Technische Universität Berlin Fachgebiet Komplexe und Verteilte IT-Systeme Sommersemester 2015	**Aufgabenblatt 6** zu – Systemprogrammierung – Prof. Dr. Odej Kao, Dr. Peter Janacik, Tutoren
Abgabetermin:	[1] – 06.07.2015 [2] – 13.07.2015

Aufgabe 6.1: Seitentauschverfahren (1,2 Punkte) (Theorie[1])

Gegeben sei ein physikalischer Speicher mit vier Kacheln. Auf folgende Seiten wird nacheinander zugegriffen:

$$R = 2, 7, 1, 7, 4, 2, 4, 5, 1, 3, 7, 8, 7, 3, 6$$

Geben Sie für die folgenden Seitentauschverfahren die jeweilige Seiten-Kachel-Zuordnung, die jeweils angegebene Datenstruktur sowie die Anzahl der Seitenfehler an. Machen Sie dabei sowohl erfolgreiche Zugriffe auf belegte Kacheln als auch Seitenzugriffsfehler kenntlich. Falls bei einem Verfahren mehrere Seiten gleich bewertet werden, soll die älteste Seite ausgelagert werden (FIFO).

a) Optimal

b) First-In-First-Out (FIFO) inklusive der Angabe der als nächstes auszulagernden Kachel

c) Least-Frequently-Used (LFU) inklusive der Angabe der Seitenzugriffe jeder Seite zu jedem Zeitpunkt

d) Least-Recently-Used (LRU) inklusive des Stapels/Stacks

Bitte stellen Sie ihre Ergebnisse in der folgenden Form dar:

Seite	2	...
Kachel 1	***2***	...
Kachel 2	-	...
Kachel 3	-	...
Kachel 4	-	...

Darstellung der Datenstruktur

Legende: **2** - (fett) Kachelzugriff auf Seite 2, ***2*** - (fett und unterstrichen) Zugriff nach Seitenzugriffsfehler auf Seite 2

Aufgabe 6.2: Speicherbelegungsstrategien (1,2 Punkte) (Theorie[1])

Die aktuelle Belegung des Speichers sei wie folgt (die dunklen Felder sind belegt, die weißen Felder sind frei):

Die folgenden Operationen treten für jeweils zusammenhängende Speicherbereiche in der angegebenen Reihenfolge auf:

$$A_1 = 2\,\text{MB}, A_2 = 3\,\text{MB}, A_3 = 10\,\text{MB}, A_4 = 2\,\text{MB}, A_5 = 4\,\text{MB},$$
$$free(A_3), A_6 = 5\,\text{MB}$$

Diese Anforderungen werden in der Reihenfolge der Ankunft verarbeitet, wobei bei free() der entsprechende Speicherbereich wieder freigegeben werden soll. Welche Speicherbereiche werden welchen Anforderungen jeweils zugeordnet, wenn als Belegungstrategie First Fit, Next Fit, Best Fit bzw. Worst Fit verwendet wird?
Stellen Sie die Belegung mindestens nach A_5 und nach der Belegung von A_6 dar. Sie können gerne mehr Zwischenschritte angeben, um die Nachvollziehbarkeit zu erhöhen.
Sollte für eine Anforderung kein passender Speicherbereich mehr verfügbar sein, überspringen Sie diese Anforderungen und fügen eine entsprechenden Bemerkung hinzu.

Aufgabe 6.3: Dateisystem (0,6 Punkte) (Theorie[1])

In blockbasierten Dateisystemen werden Dateien auf mehrere Blöcke verteilt auf der Festplatte abgelegt.

a) Was versteht man in diesem Zusammenhang unter Dateifragmentierung? Welche Verwaltungsstrukturen sind aus der Vorlesung für die Realisierung von Dateifragmentierung bekannt?

b) Welche Auswirkungen hat Fragmentierung bei Datenträgern mit langen Zugriffszeiten und warum wird bei flash-basierten Speichermedien von einer Defragmentierung (gezielte Umordnung der Dateiblöcke auf dem Medium zur Reduzierung der Fragmentierung) abgeraten?

Aufgabe 6.4: Seitentauschverfahren (1. Woche) (Tafelübung)

a) Warum ist es sinnvoll, virtuelle Speicheradressen zu verwenden? Welche Aufgabe übernehmen die Seitentauschverfahren in diesem Zusammenhang?

b) Gegeben seien ein physikalischer Speicher mit drei Kacheln und ein physikalischer Speicher mit vier Kacheln. Auf folgende Seiten wird nacheinander zugegriffen:

$$R = 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5$$

Geben Sie für beide Speicher die jeweilige Seiten-Kachel-Zuordnung, alle notwendigen Datenstrukturen sowie die Anzahl der Seitenfehler für die Seitentauschverfahren FIFO und LRU an. Vergleichen Sie die Anzahl der Seitenfehler zwischen beiden Speichern und diskutieren Sie die Ergebnisse.

Aufgabe 6.5: Buddy Verfahren (1. Woche) (Tafelübung)

In dieser Aufgabe soll das Buddy Verfahren per Hand simuliert werden. Gegeben sei ein 64 MB Speicherblock.

a) Simulieren Sie die in Tabelle 1 gegebenen Anfragen und schreiben Sie in jedem Schritt auf, wie die Freispeicher-Liste aussieht.

Schritt	Operation
1	`a1 = malloc(7 MB);`
2	`a2 = malloc( 17 MB);`
3	`a3 = malloc( 7 MB);`
4	`a4 = malloc( 4 MB);`
5	`free(a3);`
6	`free(a1);`
7	`a5 = malloc( 15 MB );`
8	`free(a2);`

Tabelle 1: Speicheranfragen

b) Wie hoch ist der maximale interne Verschnitt, der durch eine einzelne Speicheranforderung verursacht wird?
Wie groß ist nach Schritt 4 der mittlere interne Verschnitt pro Speicherallokation?

Aufgabe 6.6: Verwaltung von Speicherbelegungen (2. Woche) (Tafelübung)

- Überlegen Sie sich, wie eine Datenstruktur zur Speicherverwaltung aussehen könnte.
- Wie könnte man Speicher 'in situ' verwalten, damit die Verwaltungsinformationen zusammen mit dem Speicher in einem Block liegen?
- Wie sollte das Mengenverhältnis zwischen Verwaltungsdaten und verwaltetem Speicher sein?

Aufgabe 6.7: Seitenverwaltung (2. Woche) (Tafelübung)

Gegeben sei ein virtueller Adressraum, der aus 8 Seiten besteht (Seitengröße = Kachelgröße = 32 Byte). Für die Adressumsetzung zwischen virtuellen und physikalischen Adressen liegt die folgende Seitentabelle vor. Finden Sie für die gegebenen virtuellen Adressen die entsprechenden physikalischen Adressen.

Seitentabelle:
Adressen:

Aufgabe 6.8: Speicherfehler (2. Woche) (Tafelübung)

Erklären Sie den Unterschied zwischen einem Seitenfehler (page fault) und einem Speicherzugriffsfehler (segmentation fault). Wann genau tritt welcher der beiden Fehler auf? Wie reagiert das Betriebssystem darauf?

Adresse	Kachelnr.
0	3
1	4
2	1
3	2
4	5
5	6
6	0
7	7

Virtuelle Adresse	Seitennr. : Offset	Physikalische Adresse
000 10000		
001 11100		
010 00100		
111 00101		

Aufgabe 6.9: Verwaltung einer Seitentabelle (2 Punkte) (Praxis^2)

In dieser Aufgabe sollen Speicherzugriffe und die Speicherverwaltung über eine Memory-Management-Unit (MMU) simuliert werden. Es soll ein virtueller Adressraum von 0 bis 0x7FFF verwaltet werden, sodass die Adressbreite 15 Bit beträgt. Das Speicherlayout, also die Beschreibung der verfügbaren Geräte im physikalischen Adressraum, ist in der nachfolgenden Tabelle vorgegeben. Um die Simulation der MMU möglichst realistisch zu gestalten, sollen die Verwaltungsinformationen in einer vorgegebenen Seitentabelle untergebracht werden. **Eine Speicherung der Verwaltungsinformationen außerhalb der vorgegebenen Seitentabelle ist unzulässig.** Zur Vereinfachung ist ein Umgang mit den Daten im Speicher nicht erforderlich und auch nicht Teil der Aufgabe.

Funktion	Adresse (im Hexadezimalsystem)
RAM (Kernel User only)	0x0000
RAM (für alle)	0x1000
frei	0x4000
FLASH	0x4400
frei	0x4800
I/O Geräte	0x7800
frei	0x7804
größte Adresse	0x7FFF

Tabelle 2: Layout des physikalischen Speichers

Jede Seite/Kachel soll eine Größe von 16 Byte haben. Ein Eintrag in der Seitentabelle soll 16 Bit lang sein und wird daher in dieser Aufgabe mit dem Datentyp *int16_t* realisiert.

Neben der reinen Adressumsetzung soll zusätzlich eine einfache Rechteverwaltung realisiert werden. Zu diesem Zweck soll in einem Eintrag der Seitentabelle zusätzlich zum Index der Kachel, auf die die entsprechende Seite abgebildet werden soll, die folgenden Flags mit jeweils einem Bit gespeichert werden:

- *Available-Flag*: ist gesetzt, wenn der Eintrag gültig ist und ein Seitenzugriff erlaubt ist
- *Kernel-Flag*: ist gesetzt, wenn nur im Kernelmodus auf die Seite zugegriffen werden darf

- *Read-Only-Flag*: ist gesetzt, wenn auf diese Seite lediglich lesend zugegriffen werden darf

Eine Rechteverwaltung ist nur dann sinnvoll, wenn nicht jeder diese Rechte ändern kann. Aus diesem Grund soll zwischen einem Kernelmodus und einem Anwendermodus unterschieden werden. Das System befindet sich stets in einem der beiden Modi. Änderungen an der Seitentabelle sollen lediglich im Kernelmodus möglich sein. Bedenken Sie dies bei der Bearbeitung der Aufgabe.

Die Aufgabe besteht darin, folgende Funktionen zu implementieren:

- `void switch_mode(sys_mode_t new_mode)`

Diese Funktion simuliert den Wechsel vom Kernelmodus in den Benutzermodus und den umgekehrten Fall, indem der aktuelle Modus mit dem gegebenen Modus *new_mode* überschrieben wird. In einem realen Szenario würde der Wechsel aus dem Benutzermodus in den Kernelmodus nur durch einen entsprechenden Interrupt ausgelöst werden.

- `void init_page_table()`

Diese Funktion soll die vorgegebene Seitentabelle mit einem 1:1 Mapping zwischen dem virtuellen und dem physikalischen Adressraum initialisieren. Mit 1:1 ist dabei gemeint, dass alle virtuellen Adressen nach der Adressumsetzung auf die gleiche physikalischen Adressen führen. Jede Seite, welche auf eine Kachel abgebildet wird, welche ausschließlich physikalische Adressen in einem freien Speicherbereich enthält (siehe Speicherlayout), soll invalidiert werden. Es sollen dennoch immer alle relevanten Bits des Seiteneintrags initialisiert werden, auch wenn die Seite nach der Initialisierung nicht verfügbar ist.

- `void remap(int16_t page, int16_t tile)`

Diese Funktion ändert die Seitentabelle in der Art, dass die Seite mit dem Index *page* auf die Kachel mit dem Index *tile* abgebildet wird. Die Flags des Seitentabelleneintrags sollen dabei unverändert bleiben.

- `int16_t memory_access(int16_t address, access_type operation)`

Diese Funktion simuliert einen indirekten Speicherzugriff. Der Parameter *address* gibt dabei die virtuelle Speicheradresse an, auf welche zugegriffen werden soll. Außerdem wird mit *operation* die Art des Zugriffs auf den Speicher festgelegt (schreibend oder lesend).
Im Erfolgsfall soll die physikalische Adresse zurückgegeben werden, auf welche zugegriffen worden ist. Kommt es bei einem Speicherzugriff zu einem Fehler, so soll stattdessen der entsprechende Fehlercodes zurückgegeben werden. Die Fehlercodes sind in der *mmu.h* genauer beschrieben.

- `void close_page(int16_t page)`

Mit dieser Funktion soll die Seite mit dem Index *page* geschlossen (invalidiert) werden.

- `void open_page(int16_t page)`

Mit dieser Funktion soll die Seite mit dem Index *page* geöffnet (validiert) werden.

- **void** `page_kernel_only(int16_t page)`

Diese Funktion sorgt dafür, dass auf die Seite mit dem Index *page* nur noch im Kernelmodus zugegriffen werden darf.

- **void** `open_page_for_user(int16_t page)`

Diese Funktion sorgt dafür, dass auf die Seite mit dem Index *page* auch im Anwendermodus zugegriffen werden darf.

- **void** `page_read_only(int16_t page)`

Diese Funktion sorgt dafür, dass auf die Seite mit dem Index *page* nur noch lesend zugegriffen werden darf.

- **void** `page_read_write(int16_t page)`

Diese Funktion sorgt dafür, dass auf die Seite mit dem Index *page* sowohl lesend als auch schreibend zugegriffen werden darf.

- **void** `protect_system(`**void**`)`

In diesem Teil der Aufgabe wird angenommen, dass das Betriebssystem die ersten 4096 Bytes des RAMs einnimmt. Dieser Bereich soll daher vor Zugriffen im Anwendermodus geschützt werden. Nutzen Sie dazu die Methoden der Speicherverwaltung, die Sie zuvor implementiert haben. Es wird außerdem angenommen, dass Ein- und Ausgabegeräte ausschließlich vom Betriebssystem direkt angesprochen werden dürfen. Daher sollen auch diese vor Zugriffen im Anwendermodus geschützt werden.