Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

4 Threads

4.1 Allgemein

Prozessmodell:
Zwei unabhängige Eigenschaften eines Prozesses:
- Er hat Resourcen: Adressen, Daten, geöffnete Dateien
- und ist ausführbar

Thread:
- hat keine eigenen Resourcen (ausser Thread Local Storage, TLS)
- ist ausführbar (Umschaltung wie bei Prozess)
- erlaubt Nebenläufigkeit innerhalb einer Prozessumgebung

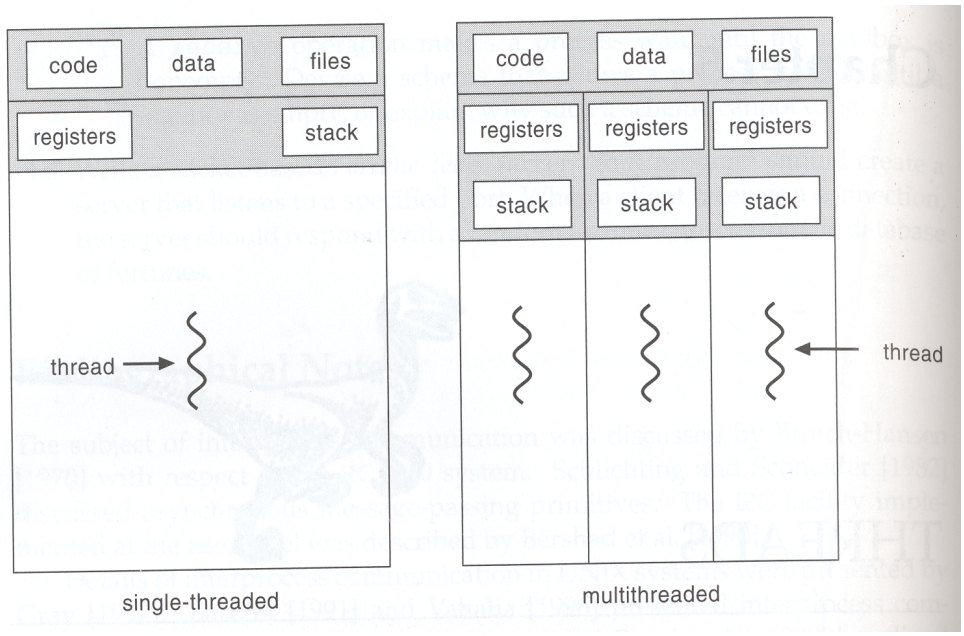

Abbildung 33: Thread vs. Prozess (nach [2])

Begriffsdefinition:
- Ein Prozess hat mindestens einen **Ausführungs-Thread**.
- Threads werden auch **LWP (Light Weight Processes)** genannt.
- **Thread-Local Storage (TLS)**: Daten, die einem bestimmten Thread zugeordnet sind (sind manchmal zur Programmierung nötig)

Prozess	***Thread***
Addressraum	Befehlszähler
Globale Variablen	Register
Geöffnete Dateien	Stack
Kindsprozesse	Zustand

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

Prozess	*Thread*
Ausstehende Alarme	
Signale, Signalhandler	
Verwaltungsdaten	

-> kein Schutz zwischen verschiedenen Threads!

4.1.1 Anwendungsbeispiel 1: MS Word

- 1 Programm bearbeitet 1 Dokument; dennoch sind verschiedene Aufgaben gleichzeitig zu erledigen:
  - Anzeige des Dokumentes, Scrollen
  - Drucken im Hintergrund
  - Speichern im Hintergrund
- Verschiedene Prozesse: Kein Zugriff auf die Daten möglich
- Kein Zugriffsschutz nötig, da ja alle Teile zu MS Word gehören

4.1.2 Anwendungsbeispiel 2: Web-Server

Der Web-Server bearbeitet verschiedene Anfragen; für jede Anfrage ist z.B. ein Zugriff auf das Datei-System nötig (blocking I/O): während dieser Zeit können keine weiteren Anfragen bearbeitet werden
Deshalb: jede Anfrage wird in einem eigenen Thread bearbeitet.
-> bessere Performance

4.2 Threaderzeugung

Vergleich von Prozesserzeugung und Threaderzeugung:

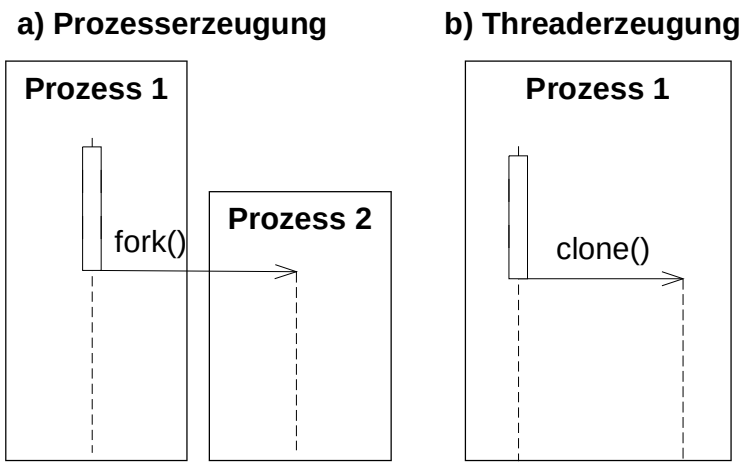

Abbildung 34: Threaderzeugung (einfache Darstellung ohne Kernel)

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

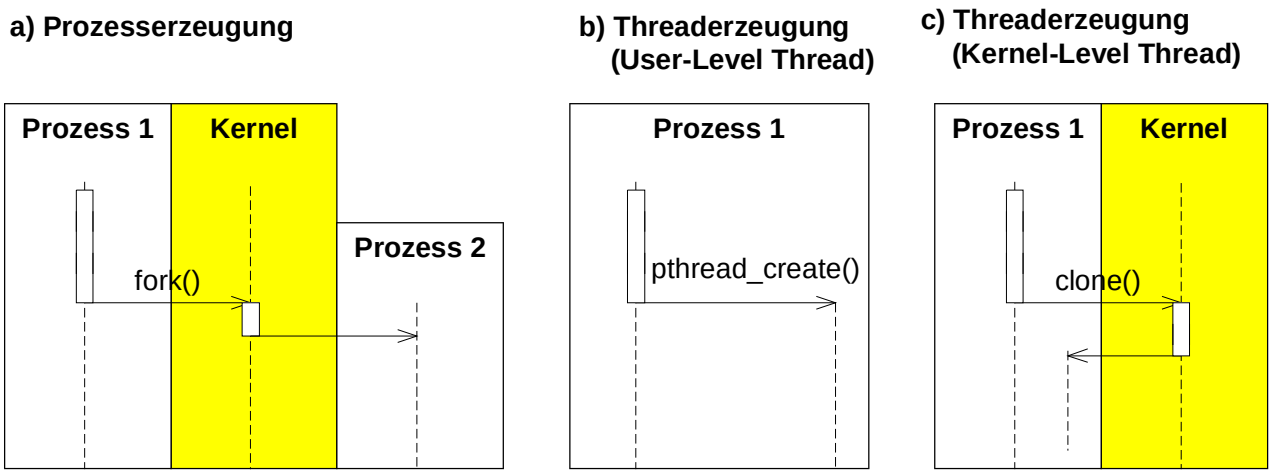

Abbildung 35: Thread-Erzeugung (Darstellung mit Kernel)

Allgemeine Parameter bei der Threaderzeugung:

- Arbeitsfunktion (worker function): eine Funktion, die im Thread abgearbeitet wird
- Stack für neuen Thread:
  - Extra Speicherbereich
  - oder Bereitstellung durchs Betriebssystem
- Funktionsparameter: 1 Parameter reicht, da typischerweise ein Zeiger auf Datenstrukturen übergeben werden kann; dort können beliebig viele Daten gespeichert werden (gemeinsamer Speicherbereich!)
- Einstellungen (Flags):
  - Optionen bei der Erzeugung, z.B. Übernahme offener Dateien, etc.
  - Sicherheitseinstellungen (Win32)

4.2.1 Linux: clone()

Dient zum Erzeugen eines neuen Threads bzw. LWP (Light-Weight-Process)

Parameter:

- Thread Worker-Funktion: enthält die Funktion, die im Thread ausgeführt werden soll
  - wenn die Funktion zurückkehrt, wird der Thread beendet
- Child-Stack: eigener Stack wird benötigt; hier kann der benötigte Speicher zur Verfügung gestellt werden
- Flags: verschiedene Einstellungen sind möglich; einige sind:
  - CLONE_PARENT: Thread hat denselben Vater wie der erzeugende Prozess
  - CLONE_FS: Dateisystem wird zwischen Kind und Vater geteilt (shared)
  - CLONE_FILES: Dateien werden zwischen Kind und Vater geteilt (shared)

4.2.2 Win32: CreateThread()

- Win32-Befehl zum Erzeugen und Vernichten eines neuen Threads: CreateThread(), ExitThread()
- In der C-Standardbibliothek: _beginthread(), _exitthread()

Parameter:

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

- Sicherheitskontext
- Stack-Größe
- Funktionszeiger auf Arbeitsfunktion (worker function), die nach dem Erzeugen des Threads aufgerufen wird
- 1 Parameter, der der Arbeitsfunktion übergeben wird (typischerweise ein Zeiger auf eine Datenstruktur)
- Flags zur Threaderzeugung, z.B.
    - CREATE_SUSPENDED: der Thread wird erzeugt, ist aber suspendiert (läuft nicht)
- Rückgabe-Parameter: Thread-ID des erzeugten Threads

4.2.3 POSIX: pthread-create()

POSIX-Threads: erzeugen mit pthread_create()

Parameter:
- Rückgabe-Parameter Thread-ID
- Thread-Attribute
- Arbeitsfunktion
- Argument

POSIX-Man-Pages:

NAME
pthread_create - thread creation

SYNOPSIS

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
    void *(*start_routine)(void*), void *arg);
```

DESCRIPTION
The *pthread_create()* function is used to create a new thread, with attributes specified by *attr*, within a process. If *attr* is NULL, the default attributes are used. If the attributes specified by *attr* are modified later, the thread's attributes are not affected. Upon successful completion, *pthread_create()* stores the ID of the created thread in the location referenced by *thread*.

The thread is created executing *start_routine* with *arg* as its sole argument. If the *start_routine* returns, the effect is as if there was an implicit call to *pthread_exit()* using the return value of *start_routine* as the exit status. Note that the thread in which *main()* was originally invoked differs from this. When it returns from *main()*, the effect is as if there was an implicit call to *exit()* using the return value of *main ()* as the exit status.

The signal state of the new thread is initialised as follows:

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

- The signal mask is inherited from the creating thread.
- The set of signals pending for the new thread is empty.

If *pthread_create()* fails, no new thread is created and the contents of the location referenced by *thread* are undefined.

RETURN VALUE
If successful, the *pthread_create()* function returns zero. Otherwise, an error number is returned to indicate the error.

4.3 User- und Kernel-Threads

Unterschiedliche Threads im User-Space und im Kernel-Space:

- User-Threads: werden vom Programm explizit erzeugt & führen eine Berechnung durch (Arbeitsfunktion)
- Kernel-Threads: laufen im Kernel, um Kernel-spezifische Aufgaben zu erledigen
  - z.B. I/O, Datenstrukturen updaten, etc.
- Bei Systemaufrufen wechselt der ausführende Thread: ein User-Thread übergibt die Kontrolle an einen Kernel-Thread

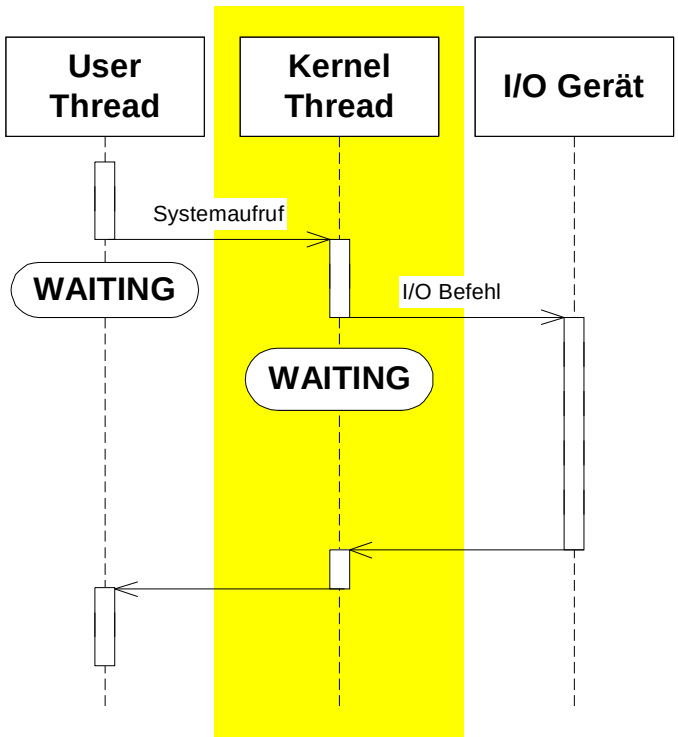

Abbildung 36: Wechsel von User-Thread zu Kernel-Thread

4.3.1 User-Level Threads

Sind auf User-Ebene implementiert durch eine System-Bibliothek, wie z.B. POSIX Pthreads.

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

Vorteile:
- Einfach zu implementieren
- Sehr effizient, da kein Übergang von User-Mode in Kernel-Mode nötig ist
- Keine Kernel-Unterstützung nötig -> können auch auf Betriebssystemen realisiert werden, die keine Thread-Unterstützung bieten

Problem:
- Prozess hat mehrere User-Level-Threads und ruft nun eine System-Funktion auf, die blockiert (z.B. I/O). Dann wird der gesamte Prozess in den Zustand WAITING versetzt, obwohl andere Threads noch lauffähig wären

4.3.2 Kernel-Level Threads

Sind Bestandteil des Kernels. Alle modernen Betriebssysteme unterstützen Threads.

Vorteile:
- Mehr Nebenläufigkeit:
    - Threads können auf Mehrprozessor-Maschinen auf verschiedenen Prozessoren laufen
    - Thread blockieren sich nicht gegenseitig

Nachteile:
- Betriebssystem-Unterstützung nötig
- Threadwechsel aufwändiger, da dies vom Kernel / Scheduler durchgeführt wird

4.3.3 Many-to-one Modell (User-Level-Threads)

Dies entspricht User-Level-Threads: mehreren Threads im User-Space ist ein Kernel-Thread zugeordnet. Wenn der Kernel-Thread blockiert, blockieren alle anderen Threads ebenso.

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

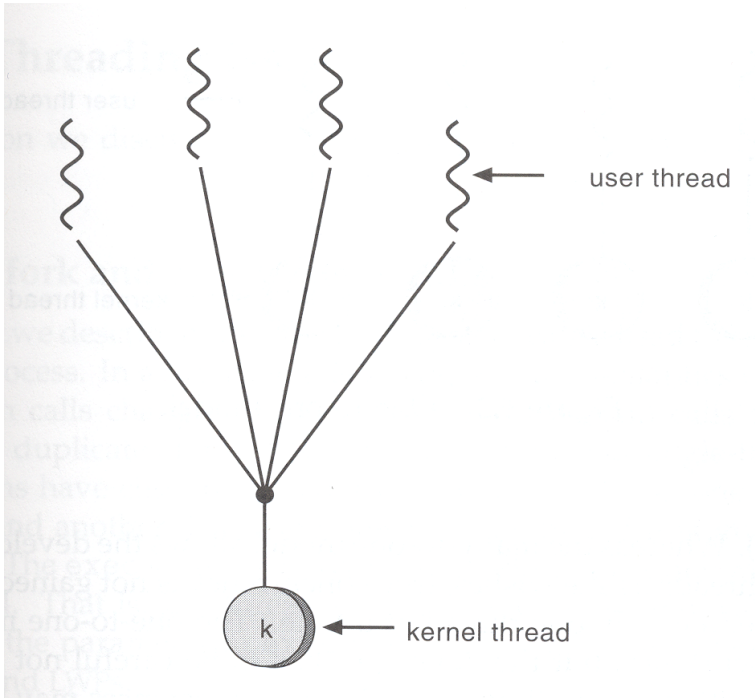

Abbildung 37: many-to-one Thread Modell (nach [2])

4.3.4 One-to-one Modell (Kernel-Level-Threads)

Jedem User-Level-Thread ist genau ein Kernel-Thread zugeordnet. Alle Threads können vom Scheduler getrennt behandelt werden.

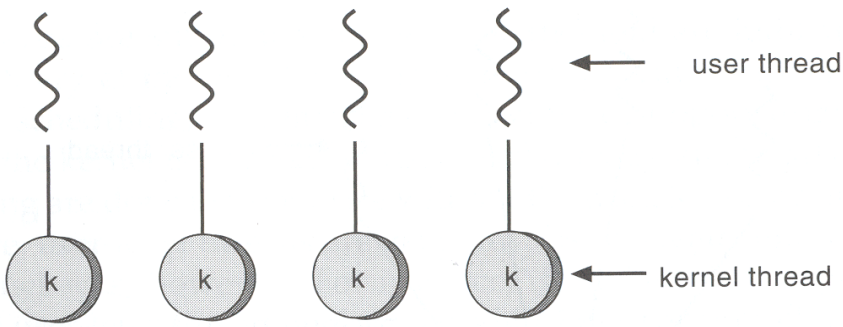

Abbildung 38: One-to-one Thread Modell (nach [2])

4.3.5 Many-to-many Modell (gemischt)

Jedem Thread im User-Space werden nach Bedarf Kernel-Threads zugeordnet.

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

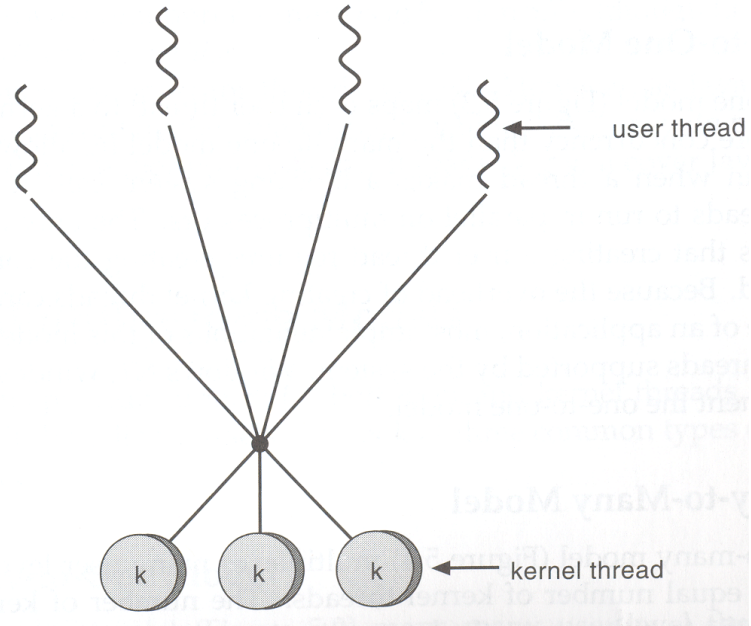

Abbildung 39: Many-to-many Thread Modell (nach [2])

Vorteile:
- Beliebig viele User-Threads möglich
- Zugeordnete Kernel-Threads erlauben hohes Mass an Nebenläufigkeit
- Viele Betriebssysteme unterstützen dieses Modell, z.B. Solaris 2

4.4 Beispiele

4.4.1 Pthreads: User-Level Bibliothek
- Implementiert den POSIX-Standard
- User-Level Threads, überall verfügbar

4.4.2 Win2000 Threads
- One-to-one Thread Modell: jeder User-Thread entspricht einem Kernel-Thread
- Zusätzliche Fiber-Bibliothek ermöglicht ein many-to-many Thread-Modell

4.4.3 Solaris 2 Threads
- Pthreads auf User-level
- Zwischenebene: LWPs
    - Jeder Prozess hat zumindest einen LWP
    - Die Thread-Bibliothek bildet die verschiedenen User-Threads nach Bedarf auf LWPs ab
    - Jeder LWP hat einen Kernel-Thread
- Standard-Kernel-Threads dienen zur Abwicklung aller Kernel-Aufgaben

Dr. Ulrich Margull
Kirchweg 6 * 93055 Regensburg
Tel. (+49) 941 78956-8 Fax –9
www.margull.de

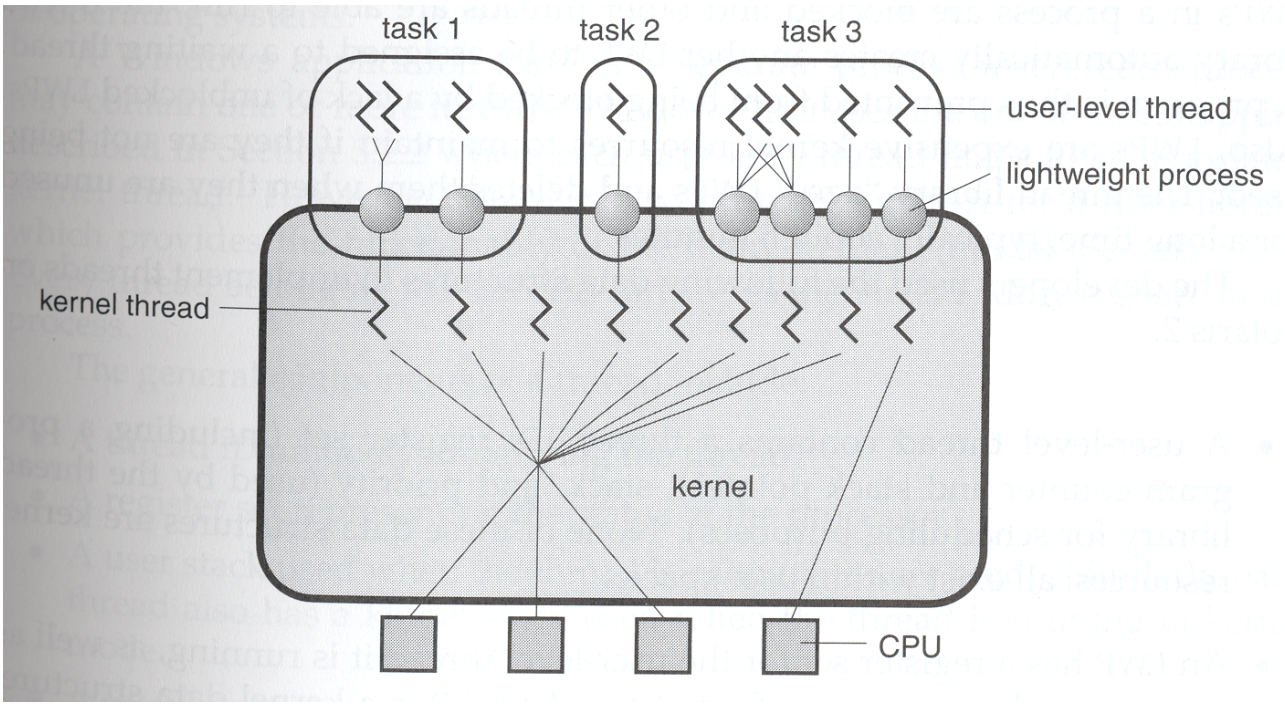

Abbildung 40: Solaris 2 Threading Modell (nach [2])

4.4.4 Linux-Threads

- clone()-Befehl, ähnlich wie fork()

4.4.5 Java Threads

Erinnerung: Java läuft in der Java Virtual Machine

- JVM verwaltet Java-Threads
    - weder User- noch Kernel-Threads im klassischen Sinne
    - Mapping auf Kernel-Threads hängt von der jeweiligen Java-VM ab
- Objekt-Orientiert:
    - Lauffähige Objekte werden von java.lang.Thread abgeleitet
    - müssen eine run() Funktion bereitstellen, die beim Starten des Thread aufgerufen wird (Arbeitsfunktion)
    - Thread wird durch o.start() erzeugt, welches implizit o.run() aufruft – o.run() wird nie direkt aufgerufen!

-> dazu gibt es eine Übung!