

# Übungen zu Systemprogrammierung 1 (SP1)

## VL 7 – Threads und Koordinierung

**Jens Schedel, Christoph Erhardt, Jürgen Kleinöder**

Lehrstuhl für Informatik 4  
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität  
Erlangen-Nürnberg

WS 2012/13 – 14. Januar bis 18. Januar 2013

[http://www4.cs.fau.de/Lehre/WS12/V\\_SP1](http://www4.cs.fau.de/Lehre/WS12/V_SP1)

## Agenda

- 7.1 Hinweise zur Evaluation
- 7.2 (Mini-)Klausurvorbereitung
- 7.3 Threads
- 7.4 Schnittstelle
- 7.5 Koordinierung
- 7.6 Aufgabe 6: palim
- 7.7 Gelerntes Anwenden



## Agenda

- 7.1 Hinweise zur Evaluation
- 7.2 (Mini-)Klausurvorbereitung
- 7.3 Threads
- 7.4 Schnittstelle
- 7.5 Koordinierung
- 7.6 Aufgabe 6: palim
- 7.7 Gelerntes Anwenden



## Hinweise zur Evaluation

- Bei Kommentaren, die sich auf einen bestimmten Übungsleiter beziehen, bitte dessen Namen **in jedem Feld** voranstellen
  - Kommentarfelder werden in der Auswertung durcheinandergewürfelt
- Frage „eigener Aufwand zur Vor- und Nachbereitung“:
  - Bitte nach Vorlesung und Übung auftrennen
  - Übung: den jeweiligen Wochenaufwand durch 2 teilen (rechnerisch: 2 x 1 Stunde Übung (Tafel + Rechner) pro Woche, Angabe je 45 Minuten)
  - Vorlesung: den jeweiligen Wochenaufwand nicht teilen (1 x 90 Minuten Vorlesung pro Woche, Aufwandsangabe je 90 Minuten)



## (Mini-)Klausurvorbereitung

- In zwei Wochen Klausurvorbereitung in der Tafelübung zur Vorbereitung auf
  - die SP1-Klausur für Mathematiker, Technomathematiker und 2-Fach-Bachelor
  - die Miniklausur zu Beginn von SP2 für alle Anderen
- Wir erarbeiten die SP1-Klausur aus dem Juli 2012 gemeinsam
  - Eine Vorbereitung der Klausur im Vorfeld der Tafelübung wird erwartet

07-palim\_handout



## Agenda

- 7.1 Hinweise zur Evaluation
- 7.2 (Mini-)Klausurvorbereitung
- 7.3 Threads
- 7.4 Schnittstelle
- 7.5 Koordinierung
- 7.6 Aufgabe 6: palim
- 7.7 Gelerntes Anwenden

07-palim\_handout



## Motivation von Threads

- UNIX-Prozesskonzept (Ausführungsumgebung mit einem Aktivitätsträger) für viele heutige Anwendungen unzureichend
  - keine parallelen Abläufe innerhalb eines virtuellen Adressraums auf Multiprozessorsystemen
  - typische UNIX-Server-Implementierungen benutzen die **fork**-Operation, um einen Server-Prozess für jeden Client zu erzeugen
    - Verbrauch unnötig vieler System-Ressourcen
  - zur besseren Strukturierung von Problemlösungen sind oft mehrere Aktivitätsträger innerhalb eines Adressraums nützlich
- Lösung: bei Bedarf weitere Threads in einem UNIX-Prozess erzeugen

07-palim\_handout



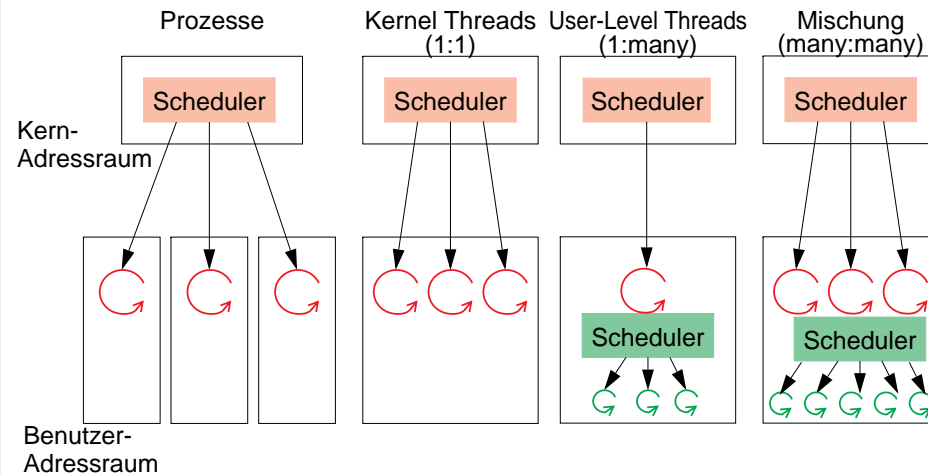
## Arten von Threads

- User-Level-Threads / federgewichtige Prozesse
  - Realisierung von Threads auf Anwendungsebene
  - Systemkern sieht nur den Prozess mit **einem** Kontrollfluss
  - + Erzeugung von Threads und Umschaltung extrem billig
  - Systemkern hat kein Wissen über diese Threads
    - in Multiprozessorsystemen keine parallelen Abläufe möglich
    - wird ein Thread blockiert, ist der gesamte Prozess blockiert
    - Scheduling zwischen den Threads schwierig
- Kernel-Level-Threads / leichtgewichtige Prozesse
  - + Gruppe von Threads nutzt gemeinsam die Betriebsmittel eines Prozesses
  - + jeder Thread ist als eigener Aktivitätsträger dem Betriebssystemkern bekannt
  - Kosten für Erzeugung und Umschaltung zwar erheblich geringer als bei „schwergewichtigen“ Prozessen, aber erheblich teurer als bei User-Level-Threads

07-palim\_handout



## Arten von Threads



07-palim\_handout



## Arten von Threads

- Mischung: eine große Zahl von User-Level-Threads wird auf eine kleinere Zahl von Kernel-Level-Threads abgebildet (many:many)
  - + User-Level-Threads sind billig
  - + die Kernel-Threads ermöglichen echte Parallelität auf einem Multiprozessor
  - + blockierender User-Level-Thread blockiert nur den Kernel-Level-Thread in dem er gerade abgewickelt wird
    - andere Kernel-Level-Threads können andere lauffähige User-Level-Threads weiterhin auszuführen

07-palim\_handout



## Agenda

- 7.1 Hinweise zur Evaluation
- 7.2 (Mini-)Klausurvorbereitung
- 7.3 Threads
- 7.4 Schnittstelle
- 7.5 Koordinierung
- 7.6 Aufgabe 6: palim
- 7.7 Gelerntes Anwenden

07-palim\_handout



## Pthreads-Schnittstelle

- Thread erzeugen

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

- thread Thread-ID
- attr Modifizieren von Attributen des erzeugten Threads (z. B. Stackgröße). NULL für Standardattribute.
- Nach der Erzeugung führt der Thread die Funktion `start_routine` mit Parameter `arg` aus
- Im Fehlerfall wird `errno` nicht gesetzt, aber ein Fehlercode als Ergebnis zurückgeliefert.
  - Um `perror(3)` verwenden zu können, muss der Rückgabewert erst in der `errno` gespeichert werden.

- Eigene Thread-ID ermitteln

```
pthread_t pthread_self(void)
```

- Die Funktion kann nie fehlschlagen.

07-palim\_handout



## Pthreads-Schnittstelle

- Thread beenden (bei Rücksprung aus `start_routine` oder):

```
void pthread_exit(void *retval)
```

- Der Thread wird beendet und `retval` wird als Rückgabewert zurück geliefert (siehe `pthread_join()`)

- Auf Thread warten, Ressourcen freigeben und Rückgabewert abfragen:

```
int pthread_join(pthread_t thread, void **retvalp)
```

- Wartet auf den Thread mit der Thread-ID `thread` und liefert dessen Rückgabewert über `retvalp` zurück.

- Ressourcen automatisch bei Beendigung freigeben:

```
int pthread_detach(pthread_t thread)
```

- Die mit dem Thread `thread` verbundenen Systemressourcen werden bei dessen Beendigung automatisch freigegeben. Der Rückgabewert der Thread-Funktion kann nicht abgefragt werden.



## Beispiel

```
static double a[100][100], b[100], c[100];

int main(int argc, char* argv[]) {
    pthread_t tids[100];
    ...
    for (int i = 0; i < 100; i++)
        pthread_create(&tids[i], NULL, mult, (void *) i);
    for (int i = 0; i < 100; i++)
        pthread_join(tids[i], NULL);
    ...
}

static void *mult(void *cp) {
    int i = (int) cp;
    double sum = 0;
    for (int j = 0; j < 100; j++)
        sum += a[i][j] * b[j];
    c[i] = sum;
    return NULL;
}
```

- Parameterübergabe bei `pthread_create()` problematisch



## Parameterübergabe bei `pthread_create()`

- Generischer Ansatz mit Hilfe einer Struktur für die Argumente

```
typedef struct {
    int index;
} param;
```

- Für jeden Thread eine eigene Argumenten-Struktur anlegen

- Speicher je nach Bedarf auf dem Heap oder dem Stack allokalieren

```
int main(int argc, char* argv[]) {
    pthread_t tids[100];
    param args[100];

    for (int i = 0; i < 100; i++) {
        args[i].index = i;
        pthread_create(&tids[i], NULL, (void*) (*) (void*) mult,
                      (void *) (&args[i]));
    }
    for (int i = 0; i < 100; i++)
        pthread_join(tids[i], NULL);
    ...
}
```



## Parameterübergabe bei `pthread_create()`

```
static void* mult (param *arg) {
    double sum = 0;
    for (int j = 0; j < 100; j++) {
        sum += a[arg->index][j] * b[j];
    }
    c[arg->index] = sum;
    return NULL;
}
```

- Zugriff auf den threadspezifischen Parametersatz über `arg`



# Agenda

- 7.1 Hinweise zur Evaluation
- 7.2 (Mini-)Klausurvorbereitung
- 7.3 Threads
- 7.4 Schnittstelle
- 7.5 Koordination
- 7.6 Aufgabe 6: palim
- 7.7 Gelerntes Anwenden

07-palim\_handout



# Koordinierung – Motivation

Welches Problem kann hier auftreten?

```
static double a[100][100], sum;

int main(int argc, char* argv[]) {
    pthread_t tids[100];
    param args[100];

    for (int i = 0; i < 100; i++) {
        args[i].index = i;
        pthread_create(&tids[i], NULL, (void*) (*)(void*)sumRow,
                      &args[i]);
    }
    for (int i = 0; i < 100; i++)
        pthread_join(tids[i], NULL);
}

static void *sumRow(param *arg) {
    double localSum = 0;
    for (int j = 0; j < 100; j++)
        localSum += a[arg->index][j];
    sum += localSum;
    return NULL;
}
```

07-palim\_handout



## Semaphore

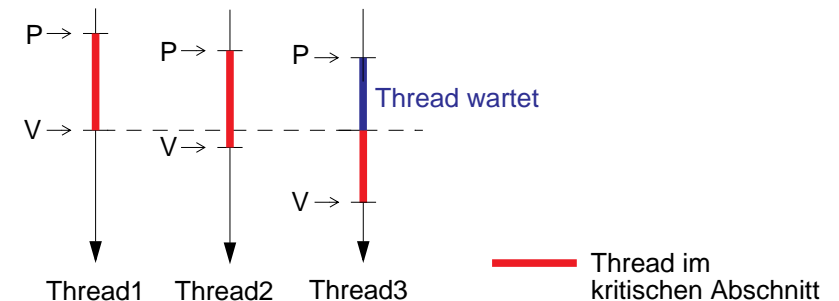
- Zur Koordination von Threads können Semaphore verwendet werden
- UNIX stellt zur Koordination von Prozessen komplexe Semaphor-Operationen zur Verfügung
  - Implementierung durch den Systemkern
  - komplexe Datenstrukturen, aufwändig zu programmieren
  - für die Koordination von Threads viel zu teuer
- Stattdessen Verwendung einer eigenen Semaphorimplementierung mit atomaren P()- und V()-Operationen

07-palim\_handout



## Limitierung von Ressourcen

- Verwendung eines zählenden Semaphors
- Beispiel: Nur zwei aktive Threads gleichzeitig gewünscht
  - Initialisierung des Semaphors mit 2

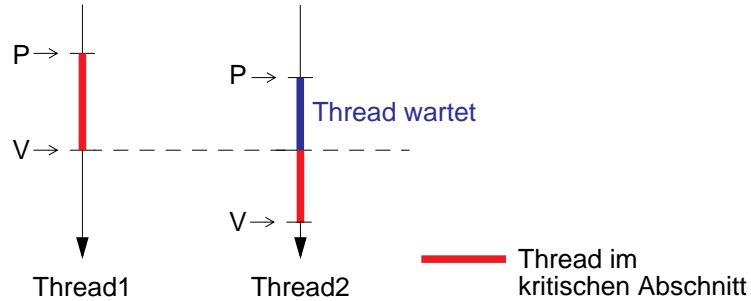


07-palim\_handout



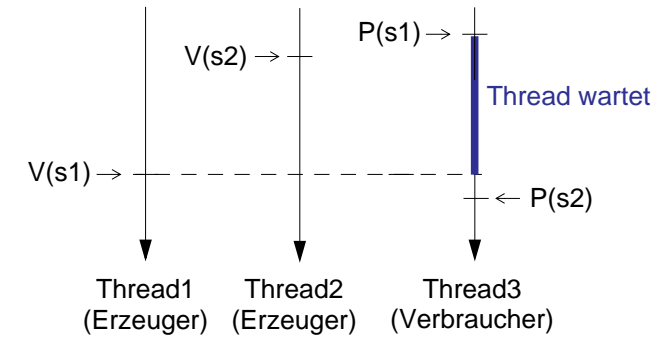
## Gegenseitiger Ausschluss

- Spezialfall des zählenden Semaphors: Binärer Semaphor
  - Initialisierung des Semaphors mit 1
- Beispiel: Schreibender Zugriff auf ein gemeinsames Datum



## Signalisierung

- Benachrichtigung eines anderen Threads über ein Ereignis
- Beispiel: Bereitstehen von Zwischenergebnissen



## SP-Semaphoren-Modul

- Semaphore erzeugen

```
SEM* semCreate (int initVal)
```
- P/V-Operationen

```
void P (SEM *sem)
void V (SEM *sem)
```
- Semaphor zerstören

```
void semDestroy (SEM *sem)
```
- Semaphoren-Modul und zugehörige Headerdatei befinden sich im pub-Verzeichnis.

## Agenda

- 7.1 Hinweise zur Evaluation
- 7.2 (Mini-)Klausurvorbereitung
- 7.3 Threads
- 7.4 Schnittstelle
- 7.5 Koordinierung
- 7.6 Aufgabe 6: palim
- 7.7 Gelerntes Anwenden

## Allgemeines

- Mehrfädige, rekursive Suche nach einer Zeichenkette in Verzeichnisbäumen
- Zur Verwendung der Pthreads-Bibliothek ist die gcc-Option **-pthread** notwendig
- Partnerabgabe!
- palim nicht ausführen auf:
  - SunRay-Servern (faii0sr0, faii0sr1, faii0sr2, faii05, faii01)
  - SunRay-Thin-Clients
  - faii09er und faii01er Rechnern



## Aufgaben der Threads

- Haupt-Thread (**main()**)
  - Startet für jeden als Parameter übergebenen Verzeichnisbaum einen eigenen crawl-Thread
  - Aktualisiert die Statusausgabe kontinuierlich, bis die Suche abgeschlossen ist, und terminiert anschließend den Prozess
- *crawl*-Thread
  - Durchsucht einen Verzeichnisbaum rekursiv
  - Wartet falls maximale Anzahl an **grep**-Threads erreicht
  - Startet für jede gefundene reguläre Datei einen eigenen grep-Thread
- *grep*-Thread
  - Öffnet reguläre Datei und zählt u. a. die Anzahl der Zeilen, die die Suchzeichenkette enthalten



## Agenda

- 7.1 Hinweise zur Evaluation
- 7.2 (Mini-)Klausurvorbereitung
- 7.3 Threads
- 7.4 Schnittstelle
- 7.5 Koordinierung
- 7.6 Aufgabe 6: palim
- 7.7 Gelerntes Anwenden



## Aktive Mitarbeit!

### „Aufgabenstellung“

- Beispiel von Folie 7–18 korrekt synchronisieren

