

# Übungen zu Systemprogrammierung 2

## Ü6 – Mehrfädige Programme

Sommersemester 2019

Simon Ruderich, Dustin Nguyen, Christian Eichler, Jürgen Kleinöder

Lehrstuhl für Informatik 4  
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Verteilte Systeme  
und Betriebssysteme



## Agenda



- 6.1 Organisation
- 6.2 Thread-Pool-Entwurfsmuster
- 6.3 Zusammenspiel von BS-Konzepten
- 6.4 Aufgabe 5: mother

## Agenda



- 6.1 Organisation
- 6.2 Thread-Pool-Entwurfsmuster
- 6.3 Zusammenspiel von BS-Konzepten
- 6.4 Aufgabe 5: mother

## Hinweise zur Evaluation



- Übungsevaluation (weiße TANs)
  - Bei Kommentaren, die sich auf einen bestimmten Übungsleiter beziehen, bitte dessen Namen **in jedem Feld** voranstellen
  - Kommentarfelder werden in der Auswertung durcheinandergewürfelt
  - Bitte hier auch die Rechnerübungen berücksichtigen
- Keine Vorlesungsevaluation im Sommersemester 2019

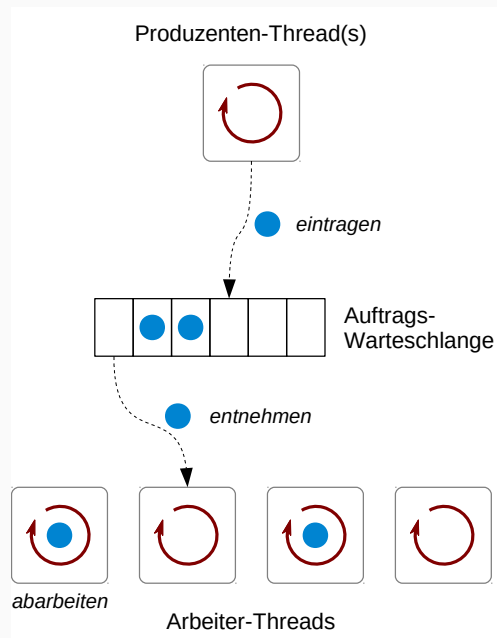


- In den letzten beiden Semesterwochen: Klausurvorbereitung in der Tafelübung zur Vorbereitung auf die Klausur
- Wir erarbeiten die Klausur Februar 2019 (WiSe 2018) gemeinsam
  - Klausur ist auf Übungsseite (SP2 ⇒ Übung ⇒ Folien) verlinkt
  - Eine Vorbereitung der Klausur im Vorfeld der Tafelübung wird erwartet
- **Voraussichtlicher** Klausurtermin: 30.07.2019

4



- Feste Menge von Arbeiter-Threads:
  - laufen endlos
  - erhalten Aufträge zur Abarbeitung
- Verteilen der Aufträge mittels zentraler, synchronisierter Warteschlange (z. B. Ringpuffer)
- Vorteil: kein ständiges Erzeugen + Zerstören von Threads für Aufträge



6



- 6.1 Organisation
- 6.2 Thread-Pool-Entwurfsmuster
- 6.3 Zusammenspiel von BS-Konzepten
- 6.4 Aufgabe 5: mother



- 6.1 Organisation
- 6.2 Thread-Pool-Entwurfsmuster
- 6.3 Zusammenspiel von BS-Konzepten
- 6.4 Aufgabe 5: mother



- Signale können ...
  - an einen Thread gerichtet sein:
    - Synchron auftretende Signale (z. B. SIGSEGV, SIGPIPE)
    - Signale, die mit pthread\_kill(3) geschickt wurden
  - an einen Prozess gerichtet sein:
    - Alle anderen Signale (z. B. mit kill(2) erzeugte Signale)
- Signalbehandlung gilt prozessweit:
  - An Thread gerichtete Signale werden von diesem bearbeitet
  - An Prozess gerichtete Signale werden von beliebigem Thread bearbeitet
- Signalmaske ist Thread-lokal:
  - Statt sigprocmask(2) muss pthread\_sigmask(3) benutzt werden:
    - Verhalten von sigprocmask(2) in mehrfädigem Prozess ist undefiniert
  - Von einem Thread blockierte Signale, die ...
    - an diesen gerichtet sind, werden verzögert
    - an dessen Prozess gerichtet sind, werden von einem anderen Thread bearbeitet

8



- Verwendung von fork(2) in mehrfädigen Prozessen grundsätzlich problematisch:
  - Bei fork(2) wird nur der aufrufende Thread geklont; alle anderen Threads sind im Kind nicht mehr vorhanden
  - Gelockte Mutexe bleiben gelockt und können nicht freigegeben oder zerstört werden
  - Kind kann inkonsistenten Zustand kopieren
- Unproblematisch, wenn geforkt wird, um exec(3) auszuführen:
  - Beim Aufruf von exec(3) ...
    - werden alle Mutexe und Bedingungsvariablen zerstört
    - verschwinden alle Threads – bis auf den aufrufenden

9



- Erinnerung: offene Dateien/Sockets/...
  - werden bei fork(2) an den neu erzeugten Kindprozess vererbt
  - bleiben bei exec(3) im neu geladenen Programm erhalten
- Dieses Verhalten ist unter Umständen unerwünscht!
  - Beispiel: Server will seine offenen Sockets nicht an ein von ihm gestartetes Programm weiterreichen
- Abhilfe: Close-on-exec-Flag für Dateideskriptoren
  - Dateideskriptoren, bei denen dieses Flag gesetzt ist, werden beim Aufruf von exec(3) automatisch geschlossen
  - Sofortiges Setzen beim Öffnen einer Datei:
 

```
int fd = open("index.html", O_RDONLY | O_CLOEXEC);
FILE *fp = fdopen(fd, "r");
```

10



- Close-on-exec-Flag für Dateideskriptoren, Fortsetzung
  - Alternativ: Setzen mit fcntl(2):
 

```
int flags = fcntl(fd, F_GETFD, 0); // Alte Flags holen
fcntl(fd, F_SETFD, flags | FD_CLOEXEC); // Neue Flags setzen
```
  - dup(2), dup2(2) setzen Close-on-exec beim neuen Dateideskriptor zurück
  - Bei Verzeichnissen: opendir(3) setzt Close-on-exec automatisch

11



6.1 Organisation

6.2 Thread-Pool-Entwurfsmuster

6.3 Zusammenspiel von BS-Konzepten

6.4 Aufgabe 5: mother



- Stark aufgebohrte Version der `sister`
- Neue Features:
  - Thread-Pool statt `fork(2)`
  - Auflistung von Verzeichnisinhalten (alphabetisch sortiert)
  - Ausführen von Perl-Skripten
- Ziel der Aufgabe:
  - Wiederholung etlicher in den SP-Übungen gelernter Konzepte