**Introduction**

* **Memory Hierarchy** from bottom to top: Magnetic tapes -> Optical Disk -> Magnetic Disk -> Electronic disk -> Main memory -> Cache -> Registers
* Process is a program in execution. Program is passive while process is active.
* User mode is bit=1 and kernel mode is bit=0
* User program executes -> call system -> execute system call (in kernel mode) -> Return from system call.
* Preemptive multithreading: Everything looks like they are running at the same time.
* Each page in virtual memory is mapped to a certain location in physical memory.
* Virtual memory is split into 2GB of user and 2GB of kernel memory.
* All pages in kernel virtual memory are mapped to the same location in physical memory.
* User virtual memory is mapped to its own memory in physical memory.
* Processor does the mapping with a paging table.
* Process is a task which is already running.
* Handle is an index of a pointer to kernel objects.
* Processes do not have to know where its virtual address is mapped to.
* Reasons for chunking memory:
  1. Make access to memory more efficient
  2. Set security attributes for specific chunk
  3. Allow processes to access the same register.
* Threads share the same memory space.

**Processes, threads and handles**

* Processes mangage resources such as threads and memory, while threads do the actual work.
* Processes have their own memory, while threads do not.
* Processes cannot access other processes’ memory, unless specified.
* Threads from same process can access one another’s memory, but cannot access memory of another thread in another process.
* Parent and child processes run cocurrenttly, but parent process can try to finish after child process, so that it can clean up after the child process.
* Multi-tasking: Quickly change from 1 thread to another.
* Multi-processing: Makes use of all processors in a computer for better efficiency.
* Loaders load a program and turn it into a process.
* Each process has a handle table. Index of a handle in one process’s handle table is different from that of another process.
* Duplicate handle can be used between 2 processes.
* Handle inheritance can only be used between parent and child.
* Difference between system call and handle: System call is requesting resource from kernel and usually requires the handle to the resource.
* Context Switches: How many times a thread is run.
* Hardware interrupt: Keyboard press, mouse click.
* A thread is represented by KTHREAD.
* Context Switching: Save state/registers of one thread and load state/registers of another thread.
* Each process have at least one thread, and each thread has instruction pointers.
* Details of a process will be in a process object and stored in the OS.
* Scheduling: Try to minimise the overhead of context switching
* Problem of round-robin simple scheduling: Threads which are intensive do not get enough time to run.
* Solution: Use priority, from 0 to 31. Highest priority runs first and has more time to run.
* Base priority is for process while dynamic prioirty is for each thread in the process.
* Starvation: Computer gives a thread which hasn’t run in a long time the hgihest priority for a while, before giving it back its original priority.

**Synchronisation Objects**

* Solution for race condition is to use synchronisation objects.
* Lock (Mutex): lock.acquire and lock.release and Win32API
* Critical section (terminology): Part of code to prevent more than one thread to enter.
* Difference between Mutex and CriticalSection (Win32API): Mutex uses object in the kernel, so when it is called, transition to kernel mode occurs. CriticalSection API does not need to move to the kernel, so it is more efficient, but can only be used with 2 or more threads in the same process.
* Semaphore has an initial counter, and decrements by 1 when a thread starts. When the thread ends, the counter increases. Anyone can acquire or release the semaphore, unlike the mutex which can only be released by the owner.
* Events are global boolean objects. Signalled: True or False.
* Interlocked operations are the simplest form of synchronisation.
* Hardware implements interlocked and is very limited in operations, but it is the fastest.
* Deadlock: Two or more processes wait for each other.
* Reference count: Stores the number of references, pointers, or handles to a resource, such as an object, a block of memory, disk space, and others. When the count reaches 0, the resource is deallocated.

**Communication between processes**

* Named pipes: Use two pipes; one for reading and the other for writing.
* File Mapping/Shared Memory: API called CreateFileMapping which creates shared memory to be used between two processes. For shared memory, new chunk will be allocated in virtual memory of both processes and in physcial memory. Virtual memory of both processes is different, but the physical memory address is the same.
* Shared memory is subject to race conditions, but named pipes are not.
* To inherit handle, pass security attributes to variable.
  + sa = win32.SECURITY\_ATTRIBUTES()
  + sa.inheritHandle = True

**Memory Management**

* CPU Addressing Modes: Real, Protected, Long
* The larger the registers, the more memory they can have.
* To calculate how much memory ton access, calculate the size of registers.
* Assuming 32-bit registers, num = 2^ 32 in bytes. In GB, num/1024/1024 = 4 GB.
* MMU is the only one which talks to memory and cache, and translates virtual to physical addresses.
* Assume a function is 100 bytes. If a process runs a function, these 100 bytes will be copied to its address space. If both processes use the same function, the physical memory chunk is the same, though from the processes’ point of view, the virtual memory chunk is unique to themselves.
* Difference between shared memory and common memory used for a function to be used between 2 processes: Shared memory allows 2 processes to know that the memory is shared while common memory will not have the processes know that the memory is being shared.
* OS creates page table and takes charge of CR3, which is a special register that stores current page table.
* Paging only occurs when you have swap space to put memory pages that you want to swap out to that space in order to use the memory page for something else. If there is no swap space, then there is no paging, however there will be context switching.
* Page fault occurs when the physical address is not found on the RAM, paged out page or non-existent memory address was accessed, or a write access was triggered for read-only page.
* Workset: How much memory in the RAM
* Private workset: Vritual memory being used by one process at the moment.
* Heap: Block of virtual memory which is readable and writable to all threads. It allocates small amounts of data quickly and acts like a reserve memory space.
* Each process has its own page table, and each page table has 3 columns: Virtual Memory Address, Physical Memory Address and Permissions (set for virtual memory address; read, write).
* Copy-on-write: When a process accesses the shared memory to change the variable to something else in its own virtual memory, OS will copy the shared memory chunk to a new chunk/page. Both processes will now have seperate pages, with no shared page.
* Processes contain stack, heap, code, system information and data.
* Stack: Local variables in a function are saved in its own private stack. It is the way back. Each thread has one stack.

**Management Mechanisms**

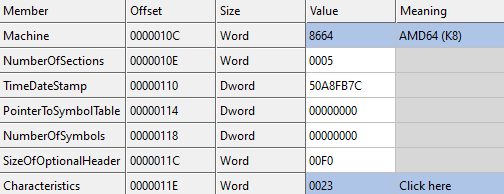
* Registry contains config of OS components and installed programs.
* Since running every service in its own process is wasteful, they are sometimes run in a container called svchost.exe.
* Services are controlled by the SCM.
* WOW64 allows execution of 32-bit apps on 64-bit Windows.
* EBP and ESP allow a thread to know where the stack is.
* EIP allows a thread to know where the current instruction is.
* Registers are in the processor/CPU while the stack is in virtual memory.

**Subwindow Systems**

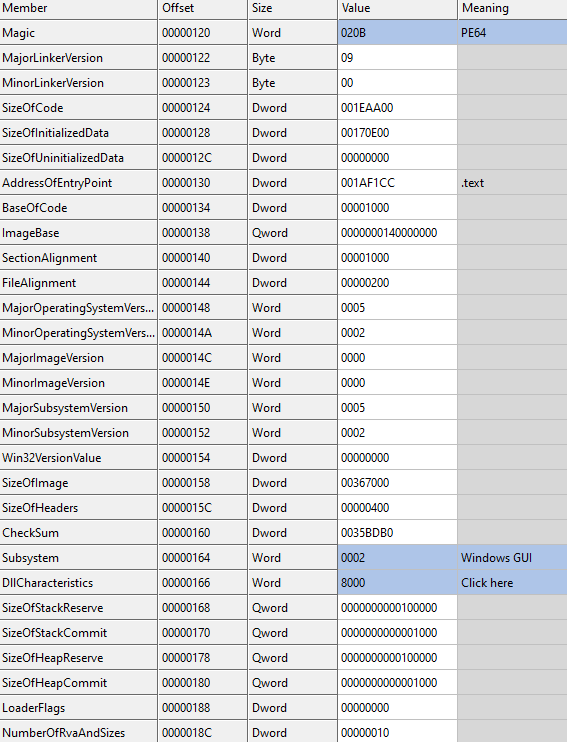
* Register new windows classes with RegisterClassEx
* Create new windows with CreateWindowEx
* Only windowed threads have thread cues.
* WM\_CHAR has the character code and exists because multiple languages exist and hence the character code is different for each language.
* WM\_KEY\_UP & WM\_KEYDOWN make use of virtual keys and will always be the same.
* Window handles are global.
* Use Spy++ to obtain handle hexadecimal.
* Reason why Win32k was moved to kernel: Kernel is faster and processes required more privileges.

**PE and DLL information**

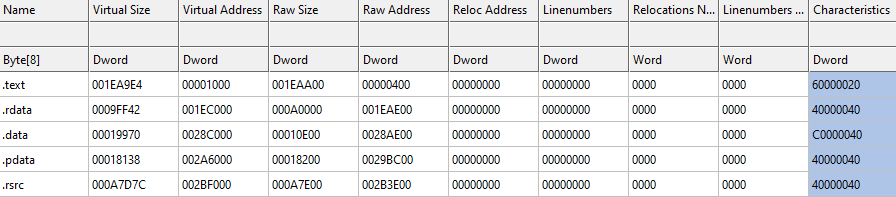
* All addresses are relative and expressed in relative virtual addressing (RVA).
* DOS header exists for legacy reasons.
* NT headers contain File header and Optional header.
* File header:



* Optional header is not actually optional and tells about the size of the environment and where to start executing the exe.
* DWORD 0x3C (File Alignment) points to PE header.



* Section header:



* .rdata contains costants, and .data contains global variables.
* Import directory contains the list of functions/DLLs used.
* Export directory is a list of functions/DLLs which other applications can use.
* Only DLLs have export directory, not exe files.
* Relocation directory changes the addresses of the instructions to the correct absolute addresses, since the intial addresses are relative.

**Loading and Linking**

* Loader loads main exe to memory and jumps to exe’s entry point.
* DLLs can be loaded on runtime.
* **Static linking**: Static linking is the result of the linker copying all library routines used in the program into the executable image.
* **Static linking Pros and Cons**: Requires more disk space and memory than dynamic linking, but is both faster and more portable, since it does not require the presence of the library on the system where it is run.
* **Dynamic linking pros**:
  + Share same DLL on disk and memory.
  + App can be upgraded without recompiling
* Use LoadLibrary API to load DLLs.
* **Dynamic linking cons:**
  + Shipping app without its DLL can risk in incomptability
* 2 types of dynamic linking: implicit and explicit
* Implicit dynamic linking: Process has not been created, and all DLLs are loaded.
* Explicit dynamic linking: Process has already begun, and new DLLs are loaded. Can be used for updates.

**Security**

* Built-in SIDs (security identifiers):
  + Everyone: S-1-1-0
  + Null Authority: S-1-0
  + Local Authority: S-1-2
  + Local: S-1-2
  + Creator Authority: S-1-3
  + Creator Owner: S-1-3-0
  + Creator Group: S-1-3-1
  + Service: S-1-5-6
  + Anonymous: S-1-5-7
  + Authenticated Users: S-1-5-11
  + Local System: S-1-5-18
  + NT Authority: S-1-5-19 (Local Service) and S-1-5-20 (Network Service)
  + Administrator: S-1-5-21domain-500
  + Guest: S-1-5-21domain-501
* The registry key **HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList** stores all SIDs.
* Security components:
  + Security Reference Monitor (SRM) -> Kernel mode
  + Local Security Authority Subsystem (LSASS) -> User mode
  + Authentication Packages
  + Winlogon
* Information of user permissions is stored in HKLM/SAM
* Use reg query to look inside a registry key value, which cannot be seen with regedit.
* Mandatory Access Control (MAC): Process determines access rights.
* Low integrity processes cannot open handles to higher integrity ones.
* Process cannot write to higher level objects cannot read higher level processes.

**When opening a process**

1. Use CreateFile function to get a handle from the file on hard disk.
2. OpenProcess using the handle.
3. A thread is created and a handle table + virtual address space are created in the process.
4. Instruction pointer (EIP) in the thread is saved in kernel memory

/OS for future use. This happens during context switching when a thread is switched out.