mikro - Introducing a C++ Mikro Kernel

Victor Apercé & Julien Freche

viaxxx@lse.epita.fr
julien.freche@lse.epita.fr
http://lse.epita.fr/

Outline I

Introduction

mikro id

- Kernel type: micro kernel
- Status: experimental now but intended to be in production (if we can :D)
- Language: C++
- Fathers: Victor Apercé, Julien Freche
- Birth: in early September 2013
- Place of birth: LSE, near Paris, France

Some figures about mikro

- The most active project of the LSE this year
- 2 repositories: kernel and User land
- 2 main authors and 2 contributors
- ~900 commits representing 5.2 commits per day
- ~27,000 lines

Kernel types

Introduction to kernel

Kernel land/User land

Kernel land

Code running at a privileged level of the CPU

Bugs are most of the time fatal

User land

Tasks running at a low privileged level

Bugs can be recovered by Kernel land

Kernel

Interface between software and hardware

- A part is running in Kernel land but not necessary all
- Core of the operating system
- 2 main types: monolithic and micro kernel

Main component of Kernel

Main components of kernel are:

- Paging (will be discussed in an other talk)
- VFS: Virtual File System
- Binary loader
- Scheduler
- Processor init (will be discussed in an other talk)
- Drivers

Monolithic/Micro kernel explained

Monolithic kernel

Almost everything privileged is in Kernel land

- Availability of dynamic module loading - most of the time
- Very big
- System functionality access: through system calls

Micro kernel

The less as possible resides in Kernel land

- No dynamic module loading
- Very light
- Kernel components are User land processes: services
- System functionality access: through IPC

Inter Process Communication

Communication between 2 User land processes

- Message passing in micro kernels
- Will be discussed in an other talk

Component location

Components	Micro kernel	Monolithic kernel
Paging	Kernel & User	Kernel land
VFS	User	
Binary Loader	User	
Scheduler	Kernel or User	
Processor Init	Kernel	
Drivers	User	

Examples

Monolithic kernels

- Windows (hybrid :D)
- Darwin, MacOS X kernel (hybrid :D)
- Linux
- *BSD
- STOS

Micro kernels

- QNX
- Mach
- L4 family, the reference

Monolithic/Micro kernel comparison

Complexity

Monolithic kernel

- Very complex
- Hard to maintain
- Kernel must be reentrant

Micro kernel

- Kernel land is simple
- User land is hard to design and complex
- Supposed to be easier to maintain
- Kernel don’t need to be reentrant

Security

Monolithic kernel

- Kernel land is big so many possible bugs
- Possibility to exploit dynamic code loading in Kernel land
- Bugs are most of the time fatal

Micro kernel

- Kernel land is simple so less possible bugs
- Bugs in User land are not fatal
- Still possible exploits in User land services
- Security can be mathematically proved like seL4

Performance

Monolithic kernel

Very fast due to system calls

Micro kernel

- Slower because of IPC context switches
- IPC must be as fast as possible to improve performance
- This problem hasn't been really solved yet

Knowledge background

Monolithic kernel

- First kernels
- Design used a lot
- Less fun, recipe already exists

Micro kernel

- Rare design
- Still researches on the topic
- But let room to innovations!

mikro

Why's mikro a micro kernel?

- More challenging
- Pretty fan of the micro kernel idea
- More fun because there's no recipe

What makes mikro different?

- Not a L4 clone but inspired from L4
- We want it to be as fast as we can
- IPC will be different from other micro kernel (see other talk)
- mikro code has been designed to be "one day" in production
- Developed with newer technologies:
  - clang is the default compiler
  - C++ is the official language of mikro

Low level C++

Common question:

Is it possible to code a kernel in C++ ?

Answer:

Yes, it is, but you will have to drop some features:

- RTTI: Run-Time Type Information
- STL: Standard Template Library
- Local static variables
- Global objects
- New and delete operators
- ...

RTTI

C++ mechanism that gives informations about an object's data type at runtime.

- This is useful for *dynamic_cast*<> and *typeid* operators.
- Used when handling exceptions.
- Virtual functions work without RTTI (using vtables).

You can port a C++ RunTime lib and an Unwind lib to restore these features.

STL

Library that provides generic templated classes and associated algorithms.

- Depends on the libc
- No stream operators
- Some of the containers may be usable if extracted from the lib.

You can port libstdc++, STLPort or uSTL for example to restore these features.

Local static variables

These variables are declared as static inside a function and are preserved across different calls.

The compiler will generate a code that will looks like:

```
__guard guard;

if (!((char*)&guard)[0])
  if (__cxa_guard_acquire (&guard))
  {
    // Effective variable init
    __cxa_guard_release (&guard);
  }
```

This code is, of course, simpler that the real generated code.
You can code __cxa_guard_* to support this feature.

Init me

Constructor of global objects have to be called before entering the C++ entry point. You can do it with the following:

```
    mov    $start_ctors, %ebx
    jmp    2f
1:
    call   *(%ebx)
    add    $4, %ebx
2:
    cmp    $end_ctors, %ebx
    jb     1b

    call   k_main
```

The symbols *start_ctors* and *end_ctors* are generated by the compiler. You can call destructors with a similar code.

No throw, please !

You will have to code new and delete operators by yourself. The usual new operator will throw an exception on error but you probably don't want that.

```
void *operator new(size_t size) noexcept;
void operator delete(void *p);
```

- The new operator will have to call your kernel internal allocator.
- Do not forget to *check* the return value.
- Do not use new in a constructor, you cannot check the return value.

Design considerations

Pick one

How to separate generic code from arch-specific code ?

- Using inheritance ?
- Using templates ?
- Using macro ?

Not so good..

Inheritance

- Parent Class: arch generic
- Inherited Class: arch specific

Considerations:

- First naive idea
- You can change program behavior at run-time. Useless here.
- Overhead due to vtables and pointer manipulation.
- Pay attention to not compile useless code.

Not so bad..

Macro

- Macro name: arch generic
- Expanded name: arch specific

Considerations:

- Less readable. You have to find the macro definition to understand.
- C style.
- Pretty powerful.

First try

Templates

- Base template: arch generic
- Specialized template: arch specific

Considerations:

- Some code has to be in the header file.
- Specialized templates do not receive methods from the base template.

You cannot easily write an arch generic interface...

Hybrid

The solution implemented in mikro is hybrid and inspired by the PIMPL design pattern.

```
class MemoryManager
{
    public:
      inline void init()
      {
        arch.init();
      }

      MemoryManagerImpl<ARCH_GENERIC> arch;
};
```

This solution is not perfect but it works and can also be used for other things like changing scheduler at compile time.

Features and progress

Implemented features

Awesome

Kernel:

- Paging
- SMP
- IPC
- vm86

Userland:

- minimal libc
- mikro lib
- vesa

Bootloader:

- module loading
- ext2 support
- configuration file

Of course it works !

Tested on:

- qemu
- bochs
- VirtualBox
- Real hardware

You can easily create a USB stick with mikro and test it on your hardware.

Missing features

Not my fault

Kernel:

- Support x86_64 and armv7
- Better scheduler
- Time management
- Improve IPC
- Probably a lot of bugs to fix

Userland:

- VFS
- Paging daemon
- Process creation in userland
- Drivers

Let’s dream

Other features we want to add if we can:

- Linux compatible driver API
- Metadata oriented file system "datameat" port on mikro
- Unix program ports
- And much more...

Conclusion

Almost done

mikro

- highly active project
- very fun to code !
- a good way to learn new stuff
- not usable yet but mainly because of the userland

I want YOU!

You can contribute to the project.

- Main page: `http://www.lse.epita.fr/projects/mikro.html`
- Kernel repo (lse): `http://git.lse.epita.fr/?p=mikro.git`
- Kernel repo (bb): `http://bitbucket.org/mikroteam/mikro`
- Userland repo (lse): `http://git.lse.epita.fr/?p=mikro-userland.git`
- Userland repo (bb): `http://bitbucket.org/mikroteam/mikro-userland`

Contacts

Julien Freche

- julien.freche@lse.epita.fr
- @JulienFreche

Victor Apercé

- viaxxx@lse.epita.fr

Mailing List:

- mikro@lse.epita.fr

Feel free to contact us if you have any questions about the project. We will be happy to answer.

Thank you for your attention