

Aufgabe 02-1(Threads)

1. Wann ist es sinnvoll, nebenläufige Programmteile mit Hilfe von Threads anstatt Prozessen zu implementieren?

Ein Programm (Prozess) besitzt mindestens einen Thread. Die Nutzung von weiteren Threads ist dann sinnvoll, wenn eine schnellere Prozessausführung innerhalb des selben Prozesses gewünscht ist, welche den gleichen Adressraum teilen. Diese ist bei einem Multikernsystem durch parallelisierung von mehreren Threads möglich. Durch die Leichtgewichtigkeit eines Threads kann es schneller zur Erstellung und gewünschten Kontextwechsels kommen. Durch das Teilen des Adressraums können Daten untereinander ausgetauscht werden.

2. Welche Vorteile bieten User-Level-Threads gegenüber den Kernel-Level-Threads? Gibt es auch Nachteile?

Die Funktion der beiden Thread-Arten unterscheidet sich, daher gehen die eigentlichen Vor- und Nachteile aus der Verwendung hervor. Vorteile eines User-Level-Threads für die Entwicklung ist die Lenkbarkeit und der Zugriff über eine einheitliche Thread-Library. Die erstellten Threads sind schneller, einfacher zu erstellen zu steuern und zu mangen. Sie funktionieren plattformunabhängig und benötigen keine Kernel mode privilegien um einen Kontextwechsel vorzunehmen. Das Kernel weiß nichts von den verschiedenen Threads aus User-Level und behandelt die Prozesse wie eine single-Thread-Application, das kann zu Nachteilen führen. Das Multithreading kann nicht zum Vorteil der Anwendung genutzt werden. Falls ein User-Level-Thread eine Blockieroperation, wird der gesamte Prozess blockiert. Dies kann bei einem Kernel-Level-Thread nicht passieren. Multithreading kann über das Kernel geschedulart werden und bei der Blockierung eines Threads, kann ein anderer Thread innerhalb des Prozesses mit der Operation vortfahren.

3. Welche Vorteile und Nachteile gibt es, wenn man Thread-Kontrollblock(TCB) als Skalare, in Arrays, Listen, Bäumen oder invertierten Tabellen speichert?

Struktur	Vorteil	Nachteil
Skalar	Einfache Implementierung	Unterstützt keine Mehrfach-Threads
Array	Schneller Indexzugriff, gut für feste Prioritäten	Unflexibel, schlecht bei dynamischer Anzahl
Liste	Einfaches Einfügen/Löschen	Langsame Suche, langsam bei prioritätsbasiertem Scheduling
Baum	Sehr gut für Prioritäten, schnelle Operationen ($O(\log n)$)	Komplexe Implementierung, zusätzlicher Overhead
Invertierte Tabelle	Schneller Lookup über ID → TCB	Benötigt Hashing oder direkte Indexierung, Kollisionsgefahr

Table 1: Vergleich von Datenstrukturen zur Speicherung von TCBs

4. In welchem Adressraum wird ein TCB gespeichert?

Der TCB eines Kernel-Threads wird im Kernel-Adressraum gespeichert, typischerweise im Kernelspeicher des Betriebssystems. Sie können auch durch Kerneloperationen gesteuert und gemangelt werden.