Universität Heidelberg
Sommersemester 2022

AG Parallele und Verteilte Systeme (PVS)
Artur Andrzejak

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

Abgabedatum: 24.05.2022, 11:00 Uhr

Aufgabe 1 (2 Punkte)

Schauen Sie das Video *The value of teamwork*[1] aus dem Kurs *Learning How to Learn*. Hier werden den beiden Gehirnhälften verschiedene Rollen und Funktionen beim Lernen und Problemlösen zugeordnet.

- Wenn Sie bei einer Aufgabe mit unterschiedlichen heuristischen Algorithmen oder Strategien stets dieselbe Lösungen erhalten, sollten Sie stutzig werden, da dies nicht plausibel ist. Beschreiben Sie die Funktionen, die den beiden Gehirnhälften im Video zugeordnet werden in eigenen Worten. Sie können dabei eine frühere Übungsaufgabe ihrer Wahl als Beispiel nehmen.
- Welchen Vorteil bietet laut Video das Arbeiten im Team beim Lernen und Problemlösen?

Aufgabe 2 (1 Punkt)

Finden Sie heraus, welche Paradigmen der Versionsverwaltung man mit Lock-Modify-Write bzw. Copy-Modify-Merge bezeichnet und beschreiben Sie beide in eigenen Worten. Welches dieser beiden entspricht einem wechselseitigen Ausschluss in der Programmierung? Lässt sich auch der andere Ansatz auf die Arbeitsweise von Programmen übertragen? Wenn ja, wie, bzw. warum nicht?

Aufgabe 3 (2 Punkte)

Fügen Sie in das Programm mit Bezeichnung Algorithmus 1 für die Platzhaltern `???` geeignet folgende Befehle ein: `shmat, shmctl, shmdt, shmget`. Erklären Sie jeden der Befehle kurz.

Kompilieren Sie das Programm und führen es aus. Welches Verhalten können Sie beobachten (ist dies wie erwartet)? Versuchen Sie, falls möglich, Ihre Beobachtungen zu erklären.

Aufgabe 4 (1 Punkt)

Shells wie Bash erlauben die Verwendung von benannten Pipes. Schreiben Sie das Beispiel aus Vorlesung 8 (zählen von *.jpg*-Dateien) so um, dass eine benannte Pipe verwendet wird.

[1] https://www.youtube.com/watch?v=XlN3c3dVqrg

Aufgabe 5 (3 Punkte)

Das Win32-API des Betriebssystems Windows benutzt `Handles` statt der Dateideskriptoren.

a) Recherchieren Sie, welchen `C/C++`-Datentyp die `Handles` haben, und welche Daten sie (wirklich) enthalten.

b) Die Programmierung der Ein-/Ausgabeumlenkung unter Win32 ist etwas komplizierter als unter Posix. Recherchieren Sie und erläutern an einem Code-Beispiel, wie man die Standard*eingabe* in eine Datei (auf der Ebene der Win32-API) umleiten kann.

Aufgabe 6 (2 Punkte)

Betrachten Sie das Übergangsdiagramm aus Vorlesung 8 („Neue Prozesszustände"). Angenommen, das Betriebssystem wird jetzt einen Prozesswechsel vornehmen. Es gebe zu diesem Zeitpunkt sowohl Prozesse im *Ready*-Zustand als auch im *Ready/suspend*-Zustand. Weiterhin habe mindestens ein Prozess, der sich im *Ready/suspend*-Zustand befindet, eine höhere Scheduling-Priorität als jeder der Prozesse, die aktuell im *Ready*-Zustand sind. Zwei entgegengesetzte Scheduling-Strategien wären:

1. Führe möglichst immer nur Prozesse aus, die im *Ready*-Zustand sind, um Swapping zu vermeiden.
2. Ziehe immer Prozesse gemäß ihrer Scheduling-Priorität vor, selbst wenn dadurch Swapping notwendig wird.

a) Was sind Vor- und Nachteile beider Strategien?

b) Schlagen Sie eine alternative Strategie vor, die das Beste aus beidem vereint.

Aufgabe 7 (1 Punkt)

In einem System mit zusammenhängendem Speicher sind folgende Lücken in Folge von Swapping entstanden (geordnet nach aufsteigenden Adressen, und in MB): 10, 4, 20, 18, 7, 9, 12 und 15.

Welche Lücken wählen *First Fit*, *Best Fit* und *Worst Fit* jeweils aus, wenn nacheinander Speichersegmente von 12 MB, 11 MB, 3 MB und 5 MB angefordert werden? Erläutern Sie ihre Lösung.

Aufgabe 8 (2 Punkte)

"Das Studierendenwerk Heidelberg stellt Studierenden in Heidelberg rund 4.800 Wohnheimzimmer in etwa 65 Wohnheimen zur Verfügung."[2]. Sie kennen die Matrikelnummer eines Kommilitonen, und wollen seine vollständige Adresse erfahren (hier ist die vollständige Adresse = Wohnheim-Adresse und Zimmernummer), d.h. Sie übersetzen die Matrikelnummer in die Wohnadresse mit Hilfe eines Verzeichnisses.

[2] http://www.stw.uni-heidelberg.de/wohnen_allgemein

a) In einer Analogie zu Paging, wie würden Sie die Begriffe in {*Matrikelnummer*, *Wohnadresse*, *Verzeichnis*} den Begriffen in {*Seitentabelle*, *Seitennummer*, *Rahmennummer*} zuordnen? Gibt es eine Entsprechung zum *Offset*?

b) Wie würde in dieser Analogie das Verzeichnis aussehen, wenn ein Verfahren ähnlich wie bei einer *direkten* Seitentabelle benutzt wird? Wie viele Einträge hätte das Verzeichnis? Schätzen Sie den Anteil der Einträge, die relevant sind. Hinweis: Überlegen Sie, wie viele verschiedene Matrikelnummern es geben kann, wenn die Stellenzahl der Matrikelnummer unverändert bleibt. Nehmen Sie weiterhin an, dass die Nummer keine Redundanzen wie Prüfziffern enthält.

Aufgabe 9 (1 Punkt)

In einem Paging-System wurde als die Seitengröße 1 KiB gewählt. Geben Sie die Seitennummern und die Offsets für jede der folgenden logischen Adressen an: 2456, 16382, 30000, 4385. Wie würden Sie die Seitennummer und Offset in der Programmiersprache C berechnen? Zwei Zeilen Code reichen als Antwort aus.

Aufgabe 10 (1+3 Punkte)

Betrachten Sie den in der Vorlesung 9 vorgestellten „Simulator der Direkten Seitentabellen in Nim" und lösen Sie folgende Aufgaben:

a. Ändern Sie den Code so ab, dass ein 64-Bit System mit einer Seitengröße von 1 Mebibyte (MiB, d.h. 2^{20} Bytes) simuliert wird. Dabei sollten die Einträge der simulierten Seitentabelle `MMUdata.page_table` jeweils 64 Bits haben.

b. (Bonusaufgabe, 3 Punkte. Diese Aufgabe kann erst nach der Einführung von hierarchischen Seitentabellen gelöst werden.) Ändern Sie das Programm so, dass die in der Vorlesung 10 vorgestellte hierarchische Seitentabelle der IA32-Architektur simuliert wird (siehe Abschnitt „Paging bei IA32-Architektur: Konkrete Details"). Allozieren Sie die „page tables" (d.h. Fragmente der 2. Stufe) nur wenn ein solches Fragment tatsächlich benötigt wird. Statt der Zeiger (pointers) kann die „page directory" die Indizes in eine Liste mit bereits allozierten „page tables" enthalten.

Algorithmus 1 Programm (mit Lücken), das shared memory verwenden soll

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <stdlib.h>
int main(){

    int i, shmID, *shared_mem, count=0, total=0,rnd;
    shmID = ???(IPC_PRIVATE, sizeof(int), IPC_CREAT | 0644);
    shared_mem = (int*)???(shmID,0,0);
    *shared_mem = 0;
    if (fork())
        for (i=0; i<500; i++){
            *shared_mem+=1;
            printf("\n Elternprozess: %i", *shared_mem);
            sleep(2);
        }
    else
        for (i=0; i<500;i++){
            *shared_mem+=1;
            printf("\n Kindprozess: %i", *shared_mem);
            rnd=rand();
            sleep(rnd%3);
        }
    ???(shared_mem);
    ???(shmID, IPC_RMID, 0);
    return 0;

}
```