Aufgabe 14: Threads

g) Beschreiben Sie die unterschiedlichen Thread-Formen:

- •schwergewichtige,
- •mittelgewichtige und
- •leichtgewichtige Threads.

Diskutieren Sie die Vor- und Nachteile dieser verschiedenen Thread-Formen.

h) Gegeben ist ein MIMD-Multiprozessor mit 6 Rechenwerken. Folgende Aufträge stehen in der Warteschlange:

Auftrag	benötigte Threads
1	3
2	4
3	2
4	3

Zeigen Sie, in welcher Reihenfolge die Aufträge gescheduled werden, wenn folgende drei Verfahren zum Gang-Scheduling von Ousterhout eingesetzt werden:

- •Matrix-Scheduling
- •fortlaufendes Scheduling
- •ungeteiltes Scheduling

Literatur:

Singhal, M., Shivaratri N.:
Advanced Concepts in Operating Systems
McGraw-Hill, Inc., 1994

Kleiman, S., Shah D., Smallders B.:
Programming with Threads
SunSoft-Press, 1996

Lösung 14:

a) die unterschiedlichen Thread-Formen besitzen folgende Eigenschaften:

Schwergewichtige Threads (Prozesse): Diese Form entspricht den Unix-Prozessen. Jeder Thread und damit jeder Aktivitätsträger besitzt einen eigenen Adreßraum.

Vorteil:

- •Kapselung der Daten der unterschiedlichen Aktivitätsträger.

Nachteil:

- •Teure Umschaltung: jeder Threadwechsel führt zu einer Invalidierung des Adreßraumes. Dadurch werden Wechsel von Aktivitätsträgern ineffizient (Cache-Invalidierung, TLB-Invalidierung).
- •Kommunikation: Inter-Prozess-Kommunikation ist schwierig und teuer.

Mittelgewichtige Threads (Kernel-Threads): Bei dieser Form bietet der Kernel mehrere Aktivitätsträger in einem Adressraum an. Das Management dieser Threads bleibt in der Hand des Kerns.

Vorteile:

- •Kontrolle: Der Kern hat volle Kontrolle über die Threads. Koordinierung und Scheduling können dadurch aufeinander abgestimmt werden.
- •Effizienz: Im Vergleich zu Prozessen sind Kernel-Threads effizienter. Bei Thread-Umschaltungen innerhalb eines Adreßraumes muß der Adreßraum nicht invalidiert werden.
- •Anwendungen, die viele Systemcalls durchführen sind zur Verwendung von Kernel-Threads geeignet.

Nachteile:

- •Management-Kosten: Thread-Management ist teuer: jede Management-Operation benötigt einen Systemcall. Bei Verwendung von Übergabe-Parametern müssen Schutz-Abfragen durchgeführt werden, um die Sicherheit des Kerns zu gewährleisten.
- •Generalität: Da die Management-Strukturen im Kern verankert sind, können diese nicht an die Anwendungen angepaßt werden. Dadurch müssen alle potentiell verwendbaren Mechanismen im Thread-Management implementiert werden. Die Folge ist ein potentieller Overhead bei der Ausführung von Management-Aktionen.

Leichtgewichtige Threads (User-Level-Threads): Diese Threads werden im User-Level ausgeführt. Durch eine Bibliothek wird die Funktionalität der Threads dem Benutzer zur Verfügung gestellt. Dabei werden vom Kern zur Verfügung gestellte Aktivitätsträger als virtuelle Prozessoren verwendet.

Vorteile:

- Effizienz: Um zwischen Threads umzuschalten, ist kein System-Call notwendig. Dadurch wird eine noch effizientere Abarbeitung der Threads als bei Kernel-Threads verwirklicht. Die Umschaltung zweier Threads einer Applikation hat annähernd den gleichen Overhead wie ein einfacher Funktionsaufruf.
- Kompatibilität: Für die Verwendung von User-Level-Threads ist keine besondere Unterstützung vom Betriebssystem notwendig.
- Flexibilität: Für unterschiedliche Problemstellungen können die Thread-Libraries angepaßt werden. Insbesondere das Scheduling der Threads einer Applikation wird damit auf User-Level und damit vom Benutzer einstellbar.

Nachteile:

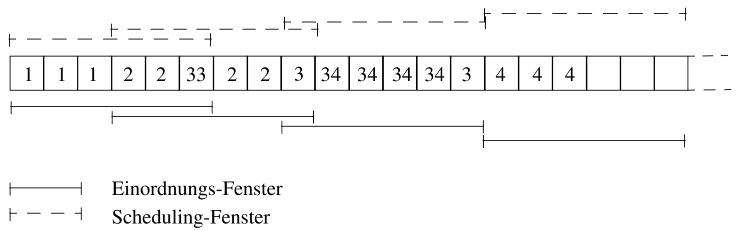
- Abhängigkeit: User-Level-Threads sind vom Kern abhängig. Werden zum Beispiel im Kern mehr virtuelle Prozessoren als physikalische Prozessoren zur Verfügung gestellt, so muß durch eine Strategie die Zuordnung gescheduled werden. Dieser Vorgang geschieht in aller Regel transparent für die User-Level-Threads, so daß diese keinen Einfluß auf diese Zuordnung haben. Die Folge sind Verhalten, die in der Spezifikation der Threads nicht geplant werden können. So zum Beispiel: keine Gewährleistungen, ineffizientere Abarbeitung, vermeidbare Spinlocks und Deadlocks.
- Verlust von Parallelität bei blockierenden Systemcalls: durch die Zuordnung mehrerer User-Level-Threads auf einen virtuellen Prozessor sind bei einem blockierenden Systemcall eines Threads sämtliche Threads des gleichen virtuellen Prozessors nicht lauffähig. Lösungen dieser Problematik bieten Konzepte wie die “Sleeping Threads” oder die “Scheduler Activations” an: wird ein Aktivitätsträger blockiert, so bietet der Kern einen Ersatz-Aktivitätsträger an (bzw. einen virtuellen Ersatz-Prozessor). Ein weiterer Lösungsvorschlag basiert auf einer Kommunikations-Möglichkeit zwischen den Schedulern in der User-Level Thread-Library und dem System-Scheduler (First-Class Threads aus dem Betriebssystem Psyche).

b) Gang-Scheduling von Ousterhout:

- *Matrix-Scheduling*: die Aufträge werden in einer Matrix angeordnet. Die Anzahl benötigter Threads darf die Zeilenlänge nicht überschreiten.

	Prozessoren					
Zeit-scheiben	1	1	1	3	3	
	2	2	2	2		
	4	4	4			

- *fortlaufendes Scheduling*: die Matrix des vorhergehenden Falls wird linearisiert. Aufträge werden in einem Fenster eingeordnet, dessen Breite der Anzahl zur Verfügung stehenden Prozessoren entspricht. Passt der Auftrag nicht in das aktuelle Fenster, wird dieses so lange nach rechts verschoben, bis zum erstenmal das linke Feld des Fensters leer ist. Dies wird so oft wiederholt, bis der Auftrag eingetragen werden kann. Gescheduled wird durch ein Scheduling-Fenster, das nach den jeweiligen Zeitquanten so weitergeschoben wird, daß der linke Eintrag des Fensters zu einem Auftrag gehört, der in der vorherigen Zuordnung noch nicht vollständig oder garnicht zugeordnet war:



- *ungeteiltes Scheduling*: Analoge Vorgehensweise zum fortlaufenden Scheduling. Allerdings werden die Aufträge nur zusammenhängend eingeordnet: