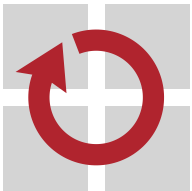
Übungen zu Systemprogrammierung 2

Ü6 – Mehrfädige Programme

Sommersemester 2020

Dustin Nguyen, Jonas Rabenstein, Christian Eichler, Jürgen Kleinöder

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



Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Agenda

6.1 Werbeblock: ICPC

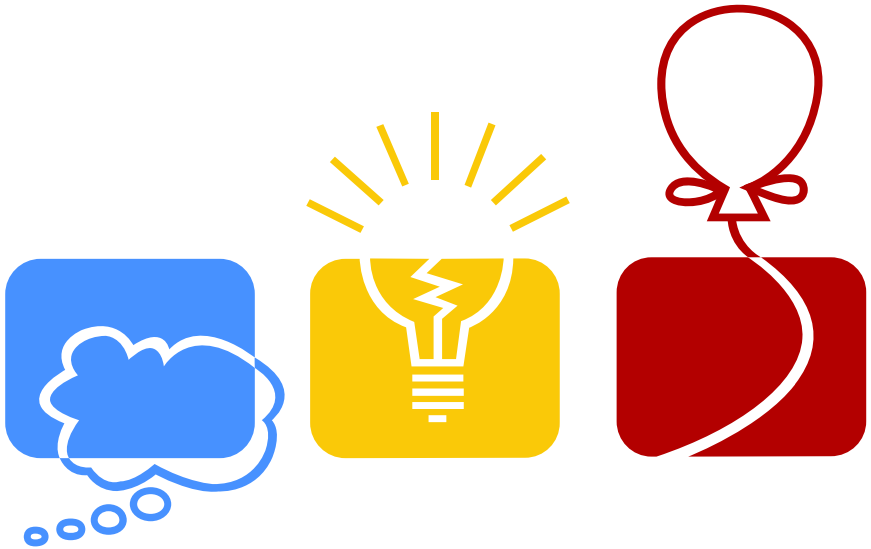
6.2 Organisation

6.3 Thread-Pool-Entwurfsmuster

6.4 Zusammenspiel von BS-Konzepten

6.5 Aufgabe 5: `mother`

ICPC – Programmierwettbewerb, auch an der FAU



Think – Create – Solve

ICPC – Was ist das?

- International Collegiate Programming Contest – veranstaltet von der ACM
- dabei sollen Teams aus 3 Studenten innerhalb von 5 Stunden ~10 algorithmische/mathematische Programmieraufgaben lösen
- Problem: nur 1 Computer steht zur Verfügung, aber kein Internet ☹
- analoge Hilfsmittel (Bücher) dürfen mitgenommen werden ☺
- dreistufiger Wettbewerb mit Local Contest in Erlangen, Regional Contest (dieses Jahr in Eindhoven) und World Finals (letztes Mal in Portugal)

Local Contest
(Erlangen)

Regional Contest
(Bath, UK)

World Finals
(Orlando)

ICPC an der FAU

- am Samstag, 25. Januar 2020 findet wieder der FAU Wintercontest statt
- von 11 bis 16 Uhr im Informatikhochhaus
- teilnehmen darf jede/r Student/in der FAU, Fachrichtung egal!
- es wird jeweils zu dritt programmiert (Einzelanmeldung möglich)
- es wird außerdem eine Practice Session für alle Neulinge stattfinden, bei der (einfache) typische Probleme gezeigt und erklärt werden
- mehr Infos/Anmeldung: https://icpc.cs.fau.de

Wichtig: Anmeldung

Zur Teilnahme am Wettbewerb ist eine Anmeldung unter https://icpc.cs.fau.de unbedingt erforderlich. Deadline: 23.01.2020.

Was bringt mir das Ganze?

- Spaß und Pizza ☺
- Programmiererfahrung und Vertiefung gelernter Algorithmen
- jeder Teilnehmer erhält eine Urkunde
- im Winter sind die Probleme etwas einfacher, perfekt für Neueinsteiger
- beim Local Contest im Sommer werden die Teams bestimmt, die zum NWERC fahren dürfen, um unsere Uni zu vertreten
- die ganz Guten dürfen zu den World Finals (z.B. 2019 Porto in Portugal) fahren, um unsere Uni zu vertreten

Wo kann ich trainieren?

- Training an der FAU:
  - FAU Online Judge: https://icpc.cs.fau.de/oj
  - einmaliges Freischalten mittels EST-Account notwendig
  - mit Problemen z.B. vom Winter 2019
  - Hilfestellung über das Online-Judge-Frontend bzw. IRC-Channel #hallowelt im IRCnet (z.B. irc.uni-erlangen.de)

- Online-Plattformen zum Trainieren von Programmieraufgaben:
  - Codeforces: http://codeforces.com
  - SPOJ: http://spoj.pl
  - UVa: http://uva.onlinejudge.org

ICPC Wintercontest an der FAU

Agenda

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 2020

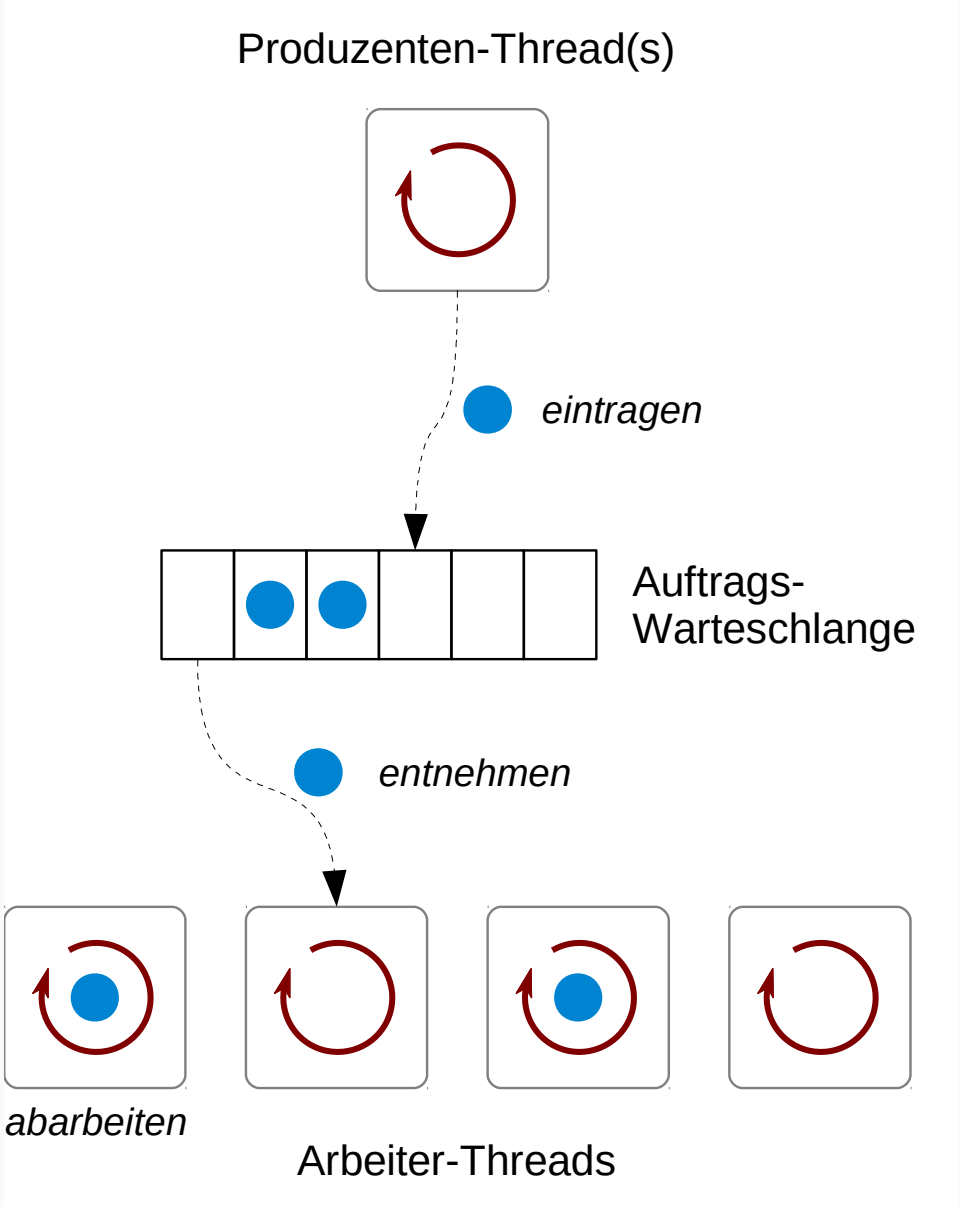
Klausurvorbereitung

- In den letzten beiden Semesterwochen: Klausurvorbereitung in der Tafelübung zur Vorbereitung auf die Klausur
- Wir erarbeiten die Klausur `Juli 2019` (SoSe 2019) gemeinsam
  - Klausur ist auf Übungsseite (SP2 $\Rightarrow$ Übung $\Rightarrow$ Folien) verlinkt
  - Eine Vorbereitung der Klausur im Vorfeld der Tafelübung wird erwartet
- **Voraussichtlicher** Klausurtermin: 19.02.2020

Agenda

Thread-Pool-Entwurfsmuster

- 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



Agenda

Threads und UNIX-Signale

- 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
  - Neue Threads „erben“ Signalmaske des Erzeugers
  - Von einem Thread blockierte Signale, die …
    - an diesen gerichtet sind, werden verzögert
    - an dessen Prozess gerichtet sind, werden von einem anderen Thread bearbeitet

Threads und Prozesse

- 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

Prozesse und offene Dateien

- 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");
```

Prozesse und offene Dateien, cont.

- *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

Agenda

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