Hausübungen 2 Betriebssysteme

Andre Rein <andre.rein@mni.thm.de>

Version v1.0, 07.06.2022: Final

Inhalt

1. Design und Implementierung eines Virtual Memory Manager

1.1. Aufgabenstellung

In dieser Hausübung implementieren Sie ein Programm, das logische in physische Adressen für einen virtuellen Adressraum der Größe 2^{16} = 65.536 Bytes übersetzt. Ihr Programm liest aus einer Datei logische Adressen ein, übersetzt diese unter Verwendung eines Translation Lookaside Buffers (TLB) und eines Page Tables in entsprechende physische Adresse und gibt den Wert des an der übersetzten physischen Adresse gespeicherten Bytes aus. Das Lernziel dieser Hausübung ist es, durch Simulation die Schritte bei der Übersetzung von logischen in physische Adressen zu verstehen, einschließlich der Behebung von Pagefaults durch Demand Paging, der Verwaltung eines TLB und der Implementierung eines Seitenersetzungsalgorithmus.

1.2. Voraussetzung

Für die Abgabe der Hausübung 2 müssen Sie die Hausübung 1 **vollständig präsentiert** und **abgegeben** haben.

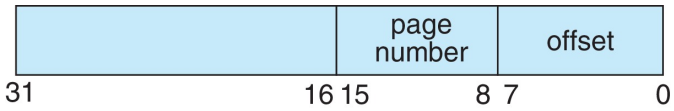
1.3. Zeitplan

Den finalen Abgabetermin entnehmen Sie bitte Moodle bzw. wird dieser auch im Abgabeskript angezeigt. ACHTUNG, diese Hausübung 2 muss in Einzelarbeit erfolgen und abgegeben werden!

Weitere Hinweise zur Abgabe der Hausübung finden sie in Section 3

2. Aufgabendetails

Ihr Programm liest eine Datei mit mehreren 32-Bit Ganzzahlen, die logische Adressen darstellen. Sie müssen sich jedoch nur mit 16-Bit-Adressen befassen, daher müssen Sie die letzten 16 Bit jeder logischen Adresse maskieren. Diese 16 Bit werden unterteilt in (1) eine 8-Bit-Pagenummer (p) und (2) einen 8-Bit Offset (d). Die Adressen sind aufgebaut, wie nachfolgend dargestellt:



Weitere Details sind:

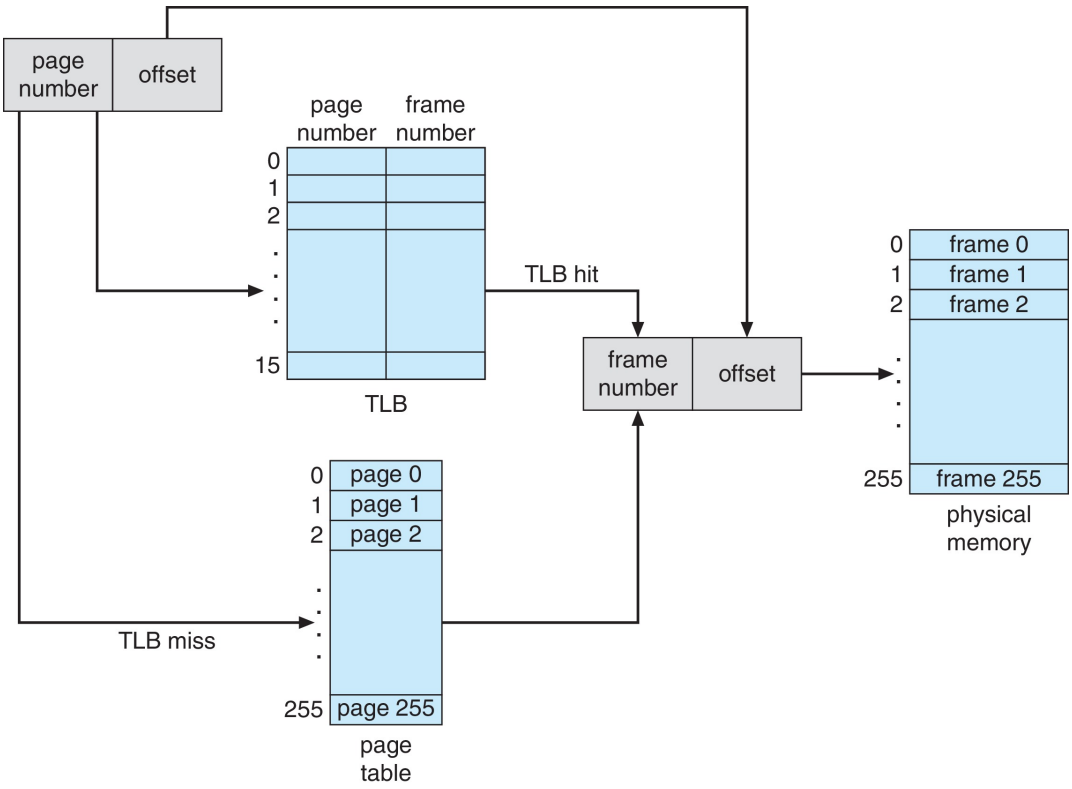
- 2^8 Einträge im Page Table
- Pagegröße 2^8 Bytes
- 16 Einträge im TLB
- Framegröße 2^8 Bytes

- Frames: 64
- Größe an physischem Speicher (64 Frames × 256-Byte Framegröße = 16384 Bytes)

2.1. Addressenübersetzung

Ihr Programm übersetzt logische in physische Adressen unter Verwendung eines TLB und einem Page Table, wie in der Vorlesung beschrieben. Zuerst wird die Pagenummer aus der logischen Adresse extrahiert. Nun wird zuerst nachgesehen ob die Pagenummer im TLB vorhanden ist. Wird die Pagenummer im TLB gefunden spricht man von einem TLB-HIT; dann wird die zugeordnete Framenummer direkt aus dem TLB bezogen und ausgegeben. Ist die Pagenummer nicht im TLB, handelt es sich um einen TLB-MISS; nun muss der Page-Table konsultiert werden.

Beim Page Table gibt es abermals zwei Fälle die unterschieden werden müssen. Wenn zu der Pagenummer eine Framenummer im Page Table eingetragen ist, wird die Framenummer ausgegeben. Andernfalls, d.h. es gibt (aktuell) keinen gültigen Eintrag, tritt ein sog. Pagefault auf. Dann ist es nötig zuerst den Speicherinhalt aus dem Backingstore in den Hauptspeicher zu laden. Hierzu wird der nächste freie Frame im Hauptspeicher ausgewählt, im Page Table eingetragen und ausgegeben.



2.2. Behandlung von Page Faults

Ihr Programm implementiert Demand-Paging wie in der Vorlesung beschrieben. Der Backingstore wird durch die Datei `BACKING_STORE.bin` repräsentiert, eine Binärdatei mit der Größe 65.536 Byte. Wenn ein Pagefault auftritt, lesen Sie eine 256 Byte lange Seite aus der Datei `BACKING_STORE.bin` ein und speichern sie in einem verfügbaren Frame im physischen Speicher.

Wenn beispielsweise eine logische Adresse mit der Pagenummer 15 zu einem Pagefault geführt hat, sollte Ihr Programm die Seite 15 aus `BACKING_STORE.bin` einlesen (denken Sie daran, dass die Pages

bei 0 beginnen und 256 Byte groß sind) und in einem Frame im physischen Hauptspeicher speichern. Sobald dieser Frame gespeichert ist (und der Page Table und TLB entsprechend aktualisiert wurden), werden nachfolgende Zugriffe auf Page 15 entweder durch den TLB oder den Page Table aufgelöst.

Sie müssen die Datei `BACKING_STORE.bin` als *Random-Access-Datei* behandeln, damit Sie beliebig zu bestimmten Positionen der Datei zum Lesen springen können. Empfohlen wird die Verwendung der Standard-C-Bibliotheksfunktionen für die Ausführung von I/Os, einschließlich `fopen()`, `fread()`, `fseek()`, und `fclose()`.

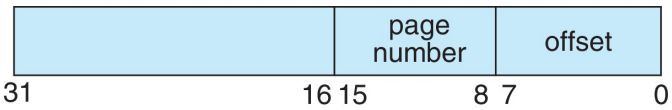
Die Größe des physischen Speichers (16384 Byte) entspricht nicht der Größe des logischen/virtuellen Adressraums (65.536 Byte). Sie müssen hierbei also eine Ersetzungsstrategie für die Zuordnung Page → Frame (ein sog. Page Replacement Strategie) verwenden. **Hierbei ist *FIFO* als Ersetzungsstrategie für den Page Table vorgegeben. Verwenden Sie ausschließlich diese Ersetzungsstrategie.**

2.3. Testdatei

Die Datei `addresses.txt` enthält ganzzahlige Werte, die logische Adressen im Bereich von 0 bis 65535 (die Größe des virtuellen Adressraums) repräsentieren. Ihr Programm öffnet diese Datei, liest jede logische Adresse ein, übersetzt diese in die entsprechende physische Adresse und gibt den Wert, der im physischen Speicher an der angegebenen physischen Adresse ist, als `unsigned` Byte-Wert aus.

2.4. Wie fängt man an?

Schreiben Sie zunächst ein einfaches Programm, das die Pagenummer und den Offset basierend von folgenden Zahlen berechnet: `1`,`256`,`32768`,`32769`,`128`,`65534`,`33153`



Der vielleicht einfachste Weg, dies zu tun, ist, die Operatoren für Bitmaskierung und Bit-Shifting zu verwenden. Sobald Sie die Pagenummer und den Offset zu einer ganzzahligen Zahl korrekt berechnen können, sind Sie bereit zu beginnen.

Im nächsten Schritt, implementieren Sie einen Page Table. *Hierbei können Sie vorerst annehmen, dass der virtuelle Adressraum und der physische Speicher gleich groß sind. Somit müssen Sie sich zunächst nicht um die Ersetzungsstrategie kümmern.* Sobald Ihr Programm unter dieser Bedingung korrekt funktioniert, reduzieren Sie die Speichergröße des physischen Speichers auf **64** verfügbare Frames. Beginnen Sie mit der Ersetzung von Frames nach dem FIFO-Prinzip sobald **keine freien Frames** mehr zur Verfügung stehen.

Sie können den TLB später integrieren, sobald Ihr Page Table ordnungsgemäß funktioniert. Denken Sie daran, dass die Adressübersetzung auch ohne TLB funktioniert; der TLB macht sie einfach nur schneller. Wenn Sie bereit sind, den TLB zu implementieren, denken Sie daran, dass er nur **16** Einträge hat, so dass Sie auch hier eine Ersetzungsstrategie verwenden müssen, wenn Sie einen

vollständig gefüllten TLB aktualisieren. Da die Implementierung des Page Tables FIFO als Strategie verwenden muss, wird empfohlen dies auch im TLB umzusetzen.

2.4.1. Mögliche Probleme nach der TLB Implementierung

Wenn Sie im Page Table einen neuen Eintrag, z.B. mit der Pagenummer `1` und dem Frame `5` anlegen dann müssen Sie prüfen ob im TLB für diesen Frame noch ein Eintrag existiert. Wenn im TLB für den Frame `5` ein Eintrag vorhanden ist, dann müssen Sie diesen Eintrag zuerst aus dem TLB entfernen. Verändern Sie hierbei nicht den internen Zähler der FIFO des TLB, ansonsten kann es passieren das sich in Ihrer Ausgabe die virtuellen Adressen ändern. Wenn Sie dieses Problem nicht beheben, sehen Sie bei einem Vergleich ihrer Ausgabe mit der `correct-64.txt` bei einigen Werten auf einmal andere Werte. Dies darf auf keinen Fall passieren und deutet auf das hier beschriebene Problem hin.

2.5. Herunterladen und Zugriff auf die Hausübung

Wenn Sie das Repository noch nicht auf Ihrem System haben, clonen Sie sich das GIT Repository von https://git.thm.de/arin07/betriebssysteme_public.

Das geht entweder mit:

```
$ git clone git@git.thm.de:arin07/betriebssysteme_public.git
```

oder

```
$ git clone https://git.thm.de/arin07/betriebssysteme_public.git
```

Ansonsten führen Sie eine Aktualisierung durch:

```
$ git pull
```

Wechseln Sie in das Verzeichnis: `cd betriebssysteme_public/hausuebung/hu2/` Dort finden Sie alles was Sie benötigen:

1. Im Verzeichnis `memory/src` finden Sie ein verwendbares Code-Skelett.
2. Implementieren Sie Ihre Lösung in der Datei `vmm.c`. Ihr Einstiegspunkt ist die Funktion `simulate_virtual_memory_accesses`, die automatisch im Verlauf des Programms aufgerufen wird.
3. Verwenden Sie zur Ausgabe nur die Funktion `print_access_results` aus der Datei `output_utility.c`

2.6. Kompilieren der Hausübung

2.6.1. Verwendung von `cmake`

```
$ cd betriebssysteme_public/hausuebung/hu2/memory/
$ mkdir build
$ cd build
$ cmake ../
$ make
```

2.7. Ausführung des Programms

Ihr Programm soll folgendermaßen ausgeführt werden:

```
$ ./vmm --backing ../BACKING_STORE.bin ../addresses.txt
```

2.8. Programmablauf

Ihr Programm liest die Datei `addresses.txt` ein, die 1.000 logische Adressen im Bereich von 0 bis 65535 enthält. Ihr Programm soll jede logische Adresse in eine physische Adresse übersetzen und den Inhalt des Bytes bestimmen, das an der richtigen physischen Adresse gespeichert ist. Als Datentyp können Sie `unsigned char` verwenden. Dieser Datentyp entspricht genau 1 Byte und verhindert die Ausgabe von negativen Werten.

Ihr Programm soll die folgenden Werte ausgeben:

1. Die zu übersetzende logische Adresse (der Integer-Wert wird aus der Datei `addresses.txt` gelesen).
2. Die entsprechend übersetzte physische Adresse.
3. Den *unsigned* Bytewert, der im physischen Speicher an der übersetzten physischen Adresse gespeichert ist.

2.8.1. Beispielausgabe

```
Virtual: 16916, Physical:    20, Value:   0, TLB hit: false, PT hit: false
Virtual: 62493, Physical:   285, Value:   0, TLB hit: false, PT hit: false
```

Die Datei `correct-256.txt` enthält die korrekten Ausgabewerte für die Datei `addresses.txt` unter der Annahme das die Speichergröße gleich der Größe des virtuellen Adressraums entspricht (256 Frames). Die Datei `correct-64.txt` enthält die Ausgabewerte für die Abgabeversion, mit reduziertem Hauptspeicher (16384 Bytes/ 64 Frames). Sie sollten diese Dateien verwenden, um festzustellen, ob Ihr Programm logische in physische Adressen korrekt übersetzt. Ihre Ausgabe muss exakt so erfolgen, wie in der Datei `correct-256.txt` bzw. `correct-64.txt` vorgegeben.

Verwenden Sie hierzu die Ausgabefunktion `print_access_results` aus der Datei `output_utility.c`.

2.9. Statistiken

Nach der Fertigstellung soll Ihr Programm die folgenden Statistiken ausgeben:

1. Pagefault Rate (Seitenfehlerrate) - Der Prozentsatz der Adressreferenzen, die zu Pagefaults geführt haben.
2. TLB-Hit Rate - Der Prozentsatz der Adressreferenzen, die bereits im TLB aufgelöst wurden. (Da die logischen Adressen in der adressen.txt zufällig generiert wurden und keine Speicherzugriffslokalität widerspiegeln, erwarten Sie keine hohe TLB-Hit Rate.)

Die Ausgaben dafür sind schon vorbereitet. Sie müssen nur noch die angelegte Struktur `stats` vom Typ `Statistics` korrekt befüllen.

3. Abgabe der Lösung

ACHTUNG: Diese Hausübung muss in Einzelarbeit bearbeitet und abgegeben werden. Die Lösungen müssen diesmal **NICHT** in den Übungsgruppen präsentiert und abgenommen werden. Die Prüfung der Korrektheit Ihres Programms erfolgt automatisiert mit Ihnen **unbekannten** Eingabewerten! Die eigentliche Abgabe erfolgt mittels dem bekannten Verfahren zur Abgabe mittel Skript.

Hierzu rufen Sie im Stammverzeichnis `betriebssysteme_public/hausuebung/hu2/` das Skript `submit.sh` auf.

Dann folgen Sie den Anweisungen auf dem Bildschirm. Beim erstmaligen Start müssen Sie zuerst Daten zu sich und Ihrem Team angeben. Das Skript führt Sie durch diese Schritte. Eine Abgabe kann mehrmals erfolgen, d.h. wenn Sie ein korrigierte Version abgeben möchten, führen Sie das Skript nochmals aus.

Die Prüfung auf Korrektheit Ihrer Lösung erfolgt direkt bei der Abgabe selbst. Sie können so oft abgeben wie Sie möchten bzw. das System dazu verwenden Ihre Lösung zu überprüfen. Sobald eine Lösung vom System akzeptiert wurde, gilt Ihre Lösung als abgegeben. In diesem Fall haben Sie alle Hausübungen für Betriebssysteme erfolgreich abgegeben. Sollten bei der Plagiatsprüfung keine Auffälligkeiten auftreten, haben Sie hiermit die Zulassung zur Klausur erworben!