Operating Systems Notes

1 OSs and Architectures

Different architectures dictates the way that an operating system must conduct its business.

As different architecture platforms have different instruction sets, it determines what are viable methods for accomplishing tasks such as *memory protection* and *control interrupts*.

1.1 OS Structure

The OS acts as an intermediary between software and hardware. It allows to mediate access and gives developers a nicer way of completing lower-level actions. This abstraction is achieved via *traps* and *exceptions*. Similarly, devices can gain attention via the use of *interrupts*.

1.2 Privileged Instructions

These instructions are restricted to use by the OS *only*, and includes direct access to *I/O devices* and *memory state management*.

This is achieved by the implementation of two *modes of operation*; *user* and *kernel* modes.

Privileged instructions can only be executed in kernel mode.

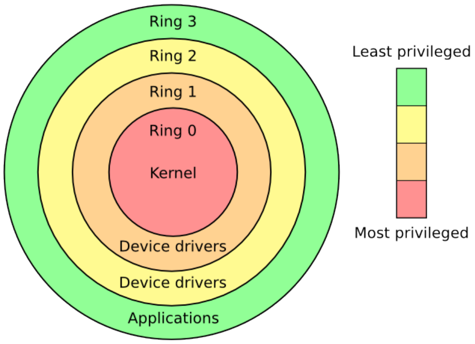

Figure 1: x86 Architecture Levels of Privilege
Courtesy of Hertzsprung of English Wikipedia

1.2.1 System Calls

If code running from within user mode tries to execute a privileged instruction, it will trigger the *illegal execution trap*, which allows for such code to gain access to privileged instructions and resources. These are called *system calls*.

An OS will define a set of system calls that makes it possible for such an interaction to take place.

Caller: The user mode process invoking the system call
Callee: The OS handling the system code

- The caller places arguments, especially the *type* of system call, in a specified location
- The callee saves the caller's state
- The callee verifies the arguments
- If valid, the code is run
- When the system call has been satisfied, the program counter is set to the return address
- Execution is returned to user mode

Figure 2: A rough breakdown of a system call

1.3 Exception Handling

A *trap* is a synchronised, intended transition that is initiated by the OS.

An *exception* is also synchronised, however it is an *unexpected* problem with some instruction.

An *interrupt* is *a*synchronous on the other hand, and is caused by some external device.

2 Memory Management

Programs must be brought from disk into mmemory and then placed into a process.

The CPU can only access data from memory, not disk.

Register access takes at most 1 CPU clock, but Main Memory can take mayn cycles, causing a *Stall*.

2.1 Base and Limit Registers

A set of *base* and *limit registers* define the logical adress space. the CPU must chech every memory access is valid between the base and the limit for that user. Failure causes a trap to the OS monitor

2.2 Virtual Address Space

Logical/Virtual addresses are independent of physical memory.

Hardware translates virtual addresses into physical ones.

Logical/Virtual addresses a process can reference is called the address space.

2.3 Memory Managment Unit (MMU)

Effectively is a hash function from logical address to physical address.

a MMU prevents the need for swapped out process to be swapped back into the same physical addresses.

Swapping is not typically supported on mobile devices, more likely to to overwite least used data.

2.4 Partitioning

Main memory is usually broken up into two partitions; The OS and user process.

Each process is contained within a single contiguous section on memory.

Realocation registers are used to protect users processes from one another and from canging the OS code.

Some old techniques include:

- Fixed Partitions - simple but causes fragmentation often
- Variable Partitions - no internal fragmentation, but can leave holes in the physical memory

Dynamic Storage-Allocation is possible using First-fit, Best-fir and Worst-fit in terms of hole filling.