

## Aufgabenblatt 7

### Musterlösung

## Vorlesung Verteilte Systeme

### Wintersemester 2025/26

#### Aufgabe 1: Prozeßmigration

Es gibt etliche Zustandskomponenten eines Prozesses, die im BS gespeichert sind, z.B.

- die Tabelle offener Dateien (der Prozeß könnte nach der Migration nicht mehr auf diese zugreifen)
- die Signalmaske (der Prozeß würde ggf. bestimmte Signale nicht mehr richtig behandeln)
- gesetzte Alarm-Timer (der Prozeß würde das Alarm-Signal nicht mehr bekommen)
- etc.

Man müßte also zumindest diese Information des BS-Kerns mit übertragen. Alternativ kann man auch wieder Systemaufrufe an den Ursprungsknoten zurücksenden (und Signale etc. von diesem an den neuen Knoten weiterleiten).

Eine Migration in der beschriebenen Art ist natürlich prinzipiell nur zwischen Rechnern möglich, die exakt dieselbe CPU-Architektur und dieselbe BS-Version benutzen.

#### Aufgabe 2: Policies dynamischer Lastverteilungs-Systeme

- a) Die **Transfer Policy** beschreibt, wie ein Computer entscheidet, ob (bzw. wann) Last verschoben werden soll. Typischerweise wird dies über Schwellwerte für die Last bestimmt. Wenn ein Computer eine Last hat, die größer als diese Schwelle ist, gibt er Tasks (Prozesse oder Jobs) ab. Umgekehrt kann ein Knoten, dessen Last kleiner als eine Schwelle ist, Tasks von anderen Knoten anfordern.

Sobald ein Knoten als Sender festgelegt ist, muss er wählen, welcher Task zu einem anderen Knoten gesendet werden soll (**Selection Policy**). Dabei muß sichergestellt werden, dass der Overhead, der durch das Verschieben der Task verursacht wird, nicht den Nutzen des Verschiebens vernichtet. Eine Methode wäre, neue Tasks zu senden, die gerade in die Warteschlange gekommen sind. Dies stellt sicher, dass ein Task, der gesendet werden muss, nicht schon durchgeführt worden ist, d.h. keinen Zustand hat.

Die **Location Policy** beschreibt, wie man einen Empfänger-Knoten für die zu verschiebenden Tasks ermittelt (bzw. einen Sender-Knoten, wenn der Empfänger den Lastausgleich initiiert). Eine sehr einfache Möglichkeit besteht darin, einen zufälligen Knoten zu nehmen. Besser wäre es, mehrere Knoten nach ihrer Last zu befragen und den am wenigsten belasteten als Empfänger zu wählen.

Die **Information Policy** beschreibt, wie/wann die Last-Information ermittelt und verteilt wird. Die Last kann z.B. regelmäßig ermittelt werden, oder ereignisgesteuert (wenn sich die Lastsituation ändert, z.B. bei Erzeugung oder Beendigung von Tasks). Die Lastinformation kann global an alle Knoten oder auch nur lokal an die nächsten Nachbarn verteilt werden.

- b) Empfänger-initiierte Verfahren können eine Verlangsamung des Systems (mit niedriger Last) ergeben. Die Leistung des Systems sinkt.

Wenn das System nie eine hohe Last hat, dann beschließen normalerweise alle Knoten durch die Transfer Policy, ein Empfänger zu sein. Wenn wir empfänger-initiierte Verfahren verwenden, wird jeder Knoten ständig Anfragen nach Tasks versenden, und es gibt keine Sender, um sie zu erfüllen. Diese Taskanfragen füllen das Netz mit nicht

notwendigem Verkehr und können tatsächliche Taskübertragungen verlangsamen und damit die Leistung verringern.

Durch die Latenz zwischen Absenden einer Anfrage und Eintreffen einer Task kann es zudem zu Leerlaufzeiten auf Knoten kommen, obwohl vielleicht insgesamt genügend Tasks vorhanden wären, um alle Knoten zu beschäftigen.

### Aufgabe 3: Uhrendrift

- a) Ein einfaches Beispiel ist das verteilte Kompilieren eines Programms mittels `make`. Make wird über ein sogenanntes Makefile gesteuert, in dem Abhängigkeiten z.B. zwischen Quell- und Objektdateien sowie Regeln, die angeben, wie Zielformate aus Quellformaten erzeugt werden können, spezifiziert sind. Aus Optimierungsgründen wird eine solche Regel nur angewendet, wenn sich seit der letzten Anwendung die Quellformate verändert hat, damit eben nur wirklich notwendige Kompilierungsvorgänge stattfinden. Dabei orientiert sich Make an den im Dateisystem abgelegten Zeitstempeln der Dateien, die die letzte Änderung angeben. Weichen nun die Uhren auf den einzelnen beteiligten Rechnern voneinander ab, so kann es zu inkonsistenten Zuständen kommen, z.B. dazu, dass eine Quellformate ein älteres Datum als die zugehörige Objektformate aufweist, obwohl der Quellcode nach dem letzten Übersetzungsvorgang verändert wurde. Fehlerhafterweise würde dann die erneute Kompilierung dieser Formate ausbleiben.
- b) Mit den zusätzlich eingefügten Schaltsekunden wird erreicht, dass manche Tage statt 86400 genau 86401 Sekunden dauern. Notwendig ist dies, da eine Erddrehung eben nicht exakt  $24 * 60 * 60$  Sekunden dauert, sondern einen kleinen Sekundenbruchteil länger. Schaltsekunden haben also nichts mit einer evtl. Ungenauigkeit der Atomuhr zu tun, sondern dienen lediglich einer Anpassung an den Kalender.

### Aufgabe 4: Uhrensynchronisation

Es gibt mehrere Möglichkeiten zur Synchronisierung verteilter Uhren. Eine von diesen ist Cristians Protokoll. Ein Client-Prozess  $P$  fordert die Zeit in einer Nachricht  $m_r$  von einem Zeit-Server  $S$  an, und empfängt den Zeitwert  $t$  in einer Nachricht  $m_t$  von  $S$ . Das Problem ist, daß wegen der Signallaufzeit von  $m_t$  die Zeit  $t$  bis zur Ankunft bei  $P$  schon wieder veraltet ist.  $P$  mißt daher die Round-Trip-Zeit  $T_{round}$  vom Versenden von  $m_r$  bis zum Empfang von  $m_t$  und setzt seine Uhr auf  $t + T_{round}/2$  ( $T_{round}/2$  ist die geschätzte Laufzeit der Nachricht  $m_t$ ).

Die Genauigkeit des Verfahrens wird eingeschränkt durch:

- die Annahme, daß die Laufzeiten von  $m_r$  und  $m_t$  gleich sind (systematischer Fehler).
- die Streuungen in den Nachrichtenlaufzeiten (statistischer Fehler).

In einem LAN entstehen Streuungen durch den Wettbewerb um das Medium (Kollisionen, Pufferzeiten in Switches) und durch die Verarbeitung der Nachrichten im Betriebssystem von  $P$  und  $S$ . Für ein LAN ist die Genauigkeit vermutlich innerhalb  $1ms$ . Im Internet entstehen zusätzliche Ungenauigkeiten, weil die Nachrichten in den Routern verzögert werden. Für WANs ist die Genauigkeit vermutlich innerhalb  $5 - 10ms$ .

Auch mit Hilfe von GPS ist eine exakte Synchronisierung von Uhren nicht möglich. Wegen der bekannten Position von Satellit und GPS-Empfänger kann die Nachrichtenlaufzeit zwar sehr genau berechnet werden, schwankt aber z.B. aufgrund von Wetterbedingungen. Zudem sind auch die Positionen mit Fehlern behaftet.