**Betriebssysteme: Sheet 02**

Feliks & Bennet

# Aufgabe 02-1(Threads)

**1.Wann is 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 mehrern 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. Voreile eines User-Level-Threads für die Enwicklung ist die Lenkbarkeuit und der Zugriff über eine einheiltlcihe Thread-Libary. Die erstellten Threads sind schneller, einfacher zu erstellen zu steuern und zu mangen. Sie funktionieren Plattformunabhängig und benötigen keine Kernel mode priviliegien um einen Kontextwechsel vorzunehmen. Das Kernel weiß nichts von den verschiedenen Threds aus User-Level und behandelt die Prozesse wie eine 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 gesammte Prozess blockiert. Dies kann bei einem Kernel-Level-Thread nicht passieren Multithreading könenn über das Kernel geschedulart werden und bei der Blockierung eines Threads, kann ein anderer Thread innerhalb des Prozesses mit der Operastion 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 gemanged werden.

## Aufgabe 02-2(Behandlung von Ausnahmen)

- Wir haben leider nicht geschafft den Code rechtzeitig zum Laufen zu bringen
- Es war einfach zu wenig Hilfestellung und kein guter Weg zu testen das es läuft
- Gdb schien uns auch nicht helfen zu wollen
- Wobei später gdb wieder ging aber nicht ausgereicht hat um Sachen ausreichend zu fixen

- Memory remap schien noch ganz ok zu gehen
- Interrupt Vektor Table und Stack initieren war etwas schwierig, wo ultimativ die Base Funktion nicht funktioniert hat und somit wir die erweiterte Funktionalität nicht geschaft haben

- Code kompeliert
- Zuvor ging der test print durch nach dem init stack aber es hat keine interrupts behandelt
- Jetzt hat es Probleme nach dem stack init und hört danach auf