GEM5 Simulator in Full System Mode

Yizi Gu

Tsinghua University

yizigu@gmail.com

October 20, 2014

Overview

1. Study Background
   - GEM5 introduction
   - Application in current research
2. Current Progress
   - OS Basics
   - How to make it work
   - How to add files/programs to the disk image

A modular platform for computer architecture research

GEM5 is a cycle accurate simulator. It is the combination of two separate simulator: GEMS and M5. It follows the object oriented design pattern, which makes the extension of modules relatively easy. There are mainly two running modes:

- SE(System call Emulation): System calls are passed to the host machine.
  - Good for testing/profiling micro-architecture design
  - Not enough for assessing the system level performance.
- FS(Full System): Simulating a bare metal machine with CPU, memory hierarchy, I/O, devices.
  - More realistic and more interesting than SE mode
  - Need appropriate kernel and file system image.

Compilation and Module Wrapping

- SCons: GEM5 is compiled under the control of SCons, a promising substitution for Make.
- Swig: A automatic wrapper for connecting C++ with python. A typical auto-generated wrapper function is shown in figure 1. The parameters could be set by python scripts which avoids re-compiling the whole source code.

```
SWIGINTERN PyObject *_wrap_LinuxArmSystemParams_machine_type_get(PyObject *SWIGUNUSEDPARM(self), PyObject *args) {
  PyObject *resultobj = 0;
  LinuxArmSystemParams *arg1 = (LinuxArmSystemParams *) 0 ;
  void *argp1 = 0 ;
  int res1 = 0 ;
  PyObject * obj0 = 0 ;
  Enums::ArmMachineType result;

  if(!PyArg_UnpackTuple(args,(char *)"LinuxArmSystemParams_machine_type_get",1,1,&obj0)) SWIG_fail;
  res1 = SWIG_ConvertPtr(obj0, &argp1,SWIGTYPE_p_LinuxArmSystemParams, 0 |  0 );
  if (!SWIG_IsOK(res1)) {
    SWIG_exception_fail(SWIG_ArgError(res1), "in method '" "LinuxArmSystemParams_machine_type_get" "', argument " "1"" of type '" "LinuxArmSystemParams *""'");
  }
  arg1 = reinterpret_cast< LinuxArmSystemParams * >(argp1);
  result = (Enums::ArmMachineType) ((arg1)->machine_type);
  resultobj = SWIG_From_int(static_cast< int >(result));
  return resultobj;
fail:
  return NULL;
}
```

Figure : Swig Wrapper function

Support for Non-Volatile Sensors Architecture Study

Basic Idea:

- Add new device(sensor) in the GEM5 source tree
- Write kernel module(driver) to support the new device
- Run system benchmark: The benchmark could be generated by using the experimental data such as the typical power supply of energy harvesting nodes.
- Calculate the metrics and evaluate the architectural design.

The typical boot process of a Linux machine

Stage 0

On start up, the BIOS program fixed in FLASH or motherboard is first executed and choose the candidate boot device(Hard Disk, Floppy Disk, USB stick etc.)

Stage 1

Execute first stage bootloader. If boot from hard disk, the 1st stage bootloader resides in the first 446 Bytes of the 512-Byte large MBR(Master Boot Record). The remaining 64 Bytes record four pieces of partition information. The bootloader makes sure there is one and only one active partition from which the machine will boot by inspecting these records.

Cont'd

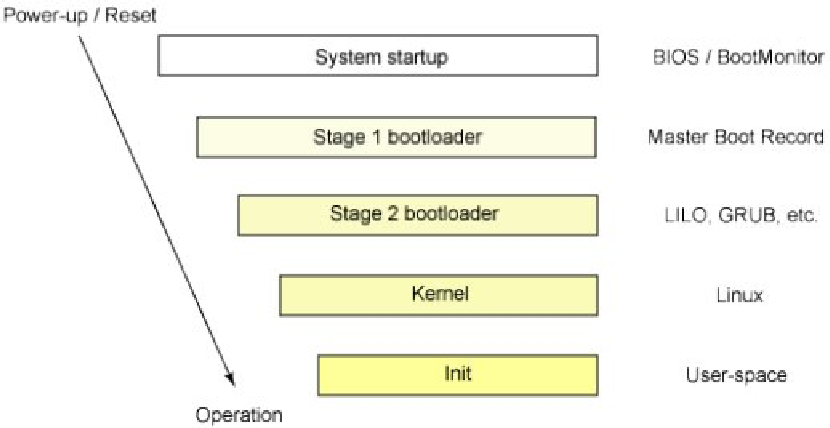

Figure : Simplified Boot Process

Cont'd

Stage 2

The second stage bootloader is loaded and executed. 1st and 2nd stage bootloaders are generally called LILO(Linux Loader)/GRUB(GRand Unified Bootloader). Initial RAM Disk(initrd) will be loaded into the memory and work as a temperate file system for further booting the complete system.

Stage 3

The bootloader then starts the compressed kernel image(zImage/bzImage). And it decompresses itself by calling several functions in head.S, misc.c. At the end of the progress, the basic software environment such as stack and page table are set. Then the start_kernel() function in init/main.c is called.

- Download the kernel and disk image from official website
- Currently the kernel vmlinux.arm.smp.fb.2.6.38.8 runs together with disk image linux-arm-ael.img smoothly.

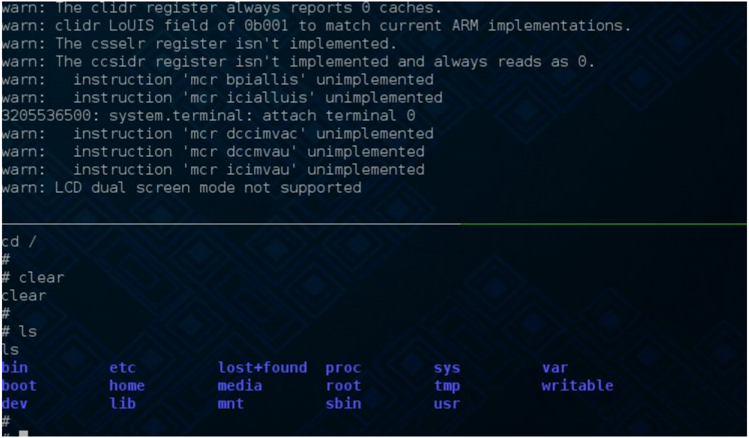

Figure : FS runtime demonstration

How to manipulate the disk image

Problem: The disk image is only a bare file system. How to add files/programs to the system efficiently?

Possible Solutions:

1. Connecting the network: May be rather slow.
2. Boot the system using virtual machine(QEMU/VirtualBox).
3. Mount the file system locally using mount command.

Solution 2 is a good approach but I haven't found the proper kernel and disk image which fit GEM5 and QEMU at the same time. So I mount the file system directly by the shell command:

```
sudo mount -o loop,offset=32256 linux-arm-ael.img /mnt
```

After mounting the image, we could copy the programs into the file tree:

```
sudo cp a.out /mnt/home/
```

Cont'd

We could cross-compile the program on the host machine and then move the binaries into the file system for simulation:

```
#
# ./a.out
Thu Jan  1 00:00:10 UTC 1970
S: mb panel
./a.out
hello world#
# 
```

```
info: Using kernel entry physical address at 0x8000
**** REAL SIMULATION ****
info: Entering event queue @ 0.  Starting simulation...
warn: The clidr register always reports 0 caches.
warn: clidr LoUIS field of 0b001 to match current ARM implementations.
warn: The csselr register isn't implemented.
warn: The ccsidr register isn't implemented and always reads as 0.
warn:   instruction 'mcr bpiallis' unimplemented
warn:   instruction 'mcr icialluis' unimplemented
warn:   instruction 'mcr dccimvac' unimplemented
warn:   instruction 'mcr dccmvau' unimplemented
warn:   instruction 'mcr icimvau' unimplemented
8690953000: system.terminal: attach terminal 0
warn: LCD dual screen mode not supported
```

Figure : FS node running imported program

Kernel Module

I first download the kernel source file whose version is in accordance with the current pre-compiled kernel version(2.6.38.8). Then cross-compiling the module using the kernel header files downloaded. Then move the .ko file into the file system and insert the module. I got a problem with the magic version that remains to be solved:

```
# insmod hello.ko
insmod hello.ko
[  158.631323] hello: version magic '2.6.38-rc8 mod_unload ARMv6 ' should be '2.6.38.8-gem5
SMP mod_unload ARMv7 '
insmod: can't insert 'hello.ko': invalid module format
```

This week's work

- Study the GEM5 FS mode further and learn how to get the simulation profiles.
- Read papers relevant to the research

Thank you