Windows CE

Windows CE (officially known as Windows Embedded), is an edition of Windows that runs on minimalistic computers, like satellite navigation systems, and uncommonly mobile phones. Windows Embedded runs as CE, rather than NT, which is why it should not be mistaken for Windows XP Embedded, which is NT. Windows CE was used in the Sega Dreamcast along with Sega's own proprietary OS for the console.

Windows Lifecycle Policy

Microsoft has stopped releasing updates and hotfixes for many old Windows operating systems, including all versions of Windows 9x, and earlier versions of Windows NT. Windows versions prior to XP are no longer supported, with the exception of Windows 2000, which is currently in the Extended Support Period, that will end on July 13, 2010. No new updates are created for unsupported versions of Windows.

DESIGN ISSUES

WINDOWS RESOURCE MANAGEMENT
This has do with the management of all the resources in the component of the Operating System e. g the Memory, the Central Processing Unit, I/O devices and other components.

✓ PROCESS MANAGEMENT

Processor

A circuit designed to automatically perform lists of logical and arithmetic operations.
Unlike microprocessors, processors may be designed from discrete components rather than be a monolithic integrated circuit.

A process is running program containing one or more threads. A process encapsulates the protected memory and environment for its threads.

Processes and Threads

In addition to being a preemptive multitasking operating system, Windows NT is also multithreaded, meaning that more than one thread of execution (or ***thread***) can execute in a single task at once.

A process comprises:

- A private memory address space in which the process's code and data are stored.
- An access token against which Windows NT makes security checks.
- System resources such as files and windows (represented as object handles).
- At least one thread to execute the code.

A thread comprises:

- A processor state including the current instruction pointer.
- A stack for use when running in user mode.
- A stack for use when running in kernel mode.

Since processes (not threads) own the access token, system resource handles, and address space, threads do NOT have their own address spaces nor do they have their own access token or system resource handles. Therefore, all of the threads in a process SHARE the same memory, access token, and system resources (including quota limits) on a "per-process" rather than a "per-thread" basis. In a multithreaded program, the programmer is responsible for making sure that the different threads don't interfere with each other by using these shared resources in a way that conflicts with another thread's use of the same resource. (As you might suspect, this can get a little tricky.)

Why Use Multithreading?

Multithreading provides a way to have more than one thread executing in the same process while allowing every thread access to the same memory address space. This allows very fast communication among threads. Threads are also easier to create than processes since they don't require a separate address space.

Inside Windows NT, processes and threads are represented as objects that are created, maintained, and destroyed by the Process Manager. These Process Manager process and thread objects contain simpler kernel process and thread objects.

Some typical examples of the use of multiple threads are using a background thread to print a document in a word processor and to recalculate a spreadsheet. When a new thread is created to do these tasks, the main thread can continue responding to user input. A single-threaded application can't respond to user input until it's done printing or recalculating or whatever.

On a uniprocessor platform, the use of multiple threads allows a user to continue using a program even while another thread is doing some lengthy procedure. But only one thread executes at a time.

On a multiprocessor platform, more than one processor may be running different threads in the same process. This has the potential for very significantly speeding up the execution of your program.

Sharing A Single Address Space--Synchronizing Access To Data

Running each process in its own address space had the advantage of reliability since no process can modify another process's memory. However, **all of a process's threads run in the same address space and have unrestricted access to all of the same resources, including memory.** While this makes it easy to share data among threads, it also makes it easy for threads to step on each other. As mentioned before, multithreaded programs must be specially programmed to ensure that threads don't step on each other.

A section of code in Windows operating system that modifies data structures shared by multiple threads is called a ***critical section***. It is important than when a critical section is running in one thread that no other thread be able to access that data structure. Synchronization is necessary to ensure that only one thread can execute in a critical section at a time. This synchronization is accomplished through the use of some type of Windows synchronization object. Programs use Windows synchronization objects rather than writing their own synchronization both to save coding effort and for efficiency: when you wait on a Windows synchronization object, you do NOT use any CPU time testing the object to see when it's ready.

Windows provides a variety of different types of synchronization objects that programs can use to coordinate threads' access to shared data structures. Synchronization objects remember their states and can be set and tested in one uninterruptable step. They also cause the thread to be suspended while waiting on an object and to automatically restart when the other thread signals that it's done.

During the initialization of a program, the program creates a synchronization object for each data structure or object that will be shared among threads.

EVERY critical section will have the following structure:

1. Wait on the synchronization object before accessing the data structure. The Windows waiting API insures that your thread is suspended until the synchronization object becomes unlocked. As soon as the synchronization object becomes unlocked, Windows sets the synchronization object to "locked" and restarts your thread.
2. Access the data structure. (This is the critical section.)
3. Unlock the synchronization object so that the data can be accessed by other threads.

The first step is critical because if it's omitted then any thread can access the data structure while you're accessing. The last step is also critical--it it's omitted, then no thread will be able to access the data even after you're done.

Using this technique on every critical section insures that only one thread can access the data at a time.

The Life Cycle Of A Thread

Each thread has a ***dispatcher state*** that changes throughout its lifetime.

The most important dispatcher states are:

- Running: only one thread per processor can be running at any time.
- Ready: threads that are in the Ready state may be scheduled for execution the next time the kernel dispatches a thread. Which Ready thread executes is determined by their priorities.
- Waiting: threads that are waiting for some event to occur before they become Ready are said to be waiting. Examples of events include waiting for I/O, waiting for a message, and waiting for a synchronization object to become unlocked.

SCHEDULLING

The Kernel's Dispatcher

The kernel's dispatcher performs scheduling and context switching.

Thread scheduling is the act of determining which thread runs on each processor at a given time.

Context switching is the act of saving one thread's volatile state (CPU register contents) and restoring another thread's state so it can continue running where it previously left off.

How Thread Priorities Affect Scheduling

The kernel's dispatcher schedules threads to run based a 32-level priority scheme. Windows guarantees that the threads that are ready that have the highest priority will be running at any given time. (That's one thread on a single-processor system.) Threads with a priority of 31 will be run before any others, while threads with a priority of 0 will run only if no other threads are ready. The range of priorities is divided in half with the upper 16 reserved for real-time threads and the lower 16 reserved for variable priority threads.

Real-time threads run at the same priority for their entire lifetime. They are commonly used to monitor or control systems that require action to be taken at very precise intervals. These threads run at higher priorities than all variable priority threads, which means that they must be used sparingly.

Variable priority threads are assigned a base priority when they are created. (A thread's base priority is determined by the process to which the thread belongs.) The priority of such threads can be adjusted dynamically by the kernel's dispatcher. A thread's dynamic priority can vary up to two priority levels above or below its base priority.

The dispatcher maintains a ***priority queue*** of ready tasks. When prompted to reschedule, it changes the state of the highest priority task to Standby. When the conditions are right, a context switch is performed to begin the thread's execution and the thread goes into the Ready state.

Lower priority threads will always be preempted when a higher priory thread enters the ready state. This is true even if the lower priority thread has time remaining in its quantum, or if the lower priority thread is running on a different processor.

Performance Tuning

In order to get the computer system to perform as users expect, Windows changes the priorities of threads over time.

Each process has a base priority. Threads in a process can alter their base priority by up to two levels up or down.

Depending on the type of work the thread is doing, Windows may also adjust the thread's dynamic priority upwards from its base priority. For instance:

- Threads that are waiting for input get a priority boost, as do threads in the foreground process. This makes the system responsive to the user.
- Threads get a priority boost after completing a voluntary wait.
- All threads periodically get a priority boost to prevent lower priority threads from holding locks on shared resources that are needed by higher priority threads.
- Compute-bound threads get their priorities lowered.

Scheduling On Multiprocessor Systems

A ***multiprocessing*** operating system is one that can run on computer systems that contain more than one processor. Windows is a ***symmetric multiprocessing (SMP)*** system, meaning that it assumes that all of the processors are equal and that they all have access to the same physical memory. Therefore, Windows can run any thread on any available processor regardless of what process, user or Executive, owns the thread.

There are also asymmetric multiprocessing (ASMP) systems in which processors are different from each other--they may address different physical memory spaces, or they may have other differences. These operating systems only run certain processes on certain processors--for instance, the kernel might always execute on a particular processor.

The design of Windows supports ***processor affinity***, whereby a process or thread can specify that it is to run on a particular set of processors, but this facility isn't supported in the first release.

Windows uses the same rules for scheduling on a multiprocessor system as it does on a single processor system, so at any given time the threads that are ready and have the highest priorities are actually running.

Using Task Manager

The *Task Manager* utility shows the applications and processes that are currently running on your computer, as well as CPU and memory usage information. To access Task Manager, press CtṛlAḷtDelete and click the Task Manager button. Alternatively, right-click an empty area in the Taskbar and select Task Manager from the pop-up menu.

The Task Manager dialog box has four main tabs: Applications, Processes, Performance, and Networking. These options are covered in the following subsections.

Managing Application Tasks

The Applications tab of the Task Manager dialog box, shown in **Figure iii**, lists all of the applications that are currently running on the computer. For each task, you will see the name of the task and the current status (running, not responding, or stopped).

To close an application, select it and click the End Task button at the bottom of the dialog box. To make the application window active, select it and click the Switch To button. If you want to start an application that isn't running, click the New Task button and specify the location and name of the program you wish to start.

- Processor Scheduling, which allows you to optimize the processor time for running programs
  or background services
- Memory Usage, which allows you to optimize memory for programs or system cache
- Virtual Memory, which is used to configure the paging file

Stopping Processes

Process Description

System Idle Process A process that runs when the processor is not executing any other threads

smss.exe Session Manager subsystem

csrss.exe Client-server runtime server service

mmc.exe Microsoft Management Console program (used to track resources used by MMC snap-ins such as System Monitor)
explorer.exe Windows Explorer interface
Ntvdm.exe MS-DOS and Windows 16-bit application support

Managing Process Priority

You can manage process priority through the Task Manager utility or through the start command-line utility. To change the priority of a process that is already running, use the Processes tab of Task Manager. Right-click the process you want to manage and select Set Priority from the pop-up menu. You can select from RealTime, High, AboveNormal, Normal, BelowNormal, and Low priorities.

Options for the start Command-Line Utility

Option Description

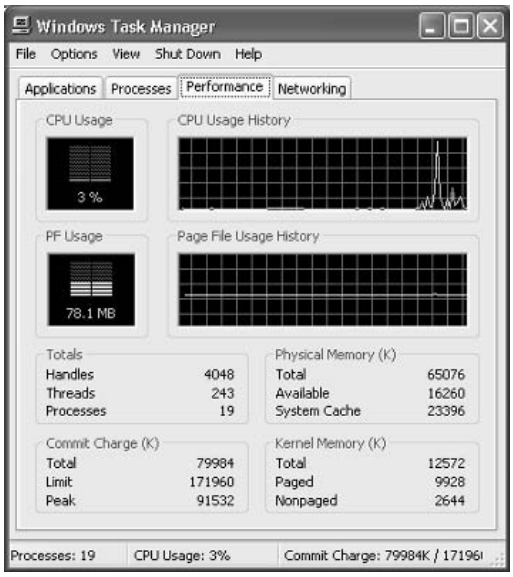
- low Starts an application in the Idle priority class.
- normal Starts an application in the Normal priority class.
- high Starts an application in the High priority class.
- realtime Starts an application in the RealTime priority class.
- abovenormal Starts an application in the AboveNormal priority class.
- belownormal Starts an application in the BelowNormal priority class.
- min Starts the application in a minimized window.
- max Starts the application in a maximized window.
- separate Starts a Windows 16-bit application in a separate memory space. By default Windows 16-bit applications run in a shared memory space, NTVDM, or NT Virtual DOS Machine.
- shared Starts a DOS or Windows 16-bit application in a shared memory space.

Managing Performance Tasks

The Performance tab shows the following information:

- CPU Usage, in real time and in a history graph
- Page File Usage, in real time and in a history graph
- Totals for handles, threads, and processes
- Physical Memory statistics
- Commit Charge memory statistics

- Kernel Memory statistics



A task manager windows

Scheduling Tasks

Windows includes a *Task Scheduler* utility that allows you to schedule tasks to occur at specified intervals. You can set any of your Windows programs to run automatically at a specific time and at a set interval, such as daily, weekly, or monthly. For example, you might schedule your Windows Backup program to run daily at 2:00 a.m.

Tuning and Upgrading the Processor

If you suspect that you have a processor bottleneck, you can try the following solutions:

- Use applications that are less processor-intensive.
- Upgrade your processor.
- If your computer supports multiple processors, add one. Windows can support up to two processors, which will help if you use multithreaded applications. You can also use processor affinity to help manage processor-intensive applications.

Monitoring and Optimizing the Processor

Processor bottlenecks can develop when the threads of a process require more processing cycles than are currently available. In this case, the process will wait in

a processor queue and system responsiveness will be slower than if process requests could be immediately served.
The most common causes of processor bottlenecks are processor-intensive applications and other subsystem components that generate excessive processor interrupts (for example, disk or network subsystems).
In a workstation environment, processors are usually not the source of bottlenecks. You should still monitor this subsystem to make sure that processor utilization is at an efficient level.

MEMORY MANAGEMENT

The memory manager implements virtual memory, provides a core set of services such as memory mapped files, copy-on-write memory, large memory support, and underlying support for the cache manager.

Memory management in Microsoft Windows operating systems has evolved into a rich and sophisticated architecture, capable of scaling from the tiny embedded platforms (where Windows executes from ROM) all the way up to the multi-terabyte NUMA configurations, taking full advantage of all capabilities of existing and future hardware designs.

With each release of Windows, memory management supports many new features and capabilities. Advances in algorithms and techniques yield a rich and sophisticated code base, which is maintained as a single code base for all platforms and SKUs.

Memory management improvements in Windows Vista focused on areas such as dynamic system address space, enhanced NUMA and large system/page support, advanced video model support, I/O and section access, and robustness and diagnosability.

Among other things, a multiprogramming operating system kernel must be responsible for managing all system memory which is currently in use by programs. This ensures that a program does not interfere with memory already used by another program. Since programs time share, each program must have independent access to memory.

Cooperative memory management, used by many early operating systems assumes that all programs make voluntary use of the kernel's memory manager, and do not exceed their allocated memory.

Memory protection enables the kernel to limit a process' access to the computer's memory. Various methods of memory protection exist, including memory segmentation and paging. All methods require some level of hardware support (such as the 80286 MMU) which doesn't exist in all computers.

Virtual memory

The use of virtual memory addressing (such as paging or segmentation) means that the kernel can choose what memory each program may use at any given time, allowing the operating system to use the same memory locations for multiple tasks.

If a program tries to access memory that isn't in its current range of accessible memory, but nonetheless has been allocated to it, the kernel will be interrupted in the same way as it would if the program were to exceed its allocated memory. When the kernel detects a page fault it will generally adjust the virtual memory range of the program which triggered it, granting it access to the memory requested. This gives the kernel discretionary power over where a particular application's memory is stored, or even whether or not it has actually been allocated yet.

In modern operating systems, application memory which is accessed less frequently can be temporarily stored on disk or other media to make that space available for use by other programs. This is called swapping, as an area of memory can be used by multiple programs, and what that memory area contains can be swapped or exchanged on demand.

✓ ERROR HANDLING

Existing operating systems include error handling tools that identify and log errors that occur during the operation of the operating systems and provide a user with the ability to view the logged errors. Although most applications running in the operating system's environment provide error messages when errors occur, these error messages are often technical in nature and may not be easily understood by the user. Therefore, the error handling tools not only identify and log these errors but, in addition, they describe the errors to the user in an easily comprehensible

format. For example, the Windows NT® operating system by Microsoft Corporation includes an event log that collects error messages from applications, device drivers and the operating system itself to a common location that the user can access. The Windows NT® operating system also includes an event viewer that permits the user to view the error messages in the order of their occurrences.

For existing error handling tools, the client application must have advance knowledge of the possible types of service failures. Otherwise, the client application would not be able to provide meaningful information regarding the nature of the error. This poses problems in a client-server environment because the client application and applications of the service providers are loosely coupled and developed independently, and it is not practical to change an application as its underlying service provider evolves. It is, therefore, difficult for a client application to obtain meaningful information for an error message within a client/server environment.

The present invention resolves the above problems by providing a mechanism for a client application that identifies the source of a service error and obtains detailed error information at the time the error occurs. In particular, the mechanism accesses the error message information specific to the service that encounters the error and, then, reports that information as part of its error handling operation. The present invention also functions in a plurality of nested subsystems in which a subsystem that detects a particular problem is identified among a group of subsystems that are called in a nested manner. Accordingly, the present invention automatically handles the operation of reporting events whose source subsystem is different from that of the reporting subsystem without requiring the reporting subsystem to have advance knowledge of the possible errors that could be encountered.

Using Event Viewer

You can use the *Event Viewer* utility to track information about your computer's hardware and software, as well as to monitor security events. Every Windows XP computer will show three types of log files (depending on your configuration you may also have other log files):

System log

Tracks events related to the Windows XP operating system. This log is useful in troubleshooting Windows XP problems.

Security log

Tracks events related to Windows XP auditing. By default auditing is not enabled.

If you enable security auditing you can select what to track and whether you will track security success and/or failure events.

Application log

Tracks events related to applications that are running on your computer. For example, you might see that your e-mail program recorded an error. These errors are useful in troubleshooting application problems or for developers to fix application problems.

INTERRUPT

Interrupts are central to operating systems as they provide an efficient way for the operating system to interact and react to its environment. The alternative is to have the operating system "watch" the various sources of input for events (polling) that require action -- not a good use of CPU resources. Interrupt-based programming is directly supported by most CPUs. Interrupts provide a computer with a way of automatically running specific code in response to events. Even very basic computers support hardware interrupts, and allow the programmer to specify code which may be run when that event takes place.

When an interrupt is received the computer's hardware automatically suspends whatever program is currently running, saves its status, and runs computer code previously associated with the interrupt. This is analogous to placing a bookmark in a book when someone is interrupted by a phone call and then taking the call. In Windows operating systems interrupts are handled by the operating system's kernel. Interrupts may come from either the computer's hardware or from the running program.

kernel is the core process of a preemptive operating system, consisting of a multitasking scheduler and the basic security services. Depending on the operating system, other services such as virtual memory drivers may be built into the kernel. The kernel is responsible for managing the scheduling of threads and processes.

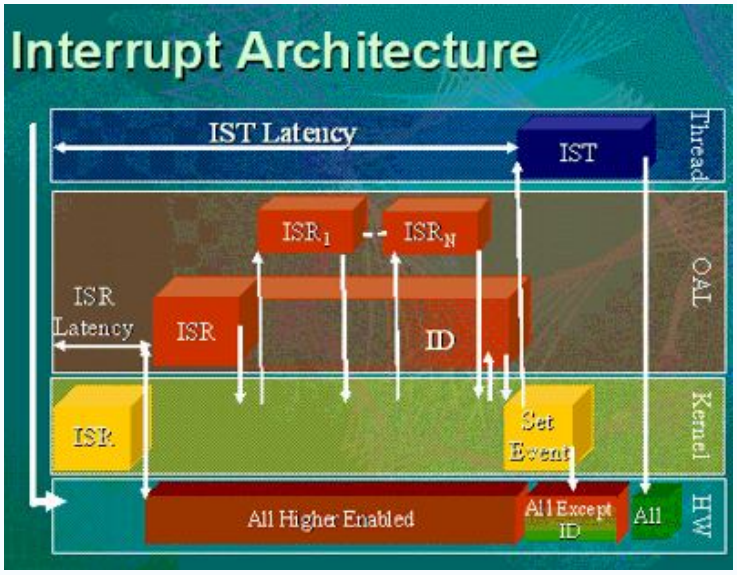
When a hardware device triggers an interrupt the operating system's kernel decides how to deal with this event, generally by running some processing code. How much code gets run depends on the priority of the interrupt (for example: a person usually responds to a smoke detector alarm before answering the phone). The processing of hardware interrupts is a task that is usually delegated to software called device drivers, which may be either part of the operating system's kernel,

part of another program, or both. Device drivers may then relay information to a running program by various means.

A program may also trigger an interrupt to the Windows operating system. If a program wishes to access hardware for example, it may interrupt the operating system's kernel, which causes control to be passed back to the kernel. The kernel will then process the request. If a program wishes additional resources (or wishes to shed resources) such as memory, it will trigger an interrupt to get the kernel's attention.

Interrupt Architecture

The first step in exploring the interrupt architecture of Microsoft® Windows® is defining an overall model of the hardware, kernel, OAL and thread interactions during an interrupt. The following diagram is an overall picture of these different levels of responsibility and the transitions that cause changes of state.



The diagram represents the major transitions during an interrupt with time increasing to the right of the diagram. The bottom most layer of the diagram is the hardware and the state of the interrupt controller. The next layer is the kernel interactions during interrupt servicing. The OAL describes the board support package (BSP) responsibilities. The top most layer represents the application or driver thread interactions needed to service an interrupt. The diagram represents the interactions during a single interrupt; representing the new ability of Windows to have shared interrupts. The activity starts with an interrupt represented by the

line at the left most section of the chart. An exception is generated causing the kernel ISR vector to be loaded onto the processor. The kernel ISR interacts with the hardware disabling all equal and lower priority interrupts on all processors except for the ARM and Strong ARM architectures. The kernel then vectors to the OAL ISR that has been registered for that particular interrupt. The OAL ISR then can either directly handle the interrupt or can use NKCallIntChain to walk a list of installed ISRs. The main ISR or any of the installed ISRs then performs any work and returns the mapped interrupt called SYSINTR for that device. If the ISR determines that its associated device is not causing the interrupt the ISR returns SYSINTR_CHAIN, which causes NKCallIntChain( ) to walk the ISR list to the next interrupt in the chain. The ISRs are called in the order that they were installed creating a priority on the calling list.

Microsoft has advanced the Windows interrupt architecture. The ability of the OS to deal with shared interrupts greatly extends the ability of Windows to support many interrupt architectures. Knowledge of this interrupt architecture greatly speeds up the investigation times into driver and latency issues. A Windows model of the operating system interaction is the key to this understanding. Shared interrupts have greatly increased the openness of Windows supporting the platform provider and application developer scenarios that are pervasive from company to company or within companies. Understanding latency sources will help in diagnosis of driver and real-time issues. The interrupt structure in Windows is well defined and understandable.

SECURITY

Security has been a hot topic with Windows for many years, and even Microsoft itself has been the victim of security breaches.[citation needed] Consumer versions of Windows were originally designed for ease-of-use on a single-user PC without a network connection, and did not have security features built in from the outset.
Windows NT and its successors are designed for security (including on a network) and multi-user PCs, but were not initially designed with Internet security in mind as much since, when it was first developed in the early 1990s, Internet use was less prevalent.
These design issues combined with flawed code (such as buffer overflows) and the popularity of Windows means that it is a frequent target of computer worm and virus writers.
Microsoft releases security patches through its Windows Update service approximately once a month (usually the second Tuesday of the month), although

critical updates are made available at shorter intervals when necessary. In Windows 2000 (SP3 and later), Windows XP and Windows Server 2003, updates can be automatically downloaded and installed if the user selects to do so.

Use these steps to develop a strong password:

- Think of a sentence that you can remember. This will be the basis of your strong password or pass phrase. Use a memorable sentence, such as "My son Aiden is three years old."
- Check if the computer or online system supports the pass phrase directly. If you can use a pass phrase (with spaces between characters) on your computer or online system, do so.
- If the computer or online system does not support pass phrases, convert it to a password. Take the first letter of each word of the sentence that you've created to create a new, nonsensical word. Using the example above, you'd get: "msaityo".

- Add complexity by mixing uppercase and lowercase letters and numbers. It is valuable to use some letter swapping or misspellings as well. For instance, in the pass phrase above, consider misspelling Aiden's name, or substituting the word "three" for the number 3. There are many possible substitutions, and the longer the sentence, the more complex your password can be. Your pass phrase might become "My SoN Ayd3N is 3 yeeRs old." If the computer or online system will not support a pass phrase, use the same technique on the shorter password. This might yield a password like "MsAy3yo".
- Finally, substitute some special characters. You can use symbols that look like letters, combine words (remove spaces) and other ways to make the password more complex. Using these tricks, we create a pass phrase of "MySoN 8N i$ 3 yeeR$ old" or a password (using the first letter of each word) "M$8ni3y0".
- Test your new password with Password Checker Password Checker is a non-recording feature on this Web site that helps determine your password's strength as you type.

- ✓ **4 steps to protect your computer**

Step 1. Keep your firewall turned on

What is a firewall?

A firewall helps protect your computer from hackers who might try to delete information, crash your computer, or even steal your passwords or credit card numbers. Make sure your firewall is always turned on.

Step 2. Keep your operating system up-to-date

What are operating system updates?

High priority updates are critical to the security and reliability of your computer. They offer the latest protection against malicious online activities. Microsoft provides new updates, as necessary, on the second Tuesday of the month.

Step 3. Use updated antivirus software

What is antivirus software?

Viruses and spyware are two kinds of usually malicious software that you need to protect your computer against. You need antivirus technology to help prevent viruses, and you need to keep it regularly updated.

Step 4. Use updated antispyware technology

What is antispyware software?