Universität Heidelberg
Sommersemester 2019

Lehrstuhl für Parallele und Verteilte Systeme (PVS)
Diego Costa, Artur Andrzejak

9. Übung zur Vorlesung „Betriebssysteme und Netzwerke“ (IBN)

Abgabedatum: 25.06.2019, 11:00 Uhr

Aufgabe 1 (4 Punkte)

Scheduling war in der Anfangszeit der Computer nicht so unsichtbar wie bei PCs oder bei heutigen interaktiven Systemen am Laptop oder auf dem Smartphone. Im Universitätsgesetz 1977 des Landes Baden-Württemberg etwa wurde den Rechenzentren in §31 Absatz 1 explizit die Aufgabe *[der] Verwaltung und [des] Betriebs der Datenverarbeitungsanlagen [...] einschließlich der Zuteilung von Rechenzeiten* auferlegt. Die Rechenzeit auf dem ersten Mainframe des URZ Heidelberg (mit 1 MHz Taktfrequenz) war Luxus: Rechnen für Studienzwecke und Forschung für Uni-Mitglieder sowie uni-interne Verwaltung wurde zwar unentgeldlich angeboten. Wollten aber Externe rechnen, erhob man 120 DM/Stunde für die Betriebskosten; beim Einsatz für kommerziellen Zwecke sogar 600 DM/Stunde (das entspricht gemäß Verbraucherpreisindex des Statistischen Bundesamts einem heutigen Preis von rund 200€ bzw. 1.000€).

Als 1969 das URZ der Uni Heidelberg (damals am Friedrich-Ebert-Platz) eröffnet wurde, beschrieb eine *Rechenordnung*, wie die Rahmenbedingungen für das Scheduling aussahen. Sie unterscheidet *Kurzläufe (bis max. 5 Min.), Normalläufe (bis max. 15 Min.)* und *Langläufe (über 15 Min.)*. Weiter hieß es, dass Kurzläufe und Normalläufe zwischen 8 und 18 Uhr bearbeitet werden - jeweils eine Stunde mit Vorzug Kurzläufe, dann eine Stunde mit Vorzug Normalläufe usw. *Langläufe werden nach 18 Uhr abgearbeitet. [...] Langläufe über 60 Minuten sind einen Tag zuvor beim Chefoperateur anzumelden.* Außerdem wurde erklärt: *Die Arbeiten sind unter Beibehaltung der Reihenfolge in die dafür bestimmten Kästen einzulegen - nach Gruppen und Betriebssystem getrennt.*

a) Welcher Typ von Umgebung (Unterscheidung nach Vorlesung 16) liegt hier vor? Wer entschied wohl, wie lange ein Programmlauf dauern würde, und wie? Wie würden Sie das obengenannte Scheduling beschreiben? Die eingesetzte IBM 360-44 hatte einen Interval Timer, der für einen Prozess in regelmäßigen Abständen runterzählte und einen Interrupt auslöste. Spekulieren Sie, wozu dieser beim Prozessmanagement und Scheduling eingesetzt worden sein könnte.

Im Jahr 1975 hatte das URZ gerade eine neue Anlage (IBM 370-168 mit 12,5 MHz) in Betrieb genommen. Ab April war den Nutzern auch Time-Sharing über 34 Terminals (Datensichtgeräte) möglich. Wenn das Timersharing-System TSO im Betriebssystem MVS aktiv war, konnte man im Dialog Programme erstellen und dann an den Batch übergeben, ohne dabei eine Lochkarte anzufassen. Im Dialog war es darüber hinaus möglich, Programme unmittelbar ablaufen zu lassen, sie am Bildschirm auszutesten und dabei auch online korrigierend in den Programmablauf einzugreifen. Jedes Batch-Programm und jede Dialogsitzung liefen in einem separaten virtuellen Adressraum von

jeweils 16 MB ab, und all diese virtuellen Speicheranforderungen mussten auf den realen Hauptspeicher von 2 MB abgebildet werden.

Um den neuen Gegebenheiten Rechnung zu tragen, wurde die Rechenordnung um URZ angepasst. Alle Jobs wurden nach ihrer Charakteristik (I/O-intensiv / CPU-intensiv / ausgewogen), der maximalen CPU-Zeit (<=10sec, <=15 min, <=60 min oder >60 min) und dem Speicherbedarf (<=256KB, <= 2MB, >2MB) in Klassen eingeteilt.

b) Wie lassen sich Batch-Betrieb und Time-Sharing durch Scheduling vereinbaren? Wie kann Scheduling helfen, den Overhead durch Paging zu minimieren? Welche Rolle spielt Spooling im Kontext von Scheduling? Und wieder: Wer hat wohl wie die Einteilung in die Klassen nach den drei Kriterien vorgenommen?

Ein Disclaimer aus der Benutzungsordnung 1969 gilt wohl heute mehr denn je am URZ: § 6, Absatz 3, *Das Rechenzentrum übernimmt keine Verantwortung für die Richtigkeit der Programmierung und der Ergebnisse.*

Aufgabe 2 (2 Punkte)

Fünf Stapelverarbeitungsjobs von A bis E kommen fast zur gleichen Zeit in einem Rechenzentrum an. Ihre geschätzten Laufzeiten sind 10, 6, 2, 4 und 8 Minuten. Ihre (extern bestimmten) Prioritäten sind 3, 5, 2, 1 und 4, wobei 5 die höchste Priorität darstellt. Bestimmen Sie für jede der folgenden Scheduling-Strategien die durchschnittliche Prozessdurchlaufzeit (Vorlesung 16). Vernachlässigen Sie dabei den Aufwand des Prozesswechsels.

1. Round Robin
2. Prioritätsscheduling
3. First Come First Served (in der Ankunftsreihenfolge A bis E)
4. Shortest Job First

Hinweis Nehmen Sie für 1. an, dass das System multiprogrammierbar ist und dass die CPU unter den Jobs gleichmäßig aufgeteilt wird. Für 2. bis 4. nehmen Sie an, dass immer nur jeweils ein einziger Job bis zum Ende ausgeführt wird. Alle Jobs sind vollständig CPU-intensiv, d.h. es gibt keine Unterbrechungen durch Ein-/Ausgabeoperationen.

Aufgabe 3 (6 Punkte)

Implementieren Sie ein Programm, dass Prozessschedulingalgorithmen simulieren kann. Nehmen Sie an, dass alle Prozesse 100% CPU-gebunden sind (kein I/O). Als Parameter erwartet es ein Quantum (für die Algorithmen mit Unterbrechbarkeit) und als Input eine Folge tatsächlicher Laufzeiten von Prozessen, in der Reihenfolge ihres „Eintreffens" beim Scheduler. Nehmen Sie der Einfachheit halber an, dass alle Prozesse bereits im Wartezustand sind, wenn die Simulation beginnt. Rechnen Sie

in ganzzahligen, unspezifischen Zeiteinheiten. Nehmen Sie weiterhin an, dass der Scheduler die tatsächliche Joblänge im Voraus kennt.

Das Programm soll in die Standardausgabe eine Folge von Tupeln (P_i, t_i) ausgeben, die angibt, in welcher Reihenfolge welcher Prozess wie lange läuft. Dabei bedeutet das Tupel (P_i, t_i), dass Prozess P_i für t_i Zeiteinheiten läuft. Geben Sie außerdem die aufummierte Turn-around-Zeit aus. Implementieren Sie die Strategien FCFS, SJF, SRTF und Round-Robin. Starten Sie das Programm jeweils einmal mit den folgenden Eingabefolgen mit dem Quantom $Q = 5$. Reichen Sie den Quelltext als auch ein Log der Programmausgaben als Lösung ein.

- 5, 3, 12, 100, 1, 2, 3, 4, 5
- 5, 4, 3, 2, 1, 100, 12, 3, 5
- 23, 17, 31, 29, 71, 2, 5, 113

Aufgabe 4 (2 Punkte)

Moderne Linux / UNIX Systeme verwenden einen 48 oder 128 Bit langen Salt-Wert, um das Erraten von Passwörtern zu erschweren. Recherchieren und erläutern Sie in eigenen Worten, wie diese Modifikation der in Vorlesung 17 vorgestellten Passwort-Authentifizierung funktioniert. Warum ist diese Technik wirksam, auch wenn der Salt-Wert im Klartext im gleichen Eintrag wie das entsprechende verschlüsselte Passwort gespeichert wird? Würde auch ein 12 Bit langer Salt-Wert ausreichen (diese Länge wurde auf älteren UNIX-Systemen tatsächlich benutzt)?

Aufgabe 5 (1 Punkt)

Gibt es eine praktisch realisierbare Möglichkeit, die MMU-Hardware so zu nutzen, dass der in der Vorlesung 17 vorgestellte Pufferüberlaufangriff verhindert werden kann? Erklären Sie, warum bzw. warum nicht.

Aufgabe 6 (1 Punkt)

Recherchieren und erläutern Sie in eigenen Worten, wie der *Return-to-libc*-Angriff funktioniert.

Aufgabe 7 (Bonus, 4 Punkte)

Im Artikel *The Linux Scheduler: a Decade of Wasted Cores*[1] untersuchen die Autoren, wie der Linux-Scheduler auf NUMA-Architekturen in der Praxis funktioniert.

[1] http://www.ece.ubc.ca/~sasha/papers/eurosys16-final29.pdf

- Erklären Sie in eigenen Worten, was *load balancing* ist und wie es funktioniert. Nehmen Sie dazu Abschnitt 2.2.1 zu Hilfe.
- Erklären Sie zwei der vier Bugs, die die Autoren gefunden haben (Abschnitt 3), in eigenen Worten.