Cook a Mini Bootable Linux System

— Grub2 + Kernel + Busybox —

C.D.Luminate

cdluminate@gmail.com

May 3, 2015

Contents

1 Introduction

1.1 Tools in Need

Before you go ahead, you should check if you had these tools installed in your system[1]

1. build-essential, Depends:
   libc6-dev, gcc, g++, make, dpkg-dev
2. grub-common, grub2-common, grub-pc, grub-pc-bin
3. QEMU - fast processor emulator

2 Trace the Boot Process

Let's trace the bootup process of the machine.
And, this is just a brief trace.

2.1 Overall Process

```
* CPU Power up
- Load BIOS from CS:IP=FFFF:0000 Entry
- Load GRUB to 0x7c00 via int 0x19
- Load vmlinuz
- real mode : arch/x86/boot/header.S : _start
- read mode : arch/x86/boot/main.c
- protected mode (0x100000): arch/x86/boot/compressed/head_64.S
- protected mode : arch/x86/boot/compressed/head64.c
- arch independent : start_kernel ();
- create init rootfs : mnt_init ();
- kernel init : rest_init ();  kernel_init ();
- load initramfs : init/initramfs.c : populate_rootfs ();
+ if cpio initrd
    /init
  else if image initrd
    /linuxrc
  fi
- userspace init : /sbin/init
```

2.2 BIOS/EFI

BIOS/EFI reads the machine code at a fixed location on hard disk, typically sector 0, then execute it.
This piece of machine code belongs to boot loader.

2.3 GRUB2

2.3.1 grub stage 1

Read then execute the first 512 Bytes, then look for file systems.

2.3.2 grub stage 2

Grub loads grub.cfg, then loads *linux* and *initrd.img* (or *initramfs.img*) into memory, finally boot them.

[1] we assume your machine is running Debian or its variant.

2.4 linux

2.4.1 bzImage

You can look up kernel doc[?]. For example, `ARCH=x86_64`, find file `arch/x86/`:

```
boot/header.S:

    293 _start:
    294         # Explicitly enter this as bytes, or the assembler
    295         # tries to generate a 3-byte jump here, which causes
    296         # everything else to push off to the wrong offset.
    297         .byte   0xeb        # short (2-byte) jump
    298         .byte   start_of_setup-1f

    456 start_of_setup:
    457 +-- 51 lines: # Force %es = %ds----------------------------------------------
    508 # Jump to C code (should not return)
    509     calll   main

boot/main.c:

    135 void main(void)
    136 +-- 48 lines: {--------------------------------------------------------------
    184     go_to_protected_mode();
    185 }

boot/pm.c:

    104 void go_to_protected_mode(void)
    105 +-- 19 lines: {--------------------------------------------------------------
    124     protected_mode_jump(boot_params.hdr.code32_start,
    125                 (u32)&boot_params + (ds() << 4));
    126 }

boot/pmjump.S:

    26 GLOBAL(protected_mode_jump)
    27 +-- 18 lines: movl %edx, %esi  # Pointer to boot_params table---------------
    45 2:  .long   in_pm32         # offset

    51 GLOBAL(in_pm32)
    52 +-- 24 lines: # Set up data segments for flat 32-bit mode-------------------
    76     jmpl    *%eax           # Jump to the 32-bit entrypoint
    77 ENDPROC(in_pm32)
```

After executing pmjump.S, the Processor is in protected mode.

```
boot/compressed/head_64.S:

    37 ENTRY(startup_32)
    38 +--142 lines: 32bit entry is 0 and it is ABI so immutable!------------------
    180     pushl   $__KERNEL_CS
    181     leal    startup_64(%ebp), %eax
    ENTRY(startup_64)
    +-- 15 lines: 64bit entry is 0x200 and it is ABI so immutable!-------------
        jmp preferred_addr

    preferred_addr:
    +-- 61 lines: #endif-------------------------------------------------------
    /*
     * Jump to the relocated address.
     */
        leaq    relocated(%rbx), %rax
        jmp *%rax

    relocated:
    +-- 24 lines: Clear BSS (stack is currently empty)-------------------------
    /*
     * Do the decompression, and jump to the new kernel..
     */
        pushq   %rsi            /* Save the real mode argument */
        movq    $z_run_size, %r9    /* size of kernel with .bss and .brk */
        pushq   %r9
        movq    %rsi, %rdi      /* real mode address */
        leaq    boot_heap(%rip), %rsi   /* malloc area for uncompression */
        leaq    input_data(%rip), %rdx  /* input_data */
        movl    $z_input_len, %ecx  /* input_len */
        movq    %rbp, %r8       /* output target address */
        movq    $z_output_len, %r9  /* decompressed length, end of relocs */
        call    decompress_kernel   /* returns kernel location in %rax */
        popq    %r9
        popq    %rsi

boot/compressed/misc.c:

    asmlinkage __visible void *decompress_kernel(void *rmode, memptr heap,
    +-- 53 lines: unsigned char *input_data,-----------------------------------
        debug_putstr("\nDecompressing Linux... ");
        decompress(input_data, input_len, NULL, NULL, output, NULL, error);
        parse_elf(output);
        /*
         * 32-bit always performs relocations. 64-bit relocations are only
         * needed if kASLR has chosen a different load address.
         */
        if (!IS_ENABLED(CONFIG_X86_64) || output != output_orig)
            handle_relocations(output, output_len);
        debug_putstr("done.\nBooting the kernel.\n");
        return output;
    }

boot/compressed/head_64.S:

    relocated:
    +-- 36 lines: Clear BSS (stack is currently empty)-------------------------
        call    decompress_kernel   /* returns kernel location in %rax */
        popq    %r9
        popq    %rsi

    /*
     * Jump to the decompressed kernel.
    419  */
    420     jmp *%rax

kernel/head_64.S:

    49 startup_64:
    50 +--111 lines: At this point the CPU runs in 64bit mode CS.L = 1 CS.D = 0,---
    161     jmp 1f

    162 ENTRY(secondary_startup_64)
    163 +--122 lines: At this point the CPU runs in 64bit mode CS.L = 1 CS.D = 0,---
    285     movq    initial_code(%rip),%rax
    286     pushq   $0      # fake return address to stop unwinder
    287     pushq   $__KERNEL_CS    # set correct cs
    288     pushq   %rax        # target address in negative space
    289     lretq

    310     GLOBAL(initial_code)
    311     .quad   x86_64_start_kernel

kernel/head64.c:

    141 asmlinkage __visible void __init x86_64_start_kernel(char * real_mode_data)
    142 +-- 47 lines: {-------------------------------------------------------------
    189     x86_64_start_reservations(real_mode_data);
    190 }

    192 void __init x86_64_start_reservations(char *real_mode_data)
    193 +--  7 lines: {-------------------------------------------------------------
    200     start_kernel();
    201 }

../../init/main.c:

    489 asmlinkage __visible void __init start_kernel(void)
    490 {
```

2.4.2 vmlinux

see `linux-4.0/init/main.c`:

```
489 asmlinkage __visible void __init start_kernel(void)
490 +---183 lines: {-----------------------------------------------------------
673     /* Do the rest non-__init'ed, we're now alive */
674     rest_init();
675 }
```

trace `rest_init()`:

```
382 static noinline void __init_refok rest_init(void)
```

```
{
+--  8 lines: int pid;-----------------------------------------------------
    kernel_thread(kernel_init, NULL, CLONE_FS);
```

trace kernel_init():

```
static int __ref kernel_init(void *unused)
{
+-- 20 lines: int ret;-----------------------------------------------------
    /*
     * We try each of these until one succeeds.
     *
     * The Bourne shell can be used instead of init if we are
     * trying to recover a really broken machine.
     */
    if (execute_command) {
        ret = run_init_process(execute_command);
        if (!ret)
            return 0;
        panic("Requested init %s failed (error %d).",
              execute_command, ret);
    }
    if (!try_to_run_init_process("/sbin/init") ||
        !try_to_run_init_process("/etc/init") ||
        !try_to_run_init_process("/bin/init") ||
        !try_to_run_init_process("/bin/sh"))
        return 0;

    panic("No working init found.  Try passing init= option to kernel. "
          "See Linux Documentation/init.txt for guidance.");
}
```

It shows that, from here the kernel executes the init program as pid 1, then init program do the Operating System initialization things.

2.5 Busybox init

```
busybox-1.23.2/init/init.c:

    int init_main(int argc, char **argv) MAIN_EXTERNALLY_VISIBLE;
    int init_main(int argc UNUSED_PARAM, char **argv)
    +-- 98 lines: {------------------------------------------------------------
            parse_inittab();
        }

    static void parse_inittab(void)
    {
    #if ENABLE_FEATURE_USE_INITTAB
        char *token[4];
        parser_t *parser = config_open2("/etc/inittab", fopen_for_read);

    if (parser == NULL)
#endif
    {
        /* No inittab file - set up some default behavior */
        /* Sysinit */
        new_init_action(SYSINIT, INIT_SCRIPT, "");
        /* Askfirst shell on tty1-4 */
        new_init_action(ASKFIRST, bb_default_login_shell, "");
//TODO: VC_1 instead of ""? "" is console -> ctty problems -> angry users
        new_init_action(ASKFIRST, bb_default_login_shell, VC_2);
        new_init_action(ASKFIRST, bb_default_login_shell, VC_3);
        new_init_action(ASKFIRST, bb_default_login_shell, VC_4);
        /* Reboot on Ctrl-Alt-Del */
        new_init_action(CTRLALTDEL, "reboot", "");
        /* Umount all filesystems on halt/reboot */
        new_init_action(SHUTDOWN, "umount -a -r", "");
        /* Swapoff on halt/reboot */
        new_init_action(SHUTDOWN, "swapoff -a", "");
        /* Restart init when a QUIT is received */
        new_init_action(RESTART, "init", "");
        return;
    }

/* Default sysinit script. */
#ifndef INIT_SCRIPT
# define INIT_SCRIPT  "/etc/init.d/rcS"
#endif
```

3 Build Linux Kernel Image

3.1 Download kernel source

Pick a kernel from the linux kernel archives.[1]
Here I use the Debian redistributed one or linux 4.0 :

```
linux-3.16.7-ckt7-1
linux-4.0
```

Then extract it to the workplace :

```
$ tar zxvf linux-3.16.7-ckt7.tar.gz -C workplace/
$ tar zxvf linux-4.0.tar.gz -C workplace/
$ cd workplace
```

3.2 Configure the Kernel

To simplify the Procedure, I just used the default kernel config for AMD64 architecture, so type

```
$ cd workplace/linux-?/
$ make x86_64_defconfig
$ make menuconfig
```

Modify some configurations as you like, via menuconfig.[2]

3.3 Compile kernel

Lets compile the kernel. Maybe you should invoke "make help" at first.

```
$ make -j4 vmlinux
$ make -j4 bzImage
```

The process takes a long while.

3.4 The kernel

After compiling, the file "`arch/x86/boot/bzImage`" is exactly what we need.

```
bzImage: Linux kernel x86 boot executable bzImage,
    version 3.16.7-ckt7 (lumin@debian) #2 SMP Sat Mar 21 09:15:07 UTC 2015,
    RO-rootFS, swap_dev 0x5, Normal VGA
```

Put this kernel file at proper place.

4 Build Static Busybox

4.1 Download Busybox source

You can download busybox source on official site.[2]
Here I use Debian Redistributed one or another official one:

```
busybox-1.22.0-9+deb8u1
busybox-1.23.2.tar.bz2
```

Extract the source pack and change directory into source tree.

4.2 Configure Busybox

```
$ cd busybox-?/
$ make defconfig
$ make menuconfig
```

Set the "`CONFIG_STATIC=y`", namely mark
Busybox Settings - Build Options - ... Static Binary
You can also mark the "dpkg" or something else as you like.

[2] For detail please look up other materials.

4.3 Compile Busybox

```
$ make -j4 busybox
$ make install
```

Then you will see a fine rootfs under directory "_install/" .
Copy all the content of _install/ to workplace/initrd/:

```
# cd _install
# mkdir -p workplace/initrd
# cp -av . workplace/initrd/
# chown -R root:root workplace/initrd/
```

Now, Busybox preparation is completed. Lets Configure the system.

5 Build Initramfs

HINT: In Debian t/e `initramfs.img` is named `initrd.img` too.

5.1 Make FHS available

```
# cd workplace/initrd/
# mkdir  boot bin dev proc sbin tmp boot etc
# mkdir  lib root run srv usr home mnt sys var
```

5.2 Configure initramfs files

You can refer to the Debian package `base-files`.

5.2.1 etc/fstab

fstab stores static information about the filesystem, so let's vim etc/fstab.

```
proc  /proc proc rw,nosuid,nodev,noexec,relatime 0 0
sysfs /sys sysfs rw,nosuid,nodev,noexec,relatime 0 0
tmpfs /run tmpfs rw,nosuid,relatime,mode=755     0 0
```

Above are important items. If you would like to invoke

```
# mount -a
```

in any script (like rcS or initramfs init) or manually, you should have this file.

5.2.2 /dev/*

```
# cd wordplace/initrd
(# mknod -m 640 dev/initrd  b 1 250)
# mknod -m 600 dev/console c 5 1
# mknod -m 666 dev/null    c 1 3
```

5.2.3 etc/hostname

Anything you like, such as `debian`.

5.2.4 etc/hosts

This is for basic network function.

```
127.0.0.1  localhost debian
::1        localhost ip6-localhost ip6-loopback debian
ff02::1    ip6-allnodes
ff02::2    ip6-allrouters
```

5.2.5 /etc/inputrc

Add this file to enable convenient keys.

```
# /etc/inputrc
set input-meta on
set output-meta on
set bell-style none
$if mode=emacs
"\e[1~": beginning-of-line
"\e[4~": end-of-line
"\e[3~": delete-char
"\e[2~": quoted-insert
"\e[1;5C": forward-word
"\e[1;5D": backward-word
"\e[5C": forward-word
"\e[5D": backward-word
"\e\e[C": forward-word
"\e\e[D": backward-word
$if term=rxvt
"\e[8~": end-of-line
"\eOc": forward-word
"\eOd": backward-word
$endif
$endif
```

5.2.6 /etc/{passwd,shadow}

this is passwd

`root:x:0:0:root:/root:/bin/bash`

and this is shadow

```
root:::0:99999:7:::
```

They are to enable root login and set root password as null.

5.3 Initramfs init

Warning : If there is no `/init` in initrd.img, kernel would regard the initrd.img as malformed/illegal one and then **panic**.

- If you don't want to create an `init` script, you can just link the init to busybox as following.
- If you want to use a true init script, following is a very simple one that works.
- Or even, you can write your own C init program.

Linking busybox init:

```
# cd workplace/initrd/
# ln -s linuxrc init
```

Creating a simple initramfs init script:

```
#!/bin/sh
printf "\x1b[1;32m *\x1b[0;m [initramfs] Loading, please wait..."
export PATH=/sbin:/usr/sbin:/bin:/usr/bin
[ -d /dev ]  || mkdir -m 0755 /dev
[ -d /root ] || mkdir -m 0700 /root
[ -d /sys ]  || mkdir /sys
[ -d /proc ] || mkdir /proc
[ -d /tmp ]  || mkdir /tmp
mkdir -p /var/lock
#mount -a
mount -t sysfs -o nodev,noexec,nosuid sysfs /sys
mount -t proc -o nodev,noexec,nosuid proc /proc
/sbin/mdev -s
#clear
printf "\x1b[1;32m *\x1b[0;32m Welcome to MiniSys on Initramfs !\x1b[m\n"
exec /sbin/init
```

With this initramfs init, you can only stay in initramfs after boot.
Creating initramfs init C program:
`bsdbar.h :`

```
LyogYnNkYmFyLmgKCiAgIHBhcnQgb2YgQnl0ZWZyZXEKICAgY2RsdW1pbmF0ZUAxNjMuY29tCiov
CiNpZm5kZWYgQlNEQkFSX0gKI2RlZmluZSBCU0RCQVJfSAoKI2luY2x1ZGUgPHVuaXN0ZC5oPgoj
aW5jbHVkZSA8c3RkaW8uaD4KCi8qIElOVEVSRkFDRSAqLwp2b2lkIEJTRGJhcl9pbml0ICh2b2lk
KTsKdm9pZCBCU0RiYXJfY2xlYXIgKHZvaWQpOwp2b2lkIEJTRGJhcl9yZWZyZXNoICh2b2lkKTsK
LyogRU5EIElOVEVSRkFDRSAqLwoKc3RhdGljIHN0cnVjdCBfYnNkYmFyIHsKCWNoYXIgYmFyOwoJ
c3RydWN0IF9ic2RiYXIgKiBuZXh0Owp9IGJhcjEsIGJhcjIsIGJhcjM7CgpzdGF0aWMgc3RydWN0
IF9ic2RiYXIgKiBfYmFyX2N1cnNvcjsKCnZvaWQKQlNEYmFyX2luaXQgKHZvaWQpCnsKICAgIC8q
IHdyaXRlIGEgcGFkZGluZyBmb3IgdGhlIGJhciAqLwoJd3JpdGUgKDIsICIgICAgIiwgNSk7Cgkv
KiBidWlsZCBhIGNoYWluIGN5Y2xlICovCgliYXIxLmJhciA9ICctJzsKCWJhcjIuYmFyID0gJ1xc
JzsKCWJhcjMuYmFyID0gJy8nOwoJYmFyMS5uZXh0ID0gJmJhcjI7CgliYXIyLm5leHQgPSAmYmFy
MzsKCWJhcjMubmV4dCA9ICZiYXIxOwoJLyogcG9pbnQgdGhlIGN1cnNvciB0byBiYXIxICovCglf
YmFyX2N1cnNvciA9ICZiYXIxOwoKCXJldHVybjsKfQoKLyogdGhpcyBmdW5jdGlvbiBpcyBmb3Ig
```

```
aW50ZXJuYWwgdXNlICovCnZvaWQKX0JTRGJhcl9yZWZyZXNoIChjaGFyIF9iYXIpCnsKCS8qIHJl
ZnJlc2ggQlNELXN0eWxlIHByb2dyZXNzIGJhciAqLwogICAgLyogd2hvbGUgYnVmZmVyIG9mIHRo
ZSBiYXIgKi8KCXN0YXRpYyBjaGFyIGJiWzRdID0gewogICAgICAgICCgJywgJyAnLCAnICcsICCg
JwogICAgfTsKCXdyaXRlICgyLCAiXGJcYlxiXGIiLCA1KTsgLyogY2xlYXIgdGhlIHByZXZpb3Vz
IGJhciAqLwoJc25wcmludGYgKGJiLCA0LCAiICVjICAiLCBfYmFyKTsgLyogcHJlcGFyZSBidWZm
ZXIgKi8KCWZmbHVzaCAoTlVMTCk7IC8qIHN5bmMgc3RkaW8gYnVmZmVyIHRvIHVzZXItZGVmaW5l
ZCBidWZmZXIgKi8KCXdyaXRlICgyLCBiYiwgNCk7IC8qIHByaW50IHRoZSBidWZmZXIgdG8gc3Rk
ZXJyICovCglyZXR1cm47Cn0KCnZvaWQKQlNEYmFyX3JlZnJlc2ggKHZvaWQpCnsKICAgIC8qIG5v
dGUgdGhhdCAnaW50IG51bScgaXMgdGhlIHByb3BvcnRpb24gdG8gZGlzcGxheSAqLwogICAgX0JT
RGJhcl9yZWZyZXNoIChfYmFyX2N1cnNvciAtPiBiYXIpOwoJX2Jhcl9jdXJzb3IgPSBfYmFyX2N1
cnNvciAtPiBuZXh0OwogICAgcmV0dXJuOwp9Cgp2b2lkCkJTRGJhcl9jbGVhciAodm9pZCkKewog
ICAgLyogY2xlYXIgdGhlIHBhZGRpbmcvYmFyIGFuZCBuZXdsaW5lKi8KCXdyaXRlICgyLCAiXGJc
YlxiXGJcbiIsIDYpOwoJcmV0dXJuOwp9CgojZW5kaWYgLyogQlNEQkFSX0ggKi8K
```

init.c :

```
LyogaW5pdC5jIC0gY2RsdW1pbmF0ZUAxNjMuY29tICovCiNpbmNsdWRlIDx1bmlzdGQuaD4KI2lu
Y2x1ZGUgPHN0ZGxpYi5oPgojaW5jbHVkZSA8c3RkaW8uaD4KI2luY2x1ZGUgPHN0cmluZy5oPgoK
I2luY2x1ZGUgImJzZGJhci5oIgoKI2RlZmluZSBzdGFyICAgICAiXHgxYlsxOzMybSAqIFx4MWJb
MDttIgojZGVmaW5lIGNpbmZvICAgICJceDFiWzM2bSIKI2RlZmluZSBjbm9ybWFsICAiXHgxYlsz
Mm0iCiNkZWZpbmUgY3dhcm4gICAgIlx4MWJbMzFtIgojZGVmaW5lIGNlbmQgICAgICJceDFiWzA7
bVxuIgoKI2RlZmluZSBJTkZPIDEKI2RlZmluZSBOT1JNQUwgMgojZGVmaW5lIFdBUk4gMwoKdm9p
ZApzZHVtcCAoaW50IGxldmVsLCBjaGFyICogc3RyaW5nKQp7CiAgICB3cml0ZSAoMSwgc3Rhciwg
c2l6ZW9mKHN0YXIpKTsKICAgIHN3aXRjaCAobGV2ZWwpIHsKICAgIGNhc2UgSU5GTzoKICAgICAg
ICB3cml0ZSAoMSwgY2luZm8sIHNpemVvZihjaW5mbykpOwogICAgICAgIGJyZWFrOwogICAgY2Fz
ZSBOT1JNQUw6CiAgICAgICAgd3JpdGUgKDEsIGNub3JtYWwsIHNpemVvZihjbm9ybWFsKSk7CiAg
ICAgICAgYnJlYWs7CiAgICBjYXNlIFdBUk46CiAgICAgICAgd3JpdGUgKDEsIGN3YXJuLCBzaXpl
b2YoY3dhcm4pKTsKICAgICAgICBicmVhazsKICAgIGRlZmF1bHQ6CiAgICAgICAgOwogICAgfQog
ICAgd3JpdGUgKDEsIHN0cmluZywgc3RybGVuKHN0cmluZykpOwogICAgd3JpdGUgKDEsIGNlbmQs
IHNpemVvZihjZW5kKSk7CiAgICByZXR1cm47Cn0KCnZvaWQKbWFpbiAoaW50IGFyZ2MsIGNoYXIg
Kiphcmd2LCBjaGFyICoqZW52KQp7CiAgICBsb25nIGNvdW50ZXIgPSAwOwogICAgc2R1bXAgKFdB
Uk4sICJXZWxjb21lIHRvIGluZmluaXRlIGluaXQgISIpOwogICAgc2xlZXAgKDEpOwogICAgc2R1
bXAgKE5PUk1BTCwgIkxvb3Agc3RhcnQgLi4uIik7CiAgICBCU0RiYXJfaW5pdCAoKTsKICAgIHdo
aWxlICgxKSB7CiAgICAgICAgY291bnRlcisrOwogICAgICAgIHVzbGVlcCAoODApOwogICAgICAg
IEJTRGJhcl9yZWZyZXNoICgpOwogICAgICAgIGlmIChjb3VudGVyID4gMTAwMDAgfHwgY291bnRl
ciA8IDApIHsKICAgICAgICAgICAgQlNEYmFyX2NsZWFyICgpOwogICAgICAgICAgICBzZHVtcCAo
SU5GTywgInBhc3NlZCAxMDAwMCBjeWNsZS4iKTsKICAgICAgICAgICAgY291bnRlciA9IDA7CiAg
ICAgICAgfSAKICAgIH0KICAgIEJTRGJhcl9jbGVhciAoKTsKICAgIHJldHVybjsKfQo=
```

To compile it, just type

```
$ gcc -O2 -o init init.c
```

5.4 Init Script

Use **either** inittab or rcS for busybox init here, and don't use both them.
Lookup busybox init for reason[2].

5.4.1 etc/init.d/rcS

Example for busybox init :

```
#!/bin/sh
printf "\x1b[1;32m*\x1b[0;m [init] Loading, please wait..."
export PATH=/sbin:/usr/sbin:/bin:/usr/bin
mount -a
#clear
printf "\x1b[32m* Welcome to MiniSys on Initramfs !\x1b[m\n"
/bin/sh
```

5.4.2 /etc/inittab

```
tty1::respawn:/sbin/getty 38400 tty1
tty2::respawn:/sbin/getty 38400 tty2
tty3::respawn:/sbin/getty 38400 tty3
tty4::respawn:/sbin/getty 38400 tty4
tty5::respawn:/sbin/getty 38400 tty5
tty6::respawn:/sbin/getty 38400 tty6
```

5.5 Wrap Initrd

```
# cd initrd/
# find . | cpio -o -H newc > ../initrd.img
# gzip -k ../initrd.img
```

you can also gzip the image to initrd.img.gz, kernel recogonizes it too.

6 Install the System into a USB stick

6.1 Partition USB stick

Assume I have an 8GB USB Flash stick, detected as /dev/sdc.

```
# parted /dev/sdc
  > mktable  (gpt)
  > mkpart (2MB-4MB as BIOS_GRUB)
  > set 1 bios_grub
  > mkpart (4MB-REST as /)
  > quit
# partprobe
# lsblk || cat /proc/partitions
```

6.2 Make Filesystem

```
# mkfs.ext4 /dev/sdc2 || mke2fs -t ext4 /dev/sdc2
# mount -t ext4 /dev/sdc2 /mnt
```

6.3 Copy Kernel and Initrd

```
# cp bzImage /mnt/boot
# cp initrd.img /mnt/boot
```

6.4 Install Grub on USB Stick

```
# grub-install --boot-directory /mnt/boot /dev/sdc
```

7 Boot Test

7.1 Boot via QEMU, without USB

Test bzImage + initrd.img.

```
# qemu-system-x86_64 -enable-kvm -m 512 -kernel bzImage -initrd initrd.img
```

7.2 Boot via QEMU, with USB

Test Grub2 + bzImage + initrd.img.

```
# qemu-system-x86_64 -enable-kvm -m 512 -hda /dev/sdc
```

7.2.1 Talk with Grub2

```
grub> ls
grub> insmod linux
grub> prefix=(hd0,gpt2)/root/grub
grub> root=(hd0,gpt2)
grub> linux /boot/bzImage [OPTIONS]
grub> initrd /boot/initrd.img
grub> boot
```

Where OPTIONS depends on your preference.

7.2.2 Put Grub2 config into boot/grub/grub.cfg

```
# grub.cfg
insmod part_gpt
insmod ext2
set root=(hd0,gpt2)

echo "* [grub] Loading linux ...\n"
linux /boot/bzImage root=/dev/ram0 init=/sbin/init
echo "* [grub] Loading initrd.img ...\n"
initrd /boot/initrd.img
echo "* [grub] Booting ...\n"
boot
```

Then the system would autostart as grub2 found grub.cfg.

8 Extend the Mini System

8.1 Script /init in initrd.img

Imitating Debian's script from update-initramfs and the script from Linux from scratch[6].
Note that, this script defined a new function "choose if you want to switch root", add corresponding kernel parameter then you can activate this function:

- `switch` is default, means that if a root filesystem is detected, then init would switch root into it.
- `noswitch` meas that, don't switch root even if an available root is detected.

```
#!/bin/sh
# initrd.img /init # C.D.Luminate <cdluminate@gmail.com>
printf "* [initrd] Loading, please wait...\n"
export PATH=/sbin:/usr/sbin:/bin:/usr/bin

# Check FHS
[ -d /dev  ] || mkdir -m 0755 /dev
[ -d /root ] || mkdir -m 0700 /root
[ -d /sys  ] || mkdir /sys
[ -d /proc ] || mkdir /proc
[ -d /tmp  ] || mkdir /tmp
[ -d /run  ] || mkdir /run
mkdir -p /var/lock
mount -n -t sysfs -o nodev,noexec,nosuid sysfs /sys
mount -n -t proc -o nodev,noexec,nosuid proc /proc
mount -n -t devtmpfs devtmpfs /dev
mount -n -t tmpfs tmpfs /run
/sbin/mdev -s

# For switch_root
mkdir /.root
mknod /dev/initrd b 1 250

# parameters
init=/sbin/init
#init=/usr/lib/systemd/systemd
root=
rootdelay=
rootfstype=auto
ro="ro"
rootflags=
device=
switch="true"

printf "* [initrd] Parse cmdline...\n"
read -r cmdline < /proc/cmdline
for param in $cmdline ; do
    case $param in
    init=*)         init=${param#init=}             ;;
    root=*)         root=${param#root=}             ;;
    rootfstype=*)   rootfstype=${param#rootfstype=} ;;
    rootflags=*)    rootflags=${param#rootflags=}   ;;
    ro)             ro="ro"                         ;;
    rw)             ro="rw"                         ;;
    switch)         switch="true"                   ;;
    noswitch)       switch="false"                  ;;
    esac
done

case "$root" in
    /dev/* ) device=$root ;;
    UUID=* ) eval $root; device="/dev/disk/by-uuid/$UUID"  ;;
    LABEL=*) eval $root; device="/dev/disk/by-label/$LABEL" ;;
    ""     ) echo "* [initrd] FATAL: No root device found.";
             switch="false" ;;
esac

printf "\x1b[32m* [initrd] Mount root device...\x1b[m\n"
if [ ! -z $root ]; then {
  if ! mount -n -t "$rootfstype" -o "$rootflags" "$device" /.root ; then
    printf "\x1b[31m* [initrd] Mount device $root : Failure\x1b[m\n"
    printf "\x1b[33m\r\nAvailable Devices:\n";
    cat /proc/partitions; printf "\x1b[m"; sleep 10;
  else
    printf "\x1b[32m* [initrd] Mount device $root : Success\x1b[m\n"
  fi
} else {
  printf "\x1b[32m* [initrd] No mounting root device \x1b[m\n"
} fi

case "$switch" in
"true")
    printf "\x1b[33m* Switching root ...\x1b[m\n";
    sleep 1;
    exec switch_root /.root "$init" "$@" ;;
*)
    printf "\x1b[33m* No Switch root ...\x1b[m\n";
    sleep 1;
    exec /bin/busybox init;;
esac
# EOF init Script
```

8.2 Prepare Stage3 rootfs

8.2.1 Make it myself

"debootstrap" for Debian or Ubuntu. For example,

```
# debootstrap ubstable ./unstable-chroot http://ftp.us.debian.org/debian
```

8.2.2 Use an already cooked one

Download the Stage3 tarball of Archlinux[3] or Gentoo[4].
Here we can use both of them. And this is hint:

- **Archlinux** stage3 does not symlink /usr/lib/systemd/systemd to /sbin/init, so you may encounter a kernel panic if you don't modify my initramfs init script. to avoid this just set the init parameter.
- **Gentoo** stage3 tarballs works well.
- **Debian** stage3 tarballs I made also works well.

[3] https://www.archlinux.org/
[4] http://www.gentoo.org/

8.2.3 Make the disk image OR copy them into USB

You can extrace the Stage3 tarball into a disk image:

```
# dd of=disk.img bs=1 seek=4G count=0
# mkfs.ext4 disk.img
# mount disk.img /mnt
# tar zxvf Stage3.tar.gz -C /mnt
# do some configurations
# umount /mnt
```

or just extract it into you USB stick.

8.3 QEMU: Boot with the new disk

```
# qemu-system-x86_64 -enable-kvm -m 512 -kernel bzImage -initrd initrd.img
    -hda disk.img -append "root=/dev/sda init=/usr/lib/systemd/systemd"
```

8.4 QEMU: Boot from the USB

```
# qemu-system-x86_64 -enable-kvm -m 512
    -hda disk.img -append "root=/dev/sda init=/usr/lib/systemd/systemd"
```

9 Compile Static Bash

9.1 Download bash source

bash-4.3.tar.gz from GNU.

9.2 Compile static Bash

```
$ ./configure --enable-static-link --without-bash-malloc
```

Then you will see bash as following:

```
$ ldd bash
bash: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux),
      statically linked, for GNU/Linux 2.6.32,
      BuildID[sha1]=ab5bcc419a27e6c54d0fb352c28019446e68dd46, not stripped
```

9.3 Make bash.tar.gz tarball

I suggest copy those files into directory bash.pkg:

- staticly linked bash excutable
- examples/startup-files/Bash_profile
- examples/startup-files/bashrc

then make dir bash.pkg as bash.tar.gz

9.4 Install static bash into initrd

Just extract the tarball into initrd/.

10 Lazy Glibc supporting Sed

If, say, I want to use the program GNU Sed in the freshly cooked system, but we lack the glibc that supporting sed. so we can use Debian's precompiled glibc, and a sed that customed by us (deleting .so links that we don't like.).

10.1 Compiling GNU Sed

```
$ ./configure --without-selinux --disable-acl
$ make -j4
$ ldd sed/sed
  linux-vdso.so.1 (0x00007ffcec738000)
  libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f870f03b000)
  /lib64/ld-linux-x86-64.so.2 (0x00007f870f410000)
$ mkdir sed.pkg/
```

See, it's ldd output is very clean. Now let's add the libc.
Then you should copy files you need into dir sed.pkg/

10.2 Wrap GLibc

Now let's grab a libc to support the sed (and other programs depending on this libc).
For convenience, I just downloaded the Debian's precompiled glibc, and re-packed the glibc together with sed.

```
$ apt-get download libc6 libc-bin
$ for DEB in $(ls *.deb); do dpkg -X ${DEB} sed.pkg/; done
$ cd sed.pkg
$ tar zcvf ../sed.tar.gz .
```

10.3 Install libc + sed into initrd

Just extract the tarball into initrd/

A Base64 Code of /init

Following is init.base64 :

```
IyEvYmluL3NoCiMgaW5pdHJkLmltZyAvaW5pdCAjIEMuRC5MdW1pbmF0ZSA8Y2RsdW1pbmF0ZUBn
bWFpbC5jb20+CnByaW50ZiAiKiBbaW5pdHJkXSBMb2FkaW5nLCBwbGVhc2Ugd2FpdC4uLlxuIgpl
eHBvcnQgUEFUSD0vc2Jpbjovd XNyL3NiaW46L2Jpbjovd XNyL2JpbgoKIyBDaGVjayBGSFMKWyAt
ZCAvZGV2ICBdIHx8IG1rZGlyIC1tIDA3NTUgL2RldgpbIC1kIC9yb290IF0gfHwgbWtkaXIgLW0g
MDcwMCAvcm9vdApbIC1kIC9zeXMgIF0gfHwgbWtkaXIgL3N5cwpbIC1kIC9wcm9jIF0gfHwgbWtk
aXIgL3Byb2MKWyAtZCAvdG1wICBdIHx8IG1rZGlyIC90bXAKWyAtZCAvcnVuICBdIHx8IG1rZGly
IC9ydW4KbWtkaXIgLXAgL3Zhci9sb2NrCm1vdW50IC1uIC10IHN5c2ZzIC1vIG5vZGV2LG5vZXhl
Yyxub3N1aWQgc3lzZnMgL3N5cwptb3VudCAtbiAtdCBwcm9jIC1vIG5vZGV2LG5vZXhlYyxub3N1
aWQgcHJvYyAvcHJvYwptb3VudCAtbiAtdCBkZXZ0bXBmcyBkZXZ0bXBmcyAvZGV2Cm1vdW50IC1u
IC10IHRtcGZzIHRtcGZzIC9ydW4KL3NiaW4vbWRldiAtcwoKIyBGb3Igc3dpdGNoX3Jvb3QKbWtk
aXIgLy5yb290Cm1rbm9kIC9kZXYvaW5pdHJkIGIgMSAyNTAKCiMgcGFyYW1ldGVycwppbml0PS9z
YmluL2luaXQKI2luaXQ9L3Vzci9saWIvc3lzdGVtZC9zeXN0ZW1kCnJvb3Q9CnJvb3RkZWxheT0K
cm9vdGZzdHlwZT1hdXRvCnJvPSJybyIKcm9vdGZsYWdzPQpkZXZpY2U9CnN3aXRjaD0idHJ1ZSIK
CnByaW50ZiAiKiBbaW5pdHJkXSBQYXJzZSBjbWRsaW5lLi4uXG4iCnJlYWQgLXIgY21kbGluZSA8
IC9wcm9jL2NtZGxpbmUKZm9yIHBhcmFtIGluICRjbWRsaW5lIDsgZG8KICAgIGNhc2UgJHBhcmFt
IGluCiAgICBpbml0PSopICAgICAgICAgaW5pdD0ke3BhcmFtI2luaXQ9fSAgICAgICAgICAgICA7
OwogICAgcm9vdD0qKSAgICAgICAgIHJvb3Q9JHtwYXJhbSNyb290PX0gICAgICAgICAgICAgOzsK
ICAgIHJvb3Rmc3R5cGU9KikgICByb290ZnN0eXBlPSR7cGFyYW0jcm9vdGZzdHlwZT19IDs7CiAg
ICByb290ZmxhZ3M9KikgICAgcm9vdGZsYWdzPSR7cGFyYW0jcm9vdGZsYWdzPX0gICA7OwogICAg
cm8pICAgICAgICAgICAgIHJvPSJybyIgICAgICAgICAgICAgICAgICAgICAgICAgOzsKICAgIHJ3
KSAgICAgICAgICAgICBybz0icnciICAgICAgICAgICAgICAgICAgICAgICAgIDs7CiAgICBzd2l0
Y2gpICAgICAgICAgc3dpdGNoPSJ0cnVlIiAgICAgICAgICAgICAgICAgICA7OwogICAgbm9zd2l0
Y2gpICAgICAgIHN3aXRjaD0iZmFsc2UiICAgICAgICAgICAgICAgICAgOzsKICAgIGVzYWMKZG9u
ZQoKY2FzZSAiJHJvb3QiIGluCiAgICAvZGV2LyogKSBkZXZpY2U9JHJvb3QgOzsKICAgIFVVSUQ9
KiApIGV2YWwgJHJvb3Q7IGRldmljZT0iL2Rldi9kaXNrL2J5LXV1aWQvJFVVSUQiICA7OwogICAg
TEFCRUw9KikgZXZhbCAkcm9vdDsgZGV2aWNlPSIvZGV2L2Rpc2svYnktbGFiZWwvJExBQkVMIiA7
OwogICAgIiIgICAgICkgZWNobyAiKiBbaW5pdHJkXSBGQVRBTDogTm8gcm9vdCBkZXZpY2UgZm91
bmQuIjsKICAgICAgICAgICAgIHN3aXRjaD0iZmFsc2UiIDs7CmVzYWMKCnByaW50ZiAiXHgxYlsz
Mm0qIFtpbml0cmRdIE1vdW50IHJvb3QgZGV2aWNlLi4uXHgxYlttXG4iCmlmIFsgISAteiAkcm9v
dCBdOyB0aGVuIHsKICBpZiAhIG1vdW50IC1uIC10ICIkcm9vdGZzdHlwZSIgLW8gIiRyb290Zmxh
Z3MiICIkZGV2aWNlIiAvLnJvb3QgOyB0aGVuCiAgICBwcmludGYgIlx4MWJbMzFtKiBbaW5pdHJk
XSBNb3VudCBkZXZpY2UgJHJvb3QgOiBGYWlsdXJlXHgxYlttXG4iCiAgICBwcmludGYgIlx4MWJb
MzNtXHJcbkF2YWlsYWJsZSBEZXZpY2VzOlxuIjsKICAgIGNhdCAvcHJvYy9wYXJ0aXRpb25zOyBw
cmludGYgIlx4MWJbbSI7IHNsZWVwIDEwOwogIGVsc2UKICAgIHByaW50ZiAiXHgxYlszMm0qIFtp
bml0cmRdIE1vdW50IGRldmljZSAkcm9vdCA6IFN1Y2Nlc3NceDFiW21cbiIKICBmaQp9IGVsc2Ug
ewogIHByaW50ZiAiXHgxYlszMm0qIFtpbml0cmRdIE5vIG1vdW50aW5nIHJvb3QgZGV2aWNlIFx4
MWJbbVxuIgp9IGZpCgpjYXNlICIkc3dpdGNoIiBpbgoidHJ1ZSIpCiAgICBwcmludGYgIlx4MWJb
MzNtKiBTd2l0Y2hpbmcgcm9vdCAuLi5ceDFiW21cbiI7CiAgICBzbGVlcCAxOwogICAgZXhlYyBz
d2l0Y2hfcm9vdCAvLnJvb3QgIiRpbml0IiAiJEAiIDs7CiopCiAgICBwcmludGYgIlx4MWJbMzNt
KiBObyBTd2l0Y2ggcm9vdCAuLi5ceDFiW21cbiI7CiAgICBzbGVlcCAxOwogICAgZXhlYyAvYmlu
L2J1c3lib3ggaW5pdDs7CmVzYWMKIyBFT0YgaW5pdCBTY3JpcHQK
```

To decode it, type :

```
$ base64 -d init.base64
```

B Misc

References

[1] The Linux Kernel Archive
https://kernel.org

[2] Busybox
http://www.busybox.net

[3] Linux kernel Doc
https://kernel.org/doc
You can also lookup the Documentation dir in kernel source tree.

[4] Linux kernel boot protocol
linux-4.0/Documentation/./x86/boot.txt

[5] GNU : grub document
http://www.gnu.org/software/grub/manual/grub.html

[6] LFS : Linux From Scratch
http://www.linuxfromscratch.org/lfs/

[7] BLFS : Beyond Linux From Scratch
http://www.linuxfromscratch.org/blfs/

[8] A Guide on initramfs from LFS
http://www.linuxfromscratch.org/hints/downloads/files/initramfs.txt

[9] A Guide on initrd from LFS
http://www.linuxfromscratch.org/hints/downloads/files/initrd.txt

[10] Download GNU Bash
http://ftp.gnu.org/gnu/bash/

[11] The kernel module dir
/lib/modules/$(uname -r)/

[12] An Article about Grub
http://blog.csdn.net/guanggY/article/details/6210774

[13] An Article about Initramfs
http://blog.csdn.net/lvqqrainbow/article/details/6536422

[14] An Article about Kernel Boot
http://blog.csdn.net/zhoudaxia/article/details/6666683

[15] An Article about Kernel Boot
http://blog.sina.com.cn/s/blog_b02f77c80101db1t.html

C LISENCE