

Class Notes Aug 25 2006.

We've discussed one thing we must do if we are to be able to run more than one program at a time. Namely:

1. We need a relocating loader, and an executable file format that tells us where all the address constants are.

Another obvious thing we need is:

2. memory allocation. When the operating system uses the relocating loader to load a program into memory, it must be loaded into a location that isn't being used by anything else. Thus the operating system must keep track of how much memory is being used and where things are located in memory.

We were talking about how we force programs to wait for I/O, even though the programmer might not want to cooperate. We make I/O instructions illegal for programs to execute. These instructions can be executed only by the operating system. Thus the OS will know about each I/O instruction that is executed, because only the OS can execute them. But how do we make this work? We need something called:

3. Dual mode operation.

The hardware must know whether the OS is executing or an ordinary program is executing. In program mode we I/O instructions are illegal and any program attempting to execute them will be terminated. In OS mode, I/O instructions will work just fine. We must be able to switch from one mode to another, but we can't allow an ordinary program to switch modes.

How do we switch modes? We use something called:

4. Interrupts.

They are also known as traps and system calls. Here's the idea. From time to time, a program will need the OS to do something. This includes things like performing I/O, recognizing when I/O is finished, and trapping erroneous operations, such as attempts to execute illegal instructions.

When something happens that requires OS intervention, the currently executing program must stop executing and the OS must start executing. This really nothing more than a jump. It's usually necessary to return to the place in the program where we left off, so we also need to save the return address somewhere. In other words, what we need is something very much like a subroutine call, but it is not caused by executing an instruction. It is generated internally in the hardware. (Usually.) The jump target must be inside the operating system, and it must be impossible for an ordinary program to change

the jump target. When the jump is executed, the CPU must switch modes from program mode to OS mode.

Here is how this works. The current mode is stored in a register called the status register. Usually it's just a single bit. There are instructions that can change the status register, but they are illegal unless executed by the OS. (Such instructions are called privileged instructions.) When an interrupt occurs, for any reason, there is an address someplace that is loaded into the program counter. This address is in a location that is inaccessible to ordinary programs. This address points into the OS. In addition to replacing the PC, there is another location that contains a new value for the status register. This location is also inaccessible to ordinary programs. The new value of the status register will cause the CPU to start executing in OS mode. The previous values of the status register and PC are saved for later restoration.

The things that cause interrupts are the following:

- a. Completion of an I/O operation.
- b. An illegal operation by an application program.
- c. An unsolicited I/O event (Key-press, mouse-move)
- d. An explicit request from a program (System call instruction).
- e. Whatever else the computer designer decides is important.

Each interrupt has a new PC and Status Register (PSW). These two values taken together are called an interrupt vector. In today's computers, return address and PSW are saved in a stack. This permits nested interrupts. In many cases, nested interrupts are prevented by additional bits in the PSW. These bits are used to turn off interrupts temporarily while a certain portion of interrupt processing is taking place.

There is no way to get into the operating system except through an interrupt.

Now, what else do we need?

## 5. Memory protection.

We need to protect interrupt vectors. We need to protect the OS from application programs, and we need to protect application programs from one another. We need hardware features that permit portions of the memory to be inaccessible to certain programs. This can be easy or complicated. In today's computers virtual memory is used as a protection mechanism. In earlier computers, complex hardware mechanisms were used. We will discuss virtual memory later.

Once we have all of this, the rest is straightforward programming. We will continue by trying to figure out how to keep track of the programs running in the system and keep track of the system resources being used by each program. Most of this is straightforward if you think about it long enough. (Not all of it, however.)