Detaillierte Testergebnisse für Testdurchlauf 1

1. Prozesse und Threads [ID: 76659]

Was ist die Motivation für Threads, wo doch schon Prozesse die Funktionalität der Aufgabenteilung bieten? Gib mindestens 2 Gründe für die Notwendigkeit von Threads.

Threads teilen sich den Speicher

Bei einem Interrupt stoppt nicht der ganze Prozess und reduziert so Zeit, in der ein Prozessor untätig sein könnte

Nicht der ganze Prozess muss auf einen User Input warten

1000 Zeichen zugelassen, Anzahl der eingegebenen Zeichen: **229**

2. Prozesse: Stack und Heap [ID: 76660]

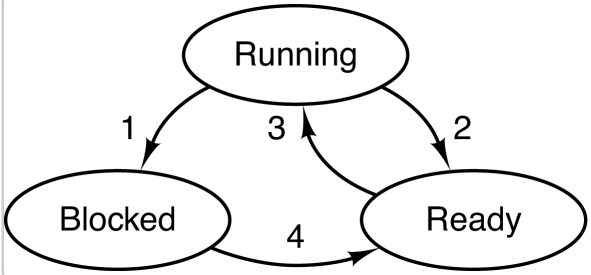
Was ist der Unterschied zwischen Stack und Heap? Definiere die beiden Begriffe und grenze sie von einander ab. (je 0.5)

Der Stack ist ein reservierter Speicherraum. Dabei wird nach dem LIFO Prinzip (Last In First Out) gearbeitet. Wird also eine Funktion augerufen kommen die Variablen dieser oben auf den Stack. Der am aktuellsten reservierte Speicher ist also der erste, der wieder freigegeben wird.

Beim Heap wird ebenfalls Speicher reserviert. Der Unterschied zum Stack ist nun, dass hier nicht nach dem LIFO Prinzip gearbeitet wird. Das heisst, vom Heap kann jederzeit ein Block reserviert und wieder freigegeben werden.

1000 Zeichen zugelassen, Anzahl der eingegebenen Zeichen: **525**

3. Prozesszustände [ID: 76661]



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Auf dem oben angezeigten Bild sind drei Prozesszustände angezeigt. Eigentlich sollten also 6 Pfeile, welche je einen Übergang eines Zustands in einen anderen definieren, gezogen werden können. Welcher der nachfolgenden Pfeile ist jedoch **unmöglich**?

- ❏ Ready -> Blocked
- ❏ Blocked -> Running

4. UNIX-Prozesserzeugung [ID: 76663]

In UNIX müssen neue Prozesse mittels des ***fork***-Systemaufrufs erzeugt werden. Was passiert beim Aufruf von ***fork***? Wähle die am meisten korrekte Antwort.

- ❍ fork erzeugt einen neuen Prozess mit einer neuen PID
- ❍ fork erzeugt eine exakte Kopie des aufrufenden Prozesses, mit neuer Prozessnummer.
- ❍ fork erzeugt eine exakte Kopie des aufrufenden Prozesses, mit neuer Prozessnummer, und passt dann den Speicherbereich oder führt allenfalls ein neues Programm aus.

5. Threads: thread_yield [ID: 76665]

Wieso sollte ein Thread freiwillig mittels Aufruf ***thread_yield*** die CPU freigeben? Schliesslich kann es sein, dass es kein periodisches Clock-Interrupt gibt und der Thread nie die CPU zurückerhalten wird.

Geben Threads die CPU nicht frei, kann kein kooperatives Scheduling gemacht werden. Auf dem Userlevel weiss das OS nicht von der Existenz von User-Threads und kann also

kein Scheduling betreiben.

Unbegrenzt Zeichen zugelassen, Anzahl der eingegebenen Zeichen: **204**

6. Vor- und Nachteile der Implementation von Threads im User-Space [ID: 76666]

Nenne einen wichtigen Vorteil für die Implementation von Threads im User Space (0.5).
Nenne einen wichtigen Nachteil (0.5).

+) Programme können verzahnt ausgeführt werden.

-) Wird ein User-Thread blockiert, kann er die Kontrolle nicht an einen anderen Thread übergeben, wodurch das ganze Programm blockiert wird. Dies kann aber umgangen werden.

Unbegrenzt Zeichen zugelassen, Anzahl der eingegebenen Zeichen: **242**

7. Preforking [ID: 76706]

Webserver-Prozesse (u.a. Apache) verwenden häufig eine Technik namens "preforking", um Anfragen so schnell als möglich bearbeiten zu können.
Worin besteht diese Technik?

Ein fork ist ein komplett eigenständiger *nix Prozess. Pre-forking bedeutet dann, dass ein Master (Parent-Prozess) die Prozesse forken kann, bevor ein Request kommt. Dadurch stehen zukünftige, ähnliche Prozesse unverzüglich zur Verfügung.

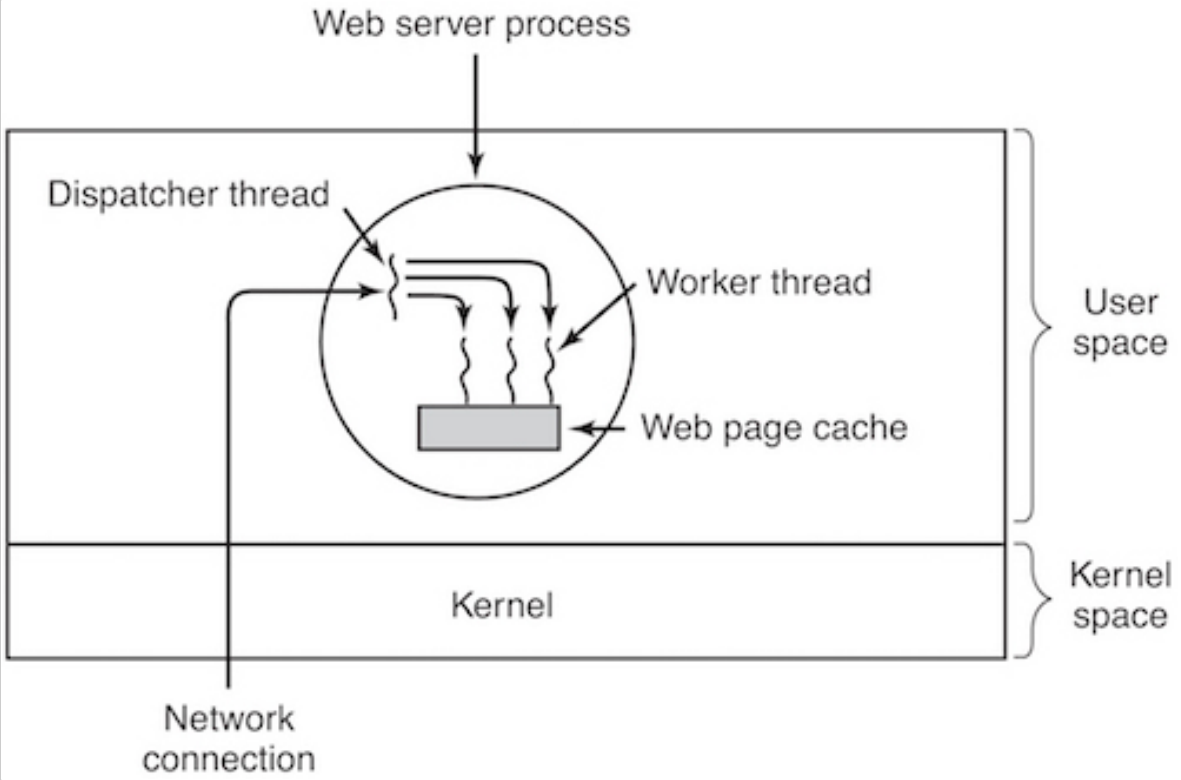
Unbegrenzt Zeichen zugelassen, Anzahl der eingegebenen Zeichen: **252**

8. User-Level und Kernel-Level Threads [ID: 76728]

Welche Aussagen bezüglich User-Level und Kernel-Level Threads sind korrekt?

- ❏ Kernel-Level Thread-Wechsel erfordert keine Interaktion mit dem Kernel
- ❏ User-Level Threads erlauben schnellere Thread-Wechsel.
- ❏ Faires Scheduling ist zwischen allen Threads eines Systems möglich.
- ❏ Kernel-Level Threads sind weniger empfindlich zur blockierenden Aufrufen.

9. User- und Kernel-Threads [ID: 76664]



Obige Grafik zeigt einen Web Server Prozess mit mehreren Threads. Wenn die einzige Möglichkeit, um eine Datei zu lesen, der blockierende ***read*** Systemaufruf ist, ist es

geeigneter User- oder Kernel-Threads zu verwenden? Wieso?

Kernel-Threads. Wenn dieser Thread dann blockiert ist, können die anderen Threads nocht weiterarbeiten.

Unbegrenzt Zeichen zugelassen, Anzahl der eingegebenen Zeichen: **113**