Emulation

Dr.-Ing. Volkmar Sieh

Department Informatik 4
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander-Universität Erlangen-Nürnberg

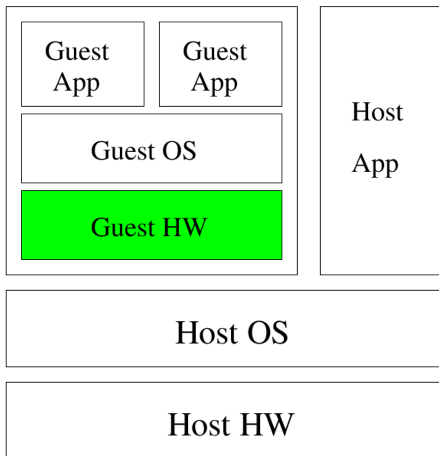
WS 2014/2015

Definition: Emulation

Wikipedia:

Als Emulation wird in der Computertechnik das funktionelle Nachbilden eines Systems durch ein anderes bezeichnet. Das nachbildende System erhält die gleichen Daten, führt die gleichen Programme aus und erzielt die gleichen Ergebnisse wie das originale System.

Emulation



Emulation

Beispiel:
System mit einem Bus mit angeschlossener CPU, RAM, ROM, einem seriellen Terminal und einer Platte.

CPU
IRQ
Bus
memory
bios_rom
disk_ctrl
serial_ctrl
Disk
Terminal

Emulation – CPU

Idee System-Simulationsschleife:

```
for (;;) {
    /* bus_step(); */
    cpu_step();
    /* ram_step(); */
    /* rom_step(); */
    ser_step();
    disk_step();
}
```

Emulation – CPU

```
cpu_step() {
    if (! power) {
        /* Do nothing... */
    } else if (ieflag && interrupt_pending) {
        cpu_exec_interrupt();
    } else {
        cpu_exec_instruction();
    }
}
```

Emulation – CPU

```
cpu_exec_instruction() {
    inst = load(ip++); /* Get Instruction */
    switch (inst) {
    case ADD: /* add $imm, %accu */
        imm = load(ip++); /* Get Op */
        accu += imm;
        break;
    case OUT: /* out %accu, port */
        port = load(ip++); /* Get Port */
        out(port, accu);
        break;
    case JMP: /* jmp addr */
        ip = load(ip); /* Get Address */
        break;
    ...
    }
}
```

Emulation – CPU

```
cpu_exec_interrupt() {
    interrupt_pending = 0;

    store(--sp, ip);
    store(--sp, ieflag);

    ieflag = 0;
    ip = load(0);
}
```

Emulation – CPU

```
cpu_interrupt() {
    interrupt_pending = 1;
}
```

Emulation – Bus

```
uint8_t load(uint32_t addr) {
    if (ROMSTART <= addr && addr < ROMSTART + ROMSIZE) {
        return rom_load(addr - ROMSTART);
    } else if (RAMSTART <= addr && addr < RAMSTART + RAMS
        return ram_load(addr - RAMSTART);
    } else {
        return 0;
    }
}

store(uint32_t addr, uint8_t val) {
    if (RAMSTART <= addr && addr < RAMSTART + RAMSIZE) {
        ram_store(addr - RAMSTART, val);
    } else {
        /* Do nothing... */
    }
}
```

Emulation – Bus

```
out(uint16_t port, uint8_t val) {
    if (SERSTART <= port && port < SERSTART + SERSIZE) {
        ser_out(port - SERSTART, val);
    } else if (DISKSTART <= port && port < DISKSTART + DI
        disk_out(port - DISKSTART, val);
    } else ...
}

uint8_t in(uint16_t port) {
    if (SERSTART <= port && port < SERSTART + SERSIZE) {
        return ser_in(port - SERSTART);
    } else if (DISKSTART <= port && port < DISKSTART + DI
        return disk_in(port - DISKSTART);
    } else ...
}
```

Emulation – Bus

```
interrupt() {
    cpu_interrupt();
}
```

Emulation – ROM

```
static const uint8_t rom[ROMSIZE] = { ... };

uint8_t rom_load(uint32_t addr) {
    addr %= ROMSIZE;
    return rom[addr];
}
```

Emulation – RAM

```
static uint8_t mem[RAMSIZE];

uint8_t ram_load(uint32_t addr) {
    addr %= RAMSIZE;
    return mem[addr];
}

ram_store(uint32_t addr, uint8_t val) {
    addr %= RAMSIZE;
    mem[addr] = val;
}
```

Emulation – serielle Schnittstelle

```
ser_out(uint16_t port, uint8_t val) {
    host_print_char(val);
}
```

Emulation – serielle Schnittstelle

```
static uint8_t key, avail;

uint8_t ser_in(uint16_t port) {
    switch (port) {
    case 0:
        avail = 0;
        return key;
    case 1:
        return avail;
    }
}
ser_step() {
    if (host_key_avail()) {
        key = host_get_key();
        avail = 1;
        interrupt();
    }
}
```

Emulation – Disk

```
static uint8_t disk[SIZE], buf, pos, status;

disk_out(uint16_t port, uint8_t val) {
    switch (port) {
    case 0: buf = val; break;
    case 1: pos = val; break;
    case 2: if (pos < SIZE) {
                if (val) disk[pos] = buf;
                else buf = disk[pos];
                status = OK;
            } else {
                status = BAD;
            }
            break;
    }
}
```

Emulation – Disk

```
uint8_t disk_in(uint16_t port) {
    switch (port) {
    case 0: val = buf; break;
    case 1: val = pos; break;
    case 2: val = status; break;
    }
    return val;
}
```

Emulation – MMU

Emulation der MMU muss *sehr* effizient sein:

- jeder Code-Zugriff braucht MMU
- jeder Daten-Zugriff braucht MMU
  - ca. 20% der Befehle lesen/schreiben Speicher
  - Grafikaufbau i.A. über `mov`-Befehle

Emulation – MMU

Aufgabe der MMU:

- Suchen der virtuellen Adresse im Translation-Lookaside-Buffer (TLB)
- nicht gefunden:
  - Umrechnung der virtuellen Adresse in eine physikalische durch Lookup in Tabellen im Hauptspeicher
  - Eintragen der virtuellen und der dazugehörigen physikalischen Adresse im TLB
- Weitergabe der zur virtuellen Adresse gehörenden physikalischen Adresse an den Hauptspeicher

Emulation – MMU

```
uint8_t virt_load(uint32_t vaddr) {
    uint32_t paddr;

again:
    found = tlb_lookup(vaddr & ~0xfff, &paddr);
    if (! found) {
        tlb_fill(vaddr & ~0xfff);
        goto again;
    }

    return phys_load(paddr | (vaddr & 0xfff));
}
```

`tlb_lookup` sollte blitzschnell ausführbar sein und nahezu immer `TRUE` zurückliefern!

Emulation – MMU

```
struct {
    uint32_t vaddr, paddr;
} tlb[SIZE];

int tlb_lookup(uint32_t vaddr, uint32_t *paddrp) {
    unsigned int hash = (vaddr >> 12) % SIZE;

    if (vaddr == tlb[hash].vaddr) {
        *paddrp = tlb[hash].paddr;
        return 1;
    } else {
        return 0;
    }
}
```

Entspricht einem Direct-Mapped Cache.