Grundlagen der Systemsoftware

Modul: InfB-GSS

Veranstaltung: 64-091

Utz Pöhlmann
4poehlma@informatik.uni-hamburg.de
6663579

Louis Kobras
4kobras@informatik.uni-hamburg.de
6658699

Marius Widmann
4widmann@informatik.uni-hamburg.de
6714203

22. Juni 2016

Zettel Nr. 4 (Ausgabe: datum1, Abgabe: 08. Juni 2016)

Aufgabe 1: Speicherverwaltung

a)

Die physikalische Addressierung besitzt 15 bit. Daher gibt es $2^{15} = 32768$ addressierbare Speicherzellen. Das sind 32 KB Arbeitsspeicher, den sich 16 Seiten teilen. Hierdurch ergibt sich eine Seitengröße von 2 KB.

c)

Da die Tabelle nur 16 Seiten ($= 2^4$) hat, werden die Seitennummern nur in den vorderen 4 Bits gespeichert. Der Rest ist Offset ($2^{12} = 4096$ Bit).

Aufgabenteil	**Hex-Wert**	**physikalische Adresse**	**Bemerkung**
i	0x5FE8	0x1FE8	Seite 5 hat den Seitenrahmen 001, also werden die oberen 4 Bit durch diese Zahl als Hexadezimalzahl ersetzt.
ii	0xFEEE	Seitenladefehler (page fault)	Seite 15 liegt nicht im Speicher, da das Present/Absent-Bit nicht gesetzt ist, weshalb ein Seitenladefehler entsteht.
iii	0xA470	0x0470	Seite 10 hat den Seitenrahmen 000, also werden die oberen 4 Bit durch diese Zahl als Hexadezimalzahl ersetzt.
iv	0x0101	0x5101	Seite 0 hat den Seitenrahmen 101, also werden die oberen 4 Bit durch diese Zahl als Hexadezimalzahl ersetzt.

d)

Pro große Seiten

1. Weniger Seiten müssen geladen werden. Die Ladezeiten sind sehr viel größer als die Lesezeiten einer Seite.

Pro kleine Seiten

1. Weniger ungenutzter Speicher, da die Allokation von Speicher zu Prozessen passgenauer ist. Insbesondere relevant bei vielen Prozessen mit eher wenig Speicherbedarf.
2. Bei einer effizienten MMU und schnellen Festplatten kann durch viele kleine Seiten der Hauptspeicher besser genutzt werden, da wenig genutzte Seiten ausgelagert werden können.

e)

Je kleiner die Seiten sind, desto weiter müssen Programme fragmentiert werden, um sie laden zu können. Zwar geht es durchaus, nur Programmfragmente im Arbeitsspeicher zu haben und diese zu bearbeiten, jedoch geht dies mit steigender Fragmentierungsintensität zu Lasten der Leistung und Effizienz.

Die Prozessgröße wird als $p = 4MiB = 4515000B$ definiert.
Die Länge eines Seitentabelleneintrages wird als $L = 8B$ definiert.
Nach [**?**] gilt für die optimale Größe einer Seite:

$$s = \sqrt{2 \cdot p \cdot L} = \lceil(\sqrt{2 \cdot 4515000 \cdot 8})\rceil B = \lceil 8499.41\rceil B \hat{=} \lceil 8.3\rceil KB = 9KB$$

Demnach sollte die Seitengröße also bei 9KB oder, falls nur Größen in Zweierpotenzen zulässig sind, 16KB groß sein.

Aufgabe 2: Seitenersetzungsalgorithmen

a)

i.

t	1	2	3	4	5	6	7	8	9	10	11	12
Angeforderte Seite	1	2	3	4	2	1	2	5	6	2	6	3
Zugriffsart	r	w	w	r	r	r	r	w	r	w	w	r
Seitenalarm	j	j	j	j	j	n	n	j	j	n	n	j
Seiten im Speicher	1	1 2	1 2 3	1 3 4	1 4 2	1 4 2	1 4 2	1 2 5	1 2 6	1 2 6	1 2 6	2 6 3

ii.

t	1	2	3	4	5	6	7	8	9	10	11	12
Angeforderte Seite	1	2	3	4	2	1	2	5	6	2	6	3
Zugriffsart	r	w	w	r	r	r	r	w	r	w	w	r
Seitenalarm	j	j	j	j	n	j	n	j	j	n	n	j
Seiten im Speicher	1	1 2	1 2 3	2 3 4	2 3 4	2 4 1	2 4 1	2 1 5	2 5 6	2 5 6	2 5 6	2 6 3

Aufgabe 3: Synchronisation

a)

```
processWriter():

do:
        wait()
while number of readers active is greater than 0 OR W is equal to 0
P(W)
write()
V(W)

processReader():

do:
        wait()
while W is equal to 0
do:
        wait()
while Mutex is equal 0
P(Mutex)
increment number of active readers
V(Mutex)

read()

do:
        wait()
while Mutex is equal 0
P(Mutex)
increment number of active readers
V(Mutex)
```

b)

Bei dieser Methode werden schreibende Prozesse benachteiligt, da sie nur auf den kritischen Abschnitt zugreifen können, sobald kein schreibender und kein lesender Prozess mehr darauf zugreift.Lesende Prozesse können allerdings schon darauf zugreifen, wenn sich noch ein lesender Prozess im kritischen Abschnitt befindet. So kann es passieren, dass immer lesende Prozesse auf den kritischen Abschnitt zugreifen und die schreibenden unendlich lange warten müssen.

Eine mögliche Lösung wäre, eine Warteschlange einzuführen, sodass die Prozesse anhand ihres Anfragezeitpunktes geordnet werden und nur weiterarbeiten können, sofern sie an der Reihe sind. Sollte also ein schreibender Prozess ankommen, würde verhindert werden, dass neu ankommende (also nicht solche, die schon vor den schreibenden Prozess angekommen sind) lesende Prozesse den kritischen Abschnitt benutzen könnten, bis der schreibende Prozess fertig ist.

Literatur

[1] http://www.matheretter.de/formeln/algebra/zahlenkonverter/

[2] http://web2.0rechner.de

[3] https://de.wikipedia.org/wiki/Not_recently_used

[4] https://de.wikipedia.org/wiki/Seitenfehler

[5] https://de.wikipedia.org/wiki/Speicherseite

[6] https://de.wikipedia.org/wiki/Bin%C3%A4rpr%C3%A4fix

[7] https://de.wikipedia.org/wiki/Semaphor_(Informatik)

[8] https://de.wikipedia.org/wiki/Paging

[9] http://hm.gesser.de/bsws2006/folien/bs2esser054up.pdf

[10] https://vsis-www.informatik.uni-hamburg.de/getDoc.php/lectures/4959/GSS-C01_a.pdf

[11] https://www.fbi.hda.de/~a.schuette/Vorlesungen/Betriebssysteme/Skript/4_Speicherverwaltung/Speicherverwaltung.pdf

[12] http://www.wi.fhflensburg.de/beispiel_riggert0.html

[13] Tanenbaum, Andrew S., *Modern Operating Systems*, 3rd Edition