Speichertypen und deren Effizienz im Linux Kernel

Seminar Effiziente Programmierung

Dominik Sander

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

27. Januar 2021

Unser Ausgangspunkt: Speicherallokation

- Das bietet uns Linux an:
  - `malloc` und `free`
  - `kmalloc` und `kfree`
  - `vmalloc` und `vfree`
- Im Kontext der effizienten Programmierung klären wir
  - die Unterschiede zwischen den Funktionen
  - die Anwendungsfelder der Funktionen
- Dafür benötigen wir einen Einblick in die Speicherverwaltung von Linux

Agenda

Physischer Speicher

Speicheradressen

- Ermöglichen eindeutigen Zugriff auf Speicherzellen
  - Kein Standard für die Adressierung
  - Kein Schutz vor unberechtigtem Zugriff
- Problem: Umgang mit physischen Speicheradressen ist schwierig
- Lösung: Virtuelle Speicheradressen

Virtual Memory

Was ist Virtual Memory?

- Wir führen einen virtuellen Adressraum ein
- Wir bilden Adressen aus dem virtuellen in den physischen Adressraum ab (Mapping)
  - Adressen zeigen auf physischen Arbeitsspeicher
  - oder auf andere Komponenten
- Zugriff auf Speicher erfolgt nur über virtuelle Adressen

Was bringt uns Virtual Memory?

- Das Konzept “Arbeitsspeicher” ist getrennt von physischer Umsetzung
  - Speicher kann auf Festplatte ausgelagert werden (Swapping)
  - Speicher kann beliebig verschoben werden
  - ...ohne dass der Prozess benachrichtigt werden muss

Was bringt uns Virtual Memory?

- Individuelle Mappings für einzelne Prozesse möglich
  - stellt jedem Prozess einen unfragmentierten Adressraum zur Verfügung
  - Prozess kann nicht auf Speicher von anderen Prozessen zugreifen
  - ...oder auf den Speicher des Kernels

Was bringt uns Virtual Memory?

- Kontrolliertes Teilen von Speicher möglich
  - Kommunikation zwischen Prozessen
  - Teilen von Ressourcen (z. B. DLLs)
- Umsetzung von Zugriffsberechtigungen möglich

Hardware-Komponenten

Hardware-Komponenten (Überblick)

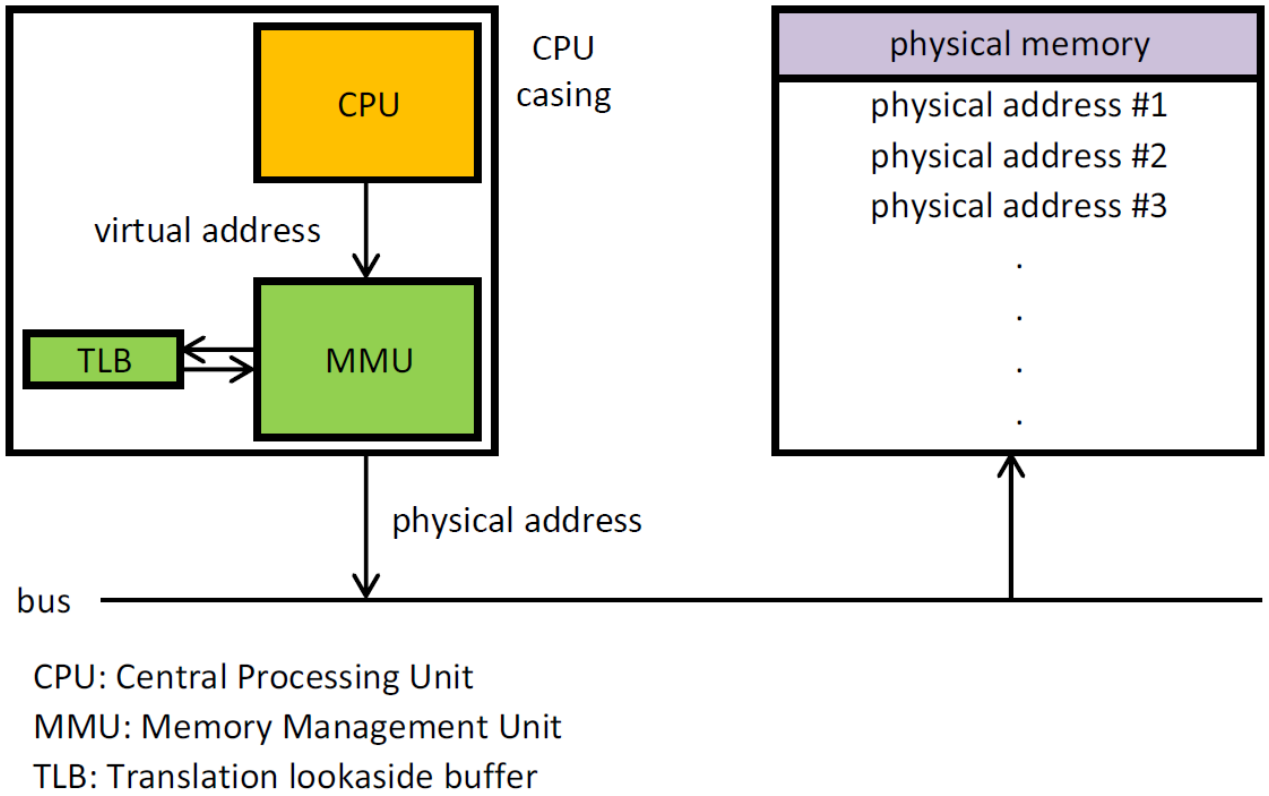

Abbildung 1: Virtual Memory im physischen Aufbau [Wik]

Memory Management Unit (MMU) [TB14, S. 195 ff.]

- Übernimmt das Mapping von virtuellen zu physischen Adressen
  - RAM, Peripheriegeräte, ...
- Wirft Page Faults bei unberechtigtem Zugriff
  - Zugriff mit einer ungültigen virtuellen Adresse
  - Zugriff mit fehlenden Berechtigungen
- Wirft Page Faults als Teil des Memory Management Designs
  - z. B. bei Zugriff auf swapped Memory

Pages [TB14, S. 195 ff.]

- Die MMU teilt den Arbeitsspeicher in Blöcke fester Größe (Pages) auf
  - Größe der Pages ist architekturabhängig (meistens 4kB)
  - eine oder mehrere virtuelle Pages werden auf physische Pages abgebildet
- Mithilfe von Page Tables ermittelt die MMU aus einer virtuellen Adresse die physische Page
  - Least significant Bits der virtuellen Adresse geben Offset in der physischen Page an
  - Page Tables sind Teil des Betriebssystems

Pages (Abbildung)

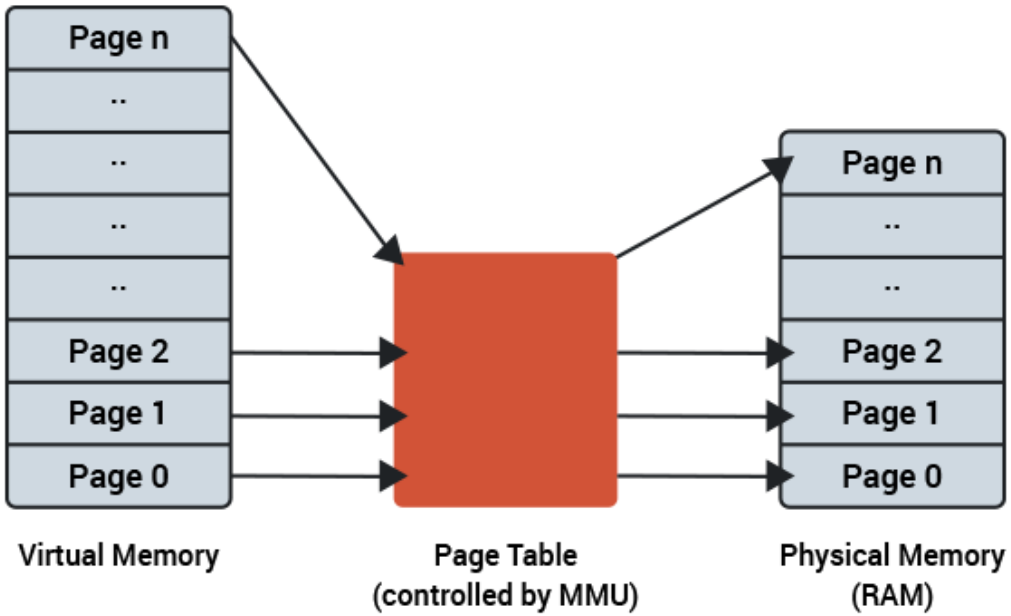

Abbildung 2: Aufteilung in Pages [Bre19]

Translation Lookaside Buffer (TLB) [TB14, S. 202 ff.]

- MMU cached aus Page Tables ermittelte Mappings in einer Liste
  - Ist in Hardware verbaut
  - Daher fester Größe
- Deutlich schnellere Ermittlung physischer Adressen als über die Page Tables des Betriebssystems
- Enthält nur die zuletzt verwendeten Mappings
  - Inklusive Permission Bits

Ermittlung der physischen Adresse (Flow Chart)

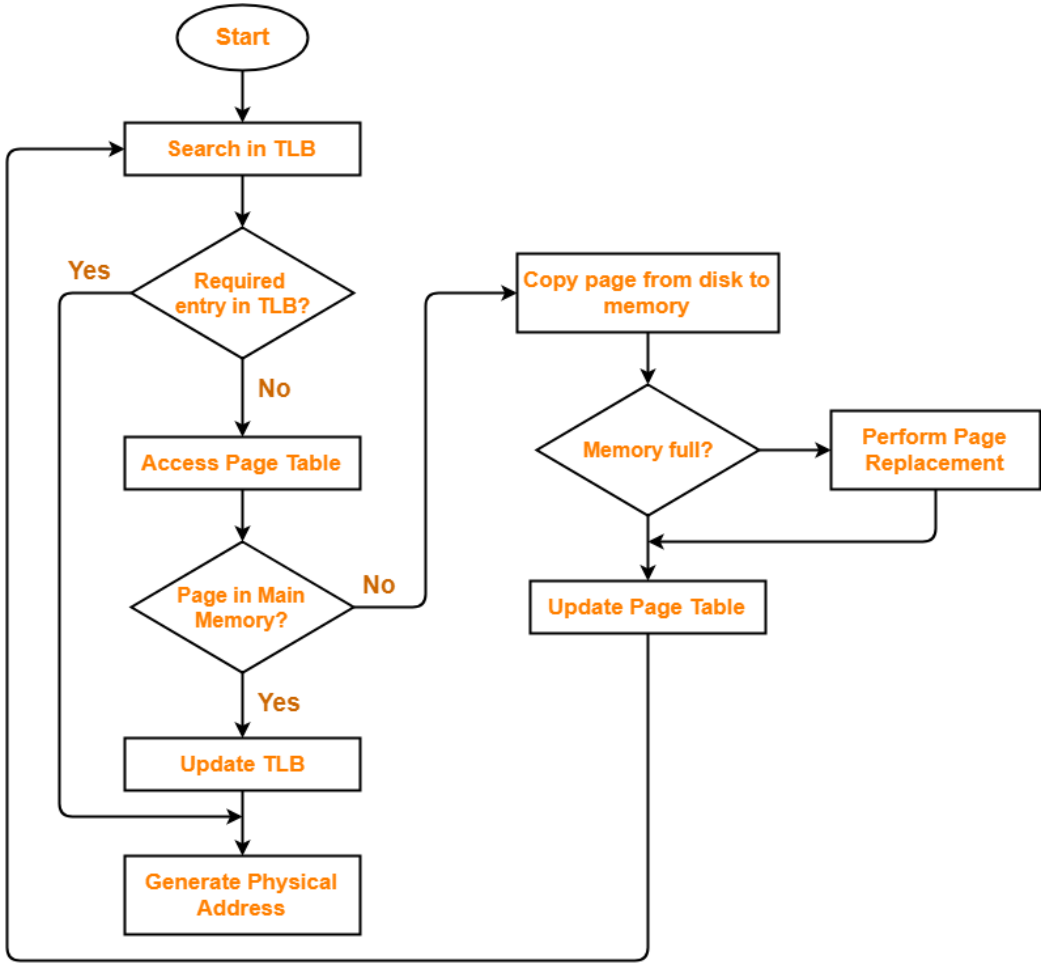

Abbildung 3: Ermitteln der physischen Adresse [Sin]

Arten von Virtual Memory

User Space & Kernel Space [CRK05, S. 19 ff.]

- Betriebssystem und Treiber werden im Kernel Space ausgeführt
  - Sind privilegiert (können alle CPU-Befehle ausführen)
  - Haben Zugriff auf den gesamten Arbeitsspeicher
- "Normale" Prozesse (z. B. Browser) werden im User Space ausgeführt
  - Können z. B. keinen `hlt` ausführen
  - Haben nur Zugriff auf den vom Kernel für den eigenen Prozess zugewiesenen Speicher
  - Hat nur über eine Schnittstelle (System Calls) Zugriff auf den Kernel

Zurück zu den `alloc`-Funktionen

- `malloc` für die Allokation von Speicher im User Space
  - wird von normalen Prozessen verwenden (z. B. Browser)
  - gibt eine User Virtual Address zurück
- `vmalloc` und `kmalloc` für die Allokationen im Kernel Space
  - wird z. B. von Treibern genutzt
  - gibt eine Kernel Virtual Address bzw. Kernel Logical Address zurück

Kernel Logical Address [CRK05, S. 413 ff.]

- Unterscheidet sich von physischer Adresse durch konstanten Offset
  - Einfache Umrechnung ohne die MMU
  - Kein Swapping
  - Somit ist der Zugriff schneller
- Virtuell zusammenhängender Speicher ist auch physisch zusammenhängend
  - ermöglicht Direct Memory Access (DMA)
- Stack von Kernel Space Programmen nutzen diesen Adressraum [Cor16]

Kernel Virtual Address [CRK05, S. 413 ff.]

- Adressen werden nicht linear auf physische Adressen abgebildet
  - virtuell zusammenhängend bedeutet nicht physisch zusammenhängend
- Ungeeignet für DMA
- Nutzt viele Vorteile von Virtual Memory
  - Abbildung auf Peripheriegeräte
  - Große Allokationen möglich

User Virtual Address

- Adressraum, der von User-Space Prozessen genutzt wird
- Jeder Prozess hat ein eigenes Mapping
  - Threads eines Prozesses teilen sich ein Mapping
- Nutzt die MMU im vollen Umfang aus

Ergebnis

Reminder: Unser Ausgangspunkt

- Das bietet uns Linux an:
  - `malloc` und `free`
  - `kmalloc` und `kfree`
  - `vmalloc` und `vfree`
- Im Kontext der effizienten Programmierung klären wir
  - wo die Unterschiede liegen
  - wo welche Funktionen verwendet werden
- Dafür benötigen wir einen Einblick in die Speicherverwaltung von Linux

Wann verwende ich was?

- `malloc` bei Prozessen im User Space
- `kmalloc` wird verwendet...
  - ...wenn physisch zusammenhängender Speicher wichtig ist
  - ...bei kleinen Allokationen
  - ...wenn schnelle Zugriffszeiten wichtig sind
  - `kmalloc` kann (im Gegensatz zu vmalloc) blockieren
- `vmalloc` wird bei großen Allokationen verwendet
  - Allokation gelingt (fast) immer
  - Speicher nicht physisch zusammenhängend
  - Zugriffe langsamer

Wann verwende ich was?

- Und bei unbekannter Größe der Allokation: `kvmalloc`
  - Versucht zuerst, im Kernel Logical Address Space zu allokieren
  - Ansonsten allokiert die Funktion im Kernel Virtual Address Space
  - Zurückgegebene Addresse kann eine Logical oder Virtual Address sein
- Freigeben des Speichers mit `kvfree`
  - Funktioniert auch mit Adressen von `kmalloc` bzw. `vmalloc`
  - Nützlich, wenn Adressraum nicht bekannt ist

Fazit

- `kmalloc` ist die erste Wahl
  - Man erhält den schnellsten Speicher
  - Öffnet DMA-Nutzung für andere Prozesse
- `kvmalloc` wenn man sich mit der Größe der Allokation unsicher fühlt
- `vmalloc` nur, wenn die Allokation wirklich groß ist
- Aber: Keine Regeln ohne Ausnahmen!

Bildquellen

[Bre19] Doug Breaker. *Understanding page faults and memory swap-in/outs: when should you worry?* 2019. URL: https://scoutapm.com/blog/understanding-page-faults-and-memory-swap-in-outs-when-should-you-worry (besucht am 21. Dez. 2020).

[Sin] Akshay Singhal. *Page Fault | Page Replacement Algorithms*. o. J. URL: https://www.gatevidyalay.com/page-fault-page-replacement-algorithms/ (besucht am 9. Dez. 2020).

[Wik] Wikipedia. *Memory management unit*. zitiert nach [TB14, S. 186 ff.]. o. J. URL: https://en.wikipedia.org/wiki/Memory_management_unit (besucht am 18. Dez. 2020).

Literatur

[com] The kernel development community. *The Linux Kernel documentation*. o. J. URL: https://www.kernel.org/doc/html/v5.0/index.html (besucht am 18. Dez. 2020).

[Cor16] Jonathan Corbet. *Virtually mapped kernel stacks*. 2016. URL: https://lwn.net/Articles/692208/ (besucht am 6. Jan. 2021).

[CRK05] Jonathan Corbet, Alessandro Rubini und Greg Kroah-Hartman. *Linux Device Drivers*. 3. Aufl. 2005.

[TB14] Andrew S. Tanenbaum und Herbert Bos. *Modern Operating Systems*. 4. Aufl. 2014.