11. Protection in the OS

Segments

- It really makes no difference whether a bug that destroys somebody else's data is an accidental mistake or malicious.
  - a faulty process may also try to execute data or perform operations on machine code
- Solution is to ensure that physical memory areas used for different purposes are protected from each other. This is the primary function of the MMU.
- Paged virtual memory makes such protection easy to implement but:
  - Pages are fixed size to fit into page frames
  - Areas we want to protect are often different sizes determined by process logic
- Introduce idea of segments which are contiguous regions of memory in a process address space of any size determined by the process logic:
- In virtual memory systems, a segment must be an integral number of pages.
- When a process is executing, it is not allowed to modify any word in memory that isn't within a segment owned by the process
- MMU *protection* operates at segment level, *physical placement* at page level.

Violations of Memory Protection

- All memory accesses must be checked to see if a segment violation has occurred.
  - If so a memory fault is signalled.
  - In a virtual memory system the handler will check if a page fault has occurred, but if there is no page on disk for this access, process is aborted before damage is done.
- Since this must be done every memory cycle it must be done by the MMU.
- The MMU maintains for each supported segment a base address and a size. If an effective address is not in the range supported by the currently active segment, the MMU generates a memory fault.
- The MMU is a device with its own registers, addressable by the CPU.
  - What if a faulty or malicious program accesses the MMU?
  - The MMU must be especially protected…
  - Only the OS should be allowed to modify the MMU.
  - How do we stop other programs from doing so?

User and System Mode

- Hardware support is needed to control access to the MMU
  - At all times, the processor is either in user mode or in system mode
  - The instructions that access the MMU are called privileged instructions
  - Privileged instructions can only be executed in system mode
- To implement this, the CPU contains a flip flop (called the CPU mode bit) which indicates the current mode. This is often in the status register.
- The hardware executes a privileged instruction as follows:
  - If system mode, perform the instruction
  - If user mode, cause an exception which transfers control immediately to the OS
- It is essential to ensure that the OS can execute its key portions in system mode.
- User processes must not be able to change the CPU to system mode without passing control back to the OS. (To do otherwise would be a hacker's dream!)
- On every exception the CPU automatically flips to system mode. The OS controls the location and content of all handler routines.
- A privileged instruction is used to set user mode before a user process is run.

More on System Mode

- Every operation that might be dangerous must be executed in system mode
  - Only the Operating System is allowed to perform these operations
  - The user program can only request such an operation to be performed
- Every I/O operation is potentially dangerous
  - I/O is complicated, so bugs are likely if users do it themselves
  - Users don't want to write I/O code anyway, since it's so complicated
  - Incorrect I/O could result in reading confidential data, or destroying data
- When your program performs an I/O operation, what it really does is build a request: a data structure that describes what operation it wants to perform.
  - Then your program executes a TRAP.
  - The OS takes control from your program and examines the request you made.
  - If a legitimate request (e.g. read a line from my own file) OS then executes privileged instructions to perform the action
  - If not legitimate (e.g. read data from somebody else's file) OS refuses the action!

Levels of Protection

- We have examined the low level basis of protection
  - Implementation requires cooperation between hardware and OS.
  - But there are many higher levels of protection which an OS uses to determine whether a requested operation is legitimate (protection policy).
- We want to have both
  - Security so that illegitimate access to information is prevented
  - Flexibility so that we can share data and use it with convenience
- When an OS supports multiple users, each non-OS process belongs to a specific user and each user has a level of privilege that determines what his/her processes are allowed to do
  - Administrator (root) can do almost anything
  - Ordinary users can modify only their own data and read data of other users who permit it.

Files

- Data in secondary memory is stored in files.
  - A file is a set of logically associated sectors.
- The MMU and system/user modes give protection to information in memory while a program is running
- Data that is stored permanently in files needs file protection, which is enforced by the operating system
- File access is essentially a specialised form of I/O
  - User programs are absolutely prohibited from performing any I/O.
  - To do I/O, a user program sends a request to the operating system.
  - The OS checks the request, to determine whether the user is the owner of the file; if the request is OK, the OS performs the actual machine language instructions to do the I/O