STOS, what's new?

Gabriel Laskar <gabriel@lse.epita.fr>
Jérémy Lefaure <blatinox@lse.epita.fr>

LSE Security System

What is STOS?

- A toy operating system
- Modular
- A teaching & experimenting tool

LSE Security System

Main principles of the STOS kernel

- Monolithic kernel
- Modular architecture
- (probably) Multi-architecture
- Simple.

LSE Security System

Emphasis on simple

- No surprises
- When in doubt, mimic the linux kernel APIs.
- Still some differences with a classic unix kernel

mkdir(3)

```c
/*
 * In stos, mkdir can be emulated with open,
 * so use it instead of creating another syscall.
 */
int mkdir(const char* pathname, mode_t mode)
{
    int fd = open(pathname, O_CREAT | O_DIRECTORY | O_EXCL, mode);
    if (fd < 0)
        return fd;
    close(fd);
    return 0;
}
```

LSE Security System

signals

- no signals planned in stos
- use something like `signalfd(2)`
- have 4 file descriptors opened by default
- still not implemented, we need to have:
  - `poll(2)`
  - threads in userland

LSE Security System

mknod()

- devfs is populated by the kernel
- every instance of a driver are in separate directory:
  - com devices (serial lines) lives in “`/dev/com/[1-4]`” for pc

STOS is now used for a kernel course!

LSE Security System

What have been done this year

- initramfs (Paul Hervot)
- command line (Paul Hervot)
- testing modules (Louis Feuvrier & Gabriel Laskar)
- porting stos to other architectures
  - arm (Jérémy Lefaure)
  - galileo (GISTRE)
- acpi (Gabriel Laskar)
- virtio (Nahim El Atmani)

LSE Security System

Testing the STOS kernel

- We now have a simple test module
- Some tests have been done (essentially for the course)

```
/**
 * register_gtest - register a global test to be executed by the test suite
 * later on in the boot process.
 *
 * @name: name of the test (will be copied)
 * @test: test function
 * @data: data to be passed to test function at execution
 * @failure: failure function to be executed when the test fails.
 */
int register_gtest(const char *name, void (*test)(struct gtest*), void *data,
     void (*failure)(struct gtest*));
```

LSE Security System

STOS on ARM

- Theoretically multi-architecture: PowerPC, Sparc64, ARM, x86 and x86_64
- Only x86 is really maintained
- Learning about kernel programming and ARM architecture

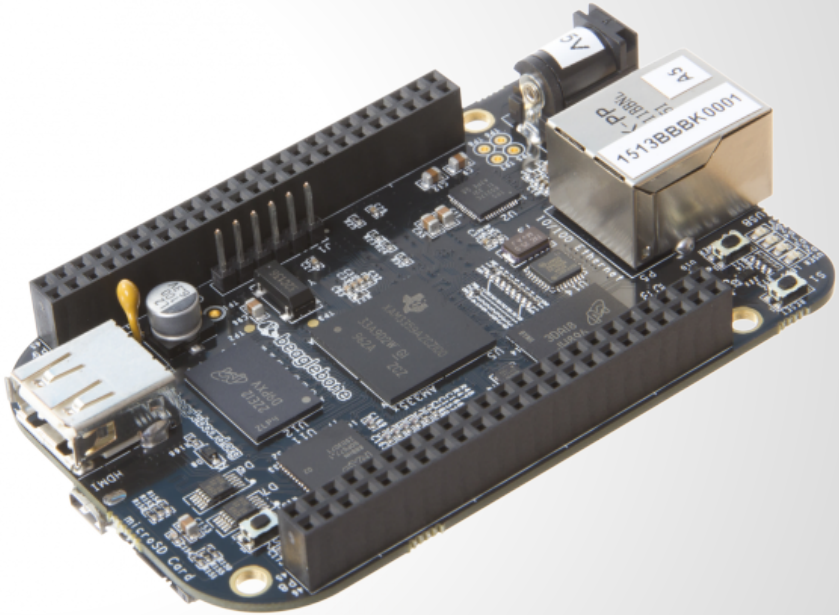
LSE Security System

Kernel development on ARM

- A lot of different versions
- Different features
- Each board has its own memory map
- Linux uses device tree

BeagleBone Black

- Processor: AM3358
- ARM Cortex-A8
- 512MB SDRAM
- Cheap (~55$)



LSE Security System

Build system

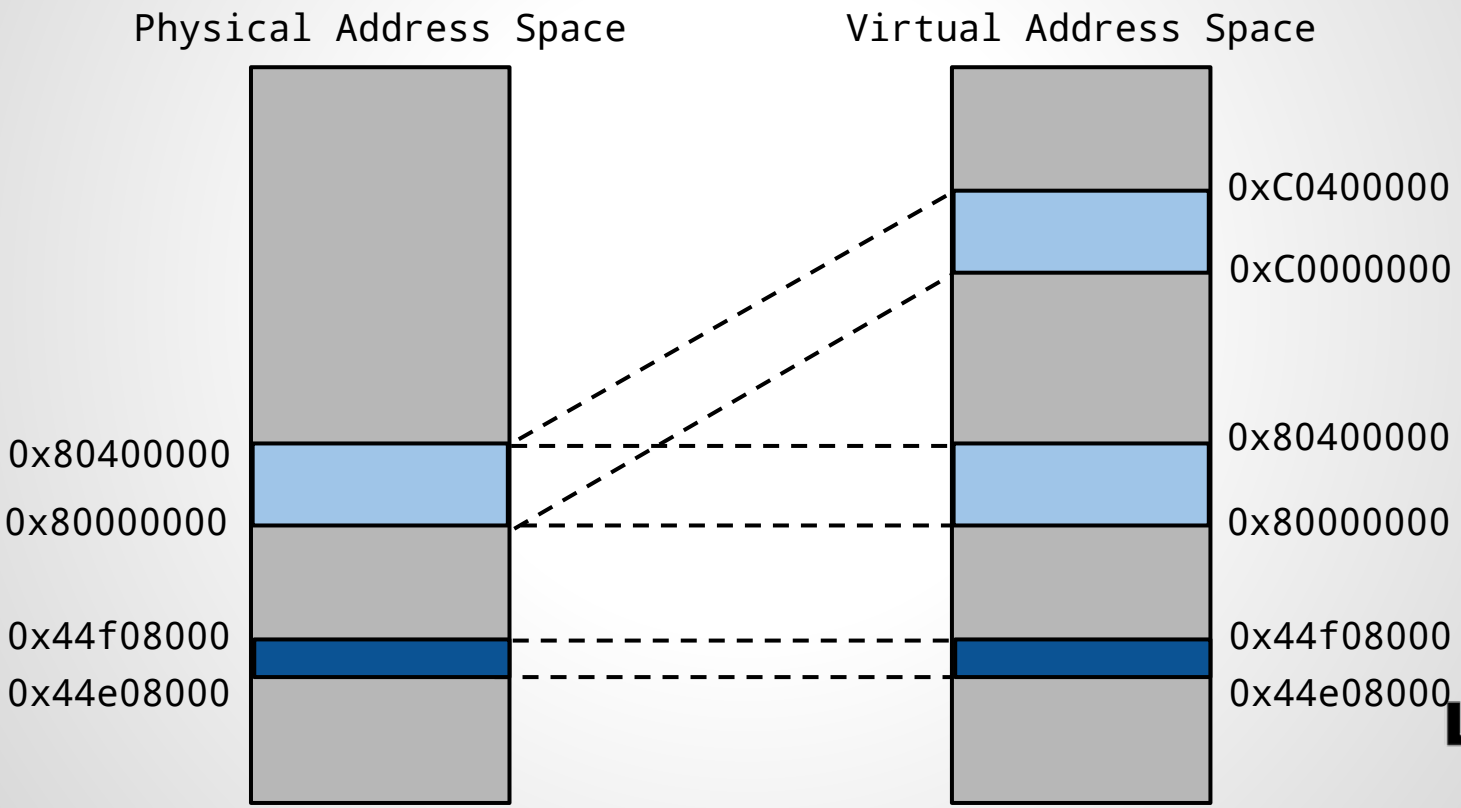
- Quite complex
- Used only for x86 target for a long time
- ARM Toolchain
- All modules in kernel mode
- Bootloader (stage 3) and kernel in one file

Stage 3

- u-boot => stage 3 => STOS core
- standalone program
- provides memory segments
- load kernel core (ELF)
- enable pagination

LSE Security System

Current paging state



Core

- Most of arch independant code (klog,...)
- Linker script
- Serial console for klog
- Backtrace

Interrupts

- Reuse old ARM code
- Fix code
- Simple API to add or remove handlers
- Should be different depending on SoC

Pagination (WIP)

- Reuse the frame allocator
- Like i386 pagination
- Arch-dependant code to write (i.e invalidate page)
- Snippets exist

TODO

- Jump to userland
- Drivers (GPIO, I2C,...)
- Other boards?

LSE Security System

Intel Galileo

LSE Security System

Goals

- PFE for GISTRE students
  - Zackary Ayoun
  - Matthieu Simon
- EFI support
- Support for the Quark x1000 SOC
- Consolidation of the STOS kernel

LSE Security System

How to reboot an OS?

- triple fault
- keyboard
- strange ioport in some bios
- apm
- acpi

Multiple usage for ACPI in STOS

- Discover devices
  - rewrite the device tree with it
  - use it to discover PNP devices
- Sleep states
  - reboot, duh !

What is still needed

- kernel thread apis
- better dependency handling
- kernel and userland timing apis
- rework device apis
- rework smp support
- poll()
- mmap() with files