks not to display directories. The flag /O:S asks to display the files ordered by size. The flag /S asks to display the files recursively in all subdirectories starting with d1.
COPY	Copy one or more files to another location.	COPY f1+f2 Backup Copy the two files, f1 and f2, from the current directory to the directory named Backup. If Backup does not exist, a new directory is created as part of the command.
WC	Count the number of words, lines, and characters in a text file.	WC -w f The flag -w specifies that WC should count and output only words in the file f.
DEL	Delete one or more files.	DEL *.jpg z* The asterisk is a wildcard character, which can be substituted for zero or more characters in a string. Thus "*.jpg" matches all file names that have the extension ".jpg" and "z*" matches all file names that start with the character z.

Feedback?

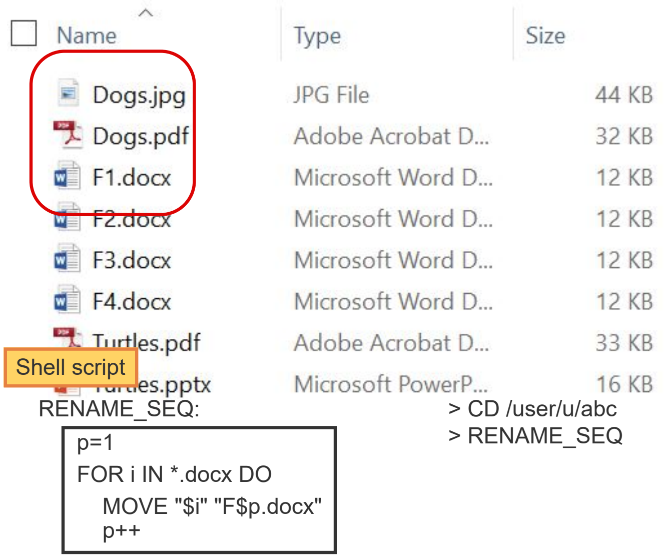
The main advantage of a GUI is the intuitive visual representation of objects. A user is able to manipulate object and invoke operations using a mouse without much training and without any knowledge of the underlying commands. A shell is the preferred interface by programmers and expert users because text-based commands are more flexible than icons by providing a wide range of parameters and control flags to modify each command.

- Multiple commands can be combined using conditional, iterative, and logic constructs to handle complex tasks efficiently. Ex: Any number of files of different types can be moved, renamed, or selectively copied in a single loop. Multiple files can be combined into one without invoking editors. Hierarchies of new directories of arbitrary complexity can be created with a single command.
- The user can create and save higher-level operations following the principles of abstraction. A ***shell script*** is a program that implements a new operation by combining multiple commands and control statement into one named unit interpreted by the shell.

PARTICIPATION ACTIVITY | 1.2.3: Using a GUI vs a shell script.

Start 2x speed

Current directory (folder) in GUI



Captions ^

1. A directory contains a large number of files of different types. The user wishes to rename all files of type docx sequentially to F1.docx, F2.docx, F3.docx, etc.
2. To start the renaming, the user right-clicks on the first file, which opens a pull-down menu.
3. The user clicks on Rename and types the new name, "F1", to replace "Birds".
4. The user then repeats the above steps for all other .docx files to perform the renaming to F1 - F4.
5. A shell allows the user to write a for-loop that iterates over all .docx files in the current directory. The same loop works regardless of the number of files in the directory.
6. Each matching file, identified by the current value $i of variable i, is renamed (MOVEd) to F$p.docx.
7. p is a variable initialized to 1 and incremented during each iteration. The current value $p of p forms part of each new name: F$p.docx.
8. The program can be saved as a shell script under a chosen name, say RENAME_SEQ.
9. The user can then select any directory, say /user/u/abc, and perform the renaming of all files in the directory by reusing the same shell script RENAME_SEQ.

Feedback?

PARTICIPATION ACTIVITY | 1.2.4: The overhead of an operation using a GUI.

A script can use an iteration variable, i. The value $i of i can be used to create a different name in each iteration. The script below creates 10 directories named D0, D1, ..., D9, where MKDIR is the command to make a directory.

```
FOR i in 0 1 2 3 4 5 6 7 8 9 DO
   MKDIR D$i
DONE
```

To create the same directories using a GUI, the user must perform some number of right and left-clicks, type some number of characters, and press Enter several times.

1) The required number of right-clicks is _____.

Check Show answer

2) The number of characters typed is

_____.

Check Show answer

3) The number of times Enter is pressed is _____.

Check Show answer

Feedback?

PARTICIPATION ACTIVITY | 1.2.5: Comparing word counts using a GUI.

A 1-line script can be written that counts the number of words in two text files and compares the counts:
IF [WC -w f1 -eq WC -w f2] THEN ECHO "equal" ELSE ECHO "not equal"
Show the order in which the different tasks below must be performed to accomplish the same goal using a GUI.

Select the definition that matches each term

1) Open file f1 using an editor (Ex: MS Word).
   - Step 1

   Correct

   To count the number of words in a file requires opening the file in an advanced text editor.

2) In the tab Review, select the function Word count.
   - Step 2

   Correct

   The editor has a menu that allows the word-count function to be applied to the open file.

3) Remember the number of words in f1.
   - Step 3

   Correct

   The number of words, along with the number of characters and lines, is displayed and must be remembered in order to proceed with the next step.

4) Repeat the previous steps for f2.
   - Step 4

   Correct

   The user must open the second file and determine the number of words in the same way.

5) Manually compare the two value of word counts.
   - Step 5

   Correct

   The two counts can then be compared manually, since invoking another application to perform the comparison would not be worth the effort.

Reset

Feedback?

System calls and supervisor calls

A ***system call*** is a request from an application for an OS service. A system call is implemented as a library function, which sets up the necessary parameters for the requested operation and issues a corresponding supervisor call.

A ***supervisor call*** (***kernel call***) is a privileged instruction that automatically transfers execution control to a well-defined location within the OS kernel. Thus supervisor calls provide the interface between the OS kernel and the higher-level software.

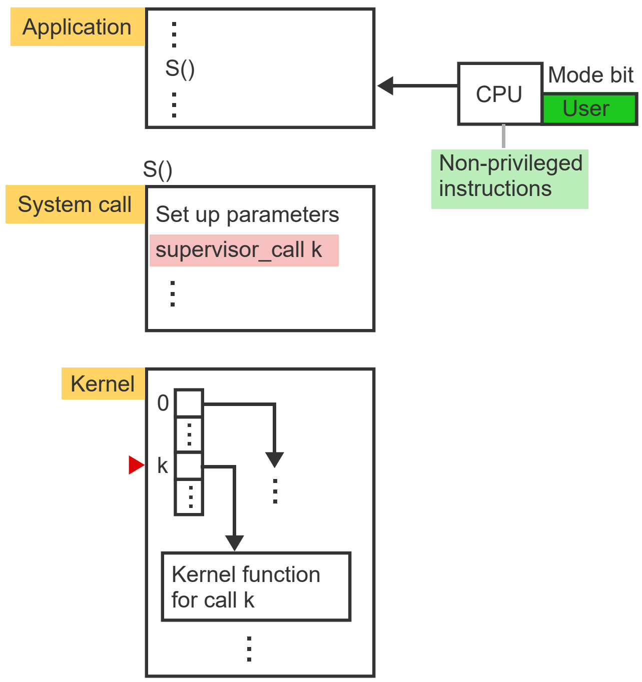
A supervisor call is similar to a function call with two special features:

1. The call switches execution from user mode to kernel mode by setting the mode bit in the CPU.
2. To prevent a call from branching to arbitrary locations within the kernel, the function to be invoked is not specified by an address but indirectly using an index into a branch vector. Thus kernel-mode execution is limited to only well-defined entry points within the kernel.

When the kernel function terminates, control is returned to the invoking library function in user mode. Depending on the type of service requested, control may return immediately or may be delayed by blocking (temporarily stopping) the invoking computation to await the completion of some event (ex: the arrival of input data from a keyboard). The system call library function then returns control to the application, also in user mode.

PARTICIPATION ACTIVITY | 1.2.6: Execution of a system call.

Start 2x speed



Captions

1. An application, running in user mode, executes a system call S() to request a service from the OS.
2. Execution transfers to the library function S(), still in user mode.
3. S sets up the parameters for the service and executes the privileged instruction supervisor_call. The parameter k identifies the kernel function using a branch vector in the kernel.
4. Execution transfers to the kernel function in kernel mode, which allows the CPU to use privileged instructions.
5. When the kernel function completes the requested service, execution returns to the system call function S in user mode.
6. The system call function completes the request and returns to the application at the instruction following S().

Feedback?

PARTICIPATION ACTIVITY | 1.2.7: System calls and supervisor calls.

1) A system call is a _____.

- privileged instruction
- library function
- kernel function

Correct

A system call is a library function that uses supervisor calls to carry out a service function.

2) A supervisor call is a _____.

- privileged instruction
- library function
- kernel function

Correct

A supervisor call is a privileged instruction that transfers control to the kernel in kernel mode.

3) A supervisor call can be executed _____.

- only in kernel mode
- by any program
- by only library functions

Correct

Any program can make direct use of a kernel function by issuing the corresponding supervisor call.

4) Changing the mode bit of the CPU _____ can only be done by a privileged instruction.

- from user mode to kernel mode
- from kernel mode to user mode
- in any way

Correct

Changing from user mode to kernel mode must be done in a controlled manner using a supervisor call instruction.

Feedback?

Interrupts and traps

An ***interrupt*** is an event that diverts the current execution of a program to a predefined location in the kernel in order to respond to an event. An interrupt is triggered by a hardware signal sent to the CPU from an external device. The two most common uses of interrupts are:

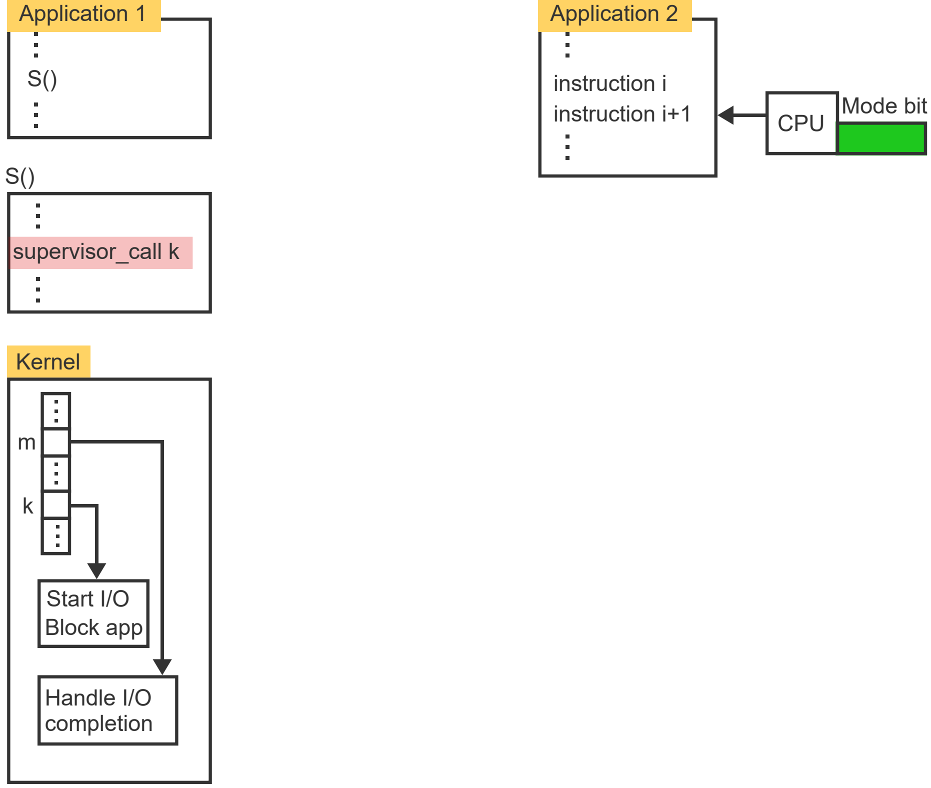
- Signal to the OS the completion of an I/O operation. The interrupt is generated by the I/O device,
- Implement time-sharing by periodically switching the CPU among multiple concurrent computations. The interrupt is generated by a countdown timer.

A ***trap*** (also called an internal interrupt) is an interrupt triggered by the currently executing instruction. Dividing by zero, executing an invalid opcode, or causing an arithmetic overflow are all errors, which result in a trap and cause the OS to abort the current program execution. Executing a supervisor call instruction is not an error but also causes a trap, since the main purpose is to transfer control to the kernel when requesting a service.

An interrupt, both external and internal, stops the execution of the current program, saves the state of the computation, and transfers control to the kernel. An ***interrupt handler*** is a kernel function, invoked whenever an interrupt occurs, that determines the cause of the interrupt and invokes the appropriate kernel function to provide the response. When the kernel function completes the requested service, the state of the interrupted computation is restored and control is returned to the instruction following the interrupt. The handling of an interrupt is thus completely transparent to the interrupted computation and provides an immediate response to asynchronous events.

PARTICIPATION ACTIVITY | 1.2.8: Processing of an interrupt.

Start 2x speed



Captions

1. Application 1 issues a system call S() requesting an I/O operation.
2. The library function S() invokes the corresponding kernel function using a supervisor call k.
3. The kernel function, in kernel mode, starts the I/O device using privileged instructions.
4. If the data transfer is expected to take a long time, application 1 is blocked.
5. The kernel selects another application to run while application 1 is waiting for the I/O device to complete the data transfer.
6. When the data transfer is complete, the device issues an interrupt to the CPU, which stops application 2 after the current instruction i and transfers control back to the kernel.
7. Using the same branch vector as a supervisor call, the kernel transfers control to function m, responsible for handling I/O completion.
8. The I/O completion routine returns to S(), which may issue another supervisor call to restart I/O and again block the application to await the I/O completion.
9. When application 1 blocks, execution transfers back to the interrupted application 2 and continues with the next instruction i+1. Application 2 is unaware of the interruption used to service application 1.

Feedback?

PARTICIPATION ACTIVITY 1.2.9: Using timeout interrupts for time-sharing.

Application 1 and 2 run concurrently. Whenever a timeout interrupt occurs, the kernel switches control between the applications. Show the order of instruction execution, assuming application 1 is currently running.

Application 1	Application 2
... instruction i (timeout interrupt) instruction i+1 ... instruction k (timeout interrupt) instruction k+1 ...	instruction 0 ... instruction j (timeout interrupt) instruction j+1 ...

Select the definition that matches each term

1) instruction i

1

Correct

Application 1 is currently running and continues until the first interrupt occurs following instruction i.

2) instruction 0

2

Correct

Execution switches to application 2, which starts with instruction 0.

3) instruction j

3

Correct

Application 2 runs until the second interrupt occurs following instruction j.

4) instruction i+1

4

Correct

Execution switches back to application 1, which continues with the instruction following the interrupt.

5) instruction k

5

Correct

Application 1 runs until the third interrupt occurs following instruction k.

6) instruction j+1

6

Correct

Execution switches back to application 2, which continues with the instruction following the interrupt.

Reset

Feedback?

CHALLENGE ACTIVITY 1.2.1: OS structure.

371440.2479476.qx3zqy7

Start

1
2
3

Two concurrent applications, a1 and a2, execute the sequences of instructions (j1, j2, j3) and (
respectively. Execution switches between the applications whenever a timeout interrupt occur
one application terminates.

If a2 starts, and interrupts occur after instructions k2 and j2, then what is the order in which th
instructions will execute?

Ex: k1, k2, j2

1 2 3

Check Next

Feedback?

EXERCISE 1.2.1: Interrupts and traps.

(a) What do interrupts and traps have in common? What are the differences between the two concepts?

Feedback?

EXERCISE 1.2.2: Multiprogramming, time-sharing, and interrupts.

(a) Is multiprogramming possible without interrupts?

(b) Is time-sharing possible without interrupts?

Feedback?