Prozessverwaltung - Threads

1. Grundlagen und Unterscheidung Prozess ⇔ Thread

   a) Erklären Sie, was ein Thread ist

   Lösungsvorschlag:

   Ein Thread ist Ausführungsfaden / Verarbeitungsstrang innerhalb eines Prozesses. Da ein Thread in seinem Aufbau einem Prozess ähnelt, jedoch weniger Ressourcen verwaltet, wird er häufig auch als leichtgewichtiger Prozess bezeichnet.

   b) Nennen Sie drei Vorteile von Threads

   Lösungsvorschlag:

   Es muss kein thread-eigener Prozessraum bereitgestellt werden

   Beim Kontextwechsel keine MMU-Umprogrammierung notwendig → schnellerer Kontextwechsel

   Daten können relativ einfach über Thread-Grenzen hinweg geteilt werden

   c) Weshalb muss ein Betriebssystem neben dem Thread-Modell auch das Prozess-Modell unterstützen?

   Lösungsvorschlag:

   Da ein Thread ohne einen umgebenden Prozess nicht existieren kann → Prozess stellt Ressourcen bereit und verwaltet diese

   Das Prozess-Modell sorgt dafür, dass die Threads zweier unterschiedlicher Prozesse keine gegenseitige Dateneinsicht / Datenmanipulation vornehmen können

Prozessverwaltung - Threads

1. Grundlagen und Unterscheidung Prozess ⇔ Thread

   d) Nennen Sie die wesentlichen Unterschiede zwischen dem Thread-Modell und dem Prozess-Modell
   Lösungsvorschlag:
   Beim Prozess-Modell werden Prozess-Daten voneinander isoliert betrachtet, wohingegen Threads sich Daten innerhalb desselben Prozesses teilen.
   Bezgl. der Parallelität: Prozess-Parallelität bedeutet, dass mehrere Benutzerprogramme parallel ablaufen, Thread-Parallelität meint die parallele Ausführung von Aufgaben innerhalb eines Benutzerprogrammes.
   Der Kontextwechsel (CPU-Scheduling) eines Prozesses ist wesentlich aufwändiger und dauert länger als der eines Threads (keine Notwendigkeit der MMU-Umprogrammierung).

   e) Aus welchen Bestandteilen besteht:

      a) Ein Prozess? **Lösungsvorschlag:** Code, Daten, Dateiverweise, Threads

      b) Ein Thread? **Lösungsvorschlag:** Stack, PC, PSW, Register

   f) Was sind Prozess-Ressourcen, die Threads innerhalb eines Prozesses gemeinsam teilen?
   Lösungsvorschlag: Code, Daten, Dateiverweise

   g) Inwiefern bringt die gemeinsame Nutzung von Ressourcen innerhalb eines Prozesses Vorteile, und inwiefern Nachteile?
   Lösungsvorschlag:
   Vorteile sind: schneller Kontextwechsel, effiziente Speichernutzung
   Nachteile sind: Ressourcenzugriff muss synchronisiert erfolgen (vgl. VL später), je nach Anwendungsfall erhöhter Programmieraufwand bei der Verwendung von Threads (zeitliches Verhalten, Debugging, …)

Prozessverwaltung - Threads

2. Implementierung des Multithreading

   a. Unterscheiden Sie die Begriffe User-Level-Thread und Kernel-Level-Thread

   Lösungsvorschlag:

   User-Level-Threads werden entsprechend ihrem Namen auf Benutzerprogramm-Ebene verwaltet, also z. B. durch eine Programmier-Laufzeitumgebung verwaltet. Kernel-Level-Threads hingegen werden auf Betriebssystem-Ebene verwaltet. Je nachdem, welches Betriebssystem eingesetzt wird, kann das Betriebssystem unterschiedlich viele KL-Threads bereitstellen, auf die UL-Threads abgebildet werden. Den KL-Threads werden CPU-Kerne und damit Rechenzeit zugeteilt.

   b. Nennen Sie drei grundsätzliche Möglichkeiten zur Implementierung des Multithreading und erklären Sie, wie dabei UL-Threads auf KL-Threads abgebildet werden

   Lösungsvorschlag:

m:1-Abbildung	m:n-Abbildung	1:1-Abbildung
Alle UL-Threads eines Prozesses werden auf genau einen KL-Thread abgebildet.	Eine beliebige Anzahl UL-Threads (m) wird auf verschiedene KL-Threads (n) abgebildet. Pro Prozess existieren mehrere KL-Threads.	Jeder UL-Thread wird auf genau einen KL-Thread abgebildet.

Prozessverwaltung - Threads

2. Implementierung des Multithreading

   c. Bei einer der Implementierungs-Varianten blockieren bei einem Warten auf I/O innerhalb eines KL-Threads alle Threads eines UL-Threads.

      a. Um welche Implementierungs-Variante handelt es sich?

         Lösungsvorschlag:

         Um die m:1-Abbildung. Hierbei „kennt“ das Betriebssystem nur den sich aktuell in Ausführung befindlichen KL-Thread und teilt damit keinem der anderen UL-Threads Rechenzeit zu.

      b. Weshalb kann mit dieser Implementierungs-Variante dennoch eine parallele Verarbeitung erreicht werden?

         Lösungsvorschlag:

         Wenn die zugrundeliegende Hardware mehr als einen Rechenkern besitzt, können mehrere KL-Threads jeweils auf die physikalisch existenten CPU-Kerne aufgeteilt werden. Dadurch ist applikationsübergreifend eine parallele Verarbeitung möglich.

   d. Bei einer m:n-Abbildung wird für die KL-Threads ein Thread-Pool verwendet.

      Welche Vorteile ergeben sich dadurch?

      Lösungsvorschlag:

      Unter der Verwendung eines Thread-Pools können Threads wiederverwendet werden, was eine schnellere Performance ermöglicht.

      Der Thread-Pool hat eine obere Grenze an zur Verfügung stehenden Threads. Wird diese erreicht, beginnen sich alle weiteren Threads in einer Warteschlage einzureihen. Dadurch wird der Verwaltungsaufwand der Threads in Grenzen gehalten.

Prozessverwaltung - Threads

2. Implementierung des Multithreading

   e. Wie unterscheidet sich bei einer m:n-Abbildung der Ansatz der Scheduler Activation von dem der Pop-up-Threads?

   Lösungsvorschlag:

   Bei der Scheduler Activation wird ein Thread-Pool zur Abbildung vorhandener UL-Threads auf KL-Threads bereitgestellt, sodass Threads wiederverwendet werden können. Pop-Up Threads verfolgen das gegenteilige Ziel, indem sie Threads für kurze Zeit erzeugen und direkt im Anschluss wieder beendigen. Des Weiteren werden Pop-Up Threads im Allgemeinen dazu verwendet, einkommende Nachrichten / Signale auf z. B. Netzwerkebene zu verarbeiten. Bei der Scheduler Activation hingegen werden Threads auf Benutzerebene erzeugt.

Prozessverwaltung - Threads

3. Threads in der Praxis

a. Welche Probleme können sich bei der Verdrängung eines Threads ergeben, wenn sich dieser mit anderen Threads gemeinsam Ressourcen teilt?

Lösungsvorschlag:

Es kann sowohl beim Lesen als auch beim Schreiben von gemeinsamen Daten zu Problem kommen. Inkonsistente Daten / unvollständige Datenstände können dabei die Folgen sein:

Der Thread kann während des Zurückschreibens von Daten unterbrochen werden (Lost Update)

Ein Thread kann fehlerhafte / unvollständige Daten eines anderen Threads wiederverwenden (Inconsistent Read)

b. Erklären Sie das Task-Konzept moderner Programmiersprachen

Lösungsvorschlag:

Ein Task bildet eine Aufgabe ab, nicht den dazu notwendigen Thread. Dadurch: Fokussierung auf die eigentliche Aufgabe und nicht auf technische Implementierungsdetails des Multithreading. Für das effiziente Erstellen und Verwalten von Threads sorgt die entsprechende Laufzeit-Umgebung.

c. Welche Vorteile bietet die Arbeit mit Tasks gegenüber der direkten Verwendung von Threads?

Lösungsvorschlag:

Fokussierung auf eigentliche Aufgabe, einfache Veröffentlichung des Task-Fortschrittes, unkompliziertes Unterbrechen / abbrechen eines bereits gestarteten Tasks, einfache Verkettung von Aufgaben bzw. Fortsetzungsaufgaben → Continuation Tasks.

Allgemein ausgedrückt: Abstraktion von Komplexität beim Erzeugen und Verwalten von parallelen Aufgaben.