

Popcorn Project: Starting with Popcorn OS

Antonio Barbalace

antoniob@vt.edu

Systems Software Research Group at Virginia Tech

<http://ssrg.ece.vt.edu>

Patches 14th Dec 2012

- Popcorn Hacking Guide
 - Generic modifications (all archs)
 - linux-3.2.14-popcorn-build.patch
 - linux-3.2.14-popcorn-syscall.patch
 - linux-3.2.14-popcorn-generic.patch
 - Architecture dependent in `arch/x86`
 - linux-3.2.14-popcorn-x86-build.patch
 - linux-3.2.14-popcorn-x86-syscall.patch
 - linux-3.2.14-popcorn-x86-generic.patch
 - linux-3.2.14-popcorn-x86-apic.patch
 - **linux-3.2.14-popcorn-x86-boot.patch**
 - linux-3.2.14-popcorn-x86-vty.patch
 - Drivers modifications (all archs) in `drivers/`
 - linux-3.2.14-popcorn-drivers-vty.patch
 - linux-3.2.14-popcorn-drivers-acpi.patch
 - linux-3.2.14-popcorn-drivers-gpu.patch
 - linux-3.2.14-popcorn-drivers-pci.patch
- Goal
 - Release a first usable version of the project (alternative to virtual machines)
 - Document the project
 - Attract contributors and enthusiasts
 - Separate the architecture dependent code (initial porting guide)

Patches 25th Mar 2013

- Goal
 - Release a new usable version of the project (replicated-kernel)
 - Add new components
 - Inter-Kernel Messaging layer
 - Remote process creation and migration
 - Other improvements and fixes

GIT Repositories

- Hosted on TO BE ANNOUNCED
 - Publicly browseable, direct r/w access is protected (ssh key required)
- Kernel code
 - TO BE ANNOUNCED
 - many different branches
 - davek – process/thread remote creation, migration
 - net_msg_integration – fast software network switch
 - shmem_tuntap – software network switch using TUN/TAP
 - bshelton_messaging – inter-kernel messaging layer
 - andy_fd_aware – NUMA aware scheduling
 - andy_load_balance – again, NUMA aware scheduling
 - mklinux-readonly – one page table per NUMA-node
 - etc.
- Utils package
 - TO BE ANNOUNCED
- Kexec repository
 - TO BE ANNOUNCED

Before Booting Popcorn OS

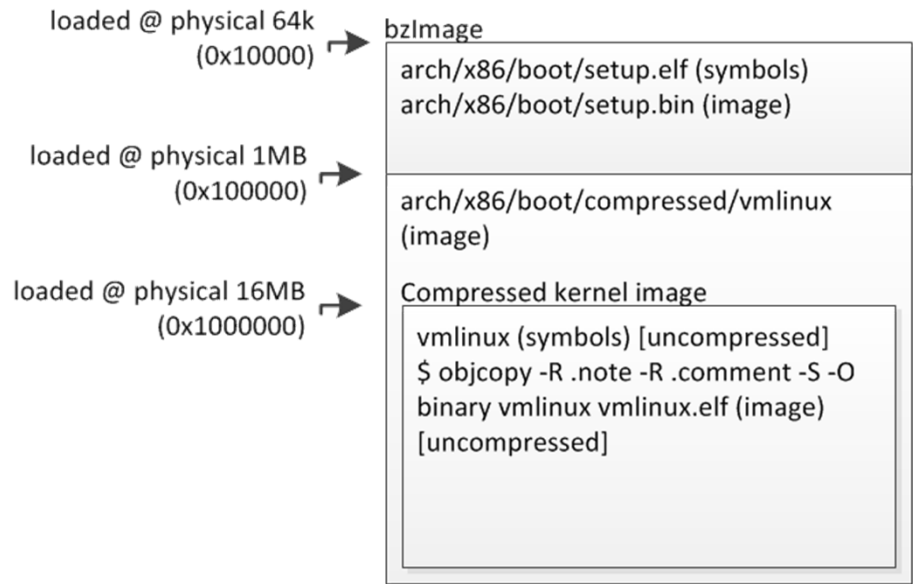
1. Start your machine with any Linux kernel with NUMA enabled
2. Download, compile and install a **Popcorn kernel**
3. Download, compile and install **Popcorn's kexec** utility
4. Download, compile and install the **Popcorn utils** package
5. Create a **ramdisk** image for the secondary kernels
6. Create the **resource** partition or cluster **configurations**
7. Copy and paste the primary kernel configuration to the boot loader (or take note of it)
8. Reboot!

Booting Popcorn OS

1. At the boot loader **select** the Popcorn kernel
2. Add the generated (slide 5 step 6) kernel **command line parameters** (if not added before)
3. When the Linux kernel is up and running, **login as root**
4. Use one of the scripts (in Popcorn utils) to **load other kernel instances** (secondary kernels)
5. The kernels will automatically and transparently form a **single OS** (this functionality can be disabled)

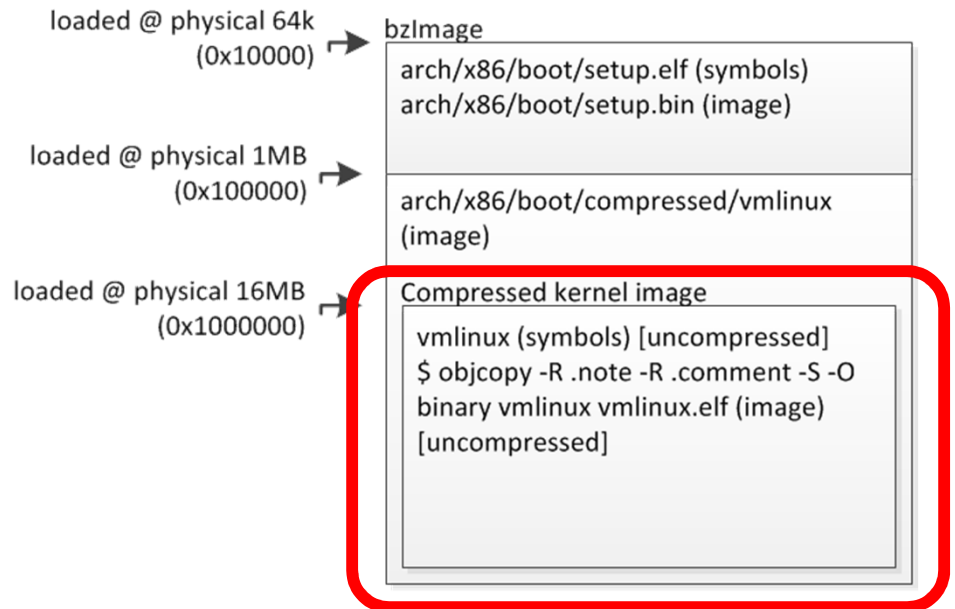
SMP Linux Boot Image (bzImage)

- x86_64 uses **bzImage** (default)
- the boot format is discussed in detail in `Documentation/x86/boot.txt`
- **bzImage** is made up of
 - a compressed and stripped version of **vmlinux**
 - that is accompanied by real-mode code for relocation and decompression



Popcorn Boot Images

- The primary Popcorn kernel boots from a conventional bzImage
- Secondary Popcorn kernels boot up from **vmlinux.elf**
 - decompression is not required, i.e. faster startup

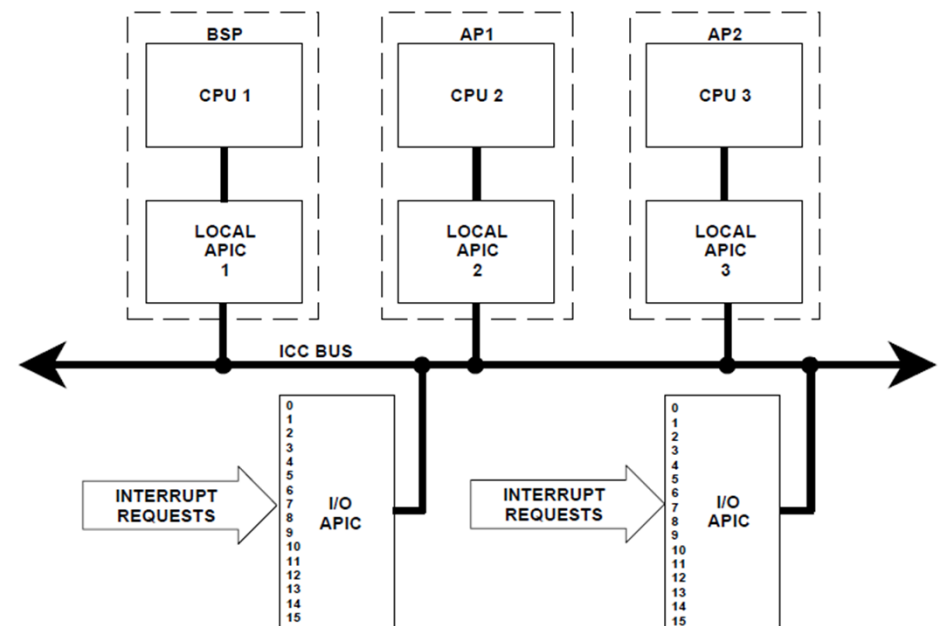


Secondary Kernels – Strategy Overview

- Use *kexec* to put the kernel image in the correct place in the physical memory (and start the boot process)
- Modify the Linux SMP boot trampoline to launch secondary kernels
- Adapt the existing Linux infrastructure to provide basic OS services in a replicated-kernel environment

SMP Linux Boot Process (x86)

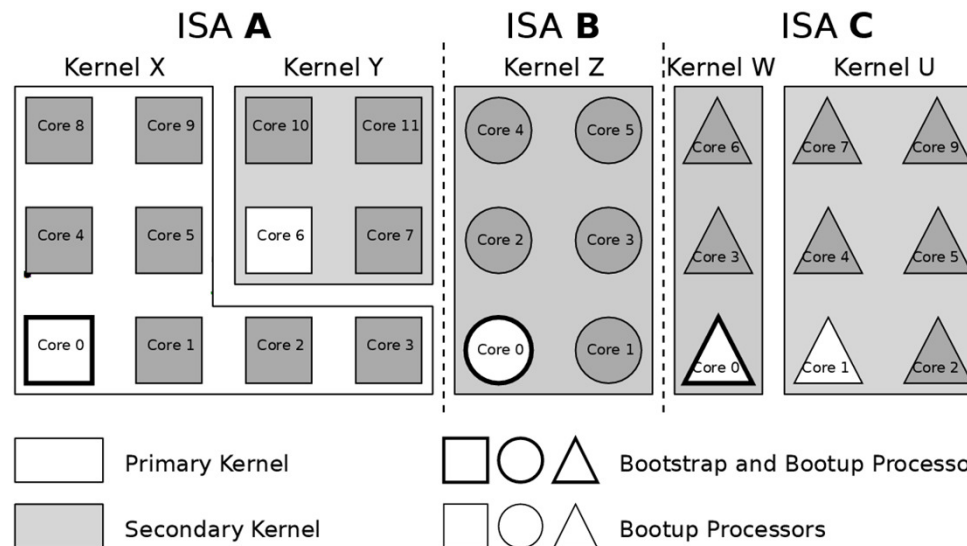
- In a multiprocessor (multi-core) x86 box there is
 - a Bootstrap Processor (**BSP**)
 - all other processors are Application Processors (**AP**)
- The BSP is the one that executes the BIOS code
- The BSP has to bring up all the APs



From Intel "MultProcessor Specification", 1997

Primary and Secondary Kernels

- The Primary kernel is the kernel that boots on the bootstrap processor (i.e. the first kernel to boot)
- Any other kernel that boots is called a Secondary Kernel
- We redefine the names in order to extend the BSP/AP processors nomenclature from the Intel specifications to kernels in a heterogeneous ISA setting



Fixes in arch/x86/kernel/head_64.S

file:///root/mklinux-patch/mklinux.git-14dec2012/linux-3.2.14-popcorn-x86-boot.patch - Kompare

File Difference Settings Help

Compare Files Save Save All Previous File Next File Previous Difference Next Difference Unapply All Unapply Difference Apply Difference Apply All

Navigation

Source Folder

- /dev/
- a/
- arch/
- x86/
- kernel/
- cpu/
- include/
- asm/

Destination Folder

- Unknown
- b/
- arch/
- x86/
- kernel/
- cpu/
- include/
- asm/

Source File

- vmlinux.l...
- trampolin...
- smpboot.c
- setup.c
- null
- head_64.S
- head64.c
- e820.c
- aperture...
- Makefile

Destination File

- vmlinux.lds.S
- trampoline.c
- smpboot.c
- setup.c
- trampoline_64_bsp.S
- head_64.S
- head64.c
- e820.c
- aperture_64.c
- Makefile

Source Line	Destination Line	Difference
240	257	Inserted 1 line
115	126	Changed 1 line
113	115	Inserted 9 lines
106	106	Inserted 2 lines

head_64.S

```
startup_64:
103  andq  $(PTRS_PER_PUD - 1), %rax
104  jz   ident_complete
105
106  leaq  (level2_spare_pgt - __START_KERNEL_map + _KERNPG_TABLE)(%rbp), %rdx
107  leaq  level3_ident_pgt(%rip), %rbx
108  movq  %rdx, 0(%rbx, %rax, 8)
109
110  movq  %rdi, %rax
111  shrq  $PMD_SHIFT, %rax
112  andq  $(PTRS_PER_PMD - 1), %rax
113  leaq  __PAGE_KERNEL_IDENT_LARGE_EXEC(%rdi), %rdx
114  leaq  level2_spare_pgt(%rip), %rbx
115  movq  %rdx, 0(%rbx, %rax, 8)
116 ident_complete:
117
118  /*
119  ENTRY (secondary_startup_64)
237  movl  initial_gs+4(%rip), %edx
238  wrmsr
239
240  /* esi is pointer to real mode structure with interesting info.
241  pass it to C */
242  movl  %esi, %edi
```

head_64.S

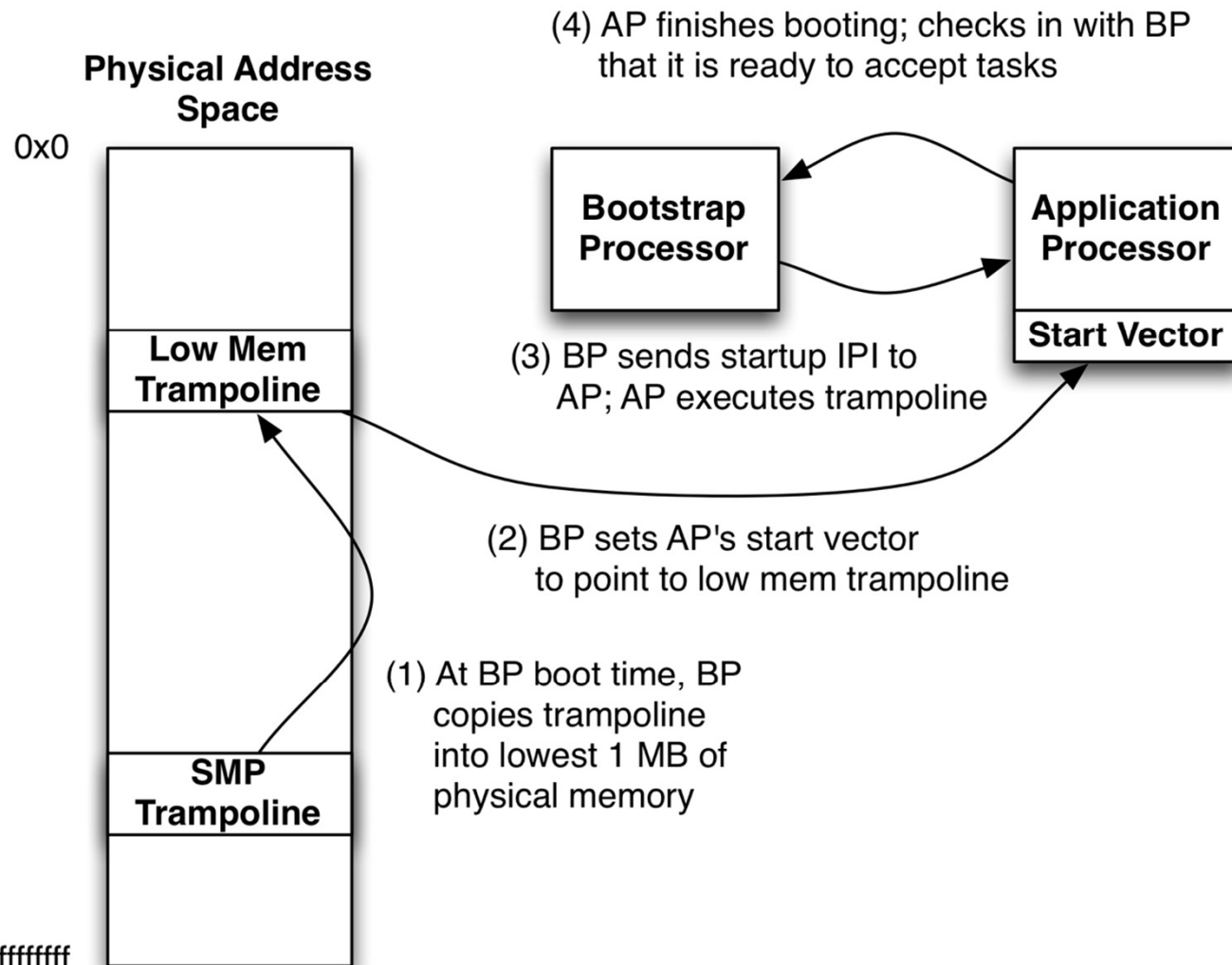
```
startup_64:
112  movq  %rdi, %rax
113  shrq  $PMD_SHIFT, %rax
114  andq  $(PTRS_PER_PMD - 1), %rax
115
116  /* MKLINUX -- at this point, %rax should be 0x0
117  * and %rdi should be rounded down to a multiple of 1 GB */
118  movq  $0, %rax
119  andq  $0xffffffffc0000000, %rdi
120
121  /* MKLINUX -- fill up level2_spare_pgt to map the 1 GB where the
122  * kernel has been loaded */
123  movq  $512, %rcx
124  leaq  __PAGE_KERNEL_IDENT_LARGE_EXEC(%rdi), %rdx
125  leaq  level2_spare_pgt(%rip), %rbx
126
127 1:  movq  %rdx, 0(%rbx, %rax, 8)
128  addq  $0x00200000, %rdx
129  incq  %rax
130  decq  %rcx
131  jnz   1b
132
133 ident_complete:
134
135  /*
136  ENTRY (secondary_startup_64)
```

Viewing diff output from file:///root/mklinux-patch/mklinux.git-14dec2012/linux-3.2.14-popcorn-x86-boot.patch

2 of 4 differences, 0 applied 9 of 14 files

The content of this slide is taken from Ben Shelton's MS Thesis

SMP Linux AP Boot Process

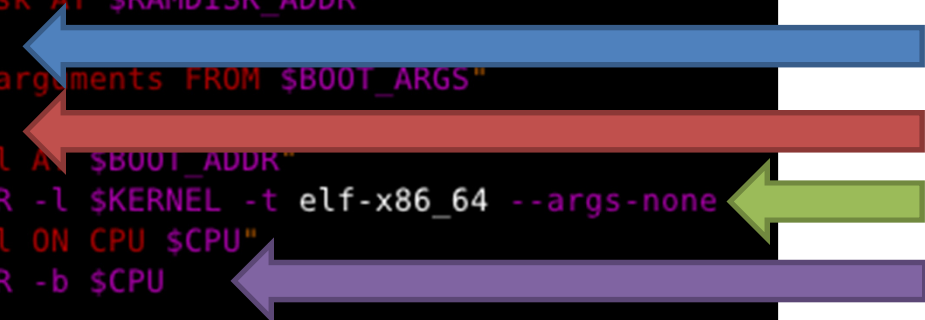


Secondary Kernels Booting (script)

- Due to not being fully integrated with *kexec*, the x86_64 version requires multiple loading steps

```
KERNEL=$1
BOOT_ARGS=$2
CPU=$3
BOOT_ADDR=$4
RAMDISK_ADDR=$5

echo "MKLINUX: loading ramdisk AT $RAMDISK_ADDR"
./copy_ramdisk $RAMDISK_ADDR
echo "MKLINUX: setting boot arguments FROM $BOOT_ARGS"
./set_boot_args $BOOT_ARGS
echo "MKLINUX: loading kernel AT $BOOT_ADDR"
./kexec_test -d -a $BOOT_ADDR -l $KERNEL -t elf-x86_64 --args-none
echo "MKLINUX: booting kernel ON CPU $CPU"
./kexec_test -d -a $BOOT_ADDR -b $CPU
~
```



Secondary Kernels Booting (cmdline)

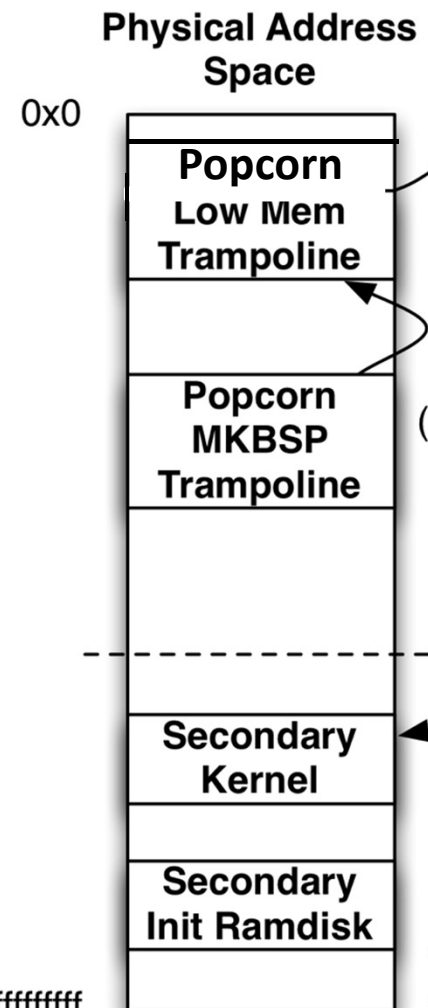
```
earlyprintk=ttyS0,115200 console=ttyS0,115200 acpi_irq_nobalance no_ipi_broadcast  
t lapic_timer=1000000 pci_dev_flags=0x8086:0x10c9:b,0x102b:0x0532:b,0x1002:0x5a1  
0:b,0x1002:0x4390:b,0x1002:0x4396:b,0x1002:0x4397:b,0x1002:0x4398:b,0x1002:0x439  
9:b mklinux debug vty_offset=0x1fac000000 present_mask=2 memmap=1920M@4608M memm  
ap=4592M$16M mem=6528M
```

- earlyprintk=ttyS0,115200
- console=ttyS0,115200
- acpi_irq_nobalance
- no_ipi_broadcast
- **lapic_timer=1000000**
- **pci_dev_flags=0x8086:0x10c9:b,0x102b:0x0532:b,0x1002:0x5a10:b,0x1002:0x4390:b,0x1002:0x4396:b,0x1002:0x4397:b,0x1002:0x4398:b,0x1002:0x4399:b**
- **mklinux**
- debug
- **vty_offset=0x1fac000000**
- **present_mask=2**
- memmap=1920M@4608M memmap=4592M\$16M mem=6528M

In this example the kernel will be loaded on core 2

Secondary Kernels Boot Process

- The kernel binary is copied to the selected physical location
- The boot **ramdisk** is copied to the designated kernel's memory area
- The secondary kernel's *boot_params* are initialized with the appropriate kernel arguments and **ramdisk** location/size
- A syscall to boot the secondary kernel is made, this sets the CPU's initial instruction pointer to point to the multi-kernel trampoline and will send an inter-processor interrupt (IPI) to the CPU to wake it up

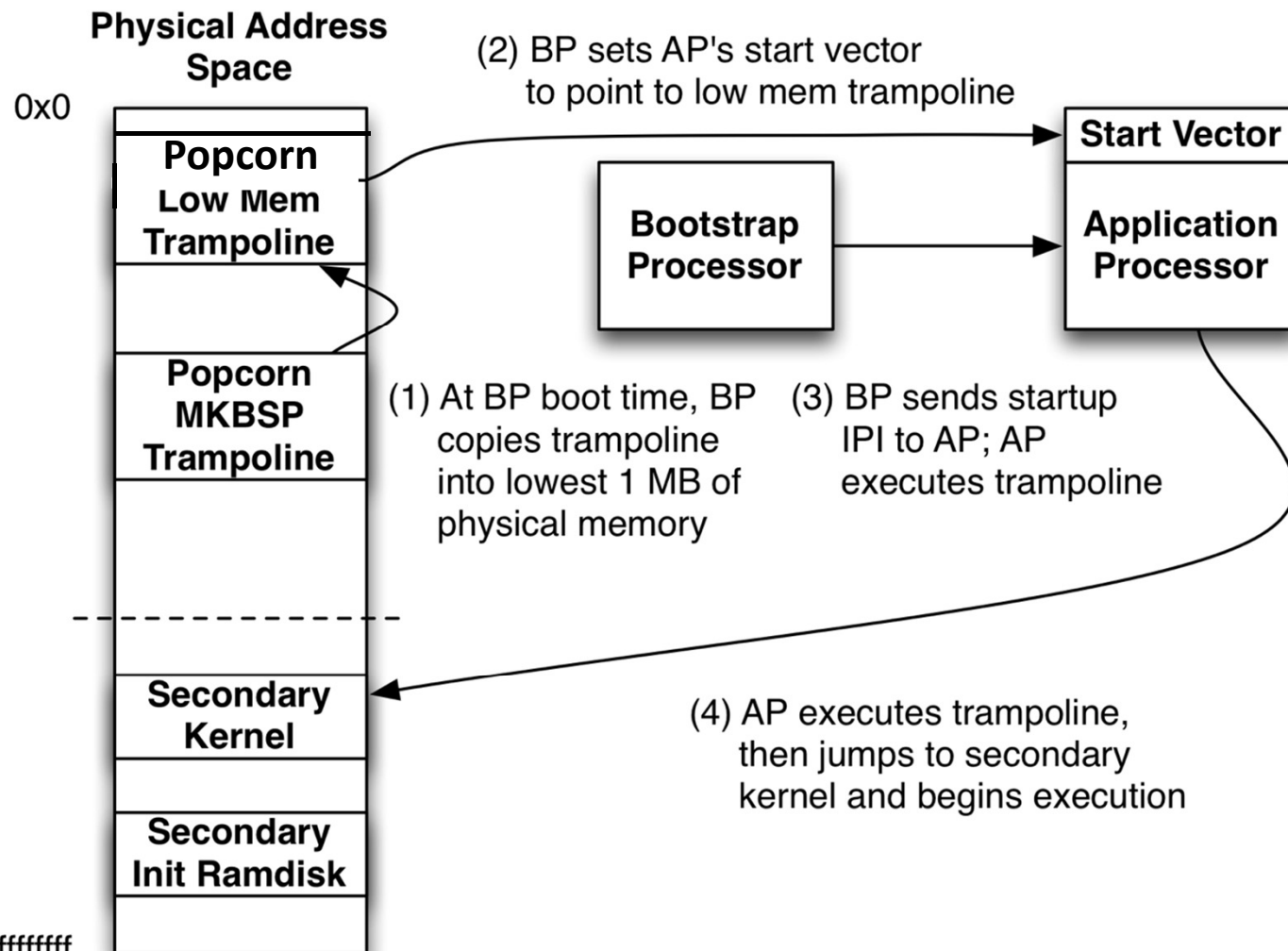


...from the trampoline

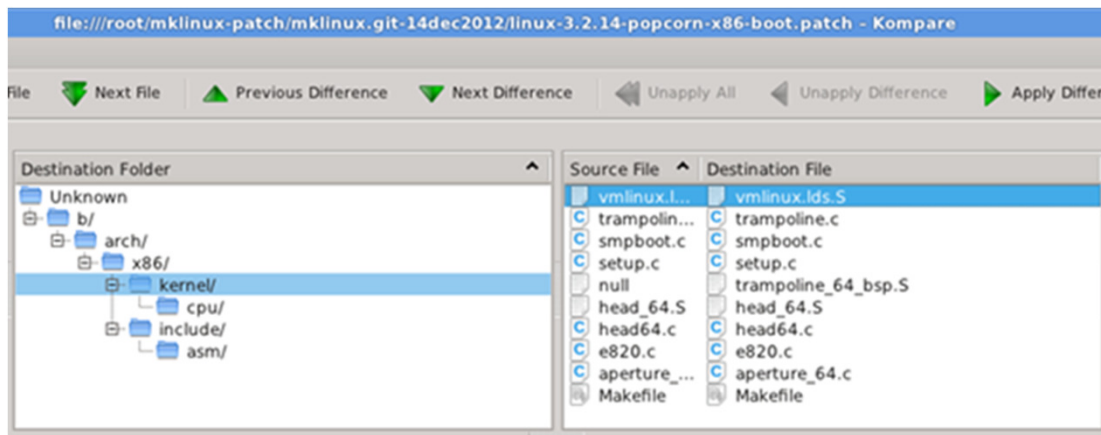
- Transition from real to protected mode, then jump to 64-bit long mode
- Load a 64-bit identity-mapped pagetable for the appropriate region
- Do a long jump to **head_64** of the guest kernel
- Fix up the kernel's pagetables, and setup the identity mappings for the 2 GB of physical address space where the kernel was loaded
- Continue to **x86_64_start_kernel()** to begin executing the kernel code itself

The content of this slide is taken from Ben Shelton's MS Thesis

Popcorn AP Boot Process



Popcorn Low Mem Trampoline



- We added a new trampoline
- There are now **two trampolines**
 - One for APs, i.e. cores that belongs to the same kernel
 - And another for secondary kernels

```
vmlinux.lds.S
SECTIONS
209     x86_trampoline_end = .;
210 }
211
212#ifdef CONFIG_POPCORN
213     .x86_trampoline_bsp : AT(ADDR(.x86_trampoline_bsp) - LOAD_OFFSET) {
214         x86_trampoline_bsp_start = .;
215         *(.x86_trampoline_bsp)
216         x86_trampoline_bsp_end = .;
217     }
218#endif
219
220     .x86_cpu_dev.init : AT(ADDR(.x86_cpu_dev.init) - LOAD_OFFSET) {
221         __x86_cpu_dev_start = .;
222         *(.x86_cpu_dev.init)
```

Setup Popcorn trampoline..

The screenshot shows a file diff tool window titled "file:///root/mklinux-patch/mklinux.git-14dec2012/linux-3.2.14-popcorn-x86-boot.patch - Kompare". The window displays a comparison between the source and destination files for the "trampoline.c" file. The source file is located at "/dev/a/arch/x86/kernel/cpu/include/asm/" and the destination file is at "Unknown/b/arch/x86/kernel/cpu/include/asm/".

The diff output shows the following changes:

Source Line	Destination Line	Difference
43	78	Inserted 11 lines
42	76	Inserted 1 line
26	29	Changed 1 line
9	11	Inserted 1 line
8	8	Inserted 2 lines

The source file "trampoline.c" contains the following code:

```
#include <asm/cacheflush.h>
#include <asm/pgtable.h>
7
unsigned char *x86_trampoline_base;
9
void __init setup_trampoline(void)
11{
    void __init setup_trampoline(void)
23    printk(KERN_DEBUG "Base memory trampoline at [%p] %llx size %zu\n",
24            x86_trampoline_base, (unsigned long long)mem, size);
25
26    memcpy(x86_trampoline_base, x86_trampoline_start, size);
27}
28
29/*
    static int __init configure_trampoline(void)
39    set_memory_x((unsigned long)x86_trampoline_base, size >> PAGE_SHIFT);
40    return 0;
41}
42arch_initcall(configure_trampoline);
```

The destination file "trampoline.c" contains the following code:

```
39void __init setup_trampoline_bsp(void)
40{
41    phys_addr_t mem;
42    size_t size = PAGE_ALIGN(x86_trampoline_bsp_end - x86_trampoline_bsp_start);
43
44    /* Has to be in very low memory so we can execute real-mode AP code. */
45    mem = memblock_find_in_range(0, 1<<20, size, PAGE_SIZE);
46    if (mem == MEMBLOCK_ERROR)
47        panic("Cannot allocate trampoline\n");
48
49    x86_trampoline_bsp_base = __va(mem);
50    memblock_x86_reserve_range(mem, mem + size, "TRAMPOLINE_BSP");
51
52    printk(KERN_DEBUG "Base memory trampoline BSP at [%p] %llx size %zu\n",
53            x86_trampoline_bsp_base, (unsigned long long)mem, size);
54
55    if (!mklinux_boot) {
56        memcpy(x86_trampoline_bsp_base, x86_trampoline_bsp_start, size);
57    } else {
58        printk("Popcorn boot: BSP trampoline will NOT be copied\n");
59    }
60}
61}
```

Resource Configurations

- In the **Popcorn utils** package there are scripts and applications to automatically (statically) subdivide hardware resources in your machine amongst kernels (and assists with booting them)
 - `generate_all.sh` creates the configurations
 - `mklinux_boot.sh` boots a secondary configuration
- ***mpart*** is the main application to divide computational and memory resources into partitions, optionally in a NUMA-aware fashion

mpart

```
antonio@gigi:~$ numactl --hardware
available: 8 nodes (0-7)
node 0 cpus: 0 1 2 3 4 5 6
node 0 size: 16381 MB
node 0 free: 15841 MB
node 1 cpus: 8 9 10 11 12 13 14 15
node 1 size: 16384 MB
node 1 free: 15893 MB
node 2 cpus: 16 17 18 19 20 21 22 23
node 2 size: 16384 MB
node 2 free: 15943 MB
node 3 cpus: 24 25 26 27 28 29 30 31
node 3 size: 16384 MB
node 3 free: 15940 MB
node 4 cpus: 32 33 34 35 36 37 38 39
node 4 size: 16384 MB
node 4 free: 15916 MB
node 5 cpus: 40 41 42 43 44 45 46 47
node 5 size: 16384 MB
node 5 free: 15895 MB
node 6 cpus: 48 49 50 51 52 53 54 55
node 6 size: 16384 MB
node 6 free: 15940 MB
node 7 cpus: 56 57 58 59 60 61 62 63
node 7 size: 16368 MB
node 7 free: 15921 MB
node distances:
node  0  1  2  3  4  5  6  7
 0: 10 16 16 22 16 22 16 22
 1: 16 10 22 16 22 16 22 16
 2: 16 22 10 16 16 22 16 22
 3: 22 16 16 10 22 16 22 16
 4: 16 22 16 22 10 16 16 22
 5: 22 16 22 16 16 10 22 16
 6: 16 22 16 22 16 22 10 16
 7: 22 16 22 16 22 16 16 10
```

- Gathers NUMA information from:
 - /sys/devices/system/node/
 - /proc/meminfo
- Outputs the kernel boot arguments for different configurations
 - Different alignments and resource reservations
 - Clustering (one kernel per NUMA-zone)
 - Partitioning (one kernel per core)

Masking Resources

- When Linux boots, it automatically discovers the PCI devices present in the system
- Each kernel assumes that it has access to all of the devices in the system unless a device is “blacklisted” using kernel arguments
- Example syntax (from slide 15)
 - **pci_dev_flags=0x8086:0x10c9:b,0x102b:0x0532:b,0x1002:0x5a10:b,0x1002:0x4390:b,0x1002:0x4396:b,0x1002:0x4397:b,0x1002:0x4398:b,0x1002:0x4399:b**

Partitioning and Clustering

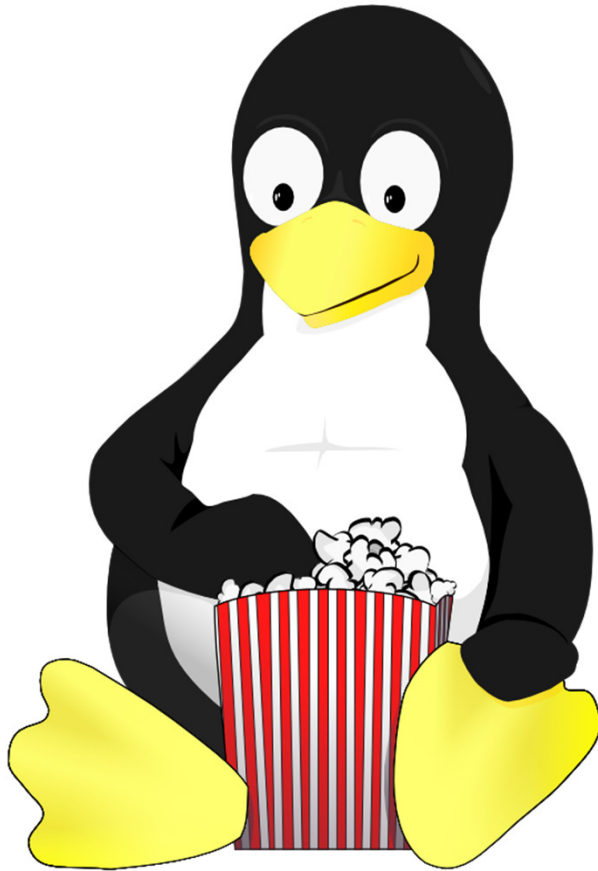
- Logical to physical translation for CPU ID and APIC ID to provide a contiguous range
- I/O APIC is set up to direct device interrupts to the kernel that owns the device
- Local APICs are set up to allow inter-processor interrupts (IPI) between kernels for synchronization and communication

Interact with Secondary Kernels

- A user can interact with any of the kernels by using any of:
 - Virtual TTY
 - example: `$ cu -l /dev/ttyX`
 - X is the core ID on which the kernel is started
 - Virtual TTYs appear to applications as regular TTYs
 - Virtual Network Switch
 - example: `$ ssh 10.1.2.X`
 - X is the core ID plus one on which the kernel is started
 - Different versions are currently available depending on the performance and resilience needed
 - Inter-Kernel Shared Memory

Questions?

Team



www.popcornlinux.org