



# hhuOS

---

Burak Akguel, 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

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



## 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

code/Arrays.cpp

C++

```
1 Util::Array<uint32_t> a = {1, 2};  
2  
3 Util::Array<uint32_t> b = a;  
4  
5 a[1] = 3;  
6  
7 printf("a[0]=%d , a[1]=%d\n", a[0], a[1]);  
8  
9 printf("b[0]=%d , b[1]=%d\n", b[0], b[1]);
```

```
a[0]=1 , a[1]=3  
b[0]=1 , b[1]=2
```

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

code/HashMaps.cpp

C++

```
1 Util::HashMap<String, uint32_t> hashMap =  
2     {{ "startup", 0xC0100020}, {"main", 0xC014F7C8}};  
3  
4 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

C++

```
1  #include <kernel/Module.h>
2
3  class Hello : public Module {
4  public:
5      Hello() = default;
6      int32_t initialize() override;
7      int32_t finalize() override;
8      String getName() override;
9      Util::Array<String> getDependencies() override;
10 };
```

Each module needs to inherit from the `Module` class

hhuOS's functionality can be extended using kernel modules

code/Hello.cpp

C++

```
1 #include "Hello.h"
2 #include "lib/libc/printf.h"
3
4 MODULE_PROVIDER { return new Hello(); };
5 int32_t Hello::initialize() { printf("Hello hhuOS!\n"); return 0; }
6 int32_t Hello::finalize() { printf("Bye hhuOS!\n"); return 0; }
7 String Hello::getName() { return "hello"; }
8 Util::Array<String> Hello::getDependencies() { return Util::Array<String>(0); }
```

In this example, the Hello module prints Hello hhuOS! on initialization

hhuOS supports loading kernel modules at runtime

code/Modules.cpp

C++

```
1 auto moduleLoader = Kernel::getService<ModuleLoader>();  
2  
3 auto file = File::open("/mod/hello.ko", "r");  
4  
5 moduleLoader->load(file);
```

Hello hhuOS!

Compiled modules can be placed on an external storage device

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

```
[PANIC] IndexOutOfBounds Exception
```

```
#00 0xC011C2C6 --- _ZN3Cpu14throwExceptionENS_9ExceptionE
#01 0xC011C2C6 --- _ZN3Cpu14throwExceptionENS_9ExceptionE
#02 0xC016C6FA --- _ZN4Util5ArrayIJEixEj
#03 0xC014F805 --- badArrayAccess
#04 0xC014F82B --- main
```

```
eax=0x00000001 ebx=0x0000C010 ecx=0xC01BAFDC edx=0xC043E404
esp=0xC01BAF54 ebp=0xC01BAF68 esi=0x00000000 edi=0xC01AA078

eflags=0x00200096
```



## **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

**Fragen**