Grundlagen der Windowsprogrammierung

1. Eigenschaften der Windows - Umgebung:

☞ Beschreiben Sie die Besonderheiten von Windowsprogrammen!

- Windows arbeitet mit **grafischen Oberflächen**. Grafische Oberflächen stehen für Benutzerkomfort und ansprechende, bildliche Darstellungen. Für den Programmierer bedeutet dies erhöhte Anforderungen an seine Programme, damit diese dem vorgegebenen Standard entsprechen.
- Windows - Anwendungen werden aus der Sicht des Benutzers in **Fenstern** ausgeführt. In den Fenstern werden dem Benutzer Informationen angezeigt, und umgekehrt kann er Eingaben an die Anwendung schicken. Für den Benutzer verschmilzt daher das Hauptfenster einer Anwendung mit der Anwendung selbst; für den Programmierer ist es »lediglich« eine visuelle Schnittstelle.
- Windows arbeitet **ereignisorientiert**. Auf der Konsole kann jeweils nur ein Programm vom Start bis zum Ende abgearbeitet werden. Windows - Anwendungen laufen dagegen nur schrittweise ab. Alle Eingaben von Seiten des Benutzers werden zuerst von Windows abgefangen. Das Ereignis wird verarbeitet und an die entsprechende Anwendung geschickt. Für eine Anwendung bedeutet dies, dass sie zur Bearbeitung eines Ereignisses aufgerufen wird, entsprechend der Eingabe reagiert und dann wieder zurücktritt, bis ihr vom Windows - Manager ein weiteres Ereignis zur Bearbeitung zugesandt wird.

2. WinNT bis Win11, 32 bis 64- Bit Betriebssysteme

2.1 Programme und Prozesse unter Windows

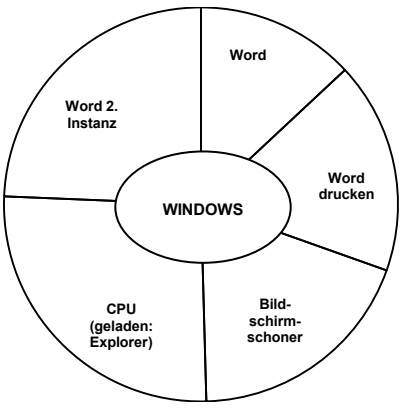
☞ Erläutern Sie den Unterschied zwischen Programm und Prozess!

WinNT bis Win11 sind multitaskfähige 32/ 64-Bit-Betriebssysteme, die dem Anwender die quasi - parallele Ausführung mehrerer Programme ermöglichen.
Windows verwaltet die laufenden Threads der Programme dazu in einer Art Ringpuffer. Die im Ringpuffer eingetragenen Programme bekommen nacheinander von Windows die Kontrolle über die CPU zugeteilt und können dann ihren Code ausführen. Da sich die Threads der Programme dabei in schneller Folge abwechseln, entsteht der Eindruck, dass die Programme parallel ablaufen.
WinNT bis Win11 sind aber nicht nur multitaskfähig, sondern auch multithreadfähig. Multithreading bedeutet, dass ein einzelnes Programm über mehrere parallel ablaufende Code - Abschnitte (die sogenannten Threads) verfügen kann. Statt von Programmen spricht man nunmehr von Prozessen. Ruft der Anwender ein Programm auf, richtet Windows einen Prozess mit 4 GByte/ 16 TByte großem Adressraum ein.
In diesen Prozessadressraum wird der auszuführende Code des Programms geladen und die Startadresse dieses Codes in Form eines Instanz - Handles an die Eintrittsfunktion (WinMain( )) des Programms zurückgeliefert. Danach richtet Windows den Hauptthread des Programms ein, der mit der Ausführung der Eintrittsfunktion beginnt. Je nach Implementierung des Programms kann dieser Hauptthread während seiner Ausführung weitere Threads erzeugen und ausführen lassen, die dann alle einem gemeinsamen Prozessadressraum angehören.
In neueren 64-Bit Betriebssystemen beträgt die theoretische Prozessgröße 16 TByte.

2.2 Die Prozessverwaltung unter Windows

☞ Welche Aufgaben erfüllt Windows beim Ausführen von Prozessen?

Ein Prozess ist nur eine Hülle, eine virtuelle Umgebung, die dem Programm vorgaukelt, es wäre das einzige laufende Programm auf dem Computer. Den ausgeführten Code bezeichnet man als Thread. Im Ringpuffer von Windows werden nicht Programme, sondern Threads verwaltet.
Prozess, Adressraum, laden des Codes und Hauptthread werden automatisch von Windows eingerichtet.
Jedes in Ausführung befindliche Programm ist ein eigener Prozess und verfügt über einen eigenen Adressraum. Auch wenn ein Programm, beispielsweise Word, mehrfach aufgerufen wird, stellt jede aufgerufene Instanz des Programms einen eigenen Prozess mit eigenem Adressraum dar.



Der Adressraum jedes Prozesses umfaßt 4 GByte / 16 TByte. Dies bedeutet, dass der Prozess diesen Speicher verwenden kann, der aber natürlich nicht mit reellem Speicher verbunden sein kann. Für die Abbildung der virtuellen Adressen auf reelle Speicherzellen sorgt Windows.
Im Allgemeinen sorgt die Speicherverwaltung von Windows dafür, dass die laufenden Prozesse sich nicht gegenseitig ihre privaten Daten überschreiben. Dies erschwert aber auch den Austausch von Adressen zwischen Prozessen, da gleiche logische Adressen unterschiedlicher Prozesse von Windows üblicherweise auch auf unterschiedlichen physischen Speicherzellen abgebildet werden.
Erzeugt ein Programm mehrere Threads, liegen diese alle in einem gemeinsamen Prozessraum und können daher problemlos Adressen untereinander austauschen.

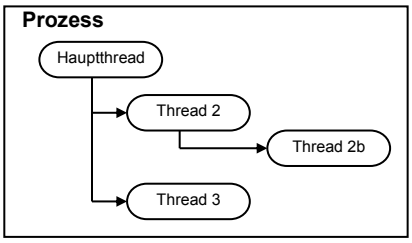
Grundlagen der Windowsprogrammierung

3. Multithreading und Multitasking unter Windows

3.1 Multithreading

☞ Erläutern Sie die Vorteile des Multithreadings!

Unter Win16 bezeichnete man in Ausführung befindlichen Code als Task. Da man unter Windows 3.x ein Programm mehrfach aufrufen kann, sind die Bezeichnungen Programm und Task nicht identisch. Stattdessen spricht man von laufenden Instanzen eines Programms.



Ab Win32 spricht man dagegen von Prozessen und Threads. Jede Instanz eines Programms entspricht nun einem Prozess, und jeder Prozess verfügt automatisch über einen Thread, der den eigentlichen auszuführenden Handlungsfaden bezeichnet.
Bis dahin gibt es noch keinen Unterschied zwischen Threads und Tasks, aber Threads haben den Vorzug, dass sie selbst neue Threads erzeugen können (wobei erzeugter und erzeugender Thread dem gleichen Prozess angehören). Da alle erzeugten Threads am Multitasking teilnehmen, hat eine Anwendung damit die Möglichkeit, zeitaufwendige Routinen (beispielsweise das Ausdrucken eines Textes) als Thread abzuspalten, so dass die Anwendung (ihr Hauptthread), während des Druckens weiter ausgeführt werden kann.
Dieses vereinfachte Modell würde die Performance des Systems stark herabsetzen. Meist haben Windows - Anwender mehrere Anwendungen geöffnet, arbeiten aber nur mit einem Programm. Nach obigem Modell würden dann alle geöffneten Anwendungen gleich lang über die CPU verfügen und die einzig arbeitende Anwendung würde quälend langsam ablaufen. Windows passt das Modell daher in mehreren Punkten an:
Threads erhalten Prioritäten zugewiesen, die das Betriebssystem dynamisch anpassen kann. Untätige Threads werden in den Hintergrund gedrängt.

3.2 Multitasking

☞ Erklären Sie den Unterschied zwischen kooperativem und preemptivem Multitasking!

Alte Betriebssysteme (z.T. Win 9x) verfügten bereits über **kooperatives Multitasking**. Beim kooperativen Multitasking übernimmt der Windows - Manager die Verwaltung gleichzeitig geöffneter Anwendungen. Windows überwacht die Eingaben des Benutzers und leitet diese der Reihe nach als Botschaften an die entsprechenden Tasks weiter. Die Anwendungen erhalten zur Bearbeitung die Kontrolle über die CPU und geben diese erst wieder ab, wenn ihre Aufgabe erledigt ist. Windows hat keine Möglichkeit, ihnen die Kontrolle über die CPU abzunehmen. Schlimmstenfalls stürzt die Anwendung ab und legt dadurch das gesamte System lahm.
Win32, d.h. ab Win 9x, NT, 2K und XP, Vista und Win10/11, verwenden dagegen das **preemptive Multitasking**, bei dem das System dem laufenden Thread praktisch jederzeit (nach Ausführung jedes CPU - Befehls) die Kontrolle über die CPU entziehen kann.
Bei der Verwaltung der Threads unter Windows teilt der Windows - Manager jedem Thread eine Zeitscheibe zu, die bestimmt, wie lange der Thread über die CPU verfügen kann. Außerdem sorgt der Prozess - Manager dafür, dass alle laufenden Threads der Reihe nach die Kontrolle über die CPU erhalten. So entsteht der Eindruck der parallelen Verarbeitung mehrerer Programme.

4. Botschaftsverarbeitung unter Windows

Windows erfordert eine botschaftsorientierte Programmierung. Botschaften dienen der Kommunikation zwischen Windows und den laufenden Anwendungen und stellen ein wichtiges Element der Multitasking - Umgebung dar. Botschaften werden aber auch zur Kommunikation von Anwendungen untereinander verwendet.
Windows kennt zwei grundlegende Wege zur Übermittlung von Botschaften:

4.1 Botschaftsübertragung über die Message Loop

Partnerarbeit:

☞ Beschreiben Sie die Abarbeitung von Ereignissen über die Message Loop!

Botschaften, die vom Anwender ausgelöst werden, laufen stets über die Message Loop. Die Message Loop ist eine besondere while - Schleife, in der eine Anwendung Botschaften von Windows empfängt und verarbeitet:

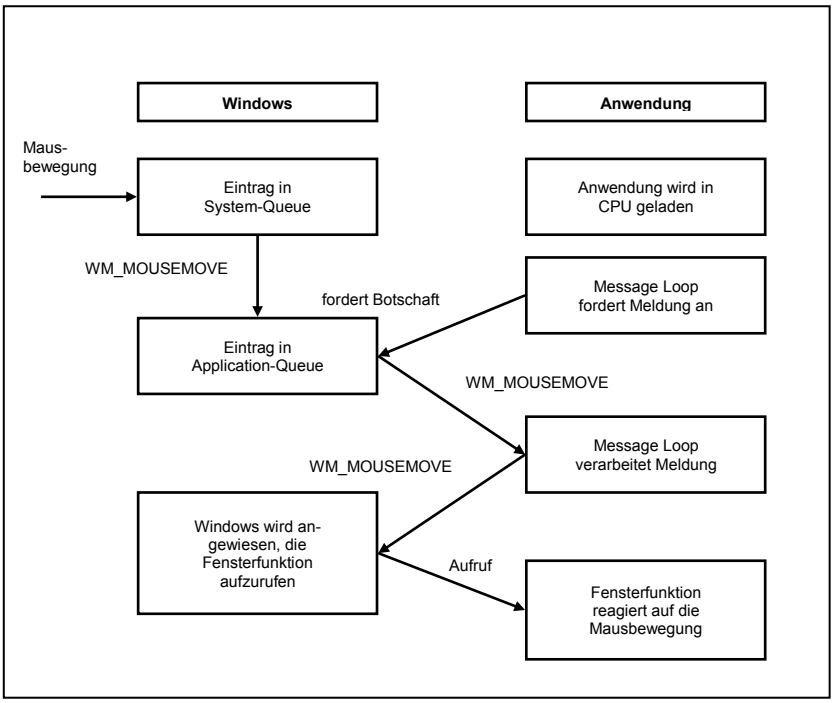
```
// MessageLoop
while (GetMessage (&Message, NULL, 0, 0))
{
   TranslateMessage(&Message);
   DispatchMessage(&Message);
}
```

Sie ist allerdings nicht die Endstation der Botschaftsverarbeitung, sondern lediglich eine Zwischenstation. Ihr Vorteil ist die chronologische Bearbeitung der einkommenden Botschaften (**fifo**, first in first out). Dies ist insbesondere für die Bearbeitung von Benutzereingaben wichtig (wenn der Anwender beim Aufsetzen eines Textes in einen bestimmten Absatz klickt und dort ein neues Wort einfügt, möchte er auch, dass zuerst der Mausklick und dann die Tastatureingaben bearbeitet werden und nicht umgekehrt).

Grundlagen der Windowsprogrammierung

Beispiel: Mausbewegung

- Ereignis tritt auf, beispielsweise die Bewegung der Maus.
- Ereignis wird in die System Message Queue eingetragen. Die System Message Queue - Warteschleife fängt erst einmal alle Eingaben ab, die von Peripheriegeräten kommen (Maus, Tastatur, Drucker etc.).
- Die System Message Queue wird abgearbeitet. Die abgefangenen Ereignisse werden ausgelesen und auf die Message Queues der zugehörigen Threads verteilt (klickt der Anwender beispielsweise in ein Fenster, wird der Thread ermittelt, der dieses Fenster erzeugt hat, und die Botschaft wird in seine Message Queue eingetragen).
- Die Botschaft wird in der Message Loop empfangen, übersetzt (in für den Programmierer leichter zu lesende Parameter) und an die bearbeitende Fensterfunktion weitergereicht.



4.2 Die Fensterfunktion WndProc(...)

Die Fensterfunktion des Fensters erhält die Botschaft. Letztendlich werden Botschaften an Fenster geschickt. Daher definiert auch jedes Fenster eine eigene Fensterfunktion (Windows Procedure), die die einkommenden Botschaften empfängt und der Verarbeitung zuführt. Zu diesem Zweck definiert jede Fensterfunktion eine switch - Anweisung. In dieser switch - Anweisung wird festgestellt, welche Botschaft empfangen wurde, und eine entsprechende Funktion zur Beantwortung der Botschaft aufgerufen.

```
// Fensterfunktion WndProc
LRESULT CALLBACK WndProc(HWND hWnd,
          UINT uiMessage, WPARAM wParam, LPARAM lParam)
{
   // beantworte Botschaften mit entsprechenden Aktionen
   switch(uiMessage)
   {
     case WM_DESTROY:
       PostQuitMessage(0);
       return(0);
     default:
       return(DefWindowProc(hWnd, uiMessage, wParam, lParam));
   }
}
```

4.3 Botschaftsübertragung ohne Message Loop

☞ Wie verläuft die Botschaftsverarbeitung ohne Message Loop?

Eine Vielzahl von Botschaften haben nur indirekt etwas mit Benutzeraktionen zu tun und werden intern von Windows verschickt (beispielsweise, wenn Windows ein Fenster darüber informiert, dass es gerade verschoben, neu dimensioniert oder geschlossen wird). In diesen Fällen wird nicht nur die System Message Queue ausgespart. Windows umgeht dabei auch die Message Queues der einzelnen Threads und schickt die Botschaft stattdessen direkt an die Fensterfunktion des betroffenen Fensters.

Beispiel: Schließen eines Fensters

- Ereignis tritt auf (der Anwender hat in der Titelleiste eines Fensters die Schaltfläche Schließen angeklickt).
- Windows schickt eine entsprechende Botschaft (WM_CLOSE) direkt an die Fensterfunktion(en).
- Die Fensterfunktion des betroffenen Fensters empfängt die Botschaft und führt sie der Verarbeitung zu.

4.4 Die Bedeutung von Windowsbotschaften

Windows kennt derzeit (ohne .NET Framework) an die 500 verschiedene Botschaften. Glücklicherweise muss der Programmierer nicht selbst für die korrekte Bearbeitung all dieser Botschaften sorgen; es genügt, wenn er alle nicht von ihm abgefangenen Botschaften ihrer vorgesehenen Standardverarbeitung zuführt (durch Aufruf der API - Funktion DefWindowProc( ).
Worauf es letztendlich ankommt, ist die für ein Programm wirklich interessanten Botschaften abzufangen und mit dem Aufruf passender Antwortfunktionen zu verbinden.