Porting Plan 9 to the PowerPC Architecture

Ian Friedman
Ajay Surie
Adam Wolbach
Plan 9
http://www.cs.bell-labs.com/plan9dist/

• OS developed by Bell Labs in order to cost-effectively manage large, centralized resources to employees at cheap, less powerful terminals

• Key Features:
  – Centralized resources allow for efficient management/sharing
  – Local name space, regardless of the box in front of you
  – Everything is (theoretically) considered a file, including devices
    • Thus, there exists a unique protocol that accesses everything
  – Cheap for large-scale implementations
Project Additions

• What We Won’t Change:
  – Fundamental operating system design
    • Specification of OS remains the same, no new features

• What We Will Add:
  – Compatibility with 32-bit PowerPC architectures
    • Booting with OpenFirmware and BootX/Yaboot
    • Memory Management
    • Device Support: Console I/O, Ethernet
    • This is the type of PowerPC downstairs: 74xx (almost certain of this)
  – Compatibility with 64-bit PowerPC architectures with 32-bit emulation
Resources

412 Lab/Wean Hall 3508 Cluster

- 1 iMac G4
  - 744x Series PowerPC
  - Used for Testing

- 3 Linux “Boxes”
  - Made of miscellaneous parts off the CS pile
  - Reasonably fast
  - Used for Coding/Compiling/Debugging

- Everything we need has been provided
Code Base

- Plan 9 existing PowerPC code
  - C Code ~ 10,000 lines
    - A lot of it is code for different PPC hardware (2 ethernet drivers, Saturn, 8260)
    - Can use Plan 9 common kernel routines (i.e. main.c) and kernel interface to ethernet (~1000 lines)
    - Probably don’t need to touch protocol code [TLS 1.0, SSL 3.0] – (~2000 lines)
    - Can completely ignore flash related code (BLAST) and UART serial code (~2000 lines)
  - Plan 9 assembly code ~ 1100 lines
    - Not sure how much will need to change
Booting Plan 9

• OpenFirmware
  – Uses Forth as its command interface
  – Provides basic hardware support
  – Loads the kernel for us
    • Can load kernel over TFTP!
  – Once the kernel is loaded, we still need to talk to OF for devices
Booting Plan 9 (cont’d)

• The Task at Hand: Figure out how to talk to OF for device support (console/ethernet)

• Investigating two possibilities:
  – Primary: BootX
    • Darwin’s bootloader
    • Not very well documented
  – Secondary/Backup: Yaboot
    • Used by PPC linux distributions
    • Also not very well documented
Memory Management

(based on http://users.rowan.edu/~shreek/fall01/comparch2/lectures/PowerPC.ppt)

• Using PowerPC 740/750 architecture specification (32-bit)

• 2 Memory Management Units
  – Distinctive behaviors for Data and Instruction fetches/translations
    • Each have their own L1 cache
    • Unified L2 cache

• Memory Support
  – Physical Memory: 64 Gigabytes ($2^{36}$)
  – Virtual Memory: 4 Pentabytes ($2^{52}$)
Memory Address Translation

• 3 Address Translation Modes
  – Page Address Translation
    • In other words, virtualization via segmentation
      – Translation from 32-bit effective address (EA), to 52-bit virtual address (VA) (by segment table), to 32-bit real/physical address (PA) (by page table)
      – Segment table comprised of 16 on-chip segment registers
      – Segmentation also used as memory-mapping for I/O devices
  – Block Address Translation
    • EA translated to PA via BAT table lookup
      – Table is actually set of pairs of on-chip registers (limiting the number of possible “Blocks”), separate for Data and Instructions
  – Real Addressing Mode
    • Effective Address = Physical Address (i.e., no virtualization)
Figure 5-1. MMU Conceptual Block Diagram—32-Bit Implementations
Memory Management

• Documentation:
  – Actual specification:
    • Google “PowerPC 740 750 RISC Microprocessor”
  – IBM Overview:
    • Google”powerpc 740/750 memory management unit”
  – Brief introduction to PPC architecture:
    http://users.rowan.edu/~shreek/fall01/comparch2/lectures/PowerPC.ppt
Lines of Code to Write

- Bootloader
  - Existing open source code available, may require minor modification
- Memory Management (~300 lines)
- Interrupts / Hardware specific (~350 lines)
- Device Drivers
  - Console (~200 lines – depending on available code for PPC)
  - Ethernet (~800 lines)
  - Clock / Timer (~100 lines)

- Total >= 1800 lines
Brief Schedule - Optimistic

• 14 weeks remaining in the semester
• Stage I (2 – 3 weeks)
  – Understand existing code / architecture
• Stage II (~4 weeks)
  – Open Firmware / Booting (~4 weeks)
  – Memory Management (4 – 6 weeks)
• Stage III (~3 weeks)
  – Console driver
  – Ethernet driver
  – Integration
• Stage IV (~2 weeks)
  – Debugging / making it work
Issues / Challenges

• Understanding existing code base
  – Poorly documented, module structure unclear
    (not the way we were taught in 410!)
  – Will probably require more time than we expect
• Understanding Plan 9 assembly
• Integrating new code – maintaining the “Plan 9 way” of kernel implementation
• Probably others we haven’t thought of…