

Chapter 13-15

1. Testat

- Systemdurchsatz (Throughput)
 - Wieviel Jobs/Zeit
 - Dafür optimal sind SJF und STCF
- Scheduling Analyse
 - berechnet/bestimmt: **Startzeit + Turnaroundtime**
 - gegeben ist: **Ankunftszeit, Servicetime**
- Exec() Family: Fehlerbehandlung des OS?
 - Keine !!
 - Aufruf der exec() Funktionen kann schief gehen
 - errno Variable muss selbst ausgewertet werden
 - Stackaufbau immer der gleiche bei allen execv() Familien

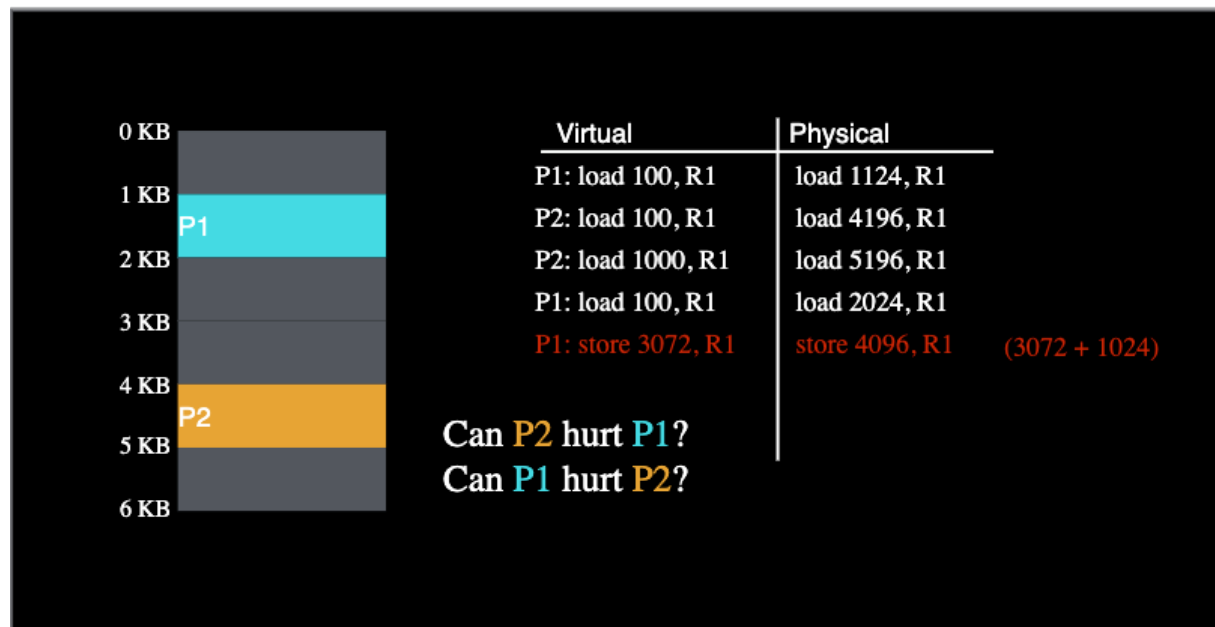
→ Nächste Wochen wird es hart sein, also früh anfangen!

Example: 64 Bit Linux System

```
cat /proc/8080/maps
# address          Perms  Offset               Dev    Inode    Pathname
00400000-00406000   r-xp  00000000           03:05  1437882  /usr/sbin/metalog
00505000-00506000   rw-p  00005000           03:05  1437882  /usr/sbin/metalog
00506000-00527000   rw-p  00006000           00:00   0       [heap]
2b76d29df000-2b76d29f5000   r-xp  00000000           03:02  61500    /lib64/ld-2.3.6.so
2b76d29f5000-2b76d29fc000   rw-p  2b76d29f5000      00:00   0
2b76d2af5000-2b76d2af6000   r--p  00016000           03:02  61500    /lib64/ld-2.3.6.so
2b76d2af6000-2b76d2af7000   rw-p  00017000           03:02  61500    /lib64/ld-2.3.6.so
2b76d2af7000-2b76d2b0c000   r-xp  00000000           03:05  311804   /usr/lib64/libpcrcr.so.0.0.1
2b76d2b0c000-2b76d2c0c000   ---p  00015000           03:05  311804   /usr/lib64/libpcrcr.so.0.0.1
2b76d2c0c000-2b76d2c23000   rw-p  00015000           03:05  311804   /usr/lib64/libpcrcr.so.0.0.1
2b76d2c23000-2b76d2d41000   r-xp  00000000           03:02  62921    /lib64/ls/libc-2.3.6.so
2b76d2d41000-2b76d2e41000   ---p  0011e000           03:02  62921    /lib64/ls/libc-2.3.6.so
2b76d2e41000-2b76d2e44000   r--p  0011e000           03:02  62921    /lib64/ls/libc-2.3.6.so
2b76d2e44000-2b76d2e47000   rw-p  00121000           03:02  62921    /lib64/ls/libc-2.3.6.so
2b76d2e47000-2b76d2e4e000   rw-p  2b76d2e47000      00:00   0
7fffd80b5000-7fffd80cb000   rw-p  7fffd80b5000      00:00   0       [stack]
ffffffff600000-ffffffffe00000 ---p  00000000           00:00   0       [vdso]

cat /proc/790/maps
00400000-004af000   r-xp  00000000           03:02  62290    /bin/bash
005ae000-005b9000   rw-p  000ae000           03:02  62290    /bin/bash
005b9000-00728000   rw-p  005b9000           00:00   0       [heap]
.....
```

→ schaue statisch und dynamisches Linked wegen der .so. Das sind dynamische Bibliotheken. Beim statisch wären diese libs wahrscheinlich im Code Bereich.



→ Nur base ist festgelegt

Who Controls the Base Register?

- Who should do translation with base register?
(A) process, (B) OS, or (C) HW

Polling option:

A

C

Professor Dr. Michael Mächtel

→ C, muss schnell sein, also die Hardware

Who Controls the Base Register?

- Who should modify the base register?
(A) process, (B) OS, or (C) HW

Professor Dr. Michael Mächtel

15

→ B

relocation.py

- In der Simulation relocation.py wird eine einfache Speicherverwaltung mittels base und bounds register simuliert. Überlegen Sie sich die base und wählen Sie dann den passenden bounds Wert so exakt wie möglich, so dass sich die folgende Abbildung ergibt.

Virtual Address	->	Physical Address	
8000		11006	Base? _____
100		3106	
2000		5006	Bounds? _____
2001		5007	

Base:

- (A) 100
- (B) 2000
- (C) 3000
- (D) 3006
- (E) 11006

D - 3006

Bounds - 100

relocation.py

- In der Simulation relocation.py wird eine einfache Speicherverwaltung mittels base und bounds register simuliert. Überlegen Sie sich die base und wählen Sie dann den passenden bounds Wert *so exakt wie möglich*, so dass sich die folgende Abbildung ergibt.

Virtual Address	->	Physical Address	
8000		11006	Base? _____
100		3106	
2000		5006	Bounds? _____
2001		5007	

Bounds:

- (A) 100
- (B) 8000
- (C) >100
- (D) >8000
- (E) 2001

D, ausprobieren selber in Simulator. Schaue auch für die quizzes