Aufgabe 12 - Virtuelle Caches:

a) Diskutieren Sie die Vorgehensweise bei der Verwaltung von Zugriffsrechten bei virtuellen Caches.

b) Was wird unter den Begriffen Ambiguities und Aliases verstanden?

c) Welche Auswirkungen haben Einsprünge in das Betriebssystem (System-Calls, Traps) in Bezug auf die Gültigkeit der Cache-Inhalte?

d) Diskutieren Sie die Schwierigkeiten bei einem fork-System-Call in Unix-Systemen. Berücksichtigen Sie dabei den Fall ohne und mit copy-on-write-Mechanismus.

e) In einem Rechensystem befinden sich 2 lauffähige Prozesse. Diese Prozesse besitzen zwei gemeinsame Speicherbereiche (*shared memory*), die bei folgenden Anfangsadressen beginnen:

	Segment 1	Segment 2
Prozeß 1	0x00200	0x10000
Prozeß 2	0x00200	0x08000

Überlegen Sie, welche Aktionen das Betriebssystem in Bezug auf eventuell vorhandene Caches unternehmen muß, wenn eine Prozeßumschaltung stattfindet. Untersuchen Sie dabei die in der Vorlesung besprochenen 4 Cachearten.

f) Welche Schwierigkeiten ergeben sich aus DMA-Transfers? Welche Möglichkeiten hat das Betriebssystem, um diese Schwierigkeiten zu beheben? Welche Probleme lassen sich mit Hilfe von Hardware sehr effizient lösen (Stichwort *Snooping*)?

Lösung Aufgabe 12:

a) Erzeugt der Zugriff auf einen virtuellen Cache einen Write-Miss, so können die Zugriffsrechte durch die MMU geprüft werden. Bei einem Write-Hit unter der Strategie Write-Through ist diese Vorgehensweise ebenfalls möglich, da auf jeden Fall in den Hauptspeicher geschrieben werden muß. Anders dagegen bei einem Cache mit Write-Back-Strategie: wird hier ein Write-Hit erzeugt, so wird bei gesetztem Modify-Bit davon ausgegegangen, daß die Daten der Cache-Line bereits schreibende Zugriffsrechte besitzen, und die Daten in den Cache geschrieben. Ist das Modify-Bit nicht gesetzt, kann nicht festgestellt werden, ob die Daten schreibende Zugriffsrechte besitzen - in diesem Fall muss über die MMU das Zugriffsrecht explizit geprüft werden. Eine alternative Möglichkeit besteht darin, das Zugriffsrecht zum Schreiben auf Daten in ein eigenes Bit der Cacheline zu übertragen. Dadurch kann im Cache selbst eine Überprüfung stattfinden.

Werden die Zugriffsrechte der Daten eines Prozesses während der Ausführung geändert, muß der Cache geleert werden, da sich noch Daten mit den alten Zugriffsrechten im Cache befinden können.

b) Ambiguities (Mehrdeutigkeiten) treten auf, wenn *verschiedene physikalische* Adressen auf die *gleiche virtuelle* Adresse abgebildet werden. Die Folge ist, daß ein virtueller Cache diese physikalischen Adressen nicht unterscheiden kann und deshalb bei einem Zugriff auf die unterschiedlichen physikalischen Adressen nur die Daten einer Adresse liefert. Dieser Fall tritt potentiell bei jedem Prozesswechsel auf.

Aliase (Synonyme, Bedeutungsgleicheit) entstehen, wenn *verschiedene virtuelle* Adressen auf die *gleiche physikalische* Adresse abgebildet werden. Durch die unterschiedlichen virtuellen Adressen können die Daten in unterschiedlichen Bereichen des Caches abgelegt werden. Dadurch sind inkonsistente Zustände möglich. Dieser Fall tritt typischerweise bei Shared-Memory auf.

c) Wird bei einem fork-Aufruf eine echte Kopie der Speicherbereiche vorgenommen, so werden im Unix-System für den Kopiervorgang die physikalischen Seiten des zukünftigen Kindprozesses temporär in den virtuellen Adressraum des Kerns abgebildet. Dadurch kann die Kopie vom virtuellen Adressraum des Vaters aus durch eine direkte Adressierung des Kind-Adressraumes vorgenommen werden. Nach der Kopie wird die Abbildung wieder aufgehoben. Folgende Aspekte sind dabei unter Verwendung eines virtuellen Caches zu berücksichtigen:

- Durch die Verwendung des Vater-Adressraumes als aktuellen Adressbereich während des fork-Aufrufes stellen Daten, die vor dem Kopieren nur im Cache und noch nicht im Hauptspeicher existierten, kein Problem dar.
- Bevor der Kind-Prozess aktiviert wird, muß gewährleistet sein, daß seine Daten im Hauptspeicher stehen. Bei einer write-through-Strategie des Caches ist dies kein Problem - die entsprechenden Einträge im Cache müssen lediglich nach dem Kopieren invalidiert werden. Wird eine write-back-Strategie verwendet, muß der Cache invalidiert und der Haupt-

speicher aktualisiert werden. Diese Aktualisierung muß vor der Auflösung der temporären Abbildung geschehen, damit die MMU die virtuellen Adressen korrekt auflösen kann.

Bei Verwendung einer copy-on-write-Strategie muß unter Verwendung eines write-through-Caches keine Invalidierung des Caches vorgenommen werden, da kein Kopieren notwendig ist. Wird ein Cache mit write-back verwendet, muß der Hauptspeicher validiert werden. Ansonsten würde ein Zurückschreiben der modifizierten Cache-Lines nach dem fork-Aufruf ein copy-on-write aktivieren, wodurch Veränderungen, die vor dem fork-Aufruf stattgefunden hatten nur in einer privaten Kopie des Vaters landen würden.
Wird nach dem fork-Aufruf eine Seite anhand des copy-on-write-Mechanismus kopiert, wird auch in diesem Fall eine temporäre Abbildung im Kernel-Adreßraum erzeugt. Dadurch müssen äquivalent zum Fall ohne copy-on-write die relevanten Bereiche des Caches in den Hauptspeicher zurückgeschrieben werden, wenn eine write-back-Strategie verwendet wird.

d) 1. Problem: Ambiguities - Mehrdeutigkeit
Falls der Kern gleiche virtuelle Adressbereiche wie der User-Level verwendet, müssen virtuelle Caches sowohl beim Betreten als auch beim Verlassen des Kerns den gesamten Cache invalidieren. Sind die Adressbereiche disjunkt, treten Ambiquities dieser Form nicht auf. Dies ist in einem Unix-System der Fall.
2. Problem: Sicherheit
Wird der Cachespeicher beim Zurückspringen in den User-Level nicht invalidiert, so kann dies bei reinen virt. Caches ein Sicherheitsproblem darstellen, da der User-Prozeß auf noch gecachte Daten des Kerns zugreifen kann.

e) Segment 1: (*Shared Memory mit gleichen Anfangsadressen*)
Die physikalischen, virtuellen und virtuellen Caches mit physikalischem Tag stellen bei dieser Konstellation keinen Problemfall dar, da eventuelle Zweideutigkeiten durch übereinstimmung der Cache-Line und des Tags aufgelöst werden. Virtuelle Caches mit Prozeßtag können bei einer write-allocate-Strategie und einem direct-mapped Caches ebenfalls ohne Invalidierung auskommen.

Segment 2: (*Shared Memory mit unterschiedlichen Anfangsadressen*)
Physikalische Caches bereiten auch bei dieser Anordnung keine Probleme. Virt. Caches mit phys. Tags können die Mehrdeutigkeiten auflösen, wenn die Anfangsadresse des Segments jeweils in der selben Cacheline liegt. Ansonsten muß das Segment vollständig invalidiert werden (Aliases!). Diese Einschränkung gilt auch für virtuelle Caches mit Prozeßtags, wenn der Cache eine write-allocate Strategie besitzt (ohne write-allocate würde ein Cache-Miss direkt in den Hauptspeicher schreiben, wodurch die Daten im Cache nicht mehr aktuell sind). Virtuelle Caches benötigen bereits bei einem Prozeßwechsel eine Validierung des Hauptspeichers (Ambiguities), so daß bei einem Shared Memory, das in unterschiedlichen Adressbereichen eingeblendet wird, kein zusätzlicher Aufwand betrieben werden muß. Anders jedoch bei der mehrfachen Einblendung eines Shared Memories im Adressraum eines Prozesses. Liegen dabei die Anfangsadressen wieder in einer Cache-Line, kann anhand einer write-allocate

Strategie die Validierung des Hauptspeichers vermieden werden. Ansonsten muß nach jedem Zugriff auf das Shared Memory der Cache invalidiert und der Hauptspeicher aktualisiert werden.

Zu beachten ist, daß die Validierung des Hauptspeichers bei einer Abbildung auf die gleiche Cache-Line nur bei einem direct-mapped Cache funktioniert!

f) Da ein DMA direkte Änderungen am HS durchführt, wird der Cache über solche Änderungen nicht unterrichtet. Das Resultat können inkonsistente Cacheinhalte sein. Durch horchen des Caches am HS-Bus (snooping) kann der Cache eventuelle Inkonsistenzen erkennen und bereinigen.