**2.Aufgabe**

2.1)

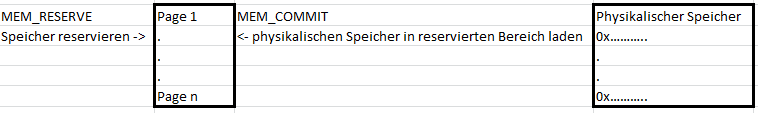
Der physikalische Adressraum ist der Adressraum des realen Arbeitsspeichers. Der virtuelle Adressraum beschreibt einen gedachten, nicht real vorhandenen Arbeitsspeicher, der als virtueller Speicher bezeichnet wird. Der virtuelle Adressraum ist im Normalfall größer als der physikalische Adressraum. Der virtuelle Speicher wird auf der Platte abgebildet. Dabei werden nicht alle Teile des Programms und analog nicht der gesamte Datenbereich benötigt. Es reicht also, nur die benötigten Programm- und Datenbereichs-Teile im Arbeitsspeicher zu halten. Bei Bedarf können die zerlegten Programme und Daten in den Speicher geladen werden, wenn sie von der CPU benötigt werden. Zur Abarbeitung ist es dann erforderlich, die virtuellen Adressen in physikalische Adressen zu transformieren.

2.2)

Die Funktion VirtualAlloc reserviert Speicher auf Page-Level, also der kleinsten Speichereinheit die von der CPU allokiert oder freigegeben wird. Die Größe einer Page ist abhängig von der CPU, sie kann entweder 1024 oder 4096 Bytes groß sein. Die Speicherallokierung erfolgt in 2 Schritten. Zuerst wird ein virtueller Speicher reserviert. Die Reservierung benötigt keinen Arbeitsspeicher. Es wird nur verhindert, dass ein Teil des virtuellen Adressraums für andere Zwecke benutzt wird. Nach dieser Reservierung können Teile des Bereichs oder der gesamte Bereich "committed" werden, wodurch physikalischer Speicher in den reservierten Bereich "gemappt" wird. Also wird die VirtualAlloc-Funktion sowohl für die Reservierung als auch dem "committen" des Speichers verwendet.

MEM\_RESERVE reserviert den virtuellen Adressspeicher, der später "committed" wird. MEM\_COMMIT allokiert den Speicher, der vom Programm schließlich benutzt wird. Auf reservierte Pages kann nicht zugegriffen werden, bis ein anderer Aufruf von VirtualAlloc mit dem Parameter MEM\_COMMIT aufgerufen wird.

Skizze Aufruf Virtual Alloc:



2.3)

In Windows CE 5.0 bezieht sich der Kernel Mode auf den Zugriffsbereich für einen Thread. Ist ein Thread im Kernel Mode, kann auf den Kernel-Adressraum zugegriffen werden. Dabei kann mithilfe eines Aufrufes von SetKMode jederzeit in den Kernel Mode wechseln bzw. diesen verlassen. Die meisten APIs sind dabei nicht im Kernel implementiert (nk.exe), also geht ein Thread normalerweise nicht in den Kernel Mode wenn eine API aufgerufen wird, es sei denn, es wird eine API aufgerufen, die in nk.exe implementiert wurde. Dann wird temporär in den Kernel Mode gewechselt.

In Windows CE 6.0 ist die Implementierung gleich wie bei 5.0, jedoch wird die SetKMode API nicht mehr unterstützt. Threads können nicht mehr zu einem beliebigen Zeitpunkt in den Kernel Mode wechseln oder den Kernel Mode verlassen. Es ist also nicht einfach so möglich, auf physikalischen Speicher zuzugreifen. Jedoch geht ein Thread nun standardmäßig in den Kernel Mode, da die meisten Module der System-APIs in Windows CE 6.0 in den Kernel-Prozess geladen werden.

Der Unterschied zwischen den beiden Versionen liegt also darin, dass in CE 6.0 ein Thread nur in den Kernel Mode wechselt, wenn Code innerhalb des Kernel-Prozesses ausgeführt wird. Wenn eine User-Applikation einen Funktionspointer an Kernel Mode Code übergibt, und dieser den Funktionspointer direkt aufruft, ist der Thread auch dann noch im Kernel Mode, während eigentlich User Code ausgeführt wird. Dies ist sehr unsicher, da der Code des User Modes alle Kerneladressen und Kernel-APIs erreichen kann. Daher soll beim Schreiben von Treibern immer die CEDDK-Funktion (CeDriverPerformCallback) aufgerufen werden. Dadurch geht der Thread wieder ordnungsgemäß in den User-Prozess zurück, damit die hier aufgerufenen Funktionen nicht auf irgendwelche Kernel-spezifischen Adressen bzw. APIs erreichen kann.