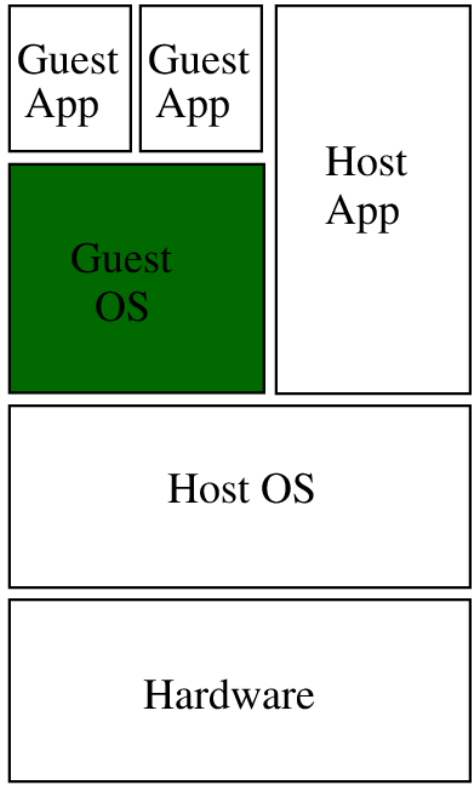
Paravirtualisierung (1)

Department Informatik 4
Systemsoftware
Friedrich-Alexander-Universität Erlangen-Nürnberg

WS 2026/2027



Paravirtualisierung



- Unter Gast-OS laufende Applikationen sehen Gast-OS-ABI.
- Gast-OS selbst sieht als „Hardware" das Gastgeber-OS.

Portierung eines Betriebssystems auf besondere „Hardware" (Gastgeber-OS)



Paravirtualisierung

Supervisor-Instuktionen nicht nutzbar. Diese müssen ersetzt werden.

Gast-OS kann nutzen

- CPU im User-Modus
- Speicher
- System-Calls des Gastgeber-OS



Paravirtualisierung

Supervisor-Instruktionen am Beispiel der x86-Architektur:

- `in{b,w,l}`, `out{b,w,l}` (Ein-/Ausgabe)
- `mov ..., %cr3` (Pointer auf Page-Tabelle laden)
- `cli`, `sti` (Disable/Enable Interrupts)
- `pushf`, `popf` (Interrupt Status speichern/laden)
- `lidt` (Pointer auf Interrupt-Tabelle laden)
- ...



Paravirtualisierung – Beispiel

Abbildung der x86-Supervisor-Instruktionen auf Unix/Linux System-Calls.

Idee:

phys. Speicher: eine Datei

virt. Speicher: eine gemappte Datei

Interrupts: Signale

Real-Time-Clock/Timer: System-Clock, Timer

Konsole: Terminal oder GUI

Netzwerkkarte: Socket

Sound-Karte: Sound-System

...



Paravirtualisierung – I/O

`in`- und `out`-Befehle in Gerätetreibern.

Gerätetreiber werden ersetzt.

Im Gerätetreiber z.B. nutzbar:

- `read`-/`write`-System-Calls (non-blocking)
- `send`-/`recv`-System-Calls (non-blocking)
- Grafik-Ausgabe
- Sound-Ausgabe



Paravirtualisierung – MMU

mmap- und munmap-System-Calls:

```
void *
mmap(void *addr,      /* virt. Adresse */
     size_t length,   /* 4096 */
     int prot,        /* read/write/execute */
     int flags,       /* shared */
     int fd,          /* File-Descriptor */
     off_t offset);   /* File-Offset (phys. Adresse) */
```

```
int
munmap(void *addr,      /* virt. Adresse */
       size_t length); /* 4096 */
```



Paravirtualisierung – MMU

Mit `mmap`- und `munmap`-System-Calls kann eine MMU nachgebildet werden.

Probleme:

- Viele System-Calls für einen Kontext-Wechsel notwendig.
- System-Calls langsam.
- Nicht beliebig viele Mappings möglich.

=> Nur gerade benötigte Teile des Adressraumes „on demand" mappen (ähnlich TLB).

- Host-OS belegt Teil des virtuellen Adressraums.
  => Guest-OS muss i.A. verschoben werden.





Paravirtualisierung – MMU

- Guest-OS läuft im User-Modus
- Guest-Applikationen laufen im User-Modus

$=>$ **keine Kernel-Protection!**

$=>$ für Protection Unterstützung des Host-OS notwendig („Kontextwechsel” zwischen Guest-OS und Guest-Applikationen)



Paravirtualisierung – Interrupts

Zum Nachbilden der Interrupts können statt dessen Linux-Signale verwendet werden:

- Programmieren der Signale: `sigaction`
- Setzen des Signal-/Supervisor-Stacks: `sigstack`
- Enable/Disable der Signale: `sigprocmask`

`sigstack` notwendig. Signal-Ursache könnte Page-Fault bei Stack-Zugriff sein!



Paravirtualisierung – Interrupts

Problem:

- Interrupts werden sehr häufig disabled/enabled.
- Aufruf des `sigprocmask`-System-Calls langsam.

Idee:

Signale immer zulassen; zugehörigen Signal-Handler aber nur aufrufen, wenn er enabled ist. Wenn Signal-Handler disabled ist, Signal als „deferred" vormerken.



Paravirtualisierung – Interrupts

```
void sig_handler() {
  if (enabled)
    sig_handler2();
  else
    deferred = 1;
}

void sig_enable() {    /* Muesste atomar sein! */
  while (deferred) {
    deferred = 0;
    sig_handler2();
  }
  enabled = 1;
}

void sig_disable() {
  enabled = 0;
}
```



Paravirtualisierung – Interrupts

Compilierte `sig_enable`-Prozedur:

```
sig_enable:
  jmp .L3
.L1:
  movl $0, deferred
  call sig_handler2
.L3:
  cmpl $1, deferred
  je .L1
  movl $1, enabled
  ret
sig_enable_end:
```



Paravirtualisierung – Interrupts

Idee: Im Interrupt-/Signal-Handler liegt `%eip` auf Stack.

`sig_enable` $<=$ `%eip` $<$ `sig_enable_end`: setze `%eip` auf `.L1`.

sonst (`enable=0`): setze `deferred` auf 1.

sonst (`enable=1`): rufe `sig_handler2()` auf.



Paravirtualisierung – Exceptions/System-Calls

Exceptions:

- Exceptions gehen standardmässig zunächst an das Host-OS.
- Sollen eigentlich an das Guest-OS (Host-Applikation) gehen.
- Normalerweise sendet ein OS Exceptions an Applikation weiter (Unix: Signal-Mechanismus)

System-Calls:

- System-Calls gehen an das Host-OS.
- Werden nicht wie Exceptions weitergeleitet.

$=>$ Spezieller Mechanismus im Host-OS notwendig!

Umleitung unter Linux mittels `ptrace`-Mechanismus (Teil der Debug-Schnittstelle) möglich (aber langsam).



Paravirtualisierung – Uhr

`gettimeofday`-System-Call des Host-OS benutzt Real-Time-Clock nur einmal beim Booten (Start-Time). Ansonsten werden Timer-Ticks gezählt (Run-Time).

Paravirtualisierung geht ähnlich vor.

Real-Time-Clock: `gettimeofday`-System-Call

Timer-Ticks: `setitimer`-System-Call

- `gettimeofday`-System-Call unproblematisch (nur einmal verwendet).
- `setitimer`-System-Call auch unproblematisch (nur ein Aufruf; danach kommen SIGALRM-Signale).



Paravirtualisierung – Uhr

`setitimer`-System-Call kann nur Signale mit einer maximalen Frequenz von 100Hz bzw. 1000Hz zustellen.

$=>$ virtuelle System-Clock ungenau.

Höhere Auflösungen aber auch nicht sinnvoll. Guest-OS kann ge-scheduled werden!

$=>$ Zeitmessungen in paravirtualisierten Systemen problematisch!

