Hasso Plattner Institut
Digital Engineering · Universität Potsdam

Übungsbesprechung Blatt 5
Vorlesung Betriebssysteme I (WS 2018/19)

Operating Systems And Middleware Group

2019-01-24

1 Virtuelle Adressen

1. Virtuelle Adressen
Beispielrechnung für eine fiktive Architektur

Architektur:

- Adressbreite: **26 bit**
- Seitengröße: **256 byte**
- **2 Statusbits** je Seite
- **Einstufige** Adressübersetzung

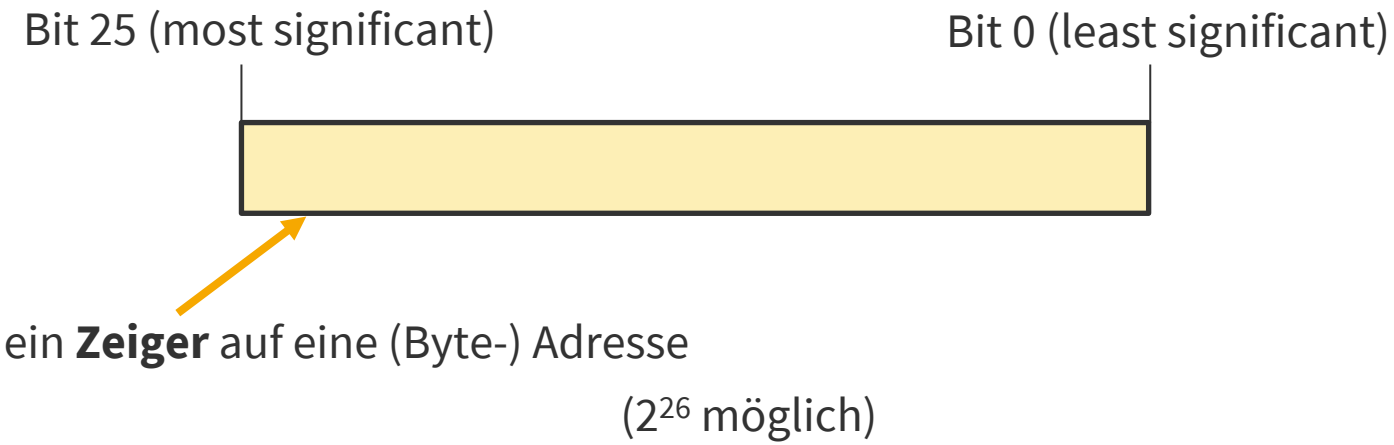
Ein Rechner mit dieser Prozessorarchitektur:

- **24 MiB** RAM
- **5 MiB** reserviert für das OS
- **8 Prozesse** laufen zeitgleich

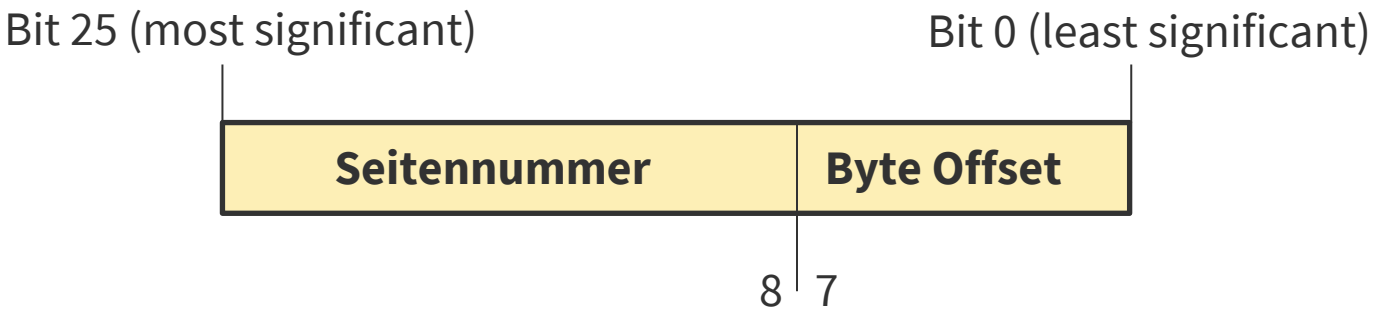
1. Virtuelle Adressen
Beispielrechnung für eine fiktive Architektur

<Wie viele Bits hat ein Seitentabelleneintrag?>

1. Virtuelle Adressen
Beispielrechnung für eine fiktive Architektur

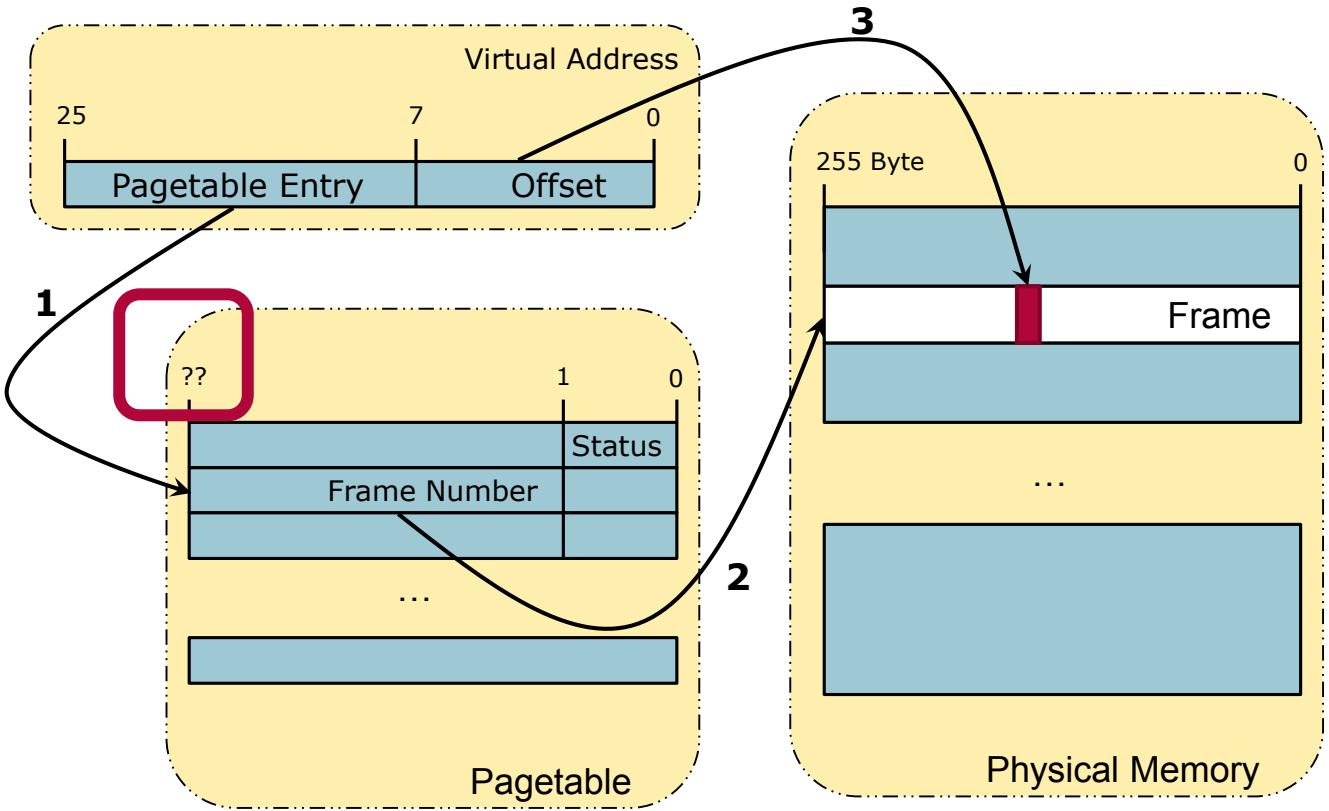


1. Virtuelle Adressen
Beispielrechnung für eine fiktive Architektur



Alle 256 Bytes eine neue Page, Adressen dazwischen per Offset gegeben

1. Virtuelle Adressen
Beispielrechnung für eine fiktive Architektur



Hasso Plattner Institut

<Wie viele Bits hat ein Seitentabelleneintrag?>

Adressraum: 2^{26} Byte = 2^{6} * 2^{20} Byte = 2^{6} MiB = **64 MiB**

Anzahl Seitentabelleneinträge: 64 MiB / 256 Byte = 2^{26-8} = 262.144 Einträge

18 bits notwendig um alle 262.144 Einträge zu zählen.

18 bits (Seitenzahl) + 2 bits (Status) = **20 bits**

1. Virtuelle Adressen
Beispielrechnung für eine fiktive Architektur

Hasso Plattner Institut

<Wie groß ist die Seitentabelle?>

20 Bits * 262.144 Einträge $= 20 \text{ bits} * 2^{18} = 5.242.880 \text{ bits}$

$= 20 \text{ bits} * 2^{10} * 2^{8} * 2^{-3} \text{ bits/B}$

$= 20 * 2^{5} \text{ KiB} =$ **640 KiB** $= 0.625 \text{ MiB}$

Hasso Plattner Institut

<Wie groß ist das maximale Working-Set?>

24 MiB	(verbauter Arbeitsspeicher)
- 8 * 0.625 MiB	(Seitentabellen für 8 Prozesse)
- 5 MiB	(reservierter Speicher für OS)

= **14 MiB**

2 Seitenersetzung

2. Seitenersetzung
Strategien

Nicht aller Speicher eines Prozesses passt in den physischen Speicher.

Der Inhalt einzelner Seitenrahmen (Frames) muss ausgetauscht werden.

Entscheidung für Frames entsprechend Page-Replacement-Algorithmus (z.B. FIFO, Second-Chance, Least-Recently-Used)

Optimierungsziel: Anzahl der Page-Faults minimieren

2. Seitenersetzung
Beispiel Szenario

Sequenzieller Zugriff auf Seiten:

2, 1, 4, 2, 3, 4, 2, 1, 3, 4, 3

Vereinfachung:

System hat 3 Seitenrahmen, bilden komplettes Working Set (kein OS, keine anderen Prozesse).

Legende:

- Zuerst im Speicher
- Kann ersetzt werden
- Page Hit
- Access-Bit

2. Seitenersetzung
First-In-First-Out (FIFO)

„Älteste“ Seite wird zuerst ersetzt

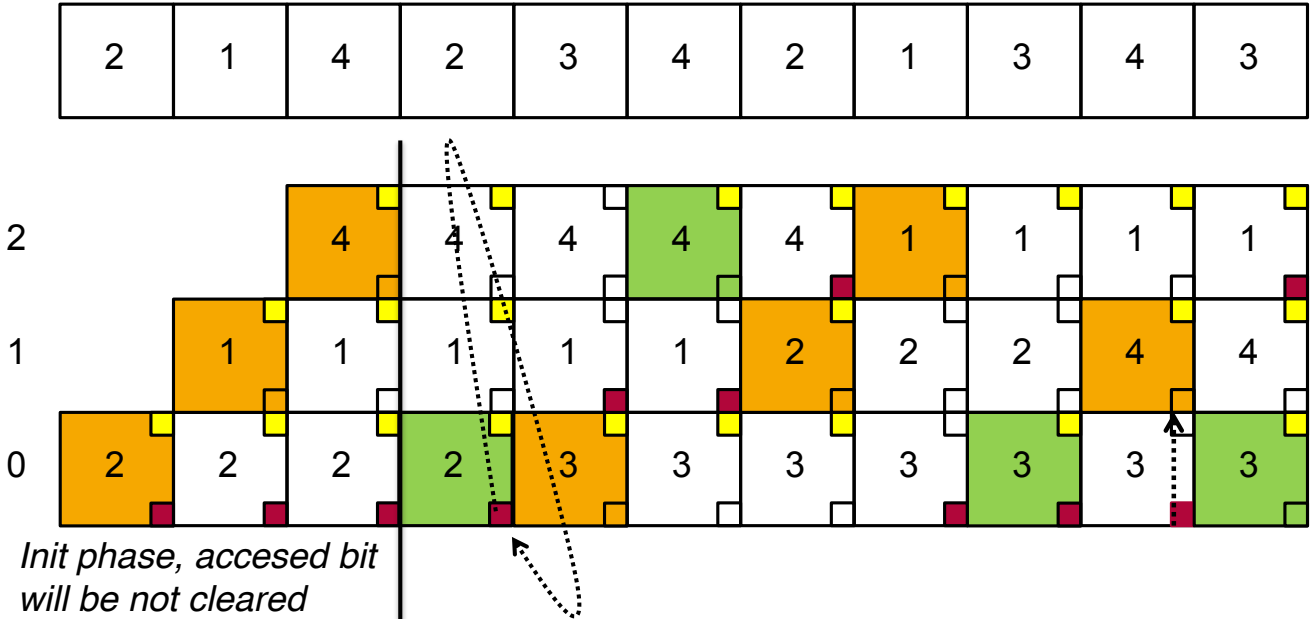
Einfache Implementierung: Queue, oder rotierender Zeiger auf Ersetzungsstelle (Ringpuffer)

	2	1	4	2	3	4	2	1	3	4	3
2			4	4	4	4	4	1	1	1	1
1		1	1	1	1	1	2	2	2	2	3
0	2	2	2	2	3	3	3	3	3	4	4

2. Seitenersetzung
Second Chance (SC)

FIFO-Erweiterung: Ersetzt „Älteste“ Seite, die nicht „kürzlich“ benutzt wurde

Implementierung mithilfe Statusbits (*access bit*)



2. Seitenersetzung
Second Chance (SC)

Erfolgreicher Seitenzugriff setzt Access-Bit (oft automatisch von MMU).

Solang unbelegte Frames existieren (Init-Phase): FIFO

Bei Page Miss (nach Init-Phase):

- Such in Queue nach erster Seite ohne gesetztes Access-Bit
- Lösch alle anderen Access-Bits
- Ersetz Seite mit gefundenem Kandidaten
- Setz Access-Bit auf neu geladener Seite
- Geh in der Queue zum nächsten Frame

2. Seitenersetzung
Least Recently Used (LRU)

Ersetzt am längsten unbenutzte Seite

Nicht effizient implementierbar ohne Zähler bei jeden Seitenzugriff.
Näherung: *Heuristic Aging* – periodisch die Access-Bits auswerten

	2	1	4	2	3	4	2	1	3	4	3
2			4	4	4	4	4	4	3	3	3
1		1	1	1	3	3	3	1	1	1	1
0	2	2	2	2	2	2	2	2	2	4	4

2. Seitenersetzung
Optimaler Algorithmus

Ersetzt Seite, die am längsten nicht genutzt werden wird

Implementierung unmöglich ohne Zukunftsvorhersage (aller kommenden Seitenzugriffe)

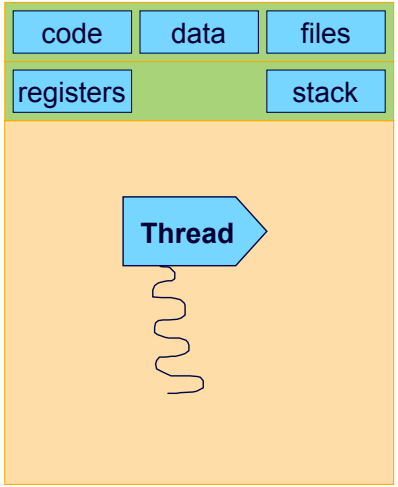
	2	1	4	2	3	4	2	1	3	4	3
2			4	4	4	4	4	4	4	4	4
1		1	1	1	3	3	3	3	3	3	3
0	2	2	2	2	2	2	2	1	2	4	4

2. Seitenersetzung
Zusammenfassung

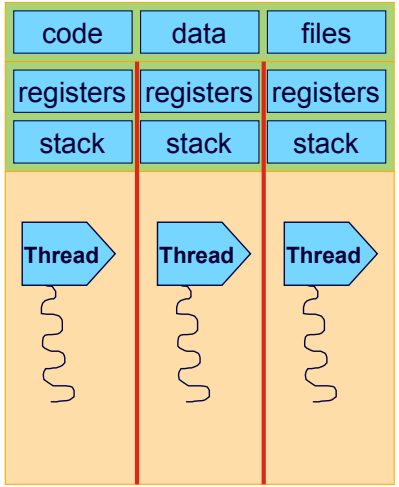
FIFO	8
Second Chance	7
Least-Recently-Used	7
Optimaler Algorithmus	5

Anzahl der Page Faults im Szenario

3 Scheduling



single-threaded



multi-threaded

Hasso Plattner Institut

Job Scheduler (long-term) ::

Wählt den nächsten zu laufenden Prozess für die Ready-Queue.
Selten gerufen, darf (relativ) langsam sein.

Betrachtet Speicherverwaltung (z.B. ausgelagerte Prozesse)

Windows hat keinen long-term Scheduler

Hasso Plattner Institut

CPU Scheduler (short-term) ::

Wählt den nächsten auf der CPU auszuführenden Thread aus

(Scheduling-)Entscheidung können anstehen, wenn ein Thread die Kontrolle abgibt (yield), d.h.:

- Von Running zu Waiting wechselt
- Von Running zu Ready wechselt
- Von Waiting zu Ready wechselt
- Terminiert

} **kooperativ**

oder unterbrochen wird (z.B. Quantumsablauf) **präemptiv**

Hasso Plattner Institut

Dispatcher ::

Weist der CPU den (vom Scheduler gewählten) Thread **zu**, d.h.:

- Führt Kontextwechsel aus
- (ggf.) wechselt in den User-Mode
- Setzt Programmzeiger auf die richtige Stelle um das Programm fortzuführen

Dispatch latency ::

Notwendige Zeit um einen Thread anzuhalten und einen anderen zu starten.

3. Scheduling
Scheduling Algorithmus: Round Robin (RR)

Hasso Plattner Institut

Präemptiver FIFO Scheduling Algorithmus

Nach Ablauf einer kleinen Zeiteinheit q (Quantum) wird aktueller thread unterbrochen und ans Ende der Bereitsschaftsschlange (Ready-Queue) gehangen

Jeder der n Ready-Threads bekommt 1/n der CPU-Zeit

Jeder Thread wartet nicht länger als (n - 1)q

Kleine q: Hoher Overhead.

3. Scheduling
Multi-Level Queue

Nicht eine Bereitschaftsschlange (Ready-Queue), sondern mehrere, z.B.:

- Nach Prioritäten
- Real-Time vs Interaktiv

Queues können unterschiedliche Scheduling-Algorithmen haben.

Beispiel Windows:

- Real-Time: Round Robin
- Interaktiv: Round Robin + Priority-Elevation + Quantum Stretching

4 BYOS – Build Your Own Scheduler

```
003
[    0.109757]  0000000000000003 ffff8802258c3dc0 ffffffff8100608f ffff8802258e
000
[    0.109928] Call Trace:
[    0.109975]  [<ffffffff810104ff>] release_ds_buffers+0xbf/0xd0
[    0.110027]  [<ffffffff8100608f>] x86_release_hardware+0x8f/0xa0
[    0.110077]  [<ffffffff810060ae>] hw_perf_event_destroy+0xe/0x20
[    0.110129]  [<ffffffff81183746>] _free_event+0xb6/0x230
[    0.110178]  [<ffffffff81183983>] put_event+0xc3/0x100
[    0.110227]  [<ffffffff811839c9>] perf_event_release_kernel+0x9/0x10
[    0.110280]  [<ffffffff8113ccae>] watchdog_nmi_disable+0x4e/0x70
[    0.110332]  [<ffffffff810a4b00>] ? sort_range+0x30/0x30
[    0.110381]  [<ffffffff8113cd2c>] watchdog_disable+0x5c/0x80
[    0.110431]  [<ffffffff810a4be8>] smpboot_thread_fn+0xe8/0x160
[    0.110482]  [<ffffffff810a1845>] kthread+0xe5/0x100
[    0.110531]  [<ffffffff810a1760>] ? kthread_create_on_node+0x1e0/0x1e0
[    0.110584]  [<ffffffff81844a0f>] ret_from_fork+0x3f/0x70
[    0.110633]  [<ffffffff810a1760>] ? kthread_create_on_node+0x1e0/0x1e0
[    0.110684] Code: 4c 89 d7 e8 9c 83 fa ff eb 8c 41 b8 01 00 00 00 48 89 d9 48
 89 da 4c 89 d6 e8 06 fc ff ff e9 73 ff ff ff 49 8b 42 20 a8 01 75 c7 <0f> 0b 0f
 1f 80 00 00 00 00 0f 1f 44 00 00 55 48 89 e5 41 57 41
[    0.112493] RIP  [<ffffffff811f00a7>] kfree+0x147/0x150
[    0.112568]  RSP <ffff8802258c3d58>
[    0.112618] ---[ end trace 80bdc4bb7e151ce8 ]---
_
```

4. Build Your Own Scheduler
Aufgabe: Userspace-Scheduler

Kernel hacking ist schwer …

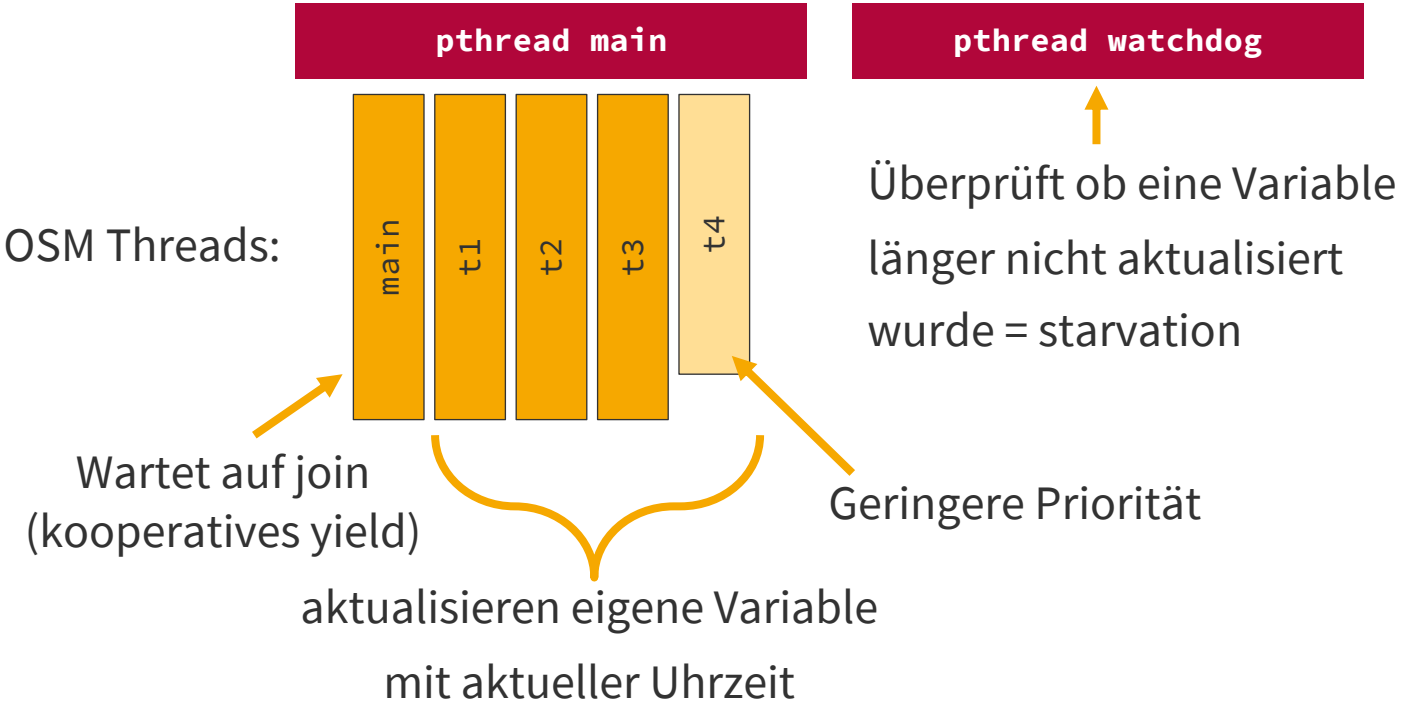
We present: **osmthread**

Eine einfache, ausbaufähige Userspace-Threading-Bibliothek

Nutzt präemtives Round-Robin-Scheduling innerhalb von Multi-Level-Queues (je eine für jede Priorität)

Aufgabe: Verhindern Sie das Verhungern niederpriorer Threads.

4. Build Your Own Scheduler
Testprogramm: Wie erkennnen wir Starvation?



end