Burak Akgül, Christian Gesse, Filip Krakowski, Fabian Ruhland

Institute of Computer Science
Heinrich-Heine-University Düsseldorf

18. Juli 2018
Introduction
• A small operating system for teaching and learning purposes
• written for x86 32-bit architecture (64-bit maybe later)
• written in C++ and x86-Assembler using g++ and nasm
• Open-Source, published under the GPL v3 license
- Round-robin based preemptive scheduling for threads
- Support for AHCI, USB (partially), PCI and VESA-Graphics
- Different memory managers
- Paging with higher half kernel
- FAT- Filesystem and VFS
Memory & Paging
Overview: Memory & Paging

- Paging is used to abstract physical memory from virtual address spaces.
- New pages can be mapped in/out dynamically.
- It is possible to create different address spaces.
- Kernel is mapped at 3GB → Higher-Half-Kernel.
- The addresses above 3GB are mapped into all address spaces as Kernelspace.
- Everything below 3GB can be used for usermode later.
Difficulties implementing paging

The bootstrapping process:

- How to allocate memory when no memory manager is available?
- How to map the Kernel-code at 3GB without losing the EIP?
- Solution: activate paging in three steps
  1. First: Create a rough basic mapping for important areas using 4MB paging
  2. Then initialize all important memory managers and the page frame allocator
  3. Set up the first 4KB-Pagedirectory and reload CR3 register
Invoking BIOS-calls:

- BIOS-calls are necessary to set up VESA-Graphics
- BUT: BIOS-calls run in 16-bit mode without paging
- every BIOS-call would crash immediately if used with Higher-Half Kernel
- Solution:
  1. Switch to a simple 4MB-Pagedirectory that maps the Kernel to low addresses
  2. jump down to low addresses with EIP and switch to 16-bit mode without paging
  3. After returning from BIOS-call restore the old state
Filesystem
• **StorageDevice** as interface for block devices
• Only 5 methods: `getSectorCount()`, `getSectorSize()`, `read()`, `write()`, `getHardwareName()`
• Implemented for AHCI, EHCI, Floppy, Virtual Drives, and Partitions
• File in `/dev` for every device (similar to Linux)
Virtual File System

- Overlay over physical file systems
- Storage devices can be mounted to any folder
- Interface FsDriver for physical file systems
- Currently, only FAT is supported as physical file system
Kernel Features
hhuOS includes some utility classes to ease implementing new features

Array  ArrayList  LinkedList  HashMap

HashSet  BlockingQueue  RingBuffer

Each utility class can be used in conjunction with any type by using template parameters
Array copies can be created using a simple assignment

```cpp
Util::Array<uint32_t> a = {1, 2};
Util::Array<uint32_t> b = a;
a[1] = 3;
printf("a[0]=%d , a[1]=%d\n", a[0], a[1]);
printf("b[0]=%d , b[1]=%d\n", b[0], b[1]);
```

```
a[0]=1 , a[1]=3
b[0]=1 , b[1]=2
```
Utility Classes : HashMaps

HashMaps can be used to store key-value pairs with any type

code/HashMaps.cpp

```
Util::HashMap<String, uint32_t> hashMap = 
    {{ "startup", 0xC0100020}, {"main", 0xC014F7C8}};

printf("Function 'main' is at 0x%08x\n", hashMap.get("main"));
```

Function 'main' is at 0xC014F7C8
hhuOS’ functionality can be extended using kernel modules

code/Hello.h

```cpp
#include <kernel/Module.h>

class Hello : public Module {
 public:
   Hello() = default;
   int32_t initialize() override;
   int32_t finalize() override;
   String getName() override;
   Util::Array<String> getDependencies() override;
};
```

Each module needs to inherit from the Module class
Kernel Modules

hhuOS’s functionality can be extended using kernel modules

In this example, the Hello module prints Hello hhuOS! on initialization
hhuOS supports loading kernel modules at runtime

```cpp
auto moduleLoader = Kernel::getService<ModuleLoader>();
auto file = File::open("/mod/hello.ko", "r");
moduleLoader->load(file);
```

Hello hhuOS!

Compiled modules can be placed on an external storage device
Bluescreen / Stacktrace

If something goes wrong, hhuOS prints out a bluescreen containing useful information such as a stacktrace

```
[PANIC] IndexOutOfBoundsException

#00 0xC011C2C6 --- _ZN3Cpu14throwExceptionENS_9ExceptionE
#01 0xC011C2C6 --- _ZN3Cpu14throwExceptionENS_9ExceptionE
#02 0xC016C6FA --- _ZN4Util5ArrayIJjEixEj
#03 0xC014F805 --- badArrayAccess
#04 0xC014F82B --- main

eax=0x00000001 ebx=0x000000C010 ecx=0xC01BAFDC edx=0xC043E404
esp=0xC01BAF54 ebp=0xC01BAF68 esi=0x00000000 edi=0xC01AA078

eflags=0x00200096
```
Future Work
Future Work

- Implement more device drivers (sound, graphics card, etc.)
- Ring protection / user mode
- Process system
- Enhanced scheduling (priorities, I/O management)
- New (custom) filesystems
- Multicore support
Demo

16.07.2010 15:26:30

hhuOS main menu

Shell

- Bug Defender
- Asciiart
- Langtons Ant
- Mandelbrot
- Mouse
- Loops and Sound
- Heap Demo
- 10 Memory Demo
- Exception Demo

A simple UNIX-like Shell
Please select an option using the arrow keys and confirm your selection using the space key.