Gambit on bare-metal

Samuel Yvon


console
Gambit v4.9.3
> (directory-files)
("FIB.SCM" "FACT.SCM" "GAMBIN~1.SCM")
>
log
Page Attribute Table
36 bit Page Size Extensions
Processor Serial Number
MultiMedia Extensions
FXSAVE and FXRSTOR
SSE instructions
Loading up disks...
Loading up IDE controllers...
Setting up IDE controller no: 0
Resetting the IDE...Reset of device
0
Reset of device 1
Reset of device 0
Reset of device 1
Word 47 hi (should be 128) = 128
Maximum number of sectors that sh
all be transferred per interrupt on
READ/WRITE MULTIPLE commands = 16
Setting up IDE controller no: 1
Resetting the IDE...Reset of device
0
Reset of device 1
Reset of device 0
Reset of device 1
Reset of device 0
Reset of device 1
Reset of device 0
Reset of device 1
Reset of device 0
Reset of device 1
Reset of device 0
Reset of device 1
Reset of device 0
Reset of device 1
Reset of device 0
Reset of device 1
Reset of device 0
Reset of device 1
Reset of device 0
Reset of device 1
Reset of device 0
Reset of device 1
Reset of device 0
Reset of device 1
Reset of device 0
Reset of device 1
Reset of device 0
Reset of device 1
Reset of device 0
Reset of device 1
Reset of device 0
Reset of device 1
Word 47 hi (should be 128) = 0
Maximum number of sectors that sh
all be transferred per interrupt on
READ/WRITE MULTIPLE commands = 0
Setting up IDE controller no: 2
Setting up IDE controller no: 3
Loading up the virtual file system.
..
Unknown partition type: 0
Unknown partition type: 0
Enabling FS2
Enabling the terminal STDOUT bridge
Loading up LIBC
Starting /dsk1/gambit/bin/gsi
The len of the gambit file is: 3851
184
Gambit file loaded...
Running program thread
SAMSUNG
SyncMaster 191T

Introduction

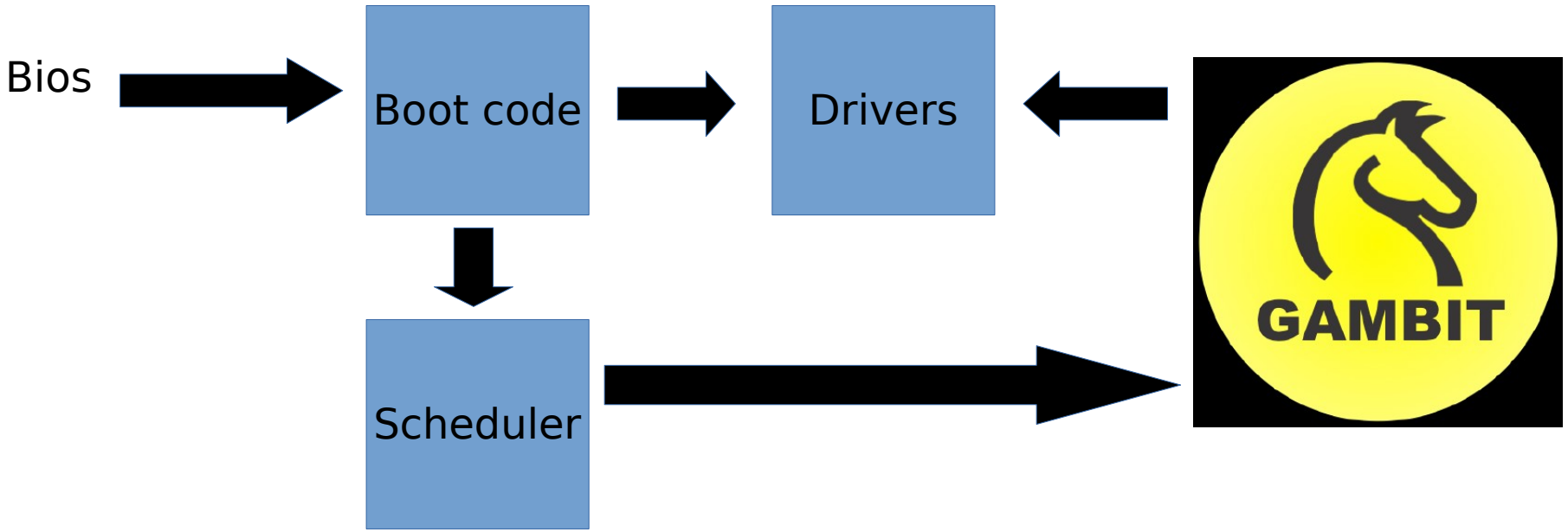
- **Surely, you can't run a compiler on bare-metal?**
  - Yes you can!
    - But why?
- **What has been done:**
  - Minimal x86 operating system designed solely to run Gambit

- X86-32 operating system
- Driver for IDE, Serial, VGA, PS2
- VFAT, VFS
- Multithreading and synchronization primitives

Motivation

- **Offers little to no overhead**
  - Performance of the generated code, not the generated code + system
- **Offers a view into kernel development with high level languages**
  - Obviously towards a Gambit Scheme implementation
  - Good base to compare a Scheme implementation with the equivalent in C (performance, effort)

Structure of *Mimosa*



Structure of *Mimosa* (cont.)

```
program_thread* task =
    CAST(program_thread*, kmalloc(sizeof(program_thread)));
new_program_thread(task, "/dsk1/home/sam/", CAST(libc_startup_fn, code), "Gambit");
thread_start(CAST(thread*, task));
```

```
program_thread* new_program_thread(program_thread* self, native_string cwd,
                                   libc_startup_fn run, native_string name) {
  new_thread(&self->super, NULL, name);
  self->super.type = THREAD_TYPE_USER;
  self->super._prio = high_priority;
  self->_code = run;
  self->_cwd = NULL;
  self->super.vtable = &_program_thread_vtable;

  program_thread_chdir(self, cwd);
  return self;
}
```

Structure of *Mimosa* (cont.)

- **System is tuned to Gambit, not the opposite**
  - Gambit runs: nothing else does (unless you want to)
  - No system call, direct access to OS services

```
struct libc_link {

  // dirent.h
  DIR *(*_opendir)(const char *__name);
  dirent *(*_readdir)(DIR *__dirp);
  int (*_closedir)(DIR *__dirp);

  // errno.h
  int *_errno;

  // math.h
  double (*_acos)(double __x);
  double (*_acosh)(double __x);
  double (*_asin)(double __x);
  double (*_asinh)(double __x);
  double (*_atan)(double __x);
  double (*_atan2)(double __y, double __x);
  double (*_atanh)(double __x);
  double (*_ceil)(double __x);
  double (*_cos)(double __x);
```

Structure of *Mimosa* (cont.)

- **Kernel has the ability to run:**
  - Interpreted Scheme Code
  - Compiled code (x86 machine)
  - Compile and run Scheme code

Perpective for the Scheme version

- **Convenience of C cancelled out by “C” issues**
  - Pointers, memory allocation…
  - Maybe get the best of both worlds with Scheme?
- **Imperative-style is appropriate sometimes**
  - Scheme allows mix of functional and imperative
- **Many challenges for the Scheme version:**
  - Garbage collection
  - Convenient access to memory mapped resources, CPU instructions and other physical hardware

Main Issues

- **Hardware is imperfect**
  - Not all components work like they always should
- **Support for modern hardware requires more human-resources**
  - Only IDE-pio has been implemented (SATA would be nice)
  - USB is a ton of work
  - Networking? Maybe port of S3 (tcp in Gambit Scheme)
- **Deprecated compiler toolchain (G++ 3.4)**
  - Awkward mix of C++ and C

Future Objectives

- **Automated benchmark capabilities**
  - More work on serial to TCP bridge
- **More drivers?**
  - SATA, USB...
  - BIOS FFI
- **Scheme implementation of:**
  - Boot-sector, basic kernel and more
  - Framework for accessing low-level resources

Conclusion

- **Fully Gambit Scheme kernel closer than ever**
- **Many low-level routines / constructs can be done in Scheme without loss of expressiveness**
- **Available soon on GitHub**
- **Demo...**