Übungen zu Systemprogrammierung 2 (SP2)

Ü6 – Mehrfädige Programme

Christoph Erhardt, Jens Schedel, Jürgen Kleinöder

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität
Erlangen-Nürnberg

SS 13 – 24. bis 28. Juni 2013

http://www4.cs.fau.de/Lehre/SS13/V_SP2

Agenda

26-Multithreading_handout

Agenda

26-Multithreading_handout

Hinweise zur Evaluation

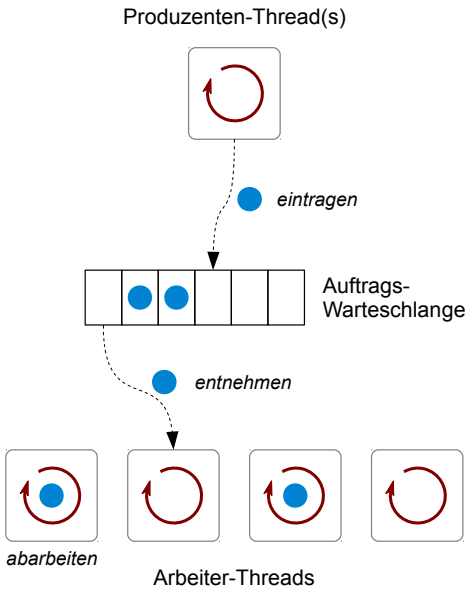
- Bei Kommentaren, die sich auf einen bestimmten Übungsleiter beziehen, bitte dessen Namen **in jedem Feld** voranstellen
  - Kommentarfelder werden in der Auswertung durcheinandergewürfelt
- Frage „eigener Aufwand zur Vor- und Nachbereitung“:
  - Bitte nach Vorlesung und Übung auftrennen
  - Übung: 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)

26-Multithreading_handout

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:
  - Von einem Thread blockierte Signale, die ...
    - an diesen gerichtet sind, werden verzögert
    - an dessen Prozess gerichtet sind, werden von anderem Thread bearbeitet
  - Statt `sigprocmask(2)` muss `pthread_sigmask(3)` benutzt werden:
    - Verhalten von `sigprocmask(2)` in mehrfädigem Prozess ist undefiniert

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(2)` auszuführen:
  - Beim Aufruf von `exec(2)` ...
    - 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(2)` 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(2)` automatisch geschlossen
  - 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)` setzt *Close-on-exec* bei dupliziertem Dateideskriptor zurück
  - Bei Verzeichnissen: `opendir(3)` setzt *Close-on-exec* automatisch

Agenda

Aufgabe 5: `mother`

Modular **Th**readed Serv**er**

- Stark aufgebohrte Version der `sister`
- Neue Features:
  - Thread-Pool statt `fork(2)`
  - Auflistung von Verzeichnisinhalten (alphabetisch sortiert)
  - Ausführen von Perl-Skripten
- (Leider) Anpassung der Schnittstelle des Anfrage-Moduls
  - `void handleRequest(FILE *fromClient, FILE *toClient)`: sieht einen `FILE*` für jede Kommunikationsrichtung vor
  - Die Funktion `void handleInternalError(int client)` wurde hinzugefügt um im Verbindungs-Modul auftretende Fehler an Clients übermitteln zu können
- Ziel der Aufgabe:
  - Wiederholung etlicher in den SP-Übungen gelernter Konzepte