QEMU Just-In-Time Code Generator and System Emulation

趙至敏 (cmchao), Andes Technology
黃敬群 (jserv), 0xlab

March 15, 2010 / 台灣科技大學資訊工程所

Rights to copy

Part I :

QEMU Just-In-Time Code Generator

Outline

- Concept
- Traditional Methods & Problems
- QEMU Approach

Who am I

- 趙至敏 (cmchao)
- 交通大學電子所系統組 (2003~2005)
- Andes Technology / 晶心科技 (2007~current)
- Open source project
  - pcmanx (Gtk+-based BBS client)
  - QEMU
- Contact: cmchao@gmail.com

Andes Technology

- High-performance/low-power 32-bit processors designer & provider
- Full SoC Solution
- Full Software Solution (toolchains, Linux and RTOS)
- Full Software Development Environment

What is QEMU

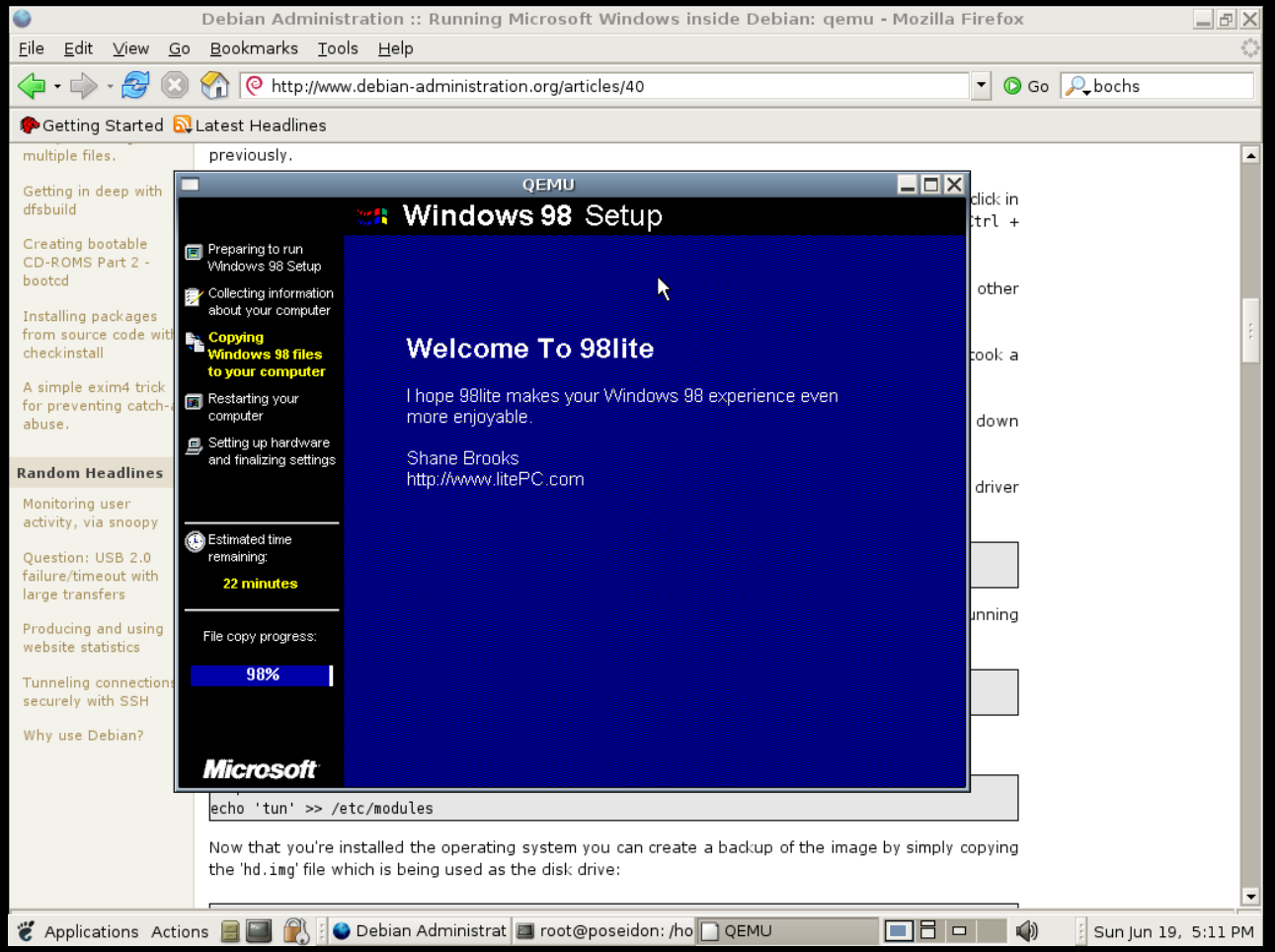
- “Quick Emulator” for system level emulation
- Created by Fabrice Bellard in 2003
- Features
  - Just-in-time(JIT) compilation support to achieve high performance (400 ~ 500 MIPS)
  - Cross-platform (most UNIX-like system and MS-Windows)
  - Lots of peripherals support
  - Lots of target hosts and targets support (full system emulation)
    - x86, arm, mips, sh4, cris, sparc, nds32, …
  - User mode emulation: can run applications compiled for another CPU.

Terminology clarification

- Emulation vs simulation
  - emulation : be you
  - simulation : be like you
- Host vs. target(guest)
  - host : where you run QEMU
  - target : the emulated computer

Screenshot (1)

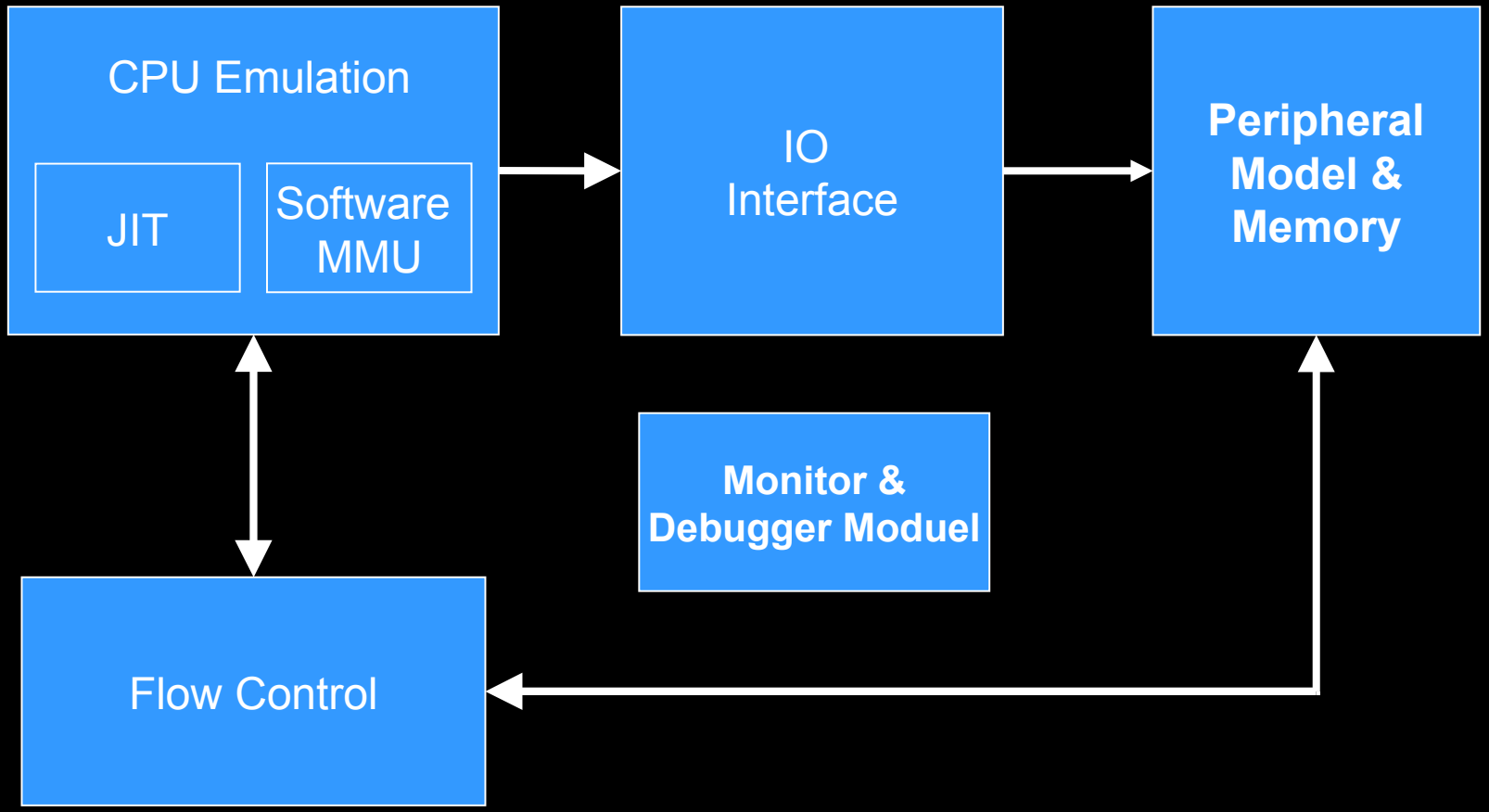
- Source : http://free.oszoo.org/img/98lite.png
- Host :x86
- Target : x86



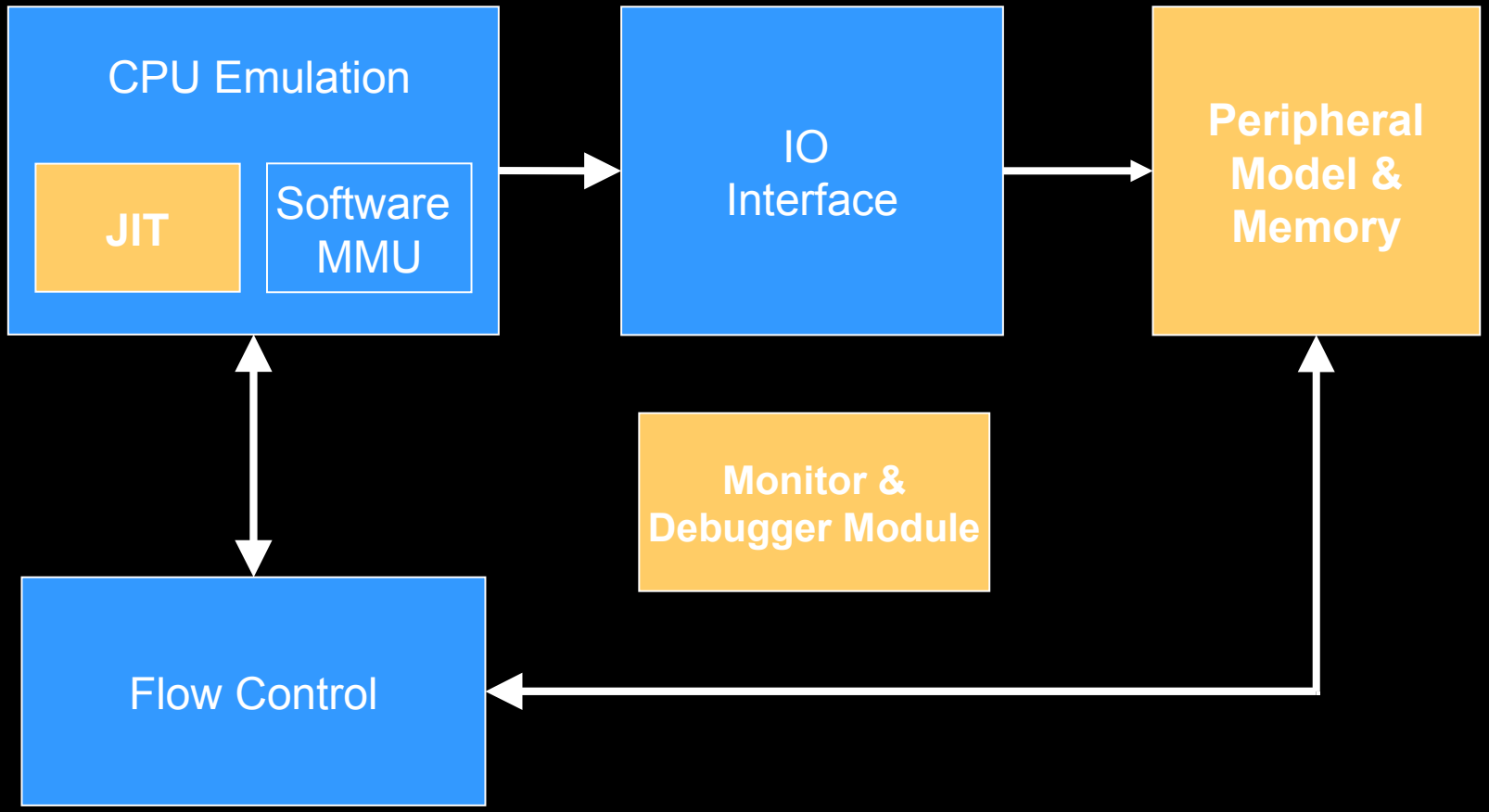
Screenshot (2)

- Source : http://www.linuxtopia.org
- Host : x86
- Target : arm

QEMU System Framework



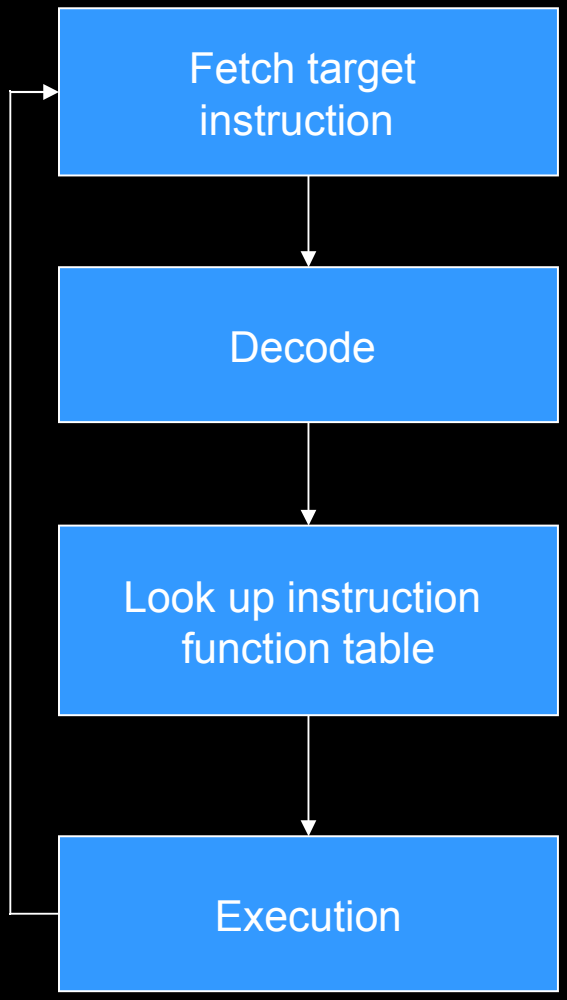
QEMU System Framework



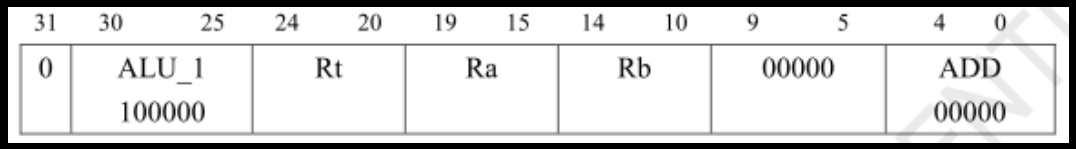
Outline

- Concept
- Traditional Methods & Problems
- QEMU Approach

Traditional CPU Emulation Approach




```
pc(0x50000):40008800
```



31	30 25	24 20	19 15	14 10	9 5	4 0
0	ALU_1 100000	Rt	Ra	Rb	00000	ADD 00000

```
Opcode : 0x10 (add), rt:0, ra:1, rb:2

 switch(opcode)
 case add :
        add(rt, ra, rb)
        break;


void add(int rt, int ra, int rb) {
    env->gpr[rt] = env->gpr[ra] +
    env->gpr[rb];
}
```

Performance Issues (1/3)

- Functional call overhead

```
void add(int rt, int ra, int rb) {
        env->gpr[rt] =
        env->gpr[ra] + env->gpr[rb];
}
```

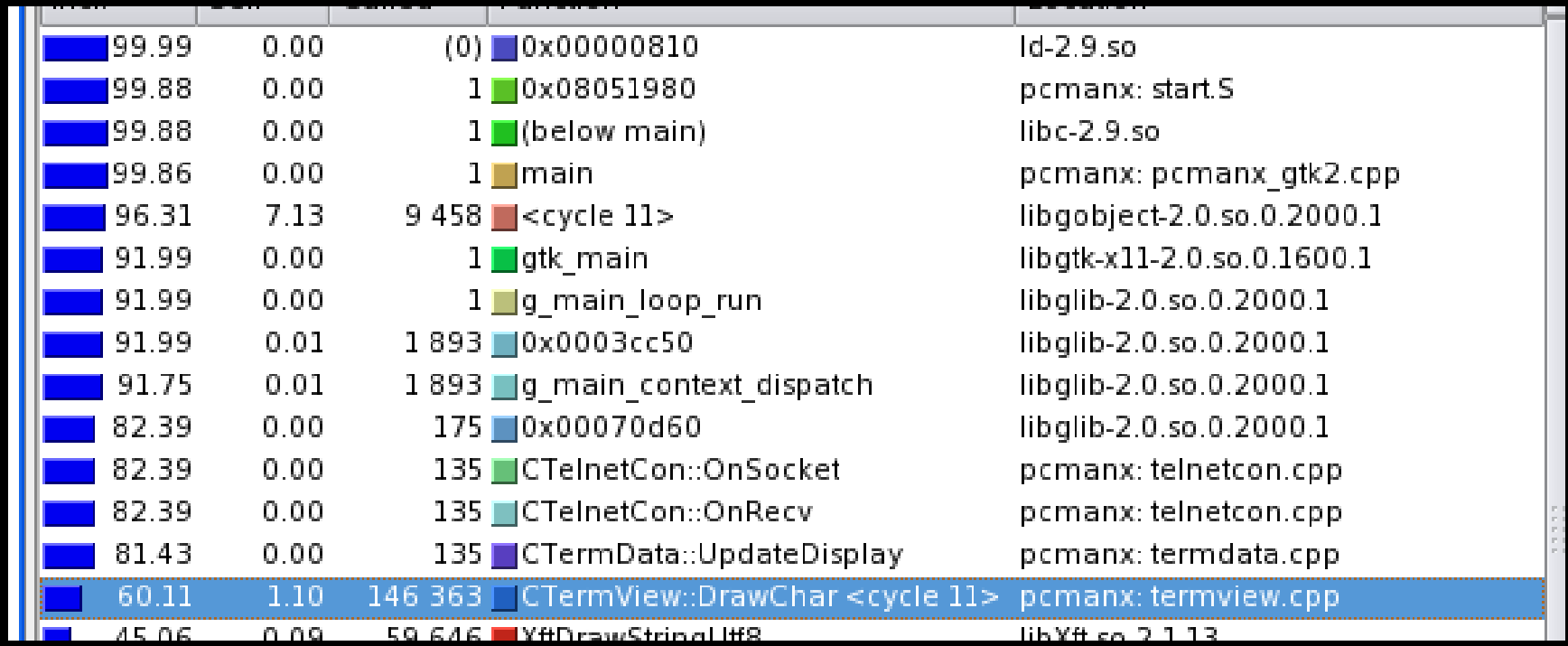
Prologue : reserve stack

Epilogue :
Release stack and
return to caller

```
pushq   %rbp
movq    %rsp, %rbp
movl    %edi, -4(%rbp)
movl    %esi, -8(%rbp)
movl    %edx, -12(%rbp)
movq    env(%rip), %r8
movl    -4(%rbp), %eax
movslq  %eax,%rsi
movq    env(%rip), %rdi
movl    -8(%rbp), %eax
movslq  %eax,%rcx
movq    env(%rip), %rdx
movl    -12(%rbp), %eax
cltq
movl    4(%rdx,%rax,4), %eax
addl    4(%rdi,%rcx,4), %eax
movl    %eax, 4(%r8,%rsi,4)
leave
ret
```

Performance Issues (2/3)

- Redundant fetch and decode
  - Space locality : programs usually run sequentially
  - Temporal locality : 80/20 rule



99.99	0.00	(0)	0x00000810	ld-2.9.so
99.88	0.00	1	0x08051980	pcmanx: start.S
99.88	0.00	1	(below main)	libc-2.9.so
99.86	0.00	1	main	pcmanx: pcmanx_gtk2.cpp
96.31	7.13	9 458	<cycle 11>	libgobject-2.0.so.0.2000.1
91.99	0.00	1	gtk_main	libgtk-x11-2.0.so.0.1600.1
91.99	0.00	1	g_main_loop_run	libglib-2.0.so.0.2000.1
91.99	0.01	1 893	0x0003cc50	libglib-2.0.so.0.2000.1
91.75	0.01	1 893	g_main_context_dispatch	libglib-2.0.so.0.2000.1
82.39	0.00	175	0x00070d60	libglib-2.0.so.0.2000.1
82.39	0.00	135	CTelnetCon::OnSocket	pcmanx: telnetcon.cpp
82.39	0.00	135	CTelnetCon::OnRecv	pcmanx: telnetcon.cpp
81.43	0.00	135	CTermData::UpdateDisplay	pcmanx: termdata.cpp
60.11	1.10	146 363	CTermView::DrawChar <cycle 11>	pcmanx: termview.cpp
45.06	0.09	59 646	XftDrawStringUtf8	libXft.so.2.1.13

Performance Issues (3/3)

- Un-matched description ability of high-level language

```
add  r0, r1, r2
void add(int rt, int ra, int rb) {
        env->gpr[rt] = env->gpr[ra] + env->gpr[rb];
}
```

From high-level language

```
pushq   %rbp
movq    %rsp, %rbp
movl    %edi, -4(%rbp)
movl    %esi, -8(%rbp)
movl    %edx, -12(%rbp)
movq    env(%rip), %r8
movl    -4(%rbp), %eax
movslq  %eax,%rsi
movq    env(%rip), %rdi
movl    -8(%rbp), %eax
movslq  %eax,%rcx
movq    env(%rip), %rdx
movl    -12(%rbp), %eax
cltq
movl    4(%rdx,%rax,4), %eax
addl    4(%rdi,%rcx,4), %eax
movl    %eax, 4(%r8,%rsi,4)
leave
ret
```

X86 Host

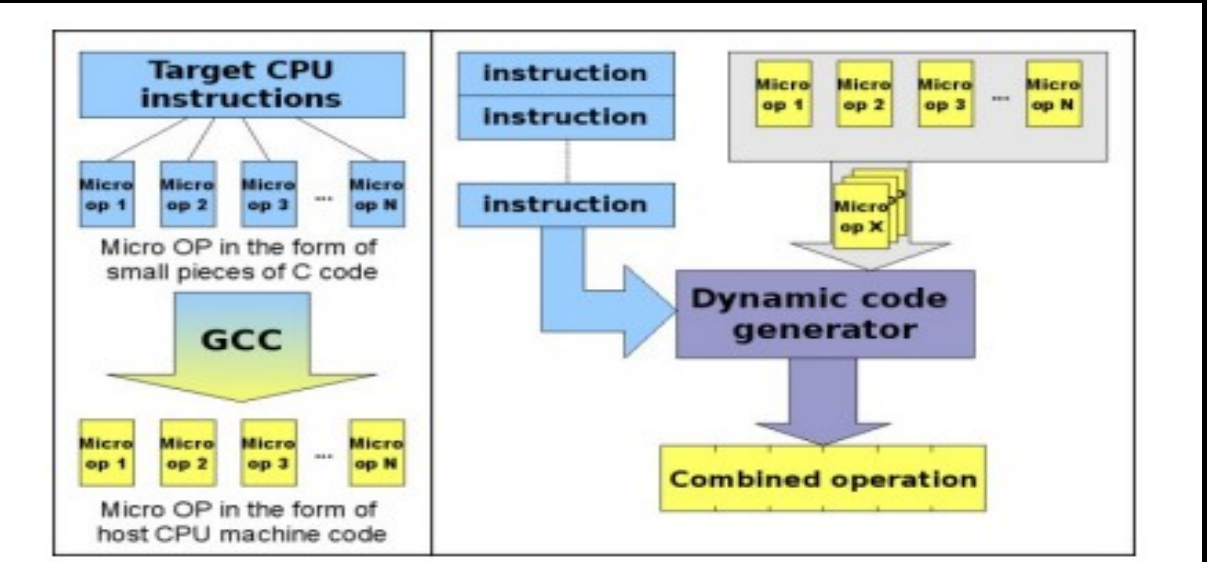
```
mov     0x7c(%r14),%r15d
mov     0x6c(%r14),%r12d
add     %r12d,%r15d
mov     %r15d,0x8(%r14)
```

Outline

- Concept
- Traditional Methods & Problems
- QEMU Approach

Goal

- Use the most simplified host instruction to describe target instruction
- Perform automatic translation and patch at run-time



Build Micro-op Template

- Micro-op : common and simple host operation

```
add  r0, r1, r2
void add(int rt, int ra, int rb) {
        env->gpr[rt] = env->gpr[ra] + env->gpr[rb];
}
```

```
mov   0x7c(%r14),%r15d      read data from register-related addr
mov   0x6c(%r14),%r12d      the same
add   %r12d,%r15d           ALU operation on registers
mov   %r15d,0x8(%r14)       store date into register-related addr
```

Build Micro-op Template

- Use compiler trick to force register use and alias register as variable

```
register struct CPUNDS32State *env asm(r14);
register target_ulong T0 asm(r15);
register target_ulong T1 asm(r12);
register target_ulong T2 asm(r13);
```

```
T0 = env->gpr[19];          mov   0x7c(%r14),%r15d
T1 = env->gpr[15];          mov   0x6c(%r14),%r12d
T0 = T0 + T1;               add   %r12d,%r15d
env->gpr[2] = T0;           mov   %r15d,0x8(%r14)
```

Build Micro-op Template

- Use function call and more compiler trick to produce related host instruction
  - -O2 : remove prologue because of leaf-function
  - __volatile__("" : : : "memory") : produce one epilog

```
void op_load_gpr_T0_gpr_r0(void) {
    T0 = env->gpr[0];
    FORCE_RET();
}
```

```
0000000000000000 <op_load_gpr_T0_gpr0>:
       0:       45 8b 3e        mov    (%r14),%r15d
       3:       c3              retq
```

```
void op_add(void) {
    T0 = T1 + T2;
    FORCE_RET();
}
```

```
0000000000001c18 <op_add>:
    1c18:       45 01 e7        add    %r12d,%r15d
    1c1b:       c3              retq
```

```
void op_addi(void){
    T0 += (int32_t)PARAM1;
    FORCE_RET();
}
```

```
0000000000002b85 <op_addi>:
    2b85:       8d 05 00 00 00 00    lea    0(%rip),%eax

    2b8b:       41 01 c7             add    %eax,%r15d
    2b8e:       c3                   retq
```

Build Micro-op Template

Build the micro-op vs host code look-up table

```
000000 <op_load_gpr_T0_gpr0>:
  0:  45 8b 3e     mov    (%r14),%r15d
  3:  c3           retq

0001c18 <op_add>:
  1c18:  45 01 e7    add    %r12d,%r15d
  1c1b:  c3          retq

002b85 <op_addi>:
  2b85:  8d 05 00 00 00 00  lea    0(%rip),%eax
  2b8b:  41 01 c7           add    %eax,%r15d
  2b8e:  c3                 retq
```

```
case INDEX_op_load_gpr_T0_gpr0:
    copy code from op_load_grp_T0_gpr0
    to code-gen buffer
break;

case INDEX_op_add:
    copy code from op_load_add
    to code-gen buffer
break;

case INDEX_op_addi:
    copy code from op_load_addi
    to code-gen buffer

    patch the constant value part
break;
```

File : *op.h* in build directory

Code-generation Flow

Target instruction

```
pc(0x1124): add r0, r1, r2
```

cached
(default size: 16MB)

Micro-op translation, implemented in advance

```
op_load_gpr_T0_r1
op_load_gpr_T1_r2
op_add
op_store_gpr_T0_r0
```

Generate code by look-up pre-built table

Set buffer executable and jump to Buffer & Execute

Linux : mprotect()

Buffer Address : 0x20000

```
mov     0x7c(%r14),%r15d
mov     0x6c(%r14),%r12d
add     %r12d,%r15d
mov     %r15d,0x8(%r14)
```

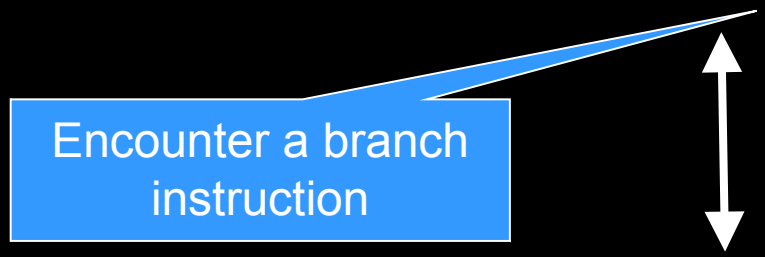
Optimization

- Use basic block as execution unit (space locality)
- Chain basic block
- Cache translated basic block (temporal locality)

Optimization : Use Basic Block

- Basic block : code that has one exit point and one problem flow alternation instruction

```
000081ac <abort>:
    81ac:       addi $sp,$sp,#-24
    81b0:       swi $fp,[$sp+#20]
    81b4:       addi $fp,$sp,#16
    81b8:       movi55 $r0,#1
    81ba:       sethi $r15,#8
    81be:       ori $r15,$r15,#0x32c
    81c2:       jral $lp,$r15
    81c6:       mov55 $sp,$fp
    81c8:       lwi $fp,[$fp+#0]
    81cc:       addi $sp,$sp,#4
    81d0:       ret5 $lp
```



Optimization : Chain Basic Block

- Certain branch target is deterministic, i.e no more judgment is required

```
void select(int *a, int *b) {
        if (*b < 10)
                *a = *b * 2;
        else
                *a = *b * 4 + 1;
}
```

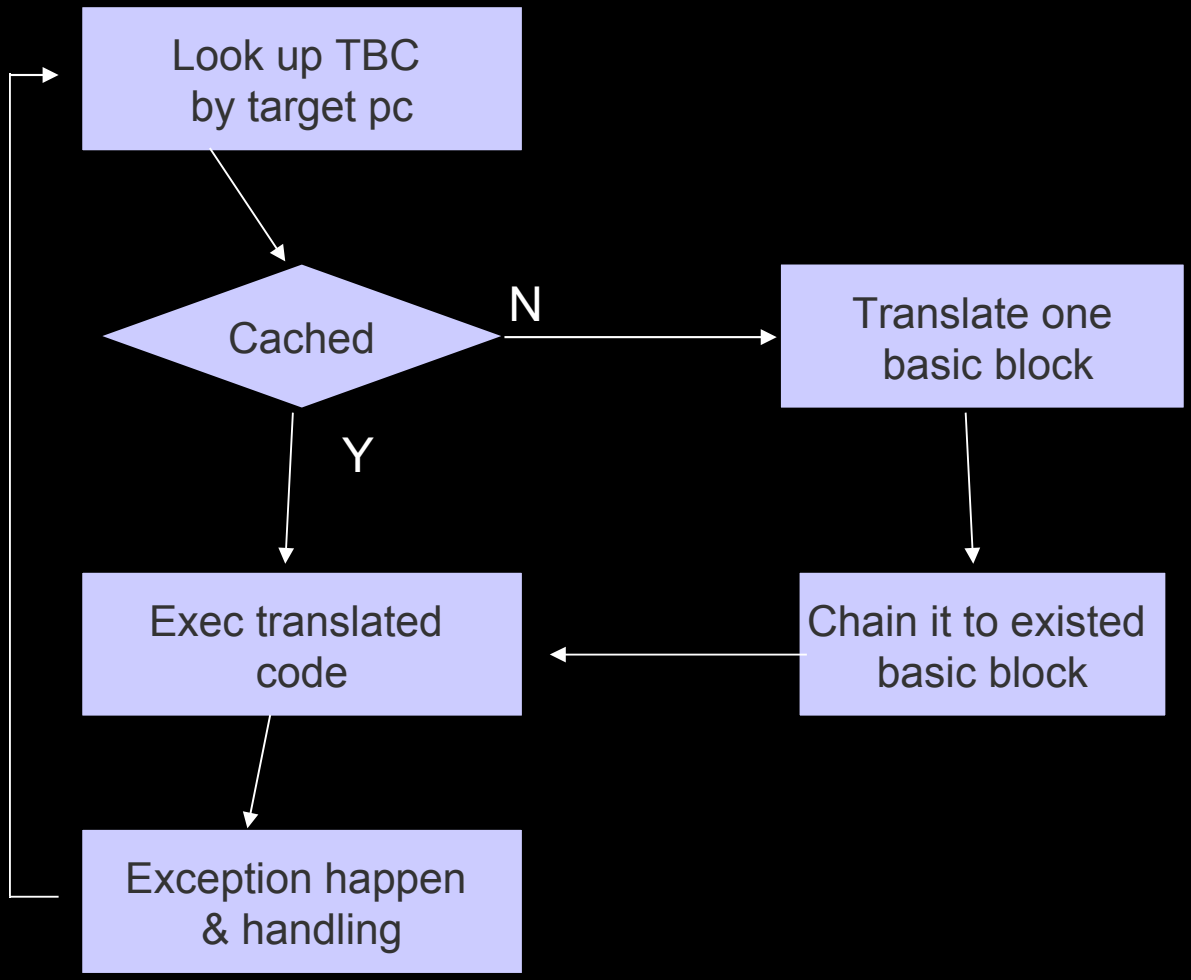
Yes condition

No condition

```
84a8:   smwa.adm $sp,[$sp],$sp,#0x8
84ac:   addi10.sp #-4
84ae:   mov55 $fp,$sp
84b0:   lwi450 $r2,[$r1]
84b2:   mov55 $r3,$r0
84b4:   slli333 $r0,$r2,#0x2
84b6:   addi333 $r0,$r0,#0x1
84b8:   add333 $r1,$r2,$r2
84ba:   sltsi45 $r2,#0xa
84bc:   beqzs8 84c2 <select+0x1a>
84be:   swi450 $r1,[$r3]
84c0:   j8 84c4 <select+0x1c>
84c2:   swi450 $r0,[$r3]
84c4:   addi $sp,$fp,#4
84c8:   lwi.bi $fp,[$sp],#4
84cc:   ret5 $lp
84ce:   srli45 $r0,#0x0
```

- Lazy condition code flag evaluation

CPU JIT Execution Flow



More Optimization

- GCC4 support

Solution: **TCG** (Tiny Code Generator)

GCC4 can not guarantee the occurrence of **retq** instruction, thus QEMU trick can not be applied

(a)
```
0000000000002b85 <op_addi>:
        2b85: 8d 05 00 00 00 00       lea    0(%ri  ),%eax

        2b8b: 41 01 c7                add    %eax,%r15d
        2b8e: c3                      retq
```

- Redundant register allocation

(b)
```
add r2, r19, r15;
Sub r3, r2, r1;
```

```
mov    0x4(%r14),%r15d
mov    0x8(%r14),%r12d
add    %r12d, %r15d
mov    %r15d, 0x8(%r14)
mov    0x8(%r14),%r15d
mov    0x4(%r14),%r12d
sub    %r12d, %r15d
mov    %r15d, 0xc(%r14)
```

- Duplicated code for different target porting

Source Code Organization

- qemu/
  - qemu-* : OS dependent API wrapper
    example: memory allocation or socket
  - target-*/ : target porting
  - tcg/ : new and unified JIT framework
  - *-user/ : user-mode emulation on different OS
  - softmmu-* : target MMU acceleration framework
  - hw/ : peripheral model
  - fpu : softfloat FPU emulation library
  - gdb : GDB stub implementation

Source Tree Organization

- qemu/target-*/
  - cpu.h : regs and other cpu status declaration
  - exec.h : declaration and certain inline function used outside
  - op_helper.c : complicated instruction which cannot be modeled with micro-op such as MMU
  - helper.c : exception handling
  - translate.c : target instruction to micro-op mapping

Part II :

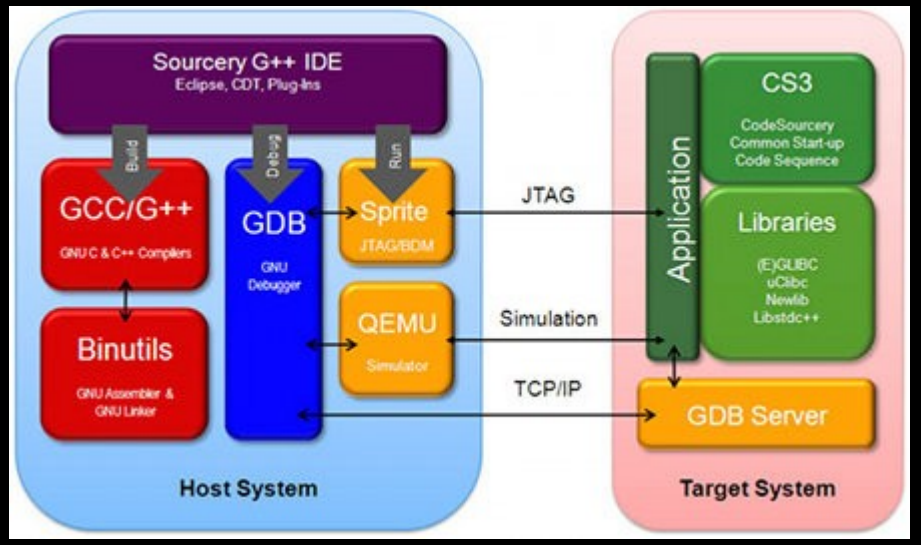
QEMU System Emulation

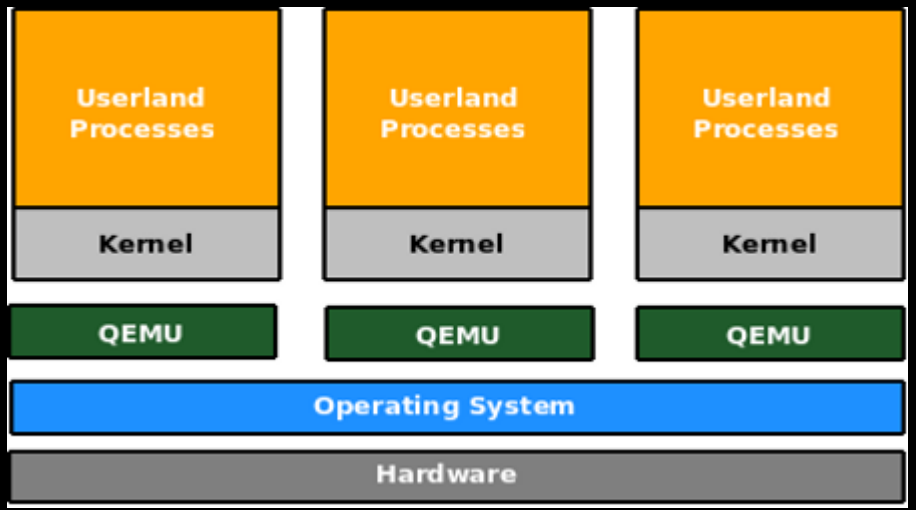
Who am I

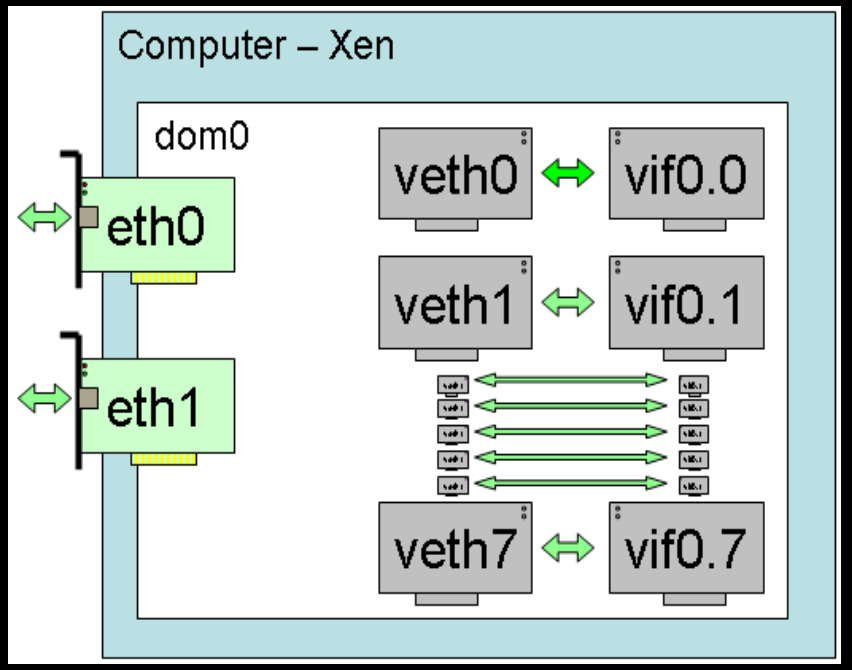
- 黃敬群 (Jim Huang / jserv)
- Software Group Lead, Openmoko (2007-2008)
- Engineer, Andes Technology / 晶心科技 (2008~2009)
- Developer & Co-founder, 0xlab (2009-)

  http://0xlab.org

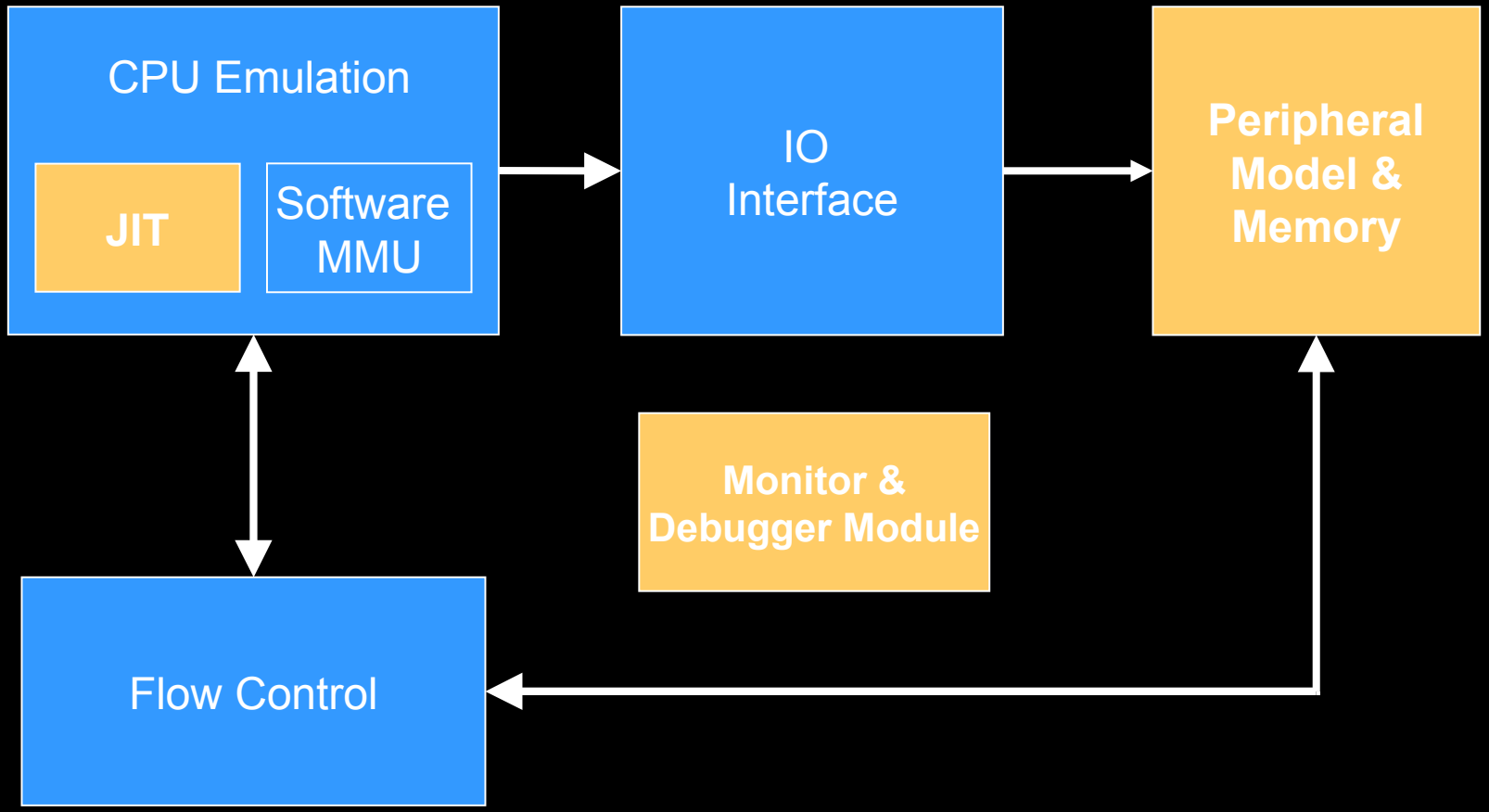
Real Applications for QEMU





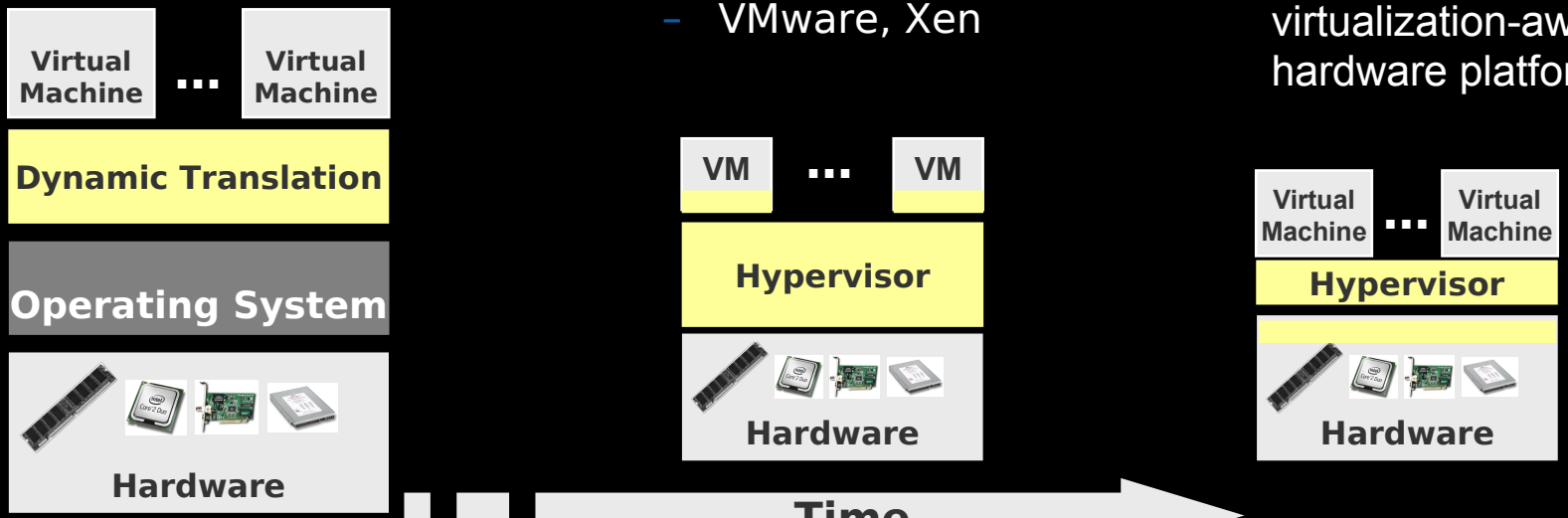


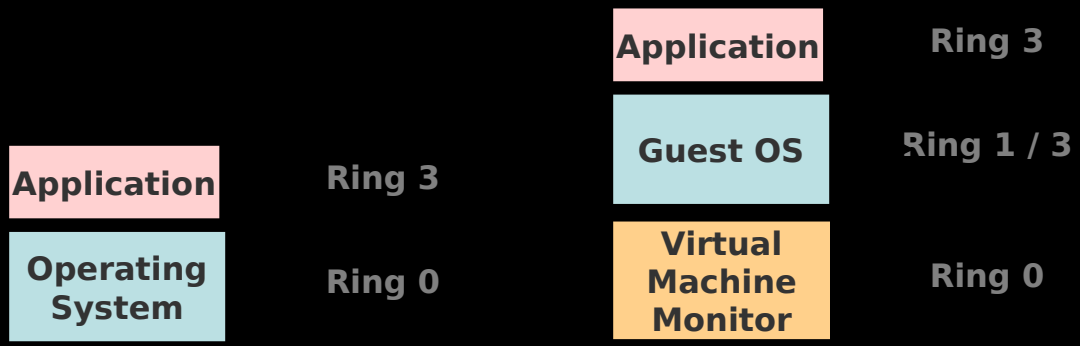
QEMU System Framework



Evolution of Software

- **Full virtualization (Binary rewriting)**
  - Software-based
  - VMware
- **Paravirtualization**
  - Cooperative virtualization
  - Modified guest
  - VMware, Xen
- **Hardware-assisted virtualization**
  - Unmodified guest
  - VMware and Xen on virtualization-aware hardware platforms

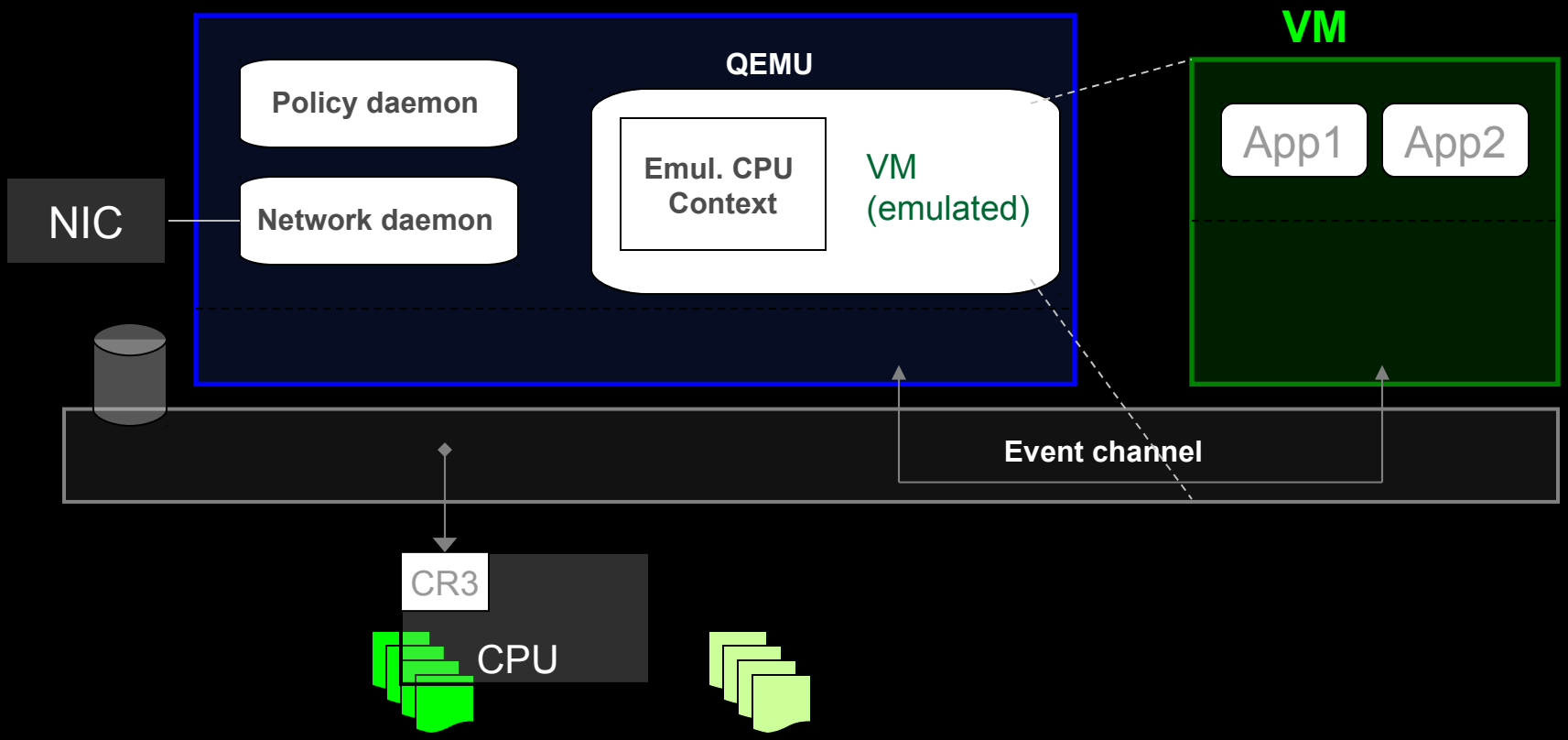


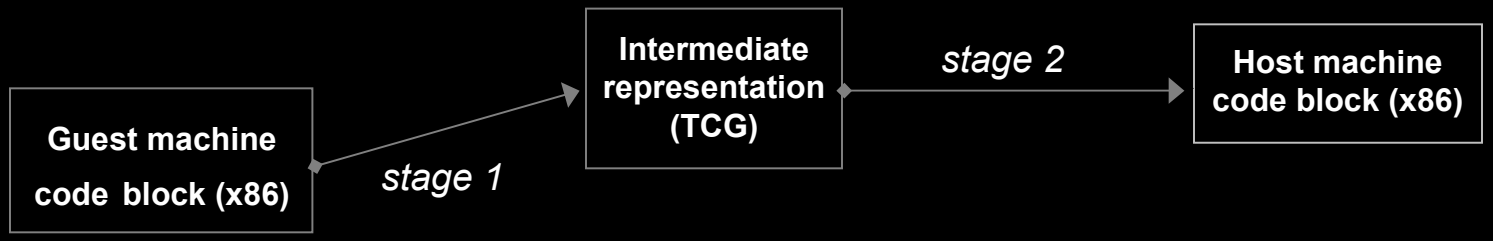


Traditional x86 Architecture

Full Virtualization

Big Picture


VM
QEMU
Policy daemon
Network daemon
NIC
Emul. CPU Context
VM (emulated)
App1
App2
Event channel
CR3
CPU


Guest machine code block (x86)
stage 1
Intermediate representation (TCG)
stage 2
Host machine code block (x86)

QEMU System Emulation

- Emulates a certain complete machine
  - Cross run unmodified operating system software or firmware
  - Can also emulate Boot ROM including different bootstrap methods.
- QEMU itself is single-threaded.
  - The speed of emulation depends on the amount of peripherals and modeling complexity.

I/O Emulation

A computer is more than a CPU

Also need I/O!

Types of I/O:

- Block (e.g. hard disk)
- Network
- Input
  - keyboard, mouse, ...
- Sound
- Video

Most performance critical (for servers):

- Network
- Block

QEMU I/O

- Memory Access
  - Basic BUS topology modeling
- SoftMMU
  - QEMU implements fast TLB caches slower target TLB
- Interrupt
- Peripherals

QEMU Peripherals

- Source: directory hw/
- Register callback functions for control register access
- QEMU Boards initialize CPU core, define Address Map, wire up devices, and load OS/Kernel images
- Timer, Interrupt controller, DMA, Flash type memory, Serial ports, IDE/SCSI, Graphics adapter, ...

```
static QEMUMachine realview_pbx_a9_machine = {
    .name = "realview-pbx-a9",
    .desc = "ARM RealView Platform Baseboard Explore for Cortex-A9",
    .init = realview_pbx_a9_init,
    .use_scsi = 1,
    .max_cpus = 4,
};

static void realview_machine_init(void)
{
    qemu_register_machine(&realview_eb_machine);
    qemu_register_machine(&realview_eb_mpcore_machine);
    qemu_register_machine(&realview_pb_a8_machine);
    qemu_register_machine(&realview_pbx_a9_machine);
}

machine_init(realview_machine_init);
device_init(realview_register_devices)
```

hw/realview.c

```
static void realview_pbx_a9_init(ram_addr_t ram_size,
                    const char *boot_device,
                    const char *kernel_filename, const char *kernel_cmdline,
                    const char *initrd_filename, const char *cpu_model)
{
    if (!cpu_model) {
        cpu_model = "cortex-a9";
    }
    realview_init(ram_size, boot_device, kernel_filename, kernel_cmdline,
                  initrd_filename, cpu_model, BOARD_PBX_A9);
}
```

```
static void realview_init(ram_addr_t ram_size,
                    const char *boot_device,
                    const char *kernel_filename, const char *kernel_cmdline,
                    const char *initrd_filename, const char *cpu_model,
                    enum realview_board_type board_type)
{
 …
    sysbus_create_simple("pl050_keyboard", 0x10006000, pic[20]);
    sysbus_create_simple("pl050_mouse", 0x10007000, pic[21]);

    sysbus_create_simple("pl011", 0x10009000, pic[12]);
    sysbus_create_simple("pl011", 0x1000a000, pic[13]);
    sysbus_create_simple("pl011", 0x1000b000, pic[14]);
    sysbus_create_simple("pl011", 0x1000c000, pic[15]);

    /* DMA controller is optional, apparently.  */
    sysbus_create_simple("pl081", 0x10030000, pic[24]);

    sysbus_create_simple("sp804", 0x10011000, pic[4]);
    sysbus_create_simple("sp804", 0x10012000, pic[5]);

    sysbus_create_simple("pl110_versatile", 0x10020000, pic[23]);

    sysbus_create_varargs("pl181", 0x10005000, pic[17], pic[18], NULL);

    sysbus_create_simple("pl031", 0x10017000, pic[10]);
…
    dev = sysbus_create_simple("realview_i2c", 0x10002000, NULL);
    i2c = (i2c_bus *)qdev_get_child_bus(dev, "i2c");
    i2c_create_slave(i2c, "ds1338", 0x68);
…
```

DSL = Damn Small Linux distribution

jserv@venux: ~/virt

```
$ /opt/bin/qemu -cdrom $HOME/Downloads/dsl-4.4.10.iso
```

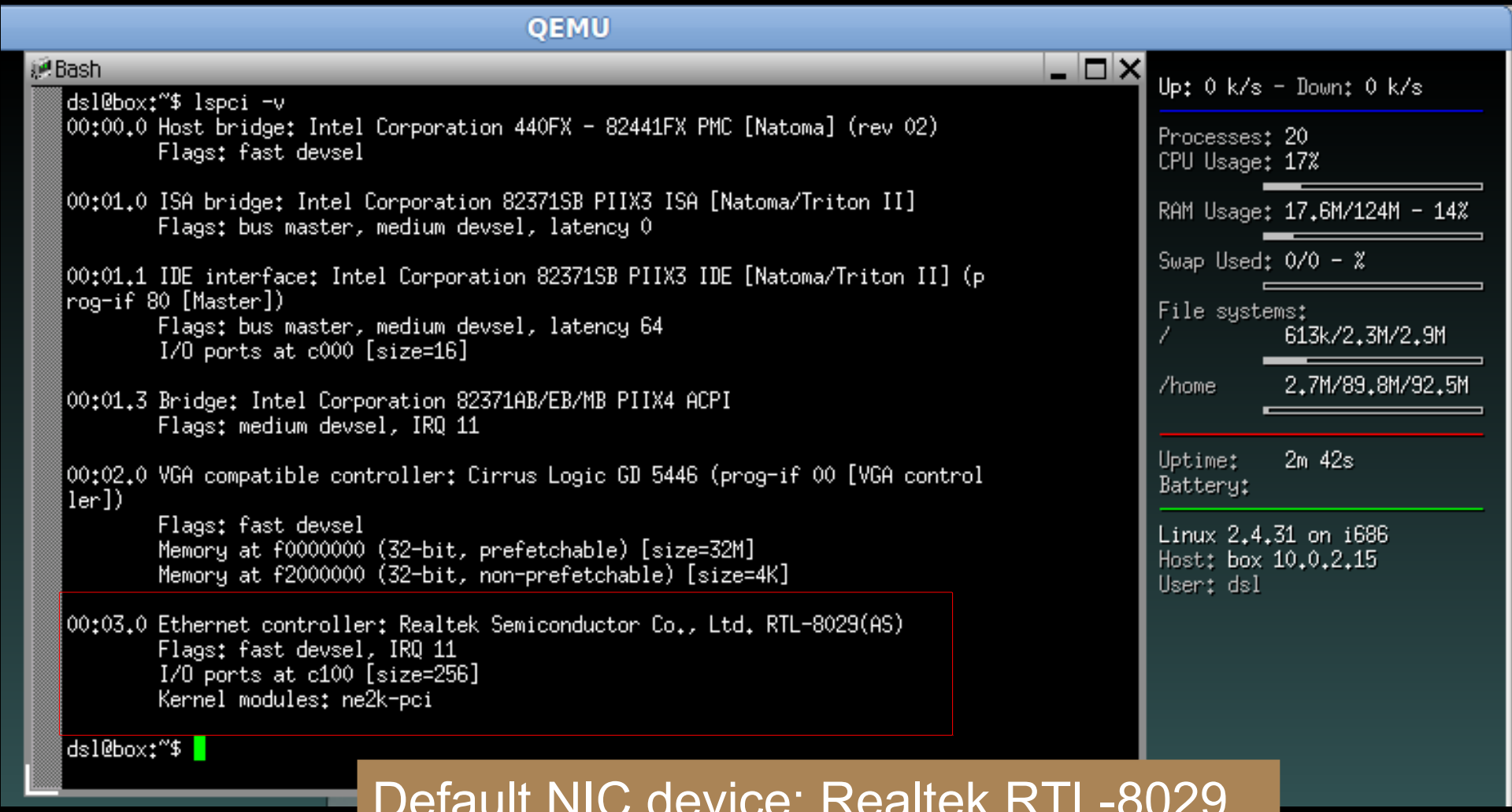
QEMU

```
QEMU 0.9.1 monitor - type 'help' for more information
(qemu) info pci
  Bus  0, device   0, function 0:
    Host bridge: PCI device 8086:1237
  Bus  0, device   1, function 0:
    ISA bridge: PCI device 8086:7000
  Bus  0, device   1, function 1:
    IDE controller: PCI device 8086:7010
      BAR4: I/O at 0xc000 [0xc00f].
  Bus  0, device   1, function 3:
    Bridge: PCI device 8086:7113
      IRQ 11.
  Bus  0, device   2, function 0:
    VGA controller: PCI device 1013:00b8
      BAR0: 32 bit memory at 0xf0000000 [0xf1ffffff].
      BAR1: 32 bit memory at 0xf2000000 [0xf2000fff].
  Bus  0, device   3, function 0:
    Ethernet controller: PCI device 10ec:8029
      IRQ 11.
      BAR0: I/O at 0xc100 [0xc1ff].
(qemu)
```

QEMU monitor

Default NIC device: Realtek RTL-8029

Switch back to VGA screen
(under Linux)



Default NIC device: Realtek RTL-8029

```
$ /opt/bin/qemu -cdrom $HOME/Downloads/dsl-4.4.10.iso \
  -net user \
  -net nic,model=atheros_wlan_linux_HPW400
```

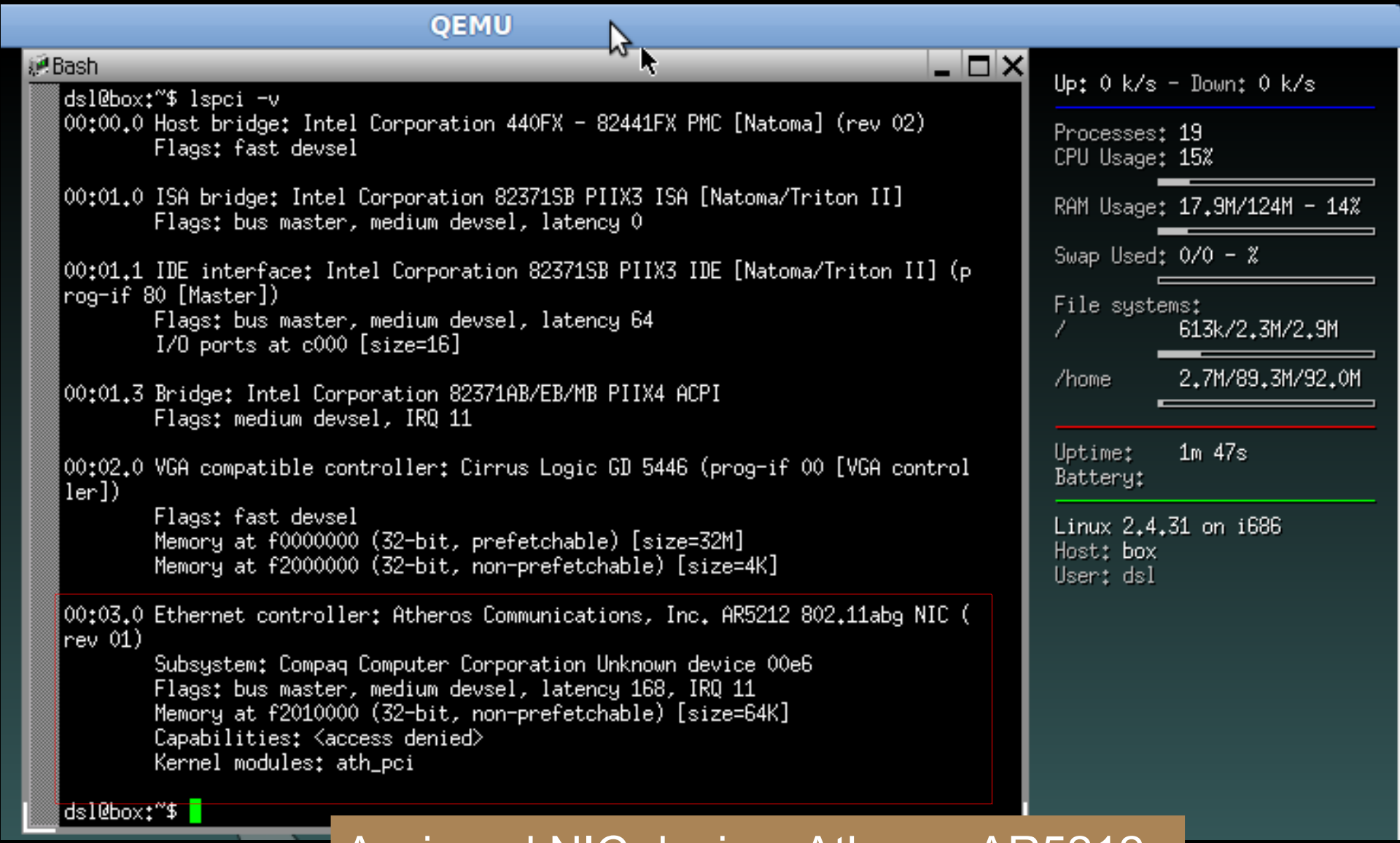
QEMU

```
QEMU 0.9.1 monitor - type 'help' for more information
(qemu) info pci
  Bus  0, device   0, function 0:
    Host bridge: PCI device 8086:1237
  Bus  0, device   1, function 0:
    ISA bridge: PCI device 8086:7000
  Bus  0, device   1, function 1:
    IDE controller: PCI device 8086:7010
      BAR4: I/O at 0xc000 [0xc00f].
  Bus  0, device   1, function 3:
    Bridge: PCI device 8086:7113
      IRQ 11.
  Bus  0, device   2, function 0:
    VGA controller: PCI device 1013:00b8
      BAR0: 32 bit memory at 0xf0000000 [0xf1ffffff].
      BAR1: 32 bit memory at 0xf2000000 [0xf2000fff].
  Bus  0, device   3, function 0:
    Ethernet controller: PCI device 168c:0013
      IRQ 11.
      BAR0: 32 bit memory at 0xf2010000 [0xf201ffff].
(qemu)
```

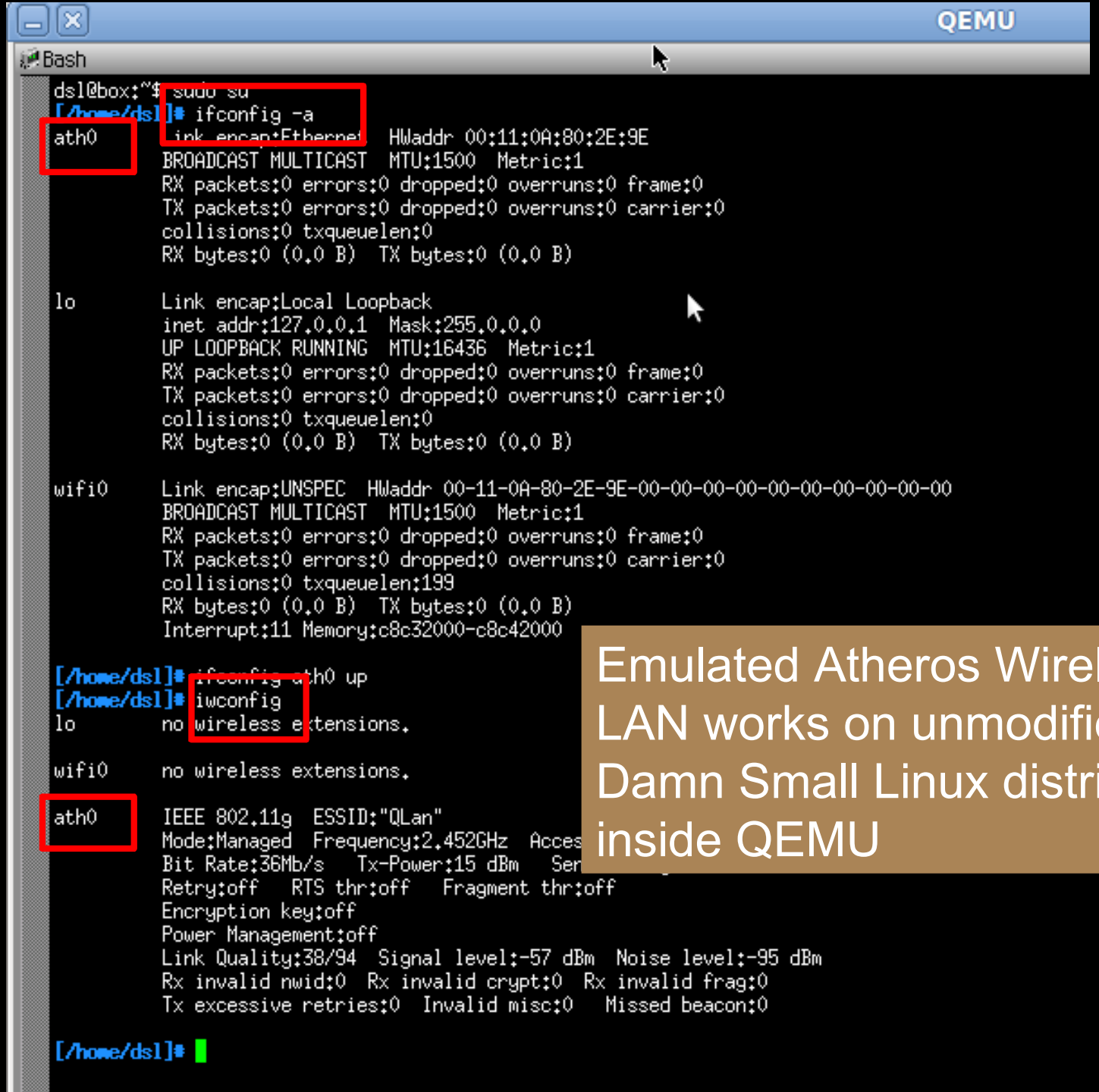
QEMU monitor

Assigned NIC device: Atheros AR5212
With proper implementation, even WirelessLAN device could be emulated.

Switch back to VGA screen
(under Linux)



Assigned NIC device: Atheros AR5212



Emulated Atheros Wireless LAN works on unmodified Damn Small Linux distribution inside QEMU

How does a NIC (network interface card) driver work?

Transmit path

Receive path

How does a NIC (network interface card) driver work?

Transmit path:

OS prepares packet to transmit in a buffer in memory
Driver writes **start address** of buffer to **register X** of the NIC
Driver writes **length** of buffer to **register Y**
Driver writes ‘1’ (**GO!**) into **register T**
NIC reads packet from memory addresses [X,X+Y) and sends it on the wire
NIC sends interrupt to host (**TX complete**, next packet please)

Receive path:

Driver prepares buffer to receive packet into
Driver writes **start address** of buffer to **register X**
Driver writes **length** of buffer to **register Y**
Driver writes ‘1’ (**READY-TO-RECEIVE**) into **register R**
When packet arrives, NIC copies it into memory at [X,X+Y)
NIC interrupts host (**RX**)
OS processes packet (e.g., wake the waiting process up)

The Emulated NIC

Emulator implements **virtual NIC** (by the specification of a real NIC like Atheros.

NIC **registers** (X, Y, Z, T, R, …) are just **variables** in emulator (host) **memory**

If **guest writes** ‘1’ to **register T**, **emulator reads** buffer from **memory [X,X+Y)** and passes it to **physical NIC** driver for transmission

When physical NIC interrupts (**TX complete**), emulator **injects** TX complete interrupt into guest

(Similar for RX path)

Pro:

Unmodified guest

(guest already has drivers for Atheros NIC)

Example: QEMU, KVM, VMware (without VMware Tools)

Cons:

Slow – every access to every NIC register causes a **VM exit**

Emulator needs to **emulate complex hardware behavior**

Para-virtualization

Add virtual NIC driver into guest (**frontend**)

Implement the virtual NIC in the hypervisor (**backend**)

Everything works just like in the emulation case except – **protocol** between frontend and backend

Protocol in emulation case:

Guest writes registers X, Y, waits at least 3 ns and writes to register T

Hypervisor **infers** guest wants to transmit packet

Paravirtual protocol can be **high-level**, e.g., ring of buffers to transmit (so NIC doesn't stay idle after one transmission), and **independent of particular NIC** registers

Pro:

Fast – no need to emulate physical device

Con:

Requires **guest driver**

Example: QEMU, KVM, VMware (with VMware Tools), Xen

How is paravirtual I/O different from paravirtual guest?

Paravirtual guest requires to modify whole OS

→ Try doing it on Windows (without source code), or even Linux (lots of changes)

→ Paravirtual I/O requires the addition of a single driver to a guest

→ Easy to do on both Windows and Linux guests

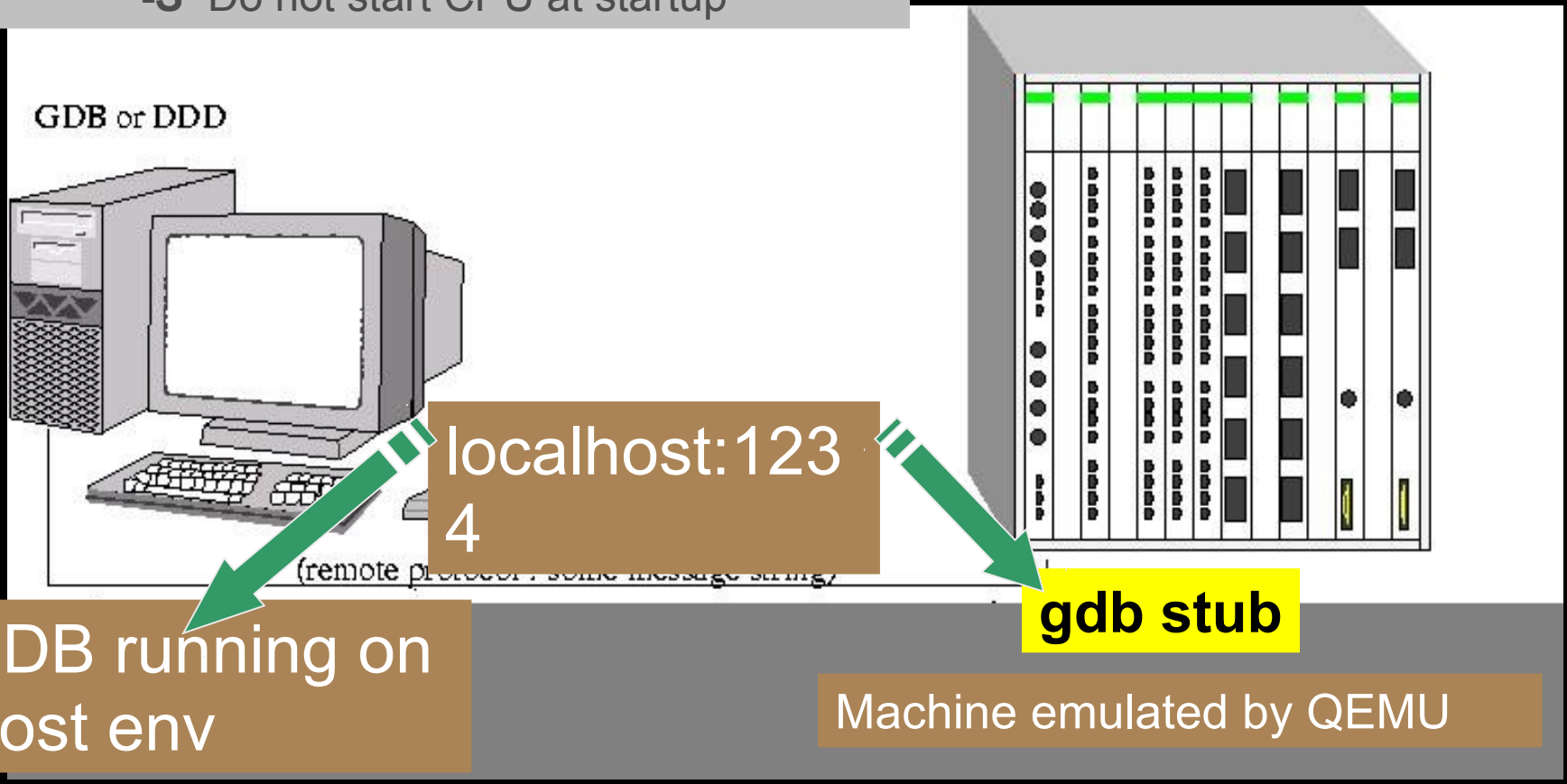
QEMU Debugging

- GDB stub is non-intrusive and controllable from first executed instruction
- Provide hardware breakpoint and watchpoint
- Fully integrated into QEMU Monitor

QEMU Remote Debugger

gdb stub – through CP/IP

- (gdb) target remote localhost:1234
- QEMU options：
  - **-s** Wait gdb connection to port 1234.
  - **-S** Do not start CPU at startup



QEMU monitor

Press Ctrl-Alt-2 after QEMU window appears to switch to QEMU monitor

```
jserv@venux:~
pxa2xx_clkpwr_write: CPU frequenc
##########################################
#       Start CuRT....
##########################################
 ID        State          Name
           Ready          idle-thre
  1        Ready          shell
  2        Running        info_
  3        Ready          stati
  4        Ready          hellc
  5        Ready          hellc
********** CuRT statistics i
Total Thread Count : 6
Total Context Switch Count :
Current Time Tick : 0
**********
$
```

Press Ctrl-C in GDB to get controlled

```
static void shell_thread_func(void *pdata){
    …
  While (1) {
        ...
        gets(buf);
```

QEMU [Stopped]

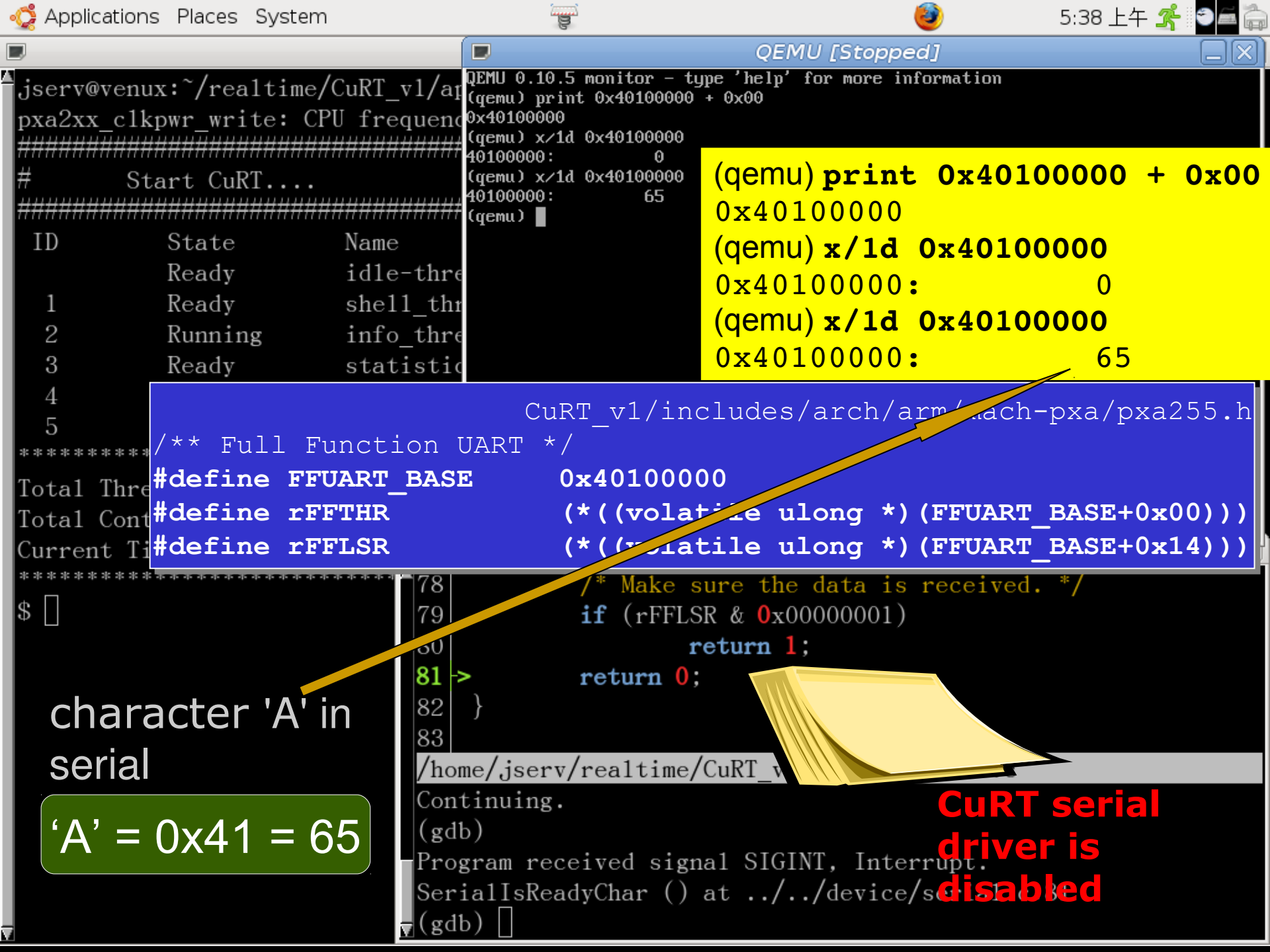
```
QEMU 0.10.5 monitor - type 'help' for more information
(qemu) info registers
R00=00000000 R01=00000020 R02=00000020 R03=00000000
R04=04040404 R05=05050505 R06=06060606 R07=07070707
R08=08080808 R09=09090909 R10=10101010 R11=a000c438
R12=12121212 R13=a000c42c R14=a0001828 R15=a0000bbc
PSR=60000013 -ZC- A svc32
(qemu)
```

(qemu) **info registers**

```
 75   */
 76  int SerialIsReadyChar(void)
 77  {
 78          /* Make sure the data is received
 79          if (rFFLSR & 0x00000001)
 80                  return 1;
 81->        return 0;
 82  }
 83
 84  /**
 85   * @brief Receives a character from serial
 86   * @retval
 87   */
```

CuRT_v1/device/serial.c

```
al SIGINT, Interrupt.
SerialIsReadyChar () at ../../device/serial.c:81
(gdb)
```


Applications Places System
5:38 上午
QEMU [Stopped]
QEMU 0.10.5 monitor - type 'help' for more information
(qemu) print 0x40100000 + 0x00
0x40100000
(qemu) x/1d 0x40100000
40100000: 0
(qemu) x/1d 0x40100000
40100000: 65
(qemu)
jserv@venux:~/realtime/CuRT_v1/ap
pxa2xx_clkpwr_write: CPU frequenc
Start CuRT....
ID State Name
Ready idle-thre
1 Ready shell_thr
2 Running info_thre
3 Ready statistic
4
5
Total Thre
Total Cont
Current Ti
$
(qemu) print 0x40100000 + 0x00
0x40100000
(qemu) x/1d 0x40100000
0x40100000: 0
(qemu) x/1d 0x40100000
0x40100000: 65
CuRT_v1/includes/arch/arm/mach-pxa/pxa255.h
/** Full Function UART */
#define FFUART_BASE 0x40100000
#define rFFTHR (*((volatile ulong *)(FFUART_BASE+0x00)))
#define rFFLSR (*((volatile ulong *)(FFUART_BASE+0x14)))
78 /* Make sure the data is received. */
79 if (rFFLSR & 0x00000001)
return 1;
81 -> return 0;
82 }
83
/home/jserv/realtime/CuRT_v
Continuing.
(gdb)
Program received signal SIGINT, Interrupt.
SerialIsReadyChar () at ../../device/se
(gdb)
character 'A' in serial
'A' = 0x41 = 65
CuRT serial driver is disabled

QEMU Mainline Status

- 0.9.1 (Jan 6, 2008)
  - Stable and stop for a long time
- 0.10 (Mar 5, 2009)
  - TCG support (a new general JIT framework)
- 0.11 (Sep 24, 2009)
  - KVM support
- 0.12
  - More KVM support
  - Code refactoring
  - new peripheral framework to support dynamic board configuration

Reference

- QEMU: http://www.qemu.org/
- Fabrice Bellard, QEMU, a Fast and Portable Dynamic Translator, USENIX 2005 Annual Technical Conference, FREENIX Track Pp. 41–46 of the Proceedings
- KVM: http://www.linux-kvm.org/