hhuOS

Christian Gesse, Fabian Ruhland

13.07.2023

Institute of Computer Science
Heinrich Heine University Düsseldorf

Facts about hhuOS

- A small operating system for teaching and learning purposes
- written for x86 32-bit architecture
- written in C++ and x86-Assembler using gcc and nasm
- Open-Source, published under the GPL v3 license
- ~ 43000 lines of code

hhuOS - Features

- Processes & Threads
  - Round-Robin based preemptive scheduling
  - Binary files are executed as processes
  - Each process has its own address space

hhuOS - Features

- Processes & Threads
  - Round-Robin based preemptive scheduling
  - Binary files are executed as processes
  - Each process has its own address space
- Adress spaces and memory management
  - Using x86-Paging mechanism (virtual/physical memory)
  - Lazy mapping implementation
  - Kernel address space is always mapped at 3 GiB (not accessible from user space)
  - Different memory managing algorithms are implemented (Free List, Bitmap, Table)

hhuOS - Features

- Unified Library
  - Single codebase for user- and kernel-space library
  - Functions requiring kernel access are outsourced into an interface (implemented two times - for kernel and user space)
  - Kernel access from user space via system calls (software interrupts)

hhuOS - Features

- Unified Library
  - Single codebase for user- and kernel-space library
  - Functions requiring kernel access are outsourced into an interface (implemented two times - for kernel and user space)
  - Kernel access from user space via system calls (software interrupts)
- Further features
  - Virtual Filesystem (can mount physical filesystems)
  - Hardware support
    - Basic I/O: Keyboard & Mouse, Serial & Parallel Ports, VESA & CGA graphics, PCI & ISA bus, PC Speaker
    - Interrupt: Programmable Interrupt Controller (PIC), APIC (bachelor thesis by Christoph Urlacher)
    - Time: Programmable Interval Timer, Real Time Clock
    - Storage: Floppy, IDE
    - Network: Realtek RTL8139 (bachelor thesis by Alexander Hansen)
  - Multiboot2 compatible: Bootable on UEFI and BIOS systems
  - Own UEFI bootloader (towboot, developed by Niklas Sombert)
  - UDP/IP-stack (based on bachelor thesis by Hannes Feil)

Overview: Memory & Paging

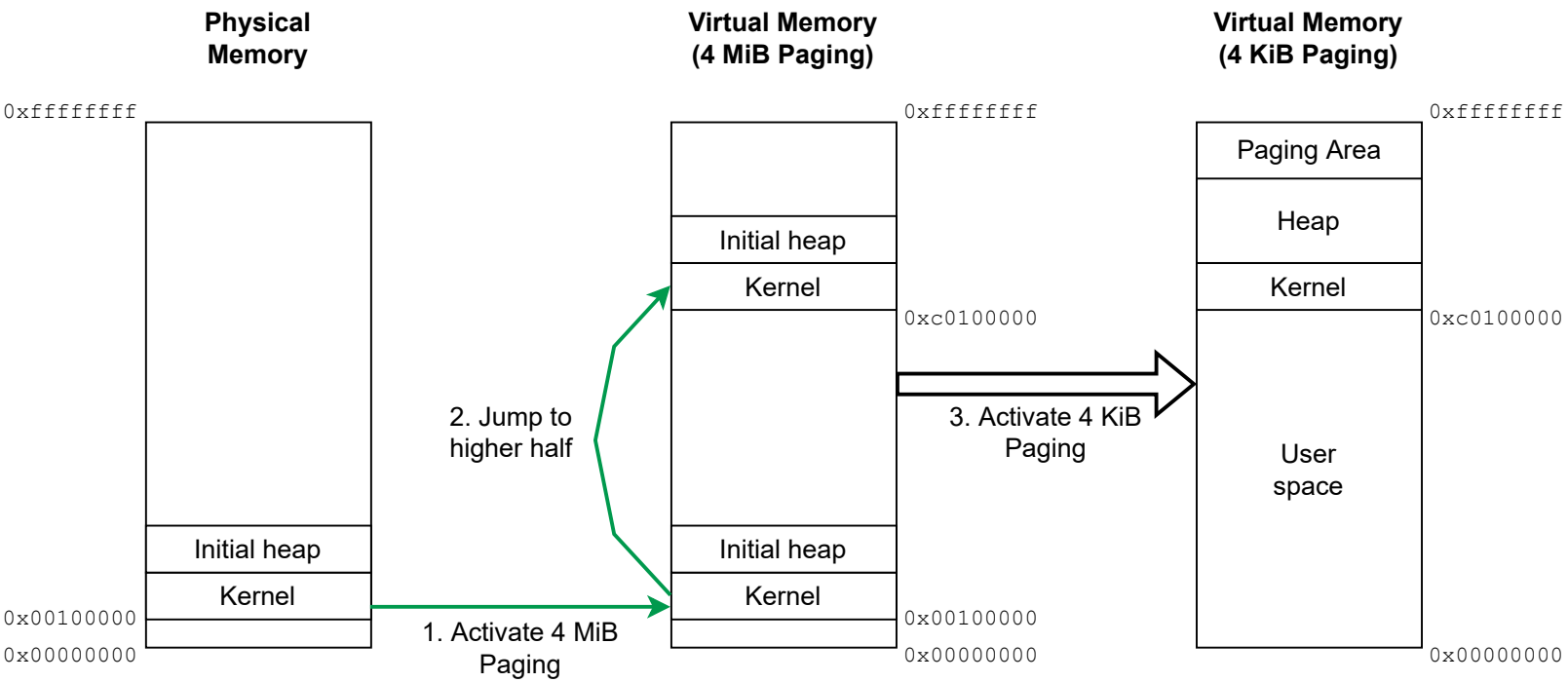
- Paging: Abstract physical memory from virtual address spaces
- New pages can be mapped in/out dynamically
- Use of different address spaces for process separation
- Kernel is mapped at 3 GiB $\rightarrow$ Higher-Half-Kernel (always visible)
- Adresses below 3 GiB are used for user space memory

Paging: The boot process

- How to allocate memory when no memory manager is available?
- How to map the Kernel-code at 3 GiB without losing the EIP?
- Solution: Activate paging in several steps

Paging: The boot process

- How to allocate memory when no memory manager is available?
- How to map the Kernel-code at 3 GiB without losing the EIP?
- Solution: Activate paging in several steps



Allocating heap memory

1. Invoke Free list manager to find free memory block in heap
2. Slice free block (with respect to alignment)
3. Return pointer to found block
4. If the block was not previously mapped, the first access will generate a pagefault
5. Fault handler is invoked during interrupt handling
6. Search unused pageframe and map it to the virtual fault address
7. Return to program

Process Scheduling

- Global scheduler manages threads in a queue
- Timer invokes scheduler every 10ms to switch to next thread
  - If current and next threads belong to different processes, address space needs to be switched (flush TLB)
- Threads can be blocked and unblocked by other threads
- Threads can sleep for a defined time
  - Scheduler manages sleeping threads in a separate queue
- It is possible to join other threads (or processes)
  - Each thread has a list of joining threads, that are reassigned to the scheduler, once the thread has finished

Storage Devices

- *StorageDevice* as interface for block devices
- Only 4 methods: *getSectorCount()*, *getSectorSize()*, *read()*, *write()*
- Implemented for IDE (hard drives), Floppy, Virtual Drives, and Partitions

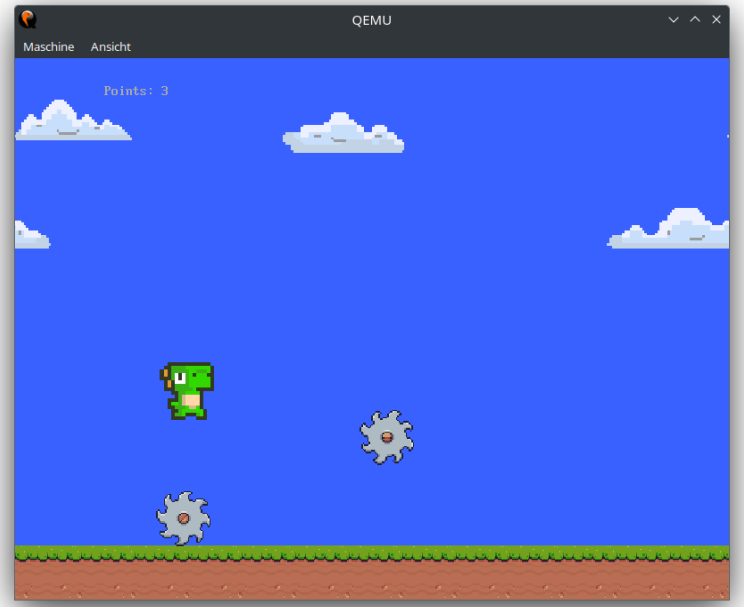
Virtual File System

- Overlay over physical file systems (e.g. FAT)
- Storage devices can be mounted to any folder
- Files and directories are represented by `Filesystem::Node` (similar to *INode*)
- Interface `Filesystem::Driver` for physical file systems
- Implemented using *FatFs*[1] to support all FAT variants

[1] `http://elm-chan.org/fsw/ff/00index_e.html`

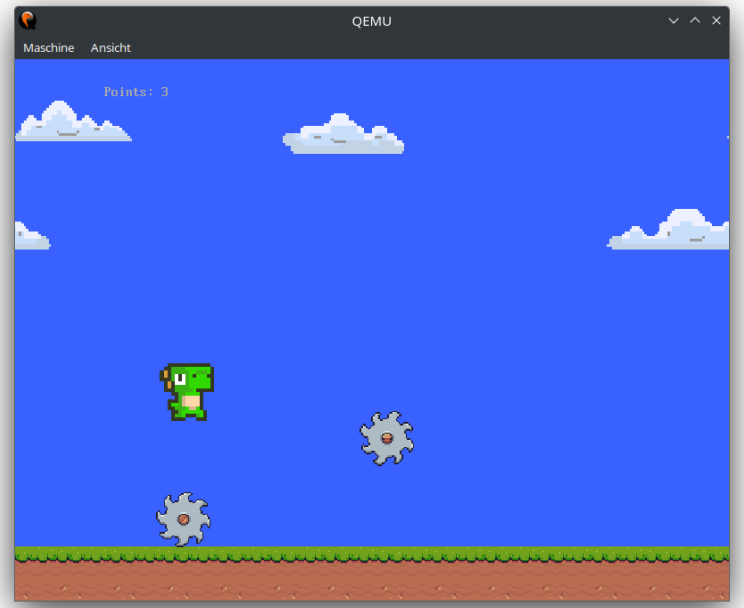
2D games

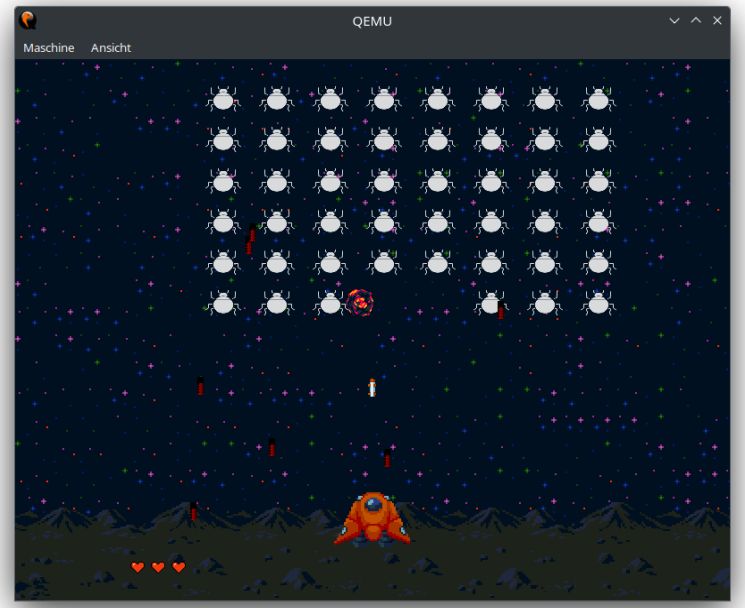
- 2D game library implemented by Malte Sehmer (bachelor thesis)
- Supports sprites (loaded from bitmap files) and animations
- Collision detection based on rectangle colliders (ploygon colliders are conceptually implemented)
- Linear and accelerated movements are implemented (e.g. gravity)



2D games

- 2D game library implemented by Malte Sehmer (bachelor thesis)
- Supports sprites (loaded from bitmap files) and animations
- Collision detection based on rectangle colliders (ploygon colliders are conceptually implemented)
- Linear and accelerated movements are implemented (e.g. gravity)

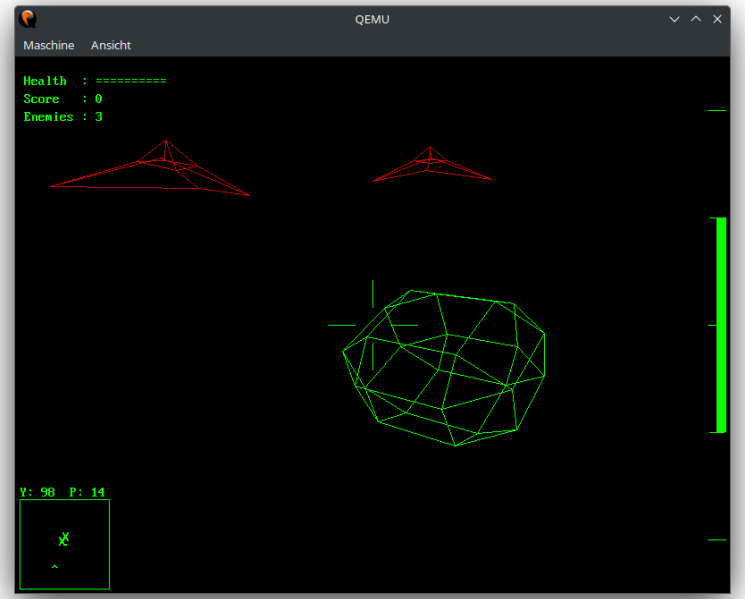


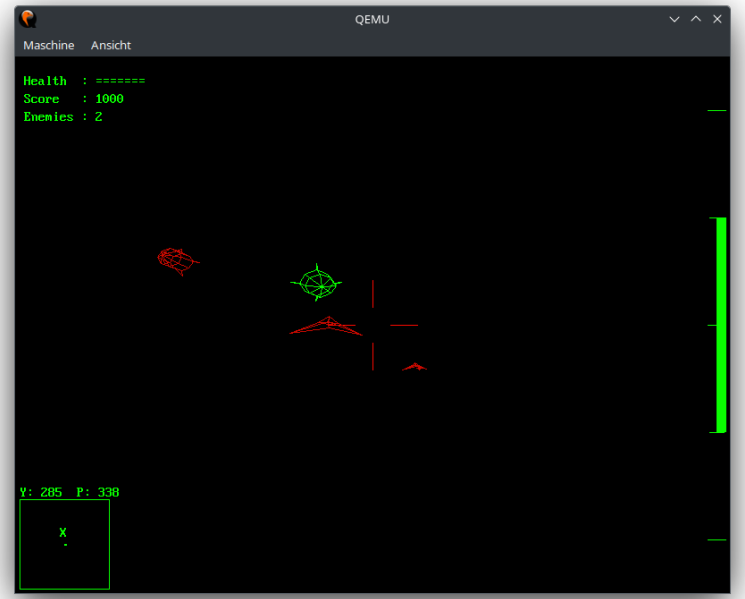


But can it run Crysis?

3D games - Work in Progress

- 3D game library implemented by Richard Schweitzer (bachelor thesis)
- Supports wireframe 3D-objects loaded from text files
- Collision detection based on spheres





Future Work

- Implement more device drivers (sound, graphics, network, etc.)
- Implement new library functions (e.g. C standard library)
- Enhance network stack (TCP)
- Better scheduling (priorities, I/O management)
- New (custom) filesystems
- Multicore support (Work in Progress)
- Support modern x86-Features (Physical Address Extensions, Long mode, HPET, etc.)