16.1

Virtual memory:

Virtual memory is a computer system technique which gives an application program the impression that it has contiguous working memory (an address space), while in fact it may be physically fragmented and may even overflow on to disk storage Systems that use this technique make programming of large applications easier and use real physical memory (e.g. RAM) more efficiently than those without virtual memory.

Page Fault:

A page is a fixed-length block of memory that is used as a unit of transfer between physical memory and external storage like a disk, and a page fault is an interrupt (or exception) to the software raised by the hardware, when a program accesses a page that is mapped in address space, but not loaded in physical memory.

Thrash:

Thrash is the term used to describe a degenerate situation on a computer where increasing resources are used to do a decreasing amount of work. In this situation the system is said to be thrashing. Usually it refers to two or more processes accessing a shared resource repeatedly such that serious system performance degradation occurs because the system is spending a disproportionate amount of time just accessing the shared resource. Resource access time may generally be considered as wasted, since it does not contribute to the ad -vancement of any process. In modern computers, thrashing may occur in the paging system (if there is not ‘sufficient’ physical memory or the disk access time is overly long), or in the communications system (especially in conflicts over internal bus access), etc.

16.2

Branch misprediction occurs when the CPU mispredicts the next instruction to be executed.

The CPU uses pipelining which allows several instructions to be processed simultaneously. But during a conditional jump, the next instruction to be executed depends on the result of the condition. Branch Prediction tries to guess the next instruction. However, if the guess is wrong, we are penalized because the instruction which was executed must be discarded.

Branch Target Buffer (BTB) reduces the penalty by predicting the path of the branch, computing the target of the branch and caching the information used by the branch.

16.3

Direct memory access (DMA) is a feature of modern [computers](http://en.wikipedia.org/wiki/Computer) that allows certain hardware subsystems within the computer to access system [memory](http://en.wikipedia.org/wiki/Computer_storage" \o "Computer storage)independently of the [central processing unit](http://en.wikipedia.org/wiki/Central_processing_unit) (CPU).

Without DMA, when the CPU is using [programmed input/output](http://en.wikipedia.org/wiki/Programmed_input/output), it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU initiates the transfer, does other operations while the transfer is in progress, and receives an[interrupt](http://en.wikipedia.org/wiki/Interrupt) from the DMA controller when the operation is done. This feature is useful any time the CPU cannot keep up with the rate of data transfer, or where the CPU needs to perform useful work while waiting for a relatively slow I/O data transfer. Many hardware systems use DMA, including [disk drive](http://en.wikipedia.org/wiki/Disk_drive)controllers, [graphics cards](http://en.wikipedia.org/wiki/Graphics_card), [network cards](http://en.wikipedia.org/wiki/Network_card) and [sound cards](http://en.wikipedia.org/wiki/Sound_card). DMA is also used for intra-chip data transfer in [multi-core processors](http://en.wikipedia.org/wiki/Multi-core_processor). Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without a DMA channel. Similarly, a processing element inside a multi-core processor can transfer data to and from its local memory without occupying its processor time, allowing computation and data transfer to proceed in parallel.

By using DMA, drivers can access the memory allocated to the user level buffer / pointer.

16.4

1. The keyboard sends a scan code of the key to the keyboard controller (Scan code for key pressed and key released is different).

2. The keyboard controller interprets the scan code and stores it in a buffer.

3. The keyboard controller sends a hardware interrupt to the processor. This is done by putting signal on “interrupt request line”: IRQ 1.

4. The interrupt controller maps IRQ 1 into INT 9.

5. An interrupt is a signal which tells the processor to stop what it was doing currently and do some special task.

6. The processor invokes the “Interrupt handler”. CPU fetches the address of “Interrupt Service Routine” (ISR) from “Interrupt Vector Table” maintained by the OS (Processor use the IRQ number for this).

7. The ISR reads the scan code from port 60h and decides whether to process it or pass the control to program for taking action.

16.6

For more complex schemes, you could set individual permissions on the range of memory pages representing the stack section you care about. For example we can mark a page as read\_only or read\_write, This will cause the code accessing the region to go through an exception on access to the specific section of the stack.

16.7

We can use following ways:

» Use obfuscators.

» Do not store any data (string, etc) in open form. Always compress or encode it.

» Use a static link so there is no DLL to attack.

» Strip all symbols.

» Use a .DEF file and an import library to have anonymous exports known only by their export ids.

» Keep the DLL in a resource and expose it in the file system (under a suitably obscure name, perhaps even generated at run time) only when running.

» Hide all real functions behind a factory method that exchanges a secret (better, proof of knowledge of a secret) for a table of function pointers to the real methods.

» Use anti-debugging techniques borrowed from the malware world to prevent reverse engineering (Note that this will likely get you false positives from AV tools ).

» Use protectors.