Freedom Embedded

Balau's technical blog on open hardware, free software and security

Compile Linux kernel 3.2 for ARM and emulate with QEMU

Posted on 2012/03/31

114

This tutorial is an updated version of this old post (https://balau82.wordpress.com/2010/03/22/compiling-linux-kernel-for-qemu-arm-emulator/), with newer software and less obsolete emulated hardware.

Every year the market produces tons of new products (http://en.wikipedia.org/wiki/List_of_applications_of_ARM_cores) that run on ARM cores, and are able to run operating systems such as Linux. While most of these products are quite expensive (think about smartphones, development kits or evaluation boards) it's possible to explore the world of Linux on ARM freely, thanks to software emulators like QEMU.

I am going to show how to compile the kernel and emulate the boot. To simplify things, the boot will not include a complete filesystem but uses a minimal ramdisk to show the kernel executing just one program.

I chose to emulate the Versatile Express (http://www.arm.com/products/tools/development-boards/versatile-express/index.php) product because it's well supported both by the mainline Linux kernel and by mainline QEMU. Moreover, this hardware platform runs on the Cortex-A9 core, which is an ARM CPU that is included in many smartphones today.

Requirements

In order to follow the same steps that I did, you need some tools.

First of all, anything I do is performed on a Linux machine, specifically a Debian testing distribution, in a `bash (http://www.gnu.org/software/bash/)` shell.

To manage the kernel compilation, GNU `make` (http://www.gnu.org/software/make/) should be installed (it is usually in build-essential package)

To compile the kernel for ARM architecture, a cross-compiler must be installed. The difference between a traditional compiler and a cross-compiler is that the traditional compiler runs on an architecture (for example x86_64) and produces binaries for the same architecture. A cross-compiler produces binaries for a different architecture (in our case ARMv7). Depending on your distribution and what works for your setup, you can choose from different toolchains:

- Emdebian (http://www.emdebian.org/crosstools.html); here some instructions (http://wiki.debian.org/EmdebianToolchain) on how to install,
- Linaro (https://launchpad.net/gcc-linaro); if you run a newish version of Ubuntu you can install it directly with "`sudo apt-get install gcc-arm-linux-gnueabi`",
- Fedora ARM cross-toolchain, (http://fedoraproject.org/wiki/Architectures/ARM/CrossToolchain)
- Sourcery Codebench (http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/) (was CodeSourcery); available for free only under registration,
- other toolchain suggestions (http://elinux.org/Toolchains) by eLinux.

Cross-compilers offer a set of programs, mainly GCC (http://gcc.gnu.org/) and binutils (http://www.gnu.org/software/binutils/), that start with a prefix indicating the architecture, the operating system of the libraries and the binary interface of the compiled programs. In my case I use Emdebian toochain, which has the "`arm-linux-gnueabi-`" prefix.

Finally the emulator that I use is QEMU, in particular the program to emulate ARM hardware is "`qemu-system-arm`". You must install the correct package depending on your distribution; sometimes distributions split the QEMU programs into different packages, for example Ubuntu packs it into the "`qemu-extras`" package.

The short story

Create a clean directory, then create a file called "`init.c`", which contains the following simple C code:

```
1  #include <stdio.h>
2
3  void main() {
4    printf("Hello World!\n");
5    while(1);
6  }
```

Then in the same directory execute the following commands in order:

```
1  wget http://www.kernel.org/pub/linux/kernel/v3.0/linux-3.2.tar.bz2 (http
2  tar xjf linux-3.2.tar.bz2
3  export ARCH=arm
```

```
export CROSS_COMPILE=arm-linux-gnueabi-
cd linux-3.2
make vexpress_defconfig
make all
cd ..
arm-linux-gnueabi-gcc -static init.c -o init
echo init|cpio -o --format=newc > initramfs
qemu-system-arm -M vexpress-a9 -kernel linux-3.2/arch/arm/boot/zImage -i
```

The kernel compilation (the "`make all`" command) could take some minutes or hours depending on your host machine power.

The last command opens a QEMU window, that shows a black background and many boot messages, and towards the end the "Hello World!" string is displayed.

The long story

The steps are:

1. Get Linux kernel source code
2. Prepare for compilation
3. Configure and compile
4. Prepare and create ramdisk
5. Emulate kernel boot and ramdisk execution

Get Linux kernel source code

The official site for mainline Linux kernel is at www.kernel.org (http://www.kernel.org/). The kernel version that I will use is the 3.2, be aware that if you want to use a different version you may have different results, even though most of the functionality used here is simple enough that it should not change between versions.

Download linux-3.2.tar.bz2 from the FTP site (ftp://ftp.kernel.org/pub/linux/kernel/v3.x/), or simply run from the command line:

```
wget http://www.kernel.org/pub/linux/kernel/v3.0/linux-3.2.tar.bz2 (http:
```

Then extract the kernel source. One way to do it is by running in the same directory:

```
1 tar xjf linux-3.2.tar.bz2
```

This will create a new subdirectory called linux-3.2 containing the full source of the Linux kernel.

Prepare for compilation

We are going to compile for ARM architecture by using a cross-toolchain, so we need to tell it somehow to the Linux build system. There are two environmental variable for this: `ARCH` and `CROSS_COMPILE`. The valid values for `ARCH` are basically the subdirectories of the "arch" directory. For `CROSS_COMPILE` we need to provide the prefix of the toolchain, which is the name of the compiler program minus the `gcc` at the end. For example if we are using `arm-linux-gnueabi-gcc`, we need to set `CROSS_COMPILE` to `arm-linux-gnueabi-`. On the terminal, run:

```
1 export ARCH=arm
2 export CROSS_COMPILE=arm-linux-gnueabi-
```

Configure and compile

We want to compile for the Versatile Express, and for this we can use the prepared configuration file in "`arch/arm/configs/vexpress_defconfig`" by running:

```
1 make vexpress_defconfig
```

This will configure the compilation for the desired hardware, by creating a file called "`.config`" that contains all the relevant options.

Then, to compile the kernel image, the command is simply:

```
1 make all
```

At the end of compilation it creates a file in "`linux-3.2/arch/arm/boot/zImage`" that is a compressed kernel image that auto-extracts in RAM. To speed-up compilation on multi-core hosts I suggest trying the parallel compilation, by launching

```
1 make -j 2 all
```

which, instead of compiling sequentially, will use a parallelism of 2 to create the objects and the final images.

Prepare and create ramdisk

In order to make the kernel do something, we can create a simple "Hello-World" user-space program. We can use a ramdisk as the first filesystem that Linux uses as root, using the "`initramfs`" scheme. More information about ramdisks can be found in the kernel source tree, in the file "`Documentation/early-userspace/README`". The first program that Linux tries to execute is "`/init`", so we can create an executable with that name. The source code is simply:

```
1  #include <stdio.h>
2
3  void main() {
4    printf("Hello World!\n");
5    while(1);
6  }
```

And it will be compiled by our cross-toolchain of choice. In order to make this program work alone, we need to compile it as a static executable, which links in the program also the libraries that it needs. In this way we can have a filesystem with only one executable without worrying about shared libraries. The command for compilation is:

```
1  arm-linux-gnueabi-gcc -static init.c -o init
```

This will create an executable for ARM called `init`. The format of the file is something like the following:

```
1  $ file init
2  init: ELF 32-bit LSB executable, ARM, version 1 (SYSV), statically linked
```

We can now create our ramdisk with the `cpio` utility, by adding just the `init` file to a new archive:

```
1  echo init|cpio -o --format=newc > initramfs
```

Be aware that the command must be run in the same directory as the `init` file. The `initramfs` file is our ramdisk. You can check its content with:

```
1  $ cpio -t < initramfs
2  init
3  1090 blocks
```

Emulate kernel boot and ramdisk execution

We have all that is needed to execute the Linux boot with QEMU, which is able to emulate the Versatile Express platform using the "`-M vexpress A9`" option. The `zImage` kernel and `initramfs` image are loaded by QEMU in the emulated RAM with the "`-kernel`" and "`-initrd`" options, pointing to the corresponding files.

We also want to display the boot messages which are sent on the console. To show them in the graphic window, we need to pass the "`console=tty1`" kernel parameter. The kernel parameters will be passed to Linux by QEMU using the "`-append`" option.

The complete command is then:

```
1 qemu-system-arm -M vexpress-a9 -kernel linux-3.2/arch/arm/boot/zImage -in
```

The command will launch QEMU and open a black window, with a Tux logo to show the graphic capabilities. The boot messages will be displayed in the graphic window, and at the end of the messages our "`Hello World!`" string will be printed.

Otherwise, QEMU can redirect the serial port of the emulated system on the host terminal, using the "`-serial stdio`" option, and Linux can display its messages on the first serial port by passing "`console=ttyAMA0`" as a kernel parameter. The command becomes:

```
1 qemu-system-arm -M vexpress-a9 -kernel linux-3.2/arch/arm/boot/zImage -in
```

The command will launch QEMU and open the black graphical window, and the boot messages will be displayed in the host terminal instead of the black window. Note that "`ttyAMA0`" is a serial port name that is dependent on the hardware that is emulated, and may not be the same for all systems.

In my tests I used QEMU version 1.0, and the result may vary if you are using a different version.

Tagged: *ARM*, *codesourcery*, *cortex-A9*, *cross compiling*, *emdebian*, *gcc*, *gnu*, *initramfs*, *kernel*, *linaro*, *linux*, *open source*, *qemu*, *ramdisk*, *versatile express*

Posted in: *Embedded (https://balau82.wordpress.com/category/software/embedded-software/)*

114 Responses "Compile Linux kernel 3.2 for ARM and emulate with QEMU" →

Nate

2012/03/31

Very helpful as usual. However, I believe that Linux should be compiled with arm-none-eabi (the "bare-metal" flavor). The reason being that libc links printf to stdout which only the kernel can set. For instance, you can use newlib (or redlib, or whatever) which is for code where there is no stdin/stdout but provides hooks for them such as _write() or __sys_write(). http://sourceware.org/newlib/libc.html#Syscalls

Balau

2012/03/31

Bare metal toolchains have the purpose of compiling code that is not running on an underlying OS, and the kernel itself is running on bare metal. So I agree that the bare metal toolchain should be used in theory, but in practice an arm-linux-… toolchain is OK too because the kernel source does not call those system functions, it implements its own (printk, kmalloc, …) so there are no userspace libraries linked into the kernel.

See also this blog post: http://marcin.juszkiewicz.com.pl/2010/10/19/how-to-cross-compile-arm-kernel-under-ubuntu-10-10/
Marcin Juszkiewicz is one of the most active developers for cross-toolchains in Canonical and Linaro projects, and surely he knows about these topics more than I do. He has no problems using an arm-linux-… toolchain to compile the kernel.

lp

2012/04/16

Thanks for the tutorial… but following the steps I get to this:

user@localhost:~/work/test/$ qemu-system-arm -M vexpress-a9 -kernel linux-3.2/arch/arm/boot/zImage -initrd initramfs -append "console=tty1"
qemu: fatal: Unimplemented cp15 register write (c9, c12, {0, 2})

R00=20000093 R01=00000000 R02=00000001 R03=80000000
R04=8047b304 R05=ffffffff R06=8059c168 R07=20000093
R08=8042b68c R09=00000000 R10=00000000 R11=00000000

R12=00000001 R13=87827f70 R14=80016384 R15=80016400
PSR=60000093 -ZC- A svc32
Aborted

I run on an Ubuntu Natty x86_64, qemu-system 0.14.50-2011.03-1-0ubuntu2, gcc-arm-linux-gnueabi 4:4.5.2-8
Any ideas would be appreciated.

Balau

2012/04/16

I have a newer QEMU and this doesn't happen, maybe the newer versions have implemented that instruction.
If trying with an updated QEMU does not work, I suggest one of two things:
– running it with "-d in_asm,cpu -D ./qemu.log -singlestep" options, to make it generate a log that can help you trace what happened before the crash
– running it with "-s -S" options and attaching with arm-linux-gnueabi-gdb, using "target remote localhost:1234" and debugging with that.

Rishi Agrawal

2012/05/09

Very good and helpful post, my qemu did not tun in one attempt as it did not had support for the machine. I had to perform the following steps which I read at http://www.cnx-software.com/2011/03/15/qemu-linaro-versatile-express-image-on-ubuntu-10-10/ to get the support.

sudo add-apt-repository ppa:linaro-maintainers/tools
sudo apt-get update
sudo apt-get install qemu-user-static qemu-system

Thanks a lot/

HC

2012/05/09

Thanks for the tutorial. I have been trying to run the test as rootfs but have not been successful. The linux kernel appears to be boot correctly, but crash with the following. Is there something obvious that I am not doing right?

#qemu-system-arm -M versatilepb -nographic -kernel uImage -initrd ./rootfs -append "root=/dev/ram rw rdinit=/test console=ttyAMA0"

…
smc91x.c: v1.1, sep 22 2004 by Nicolas Pitre
eth0: SMC91C11xFD (rev 1) at c8800000 IRQ 25 [nowait]
eth0: Ethernet addr: 52:54:00:12:34:56
mice: PS/2 mouse device common for all mice
TCP cubic registered
NET: Registered protocol family 17
VFP support v0.3: implementor 41 architecture 1 part 10 variant 9 rev 0
Freeing init memory: 108K
Kernel panic – not syncing: Attempted to kill init!
Backtrace:
[] (dump_backtrace+0x0/0x110) from [] (dump_stack+0x18/0x1c)
r6:c03020ec r5:c03020ec r4:c0315404
[] (dump_stack+0x0/0x1c) from [] (panic+0x5c/0x180)
[] (panic+0x0/0x180) from [] (do_exit+0x68/0x60c)
r3:60000013 r2:c781be48 r1:c7819be0 r0:c02c0f60
r7:c7819be0
[] (do_exit+0x0/0x60c) from [] (do_group_exit+0x94/0xc8)
r7:c79ce560
[] (do_group_exit+0x0/0xc8) from [] (get_signal_to_deliver+0x334/0x36c)
r4:0830009f
[] (get_signal_to_deliver+0x0/0x36c) from [] (do_signal+0x70/0x614)
[] (do_signal+0x0/0x614) from [] (do_notify_resume+0x20/0x54)
[] (do_notify_resume+0x0/0x54) from [] (work_pending+0x24/0x28)
r4:00000000

Balau

2012/05/09

"Attempted to kill init!" means that your "`/test`" executable is exiting somehow.
The most common cause I see is because the kernel has not been configured with the CONFIG_AEABI option.

lp

2012/05/28

it worked, thanks…

Shantanu Sharma

2012/06/18

m getting the same error.. kernel pani !! attempted to kill init…

Balau

2012/06/19

Have you checked that CONFIG_AEABI is enabled?

sourav

2012/06/22

I am trying to do the same (as mentioned above) for cortex a-15 versatile express board with linux kernel 3.4.3.I think linux-3.4.3 supports cortex a-15.However I am not able to see any boot messages being displayed after the last step as it gets displayed for a9.The console pops up and is empty.Do i need to modify anything when compiling for cortex a-15 vexpress ?

Balau

2012/06/22

The Cortex-A15 version of the Versatile Express is too different from the A9 that I used in my post. I just tried something that seems to work:
– download the 3.4.3 kernel
– configure it for vexpress_defconfig but also add support for Device Tree (CONFIG_ARCH_VEXPRESS_DT)
– create a device tree blob as indicated here:
http://www.linaro.org/members/arm/ve_12.04#tab2
– compile the kernel as in the post
– run in QEMU with something like:
qemu-system-arm -M vexpress-a15 -serial stdio -dtb ./v2p-ca15-tc1.dtb -kernel arch/arm/boot/zImage -append "console=ttyAMA0"

The graphic console (CLCD emulation) doesn't seem to work for me right now, so i redirect the console on the serial port.

sourav

2012/06/23

Thanks for the reply.I did the steps as mentioned by you.However when I give qemu-system-arm -M vexpress-a15 -serial stdio -dtb ./v2p-ca15-tc1.dtb -kernel arch/arm/boot/zImage -append "console=ttyAMA0" I get
"Device tree requested, but qemu was compiled without fdt support"

So I removed the dtb option and ran the command again.Now it displays
“Uncompressing Linux… done, booting the kernel.”
The qemu console pops up but I cant see any activity there.

Balau

2012/06/23

Giving the device tree is necessary, it won’t work without it.
Device trees contain information about the platform hardware and the memory map.

With a newer QEMU it will work. Here’s the version I’m using:

```
$ qemu-system-arm --version
QEMU emulator version 1.1.0 (Debian 1.1.0+dfsg-1), Copyright (c) 2003-2008
Fabrice Bellard
```

sourav

2012/06/25

Thanks a lot!It finally worked.I build qemu 1.1.0 with fdt support enabled and then used the command
qemu-system-arm -M vexpress-a15 -serial stdio -dtb ./v2p-ca15-tc1.dtb -kernel arch/arm/boot/zImage -append “console=ttyAMA0

Kernel panic occured.So I used qemu-system-arm -M vexpress-a15 -kernel linux-3.2/arch/arm/boot/zImage -initrd initramfs -serial stdio -append “console=ttyAMA0″.
The kernel booted and I saw Helloworld on my console!

sourav

2012/06/25

Sorry I used this command instead of the above mentined one
qemu-system-arm -M vexpress-a15 -dtb ./v2p-ca15-tc1.dtb -kernel linux-3.4.3/arch/arm/boot/zImage -initrd initramfs -serial stdio -append “console=ttyAMA0″

som

2012/06/26

Hi,

very Nice tutorial. I have been tring to do the same with a custom initramfs in which i want to run init (script file) as init. but i am getting oops “failed to execute /init” .
I have executed below command —

qemu-system-arm -M vexpress-a9 -kernel linux-3.2/arch/arm/boot/zImage -initrd initramfs -serial stdio -append “console=ttyAMA0″

here is the error log —

[…]
aaci-pl041 mb:aaci: ARM AC’97 Interface PL041 rev0 at 0x10004000, irq 43
aaci-pl041 mb:aaci: FIFO 512 entries
ALSA device list:
#0: ARM AC’97 Interface PL041 rev0 at 0x10004000, irq 43
oprofile: using arm/armv7-ca9
TCP cubic registered
NET: Registered protocol family 17
VFP support v0.3: implementor 41 architecture 3 part 30 variant 9 rev 0
input: ImExPS/2 Generic Explorer Mouse as /devices/mb:kmi1/serio1/input/input1
rtc-pl031 mb:rtc: setting system clock to 2012-06-26 10:12:06 UTC (1340705526)
Freeing init memory: 172K
Failed to execute /init
Kernel panic – not syncing: No init found. Try passing init= option to kernel. See Linux Documentation/init.txt for guidance.
[] (unwind_backtrace+0x0/0xf8) from [] (panic+0x5c/0x184)
[] (panic+0x5c/0x184) from [] (init_post+0xa0/0xc4)
[] (init_post+0xa0/0xc4) from [] (kernel_init+0x13c/0x16c)
———-

PS : CONFIG_AEABI is enabled

Any help will be appreciated.

Balau

2012/06/26

Common causes that I can think:
– the “`/init`” file does not have the execution bit enabled (fix with “`chmod +x init`” before creating the initramfs)
– the beginning of “`/init`” script contains as the “shebang” a line that can’t be executed. For example “#!/bin/bash” when bash is not present.

som

2012/06/27

Now its working. the issue was due to the /bin/busybox which was dynamically linked binary and /bin/sh was sym link to that. when i build the busybox as static, the problem was resolved. Thanks for your response.

Mix

2012/06/28

Hi, this post is very good and I especially like that every step is explained in detail. I have downloaded and compiled kernel, compiled small application (famous "hello world" example) and created ramdisk, following every step.

However, neither I see the Linux booting, nor the black window with Tux in the upper left corner is displayed.

When I execute the command:

qemu-system-arm -M vexpress-a9 -kernel ../linux-3.2/arch/arm/boot/zImage -initrd initramfs –append "console=tty1"

I see only the following messages in the same terminal window:

oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
audio: Failed to create voice `lm4549.out'
VNC server running on `127.0.0.1:5900'
l2x0_priv_write: Bad offset 900
l2x0_priv_write: Bad offset 904
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
audio: Failed to create voice `lm4549.out'

and after that nothing happens, I can only terminate qemu with "Ctrl-C"

I'm using Ubuntu 12.04 LTS, kernel version 3.2.0-25-generic, as a virtual machine run by Virtual Box on Windows 7.

Qemu version is 1.1.0

Do you have any idea what could be wrong? I'm not sure if the fact that Ubuntu is run as a virtual machine could be causing this behaviour. I'm thinking of using "pure" Linux machine instead of virtual one.

Thanks a lot in advance

Balau

2012/06/28

I think you could have the same errors if running "`qemu-system-arm -M vexpress-a9 -kernel /dev/null`", and if confirmed then your problems are not related to the way you created zImage or initramfs.
The problem is most likely in QEMU, it doesn't seem to have neither sound (/dev/dsp) nor display (it opens a VNC server) available.
I don't know if using it in a virtual machine could create problems, probably it's the cause of lack of audio; but even without audio it should be able to run your command, the other problem is that it doesn't show the display but opens a VNC server instead.
This usually happens when the display can't be opened (run xterm to disprove this possibility) or when QEMU has not been compiled with SDL libraries support. You could try to recompile from source making sure that graphic support is enabled.

Mix

2012/06/29

Thanks Balau!

Yes, you were right. I got the same result with "qemu-system-arm -M vexpress-a9 -kernel /dev/null".

First I had to install libsdl1.2-dev and then to configure Qemu like this:

./configure –enable-sdl

Now Qemu starts properly, but the kernel panics. It complains about not being able to open root device. Here's a part of printouts in Qemu:

Root-NFS: no NFS server address
VFS: Unable to mount root fs via NFS, trying floppy.
VFS: Cannot open root device "(null)" or unknown-block(2,0)
Please append a correct "root=" boot option; here are the available partitions:
Kernel panic – not syncing: VFS: Unable to mount root fs on unknown-block(2,0)
[] (unwind_backtrace+0x0/0xf8) from [] (panic+0x5c/0x184)

[] (panic+0x5c/0x184) from [] (mount_block_root+0x170/0x224)
.
.
.

I have tried with the command:

qemu-system-arm -M vexpress-a9 -kernel ../linux-3.2/arch/arm/boot/zImage -initrd initramfs –append “root=/dev/ram”

But without success. Do you have any suggestion?

Balau

2012/06/29

- You could append “console=ttyAMA0” to the current kernel parameters (and QEMU must be launched with “-serial stdio“) to display more info on the terminal.
– You could remove “root=/dev/ram” from kernel parameters.
– Try to find a line in the middle of the log that contains “Trying to unpack rootfs image as initramfs…“. The lines printed just before and just after could contain useful hints about why the kernel isn’t mounting the filesystem.

Mix

2012/06/29

Hi, I have caught longer log but I couldn’t find any info that could give me a clue what can be wrong.

The command was:

qemu-system-arm -M vexpress-a9 -kernel ../linux-3.2/arch/arm/boot/zImage -initrd initramfs -serial stdio -append “console=ttyAMA0”

In the log I just see “Unpacking initramfs…” and then again the message of not being able to mount root fs.

Here’s the part of the log, sorry for it being too long, but I just want to make things clear:

NET: Registered protocol family 1
RPC: Registered named UNIX socket transport module.
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
RPC: Registered tcp NFSv4.1 backchannel transport module.
Unpacking initramfs…

```
Freeing initrd memory: 444K
JFFS2 version 2.2. (NAND) © 2001-2006 Red Hat, Inc.
msgmni has been set to 244
io scheduler noop registered (default)
clcd-pl11x ct:clcd: PL111 rev2 at 0x10020000
clcd-pl11x ct:clcd: CT-CA9X4 hardware, XVGA display
v2m_cfg_write: writing 03c8eee0 to 00110001
v2m_cfg_write: writing 00000000 to 00710000
v2m_cfg_write: writing 00000002 to 00b10000
Console: switching to colour frame buffer device 128×48
smsc911x: Driver version 2008-10-21
smsc911x-mdio: probed
smsc911x smsc911x: eth0: attached PHY driver [Generic PHY] (mii_bus:phy_addr=ffffffff:01,
irq=-1)
smsc911x smsc911x: eth0: MAC Address: 52:54:00:12:34:56
isp1760 isp1760: NXP ISP1760 USB Host Controller
isp1760 isp1760: new USB bus registered, assigned bus number 1
isp1760 isp1760: Scratch test failed.
isp1760 isp1760: can't setup
isp1760 isp1760: USB bus 1 deregistered
isp1760: Failed to register the HCD device
Initializing USB Mass Storage driver…
usbcore: registered new interface driver usb-storage
USB Mass Storage support registered.
mousedev: PS/2 mouse device common for all mice
rtc-pl031 mb:rtc: rtc core: registered pl031 as rtc0
mmci-pl18x mb:mmci: mmc0: PL181 manf 41 rev0 at 0x10005000 irq 41,42 (pio)
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
input: AT Raw Set 2 keyboard as /devices/mb:kmi0/serio0/input/input0
aaci-pl041 mb:aaci: ARM AC'97 Interface PL041 rev0 at 0x10004000, irq 43
aaci-pl041 mb:aaci: FIFO 512 entries
ALSA device list:
#0: ARM AC'97 Interface PL041 rev0 at 0x10004000, irq 43
oprofile: using arm/armv7-ca9
TCP cubic registered
NET: Registered protocol family 17
VFP support v0.3: implementor 41 architecture 3 part 30 variant 9 rev 0
rtc-pl031 mb:rtc: setting system clock to 2012-06-29 11:40:58 UTC (1340970058)
input: ImExPS/2 Generic Explorer Mouse as /devices/mb:kmi1/serio1/input/input1
Root-NFS: no NFS server address
VFS: Unable to mount root fs via NFS, trying floppy.
VFS: Cannot open root device "(null)" or unknown-block(2,0)
Please append a correct "root=" boot option; here are the available partitions:
Kernel panic – not syncing: VFS: Unable to mount root fs on unknown-block(2,0)
```

[] (unwind_backtrace+0x0/0xf8) from [] (panic+0x5c/0x184)
[] (panic+0x5c/0x184) from [] (mount_block_root+0x170/0x224)
[] (mount_block_root+0x170/0x224) from [] (mount_root+0xbc/0xdc)
[] (mount_root+0xbc/0xdc) from [] (prepare_namespace+0x128/0x180)
[] (prepare_namespace+0x128/0x180) from [] (kernel_init+0x138/0x16c)
[] (kernel_init+0x138/0x16c) from [] (kernel_thread_exit+0x0/0x8)

Balau

2012/06/29

What is the content of initramfs?
Try to run "`cpio -t < initramfs`", it should display the "`init`" file without directory. If Linux does not see the "`/init`" file, the initramfs is not used and the kernel tries to mount root and panics.
So you should try to re-create the initramfs correctly.

Mix

2012/06/29

Hi,

The content of initramfs is just "hello", an executable file with "Hello World" example, made in accordance with your instructions:

cpio -t < initramfs
hello
892 blocks

I have re-created it, just to be sure that everything is ok, but the situation was the same.

Luckily, I took a look in older version of this post and I have found some differences in startup command, so I tried this one:

qemu-system-arm -M vexpress-a9 -kernel ../linux-3.2/arch/arm/boot/zImage -initrd initramfs -append "rdinit=/hello console=tty1"

That "rdinit=/hello" has made the difference and now the kernel mounts initramfs and I got the long-waited "Hello World!" in Qemu window.

Why was that rdinit=/hello necessary?

Thanks Balau!

Balau

2012/06/29

You missed a little thing
In my instruction the executable is called “init” and that makes the difference.
As stated in Linux “`Documentation/early-userspace/README`”, the “initramfs” way of booting Linux expects that the ramdisk is a cpio archive, it mounts it and then tries to execute “/init“.
In your case you don’t have “`/init`” but “`/hello`” so you have two options:
– use “`rdinit=/hello`” as you did
– create “init” instead of “hello” as in my blog post

Mix

2012/07/02

Oh… god… That’s what you get when you try to be “creative”

Thanks once again for this explanation. I’m glad that I made Qemu work one way or another. Without you that wouldn’t be possible.

Mix

2012/07/03

Hi,

I am stuck with another step in my work with Qemu. I’m trying to build initramfs, the one that will have shell up and running when I invoke Qemu.

So far, I’ve been using very simple initramfs that contained only one single executable.

I have tried a few possible ways of creating initramfs, but none of them has made initramfs that could be loaded, i.e. recognized by the current kernel.

1.
mkinitramfs -o initramfs

I then renamed and unzipped this filesystem (I’ve found that instruction somewhere)

mv initramfs initramfs,gz
gunzip initramfs.gz

Now with cpio -t initramfs

Again, I could see initramfs's contents with cpio -t < initramfs, with /init being executable (I have changed its permissions prior to creating filesystem). Unfortunately, I didn't manage to see if the script is working because kernel panicked with the same message: "No init found."

When I renamed my executable ("Hello World") to /init, and rebuild the filesystem, then everyting works fine, i.e. I got "Hello World" message in Qemu window.

3.
The last option I tried was to use usr/gen_init_cpio in linux-3.2 kernel that I'm using from the very begining, and whose image is provided to Qemu:

gen_init_cpio initramfs_list

where initramfs_list is a file containing only one line:

file /init /my_examples/hello 777 0 0

After executing

gen_init_cpio initramfs_list

I just get rubbish on the screen, and I don't see that anything was created.

Do you have an idea what could be wrong in any of these cases?

What would you instruct me to use as the most appropriate way for creating initramfs?

Balau

2012/07/03

1. I don't use mkinitramfs but from the man page I understand that images created by mkinitramfs must be used on Linux boxes of the same architecture of the Linux that launched the command. So, a mkinitramfs launched on x86 can't work on ARM.

2. I don't see an explicit point 2. I suppose it's the "hello world" that works.

3. Try to redirect the output to a file such as "`gen_init_cpio initramfs_list > initramfs`"

I don't have the audacity to declare "The most appropriate" way for creating initramfs. There's more than one method, pick the one you are most comfortable and successful with. In my opinion you have three easy options:
1. use cpio directly like I do in my tutorials
2. use gen_init_cpio and redirect the output
3. use "make menuconfig" and use "General Setup -> Initramfs source file(s)"

Mix

2012/07/04

Damn, it looks as if my post was corrupted somehow. Here are the first two points again, just to make everything clear:

1.
mkinitramfs -o initramfs

I then renamed and unzipped this filesystem (I've found that instruction somewhere)

mv initramfs initramfs,gz
gunzip initramfs.gz

Now with cpio -t initramfs

Again, I could see initramfs's contents with cpio -t < initramfs, with /init being executable (I have changed its permissions prior to creating filesystem). Unfortunately, I didn't manage to see if the script is working because kernel panicked with the same message: "No init found."

When I renamed my executable ("Hello World") to /init, and rebuild the filesystem, then everyting works fine, i.e. I got "Hello World" message in Qemu window.

Please let me know if you have any comments on point, as it's complete now.

Thank you for your replies.

Mix

2012/07/04

It looks corrupted again… let's try one more time…

1. mkinitramfs -o initramfs
I then renamed and unzipped this filesystem (I've found that instruction somewhere)

mv initramfs initramfs,gz
gunzip initramfs.gz

Now with cpio -t < initramfs I could see how this system looked like and there was /init that was executable.
However, when I launched Qemu with initrd pointing to this initramfs, kernel wrote a message "No init found"

Mix

2012/07/04

And now point 2:

2. I have created filesystem by myself, i.e. I have made very simple directory structure (/dev, root, /lib,…) in a separate directory. I have added my executable “hello” in /bin directory, and /init was a busybox shell script that was just supposed to mount rootfs.

The directory containing filesystem was used for creating initramfs like this:
find . -print0 | cpio –null -vo –format=newc > initramfs

Again, I could see initramfs’s contents with cpio -t < initramfs, with /init being executable (I have changed its permissions prior to creating filesystem). Unfortunately, I didn't manage to see if the script is working because kernel panicked with the same message: "No init found. "When I renamed my executable ("Hello World") to /init, and rebuild the filesystem, then everyting works fine, i.e. I got "Hello World" message in Qemu window.

Balau

2012/07/04

Wait, the /init script you have is just supposed to mount rootfs?
Could it be that the /init file is executed and mounts the rootfs and then the kernel panics because it can’t find the “init” of the rootfs?

Mix

2012/07/04

Yes, I think you’re right.

In the /init script there is:

mount -o ro /dev/sda1 /mnt/root
exec switch_root /mnt/root /sbin/init

I think I should’ve added device node(s)… (/dev/sda1, for example). Am I right?

Balau

2012/07/04

I don't know if you are right.
Maybe the /dev/sda1 node already exists in the initramfs, then the rootfs gets mounted correctly and the switch is performed, but you have no file called "/sbin/init" in the rootfs.
You could try to substitute the /init script with a link to /bin/sh and discover it yourself, and launch the commands directly from the shell.

Alto Stratus

2012/07/07

Thank you Balau, great tutorial as always! Keep it up!

hongwoo

2012/07/13

Hi~
Thank you for sharing your wisdom.

I succeeded to run init sample you said.
And I've tried to run my own small linux on qemu.
I just want to debug kernel using kgdb line by line.
This is what I did.

wget http://busybox.net/downloads/busybox-1.20.0.tar.bz2
tar xvjf busybox-1.20.0.tar.bz2
cd busybox-1.20.0
make defconfig
make install
cd _install
find . | cpio -o –format=newc > ../rootfs.img
cd ..
gzip -c rootfs.img > rootfs.img.gz

qemu-system-arm -M vexpress-a9 -m 1024M –snapshot -s -kernel ../linux/arch/arm/boot/zImage -initrd rootfs.img.gz -serial stdio -append "root=/dev/ram init=/sbin/init console=ttyAMA0 debug"

…..
rtc-pl031 mb:rtc: setting system clock to 2012-07-13 08:45:44 UTC (1342169144)
ALSA device list:
No soundcards found.
input: AT Raw Set 2 keyboard as /devices/mb:kmi0/serio0/input/input0
input: ImExPS/2 Generic Explorer Mouse as /devices/mb:kmi1/serio1/input/input1
List of all partitions:

No filesystem could mount root, tried: ext3 ext2 ext4 cramfs vfat msdos
Kernel panic – not syncing: VFS: Unable to mount root fs on unknown-block(1,0)
[] (unwind_backtrace+0x0/0xfc) from [] (panic+0x94/0x1d4)
[] (panic+0x94/0x1d4) from [] (mount_block_root+0x220/0x23c)
[] (mount_block_root+0x220/0x23c) from [] (mount_root+0xf8/0x100)
[] (mount_root+0xf8/0x100) from [] (prepare_namespace+0x190/0x1cc)
[] (prepare_namespace+0x190/0x1cc) from [] (kernel_init+0x224/0x238)
[] (kernel_init+0x224/0x238) from [] (kernel_thread_exit+0x0/0x8)
…..

And… using init=/bin/sh…,

qemu-system-arm -M vexpress-a9 -m 1024M –snapshot -s -kernel ../linux/arch/arm/boot/zImage -initrd rootfs.img.gz -serial stdio -append "root=/dev/ram init=/bin/sh console=ttyAMA0 debug"

VFP support v0.3: implementor 41 architecture 3 part 40 variant 0 rev 0
rtc-pl031 mb:rtc: setting system clock to 2012-07-13 08:46:05 UTC (1342169165)
ALSA device list:
No soundcards found.
input: AT Raw Set 2 keyboard as /devices/mb:kmi0/serio0/input/input0
input: ImExPS/2 Generic Explorer Mouse as /devices/mb:kmi1/serio1/input/input1
List of all partitions:
No filesystem could mount root, tried: ext3 ext2 ext4 cramfs vfat msdos
Kernel panic – not syncing: VFS: Unable to mount root fs on unknown-block(1,0)
[] (unwind_backtrace+0x0/0xfc) from [] (panic+0x94/0x1d4)
[] (panic+0x94/0x1d4) from [] (mount_block_root+0x220/0x23c)
[] (mount_block_root+0x220/0x23c) from [] (mount_root+0xf8/0x100)
[] (mount_root+0xf8/0x100) from [] (prepare_namespace+0x190/0x1cc)
[] (prepare_namespace+0x190/0x1cc) from [] (kernel_init+0x224/0x238)
[] (kernel_init+0x224/0x238) from [] (kernel_thread_exit+0x0/0x8)

Could you please tell me which part is going wrong way ??

Balau

2012/07/13

Have you tried "rdinit" instead of "init"?

If the ramdisk is mounted correctly, Linux searches for "/init" or whatever you pass with "rdinit" kernel parameter.
If it does not find the file, it mounts the root filesystem on "/" and searches for "/sbin/init" or whatever you pass with "init" function.
It seems that "root=/dev/ram" is not sufficient to mount the ramdisk as the root filesystem, but you should not need it.

Mary

2012/08/07

Hi,
I tried your method of running linux kernel on vexpress-a9 board on qemu. It ran successfully. The same when compiled for versatile board and when run on versatile it shows kernel panic. Why is this so? Is linux kernel and initramfs only meant to run in vexpress-a9? Doesnt it work on any other emulation boards?

Balau

2012/08/07

A kernel compiled for the versatile express will most certainly not work on other boards. There are mainly three possible areas for differences:
– the CPU (for example the versatile express has a Cortex-A9, the versatile has an ARM926EJ-S)
– the System-on-Chip architecture (peripherals, memory map, interrupts, …)
– the board (connections, displays, flash memories, …)
There is some effort around the world in developing a single kernel that can run on many boards and chips, using a “Device Tree” as a description for the hardware that the kernel uses to configure itself.

Mary

2012/08/08

Thank you so much Balau. This information was useful.

miloody

2012/08/12

hi:
Thanks for your kind sharing your knowledge.
after reading your website, I follow your instructions, but it cannot show “hello world” as what we add at init.c
I use “QEMU emulator version 1.1.1″

below is the instruction I execute my qemu
/media/sdb1/software/qemu/qemu-1.1.1/build/bin/qemu-system-arm -M vexpress-a9 –kernel linux-3.2/build.vexpress/arch/arm/boot/zImage -initrd initramfs.igz -serial stdio -append “console=ttyAMA0″

below is my log:

```
Uncompressing Linux… done, booting the kernel.
l2x0_priv_write: Bad offset 900
l2x0_priv_write: Bad offset 904
Booting Linux on physical CPU 0
Initializing cgroup subsys cpuset
Linux version 3.2.0 (root@NB) (gcc version 4.7.1 20120402 (prerelease) (crosstool-NG linaro-1.13.1-2012.04-20120426 – Linaro GCC 2012.04) ) #5 SMP Thu Jul 19 10:05:10 CST 2012
CPU: ARMv7 Processor [410fc090] revision 0 (ARMv7), cr=10c53c7d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine: ARM-Versatile Express
Memory policy: ECC disabled, Data cache writealloc
sched_clock: 32 bits at 24MHz, resolution 41ns, wraps every 178956ms
PERCPU: Embedded 7 pages/cpu @8085a000 s4864 r8192 d15616 u32768
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 32512
Kernel command line: console=ttyAMA0
PID hash table entries: 512 (order: -1, 2048 bytes)
Dentry cache hash table entries: 16384 (order: 4, 65536 bytes)
Inode-cache hash table entries: 8192 (order: 3, 32768 bytes)
Memory: 128MB = 128MB total
Memory: 121912k/121912k available, 9160k reserved, 0K highmem
Virtual kernel memory layout:
vector : 0xffff0000 – 0xffff1000 ( 4 kB)
fixmap : 0xfff00000 – 0xfffe0000 ( 896 kB)
vmalloc : 0x88800000 – 0xf8000000 (1784 MB)
lowmem : 0x80000000 – 0x88000000 ( 128 MB)
modules : 0x7f000000 – 0x80000000 ( 16 MB)
.text : 0x80008000 – 0x8041f2e0 (4189 kB)
.init : 0x80420000 – 0x80713300 (3021 kB)
.data : 0x80714000 – 0x807390c0 ( 149 kB)
.bss : 0x807390e4 – 0x80756bcc ( 119 kB)
SLUB: Genslabs=13, HWalign=32, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
Hierarchical RCU implementation.
NR_IRQS:128
Console: colour dummy device 80×30
Calibrating delay loop… 310.88 BogoMIPS (lpj=1554432)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
CPU0: thread -1, cpu 0, socket 0, mpidr 80000000
hw perfevents: enabled with ARMv7 Cortex-A9 PMU driver, 1 counters available
Brought up 1 CPUs
SMP: Total of 1 processors activated (310.88 BogoMIPS).
NET: Registered protocol family 16
L310 cache controller enabled
```

```
l2x0: 8 ways, CACHE_ID 0x410000c8, AUX_CTRL 0x02420000, Cache size: 131072 B
hw-breakpoint: debug architecture 0x0 unsupported.
Serial: AMBA PL011 UART driver
mb:uart0: ttyAMA0 at MMIO 0x10009000 (irq = 37) is a PL011 rev1
console [ttyAMA0] enabled
mb:uart1: ttyAMA1 at MMIO 0x1000a000 (irq = 38) is a PL011 rev1
mb:uart2: ttyAMA2 at MMIO 0x1000b000 (irq = 39) is a PL011 rev1
mb:uart3: ttyAMA3 at MMIO 0x1000c000 (irq = 40) is a PL011 rev1
bio: create slab at 0
SCSI subsystem initialized
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
Advanced Linux Sound Architecture Driver Version 1.0.24.
Switching to clocksource v2m-timer1
NET: Registered protocol family 2
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 4096 (order: 3, 32768 bytes)
TCP bind hash table entries: 4096 (order: 3, 32768 bytes)
TCP: Hash tables configured (established 4096 bind 4096)
TCP reno registered
UDP hash table entries: 128 (order: 0, 4096 bytes)
UDP-Lite hash table entries: 128 (order: 0, 4096 bytes)
NET: Registered protocol family 1
RPC: Registered named UNIX socket transport module.
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
RPC: Registered tcp NFSv4.1 backchannel transport module.
Unpacking initramfs…
Freeing initrd memory: 472K
JFFS2 version 2.2. (NAND) 2001-2006 Red Hat, Inc.
msgmni has been set to 239
io scheduler noop registered (default)
clcd-pl11x ct:clcd: PL111 rev2 at 0x10020000
clcd-pl11x ct:clcd: CT-CA9X4 hardware, XVGA display
v2m_cfg_write: writing 03c8eee0 to 00110001
v2m_cfg_write: writing 00000000 to 00710000
v2m_cfg_write: writing 00000002 to 00b10000
Console: switching to colour frame buffer device 128×48
smsc911x: Driver version 2008-10-21
smsc911x-mdio: probed
smsc911x smsc911x: eth0: attached PHY driver [Generic PHY] (mii_bus:phy_addr=ffffffff:01,
irq=-1)
smsc911x smsc911x: eth0: MAC Address: 52:54:00:12:34:56
isp1760 isp1760: NXP ISP1760 USB Host Controller
```

isp1760 isp1760: new USB bus registered, assigned bus number 1
isp1760 isp1760: Scratch test failed.
isp1760 isp1760: can't setup
isp1760 isp1760: USB bus 1 deregistered
isp1760: Failed to register the HCD device
Initializing USB Mass Storage driver…
usbcore: registered new interface driver usb-storage
USB Mass Storage support registered.
mousedev: PS/2 mouse device common for all mice
rtc-pl031 mb:rtc: rtc core: registered pl031 as rtc0
mmci-pl18x mb:mmci: mmc0: PL181 manf 41 rev0 at 0x10005000 irq 41,42 (pio)
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
audio: Failed to create voice `lm4549.out'
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
audio: Failed to create voice `lm4549.out'
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
audio: Failed to create voice `lm4549.out'
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
audio: Failed to create voice `lm4549.out'
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
oss: Could not initialize DAC

oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
audio: Failed to create voice `lm4549.out'
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
audio: Failed to create voice `lm4549.out'
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
audio: Failed to create voice `lm4549.out'
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
audio: Failed to create voice `lm4549.out'
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
oss: Could not initialize DAC
oss: Failed to open `/dev/dsp'
oss: Reason: No such file or directory
audio: Failed to create voice `lm4549.out'
aaci-pl041 mb:aaci: ARM AC'97 Interface PL041 rev0 at 0x10004000, irq 43
aaci-pl041 mb:aaci: FIFO 512 entries
ALSA device list:
#0: ARM AC'97 Interface PL041 rev0 at 0x10004000, irq 43
oprofile: using arm/armv7-ca9
TCP cubic registered
NET: Registered protocol family 17
VFP support v0.3: implementor 41 architecture 3 part 30 variant 9 rev 0
rtc-pl031 mb:rtc: setting system clock to 2012-08-12 15:46:05 UTC (1344786365)
Freeing init memory: 3020K
input: AT Raw Set 2 keyboard as /devices/mb:kmi0/serio0/input/input0
input: ImExPS/2 Generic Explorer Mouse as /devices/mb:kmi1/serio1/input/input1

Balau

2012/08/13

A couple of things you could try:

- Remove the "while(1)" from the program. If the kernel does not panic (attempting to kill init), then your program is not really executed.
- Check that the "\n" is there, otherwise it will not flush
- Print also on stderr with fprintf
- Recompile QEMU without sound support, to remove those "oss: Failed to open /dev/dsp" errors.
- Do not use -nographic, and use console=tty1 to see if it prints something in the black window.

sanal

2012/09/14

I am getting the error below….

qemu: fatal: Unimplemented cp15 register write (c9, c12, {0, 2})

R00=00000001 R01=00000000 R02=00000001 R03=0000001f
R04=804b35c4 R05=ffffffff R06=805d4168 R07=20000093
R08=804646c8 R09=00000000 R10=00000000 R11=00000000
R12=80498e78 R13=87827f70 R14=80000000 R15=80016c90
PSR=60000093 -ZC- A svc32
Aborted

Balau

2012/09/14

R15 indicates the program counter and R14 the link register (which is strangely at `0x80000000`)
To check where the execution has gone, you could disassemble the kernel with "`arm-linux-gnueabi-objdump -dS linux-3.2/vmlinux > vmlinux.dis`" and check what is the instruction (and the function) at `0x80016c90`.
What is the version of QEMU that you are using? Mine is 1.0.
What toolchain have you used? I used the Emdebian toolchain.
In general, did you do something differently with respect to my instructions?

touzzie

2012/09/22

could you explain step no. 10 under title "The short story" ?

Balau

2012/09/23

If you are referring to the "`echo init|cpio -o --format=newc > initramfs`" line, then the reasoning is the following:
I want to create a filesystem so that the Linux kernel can mount it at boot as a ramdisk.
Linux understands a filesystem format that can be created with the "`cpio`" utility with the "`--format=newc`" option.
`cpio` works by accepting from standard input a list of file paths, and writes on standard output the filesystem image.
So I am using the redirect bash operator "`>`" to make `cpio` write on a file instead of the terminal. I decided to call this file "`initramfs`" because it's a ramdisk.
Instead of sending the file paths to `cpio` from standard input, I decided to use the pipe operator "`|`" to send input from the preceding command.
The "`echo init`" command simply prints "`init`" which is the filepath of the "`init`" executable that we just created. I could have written "`./init`", too, because the executable is in the same directory where we are launching `cpio`.
So the command means "call `cpio` to create a filesystem containing the `init` file from the current directory, and write the filesystem in a file image called `initramfs`".

adriano

2012/11/28

hey, do you know if it is possible to communicate the QEMU with the hyperterminal of the computer? How should I configure the UART for the communication with Windows?

Thank in advance

Balau

2012/11/28

You could try something like com0com to create two virtual serial ports COMxx and COMyy, then launch qemu with "-serial COMxx" option and connect hyperterminal to COMyy.
You could also try redirecting it to telnet and then using something like Putty to connect.
I never tried QEMU on windows so I can't be sure that these solutions work.

adriano

2012/11/29

Hello Balau;
Thanks for answering. I am using the QEMU in Linux, not in Windows, but I want to communicate it with windows using the serial, so I don't know if that com0com would work… So my option was to use the physical serial port from linux with QEMU and later receive the data of the physical port in the hyperterminal. Do you think that is possible?

Balau

2012/11/29

Theoretically it should work, adding something like "-serial /dev/ttyS0" to QEMU options to connect to a physical serial port.
The help says:

> *-serial*
> *[…]*
> */dev/XXX*
> *[Linux only] Use host tty, e.g. /dev/ttyS0. The host serial port parameters are set according to the emulated ones.*

Be aware of file permissions, usually being in "dialout" group is necessary.
Then you connect a physical RS232 cable between Linux PC and Windows PC and open hyperterminal on that COM port using the parameters set from QEMU guest system.

adriano

2012/11/30

Ok, thank you, I will try to do that!

gowtham

2012/12/05

Hi balau, thanks for tutorial.
I am executing following line :
qemu-system-arm -M versatilepb -m 128M -kernel linux-3.2/arch/arm/boot/zImage -initrd initramfs -serial stdio -append "console=ttyAMA0"

I've configured kernel for versatilepb only.
the final output is…
Uncompressing Linux… done, booting the kernel.

and its hanged.

what went wrong???

Balau

2012/12/05

In my post I compiled the kernel for Versatile Express. If you try to run the same kernel on Versatile Platform Baseboard it won't run.

gowtham

2012/12/06

will it not run even if i configure and compile the kernel for versatile platform board…? I'm using qemu 1.3.0 and it says it doesnot support versatile express…

gowtham

2012/12/06

i'm sorry… my qemu version is different i guess. this is wat it is..(i installed it through sudo-apt-get install qemu-kvm-extras)

qemu-system-arm –version
QEMU PC emulator version 0.12.3 (qemu-kvm-0.12.3), Copyright (c) 2003-2008 Fabrice Bellard

Balau

2012/12/06

If you compile a kernel for the Versatile Platform Baseboard and run it on an emulated Versatile Platform Baseboard it should work.
My suggestion is to try to replicate exactly the setup that I described in this post: "Compiling Linux kernel for QEMU ARM emulator", with kernel version 2.6.33, and then try to understand what's different from yours. The version of QEMU that I used in that post is similar to yours (0.12.5+noroms-0ubuntu7.5).

gowtham

2012/12/06

hey got it..:) thanks:)
installed latest version of qemu from its source code and emulated for vexpress platform.

but in old version of qemu, getting output for versatilepb platform for linux 2.6.33 but for same steps with kernel linux 3.2 its failing.. is it kernel specific??

Balau

2012/12/06

It could be a version-specific kernel regression, I don't know. Maybe the Versatile Platform Baseboard is old and not supported very well. In order to understand the problem maybe the fastest way to do it is use gdb to attach to QEMU and see what is executing.

touzzie

2012/12/09

Thanks for the nice artricle !
Keep doing …

psychesnet

2013/01/12

Hello Balau,
I am studying qemu wiith arm, but I have some trouble need your help, thank a lot.
1. if I cat u-boot.bin uImage u-rootfs.cramfs.bin.gz > flash.bin,
and I get start address 0x24EAC 0x1CBBAC

VersatilePB # iminfo 0x24EAC 0x1CBBAC

Checking Image at 00024eac …
Legacy image found
Image Name: Linux-3.6.10
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 1731776 Bytes = 1.7 MB
Load Address: 00008000
Entry Point: 00008000
Verifying Checksum … OK

Checking Image at 001cbbac …
Legacy image found
Image Name: uboot ramdisk
Image Type: ARM Linux RAMDisk Image (uncompressed)
Data Size: 3183359 Bytes = 3 MB
Load Address: 03000000
Entry Point: 03000000

Verifying Checksum … OK
VersatilePB # bootm 0x24EAC 0x1CBBAC
……
Kernel command line: console=ttyAMA0 mem=128M root=/dev/ram0
ip=192.168.1.150:192.168.1.118:192.168.1.1:255.255.255.0
……..
List of all partitions:
1f00 131072 mtdblock0 (driver?)
No filesystem could mount root, tried: cramfs vfat msdos romfs
Kernel panic – not syncing: VFS: Unable to mount root fs on unknown-block(1,0)
—-> What's wrong with my system ????

2. if I want to use NAND simulator partition 0, do you have any idea ?
3. if I want to modify mtd mapping driver of NAND simulator partition, do you know where I can
modify it ?
4. I use qemu-img create -f raw hda.img 10M to create a hard disk, and run
qemu-system-arm -M versatilepb -kernel uImage -initrd rootfs.cramfs.bin.gz -nographic -append "console=ttyAMA0 root=/dev/ram0" -hda hda.img
Uncompressing Linux… done, booting the kernel.
……
rc.sysinit start ok.
rc.local start ok.
~ # ls /dev
~ # …..
—> Why I can not see any hda or sda device? May I wrong???

Thanks a lot.

Balau

2013/01/12

1. I did something similar in my post "Booting Linux with U-Boot on QEMU ARM", if you want to take a look. What I see is that we use different addresses, make sure that the kernel is not overwriting the ramdisk when you launch bootm, or something like that. I can't see the command you use when you launch qemu so I don't know if that may be a problem. Also, when an initramfs is mounted, "/init" is called (or anything specified by rdinit=xxx kernel parameter) so be sure that the script exists and is executable.

2, 3. don't know, sorry. You may try to ask qemu IRC channels or mailing lists.

4. In my post "Trying Debian for ARM on QEMU" I used a kernel and root filesystem that were able to use "/dev/sda". Try to analyze that filesystem and kernel to understand what's the difference. It's possible that you should add a module (or more) in your kernel that is currently missing.

flashman

2013/01/30

Hi Balau,

thanks for the post and sorry if I resume an old post

The tutorial was perfect and I started to navigate and to try linaro img from the linaro ws. Also ubuntu and android.

In linaro webpage there is a short overview to run qemu using :
qemu-system-arm -kernel vmlinuz-2.6.37-1003-linaro-vexpress -M vexpress-a9 -cpu cortex-a9 -serial stdio -m 1024 -initrd initrd.img-2.6.37-1003-linaro-vexpress -append 'root=/dev/mmcblk0p2 rw mem=1024M raid=noautodetect console=ttyAMA0,38400n8 rootwait vmalloc=256MB devtmpfs.mount=0' -sd vexpress.img

As you see the -sd option is used as secure digitale img. Is possible to "unpack" the original image of linaro "vexpress.img" and add some new file, for example apk file ?

On linaro there the procedure to build from scratch but requires too much space.

Thanks a lot!

Balau

2013/01/30

I'm sure there is a way to unpack, but I don't know it. I suppose it contains simply the data that an hard disk would have. If so, you can mount its partition as a loopback device. See here for an example: http://wiki.edseek.com/guide:mount_loopback

You could analyze the linaro-image-tools source code to understand what it is doing:
Here you can download the complete source code:
https://launchpad.net/ubuntu/raring/+source/linaro-image-tools

In particular this is the "main" of the linaro-media-create tool:

http://bazaar.launchpad.net/~ubuntu-branches/ubuntu/raring/linaro-image-tools/raring/view/head:/linaro-media-create

And this is the code that handles the partitions:

http://bazaar.launchpad.net/~ubuntu-branches/ubuntu/raring/linaro-image-tools/raring/view/head:/linaro_image_tools/media_create/partitions.py

Hope this helps.

flashman

2013/01/31

Thank you very much Balau.

I'll investigate through posted links.

flashman

flashman

2013/02/01

Hi Balau,

I found the solution. As you told me is loopback device with kpartx.

kpartx -a guest.img

In the /dev/mapper/ dir are recognized all partitions.

So you can mount one of this simply : mount /dev/mapper/loop0p1 mydir.

Adding all files or delete.

umount ….

And : kpartx -d guest.img disconnnect the image file from the partition mappings.

Thank you for your help.

Bye flashman.

PS : We are linkedin on linkedin!

Balau

2013/02/01

Glad you solved your issue!

Suresh

2013/02/04

Hi,
I have followed all the steps. But after following command

qemu-system-arm -M smdkc210 -kernel linux-3.7.5/arch/arm/boot/zImage -initrd initramfs -serial stdio -append “console=tty1″

I not getting QEMU window. Instead a message is showing

VNC server running on `127.0.0.1:5900`

What should be the cause?

Balau

2013/02/04

If you compiled the kernel yourself then you should recompile it adding SDL support.
It is possible that somewhat the Linux distribution that you are using prepared a qemu package without SDL support, but it’s unlikely.
Another remote possibility is that you are running it in a terminal that does not support graphic windows, but you could try running an “xterm” and see if something opens; if it does, it’s not the problem.

wa2707

2013/02/27

Hi Balau
Plz i need help i follow the tutorial step by step but in the final i have a black screen.
i try to compile a kernel from that link https://github.com/AndrewDB/rk3066-kernel.i'm using the compiler arm-none-eabi-gcc. i don’t know wher is the problem.
PLZ help me

Balau

2013/02/27

The problem is probably that you are compiling a kernel for rk3066 architecture but QEMU does not emulate it. In my tutorial I emulate the Versatile Express which is very different.
I don’t think there’s a QEMU that can emulate the rk3066 architecture; depending on what you are really trying to accomplish you could ask that kernel maintainer how to emulate it.

wa2707

2013/02/27

i tape this command qemu-system-arm -M versatilepb -m 128M -kernel Image -initrd rootfs -append “console=ttyAMA0 root=/dev/ram rdinit=/test” -nographic
but it still runnig, it doesn’t stop at all.when i want to close the terminal , a message is shown “There is still a process running in this terminal. Closing the terminal will kill it”
and the qemu doesn’t appear at all.

Balau

2013/02/27

Yeah it’s the “nographic” option. If you press Ctrl-A and then X it should close QEMU.
See here:

http://doc.opensuse.org/products/draft/SLES/SLES-kvm_sd_draft/cha.qemu.running.html#cha.qemu.running.devices.graphic.display

wa2707

2013/02/27

thank you for your fast answers
i have to hack linux kernel to adapt it on Rockchip 3066 which is a dual core processor type ARM Cortex A9 .
So according to your experience can you tell me:
*from where i should begin?
* what should i know about the kernel to accomplish this mission?
*what is the difference between configure kernel for arm cortex a9 and configure it for rk 3066?

Balau

2013/02/27

I don’t have experience on porting Linux, and I don’t think it’s an easy job.
You should search for Linux porting guides for ARM to help you.
It seems that Rockchip 3066 is similar to the Samsung Exynos 4, so I would start from that code base.
Maybe it’s a matter of creating a Rockchip 3066 Device Tree description and then passing it to the Linux kernel.
Otherwise, you need to change the kernel, by adding a new “mach” directory in “arch/arm/” and anything that needs to be done (as I said, I never did something like that).

To answer your last question, cortex-A9 is just the CPU. There's a whole architecture around the CPU, it's not like x86 where you run the same kernel on all the x86 you can find around. A better way to ask is "what is the difference between a kernel compiled for OMAP4430 and one for rk 3066". In order to make a kernel work for rk 3066 you need to compile it for Cortex-A9, and also you need to add the peripherals of the architecture. You need the rk 3066 reference manual and study the memory map and the programmer's model.

Hope this helps.

wa2707

2013/02/27

yup it's so helpful and it clear many steps in my mind
thank you so much

lax

2013/03/02

I want to run opencv on MeeGo OS. how to do it? Do I need cross-compilation of opencv for MeeGo??? plz reply

Balau

2013/03/03

I don't know how, I'm not experienced with MeeGo. It's very likely that cross-compilation is involved, but usually distributions have their own way to build. Maybe this is a good place to start: http://wiki.meego.com/Build_Infrastructure

lax

2013/03/03

what toolchain do I search for?????

Balau

2013/03/03

As I already said, I'm not experienced with MeeGo.
I suggest asking help of people who work with Meego, such as the official mailing list or IRC channels. See here: IRC / mailinglists / contact.
I suppose an arm-linux-gnueabi- toolchain could work (I am assuming that you need to compile for ARM even if you failed to mention it) but I'm just guessing.

lax

2013/03/04

okay, I will check it.
thanks!

Jerry

2013/04/10

It works like a charm only when using -no-graphic option you need to use ttyAMA0 ! I'm using Qemu 1.4.0

abhishek

2013/05/23

Hi balau, This post helped me a lot.
I need a little more help,
i need to connect a hdd.img
i tried the '-hda hdd.img' option. but there is no sda or hda files in the /dev on qemu booted.
I think i need to compile the kernel using scsi support. How to do that?

Balau

2013/05/26

I don't have the answer, I don't know exactly how QEMU emulates hard disks, and it also depends on what initial ramdisk you are using, for example.

My suggestion is to take a look at this tutorial that should contain what you want to do: Embedded Linux From Scratch… in 40 minutes!

You could also take a look at my blog post Trying Debian for ARM on QEMU and see what does the initrd contains (it's a gzipped cpio archive so you need to extract it with gunzip and then with "cpio -i").

ashiv

2013/05/30

Hello Balau, Great tutorial! I'm hoping you will help me out here.
After executing the last statement –> qemu-1.4.0$ qemu-system-arm -M vexpress-a9 -kernel linux-3.2/arch/arm/boot/zImage -initrd initramfs -serial stdio -append "console=tty1"
I'm getting an error in the form of –> l2x0_priv_write: Bad offset 900
l2x0_priv_write: Bad offset 904

would you happen to know what this is about?

Balau

2013/06/02

From a quick look it seems l2x0 is ARM L220 cache controller. From its reference manual it seems the 900 and 904 offsets are cache lockdown registers. From QEMU 1.4.0 source code it seems that those cache lockdown registers are not implemented. I have no experience on cache controllers, but I suppose you could try these things:
– use a different QEMU version (or wait for the new one that maybe has this behavior fixed)
– patch QEMU to add this support
– disable cache controller management in Linux
Hope this helps.

pm

2013/06/12

Hello Balau, thanks a lot for this tutorial!
I want to emulate the Xilinx Zynq Platform Baseboard for Cortex-A9 (-M xilinx-zynq-a9) using QEMU. So I can't use "make vexpress_defconfig" for configuring Linux? Which config do I need? Do I have to change anything else compared to this tutorial?
Thank you!

franc

2013/06/12

Hi Balau, Could you tell me what are the kernel options to enable for using serial capability on QEMU 1.4.0 emulating a BeagleBoard-xm machine? I read on QEMU's website that for cortex-A8 cpu's, PL011 UART where emulated. What I want to do is to communicate with the host machine using -serial flag. This is some options that a copied from a config file for qemu versatille (it's what I think relevent to the serial communication part):

CONFIG_SERIO_AMBAKMI=y CONFIG_SERIAL_8250=m CONFIG_SERIAL_8250_EXTENDED=y CONFIG_SERIAL_8250_MANY_PORTS=y CONFIG_SERIAL_8250_SHARE_IRQ=y CONFIG_SERIAL_8250_RSA=y CONFIG_SERIAL_AMBA_PL011=y CONFIG_SERIAL_AMBA_PL011_CONSOLE=y CONFIG_SERIAL_CORE=y CONFIG_SERIAL_CORE_CONSOLE=y #CONFIG_SERIAL_OF_PLATFORM is not set #CONFIG_SERIAL_OMAP is not set #CONFIG_SERIAL_OMAP_CONSOLE is not set Is there anything else that I should add or remouve becouse when the kernel boots, this is what I get: [2.366231] Serial: 8250/16550 driver, 4 ports, IRQ sharing enabled [2.455020] omap_uart.0: ttyO0 at MMIO 0x4806a000 (irq = 72) is a OMAP UART0 [2.472204] omap_uart.1: ttyO1 at MMIO 0x4806c000 (irq = 73) is a OMAP UART1 [2.492057] omap_uart.2: ttyO2 at MMIO 0x49020000 (irq = 74) is a OMAP UART2 [2.654465] console [ttyO2] enabled [2.672666] omap_uart.3: ttyO3 at MMIO 0x49042000 (irq = 80) is a OMAP UART3 I'm using Buildroot to create my image and I putted as one of the option that the login console where at ttyAMA0. As you can see, it keeps redirecting console to ttyO2. Also, this is what happen before the login question: INIT: Id "AMA0" respawning too fast: disabled for 5 minutes I' using kernel 3.2.8 for the target.

Balau

2013/06/12

@pm
I don't know if Linux mainline tree supports the Zynq, it seems that it doesn't.
Xilinx Linux tree seems to support it, you can try to download it:
https://github.com/Xilinx/linux-xlnx
You can try "make xilinx_zynq_defconfig" since there exists a file with that name in "arch/arm/configs/"
I know almost nothing about it so I don't know if it works.

avtop

2013/08/26

Hi Balau
I read BareMatal and this one also I complied 3 version of linux kernal but out 1 is only running in qemu, what I want to know is how decide compiler like:arm-none-eabi,arm-none-linux-eabi, uCLinux, and bare metal tool chains and is that make deference to compile linux kernel 2.6 under host linux kernel 3.0 or kernel 3.0 under host kernel 2.6 using ubuntu 9.10
thx for all , very very nice info for newbee
thx for all

Balau

2013/08/26

The "*-linux-*" toolchains are used to build user-space programs that run on Linux. The same with uCLinux, which is a similar but different operating system. They link the program with user-space libc libraries to implement C library functions (printf, fopen, …).
Bare metal toolchain are used to build programs that run without operating system below. They link the program with bare metal libraries and usually the developer needs to implement the system calls (open, read, write, …).
If you need to compile the Linux kernel you can use the bare metal toolchain because the kernel doesn't run on another operating system, but is an operating system in itself. It also works to compile it with linux toolchain because the linux kernel doesn't try to use C library functions such as printf and fopen. It may also compile with uCLinux toolchain but I never tried.

The version of the kernel of the host should not make a difference on the guest kernel that you compile. So if you want to compile a 3.0 kernel you can do it on a host with 2.6 kernel or on a host with 3.0 kernel.

avtop

2013/08/31

Hi Balau thx a lot for reply. I check on to computers with same and different kernel version and u-boot also,but result are not same latest version has error compiling, How can I check which version is good work on and main is is that any nice book or blog to know what is ram file system,(like how load linux) and from where to call my application after boot linux and main is I have wm8650 tab and I want use it for linux devel. platform so what is u-boot command or to set env to boot linux from MMC/USB any one, (to check without losing my NAND android) , I searched lot but didn't find any one who can help/explain in easy way to start as your post about linux startup thx a lot
Thx…….

Balau

2013/08/31

The first place to look is inside Linux kernel source Documentation folder. You can search for ramfs for example:

```
$ grep -R ramfs Documentation
```

For example in kernel 3.4.3 the file explaining ramfs is "`Documentation/filesystems/ramfs-rootfs-initramfs.txt`".

Free Electrons has some documentation too: http://free-electrons.com/docs/ , especially the "Embedded Linux from scratch…" slides.

I don't know where to find information about wm8650 and u-boot, but I suppose it's very specific to that hardware platform.

charles

2013/10/17

Hi, Balau,
i tried to test u-boot with qemu, there is no output, just a black screen.
I use u-boot code u-boot-2013.07.tar.bz2, and build as :
make CROSS_COMPILE=arm-linux-gnueabi- vexpress_ca9x4_config
make CROSS_COMPILE=arm-linux-gnueabi- all
then run
qemu-system-arm -M vexpress-a9 -m 256M -serial stdio -kernel u-boot
But i cannot see any output in qemu window.
My arm gcc toolchain is:
gcc version 4.7.1 20120402 (prerelease) (crosstool-NG linaro-1.13.1-2012.04-20120426 – Linaro GCC 2012.04)
Qemu version is:
qemu-system-arm –version
QEMU emulator version 1.6.50, Copyright (c) 2003-2008 Fabrice Bellard.
Do you know the reason ?
thank you!

Balau

2013/10/17

I tried the same with my setup, but QEMU crashed when executing an instruction that sets VBAR cp15. I have QEMU 1.1.2 which is older so that might be the problem (Debian Testing is a bit slow in these things).
My suggestion is to try to debug the problem by running:

```
$ qemu-system-arm -M vexpress-a9 -serial stdio -kernel u-boot -s -S
```

and then in another terminal (same directory):

```
$ arm-linux-gnueabi-gdb u-boot
...
(gdb) target remote localhost:1234
```

and then try to understand where the code is blocked, for example by running "`continue`" and Ctrl-C after a little while.
Keep in mind that the "`-m 256M`" option can affect program behavior because u-boot assumes

a certain memory map, and with that option you are changing the expected map, so I suggest removing it unless you are aware that you are doing it for a specific reason.

charles

2013/10/23

Hi, Balau,,
Thank you for your quick reply!
I tested using your suggested method and found that, the code is blocked when the following asm instruction is executed:
mcr p15, 0, r0, c12, c0, 0 @Set VBAR
(at file u-boot-2013.10/arch/arm/cpu/armv7/start.S).
But i don't know much about u-boot code, so it's difficult for me to figure out the cause….
Then i tried linaro qemu. With linaro qemu, u-boot can run correcly without problem. ^^

Balaji

2013/10/30

Hi Balau

I want to test a PL353 based SMC controller for supporting read and write operations of nand flash in qemu at the u-boot level. Though I got the PL353 model from the net but i dont know how to connect to Micron based nand flash. Please check the following link for pl353 model.

http://lists.gnu.org/archive/html/qemu-devel/2012-10/msg03383.html.

Can you tell me if it is possible to perform nand flash operations in qemu at the u-boot level using mtd(Memory technology devices) layer.

Thanks
balaji

Balau

2013/10/30

I don't know if I understand your question, it's confusing to me.
I know little about what you are talking about, so I am just taking a look and guessing what needs to be done.

From the link you provided it seems the model also incorporates SRAM and NAND model.
I don't understand what you say about Micron, but it seems you want to connect a certain nand (a new model?) to the pl353 model you found.

So in my opinion you probably have to remove the nand model inside that patch and place your own nand model instead. Unless the nand model inside the pl353 code is already compatible with the nand you want to connect. For sure the developer who wrote that code can help you more.

Searching u-boot-2013.07 code I can't see anything about pl35x controllers, so you probably have to write its mtd drivers.

I am not certain of what I have told you, I don't have the expertise you need.

I also feel the need to point out that your comments have little to do with the blog post you are commenting on.

Sourabh

2014/02/19

Hey ,
A great , simple guide for beginners like me
I am facing an issue as Kernel Panic :try passing init= to kernel… please help with complete command

Balau

2014/02/19

First things that come to mind: is the "`init`" file that is put into `initramfs` executable? Otherwise it needs to be set with "`chmod +x init`" before creating `initramfs`.
Also, what is displayed when you run "`cpio -t < initramfs`"?

LIKAN

2014/04/23

I try to do the same for cortex a-15. I use kernel v3.8.9. I have correted config (CONFIG_ARCH_VEXPRESS_DT), and I use device tree, wich was created in the linux kernel directory (arch/arm/boot/dts/vexpress-v2p-ca15-tc1.dtb) (Your link about creating device tree files is out of date) I use this launch script – http://pastebin.com/jyK32tYG . And I have this error log – http://pastebin.com/NCxZ0q9U How to fix these errors?

Balau

2014/04/23

Thanks for pointing out the broken link. About your error I don't know what is wrong, I never tried to emulate A15 because multi-processor is more complicated. I don't know what `physmap.enable=0` kernel parameter does, have you tried disabling it? I understand that `ksysmoops` utility might be useful to extract information from errors like yours. It seems the problem is generated in `kmem_cache_alloc` but also the message `GIC CPU mask not found - kernel will fail to boot` is suspicious. It seems to me a bug in the kernel or in a kernel module more than a problem in device trees, but I don't have enough experience about that to be sure.

foton

2014/05/01

I'm trying to emulate Fedora, with qemu, I get this error

audio: Could not init `oss' audio driver
VNC server running on 127.0.0.1:5901

How can I solve? Screen shot is here.
[https://docs.google.com/file/d/0B2uDS8XrCvL3ZmI1WHQ1VGs1WXc/edit]

Balau

2014/05/02

Usually I get the VNC line when I compile QEMU from source and I don't have SDL development libs installed. Basically QEMU is running but you can't see it because the screen is not displayed anywhere, unless you connect to QEMU with a VNC client. If you compiled it from source then you must first install the SDL development libraries and then recompile it. I don't know Fedora well so I don't know the exact steps. If you installed it from Fedora packages then it probably means Fedora has a problem with that package and that a bug should be opened to notify them of it; and in the meantime you can recompile QEMU as said before. Or if you install a VNC client, connect to QEMU and you see that it's something that works for you, then you found your work-around.
The audio issue is a separate problem but in my opinion can be ignored (the guest should start anyway) unless you need audio, in that case it will probably be a matter of playing with the `-soundhw` QEMU option or installing something like ALSA or PulseAudio on your host machine.

foton

2014/05/03

sir thanks for the reply! I'll compile again now qeme with those installed libraries to see what happens.

Abhishek

2014/08/19

Hi Balau,

Thanks for informative blog and comments.
I am trying to support i.mx6 on the qemu, using the i.mx6 build kernel for vexpress_a9 machine. It shows me following comment.
qemu: hardware error: pl181_write: Bad offset 10

CPU #0:
R00=10005010 R01=40400c12 R02=10008000 R03=10004000
R04=10008000 R05=00000000 R06=00000000 R07=000008e0
R08=60000100 R09=10000000 R10=20000000 R11=00100103
R12=60010068 R13=00000000 R14=600102ec R15=60010264
PSR=200001d3 –C- A svc32
Aborted

Any idea?

Thanks.

Balau

2014/08/19

I don't understand if you are building the kernel for i.mx6 or for vexpress_a9, and I don't understand if you are running qemu emulating an i.mx6 or a vexpress_a9.
If you are building the kernel for i.mx6 and running it on qemu emulating a vexpress_a9 it most probably won't work, because i.mx6 and vexpress_a9 most probably have different memory maps and peripherals.

abhishek

2014/08/20

Hi Balau,

Yes you are correct, I am building the kernel for i.mx6 and running it on qemu emulating a vexpressa9 machine.(It seems to be closest to i.mx6 as both use arm a9).

from your comment I understand that both of them will have different memory maps and peripherals, so it may not work.

Is there any way to support new machine type like i.mx6 in the qemu or to modify the existing machine tto support the same? Where shall i begin?

Thanks

Balau

2014/08/20

The right way to do it should be to develop in QEMU source code the emulation of i.mx6. I don't think it's an easy task, but I don't have much experience in QEMU source code. You should probably ask qemu-devel mailing list for information. See also this mail thread.

Abdul Lateef

2014/12/17

Simple and well explained, I will try to compile on QEMU ARM emulator…thanks for one pager, i was searching for these steps over internet for quite sometime.

3 Trackbacks For This Post

1. *Compiling Linux kernel for QEMU ARM emulator « Balau* →
   March 31st, 2012 → 11:25
   […] [EDIT] I have written a new updated version of this post here. […]

2. *Building a Linux distro | BlogoSfera* →
   May 21st, 2013 → 14:02
   […] world.c' for an arm machine. I gave up that. But After noon I found a blog to compile kernal here I compiled it successfully, and booted successfully Now it boots and shows a hello […]

3. *QEMU 1.5.0 released, a backward compatibility warning | Balau* →
   May 21st, 2013 → 20:51
   […] Compile Linux kernel 3.2 for ARM and emulate with QEMU […]