

Übungen zu Systemprogrammierung 2

Ü6 – Mehrfädige Programme

Wintersemester 2019/20

Simon Ruderich, Dustin Nguyen, 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



6.1 Werbeblock: ICPC

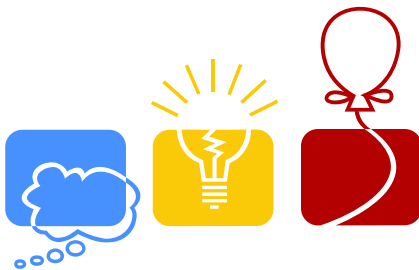
6.2 Organisation

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>

Programmierwettbewerb an der FAU 25.01.2020 – 11h bis 16h



Anmeldung bis
Donnerstag, 23. Januar

*Did you know that...?
Edition*



... cats sometimes chew on cables.
Let's hope the **one computer** you have still works.



... cats are solitary creatures.
But your team can have up to **3 students**.



... you should not feed **pizza** to cats.
You should eat it yourself.



... cats can sleep on your keyboard for up to 19 hours a day.
You have **5 hours** to solve the problems.



... the cat ate your homework.
No worries, we have extra **10+ problems** for you!

Infos und Anmeldung:

<https://icpc.cs.fau.de/>



and end 1999

```
1 // 1999
2 // 1999
3 // 1999
4 // 1999
5 // 1999
6 // 1999
7 // 1999
8 // 1999
9 // 1999
10 // 1999
11 // 1999
12 // 1999
13 // 1999
14 // 1999
15 // 1999
16 // 1999
17 // 1999
18 // 1999
19 // 1999
20 // 1999
21 // 1999
22 // 1999
23 // 1999
24 // 1999
25 // 1999
26 // 1999
27 // 1999
28 // 1999
29 // 1999
30 // 1999
31 // 1999
32 // 1999
33 // 1999
34 // 1999
35 // 1999
36 // 1999
37 // 1999
38 // 1999
39 // 1999
40 // 1999
41 // 1999
42 // 1999
43 // 1999
44 // 1999
45 // 1999
46 // 1999
47 // 1999
48 // 1999
49 // 1999
50 // 1999
51 // 1999
52 // 1999
53 // 1999
54 // 1999
55 // 1999
56 // 1999
57 // 1999
58 // 1999
59 // 1999
60 // 1999
61 // 1999
62 // 1999
63 // 1999
64 // 1999
65 // 1999
66 // 1999
67 // 1999
68 // 1999
69 // 1999
70 // 1999
71 // 1999
72 // 1999
73 // 1999
74 // 1999
75 // 1999
76 // 1999
77 // 1999
78 // 1999
79 // 1999
80 // 1999
81 // 1999
82 // 1999
83 // 1999
84 // 1999
85 // 1999
86 // 1999
87 // 1999
88 // 1999
89 // 1999
90 // 1999
91 // 1999
92 // 1999
93 // 1999
94 // 1999
95 // 1999
96 // 1999
97 // 1999
98 // 1999
99 // 1999
100 // 1999
```


6.1 Werbeblock: ICPC

6.2 Organisation

6.3 Thread-Pool-Entwurfsmuster

6.4 Zusammenspiel von BS-Konzepten

6.5 Aufgabe 5: mother

- 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
- Die Vorlesung bitte ebenfalls evaluieren (grüne TANS)
- Vorlesungsevaluation: „Dozent hat Vorlesung zu ... selbst gehalten“
 - Dozenten sind Wolfgang Schröder-Preikschat und Jürgen Kleinöder
 - Technisch bedingt wird in der Evaluation nur Wolfgang Schröder-Preikschat als Dozent genannt
 - Bitte beide Dozenten bei der Beantwortung der Frage berücksichtigen

- 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

6.1 Werbeblock: ICPC

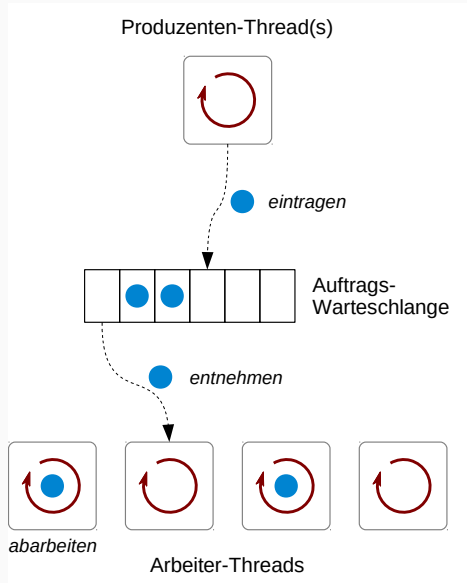
6.2 Organisation

6.3 Thread-Pool-Entwurfsmuster

6.4 Zusammenspiel von BS-Konzepten

6.5 Aufgabe 5: mother

- 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





6.1 Werbeblock: ICPC

6.2 Organisation

6.3 Thread-Pool-Entwurfsmuster

6.4 Zusammenspiel von BS-Konzepten

6.5 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
 - 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



- 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



- 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");
```



■ *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

6.1 Werbeblock: ICPC

6.2 Organisation

6.3 Thread-Pool-Entwurfsmuster

6.4 Zusammenspiel von BS-Konzepten

6.5 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