Betriebs- und Kommunikationssysteme

WS 14/15

Prof. Dr-Ing. Volker Roth

Freie Universität Berlin

8. Übung

Ausgabe	Abgabe
9.12.14	19.12.14

Bitte bei der Abgabe Name der Mitglieder einer Gruppe, Nummer der Übung/Teilaufgabe und Datum auf den Lösungsblättern nicht vergessen! Darauf achten, dass die Lösungen beim richtigen Tutor abgegeben werden. Achten Sie bei Programmieraufgaben außerdem darauf, dass diese im Linuxpool kompilierbar sind. Nutzen Sie dazu die Flags `-std=c99`, `-Wall` und `-pedantic`. Es sollten keine Warnungen auftauchen.
Zu spät abgegebene Lösungen werden nicht berücksichtigt!

Aufgabe 1: Speicherverwaltung (2+3+1=6 Punkte)

In der Vorlesung haben Sie das Prinzip des *Pagings* kennengelernt. Es beinhaltet, dass im Hauptspeicher eine begrenzte Zahl von Speicherseiten (*Pages*) fester Größe gehalten wird, die kleiner als die Anzahl der tatsächlich existierenden Seiten ist. Seiten, die gerade nicht im Hauptspeicher gehalten werden können, werden auf ein externes Speichermedium ausgelagert.

Will ein Prozess auf eine solche Seite zugreifen, muss sie zuvor vom externen Speicher in den Hauptspeicher zurückgeholt werden (*Seitenfehler, Page Fault*). Sind noch nicht alle Plätze für Speicherseiten (*Frames*) im Hauptspeicher gefüllt, stellt dies kein Problem dar. Ist jedoch kein Platz mehr vorhanden, muss eine Seite im Hauptspeicher ausgewählt werden, die ausgelagert und durch die neue Seite ersetzt wird.

Für die Auswahl der zu ersetzenden Seite gibt es verschiedene Strategien:

FIFO (First In, First Out): Die jeweils älteste Seite wird aus dem Hauptspeicher ausgelagert.

LRU (Least Recently Used): Es wird die Seite aus dem Hauptspeicher entfernt, auf die am längsten kein Zugriff erfolgt ist.

SC (Second Chance): Diese Strategie ist eine Kombination aus FIFO und LRU. Pro Seite wird ein Use-Bit gespeichert, welches gesetzt wird, wenn nach dem Zugriff, der die Seite angefordert hat, ein weiterer Zugriff erfolgt. Ersetzt wird zuerst die älteste Seite mit nicht gesetztem Use-Bit. Bei einem Seitenfehler oder wenn die Use-Bits aller Seiten gesetzt sind ($\rightarrow$ keine zusätzliche Information mehr durch die Use-Bits), werden alle Use-Bits zurückgesetzt.

LFU (Least Frequently Used): Bei dieser Strategie wird die Seite mit der geringsten Nutzungshäufigkeit ausgetauscht. Die Nutzungshäufigkeit kann auf verschiedene Art und Weise ermittelt werden. Für diese Aufgabe wird angenommen, dass die Nutzung ab Beginn des Referenzstrings gezählt wird. Ist für mehrere Seiten die gleiche Nutzungshäufigkeit aufgetreten, wird die älteste dieser Seiten zuerst ausgetauscht.

CLIMB (Aufstieg bei Bewährung): Diese Strategie funktioniert prinzipiell wie FIFO, jedoch wird eine Seite, auf die ein erneuter Zugriff erfolgt, während sie bereits im Speicher ist, in der Schlange um eine Position „verjüngt“ (also mit ihrem Nachfolger vertauscht).

OPT (Optimalstrategie): Diese Strategie ersetzt die Seite zuerst, die in Zukunft am längsten nicht mehr benötigt wird. Die Strategie ist in der Praxis nicht realisierbar, weil dazu alle zukünftigen Zugriffe bekannt sein müssen.

Seitenzugriffe werden (im Modell) als *Referenzstring* angegeben, welcher die zeitliche Reihenfolge der angeforderten Seiten enthält. Ein solcher String ist z.B. „01230544351120675231“: Erst erfolgt ein Zugriff auf die Seite 0, dann auf Seite 1, dann auf Seite 2 usw. Seitennummern seien hier immer einstellig!

a) Welche Vor- und Nachteile hat das Paging verglichen mit dem Segmenting?

b) Gegeben sei ein System mit vier Frames. Geben Sie für jede der vorgestellten Strategien die Belegung dieser vier Speicherstellen zu jedem Zeitpunkt sowie die Anzahl der auftretenden Seitenfehler an. Legen Sie dabei den oben angegebenen Referenzstring zugrunde! Zu Beginn sei der Hauptspeicher leer.

c) Unter welcher Voraussetzung kann es sinnvoll sein, bei einem Seitenfehler nicht nur eine sondern gleich mehrere aufeinanderfolgende Seiten (*Prepaging, Look Ahead*) in den Hauptspeicher zu übertragen?

Aufgabe 2: Schutz und Sicherheit in C (3 Punkte)

In dieser Aufgabe sollen sie typische Programmierfehler in C kennenlernen und lernen diese zu vermeiden. Das bereitgestellte Programmbeispiel `dif.c` ermöglicht eine vereinfachte Zählung unterschiedlicher Zeilen in zwei Dateien.

(a) Machen Sie sich mit `<stdio.h>,<string.h>` und dem Beispiel vertraut.

(b) Testen Sie das bereitgestellte Codebeispiel ausführlich. Betrachten Sie dazu die bereitgestellten Beispiele. Welche Kombinationen werden korrekt ausgeführt? Welche nicht? Nennen Sie mögliche Fehlerquellen.

(c) Betrachten Sie nun den Code genauer. Welche weiteren Fehler finden Sie noch?

(d) Nennen Sie die Codestellen für Fehler aus **beiden** Teilaufgaben.

(e) Korrigieren Sie das Codebespiel. Machen Sie kenntlich welche Codestellen sie verbessern und warum.